John L. Viescas, Michael J. Hernandez

SQL-lekérdezések földi halandóknak

Budapest, 2009

John L. Viescas
Michael J. Hernandez

SQL-lekérdezések

földi halandóknak

Gyakorlati útmutató
az SQL nyelvű adatkezeléshez

KISKAPU
Addison-Wesley

A fordítás a következő angol eredeti alapján készült:

John L. Viescas, Michael J. Hernandez: SQL Queries for Mere Mortals®: A Hands-On Guide to Data Manipulation in SQL

Authorized translation from the English language edition, entitled SQL QUERIES FOR MERE MORTALS®: A HANDS-ON GUIDE TO DATA MANIPULATION IN SQL, 2nd Edition, ISBN 0321444434, by VIESCAS, JOHN L.; HERNANDEZ, MICHAEL J., published by Pearson Education, Inc., publishing as Addison-Wesley Professional.

Lektor: *Rézműves László*
Fordítás: *Bálint István Zsolt, Borbély György, Csépány Gergely László, Fodor Ferenc Dániel, Heiner Péter, Kmeczkó Csilla, Koronczay Dávid, Rézműves László, Szabados Ernő, Varga Péter*
Műszaki szerkesztő: *Csutak Hoffmann Levente*
Tördelés: *Kis Péter*

Felelős kiadó a Kiskapu Kft. ügyvezető igazgatója
© 2009 Kiskapu Kft.
1134 Budapest, Csángó u. 8.
Fax: (+36-1) 303-1619
http://www.kiskapukiado.hu/
e-mail: kiado@kiskapu.hu

ISBN: 978 963 9637 52 8

Készült a debreceni Kinizsi Nyomdában
Felelős vezető: *Bördős János*

„Hacsak nem vagyunk profik, ez az egyetlen SQL-könyv, amire valaha szükségünk lesz. A szerzők felfedik az összetett lekérdezések rejtélyét, és olyan világosan magyarázzák el a különböző elveket és eljárásokat, hogy egy 'földi halandó' is képes lesz emberfeletti tetteket véghezvinni. Kihagyhatatlan könyv!"

– Graham Mandeno, adatbázis-szakértő

„Én elsősorban ennek a könyvnek az első kiadásából tanultam meg az SQL használatát: remekül szerkesztett könyv, amely jól felépített módon mutatja be a nyelvet. Örülök, hogy egy második kiadást is megért, így másoknak is a hasznára lehet. Onnan kezdve, hogy miként tervezzük meg az adattábláinkat, hogy az SQL segítségével hatékonyan kezelhessük az adatainkat (ami általában gondot jelent a kezdőknek), a könyv részletesen bemutatja az SQL képességeit és különféle szerkezeteit, így az olvasó a kötet elolvasása és a példák áttekintése után félig-meddig szakértővé válhat. Megtanuljuk, hogyan alakíthatunk egy angol nyelvű kérdést értelmes SQL-utasítássá, ami nagyban elősegíti, hogy elsajátíthassuk a nyelvet, a valós életből vett számos példa alapján pedig képet formálhatunk arról, hogy miként adhatunk válaszokat az SQL segítségével az adatbázisban található adatokra vonatkozó kérdésekre. Már egyetlen olyan megjegyzés elolvasása, amely egy csapdára figyelmeztet, több pénzt takaríthat meg nekünk a könyv áránál, mivel így elkerülhetjük az adott problémát, amikor lekérdezéseket írunk. Ha az adatbázisaink képességeit teljes mértékben ki szeretnénk aknázni, ezt a könyvet feltétlenül el kell olvasnunk!"

– Kenneth D. Snell, Ph.D., adatbázis-tervező és -programozó

„Nem tudom, hogy az iskolában még tanítják-e – ha nem, az elég baj –, de annak idején mi a hetedik vagy a nyolcadik osztályban megtanultuk, hogyan rajzolhatunk fel egy mondatot diagramként. Lehet, hogy már nem emlékszünk rá, hogyan csináltuk, de mindnyájunk mondatszerkesztése javult általa. John Viescas és Mike Hernandez bizonyára nem felejtették el, mert képesek rá, hogy hétköznapi angol kérdéseket alakítsanak át SQL-utasításokká, és mindenki számára könnyen érthetővé teszik az E. F. Codd-nak a relációs adatbázisok tervezéséről írott eredeti munkájában vázolt bonyolult matematikai halmazelméletet és elsőrendű predikátumlogikát. Könyvük nélkülözhetetlen minden adatbázis-tervező számára. Ha kezdő–középhaladó szintű tankönyvet keresünk az SQL-ről, ez a kötet kötelező anyag, függetlenül attól, hogy hány másik könyvet veszünk meg még."

– Arvin Meyer, MCP, MVP

„Az *SQL-lekérdezések földi halandóknak* második kiadása lépésről lépésre haladó, olvasmányos bevezetést ad az SQL-lekérdezések írásába. Példák százait tartalmazza részletes magyarázattal ellátva, és minden eszközt megad, ami ahhoz szükséges, hogy képesek legyünk megérteni az SQL-lekérdezéseket, valamint létrehozni és módosítani azokat."
– Keith W. Hare, elnök, ISO/IEC JTC1 SC32 WG3 – nemzetközi SQL-szabványügyi bizottság

„Még ma, a varázslók és kódgenerátorok korában is biztos tudással kell rendelkeznie a strukturált lekérdezőnyelvről (SQL, Structured Query Language – a szabványos nyelv a legtöbb adatbázisrendszerrel való kommunikációhoz) minden olyan adatbázis-fejlesztőnek, aki sikeres szeretne lenni. Ebben a könyvben John és Mike nagyszerűt munkát végezve életre keltenek egy általában száraz és bonyolult témát, és az anyagot logikus rendben, sok humorral és rengeteg hasznos példával körítve tálalják. Azt kellene mondanom, hogy kötetüknek minden komoly fejlesztő könyvtárában kitüntetett helyet kellene kapnia – csakhogy biztos vagyok benne, hogy olyan sűrűn fogják forgatni, hogy nem fog sok időt tölteni a polcon!"

– Doug Steele, Microsoft Access-fejlesztő, szakíró

Tartalomjegyzék

III. rész • Többtáblás lekérdezések

7. fejezet • Halmazokban gondolkodni

8. fejezet • Belső összekapcsolás

Előszó

Az SQL adatbázisnyelv nemzetközi szabvánnyá válása óta eltelt 20, illetve az SQL adatbázis-kezelő termékek piacra dobása óta eltelt 25 évben az SQL vált az adatok tárolásának, módosításának, kiolvasásának és törlésének meghatározó nyelvévé. Ma a világ – és a világgazdaság – adatainak jelentős részét SQL-adatbázisok tárolják.

Az SQL ott van mindenütt, mert az adatok kezelésének igen hatékony eszköze. Jelen van a nagyteljesítményű tranzakciófeldolgozó rendszerekben, a webes felületek mögött, sőt találkoztam már vele hálózatfigyelő eszközökben és levélszemétszűrő tűzfalakban is.

Az SQL-utasítások mai formájukban végrehajthatók közvetlenül, beágyazhatók más programozási nyelveken írt kódokba, és hozzáférhetők hívási felületeken keresztül. Megbújik a grafikus felületek fejlesztőeszközeiben, a kódgenerátorokban és a jelentéskészítőkben. Nem számít, hogy látható vagy rejtett módon, de a háttérben végrehajtott lekérdezések rendszerint SQL nyelvűek. Ahhoz tehát, hogy megértsük a mai alkalmazások működését és újakat készíthessünk, ismernünk kell az SQL-t.

Az *SQL-lekérdezések földi halandóknak* második kiadása lépésről lépésre haladó, olvasmányos bevezetést ad az SQL-lekérdezések írásába. Példák százait tartalmazza részletes magyarázattal ellátva, és minden eszközt megad, ami ahhoz szükséges, hogy képesek legyünk megérteni az SQL-lekérdezéseket, valamint létrehozni és módosítani azokat.

Adatbázis-szakértő tanácsadóként, és mind az amerikai, mind a nemzetközi SQL-szabványügyi bizottság tagjaként sok időt töltök munkám során az SQL-lel, ezért bátran elhihetik nekem, hogy ennek a könyvnek a szerzői nem csak értik az SQL-t, hanem azt is tudják, hogyan lehet elmagyarázni a működését. Ez a két képességük az, ami ezt a kötetet kivételesen értékes forrássá teszi.

Keith W. Hare
főtanácsadó, JCC Consulting, Inc.
alelnök, INCITS H2 – az Amerikai Egyesült Államok SQL-szabványügyi bizottsága
elnök, ISO/IEC JTC1 SC32 WG3 – a nemzetközi SQL-szabványügyi bizottság

A kiadó előszava

„A nyelv puszta természeténél fogva közösségi dolog,
vagyis soha nem egy konkrét dolgot, hanem
egy kompromisszumot fejez ki – azt, ami benned,
bennem és mindnyájunkban közös."
– Thomas Earnest Hulme, Elmélkedések

Megtanulni, hogy miként nyerhetünk ki és kezelhetünk információkat egy adatbázisból, általában zavarbaejtő feladat, ugyanakkor viszonylag egyszerű, ha tudjuk, mit is kérdezünk, illetve értjük, hogy milyen változtatást is próbálunk eszközölni az adatbázisban. Miután megértettük a kérdést, lefordíthatjuk egy olyan nyelvre, amelyet minden mai adatbázisrendszer megért – ami a legtöbb esetben a Structured Query Language (SQL, strukturált lekérdezőnyelv). A kérelmet egy SQL-utasítássá kell alakítanunk, hogy az adatbázis-kezelő rendszerünk tudja, hogy milyen adatokat szeretnénk kinyerni vagy módosítani. Az SQL biztosítja az eszközöket ahhoz, hogy kommunikálni tudjunk az adatbázisrendszerrel.

Adatbázis-tanácsadóként végzett sok évnyi munkánk során azt tapasztaltuk, hogy azoknak a száma, akiknek csupán adatok kiolvasására vagy egyszerű módosításokra van szükségük egy adatbázisban, messze meghaladja azokét, akiknek a feladata az adatbázist működtető programok és alkalmazások elkészítése.

Ezzel a témával eddig sajnos egyetlen könyv sem foglalkozott kizárólagosan, különösen nem a „földi halandók" szemszögéből. Természetesen sok jó könyvet írtak az SQL-ről, de a legtöbbnek az adatbázisok programozása és fejlesztése állt a középpontjában.

Ezt figyelembe véve úgy döntöttünk, hogy ideje egy olyan könyvet írni, amely azt tanítja meg, hogy miként kérdezhetünk le helyesen és hatékonyan egy adatbázist. Könyvünk első kiadása 2000-ben jelent meg. Ebben az új, javított kiadásban abba is be szeretnénk vezetni az olvasókat, hogy az SQL-t használatakor milyen alapvető eljárásokkal módosíthatják az adatbázisokban található adatokat.

Az Olvasó tehát a fenti döntésünk eredményét tartja a kezében. Könyvünk annyiban egyedülálló az SQL-lel foglalkozó kötetek között, hogy az adatbázisrendszerek megvalósításaitól szinte teljesen függetlenül tárgyalja az SQL-t. A második kiadás új példák százait, valamint a népszerű nyílt forrású MySQL adatbázis-kezelő rendszer mintaadatbázisainak könyvünkhöz igazított változatait tartalmazza. Ha elolvastuk a könyvet, minden tudással fel leszünk vértezve ahhoz, hogy bármilyen információt kiolvassunk vagy módosítsunk, amire csak szükségünk van.

Köszönetnyilvánítás

Egy ehhez hasonló könyv megírása mindig csapatmunka: szerkesztők, munkatársak, rokonok és barátok nyújtanak lelkesen támogatást, és látják el a szerzőket hasznos tanácsokkal, amikor a legnagyobb szükség van rájuk. Minket is ezek az emberek bátorítottak szüntelenül, ők segítettek a feladatra összpontosítani, és ők ösztönöztek, hogy csináljuk végig.

Először is szeretnénk köszönetet mondani Elizabeth Peterson szerkesztőnek, amiért ösztökélt, hogy készítsük el a második kiadást. Köszönet jár Kristin Weinbergernek is, aki a munka során a helyes irányba terelgetett minket. Nem feledkezhetünk meg Chuck Toporek szerkesztőről, Romny French-ről és a nagyszerű produkciós csapatról sem. Külön köszönet illeti Chrysta Meadowbrooke-ot, aki nagyszerű munkát végzett a végső kézirat korrektúrázásával, és nem csak sok-sok következetlenséget javított ki, hanem még a javításra szoruló SQL-példákra is rámutatott. Végül, szeretnénk megköszönni Karen Gettman főszerkesztő erőfeszítéseit, aki összerakta ezt a csapatot, és figyelemmel kísérte a teljes munkát.

Elismerés jár szakmai lektorainknak, Stephen Forte-nak és Keith Hare-nek is. Keith sok időt töltött az összes példa átnézésével, kiszúrt jónéhány hibát, és javaslatokat tett a szöveg jobbá tételére. Még egyszer köszönjük mindenkettőjüknek, hogy időt és energiát fektettek abba, hogy segítsenek nekünk egy igazán alapos munkát letenni az asztalra az SQL-lekérdezésekről.

Végül még egyszer külön köszönet Keith Hare-nek, aki előszót írt a kötethez. Keith, aki a nemzetközi SQL-szabványügyi bizottság elnöke, elsőrangú SQL-szakértő, akinek a tárgyban szerzett tudását és tapasztalatát igen nagyra becsüljük, és megtiszteltetésnek vesszük, hogy a könyvünk felvezetéséül szolgáló gondolatait és megjegyzéseit papírra vetette.

A szerzőkről

John L. Viescas független tanácsadó, több mint 40 évnyi tapasztalattal. Pályafutását rendszerelemzőként kezdte, és nagy adatbázis-alkalmazásokat tervezett az IBM nagygépes rendszereihez. Hat évet töltött az Applied Data Research-nél a texasi Dallasban, ahol egy több mint 30 emberből álló csapat vezetőjeként az IBM mainframe számítógépeihez készült adatbázis-kezelő termékekhez kapcsolódó kutatásokért, fejlesztésekért és a termék-

támogatásáért felelt. Miközben az Applied Data Research-nél dolgozott, diplomát szerzett üzletviteli pénzügyi ismeretekből a Dallasban található texasi egyetemen. Tanulmányait kitüntetéssel végezte.

John 1988-ban csatlakozott a Tandem Computers, Inc. gárdájához, ahol adatbázis-népszerűsítő programok fejlesztéséért és megvalósításáért felelt az Egyesült Államok nyugati részét kiszolgáló eladási részlegen, valamint műszaki oktatóanyagokat állított össze és szemináriumokat tartott a Tandem NonStop SQL nevű relációs adatbázis-kezelő rendszeréről Hawaiitól Coloradóig és Alaszkától Arizonáig. John az első könyvét (*A Quick Reference Guide to SQL*, Microsoft Press, 1989) kutatási projektként írta, amelynek célja az ANSI-86 SQL-szabvány, az IBM DB2, a Microsoft SQL Server, az Oracle Corporation Oracle rendszere és a Tandem NonStop SQL nyelvtana közötti hasonlóságok dokumentálása volt. A Tandem-től szabadságot kivéve megírta a *Running Microsoft Access* (Microsoft Press, 1992) első kiadását, amelyből eddig négy kiadás készült, a *Microsoft Office Access Inside Out* (Microsoft Press, 2004 és 2007 a *Running* sorozat folytatása) két kiadása és a *Building Microsoft Access Applications* (Microsoft Press, 2005) mellett.

John 1993-ban megalapította a saját tanácsadó cégét, amely információs rendszerek üzemeltetésével kapcsolatban ad tanácsokat különféle kis- és nagyvállalatoknak szerte a világon, elsősorban a Microsoft Access és SQL Server adatbázis-kezelő termékekre szakosodva. A cégnek irodái vannak a New Hampshire-i Nashuában, illetve a franciaországi Párizsban. 1993 óta minden évben elnyerte a „legértékesebb szakértő" címet a Microsoft Product Support Services-től a nyilvános támogatási fórumokon nyújtott szakmai segítségéért.

John webhelyét a *www.viescas.com* címen látogathatjuk meg, e-mailben pedig a *johnv@viescas.com* címre írhatunk neki.

* * *

Michael J. Hernandez veterán adatbázis-fejlesztő, több mint 20 évnyi alkalmazásfejlesztési tapasztalattal, amelyet különböző iparágakban, a legkülönfélébb ügyfeleknek végzett munka során szerzett. Mike szakterülete a relációs adatbázisok tervezése – az adatbázis-tervezésről ő írta a nagy sikerű *Database Design for Mere Mortals* (második kiadás, Addison-Wesley, 2004; magyarul: *Adatbázis-tervezés – A relációs adatbázisok alapjairól földi halandóknak*, Kiskapu, 2004) című könyvet. Egész pályája során az SQL-lel dolgozott, olyan SQL alapú adatbázis-kezelőkhöz fejlesztve alkalmazásokat, mint a Microsoft Access és a Microsoft SQL Server, emellett számos adatbázis-kezelési témájú könyvnek és folyóiratnak volt a társszerzője, illetve szakmai lektora.

Mike 2002-ben a Microsofthoz került, ahol teljes munkaidőben dolgozik. Kezdetben a Visual Studio Tools for Office (VSTO) csapatának közösségi programmenedzsereként a fejlesztők közösségi tevékenységét irányította, majd 2006-ban a VSTO termékmenedzse-

re lett. Ebben a munkakörben a feladata a jövőbeni termékstratégia kidolgozásának elősegítése, valamint a VSTO népszerűsítése a vásárlók és fejlesztők között különféle módszerekkel. Ahogy karrierje során már annyiszor tette, Mike gyakran tart előadást fejlesztői konferenciákon és felhasználói találkozókon szerte az országban, illetve az egész világon.

Valamelyik korábbi életében Mike zenészként ért el sikereket, a legkülönfélébb helyeken szórakoztatva a közönséget. Könnyed előadásmódját és kontaktusteremtő képességét a közönséggel ennek a szórakoztatóművészi örökségének tulajdonítja. Mike soha nem szűnt meg zenésznek lenni, ezért a VSTO csapatának tagjaiból zenekart verbuvált, hogy alkalma nyíljon új tömegek előtt pengetni szeretett gitárját. Mike ma is gyakran a húrok közé csap, ha akad pár szabad perce a munkaértekezletek között. Mike élvezi az élet olyan apró dolgait, mint az órákig tartó böngészés egy Barnes & Noble könyvesboltban, egy amerikai kávé szürcsölése a Starbucksban, egy finom szivar elpöfékelése, vagy a hegyi kerékpározás feleségével, Kendrával.

Mike-nak e-mailben az *mjhernandez@msn.com* címre írhatunk.

Bevezetés

„Bizonyára ön is halandó, így tévedhet."
– James Shirley – The Lady of Pleasure

Ha nem csak alkalmi számítógéphasználók vagyunk, valószínűleg használtuk már a strukturált lekérdezőnyelvet vagy más néven SQL-t – akár anélkül, hogy tudtunk volna róla. Az SQL a szabványos nyelv, amelyen keresztül a legtöbb adatbázisrendszerrel kommunikálhatunk. Minden alkalommal, amikor adatokat töltünk be egy táblázatkezelő programba vagy olvasztunk be egy dokumentumba egy szövegszerkesztő programban, több mint valószínű, hogy az SQL-t használjuk valamilyen formában. Amikor csatlakozunk az Internetre, hogy meglátogassunk egy kereskedelmi webhelyet a Weben, és megrendeljünk egy könyvet, lemezt, filmet vagy valami mást abból a sok tucatnyi termékfajtából, ami hozzáférhető, szinte biztos, hogy a weboldal a háttérben ugyancsak az SQL segítségével éri el a bolt adatbázisát. Ha tehát egy olyan adatbázisrendszerből kell adatokat kinyernünk, amely az SQL-t használja, olvassuk el ezt a könyvet, és jobban megérthetjük a nyelvet.

Földi halandók vagyunk?

Feltehetjük a kérdést: „Ki számít közönséges *földi halandónak*? Én?" Nos, a válasz nem egyszerű. Amikor elkezdtük írni ezt a könyvet, azt hittük, hogy az SQL nevű adatbázisnyelv szakértői vagyunk. Menet közben azonban rájöttünk, hogy sok szempontból mi is csak közönséges földi halandók vagyunk. Az SQL néhány konkrét megvalósítását nagyon jól ismertük, de ahogy megvizsgáltuk, hogyan alkalmazzák az SQL-t a különböző, kereskedelmi forgalomban kapható termékek, a nyelv számos addig ismeretlen finomságát felfedeztük.

Tehát ha az alábbi leírások bármelyike ránk illik, mi is földi halandók vagyunk:

- Ha olyan számítógépes alkalmazásokat használunk, amelyek segítségével adatokat érhetünk el egy adatbázisrendszerből, valószínűleg egyszerű földi halandók vagyunk. Az első alkalommal, amikor nem a várt adatokat kapjuk az alkalmazásunkba épített lekérdezőeszközöket használva, meg kell vizsgálnunk a háttérben megbúvó SQL-utasításokat, hogy kiderítsük, miért.
- Ha mostanában fedeztük fel a számos elérhető asztali adatbázis-kezelő alkalmazás egyikét, de nehezen tudjuk meghatározni és lekérdezni a szükséges adatokat, biztosan földi halandók vagyunk.
- Ha adatbázis-programozóként dolgozunk, akinek kész termékeket kell szem előtt tartania egyes bonyolult problémák megoldásakor, közönséges földi halandók vagyunk.
- Ha szakértői vagyunk egy adott adatbázis-kezelő terméknek, de most azzal a feladattal találjuk magunkat szembe, hogy a meglevő rendszerünk adatait egy másik, az SQL-t támogató rendszerbe kell beépítenünk, csupán földi halandók vagyunk.

Röviden, *bárki*, akinek egy az SQL-t ismerő adatbázisrendszert kell használnia, haszonnal forgathatja ezt a könyvet. Kezdő adatbázis-felhasználóként, aki csak most fedezte fel, hogy a számára szükséges adatokat az SQL segítségével nyerheti ki, minden alapvető ismeretet (sőt annál többet is) megtalálhatunk a kötetben, ha pedig tapasztalt felhasználók vagyunk, akinek hirtelen bonyolult feladatokat kell megoldania, vagy több olyan rendszert kell együttműködésre bírnia, amelyek az SQL-re épülnek, a könyvből ötleteket kaphatunk arra, hogy miként aknázhatjuk ki az SQL adatbázisnyelv sokrétű képességeit.

Néhány szó a könyvről

Minden, amit ebben a könyvben olvashatunk, az SQL adatbázisnyelv jelenlegi International Organization for Standardization (ISO, Nemzetközi Szabványügyi Szervezet) szabványán (az ISO/IEC 9075-2:2003 jelű dokumentumon) alapul, abban a formában, ahogy azt a népszerű adatbázisrendszerek legtöbbje ma megvalósítja. Az ISO-dokumentumot az American National Standards Institute (ANSI, Amerikai Szabványügyi Hivatal) is a magáévá tette, vagyis valóban nemzetközi szabványról van szó. A könyvben ismertetett SQL tehát *nem* kötődik egyetlen konkrét szoftvertemékhez sem.

Ahogy a 3. fejezetben részletesebben megtanulhatjuk, az SQL-szabvány egyszerre többet és kevesebbet határoz meg annál, amit a legtöbb kereskedelmi adatbázis-kezelő termék megvalósít. Az adatbázis-kezelők gyártói a haladó szolgáltatások közül még sokat nem valósítottak meg, de a szabvány magját a legtöbben támogatják.

A népszerű termékek széles körét vizsgáltuk meg, hogy meggyőződjünk róla, hogy a könyvben átadott ismeretek a gyakorlatban is alkalmazhatók. Ha egyes elterjedt termékekben azt találtuk, hogy nem ismerik a nyelv valamelyik alapszolgáltatását, a szövegben figyelmeztetést helyeztünk el, és leírtuk, milyen más szabványos módon fogalmazhatjuk meg az adatbáziskezelőhöz intézett kérelmünket. Azokban az esetekben, amikor az SQL-szabvány egy fontos részét csak néhány gyártó támogatja, ismertetjük a kérdéses utasítások formáját, és kerülő megoldásokat javaslunk.

A könyvet öt főbb részre osztottuk:

- Az I. rész (*A relációs adatbázisok és az SQL*) elmagyarázza, hogyan épülnek a mai adatbázisrendszerek egy szigorú matematikai modellre, és röviden ismerteti annak az adatbázis-lekérdező nyelvnek a történetét, amelyből a ma SQL-ként ismert nyelv kifejlődött. Ezen kívül ebben a részben egyes egyszerű szabályokat is megtanulunk, amelyeket betartva gondoskodhatunk róla, hogy az adatbázisaink felépítése egészséges legyen.
- A II. rész (*Az SQL alapjai*) a SELECT utasítás használatába, a kifejezések létrehozásába, valamint az információknak az ORDER BY záradékkal történő rendezésébe vezet be bennünket, illetve megtudhatjuk belőle, hogy miként szűrhetjük az adatokat a WHERE záradék segítségével.
- A III. rész (*Többtáblás lekérdezések*) megmutatja, hogyan fogalmazhatunk meg olyan lekérdezéseket, amelyek egynél több táblából olvasnak ki adatokat. Ebben a részben megtanuljuk, hogyan kapcsolhatunk össze táblákat egy lekérdezésben az INNER JOIN, OUTER JOIN és UNION műveletekkel, valamint hogy miként használhatunk allekérdezéseket.
- A IV. rész (*Adatok összesítése és csoportosítása*) azt tárgyalja, hogy miként nyerhetünk ki összesítő adatokat, illetve hogy miként csoportosíthatjuk és szűrhetjük az összesített információkat. Ebben a részben tanulunk majd a GROUP BY és a HAVING záradékokról.
- Az V. rész (*Adathalmazok módosítása*) azt magyarázza el, hogy miként írhatunk olyan lekérdezéseket, amelyek sorok egy adott csoportját módosítják egy táblában. Ennek a résznek a fejezeteiből tanulhatjuk meg az UPDATE, az INSERT és a DELETE utasítás használatát.

A könyv végén található függelékekben megtalálhatjuk az összes megismert SQL-elem szintaxisdiagramját, a mintaadatbázisok felépítését, az öt legelterjedtebb adatbázis-kezelő rendszerben megvalósított dátum- és időkezelő függvényeket, valamint az SQL további tanulmányozásához ajánlott könyvek jegyzékét. A könyvhöz egy CD is tartozik, amely a kötet összes mintaadatbázisát tartalmazza, több különböző formátumban.

Miről nem szól a könyv?

Bár ez a könyv a 2003-as SQL-szabványon alapul, amely a legfrissebb volt, amikor a könyvet írtuk (azóta kidolgozás alatt áll a 2007/2008-as szabványvázlat), nem tárgyalja a szabvány minden részletét. Valójában a 2003-as SQL-szabvány sok szolgáltatását még hosszú évekig nem fogják megvalósítani – ha egyáltalán megvalósítják – a fontosabb adatbázisrendszerekben. Könyvünk alapvető célja, hogy szilárd tudást nyújtson az SQL nyelvű lekérdezések írásához. A könyv során gyakran javasoljuk „az adatabázis-kezelőnk dokumentációjának fellapozását", hogy megtudjuk, működik-e egy adott szolgáltatás. Ez nem azt jelenti, hogy a főbb adatbázisrendszerek szolgáltatásainak csak a legkisebb közös nevezőjét tárgyaljuk, mindazonáltal igyekszünk figyelmeztetni, ha egyes rendszerek egy szolgáltatást másképp vagy egyáltalán nem valósítanak meg.

Ha az adatbázisunkat hibásan terveztük meg, nehézséget okozhat mást létrehozni, mint egyetlen táblát használó egyszerű lekérdezéseket, ezért egy fejezetet az adatbázis-tervezésnek szenteltünk, hogy segítsünk azonosítani a problémás helyzeteket, de az említett fejezetben csak az alapelveket ismertetjük. Az adatbázis-tervezési elvek kimerítő tárgyalása, illetve annak a vizsgálata, hogy egy adott felépítést hogyan valósíthatunk meg egy konkrét adatbázisrendszerben, túlmutat könyvünk keretein.

Ennek a könyvnek az sem a témája, hogy miként oldhatunk meg egy problémát a leghatékonyabban. Ahogy a későbbi fejezeteket olvassuk, láthatjuk majd, hogy egy adott problémára több megoldást is javasolunk. Egyes esetekben, amikor egy lekérdezés megírása egy adott módon valószínűleg minden rendszeren teljesítményproblémákat okoz, igyekszünk erre felhívni a figyelmet. Ugyanakkor minden adatbázisrendszernek megvannak a maga erősségei és gyengéi. Miután elsajátítottuk az alapokat, készen állunk arra, hogy továbblépjünk, és beleássuk magunkat annak a konkrét adatbázisrendszernek a működésébe, amelyen megtanultuk, hogyan fogalmazhatjuk meg a lekérdezéseinket úgy, hogy optimálisan fussanak.

Hogyan olvassuk a könyvet?

A könyv fejezeteit úgy terveztük meg, hogy sorrendben célszerű elolvasni őket. Minden fejezet az előző fejezetekben megtanult elvekre épül, mindazonáltal akkor sem veszünk el, ha rögtön a könyv közepére ugrunk. Például ha már ismerjük a SELECT utasítás alapvető záradékait, és a JOIN-okról szeretnénk többet tudni, nyomban a 7., 8. és 9. fejezethez lapozhatunk.

Sok fejezet végén számos mintafeladatot találhatunk, a megoldásukkal és eredményhalmaz-mintákkal együtt. Ajánlott először néhány példát tanulmányozni, hogy jobban értsük a szükséges eljárásokat, majd önállóan megpróbálni megoldani a későbbi példákat, anélkül, hogy megnéznénk az ajánlott megoldásokat.

Megjegyzendő, hogy ahol egy adott lekérdezés sorok tucatjait adja vissza az eredményhalmazban, a könyvben csak az első néhány sort mutatjuk meg, hogy képet adjunk arról, hogyan kell festenie a kapott válasznak. A saját rendszerünkön lehet, hogy nem pontosan ugyanazt az eredményt látjuk, mivel minden adatbázisrendszer, amely támogatja az SQL-t, saját optimalizálóval rendelkezik, amely kitalálja, hogyan lehet a leggyorsabban megoldani a lekérdezést. Ezen kívül az első néhány sor, amelyet az adatbázisrendszerünk visszaad, nem feltétlenül egyezik meg pontosan azokkal a sorokkal, amelyeket a könyvben láthatunk, hacsak a lekérdezés nem tartalmaz egy ORDER BY záradékot, amely megköveteli, hogy a sorok egy adott sorrendben jelenjenek meg.

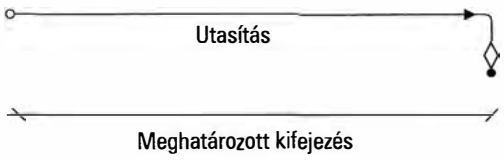
A legtöbb fejezet végén önállóan megoldandó feladatokat is találunk, amelyek lehetőséget adnak arra, hogy gyakoroljuk a fejezetben tanultakat. Ne aggódjunk: a megoldásokat a CD-n található mintaadatbázisok tartalmazzák, az olyan feladatokhoz pedig, amelyek egy kicsit nehezebbnek tűnnek, ötleteket is adunk.

Ha átrágtuk magunkat az egész könyvön, az „A" függelékben található teljes SQL-diagramok bizonyára felbecsülhetetlen értékű segítségnek bizonyulnak az ismertetett SQL-eljárások elsajátításához, a saját adatbázisaink megtervezéséhez pedig felhasználhatjuk a „B" függelék adatbázis-felépítési mintáit.

Hogyan értelmezzük a kötet diagramjait?

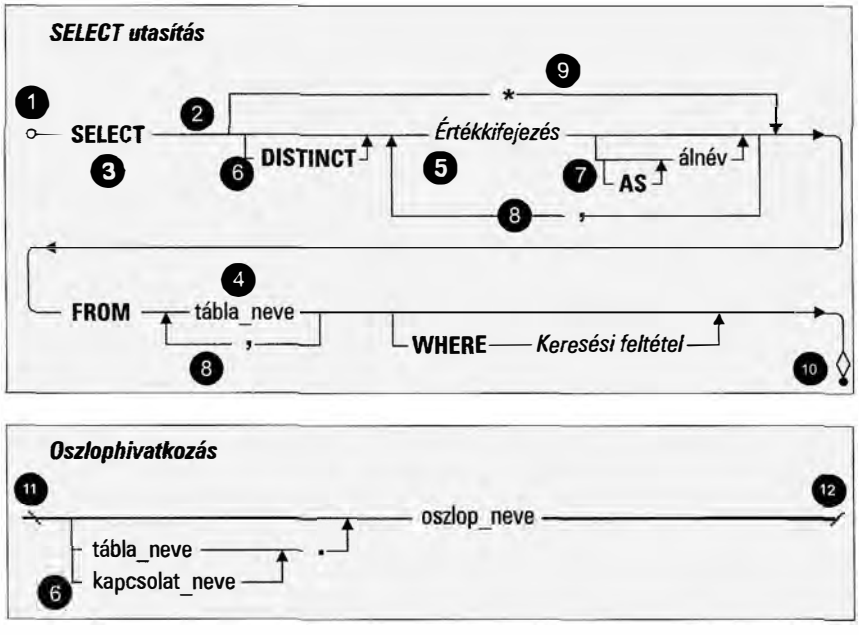
A könyvben szereplő számos diagram az SQL-ben használt egyes utasítások, kulcsszavak és kifejezések helyes használati formáját szemlélteti. Minden diagram világos képet ad a tárgyalt SQL-elem általános felépítéséről, de bármelyik diagramot használhatjuk sablonként is, hogy létrehozzuk a saját SQL-utasításainkat, vagy hogy tisztábban megértsünk egy adott példát.

A diagramok mindegyike alapelemek halmazából épül fel, és két csoport egyikébe sorolható: *utasítások* és *meghatározott kifejezések*. Az utasítások az SQL-nek a könyvben tárgyalt olyan fő műveletei, mint a SELECT utasítás, míg a meghatározott kifejezések olyan elemek, amelyekből egy utasítás egy részét építjük fel – ilyenek például az *értékkifejezések*, a *keresési feltételek*, illetve a *feltételes kifejezések*. (Ne aggódjunk – ezeket a szakkifejezéseket később kivétel nélkül elmagyarázzuk.) Egy utasítás és egy meghatározott kifejezés szintaxisdiagramja között az egyetlen különbség az, ahogy az utasításforma fő sora kezdődik, illetve végződik. A diagramokat úgy terveztük meg, hogy ezek a különbségek világosan mutassák, hogy egy teljes utasítás vagy egy utasításon belül használható kifejezés diagramját látjuk-e. Az 1. ábra mindkét diagramcsoport kezdő- és végpontját szemlélteti. Ettől a különbségtől eltekintve a diagramok ugyanazokból az elemekből épülnek fel. A 2. ábrán a szintaxisdiagramok mindkét típusára láthatunk egy-egy példát, amelyet a diagramok egyes elemeinek rövid magyarázata követ.



1. ábra

A szintaxisdiagram sorainak végpontja az utasítások és a meghatározott kifejezések esetében



2. ábra

Példák az utasítások és a meghatározott kifejezések diagramjaira

1. **Utasítás kezdőpontja** – Egy utasítás fő sorának kezdetét jelöli. Bármely elem, amely *közvetlenül* a fő sorban található, *kötelező elem*, az *alatta* található elemek pedig *nem kötelező (választható) elemek.*
2. **Az utasításforma fő sora** – Az utasítás vagy meghatározott kifejezés kötelező és választható elemeinek sorrendjét határozza meg. Az utasítás vagy meghatározott kifejezés felépítéséhez ezt a sort balról jobbra (vagy a nyilak irányában) kell követnünk.

3. **Kulcsszavak** – Egy utasítás vagy meghatározott kifejezés utasításformájának kötelező nyelvi elemét jelöli, amely az SQL nyelvtanának egyik kitüntetett szava. A diagramokban a kulcsszavak félkövér nagybetűkkel formázottak. (Amikor ténylegesen megírunk egy utasítást az adatbázis-kezelő programunkban, nem muszáj nagybetűvel írnunk a kulcsszavakat, de az utasítás is könnyebben olvasható lesz.)
4. **Konkrét elem** – Az utasításnak konkrétan megadandó érték nevét határozza meg. A konkrét (literális) elemeket egy olyan szó vagy kifejezés ábrázolja, amely az átadandó érték típusára utal. A diagramok konkrét elemei csupa kisbetűvel formázottak.
5. **Meghatározott kifejezés** – Olyan szót vagy kifejezést jelöl, amely egy olyan műveletet ábrázol, amely az adott utasításban felhasználandó értéket ad vissza. A könyv során minden olyan meghatározott kifejezést elmagyarázunk és diagramon ábrázolunk, amelyet ismernünk kell. A meghatározott kifejezéseket mindig dőlt betűvel jelöljük.
6. **Nem kötelező elem** – Olyan elemet vagy elemcsoportot jelöl, amely az utasításforma fő sora alatt jelenik meg. A nem kötelező elemek lehetnek utasítások, kulcsszavak, meghatározott kifejezések és konkrét értékek is, és a tisztánlátás kedvéért külön sorban szerepelnek. Egyes esetekben egy adott paraméterben értékek halmazát is átadhatjuk, az egyes értékeket vesszővel elválasztva (lásd a 8. pontot). Számos nem kötelező elemnek lehetnek részelemei is (lásd a 7. pontot). A nem kötelező elemek sorát általában balról jobbra kell olvasnunk, ugyanúgy, ahogy az utasításforma fő sorát olvassuk. Ha mindig a nyilakat követjük, nem tévedhetünk. Megjegyzendő, hogy egyes paraméterek több érték megadását is lehetővé teszik, ezért a nyíl jobbról balra mutathat. Miután azonban beírtuk az összes elemet, amire szükségünk van, a haladási irány visszatér a szokásos balról jobbra olvasáshoz. Szerencsére a nem kötelező elemek mindegyike ugyanúgy működik, így miután megmutattuk, hogy miként használhatunk egy választható elemet, tudni fogjuk az összes olyan nem kötelező elem használatának a módját, amellyel a szintaxisdiagramokban találkozhatunk.
7. **Nem kötelező elem részeleme** – Olyan elemet vagy elemcsoportot jelöl, amely egy nem kötelező elem alatt jelenik meg. A nem kötelező elemek részelemeinek segítségével finomhangolhatjuk az utasításainkat, hogy összetettebb feladatokat oldjunk meg velük.
8. **Paraméterlista-elválasztó** – Azt jelzi, hogy az adott paraméterben egynél több értéket is megadhatunk, mely értékeket vesszővel kell elválasztanunk.
9. **Alternatív paraméter** – Olyan kulcsszót vagy meghatározott kifejezést jelöl, amelyet egy vagy több nem kötelező elem helyett használhatunk. Az utasításformában az alternatív paraméterek sorai megkerülik azoknak a nem kötelező elemeknek a sorait, amelyekkel ezek a paraméterek felcserélhetők.
10. **Utasítás végpontja** – Egy utasítás fő sorának végét jelöli.
11. **Meghatározott kifejezés kezdőpontja** – Egy meghatározott kifejezés fő sorának kezdetét jelöli.
12. **Meghatározott kifejezés végpontja** – Egy meghatározott kifejezés fő sorának végét jelöli.

Most, hogy megismertük ezeket az elemeket, a könyv minden szintaxisdiagramját képesek leszünk elolvasni. Ha egy diagram mégis további magyarázatot igényelne, minden információt megadunk, ami ahhoz szükséges, hogy a diagramot könnyen és világosan értelmezhessük. Hogy könnyebben megértsük a diagramok működését, lássunk egy példát egy olyan SELECT utasításra, amelyet a 2. ábra alapján építettünk fel:

```
SELECT FirstName, LastName, City, DOB AS DateOfBirth
FROM Students
WHERE City = 'El Paso'
```

Ez a SELECT utasítás négy oszlopot ad vissza a Students (Tanulók) táblából, ahogy ezt a SELECT és FROM záradékokban jelöltük. Ha balról jobbra követjük az utasításforma fő sorát, láthatjuk, hogy legalább egy *értékkifejezést* meg kell adnunk. Az értékkifejezés lehet egy oszlop neve, egy oszlopnevek használatával megalkotott kifejezés, vagy egyszerűen egy (konkrét) állandó érték, amelyet meg szeretnénk jeleníteni. Az értékkifejezés paraméterlista-elválasztójával (vesszőkkel) annyi oszlopot adhatunk meg, amennyit csak akarunk, ezért tudtunk négy oszlopot kijelölni a Student táblából. Mivel nem voltunk benne biztosak, hogy mindenki, aki a fenti SELECT utasítás által visszaadott adatokat megtekinti, tudja, hogy mit jelent a DOB, egy *álnevet* rendeltünk a DOB oszlophoz, az értékkifejezést nem kötelező AS részelemként megadva. Végül a WHERE záradékot használtuk, hogy biztosítsuk, hogy a SELECT utasítás csak azokat a tanulókat adja vissza, akik El Pasóban laknak. (Ha ez most még nem világos, nem kell aggódnunk. Mindent megtanulunk majd részletesen a könyv hátralevő részében)

Az „A" függelékben minden szükséges szintaxisdiagramot megtalálunk, amelyek megmutatják a könyvben tárgyalt valamennyi utasítás és meghatározott kifejezés teljes és helyes utasításformáját.

Ha az egyes fejezetek olvasása közben ezeket a diagramokat lapozzuk fel, egyes diagramoknál némi eltérést tapasztalhatunk a fejezetben és a függelékben levő változat között. Ennek az az oka, hogy a fejezetekben szereplő diagramok a függelék diagramjainak egyszerűsített változatai. Az egyszerűsített diagramok segítenek, hogy egyszerűbben magyarázzuk el a bonyolultabb utasításokat és meghatározott kifejezéseket, és lehetőséget adnak arra, hogy az éppen tárgyalt elemekre összpontosítsunk. Emiatt nem kell, hogy fájjon a fejünk – a függelék minden diagramját tökéletesen érteni fogjuk, ha végigrágtuk magunkat a könyv anyagán.

A könyvben használt mintaadatbázisok

A könyvhöz egy CD-ROM-ot is mellékeltünk, amely öt mintaadatbázist tartalmaz, amelyeket a könyv során bemutatott lekérdezésekhez használunk. A „B" függelék diagramjain emellett ezeknek az adatbázisoknak a felépítését is ábrázoltuk.

1. **Sales Orders (Megrendelések)** – Ez egy tipikus, megrendelések bejegyzéseit tartalmazó adatbázis egy kerékpárokat és kerékpár-alkatrészeket árusító üzlet számára. (Minden adatbázisokkal foglalkozó könyvben kell lennie legalább *egy* megrendelési adatbázisnak, nem igaz?)
2. **Entertainment Agency (Szórakoztatóügynökség)** – Ezt az adatbázist úgy szerkesztettük meg, hogy előadóművészek, ügynökök, ügyfelek és lekötött előadások rekordjait kezelje. Hasonlóan terveznénk meg egy rendezvény- vagy szállodaszoba-foglalási adatbázist is.
3. **School Scheduling (Iskolai nyilvántartás)** – Ezt az adatbázis-felépítést közép- vagy főiskolai hallgatók nyilvántartásához használhatjuk. Az adatbázis nem csak az órákat tartja nyilván, hanem azt is, hogy az egyes órákat mely oktatók tartják, és hogy a tanulók milyen osztályzatokat kaptak.
4. **Bowling League (Tekebajnokság)** – Ez az adatbázis tekecsapatok nyilvántartására szolgál, és a csapatokat, azok tagjait, a lejátszott meccseket és az eredményeket tárolja.
5. **Recipes (Receptek)** – Ezt az adatbázist a kedvenc receptjeink tárolására és szerkesztésére használhatjuk. Néhány olyan receptet is találhatunk benne, amit érdemes kipróbálni.

A CD-n mind az öt mintaadatbázis megtalálható négy különböző formátumban:

- A Microsoft Office Access nevű asztali adatbázis-kezelőjének nagy népszerűsége miatt az adatbázisokat elkészítettük a Microsoft Access 2000 (Version 9.0, *.mdb* fálkiterjesztés) segítségével is. Azért a termék 9-es változatát választottuk, mert az majdnem teljesen támogatja a jelenlegi ISO/IEC SQL-szabványt, és az ebben a formátumban mentett adatbázisfájlokat az Access 2000, 2002 (XP), 2003 és 2007 segítségével is megnyithatjuk. Az említett fájlokat az MSAccess alkönyvtárban találhatjuk.
- A második formátum adatbázisfájljai a Microsoft SQL Server 2000 használatával készültek (*.mdf* fájlkiterjesztés). Ezek mellett SQL-parancsfájlokat (*.sql* fájlkiterjesztés) és kötegfájlokat (*.bat* fájlkiterjesztés) is mellékeltünk, amelyek segítségével a mintákat felvehetjük egy Microsoft SQL Server katalógusba. Az említett fájlok, amelyek az MSSQLServer alkönyvtárban találhatók, Microsoft SQL Server 2005 kiszolgálókhoz is kapcsolhatók. A Microsoft SQL Server 2005 Express Edition ingyenesen letölthető a *http://msdn.microsoft.com/vstudio/express/sql/download/default.aspx* címről.
- Az adatbázisfájlok harmadik csoportját a népszerű nyílt forrású MySQL adatbázisrendszer 5-ös változatával készítettük el. Az adatbázis szerkezetének létrehozásához, az adatok betöltéséhez és a mintanézetek létrehozásához a saját MySQL-adatkönyvtárunkban a MySQL alkönyvtárban található parancsfájlokat (*.sql* fájlkiter-

jesztés) használhatjuk, de az InnoDB adatkönyvtárunkat is beállíthatjuk úgy, hogy a MySQL alkönyvtárra mutasson. A MySQL adatbázisrendszer közösségi kiadásának egy példányát szabadon letölthetjük a *http://www.mysql.com/* címről.

- A negyedik formátum SQL-parancsfájlok sorozatából áll, amelyeket módosíthatunk, és bármely olyan fontosabb adatbázisrendszerrel használhatunk, amelyik ismeri az SQL-t. Az SQLScripts alkönyvtárban levő parancsfájlok között találunk olyat, amellyel meghatározhatjuk az egyes adatbázisok sémáit (a táblákat), olyat, amelyben INSERT utasításokkal betölthetjük az adatokat, és olyat is, amelyben CREATE VIEW utasításokkal létrehozhatjuk a lekérdezéseket. Bár ezeket a parancsfájlokat a Microsoft SQL Server segédeszközeivel készítettük, egyszerűsítettük őket, hogy általánosan használhatók legyenek a legtöbb adatbázisrendszerrel.

A mintafájlok telepítéséhez olvassuk el a CD gyökérkönyvtárában található *ReadMe.txt* fájlt. Ha a CD-mellékletet Apple Macintosh rendszerhez csatlakoztatjuk, csak a MySQL és az SQL-parancsállományok mintafájljait érhetjük el.

Megjegyzés *Bár a CREATE TABLE, CREATE INDEX, CREATE CONSTRAINT és INSERT parancsoknak nagyon gondosan csak a legelterjedtebb és legegyszerűbb formáját használtuk a minta SQL-parancsfájlokban, lehet, hogy nekünk vagy az adatbázisunk felügyelőjének kissé módosítania kell a fájlokat, hogy működjenek a mi adatbázisrendszerünkön is. Ha az adatbázisrendszert egy távoli kiszolgálón keresztül használjuk, előfordulhat, hogy az adatgazda engedélyére lesz szükségünk ahhoz, hogy az általunk megadott SQL-parancsokkal felépítsük a mintaadatbázisokat.*

A II., III. és IV. résznek a SELECT utasítással foglalkozó fejezeteihez tartozó példautasításokat és megoldásokat az egyes mintaadatbázisok „example" változatában (például: SalesOrdersExample, EntertainmentAgencyExample) találjuk. Mivel az V. rész példái módosítják a mintaadatokat, minden mintadatbázisból elkészítettünk egy „modify" változatot is (például: SalesOrdersModify, EntertainmentAgencyModify). Az V. részhez tartozó mintaadatbázisok olyan további oszlopokat és táblákat tartalmaznak, amelyek nem szerepelnek a SELECT példákban, ami lehetővé teszi, hogy szemléltessük az UPDATE, INSERT és DELETE lekérdezések képességeit.

„Mindig a Sárgaköves Úton!"

– A mumpicok királya Dorothy-nak az Óz, a nagy varázsló-ban

Most, hogy végigolvastuk a Bevezetést, készen állunk arra, hogy megtanuljuk az SQL használatát, nem igaz? Nos, talán. Jelenleg még mindig a házban vagyunk, amelyet a tornádó ostromol, és még nem hagytuk el Kansast.

Mielőtt rögtön a 4. fejezetre ugranánk, ahol majd elkészítünk egy egyszerű lekérdezést, fogadjuk meg a szerzők tanácsát, és olvassuk végig az első három fejezetet. Az 1. fejezetből képet kaphatunk arról, hogyan született meg a relációs adatbázisok elve, és hogyan vált belőlük az adatbázisoknak az iparágban ma a legszélesebb körben használt típusa. Reméljük, hogy ez megvilágít valamit azzal az adatbázisrendszerrel kapcsolatban is, amelyet éppen használunk. A 2. fejezetben megtanuljuk, hogyan finomhangolhatjuk az adatszerkezeteinket, hogy az adataink megbízhatóak és mindenekelőtt pontosak legyenek. Ha az adatszerkezeteinket rosszul terveztük meg, komoly nehézségeink lehetnek egyes SQL-utasításokkal, ezért mindenképpen ajánlott alaposan elmélyedni ebben a fejezetben.

Valójában a 3. fejezet a „Sárgaköves Út" kezdete. Itt ismerjük meg az SQL eredetét, és hogy miként nyerte el a nyelv a jelenlegi formáját. Azokról az emberekről és cégekről is olvashatunk majd, akik úttörő munkájukkal segítették a nyelv fejlesztését, és azt is megtudhatjuk, hogy miért van olyan sok változata az SQL-nek. Végül elmondjuk, hogyan vált az SQL országos és nemzetközi szabvánnyá, és arról is szót ejtünk, hogy milyen kilátásai vannak az SQL-nek a jövőben.

Miután elolvastuk az említett fejezeteket, már jó úton fogunk járni Óz birodalma felé. Csak kövessük a hátralevő fejezetekben lefektetett utat, és amikor a végére érünk a könyvnek, rájövünk majd, hogy megtaláltuk a varázslót – aki nem más, mint mi magunk.

I

A relációs adatbázisok és az SQL

1

Mit jelent az, hogy „relációs"?

„A tudás egy kis része annak a tudatlanságnak,
amit rendszerezünk és osztályozunk."
– Ambrose Bierce

A fejezet témakörei

- Az adatbázisok fajtái
- A relációs modell rövid története
- A relációs adatbázisok felépítése
- Miért hasznosak számunkra a relációs adatbázisok?
- Összefoglalás

Mielőtt fejest ugranánk az SQL világába, nem árt, ha rendelkezünk némi háttérinformációval a relációs adatbázisokról, ezért először arról fogunk beszélni, hogy miért találták ki a relációs adatbázisokat, hogyan épülnek fel, és miért hasznosak a számunkra. Ez az az alap, amelyen biztosan kell állnunk, hogy megérthessük az SQL lényegét, és tisztán lássuk, hogy miként aknázhatjuk ki legjobban az SQL képességeit.

Az adatbázisok fajtái

Mit nevezünk adatbázisnak? Amint azt bizonyára tudjuk, az adatbázis adatok rendezett gyűjteménye, amellyel valamilyen szervezett dolgot vagy szervezési folyamatot modellezünk. Tulajdonképpen nem számít, hogy papírt vagy egy számítógépes programot használunk az adatok összegyűjtésére és tárolására: amíg az adatokat konkrét célra, valamilyen rendezett formában gyűjtjük és tároljuk, adatbázisról beszélhetünk. A továbbiakban persze azt fogjuk feltételezni, hogy az adatgyűjtésre és az adatok kezelésére számítógépprogramot használunk.

Az adatbázis-kezelés területén általánosságban kétféle adatbázis van használatban: *műveleti adatbázisok* és *elemző adatbázisok*. Ma szerte a világon műveleti adatbázisok képezik számos vállalat, szervezet és intézmény gerincét. Ezt a fajta adatbázist elsősorban mindennapi adatgyűjtésre, -módosításra és -kezelésre használják. A tárolt adatok *dinami-*

kus adatok, ami azt jelenti, hogy folyamatosan változnak, és mindig friss információkat tükröznek. Az olyan szervezetek, mint az áruházak, a gyártócégek, a kórházak és klinikák, valamint a könyvkiadók műveleti adatbázisokat használnak, mivel az adataik percről percre változnak.

Ezzel szemben az elemző adatbázisok korábbról származó, adott időponthoz kapcsolódó adatokat tárolnak és rendszereznek. Az elemző adatbázisok a változások irányának nyomon követésében hasznosak, valamint a hosszú idő alatt összegyűjtött statisztikai adatok megjelenítését, illetve a taktikai és stratégiai üzleti előrejelzések készítését segítik. Az ilyen adatbázisok *statikus adatokat* tárolnak, vagyis az adatok soha (vagy csak nagyon ritkán) módosulnak, új adatok felvételére viszont gyakran sor kerülhet. Az elemző adatbázisokból kinyert információk általában nem naprakészek: egy adott időpontban készített pillanatfelvételt mutatnak az adatokról. Ilyen fajta adatbázist használnak például a kémiai laboratóriumok, a földmérő vállalatok, illetve a piackutató és -elemző cégek.

A relációs modell rövid története

Az adatbázismodelleknek többféle típusa létezik. Egyeseket – például a hierarchikus és hálózati modelleket – csak régi rendszereken használnak, míg mások – például a relációs modell – széles körben elterjedtek. Más könyvekben az objektum alapú, objektumrelációs és OLAP (online analytical processing, elektronikus elemző-feldolgozó) modellekkel is találkozhatunk. Az SQL-szabvány bővítményeket határoz meg ezeknek a modelleknek a támogatására, és léteznek kereskedelmi adatbázisrendszerek, amelyek meg is valósítanak egyes bővítményeket ezek közül. Mi azonban szigorúan a relációs modellre és a nemzetközi SQL-szabvány magjára fogunk összpontosítani.

A kezdetek

A relációs adatbázis elve 1969-ben született meg, és azóta valószínűleg a legszélesebb körben használt modellé vált az adatbázis-kezelésben. A relációs modell atyja, Dr. Edgar F. Codd (1923–2003) az IBM kutatójaként dolgozott az 1960-as években, és azt vizsgálta, hogy milyen új módszerekkel lehet nagy adatmennyiségeket kezelni. Az akkor létező adatbázismodellekkel és -termékekkel elégedetlen volt, ami arra indította, hogy matematikai elvek és szerkezetek segítségével próbálja meg megoldani az előtte álló tömérdek feladatot. Matematikusként szilárdan hitt benne, hogy a matematikának léteznek olyan ágai, amelyeket alkalmazva megoldhatók az olyan problémák, mint az adatismétlődés (redundancia) kiküszöbölése, az adatok egységességének biztosítása, illetve annak leküzdése, hogy az adatbázisok szerkezete túlzottan függjön a fizikai megvalósítástól.

Dr. Codd hivatalosan 1970 júniusában, az *A Relational Model of Data for Large Shared Databases* című, mérföldkőnek számító munkájában[1] mutatta be az általa kidolgozott új relációs modellt. A modell a matematika két ágára támaszkodott: a halmazelméletre és

[1] *Communications of the ACM*, 1970. június, 377-87. oldal

az elsőrendű predikátumlogikára. Maga a modell is a halmazelmélet egyik kulcsfogalma, a *reláció* kifejezés után kapta a nevét. (Széles körben elterjedt tévedés, hogy a relációs modell neve onnan ered, hogy a relációs adatbázisok táblái között kapcsolatok állhatnak fenn. Most, hogy tudjuk az igazságot, nyugodtan alhatunk.) Szerencsére nem kell részleteiben ismernünk a halmazelméletet vagy az elsőrendű predikátumlogikát ahhoz, hogy relációs adatbázisokat tervezhessünk és használhassunk. Ha megfelelő adatbázis-tervezési módszert követünk – például a Mike Hernandez *Database Design for Mere Mortals* (Addison-Wesley, 2004; magyarul: *Adatbázis-tervezés: A relációs adatbázisok alapjairól földi halandóknak*. Kiskapu, 2004) című könyvében ismertetett eljárásokat –, egészséges és hatékony adatbázis-szerkezetet alakíthatunk ki, amelyet bizalommal használhatunk mindenféle adat összegyűjtésére és kezelésére. (Nos rendben, természetesen egy kicsit azért *muszáj* értenünk a predikátumokhoz és a halmazelmélethez, ha bonyolultabb feladatokat is meg szeretnénk oldani. A predikátumokkal – állításokkal vagy szűrőkkel; a „predikátum" valójában csak egy fellengzős név – kapcsolatos alapvető tudnivalókat a 6., a halmazelmélet alapjait pedig a 7. fejezetben tárgyaljuk.)

A relációs adatbázisprogramok

Megjelenése óta a relációs modell az alapja az RDBMS (relational database management system, relációs adatbázis-kezelő rendszer) néven ismert adatbázistermékeknek. Ilyen terméket számos gyártó készít, és az évek során a legkülönbözőbb iparágak és szervezetek vették át a használatát, akik sokféle környezetben alkalmazzák. Az 1970-es években a nagyszámítógépek olyan programokat használtak, mint az IBM által kifejlesztett *System R* vagy a Berkeley-i Kaliforniai Egyetemen megalkotott *INGRES*. A nagygépekhez készített RDMBS rendszerek fejlesztése az 1980-as években olyan programokkal folytatódott, mint az Oracle Corporation *Oracle*, illetve az IBM *DB2* rendszere, a személyi számítógépeknek az 1980-as évek közepén bekövetkezett elterjedése pedig olyan programokat szült, mint az Ashton Tate *dBase*, az Ansa Software *Paradox* vagy a Microrim *R:BASE* szoftvere. Amikor az 1980-as évek végén, illetve az 1990-es évek elején nyilvánvalóvá vált, hogy igény mutatkozik a PC-k közötti adatmegosztásra, megszületett az ügyfél–kiszolgáló rendszerek elve, amelyet a központi helyen tárolt, közösen használható, könnyen kezelhető és biztonságossá tehető adatok ötlete kísért. Erre az elvre olyan termékek épültek, mint az Oracle *Oracle 8i*, valamint a Microsoft *SQL Server* rendszere. Körülbelül 1996 óta már abban az irányban is összehangolt erőfeszítéseket tesznek, hogy az adatbázisok elérhetőségét az Interneten is biztosítsák. A szoftvergyártók komolyan veszik a kérdést, és olyan termékekkel igyekeznek válaszolni a kihívásokra, amelyek webközpontúbbak: ilyen az Allaire cég *Cold Fusion*, a Sybase *Sybase Enterprise Application Studio* vagy a Microsoft *Visual Studio* programcsomagja. A webfejlesztésben az egyik legnépszerűbb adatbázis-kezelő a MySQL AB nyílt forrású *MySQL*-je. Ezt eredetileg Linux rendszerű webkiszolgálókhoz tervezték, de már Microsoft Windows rendszereken futó változata is létezik.

A relációs adatbázisok felépítése

A relációs modellnek megfelelően az adatok a relációs adatbázisokban *relációkban* tárolódnak, amelyeket a felhasználó táblák formájában érzékel. Minden reláció *egyedekből* (tuple, rekord) és *jellemzőkből* (mező) épül fel. Ezen kívül a relációs adatbázisoknak természetesen más tulajdonságaik is vannak – ezekkel foglalkozunk az alábbiakban.

Táblák

A táblák az adatbázisok fő szerkezeti elemei. Minden tábla mindig egyetlen konkrét tárgyat ábrázol. A rekordok és mezők logikai sorrendje a táblán belül egyáltalán nem számít. Legalább egy mezőt – az úgynevezett *elsődleges kulcsot*, amely egyedileg azonosítja a tábla egyes rekordjait– minden táblának tartalmaznia kell. (Az 1.1. ábrán például a CustomerID a Customers tábla elsődleges kulcsa.) Valójában a relációs adatbázisokban levő adatok a táblák ez utóbbi két tulajdonságának köszönhetően függetlenek attól, hogy fizikailag miként tárolódnak a számítógépen. Ez a felhasználóknak előnyös, mert így ahhoz, hogy kinyerjenek egy adatot, nem kell ismerniük az adatot tároló rekord fizikai helyét.

Customers

Customers	FirstName	LastName	StreetAddress	City	State	ZipCode
1010	Angel	Kennedy	667 Red River Road	Austin	TX	78710
1011	Alaina	Hallmark	Route 2, Box 203B	Woodinville	WA	98072
1012	Liz	Keyser	13920 S.E. 40th Street	Bellevue	WA	98006
1013	Rachel	Patterson	2114 Longview Lane	San Diego	CA	92199
1014	Sam	Abolrous	611 Alpine Drive	Palm Springs	CA	92263
1015	Darren	Gehring	2601 Seaview Lane	Chico	CA	95926

REKORDOK

MEZŐK

1.1. ábra

Példa egy táblára

A tárgy, amelyet egy adott tábla ábrázol, egy *objektum* vagy egy *esemény* lehet. Ha a tárgy egy objektum, a tábla valami olyat ábrázol, ami „megfogható": egy személyt, egy helyet vagy más fizikai dolgot. Objektumokként táblában ábrázolhatók például pilóták, termékek, gépek, hallgatók, épületek, berendezések, és így tovább. A típusától függetlenül minden objektumnak vannak olyan jellemzői, amelyek adatként tárolhatók, és ezeket az adatokat aztán szinte végtelen módon lehet feldolgozni. Erre a fajta táblára az 1.1. ábra mutat jellegzetes példát.

Amennyiben a tábla tárgya egy esemény, a tábla valami olyasmit ábrázol, ami egy adott időpontban történt, illetve történik (majd), és olyan jellemzői vannak, amelyeket rögzíteni szeretnénk. Ezeket a jellemzőket pontosan ugyanúgy tárolhatjuk adatként, és dolgozhatjuk fel mint információt, mint azokban a táblákban, amelyek valamilyen konkrét objektumot

írnak le. Olyan eseményeket rögzíthetünk például, mint a bírósági tárgyalások, az osztalékkifizetés, a laborleletek vagy egy földtani mérés. Az 1.2. ábrán például egy olyan eseményt ábrázoló táblát láthatunk, amelyet életünk során mindnyájan átélünk – egy orvosi vizetet.

Patient Visit

PatientID	VisitDate	VisitTime	Physician	BloodPressure	Temperature
92001	2006-05-01	10:30	Ehrlich	120 / 80	98.8
97002	2006-05-01	13:00	Hallmark	112 / 74	97.5
99014	2006-05-02	9:30	Fournier	120 / 80	98.8
96105	2006-05-02	11:00	Hallmark	160 / 90	99.1
96203	2006-05-02	14:00	Hallmark	110 / 75	99.3
98003	2006-05-02	9:30	Fournier	120 / 82	98.6

1.2. ábra

Eseményt ábrázoló tábla

Mezők

A mező az adatbázis legkisebb szerkezeti egysége, amely a hozzá tartozó tábla tárgyának egy jellemzőjét írja le. A mezők azok a szerkezetek, amelyek az adatokat ténylegesen tárolják. A mezőkben levő adatokat azután kinyerhetjük és információként megjeleníthetjük, szinte bármilyen elképzelhető formában. Véssük az eszünkbe, hogy az adatokból kinyert információk minősége egyenesen arányos az idő mennyiségével, amit arra fordítunk, hogy maguknak a mezőknek az adat- és szerkezeti következetességét biztosítsuk. A mezők fontosságát nem lehet túlbecsülni.

Egy megfelelően megtervezett adatbázisban minden mező kizárólag egyetlen értéket tartalmaz, és a neve azonosítja a benne tárolt érték típusát. Így nagyon egyszerűvé válik az adatok bevitele a mezőkbe: ha olyan nevű mezőket látunk, mint a FirstName (Keresztnév), a LastName (Vezetéknév), a City (Város), a State (Állam/Megye) vagy a ZipCode (Irányítószám), pontosan tudni fogjuk, hogy milyen értéket kell írnunk az egyes mezőkbe, és az adatok rendezése (például állam szerint) vagy az összes olyan személy kikeresése, akinek a vezetékneve Viescas, ugyancsak könnyű lesz.

Rekordok

A rekord egy adott tábla tárgyának egy egyedi példányát jelképezi, amely a tábla adott sorában található összes mezőt tartalmazza, függetlenül attól, hogy az egyes mezőkben szerepelnek-e értékek. A táblák meghatározásának módjából következik, hogy minden rekordot egy egyedi érték azonosít a teljes adatbázisban, és ez az érték a rekord elsődleges kulcs mezőjében tárolódik.

Az 1.1. ábrán például minden rekord egy-egy különböző vásárlót jelöl a táblában, és az adatbázisban az egyes vásárlókat a CustomerID (Vásárlóazonosító) mező azonosítja. Amint említettük, az egyes rekordok a tábla adott sorában található összes mezőt tartal-

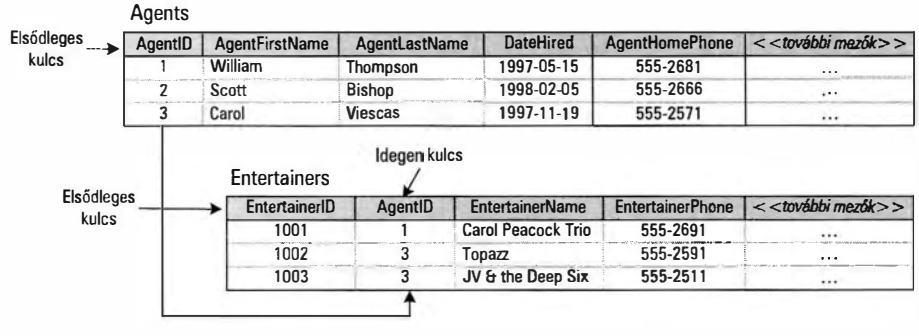
mazzák, és minden mező egy-egy tulajdonságát írja le a rekord által ábrázolt vásárlónak. A rekordok kulcsfontosságúak a táblakapcsolatok megértésében, mivel tudnunk kell, hogy egy tábla rekordjai milyen viszonyban állnak egy másik tábla rekordjaival.

Kulcsok

A kulcsok különleges mezők, amelyek meghatározott szerepet töltenek be az egyes táblákban. Ezt a szerepet a kulcs típusa határozza meg. Bár egy tábla többféle kulcsot tartalmazhat, mi csak a két legfontosabb típust tárgyaljuk: az *elsődleges kulcsot* és az *idegen kulcsot.*

Az elsődleges kulcs olyan mező vagy mezőcsoport, amely egyedileg azonosítja a táblában található összes rekordot. (Ha az elsődleges kulcs két vagy több mezőből áll, *összetett elsődleges kulcsról* beszélünk.) Két oka van annak, hogy miért az elsődleges kulcs a legfontosabb: az *értéke* egy *konkrét rekordot* azonosít a teljes adatbázisban, a *mezője* pedig ugyanott egy *adott táblát.* Az elsődleges kulcsok emellett a tábla szintjén biztosítják a következetességet, és segítenek kapcsolatokat kialakítani más táblákkal. Az adatbázisaink minden táblájának rendelkeznie kell elsődleges kulccsal.

Az 1.3. ábrán látható AgentID (Ügynökazonosító) mező jó példa az elsődleges kulcsra, mert egyedileg azonosítja az egyes ügynököket az Agents (Ügynökök) táblán belül, és a többször szereplő rekordok kiküszöbölésével gondoskodik a táblaszintű következetességről. A kulcsot ezenkívül arra is használjuk, hogy az Agents és az adatbázis más táblái (például – ahogy az ábrán láthatjuk – az Entertainers tábla) között kapcsolatokat alakítsunk ki.



1.3. ábra

Elsődleges és idegen kulcsok

Amikor eldöntjük, hogy két táblát valamilyen módon összekapcsolunk, a kapcsolatot általában úgy hozzuk létre, hogy az első tábla elsődleges kulcsát lemásoljuk és beszúrjuk a második táblába, ahol az idegen kulcs lesz. (Az *idegen kulcs* kifejezés onnan ered, hogy a második táblának már van saját elsődleges kulcsa, és az első táblából átvett elsődleges kulcs a második tábla számára „idegen".)

Az 1.3. ábrán egy jó példát láthatunk az idegen kulcsra. A példában az AgentID elsődleges kulcs az Agents táblában és idegen kulcs az Entertainers táblában. Amint láthatjuk, az Entertainers táblának már van egy elsődleges kulcsa – az EntertainerID. Ebben a kapcsolatban az AgentID az a mező, amely megteremti az összefüggést az Agents és az Entertainers tábla között.

Az idegen kulcsok nem csak abból a nyilvánvaló szempontból fontosak, hogy segítenek kapcsolatot teremteni két tábla között, hanem azért is, mert biztosítják a kapcsolatszintű következetességet, vagyis a két tábla rekordjai mindig helyes kapcsolatban fognak állni egymással, mivel az idegen kulcs mező értékeinek a hivatkozott elsődleges kulcs értékeiből *kell* származniuk. Az idegen kulcsok a rettegett „árva rekordokat" is segítenek elkerülni, amelyek klasszikus példája egy olyan rendelési rekord, amelyhez nem tartozik vásárló. Ha nem tudjuk, ki adta fel a rendelést, nem leszünk képesek feldolgozni azt, és természetesen kiszámlázni sem tudjuk, ami rontja a negyedéves eladási mutatóinkat.

Nézettáblák

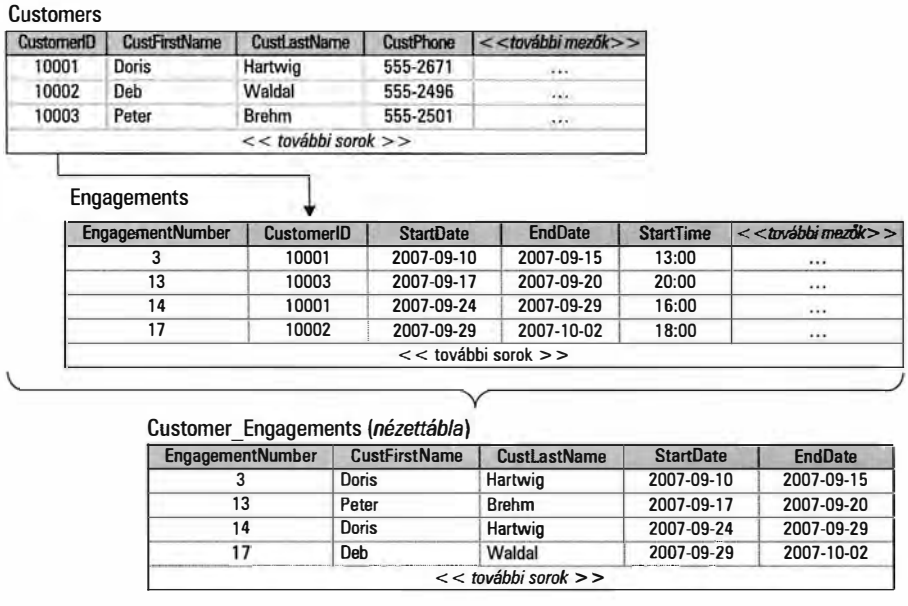
A nézet vagy nézettábla olyan virtuális tábla, amely az adatbázis egy vagy több táblájának mezőiből épül fel. A nézettáblát alkotó táblákat *alaptáblákna*k hívjuk. A relációs modell azért tekinti virtuálisnak (látszólagosnak) a nézettáblákat, mert azok adatai más táblákból származnak. Maguk a nézettáblák nem tárolnak adatokat; az adatbázisban valójában a szerkezetük az egyetlen információ, ami tárolódik róluk.

A nézettáblák lehetővé teszik, hogy az adatbázisban tárolt információkat számos különféle szempontból tekintsük meg, ami nagy rugalmasságot nyújt az adatok kezelésében. Nézettáblákat többféleképpen is létrehozhatunk – különösen akkor hasznosak, amikor több, egymással kapcsolatban álló táblára alapozzuk őket. Létrehozhatunk például egy olyan nézettáblát, amelyik olyan információkat összegez, mint a Seattle belvárosában dolgozó ácsok által ledolgozott órák teljes száma, vagy egy olyat, amelyik adott mezők szerint csoportosítja az adatokat. Ez utóbbi fajtára jó példa egy olyan nézettábla, amelyik egy adott terület államait veszi alapul, hogy megmutassa az egyes városokban az alkalmazottak teljes számát. A nézettáblák jellemző felépítését az 1.4. ábrán láthatjuk.

Sok RDBMS program a nézettáblákat ***mentett lekérdezésként*** valósítja meg, és ezen a néven vagy egyszerűen *lekérdezésként* hivatkozik rájuk. A lekérdezések a legtöbb esetben rendelkeznek a nézettáblák minden jellemzőjével, vagyis csupán a nevük az egyetlen különbség köztük. (Mi már gyakran morfondíroztunk rajta, hogy ehhez nem egy marketingcsapatnak van-e köze.) Fontos megjegyezni, hogy egyes gyártók már kezdik a valódi nevükön nevezni a lekérdezéseket, de mindegy, hogy a mi relációs adatbázis-kezelőnk hogy hívja őket, az adatbázisban valójában nézettáblákkal dolgozunk.

Könyvünk címe ettől függetlenül *SQL-lekérdezések földi halandóknak,* miközben elsősorban azt szeretné megtanítani, hogy miként építsük fel a nézettábláinkat. Ahogy a 2. fejezetből majd megtudhatjuk, egy relációs adatbázist akkor tervezünk meg helyesen, ha

az adatainkat úgy bontjuk fel, hogy minden tárgyhoz vagy eseményhez egyetlen tábla tartozzon. Ugyanakkor többnyire összefüggő tárgyakról vagy eseményekről szeretnénk információt szerezni – mely vásárlók és milyen rendelést adtak fel, mely tanárok milyen órákat tartanak, és így tovább –, ehhez pedig nézettáblát kell létrehoznunk, és tudnunk kell, hogy milyen SQL kóddal érhetjük ezt el.



1.4. ábra

Példa egy nézettáblára

Kapcsolatok

Ha egy adott tábla rekordjai valamilyen módon egy másik tábla rekordjaihoz társíthatók, azt mondjuk, hogy a táblák között kapcsolat áll fenn. A kapcsolat megteremtésének módja a kapcsolat típusától függ. Két tábla között háromféle kapcsolat állhat fenn: egy-az-egyhez, egy-a-többhöz (egy-a-sokhoz) vagy több-a-többhöz (sok-a-sokhoz) A kapcsolatok fajtáinak ismerete létfontosságú ahhoz, hogy megértsük, hogyan működnek a nézettáblák, és hogy miként kell a többtáblás SQL-lekérdezéseket megtervezni és használni. (Erről a III. részben tanulunk majd bővebben.)

Egy-az-egyhez

Két tábla akkor áll egy-az-egyhez kapcsolatban, ha az első tábla egy rekordja a második tábla *egyetlen* rekordjához kapcsolódik, és a második tábla egy rekordja is *egyetlen* rekorddal áll kapcsolatban az első táblából. Ebben a fajta kapcsolatban az egyik tábla *elsődleges tábla*, míg a másik *másodlagos tábla* lesz. A kapcsolatot úgy hozzuk létre, hogy vesszük

az elsődleges tábla elsődleges kulcsát, és beszúrjuk azt a másodlagos táblába, ahol idegen kulcs lesz belőle. Ez a kapcsolatok különleges fajtája, mivel sok esetben az idegen kulcs egyben a másodlagos tábla elsődleges kulcsaként is viselkedik.

Az egy-az-egyhez kapcsolatra az 1.5. ábrán láthatunk egy jellemző példát: itt az Agents (Ügynökök) az elsődleges, a Compensation (Díjazás) pedig a másodlagos tábla. A két tábla között olyan kapcsolat áll fenn, amelyben az Agents tábla egy-egy rekordja csak egy-egy rekordhoz kapcsolódik a Compensation táblában, és a Compensation tábla egy-egy rekordja is csak egy-egy rekordhoz társul az Agents táblából. Figyeljük meg, hogy az AgentID valóban elsődleges kulcs mindkét táblában, de egyben idegen kulcs a másodlagos táblában.

A kapcsolatban a vezető szerepet játszó elsődleges tábla kijelölése teljesen tetszőleges. Az egy-az-egyhez kapcsolatok nem túl gyakoriak; általában akkor találkozunk velük, ha egy táblát titkosítás végett két részre bontottak.

Agents

AgentID	AgentFir tName	AgentLastName	DateOfHire	AgentHomePhone	<<további mezők>>
1	William	Thompson	1997-05-15	555-2681	...
2	Scott	Bishop	1998-02-05	555-2666	...
3	Carol	Viescas	1997-11-19	555-2571	...

Compensation

Salary	Commis ionRate	<<további mezők>>
$35,000.00	4.00%	...
$27,000.00	4.00%	...
$30,000.00	5.00%	...

1.5. ábra

Példa egy-az-egyhez kapcsolatra

Egy-a-többhöz

Ha két tábla egy-a-többhöz kapcsolatban áll, akkor az első tábla egy rekordja a második tábla *több* rekordjához is kapcsolódhat, de a második tábla egy rekordja csak *egyetlen* rekorddal állhat kapcsolatban az első táblából. Ezt a kapcsolatot úgy hozzuk létre, hogy vesszük a kapcsolat „egy" oldalán álló tábla elsődleges kulcsát, és beszúrjuk azt a „több" oldalon levő táblába, ahol idegen kulcs lesz belőle.

Az egy-a-többhöz kapcsolatra az 1.6. ábrán láthatunk egy jellemző példát: itt az Entertainers (Előadók) tábla egy-egy rekordja *több* rekordhoz kapcsolódik az Engagements (Rendezvények) táblában, de az Engagements tábla egy-egy rekordja csak *egyetlen* rekordhoz társul az Entertainers táblából. Ahogy nyilván kitaláltuk, az Engagements tábla idegen kulcsa az EntertainerID.

Entertainers

EntertainerID	EntertainerName	EntertainerPhone	<<*további mezők*>>
1001	Carol Peacock Trio	555-2691	...
1002	Topazz	555-2591	...
1003	JV & the Deep Six	555-2511	...

Engagements

EngagementID	EntertainerID	CustomerID	StartDate	EndDate	<<további mezők>>
5	1003	10006	2007-09-11	2007-09-14	...
7	1002	10004	2007-09-11	2007-09-18	...
10	1003	10005	2007-09-17	2007-09-26	...
12	1001	10014	2007-09-18	2007-09-26	...

1.6. ábra

Példa egy-a-többhöz kapcsolatra

Több-a-többhöz

Két tábla akkor áll több-a-többhöz kapcsolatban, ha az első tábla egy rekordja a második tábla *több* rekordjához kapcsolódik, és a második tábla egy rekordja is *több* rekorddal áll kapcsolatban az első táblából. Ezt a kapcsolatot helyesen úgy hozzuk létre, hogy létrehozunk egy úgynevezett *kapcsoló táblát*. Ennek a táblának a segítségével egyszerűen társíthatunk rekordokat az egyik táblából a másik tábla rekordjaihoz, és a kapcsoló tábla arról is gondoskodik, hogy a kapcsolódó adatok hozzáadása, törlése vagy módosítása során semmilyen gond ne lépjen fel. A kapcsoló táblát úgy határozzuk meg, hogy lemásoljuk a kapcsolatban részt vevő mindkét tábla elsődleges kulcsát, és ezek segítségével alkotjuk meg az új tábla szerkezetét. A kulcsmezők két jól elkülöníthető szerepet töltenek be: együttesen a kapcsoló tábla összetett elsődleges kulcsát adják, külön-külön pedig idegen kulcsokként viselkednek.

Customers

CustomerID	CustFirstName	CustLastName	CustPhone	<<*további mezők*>>
10001	Doris	Hartwig	555-2671	...
10002	Deb	Waldal	555-2496	...
10003	Peter	Brehm	555-2501	...

Entertainers

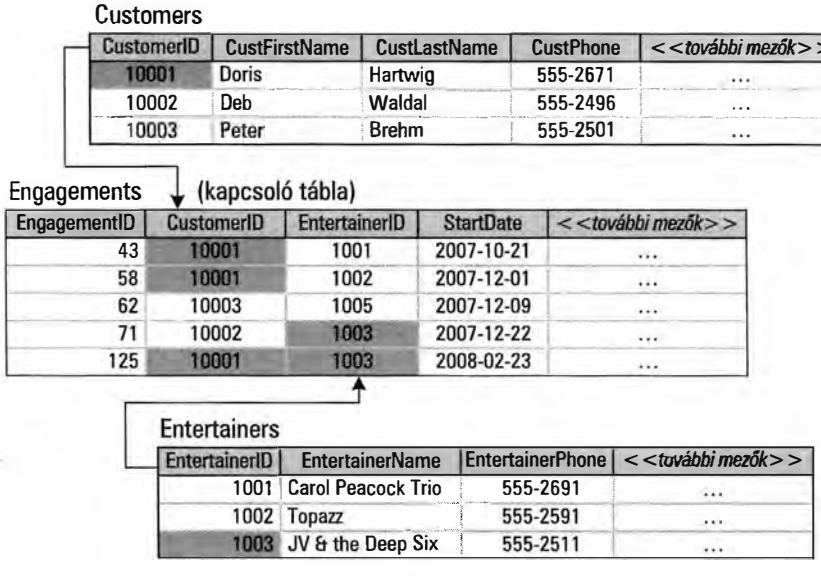
EntertainerID	EntertainerName	EntertainerPhone	<<*további mezők*>>
1001	Carol Peacock Trio	555-2691	...
1002	Topazz	555-2591	...
1003	JV & the Deep Six	555-2511	...

1.7. ábra

Feloldatlan több-a-többhöz kapcsolat

Ha egy több-a-többhöz kapcsolat nem megfelelően jött létre, *feloldatlan kapcsolatról* beszélünk. Erre az 1.7. ábrán láthatunk egy világos példát. Itt a Customers (Megrendelők) tábla egy-egy rekordja az Entertainers tábla *több* rekordjához kapcsolódhat, és az Entertainers tábla egy-egy rekordja is a Customers tábla *több* rekordjához.

A kapcsolat a több-a-többhöz viszonyban rejlő probléma miatt lesz feloldatlan. A kérdés ez: miként társíthatjuk egyszerűen az első tábla rekordjait a második tábla rekordjaihoz? Ha a kérdést az 1.7. ábrán látható táblák szerint fogalmazzuk újra, a probléma ez lesz: hogyan társíthatunk egy megrendelőt több előadóhoz vagy egy adott előadót több megrendelőhöz? (Ha egy szórakoztatóügynökséget vezetünk, bizonyára azt reméljük, hogy idővel minden megrendelő több előadót is leköt, illetve az egyes előadók fellépéseire is többen lesznek kíváncsiak.) Beszúrjuk pár megrendelő mezőjét az Entertainers táblába? Vagy adjunk néhány előadómezőt a Customers táblához? Mindkét megoldás számos problémát eredményez, amikor a kapcsolódó adatokkal dolgozunk, elsősorban az adatok következetessége terén. A problémára az a megoldás, hogy a korábban ismertetett módon létrehozunk egy kapcsoló táblát, amellyel megfelelően feloldhatjuk a több-a-többhöz kapcsolatot. A megoldás gyakorlati megvalósítását az 1.8. ábra mutatja.



1.8. ábra

Megfelelően feloldott több-a-többhöz kapcsolat

Az 1.8. ábrán a kapcsoló táblát úgy hoztuk létre, hogy vettük a CustomerID mezőt a Customers táblából, valamint az EntertainerID mezőt az Entertainers táblából, és ezekből alkottuk meg az új táblát. Az adatbázis más tábláihoz hasonlóan a kapcsoló táblának is saját ne-

ve van: Engagements (Rendezvények). Az Engagements tábla jó példa egy olyan táblára, amely egy eseményről tárol információkat. Láthatjuk például, hogy az 1003-as előadó (JV & the Deep Six) a 10001-es megrendelőnél (Doris Hartwig) lépett fel február 23-án. A kapcsoló tábla igazi előnye, hogy lehetővé teszi, hogy a kapcsolatban részt vevő mindkét táblából tetszőleges számú rekord között teremtsünk kapcsolatot. Ahogy a példa is mutatja, most már egyszerűen összekapcsolhatunk egy adott megrendelőt tetszőleges előadókkal vagy egy adott előadót akárhány megrendelővel.

Ahogy korábban mondtuk, a kapcsolatok megértése nagyon fontos ahhoz, hogy hatékony többtáblás SQL-lekérdezéseket írjunk, ezért amikor elérünk a könyv III. részéhez, érdemes visszalapozni ide, és felfrissíteni az emlékezetünket.

Miért hasznosak számunkra a relációs adatbázisok?

De miért is kell törődnünk azzal, hogy megértsük a relációs adatbázisokat? Miért érdekeljen minket, hogy milyen környezetben kezeljük az adatainkat? És egyáltalán: miért hasznosak számunkra a relációs adatbázisok? Nos, ideje, hogy megvilágosítsuk az elménket, és fejest ugorjunk a mélyvízbe!

Az idő, amit a relációs adatbázisok tanulmányozására fordítunk, olyan befektetés, ami egyértelműen megtérül. Aktív tudásra kell szert tennünk a relációs adatbázisokkal kapcsolatban, mivel ma ez a legszélesebb körben elterjedt adattárolási modell. Felejtsük el, amit korábban olvastunk a témáról vagy amit Huba az informatikai részlegről mondott nekünk – a cégek és különféle szervezetek által gyűjtött és kezelt adatok túlnyomó többségét relációs adatbázisokban tárolják. A modellnek valóban léteznek bővítései, a relációs adatbázisokat kezelő programok ma már objektumközpontú megoldásokat is alkalmaznak, és az ilyen adatbázisok mára teljesen beépültek a Világháló szövetébe, de nem számít, hogy csűrjük-csavarjuk, aprítjuk és fűszerezzük, ezek az adatbázisok ettől még relációs adatbázisok maradnak. A relációs modell 35 éve velünk él, a szerepe csak egyre erősödik, és belátható időn belül nem lesz, ami felváltsa.

Szinte minden ma használt kereskedelmi adatbázis-kezelő szoftver relációs alapokon nyugszik (bár Dr. Codd, C.J. Date és Fabian Pascal komolyan megkérdőjeleznék, hogy bármelyik kereskedelmi megvalósítás valóban relációs-e!). Ha adatbázisokkal dolgozunk, és sikert szeretnénk elérni, tudnunk kell, hogyan kell megtervezni egy relációs adatbázist, és hogyan kell azt megvalósítani valamelyik népszerű RDBMS programban – ráadásul azóta, hogy oly sok cég folytat üzleti tevékenységet az Interneten, némi webfejlesztési tapasztalat sem árt.

A relációs adatbázisok gyakorlati ismerete több szempontból is a segítségünkre lehet. Minél többet tudunk például a relációs adatbázisok tervezéséről, annál könnyebben tudunk majd egy adott adatbázishoz felhasználói alkalmazásokat fejleszteni. Meg fogunk lepődni, milyen

egyszerűvé válik az RDBMS programunk használata – mivel értjük, hogy mire szolgálnak az egyes eszközei, és hogyan használhatjuk azokat a leghatékonyabban. Gyakorlati ismereteinknek akkor is jó hasznát vesszük, amikor az SQL használatát igyekszünk elsajátítani, hiszen az SQL a relációs adatbázisok létrehozásának és kezelésének szabványos nyelve.

Hogyan tovább?

Már értjük, hogy miért fontos megismerkednünk a relációs adatbázisokkal, de azt is látnunk kell, hogy az *adatbázisok elmélete* és az *adatbázisok tervezése* két különböző dolog. Az adatbázisok elméletét azok az elvek és szabályok alkotják, amelyeken a relációs adatbázismodell alapul: ez az, amit az iskolák szentélyeiben oktatnak, és amit a valóság sötét bugyraiban gyorsan el kell hajítanunk. Az elmélet persze fontos, mert nélküle nem készíthetnénk szerkezetileg helyes relációs adatbázisokat, és az adatbázisaink adatain végzett műveletek eredménye sem lenne megjósolható. Ezzel szemben viszont az adatbázisok tervezése azokat a strukturált folyamatokat foglalja magába, amelyek révén egy relációs adatbázis felépül. A helyes adatbázis-tervezési módszerek ismerete elengedhetetlen ahhoz, hogy pontos és következetes adatokat tartalmazó adatbázisokat hozzunk létre, illetve hogy a kinyert információk is a lehető legpontosabbak és legfrissebbek legyenek.

Ha teljes vállalatra kiterjedő adatbázisokat szeretnénk tervezni és létrehozni, webes alapú internetes üzleti adatbázisokat akarunk fejleszteni, vagy adattárakat kívánunk üzemeltetni, célszerű megfontolni az adatbázisok elméletének tanulmányozását. Ez akkor is érvényes, ha az említett területek egyikében sem akarunk elmerülni, csak elismert adatbázis-szakértővé szeretnénk válni. Mindenki másnak, aki relációs adatbázisokat tervez és készít különböző rendszerekre (és úgy véljük, ők teszik ki e könyv olvasóinak nagy többségét), a helyes adatbázis-tervezési módszerek alapos ismerete elegendő. Ne feledjük, hogy egy adatbázist megtervezni viszonylag könnyű, de *megvalósítani* azt egy adott RDBMS programban egy adott rendszerre már teljesen más tészta (amit más könyvekből sajátíthatunk el).

A piacon számos jó könyvet találunk az adatbázis-tervezésről. Egyesek, például Mike Hernandez *Database Design for Mere Mortals* című könyve (Addison-Wesley, 2004; magyarul: *Adatbázis-tervezés: A relációs adatbázisok alapjairól földi halandóknak.* Kiskapu, 2004) csak az adatbázis-tervezés módszertanával foglalkoznak, míg mások, például C.J. Date *An Introduction to Database Systems* (Addison-Wesley, 2003) című műve, keverik az elméletet és a gyakorlati tervezést. (Arra viszont fel kell hívnunk a figyelmet, hogy az elméleti könyvek általában nem könnyű olvasmányok.) Miután eldöntöttük, hogy milyen irányban szeretnénk továbbhaladni, válasszuk ki és vegyük meg a megfelelő könyveket, fogjunk egy csésze dupla eszpresszót (vagy ha más kedvenc italunk van, akkor azt), és merüljünk el bennük. Ha már általánosságban magabiztosan mozgunk a relációs adatbázisok körében, rá fogunk jönni, hogy mélyebben meg kell ismerkednünk az SQL-lel is – ezért olvassuk ezt a könyvet.

Összefoglalás

A fejezetet a ma használatos adatbázisok különböző típusainak rövid ismertetésével kezdtük. Megtudtuk, hogy a dinamikus adatokkal dolgozó vállalatok és intézmények műveleti adatbázisokat használnak, és így biztosítják, hogy a kinyert információk mindig a lehető legfrissebbek és legpontosabbak legyenek, míg a statikus adatokat kezelő szervezetek elemző adatbázisokat alkalmaznak.

Ez után röviden áttekintettük a relációs adatbázismodell történetét. Elmeséltük, hogy a modellt Dr. E. F. Codd alkotta meg a matematika egyes ágaira alapozva, és hogy a modell már több mint 35 éve létezik. Adatbázis-kezelő programokat, ahogy ma ismerjük őket, különféle számítógépes környezetekhez készítenek, a teljesítményük, a hatékonyságuk és a képességeik köre pedig az 1970-es évek óta folyamatosan nő. A nagygépektől az asztali és webes alkalmazásokig sok-sok vállalat gerincét RDBMS programok képezik.

Ezt követően a relációs adatbázisok felépítését vizsgáltuk. Bemutattuk az alapvető összetevőiket, és röviden elmagyaráztuk a szerepüket. Tanultunk a kapcsolatok három lehetséges típusáról, és megértettük, miért fontosak, nem csak magának az adatbázis szerkezetének a szempontjából, hanem abból a szempontból is, hogy miként segítik az SQL működésének jobb megértését.

Végül elmagyaráztuk, hogy milyen hasznunk származik a relációs adatbázisok tanulmányozásából, és hogy miként kell megtervezni őket. Most már tudjuk, hogy a ma használatos adatbázisok közül a relációs adatbázisok a legelterjedtebbek, és szinte minden adatbázis-kezelő szoftver mögött, amivel találkozhatunk, egy relációs adatbázis áll. Ezen kívül arról is képet kaptunk, hogy merre haladhatunk, ha kicsit többet szeretnénk tudni a relációs adatbázisok elméletéről és tervezéséről.

A következő fejezetben arra tanulunk meg néhány módszert, hogy miként finomhangolhatunk meglevő adatbázis-szerkezeteket.

A helyes adatbázis-szerkezet kialakítása

„Házainkat a saját képünkre formáljuk,
ezért ők is formálnak minket."
– Sir Winston Churchill

A fejezet témakörei

- Mi a célja ennek a fejezetnek?
- Miért lényeges a helyes szerkezet?
- A mezők finomhangolása
- A táblák finomhangolása
- Szilárd kapcsolatok kialakítása
- Ennyi az egész?
- Összefoglalás

Könyvünk legtöbb olvasója bizonyára egy meglevő adatbázis-szerkezettel dolgozik, amelyet (reméljük) a kedvenc RDBMS programjára ültetett. Nem tudhatjuk, hogy az Olvasó – vagy az, aki elkészítette az adatbázist – ténylegesen rendelkezett-e az adatbázis megfelelő megtervezéséhez szükséges tudással, képzettséggel vagy idővel, ezért a legrosszabbat feltételezzük: valószínűleg egy pár olyan táblánk van, amelyekre ráférne némi finomhangolás. Ebben a fejezetben szerencsére éppen olyan eljárásokról fogunk tanulni, amelyek segítenek gatyába rázni az adatbázisunkat, és gondoskodnak róla, hogy könnyen kinyerhessük a szükséges információkat a táblákból.

Mi a célja ennek a fejezetnek?

Felmerülhet bennünk a kérdés, hogy miért foglalkozunk ebben a könyvben adatbázis-tervezéssel, különösen az egyik első fejezetben. Nos, ennek oka egyszerű: ha az adatbázisainkat rosszul tervezzük meg, akkor sok SQL-utasítás, amit a könyv hátralevő részében megtanulunk, a legjobb esetben is nehezen megvalósítható lesz, de az is megtörténhet, hogy szinte hasznavehetetlenné válik. Ha azonban az adatbázisaink szerkezete helyes, a könyvben bemutatott fogásokat hatékonyan használhatjuk.

A fejezetben nem foglalkozunk az adatbázis-tervezés finomságaival, csak az a cél, hogy az adatbázisainkat viszonylag jó formába hozzuk. Melegen ajánlott végigolvasni a fejezetet, hogy a tábláink szerkezete biztosan helyes legyen.

Megjegyzés *Fontos megjegyeznünk, hogy az adatbázisok logikai felépítését fogjuk tárgyalni. Nem mutatjuk be, hogyan kell létrehozni vagy megvalósítani egy adatbázist egy olyan adatbázis-kezelő rendszerben, amelyik ismeri az SQL-t, mivel – ahogy a bevezetőben említettük – ezek a témakörök kívül esnek a könyv keretein.*

Miért lényeges a helyes szerkezet?

Ha az adatbázisaink szerkezete nem megfelelő, a látszólag egyszerű információk kinyerésével is gondjaink lehetnek, adataink nehezen kezelhetővé válnak, és minden alkalommal kínszenvedés lesz mezőket adni a táblákhoz vagy mezőket törölni onnan. A rosszul megválasztott felépítés az adatbázist az adatok következetessége, a táblák kapcsolatai és a pontos információk kinyerése szempontjából is érinti, és ezek csak a jéghegy csúcsát jelentik. Ha el szeretnénk kerülni a kínlódást, gondoskodjunk róla, hogy az adatbázisaink szerkezete helyes legyen.

Az említett problémák többsége elkerülhető, ha már az elején megfelelően tervezzük meg az adatbázisunkat, de akkor sincs minden veszve, ha ezen a lépésen már túl vagyunk: csak annyit kell tennünk, hogy alkalmazzuk a következőkben bemutatott eljárásokat, és máris élvezhetjük a helyes szerkezet előnyeit. Azzal azonban tisztában kell lennünk, hogy a végső szerkezet minősége egyenesen arányos annak az időnek a mennyiségével, amit a finomhangolásra fordítunk. Minél gondosabban és türelmesebben alkalmazzuk a bemutatott módszereket, annál biztosabban járunk sikerrel.

Most pedig lássuk az első lépést a helyes adatbázis-szerkezet kialakítása felé: a mezők finomhangolását.

A mezők finomhangolása

Mivel a mezők az adatbázisok alapvető szerkezetei, gondoskodnunk kell róla, hogy tökéletes formában legyenek, mielőtt a táblák egészének finomhangolásába kezdhetnénk. Sok esetben a mezők kijavítása megoldja az adott tábla számos problémáját, és elejét veheti más lehetséges gondoknak is.

Mikor helyes egy név? (Első rész)

Ahogy az előző fejezetben megtanultuk, a mezők a hozzájuk tartozó tábla tárgyának egy-egy jellemzőjét írják le. Ha egy mezőnek megfelelő nevet adunk, képesek leszünk azonosítani, hogy milyen jellemzőt is jelöl. A kétértelmű, homályos jelentésű vagy bonyolult név előbb-utóbb biztosan gondot okoz, és arra utal, hogy a mező célját nem gondoltuk át alaposan. Vizsgáljuk át a mezőink nevét az alábbi ellenőrzőlista szerint:

- *A név az egész cégen belül jelentést hordoz, és leírja a mező célját?* Ha az adatbázist több részleg is használni fogja, mindenképpen olyan nevet válasszunk, ami mindenki számára, aki hozzáfér az adott mezőhöz, jelentéssel bír. A jelentés fontos, mert ha olyan szót használunk, ami különböző embereknek mást és mást jelent, magunk keressük a bajt.
- *A mező neve világos és egyértelmű?* A PhoneNumber (Telefonszám) például olyan mezőnév, ami nagyon félrevezető lehet. Milyen telefonszámot tartalmaz a mező? Otthoni vagy munkahelyi számot? Mobilszámot? Tanuljunk meg konkrétan fogalmazni! Ha az említettek telefonszámok mindegyikét rögzítenünk kell, hozzunk létre HomePhone (Otthoni telefon), WorkPhone (Munkahelyi telefon) és CellPhone (Mobil) mezőket.
  A mezőnevek világossá és egyértelművé tétele mellett arra is ügyeljünk, hogy ugyanazt a mezőnevet ne használjuk több táblában. Tegyük fel például, hogy három táblánk van Customers (Vásárlók), Vendors (Beszállítók) és Employees (Alkalmazottak) néven. Kétségtelen, hogy mindhárom táblában szükség lesz City (Város) és State (Állam) mezőkre, és mindegyiknek azonos lesz a neve az egyes táblákban. Ezzel addig nincs gond, amíg nem kell hivatkoznunk egy adott mezőre. De hogyan különböztetjük meg például a Vendors tábla City mezőjét a Customers tábla és az Employees tábla City mezőitől? A válasz egyszerű: adjunk egy rövid előtagot mindegyik mező nevéhez. A Vendors tábla mezője lehet mondjuk VendCity, a Customers tábláé CustCity, az Employees tábláé pedig EmpCity. Így a három mezőre már világosan hivatkozhatunk. (Ezt a módszert más általános mezők – például FirstName, LastName, Address, vagyis Keresztnév, Vezetéknév és Cím – esetében is alkalmazhatjuk.)
  A legfontosabb, amit meg kell jegyeznünk, hogy ügyeljünk rá, hogy az adatbázis minden mezője egyedi nevet kapjon, és a név csak egyetlen egyszer szerepeljen a teljes adatbázis-szerkezetben. Ez alól a szabály alól az egyetlen kivételt az jelenti, amikor egy mezőt arra használunk, hogy kapcsolatot teremtsünk két tábla között.
- *A mező neveként rövidítést vagy betűszót használtunk?* Ha igen, változtassuk meg! A betűszavak nehezen megfejthetők, viszont könnyű félreértelmezni őket. Képzeljük el, hogy egy olyan mezőt látunk, amelynek a neve CAD_SW. Honnan tudjuk, hogy mit jelöl a mező? A rövidítésekkel is csínján kell bánnunk; óvatosan használjuk őket – csak akkor, ha a mező nevét világosabbá teszi, mert nem szabad, hogy elterelje a figyelmet a név jelentéséről.
- *Olyan nevet adtunk, ami rejtve vagy nyíltan egynél több jellemzőre utal?* Az ilyen neveket könnyű kiszúrni, mert általában szerepel bennük az *and* (és) vagy az *or* (vagy) szó. Az olyan mezőnevek, amelyek fordított perjelet (\), kötőjelet (-) vagy „és" (&) jelet tartalmaznak, szintén egyértelműen jelzik, hogy több jellemzőre utaló névről van szó. Tehát ha olyan mezőkkel rendelkezünk, mint a Phone/Fax vagy az Area or Location, akkor vizsgáljuk felül, hogy milyen adatot tárolnak, és ha lehet, bontsuk fel őket kisebb, elkülöníthető mezőkre.

Megjegyzés *Az SQL-szabvány szabályos azonosítóként határozza meg azokat a neveket, amelyeknek betűvel kell kezdődniük, és csak betűket, számokat, valamint aláhúzásjelet tartalmazhatnak. A szóközök nem megengedettek. A szabvány a határolt azonosítót is meghatározza: ez olyan – (kettős) idézőjelekkel körülvett – név, amelynek betűvel kell kezdődnie, és csak betűket, számokat, aláhúzásjelet, szóközöket és meghatározott különleges karaktereket tartalmazhat. Mivel sok SQL-megvalósítás csak a szabályos azonosítókat támogatja, azt javasoljuk, hogy a mezőnevek esetében kizárólag ezt a formát kövessük.*

Miután az ellenőrzőlista segítségével felülvizsgáltuk a mezőneveinket, egy feladatunk marad: gondoskodnunk kell róla, hogy a mezőnevekben csak egyes számot használjunk. Az olyan többes számú nevek, mint a Categories (Kategóriák), azt sugallják, hogy a mező egy rekordhoz két vagy több értéket is tartalmazhat, ami nem túl jó ötlet. A mezőnévnek azért kell egyes számúnak lennie, mert a hozzá tartozó tábla tárgyának egyetlen jellemzőjét írja le. A táblák neve ezzel szemben többes számú kell legyen, mert hasonló objektumok vagy események gyűjteményét ábrázolják. Ha tartjuk magunkat ehhez a szabályhoz, a táblaneveket könnyen megkülönböztethetjük a mezőnevektől.

A durva élek lecsiszolása

Most, hogy tisztába tettük a mezőneveket, összpontosítsunk magára a mezőknek a szerkezetére. Ha biztosak vagyunk benne, hogy a mezőink szerkezete megfelelő, akkor is tehetünk néhány dolgot annak érdekében, hogy a mezők a lehető leghatékonyabbak legyenek. Az alábbi ellenőrzőlistát követve vizsgálhatjuk meg, hogy kell-e még finomítanunk a mezőkön:

- *Gondoskodjunk róla, hogy a mező a tábla tárgyának egy konkrét jellemzőjét írja le!* A lényeg itt az, hogy állapítsuk meg, hogy a mező valóban az adott táblába tartozik-e. Ha nem létfontosságú a táblához, távolítsuk el, vagy helyezzük át egy másik táblába. Az egyetlen kivétel e szabály alól, amikor a mezőt arra használjuk, hogy kapcsolatot teremtsünk az adott tábla és az adatbázis más táblái között, vagy amikor azért adjuk a mezőt a táblához, hogy az adatbázisprogramnak segítsünk egy adott feladat elvégzésében. A 2.1. ábrán látható Classes (Órák) táblában például a StaffLastName (Tanár vezetékneve) és StaffFirstName (Tanár keresztneve) mezők feleslegesek, mivel StaffID (Tanárazonosító) mezőnk is van. A StaffID kapcsolja össze a Classes és a Staff (Tanári kar) táblát, és egy nézettábla vagy egy SELECT SQL-lekérdezés segítségével egyszerre mindkét táblából megtekinthetünk adatokat. Ha egy táblában felesleges mezők találhatók, törölhetjük őket, de egy másik tábla alapjaként is felhasználhatjuk azokat, ha az adatbázis szerkezetében máshol nem bukkanak fel. (Ennek módját a fejezet későbbi részében mutatjuk be.)
- *Gondoskodjunk róla, hogy a mező csak egyetlen értéket tartalmazzon!* Az olyan mezőket, amelyek több azonos típusú értéket is tartalmazhatnak, *többértékű mezőknek* hívjuk (ilyen például egy több telefonszámot tároló mező), azokat pedig,

amelyekbe kettő vagy több *különböző* érték is kerülhet, *többrészes mezőknek* (ilyen például egy elem sorszámát és leírását egyaránt tároló mező). A többértékű és többrészes mezők komoly kavarodást okozhatnak az adatbázisban, különösen ha szerkeszteni, törölni vagy rendezni próbáljuk az adatokat. Ha gondoskodunk róla, hogy minden mező csak egy értéket tároljon, nagy lépést teszünk az adatok következetességének és az információk pontosságának biztosítása érdekében, de egyelőre elég annyi, ha azonosítunk minden többértékű és többrészes mezőt, és feljegyezzük őket. Azt, hogy miként oldhatjuk fel őket, a következő részben tanuljuk meg.

Staff

StaffID	StaffFirstName	StaffLastName	StaffStreetAddress	StaffCity	StaffState	*<<további mezők>>*
98014	Peter	Brehm	722 Moss Bay Blvd.	Kirkland	WA	...
98019	Mariya	Sergienko	901 Pine Avenue	Portland	OR	...
98020	Jim	Glynn	13920 S.E. 40th Street	Bellevue	WA	...
98021	Tim	Smith	30301 166th Ave. N.E.	Seattle	WA	...
98022	Carol	Viescas	722 Moss Bay Blvd.	Kirkland	WA	...
98023	Alaina	Hallmark	Route 2, Box 203 B	Woodinville	WA	...

Classes

ClassID	Class	ClassroomID	StaffID	StaffLastName	StaffFirstName	*<<további mezők>>*
1031	Art History	1231	98014	Brehm	Peter	...
1030	Art History	1231	98014	Brehm	Peter	...
2213	Biological Principles	1532	98021	Smith	Tim	...
2005	Chemistry	1515	98019	Sergienko	Mariya	...
2001	Chemistry	1519	98023	Hallmark	Alaina	...
1006	Drawing	1627	98020	Glynn	Jim	...
2907	Elementary Algebra	3445	98022	Viescas	Carol	...

2.1. ábra

Tábla felesleges mezőkkel

- *Gondoskodjunk róla, hogy a mező ne számítás vagy összefűzés eredményét tárolja!* Egy helyesen megtervezett táblában nem lehetnek számított mezők. A problémát magának a számított mezőnek az értéke jelenti. A táblázatkezelők táblázataival ellentétben az adatbázistáblák nem tárolnak számításokat, így ha a számításban szereplő valamelyik érték megváltozik, a mezőben tárolt eredmény nem frissül. Az értéket csak saját kezűleg frissíthetjük, vagy valamilyen eljárásközpontú kódot kell írnunk, ami ezt önműködően elvégzi. Bármelyik megoldást válasszuk is, nekünk, a fejlesztőnek, és a felhasználónak is kényelmetlen az érték frissítését biztosítani. A számításokat ajánlott ehelyett SELECT utasításokba ágyazni; ennek előnyeiről majd akkor tanulunk, ha eljutottunk az 5. fejezethez.

- *Gondoskodjunk róla, hogy a mező csak egyetlen egyszer szerepeljen a teljes adatbázisban!* Ha elkövettük azt a gyakori hibát, hogy ugyanazt a (például CompanyName, vagyis Cégnév) mezőt az adatbázis több táblájába is beszúrtuk, az adatok következetlenné válhatnak, ha az egyik táblában módosítjuk a mező értékét, de megfeledkezünk ugyanezt a változtatást mindenütt elvégezni, ahol a mező előfordul. Ezt a problémát úgy kerülhetjük el teljesen, ha gondoskodunk róla, hogy egy mező csak egyszer szerepeljen az egész adatbázis-szerkezetben. (Az egyetlen kivételt ez alól a szabály alól az jelenti, ha a mezőt arra használjuk, hogy kapcsolatot teremtsünk két tábla között.)

Megjegyzés *Egyes kereskedelmi forgalomban kapható adatbázis-kezelő rendszerek legújabb változatai megengedik olyan oszlopok meghatározását is, amelyek egy számított kifejezés eredményeként állnak elő. Ha az általunk használt adatbázisrendszer rendelkezik ezzel a szolgáltatással, akkor meghatározhatunk számított mezőket, de tartsuk észben, hogy az adatbázisrendszer további erőforrásokat igényel ahhoz, hogy a számított értéket mindig frissen tartsa, ha a kifejezésben szereplő valamelyik mező értéke megváltozna.*

A többrészes mezők feloldása

Ahogy korábban említettük, a többrészes és többértékű mezők tönkretehetik az adatok következetességét, ezért fel kell oldanunk őket, hogy megelőzzünk minden problémát. Az, hogy melyikkel kezdjük, teljesen mindegy, ezért először a többrészes mezőkkel foglalkozunk.

Customers

CustomerID	CustomerName	StreetAddress	PhoneNumber	<<további mezők>>
1001	Suzanne Viescas	15127 NE 24th, #383, Redmond, WA 98052	425 555-2686	...
1002	William Thompson	122 Spring River Drive, Duvall, WA 98019	425 555-2681	...
1003	Gary Hallmark	Route 2, Box 203B, Auburn, WA 98002	253 555-2676	...
1004	Robert Brown	672 Lamont Ave, Houston, TX 77201	713 555-2491	...
1005	Dean McCrae	4110 Old Redmond Rd., Redmond, WA 98052	425 555-2506	...
1006	John Viescas	15127 NE 24rh, #383, Redmond, WA 98052	425 555-2511	...
1007	Mariya Sergienko	901 Pine Avenue, Portland, OR 97208	503 555-2526	...
1008	Neil Patterson	233 West Valley Hwy, San Diego, CA 92199	619 555-2541	...

TÖBBRÉSZES MEZŐK

2.2. ábra

Tábla többrészes mezőkkel

Néhány egyszerű kérdés megválaszolásával megállapíthatjuk, hogy egy mező többrészes-e: „Felbonthatom ennek a mezőnek a jelenlegi értékét kisebb, elkülöníthető részekre?", „Gondot okozhat egy konkrét információ kinyerése, mert egy olyan mezőben van elte-

metve, ami más információkat is tartalmaz?". Ha bármelyik kérdésre igennel válaszolunk, többrészes mezővel van dolgunk. A 2.2. ábrán láthatunk egy rosszul megtervezett táblát, amely több többrészes mezőt is tartalmaz.

A Customers tábla két többrészes mezőt tartalmaz: a CustomerName (Vásárló neve) és a Street Address (Cím) mezőket. Ezen kívül még egy mező értéke állhat több részből is – a PhoneNumber-é. Hogyan tudunk vezetéknév vagy irányítószám szerint rendezni? Sehogy, mert ezek az értékek olyan mezőkbe ágyazódnak, amelyek más információkat is tárolnak. Viszont ha megfigyeljük, ezek a mezők kisebb mezőkre bonthatók. A CustomerName-ből például két külön mezőt – CustFirstName és CustLastName (Vásárló keresztneve és Vásárló vezetékneve) – alkothatunk. (Figyeljük meg, hogy közben a fejezet korábbi részében javasolt elnevezési módszert alkalmaztuk, és a Cust előtaggal láttuk el a FirstName és LastName mezőket.) Ha egy többrészes mezőre bukkantunk egy táblában, állapítsuk meg, hogy az általa tárolt érték hány részből épül fel, majd bontsuk fel a mezőt megfelelő számú kisebb mezőre. A 2.3. ábrán láthatjuk, hogy miként bonthatjuk fel a Customers tábla két többrészes mezőjét.

Customers

CustomerID	CustFirstName	CustLastName	CustAddress	CustCity	CustState	CustZipcode
1001	Suzanne	Viescas	15127 NE 24th, #383	Redmond	WA	98052
1002	William	Thompson	122 Spring River Drive	Duvall	WA	98019
1003	Gary	Hallmark	Route 2, Box 203B	Auburn	WA	98002
1004	Robert	Brown	672 Lamont Ave	Houston	TX	77201
1005	Dean	McCrae	4110 Old Redmond Rd.	Redmond	WA	98052
1006	John	Viescas	15127 NE 24th, #383	Redmond	WA	98052
1007	Mariya	Sergienko	901 Pine Avenue	Portland	OR	97208
1008	Neil	Patterson	233 West Valley Hwy	San Diego	CA	92199

2.3. ábra

Többrészes mezők feloldása a Customers táblában

Megjegyzés *A CustomerName és StreetAddress mezők felbontása mellett az sem árt, ha az adatbázisban a telefonszámokat két külön mezőben – körzetszám és helyi telefonszám – tároljuk. A legtöbbi üzleti adatbázis egyetlen mezőben tárolja a telefonszámokat, a demográfiai adatokat elemző alkalmazásokban azonban lényeges lehet a körzetszám elkülönítése (ezt a szűkre szabott hely miatt sajnos nem állt módunkban bemutatni a 2.3. ábrán).*

Néha nem könnyű felismerni egy többrészes mezőt. Nézzük meg például a 2.4. ábrán látható Instruments (Hangszerek) táblát. Első pillantásra nem veszünk észre egyetlen többrészes mezőt sem, de ha közelebbről megnézzük, láthatjuk, hogy az InstrumentID (Hangszerazonosító) valójában több részből áll. Az ebben a mezőben tárolt érték kétféle információt is megad: a hangszercsoportot – például AMP (amplifier, erősítő), GUIT (guitar, gitár) vagy MFX (multi-

effects unit, multieffekt) –, valamint a hangszer azonosítószámát. Ezt a két értéket szét kell választanunk, és a saját mezőikben kell tárolnunk, hogy biztosítsuk az adatok következetességét. Képzeljük el, micsoda nehézséget jelentene a mező frissítése, ha az MFX hangszercsoport rövidítése például MFU-ra változna: kódot kellene írnunk, ami feldolgozza a mező értékét, megnézi, hogy szerepel-e benne az MFX előtag, és ha igen, MFU-ra cseréli azt. Nem mintha ezt nem *tudnánk* megcsinálni, de az biztos, hogy a szükségesnél több munkát kellene végeznünk, sőt egyáltalán nem kellene foglalkoznunk a kérdéssel, ha az adatbázist helyesen terveztük volna meg. Ha tehát a példában szereplőhöz hasonló mezőkkel rendelkezünk, bontsuk azokat kisebb mezőkre, hogy a mezőszerkezet hatékony legyen.

Instruments

InstrumentID	Manufacturer	InstrumentDescription	<<*további mezők*>>
GUIT2201	Fender	Fender Stratocaster	...
MFX3349	Zoom	Player 2100 Multi-Effects	...
AMP1001	Marshall	JCM 2000 Tube Super Lead	...
AMP5590	Crate	VC60 Pro Tube Amp	...
SFX2227	Dunlop	Cry Baby Wah-Wah	...
AMP2766	Fender	Twin Reverb Reissue	...

2.4. ábra

Példa rejtett többrészes mezőre

A többértékű mezők feloldása

A többrészes mezők feloldása egyáltalán nem nehéz, a többértékű mezőké azonban már egy kicsit bonyolultabb, és több munkát igényel. A többértékű mezőket szerencsére könnyen felismerhetjük. Az ilyen típusú mezőkben tárolt adatok szinte kivétel nélkül vesszőket, pontosvesszőket vagy más szokványos elválasztó karaktereket tartalmaznak, amelyek a mezőben levő különböző értékek között állnak. A többértékű mezőre a 2.5. ábrán láthatunk egy példát.

Pilots

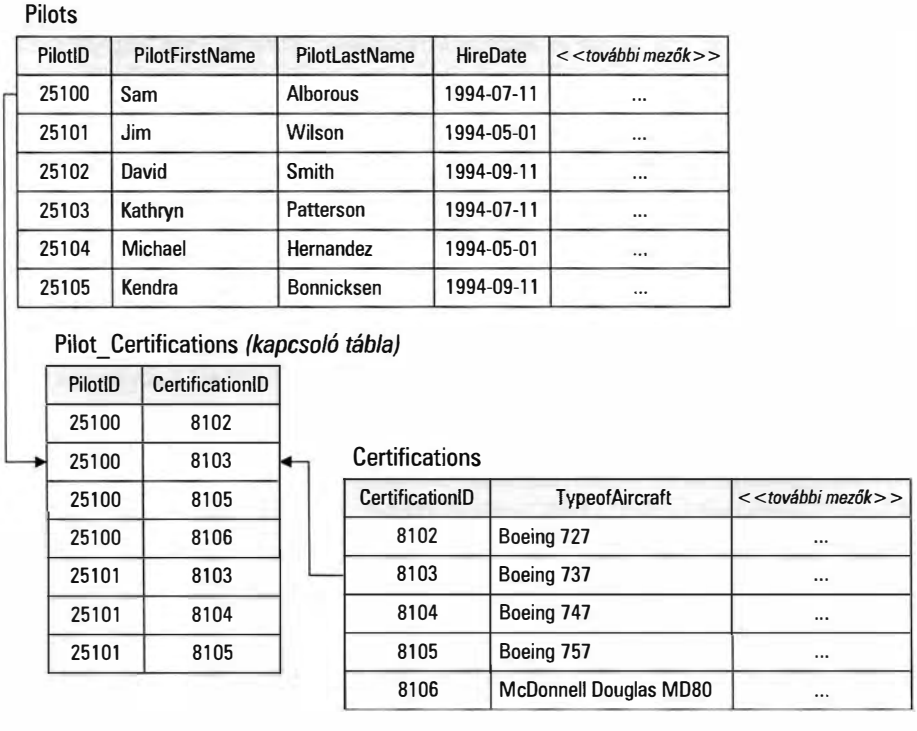
PilotID	PilotFirstName	PilotLastName	HireDate	Certifications	<<*további mezők*>>
25100	Sam	Alborous	1994-07-11	727, 737, 757, MD80	...
25101	Jim	Wilson	1994-05-01	737, 747, 757	...
25102	David	Smith	1994-09-11	757, MD80, DC9	...
25103	Kathryn	Patterson	1994-07-11	727, 737, 747, 757	...
25104	Michael	Hernandez	1994-05-01	737, 757, DC10	...
25105	Kendra	Bonnicksen	1994-09-11	757, MD80, DC9	...

2.5. ábra

Tábla többértékű mezővel

A példában minden pilóta adott gépekre szóló jogosítványokkal rendelkezik, és ezeket egyetlen mező, a Certifications (Jogosítványok) tárolja. Az adatok tárolási módja komoly gondot jelent, mert az adatok következetességét valószínűleg ugyanolyan nehéz lesz biztosítani, mint a többrészes mezők esetében. Ha közelebbről megnézzük az adatokat, láthatjuk, hogy egy SQL-lekérdezéssel nehéz lesz keresést végrehajtani közöttük, vagy rendezni őket. Mielőtt azonban megfelelően feloldhatnánk a mezőt, először meg kell értenünk, hogy valójában milyen kapcsolat áll fenn a többértékű mezők és az őket eredetileg tartalmazó táblák között.

A többértékű mezőkben levő értékek több-a-többhöz kapcsolatban állnak a szülőtábla minden rekordjával: egy többértékű mező egy konkrét értéke bármennyi rekordhoz kapcsolódhat a szülőtáblában, és a szülőtábla egyes rekordjait is bármennyi értékhez társíthatjuk a többértékű mezőben. A 2.5. ábra Certifications mezőjében szereplő repülőgéptípusokat például egyenként bármennyi pilótához kapcsolhatjuk, és egy pilótát is bármennyi repülőgéphez társíthatunk a Certifications mezőben. Ezt a több-a-többhöz kapcsolatot ugyanúgy oldhatjuk fel, mint bármely más több-a-többhöz kapcsolatot egy adatbázisban: egy kapcsoló tábla segítségével.



Pilots

PilotID	PilotFirstName	PilotLastName	HireDate	<<további mezők>>
25100	Sam	Alborous	1994-07-11	...
25101	Jim	Wilson	1994-05-01	...
25102	David	Smith	1994-09-11	...
25103	Kathryn	Patterson	1994-07-11	...
25104	Michael	Hernandez	1994-05-01	...
25105	Kendra	Bonnicksen	1994-09-11	...

Pilot_Certifications *(kapcsoló tábla)*

PilotID	CertificationID
25100	8102
25100	8103
25100	8105
25100	8106
25101	8103
25101	8104
25101	8105

Certifications

CertificationID	TypeofAircraft	<<további mezők>>
8102	Boeing 727	...
8103	Boeing 737	...
8104	Boeing 747	...
8105	Boeing 757	...
8106	McDonnell Douglas MD80	...

2.6. ábra

Többértékű mező feloldása kapcsoló tábla segítségével

A kapcsoló táblát úgy hozzuk létre, hogy a többértékű mezőből és az eredeti tábla elsődleges kulcs mezőjének *másolatából* új táblát alkotunk. Az újonnan létrehozott kapcsoló táblának adunk egy megfelelő nevet, a két mezőt összetett elsődleges kulccsá tesszük (vagyis a két mező értéke együttesen fogja az új tábla rekordjait egyedileg azonosítani), végül pedig a kapcsoló tábla két mezőjének értékeit egy-az-egyhez kapcsolattal egymáshoz társítjuk. Az eljárás menetét a 2.5. ábrán látott Pilots tábla esetében a 2.6. ábra szemlélteti.

Vessük össze Sam Alborous (PilotID 25100) bejegyzését a régi Pilots és az új Pilot_Certifications táblában! Az új kapcsoló tábla legfőbb előnye, hogy most már *tetszőleges* számú jogosítványt kapcsolhatunk egy pilótához, és bizonyos kérdéseket sokkal könnyebben tehetünk fel az adatbázisnak. Megállapíthatjuk például, hogy mely pilóták jogosultak Boeing 747-est vezetni, vagy lekérhetjük egy adott pilóta jogosítványainak listáját. Emellett azt is tapasztalhatjuk, hogy az adatokat bármilyen sorrendbe rendezhetjük, hátrányos hatások nélkül.

Megjegyzés *Egyes adatbázis-kezelő rendszerek – elsősorban a Microsoft Office Access 2007 – megengedik, hogy kifejezetten többértékű mezőket határozzunk meg. Az adatbázisrendszer ugyanakkor ezt úgy teszi lehetővé, hogy egy rejtett rendszertáblát hoz létre, ami igen hasonló a 2.6. ábrán látható kapcsoló táblához. Véleményünk szerint jobb, ha látjuk és kézben tartjuk a tábláinkat, ezért azt javasoljuk, hogy magunk hozzuk létre a helyes adatszerkezeteket, és ne az adatbázisrendszer szolgáltatásaira támaszkodjunk.*

Ha követjük az ebben a részben bemutatott eljárásokat, a mezőink megfelelőek lesznek. Most pedig, hogy finomhangoltuk a mezőket, fordítsuk a figyelmünket a fejezet második témájára, és vegyük szemügyre a táblák szerkezetét!

A táblák finomhangolása

A táblák jelentik minden SQL-lekérdezés alapját. Nem telik sok időbe, és rájövünk majd, hogy a rosszul megtervezett táblák gondokat okozhatnak az adatok következetességével kapcsolatban, és a többtáblás SQL-lekérdezések írásakor is megnehezítik a munkát. Mindezek miatt gondoskodnunk kell róla, hogy a tábláinkat a lehető leghatékonyabban építsük fel, hogy könnyen kinyerhessük belőlük azokat az információkat, amelyekre szükségünk van.

Mikor helyes egy név? (Második rész)

A mezőkről szóló részben megtanultuk, hogy milyen fontos, hogy egy mezőnek megfelelő nevet adjunk, és hogy miért kell komoly figyelmet fordítanunk a mezők elnevezésére. Ebben a részben látni fogjuk, hogy ugyanez vonatkozik a táblákra is. Ahogy mondtuk, a táblák egyetlen tárgyat kell, hogy ábrázoljanak – ha egynél több tárgyuk van, akkor kisebb táblákra kell bontanunk őket. A tábla nevének egyértelműen azonosítania kell

a tábla tárgyát: ha kétértelmű, homályos vagy bonyolult, akkor biztosak lehetünk benne, hogy a tábla tárgyát nem gondoltuk át kellőképpen. A táblanevek megfelelőségét az alábbi ellenőrzőlista szerint vizsgálhatjuk meg:

- *A név egyedi és kellően leíró ahhoz, hogy az egész cégen belül jelentéssel bírjon?* Az egyedi nevek biztosítják, hogy az adatbázis minden táblája más tárgyat ábrázoljon, és hogy a cégen belül mindenki számára világos legyen, hogy mit tartalmaznak az egyes táblák. Egyedi, leíró neveket találni nem mindig könnyű, de hosszú távon egyértelműen kifizetődik.
- *A név pontosan, világosan és egyértelműen azonosítja a tábla tárgyát?* Ha egy tábla neve homályos vagy nem egyértelmű, a nyakunkat tehetjük rá, hogy a tábla egynél több tárgyat ábrázol. A Dates (Dátumok) például homályos táblanév, mert nehéz pontosan megállapítani, hogy mit tartalmaz a tábla, hacsak nincs kéznél egy leírás róla. Tegyük fel például, hogy egy szórakoztatóügynökség adatbázisában találunk egy ilyen nevű táblát. Ha közelebbről megvizsgáljuk a táblát, valószínűleg kiderül, hogy az ügyfelekkel való találkozók időpontját, illetve az ügynökség előadóinak lekötött előadásait tárolja – vagyis világos, hogy két tárgya van. Ebben az esetben két táblára kell bontanunk a táblát, és az új táblákat megfelelő nevekkel – például Client_Meetings (Ügyfelek találkozói) és Entertainer_Schedules (Előadók fellépései) – kell ellátnunk.
- *Tartalmaz a név olyan szavakat, amelyek fizikai jellemzőkre utalnak?* Kerüljük az olyan szavak használatát a táblanevekben, mint a *File*, a *Record* vagy a *Table*, mivel ezek csak szükségtelen zavart okoznak. Ha egy tábla neve ilyen jellegű szót tartalmaz, a tábla valószínűleg több tárgyat ábrázol. Vegyük például az Employee_Record (Alkalmazottak bejegyzései) nevet: felületes ránézésre úgy tűnik, hogy ezzel a névvel nincs semmi baj, de ha belegondolunk, hogy egy alkalmazott bejegyzésének mit kell tartalmaznia, rájöhetünk, hogy itt bizony gondok lehetnek. A név olyan szót tartalmaz, amelyet el kell kerülnünk, a táblának pedig három tárgya is lehet: az alkalmazottak, a részlegek és a fizetés. Bontsuk tehát az eredeti táblát (Employee_Record) három új táblára, hogy mindhárom tárgy külön táblába kerüljön.
- *A tábla neveként rövidítést vagy betűszót használtunk?* Ha igen, azonnal változtassuk meg a nevet! Egy rövidítésből ritkán található ki a tábla tárgya, a betűszavak pedig általában nehezen megfejthetők. Tegyük fel például, hogy a cégünk adatbázisában van egy tábla, amelynek a neve SC. Honnan tudjuk, hogy mit ábrázol a tábla, ha nem ismerjük maguknak a betűknek a jelentését? Tény, hogy így nehéz azonosítani a tábla tárgyát, sőt az is lehet, hogy a tábla neve a cég egyes részlegeinek mást és mást jelent: a személyzeti osztály dolgozói azt hihetik, hogy az SC a Steering_Committees (Ügyvivő bizottságok) rövidítése, az informatikai részlegen dolgozók azt, hogy a System_Configurations-é (Rendszerbeállítások), a biztonságiak pedig nyilván biztosak benne, hogy a jelentése Security_Codes (Biztonsági kódok). A példa világosan mutatja, hogy miért kell kerülnünk a rövidítéseket és betűszavakat a táblanevekben.

- *Olyan nevet adtunk, ami rejtve vagy nyíltan egynél több tárgyra utal?* Ez az egyik leggyakoribb hiba, amit egy tábla elnevezésénél elkövethetünk, de az ilyen neveket viszonylag könnyű kiszúrni, mert általában szerepel bennük az *and* (és) vagy az *or* (vagy) szó, vagy olyan karakterek, mint a fordított perjel (\), a kötőjel (-) vagy az „és" (&) jel. Jellemző példa az ilyen nevekre a Facility/Building vagy a Department or Branch. Ha egy táblát így neveztünk el, akkor vizsgáljuk felül, hogy valóban egynél több tárgyat ábrázol-e. Amennyiben igen, bontsuk fel a táblát kisebb táblákra, és adjunk az új tábláknak megfelelő neveket.

Megjegyzés *Emlékezhetünk, hogy az SQL-szabvány szabályos azonosítóként határozza meg azokat a neveket, amelyeknek betűvel kell kezdődniük, és csak betűket, számokat, valamint aláhúzásjelet tartalmazhatnak. A szóközök nem megengedettek. A szabvány a határolt azonosítót is meghatározza: ez olyan – (kettős) idézőjelekkel körülvett – név, amelynek betűvel kell kezdődnie, és csak betűket, számokat, aláhúzásjelet, szóközöket és meghatározott különleges karaktereket tartalmazhat. Mivel sok SQL-megvalósítás csak a szabályos azonosítókat támogatja, azt javasoljuk, hogy a táblanevek esetében is kizárólag ezt a formát kövessük.*

Miután az ellenőrzőlista segítségével felülvizsgáltuk a táblaneveket, még egy feladatunk marad: ellenőrizzünk minden táblanevet még egyszer, és gondoskodjunk róla, hogy a nevekben többes számot használjunk. A táblanevekben azért kell többes számot használnunk, mert a táblák az általuk ábrázolt tárgy *példányainak gyűjteményét* tárolják. Egy Employees nevű tábla például nem csak egy, hanem sok alkalmazott adatait tartalmazza. Emellett, ha tartjuk magunkat a többes számhoz, az abban is segít, hogy a táblaneveket megkülönböztessük a mezőnevektől.

A helyes szerkezet biztosítása

Most, hogy a táblaneveket felülvizsgáltuk, összpontosítsunk a táblák szerkezetére! Ahhoz, hogy hatékonyan tároljuk az adatainkat, és pontos információkat nyerjünk ki, létfontosságú, hogy a táblákat helyesen tervezzük meg. Érdemes időt szánni arra, hogy meggyőződjünk a táblák megfelelő felépítéséről, mert a befektetett idő busásan megtérül, amikor összetett, többtáblás SQL-lekérdezéseket kell majd írnunk. Az alábbi ellenőrzőlistát használhatjuk arra, hogy megállapítsuk, hogy a tábláink szerkezete helyes-e:

- *Gondoskodjunk róla, hogy minden tábla egyetlen tárgyat ábrázoljon!* Igen, tudjuk, hogy ezt már sokszor mondtuk, de nem lehet eléggé hangsúlyozni. Ha biztosítjuk, hogy minden tábla egyetlen tárgyat ábrázoljon, jelentősen csökkentjük annak a kockázatát, hogy az adatok következetessége sérüljön. Azt sem szabad elfelejtenünk, hogy a táblák tárgya egy objektum vagy esemény lehet. „Objektum" alatt valami fizikai dolgot értünk – alkalmazottakat, beszállítókat, gépeket, épületeket vagy részlegeket –, míg „esemény" valami olyasmi lehet, ami egy adott időben történik, és olyan jellemzői vannak, amiket rögzíteni szeretnénk. Az eseményekre a legjobb

példa az, amit életünk során mindnyájan átélünk – egy orvosi vizitet. A vizit nem kézzelfogható, de vannak rögzíthető jellemzői: például a vizit dátuma és időpontja, a páciens vérnyomása, vagy a beteg testhőmérséklete.

- *Gondoskodjunk róla, hogy minden táblának legyen elsődleges kulcsa!* Két oka van annak, hogy minden táblához kell elsődleges kulcsot rendelnünk. Először is, az elsődleges kulcs egyedileg azonosítja a táblában található összes rekordot. Másodszor, segítségükkel alakítunk ki kapcsolatokat a táblák között. Ha nem rendelünk elsődleges kulcsot minden táblához, az adatok következetessége sérülhet, és a többtáblás SQL-lekérdezések egyes fajtáival is gondjaink támadhatnak. Az elsődleges kulcsok helyes megadásához a fejezet későbbi részében adunk tanácsokat.
- *Gondoskodjunk róla, hogy egyetlen tábla se tartalmazzon többrészes vagy többértékű mezőket!* Elméletileg ezeket a problémákat már kiküszöböltük, amikor finomhangoltuk a mezőszerkezetet, mindazonáltal nem árt még egyszer felülvizsgálni a mezőket, hogy meggyőződjünk róla, hogy minden többrészes és többbértékű mezőt megszüntettünk.
- *Gondoskodjunk róla, hogy a táblákban ne legyenek számított mezők!* Bár meg lehetünk győződve róla, hogy a jelenlegi táblaszerkezetben már nincsenek számított mezők, előfordulhat, hogy a mezők finomhangolása során átsiklottunk egy-kettő felett. Most jó alkalom nyílik arra, hogy még egyszer áttekintsük a táblaszerkezeteket és eltávolítsunk minden olyan számított mezőt, amiről esetleg megfeledkeztünk.
- *Gondoskodjunk róla, hogy egyetlen táblában se legyenek feleslegesen többször szereplő mezők!* A rosszul megtervezett táblák egyik fő ismertetőjele, hogy más táblák mezőinek másolatait tartalmazzák. A mezők megkettőzésére két okunk lehet: hivatkozni szeretnénk egy adatra, vagy jelezni szeretnénk, hogy egy adott érték több példányban is létezik. Az ilyen többször szereplő mezők azonban mindenféle nehézséget támasztanak az adatok kezelése során, és az információk kinyerését is megnehezítik a táblákból. A többször szereplő mezők kérdésével ezért az alábbiakban külön is foglalkozunk.

A feleslegesen többször szereplő mezők feloldása

A megfelelő felépítés biztosításának valószínűleg a többször szereplő mezők kezelése a legnehezebb része. Lássunk néhány példát, amelyek szemléltetik, hogy miként oldhatjuk fel helyesen a másolt mezőket tartalmazó táblákat!

A 2.7. ábrán egy olyan táblát láthatunk, amelynek másolt mezői hivatkozásként szolgálnak.

Ebben az esetben a StaffLastName és a StaffFirstName a Classes táblában is szerepel, hogy aki ezt a táblát nézi, lássa az adott órához tartozó tanár nevét. Ugyanakkor a Classes és a Staff táblák között fennálló egy-a-többhöz kapcsolat miatt (egy tanár több órát is tarthat, de egy óra csak egy adott tanárhoz tartozhat) ezek a mezők feleslegesek. A két tábla között a kapcsolatot a StaffID teremti meg, maga a kapcsolat pedig lehetővé teszi, hogy egy SQL-lekérdezésben egyidejűleg mindkét táblából lekérhessünk adatokat. Ezt észben tartva

biztonságosan törölhetjük a StaffLastName és a StaffFirstName mezőt a Classes táblából; ez semmilyen hátrányos következménnyel nem jár. A Classes tábla javított szerkezetét a 2.8. ábrán láthatjuk.

Staff

StaffID	StaffFirstName	StaffLastName	StaffStreetAddress	StaffCity	StaffState	<<*további mezők*>>
98014	Peter	Brehm	722 Moss Bay Blvd.	Kirkland	WA	...
98019	Mariya	Sergienko	901 Pine Avenue	Portland	OR	...
98020	Jim	Glynn	13920 S.E. 40th Street	Bellevue	WA	...
98021	Tim	Smith	30301 166th Ave. N.E.	Seattle	WA	...
98022	Carol	Viescas	722 Moss Bay Blvd.	Kirkland	WA	...
98023	Alaina	Hallmark	Route 2, Box 203 B	Woodinville	WA	...

Ezek a mezők feleslegesek

Classes

ClassID	Class	ClassroomID	StaffID	StaffLastName	StaffFirstName	<<*további mezők*>>
1031	Art History	1231	98014	Brehm	Peter	...
1030	Art History	1231	98014	Brehm	Peter	...
2213	Biological Principles	1532	98021	Smith	Tim	...
2005	Chemistry	1515	98019	Sergienko	Mariya	...
2001	Chemistry	1519	98023	Hallmark	Alaina	...
1006	Drawing	1627	98020	Glynn	Jim	...
2907	Elementary Algebra	3445	98022	Viescas	Carol	...

2.7. ábra

Hivatkozásként szolgáló másolt mezőket tartalmazó tábla

Ha a táblában megtartjuk az említett felesleges mezőket, az adatok következetességének biztosítása komoly gondot jelenthet. Gondoskodnunk kell róla, hogy a Classes tábla StaffLastName és StaffFirstName mezőinek értéke mindig megegyezzen a Staff tábla hasonló mezőinek értékével. Tegyük fel például, hogy egy tanárnő férjhez megy, és úgy dönt, hogy attól a naptól fogva a férjezett nevét használja. Ekkor nem csak a Staff táblában kell módosítanunk a hozzá tartozó rekordot, hanem a Classes táblában is ki kell javítanunk a név minden előfordulását. Ez persze nem lehetetlen feladat (legalábbis technikailag), de így sokkal többet kell dolgoznunk a szükségesnél. A relációs adatbázisok használatának ráadásul éppen az lenne az egyik legnagyobb előnye, hogy egy adatot elég egyszer bevinni az adatbázisba. (Az egyetlen kivétel ez alól a szabály alól, ha egy mezőt arra használunk, hogy kapcsolatot teremtsünk két tábla között.) Mint mindig, most is az a legjobb, ha minden mező felesleges példányait eltávolítjuk az adatbázis tábláiból.

Staff

StaffID	StaffFirstName	StaffLastName	StaffStreetAddress	StaffCity	StaffState	<<*további mezők*>>
98014	Peter	Brehm	722 Moss Bay Blvd.	Kirkland	WA	...
98019	Mariya	Sergienko	901 Pine Avenue	Portland	OR	...
98020	Jim	Glynn	13920 S.E. 40th Street	Bellevue	WA	...
98021	Tim	Smith	30301- 166th Ave. N.E.	Seattle	WA	...
98022	Carol	Viescas	722 Moss Bay Blvd.	Kirkland	WA	...
98023	Alaina	Hallmark	Route 2, Box 203 B	Woodinville	WA	...

Classes

ClassID	Class	ClassroomID	StaffID	<<*további mezők*>>
1031	Art History	1231	98014	...
1030	Art History	1231	98014	...
2213	Biological Principles	1532	98021	...
2005	Chemistry	1515	98019	...
2001	Chemistry	1519	98023	...
1006	Drawing	1627	98020	...
2907	Elementary Algebra	3445	98022	...

2.8. ábra

A másolt hivatkozó mezők feloldása

A 2.9. ábra egy másik tiszta példát mutat egy többször szereplő mezőket tartalmazó táblára, és jól szemlélteti, hogyan szokták hibásan egy adott típusú érték több előfordulását többször szereplő mezőkkel jelezni. Ebben az esetben három Committee (Bizottság) mező rögzíti azoknak a bizottságoknak a nevét, amelyeknek az adott alkalmazott a tagja.

Employees

EmployeeID	EmpLastName	EmpFirstName	Committee1	Committee2	Committee3	<<*további mezők*>>
7004	Gehring	Darren	Steering			...
7005	Kennedy	John	ISO 9000	Safety		...
7006	Thompson	Sarah	Safety	ISO 9000	Steering	...
7007	Wilson	Jim				...
7008	Seidel	Manuela	ISO 9000			...
7009	Smith	David	Steering	Safety	ISO 9000	...
7010	Patterson	Neil				...
7011	Viescas	Michael	ISO 9000	Steering	Safety	...

2.9. ábra

Adott típusú értékek több előfordulását jelző többször szereplő mezőket tartalmazó tábla

Nem túl nehéz kitalálni, hogy ezek a többször szereplő mezők miért okoznak majd gondot. Az egyik probléma a Committee mezők száma a táblában. Mi történik, ha néhány alkalmazott négy bizottságba is bekerül? És egyáltalán honnan tudjuk előre, hogy hány Committee mezőre lesz szükség? Ha egyes alkalmazottak háromnál több bizottságban foglalnak helyet, további Committee mezőket kell a táblához adnunk.

A második probléma az információk kinyerésével kapcsolatos. Hogyan nyerjük ki azokat az alkalmazottakat, akik jelenleg az ISO 9000 bizottság tagjai? A feladat nem lehetetlen, de nem is egyszerű. A kérdés pontos megválaszolásához három különálló lekérdezést kell végrehajtanunk (vagy meg kell adnunk egy keresési feltételt, amely három különböző mezőt vizsgál), mert nem tudhatjuk, hogy az ISO 9000 érték a három Committee mező közül melyikben tárolódik. Így aztán a feladat végrehajtása a valóban szükségesnél több időt és erőfeszítést igényel.

A harmadik problémát az adatok rendezése jelenti. Az adatokat semmilyen praktikus módszerrel nem tudjuk bizottság szerint rendezni, és a bizottságok nevét sem tudjuk sehogy helyesen ábécésorrendbe állítani. Ezek apró problémáknak tűnnek, de igencsak frusztrálóak lehetnek, ha az adatokat valamilyen rendezett formában szeretnénk áttekinteni.

Employees

EmployeeID	EmpLastName	EmpFirstName	EmpCity	*<<további mezők>>*
7004	Gehring	Darren	Chico	...
7005	Kennedy	John	Portland	...
7006	Thompson	Sarah	Lubbock	...
7007	Wilson	Jim	Salem	...
7008	Seidel	Manuela	Medford	...
7009	Smith	David	Fremont	...
7010	Patterson	Neil	San Diego	...
7011	Viescas	Michael	Redmond	...

Committee_Members

EmployeeID	Committee
7004	Steering
7005	ISO 9000
7005	Safety
7006	Safety
7006	ISO 9000
7006	Steering
7008	ISO 9000
7009	Steering

2.10. ábra

A javított Employees tábla és az új Committee_Members tábla

Ha figyelmesen megnézzük a 2.9. ábrán látható Employees táblát, hamar rájöhetünk, hogy az alkalmazottak és az őket a tagjaik sorában tudó bizottságok között valójában több-a-többhöz kapcsolat áll fenn. Egy alkalmazott bármennyi bizottságnak a tagja lehet, és egy bizottság is bármennyi tagból állhat. Ezért ezeket a többször szereplő mezőket is ugyanúgy oldhatjuk fel, mint bármely más több-a-többhöz kapcsolatot: egy kapcsoló tábla létrehozásával. Az Employees tábla esetében a kapcsoló táblát úgy hozzuk létre, hogy lemásoljuk az elsődleges kulcsot (EmployeeID), és hozzáveszünk egyetlen Commitee mezőt. Adjunk megfelelő nevet az új táblának (például Committee_Members, vagyis Bizottsági tagok),

tegyük összetett elsődleges kulccsá az EmployeeID és Committee mezőket, töröljük a Committee mezőket az Employees táblából, és készen is vagyunk. (Az elsődleges kulcsokról a fejezet későbbi részében részletesebben is tanulunk.) A javított Employees táblát és az új Committee_Members táblát a 2.10. ábrán láthatjuk.

Bár az eredeti Employees táblában többször szereplő mezőket feloldottuk, még nem vagyunk teljesen készen. Ha felidézzük, hogy a bizottságok és a tagjaikként dolgozó alkalmazottak között több-a-többhöz kapcsolat van, joggal merülhet fel a kérdés, hogy hol van a Committees (Bizottságok) tábla? Nos, ilyen táblánk nincs – egyelőre. A bizottságoknak nagy valószínűséggel más olyan jellemzőik is vannak, amelyeket rögzíteni szeretnénk: például a terem száma vagy neve, ahol a bizottság ülésezik, vagy a hónapnak az a napja, amikor az ülésekre sor kerül. Hozzunk hát létre egy igazi Committees táblát, amely olyan mezőket tartalmaz, mint a CommitteID (Bizottságazonosító), a CommitteeName (Bizottság neve), a MeetingRoom (Ülésterem) és a MeetingDay (Ülésnap). Ha az új táblával készen vagyunk, cseréljük a Committee_Members tábla Committee mezőjét az új Committees tábla CommitteeID mezőjére. A végső táblaszerkezeteket a 2.11. ábrán láthatjuk.

Employees

EmployeeID	EmpLastName	EmpFirstName	EmpCity	<<*további mezők*>>
7004	Gehring	Darren	Chico	...
7005	Kennedy	John	Portland	...
7006	Thompson	Sarah	Lubbock	...
7007	Wilson	Jim	Salem	...
7008	Seidel	Manuela	Medford	...
7009	Smith	David	Fremont	...
7010	Patterson	Neil	San Diego	...
7011	Viescas	Michael	Redmond	...

Committee_Members

EmployeeID	CommitteeID
7004	103
7005	104
7005	102
7006	102
7006	104
7006	103
7008	104
7009	103

Committees

CommitteeID	CommitteeName	MeetingRoom	MeetingDay
100	Budget	11-C	Tuesday
101	Christmas	9-F	Monday
102	Safety	12-B	Monday
103	Steering	12-D	Tuesday
104	ISO 9000	Main-South	Wednesday

2.11. ábra

A végső Employees, Committee_Members és Committees táblaszerkezetek

A táblák ilyetén felépítése hatalmas előnyt jelent, mert így egy tagot akárhány bizottsághoz kapcsolhatunk, és egy bizottságba is felvehetünk akárhány alkalmazottat, majd egy SQL-lekérdezéssel egyidejűleg mindhárom táblából megtekinthetünk információkat.

Most már közeledünk afelé, hogy befejezzük a táblaszerkezeteink finomhangolását. Az utolsó feladatunk az, hogy gondoskodjunk róla, hogy az egyes táblákon belül minden rekord egyedileg azonosítható legyen, és hogy magát a táblát is képesek legyünk azonosítani a teljes adatbázisban.

A kulcs az azonosítás

Ahogy az 1. fejezetben megtanultuk, az elsődleges kulcs az egyik legfontosabb elem, mert ez az, ami egyedileg azonosítja a rekordokat a táblában, és a táblát is a teljes adatbázisban. Ezen kívül két tábla között is ez teremti meg a kapcsolatot, ezért az elsődleges kulcs szerepét nem lehet eléggé hangsúlyozni: az adatbázis minden táblájának rendelkeznie kell eggyel.

Meghatározása szerint az elsődleges kulcs egy mező vagy mezőcsoport, amely egyedileg azonosít minden rekordot egy táblában. Amennyiben a kulcs egyetlen mezőből áll, *egyszerű elsődleges kulcsról* beszélünk (vagy röviden elsődleges kulcsról), míg ha két vagy több mező alkotja, *összetett elsődleges kulcsnak* nevezzük. Amikor csak lehetséges, egyszerű elsődleges kulcsot határozzunk meg, mert hatékonyabb, és sokkal könnyebben használható a táblák közötti kapcsolatok megteremtésére. Összetett elsődleges kulcsot csak akkor alkalmazzunk, amikor valóban erre van szükség (például ha egy kapcsoló táblát szeretnénk meghatározni és létrehozni).

Elsődleges kulcsként meglevő mezőt vagy mezőket is használhatunk, amennyiben azok az alábbi ellenőrzőlista valamennyi feltételét kielégítik. Ha egy mező vagy mezőcsoport nem felel meg *minden* követelménynek, használjunk egy másik mezőt, vagy határozzunk meg egy új mezőt a tábla elsődleges kulcsaként. Szánjuk rá az időt, és az alábbi ellenőrzőlista segítségével állapítsuk meg, hogy az adatbázisunk minden elsődleges kulcsa megfelelő-e:

- *A mezők egyedileg azonosítanak minden rekordot a táblában?* Egy táblában minden rekord a tábla tárgyának egy-egy példányát jelképezi. Egy jó elsődleges kulcs biztosítja, hogy a tábla minden rekordját pontosan azonosítani lehessen, illetve hogy képesek legyünk a rekordokra pontosan hivatkozni az adatbázis más tábláiból, valamint segít elkerülni, hogy a táblába többször szereplő rekordok kerüljenek.
- *A mező vagy mezőcsoport egyedi értékeket tartalmaz?* Amíg az elsődleges kulcs mező értékei egyediek, biztosak lehetünk benne, hogy a tábla nem fog többször szereplő rekordokat tartalmazni.
- *Lehetnek ezekben a mezőkben ismeretlen értékek?* Ez nagyon fontos kérdés, mert az elsődleges kulcs nem tartalmazhat ismeretlen értékeket. Ha úgy gondoljuk, hogy – bár valószínűtlen – előfordulhat, hogy a mező ismeretlen értékeket is tartalmaz, azonnal zárjuk ki az elsődleges kulcs szerepéből.

- *Megengedett, hogy ezekből a mezőkből hiányozzon az érték?* Ha erre a kérdésre igennel felelünk, a mezőt nem használhatjuk elsődleges kulcsként. Ha egy mezőben nem kötelező értéknek szerepelnie, az azt jelenti, hogy bizonyos esetekben ismeretlen értéke is lehet, márpedig ahogy az előző kérdésnél említettük, az elsődleges kulcs nem tartalmazhat ismeretlen értékeket.
- *Többrészes mezőről van szó?* Bár mostanra már meg kellett szabadulnunk minden többrészes mezőtől, érdemes még egyszer feltenni magunknak a kérdést. Ha korábban elsiklottunk egy többrészes mező felett, oldjuk fel most, és próbáljunk másik mezőt elsődleges kulcsként alkalmazni, vagy a szétválasztással újonnan létrehozott mezőket együtt, összetett elsődleges kulcsként használjuk.
- *Módosítható ezeknek a mezőknek az értéke?* Az elsődleges kulcs mezők értékének állandónak kell lennie, vagyis e mezők értéke nem változhat meg, hacsak nincs erre igazán nyomós okunk. Ha egy mező értéke megváltozhat, a mező nehezen lesz képes megfelelni az ellenőrzőlista többi követelményének.

Ahogy korábban említettük, egy mezőnek vagy mezők kombinációjának a fenti ellenőrzőlista minden követelményét tökéletesen ki kell elégítenie ahhoz, hogy elsődleges kulcsként használhassuk. A 2.12. ábrán a Pilots tábla elsődleges kulcsa a PilotID – de megfelel a PilotID az ellenőrzőlista minden pontjának? Ha igen, az elsődleges kulcs megfelelő, de ha nem, vagy módosítanunk kell, hogy eleget tegyen a feltételeknek, vagy másik mezőt kell választanunk elsődleges kulcsnak.

Pilots

PilotID	PilotFirstName	PilotLastName	HireDate	Position	PilotAreaCode	PilotPhone
25100	Sam	Alborous	1994-07-11	Captain	206	555-3982
25101	Jim	Wilson	1994-05-01	Captain	206	555-6657
25102	David	Smith	1994-09-11	FirstOfficer	915	555-1992
25103	Kathryn	Patterson	1994-07-11	Navigator	972	555-8832
25104	Michael	Hernandez	1994-05-01	Navigator	360	555-9901
25105	Kendra	Bonnicksen	1994-09-11	Captain	206	555-1106

2.12. ábra

A PilotID megfelelő elsődleges kulcs?

Itt a PilotID éppenséggel megfelelő elsődleges kulcs, mert az ellenőrzőlista valamennyi követelményének eleget tesz. De mit történik, ha nem találunk olyan mezőt, ami megfelelne elsődleges kulcsnak? Vegyük például a 2.13. ábrán látható Employees táblát: van a táblában olyan mező, amit elsődleges kulcsként használhatnánk?

Teljesen világos, hogy ez a tábla nem tartalmaz olyan mezőt (vagy mezőcsoportot), ami elsődleges kulcs lehetne, az EmpPhone kivételével ugyanis minden mezőben vannak azonos értékek, ráadásul az EmpZip, az EmpAreaCode és az EmpPhone ismeretlen

értéket is tartalmaz. Kísértést érezhetünk, hogy az EmpLastName és az EmpFirstName kombinációját használjuk kulcsként, de nincs rá biztosíték, hogy egyszer csak nem veszünk majd fel valakit, akinek szintén Jim Wilson vagy David Smith a neve. Ezen kívül, mivel a tábla minden mezőjében megváltozhatnak az értékek, nyilvánvaló, hogy egyik mezőt sem használhatjuk a tábla elsődleges kulcsaként.

Employees

EmpLastName	EmpFirstName	EmpCity	EmpState	EmpZip	EmpAreaCode	EmpPhone	HireDate
Gehring	Darren	Chico	CA	95926			1998-12-31
Kennedy	John	Portland	OR	97208	503	555-2621	1998-05-01
Thompson	Sarah	Redmond	WA	98052	425	555-2626	1998-09-11
Wilson	Jim	Salem	OR				1998-12-27
Seidel	Manuela	Medford	OR	97501	541	555-2641	1998-05-01
Smith	David	Fremont	CA	94538	510	555-2646	1998-09-11
Patterson	Neil	San Diego	CA	92199	619	555-2541	1998-05-01
Viescas	Michael	Redmond	WA	98052	425	555-2511	1998-09-11
Viescas	David	Portland	OR	97207	503	555-2633	1998-10-15

2.13. ábra

Ennek a táblának van elsődleges kulcsa?

Employees

EmployeeID	EmpLastName	EmpFirstName	EmpCity	EmpState	EmpZip	<<*további mezők*>>
98001	Gehring	Darren	Chico	CA	95926	...
98002	Kennedy	John	Portland	OR	97208	...
98003	Thompson	Sarah	Redmond	WA	98052	...
98004	Wilson	Jim	Salem	OR		...
98005	Seidel	Manuela	Medford	OR	97501	...
98006	Smith	David	Fremont	CA	94538	...
98007	Patterson	Neil	SanDiego	CA	92199	...
98008	Viescas	Michael	Redmond	WA	98052	...
98009	Viescas	David	Portland	OR	97207	...

2.14. ábra

Az Employees tábla az új mesterséges elsődleges kulccsal

Mit tehetünk ilyen esetben? Nos, létrehozunk egy mesterséges elsődleges kulcsot, vagyis meghatározunk és a táblához adunk egy mezőt, amelynek az egyetlen szerepe, hogy a tábla elsődleges kulcsa legyen. Ennek az az előnye, hogy olyan mezőt hozhatunk létre, amely biztosan megfelel az ellenőrzőlistának. Miután a mezőt a táblához adtuk, kijelöljük

elsődleges kulcsként, és készen is vagyunk – ennyi az egész. A 2.14. ábrán megtekinthetjük, hogyan fest az Employees tábla egy EmployeeID nevű mesterséges elsődleges kulccsal.

Megjegyzés *Bár mesterséges elsődleges kulcsokkal könnyen megoldhatjuk az ilyen problémákat, ezek a mezők nem zárják ki, hogy a táblában többször szereplő adatok legyenek. Ha valaki például felvesz egy rekordot egy John Kennedy nevű személy számára a táblába, és új egyedi mesterséges EmployeeID-értéket rendel hozzá, honnan fogjuk tudni, hogy a második John Kennedy nem azonos-e a táblában már szereplő 98002-es alkalmazottal?*

A megoldás az, hogy az alkalmazás kódját kiegészítjük egy ellenőrző eljárással, ami megvizsgálja, hogy egy név nem szerepel-e többször, és figyelmezteti a felhasználót. Sok adatbázisrendszerben lehetőségünk van úgynevezett kioldóként (trigger) megírni egy ilyen ellenőrző kódot, amit az adatbázis-kezelő minden alkalommal önműködően lefuttat, amikor egy sor módosul, új sor kerül a táblába, vagy törlünk egy sort onnan. A kioldók tárgyalása azonban túlmutat könyvünk keretein, ezért ha többet szeretnénk tudni róluk, a részleteket az adatbázisrendszerünk leírásában kell keresnünk.

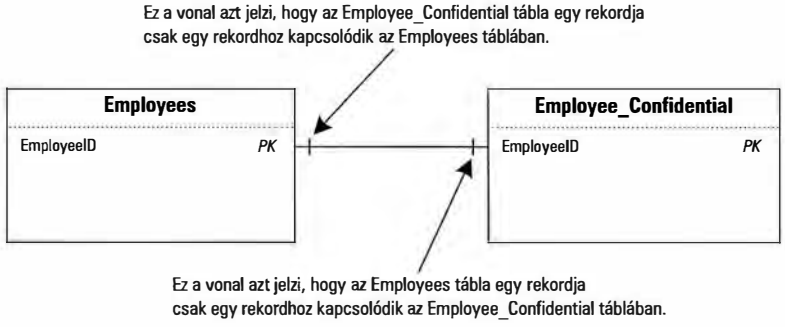
Most már minden megtettünk annak érdekében, hogy megerősítsük és finomhangoljuk a tábláink szerkezetét. A következő részben azt vizsgáljuk, hogy miként gondoskodhatunk arról, hogy a táblakapcsolatok is helyesek legyenek.

Szilárd kapcsolatok kialakítása

Az 1. fejezetben megtanultuk, hogy két tábla között akkor áll fenn kapcsolat, ha az első tábla rekordjai valamilyen módon összefüggnek a második tábla rekordjaival. Arról is beszéltünk, hogy maga a kapcsolat háromféle lehet: egy-az-egyhez, egy-a-többhöz és több-a-többhöz. A kapcsolatok egyes fajtáinak kialakítását is tanultuk – az alábbiakban röviden ezt ismételjük át.

Megjegyzés *Az ebben a részben látható diagramok jelölései Mike Hernandez Database Design for Mere Mortals (Addison-Wesley, 2004; magyarul: Adatbázis-tervezés: A relációs adatbázisok alapjairól földi halandóknak. Kiskapu, 2004) című könyvének diagramkészítési módszerét követik. A PK elsődleges kulcsot (primary key) jelöl, az FK idegen kulcsot (foreign key), a CPK pedig egy olyan mezőt, amely egy összetett elsődleges kulcs (composite primary key) része.*

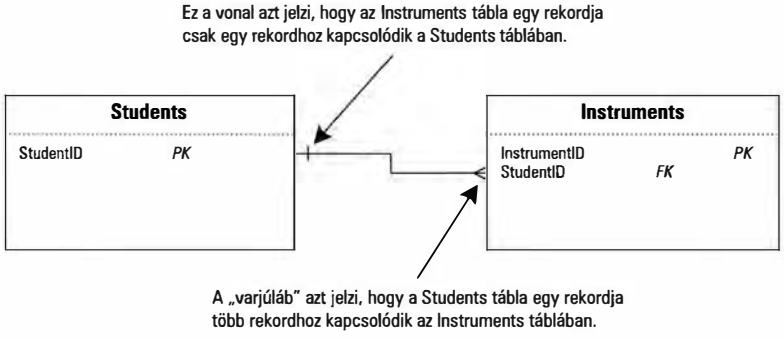
- **Egy-az-egyhez kapcsolatot** úgy alakítunk ki, hogy vesszük az elsődleges tábla elsődleges kulcsát, és beszúrjuk az alárendelt táblába, ahol idegen kulcs lesz belőle. Ez különleges fajta kapcsolat, mert sok esetben az idegen kulcs az alárendelt tábla elsődleges kulcsaként is viselkedik. Ennek a fajta kapcsolatnak a diagramját a 2.15. ábra mutatja.



2.15. ábra

Az egy-az-egyhez kapcsolat diagramja

- **Egy-a-többhöz kapcsolatot** úgy hozunk létre, hogy vesszük a kapcsolat „egy" oldalán álló tábla elsődleges kulcsát, és beszúrjuk azt a „több" oldalon levő táblába, ahol idegen kulcs lesz belőle. Ennek a fajta kapcsolatnak a diagramját a 2.16. ábra mutatja.



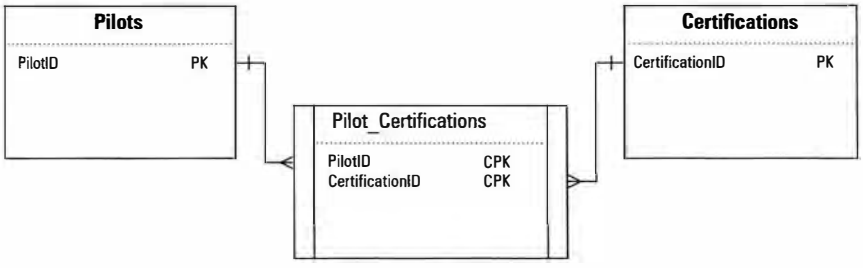
2.16. ábra

Az egy-a-többhöz kapcsolat diagramja

- **Több-a-többhöz kapcsolatot** úgy hozunk létre, hogy létrehozunk egy kapcsoló táblát. A kapcsoló táblát úgy határozzuk meg, hogy lemásoljuk a kapcsolatban részt vevő mindkét tábla elsődleges kulcsát, és ezek segítségével alkotjuk meg az új tábla szerkezetét. A kulcsmezők két jól elkülöníthető szerepet töltenek be: együttesen a kapcsoló tábla összetett elsődleges kulcsát adják, külön-külön pedig idegen kulcsokként viselkednek. Ennek a fajta kapcsolatnak a diagramját a 2.17. ábra mutatja.

Ahhoz, hogy biztosak lehessünk benne, hogy az adatbázis táblái közötti kapcsolatok szilárdak, minden kapcsolathoz tulajdonságokat kell meghatároznunk. Ezek a tulajdonságok adják meg, hogy mi történik, amikor törlünk egy rekordot, hogy milyen szerepe van az adott táblának a kapcsolatban, és hogy az egyes táblák milyen mértékben vesznek részt a kapcsolatban.

A több-a-többhöz kapcsolatokat mindig egy kapcsoló tábla segítségével oldjuk fel. Ebben a példában a Pilot_Certifications a kapcsoló tábla. Egy pilóta akárhány jogosítvánnnyal rendelkezhet, és egy jogosítványt is bármennyi pilótához kapcsolhatunk.



2.17. ábra

A több-a-többhöz kapcsolat diagramja

Mielőtt azonban belefognánk a kapcsolatok tulajdonságainak tárgyalásába, egy dolgot világossá kell tennünk: a tulajdonságokat általános logikai rendszerben mutatjuk be. Ezek a tulajdonságok azért fontosak, mert lehetővé teszik, hogy kikényszerítsük a kapcsolatok következetességét (erre hivatkoznak egyes adatbázisrendszerek *hivatkozási épségként*), programtól függően azonban más-más módon valósíthatjuk meg őket. Utána kell néznünk az adatbázis-kezelő szoftverünk leírásában, hogy a kapcsolattulajdonságok támogatottak-e, és ha igen, hogyan határozhatjuk meg azokat.

Törlési szabály meghatározása

A *törlési szabályok* adják meg, hogy mi történik, ha egy felhasználó olyan kérelmet ad ki, amellyel egy egy-az-egyhez kapcsolat elsődleges táblájából vagy egy egy-a-többhöz kapcsolat „egy" oldalán álló táblából szeretne rekordokat törölni. A szabály meghatározásával az árva rekordokkal szemben védhetjük meg magunkat. (Az *árva rekordok* olyan rekordok, amelyek egy egy-az-egyhez kapcsolat alárendelt táblájában találhatók, és nem kapcsolódik hozzájuk rekord az elsődleges táblában, illetve olyan rekordok, amelyek egy egy egy-a-többhöz kapcsolat „több" oldalán álló táblában találhatók, és nincs párjuk az „egy" oldalon levő táblában.)

Egy kapcsolatra kétféle törlési szabályt határozhatunk meg: *korlátozó* és *többszintű* törlést.

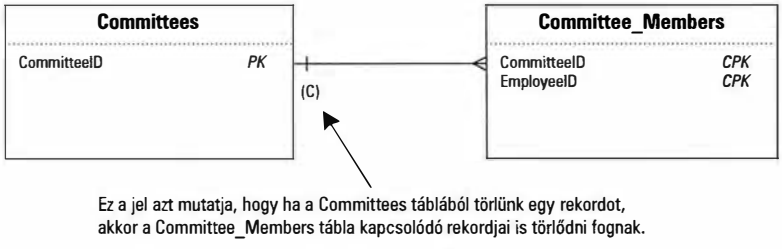
- A **korlátozó törlési szabály** nem engedi meg a kívánt rekord törlését, amennyiben az egy-az-egyhez kapcsolat alárendelt táblájában vagy az egy-a-többhöz kapcsolat „több" oldalán álló táblában rekordok kapcsolódnak hozzá. Minden kapcsolódó rekordot még az előtt el *kell* távolítanunk, hogy a kérdéses rekordot törölhetnénk. Azokban az adatbázisrendszerekben, amelyek megengedik a kapcsolati szabályok meghatározását, általában ez az alapértelmezett, sőt néha az egyetlen választható típus.

- Ha **többszintű törlési szabályt** léptetünk életbe, egy a kapcsolat „egy" oldalán álló rekord törlésének hatására a rendszer automatikusan töröl minden kapcsolódó rekordot az egy-az-egyhez kapcsolat alárendelt táblájából, illetve az egy-a-többhöz kapcsolat „több" oldalán álló táblából. Ezt a szabályt nagyon körültekintően használjuk, nehogy végül olyan rekordokat is töröljünk, amelyeket nem szerettünk volna! A többszintű törlést nem minden adatbázisrendszer támogatja.

Az alkalmazott törlési szabály típusától függetlenül mindig nagyon gondosan vizsgáljuk meg a kapcsolatot, hogy megállapítsuk, melyik fajta szabály a megfelelő. Ezt egy nagyon egyszerű kérdéssel eldönthetjük. Először válasszunk ki egy táblapárt, majd tegyük fel a következő kérdést: „Ha egy rekord törlődik az [elsődleges vagy az „egy" oldalon álló tábla neve] táblából, törlődjenek az [alárendelt vagy a „több" oldalon álló tábla neve] tábla kapcsolódó rekordjai is?".

A kérdést azért általános formában fogalmaztuk meg, hogy megértsük a mögötte húzódó elvet. A kérdés szögletes zárójelben levő kifejezéseit a megfelelő táblanevekre kell cserélnünk, tehát egy konkrét esetben a kérdés így fog festeni: „Ha egy rekord törlődik a Committees táblából, törlődjenek a Committee_Members tábla kapcsolódó rekordjai is?".

Ha a fenti kérdésre nemmel felelünk, korlátozó törlési szabályt alkalmazzunk; ha igennel, többszintű törlési szabályt. Végsősoron a válasz nagymértékben függ attól, hogy miként használjuk az adatbázisban tárolt adatokat. Ezért kell gondosan megvizsgálnunk a kapcsolatot, és megbizonyosodnunk arról, hogy a megfelelő típusú szabályt választottuk. A példában említett kapcsolat törlési szabályának diagramját a 2.18. ábrán láthatjuk. Megfigyelhetjük, hogy a korlátozó (restrict) törlési szabály jelzése (R), míg a többszintű (cascade) törlési szabályé (C).



2.18. ábra

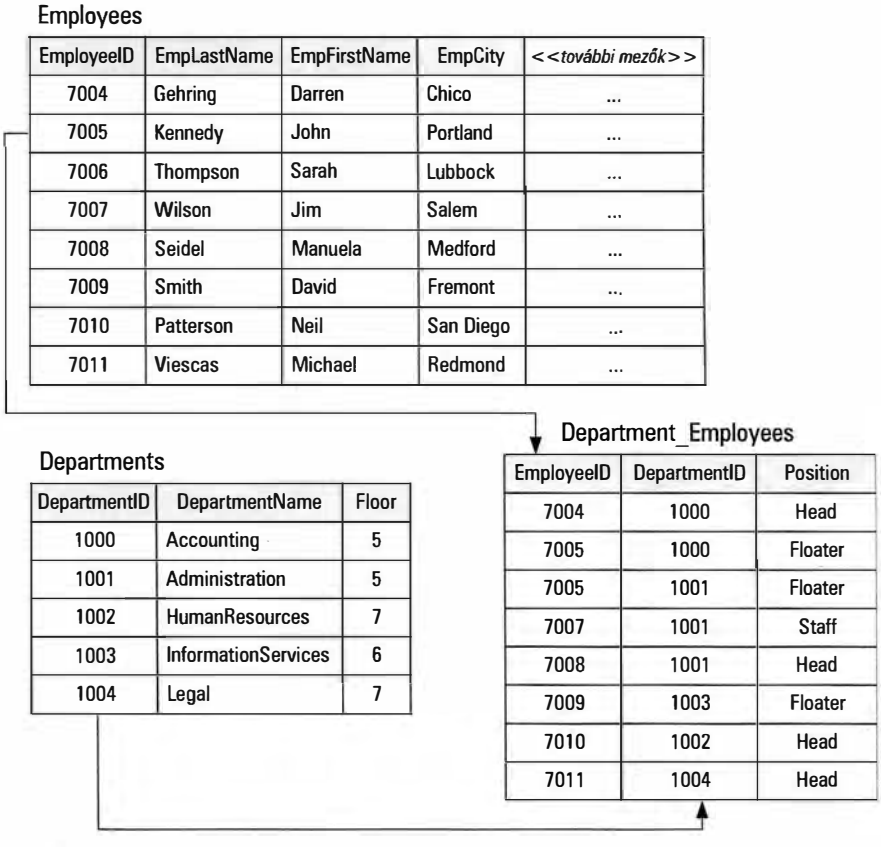
A Committees és Committee_Members táblákra vonatkozó törlési szabály diagramja

A táblák szerepének beállítása

Amikor két táblát összekapcsolunk, mindkét tábla adott szerepet kap a kapcsolatban. A *szerep* (a *részvétel típusa*), amelyet egy adott táblához rendelünk, határozza meg, hogy a táblában léteznie kell-e rekordnak, mielőtt bevihetnénk egy rekordot a másik táblába. Kétféle szerep lehetséges:

- **Kötelező** – A táblában legalább egy rekordnak lennie kell, mielőtt bármilyen rekordot bevihetnénk a másik táblába.
- **Nem kötelező** – Nem követelmény, hogy a táblában legyen rekord, mielőtt bármilyen rekordot bevihetnénk a másik táblába.

Az, hogy a táblákhoz milyen szerepet rendelünk, legnagyobb részt az adott adatbázisrendszer üzleti logikájától függ. Tegyük fel például, hogy egy több részlegből álló nagy cégnek dolgozunk, és a cég számára készített adatbázisban található egy Employees (Alkalmazottak), egy Departments (Részlegek) és egy Department_Employees (Részleg alkalmazottai) tábla. Az alkalmazottak minden fontos adatát az Employees tábla tárolja, míg a részlegekét a Departments tábla. A Department_Employees kapcsoló tábla, amely lehetővé teszi, hogy egy adott alkalmazottat tetszőleges számú részleghez kapcsoljunk. Az említett táblákat a 2.19. ábrán láthatjuk. (Az ábrán egyszerű nyilak mutatnak a kapcsolat „több" oldalára.)

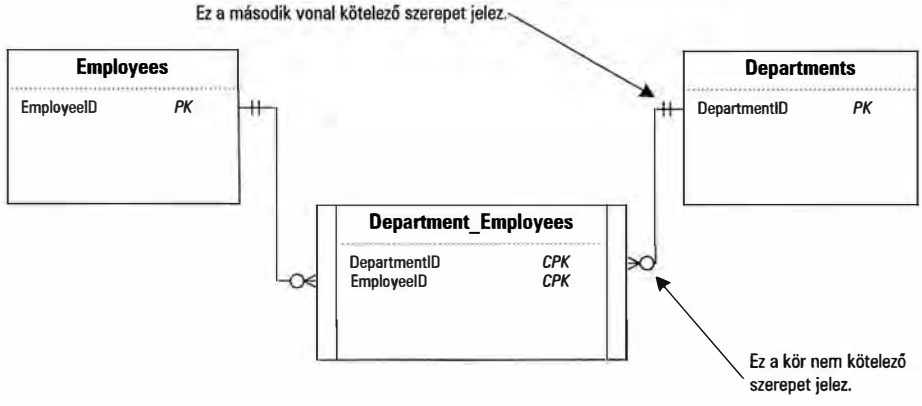


2.19. ábra

Az Employees, Departments és Department_Employees táblák

A legutóbbi értekezleten azt a feladatot kaptuk, hogy egyes alkalmazottakat osszunk be az új Research and Development (Kutatás-fejlesztés) részlegbe. A problémát a következő jelenti: szeretnénk biztosak lenni benne, hogy az új részleg a Departments táblába kerül, hogy alkalmazottakat rendelhessünk a részleghez a Department_Employees táblában. Ez az, ahol a táblák szerepe jelentést nyer. Állítsuk a Departments tábla szerepét „kötelezőre", a Department_Employees tábla szerepét pedig „nem kötelezőre". Ezekkel a beállításokkal biztosítjuk, hogy egy részlegnek léteznie kell a Departments táblában, mielőtt alkalmazottakat vehetnénk fel bele a Department_Employees táblában.

Ugyanúgy, ahogy a törlési szabály esetében, most is gondosan vizsgáljuk meg, hogy a kapcsolatban részt vevő tábláknak milyen típusú szerep felel meg. A szerepeket úgy ábrázolhatjuk diagramon, ahogy a 2.20. ábra mutatja.



2.20. ábra

A Departments és Department_Employees táblák szerepének diagramja

A részvétel mértékének beállítása

Most, hogy meghatároztuk, ***hogyan*** vesznek részt az egyes táblák a kapcsolatban, meg kell állapítanunk azt is, hogy ***milyen mértékben*** teszik ezt. Ezt úgy végezhetjük el, hogy meghatározzuk, hogy az egyik táblában legalább és legfeljebb hány rekord kapcsolódhat a másik tábla egy rekordjához. Ez a művelet a tábla ***részvételi mértékének*** meghatározása. A részvétel mértékét egy adott tábla esetében két zárójelben álló számmal adjuk meg, amelyeket vessző választ el egymástól. Az első szám a kapcsolódó rekordok lehetséges legkisebb számát jelzi, míg a második szám azok lehetséges legnagyobb számát. Az (1,12) részvételi mérték például azt mutatja, hogy legalább egy, de legfeljebb tizenkettő kapcsolódó rekord lehet.

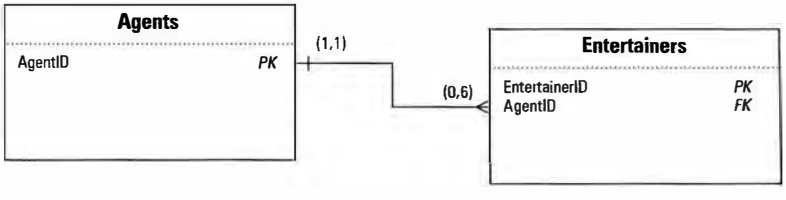
Az adatbázis különböző tábláihoz meghatározott részvételi mérték nagyban függ attól, hogy a cégen belül miként használjuk és tekintjük meg az adatokat. Tegyük fel például, hogy egy rendezvényszervező cég ügynökeként dolgozunk, és az adatbázisunkban talál-

ható egy Agents (Ügynökök) és egy Entertainers (Előadók) tábla. Tegyük fel továbbá, hogy a két tábla egy-a-többhöz kapcsolatban áll egymással: az Agents tábla egy rekordja az Entertainers tábla több rekordjához kapcsolódhat, de az Entertainers tábla egy rekordja csak egyetlen rekorddal állhat kapcsolatban az Agents táblából (vagyis általánosságban azt biztosítjuk ezzel, hogy egy előadót csak egy ügynök képviseljen, nehogy rivalizálásra kerüljön sor, ami mindenkinek csak kárt okozna).

A kapcsolat „több" oldalán levő rekordok lehetséges legnagyobb száma szinte minden esetben végtelen. Néha azonban az üzleti szabályok azt diktálják, hogy korlátozzuk ezt a részvételi mértéket, például ha meg akarjuk szabni, hogy egy adott órára legfeljebb hány hallgató iratkozhat fel. Az ügynökség példáját folytatva tegyük fel, hogy a főnök biztos szeretne lenni benne, hogy minden ügynök egyformán jó lehetőségeket kap, a köztük levő versenyt pedig a lehető legkisebb mértékre szeretné visszaszorítani, ezért bevezeti azt a szabályt, hogy egy ügynök legfeljebb hat előadót képviselhet. (Nem biztos benne, hogy ez hosszú távon is működni fog, de azért megpróbálja.) Az új szabály életbe léptetéséhez a két tábla részvételi mértékét az alábbiak szerint kell beállítanunk:

Agents	(1,1) – Egy előadó legalább és legfeljebb is csak egy ügynökhöz rendelhető.
Entertainers	(0,6) – Egy ügynöknek nem feltétlenül kell képviselnie egyetlen előadót sem, viszont adott időben nem képviselhet többet hat előadónál.

A fenti táblák részvételi mértékét a 2.21. ábra diagramja mutatja.



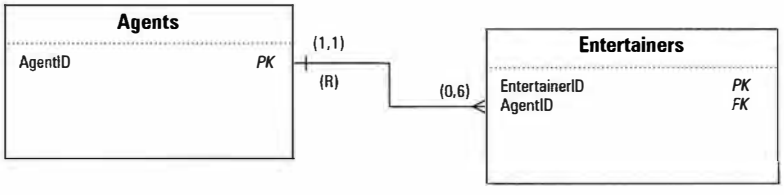
2.21. ábra

Az Agents és Entertainers táblák részvételi mértékének diagramja

Miután beállítottuk a részvétel mértékét, el kell döntenünk, hogy miként érvényesítse (kényszerítse ki) az adatbázisrendszer a kapcsolatot. A kikényszerítés módja az adatbázisrendszer nyújtotta lehetőségektől függ. A legegyszerűbb kikényszerítési mód, amelyet a legtöbb adatabázisrendszer támogat, a „több" oldalon levő tábla idegen kulcs mezőjében levő értékek korlátozása úgy, hogy a felhasználó ne írhasson ebbe a mezőbe olyan értéket, ami nem szerepel az „egy" oldalon levő táblában. Ezt a diagramon egy zárójelben levő R betűvel jelezzük az „egy" oldalon levő táblára mutató kapcsolati vonal mellett (lásd a 2.22. ábrát).

Egyes adatbázisrendszerek lehetővé teszik, hogy olyan szabályt határozzunk meg, amely átviszi a kulcsértéket az „egy" oldalon levő táblából a „több" oldalon levő táblába – (C), vagyis többszintű (cascade) módosítás – , ha a felhasználó megváltoztatja az elsődleges

kulcs értékét az „egy" táblában. Az adatbázisrendszer lényegében kijavítja a „több" tábla kapcsolódó sorainak idegen kulcs értékét, amikor módosítjuk az „egy" tábla elsődleges kulcs értékeit. Egyes adatbázis-kezelők arra is képesek, hogy automatikusan töröljék (D) a „több" tábla kapcsolódó sorait, ha törlünk (delete) egy sort az „egy" táblából. A részleteket az adatbázisrendszerünk leírásából deríthetjük ki.



2.22. ábra

Az Agents és Entertainers táblák kapcsolattulajdonságainak teljes diagramja

Megjegyzés *Ahhoz, hogy ténylegesen kikényszerítsük a részvétel mértékére vonatkozó korlátozásokat, meg kell adnunk egy vagy több kioldót vagy megszorítást az adatbázis meghatározásában (amennyiben az adatbázis-kezelőnk támogatja ezeket a szolgáltatásokat).*

Ennyi az egész?

Az ebben a fejezetben elsajátított módszerek alkalmazásával megtettük a szükséges lépéseket afelé, hogy az adatbázisunkban alapszinten biztosítsuk az adatok következetességét. A következő lépés, hogy megvizsgáljuk, hogyan használják és tekintik meg az adatokat a cégen belül, hogy kialakíthassuk és életbe léptethessük az adatbázis üzleti szabályait. Ahhoz azonban, hogy valóban a legtöbbet hozhassuk ki az adatbázisból, vissza kell térnünk a rajtkockába, és egy megfelelő tervezési módszerrel alaposan meg kell terveznünk az adatbázist. Sajnos ezek a témakörök kívül esnek könyvünk keretein, de a helyes adatbázis-tervezési elveket megtanulhatjuk olyan könyvekből, mint Michael J. Hernandez *Database Design for Mere Mortals* (Addison-Wesley, 2004; magyarul: *Adatbázis-tervezés: A relációs adatbázisok alapjairól földi halandóknak*. Kiskapu, 2004) vagy Thomas Connolly és Carolyn Begg *Database Systems: A Practical Approach to Design, Implementation, and Management* (4. kiadás, Addison-Wesley, 2004) című kötete. A legfontosabb, amit az eszünkbe kell vésnünk, hogy minél szilárdabb az adatbázisunk szerkezete, annál könnyebb lesz mind információkat kinyerni az adatbázis adataiból, mind alkalmazásokat fejleszteni az adatbázishoz.

Összefoglalás

A fejezetet annak a rövid összefoglalásával kezdtük, hogy miért fontos számunkra, hogy az adatbázisunk szerkezete helyes legyen. Megtanultuk, hogy a rosszul megtervezett táblák számos problémát okozhatnak, főként az adatok következetességével kapcsolatban.

Ezt követően a táblák mezőinek finomhangolásával foglalkoztunk. Megtudtuk, hogy nagyon fontos, hogy a mezőinknek megfelelő neveket adjunk, mert így biztosíthatjuk, hogy minden név értelmes legyen, és könnyebben megtalálhatjuk a magában a mezőszerkezetben levő rejtett problémákat is. Most már tudjuk, hogyan kell néhány egyszerű szabály érvényesítésével finomhangolni a mezők szerkezetét. Ezek a szabályok gondoskodnak róla, hogy minden mező a tábla tárgyának csak egy jellemzőjét írja le, az egyes mezők csak egy értéket tartalmazzanak, illetve hogy ne legyen bennük számított érték. A többrészes és többértékű mezők okozta gondokról is beszéltünk, és azt is megtanultuk, hogyan oldhatjuk fel az ilyen mezőket.

A következő kérdés, amit tárgyaltunk, a táblák finomhangolása volt. Megtudtuk, hogy a megfelelő táblanevek éppen olyan fontosak, mint a jó mezőnevek, mégpedig ugyanazokból az okokból. Most már tudjuk, hogyan adhatunk a tábláinknak jelentéssel bíró neveket, és hogyan biztosíthatjuk, hogy minden tábla csak egy tárgyat ábrázoljon. Ez után azokat a szabályokat tárgyaltuk, amelyek segítségével az egyes táblák szerkezetének helyességét biztosíthatjuk. Bár ezek a szabályok látszólag hasonló szerepet töltenek be, mint azok az eljárások, amelyeket a mezők szerkezetének finomhangolásakor alkalmaztunk, láthattuk, hogy a táblaszerkezetek finomhangolását végző szabályok valójában egy újabb szintet jelentenek afelé, hogy a tábláink szerkezete a lehető legegészségesebb legyen.

A következő téma, amellyel foglalkoztunk, az elsődleges kulcsok szerepe volt. Megtanultuk, miért fontos, hogy az adatbázis minden táblája rendelkezzen elsődleges kulccsal, és már azt is tudjuk, hogy az elsődleges kulcsoknak meghatározott tulajdonságokkal kell rendelkezniük, illetve hogy az elsődleges kulcs szerepét betöltő mezőt nagyon gondosan kell kiválasztanunk a táblából. Azt is megtudtuk, hogy mesterséges elsődleges kulcsot is létrehozhatunk, ha a táblában nem található olyan mező, amely az elsődleges kulcsok számára kötelező tulajdonságok mindegyikével rendelkezne.

A fejezetet a szilárd kapcsolatok kialakításának tárgyalásával zártuk. Miután ismétlésképpen újra áttekintettük a kapcsolatok három lehetséges típusát, megtanultuk, hogyan ábrázolhatók ezek a kapcsolatok egy diagramon. Ezt követően megtanultuk, hogyan határozhatunk meg egy törlési szabályt egy adott kapcsolathoz, illetve hogyan jelölhetjük azt a diagramon. Elmondtuk, hogy a törlési szabályok azért fontosak, mert védelmet nyújtanak az árva rekordok megjelenése ellen. Az utolsó két témakör, amelyről szót ejtettünk, a tábláknak a kapcsolatban betöltött szerepe, valamint a táblák részvételének mértéke volt. Megtanultuk, hogy a tábla részvétele lehet kötelező vagy nem kötelező, és hogy két tábla kapcsolatához megadhatjuk a kapcsolódó rekordok legkisebb és legnagyobb számát.

A következő fejezetben az SQL történetéről olvashatunk rövid összefoglalást, illetve arról, hogy a nyelv fejlődése miként ért el a jelenlegi változathoz, az SQL:2003-hoz.

3

Az SQL rövid története

„Csak egy vallás létezik – bár sok változata van."
– George Bernard Shaw
Barátságos és barátságtalan színművek

A fejezet témakörei

- Az SQL eredete
- Az első megvalósítások
- És megszületik egy szabvány...
- Az ANSI/ISO szabvány fejlődése
- Kereskedelmi megvalósítások
- Mit tartogat a jövő?
- Miért érdemes megtanulnunk az SQL használatát?
- Összefoglalás

A történelemírók – gyakran homályosan és megkérdőjelezhető hitelességgel – különféle eseményekről, politikai fondorlatokról és emberi gyengeségekről számolnak be. Ebben a tekintetben az SQL története sem más. Az SQL ilyen-olyan formában már a relációs modell megjelenése óta létezik, és hosszú, ám korántsem makulátlan múltjáról számos beszámoló született. Ebben a fejezetben azonban közelebbi pillantást vetünk az adatbázis-kezelő nyelv eredetére, fejlődésére és jövőjére. Két célunk van: először is képet szeretnénk adni arról, hogy miként ért az SQL olyan nyelvvé, amelyet ma az adatbázisrendszerek többsége használ, valamint azt is érzékeltetni szeretnénk, hogy miért fontos, hogy megtanuljuk az SQL használatát.

Az SQL eredete

Ahogy az 1. fejezetben megtanultuk, a relációs adatbázismodellt Dr. E. F. Codd 1970-ben mutatta be a világnak. Ez után a mérföldkőnek számító pillanat után az egyetemek, kutatólaboratóriumok és más hasonló intézmények igyekeztek egy olyan nyelvet kifejleszteni, amelyet egy a relációs modellt támogató adatbázisrendszer alapjaként használhatnának.

A kezdeti erőfeszítések gyümölcseként az 1970-es évek elején-közepén több nyelv is született, később pedig kidolgozták az SQL alapjait és kifejlesztették a ma használatos SQL alapú adatbázis-kezelőket. De milyen elődei voltak az SQL-nek, hogyan fejlődött, és mit tartogat számára a jövő? Ahhoz, hogy válaszolni tudjunk ezekre a kérdésekre, vissza kell mennünk az időben a kaliforniai San Joséba, az IBM cég Santa Teresa Research Laboratory névre hallgató intézetébe.

Az 1970-es évek elején az IBM System/R néven elindított egy nagyszabású kutatási projektet, amelynek a célja a relációs modell alkalmazhatóságának bizonyítása, valamint a relációs adatbázisok tervezése és megvalósítása terén történő tapasztalatszerzés volt. A kutatók kezdeti erőfeszítései az 1974-es és 1975-ös évben sikeresnek bizonyultak: megalkották a relációs adatbázis alapszintű prototípusát.

Amellett, hogy működőképes relációs adatbázist igyekeztek kifejleszteni, a kutatók annak érdekében is erőfeszítéseket tettek, hogy meghatározzanak egy adatbázis-kezelő nyelvet. Az ilyen nyelv megalkotására tett első kísérletek közül kétségkívül az ebben a laboratóriumban végzett munka bizonyult a legnagyobb kereskedelmi jelentőségűnek. 1974-ben Dr. Donald Chamberlin és munkatársai kifejlesztették a SEQUEL (Structured English Query Langugage, strukturált angol lekérdezőnyelv) nyelvet, amely lehetővé tette a felhasználóinak, hogy világosan meghatározott, az angolhoz hasonló nyelvű mondatokkal kérdezzenek le adatokat egy relációs adatbázisból. Dr. Chamberlin és csapata az új nyelvet először a SEQUEL-XRM nevű mintaadatbázisban valósította meg.

A SEQUEL-XRM sikere és a kezdeti visszajelzések arra bátorították Dr. Chamberlint és társait, hogy folytassák a kutatómunkát. 1976-77-ben teljesen áttervezték a SEQUEL-t, és az új változatot SEQUEL/2-nek nevezték el. Később azonban a nevet jogi okokból SQL-re (Structured Query Language, strukturált lekérdezőnyelv) kellett változtatniuk (valaki más ugyanis már használta a SEQUEL betűszót). Sokan a mai napig úgy ejtik ki az SQL nevét, mint a *sequel* szót, bár a „hivatalos" (a legszélesebb körben elfogadott) kiejtés „es-kú-el" (illetve angolul „esz-kjú-el"). Az SQL több új szolgáltatást is kínált, például támogatta a többtáblás lekérdezéseket, valamint az adatok több felhasználó általi megosztott elérését.

Az SQL megjelenése után nem sokkal az IBM egy új és még nagyratörőbb terv megvalósításába fogott: egy olyan adatbázis-prototípust igyekeztek megalkotni, amely még kézzelfoghatóbbá teszi a relációs modell alkalmasságát. Az új prototípust System R-nek nevezték el, és az SQL egy kiterjedt részhalmazára alapozták. Miután az előkészítő munka nagyja befejeződött, az IBM több belső hálózatára, illetve néhány kiválasztott ügyfelére telepítette a System R-t tesztelés és értékelés céljából, majd a tapasztalatok és a felhasználók visszajelzései alapján számos változtatást eszközölt a rendszeren és az SQL-en. Az IBM 1979-ben fejezte be a kísérletet, ami azzal a megállapítással zárult, hogy a relációs modell valóban alkalmas adatbázis-technológia, amelyben komoly üzleti lehetőségek vannak.

Megjegyzés *Az említett projekt egyik legfontosabb eredményének tulajdonították az SQL kifejlesztését is, pedig az SQL valójában a SQUARE (Specifying Queries As Relational Expressions, lekérdezések meghatározása relációs kifejezésként) kutatási nyelvben gyökerezik. Ezt a nyelvet 1975-ben dolgozták ki (még a System R projekt előtt), és arra tervezték, hogy az angol nyelvűhöz hasonló mondatokkal valósítsanak meg relációs algebrát.*

Felmerülhet bennünk a kérdés, hogy ha az IBM arra a következtetésre jutott, hogy a relációs modellben üzletei lehetőségek rejlenek, miért fejezte be a projektet? Könyvünk egyik szerzője, John, emlékszik rá, hogy látott egy bemutatót a System R-ről az 1970-es évek végén, ami sok tekintetben lenyűgözte, de az akkor rendelkezésre álló hardveren még a legegyszerűbb lekérdezés végrehajtása is percekig tartott. Világos volt, hogy az SQL-nek van jövője, de egyértelműen jobb hardver és szoftver szükségeltetett hozzá, hogy a termék vonzóvá váljon a piac számára.

Az első megvalósítások

Az 1970-es években az IBM kutatólaboratóriumában végzett munkát nagy érdeklődéssel kísérték a különböző műszaki folyóiratok, és az új relációs modell előnyeiről élénken vitáztak az adatbázis-technológiai szemináriumokon. Az évtized vége felé világossá vált, hogy az IBM nagyon szeretne a relációs adatbázismodellre és az SQL-re épülő termékeket fejleszteni, és el is kötelezte magát ebben az irányban. Ez természetesen sok gyártót töprengésre késztetett: vajon mikor dobja piacra az első ilyen termékét az IBM? Egyesek jó érzékkel úgy döntöttek, hogy a lehető leggyorsabban saját termékek fejlesztésébe kezdenek, és nem várják meg, amíg az IBM piacvezetővé válik.

1977-ben a kaliforniai Menlo Park mérnökei megalapították a Relational Software, Inc. nevű céget, azzal a szándékkal, hogy egy új relációs adatbázisterméket készítsenek, amely az SQL-en alapul. A termék az Oracle nevet kapta. Az Oracle 1979-ben került a piacra, két évvel megelőzve az IBM első hasonló termékét, és így az első kereskedelmi forgalomban kapható relációs adatbázis-kezelő rendszerré (RDBMS, relational database management system) vált. Az Oracle egyik előnye az volt, hogy a Digital VAX miniszámítógépein futott, nem pedig a költségesebb IBM-nagygépeken. A Relational Software azóta Oracle Corporationre változtatta a nevét, és a mai napig az egyik piacvezető az RDBMS-szoftvergyártók között.

Mindeközben Michael Stonebraker, Eugene Wong és a kaliforniai Berkeley Egyetem számítógépes laboratóriumának más professzorai ugyancsak a relációs adatbázis-technológia terén végeztek kutatásokat. Az IBM csapatához hasonlóan kifejlesztettek egy relációsadatbázis-prototípust, amelynek a neve INGRES lett. Az INGRES-be beépítették a QUEL (Query Language) nevű adatbázis-kezelő nyelvet is, amely az SQL-lel összevetve strukturáltabb nyelv, de kevésbé támaszkodik a természetes (angol) nyelvű utasításokra. Az INGRES-t végül SQL alapú RDBMS rendszerré alakították át, amikor nyilvánvalóvá vált, hogy az SQL

lesz az adatbázis-kezelő nyelvek szabványa. 1980-ban a Berkeley-ről távozó néhány professzor megalapította a Relational Technology, Inc. céget, és 1981-ben bejelentették az INGRES első kereskedelmi változatának megjelentetését. A Relational Technology azóta számos átalakuláson esett keresztül, és ma a Computer Associates International, Inc. része. Az INGRES azonban még mindig az iparág egyik piacvezető adatbázisterméke.

De térjünk vissza az IBM-hez, akik a saját RDBMS rendszerüket, amelyet SQL/Data System-nek (SQL/DS) neveztek el, 1981-ben jelentették be, és 1982-ben dobták piacra. 1983-ban új változatot készítettek belőle a VM/CMS operációs rendszerre (ez az IBM egyik nagygépeken futó rendszere volt), valamint egy új RDBMS-terméket is bejelentettek Database 2 (DB2) néven, amelyet az IBM nagygépeken a cég MVS operációs rendszerével lehetett használni. A DB2 1985-ben került a piacra, és azóta az IBM vezető RDBMS-termékévé vált, technológiáját pedig a teljes IBM-termékvonalba beépítették. Az IBM egyébként nem változtatta meg a nevét – még mindig IBM-nek hívják.

A System R kutatási projekt óta eltelt több mint 25 év során a relációs modell olyan erővé vált, amely az iparágra szinte minden szinten hatást gyakorol, és többmilliárd dolláros üzletté nőtte ki magát.

És megszületik egy szabvány...

Az adatbázisnyelvek kifejlesztésére irányuló vállalkozások szerteágazóságát figyelembe véve felmerülhet bennünk a kérdés, hogy gondolt-e valaki a szabványosításra. Az adatbázis-kutatók közössége persze megvitatta ennek a lehetőségét, de nem jutottak egyezségre abban, hogy kinek kellene kidolgoznia a szabványt, vagy hogy az melyik nyelven alapuljon. Ezért minden gyártó önállóan folytatta a saját adatbázis-terméke fejlesztését, abban reménykedve, hogy az – és vele a benne használt SQL-nyelvjárás – lesz majd az iparági szabvány.

A vásárlók visszajelzései és a piacon mutatkozó igények sok gyártót arra késztettek, hogy bizonyos elemeket beleépítsenek a saját SQL-nyelvjárásukba, így idővel kialakult egy nem hivatalos szabvány, amely mai szemmel nézve rövid leírás volt, mivel csak azokra az elemekre tért ki, amelyek a különböző SQL-változatokban megegyeztek. Mindazonáltal ez a szabványféleség (bármilyen is volt) megadta az adatbázis-felhasználóknak azokat az alapkövetelményeket, amelyek alapján értékelhették a piacon jelen levő különböző adatbázisprogramokat, és a legfontosabb tudnivalókat is összefoglalta, amelyeket minden adatbázisprogramban hasznosíthattak.

Egyre nagyobb igény jelentkezett egy hivatalos relációs adatbázisnyelv-szabványra, amire válaszul az Amerikai Nemzeti Szabványügyi Intézet (ANSI, American National Standards Institute) 1982-ben megbízta az alá tartozó X3 nevű szervezet adatbázis-technológiai bizottságát, az X3H2-t, hogy készítsen tervezetet egy ilyen szabványhoz. (Az X3 az ANSI által felügyelt számos szervezet egyike, az X3H2 azonban csak egy azok közül a műszaki bizottságok közül, amelyek az X3 hatáskörébe tartoznak. Az X3H2-t adatbázis-szakértők

és a fontosabb SQL alapú adatbázisokat gyártó cégek képviselői alkotják.) A bizottság kezdetben áttekintette és megvitatta a különböző javasolt nyelvek előnyeit és hátrányait, és egy az INGRES adatbázisnyelvén, a QUEL-en alapuló szabvány kidolgozásába is belefogott. A piac erői és az IBM növekvő elkötelezettsége az SQL mellett azonban meggyőzték a bizottságot, hogy a szabványtervezet alapjául inkább az SQL-t válassza.

Az X3H2 bizottság szabványtervezete nagyrészt az IBM DB2-jének SQL-nyelvjárását vette alapul. A bizottság a következő két év során több változatot is kidolgozott, és bizonyos fokig tovább is fejlesztette az SQL-t. Ennek a továbbfejlesztésnek azonban az lett a szerencsétlen eredménye, hogy az új szabvány összeegyeztethetetlenné vált a már létező fontosabb SQL-nyelvjárásokkal. Az X3H2 hamarosan ráébredt, hogy az SQL-en végrehajtott módosítások nem eredményeztek elég jelentős javulást ahhoz, hogy ellensúlyozzák a csereszabatosság sérülését, ezért a bizottság visszatért a szabvány eredeti változatához.

Az ANSI 1986-ban, ANSI X3.135-1986 Database Language SQL néven szentesítette az X3H2 szabványtervezetét, amely végül SQL/86 néven vált széles körben ismertté. Bár az X3H2 kisebb javításokat is eszközölt a szabványon, mielőtt azt az ANSI elfogadta volna, az SQL/86 csupán a legkisebb közös részhalmazát határozta meg azoknak a követelményeknek, amelyeknek az adatbázisgyártóknak eleget kellett tenniük. Lényegében tehát csak hivatalos szintre emelte azokat az elemeket, amelyek a különböző SQL-nyelvjárásokban megegyeztek, és amelyeket a legtöbb adatbázisgyártó már megvalósított. Mindazonáltal az új szabvány végre konkrét alapot biztosított, ahonnan kiindulva a nyelvet és annak megvalósításait tovább lehetett fejleszteni.

A Nemzetközi Szabványügyi Szervezet (ISO, International Organization for Standardization) a saját szabványtervezetét (ami pontosan megfelelt az ANSI SQL/86-nak) 1987-ben hagyta jóvá nemzetközi szabványként, és ISO 9075-1987 Database Language SQL néven tette közzé. (Általában mindkét szabványra ma is csak SQL/86-ként szoktak hivatkozni.) Az adatbázisgyártók nemzetközi közössége innen kezdve ugyanazokra a szabványokra építhetett, mint az Egyesült Államokban tevékenykedő társaik. Mindazonáltal annak ellenére, hogy az SQL hivatalos szabvánnyá vált, a nyelv még messze nem állt teljesen készen.

Az ANSI/ISO szabvány fejlődése

Az SQL/86-ot hamarosan nyilvános bírálatok érték, egyrészt a kormányzat, másrészt az iparág olyan szaktekintélyei részéről, mint C. J. Date. A kritikusok olyan dolgokat róttak fel a szabványnak, mint az SQL-nyelvtan felesleges ismétlődései (ugyanazt a lekérdezést többféleképpen is meg lehetett határozni), valamint bizonyos relációs műveletek támogatásának, valamint a hivatkozási épségnek a hiánya. Bár az X3H2 már az előtt tisztában volt ezekkel a problémákkal, mielőtt közzétették volna az SQL/86-ot, a bizottság úgy döntött, hogy célszerűbb hamarabb megjelentetni a szabványt (még ha később finomítani kell is), mint hogy egyáltalán ne legyen szabvány.

Mind az ISO, mind az ANSI úgy felelt a hivatkozási épséget érintő kritikákra, hogy finomított a szabványán. Az ISO 1989 közepén tette közzé az ISO 9075:1989 Database Language SQL with Integrity Enhancements nevű változatot, az ANSI pedig még ugyanabban az évben megjelentette az X3.135-1989 Database Language SQL with Integrity Enhancements szabványt, amelyre gyakran SQL/89-ként szoktak hivatkozni. Az ANSI-bizottság munkája azonban nem ért véget az évre: az X3H2 továbbra is megoldást igyekezett találni egy fontos kérdésre, amelyet a kormányzat vetett fel.

Egyes állami felhasználók panaszkodtak, hogy a szabvány nem írja le kifejezetten, hogy miként lehet az SQL-t beágyazni egy hagyományos programozási nyelvbe. (A szabvány tartalmazott ugyan erre vonatkozó leírást, de csupán függelékként.) Amiatt aggódtak, hogy a gyártók esetleg nem hordozható megvalósításokkal állnak majd elő a beágyazott SQL-hez, hiszen a szabvány ehhez nem ír elő követelményeket. Az X3H2 válaszul kidolgozott egy második szabványt, amely megkövetelte a beágyazás leírásának való megfelelőséget; ez volt az ANSI X3.168-1989 Database Language Embedded SQL. Érdekes, hogy az ISO nem tett közzé hasonló szabványt, mert a nemzetközi közösségnek nem voltak hasonló aggodalmai. Így aztán az ISO-nak nem volt szabványleírása az SQL beágyazására más programozási nyelvekbe, és ez a helyzet csak akkor változott meg, amikor az ISO megjelentette az SQL/92 szabványt.

Az SQL/86 és az SQL/89 még távol állt attól, hogy teljes szabvány legyen, mert hiányzott belőlük néhány a kereskedelmi adatbázisrendszerekben elengedhetetlen szolgáltatás. A két szabvány egyike sem írta le például, hogy miként lehet módosítani az adatbázis szerkezetét (akár magán az adatbázisrendszeren belül), miután meghatároztuk azt. Egyetlen szerkezeti elem (például a táblák és az oszlopok) sem volt módosítható vagy törölhető, és az adatbázis biztonságán sem lehetett változtatni. Létrehozhattunk például a CREATE utasítással egy táblát, de a szabvány nem határozta meg sem a táblák törlésére szolgáló DROP parancsot, sem a módosító ALTER utasítást. Hozzáférést adhattunk egy táblához a GRANT paranccsal, de a szabványban nem szerepelt a REVOKE parancs, amellyel visszavonhattuk volna az engedélyt. Ironikus módon ezekkel a képességekkel minden kereskedelmi SQL alapú adatbázis-kezelő rendelkezett, de egyik szabványba sem kerültek be, mert az egyes gyártók más-más módon valósították meg őket. Számos más olyan szolgáltatás is helyet kapott sok SQL alapú adatbázis-kezelőben, amely kimaradt a szabványokból – ismét csak a megvalósítások különbözősége miatt.

Mire az SQL/89-t befejezték, már mind az ANSI, mind az ISO megkezdte a munkát az SQL-szabvány javított változatán, amely a nyelvet valóban teljessé és hatékonnyá volt hivatott tenni. Az új változatba, amelyre (természetesen) SQL/92-ként hivatkoztak, olyan szolgáltatásokat kívántak beépíteni, amelyeket a fontosabb adatbázisgyártók már széles körben megvalósítottak. Az ANSI és az ISO egyik legfontosabb célkitűzése azonban az volt, hogy elkerüljék, hogy a szabvány ismét „a legkisebb közös osztó" elvét kövesse,

ezért úgy döntöttek, hogy olyan képességekkel is bővítik a nyelvet, amelyek még nem váltak széles körben elfogadottá, valamint olyan új szolgáltatásokat is belevesznek, amelyek jelentősen túlmutatnak a létező megvalósításokon.

Az új ANSI, illetve ISO szabvány – az X3.135-1992 Database Language SQL, illetve az ISO/IEC 9075:1992 Database Language SQL – 1992 októberében jelent meg. (A dokumentumok már 1991 végén készen álltak, de a rákövetkező évben még finomítottak rajtuk.) Az SQL/92 leírás lényegesen hosszabb volt az SQL/89-nél, de bővebb anyagot is tartalmazott. Meghatározta például azokat a módszereket, amelyekkel az adatbázis szerkezetét annak meghatározása után módosíthatjuk, további műveleteket adott a karakterláncok, valamint a dátumok és időpontok kezelésére, és új biztonsági szolgáltatásokat is leírt. Az SQL/92 tehát hatalmas ugrást jelentett bármelyik elődjéhez képest.

Szerencsére a szabványügyi bizottságok bizonyos fokig felkészültek az új helyzetre. Ahhoz, hogy az új szabványra történő fokozatos és gördülékeny átállást biztosítsák, az ANSI és az ISO az SQL/92-nek három szintjét határozta meg:

ENTRY SQL (Belépő szintű SQL) Ez a szint hasonló az SQL/89-hez, de a korábbi szabványt olyan szolgáltatásokkal bővíti, amelyek segítenek az SQL/92-re történő átállásban, illetve kijavítják az SQL/89 hibáit. Az elv az volt, hogy ezt a szintet könnyen meg lehessen valósítani, mivel a szolgáltatásainak többsége már rendelkezésre áll a legtöbb meglévő termékben.

INTERMEDIATE SQL (Középszintű SQL) Ez a szint az új szabvány legtöbb szolgáltatását tartalmazza. A két bizottság döntését, miszerint e szint részévé tesznek bizonyos szolgáltatásokat, több tényező befolyásolta. Az általános cél az volt, hogy úgy javítsanak a szabványon, hogy az SQL jobban támogassa a relációs modell elveit, illetve hogy a nyelvtanból kiküszöböljék a homályos vagy kétértelmű elemeket. Az egyértelmű volt, hogy a gyártók által így vagy úgy már megvalósított és az említett célokkal összhangban levő szolgáltatások helyet kapnak ezen a szinten. Az SQL-adatbázisrendszerek felhasználói által igényelt szolgáltatásokat aszerint bírálták el, hogy eleget tesznek-e ezeknek a követelményeknek, és hogy a legtöbb gyártó viszonylag könnyen képes lesz-e megvalósítani őket. Ennek a szintnek tehát az volt a célja, hogy biztosítsa, hogy egy adott termék a lehető leghatékonyabb megvalósítást legyen képes nyújtani.

FULL SQL (Teljes SQL) Ez a szint a teljes SQL/92 szabványt magába foglalja, és természetesen itt kaptak helyet azok az összetettebb szolgáltatások, amelyek az első két szintből kimaradtak. Ez a szint tehát olyan szolgáltatásokat tartalmaz, amelyekről – bár a felhasználói igények kielégítésére, illetve a nyelv további „megtisztítására" elég fontosnak találtattak – úgy gondolták, hogy a legtöbb gyártó csak nehezen tudná azonnal megvalósítani. Sajnos a Full SQL-nek való megfelelőség még nem követelmény, ezért egyhamar nem várhatjuk, hogy minden adatbázistermék teljesen megvalósítja a szabványt.

Bár számos adatbázisgyártó igyekezett megvalósítani az SQL/92-ben leírtakat, a saját szolgáltatások fejlesztését és megvalósítását is folytatták. Ezeket az SQL-szabványt kiegészítő szolgáltatásokat nevezzük *bővítéseknek*. Egyes gyártók például több adattípust biztosítanak az SQL/92-ben meghatározott hat típusnál. Bár a bővítések további képességekkel ruháznak fel egy adott terméket, és lehetővé teszik a gyártóknak, hogy megkülönböztessék a programjukat más termékektől, hátrányaik is vannak. A bővítésekkel kapcsolatban az jelenti a fő problémát, hogy hatásukra az adott gyártó SQL-nyelvjárása távolabb kerül az eredeti szabványtól, ami megakadályozza, hogy az adatbázis-fejlesztők olyan hordozható alkalmazásokat készítsenek, amelyek bármely SQL alapú adatbázis-kezelőből futtathatók.

Egyéb SQL-szabványok

Az ANSI/ISO SQL-szabvány ma a legszélesebb körben elfogadott szabvány, ami természetesen azt is jelenti, hogy a többi létező szabvány ugyancsak tartalmazza az SQL-t valamilyen formában. Lássunk néhányat a jelentősebb alternatív szabványok közül:

X/OPEN Az X/OPEN európai gyártók csoportja, amely egy olyan szabványgyűjteményt dolgozott ki, amelynek az a célja, hogy elősegítse egy UNIX alapú hordozható alkalmazáskörnyezet kialakítását. Az európai piacon komoly jelentőséggel bír, hogy egy alkalmazást módosítás nélkül át lehessen vinni egy számítógéprendszerről egy másikra. Bár az X/OPEN tagjai a szabványgyűjtemény részévé tették az SQL-t, az ő változatuk több ponton is eltér az ANSI/ISO szabványoktól.

SAA Az IBM mindig is saját SQL-nyelvjárást használt, amelyet beleépítettek a Systems Application Architecture (SAA) nevű szabványleírásukba. Az SAA egyik fő célja az volt, hogy az IBM SQL-nyelvjárását a cég teljes adatbázistermék-kínálatának részévé tegyék, és bár ezt a célt soha nem érték el, az SQL most is fontos szerepet játszik az IBM adatbázistermékeinek egységessé tételében.

FIPS Az NIST (National Institute of Standards and Technology, Nemzeti Szabványügyi és Technológiai Intézet) 1987-ben tette az SQL FIPS-szabvánnyá (a FIPS a Federal Information Processing Standard, vagyis szövetségi információfeldolgozási szabvány rövidítése). Az eredetileg FIPS PUB 127 néven közzétett szabvány azt a szintet írja le, amelyet egy RDBMS-nek teljesítenie kell ahhoz, hogy megfeleljen az ANSI/ISO szabványnak. Az Egyesült Államok kormányzati szervei azóta csak olyan relációs adatbázistermékeket használhatnak, amelyek megfelelnek az érvényben levő FIPS-szabványnak.

ODBC 1989-ben adatbázisgyártók egy csoportja megalakította az SQL Access csoportot, azzal a céllal, hogy megoldják az adatbázisok együttműködésének problémáját. Bár ezen gyártók első erőfeszítései nem igazán jártak sikerrel, az eredeti célkitűzés mellett arra is kísérletet tettek, hogy kidolgozzanak egy módszert arra, hogy egy SQL-adatbázist egy felhasználói felületi nyelvhez lehessen kapcsolni. Ennek az eredménye lett a CLI (Call-Level Interface) szabvány, amelyet 1992-ben tettek közzé. Ugyanebben az évben a Microsoft megjelentette a saját ODBC (Open Database Connectivity) leírását is, amely a CLI-n alapult. Az ODBC azóta az adatok elérésének és megosztásának *de facto* szabványává vált az ODBC-t támogató SQL adatbázisok között.

A fenti szabványokat folyamatosan továbbfejlesztik, hogy igazodjanak az ANSI/ISO SQL újabb változataihoz, de néha ettől függetlenül is javítanak rajtuk.

1997-ben az ANSI X3 szervezete új nevet kapott: National Committee for Information Technology Standards (NCITS, Nemzeti Információtechnológiai Szabványok Bizottsága). Az SQL-szabványért felelős műszaki bizottság neve is megváltozott; ezt ma ANSI NCITS-H2-nek hívják. Az SQL-szabvány rohamosan növekvő bonyolultsága miatt az ANSI és az ISO szabványügyi bizottságai az SQL3 szabvány kidolgozásának megkezdésekor (a név onnan ered, hogy ez lesz a szabvány harmadik nagyobb változata) megegyeztek abban, hogy a szabványt tizenkét önállóan számozott részre, valamint egy függelékre bontják, hogy az egyes részeken párhuzamosan folyhasson a munka. Ezenkívül 1997 óta két további részt is meghatároztak. Az SQL-szabvány egyes részeinek nevét és leírását a 3.1. táblázatban soroltuk fel, azzal együtt, hogy az adott rész milyen állapotban volt 2007 elején.

3.1. táblázat *Az SQL-szabvány felépítése*

Cím	Állapot	Leírás	Oldalak száma az SQL:2003-ban
Part 1: SQL/Framework	1999-ben befejezték, 2003-ban átdolgozták.	A szabvány egyes részeit írja le, és olyan információkat tartalmaz, amelyek minden részre vonatkoznak.	81
Part 2: SQL/Foundation	Ez az 1992-es alapszabvány, amelyet 1999-ben és 2003-ban átdolgoztak.	Az SQL nyelv adatmeghatározó és adatkezelő részeinek nyelvtanát és jelentéstanát írja le.	1267
SQL/OLAP (Online Analytical Processing)	1999-ben összevonták a Foundation résszel.	Az elemző feldolgozás függvényeit és műveleteit írja le. (Az SQL/Foundation kiegészítésének szánják.)	
Part 3: SQL/CLI (Call-Level Interface)	1995-ben befejezték, 1999-ben és 2003-ban kibővítették.	Ez a rész, amelyet az SQL Access csoport dolgozott ki, megfelel a Microsoft ODBC szabványának.	405
Part 4: SQL/PSM (Persistent Stored Modules)	1996-ban befejezték. A tárolt eljárásokat és a CALL utasítást 1999-ben áttették a Foundation részbe, a szabvány többi részét 2003-ban átdolgozták.	Azokat az eljárásközpontú SQL-utasításokat határozza meg, amelyek felhasználói függvényekben és eljárásokban használhatók. (A tárolt eljárásokat, a tárolt függvényeket, a CALL utasítást és az eljáráshívást végül áthelyezték az SQL/Foundation részbe.)	184

3.1. táblázat *Az SQL-szabvány felépítése*

Cím	Állapot	Leírás	Oldalak száma az SQL:2003-ban
Part 5: SQL/Bindings	Az SQL beágyazásának leírását 1999-ben önálló részbe helyezték, majd 2003-ban a Foundation részbe foglalták.	Azt írja le, hogy miként ágyazhatjuk be az SQL-t nem objektumközpontú programozási nyelvekbe. Az SQL következő változatában már az SQL/Foundation része lesz.	
Part 6: Transaction (XA Specialization)	1999-ben törölték.	Az SQL szakosítása az X/OPEN XA szabványának megfelelően.	
Part 7: SQL/Temporal	2003-ban visszavonták.	Az ideiglenes adatok tárolását és kinyerését határozza meg. A követelményeket és a részleteket illetően nézetkülönbség alakult ki, ezért a munka az utóbbi években elakadt.	
Part 8: SQL/Objects-Extended Objects	1999-ben összevonták a Foundation résszel.	Azt határozza meg, hogy az RDBMS-nek hogyan kell kezelnie az alkalmazás által meghatározott elvont adattípusokat.	
Part 9: SQL/MED (Management of External Data)	Az ISO-változat 2003-ban elkészült.	Az SQL/Foundation részt olyan nyelvi elemekkel és meghatározásokkal egészíti ki, amelyek lehetővé teszik az SQL-nek, hogy nem SQL adatforrásokat (fájlokat) is elérjen.	498
Part 10: SQL/OLB (Object Language Bindings)	1998-ban befejezték (csak ANSI-szabvány), 1999-ben az ISO átdolgozta, majd 2003-ban ismét módosította.	Az SQL-nek a Java programozási nyelvbe történő beágyazását írja le nyelvtani és jelentéstani szempontból. Megfelel egy másik ANSI szabványnak, az SQLJ Part 0-nak.	405
Part 11: SQL/Schemata	A Foundation részből vették ki; 2003-ban elkészült.	Információs és meghatározási sémákat ír le.	296

3.1. táblázat *Az SQL-szabvány felépítése*

Cím	Állapot	Leírás	Oldalak száma az SQL:2003-ban
Part 12: SQL/Replication	A projekt 2000-ben indult, de a haladás hiánya miatt 2003-ban ejtették.	Az SQL adatbázisok többszörözésének támogatását és lehetőségeit írja le.	
Part 13: JRT–SQL Routines Using the Java Programming Language	1999-ben befejezték (csak ANSI-szabvány, az SQL/92 alapján). 2003-ban nemzetközi szabvánnyá dolgozták át.	Azt határozza meg, hogy miként használhatunk Java kódot egy SQL adatbázison belül.	204
Part 14: SQL/XML-SQL Routines Using the eXtensible Markup Language	2003-ban befejezték, 2006-ban kibővítették.	Azt határozza meg, hogy miként használhatunk XML-t egy SQL adatbázison belül. A W3C XQuery V1.1 szabványához igazodik.	266

Kereskedelmi megvalósítások

Ahogy a fejezetben korábban olvashattuk, az SQL-t először nagygépes környezetben használták. Az olyan termékek, mint a DB2, az INGRES vagy az Oracle 1979 óta léteznek, és elfogadottá tették az SQL-t mint a relációs adatbázisok kezelésének elsődleges nyelvét. Az 1980-as években a relációs adatbázisok megjelentek az asztali személyi számítógépeken is, és az olyan programok, mint az R:BASE, a dBase IV és a Super Base egyszerűen elérhetővé tették a táblákban tárolt adatokat a hétköznapi felhasználók számára is. Az SQL azonban csak az 1980-as évek végén, illetve az 1990-es évek elején vált az asztali relációs adatbázisok meghatározó nyelvévé. A termék, amelynek az áttörés köszönhető, kétségkívül a Microsoft Access 1992-ben megjelent 1-es változata volt.

Az 1990-es évek elején elérkezett az ügyfél–kiszolgáló rendszerek kora is, az olyan RDBMS programokat pedig, mint a Microsoft SQL Server vagy az Informix-SE, úgy tervezték, hogy a legkülönfélébb többfelhasználós környezet felhasználóinak nyújtsanak adatbázis-szolgáltatásokat. 2000 óta arra is összehangolt erőfeszítések történnek, hogy az adatbázisok információit az Interneten keresztül is elérhetővé tegyék. Az e-kereskedelem vonzóvá vált a cégek számára, és azok is, akik még nem jelentek meg a Világhálón, igyekeznek gyorsan pótolni ezt a hiányosságot. Ennek eredményeképpen az adatbázis-fejlesztőknek hatékonyabb ügyfél–kiszolgáló adatbázisokra van szükségük, és a bevált RDBMS-termékek olyan új változataira, amelyek segítségével a webhelyek igényeit kielégítő adatbázisokat fejleszthetnek és tarthatnak fenn.

Megpróbálhatnánk felsorolni az SQL-t támogató összes kereskedelmi forgalomban levő terméket, de a lista sok-sok oldalnyi lenne, ezért elég, ha megállapítjuk, hogy az SQL még sokáig a kereskedelmi adatbázisrendszerek része lesz.

Mit tartogat a jövő?

Amikor könyvünk első kiadását írtuk 1999-ben, a szabványügyi bizottságok éppen az utolsó simításokat végezték a régóta ígért SQL3-on. Azóta megjelent az SQL:1999 és az SQL:2003 is, 2007 közepén pedig mind az ANSI-, mind az ISO-bizottság keményen dolgozott az SQL:2007 javított kiadásán. A tervek szerint alapos felülvizsgálat vár az SQL/XML részre (Part 14). A nemzetközi bizottság egy önálló SQL/MM–Multimedia szabványon is munkálkodik, amely maga is öt részből – Framework (keretrendszer), Full Text (teljes szöveg), Spatial (térgrafika), Still Image (állókép) és Data Mining (adatbányászat) – fog állni. Bár a szabványügyi bizottságok a kereskedelmi megvalósítások mögött alaposan lemaradva kezdték meg a munkát 1986-ban, az SQL-szabvány azóta egyértelműen behozta a hátrányt, és sok területen már a hozzáférhető adatbázisrendszerek szolgáltatásai előtt jár.

Miért érdemes megtanulnunk az SQL használatát?

Ha megtanuljuk az SQL használatát, birtokába jutunk mindazoknak a képességeknek, amelyek révén bármilyen relációs adatbázisból információkat nyerhetünk ki, ezen felül segít, hogy megértsük, mi zajlik az RDBMS-termékek grafikus lekérdezőfelülete mögött. Az SQL ismerete révén összetett lekérdezéseket írhatunk, és rendelkezni fogunk azzal a tudással is, ami a lekérdezések esetleges hibáinak elhárításához szükséges.

Mivel az SQL a legkülönfélébb RDBMS-termékek része, tudásunkat sokféle rendszeren hasznosíthatjuk. Ha megtanuljuk például, hogy miként kell használni az SQL-t egy olyan programban, mint a Microsoft Access, ennek a képességünknek akkor is hasznát látjuk, ha a cégünk úgy dönt, hogy Microsoft SQL Serverre, az Oracle Corporation Oracle rendszerére vagy az IBM DB/2-jére tér át. Nem kell újra elsajátítanunk az SQL használatát – csak az első termék nyelvjárása és a másik program SQL-változata közötti különbségekkel kell tisztában lennünk.

Nem lehet elégszer hangsúlyozni, hogy az SQL még sokáig velünk marad. Számos gyártó fektetett rengeteg pénzt, időt és kutatói erőforrást abba, hogy az SQL-t beépítse a saját RDBMS-termékeibe, és hatalmas a száma azoknak a cégeknek és intézményeknek is, amelyek információtechnológiai infrastrukúrájukat ezekre a termékekre építették. Amit ebben a fejezetben olvashattunk az SQL-ről, annak alapján feltételezhetjük, hogy a fejlődése nem áll meg: folyamatosan megújul majd, hogy igazodjon a változó igényekhez és piaci követelményekhez.

Összefoglalás

A fejezetet az SQL eredetének rövid áttekintésével kezdtük. Megtudtuk, hogy az SQL a relációs adatbázisokhoz készült nyelv, amelyet nem sokkal a relációs modell megjelenése után alkottak meg, és hogy a fejlődése eleinte szorosan kapcsolódott magának a relációs modellnek a fejlődéséhez.

Ez után arról olvashattunk, hogy kezdetben miként valósították meg a különböző adatbázisgyártók a relációs modellt. Megtudtuk, hogy az első relációs adatbázisokat nagyszámítógépes környezetekhez készítették, és láttuk, hogyan vált az IBM és az Oracle az iparág vezető erejévé.

Ezt követően az ANSI SQL-szabvány megszületését ismertettük: megtudtuk, hogy már létezett egy nem hivatalos szabvány, amikor az ANSI úgy döntött, hogy készít egy hivatalos változatot, és olvashattunk az ANSI X3H2 bizottságának az első szabványváltozaton végzett munkájáról. Elmondtuk, hogy bár az új szabvány a szolgáltatásoknak csupán a legkisebb közös részhalmazát határozta meg, végre konkrét alapot biztosított, ahonnan kiindulva a nyelvet tovább lehetett fejleszteni, és arról is szót ejtettünk, hogy az ISO is megjelentette a saját szabványát, amely pontosan megfelelt az ANSI leírásának.

Következő témánk az ANSI/ISO szabvány fejlődése volt. Elmondtuk, hogy a szabvány első változatát sokan bírálták, és hogy az ANSI/ISO a bírálatokra a szabvány többszöri felülvizsgálatával válaszolt. Láttuk, hogyan vezetett az egyik változat a másikhoz, és hogy miként érkeztünk el végül az SQL/92 szabványhoz. Ismertettük, hogy a szabvány több megfelelőségi szintet határoz meg, ami lehetővé teszi a gyártóknak, hogy a lehető leggördülékenyebben építsék be a szabvány új szolgáltatásait a termékeikbe. Ez után összefoglaltuk, hogyan fejlődött tovább az SQL-szabvány 1992 óta, és a kereskedelmi SQL-adatbázisok fejlődésére is vetettünk egy gyors pillantást.

A fejezet végén az SQL jövőjéről elmélkedtünk. Megtudhattuk, hogy az SQL:2003 sokkal bonyolultabb szabvány az SQL/92-nél, elmagyaráztuk, miért fog folytatódni az SQL fejlesztése, és azt is megindokoltuk, hogy miért érdemes megtanulnunk a nyelv használatát.

II

Az SQL alapjai

4

Egyszerű lekérdezések írása

„Gondolkodjunk úgy, mint a bölcsek,
de beszéljünk úgy, mint az emberek."
– William Butler Yeats

A fejezet témakörei

- Bemutatkozik a SELECT
- A SELECT utasítás
- Egy kis kitérő: adat kontra információ
- A kérelmek lefordítása SQL-re
- A sorismétlés kiküszöbölése
- Az információk rendezése
- Mentés
- Példák
- Összefoglalás
- Önálló feladatok

Most, hogy egy kicsit megismerkedtünk az SQL történetével, lépjünk tovább, és ismerkedjünk meg magával a nyelvvel! Amint a bevezetőben is említettük, a könyvben a legnagyobb hangsúlyt a nyelv adatkezelő képességeinek bemutatására fektetjük majd. Vegyük hát szemügyre elsőként az SQL igazi igáslovának tekinthető SELECT utasítást!

Bemutatkozik a SELECT

A SELECT utasítást tekinthetjük – minden más kulcsszót megelőzve –, az SQL nyelv lelkének. Sarokkövét képezi a leghatékonyabb és legösszetettebb utasításoknak, segítségével változtathatjuk az adatbázisunk tábláiban tárolt adatokat beszédes információkká. Ráadásul a SELECT-et más kulcsszavakkal és kifejezésekkel számtalan módon össze is kapcsolhatjuk. Ki? Mi? Hol? Mikor? Mennyi? Milyen esetben? Szinte minden kérdéstípusra válaszolhatunk vele. Ha jól terveztük meg az adatbázisunkat, és rendelkezésünkre állnak a megfelelő

adatok, minden feleletet megkaphatunk, amire csak szükségünk lehet például a cégünk számára fontos döntések meghozatalához. Amikor pedig az 5. fejezethez érünk, észre fogjuk venni, hogy sok mindent, amit a SELECT-ről tanultunk, az UPDATE, INSERT, és DELETE utasítások írása során is alkalmazhatunk majd.

Az SQL nyelvben a SELECT művelet három kisebb művelettípusra bontható, amelyekre a későbbiekben SELECT utasítás, SELECT kifejezés és SELECT lekérdezés néven fogunk hivatkozni. Így egyszerűbb lesz átlátnunk és megértenünk a benne rejlő összetett lehetőségeket. Mindegyik művelettípusnak megvannak a jellemző kulcsszavai és záradékai, ami nagyfokú rugalmasságot biztosít a végső lekérdezés összeállításához, hogy az minél tökéletesebben kifejezhesse az adatbázisnak feltenni kívánt kérdést. Amint a következő fejezetekben megtanuljuk majd, az egyes műveletek számtalan módon kombinálhatók is egymással, így igen összetett kérdéseket is feltehetünk.

Ezt a fejezetet a SELECT utasítás tárgyalásával kezdjük, és röviden áttekintjük a SELECT lekérdezéseket. Ezt követően részleteiben megvizsgáljuk a SELECT utasításokat az 5. és 6. fejezetben.

Megjegyzés *A téma szakirodalmában előfordul, hogy a tábla helyett a reláció kifejezést olvashatjuk, az adatbázis soraira a rekord – egyes angol nyelvű könyvekben ritkábban a tuple –, az oszlopaira pedig a jellemző (attribútum) vagy mező elnevezést alkalmazzák, jóllehet az SQL-szabvány következetesen tábla, sor, és oszlop néven hivatkozik az adatbázisszerkezet ezen elemeire. Mi – összhangban a szabvánnyal – az utóbbi három fogalmat fogjuk használni a továbbiakban is.*

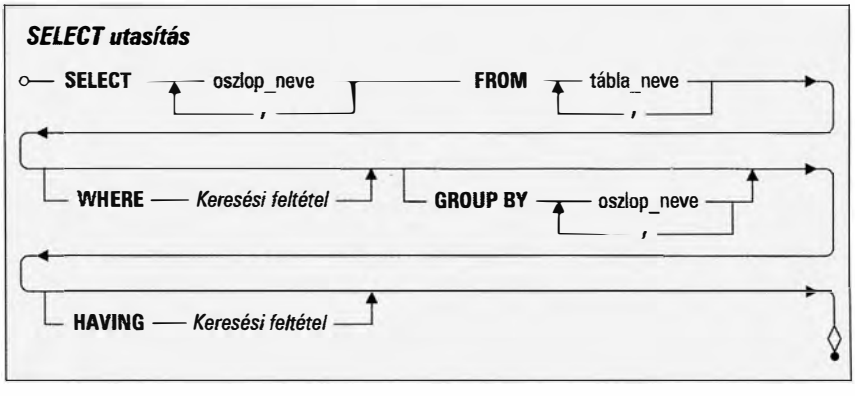
A SELECT utasítás

A SELECT utasítás képezi minden kérdés alapját, amit felteszünk az adatbázisnak. Amikor megfogalmazunk és végrehajtunk egy SELECT utasítást, lekérdezést intézünk az adatbázis felé. (Tudjuk, hogy magától értetődőnek hangzik, de szeretnénk biztosak lenni benne, hogy a téma bizonyos sarokpontjait minden olvasónk azonosan értelmezi.) A gyakorlatban sok relációs adatbázis-kezelő (RDBMS) rendszer lehetőséget ad rá, hogy a SELECT utasításainkat *lekérdezés*, *nézettábla*, *függvény* vagy *tárolt eljárás* formájában mentsük. Ha valakitől azt halljuk, hogy végrehajt egy lekérdezést, tudjuk, hogy valamilyen SELECT utasítást készül futtatni az adatbázison. Adatbázis-kezelőtől függően a SELECT utasításokat végrehajthatjuk közvetlenül egy parancsablakból, interaktív Query by Example (QBE) szerkesztőrácsból, de programkód belsejéből is. Függetlenül attól, hogy miként határozzuk meg és futtatjuk, a SELECT utasítás nyelvtani szabályai ugyanazok maradnak.

Megjegyzés *Sok adatbázis-kezelő biztosít bővített lehetőségeket az eredeti SQL-szabványhoz képest, és teszi lehetővé összetettebb utasítások (például If...Then...Else) használatát függvényekben vagy tárolt eljárásokban, de a pontos utasí-*

tásformák minden termék esetében egyediek. Akár csak egy-két ilyen bővített programozási nyelv – például a Microsoft SQL Server Transact-SQL-je vagy az Oracle PL/SQL-je – bemutatása is messze túlmutatna ennek a könyvnek a keretein, ezért az adatbázisrendszerünk függvényeinek és tárolt eljárásainak létrehozásához továbbra is az alapvető SELECT utasítást fogjuk használni. Könyvünkben mindvégig a nézettábla elnevezést fogjuk alkalmazni, amikor mentett SQL-utasításokról ejtünk szót, jóllehet ezeket valószínűleg egy függvénybe vagy eljárásba ágyazzuk majd.

A SELECT utasítások egymástól független kulcsszavakból épülnek fel, amelyeket *záradékoknak* (clause) hívunk. A SELECT utasítások ezeknek a záradékoknak a különféle beállításaitól függően képesek a kért információk visszaadására. A záradékok egy része kötelezően megadandó, másokat elhagyhatunk. Emellett az egyes záradékokhoz is tartoznak saját kulcsszavak, amelyek között szintén vannak kötelezők és elhagyhatók. Ezeket az adott záradék használja annak érdekében, hogy azokat az információkat kérje le, amelyeket a teljes SELECT utasítás igényel. A SELECT utasítást és a záradékait a 4.1. ábrán láthatjuk.



4.1. ábra
A SELECT utasítás diagramja

Megjegyzés *A 4.1. ábrán látható szintaxisdiagram egy egyszerű SELECT utasítást mutat be. A későbbiekben ezt a diagramot fogjuk bővíteni, ahogy az új kulcsszavakkal és záradékokkal megismerkedünk. Ha van már tapasztalatunk SQL-utasításokkal, legyünk türelmesek, ezeknek is hamarosan eljön az idejük.*

Az alábbiakban összefoglaljuk a SELECT utasítás záradékait.

- SELECT – Ez a SELECT utasítás elsődleges záradéka, amelyet minden esetben kötelező megadni. Arra használjuk, hogy meghatározzuk azokat az oszlopokat, amelyeket a lekérdezés eredményében vissza szeretnénk kapni. Az oszlopokat abból a táblából vagy nézettáblából választjuk ki, amelyeket a FROM záradékban meghatároztunk. (Származhatnak egyszerre több táblából is, de erre bővebben a 3. fejezetben térünk

ki.) Használhatunk összesítő függvényeket is, mint a Sum(HoursWorked) (Sum(Munkaórák)), vagy matematikai műveleteket, mint a Quantity * Price (Mennyiség * Ár).

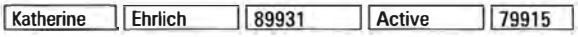
- **FROM** – Ez a második legfontosabb záradék, és ezt is kötelező megadnunk. Arra használjuk, hogy megnevezzük a táblákat vagy nézettáblákat, amelyeknek az oszlopaiból a SELECT záradékban meghatározott elemek származnak. Ezt a záradékot sokkal összetettebb módon is használhatjuk; részletesebben ezzel is a későbbi fejezetekben fogunk foglalkozni.
- **WHERE** – Ez egy elhagyható záradék, és arra használhatjuk, hogy szűkítsük a FROM záradék által visszaadott sorok körét. A WHERE kulcsszót egy kifejezés követi, amelyet *állításnak* (predikátumnak) nevezünk, és amelyet kiértékelve igaz, hamis vagy ismeretlen értéket kaphatunk. A kifejezés kiértékelésére szabványos összehasonlító műveleteket, logikai műveleteket, és még néhány különleges műveletet használhatunk. A WHERE záradék összes elemét a 6. fejezetben tárgyaljuk majd.
- **GROUP BY** – Amikor úgynevezett összesítő függvényeket használunk a SELECT záradékban összegző jellegű információk előállításához, a GROUP BY záradékot használjuk az adatok csoportokra bontásához. Adatbázis-kezelőnk a GROUP BY után megadott oszlop vagy oszlopok szerint csoportosítja az adatokat. A GROUP BY záradék használata nem kötelező. Részletesebben a 13. fejezetben foglalkozunk vele.
- **HAVING** – A HAVING záradékot az összesítő függvények által szolgáltatatott csoportosított adatok szűrésére használhatjuk. A WHERE záradékkal megegyezően a HAVING kulcsszót egy kifejezés követi, amely igaz, hamis vagy ismeretlen értéket adhat. A kifejezés kiértékelésére szabványos összehasonlító műveleteket, logikai műveleteket, és még néhány különleges műveletet használhatunk. A HAVING megadása sem kötelező; részletesebben a 14. fejezetben ismerhetjük meg.

Először a legegyszerűbb SELECT utasításokkal látunk munkához, és a SELECT és FROM záradékokra összpontosítunk, majd egyesével bővítjük a fegyvertárunkat más záradékokkal, ahogy haladunk fejezetről fejezetre, hogy egyre összetettebb SELECT utasításokat építsünk.

Egy kis kitérő: adat kontra információ

Mielőtt első lekérdezésünket nekiszegeznénk az adatbázisunknak, tisztáznunk kell, hogy lényeges különbség van *adat* és *információ* között. Egyszerűen megfogalmazva, adat az, amit az adatbázisban tárolunk, információ pedig az, amit az adatbázisból kinyerünk. Ez a szétválasztás azért fontos a számunkra, mert segít a helyes szemléletmód kialakításában. Tartsuk szem előtt, hogy az adatbázis arra szolgál, hogy beszédes információkat szolgáltasson valakinek, aki az adott cégnél dolgozik. Az információ azonban csak akkor állhat majd a rendelkezésünkre, ha a megfelelő adatok megtalálhatók az adatbázisban, és magát az adatbázist is úgy építettük fel, hogy támogassa ezeknek az információknak a megszerzését. Vizsgáljuk meg ezeket a fogalmakat részletesebben!

Az adatbázisunk mezőiben tárolt értékek az adatok. Az adatok állandónak tekinthetők, abban az értelemben, hogy változatlanok maradnak, amíg saját kezűleg vagy valamilyen automatizált eljárással nem módosítjuk őket. A 4.2. ábrán szemügyre vehetünk néhány példaadatot.

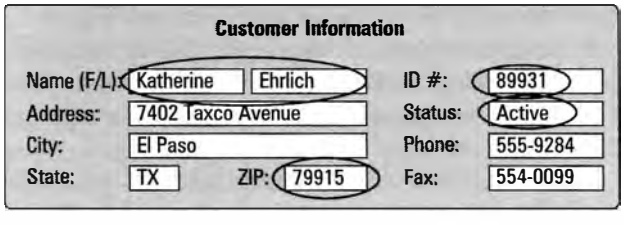


4.2. ábra

Példa egyszerű adatokra

Ezek az adatok nem mondanak nekünk túl sokat, például nem tudjuk eldönteni, hogy a 89931-es érték mit takar. Irányítószám? Vagy talán egy alkatrész azonosítókódja? Még ha tudnánk is, hogy egy vásárló azonosítójáról van szó: vajon Katherine Ehrlich-hez tartozik? Nem tudhatjuk, amíg az adatokat fel nem dolgoztuk. Az adatok feldolgozása után mindez értelmet nyer, és az adatok használhatóvá válnak. Amikor már olyan módon jelenik meg, hogy dolgozunk vele vagy megtekintjük, az adatból információ lesz. Az információkat dinamikusnak tekintjük, mivel az adatbázisban tárolt adatok függvényében folyamatosan változnak, és számtalan módon dolgozhatjuk fel és jeleníthetjük meg őket. A SELECT utasítások eredményeként előálló információkat megjeleníthetjük űrlapok formájában a számítógép képernyőjén, vagy jelentés formájában papírra nyomtatva. Ne felejtsük el, hogy az adatokat fel kell dolgoznunk ahhoz, hogy értelmes, jelentéssel bíró információvá változtassuk azokat.

A 4.3. ábra az előző példában szereplő adatokat mutatja be, miután információvá alakítottuk őket egy ügyfelünk képernyőjén. Jól szemlélteti, hogyan kezelhetők úgy az adatok, hogy értelmesek legyenek bárki számára, aki megtekinti azokat.



4.3. ábra

Adatok információvá alakítása

Amikor SELECT utasításokat használunk, a záradékok segítségével az *adatokon* végzünk műveleteket, de az utasítás maga már *információval* tér vissza. Világos, ugye?

Van még egy kérdés, amelyre választ kell találnunk. Ha végrehajtunk egy SELECT utasítást, az rendszerint egy vagy több oszlopnyi információval tér vissza. (A pontos szám persze attól is függ, hogyan építettük fel az utasítást.) Ezeknek a soroknak az összességét *eredményhal-*

maznak nevezzük – ezt a fogalmat használjuk majd a könyv hátralévő részében. Ez a név jól érzékelteti, hogy amikor relációs adatbázist használunk, mindig adatok halmazával dolgozunk. (Emlékezzünk rá, hogy a relációs modell részben a halmazelméleten alapul.) Az eredményhalmazban könnyedén áttekinthetjük az információkat, sok esetben módosíthatjuk is az adatokat. Minden attól függ, hogy miként szerkesztettük meg a SELECT utasításunkat.

Csapjunk hát a lovak közé, és kezdjünk hozzá a SELECT utasítások használatához!

A kérelmek lefordítása SQL-re

Amikor információkért folyamodunk az adatbázishoz, ezt általában kérdések vagy kérdésként is megfogalmazható utasítások formájában tesszük. Például ilyen utasításokat fogalmazhatunk meg:

„Mely városokban laknak a vásárlóink?"
„Mutasd meg az alkalmazottak aktuális listáját és a telefonszámaikat!"
„Milyen órákat kínálunk jelenleg?"
„Add meg a tanári kar tagjainak a nevét és a dátumokat, amikor felvettük őket!"

Miután már tudjuk, mit akarunk kérdezni, a kérelmeket szabályszerű alakra fordíthatjuk. Ezt az alábbi minta szerint tehetjük meg:

Select <elem> from the <forrás>

Érdemes azzal kezdenünk, hogy a felvetett kérdésben az olyan szavakat és kifejezéseket, mint az *„Írd ki"*, *„Mutasd meg"*, *„Melyek azok..."*, *„Kik azok..."* a „Select" (kiválaszt) szóra cseréljük. Következő lépésként keressük meg a főneveket a kérdésben, és döntsük el, hogy az adott főnév megfeleltethető-e a keresett elemnek vagy egy táblának, amelyben az elem tárolódik. Ha egy elemről van szó, akkor azt helyettesítsük be az <elem> helyére, ha egy tábla nevéről, akkor pedig a <forrás> helyére. Ha az első példakérdésünket lefordítjuk, valami ilyesmit kapunk:

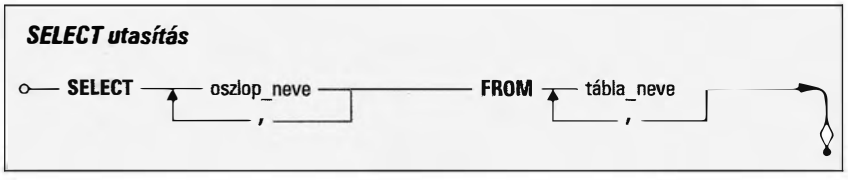
Select city from the customers table
(Válaszd ki a várost a vásárlók táblájából)

Miután a mondatot „szabályosabb" formára fordítottuk, át kell alakítanunk egy teljesen kész SELECT utasítássá, hogy megfeleljen az SQL nyelvtani szabályainak, amint az a 4.4. ábrán látható. Ehhez az első lépés az, hogy letisztázzuk a lefordított mondatot. Húzzuk ki azokat a szavakat, amelyek nem oszlopot vagy táblát megnevező főnevek, vagy nem tartoznak az SQL nyelvi elemei közé. Ezután a mondatunk így fog kinézni:

Select city from ~~the~~ customers ~~table~~

Hagyjuk el a kihúzott szavakat, és máris megkaptuk a kész SQL-utasítást:

```
SELECT City FROM Customers
```



4.4. ábra

Az egyszerű SELECT utasítás formája

Ezt a háromlépéses eljárást alkalmazhatjuk minden kérelem esetében, amelyet egy adatbázishoz szeretnénk intézni. A könyv legnagyobb részében mi is ezt a módszert fogjuk alkalmazni. Mindenkit bátorítunk a használatára, amíg egy kis gyakorlatot nem szerez az utasítások felépítésében. Ha már rutinosan szerkesztünk SQL-utasításokat, ezeket a lépéseket egy műveletben sem okoz majd gondot elvégezni.

Amíg a nyelv tanulásának ennyire az elején vagyunk, jobbára csak oszlopokkal és táblákkal dolgozunk. A 4.4. ábra szintaxisdiagramja is ezt tükrözi (oszlop_neve a SELECT záradékban, illetve a tábla_neve a FROM záradékban). A következő fejezetben megtanuljuk, hogyan építhetünk fel összetettebb SELECT utasításokat további kulcsszavak bevonásával.

Észrevehettük, hogy az előző példában a kérelem nagyon egyszerű volt. Könnyű volt lefordított utasítássá alakítani és behelyettesíteni az oszlopneveket. De mi a helyzet akkor, ha a kérelem nem ilyen egyszerű és könnyen lefordítható, hanem nehézséget okoz azonosítani az oszlopokat, amelyekre majd a SELECT záradékban szükségünk lesz? A legegyszerűbb lehetőségünk, hogy finomítjuk a kérdést, és igyekszünk konkrétabbá tenni azt. Például a *„Mutasd meg a vásárlóinkhoz kapcsolódó információkat!"* kérelmet úgy pontosíthatjuk, ha úgy fogalmazunk, hogy *„Írd ki minden vásárló nevét, városát és telefonszámát!"*. Ha a kérelem finomítása nem oldja meg a problémát, még mindig van két lehetőségünk. Az első, hogy a FROM záradékban megadott táblák között meghatározzuk azokat, amelyek tartalmazzák azokat az oszlopokat, amelyek a nekünk kellő adatokat tárolják, ami megkönnyítheti a kérelem lefordítását. A másik lehetőség, hogy közelebbről megvizsgáljuk a kérelmet, és megfigyeljük, hogy mely szavak vagy kifejezések *utalnak* oszlopnevekre. A módszerek közül az alapján kell választanunk, hogy mi volt az eredeti kérelem. Ha nehézségeink támadnak az utasítások megfogalmazásában, idézzük fel ezeket a módszereket! Most viszont nézzünk meg néhány példát mindkét módszerre, hogy lássuk, hogyan alkalmazhatók jellemzőnek mondható helyzetekben!

Az első módszer bemutatására próbáljuk meg lefordítani az alábbi kérelmet:

„Add meg minden alkalmazott nevét és címét!"

Ez első ránézésre világos kérésnek tűnik. De ha jobban megfigyeljük, észrevesszük, hogy van egy kis probléma: bár tudjuk, hogy melyik táblára lesz szükségünk (Employees, vagyis Alkalmazottak), a kérelem nem ad felvilágosítást arról, hogy mely oszlopokat kell megadnunk a SELECT záradékban. Igaz ugyan, hogy a *„név"* és a *„cím"* szerepel a kérelemben, ezek azonban nagyon általános meghatározások. A problémát úgy oldhatjuk meg, ha azonosítjuk, hogy mely táblákra lesz szükségünk a lekérdezésben, mégpedig úgy, hogy megvizsgáljuk a kérelmet, hogy milyen oszlopokra utalhat. Ezek után használhatjuk az oszlopok neveit a lefordított utasításban. (Ha ez segít jobban elképzelni a lekérdezésünket, dönthetünk úgy, hogy a fordításban még általános oszlopneveket használunk, a végső SQL-utasításban azonban már a valós oszlopneveket kell alkalmaznunk.) Esetünkben az Employees tábla oszlopait megvizsgálva a 4.5. ábrán látható felépítéssel találkozunk, amelynek alapján meghatározhatjuk, hogy mit kell behelyettesítünk a „név" és „cím" szavak helyére.

4.5. ábra
Az Employees tábla felépítése

Hogy teljesen lefedjük a „names" (nevek) és „addresses" (címek) fogalmakat, a táblából összesen hat oszlopra lesz szükségünk. Az EmpFirstName (Alkalmazott keresztneve) és az EmpLastName (Alkalmazott vezetékneve) megfelel a „names"-nek, az EmpStreetAddress (cím), az EmpCity (város), az EmpState (állam) és az EmpZipCode (irányítószám) pedig az „addresses"-nek a kérelemben. Most fordítsuk le a teljes kérelmet, amelyet a biztonság kedvéért meg is ismétlünk (a lefordított utasításban az oszlopneveket általánosan fogalmazzuk meg, míg az SQL-utasításban a tényleges oszlopneveket használjuk):

„Add meg minden alkalmazott nevét és címét!"

Fordítás Select first name, last name, street address, city, state, and ZIP Code from the employees table (Válaszd ki a vezetéknevet, keresztnevet, utcát, várost, államot és irányítószámot az alkalmazottak táblájából.)

Tisztázás Select first name, last name, street address, city, state, ~~and~~ ZIP Code from ~~the~~ employees ~~table~~

SQL
```
SELECT EmpFirstName, EmpLastName, EmpStreetAddress,
    EmpCity, EmpState, EmpZipCode
FROM Employees
```

Megjegyzés *Ebben a példában jól megfigyelhető, hogyan használhatunk több oszlopot egy SELECT záradékban. Ezt a lehetőséget a későbbiekben részletesebben is tárgyaljuk.*

A következő példában a második módszer alkalmazását szemléltetjük, azaz felkutatjuk a táblában a kérelem által sugallt oszlopokat. Tegyük fel, hogy az alábbi kérelmet szeretnénk lefordítani:

„Milyen fajta órákat kínálunk jelenleg?"

Első pillantásra bonyolultnak tűnik meghatározni a lefordított utasítást ebből a kérdésből, hiszen nem jelöl ki számunkra oszlopokat vagy bármilyen kiválasztandó elemet, ezek hiányában pedig nem tudjuk megfogalmazni a lefordított utasítást. Mit tehetünk ebben az esetben? Vizsgáljuk meg közelebbről szavanként az egész mondatot, és szűrjük ki, hogy mely pontok *utalhatnak* bizonyos oszlopokra a Classes (Órák) táblából. Mielőtt tovább olvasnánk, szánjunk néhány pillanatot a kérelem tanulmányozására. Találunk ilyen szót?

Esetünkben „fajta" („kind") szó utalhat oszlopnévre a Classes táblában. Miért? Mert az osztályok egy fajtája értelmezhető osztályok egy csoportjaként. Ha van egy category (csoport) oszlopunk a Classes táblában, akkor már tudjuk, hogy melyik oszlopra van szükségünk, hogy teljessé tegyük a lefordított mondatot és ezáltal a SELECT utasítást. Tegyük fel, hogy tényleg van category oszlop a Classes táblában. Most már elvégezhetjük a szokásos három lépést:

„Milyen fajta órákat kínálunk jelenleg?"

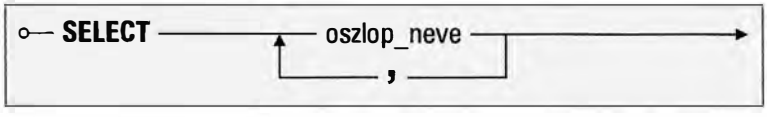
Fordítás	Select category from the classes table (Válaszd ki a csoportot az órák táblájából.)
Tisztázás	Select category from ~~the~~ classes ~~table~~
SQL	`SELECT Category` `FROM Classes`

A példából láthatjuk, hogy a módszer alkalmazása során szinonimákat használtunk bizonyos szavak vagy kifejezések cseréjéhez. Ha meghatároztunk egy szót vagy kifejezést, amely oszlopnévre utalhat, próbáljuk meg egy szinonimájára cserélni. Ez a szinonima valószínűleg meg fog határozni egy oszlopot, amely megtalálható az adatbázisban. Amennyiben az első eszünkbe jutó rokon értelmű szó nem felel meg, próbálkozzunk egy másikkal. Ismételjük ezt az eljárást addig, amíg meg nem találjuk azt a kifejezést, amelynek megfelelő oszlop létezik az adatbázisban, vagy amíg meg nem bizonyosodunk róla, hogy sem az eredeti szó, sem annak szinonimái nem találhatók meg ott oszlopnévként.

Megjegyzés *Hacsak nem jelezzük másképp, a példák SQL-utasításaiban minden oszlop- és táblanév a B függelékben található mintaadatbázisokra vonatkozik. Ez a szabály az összes példára érvényes a könyv hátralevő részében.*

Szélesítsük a látókörünket!

A SELECT utasításokban több oszlopot is ugyanolyan könnyen lekérhetünk, mint egyet, csak soroljuk fel az oszlopok neveit a SELECT záradékban egymástól vesszővel elválasztva. A 4.6. ábra szintaxisdiagramja bemutatja, hogyan adhatunk meg egynél több oszlopot, ezt egy balról jobbra mutató nyíllal szemléltetve az oszlop_neve alatt. A vessző a vonal közepén azt jelöli, hogy egy vesszőt kell beszúrnunk, mielőtt a következő oszlopnevet begépeljük.



4.6. ábra

Több oszlop megadásának módja a SELECT záradékban

Azzal, hogy több oszlopot is megadhatunk egy SELECT utasításban, lehetőségünk nyílik megfogalmazni az alábbihoz hasonló kérelmeket:

„Mutasd meg az alkalmazottak aktuális listáját és a telefonszámaikat!"

Fordítás Select the last name, first name, and phone number of all our employees from the employees table (Válaszd ki minden alkalmazott keresztnevét, vezetéknevét és telefonszámát az alkalmazottak táblájából.)

Tisztázás Select ~~the~~ last name, first name, ~~and~~ phone number ~~of all our employees~~ from ~~the~~ employees ~~table~~

SQL
```
SELECT EmpLastName, EmpFirstName, EmpPhoneNumber
FROM Employees
```

„Milyen nevű és árú termékeket kínálunk, és milyen termékcsoportokba soroltuk az egyes elemeket?"

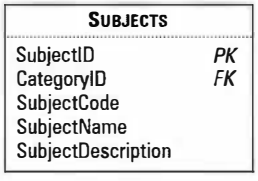
Fordítás Select the name, price, and category of every product from the products table (Válaszd ki minden termék nevét, árát és termékcsoportját a termékek táblájából.)

Tisztázás Select ~~the~~ name, price, ~~and~~ category ~~of every product~~ from ~~the~~ products ~~table~~

SQL
```
SELECT ProductName, RetailPrice, Category
FROM Products
```

Ha több oszlopot adunk meg a SELECT utasítás számára, több információt tekinthetünk át egyszerre. Az oszlopok sorrendje nem kötött; tetszőleges sorrendben megadhatjuk őket, ami lehetőséget nyújt arra, hogy ugyanazokat az információkat többféleképpen jelenítsük meg. Például tegyük fel, hogy a 4.7. ábrán látható táblával dolgozunk, és az alábbi kérelmet fogalmazzuk meg az adatbázis felé:

„Mutass egy listát a tantárgyakról és a tantárgycsoportokról, amelyek alá tartoznak, valamint a kódjaikat, amelyeket a katalógusunkban használunk! A név legyen elöl, azt kövesse a tantárgycsoport, végül a kód!"



4.7. ábra

A Subjects tábla felépítése

Ezt a kérelmet is le tudjuk fordítani a megfelelő SELECT utasításra, pedig a kérelmet megfogalmazó személy az oszlopokat egy bizonyos sorrendben szeretné megkapni: egyszerűen soroljuk fel az oszlopok neveit a megfelelő sorrendben, amikor meghatározzuk a lefordított utasítást. A mondatok SELECT utasítássá alakítása az alábbiak szerint alakul:

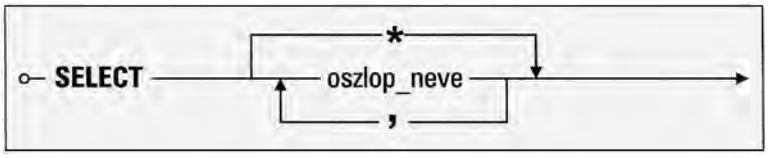
Fordítás	Select the subject name, category ID, and subject code from the subjects table (Válaszd ki a tantárgy nevét, a tantárgycsoport azonosítóját és a tantárgy kódját a tantárgyak táblájából.)
Tisztázás	Select ~~the~~ subject name, category ID, ~~and~~ subject code from ~~the~~ subjects ~~table~~
SQL	`SELECT SubjectName, CategoryID, SubjectCode FROM Subjects`

Rövidítés használata az összes oszlop lekéréséhez

A SELECT záradékban korlátlan mennyiségű oszlopot felsorolhatunk – akár az összeset is a forrástáblából. A következő példa egy olyan SELECT utasítást mutat be, amely a 4.7. ábrán szereplő Subject tábla összes oszlopát felsorolja:

```
SQL     SELECT SubjectID, CategoryID, SubjectCode,
            SubjectName, SubjectDescription
        FROM Subjects
```

Ha egy olyan táblának szeretnénk az összes oszlopát lekérdezni, amelyik sok oszlopot tartalmaz, akkor nagyon sokat kellene gépelnünk. Szerencsére az SQL-szabvány rendelkezésünkre bocsátja a csillag karaktert, amelyet az utasításainkban rövidítésként használhatunk. A 4.8. ábra szintaxisdiagramja azt ábrázolja, hogy a csillagot helyettesítő karakterként használhatjuk az oszlopok felsorolása helyett a SELECT záradékban.



4.8. ábra

A csillag rövidítés használati formája

A csillagot közvetlenül a SELECT után kell megadnunk, hogy az összes oszlopot megkapjuk a FROM záradékban szereplő táblából. Például az előző SELECT utasítás az alábbiak szerint alakul a rövidítés használatával:

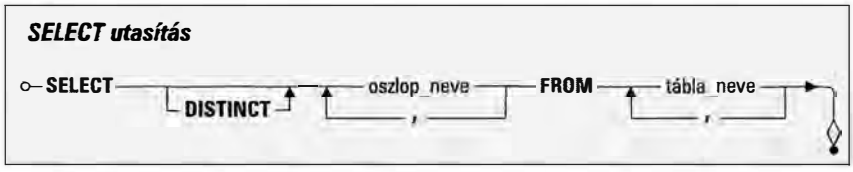
```
SQL     SELECT *
        FROM Subjects
```

Így minden bizonnyal kevesebbet kell gépelnünk! Viszont egy új probléma is felmerül, ha ilyen SELECT utasításokat írunk: a csillag azokat az oszlopokat jelöli, amelyek *jelenleg megtalálhatók* a forrástáblában, tehát ha oszlopokat veszünk fel vagy törlünk, az befolyásolja a SELECT utasítás által visszaadott eredményhalmazt (bár különös módon az SQL-szabvány azt állítja, hogy ennek *nem szabad* hatással lennie az eredményhalmazra). Ez persze csak akkor lényeges, ha az eredményhalmazban következetesen ugyanazokat az oszlopokat kell visszakapnunk. Az adatbázis-kezelőnk nem fog figyelmeztetni arra, hogy egy oszlopot töröltünk a forrástáblából, ha a SELECT záradékban csillagot alkalmaztunk, de figyelmeztetést fogunk kapni, ha nem található egy oszlop, amelyet kimondottan felsoroltunk. A mi esetünkben ugyan ez nem igazán jelent gondot, de fontossá válhat, ha mélyebben elmerülünk az SQL-programozás világában, ezért alapszabályként jegyezzük meg: csak akkor használjunk csillagot, ha „gyors és piszkos" lekérdezést szeretnénk az adatbázishoz intézni, hogy láthassuk az összes információt egy adott táblában. Minden más esetben soroljuk fel az oszlopokat, amelyekre szükségünk van a lekérdezésben. Ennek eredményképpen pontosan azokat az információkat fogjuk megkapni, amelyekre szükségünk van.

Az eddigi példák egyszerű kérelmeken alapultak, amelyek egyetlen táblából igényeltek csak oszlopokat. A harmadik fejezetben megtanuljuk, hogyan fogalmazhatunk meg összetett kérelmeket, amelyekhez akár több tábla oszlopaira is szükség lehet.

A sorismétlés kiküszöbölése

Amikor SELECT utasításokkal dolgozunk, elkerülhetetlenül akadhatnak olyan eredményhalmazaink, amelyekben bizonyos sorok többször szerepelnek. Ilyenkor nem kell kétségbe esnünk: használjuk a DISTINCT kulcsszót a SELECT utasításban, és az eredményhalmazt könnyedén megtisztíthatjuk a több példányban szereplő soroktól. A DISTINCT kulcsszó szintaxisdiagramját a 4.9. ábrán láthatjuk.



4.9. ábra

A DISTINCT kulcsszó utasításformája

Amint a diagramon is látható, a DISTINCT elhagyható kulcsszó, amely az oszlopok listáját előzi meg a SELECT záradékban, és arra kéri az adatbázis-kezelőt, hogy a visszaadott oszlopokat *egy egységként*, sorról sorra értékelje ki, és dobjon el minden olyan sort, amely már megtalálható az eredményben. Az eredményhalmaz így már csak teljesen egyedi sorokat fog tartalmazni. A következő példa szemlélteti, hogy milyen különbséget jelent a DISTINCT kulcsszó egyes esetekben. Tegyük fel, hogy az alábbi kérelemmel fordulunk az adatbázishoz:

„Mely városok képviseltetik magukat a tekebajnokságunkon?"

Nem tűnik bonyolult kérdésnek, úgyhogy lássuk az átalakítást:

Fordítás	Select city from the bowlers table (Válaszd ki a várost a tekejátékosok táblájából.)
Tisztázás	Select city from ~~the~~ bowlers ~~table~~
SQL	`SELECT City` `FROM Bowlers`

Ezzel a SELECT utasítással az a gond, hogy minden város *minden előfordulását* megjeleníti a Bowlers (Tekejátékosok) táblából. Például ha 20 ember érkezett Bellevue városából, 7 ember Kent-ből és 14 Seattle-ből, az eredményhalmaz 20 példányban fogja tartalmazni Bellevue-t, 7 példányban Kent-et és 14 példányban Seattle-t. Ezek a többször szereplő információk szükségtelenek, mert nyilván csak *egyszer* szeretnénk látni minden város nevét, ami szerepel a Bowlers táblában. Ezt a problémát úgy oldhatjuk meg, ha a DISTINCT kulcsszót használjuk a SELECT utasításban.

Fordítsuk le újra a kérelmet, ezúttal a DISTINCT használatával. Figyeljük meg, hogy a „distinct" („eltérő") szó a fordítási és a tisztázási lépésben is szerepel:

„Mely városok képviseltetik magukat a tekebajnokságunkon?"

Fordítás	Select the distinct city values from the bowlers table (Válaszd ki az eltérő városértékeket a tekejátékosok táblájából.)
Tisztázás	Select ~~the~~ distinct city ~~values~~ from ~~the~~ bowlers ~~table~~

```
SQL     SELECT DISTINCT City
        FROM Bowlers
```

Most már pontosan azt az eredményt kapjuk, amit kerestünk: egy példányban minden város nevét, ami előfordul a Bowlers táblában.

A DISTINCT kulcsszó több oszlopon is hasonlóan használható. Módosítsuk az előző példában szereplő lekérdezést úgy, hogy a „city" (város) és a „state" (állam) oszlopokat is lekérjük a Bowlers táblából. Az új SELECT utasításunk így fog festeni:

```
SELECT DISTINCT City, State FROM Bowlers
```

Ez a SELECT utasítás egy olyan eredményhalmazzal tér vissza, amely csak egyedi sorokat tartalmaz, és határozottan megkülönbözteti az ugyanolyan nevű városokat, például különbözőnek tekinti a „Portland, ME,", a „Portland, OR,", a „Hollywood, CA," és a „Hollywood, FL," sorokat. (A városnév utáni két betűs rövidítés különböző amerikai államokra utal, amelyeket a tábla state oszlopában tárolunk.) Érdemes megjegyeznünk, hogy a legtöbb adatbázis-kezelő rendszer a kimenetet aszerint rendezi, hogy milyen sorrendben adtuk meg az oszlopokat a lekérdezésben. A mi esetünkben az alábbi lesz az értékek sorrendje: „Hollywood, CA,", „Hollywood, FL,", „Portland, ME," és „Portland, OR," (jóllehet az SQL-szabvány nem követeli meg, hogy az eredmény ilyen sorrendben legyen). Ha egy kívánt sorrendet szeretnénk biztosítani, olvassunk tovább, mert a következő részben az erre szolgáló ORDER BY záradékról fogunk tanulni.

A DISTINCT kulcsszó nagyon hasznos eszköz, ha megfelelően alkalmazzuk, de csak akkor használjuk, ha tényleg egyedi sorokat szeretnénk kapni az eredményhalmazban.

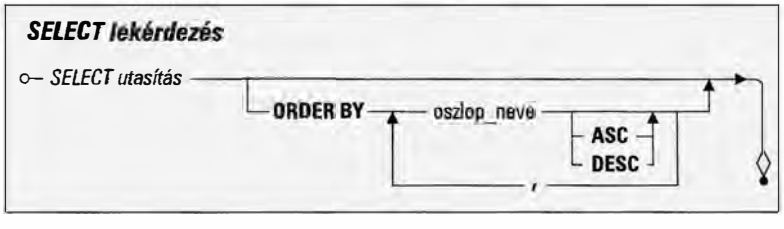
Figyelem! *A grafikus felhasználói felülettel rendelkező adatbázis-kezelők általában sorokból és oszlopokból álló rácsként jelenítik meg a lekérdezések eredményhalmazát. Ha új értéket gépelünk egy oszlopba vagy sorba, az adatbázis-kezelő automatikusan frissíti a táblában tárolt értéket. (Tulajdonképpen egy UPDATE lekérdezést hajt végre a színfalak mögött – erről bővebben olvashatunk a 15. fejezetben.)*

Ugyanakkor az összes adatbázis-kezelőben, amelyet a könyv szerzői tanulmányoztak, az eredményhalmazban szereplő sorok nem voltak frissíthetők, ha a DISTINCT kulcsszó is szerepelt a lekérdezésben. Ahhoz, hogy frissítsünk egy oszlopot egy sorban, az adatbázis-kezelőnek tudnia kell egyedileg azonosítania azt a sort és oszlopot, amelyet meg akarunk változtatni. A DISTINCT használatával azonban azok az értékek, amelyeket az eredmény egyes soraiban látunk, akár tucatnyi egyező sor kiértékeléséből is származhatnak. Ha megpróbálunk frissíteni egy oszlopot, az adatbázis-kezelő nem fogja tudni, hogy melyik sort kell frissítenie, és azt sem fogja érteni, ha mi úgy gondoltuk, hogy az összes olyan sort módosítani szeretnénk, ami az adott értéket tartalmazza.

Az információk rendezése

A fejezet elején azt olvashattuk, hogy a SELECT művelet három kisebb műveletre tagolható: a SELECT utasításra, a SELECT kifejezésre és a SELECT lekérdezésre. Arról is olvashattunk, hogy ezeket a műveleteket sokféleképpen kombinálhatjuk annak érdekében, hogy választ kaphassunk a bonyolultabb kérdésekre. Ahhoz, hogy egy eredményhalmaz sorait rendezett formában kapjuk meg, szintén ezeknek a műveleteknek a kombinációjára van szükség.

Alapesetben a SELECT utasítás által szolgáltatott eredményhalmaz sorai nem rendezettek. A sorrendjük attól függően alakul, hogy az adatok fizikailag milyen sorrendben szerepelnek a táblában. (Ez dinamikusan változhat az adatbázis-kezelő döntése alapján, attól függően, hogy miként tudja a leghatékonyabban kiszolgálni a kérésünket.) Az egyetlen módja, hogy az eredményhalmazt rendezetten kapjuk meg, ha a SELECT utasításunkat egy SELECT lekérdezésbe ágyazzuk, amint az a 4.10. ábrán is megfigyelhető. A SELECT lekérdezés tulajdonképpen egy ORDER BY záradékkal ellátott SELECT utasítás. Ennek a záradéknak a segítségével meghatározhatjuk az eredményhalmaz sorainak sorrendjét. A későbbi fejezetekben látni fogjuk, hogy a SELECT utasítást másik SELECT utasításba vagy SELECT kifejezésbe is beágyazhatjuk, és így összetett kérelmeket fogalmazhatunk meg. Ezzel ellentétben a SELECT lekérdezés semmilyen formában nem ágyazható be.



4.10. ábra
A SELECT lekérdezés szintaxisdiagramja

Megjegyzés *A könyvben az SQL-szabványban szereplő, illetve az adatbázis-kezelők által széles körben használt fogalmakat használjuk. A 2003-as SQL-szabvány azonban az ORDER BY záradékot egy kurzor (sormutató) részeként határozza meg. A kurzor olyan objektum, amelyet egy alkalmazáson belül határozunk meg egy tömb részeként (a tömb értékek listája, amely egy logikai táblát, például egy allekérdezést alkot – az allekérdezésekről a 11. fejezetben olvashatunk), vagy egy skaláris allekérdezés részeként (amely egy olyan allekérdezés, amely egyetlen értékkel tér vissza). A kurzorok és tömbök részletes tárgyalása túlmutat ennek a könyvnek a keretein. Mivel az SQL-nek szinte minden megvalósítása lehetővé teszi az ORDER BY záradék használatát a SELECT utasítások végén, amely ebben a formában nézettáblaként is menthető, a könyv szerzői bevezették a SELECT lekérdezés fogalmát ennek az utasítástípusnak a leírására. Ez lehetővé teszi a számunkra, hogy könnyebben tárgyalhassuk a lekérdezések eredményeképpen létrejövő, online vagy jelentés formájában megtekinthető kimenetek rendezésének lehetőségét. Habár tudatában vagyunk annak, hogy a 2007/2008-as szabványtervezet megengedi az ORDER BY több más helyen való használatát is, ebben a könyvben a SELECT lekérdezés fogalmára támaszkodva járjuk körül ezt a témát.*

Az ORDER BY segítségével egy adott SELECT utasítás eredményhalmazát annak egy vagy több oszlopa szerint, növekvő és csökkenő sorrendben egyaránt rendezhetjük. Csak azokat az oszlopokat használhatjuk az ORDER BY záradékban, amelyeket a SELECT záradékban is felsoroltunk. (Erről a megszorításról az SQL-szabvány is rendelkezik, ennek ellenére néhány gyártó lehetővé teszi, hogy ezt teljes mértékben figyelmen kívül hagyjuk. A könyv példáiban azonban betartottuk a szabvány vonatkozó korlátozásait.) Az oszlopokat egymástól vesszővel elválasztva kell megadnunk. A SELECT lekérdezés akkor tér vissza az eredményhalmazzal, ha a rendezést befejezte.

Megjegyzés *Az ORDER BY záradék nem változtatja meg a táblában lévő sorok fizikai sorrendjét. Ennek megvalósításához tanulmányozzuk át az adatbázis-kezelő szoftverünk dokumentációját.*

Mielőtt továbblépnénk: jelsorrend

Mielőtt közelebbről szemügyre vennénk néhány konkrét példát a SELECT lekérdezések használatára, néhány szót kell ejtenünk a jelsorrendről. Az ORDER BY záradék attól függően állapítja meg a rendezés során az értékek sorrendjét, hogy az adatbázis-kezelő szoftver milyen jelsorrendet használ. A jelsorrend szabja meg az operációs rendszer által meghatározott nyelvi karakterkészlet karaktereinek elsőbbségi sorrendjét, például eldönti, hogy a kisbetűk előnyt élvezzenek-e a nagybetűkkel szemben, vagy hogy a kis- és nagybetűket egyáltalán meg kell-e különböztetni. Az alapértelmezett jelsorrend meghatározásához tanulmányozzuk át az adatbázis-kezelő rendszerünk dokumentációját, esetleg kérdezzük meg az adatbázisunk rendszergazdáját. A jelsorrendekről bővebben is beszélünk a 6. fejezetben.

Rendezzük a sorainkat!

Az ORDER BY záradék segítségével sokkal kifejezőbb módon jeleníthetjük meg a lekérdezett információkat. Ez egyaránt igaz az egyszerű és bonyolult lekérdezésekre. Mostantól úgy érdemes megfogalmaznunk a kérdéseinket, hogy a rendezési igényeinket is kifejezzük velük. Például egy olyan kérdés, mint a *„Milyen fajta órákat kínálunk jelenleg?"* úgy lenne átfogalmazható, hogy *„Készíts listát az általunk kínált órák típusairól, és jelenítsd meg őket ábécésorrendben!"*

Mielőtt munkához látnánk a SELECT lekérdezéssel, kicsit igazítanunk kell a lefordított utasításainkon: a mondat vége egy új résszel bővül, hogy kifejezhessük a sorrendre vonatkozó elvárásainkat. Használjuk az alábbi formát az utasítások lefordításához:

Select <elem> from the <forrás> **and order by** <oszlop(ok)>

Most, hogy a kérelmeket olyan részekkel bővítettük, mint a „rendezd az eredményt városok szerint" vagy „jelenítsd meg évszámok szerinti sorrendben", esetleg „írd ki őket vezeték-, majd keresztnév szerint", vizsgáljuk meg a kérdésünket alaposabban, és döntsük el, milyen oszlopokra lesz szükségünk a rendezéshez. Ez egyszerű feladat, mivel a kérelmek-

ben általában a fentiekhez hasonló megfogalmazásokat használunk, és a szükséges oszlopok általában magától értetődőek. Ha kiválasztottuk az oszlopot vagy oszlopokat, egyszerűen helyettesítsük be azokat az <oszlop(ok)> helyére a lefordított utasításban. Nézzük meg, hogyan valósul meg ez a gyakorlatban egy egyszerű kérelem esetében:

„Készíts listát az általunk kínált órák típusairól, és jelenítsd meg őket ábécésorrendben!"

Fordítás Select category from the classes table and order by category
(Válaszd ki a tantárgycsoportokat az órák táblájából, és rendezd azokat tantárgycsoport szerint.)

Tisztázás Select category from ~~the~~ classes ~~table and~~ order by category

SQL
```
SELECT Category
FROM Classes
ORDER BY Category
```

Ebben a példában biztosak lehetünk benne, hogy a rendezés a Category oszlop szerint történik, mivel ezt az egy oszlopot adtuk meg a lekérdezésben. Azt is nyugodtan feltételezhetjük, hogy a rendezés növekvő sorrendben fog történni, mivel a lekérdezésben nem kértük ennek az ellenkezőjét. Az SQL-szabvánnyal összhangban a növekvő sorrendben történő rendezés lesz az alapértelmezett, amennyiben nem adjuk meg a rendezés sorrendjét. Ha teljesen biztosak szeretnénk lenni a dolgunkban, szúrjuk be az ASC (ascending, növekvő) kulcsszót a Category név után az ORDER BY záradékban.

Az alábbi kérelemben az oszlop, ami szerint rendezni kell, még egyértelműbben meghatározott:

„Sorold fel a beszállítók neveit, irányítószám szerinti sorrendben!"

Fordítás Select vendor name and ZIP Code
from the vendors table and order by ZIP Code
(Válaszd ki a beszállítók nevét és irányítószámát a beszállítók táblájából, és rendezd irányítószám szerint.)

Tisztázás Select vendor name ~~and~~ ZIP Code
from ~~the~~ vendors ~~table and~~ order by ZIP Code

SQL
```
SELECT VendName, VendZipCode
FROM Vendors
ORDER BY VendZipCode
```

A legtöbben egyértelműen a tudtunkra adják, ha az információkat csökkenő sorrendbe rendezve szeretnék látni. Ha ez a helyzet, és az eredményhalmazt fordított sorrendben kell megjelenítenünk, szúrjuk be a DESC (descending, csökkenő) kulcsszót a megfelelő oszlop után az ORDER BY záradékban. Az alábbi példa bemutatja, hogyan kell módosítanunk az előző példában látott SELECT utasítást ahhoz, hogy az információk irányítószám szerint csökkenő sorrendben jelenjenek meg:

SQL
```
SELECT VendName, VendZipCode
FROM Vendors
ORDER BY VendZipCode DESC
```

A következő példa egy bonyolultabb kérelmet mutat be, amelyhez több oszlop szerinti rendezésre lesz szükség. Az egyetlen különbség az előző két példához képest, hogy ez esetben több oszlopot sorolunk fel az ORDER BY záradékban (ügyeljünk rá, hogy az oszlopokat vesszővel válasszuk el egymástól, ahogy azt a 4.10. ábra szintaxisdiagramján láthattuk):

„Jelenítsd meg az alkalmazottak nevét, telefonszámát és azonosítóját, vezeték-, majd keresztnév szerint rendezve!"

Fordítás Select last name, first name, phone number, and employee ID from the employees table and order by last name and first name (Válaszd ki a vezetékneveket, keresztneveket, telefonszámokat és alkalmazott-azonosítókat az alkalmazottak táblájából, és rendezd vezetéknév, majd keresztnév szerint.)

Tisztázás Select last name, first name, phone number, ~~and~~ employee ID from ~~the~~ employees ~~table and~~ order by last name ~~and~~ first name

SQL
```
SELECT EmpLastName, EmpFirstName,
    EmpPhoneNumber, EmployeeID
FROM Employees
ORDER BY EmpLastName, EmpFirstName
```

Az ORDER BY záradék egy érdekes lehetősége, hogy különböző rendezési sorrendet adhatunk meg az egyes oszlopok számára. Az előző példát alapul véve kérhetjük a vezetékneveket az ábécé szerint csökkenő, a keresztneveket pedig növekvő sorrendben. Az ehhez szükséges módosítások után a SELECT utasítás így fest:

SQL
```
SELECT EmpLastName, EmpFirstName, EmpPhoneNumber,
    EmployeeID
FROM Employees
ORDER BY EmpLastName DESC, EmpFirstName ASC
```

Bár az ASC kulcsszót nem muszáj kiírnunk, az utasítás sokkal egyértelműbb lesz, ha megtesszük.

Ez a példa egy érdekes kérdést is felvet: van-e jelentősége az oszlopsorrendnek az ORDER BY záradékban? A válasz: *„Igen!"* Az oszlopok sorrendje azért fontos, mert az adatbázis-kezelő az oszlopokat balról jobbra haladva értékeli ki, ráadásul a sorrend jelentősége a megadott oszlopok számával egyenes arányban nő. Mindig olyan sorrendben soroljuk fel az oszlopokat, amilyen sorrendben az eredményt is rendezni szeretnénk.

Megjegyzés *A Microsoft által gyártott adatbázis-kezelők (Microsoft Office Access és Microsoft SQL Server) tartalmaznak egy érdekes bővítést, amelynek a segítségével a sorok egy részhalmazát is lekérdezhetjük az ORDER BY szerinti rendezés alapján, ha a SELECT záradékban megadjuk a TOP kulcsszót. Az alábbi lekérdezéssel például megtalálhatjuk az öt legdrágább terméket a Sales Orders adatbázisban:*

```
SELECT TOP 5 ProductName, RetailPrice
    FROM Products
    ORDER BY RetailPrice DESC
```

Az adatbázis-kezelő ár szerint csökkenő sorrendbe rendezi a Products tábla sorait, majd visszatér az első öt sorral. Mindkét adatbázisrendszer lehetőséget ad arra is, hogy a kért sorok számát az összes sor százalékában adjuk meg. Az alábbi lekérdezéssel például megkaphatjuk az ár alapján a felső 10 százalékba eső termékeket:

```
SELECT TOP 10 PERCENT ProductName, RetailPrice
    FROM Products
    ORDER BY RetailPrice DESC
```

Ha az ORDER BY-t egy nézettáblában szeretnénk megadni, az SQL Servernek mindenképpen meg kell adnunk a TOP kulcsszót is, tehát ha az összes sorra szükségünk van, akkor TOP 100 PERCENT-et kell írnunk. Ezért tapasztalhatjuk azt, hogy az SQL Server minden olyan példanézettáblájában, amely ORDER BY záradékot tartalmaz, a TOP 100 PERCENT kulcsszavak is megtalálhatók. A Microsoft Access nem tartalmazza ezt a megkötést.

Mentés

Mentsük a SELECT utasításainkat – minden fontos adatbázis-kezelő lehetőséget biztosít erre. Az utasítások mentésével elkerülhetjük, hogy újra és újra meg kelljen írnunk őket, valahányszor ugyanazt a kérelmet szeretnénk intézni az adatbázishoz. Ha a mentés során beszédes nevet választunk, az segít majd emlékeznünk, hogy milyen információkat kérdez le számunkra az utasítás. Ha az adatbázis-kezelőnk lehetőséget ad rá, adjunk tömör leírást az utasítás céljáról. Ennek hasznosságát majd akkor értjük meg igazán, ha bizonyos SELECT utasításokkal már nem találkoztunk egy ideje, és vissza kell emlékeznünk rá, hogy miért hoztuk létre őket annak idején.

A mentett SELECT utasítások néhány adatbázisszoftverben lekérdezésként, másokban pedig nézettáblaként, függvényként vagy tárolt eljárásként jelennek meg. A megvalósítás módjától függetlenül minden adatbázis-kezelő lehetőséget ad a mentett utasítások későbbi végrehajtására és eredményhalmazuk használatára.

Megjegyzés *A továbbiakban a lekérdezés szó a mentett SELECT utasításokra, a végrehajtás kifejezés pedig a velük való műveletekre vonatkozik majd.*

A mentett lekérdezéseket általában kétféleképpen hajthatjuk végre: egy interaktív eszköz (például egy eszköztárgomb vagy lekérdezésrács) segítségével, vagy egy kódblokkból. Az első módszert gyakran fogjuk alkalmazni, a másodikkal azonban addig nem is kell törődnünk, amíg el nem kezdjük használni az adatbázis-kezelőnk programozási nyelvét. Mi arra vállalkoztunk, hogy megtanítsuk az SQL-utasítások megfogalmazását és használatát, viszont az Olvasó feladata elsajátítani a lekérdezések létrehozásának, mentésének és végrehajtásának módját a saját adatbázis-kezelő szoftverében.

Példák

Miután megismertük a SELECT utasítás és a SELECT lekérdezés alapvető jellemzőit, nézzünk néhány példát, hogyan alkalmazhatók ezek a műveletek különböző helyzetekben! Az alábbi példák mindegyike a mintaadatbázisokra támaszkodik, és a SELECT utasítások, a SELECT lekérdezések, valamint a lefordított utasítások számára az oszlopok meghatározását segítő két módszer használatát mutatja be.

Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név kettős célt szolgál: azonosítja az eredményhalmazt, és a példában szereplő SQL-utasításhoz rendelt névként is szolgál.

Ha meglepődnénk azon, hogy minden SQL-utasításnak nevet adtunk, jó ha tudjuk, hogy azért tettünk így, mert mentettük őket. Valójában a példákban szereplő összes SQL-utasítást elneveztük és mentettük, amelyek itt vagy a könyv további részeiben szerepelnek.

Minden lekérdezést a megfelelő mintaadatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH04"-gyel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve betölthetjük és kipróbálhatjuk, ezáltal lehetőségünk nyílik rá, hogy megfigyeljük az utasításokat működés közben, mielőtt megpróbálnánk saját magunk megírni azokat.

Megjegyzés *Emlékeztetőül: a példákban szereplő oszlop- és táblanevek a B függelékben található mintaadatbázisokból származnak.*

Sales Orders adatbázis

„Mutasd meg az összes beszállító nevét!"

Fordítás Select the vendor name from the vendors table
(Válaszd ki a beszállítók nevét a beszállítók táblájából.)

Tisztázás Select ~~the~~ vendor name from ~~the~~ vendors ~~table~~

SQL
```
SELECT VendName
FROM Vendors
```

CH04_Vendor_Names (10 sor)

VendName
Shinoman, Incorporated
Viscount
Nikoma of America
ProFormance
Kona, Incorporated
Big Sky Mountain Bikes
Dog Ear
Sun Sports Suppliers
Lone Star Bike Supply
Armadillo Brand

„Milyen nevű és árú termékeket kínálunk?"

Fordítás Select product name, retail price from the products table
(Válaszd ki a termékek nevét és fogyasztói árát a termékek táblájából.)

Tisztázás Select product name, retail price from ~~the~~ products ~~table~~

SQL
```
SELECT ProductName, RetailPrice
FROM Products
```

CH04_Product_Price_List (40 sor)

ProductName	Retail Price
Trek 9000 Mountain Bike	$1,200.00
Eagle FS-3 Mountain Bike	$1,800.00
Dog Ear Cyclecomputer	$75.00
Victoria Pro All Weather Tires	$54.95
Dog Ear Helmet Mount Mirrors	$7.45
Viscount Mountain Bike	$635.00
Viscount C-500 Wireless Bike Computer	$49.00
Kryptonite Advanced 2000 U-Lock	$50.00
Nikoma Lok-Tight U-Lock	$33.00
Viscount Microshell Helmet	$36.00
<< *további sorok* >>	

„Mely államokban élnek vásárlóink?"

Fordítás Select the distinct state values from the customers table
(Válaszd ki az eltérő állam értékeket a vásárlók táblájából.)

Tisztázás Select ~~the~~ distinct state ~~values~~ from ~~the~~ customers ~~table~~

SQL
```
SELECT DISTINCT CustState
FROM Customers
```

CH04_Customer_States (4 sor)

CustState
CA
OR
TX
WA

Entertainment Agency adatbázis

„Sorold fel az összes előadót és a városukat, az eredményt városok, majd nevek szerint növekvő ábécésorrendbe rendezve!"

Fordítás Select city and stage name from the entertainers table and order by city and stage name
(Válaszd ki a városokat és a művészneveket az előadók táblájából, és rendezd város, majd művésznév alapján.)

Tisztázás Select city and stage name from the entertainers table and order by city and stage name

SQL
```
SELECT EntCity, EntStageName
FROM Entertainers
ORDER BY EntCity ASC, EntStageName ASC
```

CH04_Entertainer_Locations (13 sor)

EntCity	EntStageName
Auburn	Caroline Coie Cuartet
Auburn	Topazz
Bellevue	Jazz Persuasion
Bellevue	Jim Glynn
Bellevue	Susan McLain
Redmond	Carol Peacock Trio
Redmond	JV & the Deep Six
Seattle	Coldwater Cattle Company
Seattle	Country Feeling
Seattle	Julia Schnebly
<< *további sorok* >>	

„Add meg a rendezvénydátumok egyedi értékeket tartalmazó listáját! Az, hogy egy-egy dátumhoz hány rendezvény tartozik, nem érdekes!"

Fordítás Select the distinct start date values from the engagements table
(Válaszd ki az eltérő kezdődátum-értékeket a rendezvények táblájából.)

Tisztázás Select ~~the~~ distinct start date ~~values~~ from ~~the~~ engagements ~~table~~

SQL
```
SELECT DISTINCT StartDate
FROM Engagements
```

CH04_Engagement_Dates (64

StartDate
2007-09-01
2007-09-10
2007-09-11
2007-09-15
2007-09-17
2007-09-18
2007-09-24
2007-09-29
2007-09-30
2007-10-01
<< további sorok >>

School Scheduling adatbázis

„Mutass meg minden információt az órákról!"

Fordítás Select all columns from the classes table
(Válaszd ki az összes oszlopot az órák táblájából.)

Tisztázás Select ~~all-columns~~ * from ~~the~~ classes ~~table~~

SQL
```
SELECT *
FROM Classes
```

CH04_Class_Information (76 sor)

ClassID	SubjectID	ClassRoomID	Credits	StartTime	Duration	<<további oszlopok>>
1000	11	1231	5	10:00	50	...
1002	12	1619	4	15:30	110	...
1004	13	1627	4	08:00	50	...
1006	13	1627	4	09:00	110	...
1012	14	1627	4	13:00	170	...
1020	15	3404	4	13:00	110	...
1030	16	1231	5	11:00	50	...
1031	16	1231	5	14:00	50	...
1156	37	3443	5	08:00	50	...
1162	37	3443	5	09:00	80	...
<< további sorok >>						

„Add meg a kampusz épületeinek listáját, valamint minden épülethez, hogy hány emeletes! Rendezd a listát épületek szerint, növekvő sorrendben!"

Fordítás Select building name and number of floors from the buildings table, ordered by building name
(Válaszd ki az épületek nevét és a szintek számát az épületek táblájából, az épület neve szerint rendezve.)

Tisztázás Select building name ~~and~~ number of floors from ~~the~~ buildings ~~table~~, order~~ed~~ by building name

SQL
```
SELECT BuildingName, NumberOfFloors
FROM Buildings
ORDER BY BuildingName ASC
```

CH04_Building_List (6 sor)

BuildingName	NumberOfFloors
Arts and Sciences	3
College Center	3
Instructional Building	3
Library	2
PE and Wellness	1
Technology Building	2

Bowling League adatbázis

„Milyen helyszíneken rendezünk tornákat?"

Fordítás Select the distinct tourney location values from the tournaments table
(Válaszd ki a különböző tornahelyszín-értékeket a tornák táblájából.)

Tisztázás Select ~~the~~ distinct tourney location ~~values~~ from ~~the~~ tournaments table

SQL
```
SELECT DISTINCT TourneyLocation
FROM Tournaments
```

CH04_Tourney_Locations (7 sor)

TourneyLocation
Acapulco Lanes
Bolero Lanes
Imperial Lanes
Red Rooster Lanes
Sports World Lanes
Thunderbird Lanes
Totem Lanes

„Sorold fel az összes torna dátumát és helyszínét! A dátumok csökkenő sorrendben legyenek, a helyszínek pedig ábécésorrendben!"

Fordítás Select tourney date and location from the tournaments table and order by tourney date in descending order and location in ascending order
(Válaszd ki a tornák dátumát és helyszínét a tornák táblájából, és rendezd dátum szerint csökkenő sorrendben, majd helyszín szerint növekvő sorrendben.)

Tisztázás Select tourney date ~~and~~ location from ~~the~~ tournaments ~~table~~ ~~and~~ order by tourney date ~~in~~ descending ~~order and~~ location ~~in~~ ascending ~~order~~

SQL
```
SELECT TourneyDate, TourneyLocation
FROM Tournaments
ORDER BY TourneyDate DESC, TourneyLocation ASC
```

CH04_Tourney_Dates (14 sor)

TourneyDate	TourneyLocation
2008-08-15	Totem Lanes
2008-08-08	Imperial Lanes
2008-08-01	Sports World Lanes
2008-07-25	Bolero Lanes
2008-07-18	Thunderbird Lanes
2008-07-11	Red Rooster Lanes
2007-12-04	Acapulco Lanes
2007-11-27	Totem Lanes
2007-11-20	Sports World Lanes
2007-11-13	Imperial Lanes
<< *további sorok* >>	

Recipes adatbázis

„Milyen típusú receptjeink vannak, és melyek az egyes típusokhoz tartozó receptek nevei? Rendezd az információkat típus és receptnév szerint!"

Fordítás Select recipe class ID and recipe title from the recipes table and order by recipe class ID and recipe title
(Válaszd ki a receptosztály-azonosítókat és receptek nevét a receptek táblájából, receptosztály-azonosító és receptnév szerint rendezve.)

Tisztázás Select recipe class ID ~~and~~ recipe title from ~~the~~ recipes ~~table~~ ~~and~~ order by recipeclass ID ~~and~~ recipe title

SQL
```
SELECT RecipeClassID, RecipeTitle
FROM Recipes
ORDER BY RecipeClassID ASC, RecipeTitle ASC
```

CH04_Recipe_Classes_And_Titles (15 sor)

RecipeClassID	RecipeTitle
1	Fettuccini Alfredo
1	Huachinango Veracruzana (Red Snapper,Veracruz style)
1	Irish Stew
1	Pollo Picoso
1	Roast Beef
1	Salmon Filets in Parchment Paper
1	Tourtière (French-Canadian Pork Pie)
2	Asparagus
2	Garlic Green Beans
3	Yorkshire Pudding
<< *további sorok* >>	

„Sorold fel az eltérő receptosztály-azonosítókat a receptek táblájából!"

Fordítás Select the distinct recipe class ID values from the recipes table (Válaszd ki az eltérő receptosztály-azonosító értékeket a receptek táblájából.)

Tisztázás Select ~~the~~ distinct recipe class ID ~~values~~ from ~~the~~ recipes ~~table~~

SQL
```
SELECT DISTINCT RecipeClassID
FROM Recipes
```

CH04_Recipe_Class_Ids (6 so

RecipeClassID
1
2
3
4
5
6

Összefoglalás

Ebben a fejezetben megismerkedtünk a SELECT művelettel, és megtanultuk, hogy ez az egyik az SQL négy alapvető adatkezelő művelete közül. (A másik három művelet az UPDATE, az INSERT és a DELETE, amelyekkel az 5. fejezetben foglalkozunk majd.) Megtanultuk, hogyan bontható fel a SELECT művelet három kisebb műveletre: a SELECT utasításra, a SELECT kifejezésre és a SELECT lekérdezésre.

Ez után a SELECT utasítás részleteit tárgyaltuk, és megismerkedtünk az utasítást felépítő záradékokkal. Megtanultuk, hogy a SELECT és a FROM záradék alapvető, és mindig meg kell adnunk őket, ha információkért fordulunk az adatbázishoz, valamint azt is, hogy a további záradékok – a WHERE, a GROUP BY és a HAVING – akár el is hagyhatók, ám a segítségükkel szűrhetővé válnak a SELECT által szolgáltatott információk.

Ezt követően röviden kitértünk az adat és az információ különbözőségére, és megtanultuk, hogy az adatbázis soraiban tárolt értékek adatok, és hogy az információ olyan adat, amelyet feldolgoztunk annak érdekében, hogy jelentést hordozzon valaki számára. Megtudtuk, hogy a SELECT utasítás által sorokként visszaadott információkat eredményhalmaznak nevezzük.

Ezt követően arról szóltunk, hogy miként kérhetünk le információkat egy adatbázisból. Először megtanultuk a SELECT utasítás alapvető formáját, majd elsajátítottuk a SELECT utasítás helyes felépítését egy háromlépéses eljárással, amellyel a hétköznapi nyelven megfogalmazott kérdésünket nyelvtanilag helyes SQL-utasítássá alakíthatjuk. Megtudtuk, hogy a SELECT záradékban kettő vagy több oszlopot is megadhatunk, kibővítve ezáltal az adatbázisból egy lépésben lekérdezhető információk mennyiségét. Ezt követően vetettünk egy gyors pillantást a DISTINCT kulcsszóra, amelyről megtudtuk, hogy a segítségével eltüntethetjük a többször előforduló sorokat az eredményhalmazból.

A továbbiakban a SELECT lekérdezésről olvastunk. Megtanultuk, hogyan kombinálható a SELECT utasítással az eredményhalmaz rendezése érdekében, és megtudtuk, hogy mindez azért szükséges, mert a SELECT lekérdezés az egyetlen SELECT művelet, amely ORDER BY záradékot tartalmaz. Megtanultuk, hogy az ORDER BY záradékot kell alkalmaznunk, ha az információkat egy vagy több oszlop alapján rendezni szeretnénk, valamint hogy minden oszlopra egyenként beállíthatjuk, hogy növekvő vagy csökkenő sorrendben legyen-e rendezve. Ez után röviden tárgyaltuk a SELECT utasítások mentésének lehetőségét, és megtanultuk, hogy a későbbi felhasználás céljára lekérdezés vagy nézettábla formájában menthetjük őket.

Végül számos példát tanulmányoztunk, amelyek a mintaadatbázisok különféle tábláira támaszkodtak. A példák bemutatták, hogyan alkalmazhatjuk a fejezetben megismert elveket és eljárásokat bizonyos helyzetekben. A következő fejezetben behatóbban megismerkedünk a SELECT záradékkal, és meglátjuk, hogyan nyerhetünk ki mást is, mint információkat az oszlopokból.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Order adatbázis

1. *„Mutass meg minden információt az alkalmazottainkról!"*
   A megoldás itt található: CH04_Employee_Information (8 sor).
2. *„Mutasd meg azoknak a városoknak a listáját ábécésorrendben, ahol beszállítóink vannak, az egyes városokban található beszállítók nevével együtt!"*
   A megoldás itt található: CH04_Vendor_Locations (10 sor).

Entertainment Agency adatbázis

1. *„Add meg az összes ügynökünk nevét és telefonszámát, és rendezd őket vezetéknév, majd keresztnév alapján!"*
   A megoldás itt található: CH04_Agent_Phone_List (9 sor).
2. *„Adj információkat minden rendezvényünkről!"*
   A megoldás itt található: CH04_Engagement_Information (111 sor).
3. *„Sorold fel az összes rendezvényünket és a hozzájuk rendelt kezdő dátumot! Rendezd őket dátum alapján csökkenő, rendezvény alapján pedig növekvő sorrendbe!"*
   A megoldás itt található: CH04_Scheduled_Engagements (111 sor).

School Scheduling adatbázis

1. *„Mutass teljes körű listát az általunk kínált tantárgyakról!"*
   A megoldás itt található: CH04_Subject_List (56 sor).
2. *„Milyen címek tartoznak a karunkhoz?"*
   A megoldás itt található: CH04_Faculty_Titles (3 sor).
3. *„Sorold fel minden tanár nevét és telefonszámát, és rendezd a listát keresztnév, majd vezetéknév szerint!"*
   A megoldás itt található: CH04_Staff_Phone_List (27 sor).

Bowling League adatbázis

1. *„Írd ki az összes csapatot ábécésorrendben!"*
   A megoldás itt található: CH04_Team_List (8 sor).
2. *„Mutass meg minden pontszámokra vonatkozó információt minden tagunkhoz!"*
   A megoldás itt található: CH04_Bowling_Score_Information (1344 sor).
3. *„Sorold fel a tekejátékosokat a lakcímükkel együtt, és rendezd ábécésorrendbe!"*
   A megoldás itt található: CH04_Bowler_Names_Addresses (32 sor).

Recipes adatbázis

1. *„Sorold fel az összes nyilvántartott hozzávalót!"*
   A megoldás itt található: CH04_Complete_Ingredients_List (79 sor).
2. *„Mutass meg minden fontosabb receptinformációt, a recept neve szerint ábécésorrendbe rendezve!"*
   A megoldás itt található: CH04_Main_Recipe_Information (15 sor).

5

Kapjunk többet egyszerű oszlopoknál!

„A tények makacs dolgok"
– Tobias Smollett
Gil Blas de Santillane

A fejezet témakörei

- Mik azok a kifejezések?
- Milyen típusú adatokat szeretnénk megjeleníteni?
- Adattípusok megváltoztatása: a CAST függvény
- Konkrét értékek meghatározása
- A kifejezések típusai
- Kifejezések használata a SELECT záradékban
- A „semmit" jelképező érték: a Null
- Példák
- Összefoglalás
- Önálló feladatok

A 4. fejezetben megtanultuk, hogyan használhatjuk a SELECT utasítást arra, hogy információkat szerezzünk egy tábla egy vagy több oszlopából. Ez a módszer jól használható, amíg olyan egyszerű kérdéseket intézünk az adatbázishoz, amelyek révén alapvető adatokhoz szeretnénk hozzájutni, amikor azonban összetettebb kérdésekkel kezdünk foglalkozni, elkerülhetetlenné válik az SQL-szókincsünk gazdagítása. Ebben a fejezetben megismerkedünk a *kifejezések* fogalmával, amelyek segítségével lehetőségünk nyílik az adataink módosítására és új adatoszlopok előállítására a meglévő adatokból. Ez után arról ejtünk szót, hogy az oszlopokban tárolt adatok *típusa* miként befolyásolja a kifejezéseket és lekérdezéseket, majd rövid kitérőt teszünk, hogy megismerkedjünk a CAST függvény használatával, amelynek a segítségével a kifejezéseinkben megváltoztathatjuk az adatok típusát. Ez után megtanulunk állandókat (literális értékeket) létrehozni, amelyeket kreatívan használhatunk a lekérdezésekben. Elsajátítjuk, hogyan határozhatjuk meg nagy pontossággal a lekérdezett információk körét, kifejezések segítségével kezelve az adatokat. Végül megismerkedünk a különleges Null értékkel, és megtanuljuk hogyan befolyásolja a Null a munkánkat.

Mik azok a kifejezések?

Ha többre van szükségünk egyszerű oszlopoknál, kifejezéseket kell írnunk. A *kifejezés* nem más, mint valamilyen művelet számokkal, karakterláncokkal, dátumokkal és időpontokkal. Egyaránt használhatunk bennük adatokat a tábláink meghatározott oszlopaiból, állandó (literális) értékeket, vagy a kettő kombinációit. (A literális értékek létrehozását a fejezet későbbi részében mutatjuk be.) Miután az adatbázis elvégezte a kifejezésben meghatározott műveleteket, egy értéket ad vissza az SQL-utasításnak, amelyet az feldolgoz. A kifejezések a lekérdezett információk körének szűkítésére és bővítésére egyaránt jól használhatók, és különösen hasznosnak bizonyulhatnak, ha „mi lenne, ha?" típusú kérdésre keressük a választ. Lássunk néhány példát a kifejezések segítségével megválaszolható kérdésekre:

„Összesen mennyi az értéke az egyes termékekből raktáron levő daraboknak?"
„Adj egy címlistát az alkalmazottainkról, amelyben a vezetéknév szerepel elöl!"
„Mutasd meg az egyes órák kezdő és záró időpontját, valamint az óra időtartamát!"
„Mutasd meg a tekejátékosok tiszta pontszáma és büntetőpontszámok által terhelt pontszáma közötti különbséget!"
„Mi a becsült óradíja az egyes rendezvényeknek?"
„Mi lenne, ha megemelnénk a termékeink árát 5 százalékkal?"

Ahogy végighaladunk ezen a fejezeten, fokozatosan megtanuljuk, hogy a kifejezések használata nagyon értékes része az SQL-ismereteinknek. Kifejezésekkel szétdarabolhatjuk és újraépíthetjük a nyers adatokat, hogy érthetőbb és használhatóbb eredményt kapjunk a lekérdezéseinkben, és nagy hasznukat vesszük majd a 6. fejezetben is, amikor adatok szűrésére és több tábla összekapcsolására használjuk őket.

Milyen típusú adatokat szeretnénk megjeleníteni?

A kifejezésekben használt adatok típusa befolyásolja a kifejezés visszatérési értékét, ezért kezdetnek meg kell ismerkednünk az SQL szabványos adattípusaival. Adatbázisunk minden oszlopához egy *adattípus* rendelődik, amely meghatározza, hogy milyen értékeket tárolhat az adott oszlop, valamint azt is korlátozza, hogy milyen műveleteket végezhetünk az oszlop adataival. Ismernünk kell az alapvető adattípusokat, hogy hozzákezdhessünk a literális értékekkel való munkához vagy oszlopok és literális értékek együttes alkalmazásához a kifejezésekben, hogy ezeket hibátlanul létrehozhassuk, és a megfelelő értékekkel térjenek vissza.

Az SQL-szabvány hét általános kategóriát jelöl ki az adatok típusainak: `CHARACTER` (karakteres vagy szöveges), `NATIONAL CHARACTER` (nemzetközi karakteres/szöveges), `BINARY` (bináris), `NUMERIC` (szám), `BOOLEAN` (logikai), `DATETIME` (dátum/idő) és `INTERVAL` (időköz vagy tartomány). Mindegyik kategória egy vagy több egyedileg elnevezett adattípust tartalmaz, amelyekről az alábbiakban közlünk egy tömör összefoglaló listát. (Az alábbi listában a szám kategóriát két alcsoportra bontottuk, pontos szám és közelítő szám néven.)

CHARACTER	A karakteres adattípus rögzített vagy változó hosszúságú karakterláncot tárolhat, amely egy vagy több nyomtatható karaktert tartalmaz. Az elfogadott karakterek általában a szabványos amerikai információátviteli kódolású (ASCII) vagy a bővített bináris kódolású (EBCDIC) karakterkészleteken alapulnak. A rögzített hosszúságú karakteres adattípus CHARACTER vagy CHAR néven, a változó hosszúságú karakteres típusok CHARACTER VARYING, CHAR VARYING vagy VARCHAR néven ismeretesek. Megadhatjuk a tárolandó adatok hosszát, de a legnagyobb tárolható hosszúság az adott adatbázis-kezelő rendszertől függ (Ez a szabály a nemzetközi karakteres adattípusokra is vonatkozik.) Ha a tárolandó karakterlánc hossza túllépi a rendszer által meghatározott legnagyobb értéket (ez általában 255 vagy 1024 karakter), CHARACTER LARGE OBJECT, CHAR LARGE OBJECT vagy CLOB adattípust kell alkalmaznunk. Sok rendszerben a CLOB adattípus TEXT vagy MEMO néven érhető el.
NATIONAL CHARACTER	A nemzetközi karakteres adattípus lényegében megegyezik az előzőekben megismert karakteres adattípussal, azzal a kivétellel, hogy a karakterei az ISO által meghatározott idegen nyelvű karakterkészletekből származnak. A rögzített hosszúságú nemzetközi karakteres típus neve NATIONAL CHARACTER, NATIONAL CHAR vagy NCHAR, míg a változó hosszúságú nemzetközi karakteres típusé NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING vagy NCHAR VARYING. Ha a tárolandó karakterlánc hossza túllépi a rendszer által meghatározott legnagyobb értéket (ez általában 255 vagy 1024 karakter), NATIONAL CHARACTER LARGE OBJECT, NCHAR LARGE OBJECT vagy NCLOB adattípust kell alkalmaznunk. Sok rendszerben az NCLOB adattípus NTEXT néven érhető el.
BINARY	A bináris adatok (képek, zenék, videók), illetve az olyan összetett beágyazott dokumentumok, mint a szövegszerkesztők és táblázatkezelők fájljainak tárolására a BINARY LARGE OBJECT (vagy BLOB) adattípus szolgál. Sok rendszerben ez az adattípus BINARY, BIT vagy BIT VARYING néven érhető el.
Pontos szám	Ez az adattípus egész, illetve tizedesjegyeket tartalmazó számokat tárol. A pontosságot (a szám összes jegye, az egészt és a tizedesjegyeket is beleértve) és a méretet (a tizedesjegyek száma a tizedesvesszőtől jobbra) a pontos számtípus esetében a felhasználó is beállíthatja, de be kell tartania az adott adatbázis-kezelő rendszer korlátozásait. A NUMERIC, DECIMAL, DEC, SMALLINT, INTEGER, INT és BIGINT elnevezésű típusok

mind ezt az adattípust jelölik. Egy dolgot azonban szem előtt kell tartanunk: az SQL-szabvány szerint – és a legtöbb adatbázis-kezelő rendszerben – a BIGINT típus nagyobb értéktartományt vehet fel, mint az INTEGER, az INTEGER pedig nagyobb értéktartományt képvisel, mint a SMALLINT. Az egyes típusokhoz tartozó értéktartományok tekintetében az adatbázis-kezelőnk dokumentációjából tájékozódhatunk. Néhány rendszer egy TINYINT típust is a rendelkezésünkre bocsát, amelyhez a SMALLINT-nél is kisebb értéktartomány tartozik.

Közelítő szám

Ez az adattípus tizedestörtek és exponenciális számok tárolására szolgál. A FLOAT, REAL és DOUBLE PRECISION néven előforduló típusok tartoznak ebbe a körbe. A közelítő számtípusoknak nincs meghatározott pontosságuk és méretük, de az SQL-szabvány csak a FLOAT típusnál engedélyezi, hogy a pontosságot a felhasználó adja meg. Az ilyen típusokhoz tartozó méretet mindig a konkrét adatbázis-kezelő rendszer korlátozza. Az SQL-szabvány és a legtöbb adatbázis-kezelő a DOUBLE PRECISION típus értéktartományát nagyobbként határozza meg a REAL és a FLOAT adattípusok értéktartományánál. Ennek tekintetében is ellenőrizzük a rendszerünk leírását.

BOOLEAN

Ez az adattípus igaz (true) és hamis (false) értékek tárolására szolgál, és jellemzően egyetlen bitet használ fel. Néhány rendszer a BIT, INT vagy TINYINT nevet használja a logikai adattípus neveként.

DATETIME

Ebben az adattípusban dátumok, időpontok, valamint ezek kombinációi tárolódnak. Az SQL-szabvány a dátumokra az év-hó-nap jelölésmódot, míg az időpontokra a 24 órás időpont-megadást írja elő. A legtöbb adatbázis-kezelő rendszerben a dátumok megadhatók az angolszász országokban elterjedtebb hó/nap/év vagy nap/hó/év formátumban, az időpontok pedig a 12 órás de./du. formátumban is. A továbbiakban a könyvben az SQL-szabvány által előírt jelölésmódot követjük. Három típus tartozik ebbe a kategóriába: a DATE, a TIME és a TIMESTAMP. A TIMESTAMP (időbélyeg) adattípus dátum és időpont együttes tárolására alkalmas. A típusok elnevezései és használatuk módja eltérő lehet az egyes adatbázis-kezelő rendszerek esetében. Vannak rendszerek, amelyek a DATE típusban egyaránt tárolnak dátumot és időt is, míg mások erre a TIMESTAMP vagy a DATETIME adattípust használják. Ezek tekintetében is az adatbázis-kezelő rendszerünk dokumentációjára támaszkodhatunk.

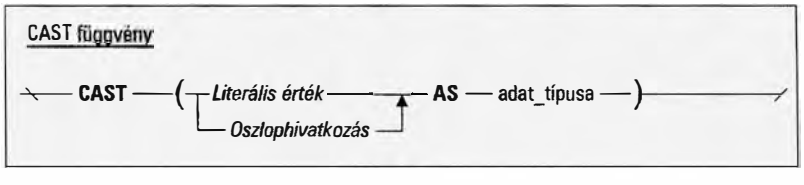
INTERVAL	Ez az adattípus két időpont (DATETIME) különbségének tárolására alkalmas, amely év, hónap; év/hó; nap, idő; vagy nap/idő formájában jelenhet meg. A főbb adatbázis-kezelők közül nem mindegyik támogatja az INTERVAL adattípust, ezért ez esetben is javasoljuk a rendszer leírásának tanulmányozását.

Sok adatbázis-kezelő rendszer biztosít további adattípusokat is, amelyek *bővített adattípus* néven ismeretesek az SQL-szabványban. (Néhányat felsoroltunk közülük az előző adattípus-csoportokat bemutató felsorolásban.) Ilyen bővített adattípus például a MONEY/CURRENCY és a SERIAL/ROWID (ez utóbbi egyedi sorazonosítókat tárol).

Mivel mi elsősorban az SQL *adatkezelő* részével foglalkozunk, csak az adatbázis-kezelőnk által biztosított adattípusok egy részére lesz szükségünk. A típusok ismerete segít a kifejezések helyes megírásában és megfelelő végrehajtásában, ezért érdemes időt szánnunk az adatbázis-kezelőnk típusainak megismerésére.

Adattípusok megváltoztatása: a CAST függvény

Amikor kifejezéseket szerkesztünk, ügyelnünk kell rá, hogy a bennük használt oszlopok és literálisok összeegyeztethetők legyenek a rajtuk végzett műveletekkel – például nem lenne szerencsés egy szöveget összeadnunk egy számmal. Ám ha egy szöveges (karakteres) oszlop vagy állandó egy számot tartalmaz, a CAST függvény segítségével valódi számmá alakíthatjuk azt, mielőtt elvégezzük az összeadást. A majdnem minden adatbázis-kezelőben támogatott CAST függvény utasításformáját az 5.1. ábrán láthatjuk.



5.1. ábra

A CAST függvény szintaxisdiagramja

A CAST függvény a neki átadott literális értéket vagy oszloptartalmat a meghatározott adattípusúvá alakítja. Segítségével biztosíthatjuk, hogy a kifejezéseinkben szereplő adatok típusa egymással *összeegyeztethető* legyen, vagyis minden oszlop vagy literális érték egyaránt szöveget, számot vagy dátum-idő értékeket tartalmazzon. (Mint minden szabálynál, itt is vannak kivételek, de ezekkel később foglalkozunk.) A kifejezésben szereplő minden értéknek összeegyeztethetőnek kell lennie a rajta végzett művelettel, hogy a kifejezés helyesen működjön; ellenkező esetben az adatbázis-kezelő hibát jelez.

Megjegyzés *A legtöbb kereskedelmi forgalomban kapható adatbázis-kezelő rendszer támogatja a CAST függvényt, de nem mindegyik. Ezekben a rendszerekben mást függvényekkel érhetjük el ugyanezt az eredményt. Ezek megismeréséhez tanulmányozzuk át a rendszerünk dokumentációját.*

Az adatok átalakítása egyik típusról a másikra viszonylag egyszerű, de az alábbi megszorításokat szem előtt kell tartanunk.

- Hívjuk ezt a szabályt a „ne rakjunk egy 10 kilós zsákot egy öt kilós dobozba" szabálynak. Amint korábban olvashattuk, a karakteres adattípusok esetében lehetőségünk van beállítani a tárolt adatok legnagyobb hosszát. Amikor megpróbálunk átalakítani egy szöveges mezőt (például `VARCHAR`-t) egy más típusú szöveges mezővé (például `CHARACTER`-ré), és az eredeti adat hossza nagyobb, mint az átalakított adat számára kijelölt adattípus lehetséges legnagyobb mérete, az adatbázis-kezelő csonkolja az eredeti szöveget. Ilyen esetben a rendszer figyelmeztet a csonkolás tényére.
- A „ne próbáljunk négyszögletes pecket dugni kerek lyukba" szabály: bármilyen karakteres értéket vagy oszlopot átalakíthatunk más típusúvá, de a forrásadatnak a céltípusra alakíthatónak kell lennie. Például egy szöveges mezőben található, öt karakterből álló irányítószámot átalakíthatunk szám típusúvá, de a kanadai irányítószámok esetében, amelyek betűket is tartalmaznak, már problémákba fogunk ütközni. Az adatbázis-kezelők figyelmen kívül hagyják a szövegek elején és végén lévő szóközöket a karakteres típusok számmá vagy dátummá alakításakor. A legtöbb adatbázis-kezelő a szöveges adatként megadott dátumok és időpontok sokféle fomáját képes felismerni. Ezzel kapcsolatban rendszerünk dokumentációja ad további tájékoztatást.
- A „10 kilós zsák" szabály második változata: amikor egy szám típusú oszlop értékét egy másik számtípussá alakítjuk, az átalakítandó értéknek illeszkednie kell a számára megadott új adattípushoz. Például, ha egy 32 767-nél nagyobb `REAL` típusú értéket szeretnénk `SMALLINT`-té alakítani, valószínűleg hibaüzenetet kapunk, ha pedig tizedestörtet próbálunk `INTEGER` vagy `SMALLINT` típusúvá alakítani, az adatbázis-kezelő rendszer a tizedesvesszőtől jobbra lévő számjegyeket levágja vagy kerekíti.
- A fentiek ellenére lehet „négyszögletes pecket kerek lyukba dugni" – bizonyos korlátozásokkal. Szám típusú oszlopok karakteres típussá alakítása három fajta eredménnyel zárulhat:
  1. Az átalakítás sikeresen megtörténik.
  2. A rendszer szóközökkel tölti fel a maradék helyet, ha az adat rövidebb, mint a karakteres típusú oszlop számára beállított hossz.
  3. Az adatbázis-kezelő hibaüzenetet küld, ha a karakteres típusúvá alakított adat hossza nagyobb, mint a karakteres oszlop számára beállított hossz.

Megjegyzés *Az SQL-szabvány ugyan ezeket a megszorításokat adja meg, ettől azonban a saját adatbázis-kezelőnk eltérhet az átalakítások elvégzése során. Néhány adatbázis-kezelő rendszer automatikusan elvégzi a típusok átalakítását, anélkül, hogy alkalmaztuk volna a CAST függvényt. Például hozzáfűzhetünk egy szöveghez egy számot, vagy összeadhatunk egy számot tartalmazó szöveget egy másik számmal anélkül, hogy külön műveletben elvégeznénk a típusok átalakítását. A részletek tekintetében tanulmányozzuk át az adatbázis-kezelőnk leírását.*

Fontos tudnunk, hogy a fenti felsorolás nem tartalmazza az SQL-szabványban leírt összes korlátozást. Csak azokat soroltuk fel, amelyek a könyvben használt adattípusokkal összefüggnek. Ha részletesebben szeretnénk olvasni az adattípusokról és a típusok átalakításakor felmerülő problémákról, a D függelékben felsorolt könyvekben találhatunk részletesebb információkat.

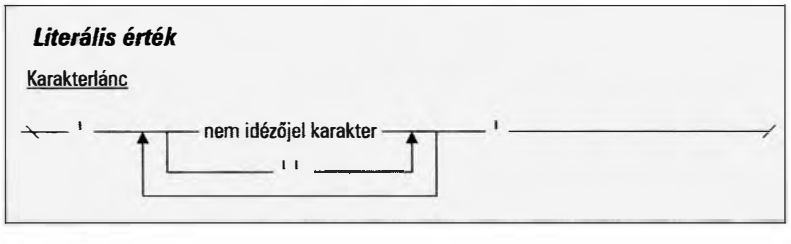
Figyeljük a CAST függvény alkalmazását a könyv további részében: akkor fogjuk használni, ha biztosak szeretnénk lenni abban, hogy bizonyos adatok biztosan egy adott típusba tartoznak.

Konkrét értékek meghatározása

Az SQL-szabvány lehetőséget ad a SELECT utasítás által szolgáltatott információk rugalmas kibővítésére, lehetővé téve állandó értékek, például karakterláncok, számok, dátumok és időpontok használatát, illetve ezek változatos kombinációit a SELECT utasításban használt minden érvényes kifejezésben. Az SQL-szabvány az ilyen módon megadott értékeket *literális értékeknek* nevezi, és leírja a meghatározásuk szabályait is.

Karakterlánc-literálok

A *karakterlánc-literál aposztrófok* (egyszeres idézőjelek) közé zárt karakterek sorozata. Bár valószínűleg hozzászoktunk, hogy általában (kettős) idézőjelek közé zárjuk a karakterláncokat, itt úgy szeretnénk bemutatni ezeket a megoldásokat, ahogy az SQL-szabvány megadja azokat. A karakterlánc-literálok diagramját az 5.2. ábrán láthatjuk.



5.2. ábra

A karakterlánc-literálok szintaxisdiagramja

Néhány példa ilyen karakterlánc-literálokra:

```
'Ez egy példamondat a karakterlánc-literálok szemléltetésére.'
'Here"s yet another!'
'B-28'
'Seattle'
```

Vegyük észre, hogy ami kettős idézőjelnek tűnik a diagramon és a második példamondatban, az valójában két egymást követő aposztróf, amelyek között nem hagytunk szóközt, ezért nagyon hasonlóan néznek ki, mint egy kétszeres idézőjel. A szabvány szerint az adatbázis-kezelő rendszerek így tudják megkülönböztetni egymástól a karakterlánc kezdetét és végét jelző aposztrófot egy a mondat közepén lévő aposztróftól. Az alábbi két sor bemutatja, hogyan adhatunk meg aposztrófot egy karakterlánc közepén:

```
SQL                     'The Vendor"s name is: '
Megjelenített eredmény  The Vendor's name is:
```

Amint korábban említettük, a karakterlánc-literálok segítségével kibővíthetjük a SELECT által adott információkat. A lekérdezések által visszaadott információ általában könnyen áttekinthető, de ha a helyzet szükségessé teszi, még világosabbá tehető. Például ha végrehajtjuk az alábbi SQL-utasítást, az eredményhalmaz csak a beszállítók weboldalának címét és a beszállítók nevét fogja tartalmazni:

```
SQL                     SELECT VendWebPage, VendName
                        FROM Vendors
```

Ha azt szeretnénk, hogy a lekérdezés több információt adjon, megadhatunk egy karakterláncot, amelynek a segítségével kiegészítő, magyarázó szöveget adhatunk a SELECT záradékhoz. Ezt az eljárást körültekintően használjuk, mivel a karakterlánc-literál az eredményhalmaz minden sorában szerepelni fog. Az előző példa lekérdezését egy karakterlánc-literállal kibővítve az alábbihoz hasonló lekérdezést kapunk:

```
SQL                     SELECT VendWebPage, 'is the Web site for',
                           VendName
                        FROM Vendors
```

Az ezáltal a SELECT utasítás által előállított eredmény egy sora ehhez hasonlóan néz ki:

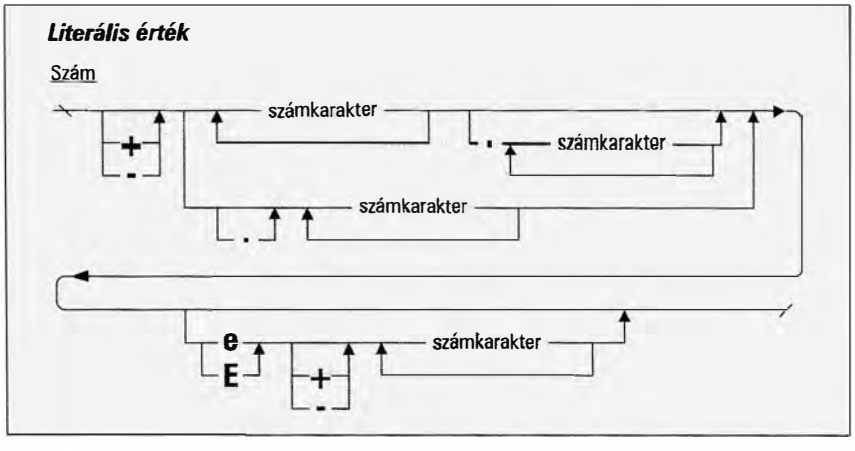
www.viescas.com	is the Web site for	Viescas Consulting, Inc.

Ez egyértelműbbé teszi a megjelenített információkat, hiszen pontosan tisztázza az internetcím hovatartozását. Természetesen ez csak egy nagyon egyszerű példa annak szemléltetésére, hogy miként használhatjuk a karakterlánc-literálokat. A későbbiekben a kifejezésekben való használatukkal is megismerkedünk.

Megjegyzés *A fenti megoldásnak különösen akkor vehetjük hasznát, ha régről örökölt adatbázisokkal kell dolgoznunk, amelyek megfejthetetlen oszlopneveket tartalmaznak. Ha az adatbázisaink létrehozása során azonban követjük a 2. fejezet javaslatait, erre a módszerre nem túl gyakran lesz szükségünk.*

Számliterálok

A számliterál a SELECT utasításban használható literálok másik típusa. Ahogy a neve is elárulja, előjelet, számot, tizedespontot, exponenciális jelet és exponenciális számot tartalmazhat. A számliterálok diagramját az 5.3. ábrán láthatjuk.



5.3. ábra

A számliterálok szintaxisdiagramja

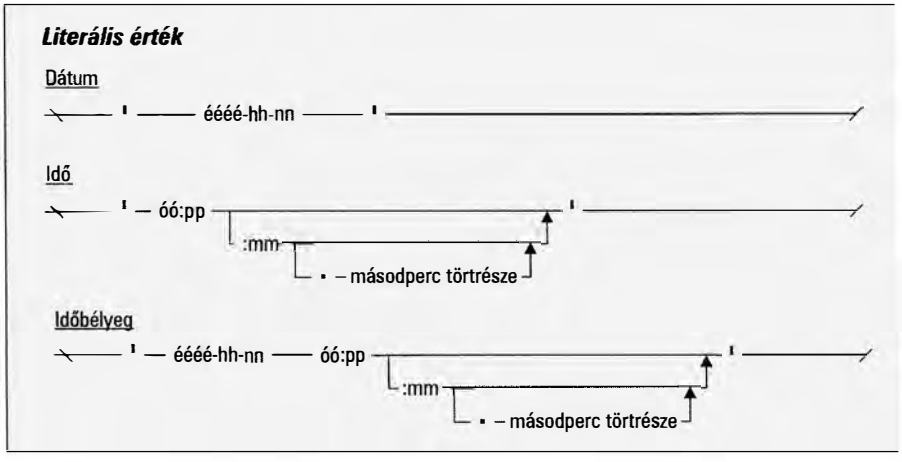
Néhány példa számliterálokra:

```
427
-11.253
.554
0.3E-3
```

A számliterálok kifejezésekben alkalmazva a leghasznosabbak a számunkra (például szorzást vagy összeadást tesznek lehetővé egy meghatározott értékkel), ezért részletesebb megismerésüket a fejezet későbbi részeire halasztjuk.

Dátum-idő literálok

A SELECT utasításokon belül a *dátumliterálok*, *időliterálok*, illetve *időbélyeg-literálok* segítségével különböző dátum- és időértékeket adhatunk meg. Az SQL-szabvány ezekre *dátum-idő literál* (*datetime literal*) néven hivatkozik. Amint az 5.4. ábrán megfigyelhetjük, ezeknek a literáloknak a megadása igen egyszerű.



5.4. ábra

A dátum-idő literálok szintaxisdiagramja

Dátum-idő literálokat, illetve azok különbségét kifejező literálokat használva az alábbiakat kell észben tartanunk:

Dátum — A dátumliterálok év-hó-nap alakban adhatók meg. A könyvben is végig ezt a formát követjük majd, de sok adatbázis-kezelő megengedi az Egyesült Államokban elterjedt hó/nap/év formátumú, valamint a világ számos országában használt nap/hó/év forma használatát is. Az SQL-szabvány megköveteli a DATE kulcsszó használatát a literál előtt, de majdnem minden, a piacon elterjedt szoftver megengedi, hogy a literális értéket egyszerűen határoló karakterek között (ezek általában aposztrófok) adjuk meg. Egy kivételt találtunk: a MySQL-t, amely a dátumliterálok megadását (kettős) idézőjelek között követeli meg, majd az így megadott karakterlánc a CAST függvény segítségével DATE adattípusúvá alakítandó, mielőtt dátumműveletekhez használhatnánk.

Idő — Időpont megadásához a 24 órás formátum alkalmazandó. A du. 07:00 időpontot például 19:00-ként adhatjuk meg. Az SQL-szabvány megköveteli a TIME kulcsszó használatát is a literál előtt, de majdnem minden kereskedelmi szoftverben elegendő csupán a literális értéket megadni határoló (általában aposztróf) karakterek között. Egy kivételt találtunk: a MySQL-t, amelyben az időliterálokat (kettős) idézőjelek között kell megadnunk, majd a CAST függvénnyel a karakterláncot TIME típusúvá kell alakítanunk, mielőtt időműveleteket végezhetnénk vele.

Időbélyeg — Az időbélyeg-literál nem más, mint egy dátum és egy időpont együttes megadása egy szóközzel elválasztva. A dátum és az időpont megadásának formai szabályai megegyeznek a dátum- és időliterálok megadásánál leír-

takkal. Az SQL-szabvány megköveteli a TIMESTAMP kulcsszó alkalmazását a literál előtt, de majdnem minden kereskedelmi szoftver, amely támogatja a TIMESTAMP adattípust, lehetővé teszi, hogy a literál értékét egyszerűen határoló karakterek (általában aposztrófok) között adjuk meg.

Megjegyzés *Bizonyos rendszerekben az időköz-literálok (időtartomány-literálok) használata is támogatott a dátum- és időliterálokkal műveletet végző kifejezésekben, de ezzel a literáltípussal a könyvben bővebben nem foglalkozunk. További részleteket az adatbázis-kezelőnk dokumentációjából tudhatunk meg.*

A DATE, TIME, TIMESTAMP és INTERVAL adattípusoknak az SQL-szabvány által meghatározott formátumát az „A" függelékben található diagramok mutatják be.

Néhány példa dátum- és időliterálokra:

```
'2007-05-16'
'2016-11-22'
'21:00'
'03:30:25'
'2008-09-29 14:25:00'
```

Ha MySQL adatbázis-kezelőt használunk, ne felejtsük el, hogy minden karakterlánc formájában megadott dátumot és időpontot kifejezetten át kell alakítanunk a CAST függvénnyel:

```
CAST('2016-11-22' AS DATE)
CAST('03:30:25' AS TIME)
CAST('2008-09-29 14:25:00' AS DATETIME)
```

Amint az előzőekben már megjegyeztük, ha követni szeretnénk az SQL-szabványt, mindegyik literál előtt alkalmaznunk kell a megfelelő kulcsszavakat. A DATE és TIME kulcsszavak az SQL-szabvány szerint a dátum- és időliterálok szükséges alkotóelemei, az adatbázis-kezelő rendszerek azonban ritkán támogatják ezeket a kulcsszavakat, és csak a literál karakterlánc részének megadását igénylik, ezért a könyv példáiban tartózkodni fogunk a kulcsszavak használatától, és csak az aposztrófokat használjuk a dátum- és időliterálok megadásakor. Az így megadott dátumok és időpontok kifejezésekben való használatával a fejezet későbbi részében ismerkedünk meg. A dátum- és időliteráloknak az SQL-szabvány szerinti formátumát az „A" függelék diagramjai mutatják be részletesen.

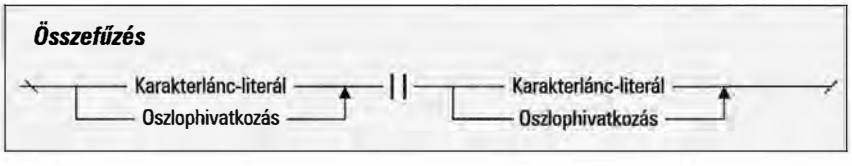
A kifejezések típusai

Az SQL-utasítások írása során a kifejezéseknek általában az alábbi három típusát használjuk.

Összefűzés	Két vagy több, karaktereket tartalmazó oszlop vagy literál egyesítése egyetlen karakterlánccá.
Matematikai kifejezés	Összeadás, kivonás, szorzás vagy osztás, számadatokat tartalmazó oszlopokkal vagy literálokkal.
Dátum- és időműveletek	Dátumok és időpontok összeadása vagy kivonása.

Összefűzés

Az SQL-szabvány két függőleges vonalat határoz meg az összefűzés műveleti jeleként. Két karakteres elem összefűzése úgy történik, hogy az egyik elemet az összefűző műveleti jel egyik oldalán szerepeltetjük, a másikat pedig a másikon. Eredményül egyetlen karakterláncot kapunk, amely a két elem összefűzéseként áll elő. Az összefűző kifejezések szintaxisdiagramját az 5.5. ábrán láthatjuk.



5.5. ábra

Az összefűző kifejezés szintaxisdiagramja

Megjegyzés *A jelentősebb adatbázis-kezelő rendszerek közül csak az IBM DB2, az Informix, valamint az Oracle támogatja az SQL-szabványban meghatározott összefűző műveleti jelet. A Microsoft Office Access az & és + jeleket használja, a Microsoft SQL Server és Ingres csak a + jelet támogatja, a MySQL-ben pedig a CONCAT függvényt kell használnunk. A könyv példáiban a szabványos || műveleti jelet használjuk, a CD-n található mintaadatbázisokban pedig minden adatbázisrendszerhez (Microsoft Access, Microsoft SQL Server és MySQL) az általa támogatott megfelelő műveleti jelet.*

Az alábbiakban láthatjuk az összefűzés műveletének elvét:

```
Kifejezés   ElsőElem || MásodikElem
Eredmény    ElsőElemMásodikElem
```

Kezdjük a világ legegyszerűbb példájával: fűzzünk össze két karakterlánc-literált, például egy kereszt- és egy vezetéknevet:

```
Kifejezés   'Mike' || 'Hernandez'
Eredmény    MikeHernandez
```

Két dolgot figyelhetünk meg ebben a példában. Először is, mindkét nevet aposztrófok közé kell írnunk, mivel két karakterlánc-literálról van szó. Másodszor, az eredményben a kereszt- és a vezetéknév közvetlenül egymás után szerepel. Bár az összefűzés művelete sikeres volt, valószínűleg ebben az esetben nem ezt szeretnénk elérni. Megoldásként helyezzünk egy szóközt a két név közé, mégpedig egy újabb karakterliterál hozzáadásával, amely egyetlen szóközt tartalmaz:

```
Kifejezés   'Mike' || ' ' || 'Hernandez'
Eredmény    Mike Hernandez
```

A fenti példa azt mutatja, hogy több összefűző műveleti jel alkalmazásával több karakterláncot is összefűzhetünk. Az összefűzendő értékek számának tekintetében semmilyen korlátozás nincs, viszont az összefűzés eredményeképpen kapott karakterlánc hossza korlátozott. Az eredményül kapott karakterlánc hossza általában nem lehet nagyobb, mint a változó hosszúságú karakteres adattípus által megengedett legnagyobb méret. Az adatbázis-kezelő rendszerek ezt a problémát különbözőképpen kezelik, ezért a további részletekért tanulmányozzuk át a rendszerünk leírását.

Két vagy több karakterlánc összefűzése remek lehetőség, de ugyanilyen módon össze tudunk fűzni kettő vagy több karakteres típusú oszlopot is. Például ha van két oszlopunk CompanyName (Cégnév) és City (Város) néven, írhatunk egy kifejezést, amelyben az oszlopneveket szerepeltetve lehetőségünk nyílik a tartalmuk összefűzésére. Az alábbi példában a két oszlop tartalmát egy karakterlánccal fűzzük össze:

```
Kifejezés   CompanyName || ' is based in ' || City
Eredmény    DataTex Consulting Group is based in Seattle
```

A CompanyName és City oszlopneveket nem kell aposztrófok közé tennünk, mivel ezek oszlophivatkozások. (Emlékszünk még az előző fejezetben megismert oszlophivatkozásokra?) Ilyen oszlophivatkozásokat bármilyen kifejezésben használhatunk, amint azt a további példákban majd láthatjuk is.

Biztosan észrevettük, hogy a példákban mindig karakterláncokat fűztünk össze. Ezek után valószínűleg kíváncsiak vagyunk, hogy kell-e tennünk valami különlegeset, ha számokat vagy dátumokat szeretnénk összefűzni. Nos, a legtöbb adatbázis-kezelő rendszer esetében találkozunk némi különbséggel. Ha az adatbázis-kezelő észreveszi, hogy karakteres típusú oszlopot vagy literált szeretnénk összefűzni számmal vagy dátumértékkel, automatikusan elvégzi a szám vagy dátum átalakítását, és az összefűzés gond nélkül megtörténik.

Viszont nem mindig bízhatjuk az adatbázis-kezelőre, hogy az átalakítást csendben, a háttérben elvégezze. Ha karakterlánc-literálokat vagy karakteres típusú oszlopok értékét szeretnénk dátumliterállal vagy szám, illetve dátum típusú oszlopok értékével összefűzni, a CAST függvényt kell használnunk az értékek karakterlánccá alakításához. Az alábbi példában a CAST használatával átalakítjuk a DateEntered nevű, dátum típusú oszlop értékét:

```
Kifejezés   EntStageName || ' was signed with our agency on '
            || CAST(DateEntered as CHARACTER(10))
Eredmény    Modern Dance was signed with our agency on 1995-05-16
```

Megjegyzés *A fenti példában kifejezetten megadtuk a CHARACTER adattípus hosszát, mivel az SQL-szabvány abban az esetben, ha nem adunk meg hosszértéket, alapértelmezésként az 1 értéket használja. Azt tapasztaltuk, hogy a főbb adatbázis-kezelők nem ragaszkodnak ehhez a viselkedéshez, hanem olyan hosszú karakterláncot hoznak létre, amelybe belefér az átalakítás eredménye. A részleteket az adatbázis-kezelőnk dokumentációjában találjuk, de ha biztosra szeretnénk menni, mindig adjunk meg konkrét hosszértéket.*

Számliterálok vagy szám típusú oszlopok értékének karakterlánchoz való fűzésekor szintén a CAST függvényt kell használnunk. A következő példában a RetailPrice nevű, szám típusú oszlop értékét alakítjuk át a CAST-tal:

```
Kifejezés   ProductName || ' sells for ' || CAST(RetailPrice
            AS CHARACTER(8))
Eredmény    Trek 9000 Mountain Bike sells for 1200.00
```

Az összefűző kifejezés egyidejűleg is tartalmazhat karakterláncokat, dátum-idő értékeket és számértékeket. Az alábbi példában egyetlen kifejezésben mindhárom adattípust használjuk:

```
Kifejezés   'Order Number ' || CAST(OrderNumber AS CHARACTER(2))
            || ' was placed on ' ||
            CAST(OrderDate AS CHARACTER(10))
Eredmény    Order Number 1 was placed on 2007-09-01
```

Megjegyzés *Az SQL-szabvány függvények sokaságát írja le, amelyekkel információkat nyerhetünk ki az oszlopokból, vagy több sor értékével végezhetünk számításokat. Ezek közül néhánnyal részletesebben foglalkozunk a 12. fejezetben. A legtöbb kereskedelmi adatbázis-kezelő rendszer további függvényeket is nyújt, amelyekkel karakterláncok részeivel végezhetünk műveleteket, vagy dátum-, idő- és pénznemértékeket formázhatunk. A részletekért tanulmányozzuk át a rendszerünk leírását.*

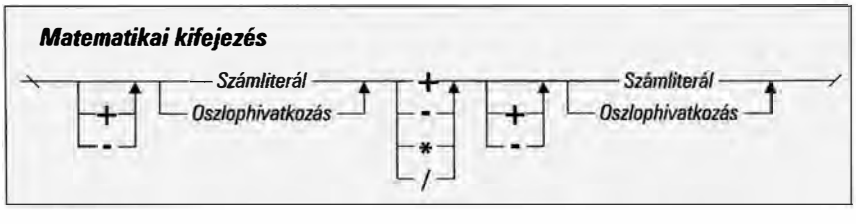
Most, hogy megismerkedtünk a különböző forrásból származó adatok egyetlen karakterlánccá való összefűzésével, ismerjük meg a kifejezések másik csoportját is: azokat a kifejezéseket, amelyeket szám típusú adatokból építhetünk fel.

Matematikai kifejezések

Az SQL-szabvány az összeadás, kivonás, szorzás és osztás műveletét értelmezi a szám típusú adatok között. Igen, ezek valóban nagyon korlátozott lehetőségek. Szerencsére a legtöbb RDBMS szoftver műveletek sokkal szélesebb választékát – például maradékképzés, gyökvonás, exponenciális és abszolút érték – biztosítja. Tudományos, trigonometriai, statisztikai és matematikai függvényeket is rengeteget ismernek. A könyvben azonban csak az SQL-szabványban leírt műveletekre összpontosítunk.

A műveletek *végrehajtási sorrendje* (kiértékelési sorrendje) fontos szempont a matematikai kifejezések írásakor. Az SQL-szabvány a sorrend tekintetében nem tesz különbséget a szorzás és az osztás között, csak azt írja le, hogy ezeket az összeadások és kivonások előtt kell végrehajtani. Ez valamelyest ellentmond a műveletek iskolában tanult sorrendjének, ahol a szorzás az osztás előtt szerepel, az osztás az összeadás előtt, az összeadás pedig a kivonás előtt, de megfelel a modern programozási nyelvek által használt műveleti sorrendnek. A matematikai kifejezések kiértékelése balról jobbra történik. Ez – attól függően, hogy miként építettük fel a kifejezést – néha érdekes eredményt adhat, ezért javasoljuk, hogy ahol csak lehet, használjunk zárójeleket az összetett matematikai kifejezéseknél, hogy azok az elvárásainknak megfelelően értékelődjenek ki.

Ha még emlékszünk rá, hogyan írtunk egyenleteket matematikaórán az iskolában, akkor az SQL-kifejezések megírása sem fog gondot okozni. Lényegében egy előjeles vagy előjel nélküli számérték, egy matematikai műveleti jel és egy másik előjeles vagy előjel nélküli számérték alkot egy egyszerű kifejezést. A matematikai kifejezések felépítését az 5.6. ábra mutatja.



5.6. ábra

A matematikai kifejezések szintaxisdiagramja

Az alábbiakban néhány példát láthatunk matematikai kifejezésekre, amelyek számszerű literális értékeket, oszlophivatkozásokat és a kettő kombinációját tartalmazzák:

```
25 + 35
-12 * 22
RetailPrice * QuantityOnHand
TotalScore / GamesBowled
RetailPrice - 2.50
TotalScore / 12
```

Amint már említettük, az összetett matematikai kifejezések esetében tanácsos zárójeleket használnunk. Az alábbi egyszerű példa egy ilyen kifejezést mutat be:

Kifejezés (11 – 4) + (12 * 3)
Eredmény ***43***

Figyeljük meg, hogyan befolyásolja a zárójelek elhelyezése a kifejezés eredményét. A következő példában szereplő két kifejezés világosan megmutatja ennek jelentőségét. Mindkét kifejezést ugyanazok a számok és műveleti jelek alkotják, csak a zárójelezésükben térnek el egymástól, ennek következtében mégis teljesen különböző értékkel fognak visszatérni:

Kifejezés `(23 * 11) + 12`
Eredmény ***265***

Kifejezés `23 * (11 + 12)`
Eredmény ***529***

Könnyű belátni, hogy miért kell óvatosan bánnunk a zárójelekkel, ez azonban ne tántorítson el bennünket a használatuktól: nélkülözhetetlenek lesznek, amikor bonyolult kifejezésekkel akad dolgunk.

Zárójelek segítségével lehetőségünk van műveletek egymásba ágyazására is a kifejezéseken belül. Amikor egymásba ágyazott, zárójelezett műveleteket használunk, ezek kiértékelését az adatbázis-kezelő balról jobbra és „bentről kifelé" végzi. Az alábbi példában egy zárójelek segítségével egymásba ágyazott műveleteket tartalmazó kifejezést láthatunk:

Kifejezés `(12 * (3 + 4)) - (24 / (10 + (6 - 4)))`
Eredmény ***82***

A kifjezésben szereplő műveletek végrehajtása nem olyan bonyolult, mint amilyennek tűnik. Az adatbázis-kezelő a következő sorrendben értékeli ki a műveleteket:

1. (3 + 4) = 7
2. (12 * 7) = 84 *12-vel szorozzuk az első művelet eredményét*
3. (6 - 4) = 2

4. (10 + 2) = 12 *10-et hozzáadunk a harmadik művelet eredményéhez*
5. (24 / 12) = 2 *a 24-et elosztjuk a negyedik művelet eredményével*
6. 84 - 2 = 82 *a 84-ből kivonjuk a második művelet eredményét*

Látható, hogy a rendszer balról jobbra halad, de a zárójelek közé zárt kifejezéseket bentről kifelé értékeli ki. Gyakorlatilag a (12 * (3 + 4)) és a (24 / (10 + (6 - 4))) kifejezések ugyanazon a szinten vannak, ezért a rendszer a bal oldali kifejezést értékeli ki először, bentről kifelé haladva. Ez után tovább halad a második zárójelekkel határolt kifejezésre, és ezt is elkezdi bentről kifelé haladva kiértékelni. Végső műveletként a bal oldali kifejezés eredményéből kivonja a jobb oldali kifejezés eredményét. (Ha úgy érezzük, hogy megfájdult a fejünk, akkor jó, ha tudjuk, hogy nem vagyunk egyedül!)

Bár az előző példában számliterálokat használtunk, ugyanilyen egyszerűen használhatunk oszlophivatkozásokat vagy a kettő tetszőleges kombinációját is. A legfontosabb, amit meg kell jegyeznünk, hogy mindig tervezzük meg és nagy gondossággal állítsuk össze a matematikai kifejezésünket, hogy valóban az általunk várt eredményt adja vissza. Használjunk zárójeleket, hogy tisztán jelöljük a műveletek végrehajtásának sorrendjét – ha odafigyelünk erre, a várt eredményt fogjuk kapni.

Amikor matematikai kifejezésekkel dolgozunk, győződjünk meg róla, hogy a kifejezésben használt értékek egymással összeegyeztethetők. Ez különösen igaz az olyan kifejezésekre, amelyek oszlophivatkozásokat tartalmaznak. Erre a célra használhatjuk a CAST függvényt, pontosan úgy, ahogy azt az összefűző kifejezéseknél tettük. Tegyük fel például, hogy van egy TotalLength nevű oszlopunk, amely INTEGER típusú adatokat tárol, és a 345 értéket tartalmazza, valamint egy oszlopunk Distance néven, amelyben a REAL típusú 138,65 érték található. Ahhoz, hogy hozzáadjuk a Distance oszlop értékét a TotalLength oszlop értékéhez, vagy a Distance oszlop értékét kell INTEGER típusúvá alakítanunk a CAST függvény segítségével, vagy a TotalLength oszlop értékét REAL típusúvá, attól függően, hogy végeredményül INTEGER vagy REAL típusú adatot szeretnénk kapni. Ha azt feltételezzük, hogy csak az értékek egész részével szeretnénk törődni az összeadás során, akkor ezt az alábbi kifejezéssel tehetjük meg:

```
Kifejezés   TotalLength + CAST(Distance AS INTEGER)
Eredmény    483
```

Nem ezt az eredményt vártuk? Akkor valószínűleg arra számítottunk, hogy a 138,65 egész számmá alakítása kerekítéssel történik. Az SQL-szabvány állítása szerint az átalakítások során kerekítésnek kell történnie, ennek ellenére a CAST függvény az adatbázis-kezelő rendszertől függően ettől eltérően működhet. A legtöbb adatbázis-kezelő egyszerűen levágja a tizedesjegyeket az egésszé alakításkor. Tehát a mi rendszerünk is valószínűleg így tesz, ezért a példában a 345-höz 138-at ad hozzá, nem pedig a kerekített értéket, 139-et.

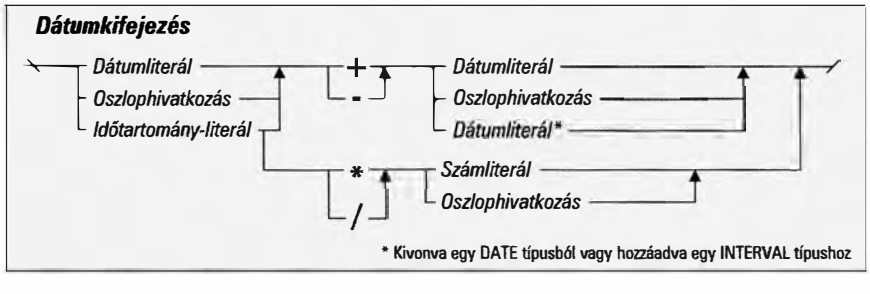
Ha elfelejtettük biztosítani az oszlopok értékének megfelelőségét egy kifejezésben, az adatbázis-kezelőnk hibaüzenetet küldhet, és ha így tesz, valószínűleg megszakítja a kifejezésen belül végzett további műveletek végrehajtását. A legtöbb RDBMS rendszer viszont az ehhez hasonló átalakításokat automatikusan kezeli, anélkül, hogy bármilyen figyelmeztetést jelenítene meg. Ilyenkor valószínűleg az történik, hogy az összes számértéket automatikusan a legösszetettebb adattípusra alakítja, mielőtt kiértékeli a kifejezést. Az előző példát alapul véve a relációs adatbázis-kezelőnk valószínűleg legszívesebben a TotalLength oszlopot alakítaná REAL típusúvá (ez az adattípus összetettebb a két adattípus közül), majd REAL típusú értéket használna minden INTEGER érték tárolására, mivel a REAL típusú oszlopokban ezek gond nélkül ábrázolhatók. Ez azonban valószínűleg nem az, amit mi el szeretnénk érni. Azok a relációs adatbázis-kezelők, amelyek nem végzik el automatikusan ezt az átalakítást, általában tudatják, hogy adattípus-eltérési probléma merült fel, ezért jó, ha tudjuk, mire van szükségünk, hogy kijavítsuk a kifejezést. Tehát ahogy azt most megtanultuk, matematikai kifejezéseket alkotni viszonylag egyszerű, ha előtte szánunk egy kis időt a tervezésre, és tudjuk, hogyan használhatjuk a CAST függvényt a céljaink elérésére. Most pedig vizsgáljuk meg, hogyan írhatunk olyan kifejezéseket, amelyekben dátumokat és időpontokat adunk össze, illetve vonunk ki egymásból.

Dátum- és időműveletek

Az SQL-szabvány a dátumok és időpontok esetében az összeadás és a kivonás műveletét értelmezi. Bizonyára meglepődünk, de sok relációs adatbázis-kezelő eltérően valósítja meg ezeket a műveleteket. Néhányukban ezeket egyszerű matematikai kifejezés formájában is megadhatjuk, míg másokban külön beépített függvények vannak erre a célra. Arról, hogy a mi esetünkben hogyan valósulnak meg ezek a műveletek, a rendszerünk dokumentációjából tájékozódhatunk. A könyvben a dátum- és időkifejezéseket általánosságban tárgyaljuk, mert az a célunk, hogy azok működéséről adjunk képet.

Dátumkifejezések

Az 5.7. ábra a dátumkifejezések szintaxisdiagramját ábrázolja, úgy, ahogy az az SQL-szabványban szerepel. Amint azt megfigyelhetjük, a kifejezések létrehozása nagyon egyszerű: vesszük az első értéket, majd hozzáadjuk vagy kivonjuk belőle a másodikat.



5.7. ábra

A dátumkifejezések szintaxisdiagramja

Az SQL-szabvány továbbá leírja az érvényes műveleteket és a hozzájuk tartozó eredményeket:

DATE plusz vagy mínusz INTERVAL eredményeként DATE típust kapunk.
DATE mínusz DATE eredményeként INTERVAL típust kapunk.
INTERVAL plusz DATE eredményeként DATE típust kapunk.
INTERVAL plusz vagy mínusz INTERVAL eredményeként INTERVAL típust kapunk.
INTERVAL szorozva vagy osztva NUMBER eredményeként INTERVAL típust kapunk.

Jegyezzük meg, hogy az SQL-szabvány értelmében csak DATE típusú értéket vonhatunk ki DATE típusúból, és csak DATE típusú értéket adhatunk INTERVAL típusúhoz.

Oszlophivatkozások használatakor ügyelnünk kell rá, hogy a fentieknek megfelelően DATE vagy INTERVAL típusú értéket tartalmazzanak. Ha az oszlop nem megfelelő típusú adatot tartalmaz, a CAST függvény segítségével átalakíthatjuk az összeadandó vagy kivonandó értékeket. A szabvány konkrétan meghatározza, hogy ezeket a műveleteket csak a fenti adattípusokkal végezhetjük el, de sok adatbázis-kezelő automatikusan átalakítja az oszlopok adatait. Ezért hogy megtudjuk, szükségünk lesz-e valamilyen átalakításra, tanulmányozzuk a rendszerünk dokumentációját.

Az INTERVAL adattípust csak néhány rendszer támogatja, viszont közel mindegyik lehetővé teszi egész szám kivonását vagy hozzáadását egy dátumértékhez. Ezt a műveletet napok hozzáadásaként, illetve kivonásaként foghatjuk fel. Segítségével olyan kérdésekre válaszolhatunk például, hogy *„Milyen dátumot írtunk kilenc nappal ezelőtt?"* vagy *„Mi volt az öt nappal ezelőtti dátum?"*. Megemlítendő még, hogy néhány rendszer esetében lehetőségünk van törtszámok hozzáadására, illetve kivonására is. Például 3,5-et egy időponthoz hozzáadva a Microsoft Access három napot és 12 órát ad az időponthoz.

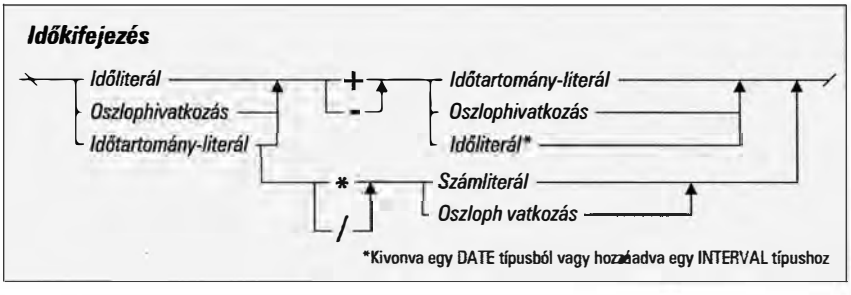
Amikor kivonunk egy dátumból egy másikat, tulajdonképpen a két dátum között eltelt időt kapjuk meg. Például ha egy szerződés aláírásának időpontját kivonjuk a jelenlegi dátumból, megkapjuk, hogy az alkalmazottunk mennyi ideje dolgozik a cégnél. A szabvány szerint ugyan csak időtartományt adhatnánk dátumhoz, sok adatbázis-kezelő rendszer (különösen azok, amelyek nem támogatják az INTERVAL adattípust) mégis megengedi, hogy számot vagy dátumot adjunk hozzá. Az ilyen jellegű számítások segítségével válaszolhatunk például egy olyan kérdésre, mint a *„Mikor lesz az alkalmazottunk következő értékelése?"*.

Könyvünkben egyszerű műveleteket mutatunk be dátumok használatával, és feltételezzük, hogy legalább egész számokban kifejezett napokat hozzá tudunk adni dátumértékekhez. Azt is feltételezzük, hogy egy dátumot kivonva egy másikból a két dátum között eltelt napok számát kapjuk. Ezeket az egyszerű elveket alkalmazva a későbbiekben gond nélkül megírhatjuk a legtöbb dátumkifejezést, amire szükségünk lesz. Álljon itt néhány példa dátumkifejezésekre, amelyeket alkalmazhatunk:

```
'2007-05-16' - 5
'2007-11-14' + 12
ReviewDate + 90
EstimateDate - DaysRequired
'2007-07-22' - '2007-06-13'
ShipDate - OrderDate
```

Időkifejezések

Kifejezéseket az előzőekhez hasonlóan időértékek felhasználásával is létrehozhatunk. Az 5.8. ábrán láthatjuk, hogy a dátum- és időkifejezések utasításformája nagyon hasonló; a dátumkifejezésekre érvényes szabályok és korlátozások érvényesek az időkifejezésekre is.



5.8. ábra

Az időkifejezések szintaxisdiagramja

Az SQL-szabvány továbbá leírja az érvényes műveleteket és a hozzájuk tartozó eredményeket:

TIME plusz vagy mínusz INTERVAL eredményeként TIME típust kapunk.
TIME mínusz TIME eredményeként INTERVAL típust kapunk.
INTERVAL plusz vagy mínusz INTERVAL eredményeként INTERVAL típust kapunk.
INTERVAL szorozva vagy osztva NUMBER eredményeként INTERVAL típust kapunk.

Jegyezzük meg, hogy az SQL-szabvány értelmében csak TIME típusú értéket vonhatunk ki TIME típusúból, és csak TIME típusú értéket adhatunk INTERVAL típusúhoz.

Az összes buktató, amit elmondtunk a dátumkifejezésekkel kapcsolatban, érvényes az időkifejezésekre is. Továbbá azok a rendszerek, amelyek támogatják a dátumot és időt egyaránt tartalmazó DATETIME adattípust, annak időrészét egy nap törtrészeként kifejezve tárolják, legalább másodperc pontosságig. A DATETIME típust támogató rendszerekben az összeadás és kivonás során általában használhatunk tört értékeket is; például a 0,25 hat órát (negyed napot) jelent. A könyv az feltételezi, hogy rendszerünk az időliterálok, illetve -oszlopok összeadását és kivonását is támogatja. A tizedestörtek hozzáadásáról és kivoná-

sáról viszont semmilyen előfeltételezéssel nem élünk. Ismételten azt kell mondanunk, ellenőrizzük a rendszerünk dokumentációját, hogy megtudjuk, hogy az esetünkben mely műveletek támogatottak.

A fentiek tükrében nézzünk meg néhány példát az időkifejezésekre:

```
'14:00' + '00:22'
'19:00' - '16:30'
StartTime + '00:19'
StopTime - StartTime
```

Korábban már mondtuk, hogy a dátum- és időkifejezésekkel csak általánosságban fogunk foglalkozni. Az a célunk, hogy az Olvasó a dátum- és időkifejezések lényegét értse meg, valamint hogy általános képet kapjon a létrehozható kifejezések fajtáiról. Sajnos a legtöbb adatbázis-kezelő nem valósítja meg tökéletesen a szabvány előírásait az időkifejezések esetében, és sok közülük csak részlegesen támogatja a dátumkifejezésekkel kapcsolatos meghatározásokat. Szerencsére azonban minden rendszer biztosít függvényeket a dátumokkal és időpontokkal való munkához. Ezen függvényeknek az öt legfontosabb rendszerben való megvalósításáról a „C" függelékben találunk összefoglalót. Melegen ajánlott tanulmányozni az adatbázis-kezelőnk dokumentációját, hogy megtudjuk, milyen függvényeket támogat.

Most, hogy megismertük a különféle kifejezések létrehozását, következő lépésként tanuljuk meg használni is őket!

Kifejezések használata a SELECT záradékban

A kifejezések használatának ismerete az egyik legfontosabb, amit megtanulhatunk ebből a könyvből. Amikor az SQL nyelvvel dolgozunk majd, széles körben fogunk használni kifejezéseket. Például minden bizonnyal kifejezéseket kell írnunk ahhoz, hogy

- olyan oszlopot jelenítsünk meg egy lekérdezésben, amelynek az értékét számítással kaptuk,
- megkeressünk egy értéket egy bizonyos oszlopban,
- szűkítsük egy eredményhalmaz sorainak körét, vagy
- összekapcsoljunk két táblát egy JOIN művelet segítségével.

A könyv hátralévő részében mindezt (és még sok egyéb dolgot) el fogjuk sajátítani. Kezdjük azzal, hogy megnézzük, hogyan használhatunk egyszerű kifejezéseket a SELECT záradékban.

Megjegyzés *A fejezetben végig a 4. fejezetben megismert „Kérelem – Fordítás – Tisztázás – SQL" módszert fogjuk használni.*

A SELECT záradékban egyszerű kifejezések használatával egyértelműbbé tehetjük az eredményhalmazban megjelenő információkat, illetve kiszélesíthetjük a kapott információk körét. Például írhatunk kifejezéseket, hogy összefűzzünk vezeték- és keresztneveket, kiszámoljuk az árát egy terméknek, meghatározzuk, hogy mennyi időt vesz majd igénybe egy projekt véghezvitele, vagy megtudjuk az ügyfelünkkel való következő találkozó dátumát. Lássuk, hogyan használhatunk egy összefűző kifejezést, matematikai kifejezést vagy dátumkifejezést a SELECT záradékban! Először összefűző kifejezésekkel dolgozunk majd.

Az összefűző kifejezések használata

A matematikai és dátumkifejezésekkel ellentétben az összefűző kifejezéseket csak a kapott információk olvashatóságának javítására használjuk. Tegyük fel, hogy az alábbi kérelmet intézzük az adatbázishoz:

„Mutasd meg a jelenleg nálunk dolgozók listáját és a telefonszámaikat!"

Miközben ezt a kérelmet SELECT utasítássá alakítjuk, némileg javíthatunk az eredményhalmaz kimenetén, ha egy oszloppá fűzzük össze a vezeték- és keresztneveket. A kérelmet például így alakíthatjuk át:

Fordítás	Select the first name, last name, and phone number of all our employees from the employees table (Válaszd ki minden alkalmazott vezetéknevét, keresztnevét és telefonszámát az alkalmazottak táblájából.)
Tisztázás	Select ~~the~~ first name, last name, ~~and~~ phone number ~~of all our employees~~ from ~~the~~ employees ~~table~~

SQL	`SELECT EmpFirstName || ' ' || EmpLastName,`
	`'Phone Number: ' || EmpPhoneNumber`
	`FROM Employees`

Ennek eredményeképpen egy ehhez hasonló sort fogunk kapni:

Mary Thompson	Phone Number: 555-251

Figyeljük meg, hogy azon kívül, hogy összefűztük a keresztnevet egy szóközzel, majd a vezetéknévvel, a `"Phone Number: "` karakterlánc-literált is hozzáfűztük a `phone number` oszlop értékéhez. Ebben a példában tisztán látható, hogy milyen könnyen javíthatjuk a SELECT utasítás olvashatóságát egy összefűző kifejezés segítségével. Ne felejtsük el, hogy különböző típusú adatokat is összefűzhetünk a CAST függvénnyel. Például összefűzhetjük egy szöveget tartalmazó oszlop értékét egy számot tartalmazó oszlop értékével az alábbiak mintájára:

„Mutasd meg a beszállítók listáját és az azonosító számaikat!"

Fordítás Select the vendor name and vendor ID from the vendors table
(Válaszd ki a beszállítók nevét és a beszállítók azonosítóját a beszállítók táblájából.)

Tisztázás Select ~~the~~ vendor name ~~and~~ vendor ID from ~~the~~ vendors ~~table~~

SQL
```
SELECT 'The ID Number for ' || VendName ||
   ' is ' || CAST(VendorID AS CHARACTER)
FROM Vendors
```

Az összefűző kifejezés hasznos eszköz lehet egy SELECT utasításban, de csak akkor, ha bölcsen használjuk. Ha összefűző kifejezéseink hosszú karakterlánc-literálokat tartalmaznak, ne felejtsük el, hogy ezek a literálok az eredményhalmaz minden sorában megjelennek, ez az ismétlődő információ pedig inkább összezavarja a végeredményt, ahelyett hogy kiegészítené azt. Ezért megfontoltan döntsünk a literálok használatáról az összefűző kifejezésekben, hogy valóban megkönnyítsék a munkánkat.

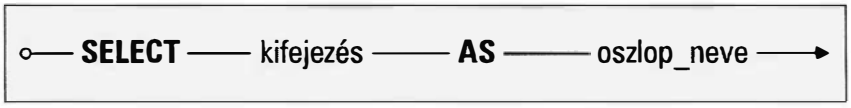
A kifejezések elnevezése

Amikor egy SELECT záradékban kifejezést használunk, az eredményhalmaz új oszloppal bővül, amelyben megjelenik a kifejezésben meghatározott műveletek értéke. Ezt az új oszlopot számított oszlopnak (vagy származtatott oszlopnak nevezzük). Az alábbi SELECT utasítás eredményhalmaza például három oszlopot fog tartalmazni – két „valódi" oszlopot és egy számított oszlopot:

SQL
```
SELECT EmpFirstName || ' ' || EmpLastName,
   EmpPhoneNumber, EmpCity
FROM Employees
```

A két valódi oszlop természetesen az EmpPhoneNumber és az EmpCity, a számított oszlop pedig az összefűző kifejezés eredményeképpen áll elő a SELECT záradék elején. Az SQL-szabványnak megfelelően az új oszlopot el is nevezhetjük (nem kötelező) az AS kulcsszó használatával. (Gyakorlatilag bármely oszlophoz új nevet rendelhetünk az AS záradék segítségével.) Mindazonáltal majdnem minden adatbázis-kezelő rendszer megköveteli a számított oszlopok elnevezését. Néhányuk konkrét névadást igényel, míg mások automatikusan állítanak elő egy nevet. Nézzünk utána, hogy a rendszerünk hogyan kezeli ezt, mielőtt munkához látunk a példákkal. Ha a későbbiekben hivatkozni szeretnénk egy kifejezés eredményére egy lekérdezésben, mindenképpen nevet kell adnunk neki.

A kifejezések elnevezésének utasításformáját az 5.9. ábra mutatja. Névként bármilyen érvényes karakterlánc-literált használhatunk (aposztrófok közé zárva). Néhány adatbázis-kezelő nem foglalkozik ezzel a követelménnyel a kifejezés ellenőrzésekor, és az aposztrófokat csak akkor követeli meg, ha az adott név szóközöket tartalmaz. Mi azonban erősen javasoljuk, hogy kerüljük a szóközök használatát a nevekben, ezek ugyanis néhány adatbázis-kezelő programozási nyelvében megnehezíthetik a dolgunkat.



5.8. ábra

A kifejezések elnevezésének szintaxisdiagramja

Most pedig módosítsuk az előző példa SELECT utasítását, és adjunk nevet az összefűző kifejezésnek:

```
SQL     SELECT EmpFirstName || ' ' || EmpLastName AS
           EmployeeName, EmpPhoneNumber, EmpCity
        FROM Employees
```

A SELECT utasítás eredményhalmaza most már három oszlopot tartalmaz, EmployeeName, EmpPhoneNumber és EmpCity néven. Az AS kulcsszó segítségével nem csak a kifejezéseket nevezhetjük el, de álnevet is rendelhetünk egy valódi oszlophoz. Tegyük fel, hogy van egy DOB nevű oszlopunk, és attól tartunk, hogy néhány felhasználónk nem lesz tisztában ennek a rövidítésnek a jelentésével. Szerencsére minden félreértést tisztázhatunk egy álnév megadásával:

```
SQL     SELECT EmpFirstName || ' ' || EmpLastName AS
           EmployeeName, DOB AS DateOfBirth
        FROM Employees
```

Ez a SELECT utasítás egy két oszlopból álló eredményhalmazzal tér vissza, amelyek az EmployeeName és DateOfBirth nevet viselik. Ezzel gyakorlatilag minden lehetséges félreértést tisztáztunk a megjelenő információkkal kapcsolatban.

A számított oszlopok elnevezése kis mértékben befolyásolja a jól bevált fordítási folyamatot. Az előző példa kérelmének lefordítása például így történhet:

„Adj egy listát az alkalmazottak nevével és születésük dátumával!"

Fordítás Select first name and last name as employee name and DOB as date of birth from the employees table
(Válaszd ki a keresztnevet és a vezetéknevet EmployeeName (alkalmazott neve) néven és a DOB értékét DateOfBirth (születési dátum) néven az alkalmazottak táblájából.)

Tisztázás Select first name ~~and~~ || '' || last name as EmployeeName ~~and~~ DOB as DateOfBirth from ~~the~~ employees ~~table~~

```
SQL     SELECT EmpFirstName || ' ' || EmpLastName
        AS EmployeeName, DOB AS DateOfBirth
        FROM Employees
```

Ha már egy kis gyakorlatot szereztünk a kifejezések használatában, nem lesz rá szükségünk rá, hogy ilyen konkrétan kidolgozzuk a lefordított utasításokat, amint azt az előzőekben tettük. A kifejezéseket közvetlenül a SELECT utasítás írása során könnyedén képesek leszünk azonosítani és megfogalmazni.

Megjegyzés *A könyv hátralévő részében az SQL-utasításokban szereplő számított oszlopokat mindig megfelelően elnevezzük majd.*

A matematikai kifejezések használata

A matematikai kifejezések minden bizonnyal a legsokoldalúbbak a kifejezések három típusa közül, és valószínűleg nagyon gyakran használjuk majd őket. Például matematikai kifejezéseket használunk, hogy összegezzük egy sor elemeit, hogy meghatározzuk, mennyi az átlagos pontszáma bizonyos tesztsorozatoknak, hogy kiszámoljuk az eltérést két laboratóriumi eredmény között, vagy hogy megbecsüljük egy épület befogadóképességét. A működőképes kifejezések írásának az a trükkje, hogy némi figyelmes tervezés előzze meg.

Az alábbiakban egy példát láthatunk a matematikai kifejezések használatára egy SELECT utasításban:

„Jelenítsd meg minden ügynök nevét és várható bevételét (alapbér és teljesítményarányos juttatások), feltételezve, hogy minden ügynök 50 000 dollár értékű megrendelést szerez!"

Fordítás Select first name and last name as agent name and salary plus 50000 times commission rate as projected income from the agents table
(Válaszd ki a keresznevet és vezetéknevet AgentName (ügynök neve) néven és az alapbért összegezve az értékesítési hányados 50 000-szeresével Projected Income (tervezett bevétel) néven az ügynökök táblájából.)

Tisztázás Select first name ~~and~~ || ' ' || last name as AgentName, ~~and~~ salary ~~plus~~ + 50000 ~~times~~ * commission rate as Projected Income from ~~the~~ agents ~~table~~

SQL
```
SELECT AgtFirstName || ' ' || AgtLastName
   AS AgentName, Salary + (50000 * CommissionRate)
   AS ProjectedIncome
FROM Agents
```

Figyeljük meg, hogy zárójelezéssel tettük egyértelművé, hogy az értékesítési hányadost kell megszoroznunk 50 000-rel és utána hozzáadni az alapbérhez, és nem az 50 000-ret hozzáadni az alapbérhez és aztán megszorozni az értékesítési hányadossal. Amint a példából látható, nincs olyan megkötés, hogy csak egyféle kifejezést használhatunk a SELECT utasításokban, sőt kifejezések széles választékát alkalmazhatjuk, hogy lekérjük a szükséges információkat az eredményhalmazban. Az előző SQL-utasítást így is leírhatjuk:

```
SQL     SELECT AgtFirstName || ' ' || AgtLastName
           || ' has a projected income of ' ||
           CAST(Salary + (50000 * CommissionRate) AS CHARACTER)
           AS ProjectedIncome
        FROM Agents
```

A matematikai kifejezések segítségével lekérdezhető információk köre szinte végtelen, de mindig meg kell terveznünk a kifejezéseinket, és szükség esetén használnunk kell a CAST függvényt.

A dátumkifejezések használata

A dátumkifejezéseket a matematikai kifejezésekkel megegyező módon használhatjuk: egyszerűen értékeket adunk össze vagy vonunk ki egymásból. A legkülönbözőbb feladatok során láthatjuk hasznukat: meghatározhatjuk egy szállítás becsült dátumát, egy projekt befejeztéig hátralévő napok számát vagy a következő találkozó időpontját egy ügyfelünkkel. Az alábbi példa egy dátumkifejezés használatát szemlélteti egy SELECT záradékban:

„Hány napot vett igénybe az egyes megrendelések teljesítése?"

Fordítás Select the order number and ship date minus order date as days to ship from the orders table
(Válaszd ki a rendelés számát és a teljesítés dátumát mínusz a rendelés dátumát DaysToShip (teljesítési idő) néven a rendelések táblájából.)

Tisztázás Select ~~the~~ order number ~~and~~ ship date ~~minus~~ – order date as DaysToShip
from ~~the~~ orders ~~table~~

```
SQL     SELECT OrderNumber, CAST(ShipDate - OrderDate
           AS INTEGER) AS DaysToShip
        FROM Orders
```

Az időkifejezések esetében ugyanígy járhatunk el:

„Mi lenne az egyes órák kezdési időpontja, ha az órákat tíz perccel később kezdenénk a jelenlegi kezdési időpontnál?"

Fordítás Select the start time and start time plus 10 minutes as new start time from the classes table
(Válaszd ki a kezdési időpontot és a kezdési időpont plusz 10 percet NewStartTime (új kezdési időpont) néven az órák táblájából.)

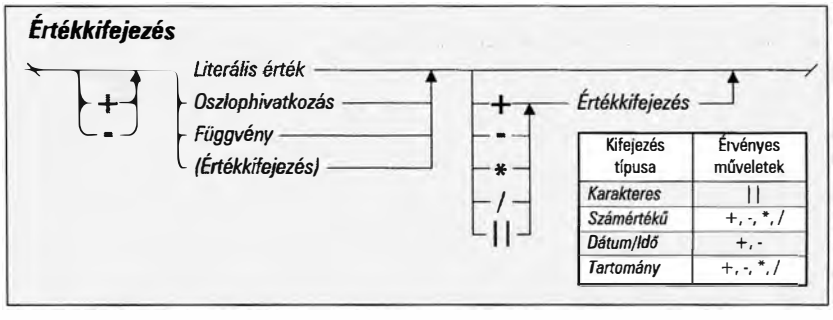
Tisztázás Select ~~the~~ start time ~~and~~ start time ~~plus~~ + '00:10' ~~minutes~~ as NewStartTime
from ~~the~~ classes ~~table~~

```
SQL     SELECT StartTime, StartTime + '00:10'
           AS NewStartTime
        FROM Classes
```

Ahogy korábban említettük, minden adatbázis-kezelő rendszer biztosít függvényeket a dátumműveletekhez. A könyvben ötletet adtunk, hogyan használhatjuk a dátumokat és időpontokat a SELECT utasításokban, de ismét meg kell jegyeznünk, hogy tanácsos áttanulmányozni az adatbázis-kezelőnk dokumentációját a dátum- és időfüggvényekkel kapcsolatos részletek megismerése végett.

Egy rövid kitérő: értékkifejezések

Már tudjuk, hogyan használjunk oszlophivatkozásokat, literális értékeket és kifejezéseket egy SELECT záradékban. Azt is tudjuk, hogyan rendelhetünk nevet egy oszlophivatkozáshoz vagy kifejezéshez. Most meg fogjuk látni, hogyan válik mindez egy nagyobb rendszer részévé. Az SQL-szabvány az oszlophivatkozásokra, literális értékekre és kifejezésekre együttesen *értékkifejezés* néven hivatkozik. Az 5.10. ábra bemutatja, hogyan adhatunk meg értékkifejezéseket.



5.10. ábra

Az értékkifejezések szintaxisdiagramja

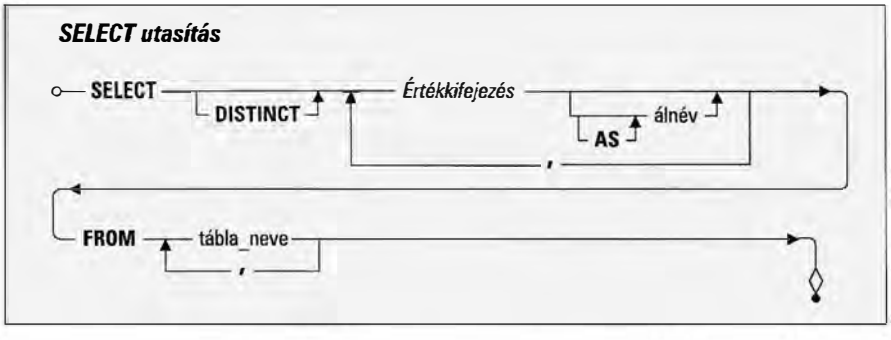
Vizsgáljuk meg közelebbről az értékkifejezések alkotóelemeit:

- Az értékkifejezés egy nem kötelező plusz- vagy mínuszjellel kezdődik. Ezeket a jeleket akkor kell megadnunk, ha azt szeretnénk, hogy az értékkifejezés egy előjeles értékkel térjen vissza. Az érték maga lehet számliterál, szám típusú oszlop értéke, függvényhívás, amely számmal tér vissza (lásd a CAST függvénynél leírtakat a fejezet korábbi részében) vagy egy matematikai kifejezés visszatérési értéke. A plusz- és mínuszjelet a karakteres típusú adatokat visszaadó kifejezések esetében nem használhatjuk.
- Megfigyelhetjük, hogy az első felsorolás az ábrán tartalmazza az (*Értékkifejezés*) elemet is. Ez azt jelenti, hogy összetett értékkifejezéseket is létrehozhatunk, amelyek más értékkifejezéseket foglalnak magukba. Ezeknek szintén meglehetnek a saját összefűző vagy matematikai műveletei. Zárójelezéssel kényszeríthetjük az adatbázis-kezelőt, hogy egy bizonyos értékkifejezést értékeljen ki először.
- Az értékkifejezések következő eleme a műveleti jel. Amint a belső táblázatban láthatjuk, az utasításforma elején szereplő kifejezés típusa határozza meg, hogy mely műveletek használatára van lehetőségünk.

- Nem, nem káprázik a szemünk: a műveleti jel után ismételten következhet egy *értékkifejezés*. Az, hogy egy értékkifejezésen belül használhatunk más értékkifejezéseket, nagyon összetett kifejezések létrehozását teszi lehetővé.

Egyszerűen fogalmazva tehát az értékkifejezés olyan értéket ad vissza, amelyet az SQL-utasításunk valamelyik összetevője használ majd. Az SQL-szabvány a legkülönbözőbb utasításokban és elemekben írja le az értékkifejezések használatát. Akárhol is használjuk, egy értékkifejezést mindig a fent leírt módon építhetünk fel.

Hogy mindezt jobban el tudjuk képzelni, az alábbiakban megfigyelhetjük az értékkifejezés használatát egy SELECT utasításban. Az 5.11. ábra a SELECT utasítás szintaxisdiagramjának (4. fejezet, 4.9. ábra) módosított változatát mutatja be. Az új forma nagyon rugalmas: literálok, oszlophivatkozások, kifejezések vagy ezek tetszőleges kombinációjának használatát teszi lehetővé egyetlen SELECT utasításban. Az értékkifejezéseinket el is nevezhetjük az AS kulcsszó használatával.



5.11. ábra

Az értékkifejezést tartalmazó SELECT utasítások szintaxisdiagramja

A könyv további részében *értékkifejezés* alatt oszlophivatkozást, literális értéket vagy kifejezést értünk (természetesen ezek közül a megfelelőt). A későbbi fejezetekben azt is tárgyaljuk, hogyan használhatjuk az értékkifejezéseket más utasításokban, és bemutatunk néhány további dolgot, amit egy értékkifejezés kifejezhet. Most pedig térjünk vissza az eredeti menetrendünkhöz!

A „semmit" jelképező érték: a Null

Mint azt már tudjuk, a táblák oszlopokból és sorokból állnak. Minden oszlop a tábla által leírt dolog egy tulajdonságát képviseli, míg az egyes sorok az adott dolog egy konkrét példányát jelképezik. A sorokat az oszlopok értékeinek teljes halmazaként is felfoghatjuk – minden sor pontosan egy értéket tartalmaz a tábla minden oszlopából. A 5.12. ábra egy jellemző táblát szemléltet.

Customers

CustomerID	CustFirstName	CustLastName	CustStreetAddress	CustCity	CustCounty	CustState
1001	Suzanne	Viescas	15127 NE 24th, #383	Redmond	King	WA
1002	William	Thompson	122 Spring River Drive	Duvall	King	WA
1003	Gary	Hallmark	Route 2, Box 203B	Auburn	King	WA
1004	Robert	Brown	672 Lamont Ave	Houston		TX
1005	Dean	McCrae	4110 Old Redmond Rd.	Redmond		WA
1006	John	Viescas	15127 NE 24th, #383	Redmond	King	WA
1007	Mariya	Sergienko	901 Pine Avenue	Portland		OR
1008	Neil	Sergienko	233 West Valley Hwy	San Diego	San Diego	CA

5.12. ábra

Egy vásárlók adatait tartalmazó jellemző tábla

Az eddigiekben megismertük, hogy kérdezhetünk le információkat egy táblából a SELECT utasítás segítségével, és hogyan módosíthatjuk ezeket kifejezések használatával. Mindez gördülékenyen ment eddig, mivel azt feltételeztük, hogy a tábla minden oszlopa tartalmaz adatokat. Az 5.12. ábra azonban jól szemlélteti, hogy előfordulhatnak sorok, amelyekben egyes oszlopok nem tartalmaznak értéket. Attól függően, hogy miként szeretnénk felhasználni az adatokat, ezeknek az értékeknek a hiánya különbözőképpen érintheti a SELECT utasításainkat és értékkifejezéseinket. Mielőtt részletesen kifejtjük, mire céloznak az előző mondatok, vizsgáljuk meg, hogyan viszonyul az SQL a hiányzó értékekhez!

Bemutatkozik a Null érték

Az SQL-ben a *Null* a hiányzó vagy ismeretlen értékeket testesíti meg. Tudnunk kell, hogy a Null *nem* azonos sem a nullával, sem pedig az egy vagy több szóközt tartalmazó vagy nulla hosszúságú karakterláncokkal. Ennek oka egészen egyszerű:

- A nulla értéknek számtalan jelentése lehet: vonatkozhat számlaegyenlegre, a rendelkezésre álló első osztályra szóló jegyek mennyiségére, vagy hogy hány darab van raktáron egy bizonyos termékből. Egy vagy több szóköz karakter pedig – bár számunkra nem sok jelentéssel bír – az SQL számára egyáltalán nem értelmetlen: egy három szóközből álló karakterlánc (`'   '`) éppen úgy teljes értékű karakteres adatnak számít, mint egy betűkből álló karakterlánc (`'karakterlánc'`).
- A nulla hosszúságú karakterlánc – azaz két egymást szóközök nélkül követő aposztróf (`''`) – is bizonyos esetekben szintén jelentéssel bírhat. Egy alkalmazottak adatait tároló táblában a KözépsőNév oszlopban szereplő nulla hosszúságú karakterlánc például azt jelentheti, hogy az adott alkalmazottnak nincs középső neve. Jó ha tudjuk azonban, hogy egyes adatbázis-kezelők (például az Oracle) a VARCHAR típusú mezőkben levő nulla hosszúságú karakterláncokat Null-nak tekintik.

A Null érték megfelelően használva egészen hasznos lehet. Az 5.12. ábra Customers (Vásárlók) táblája nagyon jó példa erre. A CustCountry oszlopban minden üresen hagyott mező egy hiányzó vagy ismeretlen államnevet jelent az adott sor esetében, ahol feltűnik. A Null értékek helyes használatához meg kell értenünk, hogy honnan származnak.

A hiányzó értékek általában emberi hiba eredményei. Tekintsünk például a Robert Brown nevét tartalmazó sorra: Brown úr adatainak rögzítése során elfelejtették megkérdezni, hogy melyik államban él. Ez az adat tehát hiányozni fog, az adatbázisban pedig Null értékként jelenik meg. Később természetesen észrevehetjük a hibát, és megkérdezhetjük Brown úrtól, hogy melyik államban is lakik.

Egy táblában több okból is előfordulhatnak ismeretlen értékek. Az egyik ok lehet például, hogy az oszlopban szereplő érték még nem ismert az adatok rögzítésekor. Például ha van egy Categories (Tantárgycsoportok) nevű táblánk a School Scheduling (Iskolai nyilvántartás) adatbázisban, akkor az valószínűleg még nincs kitöltve az újonnan induló órák esetében, amelyeket majd csak az őszi félévben szeretnénk meghirdetni. A másik helyzet, amikor ismeretlen értékek fordulhatnak elő egy táblában, ha az adott érték valóban ismeretlen a számunkra. Tekintsük meg ismét az 5.12. ábra Costumers tábláját, azon belül is a Dean McCrae nevét tartalmazó sort. Tegyük fel, hogy McRae úr adatainak rögzítésekor megkérdezzük tőle, hogy melyik államban él. Ha egyikünk sem tudja, hogy az adott város melyik államhoz tartozik, akkor a Country oszlop értéke ebben a sorban ténylegesen ismeretlennek mondható. Ez a sorban egy Null érték formájában nyilvánul majd meg. Ez természetesen később javítható, ha valamelyikünk utánanéz az állam nevének.

Egy oszlop értéke akkor is Null lesz, ha semmilyen érték nem megfelelő a számára az adott sor esetében. Például ha egy olyan alkalmazotti táblával van dolgunk, amely tartalmaz egy HaviBér és egy Óradíj nevű oszlopot is, valamelyik oszlop értéke a kettő közül mindig Null lesz, hiszen nem fizethetünk valakinek egyszerre havi meghatározott összeget és órabért is.

Fontos megemlítenünk, hogy nagyon vékony a határ aközött, hogy egy adat „nem értelmezett" és aközött, hogy „nem értelmezhető". Az előző példában a két oszlop közül az egyik nem értelmezett. De tegyük fel, hogy egy ügyféltáblával akad dolgunk, amely tartalmaz egy Hajszín nevű oszlopot, és éppen frissítünk egy sort, amely egy már létező férfi ügyfelünk adatait tartalmazza. Abban az esetben, ha az ügyfelünk kopasz, az oszlop értéke nem értelmezhető. Bár használhatnánk Null értéket is a nem értelmezhető értékek kifejezéseként, a hasonló helyzetekben egy létező érték – például „Nem értelmezhető" (angol környezetben általában az N/A – Not Applicable értéket használják ilyen esetekben) – alkalmazását javasoljuk. Ez hosszú távon jobban érthetővé teszi az információkat.

Amint láthattuk, az, hogy engedélyezzük-e a Null értékek megjelenését egy táblában, attól függ, hogyan szeretnénk majd feldolgozni az adatokat. Miután megismerkedtünk a Null értékek pozitív tulajdonságaival, foglalkozzunk kicsit az árnyoldalukkal is!

A Null értékek hátulütői

A Null értékek legfőbb hátulütője, hogy kedvezőtlenül viselkednek a matematikai műveletek során. Minden művelet, amelyben egy Null érték is felbukkan, Null értéket ad eredményül. Ennek logikája a elfogadható – ha egy bizonyos érték ismeretlen, akkor a művelet végeredménye is szükségszerűen ismeretlen lesz. Figyeljük meg, hogyan befolyásolja a Null érték a művelet eredményét a következő példában:

```
(25 * 3) + 4 = 79
(Null * 3) + 4 = Null
(25 * Null) + 4 = Null
(25 *3) + Null = Null
```

Ha a művelet Null értéket is tartalmaz, minden esetben azonos eredményt kapunk. Tegyük fel például, hogy végrehajtjuk az alábbi SELECT utasítást, és az az 5.13. ábrán látható eredményhalmazzal tér vissza:

```
SQL     SELECT ProductID, ProductDescription, Category,
           Price, QuantityOnHand, Price *
           QuantityOnHand AS TotalValue
        FROM Products
```

ProductID	ProductDescription	Category	Price	QuantityOnHand	TotalValue
70001	Shur-Lok U-Lock	Accessories		12	
70002	SpeedRite Cyclecomputer		65.00	20	1,300.00
70003	SteelHead Microshell Helmet	Accessories	36.00	33	1,118.00
70004	SureStop 133-MB Brakes	Components	23.50	16	376.00
70005	Diablo ATM Mountain Bike	Bikes	1,200.00		
70006	UltraVision Helmet Mount Mirrors		7.45	10	74.50

5.13. ábra

A null értékek viselkedése matematikai műveletekben

A művelet, amely a TotalValue oszlopban megjelenő értéket adja, sikeresen végrehajtódik, amíg a Price és QuantityOnHand oszlopok érvényes számértékeket tartalmaznak. Egyéb esetben a QuantityOnHand oszlop Null-t ad eredményül. A jó hír az, hogy a TotalValue oszlop a megfelelő értéket fogja tartalmazni, ha a Price és QuantityOnHand mezők Null értékeit érvényes számokkal helyettesítjük. A probléma teljes egészében elkerülhető, ha ügyelünk rá, hogy a matematikai kifejezésekben részt vevő oszlopok ne tartalmazzanak Null értékeket.

Nem most beszéltünk utoljára a Null értékekről. A 12. fejezetben azt is megnézzük, hogy miként hatnak a Null-ok az információkat összegző SELECT utasításokra.

Példák

Most, hogy tudjuk, hogyan használhatunk különféle kifejezéseket a SELECT utasítás SELECT záradékában, tekintsük át a következő néhány oldalon szereplő példákat, amelyek a mintaadatbázisok tábláit használják. A példák szemléltetik, hogyan használhatjuk a kifejezéseket kimeneti oszlopok előállítására.

Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név ugyanaz, mint a CD-mellékleten található megfelelő SQL-lekérdezés neve. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH05"-tel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *A következő példákban összevontuk a Fordítás és Tisztázás lépéseit, hogy megszokjuk, hogyan végezhetjük el tömörebb formában a műveletet. A fejezetek törzsében ugyanakkor továbbra is a háromlépéses módszert használjuk majd, de a példák során már alkalmunk nyílik gyakorolni a tömörebb megoldást is.*

Sales Orders adatbázis

„Mennyi az egyes termékeink leltári értéke?"

Fordítás/Tisztázás SQL

Select ~~the~~ product name, retail price ~~times~~ * quantity on hand as InventoryValue from ~~the~~ products ~~table~~

```
SELECT ProductName,
   RetailPrice * QuantityOnHand AS
   InventoryValue
FROM Products
```

CH05_Product_Inventory_Value (40 sor)

ProductName	InventoryValue
Trek 9000 Mountain Bike	$7,200.00
Eagle FS-3 Mountain Bike	$14,400.00
Dog Ear Cyclecomputer	$1,500.00
Victoria Pro All Weather Tires	$1,099.00
Dog Ear Helmet Mount Mirrors	$89.40
Viscount Mountain Bike	$3,175.00
Viscount C-500 Wireless Bike Computer	$1,470.00
Kryptonite Advanced 2000 U-Lock	$1,000.00
<< *további sorok* >>	

„Hány nap telt el a rendelés és a teljesítés dátuma között az egyes rendelések esetében?"

Fordítás/ Select ~~the~~ order number, order date, ship date, ship date
Tisztázás ~~minus~~ – order date as DaysElapsed from ~~the~~ orders ~~table~~

SQL
```
SELECT OrderNumber, OrderDate, ShipDate,
   CAST(ShipDate - OrderDate AS INTEGER)
   AS DaysElapsed
FROM Orders
```

CH05_Shipping_Days_Analysis (944 sor)

OrderNumber	OrderDate	ShipDate	DaysElapsed
1	2007-09-01	2007-09-04	3
2	2007-09-01	2007-09-03	2
3	2007-09-01	2007-09-04	3
4	2007-09-01	2007-09-03	2
5	2007-09-01	2007-09-01	0
6	2007-09-01	2007-09-05	4
7	2007-09-01	2007-09-04	3
8	2007-09-01	2007-09-01	0
9	2007-09-01	2007-09-04	3
10	2007-09-01	2007-09-04	3
<< *további sorok* >>			

Entertainment Agency adatbázis

„Mennyi ideig tartanak az egyes rendezvények?"

Fordítás/ Select ~~the~~ engagement number, end date ~~minus~~ – start date
Tisztázás ~~plus one~~ + 1 as DueToRun from ~~the~~ engagements ~~table~~

SQL
```
SELECT EngagementNumber,
   CAST(CAST(EndDate - StartDate AS INTEGER) + 1
   AS CHARACTER)
   || ' day(s)' AS DueToRun
FROM Engagements
```

Megjegyzés *Azért kell hozzáadnunk egyet a dátumok különbségéhez a kifejezésben, mert a kérdés arra vonatkozik, hogy hány napig tart egy-egy rendezvény, így máskülönben 0 napos időtartamot kapnánk az egyazon dátummal kezdődő és végződő rendezvények esetében. A kivonás eredményét a CAST függvény segítségével alakítjuk át INTEGER típusúra, hogy az 1 hozzáadását el tudjuk végezni. Ez után ismét a CAST használatára van szükség, hogy CHARACTER típusúvá alakítsuk az értéket az összefűzéshez.*

CH05_Engagement_Lengths (111 sor)

EngagementNumber	DueToRun
2	5 day(s)
3	6 day(s)
4	7 day(s)
5	4 day(s)
6	5 day(s)
7	8 day(s)
8	8 day(s)
9	11 day(s)
10	10 day(s)
11	2 day(s)
<< *további sorok* >>	

„Mennyi az egyes szerződéseink nettó értéke?"

Fordítás/ Tisztázás: Select ~~the~~ engagement number, contract price, contract price ~~times~~ * 0.12 as OurFee, contract price ~~minus~~ – (contract price ~~times~~ * 0.12) as NetAmount from ~~the~~ engagements ~~table~~

SQL:

```
SELECT EngagementNumber, ContractPrice,
   ContractPrice * 0.12 AS OurFee,
   ContractPrice -(ContractPrice * 0.12)
   AS NetAmount
FROM Engagements
```

CH05_Net_Amount_Per_Contract (111 sor)

EngagementNumber	ContractPrice	OurFee	NetAmount
2	$200.00	$24.00	$176.00
3	$590.00	$70.80	$519.20
4	$470.00	$56.40	$413.60
5	$1,130.00	$135.60	$994.40
6	$2,300.00	$276.00	$2,024.00
7	$770.00	$92.40	$677.60
8	$1,850.00	$222.00	$1,628.00
9	$1,370.00	$164.40	$1,205.60
10	$3,650.00	$438.00	$3,212.00
11	$950.00	$114.00	$836.00
<< *további sorok* >>			

School Scheduling adatbázis

„Írd ki, hogy 2007. október 1-ig a tanári kar egyes tagjai hány évet töltöttek az iskola csapatában, az eredményt pedig rendezd kereszt- és vezetéknév szerint!"

Fordítás/ Tisztázás: Select last name || ',' || ~~and~~ first name ~~concatenated with a comma~~ as Staff, date hired, ~~and~~ (('2007-10-01' ~~minus~~ – date hired) ~~divided by~~ / 365) as YearsWithSchool from ~~the~~ staff ~~table and sort~~ order by last name ~~and~~ first name

SQL:

```
SELECT StfLastName || ', ' || StfFirstName AS Staff,
   DateHired,
   CAST(CAST('2007-10-01' - DateHired AS INTEGER)
   / 365 AS INTEGER)
   AS YearsWithSchool
FROM Staff
ORDER BY StfLastName, StfFirstName
```

CH05_Length_Of_Service (27 sor)

Staff	DateHired	YearsWithSchool
Alborous,Sam	1982-11-20	25
Black, Alastair	1988-12-11	19
Bonnicksen, Joyce	1986-03-02	22
Brehm, Peter	1986-07-16	21
Brown, Robert	1989-02-09	19
Coie, Caroline	1983-01-28	25
DeGrasse, Kirk	1988-03-02	20
Ehrlich, Katherine	1985-03-08	23
Glynn, Jim	1985-08-02	22
Hallmark, Alaina	1984-01-07	24
<< további sorok >>		

Megjegyzés *A fenti SELECT utasításban lévő kifejezés formailag tökéletes, és az elvárásainknak megfelelően működik, de szökőévek esetén rossz választ ad. A probléma az adatbázis-kezelőnk által biztosított megfelelő dátumkezelő függvény használatával hidalható át. Amint azt korábban említettük, a legtöbb adatbázis-kezelő rendszer a saját tagfüggvényeit használja a dátumokkal és időpontokkal való műveletekhez.*

„Sorold fel a tanári kar tagjait a bérükkel és egy tervezett 7 százalékos jutalommal együtt!"

Fordítás/ Tisztázás: Select ~~the~~ last name || ',' || ~~and~~ first name as StaffMember, salary, ~~and~~ salary ~~times~~ * 0.07 as Bonus from ~~the~~ staff ~~table~~

```
SQL     SELECT StfLastName || ', ' || StfFirstName
           AS Staff, Salary, Salary * 0.07 AS Bonus
        FROM Staff
```

CH05_Length_Of_Service (27 sor)

Staff	DateHired	YearsWithSchool
Alborous,Sam	$60,000.00	$4,200.00
Black, Alastair	$60,000.00	$4,200.00
Bonnicksen, Joyce	$60,000.00	$4,200.00
Brehm, Peter	$60,000.00	$4,200.00
Brown, Robert	$49,000.00	$3,430.00
Coie, Caroline	$52,000.00	$3,640.00
DeGrasse, Kirk	$45,000.00	$3,150.00
Ehrlich, Katherine	$45,000.00	$3,150.00
Glynn, Jim	$45,000.00	$3,150.00
Hallmark, Alaina	$57,000.00	$39,900.00
<< *további sorok* >>		

Bowling League adatbázis

„Sorold fel az összes tekejátékost és a lakcímeiket egy levelezési címlistának megfelelően formázva és irányítószám szerint rendezve!"

Fordítás/ Tisztázás: Select first name || ' ' || and last name as FullName, BowlerAddress, city || ', ' || state || ' ' || and ZIP Code as CityStateZip from the bowlers table and order by ZIP Code

```
SQL     SELECT BowlerFirstName || ' ' || BowlerLastName AS
        FullName,
           Bowlers.BowlerAddress,
           BowlerCity || ', ' || BowlerState || ' ' ||
           BowlerZip AS
           CityStateZip
        FROM Bowlers
        ORDER BY BowlerZip
```

CH05_Names_Addresses_For_Mailing (32 sor)

FullName	BowlerAddress	CityStateZip
Kathryn Patterson	Mary Thompson	Auburn,WA 98002
Rachel Patterson	16 Maple Lane	Auburn,WA 98002
Ann Patterson	16 Maple Lane	Auburn,WA 98002
Neil Patterson	16 Maple Lane	Auburn,WA 98002
Megan Patterson	16 Maple Lane	Auburn,WA 980025
Carol Viescas	16345 NE 32nd Street	Bellevue,WA 98004
Sara Sheskey	16345 NE 32nd Street	Seattle,WA 98011
Richard Sheskey	17950 N 59th	Seattle,WA 98011
William Thompson	122 Spring Valley Drive	Duvall,WA 98019
Mary Thompson	122 Spring Valley Drive	Duvall,WA 98019
<< további sorok >>		

„Mennyi volt az egyes mérkőzések és játszmák során a különböző tekejátékosok büntetőpontokkal terhelt és tiszta pontszáma közötti különbség?"

Fordítás/ Tisztázás: Select bowler ID, match ID, game number, handicap score, raw score, handicap score ~~minus~~ – raw score as PointDifference from ~~the~~ bowler scores ~~table and~~ order by bowler ID, match ID, game number

SQL

```
SELECT BowlerID, MatchID, GameNumber,
   HandiCapScore, RawScore,
   HandiCapScore - RawScore AS PointDifference
FROM Bowler_Scores
ORDER BY BowlerID, MatchID, GameNumber
```

CH05_Handicap_vs_RawScore (1344 sor)

BowlerID	MatchID	GameNumber	HandiCapScore	RawScore	PointDifference
1	1	1	192	146	46
1	1	2	192	146	46
1	1	3	199	153	46
1	5	1	192	145	47
1	5	2	184	137	47
1	5	3	199	152	47
1	10	1	189	140	49
1	10	2	186	137	49
1	10	3	210	161	49
<< további sorok >>					

Összefoglalás

A fejezetet a kifejezések rövid áttekintésével kezdtük, majd kifejtettük, hogy az adattípusok megértése nagyon fontos, mielőtt kifejezések felépítésébe fognánk, és megismertük az összes fontos adattípust, amellyel a későbbiekben találkozhatunk. Ez után megtanultuk a CAST függvény használatát, és megtudtuk, hogy gyakran használjuk majd kifejezésekben az adatok típusának megváltoztatására, hogy azok összeegyeztethetők legyenek a kifejezés által elvárt adattípusokkal.

Ezt követően sorra vettük az összes lehetséges módszert, ahogyan egy állandó értéket – egy literált – felhasználhatunk a kifejezésekben, majd megismertük, hogy miként használhatjuk a kifejezéseket az adatbázisból lekérdezett információk körének bővítésére és szűkítésére. Megtanultuk, hogy a kifejezés olyan művelet, amelynek részeként számokat, karakterláncokat, dátumokat és időpontokat egyaránt alkalmazhatunk.

Ez után folytattuk a kifejezések tárgyalását, és behatóbban megismerkedtünk azok típusaival. Láttuk, hogyan fűzhetünk össze karakterláncokat egymással, és hogyan fűzhetjük őket össze más adattípusokkal a CAST függvény segítségével. Megismerkedtünk a matematikai kifejezésekkel, és megtanultuk, hogy a műveletek végrehajtási sorrendje hogyan befolyásolja a műveletek végeredményét. Ezt a részt a dátum- és időkifejezések tárgyalásával zártuk. Miután megismertük, hogy az SQL-szabvány hogyan kezeli a dátumokat és időpontokat, azt is megtudtuk, hogy az adatbázis-kezelők általában saját, egyedi tagfüggvényeket nyújtanak a dátumokkal való munkához.

Ez után továbbléptünk a kifejezések SELECT utasításokban való használatát bemutató témakörre, és megtanultuk, hogyan ágyazhatjuk a kifejezéseinket a SELECT záradékba. Láttuk, hogyan használhatunk egyidejűleg literálokat és oszlophivatkozásokat a kifejezésekben, valamint hogy miként adhatunk nevet a kifejezések eredményét tartalmazó oszlopoknak. Mielőtt lezártuk volna a témát, tettünk egy rövid kitérőt, és megismerkedtünk az értékkifejezés fogalmával. Kiderült, hogy az SQL-szabvány ezen a néven hivatkozik együttesen az oszlophivatkozásokra, literális értékekre és kifejezésekre a különféle SQL-záradékokban. (Természetesen erről később többet is megtudhatunk majd.)

A fejezetben végezetül a Null értéket tárgyaltuk. Megtanultuk, hogy a Null hiányzó vagy ismeretlen értéket jelöl. Megfigyelhettük a Null értékek helyes használatát, és megtudhattuk, hogy megfelelően alkalmazva a Null értékek nagyon hasznosak lehetnek a számunkra. Megismerkedtünk ugyanakkor a Null árnyoldalával is, vagyis azzal, hogy a Null kedvezőtlenül befolyásolhatja a matematikai műveletek eredményét. Megtanultuk, hogy minden Null értékeket érintő matematikai művelet Null eredménnyel tér vissza, valamint azt, hogy miként tehetik a Null értékek az eredményhalmazunkat pontatlanná.

A következő fejezetben annak az elvéről olvashatunk, hogy miként kérdezhetjük le adatok meghatározott csoportját, majd megismerkedünk a WHERE záradékkal, amellyel szűrhetjük a SELECT utasítás által szolgáltatott információkat.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Mi lenne, ha csökkentenénk a termékeink árát 5 százalékkal?"*
   A megoldás itt található: CH05_Adjusted_Wholesale_Prices (90 sor).
2. *„Mutasd meg az egyes vásárlók rendeléseit csökkenő dátumsorrendben!"*
   (Tipp: valószínűleg több oszlop szerint is rendeznünk kell az információkat a megfelelő végeredményhez.)
   A megoldás itt található: CH05_Orders_By_Costumer_And_Date (944 sor).
3. *„Állítsd össze a beszállítók neveinek teljes körű listáját a címeikkel együtt, a beszállító neve szerint rendezve!"*
   A megoldás itt található: CH05_Vendors_Addresses (10 sor).

Entertainment Agency adatbázis

1. *„Add meg az összes megrendelőnk nevét, város szerint rendezve!"*
   (Tipp: valószínűleg szükségünk lesz egy ORDER BY záradék használatára az egyik oszlopon.)
   A megoldás itt található: CH05_Customers_By_City (15 sor).
2. *„Sorold fel az összes előadót és a webhelyeik címét!"*
   A megoldás itt található: CH05_Entertainer_Web_Sites (13 sor).
3. *„Mutasd meg az egyes ügynökeink első hat hónapos teljesítményértékelésének dátumát!"*
   A megoldás itt található: CH05_First_Performance_Review (9 sor).

School Scheduling adatbázis

1. *„Sorold fel a tanári kar tagjait bérezés szerint csökkenő sorrendben!"*
   A megoldás itt található: CH05_Staff_List_By_Salary (27 sor).
2. *„Sorold fel a tanári kar tagjait a telefonszámaikkal együtt!"*
   A megoldás itt található: CH05_Staff_Member_Phone_List (27 sor).
3. *„Írd ki az összes hallgató nevét, és rendezd őket a lakóhelyük városa szerint!"*
   A megoldás itt található: CH05_Students_By_City (18 sor).

Bowling League adatbázis

1. *„Mutasd meg a következő évi tornák dátumát az összes tornahelyszínhez!"*
   A megoldás itt található: CH05_Next_Years_Tourney_Dates (14 sor).
2. *„Írd ki a bajnokság minden résztvevőjének nevét és telefonszámát!"*
   A megoldás itt található: CH05_Phone_List (32 sor).
3. *„Mutasd meg az összes csapat felállását!"*
   A megoldás itt található: CH05_Team_Lineups (32 sor).

6

Az adatok szűrése

„Van hat becsületes szolgám
(Mindent tőlük tudok)
Nevük Mi, Miért és Hogyan
és Hol és Ki és Mikor."
Rudyard Kipling
„I keep six honest-serving men"

A fejezet témakörei

- A lekérdezések finomítása a WHERE segítségével
- A keresési feltételek megfogalmazása
- Több feltétel használata
- Visszatérés a Null-okhoz: egy figyelmeztető megjegyzés
- Feltételek más megfogalmazásban
- Példák
- Összefoglalás
- Önálló feladatok

Az előző két fejezetben megtárgyaltuk, hogy mely módszerekkel kérdezhetjük le az összes adatot egy adott táblából. Szóba került az is, hogy miként alkothatunk olyan kifejezéseket, amelyekkel bővíthetjük vagy szűkíthetjük a keresett információk halmazát. Ebben a fejezetben megmutatjuk, hogyan lehet finomhangolni az információ szűrését a WHERE záradék használatával.

A lekérdezések finomítása a WHERE segítségével

Az eddig használt SELECT utasítások egy adott tábla minden sorát visszaadták az utasítás eredményeként. Ez nagyszerűen működik, ha a tábla összes információjára szükségük van, de mit tegyünk, ha csak azokra a sorokra van szükségünk, amelyek egy adott személyhez vagy helyhez kapcsolódnak, vagy egy adott számértékhez, illetve időszakhoz

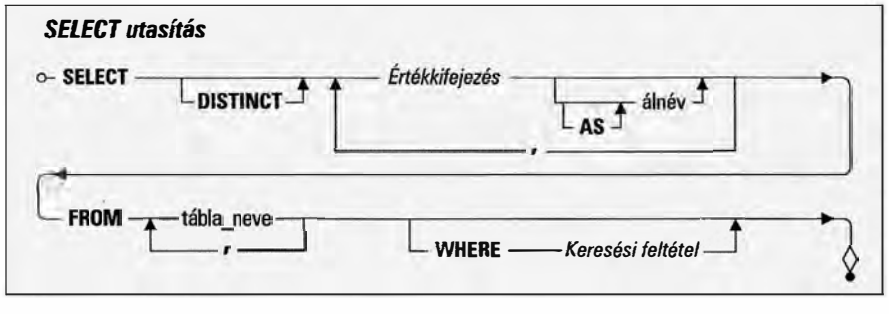
köthetők? Ezek cseppet sem szokatlan kérdések. Valójában ezek bújnak meg azon kérdéseink sokasága mögött, amelyeket nap mint nap felteszünk az adatbázis-kezelőnek. Előfordulhatnak például a következő kérelmek:

„Kik a vásárlóink Seattle-ben?"
„Mutass egy friss listát a bellevue-i alkalmazottainkról a telefonszámukkal együtt!"
„Milyen zenei órákat kínálunk jelenleg?"
„Adj egy listát a három kreditet érő zenei óráinkról!"
„Mely előadóknak van honlapjuk?"
„Adj egy listát a Caroline Coie Trio lekötött fellépéseiről!"
„Adj egy listát azokról a vásárlókról, akik májusban rendeltek valamit!"
„Kiket vettünk fel 1985. május 16-án?"
„Mi a Red Rooster Lanes turnéjának jelenlegi időbeosztása?"
„Mely tekézők vannak az ötös csapatban?"

Az ilyen kérdések megválaszolásához megint bővítenünk kell az SQL-szókincsünket: most egy záradékot (clause) adunk a SELECT utasításhoz, mégpedig a WHERE záradékot.

A WHERE záradék

A WHERE záradék a SELECT utasítással együtt használható, egy táblából lekérdezett adatok szűrésére. A WHERE záradék egy *keresési feltételt* tartalmaz, amit szűrőként használ. Ez a keresési feltétel adja meg a lehetőséget, hogy csak a szükséges sorokat válasszuk ki, illetve hogy kizárjuk azokat, amelyek nem kellenek. Az adatbázis-kezelő rendszerünk a FROM záradékkal kiválasztott logikai tábla minden sorában ellenőrzi a keresési feltételt. A 6.1. ábrán a SELECT utasítás formája látható a WHERE záradékkal.



6.1. ábra

A SELECT utasítás szintaxisdiagramja a WHERE záradékkal

Egy keresési feltétel egy vagy több *állítást* (amelyek egy-egy kifejezésnek felelnek meg) tartalmaz, és egy vagy több értékkifejezést vizsgál, aminek az eredményeképp igaz, hamis vagy ismeretlen értéket ad vissza. Később megtanuljuk, hogy ezek az állítások az AND és OR logikai műveletekkel egy keresési feltételbe is összevonhatók. Amikor a teljes keresési

feltétel igazra értékelődik ki egy sor esetében, akkor az a sor megjelenik az eredményben. Megjegyezzük, hogy ha egy keresési feltétel csak egy állítást tartalmaz, akkor a ***keresési feltétel*** (search condition) és az *állítás* (predikátum, predicate) azonos fogalmak.

Az 5. fejezetben szerepelt, hogy az értékkifejezések oszlopneveket, literális értékeket, függvényeket vagy más értékkifejezéseket tartalmazhatnak. Amikor megfogalmazunk egy állítást, általában egy olyan oszlopra vonatkozó értékkifejezést használunk, ami a FROM záradékkal megadott táblákban található.

A legegyszerűbb és valószínűleg a legtöbbször használt állítás egy értékkifejezést (egy oszlopot) hasonlít össze egy másikkal (egy literálissal). Például ha csak azokat a sorokat szeretnénk látni a Customers (Vásárlók) táblából, ahol a vásárló vezetékneve Smith, megfogalmazhatunk egy állítást, ami összehasonlítja a vezetéknév oszlopát a literális „Smith" értékkel.

SQL
```
SELECT CustLastName
FROM Customers
WHERE CustLastName = 'Smith'
```

Az állítás a WHERE záradékban megegyezik azzal, mintha a következő kérdést tennénk fel minden egyes sornak a Customers táblában: „A vásárló vezetékneve azonos a 'Smith'-szel?". Ha a Customers tábla egy adott sorában erre a kérdésre igen (igaz) a válasz, akkor az a sor szerepelni fog az eredményben.

Az SQL-szabvány tizennyolc állítást határoz meg, de csak az öt legfontosabbat tárgyaljuk ebben a fejezetben, amelyek a következők: összehasonlítás, BETWEEN, IN, LIKE és IS NULL.

Összehasonlítás	A hat összehasonlító műveleti jellel (=, <>, <, >, <=, >=) egy értékkifejezést egy másik értékkifejezéssel hasonlíthatunk össze.
BETWEEN (Tartomány)	A BETWEEN állítás segítségével eldönthetjük, hogy egy adott értékkifejezés értékek egy adott tartományába esik-e. A tartományt két értékkifejezéssel lehet megadni, amelyeket az AND kulcsszóval kötünk össze.
IN (Tagság)	Segítségével eldönthető, hogy az adott értékkifejezés megegyezik-e egy bizonyos listában szereplő elemek valamelyikével.
LIKE (Mintaillesztés)	A LIKE állítással eldönthető, hogy egy karakterlánc formátumú értékkifejezés megegyezik-e egy adott karakterlánc mintájával.
IS NULL	Az IS NULL állítást annak eldöntésére használhatjuk, hogy egy értékkifejezés Null értékű-e.

Megjegyzés *Nem érdemes a többi tizenhárom, az SQL-szabványban szereplő állítással foglalkozni. Tizenegyhez ugyanis nem találtunk semmiféle kereskedelmi megvalósítást, a maradék kettőt pedig – Quantified és EXISTS – a 11. fejezetben tárgyaljuk.*

A WHERE záradék használati területei

Mielőtt felfedezzük az SQL-szabványban meghatározott alapvető állításokat, nézzünk megint egy egyszerű példát a WHERE záradék használatára. Az alábbiakban részletesen leírjuk, hogy milyen lépések során lehet felépíteni a lekérdezést.

Megjegyzés *Ebben a fejezetben végig a „Kérelem – Fordítás – Tisztázás – SQL" módszert alkalmazzuk, amit a 4. fejezetben vezettünk be.*

Tegyük fel, hogy a következő kérdésre szeretnénk választ kapni az adatbázisunk segítségével:

„Hogy hívják a Washington államban élő vásárlóinkat?"

Amikor ilyen kérelemhez állítunk össze fordítási utasítást, meg kell próbálnunk minél világosabban és átláthatóbban megfogalmazni, hogy mely információkra van szükségünk. Több munkába fog kerülni a kérelem átfogalmazása, mint ahogy eddig megszoktunk, de az eredmény indokolni fogja ezt a többletmunkát. A feltett kérdést így fogalmazhatjuk át:

Fordítás Select first name and last name from the customers table for those customers who live in Washington State
(Válaszd ki azoknak a vásárlóknak a vezeték- és keresztnevét a vásárlók táblájából, akik Washington államban laknak.)

Ezt az állítást a megszokott módon tisztázzuk le, de most két további feladatot is elvégzünk. Először keressünk olyan szavakat és kifejezéseket, amelyek valamilyen megszorításra utalnak. Biztos találatnak számítanak a „hol" és a „ki", illetve a „számára" és a „-nak/nek" ragokra végződő szavak. Szerepeljen itt néhány példa ezekre a keresett kifejezésekre:

„...akik Bellevue-ben élnek!"
„...mindenkit, akinek az irányítószáma 98125!"
„...akik májusban adtak le rendelést!"
„...a kaliforniai beszállítókhoz tartozókat!"
„...akiket 1985. május 16-án vettek fel!"
„...ahol a körzetszám 425!"
„...a Mike Hernandezhez tartozókat!"

Amikor ilyen megszorítást találunk, készen állunk a második feladatra. Tanulmányozzuk a kifejezést, és próbáljuk meghatározni, hogy melyik oszlopban, milyen értéket és hogyan kell keresni. A válaszaink ezekre a kérdésekre segítenek majd megalkotni a keresési feltételeket a WHERE záradék számára. Alkalmazzuk ezeket a kérdéseket a lefordított állításra:

Melyik oszlopban kell keresni? **State (Állam)**
Milyen értéket kell keresni? **„WA"**
Hogyan kereshetünk az oszlopban? Az **„egyenlő" műveleti jellel**

A válasz megtalálásához ismernünk kell a tábla szerkezetét. Ha szükséges, készítsünk másolatot a tábla szerkezetéről, mielőtt nekiállunk megválaszolni ezeket a kérdéseket.

Megjegyzés *Néha a válaszok ezekre a kérdésekre nyilvánvalóak, néha azonban csak rejtett utalásokat találunk. Ebben a fejezetben a példák során megmutatjuk, hogyan választhatjuk ki és fejthetjük meg a helyes válaszokat.*

Miután megválaszoltuk a fenti kérdéseket, használjuk fel őket a megfelelő feltételek meghatározásához. A következő lépésben kihúzzuk az eredeti megszorítást, és a WHERE szóra, valamint az imént megalkotott keresési feltételre cseréljük le. A Tisztázás után így néz ki az állításunk:

Tisztázás Select first name ~~and~~ last name from ~~the~~ customers ~~table for those customers who live in~~ where state ~~is equal to~~ = 'WA' ~~Washington State~~

Ezt utána egy rendes SELECT utasításba írjuk át:

SQL
```
SELECT CustFirstName, CustLastName
FROM Customers
WHERE CustState = 'WA'
```

A SELECT utasítás eredményében csak azok a vásárlók fognak szerepelni, akik Washington államban laknak.

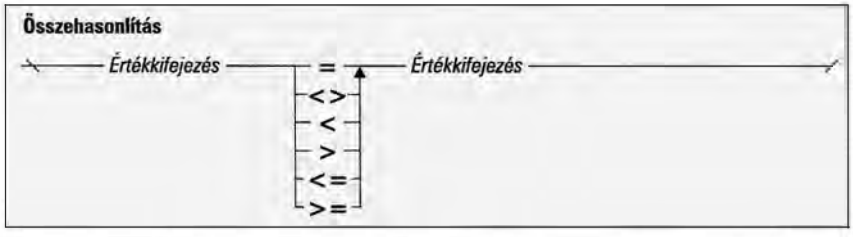
A WHERE záradék meghatározásáról mindössze ennyit kell tudnunk. Ahogy ennek a résznek az elején is említettük, csupán meg kell fogalmaznunk a megfelelő keresési feltételeket, és elhelyezni ezeket a WHERE záradékban. Az igazi munkát azonban a keresési feltételek megfogalmazása jelenti.

A keresési feltételek megfogalmazása

Most hogy láttuk, miként épül fel egy egyszerű WHERE záradék, vessünk közelebbi pillantást az öt alapvető állításra.

Összehasonlítás

A leggyakrabban használt feltétel az összehasonlító állítás, amellyel két értékkifejezést hasonlíthatunk össze. Ahogy a 6.2. ábrán látható, hatféle összehasonlítást határozhatunk meg az összehasonlító műveleti jelekkel.



6.2. ábra

Az összehasonlító feltétel szintaxisdiagramja

=	egyenlő	<	kisebb	<=	kisebb vagy egyenlő
<>	nem egyenlő	>	nagyobb	>=	nagyobb vagy egyenlő

Karakterláncok összehasonlítása: vigyázat!

Könnyű összehasonlítani a számértékű vagy dátum és idő formátumú adatokat, de a karakterláncok összehasonlításával vigyázni kell. Például nem azt az eredményt kaphatjuk, amire számítunk, ha összehasonlítjuk a ránézésre hasonló „Mike" és „MIKE" karakterláncokat. A karakterláncok összehasonlításában a döntő tényező a használt adatbázisrendszer karaktersorrendje. Ez a sorrend dönti el a karakterláncok sorrendjét is, és a többi összehasonlító feltételre is hatással van.

Mivel az SQL nyelvet a különféle gyártók eltérő rendszerekre és helyi nyelvekhez igazodva valósítják meg, az SQL-szabvány nem határoz meg semmilyen karaktersorrendet. A karakterek sorrendje a „legkisebbtől" a „legnagyobbig" a használt adatbázisrendszeren és annak beállításain múlik.

Sok adatbázis-kezelő rendszer az ASCII karaktersorrendet használja, ami a számokat a betűk elé helyezi, és a nagybetűk a kisbetűk előtt szerepelnek. Ha az adatbázisrendszerünk az ASCII sorrendet támogatja, akkor a karakterek sorrendje a következő, a legkisebbtől a legnagyobbig:

...0123456789...ABC...XYZ...abc...xyz...

Néhány rendszeren viszont a kis- és nagybetűk megegyeznek. Ilyen esetben a kis *a* betű azonosnak tekintendő a nagy *A* betűvel. Ha az adatbázisrendszerünk ezt a sorrendet használja az ASCII sorrendet alapul véve, a sorrend a következőképp alakul:

...0123456789...{Aa}{Bb}{Cc}...{Xx}{Yy}{Zz}...

A kapcsos zárójelekben ({}) lévő karakterek azonosnak számítanak, mivel nem teszünk különbséget a kis- és nagybetűk között, és ezt a sorba rendezésnél is figyelembe kell venni.

Az IBM rendszereken futó adatbázis-kezelő rendszerek az IBM szabadalmaztatott EBCDIC karaktersorrendjét használják. Egy olyan adatbázisrendszerben, amely EBCDIC sorrendet használ, először jönnek a kisbetűk, utánuk a nagybetűk és végül a számok. Ha az adatbázisrendszerünk az EBCDIC-et támogatja, a karakterek sorrendje a következő:

...abc...xyz...ABC...XYZ...0123456789...

Mindezt jobban átláthatjuk, ha megnézzük a következő mintaoszlopon, hogy miként befolyásolják a különböző karaktersorrendek a kisebb, a nagyobb vagy az egyenlő relációkat az adatbázisban. A táblában az értékeket az ASCII karakterkészlet szerint rendeztük sorba, figyelve a kis- és nagybetűk közti különbségre (előbb jönnek a számok, majd a nagybetűk, végül a kisbetűk):

Company Name
3rd Street Warehouse
5th Avenue Market
Al's Auto Shop
Ashby's Cleaners
Zebra Printing
Zercon Productions
allegheny & associates
anderson tree farm
zorn credit services
ztech consulting

Most hagyjuk el a kis- és nagybetűk közti különbséget, és nézzük, mi lesz az eredménye:

Company Name
3rd Street Warehouse
5th Avenue Market
Al's Auto Shop
allegheny & associates
anderson tree farm
Ashby's Cleaners
Zebra Printing
Zercon Productions
zorn credit services
ztech consulting

Végül nézzük meg, hogy ezek az adatokat hogyan rendezhetjük sorba egy IBM rendszeren az EBCDIC sorrendet használva (kisbetűk, nagybetűk és számok):

Company Name
allegheny & associates
anderson tree farm
zorn credit services
ztech consulting
Al's Auto Shop
Ashby's Cleaners
Zebra Printing
Zercon Productions
3rd Street Warehouse
5th Avenue Market

Váratlan eredményt kaphatunk akkor is, ha olyan, eltérő hosszúságú karakterláncokat próbálunk összehasonlítani, mint a „John" és a „John " vagy a „Mitch" és a „Mitchell". Szerencsére az SQL-szabvány pontosan leírja, mi a teendő ekkor. Mielőtt az adatbázis-kezelő rendszer összehasonlítaná a két nem azonos hosszúságú karakterláncot, a rövidebbet ki kell egészítenie az *alapértelmezett kitöltő karakterrel* annyira, hogy a hosszuk megegyezzen. (Az alapértelmezett kitöltő karakter a legtöbb rendszeren a szóköz.) Ezután az adatbázisrendszer a beállított karaktersorrendet figyelembe véve eldönti, hogy így azonos-e a két karakterlánc. Ennek eredményeképpen a „John" és a „John " egyezni fog (a kiegészítés után), de a „Mitch " és a „Mitchell" nem.

Megjegyzés *Néhány rendszer eltér az SQL-szabványtól abban, hogy figyelmen kívül hagyja a záró üres karaktereket, ahelyett hogy kibővítené a rövidebb karakterláncot. Ezért a „John" és a „John " bizonyos rendszerekben megegyeznek, de más indokok miatt – mivel a második elemnél a záró üres karakterek összehasonlítására nem kerül sor. Vizsgáljuk meg, hogy az adatbázisrendszerünk hogyan működik, azaz próbáljuk ki, hogy miként kezeli ezeket az összehasonlításokat, és hogy az elvárt eredményt kapjuk-e.*

Zárszóként még annyit, hogy mindig olvassunk utána a kisbetűk, nagybetűk és számok sorrendjének az adatbázisrendszerünk kézikönyvében.

Egyenlőség és egyenlőtlenség

Ugyan már láttunk rá példákat, de vizsgáljunk meg figyelmesebben egy egyenlőségi feltételt az „egyenlő" műveletei jel használatával! Tegyük fel, hogy a következő kérelmet szeretnénk az adatbázishoz intézni:

„Mutasd meg azoknak az ügynököknek a vezeték- és keresztnevét, akiket 1977. március 14-én vettek fel!"

Mivel egy bizonyos felvételi dátum szerint keresünk, a szükséges információ lekéréséhez az egyenlőségi összehasonlítást használhatjuk, az „egyenlő" műveleti jellel. Ha most alkalmazzuk a fordítási eljárást erre a kérelemre, megkapjuk a szükséges SELECT utasítást.

Fordítás Select first name and last name from the agents table for all agents hired on March 14, 1977
(Válaszd ki azoknak az ügynököknek a vezeték- és keresztnevét az ügynökök táblájából, akiket 1977. március 14-én vettek fel.)

Tisztázás Select first name ~~and~~ last name from ~~the~~ agents ~~table for all agents hired on~~ where date hired = ~~March 14, 1977~~ '1977-03-14'

SQL
```
SELECT AgtFirstName, AgtLastName
FROM Agents
WHERE DateHired = '1977-03-14'
```

Ebben a példában egy bizonyos oszlopban lévő értékek között kerestük az adott dátum értékét. Lényegében egy *belefoglaló* folyamatot hajtottunk végre – egy adott sor az Agents (Ügynökök) táblából belekerül az eredmények közé, de *csak akkor*, ha a DateHired (Felvétel dátuma) oszlop értéke az adott dátum értékével azonos. De mi a teendő, ha pont az ellenkezőjét szeretnénk, és *kizárni* kívánunk adott sorokat az eredményből? Ebben az esetben a „nem egyenlő" műveleti jelet használhatjuk az összehasonlítás folyamán.

Tegyük fel, hogy a következő kérelmet szeretnénk az adatbázishoz intézni:

„Add meg az összes beszállító nevét és telefonszámát, kivéve azokét, akik itt vannak Bellevue-ben!"

Könnyen kitalálható, hogy ki kell zárnunk az eredményből azokat a beszállítókat, akiknek a telephelye Bellevue-ben található, és ehhez egy „nem egyenlő" feltételt kell keresni. A „kivéve" szó biztos jele annak, hogy egy „nem egyenlő" feltétel pont jó lesz. Ezt észben tartva lássuk a kérelem lefordítását:

Fordítás Select vendor name and phone number from the vendors table for all vendors except those based in 'Bellevue'
(Válaszd ki a beszállítók nevét és telefonszámát a beszállítók táblájából, kivéve azokat, akiknek a telephelye „Bellevue"-ben található.)

Tisztázás Select vendor name ~~and~~ phone number from ~~the~~ vendors ~~table for all vendors except those based in~~ where city <> 'Bellevue'

SQL
```
SELECT VendName, VendPhone
FROM Vendors
WHERE VendCity <> 'Bellevue'
```

Megjegyzés *Az SQL-szabvány a <> jelet használja a „nem egyenlő" műveletre. Számos RDBMS (relációs adatbázis-kezelő rendszer) program egyéb jelöléseket is kínál, például a != (ezt a Microsoft SQL Server és a Sybase támogatja) és a ¬= (ezt az IBM DB2 terméke támogatja) jeleket. Olvassunk utána az adatbázisrendszerünk kézikönyvében, hogy milyen jelöléseket használhatunk erre a műveletre.*

Ezzel az egyszerű feltétellel hatékonyan kizártuk a bellevue-i beszállítókat az eredményből. Később ebben a fejezetben megmutatunk egy másik módszert is arra, hogy miként zárhatunk ki sorokat az eredmények közül.

Kisebb mint és nagyobb mint

Sokszor olyan sorokra van szükségünk, ahol egy oszlopban lévő bizonyos adat nagyobb vagy kisebb, mint az összehasonlítandó érték. Ez a fajta összehasonlítás magába foglalja a „kisebb" (<), „kisebb vagy egyenlő" (<=), „nagyobb" (>) és „nagyobb vagy egyenlő" (>=) összehasonlító műveleteket. Az összehasonlítandó adat típusa határozza meg a fenti műveletek esetében a relációkat.

KARAKTERLÁNCOK	Ez az összehasonlítás eldönti, hogy az első értékkifejezés megelőzi-e (<) vagy követi-e (>) a második értékkifejezést a használt adatbázisrendszer karaktersorrendje szerint. Például az a < c úgy értelmezhető, hogy „Az *a* megelőzi a *c*-t?" A karaktersorrendekről lásd az előző, *Karakterláncok összehasonlítása: vigyázat!* című részt.
SZÁMOK	Ez az összehasonlítás eldönti, hogy az első értékkifejezés kisebb (<) vagy nagyobb (>), mint a második értékkifejezés. Például a *10* > *5* úgy értelmezhető, hogy „A *10* nagyobb, mint az *5*?"
DÁTUMOK/IDŐPONTOK	Ez az összehasonlítás eldönti, hogy az első értékkifejezés korábban (<) vagy később (>) van-e, mint a második értékkifejezés. Például a *'2007-05-16'* < *'2007-12-15'* úgy értelmezhető, hogy „2007. május 16-a korábban van-e, mint 2007. december 15-e?" A dátumok és időpontok feldolgozására időrendi sorrendben kerül sor.

Nézzük meg, hogy használhatjuk ezeket az összehasonlító állításokat egy kérdés megválaszolásához:

> *„Van olyan rendelésünk, amelynél a szállítási dátum véletlenül a megrendelés előtti időpont?"*

Ehhez a „kisebb mint" műveletet fogjuk használni, mert azt szeretnénk megtudni, hogy van-e olyan szállítási időpont, ami megrendelés előtti dátumot jelöl. Íme a fordítás:

Fordítás Select order number from the orders table where the ship date
is earlier than the order date
(Válaszd ki azoknak a megrendeléseknek a számát a megrendelések táblájából, ahol a szállítási dátum korábbi, mint a megrendelési dátum.)

Tisztázás Select order number from ~~the~~ orders ~~table~~ where ~~the~~ ship date ~~is earlier than the~~ < order date

SQL
```
SELECT OrderNumber
FROM Orders
WHERE ShipDate < OrderDate
```

A SELECT utasítás eredménye csak azokat a sorokat fogja tartalmazni, amelyek megfelelnek a keresési feltételnek.

A következő példához egy „nagyobb mint" műveletet kell segítségül hívni:

„Van olyan óra, ami több mint négy kreditet ér?"

Fordítás Select class ID from the classes table for all classes that earn
more than four credits
(Válaszd ki azoknak az óráknak a sorszámát az órák táblájából, amelyek több mint négy kreditesek.)

Tisztázás Select class ID from ~~the~~ classes ~~table for all classes that earn more than four~~ where credits > 4

SQL
```
SELECT ClassID
FROM Classes
WHERE Credits > 4
```

A SELECT eredményei között csak azok az órák fognak szerepelni, amelyek öt vagy annál több kreditet érnek, például az Intermediate Algebra (Középfokú algebra) vagy az Engineering Physics (Kísérleti fizika).

Most nézzünk olyan példákat, ahol az értékek nem csak szigorúan kisebb vagy nagyobb volta érdekel minket, hanem egyenlők is lehetnek az összehasonlítandó értékekkel.

„Mutasd meg azoknak a nevét, akiket 1989. január 1-e óta vettünk fel!"

Itt egy „nagyobb vagy egyenlő" összehasonlítást használunk, mert az összes felvételi dátum érdekel minket 1989. január 1-e és a mai nap között, *beleértve* azokat is, akiket azon a napon vettünk fel. A fordítás közben figyeljünk oda, hogy minden szükséges oszlopot belefoglaljunk a lekérdezésbe:

Fordítás Select first name and last name as EmployeeName
from the employees table
for all employees hired since January 1, 1989

(Válaszd ki azoknak az alkalmazottaknak a vezeték- és keresztnevét EmployeeName néven az alkalmazottak táblájából, akiket 1989. január 1-e óta vettünk fel.)

Tisztázás Select first name ~~and~~ || ' ' || last name as EmployeeName from ~~the~~ employees ~~table for all employees hired since~~ where date hired >= ~~January 1, 1989~~ '1989-01-01'

SQL
```
SELECT FirstName || ' ' || LastName
   AS EmployeeName
FROM Employees
WHERE DateHired >= '1989-01-01'
```

Egy másik kérelem, amit az adatbázisrendszerhez intézhetünk:

„Mutasd meg azoknak a termékeknek a listáját, amelyeknek a fogyasztói ára legfeljebb ötven dollár!"

A kérelemből azonnal következik, hogy egy „kisebb vagy egyenlő" összehasonlítást kell használnunk. Ezzel biztosítjuk, hogy a SELECT utasítás csak azokat a termékeket adja vissza, amelyeknek az ára pontosan az egy centtől ötven dollárig terjedő tartományban található. Fordítsuk le a kérelmet:

Fordítás Select product name from the products table for all products with a retail price of fifty dollars or less
(Válaszd ki azoknak a termékeknek a neveit a termékek táblájából, amelyek fogyasztói ára ötven vagy annál kevesebb dollár.)

Tisztázás Select product name from ~~the~~ products ~~table for all products with a~~ where retail price ~~of~~ <= 50 ~~fifty dollars or less~~

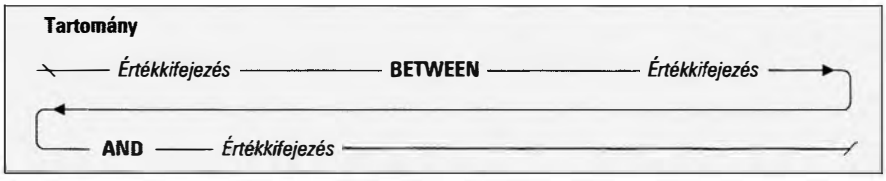
SQL
```
SELECT ProductName
FROM Products
WHERE RetailPrice <= 50
```

Az eddigi példák csak egyféle összehasonlítást alkalmaztak. Később ebben a fejezetben mutatunk példákat az AND és az OR művelettel összefűzött feltételekre is.

Tartomány

Egy értékkifejezést egy tartományfeltétellel adott értékek egy tartományához is hasonlíthatjuk. A 6.3. ábrán látható ennek a feltételnek az utasításformája.

A tartományfeltétel egy adott értékkifejezést hasonlít össze két másik értékkifejezés által meghatározott tartománnyal. A BETWEEN ... AND állítás adja meg a tartományt, ahol a második értékkifejezés jelöli a tartomány alsó határát, a harmadik pedig a felső határát. Az alsó és a felső határ is a tartomány része. Egy sor csak akkor kerül bele az eredménybe, ha az első értékkifejezés az előírt tartományba esik.



6.3. ábra

A tartományfeltétel szintaxisdiagramja

Van azonban egy kis „probléma" a BETWEEN ... AND használatával. Az SQL-szabvány valójában kétféle BETWEEN-összehasonlítást határoz meg: ASYMMETRIC (aszimmetrikus) és SYMMETRIC (szimmetrikus). Az alapértelmezett ASYMMETRIC azt okozza, hogy a Value1 BETWEEN Value2 AND Value3 azonos lesz a Value1 >= Value2 AND Value1 <= Value3 kifejezéssel. Ez azt jeleni, hogy az állítás akkor használható, ha a Value2 kisebb, mint a Value3, vagy egyenlő azzal. Vegyük például az alábbi kifejezést:

```
MyColumn BETWEEN 5 AND 10
```

Az SQL-szabvány szerint ezt a kifejezést így kell értelmezni:

```
MyColumn >= 5 AND MyColumn <= 10
```

Azaz ha a nagyobb értéket tesszük az első helyre...

```
MyColumn BETWEEN 10 AND 5
```

...az az SQL-szabvány értelmében megegyezik ezzel a kifejezéssel:

```
MyColumn >=10 AND MyColumn <= 5
```

Márpedig ez a kifejezés soha nem lehet igaz! (Az oszlop értéke nem tud egyszerre nagyobb lenni 10-nél vagy egyenlő azzal, *és* kisebb lenni 5-nél vagy egyenlő 5-tel.) Ennek ellenére néhány adatbázis-kezelő rendszer megengedi, hogy a Value2 nagyobb legyen, mint a Value3 – ami megegyezik a SYMMETRIC kulcsszó használatával az SQL-szabványban. (Egyelőre nem tudunk olyan megvalósításról, ami támogatná az ASYMMETRIC és SYMMETRIC kulcsszavakat.) Olvassunk utána az adatbázisrendszerünk kézikönyvében ezeknek a részleteknek.

A következő néhány példa bemutatja a tartományfeltétel használatát.

„A tanári kar mely tagjait vettük fel 1986 júliusában?"

A tartományfeltétel itt alkalmazható, mivel adott dátumok között felvettek nevei érdekelnek minket (ebben az esetben az 1986. július 1. és 1986. július 31. között felvetteké). Hajtsuk végre a fordítást ezen a kérelmen, és írjuk meg a megfelelő SELECT utasítást:

Fordítás Select first name and last name from the staff table where the date hired is between July 1, 1986, and July 31, 1986
(Válaszd ki a tanárok táblájából azoknak a vezeték- és keresztnevét, akiket 1986. július 1. és 1986. július 31. között vettünk fel.)

Tisztázás Select first name ~~and~~ last name from ~~the~~ staff ~~table~~ where ~~the~~ date hired ~~is~~ between ~~July 1, 1986~~ '1986-07-01' and ~~July 31, 1986~~ '1986-07-31'

SQL
```
SELECT FirstName, LastName
FROM Staff
WHERE DateHired
   BETWEEN '1986-07-01' AND '1986-07-31'
```

Figyeljük meg, hogy a dátumokat hónap és nap szerint részletezve tüntettük fel a lefordított kérelemben. Használjuk ezt a módszert a fordításra, hogy minél pontosabb kérelmet kapjunk, és a megfelelő SELECT utasítást fogalmazhassuk meg.

A tartományfeltételt egy karakterláncra is hatékonyan alkalmazhatjuk, amint az a következő példában is látható:

„Adj egy listát azokról a hallgatókról – a telefonszámukkal együtt –, akiknek a vezetékneve B-vel kezdődik!"

Fordítás Select last name, first name, and phone number
from the students table
for all students whose last name begins with the letter 'B'
(Válaszd ki azoknak a hallgatóknak a vezeték- és keresztnevét a telefonszámukkal együtt a hallgatók táblájából, akiknek a vezetékneve B-vel kezdődik.)

Tisztázás Select last name, first name, ~~and~~ phone number
from ~~the~~ students ~~table~~
~~for all students whose name begins with the letter 'B'~~
where last name between 'B' and 'Bz'

SQL
```
SELECT StudLastName, StudFirstName, StudPhoneNumber
FROM Students
WHERE StudLastName BETWEEN 'B' AND 'Bz'
```

Amikor egy karakterlánchoz adunk meg tartományfeltételt, alaposan gondoljuk át, hogy milyen értékeket szeretnénk belefoglalni. Ehhez a feladathoz például háromféle megoldás is elképzelhető a kezdő és a záró értékeket tekintve, és az eredmény igencsak különböző lesz!

BETWEEN 'A' AND 'C' Feltehetőleg nem sokszor fordul elő, hogy az 'A' betűvel jelöljük meg a tartomány kezdetét, hiszen akkor az összes 'A' betűvel kezdődő név benne lenne az eredményben. Ennek ellenére ez igen gyakori hiba.

BETWEEN 'B' AND 'C'	Ilyen módon jelölve a kezdő és záró értékeket ebben a példában valószínűleg a megfelelő eredményt kapjuk vissza, de váratlan eredményt is kaphatunk, az összehasonlított adatoktól függően. Ne felejtsük el, hogy a BETWEEN művelet a kezdő és a záró értéket is *belefoglalja* a tartományba. Ez alapján a példánkban, ha egy hallgató vezetékneve a 'C' betű lenne, akkor ő is szerepelne az eredményben.
BETWEEN 'B' AND 'Bz'	Ez a legpontosabb jelölése a kezdő és záró értékeknek – a legtöbb esetben így megkapjuk a szükséges eredményt. A pontos határok megállapításához végsősoron ismernünk és értenünk kell az adatainkat.

Még egy dolog, mielőtt befejezzük a BETWEEN tárgyalását. Figyeljük meg a korábbi 6.3. ábrát, ahol az látható, hogy egy értékkifejezés nem csak a BETWEEN záradékban használható, hanem az első értékként is. Már volt szó róla, hogy egy értékkifejezés lehet egyszerűen egy oszlop neve, vagy egy egyszerű literális érték, de lehet egy összetett karakter, egy matematikai kifejezés vagy időpont kifejezése is. Amikor olyan táblánk van, amely két külön oszlopban tartalmazza a tartomány elejét és végét jelölő értékeket (például a StartDate és EndDate oszlopok az Entertainment Agency mintaadatbázis Engagements táblájában), a BETWEEN használható olyan sorok keresésére is, amelyekben egy érték a két oszlop értéke közé esik. Egy példa erre:

„Mutasd meg az összes olyan rendezvényt, ami 2007. október 10-én esedékes!"

Fordítás
Select engagement number, start date, and end date
from the engagements table
for engagements where October 10, 2007,
is between the start date and the end date
(Válaszd ki a rendezvény számát, valamint a kezdő és záró dátumát
a rendezvények táblájából, azoknak a rendezvényeknek az esetében,
amelyeknél 2007. október 10-e a kezdő és a záró dátum közé esik.)

Tisztázás
Select engagement number, start date, ~~and~~ end date
from ~~the~~ engagements ~~table~~
~~for engagements~~ where ~~October 10, 2007 is~~ '2007-10-10'
between ~~the~~ start date and ~~the~~ end date

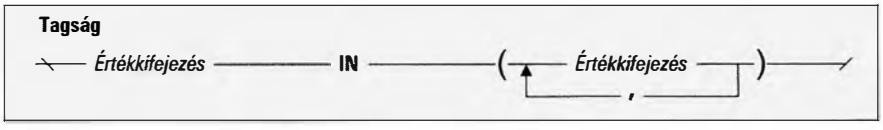
SQL

```
SELECT EngagementNumber, StartDate, EndDate
FROM Engagements
WHERE '2007-10-10' BETWEEN StartDate AND EndDate
```

Eddig arra láttunk példákat, hogy miként szűkíthetünk egy lekérdezést adatok széles tartományára, illetve bizonyos szűkebb adattartományokra. Most nézzük meg, hogyan finomíthatunk tovább egy lekérdezést, ha értékek egy listáját adjuk meg feltételként!

Tagság beállítása

Tagsági feltételt akkor használunk, ha egy értéket egy listában előre megadott értékekhez kívánunk hasonlítani. Amint a 6.4. ábrán látható, a tagság feltétele az IN állítást használja annak eldöntésére, hogy az első értékkifejezés értéke egyezik-e egy zárójellel jelölt lista valamelyik elemével, amit egy vagy több értékkifejezés határoz meg.



6.4. ábra

A tagsági feltétel szintaxisdiagramja

Elméletileg ugyan nincs felső határa a listába foglalható elemeknek, mégis jobb ötlet, ha csak néhányat használunk. Van két másik feltételünk is már, amelyekkel értékek szélesebb tartományát jelölhetjük meg. A tagsági feltételt legjobban egy véges elemszámú listával használhatjuk, ahogy azt látni fogjuk a következő példákban.

Egy lehetséges kérelem az adatbázisunkhoz:

> *„Sorold fel azokat a tekeklubokat, amelyek 2007. szeptember 18-án, október 9-én és november 6-án rendezik a versenyeket!"*

Ez a kérelem éppen alkalmas a tagsági feltételhez, mivel adott dátumokra kérdez rá. Ha a kérelem nem lenne ennyire pontosan megfogalmazva, akkor egy tartományfeltételt használhatnánk. A kérelmet így fordíthatjuk le:

Fordítás Select tourney location from the tournaments table where the tourney date is in this list of dates: September 18, 2007; October 9, 2007; November 6, 2007
(Válaszd ki a verseny helyszínét a versenyek táblájából a következő napokra: 2007. szeptember 18., 2007. október 9., 2007. november 6.)

Tisztázás Select tourney location from ~~the~~ tournaments ~~table~~ where ~~the~~ tourney date ~~is~~ in ~~this list of dates:~~ (~~September 18, 2007;~~ '2007-09-18',~~October 9 2007;~~ '2007-10-09',~~November 6, 2007~~ '2007-11-06')

SQL

```
SELECT TourneyLocation
FROM Tournaments
WHERE TourneyDate
   IN ('2007-09-18', '2007-10-09',
   '2007-11-06')
```

Egy másik kérelem, amihez szintén tagsági feltételt kell használni:

„Mely előadókat képviseljük Seattle-ben, Redmondban és Bothellben?"

Fordítás Select stage name from the entertainers table for all entertainers based in 'Seattle', 'Redmond', or 'Bothell'
(Válaszd ki azoknak az előadóknak a művésznevét az előadók táblájából, akik 'Seattle'-ben, 'Redmond'-ban vagy 'Bothell'-ben dolgoznak.)

Tisztázás Select stage name from ~~the~~ entertainers ~~table for all entertainers based~~ where city in ('Seattle','Redmond', ~~or~~ 'Bothell')

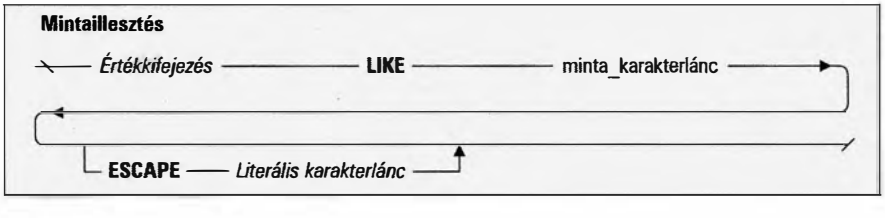
SQL
```
SELECT EntStageName
FROM Entertainers
WHERE EntCity
   IN ('Seattle', 'Redmond', 'Bothell')
```

Vegyük észre, hogy a „vagy" szót használtuk a fordítás folyamán az eredeti kérelemben szereplő „és" szó helyett. Ennek az oka egyszerű: csak egy bejegyzés található az EntCity oszlopban egy adott előadóhoz. Egy adott sor nem tartalmazhatja a Seattle *és* Redmond és Bothell városneveket egyszerre, de tartalmazhatja a Seattle *vagy* Redmond *vagy* Bothell városneveket. Ez nyilvánvalónak tűnhet most, de a pontosan megfogalmazott szavak és kifejezések segítenek jobban átlátható fordítást és tisztázott álkódot alkotni, amelyekkel a kérelemhez legjobban illő SELECT utasításokat kaphatunk. Később ebben a fejezetben látni fogjuk, hogy ezek a kis dolgok mennyire fontosak, ha több feltételt használunk egyszerre.

Az összes eddig tanult feltétel teljes értékkifejezéseket használt, most azonban egy olyan feltételt tanulunk meg, amely részleges értékek használatát is megengedi.

Mintaillesztés

A mintaillesztési feltétel akkor hasznos, ha olyan értékeket kell keresnünk, amelyek megegyeznek egy adott mintakarakterlánccal, vagy amikor a szükséges információnak csak egy része áll rendelkezésre a keresési feltételhez. A 6.5. ábrán láthatjuk ennek a feltételnek az utasításformáját.



6.5. ábra

A mintaillesztési feltétel szintaxisdiagramja

Ez a feltétel egy értékkifejezés értékét veszi, és a LIKE állítással ellenőrzi, hogy illik-e rá az adott mintakarakterlánc. A mintakarakterlánc a szokásos karakterek logikus kombinációjából állhat, és két helyettesítő karaktert tartalmazhat: a százalékjelet (%) és az aláhúzást (_). A százalékjel nulla vagy több tetszőleges karakternek felel meg, az aláhúzás pedig egyetlen tetszőleges karaktert jelent. A mintakarakterlánccal határozzuk meg, hogy milyen adatokat szeretnénk visszakapni a lekérdezés folyamán. A különféle mintakarakterláncokra a 6.1. táblázatban láthatunk példákat.

Megjegyzés *Az egyik legnépszerűbb adatbázis-kezelő rendszer, a Microsoft Office Access, a csillagot (*) használja a százalékjel (%) helyett, illetve a kérdőjelet (?) az aláhúzás (_) helyett. Az Access a kettőskeresztet (#) is támogatja, amellyel számjegyeket helyettesíthetünk egy adott helyen. Ha Microsoft Accesst használunk, ezeket a karaktereket helyettesítsük be a LIKE állítás mintakarakterláncába.*

6.1. táblázat *Példák mintakarakterláncokra*

Mintakarakterlánc	Minta jelentése	A mintára illeszkedő karakterláncok
`'Sha%'`	Bármilyen hosszú karakterlánc lehet, de „Sha"-val kell kezdődnie.	Shannon, Sharon, Shawn
`'%son'`	Bármilyen hosszú karakterlánc lehet, de „son"-ra kell végződnie.	Benson, Johnson, Morrison
`'%han%'`	Bármilyen hosszú karakterlánc lehet, de tartalmaznia kell a „han" szótöredéket.	Buchanan, Handel, Johansen, Nathanson
`'Ro_'`	A karakterlánc csak három karakter hosszú lehet, és „Ro"-val kell kezdődnie.	Rob, Ron, Roy
`'_im'`	A karakterlánc csak három karakter hosszú lehet, és „im"-re kell végződnie.	Jim, Kim, Tim
`'_ar_'`	A karakterlánc csak négy karakter hosszú lehet, és a második és harmadik karakterként az „ar"-t kell tartalmaznia.	Bart, Gary, Mark
`'_at%'`	Bármilyen hosszú karakterlánc lehet, de a második és harmadik karakterként az „at"-ot kell tartalmaznia.	Gates, Matthews, Patterson
`'%ac_'`	Bármilyen hosszú karakterlánc lehet, de az utolsó előtti két karakternek „ac"-nak kell lennie.	Apodaca, Tracy, Wallace

Nézzük meg, hogyan használhatjuk a mintaillesztési feltételt a következő kérelemnél:

„Adj egy listát azokról a vásárlókról, akiknek a vezetékneve 'Mar'-ral kezdődik!"

Az ilyen kérelmeknél jellemzően található olyan kifejezés, ami a mintaillesztés használatára utal. Íme néhány példa az ehhez hasonló kifejezésekre:

„...'Her'-rel kezdődik!"
„...az első két karakter a 'Ba'!"
„...tartalmazza a 'Park' szót!"
„...tartalmazza a 'han' betűsorozatot!"
„...a közepén szerepel az 'ave' szótöredék!"
„...'son'-ra végződik!"
„...a végén 'ez' van!"

Figyelem! *A legtöbb adatbázisrendszerben a karakterláncok összehasonlításánál a kis- és nagybetűk különbözőknek számítanak. Számos nagyobb adatbázisrendszerben a rendszergazdáknak telepítéskor lehetőségük van beállítani, hogy az összehasonlításnál a kis- és nagybetűk azonosnak vagy különbözőnek számítsanak-e. Ha az adatbázisrendszerünk megkülönbözteti a kis- és nagybetűket, akkor a* `LIKE '%chi%'` *meg fogja találni a „roast chicken" kifejezést, de a „Chicken a la King" kifejezést már nem, mivel a mintában található kis 'c' nem egyezik az oszlopban található nagy 'C' betűvel. Olvassunk utána az adatbázisrendszerünk kézikönyvében, hogy kell-e foglalkoznunk a kis- és nagybetűk közötti eltéréssel.*

Láthatjuk, hogy viszonylag könnyű a mintakarakterlánc típusát eldönteni a kérdés alapján. Ha már tudjuk, hogy melyik minta kell nekünk, akkor folytathatjuk a kérelem lefordításával:

Fordítás Select last name and first name from the customers table
where the last name begins with 'Mar'
(Válaszd ki azoknak a vásárlóknak a vezeték- és keresztnevét a vásárlók táblájából, akiknek a vezetékneve a 'Mar' betűkkel kezdődik.)

Tisztázás Select last name ~~and~~ first name from ~~the~~ customers ~~table~~
where ~~the~~ last name ~~begins with~~ like 'Mar%'

SQL
```
SELECT CustLastName, CustFirstName
FROM Customers
WHERE CustLastName LIKE 'Mar%'
```

A SELECT utasítás eredménye olyan neveket fog tartalmazni, mint a Marks, a Marshall, a Martinez és a Marx, mivel csak az első három karaktert vizsgáltuk a vezetéknévből.

Példa egy másik, szintén a mintaillesztési feltétellel megalkotható kérelemre:

„Adj egy listát azokról a beszállítókról, akiknek a címében a 'Forest' szó szerepel az utcanévben!"

Fordítás Select vendor name from the vendors table where the street address contains the word 'Forest'

(Válaszd ki azoknak a beszállítóknak a nevét a beszállítók táblájából, akiknek a címében az utcanév tartalmazza a 'Forest' szót.)

Tisztázás Select vendor name from ~~the~~ vendors ~~table~~ where ~~the~~ street address ~~contains the word~~ like '%Forest%'

SQL
```
SELECT VendName
FROM Vendors
WHERE VendStreetAddress LIKE '%Forest%'
```

Ebben az esetben a beszállítók táblájából egy sor csak akkor kerül be az eredménybe, ha a címben az utcanév olyan, mint a Forest Park Place, a Forest Ridge Avenue, az Evergreen Forest Drive vagy a Black Forest Road.

A mintakarakterlánccal ugyan bármilyen mintára rá tudunk keresni a megfelelő helyettesítő karaktereket használva, de könnyen problémákba ütközhetünk, ha százalékjelre vagy aláhúzásra szeretnénk keresni. Például az MX_445 karakterlánc megkeresése gondot okoz, mert aláhúzás karaktert tartalmaz. Egy ilyen lehetséges problémán a LIKE állítás ESCAPE kapcsolójával lehetünk úrrá, ahogy a 6.5. ábrán láthattuk.

Az ESCAPE kapcsoló lehetővé teszi, hogy kijelöljünk egy literális karakterláncot – amit *védőkarakternek* hívunk –, amivel jelezzük az adatbázis-kezelő rendszernek, hogy a százalékjeleket vagy aláhúzásokat sima karakterként kezelje a mintakarakterláncban. A védőkaraktert az ESCAPE kulcsszó után írjuk, és aposztrófok közé tesszük, ahogy a literális karakterláncokat szoktuk. Ha a mintakarakterláncban egy helyettesítő karakter előtt védőkarakter található, az adatbázisrendszer a helyettesítő karaktert *literálisan* („szó szerint") értelmezi.

Egy lehetséges példa az ESCAPE használatára:

„Mutass egy listát azokról a termékekről, amelyeknek a termékkódja 'G_00'-val kezdődik, és egy számmal vagy betűvel végződik!"

Fordítás Select product name and product code from the products table where the product code begins with 'G_00' and ends in a single number or letter
(Válaszd ki a terméknevet és termékkódot a termékek táblájából azokra a termékekre, amelyeknek a termékkódja a 'G_00' karakterekkel kezdődik, és egy számjeggyel vagy betűvel végződik.)

Tisztázás Select product ~~name and~~ product code from ~~the~~ products ~~table~~ where ~~the~~ product code ~~begins with~~ like 'G_00_' ~~and ends in a single number or letter~~

SQL
```
SELECT ProductName, ProductCode
FROM Products
WHERE ProductCode LIKE 'G\_00_' ESCAPE '\'
```

Látható, hogy ehhez a kérelemhez az ESCAPE kapcsolót kell használni – másképp az adatbázis-kezelő rendszer az aláhúzást helyettesítő karakterként értelmezné. Figyeljük meg, hogy a védőkaraktert belefoglaltuk a Tisztázás pontba is. Ajánlott így letisztázni az álkódot, mert így biztosan emlékezni fogunk, hogy az ESCAPE kapcsolót nem szabad kifelejtenünk a SELECT utasításból.

Ez a SELECT utasítás meg fogja találni az olyan termékkódokat, mint a G_002 vagy a G_00X. Egy vagy két olyan karaktert keresünk, amelyek alapértelmezés szerint helyettesítő karakterek, ezért használnunk *kell* az ESCAPE záradékot. Ha arra keresünk, hogy `LIKE 'G_00_'`, akkor az adatbázis-kezelő rendszer minden olyan sort visszaad eredményként, ahol a termékkód első betűje 'G', valamilyen karakter áll a második helyen (a helyettesítő karakter miatt), nullák vannak a harmadik és negyedik helyen, és valamilyen karakterrel zárul az ötödik helyen. Amikor a „\"-t állítjuk be védőkarakterként, az adatbázisrendszer figyelmen kívül hagyja a védőkaraktert, de az első aláhúzást literálisan értelmezi, nem helyettesítőként. Mivel a második aláhúzás elé nem írtunk védőkaraktert, azt az adatbázisrendszer valódi helyettesítőként értelmezi.

Tartsuk észben, hogy védőkarakterként nem szabad olyan karaktert használnunk, ami a megtalálandó értékekben szerepelhet. Nem jó ötlet az &-t használni védőkarakterként, ha olyan értékekre keresünk, mint a Martin & Lewis, a Smith & Kearns vagy a Hernandez & Viescas. Emlékezzünk arra is, hogy a védőkarakter csak a közvetlenül utána következő karakterre van hatással, de annyi védőkaraktert használhatunk a mintakarakterláncban, amennyire szükség van.

Null

Megtanultuk, hogyan kereshetünk teljes és részleges értékekre, így most rátérhetünk az *ismeretlen* értékekre. Az 5. fejezetben megtanultuk, hogy a Null *nem azonos* a nullával, az egy vagy több üres karaktert tartalmazó karakterlánccal vagy a nulla hosszúságú karakterlánccal (ami olyan karakterlánc, amiben egyetlen karakter sem szerepel), mert ezek mind jelenthetnek valamit a megfelelő környezetben. Azt is megtanultuk, hogy a Null *azonos* a hiányzó vagy ismeretlen értékekkel. Egy értékkifejezés Null értékének meghatározásához a *Null feltételt* használhatjuk, ahogy a 6.6. ábrán láthatjuk.

6.6. ábra

A Null feltétel szintaxisdiagramja

Ez a feltétel veszi az értékkifejezés értékét, és az IS NULL állítással eldönti, hogy Null értékű-e. Ez elég egyszerű művelet. Nézzük meg a következő példák segítségével, hogyan használhatjuk ezt a feltételt.

„Adj egy listát azokról a vásárlókról, akik nem adták meg, hogy melyik megyében laknak!"

Fordítás Select first name and last name as Customer from the customers table where the county name is unspecified
(Válaszd ki azoknak a vásárlóknak a vezeték- és keresztnevét Customers néven a vásárlók táblájából, akik nem adták meg a megyét, ahol laknak.)

Tisztázás Select first name || ' ' || ~~and~~ last name as Customer from ~~the~~ customers ~~table~~ where ~~the~~ county name is null ~~unspecified~~

SQL
```
SELECT CustFirstName || ' ' || CustLastName
   AS Customer
FROM Customers
WHERE CustCounty IS NULL
```

A fenti SELECT utasítás eredményében csak azok a vásárlók fognak szerepelni, akik nem tudták vagy nem emlékeztek, hogy melyik megyében laknak, illetve azok, akik Washington D.C.-ben élnek (mivel Washington az egyetlen város az Egyesült Államokban, amelyik egyik megyébe sem tartozik).

Egy másik lehetséges kérdés:

„Mely rendezvényeknek nincs még szerződési összege?"

Fordítás Select engagement number and contract price from the engagements table for any engagement that does not have a contract price
(Válaszd ki azoknak a rendezvényeknek a számát és szerződési összegét, amelyeknek nincs ára.)

Tisztázás Select engagement number ~~and~~ contract price from ~~the~~ engagements ~~table for any engagement that does not have a~~ where contract price is null

SQL
```
SELECT EngagementNumber, ContractPrice
FROM Engagements
WHERE ContractPrice IS NULL
```

Ha felszínesen foglalkozunk ezzel a kérelemmel, akkor egyszerűnek tűnik – csupán a 0 árú rendezvényeket kell megkeresni. Ezzel azonban vigyázzunk, mert hibás feltevésekhez vezethet. Ha a példában az ügynökség egy reklámcélú rendezvény szerződési összegeként 0-t tüntet fel, akkor a 0 valós, jelentéssel bíró érték. Ezek szerint minden, még meg nem határozott vagy el nem döntött árnak az értéke valójában Null (vagy annak kellene lennie).

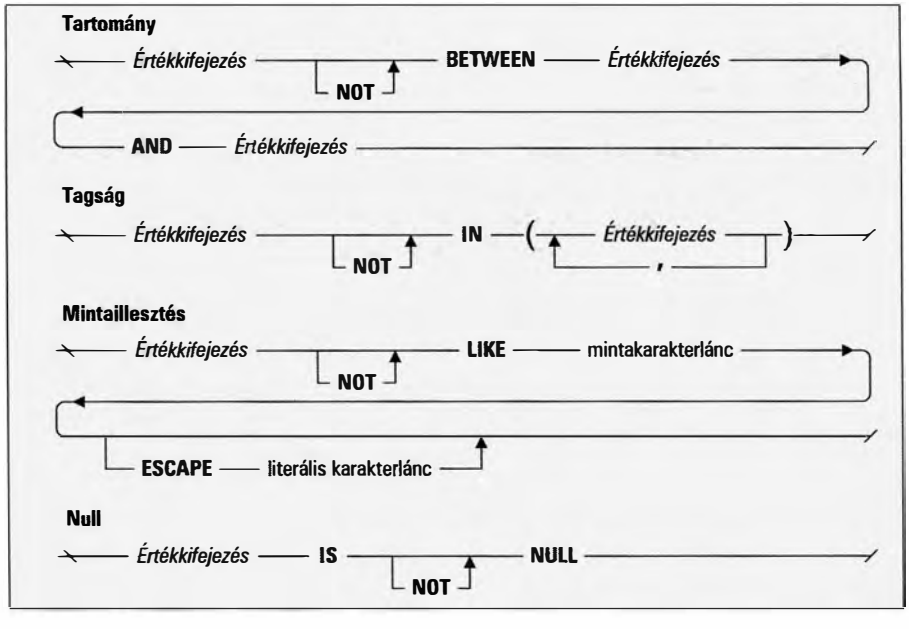
Ez a példa mutatja, hogy értenünk kell a táblákban szereplő adatokat, hogy értelmes, pontos lekérdezéseket írhassunk. Ha végrehajtunk egy SELECT utasítást, és úgy tűnik, hogy a kapott eredmény hibás, ne essünk kétségbe. Az első ötletünk valószínűleg az lesz,

hogy az egész SELECT utasítást újraírjuk, mert azt gondolhatjuk, hogy valamilyen végzetes formai hibát ejtettünk. Mielőtt azonban bármilyen drasztikus lépést tennénk, nézzük át újra az adatokat, amelyekkel éppen dolgozunk, és tudjuk meg, hogy pontosan hogyan használhatjuk azokat. Ha jobban megértettük az adatokat, sokszor észrevesszük majd, hogy csak néhány kisebb javítást kell eszközölnünk a SELECT utasításon a megfelelő adatok lekérdezéséhez.

Megjegyzés *A Null értékek megkereséséhez a Null feltételt kell használnunk. Egy olyan feltétel, mint az* `<Értékkifejezés> = Null`, *érvénytelen, mert egy értékkifejezés értéke nem hasonlítható össze egy olyan dologgal, ami – a meghatározása szerint – ismeretlen.*

Sorok kizárása a NOT művelettel

Eddig arról volt szó, hogy miként tudunk egyes sorokat *belefoglalni az* eredménybe. Most nézzük meg, hogyan lehet *kizárni* sorokat az eredményből a NOT művelet segítségével. Már mutattunk egy egyszerű módszert a sorok kizárására az egyenlőségi összehasonlító feltételben a „nem egyenlő" művelettel. Más feltételeknél is zárhatunk ki sorokat a NOT művelettel. Ahogy a 6.7. ábrán látható, a NOT lehetséges kiegészítője a BETWEEN, IN, LIKE és IS NULL állításoknak. Amikor a feltétel a NOT műveletet is tartalmazza, akkor a SELECT utasítás minden olyan sort el fog dobni, amelyekre az említett állítások bármelyike illeszkedik. Az eredményben szereplő sorok tehát azok lesznek, amelyekre *nem illeszkedik* a feltétel.



6.7. ábra

A NOT művelet szintaxisdiagramja

A következő példák megmutatják, hogyan használhatjuk a NOT műveletet egy keresési feltétel részeként:

„Mutasd meg az összes felvett megrendelést, kivéve azokat, amelyeket júliusban adtak fel!"

Egy ehhez hasonló kérdéshez olyan SELECT utasítást kell írnunk, amely kizárja az adott feltételnek eleget tevő sorokat. Ez sokszor olyan kifejezéseket tartalmaz, amelyek sugallják a NOT művelet használatát. Lássunk néhány ehhez hasonló kifejezést, amellyel találkozhatunk:

„...ami nem 'Her'-rel kezdődik!"
„...amelyek nem az Administrative vagy Personnel osztályokhoz tartoznak!"
„...akiknek van faxszámuk!"
„...akiket június 1. előtt vagy augusztus 31. után vettek fel!"

Néha egy kis elemző munkát kell végezni, hogy jól tudjuk lefordítani a kérdéses kifejezést. Egyes kifejezések, például a fenti példák közül a harmadik, nem indokolja egyértelműen a NOT művelet használatát. Ebben az esetben az a lényeg, hogy ***ki akarjuk zárni*** azokat, akiknek *nincs* faxszámuk. Ahogy ilyen kifejezéseket tartalmazó lekérdezésekkel kezdünk foglalkozni, gyakran előfordul, hogy részletesen elemezni kell a kérelmet, és esetleg újra is kell írni, hogy meghatározzuk az eredményből kizárandó sorokat. Erre nincs egyértelmű alapszabály, de egy kis türelemmel és gyakorlással könnyebben kitalálhatjuk, hogy szükségünk van-e a NOT műveletre egy adott feladatnál.

Miután meghatároztuk, hogy kell-e egyáltalán bizonyos információkat kizárni az eredményből, folytathatjuk a fordítással:

„Mutass egy listát a felvett rendelésekről, kivéve azokat, amelyeket októberben adtak fel!"

Fordítás Select order ID and order date from the orders table where the order date does not fall between October 1, 2007, and October 31, 2007
(Válaszd ki a megrendelés számát és dátumát a megrendelések táblájából, azoknak a rendeléseknek az esetében, amelyeknek a megrendelési dátuma nem 2007. október 1. és 2007. október 31. közé esik.)

Tisztázás Select order ID ~~and~~ order date from ~~the~~ orders ~~table~~ where ~~the~~ order date ~~does~~ not ~~fall~~ between ~~October 1,2007,~~ '2007-10-01' and ~~October 31,~~ 2007 '2007-10-31'

SQL
```
SELECT OrderID, OrderDate
FROM Orders
WHERE OrderDate NOT BETWEEN '2007-10-01'
   AND '2007-10-31'
```

Ezzel a SELECT utasítással olyan eredményt kapunk, amelyben *nincs* rendelés a 2007. október 1. és 2007. október 31. közötti időszakból, de ezen kívül minden megrendelést tartalmazni fog az Orders (Megrendelések) táblából. Az eredmények listáját még tovább szűkíthetjük, ha több feltételt alkalmazunk, úgy, hogy az eredmény csak a 2007-ben feladott rendeléseket tartalmazza (ezt a következő részben tárgyaljuk).

Tegyük fel, hogy a következő kérelmet szeretnénk az adatbázishoz intézni:

„Mutasd meg a kar összes olyan dolgozójának a személyi számát, aki nem egyetemi tanár vagy docens!"

Fordítás Select staff ID and title from the faculty table where the title is not 'professor' or 'associate professor'
(Válaszd ki a tanárazonosítót és a rangot a tanszék táblájából azokra, akiknek a rangja nem 'egyetemi tanár' vagy 'docens'.)

Tisztázás Select staff ID ~~and~~ title from ~~the~~ faculty ~~table~~ where ~~the~~ title ~~is~~ not in ('professor', ~~or~~ 'associate professor')

SQL
```
SELECT StaffID, Title
FROM Faculty
WHERE Title
NOT IN ('Professor', 'Associate Professor')
```

Ebben az esetben ki kell zárni minden olyan tanárt, akinek a rangja megegyezik a kérelemben megfogalmazott valamelyik címmel, ezért a tagsági feltételt alkalmazzuk a NOT művelettel együtt a megfelelő sorok kiválasztásához.

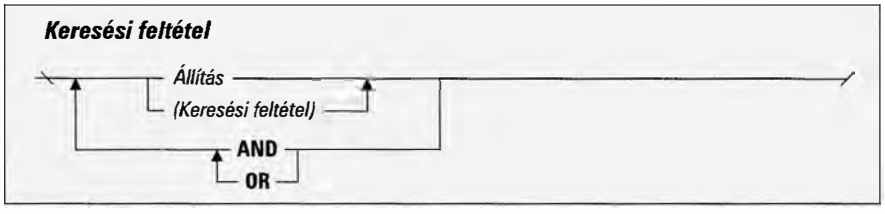
A sorok kizárása az eredményből viszonylag egyszerű műveletté válik, amint hozzászoktunk a kérelem helyzettől függő elemzéséhez és újrafogalmazásához. A folyamat lényege – ahogy eddig is láthattuk – az, hogy meg tudjuk határozni az adott kérelemhez használandó feltétel típusát.

Több feltétel használata

Az eddigi kérdések egyszerűek voltak, és csak egy feltétel kellett a megválaszolásukhoz. Most megnézzük, hogyan válaszolhatunk összetett kérdésekre több feltételt használva. Kezdésként nézzük a következő kérelmet:

„Adj egy listát azoknak a vásárlóknak a vezeték- és keresztnevéről, akik Seattle-ben élnek, és a vezetéknevük 'H' betűvel kezdődik!"

Az eddig tanultak alapján biztosak lehetünk abban, hogy a kérelem egy egyenlőségi összehasonlító és egy mintaillesztési feltételt igényel. Felismertük a szükséges feltételeket, de hogyan vonhatjuk azokat össze egy keresési feltételbe? A választ az SQL-szabvány keresési feltételének utasításformája adja, ahogy a 6.8. ábrán látható.



6.8. ábra

A keresési feltétel szintaxisdiagramja

AND és OR

Két vagy több feltételt az AND és OR műveletekkel vonhatunk össze, és a kérdés megválaszolásához szükséges összes feltétel alkot egyetlen keresési feltételt. Ahogy a 6.8. ábrán is látható, egy már összevont keresési feltételt is össze lehet vonni egyéb feltételekkel, ha a keresési feltételt zárójelbe tesszük. Ez igen összetett WHERE záradékok készítését teszi lehetővé, amelyek pontosan megjelölik az eredménybe bekerülő sorokat.

Az AND használata

Az egyik módszer két vagy több feltétel összevonására az AND művelet használata. Ezt akkor használjuk, ha az összes összevont feltételnek teljesülnie kell egy sorra, hogy az az eredmények közé kerülhessen. Használjuk fel a jelenlegi rész elején szereplő mintakérdést példaként ennek a műveletnek a bemutatására:

„Adj egy listát azoknak a vásárlóknak a vezeték- és keresztnevéről, akik Seattle-ben élnek, és a vezetéknevük 'H' betűvel kezdődik!"

Fordítás Select first name and last name from the customers table where the city is 'Seattle' and the last name begins with 'H'
(Válaszd ki azoknak a vásárlóknak a vezeték- és keresztnevét a vásárlók táblájából, akiknek a városa 'Seattle', és a vezetéknevük 'H' betűvel kezdődik.)

Tisztázás Select first name ~~and~~ last name from ~~the~~ customers ~~table~~ where ~~the~~ city ~~is~~ = 'Seattle' and ~~the~~ last name begins ~~with like~~ 'H%'

SQL
```
SELECT CustFirstName, CustLastName
FROM Customers
WHERE CustCity = 'Seattle'
   AND CustLastName LIKE 'H%'
```

Figyelembe vettük mind az egyenlőségi összehasonlító feltételt, mind a mintaillesztési feltételt, ahogy a kérelem megköveteli, és az AND művelettel biztosítottuk, hogy mindkettő igaz legyen azokra a sorokra, amelyek bekerülnek az eredménybe. Ha egy sorra valamelyik feltétel nem teljesül, az nem fog bekerülni az eredmények közé.

Annyi feltételt használhatunk egyszerre, amennyire csak szükségünk van az adott kérelemhez, csak ne felejtsük el, hogy az AND művelettel összevont összes feltételnek igaznak *kell* lennie egy sorra, hogy az az eredmények között szerepelhessen, és emlékezzünk arra is, hogy a teljes keresési feltételnek igaznak kell lennie egy sorra, hogy azt viszontlássuk az eredményben. Az eredményt két állításnak az AND művelettel történő összevonása esetén a 6.9. ábra mutatja. Ha *valamelyik* kifejezés hamisra értékelődik ki, akkor a sort a lekérdezés nem adja vissza.

	Második kifejezés		
Első kifejezés	**AND**	**Igaz**	**Hamis**
	Igaz	Igaz (Sorok kijelölve)	Hamis (Sorok kihagyva)
	Hamis	Hamis (Sorok kihagyva)	Hamis (Sorok kihagyva)

6.9. ábra

Két állítás összevonásának eredménye az AND művelet használatakor

Az OR használata

A másik lehetőség két vagy több feltétel összevonására az OR művelettel adódik. Ezt akkor használjuk, ha az összevont feltételek közül elég *valamelyiknek* teljesülnie ahhoz, hogy egy sor az eredménybe kerüljön. Lássunk egy példát az OR használatára egy keresési feltételben:

„Mutasd meg azoknak a tanároknak a nevét, városát és államát, akik Seattle-ben élnek, vagy Oregon államból jöttek!"

Fordítás Select first name, last name, city, and state from the staff table
where the city is 'Seattle' or the state is 'OR'
(Válaszd ki azoknak a személyeknek a vezeték- és keresztnevét, városát és államát a tanárok táblájából, akiknek a városa 'Seattle' vagy az állama 'OR'.)

Tisztázás Select first name, last name, city, ~~and~~ state from ~~the~~ staff ~~table~~
where ~~the~~ city ~~is~~ = 'Seattle' or ~~the~~ state ~~is~~ = 'OR'

SQL
```
SELECT StfFirstName, StfLastName, StfCity, StfState
FROM Staff
WHERE StfCity = 'Seattle' OR StfState = 'OR'
```

Ebben az esetben mindkét egyenlőségi összehasonlító feltételt belevettük a lekérdezésbe, és az OR művelettel biztosítottuk, hogy már az *egyik feltétel* teljesülésekor is igaz legyen a kifejezés. Amíg a sorok teljesítik valamelyik feltételt, addig be fognak kerülni az ered-

mények közé. Hogy könnyebben átlássuk, hogyan is működik mindez, nézzük meg a 6.10. ábrát, amelyen azt láthatjuk, hogy milyen eredményt ad, ha két állítást összevonunk az OR művelettel.

6.10. ábra

Két állítás összevonásának eredménye az OR művelet használatakor

Viszonylag könnyű eldönteni, hogy mikor kell az AND műveletet használni a feltételek összevonására, viszont az OR használata már nem ennyire egyértelmű. Például nézzük a következő kérelmet:

> *„Mutass egy listát azoknak a beszállítóknak a nevével és telefonszámával, akiknek a telephelye Washingtonban vagy Kaliforniában található!"*

Először nyilván az AND művelet használatára gondolnánk, mert nyilvánvalónak tűnik a feltétel – azok a beszállítók érdekelnek minket, akiknek a telephelye Washingtonban *és* Kaliforniában található. Ez azonban sajnos tévedés. Ha átgondoljuk, egy beszállító *vagy* Washingtonban, *vagy* Kaliforniában lehet, mert az államok oszlopába csak egyetlen állam nevét lehet beírni. Ugye mennyivel átláthatóbb most a feltétel? A fejezetben korábban már említettük, hogy meg kell szoknunk a kérelmek tanulmányozását és elemzését, ahogy egyre összetettebb kérelmekkel találkozunk. Próbáljuk meg minél ügyesebben megkeresni a kérelemben rejlő feltételeket.

Folytassuk tehát a kérelem lefordításával:

> *„Mutass egy listát azoknak a beszállítóknak a nevével és telefonszámával, akiknek a telephelye Washingtonban vagy Kaliforniában található!"*
>
> Fordítás Select name, phone number, and state from the vendors table
> where the state is 'WA' or 'CA'
> (Válaszd ki azoknak a beszállítóknak a nevét, telefonszámát és államát a beszállítók táblájából, akiknek az állama 'WA' vagy 'CA'.)

Tisztázás Select name, phone number, ~~and~~ state from ~~the~~ vendors ~~table~~
where ~~the~~ state ~~is~~ = 'WA' or state = 'CA'

SQL
```
SELECT VendName, VendPhoneNumber, VendState
FROM Vendors
WHERE VendState = 'WA' OR VendState = 'CA'
```

Mindkét egyenlőségi összehasonlító feltételt belevettük a lekérdezésbe, és az OR műveletel biztosítottuk, hogy legalább az egyik teljesüljön. Figyeljük meg, hogy a „state" (állam) kétszer jelenik meg a Tisztázás és az SQL-utasítás keresési feltételei között. Erre azért van szükség, mert az összehasonlító feltételek utasításformája a következő:

```
Értékkifejezés <összehasonlító művelet> Értékkifejezés
```

Ne felejtsük el, hogy nem hagyhatunk ki semmilyen záradékot, kulcsszót vagy adott kifejezést az utasításból, kivéve, ha az feltételes eleme az utasításnak. Ezért egy olyan feltétel teljesen érvénytelen, mint a `WHERE VendState = 'WA' OR 'CA'`. Megkérdezhetnénk, miért is van ez, de a kifejezésekre alkalmazott műveletek sorrendjének tárgyalásakor – a kiértékelési sorrendnél – ezt részletesen el fogjuk magyarázni.

Ebben az esetben az adatbázis-kezelő rendszerünk a kifejezést szigorúan balról jobbra haladva értelmezi, tehát először a `VendState = 'WA'`-t értékeli ki. Az eredmény minden olyan sorra igaz lesz, ahol az állam Washington, és hamis, ha nem. A következő lépésben az adatbázis-kezelő ezt az igaz vagy hamis eredményt hasonlítja össze az 'OR' segítségével a literális `'CA'` szöveggel – ami viszont nem igaz vagy hamis érték! Az adatbázisrendszer ekkor hibát jelezhet (a `'CA'` – ami egy literális karakterlánc – érvénytelen adattípus az OR művelet számára), vagy csak azokat a sorokat adja vissza, ahol az állam Washington.

Mindig bizonyosodjunk meg arról, hogy a feltételeink teljesen és hibátlanul vannak megfogalmazva. Ha nem így teszünk, akkor a SELECT utasításunk keresési feltétele hibás lehet.

Megjegyzés *Ezt a példát arra használtuk, hogy egy az OR művelet esetében sokszor előforduló hibára hívjuk fel a figyelmet. Viszont ha arra gondolunk, hogy egy tagsági feltételt – például* `WHERE VendState IN ('WA', 'CA')` *– használjunk a kérelemhez, teljesen igazunk van. Néhány esetben nem csupán egy módszer adódik egy feltétel kifejezésére.*

Az AND és az OR együttes használata

Az AND és OR műveleteket különösen bonyolult kérdelmekhez együtt is használhatjuk. Például a következő kérelemhez mindkét műveletet egyszerre kell használni:

„Sorold fel a tanári kar azon tagjainak a nevét, akiknek a körzetszáma 425, és a telefonszámuk 555-tel kezdődik, valamint azoknak a nevét is, akiket 2007. október 1. és december 31. között vettünk fel!"

Mostanra könnyen el tudjuk dönteni, hogy milyen feltételek kellenek ehhez a kérelemhez. Felismerhetjük, hogy háromféle feltételt kell alkalmaznunk: egy egyenlőségi összehasonlító feltételt a körzetszámhoz, egy mintaillesztési feltételt a telefonszámhoz, és egy tartományfeltételt azoknak a tanároknak a megtalálásához, akiket október 1. és december 31. vettünk fel. Már csak azt kell kitalálni, hogy miként vonhatjuk ezeket össze.

Az összehasonító és a mintaillesztési feltételt egy AND művelettel kell egyesítenünk, mert ezek határozzák meg a keresett telefonszámot, és mindkét feltételnek teljesülnie kell ahhoz, hogy egy sor bekerüljön az eredménybe. Az egyesített feltételt utána egy egységként kezeljük, és az OR művelet segítségével összevonjuk a tartományfeltétellel. Így egy sor akkor kerül be az eredménybe, ha *vagy* az egyesített feltételt, *vagy* a tartományfeltételt teljesíti.

A kérelem és a fordítása tehát így fest:

„Sorold fel a tanári kar azon tagjainak a nevét, akiknek a körzetszáma 425, és a telefonszámuk 555-tel kezdődik, valamint azoknak a nevét is, akiket 2007. október 1. és december 31. között vettünk fel!"

Fordítás Select first name, last name, area code, phone number, and date hired from the staff table where the area code is 425 and the phone number begins with 555 or the date hired falls between October 1, 2007, and December 31, 2007
(Válaszd ki azoknak a tanároknak a vezeték- és keresztnevét, körzetszámát, telefonszámát és felvételi dátumát a tanárok táblájából, akiknek a körzetszáma 425, és a telefonszámuk 555-tel kezdődik, vagy a felvételi dátumuk 2007. október 1. és 2007. december 31. közé esik.)

Tisztázás Select first name, last name, area code, phone number, ~~and~~ date hired from ~~the~~ staff ~~table~~ where ~~the~~ area code ~~is~~ = '425' and the phone number ~~begins with~~ like '555%' or ~~the~~ date hired ~~falls~~ between ~~October 1, 2007,~~ '2007-10-01' and ~~December 31, 2007~~ '2007-12-31'

SQL
```
SELECT StfFirstName, StfLastName, StfAreaCode,
   StfPhoneNumber, DateHired
FROM Staff
WHERE (StfAreaCode = '425'
   AND StfPhoneNumber LIKE '555%')
OR DateHired
   BETWEEN '2007-10-01' AND '2007-12-31'
```

Az előző példa jól mutatja, hogyan használhatunk egy keresési feltételt egy másik keresési feltételen belül. Mielőtt lefordítottuk a kérelmet, azt mondtuk, hogy az összehasonlító és a mintaillesztési feltételt egyesítenünk kell az AND művelettel, és egy egységként kell kezelni. Amikor feltételeket egy egységbe vonunk össze, akkor az az egység is keresési feltétel lesz, és a példában látható módon zárójelek közé kell tenni.

Egy másik példa az AND és az OR használatára:

„Mutasd meg azoknak az egyetemi tanároknak és docenseknek a nevét és rangját, akiket 1989. május 16-én vettek fel!"

Fordítás Select first name, last name, title, and date hired from the staff table where the title is 'professor' or 'associate professor' and the date hired is May 16, 1989
(Válaszd ki azoknak a személyeknek a vezeték- és keresztnevét, rangját és felvételi dátumát a tanárok táblájából, akiknek a rangja 'egyetemi tanár' vagy 'docens', és a felvételi dátumuk 1989. május 16.)

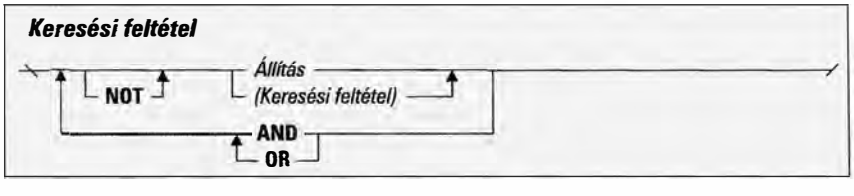
Tisztázás Select first name, last name, title, ~~and~~ date hired from ~~the~~ staff ~~table~~ where ~~the~~ title ~~is~~ = 'professor' or title = 'associate professor' and ~~the~~ date hired ~~is~~ = ~~May 16, 1989~~ '1989-05-16'

SQL
```
SELECT StfFirstName, StfLastName, Title, DateHired
FROM Staff
WHERE (Title = 'Professor' OR Title =
   'Associate Professor') AND DateHired =
   '1989-05-16'
```

Észrevehetjük, hogy az OR-ral összevont két feltételt egy keresési feltételként kezeltük. Ez a példa pusztán megerősíti azt a tényt, hogy egy keresési feltétel megfogalmazható akár az AND, akár az OR művelettel, de ne felejtsük el: a lényeg az, hogy zárójelbe tegyük a keresési feltételt.

Sorok kizárása: második felvonás

Úgy tűnhet, mintha ezzel egyszer már találkoztunk volna... Nincs ok az aggodalomra, a témával már tényleg foglalkoztunk az előző részekben – legalábbis egy szintig. Ebben a fejezetben korábban már megtanultuk, hogy a NOT művelet feltételes kiegészítése a BETWEEN, IN, LIKE és IS NULL állításoknak. De ahogy a 6.11. ábrán is látható, a NOT egy keresési feltétel első kulcsszava is lehet, és lehetővé teszi, hogy ugyanúgy zárjunk ki vele sorokat, mint az állításoknál. Ezt a sajátos NOT műveletet egy feltétel (állítás) vagy egy beágyazott keresési feltétel *előtt* használhatjuk. Ugyanazt a feltételt itt is többféle módon megfogalmazhatjuk.



6.11. ábra

A NOT művelet használata egy keresési feltételben

Tegyük fel, hogy a következő kérelmet szeretnénk megfogalmazni:

„Mutasd meg azoknak a versenyeknek a helyszínét és időpontját, amelyeket nem a Bolero Lanes, az Imperial Lanes vagy a Thunderbird Lanes pályán tartanak!"

Nem nehéz rájönni, hogy egy tagsági feltételt fogunk alkalmazni ehhez a kérdéshez, csupán meg kell azt fogalmaznunk. Az egyik megközelítés, hogy a NOT műveletet használjuk az állításon belül:

```
WHERE TourneyLocation NOT IN ('Bolero Lanes', 'Imperial Lanes',
    'Thunderbird Lanes')
```

A másik megközelítés szerint a NOT műveletet első kulcsszóként is írhatjuk a keresési feltétel elé:

```
WHERE NOT TourneyLocation IN ('Bolero Lanes', 'Imperial Lanes',
   'Thunderbird Lanes')
```

Mindkét feltétel ki fogja zárni azokat a versenyeket, amelyeknek a helyszíne a Bolero Lanes, az Imperial Lanes vagy a Thunderbird Lanes. Annak azonban, hogy a NOT-ot a keresési feltétel előtt használjuk, az az egyik előnye, hogy egy összehasonlító feltételre is alkalmazható. (Emlékezzünk vissza, hogy az összehasonlító feltétel utasításformája nem tartalmazza a NOT kulcsszót lehetséges műveletként.) Így tehát egy összehasonlító feltételt sorok kizárására is használhatunk. Ennek alkalmazását a következő példa mutatja:

„Mutasd meg azokat a tekézőket, akik nem Bellevue-ben élnek!"

Fordítás	Select first name, last name, and city from the bowlers table where the city is not 'Bellevue' (Válaszd ki azoknak a tekézőknek a vezeték- és keresztnevét, illetve városát a tekézők táblájából, akiknek a városa nem 'Bellevue'.)
Tisztázás	Select first name, last name, ~~and~~ city from ~~the~~ bowlers ~~table~~ where ~~the~~ city ~~is~~ not = 'Bellevue'

SQL	`SELECT BowlerFirstName, BowlerLastName, BowlerCity`
	`FROM Bowlers`
	`WHERE NOT BowlerCity = 'Bellevue'`

Igen, tudjuk jól, hogy ezt a feltételt a `WHERE BowlerCity <> 'Bellevue'` formában is kifejezhettük volna. Ez a példa csak azt hangsúlyozza, hogy egy feltételt számos módon megfogalmazhatunk.

Most, hogy megtanultuk, hogyan használhatjuk a NOT műveletet egy egyszerű és egy összetett keresési feltételben, ismerkedjünk meg azzal a problémával, amikor egy keresési feltételt két NOT művelettel írunk fel, és ez a sorokat *belefoglalja* az eredménybe, nem pedig *kizárja*. Lássunk egy példát erre:

„Mely alkalmazottaink nem tanárok vagy tanársegédek?"

Fordítás Select first name, last name, and title from the staff table where the title is not 'teacher' or 'teacher's aide'
(Válaszd ki azoknak a vezeték- és keresztnevét, illetve rangját a tanárok táblájából, akiknek a címe nem 'tanár' vagy 'tanársegéd'.)

Tisztázás Select first name, last name, ~~and~~ title from ~~the~~ staff ~~table~~ where ~~the~~ title ~~is~~ not in ('teacher', ~~or~~ 'teacher"s aide')

SQL
```
SELECT StfFirstName, StfLastName, Title
FROM Staff
WHERE NOT Title
NOT IN ('Teacher', 'Teacher''s Aide')
```

Megjegyzés *Csodálkozhatunk, hogy mit keres két aposztróf a* `'Teachers"s Aide'` *literális karakterláncban. Az SQL-szabvány azt mondja, hogy aposztróffal kell elválasztani a literális karakterláncokat és időpontokat. Amikor egy aposztrófot szeretnénk egy literális karakterláncba írni, akkor „meg kell győzni" az adatbázis-kezelő rendszerünket erről a szándékunkról az aposztróf kétszeres beírásával. Ha nem így teszünk, akkor az aposztróf a karakterlánc végét fogja jelölni, a második aposztróf után következő „s Aide'" szövegre pedig az adatbázisrendszer nyelvtani hibát jelez.*

Feltesszük természetesen, hogy a két NOT művelet egyike figyelmetlenségből került bele a lekérdezésbe. A SELECT utasítás így is végrehajtható, de rossz sorokat fogunk eredményként kapni. Ebben az esetben a két NOT művelet kioltja egymást – mint két mínusz előjel a matematikában vagy a logikában –, és az IN állítás fogja az eredménybe kerülő sorokat meghatározni. Tehát ahelyett, hogy a tanárokon és tanársegédeken *kívül mindenkit* látnánk az eredmények között, *csak a* tanárok és tanársegédek fognak az eredményben szerepelni. Ámbár szándékosan nem írunk ilyen keresési feltételt, véletlenül könnyen előfordulhat. Emlékezzünk arra, hogy gyakran az egyszerű hibák okozzák a legtöbb gondot.

Kiértékelési sorrend

Az SQL-szabvány meghatározza, hogy egy adatbázis-kezelő rendszernek hogyan kell kiértékelnie az egyes feltételeket egy keresési feltételen belül, illetve a kiértékelés sorrendjét. Ebben a fejezetben megtanultuk, *hogyan* értékeli ki az adatbázisrendszer az egyes típusú feltételeket, most pedig megnézzük, hogy az adatbázisrendszer *mikor* értékeli ki az egyes feltételeket.

Alapértelmezés szerint az adatbázisban balról jobbra haladva történik a feltételek kiértékelése. Ez különösen igaz az egyszerű feltételekre. A következő példában a SELECT utasítás először megkeresi azokat a sorokat, ahol a szállítási dátum egyenlő a megrendelési dátummal, és utána határozza meg, hogy mely sorokban 1001 a megrendelő száma. Azok a sorok, amelyek mindkét feltételt teljesítik, bekerülnek az eredmények közé.

```
SQL     SELECT CustomerID, OrderDate, ShipDate
        FROM Orders
        WHERE ShipDate = OrderDate
           AND CustomerID = 1001
```

Ha azt szeretnénk, hogy a SELECT utasítás előbb egy bizonyos megrendelői számot keressen, és utána értékelje ki a szállítási dátumot, csak cseréljük meg a két feltétel helyét. Később ebben a részben elmagyarázzuk, hogy esetleg miért szeretnénk ilyet tenni. Amikor egy keresési feltétel többféle egyszerű feltételből áll, az adatbázis egy adott sorrendben értékeli ki azokat, az egyes feltételekben szereplő műveletektől függően. Az SQL-szabvány a következő elsőbbségi sorrendet (vagy kiértékelési sorrendet) határozza meg a műveletek kiértékelésére:

Kiértékelési sorrend	Művelet típusa
1	Pluszjel (+), mínuszjel (–)
2	Szorzás (*), osztás (/)
3	Összeadás (+), kivonás (–)
4	=, <>, <, >, <=, >=, BETWEEN, IN, LIKE, IS NULL
5	NOT
6	AND
7	OR

A következő SELECT utasításban egy olyan keresési feltételre láthatunk példát, amelynek a kiértékelésekor az adatbázisnak az elsőbbségi sorrendet kell követnie. Ebben az esetben az adatbázis elvégzi az összeadás műveletét, végrehajtja az összehasonlításokat, és eldönti, hogy valamelyik feltétel igaz lett-e. Ha valamelyik sor kielégítette valamelyik feltételt, akkor az bekerül az eredmények közé.

```
SQL     SELECT CustomerID, OrderDate,ShipDate
        FROM Orders
        WHERE CustomerID = 1001
           OR ShipDate = OrderDate + 4
```

A feltételek elsőbbsége

Jelentősen javíthatjuk a keresési feltételeink pontosságát, ha megértjük az elsőbbségi sorrend lényegét. Ezzel a tudással képesek leszünk megalkotni a szükséges feltételeket, viszont a félreérthető feltételek megfogalmazásával vigyázni kell, mert váratlan eredményt kaphatunk. Nézzünk meg a következő példán egy lehetséges problémát:

```
SQL     SELECT CustFirstName, CustLastName, CustState, CustZipCode
        FROM Orders
        WHERE CustLastName = 'Patterson'
           AND CustState = 'CA'
           OR CustZipCode LIKE '%9'
```

Ebben a példában nehéz eldönteni, hogy mi a valódi célja a keresési feltételnek, mivel kétféle módon is értelmezhetjük:

1. A Kalifornia államban élő Patterson nevűeket keressük, *vagy* bárkit, akinek az irányítószáma 9-esre végződik.
2. A Patterson nevűeket keressük, *és* bárkit, aki Kaliforniában él, vagy az irányítószáma 9-esre végződik.

Ha emlékszünk a kiértékelés sorrendjére, akkor tudjuk, hogy az első megoldás lenne a jó, mivel a rendszernek az AND kulcsszót az OR művelet előtt kellene kiértékelnie. De biztos mindig emlékezni fogunk a kiértékelési sorrendre? Ezt a bizonytalanságot elkerülhetjük, és átláthatóbb keresési feltételt írhatunk, ha zárójeleket használunk, amelyekkel összevonunk egyes feltételeket, és megadjuk a kiértékelésük sorrendjét. Például a keresési feltételnek az első megoldás szerinti értelmezéséhez a WHERE záradékot a következő módon írhatjuk:

```
WHERE (CustLastName = 'Patterson' AND CustState = 'CA') OR
CustZipCode LIKE '%9'
```

A zárójelek biztosítják, hogy az adatbázis *előbb* elemzi és értékeli ki a két összehasonlító feltételt, mint a mintaillesztési feltételt.

Ha viszont a második megoldás szerinti értelmezést szeretnénk megvalósítani, akkor a következő WHERE záradékot használhatjuk:

```
WHERE CustLastName = 'Patterson' AND (CustState = 'CA' OR CustZipCode
LIKE '%9')
```

Ebben az esetben az adatbázis az első összehasonlító feltételt az *után* elemzi és értékeli ki, miután végzett a második összehasonlító és mintaillesztési feltétellel.

A feltételek zárójelezése mostanra már ismerős lehet, hiszen a fejezetben megtanultuk, hogyan tehetjük ezt meg, amikor a feltételek összevonása a cél. Most csak megpróbáljuk kihangsúlyozni, hogy a zárójelek helyének komoly hatása lehet az eredményre nézve. A zárójelekbe tett feltételekből akárhányat használhatunk, és még egymásba is ágyazhatjuk őket szükség szerint. A kifejezésekhez hasonlóan a keresési feltételek feldolgozása is balról jobbra történik, illetve a legbelsőtől a külsők felé haladva a zárójelekben, *kivéve* ha két vagy több feltétel azonos szinten van, ilyenkor ugyanis az adatbázisrendszer előbb az AND, majd az OR műveleteket értelmezi. Az adatbázis a következő módon kezeli a zárójelezett keresési feltételeket:

- A zárójeles keresési feltételek feldolgozása a nem zárójelezett keresési feltételek előtt történik.
- Két vagy több zárójelezett keresési feltétel feldolgozása balról jobbra megy végbe.
- Az egymásba ágyazott zárójeles keresési feltételek feldolgozására a legbelsőtől a legkülsőig kifelé haladva kerül sor.

Miután az adatbázis elemezte az adott zárójeles feltételt, az összes kifejezést a szokásos kiértékelési sorrendben hajtja végre. Ha megfelelő odafigyeléssel fordítottuk le a kérdésünket, és hatékonyan használtuk a zárójeleket a keresési feltételben, akkor pontosabb eredményt fogunk kapni.

A kevesebb több

Ebben a részben már említettük, hogy az adatbázis kezdetben balról jobbra halad a feltételek kiértékelésével, és az elsőbbségi sorrendet akkor alkalmazza, ha összetett feltételt adtunk meg. Arról is volt szó, hogy a zárójelek használatának közvetlen hatása van az eredményre. Most egy egyszerű, általános tanácsot adunk a keresési folyamat felgyorsítására: kérjünk kevesebbet, azaz kevesebb oszlopot használjunk a kérdés megválaszolásához, és a keresési feltételt a lehető legjobban szűkítsük le, így az adatbázis-kezelő a lehető legkevesebb sort fogja feldolgozni. Ha több feltételt kell használnunk, bizonyosodjunk meg arról, hogy először a legtöbb sort kizáró feltétel feldolgozására kerül sor, és az adatbázis reménhetőleg gyorsabban megtalálja a szükséges választ. (Itt lehet igazán előnyös, ha alaposan ismerjük az elsőbbségi sorrendet.)

Ezt a tippet egy korábban használt példán mutatjuk be:

SQL
```
SELECT CustomerID, OrderDate, ShipDate
FROM Orders
WHERE ShipDate = OrderDate
AND CustomerID = 1001
```

Ennél az esetnél egy sornak mindkét feltételt teljesítenie kell, hogy az eredmények között szerepelhessen. Az állítások ilyen sorrendje arra utasítja az adatbázis-kezelőt, hogy hasonlítson össze minden szállítási dátumot a hozzá tartozó megrendelési dátummal. Ennek a feltételnek a megvizsgálása a táblában szereplő sorok számától függően jelentős időt is igénybe vehet. Ezután az adatbázis azon sorok között, amelyekre az előző feltétel igaz lett, megkeresi az 1001-es vásárlóazonosító számot tartalmazó sorokat.

Lássunk egy talán jobban megfogalmazott lekérdezést ugyanezzel a feltétellel:

SQL
```
SELECT CustomerID, OrderDate, ShipDate
FROM Orders
WHERE CustomerID = 1001
AND ShipDate = OrderDate
```

Most az adatbázis-kezelő várhatóan előbb a vásárlóazonosító számát fogja keresni.
Ez a feltétel feltehetőleg kevesebb sort eredményez, ami azt jelenti, hogy az adatbázisnak kevesebb idő kell a szállítási dátum feltételének kiértékeléséhez.

Ezt a módszer érdemes átvinni a napi gyakorlatba és alkalmazni a keresési feltételek megfogalmazásánál. Ez hosszú távon biztosítja, hogy a SELECT utasításainkat gyorsan és hatékonyan végre lehessen hajtani. Tanulmányozzuk az adatbázisrendszerünk kézikönyvét, hogy milyen más eljárásokkal tudjuk tovább optimalizálni a SELECT utasításainkat.

Megjegyzés *Jóformán az összes kereskedelmi forgalomban kapható adatbázis-rendszer tartalmaz lekérdezés-optimalizáló eljárásokat, amelyek átnézik a lekérdezésünket, és megpróbálják kitalálni, hogyan tudna az a lehető leggyorsabban lefutni. Az optimalizáló eljárások hatékonyságára az adatbázis rendszergazdája által beállított oszlopindexeknek van a legnagyobb befolyásuk, de nem árt, ha gyakoroljuk, hogy a legtöbb sort kizáró feltételt adjuk meg első feltételként, mert ezzel is segíthetjük az adatbázisrendszer saját optimalizáló eljárásait.*

Most, hogy megértettük a keresési feltételek összevonásának módszereit, tegyünk egy kis kitérőt valami még összetettebb felé. Mit tehetünk, ha azokat a sorokat akarjuk megtalálni, amelyekben értékek egy tartománya megegyezik más értékek egy tartományával? Olvassunk tovább!

Egymást átfedő tartományok ellenőrzése

A BETWEEN nagyon jól működik, ha egyetlen oszlopban keresünk olyan értéket, ami egy adott tartományban van. Azt is megtanultuk, hogy egy értéket is vizsgálhatunk, hogy benne van-e két oszlop (alsó és felső határ) által meghatározott tartományban. De mit kell tennünk, ha el szeretnénk eldönteni, hogy egy tartomány átfed-e egy másikat? Például tudni szeretnénk, hogy mely rendezvényekre (mindegyiknek van kezdő és záró dátuma) kerül sor 2007. november 12. és 2007. november 18. között. Első gondolatunk a BETWEEN használata lehet:

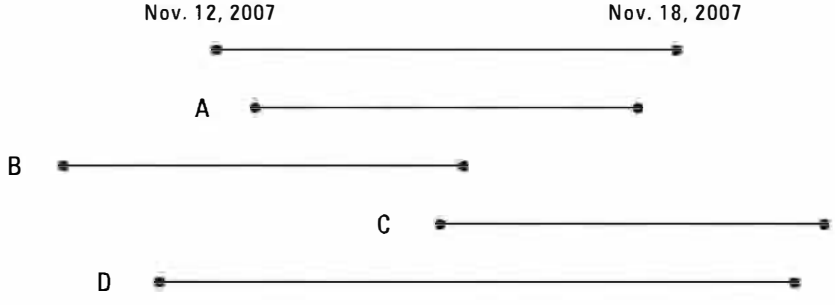
„Mutasd meg azokat a rendezvényeket, amelyekre 2007. november 12. és 2007. november 18. között kerül sor!"

Fordítás
```
Select engagement number, start date, and end date
from the engagements table
where start date is between November 12, 2007, and
  November 18, 2007
and end date is between November 12, 2007, and
  November 18, 2007
```
(Válaszd ki azoknak a rendezvényeknek a számát, illetve a kezdő és záró dátumát a rendezvények táblájából, amelyeknek a kezdő dátuma 2007. november 12. és 2007. november 18., a záró dátuma pedig 2007. november 12. és 2007. november 18. közötti.)

Tisztázás
```
Select engagement number, start date, ~~and~~ end date
from ~~the~~ engagements ~~table~~
where start date ~~is~~ between ~~November 12, 2007~~ '2007-11-12'
```

```
                and November 18, 2007 '2007-11-18'
              and end date is between November 12, 2007 '2007-11-12'
                and November 18, 2007 '2007-11-18'
SQL           SELECT EngagementNumber, StartDate, EndDate
              FROM Engagements
              WHERE StartDate BETWEEN '2007-11-12' AND '2007-11-18'
              AND EndDate BETWEEN '2007-11-12' AND '2007-11-18'
```

Majdnem jó, de nem teljesen. Valójában azok a rendezvények érdekelnek minket, amelyeknek bármelyik időpontja a két adott novemberi nap közé esik. Hogy megértsük, hogy miért nem működik egy ilyen egyszerű kombinációja a BETWEEN záradékoknak, nézzük meg a 6.12. ábrát.



6.12. ábra

Az adott időszakba eső rendezvények

Ahogy ezen az ábrán láthatjuk, a rendezvényeknek négyféle olyan időszaka van, ami részben vagy teljesen a kívánt héten belülre esik. Néhány rendezvény teljesen része az időszaknak, ezeket az A vonal jelöli. Néhány a megadott időszak előtt kezdődik, de abban ér véget; ezeket jelöli a B vonal. Mások viszont az időszakban kezdődnek, de utána fejeződnek be; ezeket mutatja a C vonal. Végül, vannak olyan rendezvények, amelyek az időszak előtt kezdődnek, és utána fejeződnek be; ezeket a D vonal jelöli.

Ha átgondoljuk, hogyan tettük fel a kérdést, az arra adott válasz csak az A vonalhoz tartozó rendezvényeket fogja megadni. A B azért nem lesz jó, mert a kezdő dátum nem november 12. és november 18. közé esik, még ha a rendezvény egy részére az adott időtartamban kerül is sor. A C-t azért kell kizárni, mert a záró dátum nincs az adott napok között, a D pedig azért nem lesz az eredmények között, mert mind a kezdő, mind a záró dátum az adott időszakon kívülre esik, még ha maga a rendezvény ***végig*** az adott időszak alatt tart is.

Tehát mi a megoldás erre a problémára? Írhatunk feltételeket mind a négy eshetőségre külön:

```
WHERE (StartDate BETWEEN '2007-11-12' AND '2007-11-18'
AND EndDate BETWEEN '2007-11-12' AND '2007-11-18')
OR (StartDate <= '2007-11-12')
AND EndDate BETWEEN '2007-11-12' AND '2007-11-18')
OR (StartDate BETWEEN '2007-11-12' AND '2007-11-18'
AND EndDate >= '2007-11-18')
OR (StartDate <= '2007-11-12'
AND EndDate >= '2007-11-18')
```

Nem túl szép, ugye? De nézzük meg jobban! Mi az az egy dolog, ami minden kezdő dátumnál azonos? Mindegyik kisebb vagy egyenlő, mint az időtartam záró dátuma! Ugyanígy a záró dátumok mind nagyobbak vagy egyenlők, mint az időszak kezdő dátuma. Tehát az egyszerű válasz a következő:

SQL

```
SELECT EngagementNumber, StartDate, EndDate
FROM Engagements
WHERE StartDate <= '2007-11-18'
AND EndDate >= '2007-11-12'
```

Nem sokkal egyszerűbb? Tartsuk észben ezt a megoldást – szükségünk lesz rá a fejezet végén található egyik mintafeladat megoldásához. Most pedig térjünk vissza a szokásos programozáshoz – nézzük át a Null helyzetét még egyszer.

Visszatérés a Null-okhoz: egy figyelmeztető megjegyzés

Éppen ideje, hogy visszatérjünk a Null értékekhez. Az 5. fejezetben megtanultuk, hogy a Null egy érték hiányát jelenti, egy Null értéket feldolgozó kifejezés pedig Null értékkel fog visszatérni. Ugyanez igaz a keresési kifejezésekre is. Egy Null értéket kiértékelő állítás *soha nem lehet igaz* – viszont (bármilyen furcsa is) az állítás hamis sem lehet! Az SQL-szabvány bármely, Null értéket kiértékelő állítás eredményeként *ismeretlent* határoz meg. Emlékezzünk arra, hogy egy állításnak igaznak kell lennie azokra a sorokra, amelyek bekerülnek az eredménybe, tehát hamis vagy ismeretlen értékek esetén a sort ki kell zárni.

Ennek a tisztázásához tanulmányozzuk a 6.13. és 6.14 ábrákat, amelyeken az először a 6.9. és 6.10. ábrákon látott igazságtáblázatok szerepelnek, ezúttal azonban azokkal az ismeretlen értékekkel együtt, amiket egy Null érték kiértékelésekor kapunk.

Látható, hogy egy állítás kiértékelése egy Null értékű oszlopon igencsak megkavarja az eredményt! Például tegyük fel, hogy van egy egyszerű összehasonlító állításunk: A = B. Ha akár A, akár B egy adott sorra Null értékű, akkor az összehasonlítás eredménye ismeretlen lesz. Mivel ez nem igaz eredmény, a sort az adatbázis-kezelő nem jelöli ki. Ha A = B nem igaz, azt várhatnánk, hogy a NOT (A = B) igaz lesz. Nem! Ez szintén ismeretlen eredményt ad. A 6.15. ábra segít megérteni, hogy is van ez.

Második kifejezés

Első kifejezés / AND	Igaz	Hamis	Ismeretlen
Igaz	Igaz (Sorok kijelölve)	Hamis (Sorok kihagyva)	Ismeretlen (Sorok kihagyva)
Hamis	Hamis (Sorok kihagyva)	Hamis (Sorok kihagyva)	Hamis (Sorok kihagyva)
Ismeretlen	Ismeretlen (Sorok kihagyva)	Hamis (Sorok kihagyva)	Ismeretlen (Sorok kihagyva)

6.13 ábra

Két állítás összevonásának eredménye az AND művelet használatakor, amikor valamelyik kifejezés Null (ismeretlen) is lehet

Második kifejezés

Első kifejezés / OR	Igaz	Hamis	Ismeretlen
Igaz	Igaz (Sorok kijelölve)	Igaz (Sorok kijelölve)	Igaz (Sorok kijelölve)
Hamis	Igaz (Sorok kijelölve)	Hamis (Sorok kihagyva)	Ismeretlen (Sorok kihagyva)
Ismeretlen	Igaz (Sorok kijelölve)	Ismeretlen (Sorok kihagyva)	Ismeretlen (Sorok kihagyva)

6.14. ábra

Két állítás összevonásának eredménye az OR művelet használatakor, amikor valamelyik kifejezés Null (ismeretlen) is lehet

(Kifejezés)	NOT (Kifejezés)
Igaz	**Hamis**
Hamis	**Igaz**
Ismeretlen	**Ismeretlen**

6.15. ábra

A NOT művelet alkalmazása igaz/hamis/ismeretlen értékre

Tegyük fel, hogy a következő lekérdezést szeretnénk intézni az adatbázishoz:

„Mutasd meg azoknak a nevét és a telefonszámát King megyéből, akiknek a vezetékneve Hernandez!"

Fordítás Select first name, last name, and phone number from the customers table where the county name is 'King' and the last name is 'Hernandez'
(Válaszd ki azoknak a vásárlóknak a vezeték- és keresztnevét, illetve a telefonszámát a vásárlók táblájából, akiknek a megyéje 'King', és a vezetéknevük 'Hernandez'.)

Tisztázás Select first name, last name, ~~and~~ phone number
from ~~the~~ customers ~~table~~
where ~~the~~ county ~~name is~~ = 'King'
and ~~the~~ last name ~~is~~ = 'Hernandez'

SQL
```
SELECT CustFirstName, CustLastName, CustPhoneNumber
FROM Customers
WHERE CustCounty = 'King'
   AND CustLastName = 'Hernandez'
```

Jól tudjuk, hogy egy sornak *mindkét* feltételt teljesítenie kell, hogy bekerülhessen az eredmények közé. Ha vagy a megye neve, vagy a vezetéknév Null, akkor az adatbázis az adott sort eldobja.

Nézzük meg a következő lekérdezést:

„Mutasd meg azoknak a dolgozóknak a nevét, akik végzett tanácsadók, vagy akiket 2007. szeptember 1-én vettek fel!"

Fordítás Select last name and first name from the staff table where the title is 'graduate counselor' or date hired is September 1, 2007
(Válaszd ki azoknak a vezeték- és keresztnevét a tanárok táblájából, akiknek a címe 'graduate counselor', vagy akiknél a felvétel dátuma 2007. szeptember 1.)

Tisztázás Select last name ~~and~~ first name
from ~~the~~ staff ~~table~~
where ~~the~~ title ~~is~~ = 'graduate counselor'
or date hired ~~is~~ = ~~September 1, 2007~~ '2007-09-01'

SQL
```
SELECT StfLastName, StfFirstName
FROM Staff
WHERE Title = 'Graduate Counselor'
   OR DateHired = '2007-09-01'
```

Ugyan azt várhatnánk, hogy a Null értékeknek hasonló hatásuk van az OR műveletttel összevont feltételekre, mint az AND használatakor, de ez nem szükségszerű. Egy sor akkor is bekerülhet az eredménybe, ha a fenti feltételek *valamelyikét* teljesíti. Vessünk ismét egy pillantást a 6.14. ábrára. A Title és a DateHired értékeire alapozva a 6.2. táblázat megmutatja, hogy az adatbázis miként dönti el, hogy egy sor szerepelhet-e az eredmények között az OR művelet használatánál.

6.2. táblázat *Az eredmények meghatározása az OR használatánál*

A Title értéke	A DateHired értéke	Eredmény
Graduate Counselor	2007-09-01	A sor szerepel az eredményben, mert mindkét feltételt kielégíti.
Graduate Counselor	2007-11-15	A sor szerepel az eredményben, mert teljesíti az első feltételt.
Registrar	2007-09-01	A sor szerepel az eredményben, mert teljesíti a második feltételt.
Graduate Counselor	Null	A sor szerepel az eredményben, mert teljesíti az első feltételt.
Null	2007-09-01	A sor szerepel az eredményben, mert teljesíti a második feltételt.
Null	Null	A sort kizártuk, mert egyik feltételt sem teljesíti.

Ha azt feltételezzük, hogy az eredmény hibás adatokat tartalmaz, ellenőrizzük az adott oszlopokat a Null feltétellel. Ezzel lehetőségünk adódik arra, hogy szükség szerint foglalkozzunk a Null értékekkel, és utána újra lefuttassuk az eredeti SELECT utasítást. Például

ha úgy gondoljuk, hogy néhány „graduate counselor" (végzett tanácsadó) hiányzik az eredményből, lefuttathatjuk a következő SELECT utasítást, hogy lássuk, hogy a feltételezésünk igaz-e:

SQL
```
SELECT StfLastName, StfFirstName, Title
FROM Staff
WHERE Title IS NULL
```

Ha vannak Null értékek a Title oszlopban, akkor ennek a SELECT utasításnak az eredménye az összes olyan dolgozót tartalmazni fogja, akinek nincs meghatározva a rangja az adatbázisban. Ezután ezeken az adatokon elvégezhetjük a szükséges teendőket, majd visszatérhetünk az eredeti SELECT utasításhoz.

Még nem fejeztük be a Null értékek tárgyalását. Még egyszer visszatérünk a Null-okhoz a 12. fejezetben, amikor az adatösszesítő SELECT utasításokat tárgyaljuk.

Feltételek más megfogalmazásban

Az ebben a fejezetben tanultak egyik előnye, hogy most már számos módon ki tudunk fejezni egy adott feltételt. Nézzünk erre egy példát a következő lekérdezéssel:

„Add meg minden olyan alkalmazott nevét, akit 2007 októberében vettek fel!"

A kérdés megválaszolásához azokat a felvételi dátumokat kell megkeresnünk, amelyek 2007. október 1. és 2007. október 31. közé esnek. Az eddig tanultakra alapozva ezt a feltételt kétféle módon írhatjuk fel:

```
DateHired BETWEEN '2007-10-01' AND '2007-10-31'
DateHired >= '2007-10-01' AND DateHired <= '2007-10-31'
```

Mindkét feltétel ugyanazokat a sorokat fogja beválasztani az eredmények közé – az, hogy melyiket választjuk, csak tőlünk függ. Néhányan az első kifejezést találják érthetőbbnek, míg egyesek a másodikat részesítik előnyben.

Néhány példa egyenértékű feltételekre:

„Mutasd meg azokat a beszállítókat, akiknek a telephelye Kaliforniában, Oregonban vagy Washingtonban található!"

```
VendState IN ('CA', 'OR', 'WA')
VendState = 'CA' OR VendState = 'OR' OR VendState = 'WA'
```

„Adj egy listát azokról a vásárlókról, akiknek a vezetékneve 'H'-val kezdődik!"

```
CustLastName >= 'H' AND CustLastName <= 'HZ'
CustLastName BETWEEN 'H' AND 'HZ'
CustLastName LIKE 'H%'
```

„Mutasd meg azokat a hallgatókat, akik nem Seattle-ben vagy Redmondban élnek!"

```
StudCity <> 'Seattle' AND StudCity <> 'Redmond'
StudCity NOT IN ('Seattle', 'Redmond')
NOT (StudCity = 'Seattle' OR StudCity = 'Redmond')
```

Egy feltétel megfogalmazásának nincs rossz megoldása, de lehet hibásan fogalmazni, ha nagyvonalúan figyelmen kívül hagyjuk a nyelvtant. (Mint tudjuk, ekkor a feltétel kiértékelése hibás lesz.) Ennek ellenére néhány adatbázisrendszer egyes feltételtípusokat megpróbálhat optimálisan végrehajtani a gyorsabb feldolgozás érdekében, és ekkor ajánlott ezeket használni. Nézzünk utána az adatbázisrendszerünk kézikönyvében, hogy vannak-e ilyen előnyben részesített módszerek a feltételek megfogalmazásánál.

Példák

Megtanultuk az összes szükséges technikát, ami a megbízható keresési feltételek elkészítéséhez szükséges. Nézzünk hát példákat a különböző keresési feltételekre a mintaadatbázisokból vett táblák segítségével! Ezek a példák megmutatják, hogyan használjuk a keresési feltételeket az adatok szűrésére.

Az SQL-utasítások után a mintalekérdezések eredményeit is mellékeltük; ezeket kell kapnunk a műveletek végrehajtása után. Az eredmény felett látható cím a lekérdezés neve, amelyen a lekérdezést megtalálhatjuk a CD melléklet mintaadatai között. Minden lekérdezést a megfelelő mintaadatbázisban helyeztünk el (ahogy a példáknál jelöltük), és az ehhez a fejezethez tartozó példákat a „CH06" előtaggal láttuk el. A példákat a könyv bevezetésében leírt módon tölthetjük be és próbálhatjuk ki.

Megjegyzés *A következő példáknál összevontuk a Fordítás és Tisztázás részeket, hogy tovább gyakorolhassuk az egyesítés folyamatát.*

Sales Orders adatbázis

„Mutasd meg a 1001-es vásárlói számmal rendelkező vásárló megrendeléseit!"

Fordítás/ Tisztázás: Select ~~the~~ order number ~~and~~ customer ID from ~~the~~ orders ~~table~~ where ~~the~~ customer ID ~~is equal to~~ = 1001
(Válaszd ki a megrendelés számát és a vásárló azonosítóját a megrendelések táblájának azon soraiból, ahol a vásárlóazonosító értéke 1001.)

SQL

```
SELECT OrderNumber, CustomerID
FROM Orders
WHERE CustomerID = 1001
```

CH06_Orders_for_Customer_1001 (44 sor)

OrderNumber	CustomerID
2	1001
7	1001
16	1001
52	1001
55	1001
107	1001
137	1001
138	1001
151	1001
154	1001
<< további sorok >>	

„Mutasd meg a 'Dog' kezdetű termékek ábécérendbe szedett listáját!"

Fordítás/ Tisztázás: Select ~~the~~ product name from ~~the~~ products ~~table~~ where ~~the~~ product name like 'Dog%' ~~and~~ order by product name
(Válaszd ki azoknak a termékeknek a nevét a termékek táblájából, amelyeknél a terméknév 'Dog'-gal kezdődik.)

SQL

```
SELECT ProductName
FROM Products
WHERE ProductName LIKE 'Dog%'
ORDER BY ProductName
```

CH06_Products_That_Begin_With_DOG (4 sor)

ProductName
Dog Ear Aero-Flow Floor Pump
Dog Ear Cyclecomputer
Dog Ear Helmet Mount Mirrors
Dog Ear Monster Grip Gloves

Megjegyzés *Csak emlékeztetni szeretnénk arra, hogy az ORDER BY záradékot a SELECT utasítás végén kell elhelyezni. Ha szükséges, olvassuk át újra a 4. fejezet Az információk rendezése című részét.*

Entertainment Agency adatbázis

„Mutass egy névsorrendbe szedett listát a bellevue-i, redmondi és woodinville-i előadókról!"

Fordítás/ Tisztázás: Select stage name, phone number, ~~and~~ city
from ~~the~~ entertainers ~~table~~
where ~~the~~ city ~~is~~ in ('Bellevue', 'Redmond', ~~or~~ 'Woodinville')
~~and~~ order by stage name
(Válaszd ki azoknak az előadóknak a művésznevét, telefonszámát és városát, akiknek a városa 'Bellevue', 'Redmond' vagy 'Woodinville', és rendezd a listát a művésznév szerint.)

SQL:
```
SELECT EntStageName, EntPhoneNumber, EntCity
FROM Entertainers
WHERE EntCity
IN ('Bellevue', 'Redmond', 'Woodinville')
ORDER BY EntStageName
```

CH06_Eastside_Entertainers (7 sor)

EntStageName	EntPhoneNumber	EntCity
Carol Peacock Trio	555-2691	Redmond
Jazz Persuasion	555-2541	Bellevue
Jim Glynn	555-2531	Bellevue
JV & the Deep Six	555-2511	Redmond
Katherine Ehrlich	555-0399	Woodinville
Modern Dance	555-2631	Woodinville
Susan McLain	555-2301	Bellevue

„Mutasd meg az összes olyan rendezvényt, amely négy napig tart!"

Fordítás/ Tisztázás: Select engagement number, start date, ~~and~~ end date
from ~~the~~ engagements ~~table~~
where ~~the~~ CAST(end date ~~minus~~ - start date AS INTEGER) ~~is~~ ~~equal-to~~ = 3
(Válaszd ki azoknak a rendezvényeknek a számát és kezdő-, illetve záró dátumát a rendezvények táblájából, amelyeknél a CAST(záró dátum mínusz kezdő dátum AS INTEGER) értéke 3.)

SQL:
```
SELECT EngagementNumber, StartDate, EndDate
FROM Engagements
WHERE CAST(EndDate – StartDate AS INTEGER) = 3
```

CH06_Four-Day Engagements (15 sor)

EngagementNumber	StartDate	EndDate
5	2007-09-11	2007-09-14
13	2007-09-17	2007-09-20
17	2007-09-29	2007-10-02
21	2007-09-30	2007-10-03
56	2007-11-25	2007-11-28
58	2007-12-01	2007-12-04
59	2007-12-01	2007-12-04
63	2007-12-18	2007-12-21
70	2007-12-23	2007-12-26
95	2008-01-15	2008-01-18
<< további sorok >>		

Megjegyzés *Minden rendezvény a kezdő dátumtól a záró dátumig tart, és ezek is beleszámítanak az időtartamba. Amikor kivonjuk a StartDate értékét az EndDate értékéből, eggyel kevesebb napot kapunk eredményül, mint a rendezvény teljes időtartama, ezért hasonlítjuk össze a számítás eredményét a 3-mal és nem a 4-gyel.*

School Scheduling adatbázis

„Adj egy név szerint rendezett listát az évi 40 000 és 50 000 dollár közötti fizetést kapó dolgozók nevével és fizetésével!"

Fordítás/ Tisztázás
Select first name, last name, ~~and~~ salary
from ~~the~~ staff ~~table~~
where ~~the~~ salary ~~is~~ between 40000 and 50000~~, then~~
order by last name, ~~and~~ first name
(Válaszd ki azoknak a dolgozóknak a kereszt- és vezetéknevét, illetve fizetését a dolgozók táblájából, akiknek a fizetése 40 000 és 50 000 közötti, és rendezd a listát vezetéknév, majd keresztnév szerint.)

SQL

```
SELECT StfFirstName, StfLastName, Salary
FROM Staff
WHERE Salary BETWEEN 40000 AND 50000
ORDER BY StfLastname, StfFirstName
```

CH06_Staff_Salaries_40K_TO_50K (14 sor)

StfFirstName	StfLastName	Salary
Robert	Brown	$49,000.00
Kirk	DeGrasse	$45,000.00
Katherine	Ehrlich	$45,000.00
Jim	Glynn	$45,000.00
Liz	Keyser	$48,000.00
Ann	Patterson	$45,000.00
Maria	Patterson	$48,000.00
Mariya	Sergienko	$45,000.00
Tim	Smith	$40,000.00
Caleb	Viescas	$45,000.00
<< *további sorok* >>		

„Mutasd meg azoknak a hallgatóknak a listáját, akiknek a vezetékneve 'Kennedy', vagy akik Seattle-ben élnek!"

Fordítás/ Select first name, last name, ~~and~~ city
Tisztázás from ~~the~~ students ~~table~~
where ~~the~~ last name ~~is~~ = 'Kennedy'
or ~~the~~ city ~~is~~ = 'Seattle'
(Válaszd ki azoknak a hallgatóknak a keresztnevét, vezetéknevét és városát a hallgatók táblájából, akiknek a vezetékneve 'Kennedy' vagy a városa 'Seattle'.)

SQL
```
SELECT StudFirstName, StudLastName, StudCity
FROM Students
WHERE StudLastName = 'Kennedy'
OR StudCity = 'Seattle'
```

Seattle_Students_And_Students_Named_Kennedy (4 sor)

StudFirstName	StudLastName	StudCity
Doris	Hartwig	Seattle
John	Kennedy	Portland
Kendra	Bonnicksen	Seattle
Richard	Lum	Seattle

Bowling League adatbázis

„Írd ki azoknak a csapatoknak az azonosítószámát, amelyek egy vagy több meccset nyertek az első tízből a 3. játékban!"

Fordítás/ Tisztázás: Select ~~the~~ team ID, match ID, ~~and~~ game number from ~~the~~ match_games ~~table~~ where ~~the~~ game number ~~is~~ = 3 and ~~the~~ match ID ~~is~~ between 1 and 10

(Válaszd ki azokat a csapatazonosítókat, mérkőzésazonosítókat és játékszámokat a játékok táblájából, amelyeknél a játék száma 3, és a mérkőzés azonosítója 1 és 10 közé esik.)

SQL:

```
SELECT WinningTeamID, MatchID, GameNumber
FROM Match_Games
WHERE GameNumber = 3 AND MatchID BETWEEN 1 AND 10
```

Game3_Top_Ten_Matches (10 sor)

WinningTeamID	MatchID	GameNumber
1	1	3
3	2	3
5	3	3
7	4	3
3	5	3
4	6	3
5	7	3
8	8	3
2	9	3
1	10	3

„Írd ki azokat a tekézőket a 3., 4. és 5. csapatból, akiknek a vezetékneve 'H'-val kezdődik!"

Fordítás/ Tisztázás: Select first name, last name, ~~and~~ team ID from ~~the~~ bowlers ~~table~~ where ~~the~~ team ID ~~is either~~ in (3, 4, ~~or~~ 5) and ~~the~~ last name ~~begins with the letter~~ like 'H%'

(Válaszd ki azoknak a kereszt- és vezetéknevét, illetve csapatazonosítóját a játékosok táblájából, akiknek a csapatazonosítója 3, 4 vagy 5), és a vezetékneve 'H' betűvel kezdődik.)

SQL:

```
SELECT BowlerFirstName, BowlerLastName, TeamID
FROM Bowlers
WHERE (TeamID IN (3,4,5))
AND (BowlerLastName LIKE 'H%')
```

H_Bowlers_Teams_3_Through_5 (4 sor)

BowlerFirstName	BowlerLastName	TeamID
Elizabeth	Hallmark	4
Gary	Hallmark	4
Kendra	Hernandez	5
Michael	Hernandez	5

Recipes adatbázis

„Sorold fel azokat a recepteket, amelyekhez nem tartozik megjegyzés!"

Fordítás/ Tisztázás: Select ~~the~~ recipe title from ~~the~~ recipes ~~table~~ where notes is ~~empty~~ Null
(Válaszd ki azoknak a recepteknek a címét a receptek táblájából, amelyeknél a megjegyzés mező üres.))

SQL:
```
SELECT RecipeTitle FROM Recipes
WHERE Notes IS NULL
```

CH06_Recipes_With_No_Notes (6 sor)

RecipeTitle
Irish Stew
Salsa Buena
Fettuccini Alfredo
Mike's Summer Salad
Roast Beef
Yorkshire Pudding

„Mutasd meg azokat a hozzávalókat, amelyek a húsfélék csoportjába (a hozzávalók 2. osztályába) tartoznak, a csirkét kivéve!"

Fordítás/ Tisztázás: Select ingredient name from ~~the~~ ingredients ~~table~~ where ingredient class ID ~~is equal to~~ = 2 and ingredient name ~~does~~ not ~~contain~~ like '%chicken%'
(Válaszd ki azoknak a hozzávalóknak a nevét a hozzávalók táblájából, amelyeknek a hozzávalóosztálya 2, és a hozzávaló neve nem tartalmazza a 'chicken' szót.)

SQL:
```
SELECT IngredientName FROM Ingredients
   WHERE (IngredientClassID = 2)
   AND (IngredientName NOT LIKE '%chicken%')
```

CH06_Meats_That_Are_Not_Chicken (5 sor)

IngredientName
Beef
Bacon
T-bone Steak
New York Steak
Ground Pork

Összefoglalás

Ebben a fejezetben annak az elvével ismerkedtünk meg, hogy miként szűrhetjük az adatokat a WHERE záradékban megadott keresési feltételekkel. Megtanultuk, hogy a keresési feltételek állítások kombinációjából állnak, amelyek meghatározzák az eredménybe kerülő adatokat, és hogy minden állítás egyféle szempontból vizsgál egy adott értékkifejezést, majd bemutattuk az öt alapvető állítást.

A tárgyalást ennek az öt alapvető állításnak a részletezésével folytattuk, amelyeket a WHERE záradék keresési feltételében használhatunk. Megtanultuk, hogyan hasonlíthatunk össze értékeket, és hogyan ellenőrizhetjük, hogy egy érték más értékek egy adott tartományába esik-e. Azt is megtanultuk, hogyan vizsgálhatjuk meg, hogy egy érték egy megadott lista valamelyik elemével egyezik-e, vagy hogy része-e egy bizonyos mintakarakterláncnak. Ezenkívül megtanultuk, hogy a NOT művelettel kizárhatunk sorokat az eredményből.

Ezt követően megvizsgáltuk, hogyan adhatunk meg egyszerre több feltételt, amelyeket az AND és az OR műveletekkel fűzünk össze. Megtanultuk, hogy egy sornak minden AND művelettel összevont feltételt ki kell elégítenie, hogy bekerülhessen az eredménybe, míg az OR művelettel összekapcsolt feltételeknél csak egy feltételnek kell igaznak lennie. Az AND és az OR együttes használatával felírható összetett lekérdezésekről is szót ejtettünk, majd megint a NOT művelettel foglalkoztunk, és megtanultuk, hogy ezt a műveletet két külön szinten használhatjuk a keresési feltételekben a sorok kizárására.

Az elsőbbségi sorrend volt a következő a tárgyalt témák sorában: megtanultuk, hogyan elemzi és értékeli ki az adatbázis a feltételeket, így most már tudjuk, hogy az adatbázis egy bizonyos sorrendet követ a kiértékeléskor, amit az egyes feltételekben használt műveletek határoznak meg. Azt is megtanultuk, hogyan alkalmazhatjuk a zárójeleket az adatbázis által használt sorrend megváltoztatására egyes feltételeknél, és hogyan kerülhetjük el a félreérthető feltételeket.

Tettünk egy rövid kitérőt, hogy megmutassuk, hogyan kereshetünk egymást átfedő tartományokat. A megoldás meglepően egyszerű, és a BETWEEN használatát sem igényli. Ez után még egyszer visszatértünk a Null-okhoz. Megtanultuk, hogy a Null hasonló

hatással van a feltételekre, mint a kifejezésekre, és azt is megtudtuk, hogy ajánlott Null értékeket keresni az adataink között, ha arra gyanakszunk, hogy az eredményben hibás adatok szerepelnek.

Végül megtárgyaltuk, hogy ugyanaz a feltétel számos különböző módon megfogalmazható. Már tudjuk például, hogy három eltérő módon kereshetjük meg azokat, akiknek a vezetéknéve a „H" betűvel kezdődik.

A könyv következő részében a *halmazokat* és a rajtuk végezhető műveleteket mutatjuk be. A halmazok megismerése után térhetünk rá, hogy miként adhatunk meg több táblával dolgozó SELECT utasításokat. A fejezet utolsó részében pedig lássunk néhány önállóan kidolgozható kérelmet!

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Add meg az összes olyan beszállító nevét, akik Ballardban, Bellevue-ben vagy Redmondban működnek!"*
   A megoldás itt található: CH06_Ballard_Bellevue_Redmond_Vendors (3 sor).
2. *„Mutass egy ábécérendbe szedett listát a fogyasztói áron 125 dolláros vagy drágább termékekről!"*
   (Tipp: a listát egy korábbi fejezetben tanult záradékkal rendezhetjük ábécérendbe.)
   A megoldás itt található: CH06_Products_Priced_Over_125 (13 sor).
3. *„A velünk kapcsolatban álló beszállítók közül kiknek nincs webhelyük?"*
   A megoldás itt található: CH06_Vendors_With_No_Website (4 sor).

Entertainment Agency adatbázis

1. *„Mutass egy listát a 2007 októberében megtartott rendezvényekről."*
   (Tipp: ezt a problémát egy értéktartományt egy másik tartománnyal – október első és utolsó napja – összehasonlítva oldhatjuk meg.)
   A megoldás itt található: CH06_October_2007_Engagements (23 sor).
2. *„Mutasd meg azokat a 2007. októberi rendezvényeket, amelyek dél és délután 5 óra között kezdődtek!"*
   A megoldás itt található: CH06_October_Dates_Between_Noon_and_Five (17 sor).
3. *„Sorold fel az összes olyan rendezvényt, amelynek a kezdő és záró dátuma azonos!"*
   A megoldás itt található: CH06_Single_Day_Engagements (5 sor).

School Scheduling adatbázis

1. *„Mutasd meg azokat a dolgozókat, akik postafiókot használnak címként!"*
   A megoldás itt található: CH06_Staff_Using_POBoxes (3 sor).
2. *„Mutasd meg, mely hallgatók laknak a Pacific Northwesten kívül!"*
   A megoldás itt található: CH06_Students_Residing_Outside_PNW (5 sor).
3. *„Sorold fel az összes olyan tantárgyat, amelynek a tárgykódja az 'MUS' karakterekkel kezdődik!"*
   A megoldás itt található: CH06_Subjects_With_MUS_In_SubjectCode (4 sor).

Bowling League adatbázis

1. *„Adj egy listát a 2007 szeptemberében tartott versenyekről!"*
   A megoldás itt található: CH06_September_2007_Tournament_Schedule (4 sor).
2. *„Mi az időbeosztása a versenyeknek a Bolero, a Red Rooster és a Thunderbird Lanes pályán?"*
   A megoldás itt található: CH06_Eastside_Tournaments (6 sor).
3. *„Sorold fel azokat a tekézőket, akik az Eastside-on laknak (Bellevue, Bothell, Duvall, Redmond, Woodinville), és akiknek a csapata az 5-ös, a 6-os, a 7-es vagy a 8-as!"*
   (Tipp: a városok listájához az IN, a csapatok számához pedig a BETWEEN műveletet használjuk.)
   A megoldás itt található: CH06_Eastside_Bowlers_On_Teams_5_Through_8 (9 sor).

Recipes adatbázis

1. *„Sorold fel az összes olyan főétel (receptosztályuk az 1-es) receptjét, amelyikhez tartozik megjegyzés!"*
   A megoldás itt található: CH06_Main_Courses_With_Notes (4 sor).
2. *„Mutasd meg az első öt receptet!"*
   (Tipp: használjuk a BETWEEN műveletet a tábla elsődleges kulcsán.)
   A megoldás itt található: CH06_First_5_Recipes (5 sor).

Többtáblás lekérdezések

7

Halmazokban gondolkodni

„Vidám ebédet ad kis asztal, nagy szivesség"
– William Shakespeare
Tévedések vígjátéka, 3. felvonás, 1. szín, fordította: Szász Imre

A fejezet témakörei

- Mik azok a halmazok?
- Halmazműveletek
- Metszet
- Különbség
- Unió
- Halmazműveletek az SQL-ben
- Összefoglalás

Az eddigiekben láttuk, hogyan gyűjthetjük halmazba az adatokat a megfelelő oszlopokon végrehajtott lekérdezésekkel vagy oszlopokat tartalmazó kifejezésekkel (SELECT), hogyan rendezhetjük a sorokat (ORDER BY), és szűrhetjük a kapott sorokat (WHERE). Eddig arra összpontosítottunk, hogy egyetlen táblával végezzünk el alapvető feladatokat. Mi van azonban akkor, ha több táblában tárolt adatot akarunk lekérdezni? Mit kell tennünk, ha össze akarunk hasonlítani vagy szembe akarunk helyezni egy vagy több táblából származó adathalmazokat?

Úgy könnyű ételt készíteni, ha egy zacskó krumplit vagy egy csomag sárgarépát csak hámozni, szeletelni és kockázni kell. Mostantól fogva a problémák nagy része, amelyeknek megmutatjuk a megoldását, *több* tábla lekérdezésével jár majd. Nem csak azt mutatjuk meg, hogyan kell jó ragut készíteni, hanem azt is, hogyan legyünk jó szakácsok!

Mielőtt belevetnénk magunkat a fejezet mélységeibe, meg kell jegyeznünk, hogy ez a rész azokról a *fogalmakról* szól, amelyeket meg kell értenünk ahhoz, hogy össze tudjunk kapcsolni két vagy több adathalmazt. Ezen kívül rövid áttekintést nyújtunk az ide kapcso-

lódó, az SQL-szabványban lefektetett utasításformákról is, amelyek közvetlenül támogatják ezeket a fogalmakat. Elöljáróban azonban meg kell említenünk, hogy az SQL sok, jelenleg forgalomban lévő kereskedelmi megvalósítása még nem támogatja ezt a „tiszta" nyelvtant. A későbbi fejezetekben megmutatjuk, hogyan lehet az itt tanult fogalmakat a legtöbb nagy adatbázisrendszer által széleskörűen támogatott SQL-utasításkészlet segítségével megvalósítani. E helyen nem a szabályok betűjét, hanem azok szellemét tartjuk szem előtt.

Mik azok a halmazok?

Ha az 1960-as évek közepe után voltunk tinédzserek, bizonyára tanultunk halmazelméletet a matematika órákon. (Emlékszünk még?) Az alapok elsajátítása során bizonyára felmerült a kérdés: hol hasznosíthatjuk majd mindezt?

Most a relációs adatbázisokról és az SQL-nek nevezett cikornyás nyelvről próbálunk meg új ismereteket szerezni, hogy alkalmazásokat fejlesszünk, különböző problémákat oldjunk meg, vagy csak egyszerűen választ kapjunk a kérdéseinkre. Figyelemmel kísértük az algebra órákat? Ha igen, az SQL-lel könnyebben meg fogjuk tudni oldani a problémákat – különösen az összetetteket.

Tulajdonképpen már a könyv elejétől fogva halmazokkal dolgozunk. Az 1. fejezetben a relációs adatbázisok alapvető szerkezetéről tanultunk: a táblák rekordokat tartalmaznak, ezeket pedig egy vagy több mező osztja fel. (Emlékezzünk rá, hogy az SQL-ben a rekordok sorokat, a mezők oszlopokat jelentenek). Minden egyes tábla az adatbázisunkban egy egyedtípus tulajdonságainak *halmaza*. A 2. fejezetben megtanultuk, hogyan biztosítsuk, hogy az adatbázis szerkezete egészséges legyen. Minden tábla olyan adatok *halmazát* kell, hogy tartalmazza, amely egyetlen egyedtípust vagy műveletet határoz meg.

A 4. fejezetben láttuk, hogyan kaphatjuk meg egy egyszerű SELECT utasítással olyan adatok *halmazát*, amely egyetlen tábla meghatározott sorait tartalmazza, és hogyan rendezhetjük ezeket az *eredményhalmazokat*. Az 5. fejezet arról szólt, hogy miként kérdezhetjük le kifejezések segítségével több oszlopból adatok *halmazát*, a 6. fejezetben pedig megtanultuk tovább szűkíteni a táblákból kapott *adathalmazt* a lekérdezéshez fűzött szűrő (WHERE záradék) segítségével.

Látható, hogy egy halmazt akár egyetlen tábla egyetlen oszlopa és egyetlen sora is meghatározhat. Valójában olyan SQL-lekérdezést is írhatunk, amely nem ad vissza egyetlen sort sem, csak egy üres halmazt. Néha hasznos lehet rájönnünk, hogy valami *nem* létezik. Egy halmazt azonban több tábla több oszlopából (beleértve a kifejezések segítségével létrehozott oszlopokat) és sorából is megkaphatunk. Az eredményhalmaz minden sora a halmaz *eleme*. Az oszlopokban szereplő értékek az elemeket leíró *jellemzők*, amelyek olyan adatok, amelyek meghatározzák a halmaz adott elemét. A következő néhány fejezetben megmutatjuk,

hogyan lehet több adathalmazból információt lekérdezni, majd ezeket a halmazokat összekapcsolni, hogy bonyolultabb kérdéseket is megválaszolhassunk. Először azonban meg kell tanulnunk pár dolgot a halmazokról és azok logikai kapcsolatairól.

Halmazműveletek

Az 1. fejezetben láttuk, hogyan alkotta meg Dr. E. F. Codd a relációs adatmodellt, amin a legtöbb modern adatbázis és az SQL alapjai nyugszanak. A modell alappillérjeit a matematika két ága (a halmazelmélet és az elsőrendű predikátumkalkulus) képezte.

Miután elsajátítottuk, hogyan kaphatunk választ egyetlen táblából, azt kell megtanulnunk, hogy miként használhatjuk adatok eredményhalmazát bonyolultabb feladatok megoldására. Ezek az összetett feladatok általában valamelyik halmazművelet alkalmazását követelik meg, hogy össze lehessen kapcsolni két vagy több táblából származó adatot. Néha két különböző eredményhalmazt kell létrehoznunk ugyanabból a táblából, majd újra össze kell kapcsolnunk azokat, hogy megkapjuk a választ a kérdésünkre.

A három leggyakoribb halmazművelet a következő:

- **Metszet** – Akkor használjuk, ha két vagy több különböző halmaz közös elemeire vagyunk kíváncsiak: „Mutasd meg a bárányhúst és rizst tartalmazó recepteket!" „Mutasd meg azokat a vevőket, akik biciklit és bukósisakot rendeltek!"
- **Különbség** – Akkor használjuk, ha azokat az elemeket akarjuk megtalálni, amelyek benne vannak egy halmazban, de nincsenek benne egyik másikban sem: „Mutasd meg azokat a recepteket, amelyekben szerepel bárányhús, *de nincs* rizs!" „Mutasd meg azokat a vevőket, akik rendeltek biciklit, *de nem* rendeltek bukósisakot!"
- **Unió** – Akkor használjuk, ha két vagy több hasonló halmazt egyesítünk: „Mutass meg minden receptet, amelyben bárányhús *vagy* rizs szerepel!" „Mutasd meg azokat a vevőket, akik biciklit *vagy* bukósisakot rendeltek!"

A következő három rész ezt a három halmazműveletet fogja bővebben tárgyalni, vagyis azokat, amelyeket már az általános iskolai matematika órákon megtanultunk. Arról, hogy ezek a műveletek hogyan valósíthatók meg „tiszta" SQL nyelven, a *Halmazműveletek az SQL-ben* című rész nyújt áttekintést.

Metszet

Nem, most nem rézmetszetekről lesz szó. Két halmaz *metszete* azok közös elemeit tartalmazza. Először halmazelméleti szempontból vesszük szemügyre, hogy mit értünk metszet alatt, majd megnézzük, hogy metszetek alkalmazásával hogyan oldhatunk meg üzleti problémákat.

Metszet a halmazelméletben

A metszet nagyon hatékony matematikai eszköz, amit a kutatók és mérnökök is gyakran használnak. Kutatóként érdekeltek lehetünk abban, hogy megtaláljuk két kémiai vagy fizikai mérési sor közös pontjait. Például egy gyógyszerkutatónak van két vegyületcsoportja, amelyek – úgy tűnik – előnyös tulajdonságokkal bírnak. Ha sikerül megtalálni a két csoportban a közös részt (a metszetet), az hozzásegíthet ahhoz, hogy megtudjuk, mi teszi a két vegyületcsoportot hatásossá. Vagy például egy mérnököt az érdekelhet, hogy mi a merev, de törékeny és a puha, de ellenálló ötvözetek metszete.

Most nézzük meg közelebbről, hogyan működik a metszetképzés. Vegyünk két számhalmazt, ahol minden egyes szám a halmaz eleme. Az első számhalmaz a következő elemeket tartalmazza:

1, 5, 8, 9, 32, 55, 78

A második számhalmaz elemei:

3, 7, 8, 22, 55, 71, 99

A két számhalmaz metszetét azok a számok alkotják, amelyek mindkét halmazban előfordulnak:

8, 55

A halmazok egyes elemei nem csak egyszerű értékek lehetnek. Valójában amikor valamilyen problémát oldunk meg az SQL-lel, valószínűleg sorok halmazával dolgozunk majd.

A halmazelméletben, ha egy elem több mint egy egyszerű szám vagy más érték, a halmaz minden eleme (vagy egyedtípusa) több jellemzővel (vagy bittel) bír, amelyek a halmaz minden elemét jellemezni tudják. Kedvenc ragunk receptje például egy összetett eleme az összes recept halmazának, amely sok különböző hozzávalót foglal magában. Az összetett „ragu" halmazelem minden hozzávalója az elem egy-egy jellemzője.

Összetett elemeket tartalmazó halmazok metszetének megtalálásához azokat az elemeket kell megkeresnünk, amelyek minden jellemzőjükben megegyeznek. Ez azt jelenti, hogy minden összehasonlítandó halmazelem azonos számú és típusú jellemzővel kell, hogy rendelkezzen. Vegyük példának az alábbi, összetett halmazt, amelyben minden sor a halmaz (a ragu receptje) egy elemét jelképezi, és minden oszlop egy bizonyos jellemzőt határoz meg:

Burgonya	Víz	Bárányhús	Zöldborsó
Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Tészta	Víz	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Tészta	Víz	Sertéshús	Hagyma

A második halmaz a következőképpen néz ki:

Burgonya	Víz	Bárányhús	Hagyma
Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Tészta	Zöldség alaplé	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Bab	Víz	Sertéshús	Hagyma

A két halmaz metszetébe egyetlen elem kerül, amelynek minden jellemzője megegyezik a két halmazban:

Burgonya	Marha alaplé	Marhahús	Fejes káposzta

Eredményhalmazok metszete

Ha a korábbi példák táblázatok sorainak vagy eredményhalmaznak tűnnek, akkor jó nyomon járunk. Amikor egy SQL-lekérdezés eredményeképpen kapott adathalmaz soraival foglalkozunk, az oszlopok az egyes jellemzőket határozzák meg. Tegyük fel, hogy a következő sorokat egy lekérdezés eredményeképpen kaptuk (a receptek John szakácskönyvéből valók):

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Csirkeragu	Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Vega ragu	Tészta	Víz	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Tészta	Víz	Sertéshús	Hagyma

Egy másik eredményhalmaz a következőképpen néz ki (ezek a receptek Mike szakácskönyvében szerepelnek):

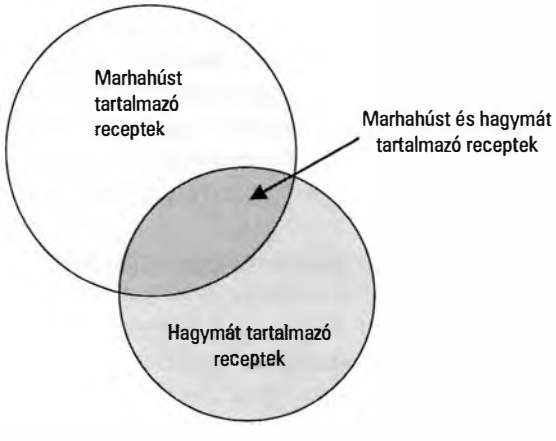
Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Pulykaragu	Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Vega ragu	Tészta	Zöldség	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Bab	Víz	Sertéshús	Hagyma

A két halmaz metszetébe az a két elem kerül, amelyek a két halmazban minden jellemzőjükben megegyeznek. Ez az a két recept, ami Mike és John szakácskönyvében azonos:

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta

Halmazdiagram segítségével néha könnyebb megérteni, hogyan működik a metszetképzés. A ***halmazdiagram*** elegáns, ám mégis egyszerű módja annak, hogy adathalmazokat jelenítsünk meg, és grafikusan ábrázoljuk a halmazok metszetét vagy átfedését. Erre a fajta diagramra az Euler- vagy Venn-diagram elnevezés is használatos. (Leonard Euler svájci matematikus a 18. században élt. John Venn a Cambridge-i Egyetem kutató ösztöndíjasaként 1880-ban megjelent cikkében használta a logikai diagramoknak ezt a különleges fajtáját. Láthatjuk tehát, hogy „halmazokban gondolkodni" nem éppen mai találmány.)

Tegyük fel, hogy van egy kedvenc recepteket tartalmazó adatbázisunk. Szeretjük, ahogy a hagyma kiemeli a marhahús ízét, ezért minket minden olyan recept érdekel, amiben van hagyma és marhahús. A 7.1. ábrán látható halmazdiagram segít a problémamegoldás szemléltetésében.



7.1. ábra

Marhahúst és hagymát tartalmazó receptek

A felső kör a marhahúst tartalmazó recepteket ábrázolja, az alsó a hagymát tartalmazókat. A mindkét hozzávalót tartalmazó receptek ott vannak, ahol a két kör átfedi egymást (a két halmaz metszetében). Ahogy az várható, az SQL-lel először lekérdezünk minden marhahúst

tartalmazó receptet, majd egy második lekérdezéssel azokat a recepteket gyűjtjük össze, amelyek hagymát tartalmaznak. Később látni fogjuk, hogy a két lekérdezés összekapcsolásához használhatunk egy külön SQL-parancsot (INTERSECT), hogy megkapjuk a végső választ.

Igen, tudjuk, mire gondol az Olvasó. Ha a receptkönyv úgy néz ki, mint a fenti példában, akkor a következőt is mondhatnánk:

„Mutasd meg azokat a recepteket, amelyek húsból marhahúst, zöldségféléből pedig hagymát tartalmaznak!"

Fordítás Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is onions
(Válaszd ki azokat a recepteket a receptek táblájából, amelyeknek a hozzávalói közül a hús marhahús, a zöldségféle pedig hagyma.)

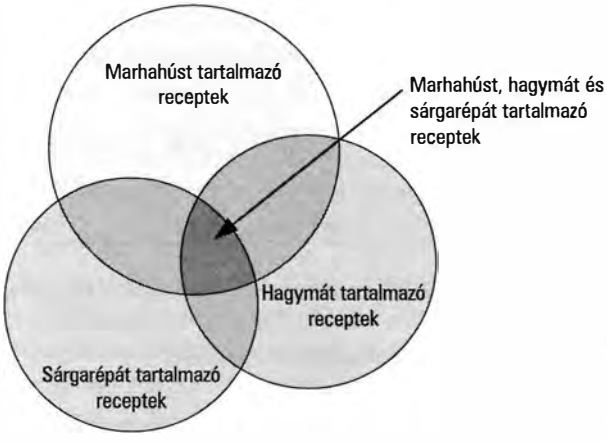
Tisztázás Select ~~the~~ recipe name from ~~the~~ recipes ~~table~~ where meat ingredient ~~is~~ = beef and vegetable ingredient ~~is~~ = onions

SQL
```
SELECT RecipeName
FROM Recipes
WHERE MeatIngredient = 'Beef'
   AND VegetableIngredient = 'Onions'
```

Itt álljunk meg egy pillanatra! Ha emlékszünk a második fejezetben tanultakra, akkor tudjuk, hogy valószínűleg egyetlen Recipes (Receptek) tábla nem fogja megfűszerezni az életünket. Mi van azokkal a receptekkel, amelyek a húson és a zöldségfélén kívül más hozzávalót is tartalmaznak? Mi van azzal a ténnyel, hogy egyes receptek sok hozzávalót tartalmaznak, míg mások keveset? Egy helyesen megtervezett receptadatbázisban van egy Recipe_Ingredients (Recepthozzávalók) tábla, amely receptenként és hozzávalónként egy sorral rendelkezik. A hozzávalók minden sorában csak egyetlen hozzávaló szerepelhet, így nem fordulhat elő, hogy egy sorban egyszerre szerepel marhahús és hagyma. Először meg kell keresnünk a marhahúst tartalmazó sorokat, majd a hagymát tartalmazó sorokat, végül a RecipeID (Receptazonosító) alapján ezek metszetét kell képeznünk. (Ha nem világos, mi a kifogásunk az előző Recipes tábla ellen, lapozzunk vissza a 2. fejezethez!)

Hogyan oldunk meg ennél összetettebb feladatokat? Mondjuk, az eddigiekhez még sárgarépát is szeretnénk adni. A megoldást a 7.2. ábrán látható halmazdiagram szemlélteti.

Ráéreztünk az ízére? Annyit kell megjegyeznünk, hogy ha összetett feltételek alapján kell valamilyen feladatot megoldanunk, a halmazdiagram értékes segítséget nyújthat az SQL-eredményhalmazok metszetének szemléltetésében.



7.2. ábra
Marhahúst, hagymát és sárgarépát tartalmazó receptek

Metszetképzéssel megoldható feladatok

Mint bizonyára kitaláltuk, metszetképzéssel két vagy több adathalmaz egyező elemeit határozhatjuk meg. Az alábbiakban néhány példával szemléltetjük, hogy a mintaadatbázisokból származó adatokkal hogyan oldhatunk meg feladatokat metszetképzés segítségével:

„Mutasd meg az ugyanolyan nevű vevőket és alkalmazottakat!"
„Keresd meg azokat a vevőket, akik biciklit és bukósisakot is rendeltek!"
„Sorold fel azokat az előadókat, akik a Bonnicksen és a Rosales nevű megrendelőknél játszottak!"
„Mutasd meg azokat a hallgatókat, akiknek rajzból és számítástechnikából legalább 85 az átlagpontszámuk!"
„Keresd meg azokat a tekejátékosokat, akiknek a tiszta pontszáma legalább 155 volt a Thunderbird és a Bolero pályán!"
„Mutasd meg a marhahúst és fokhagymát tartalmazó recepteket!"

A metszetekre vonatkozó korlátozás, hogy minden eredményhalmazban az összes oszlop értékének egyeznie kell. Ez jól működik, ha ugyanabból a táblából származó két vagy több halmazból készítünk metszetet (például azoknak a vevőknek a halmazaiból, akik biciklit, illetve bukósisakot rendeltek). Akkor is jól működik, ha különböző táblákból származó, de azonos oszlopokkal rendelkező halmazokból (például a vevők és az alkalmazottak neve) képzünk metszetet. Számos esetben azonban olyan megoldást keresünk, ahol az egyes halmazoknak csak néhány oszlopa egyezik meg. Az ilyen jellegű problémákra az SQL a JOIN (összekapcsolás) műveletet biztosítja, amely a kulcstulajdonságokból képez metszetet. Lássunk néhány problémát, amelynek a megoldásában a JOIN segíthet:

„Mutasd meg azokat a vevőket és alkalmazottakat, akik ugyanabban a városban élnek!" (JOIN a város nevére alkalmazva)
„Sorold fel a megrendelőket és az előjegyzett előadókat!" (JOIN a rendezvény számára alkalmazva)

„Keresd meg azokat az ügynököket és előadókat, akiknek ugyanaz az irányítószámuk!" (JOIN az irányítószámra alkalmazva)
„Mutasd meg azokat a hallgatókat, akiknek a keresztneve azonos a tanáráéval!" (JOIN a keresztnévre alkalmazva)
„Keresd meg azokat a tekejátékosokat, akik ugyanabban a csapatban játszanak!" (JOIN a csapat azonosítójára alkalmazva)
„Keresd meg a sárgarépát tartalmazó recepteket!" (JOIN a hozzávalók azonosítójára alkalmazva)

Egyet se féljünk, a következő fejezetben megmutatjuk, hogyan kell ilyen (és ennél bonyolultabb) problémákat JOIN műveletekkel megoldani. Mivel az SQL kevés kereskedelmi megvalósítása támogatja az INTERSECT-et, azt is megmutatjuk, hogy miként oldhatunk meg olyan feladatokat, amelyekhez egyébként az INTERSECT lenne szükséges.

Különbség

Mennyi 21 és 10 között a különbség? Ha a válaszunk 11, akkor jó nyomon járunk. A különbségképzés (vagy kivonás) során vesszük az egyik halmaz elemeit, és eltávolítjuk közülük a másik halmazzal közös elemeket. Azok az elemek maradnak, amelyek csak az egyik halmazban vannak benne, de *nincsenek* benne a másikban. (Amint később látni fogjuk, az SQL-szabványban erre használt kulcsszó az EXCEPT.)

Különbség a halmazelméletben

A különbségképzés szintén hatékony matematikai eszköz. Kutatóként érdekelhetnek olyan kérdések, hogy mi két fizikai vagy kémiai mérési sor különbsége. Például gyógyszerkutatóként lehet két hasonló vegyületcsoportunk, de az egyik rendelkezik egy bizonyos előnyös hatással, a másik viszont nem. A molekulák közötti különbség segíthet felfedni, hogy az egyik csoport tagjai miért rendelkeznek előnyös tulajdonsággal, a másik csoport elemei pedig miért nem. Mérnökként lehet két hasonló tervünk, de az egyik jobban működik, mint a másik. A két terv közötti különbség megtalálása alapvető fontosságú lehet, hogy ki tudjuk szűrni a következő épületekből a szerkezeti hibákat.

Nézzük meg a következő két számhalmaz példáján, hogy miként működik a különbségképzés! Az első számhalmaz elemei a következők:

1, 5, 8, 9, 32, 55, 78

A második számhalmaz elemei:

3, 7, 8, 22, 55, 71, 99

Az első halmaz elemeiből kivonva a második halmaz elemeit megkapjuk azokat a tagokat, amelyek benne vannak az első halmazban, de nincsenek benne a másodikban:

1, 5, 7, 9, 32, 78

Az előző különbségképzést meg is fordíthatjuk, hogy a második halmaz elemeiből vonjuk ki az első halmaz elemeit. Az eredmény:

3, 7, 22, 71, 99

Az egyes halmazelemek nem szükségszerűen egyszerű értékek. Valójában sokkal valószínűbb, hogy sorok halmazaival fogunk foglalkozni, ha az SQL-lel oldunk meg különböző feladatokat.

A fejezet elején már mondtuk, hogy ha egy halmazelem több mint egy szám vagy egy egyszerű érték, akkor a halmazelemek több jellemzővel írhatók le (ezek a halmazelemek tulajdonságait jellemző bitek). A kedvenc ragunk receptje például az összes receptet tartalmazó halmaz összetett eleme, ahol a halmaz több különböző hozzávalót tartalmaz. Minden hozzávalóra úgy gondolhatunk, mint az összetett ragu elem egy-egy jellemzőjére.

Összetett elemeket tartalmazó halmazok különbségét úgy képezzük, hogy először megkeressük azokat az elemeket a második halmazból, amelyek minden tulajdonságukban megegyeznek az első halmaz elemeivel. Ne feledjük, mindkét halmaz összehasonlításra váró megfelelő elemének azonos számú és típusú jellemzővel kell rendelkeznie. Ez után a két halmaz egyező elemeit eltávolítjuk az első halmaz elemei közül, így a különbséget kapjuk eredményül. Tegyük fel, hogy van egy olyan összetett halmazunk, mint amilyet alább láthatunk. Minden sor egy halmazelemet (ragureceptet) jelöl, míg az egyes oszlopok egy-egy jellemzőt (egy hozzávalót) határoznak meg.

Burgonya	Víz	Bárányhús	Zöldborsó
Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Tészta	Víz	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Tészta	Víz	Sertéshús	Hagyma

A második halmaz a következőképpen néz ki:

Burgonya	Víz	Bárányhús	Hagyma
Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Tészta	Zöldség alaplé	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Bab	Víz	Sertéshús	Hagyma

Az első és a második halmaz különbségét azok az elemek alkotják, amelyek benne vannak az első halmazban, de nincsenek benne a másodikban:

Burgonya	Víz	Bárányhús	Zöldborsó
Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Tészta	Víz	Tofu	Cukorborsó
Tészta	Víz	Sertéshús	Hagyma

Eredményhalmazok különbsége

Amikor SQL-lel lekérdezett sorokkal dolgozunk, a jellemzők az egyes oszlopok. Tegyük fel, hogy a következő sorokat egy lekérdezés eredményeképpen kaptuk (a receptek John szakácskönyvéből valók):

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Csirkeragu	Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Vega ragu	Tészta	Víz	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Tészta	Víz	Sertéshús	Hagyma

Egy második lekérdezés eredményei lentebb találhatók (ezek a receptek Mike szakácskönyvéből valók):

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Pulykaragu	Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Vega ragu	Tészta	Zöldség	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Bab	Víz	Sertéshús	Hagyma

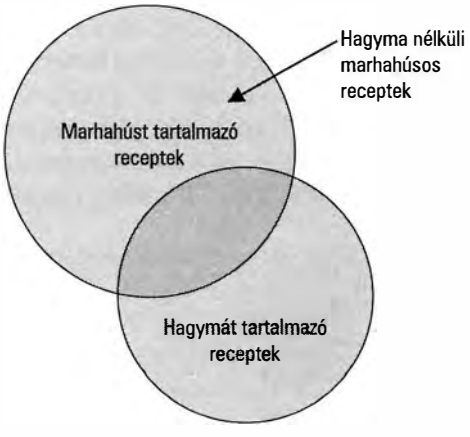
John és Mike receptjeinek különbsége (John receptjei közül kivonjuk Mike receptjeit) azok a receptek, amelyek benne vannak John szakácskönyvében, de *nincsenek* benne Mike-éban:

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Vega ragu	Tészta	Víz	Tofu	Cukorborsó
Sertésragu	Tészta	Víz	Sertéshús	Hagyma

Meg is fordíthatjuk a feladatot: tegyük fel, hogy azok a receptek érdekelnek, amelyek benne vannak Mike szakácskönyvében, de nincsenek benne Johnéban. Íme az eredmény:

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Vega ragu	Tészta	Víz	Tofu	Cukorborsó
Sertésragu	Tészta	Víz	Sertéshús	Hagyma

Itt is alkalmazhatunk halmazdiagramot, hogy szemléltessük, hogyan működik a különbségképzés. Tegyük fel, hogy van egy kedvenc recepteket tartalmazó adatbázisunk, és mivel nem szeretjük a hagymás marhahúst, olyan recepteket keresünk, amelyekben van marhahús, de nincs hagyma. A 7.3. ábra szemlélteti, hogyan oldhatjuk meg ezt a feladatot.



7.3. ábra

Hagyma nélküli marhahúsos receptek

A felső teljes kör a marhahúst tartalmazó recepteket ábrázolja, míg az alsó teljes kör a hagymát tartalmazókat. Bizonyára emlékszünk rá az INTERSECT tárgyalásából, hogy ahol a két kör átfedi egymást, ott vannak a mindkét hozzávalót tartalmazó receptek. A felső kör világosszürkével színezett, az átfedésen kívüli része a hagyma nélküli marhahúsos receptek halmazát jelöli. Hasonlóképpen, az alsó kör átfedésen kívüli része azokat a recepteket jelképezi, amelyek tartalmaznak hagymát, de nincs bennük marhahús.

Bizonyára tudjuk, hogy először a marhahúst tartalmazó recepteket kell egy SQL-paranccsal lekérdeznünk, majd egy következő SQL-paranccsal a hagymát tartalmazókat. (Mint később látni fogjuk ebben a fejezetben, a két lekérdezést egy külön SQL-kulcsszóval, az EXCEPT-tel kapcsolhatjuk össze, hogy megkapjuk a végső választ.)

Megint csapdába estünk? (*Valóban* elolvastuk a 2. fejezetet, ugye?) Ha a receptes táblánk a korábbi példákhoz hasonló, akkor gondolhatnánk, hogy nyugodtan mondhatjuk a következőt:

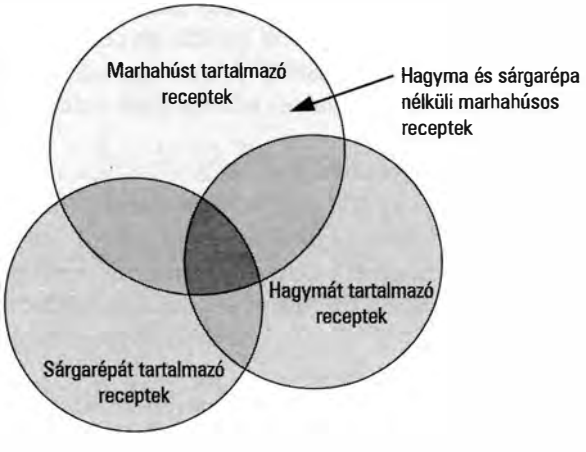
„Mutasd meg azokat a recepteket, amelyek húsféléből marhahúst tartalmaznak, de a zöldségfélékből nincs bennük hagyma!"

Fordítás Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is not onions
(Válaszd ki azokat a recepteket a receptek táblájából, ahol a hozzávalók közül a hús marhahús, de a zöldségféle nem hagyma.)

Tisztázás Select ~~the~~ recipe name from ~~the~~ recipes ~~table~~ where meat ingredient ~~is~~ = beef and vegetable ingredient ~~is not~~ <> onions

SQL
```
SELECT RecipeName
FROM Recipes
WHERE MeatIngredient = 'Beef'
   AND VegetableIngredient <> 'Onions'
```

Tehát még egyszer: amint azt a 2. fejezetben megtanultuk, egyetlen Recipes tábla alkalmazása nem valami forró gondolat. Mi van azokkal a receptekkel, amelyek húson és zöldségen kívül más hozzávalót is tartalmaznak? Mi van azzal a ténnyel, hogy egyes receptek sok, míg mások csak kevés hozzávalót tartalmaznak? Egy helyes módon megtervezett Recipes adatbázis tartalmaz egy külön Recipe_Ingredients táblát, ami receptenként és hozzávalónként egy sort tartalmaz. A hozzávalók minden sorában csak egy hozzávaló szerepel, így egyetlen sor sem tartalmazhatja a marhahúst és hagymát egyszerre. Először meg kell keresnünk minden marhahúst tartalmazó sort, majd minden hagymát tartalmazó sort, végül a RecipeID alapján ezek különbségét kell képeznünk.



7.4. ábra

A hagyma és sárgarépa nélküli marhahúsos receptek

Hogyan oldhatunk meg összetettebb feladatokat? Tegyük fel, hogy a sárgarépát is utáljuk. A megoldást a 7.4. ábrán feltüntetett halmazdiagram szemlélteti. Először meg kell keresnünk a marhahúst tartalmazó recepteket, majd ennek a különbségét kell képeznünk a hagymát vagy sárgarépát tartalmazó receptek halmazával. Az eredményből ez után ki kell vonnunk a megmaradt halmazt (hagyma vagy sárgarépa), hogy csak azok a receptek maradjanak, amelyekben van marhahús, de nincs sárgarépa és hagyma (a felső kör világos színű területe).

Különbségképzéssel megoldható feladatok

A metszetképzéstől eltérően (ahol két halmaz közös elemeit keressük) a különbségképzés során azokat az elemeket keressük, amelyek benne vannak az egyik halmazban, de nincsenek benne egyik másikban sem. A következőkben a mintaadatbázisokból néhány, különbségképzéssel megoldható példát mutatunk be:

„Mutasd meg azokat a vevőket, akiknek az alkalmazottaktól eltérő nevük van!"
„Keresd meg azokat a vásárlókat, akik rendeltek biciklit, de nem rendeltek bukósisakot!"
„Mutasd meg azokat az előadókat, akik a Bonnicksen nevű megrendelőnél játszottak, de nem játszottak a Rosales nevű megrendelőnél!"
„Mutasd meg azokat a hallgatókat, akiknek rajzból az átlagos pontszámuk legalább 85, de számítástechnikából nem értek el legalább 85 pontos átlagot!"
„Keresd meg azokat a tekejátékosokat, akiknek a tiszta pontszáma legalább 155 volt a Thunderbird pályán, de ennél kevesebb volt a Bolero pályán!"
„Mutasd meg azokat a recepteket, amelyek tartalmaznak marhahúst, de nincs bennük fokhagyma!"

A különbségképzés korlátja, hogy az értékeknek az eredményhalmazok minden oszlopában egyezniük kell. Ez jól működik, ha ugyanabból a táblából képzett halmazok között keressük a különbségeket (például külön azokat a vevőket, akik biciklit, és azokat, akik bukósisakot rendeltek). Akkor is jól működik, ha hasonló oszlopokat tartalmazó táblákból származó halmazok (például a vevők és az alkalmazottak neve) különbségét vesszük.

Sok esetben azonban a megoldáshoz a halmazok egyes oszlopainak értékei csak részben kell, hogy egyezzenek. Ilyen problémákra biztosítja az SQL az OUTER JOIN (külső összekapcsolás) műveletet, amely egy olyan metszetet képez a kulcstulajdonságokból, ami magában foglalja azokat a sorokat az egyik vagy mindkét halmazból, amelyek nem egyeznek meg. A következőkben néhány példát láthatunk az OUTER JOIN-nal megoldható feladatokra:

„Mutasd meg azokat a vevőket, akik nem ugyanabban a városban élnek, mint az alkalmazottak!" (OUTER JOIN a város nevére alkalmazva)
„Sorold fel a megrendelőket és azokat az előadókat, akiket nem foglaltak le!" (OUTER JOIN a rendezvény számára alkalmazva.)

„Keresd meg azokat az ügynököket, akiknek az irányítószáma minden előadóétől különbözik!" (OUTER JOIN az irányítószámra alkalmazva)
„Mutasd meg azokat a hallgatókat, akiknek minden tanár nevétől eltérő keresztnevük van!" (OUTER JOIN a keresztnévre alkalmazva)
„Keresd meg azokat a tekejátékosokat, akiknek az átlaga legalább 150, és soha nem volt olyan játékuk, hogy legalább 125 pontot ne értek volna el!" (OUTER JOIN a tekejátékos két különböző táblából származó azonosítójára alkalmazva)
„Mutasd meg azoknak a recepteknek az összes hozzávalóját, amelyek nem tartalmaznak sárgarépát!" (OUTER JOIN a recept azonosítójára alkalmazva)

Ne aggódjunk, a 9. fejezetben látni fogjuk, hogyan lehet megoldani a fenti (és még további) feladatokat az OUTER JOIN alkalmazásával. Mivel az SQL kevés kereskedelmi megvalósítása támogatja az EXCEPT-et (ami a különbségképzés kulcsszava), meg fogjuk mutatni, hogy az OUTER JOIN-nal hogyan lehet olyan feladatokat megoldani, amelyekhez egyébként az EXCEPT lenne szükséges.

Unió

Eddig arról volt szó, hogy miként kereshetjük meg két halmaz közös elemeit (a két halmaz metszetét), valamint a halmazok különböző elemeit (a halmazok különbségét).
A halmazműveletek harmadik fajtája két halmaz összeadását (unió) foglalja magában.

Unió a halmazelméletben

Az *unió* lehetővé teszi, hogy két, hasonló adatokat tartalmazó halmazt összeadjunk. Kutatóként lehet, hogy fizikai vagy kémiai mérési eredmények két halmazát akarjuk egyesíteni. Például egy gyógyszerkutató két halmazba gyűjtheti azokat a vegyületeket, amelyek valamilyen kedvező tulajdonsággal bírnak. A kémikus egyesítheti a két halmazt, hogy a hatékony vegyületekről egyetlen listát állítson össze.

Nézzük meg, hogy működik az unióképzés két számhalmazon. Az első halmazba tartozó számok a következők:

1, 5, 8, 9, 32, 55, 78

A második számhalmaz tagjai:

3, 7, 8, 22, 55, 71, 99

A két számhalmaz uniójába azok a számok tartoznak, amelyeket a két számhalmaz egyesítésével kapunk:

1, 5, 8, 9, 32, 55, 78, 3, 7, 22, 71, 99

Figyeljük meg, hogy a két halmaz közös elemei, a 8-as és 5-ös, csak egyszer szerepelnek az eredményben. Az eredményhalmazban a számoknak nincs szükségszerűen kitüntetett sorrendje. Amikor az adatbázisrendszerünkben kiadjuk a UNION (unió) parancsot,

a kapott értékek nem szükségszerűen lesznek sorrendben, feltéve, hogy nem adjuk meg kifejezetten az ORDER BY záradékot. Az SQL-ben lehetőségünk van a UNION ALL paranccsal is futtatni a lekérdezést, ha a kétszer szereplő elemeket is látni szeretnénk.

Az egyes halmazok elemei nem csak puszta értékek lehetnek. Valójában alighanem sorok halmazával fogunk foglalkozni, amikor az SQL-lel dolgozunk.

Két vagy több halmaz összetett elemeinek unióját akkor határozhatjuk meg, ha az egyesítendő halmazok minden eleme azonos számú és típusú jellemzővel bír. Tegyük fel, hogy van egy a lentihez hasonló összetett halmazunk. Egy sor egy halmazelemet (ragureceptet) jelöl, míg az egyes oszlopok egy-egy jellemzőt (egy hozzávalót) határoznak meg.

Burgonya	Víz	Bárányhús	Hagyma
Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Tészta	Víz	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Tészta	Víz	Sertéshús	Hagyma

A második halmaz a következőképpen néz ki:

Burgonya	Víz	Bárányhús	Hagyma
Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Tészta	Zöldség alaplé	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Bab	Víz	Sertéshús	Hagyma

A két halmaz uniója mindkét halmaz elemeit magában foglalja. A kétszer szereplő elemeket nem tüntettük fel.

Burgonya	Víz	Bárányhús	Zöldborsó
Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Tészta	Víz	Tofu	Cukorborsó
Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Tészta	Víz	Sertéshús	Hagyma
Burgonya	Víz	Bárányhús	Hagyma
Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Tészta	Zöldség alaplé	Tofu	Cukorborsó
Bab	Víz	Sertéshús	Hagyma

Eredményhalmazok uniója

Az összetett elemeket kis lépés választja el az SQL-eredményhalmazok soraitól. Amikor egy adathalmazból az SQL segítségével lekérdezett sorokkal foglalkozunk, a jellemzők az egyes oszlopok. Tegyük fel, hogy a következő táblázat egy lekérdezés eredményeképpen kapott sorok halmazát tartalmazza (a receptek John szakácskönyvéből valók):

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Csirkeragu	Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Vega ragu	Tészta	Víz	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Tészta	Víz	Sertéshús	Hagyma

Egy második lekérdezés eredménye a következőképpen nézhet ki (ezek a receptek Mike szakácskönyvében találhatók):

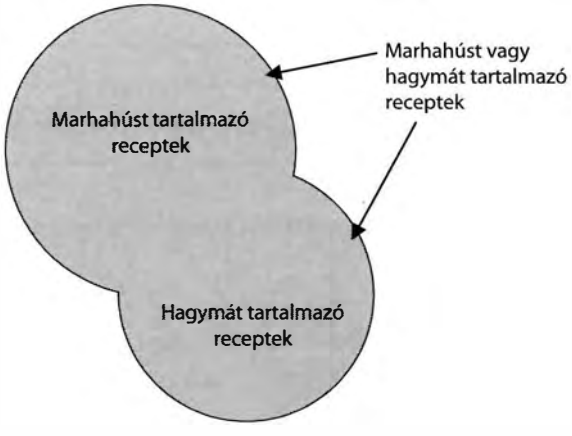
Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Pulykaragu	Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Vega ragu	Tészta	Zöldség	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Bab	Víz	Sertéshús	Hagyma

A két halmaz unója mindkét halmaz összes sorát tartalmazza. Lehet, hogy John és Mike eldöntötték: közösen írnak szakácskönyvet.

Recept	Köret	Alaplé	Hús	Zöldség
Bárányragu	Burgonya	Víz	Bárányhús	Zöldborsó
Csirkeragu	Rizs	Csirke alaplé	Csirkehús	Sárgarépa
Vega ragu	Tészta	Víz	Tofu	Cukorborsó
Ír ragu	Burgonya	Marha alaplé	Marhahús	Fejes káposzta
Sertésragu	Tészta	Víz	Sertéshús	Hagyma
Pulykaragu	Rizs	Csirke alaplé	Pulykahús	Sárgarépa
Vega ragu	Tészta	Zöldség	Tofu	Cukorborsó
Sertésragu	Bab	Víz	Sertéshús	Hagyma

Tegyük fel, hogy van egy a kedvenc receptjeinket tartalmazó adatbázisunk, és szeretjük a marhahúsos vagy hagymás recepteket, ezért szeretnénk egy olyan listát, amelyik tartalmazza az egyik hozzávalót. A 7.5. ábra szemlélteti, hogyan lehet megoldani ezt a feladatot.

A felső kör jelöli a marhahúst tartalmazó recepteket, az alsó a hagymát tartalmazókat. A két halmaz uniójába azok a receptek kerülnek, amelyek a két hozzávaló közül az egyiket tartalmazzák. A kétszer szereplő elemeket (ahol a két halmaz átfedi egymást) eltávolítottuk. Mint bizonyára tudjuk, először a marhahúsos recepteket kell lekérdeznünk az SQL-lel, majd egy második lekérdezéssel a hagymát tartalmazó recepteket gyűjtjük ki. Mint később látni fogjuk, a UNION az az SQL-kulcsszó, amivel összekapcsolhatjuk a két lekérdezést, hogy megkapjuk a végső választ.



7.5. ábra

Marhahúst vagy hagymát tartalmazó receptek halmazai

Már megtanultuk, hogy nem jó ötlet olyan receptes adatbázist tervezni, amiben egy tábla van. Ehelyett egy helyes módon megtervezett adatbázisban van egy Recipe_Ingredients tábla, amely receptenként és hozzávalónként egy sort tartalmaz. Minden sor egyetlen hozzávalót fog tartalmazni, így egy sorban sem szerepelhet egyidejűleg marhahús és hagyma. Először azokat a recepteket kell megkeresnünk, amelyekben van marhahúst tartalmazó sor, majd azokat, amelyekben van hagymát tartalmazó sor, végül ezek unióját kell képeznünk.

Unióval megoldható feladatok

Az unió lehetővé teszi, hogy két hasonló halmaz sorait „összekeverjük" – azzal a járulékos haszonnal, hogy az eredményben nem fordul majd elő kétszer ugyanaz a sor. Az alábbiakban néhány olyan feladatot láthatunk a mintaadatbázisokból, amit unióképzéssel lehet megoldani:

„Mutasd meg minden vásárló és alkalmazott nevét és címét!"
„Sorold fel azokat a vásárlókat, akik biciklit vettek, azokkal együtt, akik bukósisakot rendeltek!"
„Sorold fel azokat az előadókat, akik a Bonnicksen nevű megrendelőnél játszottak, azokkal együtt, akik a Rosales nevű megrendelőnél léptek fel!"
„Mutasd meg azokat a hallgatókat, akiknek rajzból átlagosan legalább 85 pontjuk van, azokkal a hallgatókkal együtt, akik számítástechnikából értek el ugyanilyen eredményt!"
„Keresd meg azokat a tekejátékosokat, akiknek a Thunderbird pályán legalább 155, a Bolero pályán pedig legalább 140 lett a tiszta pontszáma!"
„Mutasd meg azokat a recepteket, amelyek marhahúst tartalmaznak, azokkal együtt, amelyekben van fokhagyma!"

Mint a többi halmazműveletnél, itt is érvényes az a megszorítás, hogy minden eredményhalmazban meg kell egyezniük az oszlopok értékeinek. Ez jól működik, ha ugyanabból a táblából képezünk két vagy több halmazt (például azok a vevők, akik biciklit rendeltek és azok a vevők, akik bukósisakot rendeltek). Akkor is jól működik az unióképzés, ha olyan halmazokból származó táblákból képezünk halmazokat, amelyek azonos oszlopokat tartalmaznak (például: vásárlók neve és címe, valamint az alkalmazottak neve és címe). A 10. fejezetben részletesen elmagyarázzuk, hogyan kell használni az SQL UNION műveletét.

Sok esetben, ahol egyébként ugyanabból a táblából származó sorokat egyesítenénk, az is megteszi, ha egy összetett tábla-összekapcsolási feltételekkel ellátott DISTINCT-et alkalmazunk (a többször szereplő sorok felesleges példányainak eltávolítására). A 8. fejezetben, amikor a belső összekapcsolásokról tanulunk, mindent meg fogunk mutatni arról, hogy miként oldhatunk meg ilyen feladatokat a JOIN-nal.

Halmazműveletek az SQL-ben

Most, hogy megvannak a halmazműveletekről szerzett alapismereteink, nézzük meg, hogy az SQL-ben ezek hogyan valósíthatók meg!

A klasszikus halmazműveletek és az SQL

Amint korábban említettük, egyelőre nem sok kereskedelmi adatbázisrendszer valósítja meg közvetlenül a metszetképzést (INTERSECT) vagy a különbségképzést (EXCEPT). A jelenlegi SQL-szabvány azonban világosan meghatározza, hogy ezeket a halmazműveleteket hogyan kellene megvalósítani. Úgy gondoljuk, hogy ezek a halmazműveletek elég fontosak ahhoz, hogy legalább áttekintést adjunk a műveletek nyelvtanáról.

Ígértük, hogy későbbi fejezetekben a JOIN segítségével más lehetőségeket is megmutatunk, hogy egy metszet- vagy különbségképzésre visszavezethető problémát megoldjunk. Mivel a legtöbb adatbázisrendszer támogatja a UNION műveletet, a 10. fejezetet ennek szenteltük. A fejezet hátralevő része a három halmazműveletről nyújt áttekintést.

Közös elemek megtalálása: INTERSECT

Tegyük fel, hogy a következő egyszerű problémát szeretnénk megoldani:

„Mutasd meg azokat a rendeléseket, amelyek biciklit és bukósisakot is tartalmaznak!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of bike and helmet product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a biciklik és bukósisakok termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike and helmet product numbers

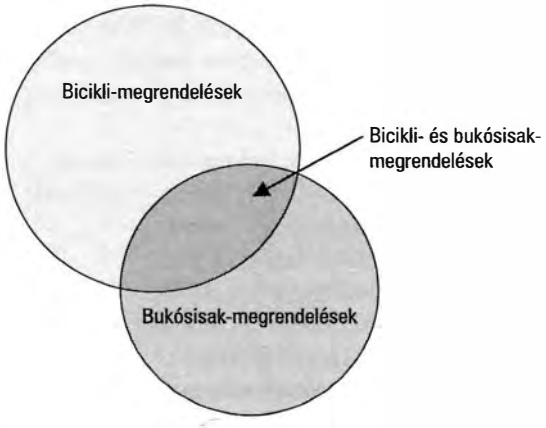
SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 10, 11, 25, 26)
```

Megjegyzés *Az SQL-ben járatos olvasók jogosan kérdezhetik, hogy miért nem kapcsoltuk össze egy JOIN-nal az Order_Details (Megrendelések_részletei) és Products (Termékek) táblákat, majd kerestünk bicikli- és bukósisak-termékneveket. A válasz egyszerű: még nem vezettük be a JOIN fogalmát, így a példában egyetlen táblát és az IN-t használtuk, valamint a biciklik és bukósisakok ismert termékazonosítóinak listáját.*

Ezzel látszólag sikerül megoldani a dolgot, a válasz azonban olyan megrendeléseket tartalmaz, amelyekben bicikli *vagy* bukósisak van. Minket azonban azok érdekelnek, amelyekben bicikli *és* bukósisak is található. Könnyebb megérteni a problémát, ha külön halmazként ábrázoljuk a bicikli- és bukósisak-megrendeléseket. A 7.6. ábra halmazdiagramon keresztül szemlélteti a megrendelések két halmaza közötti egyik lehetséges kapcsolatot.

Tulajdonképpen nem lehet előre megjósolni, hogy milyen kapcsolat áll fenn két adathalmaz között. A 7.6. ábrán egyes megrendeléseknél csak bicikli szerepel a rendelt tételek listáján, de nincs bukósisak. Más megrendeléseknél csak bukósisak szerepel, de nincs bicikli. A két halmaz átfedése vagy metszete az a rész, ahol a biciklit és bukósisakot tartalmazó megrendelések szerepelnek. A 7.7. ábra azt a lehetőséget szemlélteti, amikor *minden* megrendelésben szerepel bicikli és bukósisak, de vannak olyan bicikli-megrendelések, amelyekhez nem rendeltek bukósisakot.

Az, hogy a lekérdezésben „mindkettő" szerepel, azt sugallja, hogy a megoldást két adathalmazra kell bontani, majd a két halmazt valamilyen módon össze kell kapcsolni. (A lekérdezést is két részre kell bontani.)



7.6. ábra

Egy lehetséges kapcsolat a megrendelések két halmaza között

„Mutasd meg a bicikli-megrendeléseket!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a biciklik termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

„Mutasd meg a bukósisak-megrendeléseket!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a bukósisakok termékazonosítóinak listájában.)

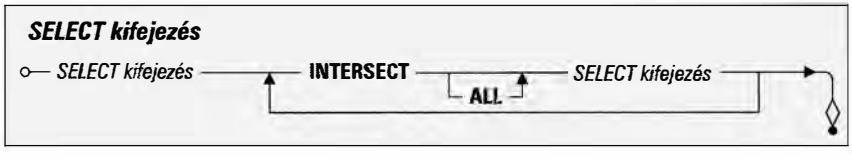
Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

7.7. ábra

A bukósisak-megrendelések egyben bicikli-megrendeléseket is tartalmaznak

Most már készen állunk, hogy – mint bizonyára kitaláltuk – a két halmaz *metszetét* képezve megkapjuk a végső választ. A megoldást a 7.8. ábrán látható szintaxisdiagram szemlélteti. (Jegyezzük meg, hogy több INTERSECT-et is használhatunk, így több SELECT állítást is összekapcsolhatunk.)



7.8. ábra

Két SELECT állítás összekapcsolása INTERSECT-tel

Most már összekapcsolhatjuk a lekérdezés két részét egy INTERSECT művelettel, hogy megkapjuk a helyes választ:

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
INTERSECT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

A rossz hír, hogy az SQL kereskedelmi megvalósításai közül egyelőre nem sok támogatja az INTERSECT műveletet – de nincs minden veszve! Emlékezzünk rá, hogy egy tábla elsődleges kulcsa egyértelműen azonosít minden sort. (Elég, ha a mezők közül csak az elsődleges kulcs egyezik, így megtaláljuk a metszet egyes sorait.) A 8. fejezetben mutatunk egy másik módszert (JOIN), amivel a problémát más módon is meg lehet megoldani.
A jó hír, hogy az SQL legtöbb kereskedelmi megvalósítása támogatja a JOIN-t.

Hiányzó elemek megtalálása: EXCEPT (különbség)

Rendben, térjünk vissza megint a biciklik és bukósisakok problémájára. Tegyük fel, hogy a következő, egyszerűnek tűnő lekérdezést szeretnénk megoldani:

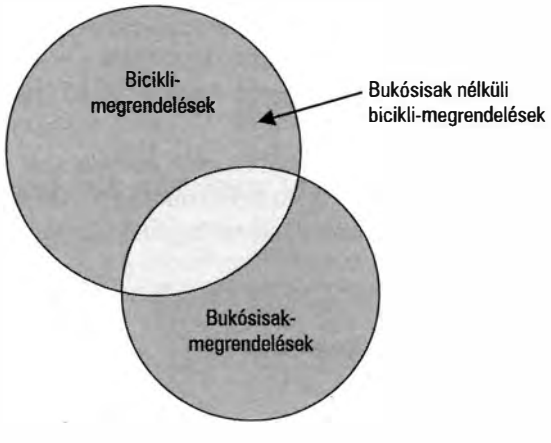
„Mutasd meg azokat a bicikli-megrendeléseket, amelyekben nem szerepel bukósisak!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers and product number is not in the list of helmet product numbers (Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a biciklik termékazonosítóinak listájában, és a termékazonosító nincs benne a bukósisakok termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers and product number ~~is~~ not in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
   AND ProductNumber NOT IN (10, 25, 26)
```

Sajnos a válaszba csak azok a megrendelések kerülnek, amelyekben kizárólag bicikli szerepel. A gond az, hogy az első IN záradék biciklit tartalmazó sorokat ad vissza, a második IN záradék viszont egyszerűen kiszűri a bukósisakot tartalmazó sorokat. Könnyebb megérteni a problémát, ha külön halmazként ábrázoljuk a bicikli- és bukósisak-megrendeléseket. A 7.9. ábra halmazdiagramon keresztül szemlélteti a megrendelések két halmaza közötti egyik lehetséges kapcsolatot.



7.9. ábra

Bukósisak nélküli bicikli-megrendelések

Az, hogy a lekérdezésben „különbség" vagy „de nem" szerepel, azt sugallja, hogy a megoldást két adathalmazra kell bontani, majd a két halmazt valamilyen módon össze kell kapcsolni. (A lekérdezést is két részre kell bontani.)

„Mutasd meg a bicikli-megrendeléseket!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a biciklik termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

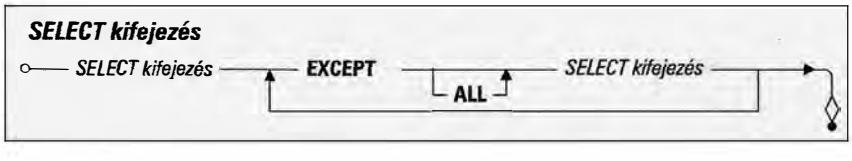
„Mutasd meg a bukósisak-megrendeléseket!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a bukósisakok termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Most már készen állunk, hogy – mint bizonyára kitaláltuk – a két halmaz *különbségét* képezve megkapjuk a végső választ. Az SQL az EXCEPT kulcsszót használja a különbségképző művelet jelölésére. A megoldást a 7.10. ábrán látható szintaxisdiagram szemlélteti.



7.10. ábra

Két SELECT állítás összekapcsolása EXCEPT-tel

Most már összekapcsolhatjuk a lekérdezés két részét egy EXCEPT művelettel, hogy megkapjuk a helyes választ:

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
EXCEPT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Emlékezzünk vissza arra, amikor a különbségképzést tárgyaltuk: a halmazok sorrendje számít. Ebben az esetben a bicikliket kérdezzük le, „kivéve" a bukósisakokat. Ha ennek az ellenkezőjét – a bicikli nélküli bukósisak-megrendeléseket – keressük, a következőképpen fordíthatjuk meg a lekérdezést:

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
EXCEPT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

Sajnos az SQL kereskedelmi megvalósításai közül egyelőre nem sok támogatja az EXCEPT műveletet, de kapjunk a sisakunkhoz! Emlékezzünk rá, hogy egy tábla elsődleges kulcsa egyértelműen azonosít minden sort. (Nem kell egy sor minden mezőjének megfelelnie a feltételnek, csak az elsődleges kulcsnak, hogy megtaláljuk a különböző sorokat.) A 9. fejezetben mutatunk egy másik módszert (OUTER JOIN) is, amivel a problémát más módon lehet megoldani. A jó hír, hogy az SQL legtöbb kereskedelmi megvalósítása támogatja az OUTER JOIN-t.

Halmazok egyesítése: UNION

Még egy utolsó megoldandó feladat a biciklikkel és bukósisakokkal, és utána továbbpedálozunk a következő fejezetre. Tegyük fel, hogy a következő, a felszínen egyszerűnek tűnő problémát akarjuk megoldani:

„Mutasd meg azokat a megrendeléseket, amelyekben vagy bicikli, vagy bukósisak szerepel!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of bike and helmet product numbers

(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a biciklik és bukósisakok termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike and helmet product numbers

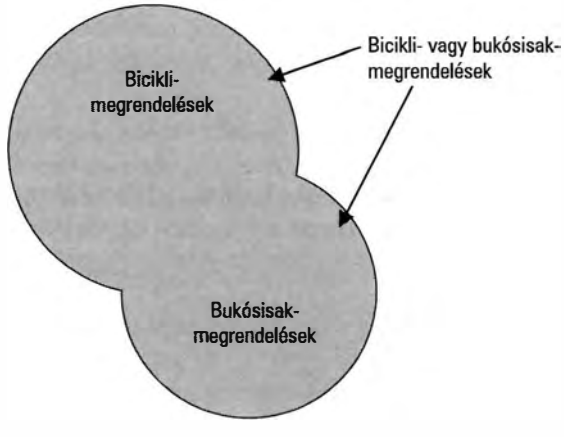
SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 10, 11, 25, 26)
```

Tulajdonképpen ez így tökéletesen működik – de akkor miért használjunk UNION-t, hogy megoldjuk a feladatot? Az igazság az, hogy nem kell, de ha összetettebb problémán dolgozunk, jól jöhet a UNION.

„Sorold fel a biciklit rendelő vevőket, azokkal a beszállítókkal együtt, akik biciklit kínálnak!"

Sajnos ennek a kérdésnek a megválaszolásához néhány JOIN műveletet magában foglaló lekérdezést kell szerkesztenünk, majd a UNION-nal kell megtalálnunk a végső választ. Mivel még nem mutattuk meg, hogyan kell a JOIN-t alkalmazni, ennek a problémának a megoldása a 10. fejezetig váratni fog magára. Emiatt máris érdemes tovább olvasni, nem igaz?

Térjünk vissza a „biciklik és bukósisakok" problémára és oldjuk meg a UNION-nal. Ha a bicikli- és bukósisak-rendeléseket két külön halmazként ábrázoljuk, könnyebben meg fogjuk érteni a problémát. A megrendelések két halmaza közötti egyik lehetséges kapcsolatot a 7.11. ábra szemlélteti.



7.11. ábra
Bicikli- vagy bukósisak-megrendelések

Az, hogy a lekérdezésben „vagy" vagy „együtt" szerepel, azt sugallja, hogy a megoldást két adathalmazra kell bontani, majd a két halmazt egy UNION-nal össze kell kapcsolni. Ezt a lekérdezést két részre bonthatjuk.

„Mutasd meg a bicikli-megrendeléseket!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a biciklik termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

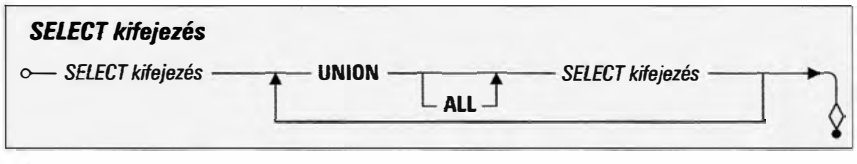
„Mutasd meg a bukósisak-megrendeléseket!"

Fordítás Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
(Válaszd ki azokat a különböző megrendelés-azonosítókat a megrendelések részleteit tartalmazó táblából, ahol a termékazonosító megtalálható a bukósisakok termékazonosítóinak listájában.)

Tisztázás Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Most már készen állunk, hogy – mint bizonyára kitaláltuk – a két halmaz *uniójá*t képezve megkapjuk a végső választ. A megoldást a 7.12. ábrán látható szintaxisdiagram szemlélteti.



7.12. ábra

Két SELECT állítás összekapcsolása UNION-nal

Most már összekapcsolhatjuk a lekérdezés két részét egy UNION művelettel, hogy megkapjuk a helyes választ:

```
SQL     SELECT DISTINCT OrderNumber
        FROM Order_Details
        WHERE ProductNumber IN (1, 2, 6, 11)
        UNION
        SELECT DISTINCT OrderNumber
        FROM Order_Details
        WHERE ProductNumber IN (10, 25, 26)
```

A jó hír, hogy az SQL kereskedelmi megvalósításai közül a legtöbb támogatja a UNION műveletet. Mint az előbbi példákból is valószínűleg kiderült, a UNION lehet a nehezebb megoldás, ha egy „vagy" eredményt akarunk megkapni egyetlen táblából. A UNION akkor a leghatékonyabb, ha több különböző, de hasonló szerkezetű táblából akarunk listát készíteni. A 10. fejezetben részletesen is megismerkedünk majd a UNION-nal.

Összefoglalás

A fejezet elején a halmaz fogalmával ismerkedtünk meg, majd az SQL-ben megvalósított főbb halmazműveleteket (metszet, különbség és unió) tárgyaltuk részletesen. Megmutattuk, hogyan használhatunk halmazdiagramot a megoldásra váró probléma szemléltetésére. Végül étvágygerjesztőként bemutattuk az alapvető SQL-utasításformákat és a három halmazművelet kulcsszavát (INTERSECT, EXCEPT és UNION).

Ezen a ponton valószínűleg így gondolkodunk: „Várjunk csak, megmutattak nekem három halmazműveletet, amiből kettőt valószínűleg nem fogok tudni használni?" Gondoljunk vissza a fejezet címére: *Halmazokban gondolkodni.* Ha szeretnénk egyáltalán sikeresen megoldani összetett feladatokat, a problémát eredményhalmazokra kell bontanunk, amelyeket azután újra össze kell kapcsolnunk.

Szóval, ha a probléma úgy néz ki, hogy „ennek ilyennek és annak meg olyannak kell lennie", akkor először valószínűleg meg kell oldanunk „ennek", majd utána „annak" a problémáját, majd össze kell őket kapcsolnunk, hogy megkapjuk a végső választ. Az SQL-szabvány meghatároz egy kényelmes INTERSECT műveletet, de egy INNER JOIN éppen olyan jól működhet. Erről bővebben a 8. fejezetben lesz szó.

Hasonló módon, ha a probléma úgy írható le, hogy „ennek ilyennek kell lennie, *de nem lehet* olyan", akkor először „ennek", majd „annak" a problémáját kell megoldanunk, végül „ebből" ki kell vonnunk „azt", hogy megkapjuk a választ. Megmutattuk az SQL-szabvány EXCEPT műveletét, de egy OUTER JOIN-nal is sikerülhet a dolog. Erről a 9. fejezetben lesz részletesen szó.

Végül megmutattuk, hogyan adhatunk össze adathalmazokat a UNION segítségével. Mint ígértük, a UNION-nal a 10. fejezetben fogunk mélységeiben megismerkedni.

8

Belső összekapcsolás

„Ne fogd vissza az ihleted és a képzeleted;
ne válj a modelled rabszolgájává."
– Vincent van Gogh

A fejezet témakörei

- Mi az a JOIN?
- Az INNER JOIN
- Az INNER JOIN alkalmazási lehetőségei
- Példák
- Összefoglalás
- Önálló feladatok

Eddig elsősorban az egyetlen tábla segítségével történő problémamegoldásra összpontosítottunk. Már tudjuk, hogyan kaphatunk egyszerű válaszokat egy táblából, és azt is tudjuk, hogy kifejezések segítségével vagy az eredmény rendezésével hogyan kaphatunk kicsit bonyolultabb válaszokat. Más szavakkal, szemet, orrot, szájat már tudunk rajzolni. Ebben a fejezetben megtanuljuk, hogyan kell összekötni a részeket, hogy arckép legyen belőlük.

Mi az a JOIN?

A 2. fejezetben hangsúlyoztuk, hogy milyen fontos, hogy a különböző dolgokra vonatkozó adatokat szétválasszuk és külön táblákban tároljuk. A valós életben a legtöbb probléma megoldásához több külön táblában szereplő adatokat kell összekapcsolnunk – vásárlókat a megrendeléseikkel, ügyfeleket a foglalásaikkal, játékosokat a pontszámaikkal, hallgatókat az általuk felvett órákkal, vagy recepteket a szükséges hozzávalókkal. Ahhoz, hogy ezeket az összetettebb feladatokat megoldjuk, több táblát kell összekapcsolnunk, így kapjuk meg a keresett válaszokat. Ehhez a JOIN (összekapcsolás) kulcsszót kell használnunk.

Az előző fejezetből kiderült, milyen hasznos lehet egy probléma megoldásánál két adathalmaz metszetét létrehozni. Ugyanakkor, ha visszaemlékszünk, az INTERSECT mindkét eredményhalmaz összes oszlopát összehasonlítja. A JOIN szintén metszetet hoz létre,

azzal a különbséggel, hogy az adatbázis-kezelő rendszer a JOIN esetében csak az általunk meghatározott oszlopokat veszi figyelembe. Így a JOIN segítségével két nagyon különböző szerkezetű tábla metszetét is létrehozhatjuk a kiválasztott oszlopokban megegyező értékek alapján. Például egy JOIN segítségével összepárosíthatjuk a vásárlókat a megrendeléseikkel, ha a Customers (Vásárlók) tábla CustomerID (Vásárlóazonosító) és az Orders (Megrendelések) tábla CustomerID mezőit hasonlítjuk össze.

Amint később látni fogjuk, a JOIN-t az SQL-utasítások FROM záradékában alkalmazzuk. A JOIN egy „logikai táblát" határoz meg, ami két tábla vagy eredményhalmaz összekapcsolásából jön létre. Azáltal, hogy a JOIN-t a FROM záradékba helyezzük, egy ilyen összekapcsolt táblát határozunk meg, és *ebből* fogja a lekérdezés kiválogatni a végső eredményhalmazt. Más szavakkal, a JOIN annak az egyetlen táblának a helyén áll, amelyet a FROM záradékban a korábbi fejezetekben megismert módon használtunk. Amint az ki fog derülni a fejezet során, több JOIN műveletet is megadhatunk, amivel kettőnél több táblából hozhatunk létre összetett eredményhalmazokat.

Az INNER JOIN

Az SQL-szabvány a JOIN művelet végrehajtásának többféle módját is meghatározza, amelyek közül a leggyakoribb az úgynevezett INNER JOIN (belső összekapcsolás). Képzeljük el egy pillanatra, hogy össze akarjuk kapcsolni a hallgatókat azokkal az órákkal, amelyeket felvettek. Elképzelhető, hogy némelyik hallgató – bár felvették az iskolába – még nem vett fel egyetlen órát sem, és lehetnek olyan óráink is, amelyeket ugyan meghirdettek, de még egyetlen hallgató sem vette fel.

A Students (Hallgatók) és a Classes (Órák) tábla közötti INNER JOIN a Students táblából fog sorokat visszaadni, összekapcsolva a Classes tábla soraival (a Students_Schedules – Hallgatók_órabeosztása – táblán keresztül) –, de nem fogja visszaadni sem azokat a hallgatókat, akik még semmilyen órát nem vettek fel, sem azokat az órákat, amelyeket még egyetlen hallgató sem vett fel. Az INNER JOIN csak azokat a sorokat adja vissza, ahol a kapcsolódó értékek mindkét táblában vagy eredményhalmazban megegyeznek.

Mit „szabad" JOIN segítségével összekapcsolni?

Legtöbbször az egyik tábla elsődleges kulcsát és másik tábla megfelelő idegen kulcsát adjuk meg a JOIN-nak, mint összekötő pontot. Ha emlékszünk még a 2. fejezetből, az idegen kulcsnak ugyanolyan adattípusúnak kell lennie, mint a megfelelő elsődleges kulcsnak. Mindazonáltal „szabad" JOIN-nal összekapcsolni két táblát vagy eredményhalmazt bármely olyan oszlop szerint, amely az SQL-szabvány alapján „JOIN-megfelelő" adattípusba tartozik.

Általánosságban, a karakter típusú oszlopokat összekapcsolhatjuk másik karakter típusú oszloppal vagy kifejezéssel, bármilyen (például egész) szám típusú oszlopot bármilyen más (mondjuk például lebegőpontos) szám típusú oszloppal, valamint bármilyen dátum

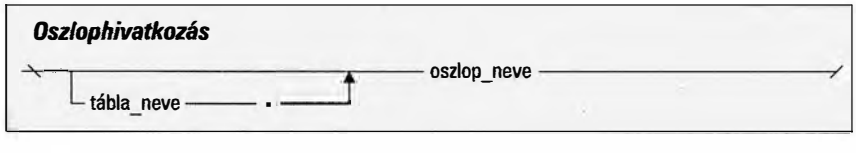
típusú oszlopot bármilyen dátum típusú oszloppal. Ez lehetővé teszi, hogy például a Customers tábla sorait összekapcsoljuk az Employees (Alkalmazottak) tábla soraival a városnevet vagy az irányítószámot tartalmazó oszlop szerint (mondjuk ha kíváncsiak vagyunk rá, hogy mely vásárlók és alkalmazottak élnek ugyanabban a városban vagy postakörzetben).

Megjegyzés *Az, hogy lehetséges bármely két tábla bármely JOIN-megfelelő oszlopa között JOIN-t használni, még nem feltétlenül jelenti azt, hogy ajánlatos is. Ahhoz, hogy a JOIN értelmes legyen, az összekapcsolt oszlopoknak ugyanolyan jelentésű adatokat kell tartalmazniuk. Például nincs értelme a vásárlók neveit az alkalmazottak postacímével kapcsolni össze, habár mindkét oszlop karakter adattípusú. Az eredményhalmazunk üres lesz, hacsak nem írt valaki véletlenül nevet az alkalmazott postacímét tartalmazó oszlopba. Hasonlóképpen, semmi értelme a StudentID és a ClassID közötti JOIN-nak, hiába szám típusú mind a kettő. Lehet, hogy lesz néhány sor az eredményhalmazban, de ezeknek nem lesz semmilyen jelentésük.*

Még amikor értelmes is bizonyos oszlopokat összekapcsolni, előfordulhat, hogy olyan lekérdezést hozunk létre, aminek a megválaszolása nagyon sokáig tart. Például ha olyan oszlopokon használunk JOIN-t, amelyekre az adatbázis rendszergazdája nem adott meg indexet, akkor az adatbázisrendszert jelentős többletmunkára kényszerítjük, valamint ha kifejezésekre alkalmazunk JOIN-t – például két táblából összefűzött vezeték- és keresztnevekre –, akkor az adatbázisrendszerünknek nem csak létre kell hoznia a kifejezésből kapott oszlopot minden sorhoz, de lehet, hogy a megfelelő válaszhoz többször is végig kell futnia mindkét tábla összes adatán.

Oszlophivatkozások

Mielőtt rátérnénk a JOIN nyelvtanára, szót kell ejtenünk még egy kulcsfontosságú tényről. Mivel az eddig alkalmazott lekérdezések egyetlen táblára vonatkoztak, az oszlopnevek használata egyértelmű volt. Viszont ha elkezdünk több táblát érintő lekérdezésekkel foglalkozni (például a JOIN-ok használatánál), akkor általában kettő vagy több táblára hivatkozunk, amelyek viszont tartalmazhatnak azonos nevű oszlopokat. Ha emlékszünk még a 2. fejezetre, ott azt a tanácsot kaptuk, hogy az egyik táblából az elsődleges kulcsot a megfelelő másik táblába átmásolva – a nevével együtt – hozzunk létre egy idegen kulcsot.



8.1. ábra

Oszlophivatkozás szintaxisdiagramja

Hogyan tudjuk tehát kristálytisztán elmagyarázni az adatbázis-kezelő rendszernek, hogy egy mező melyik másolatára gondolunk az adott lekérdezésben? Az egyszerű válasz az, hogy olyan *oszlophivatkozást* használunk, amely tartalmazza a tábla nevét. A 8.1. ábrán látható egy oszlophivatkozás diagramja.

Habár az oszlopnevet használhatjuk önmagában is bármely SQL-utasítás záradékában, kifejezetten meg is jelölhetjük az oszlopnevet a szülőtábla nevével. Ha az oszlopnév nem egyedi a FROM záradékban használt összes tábla között, akkor *muszáj* megjelölni az oszlopnevet a szülőtábla nevével. Alább láthatjuk, hogyan néz ki egy egyszerű SELECT utasítás az Employees táblán, amelyben ilyen minősített oszlopneveket használunk:

```
SQL     SELECT Employees.FirstName, Employees.LastName,
        Employees.PhoneNumber
        FROM Employees
```

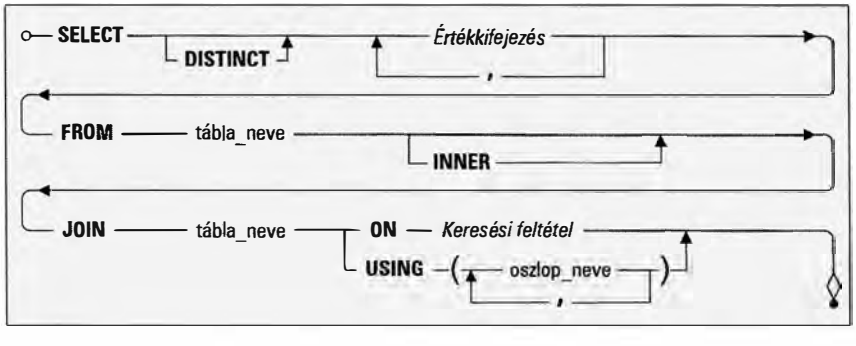
Most, hogy ezt a kis trükköt is ismerjük, haladjunk tovább és vizsgáljuk meg a JOIN utasításformáját!

Utasításforma

Az eddig tanultakat felfoghatjuk egy kis országúti élményautózásnak vagy egy rövid hétvégi bevásárlókörútnak. Eljött az ideje, hogy bekapcsoljuk a biztonsági övet, és kimerészkedjünk az autópályára – vizsgáljuk meg az INNER JOIN utasításformáját!

A táblák használata

Kezdjünk először egy egyszerű feladattal – egy két tábla közötti INNER JOIN-nal. A 8.2. ábrán látható a lekérdezésnél használatos utasításforma.



8.2. ábra

Egy két tábla között INNER JOIN-t használó lekérdezés szintaxisdiagramja

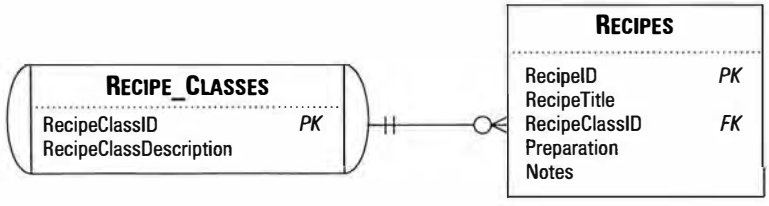
Amint látható, a FROM záradék csak egy picivel válik bonyolultabbá. (Az egyszerűség kedvéért a WHERE és az ORDER BY záradékoktól egyelőre eltekintettünk.) Egyetlen táblanév helyett két táblanevet adunk meg, és a JOIN kulcsszóval kapcsoljuk őket össze. Vegyük észre, hogy az INNER kulcsszó, ami elhagyható, a JOIN típusát adja meg. Amint a következő fejezetből megtudhatjuk majd, használható még úgynevezett OUTER JOIN (külső összekapcsolás) is. Ha nem adjuk meg kifejezetten, hogy milyen típusú JOIN-t szeretnénk, akkor az alapértelmezés az INNER. Tanácsos mindig egyértelműen kiírni, hogy milyen típusú JOIN-t szeretnénk, hogy a lekérdezés jellege tiszta és világos legyen.

Megjegyzés *Azok, akik követik az A függelékben található teljes szintaxisdiagramokat, a „Táblahivatkozás JOIN Táblahivatkozás" alakkal találkozhatnak az Összekapcsolt tábla kifejezés meghatározásánál. A Táblahivatkozás egy tábla_neve vagy egy Összekapcsolt tábla lehet, a SELECT utasítás FROM záradéka pedig egy Táblahivatkozás-t használ. Ezeket az összetett meghatározásokat fent „egybegyúrtuk", hogy kényelmesebben, egyetlen diagramon ábrázolhassuk az egyszerű, két tábla közötti JOIN-t. A fejezet hátralévő részében a diagramokban hasonló egyszerűsítéssel élünk.*

Az INNER JOIN legfontosabb része a második táblát követő ON vagy USING záradék, amely megmondja az adatbázis-kezelőnek, hogy pontosan hogyan végezze el a JOIN műveletet. Az adatbázis-kezelő rendszernek ahhoz, hogy megoldja a JOIN-t, az első tábla minden sorát logikailag össze kell kapcsolnia a második tábla minden sorával. (Ezt az összekapcsolást, amelyben egy tábla minden sora összekapcsolódik egy másik tábla minden sorával, *Descartes-szorzatnak* nevezik.) A rendszer ez után alkalmazza az ON vagy USING záradékban megadott feltételeket, kiválogatva a megfelelő visszaadandó sorokat.

A 6. fejezetben megtanultuk, hogyan használható a *keresési feltétel* a WHERE záradékban. Keresési feltételt használhatunk a JOIN utasítás ON záradékában is, megadva egy logikai tesztet, amelynek teljesülnie kell ahhoz, hogy visszakapjunk egy adott összekapcsolt sorpárt. Ne felejtsük el, hogy csak olyan keresési feltételnek van értelme, amelyben mindkét táblából legalább egy-egy oszlopot összehasonlítunk. Akármilyen bonyolult keresési feltételt írhatunk, de általában egyszerű egyenlőségi tesztet fogunk alkalmazni az egyik tábla elsődleges kulcsot tartalmazó oszlopa és a másik tábla idegen kulcsot tartalmazó oszlopa között.

Nézzünk meg egy egyszerű példát! Egy jól átgondolt adatbázisban az összetett osztálynéveket érdemes egy másik táblában tárolni, és a neveket egy egyszerű kulcsértékkel kapcsolni vissza az eredeti táblához. Ez segít megelőzni az adatbevitel során keletkező hibákat. Bárki, aki az adatbázisunkat használja, az osztálynevek listájából választ, nem pedig saját kezűleg gépeli be a nevet (ami esélyt ad a félregépelésre) minden sor esetében. Például a Recipes (Receptek) mintaadatbázis a receptosztályokat külön táblában sorolja fel, elválasztva maguktól a receptektől. A Recipe_Classes (Receptosztályok) és a Recipes táblák közötti kapcsolat a 8.3. ábrán látható.



8.3. ábra

A receptosztályok adatai másik táblában szerepelnek, elválasztva a Recipes táblától

Amikor receptekről, illetve a megfelelő receptosztály-leírásokról (RecipeClassDescription) akarunk információt lekérdezni az adatbázisból, akkor nem a Recipes táblában szereplő RecipeClassID kódszámokat akarjuk látni. Lássuk tehát, hogyan oldható meg ez a feladat a JOIN segítségével!

Megjegyzés *Ebben a fejezetben mindenhol a 4. fejezetben használt „Kérelem – Fordítás – Tisztázás – SQL" módszert alkalmazzuk.*

„Sorold fel az adatbázisunkban szereplő összes recept címét, az elkészítés menetét, és a receptosztály leírását!"

Fordítás Select recipe title, preparation, and recipe class description recipe class ID
from the recipe classes table joined with the recipes table on
in the recipe classes table matching recipe class ID in the recipes table
(Válaszd ki a recept címét, az elkészítés menetét és a receptosztály leírását a receptosztályok táblájával összekapcsolt recepttáblából, ahol az összekapcsolás a receptosztályok táblájában található receptosztály-azonosító és a receptek táblájában levő receptosztály-azonosító szerint történik.)

Tisztázás Select recipe title, preparation, ~~and~~ recipe class description
from ~~the~~ recipe classes ~~table~~
inner ~~joined with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipe classes table matching~~ = recipes.recipe class ID ~~in the recipes table~~

SQL
```
SELECT RecipeTitle, Preparation,
   RecipeClassDescription
FROM Recipe_Classes
INNER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
  Recipes.RecipeClassID
```

Megjegyzés *Amint látható, a Tisztázás lépésben használt kifejezések már hasolítanak a végső, az SQL kódban használt záradékokhoz. Ahogy egyre bonyolultabb lekérdezéseket kezdünk írni, érdemes lesz ezt a módszert követni, ami segít a tisztázástól eljutni a végső SQL-megoldásig.*

Amikor a FROM záradékban több táblát használunk, mindig a megfelelő tábla nevével minősített oszlopneveket használjuk, hogy teljesen világos legyen, melyik tábla melyik oszlopára gondolunk. (Már látszik, miért szakítottunk időt a táblahivatkozások leírására!) Vegyük észre, hogy az ON záradékban azért volt *muszáj* minősíteni a RecipeClassID (Receptosztály-azonosító) neveket, mert több ilyen nevű oszlop is létezik – az egyik a Recipes táblában, a másik a Recipe_Classes táblában. Ugyanezért nem kellett a SELECT záradékban a RecipeTitle (Receptcím), a Preparation (Elkészítés) és RecipeClassDescription (Receptosztály leírása) oszlopneveket minősíteni, mivel ezek mind csak egyszer fordulnak elő az összes tábla között. Ha a kimenetben látni szeretnénk a RecipeClassID-t is, akkor meg kell mondanunk az adatbázis-kezelőnek, hogy *melyik* RecipeClassID-re van szükségünk – arra, amelyik a Recipe_Classes táblában szerepel, vagy arra, amelyik a Recipes táblában található. Egy olyan lekérdezés, ahol minden név minősített, a következőképpen néz ki:

SQL
```
SELECT Recipes.RecipeTitle,
   Recipes.Preparation,
   Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
INNER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
```

Megjegyzés *Bár a legtöbb forgalomban lévő SQL-megvalósítás támogatja a JOIN kulcsszót, vannak olyanok is, amelyek nem. Ha az adatbázis-kezelő rendszerünk nem támogatja a JOIN-t, akkor is meg tudjuk oldani a feladatot, úgy, hogy a FROM záradékban felsoroljuk az összes táblát, amire szükségünk van, és az ON záradék helyett a WHERE záradékba tesszük a keresési feltételt. A JOIN-t nem támogató adatbázisrendszerekben tehát így néz ki a feladatunk megoldása:*

```
SELECT Recipes.RecipeTitle, Recipes.Preparation,
   Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
```

Tulajdonképpen egy kezdő számára ez a forma sokkal természetesebbnek tűnhet az egyszerű lekérdezések esetében. Az SQL-szabvány azonban lehetővé teszi, hogy kizárólag a FROM záradékon belül pontosan meghatározzuk a végső eredményhalmaz forrását. Fogjuk fel a FROM záradékot úgy, mint ami pontosan meghatároz egy összekapcsolt eredményhalmazt, amiből az adatbázis-kezelő kinyeri a kívánt választ. Az SQL-szabvány szerint a WHERE záradék csak arra szolgál, hogy a FROM záradék által meghatározott eredményhalmazból kiszűrjön bizonyos sorokat.

Ugye, hogy nem olyan bonyolult? De vajon mi történt a 8.2. ábrán látható USING záradékkal? Ha a két táblában a két összehasonlítandó oszlopnak ugyanaz a neve, és az összekapcsolás feltételeként egyszerűen ezek egyenlőségét akarjuk használni, akkor használhatjuk a USING záradékot a megfelelő oszlopnévvel. Nézzük meg az előző példát a USING záradék segítségével megvalósítva:

„Sorold fel az adatbázisunkban szereplő összes recepthez tartozó címet, az elkészítés menetét és a receptosztály leírását!"

Fordítás Select recipe title, preparation, and recipe class description
from the recipe classes table joined with the recipes table
using recipe class ID
(Válaszd ki a recept címét, az elkészítés menetét és a receptosztály leírását a receptosztályok táblájával összekapcsolt recepttáblából, ahol az összekapcsolás a receptosztályok táblájában található receptosztály-azonosító szerint történik.)

Tisztázás Select recipe title, preparation, ~~and~~ recipe class description
from ~~the~~ recipe classes ~~table~~
~~joined with the~~ recipes ~~table~~
using recipe class ID

SQL
```
SELECT Recipes.RecipeTitle, Recipes.Preparation,
   Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
INNER JOIN Recipes
USING (RecipeClassID)
```

Néhány adatbázis-rendszer még nem támogatja a USING kulcsszót. Ha az általunk használt rendszerben nem használható a USING, ugyanezt az eredményt mindig elérhetjük egy ON záradékkal és egy egyenlőségi összehasonlítással.

Megjegyzés *Az SQL-szabvány meghatároz egy úgynevezett NATURAL JOIN-t (természetes összekapcsolás) is, ami két megadott táblát minden azonos nevű oszlop összehasonlításával kapcsol össze. Ha csak a két kívánt összekapcsoló oszlopnak van azonos neve a két táblában, és az adatbázis-kezelő rendszer támogatja a NATURAL JOIN-t, akkor a példánkat a következő módon is megoldhatjuk:*

```
SELECT Recipes.RecipeTitle, Recipes.Preparation,
   Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
NATURAL INNER JOIN Recipes
```

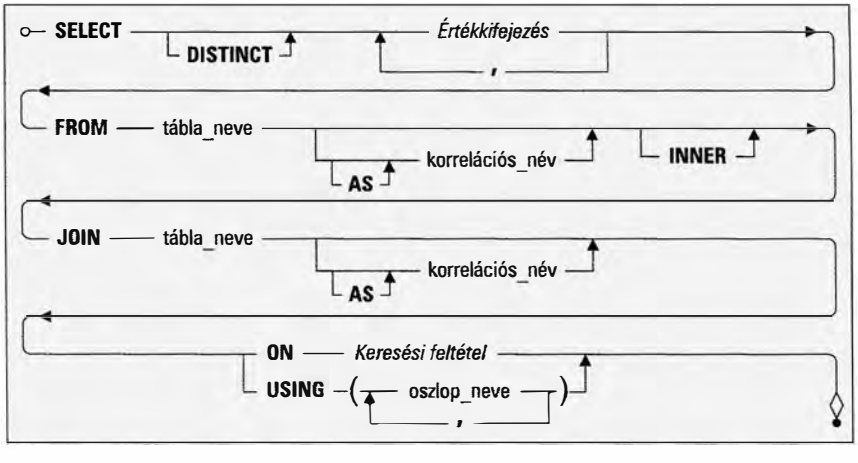
A NATURAL kulcsszó használata esetén ne adjunk meg ON vagy USING záradékot. Ne felejtsük el, hogy az INNER kulcsszó használata nem kötelező. Ha NATURAL JOIN-t írunk, akkor a rendszer INNER típusú összekapcsolást feltételez.

Amint ebben a részben korábban már említettük, az adatbázisrendszer először létrehozza az első és a második tábla sorainak összes lehetséges kombinációját, majd alkalmazza az ON vagy USING záradékban megadott feltételt. Ez alapján úgy tűnhet, hogy rengeteg többletmunkára kényszerítjük a rendszert, hiszen először létrehozza az összes lehetséges kombináció halmazát, majd ezek közül alkalomadtán egészen kevés illeszkedő sort fog kiszűrni a számunkra.

Valójában biztosak lehetünk benne, hogy az összes modern relációsadatbázis-kezelő mindig kiértékeli a teljes JOIN záradékot, mielőtt elkezdi kiolvasni a sorokat. Az eddigiekben bemutatott példafeladatot sok adatbázis-kezelő rendszer úgy kezdi el megoldani, hogy vesz egy sort a Recipes_Classes táblából, majd egy belső hivatkozást követ – egy indexet (ha a tábla tervezője megadott ilyet) –, amelynek a segítségével gyorsan megtalálja az összetartozó sorokat a Recipes táblából, amelyek illeszkednek a Recipe_Classes tábla első sorához, mielőtt a Recipes_Classes következő sorára lépne. Más szóval, az adatbázisrendszer okosan egy optimális eljárást követ, amelynek során csak az összetartozó sorokat kell kiolvasnia. Ez a néhány száz soros táblákból álló adatbázisok esetében talán nem tűnik fontosnak, de ha a rendszernek többszázezer sorral kell megbirkóznia, akkor a különbség ég és föld!

Korrelációs nevek (álnevek) hozzárendelése táblákhoz

Az SQL-szabvány meghatároz egy módszert, amivel álnevet – a szabvány szóhasználatával *korrelációs nevet* – adhatunk bármelyik táblának, amelyet a FROM záradékban használunk. Ez a lehetőség igen hasznos lehet, amikor olyan összetett lekérdezéseket írunk, amelyek hosszú, bonyolult nevű táblákat használnak. A hosszú nevű tábláknak adhatunk egy egyszerű nevet, amelyet használva kényelmesebb az oszlophivatkozások megadása.



8.4. ábra

Korrelációs név (álnév) hozzárendelése egy táblához a FROM záradékban

A 8.4. ábrán látható, hogyan rendelhetünk korrelációs nevet egy táblához a FROM záradékban.

Ahhoz, hogy egy táblának korrelációs nevet adjunk, a tábla neve után helyezzük el a nem kötelező AS kulcsszót, majd a kívánt korrelációs nevet. (Mint minden nem kötelező kulcsszó esetében, ajánlatos az AS kulcsszót minden esetben kitenni, így utólag könnyebb lesz a lekérdezést olvasni, illetve megérteni.) Miután egy táblához hozzárendeltünk egy korrelációs nevet, onnantól kezdve azt a tábla eredeti neve helyett használhatjuk bármely záradékban, beleértve a SELECT záradékot, az ON és WHERE záradék keresési feltételeit, és az ORDER BY záradékot. Elsőre persze összezavarodhatunk, mivel a SELECT záradékot általában előbb írjuk meg, mint a FROM záradékot. Ha a FROM záradékban álnevet szándékozunk adni egy táblának, akkor a SELECT záradékban az oszlopnevek minősítésénél is ezt az álnevet kell használnunk.

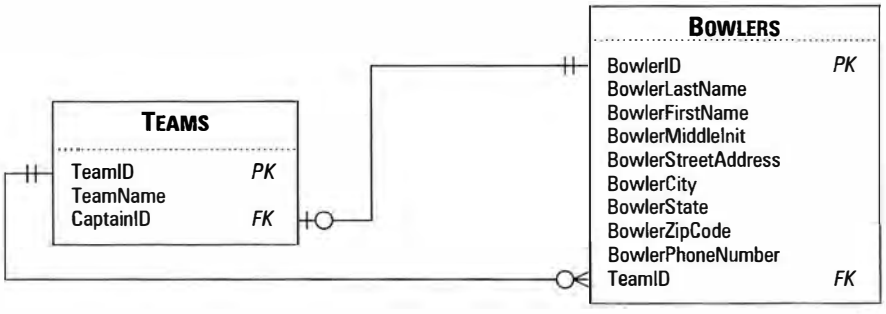
Alakítsuk át az eddig használt példafeladat megoldását korrelációs nevek segítségével, csak a próba kedvéért. Íme a lekérdezés, amelyben a Recipes tábla korrelációs neve R lesz, a Recipe_Classes tábláé pedig RC:

```
SQL     SELECT R.RecipeTitle, R.Preparation,
           RC.RecipeClassDescription
        FROM Recipe_Classes AS RC
        INNER JOIN Recipes AS R
        ON RC.RecipeClassID = R.RecipeClassID
```

Tegyük fel, hogy hozzá akarunk toldani a lekérdezéshez egy szűrőt, ami csak a Main course (Főétel) vagy a Dessert (Desszert) osztályba tartozó recepteket adja vissza. (A szűrőkről a 6. fejezetben olvashattunk.) Miután megadtuk a korrelációs nevet, a továbbiakban a táblára való összes hivatkozásban az új nevet kell használnunk. Íme az SQL-példa:

```
SQL     SELECT R.RecipeTitle, R.Preparation,
        RC.RecipeClassDescription
        FROM Recipe_Classes AS RC
        INNER JOIN Recipes AS R
        ON RC.RecipeClassID = R.RecipeClassID
        WHERE RC.RecipeClassDescription = 'Main course'
        OR RC.RecipeClassDescription = 'Dessert'
```

Nem muszáj minden táblához korrelációs nevet rendelni. Az előző példában megtehettük volna, hogy csak a Recipes vagy csak a Recipe_Class táblához rendelünk korrelációs nevet. Bizonyos esetekben azonban ***muszáj*** egy táblához korrelációs nevet rendelnünk egy összetett JOIN-on belül. Ugorjunk most egyet, és térjünk át a Bowling League (Tekebajnokság) adatbázisra, és vizsgáljunk meg egy olyan esetet, ahol ez a helyzet. A 8.5. ábrán látható a Teams (Csapatok) és a Bowlers (Játékosok) táblák közötti kapcsolat.



8.5. ábra

A Teams és a Bowlers táblák közötti kapcsolat

Amint látható, a TeamID (Csapatazonosító) egy idegen kulcs a Bowlers táblában, amely segít megtalálni az információt egy csapat összes játékosáról. A csapat játékosai közül az egyik egyben csapatkapitány is, ezért a Bowlers tábla BowlerID-ja (Játékosazonosító) és a Teams tábla CaptainID-ja (Csapatkapitány-azonosító) között is van egy összekapcsolás.

Ha egyetlen lekérdezéssel akarjuk kiíratni a csapatnevet, a csapatkapitány nevét, és az összes játékos nevét, akkor a Bowlers táblának *kétszer* kell szerepelnie a lekérdezésben – egyszer azért, hogy összekapcsoljuk a CaptainID-t a csapatkapitány nevével, és egyszer azért, hogy összekapcsoljuk a TeamID-val, hogy megkapjuk az adott csapat tagjainak a listáját. Ebben az esetben *muszáj* álnevet rendelni a Bowlers tábla egyik vagy mindkét példányához, hogy az adatbázisrendszer meg tudja különböztetni az egyik példányt, amelyik a csapatkapitány nevéhez kapcsolódik, a másik példánytól, amelyikből a csapattagok listáját kapjuk majd meg. A fejezetben később láthatunk egy olyan példát, ahol a megoldáshoz egy táblából több példányt kell használni, illetve ezekhez álneveket kell rendelni. Az említett példa, amely a Bowling League adatbázist használja, a *Példák* rész *Kettőnél több tábla* című részében található.

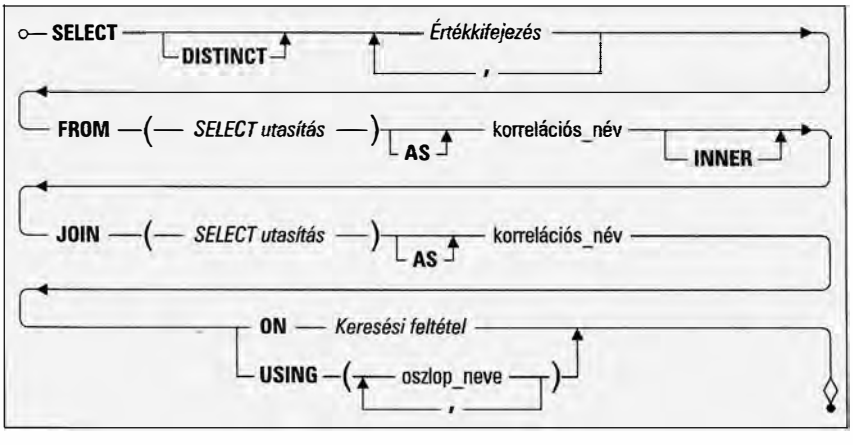
SELECT utasítások beágyazása

Tegyük a dolgokat kicsit érdekesebbé! A legtöbb SQL-megvalósításban a FROM záradékban a táblanév helyére behelyettesíthető egy teljes SELECT utasítás. Az SQL-szabvány az ilyen beágyazott SELECT utasítást *származtatott táblának* hívja. Ha belegondolunk, a SELECT utasítás valójában egy mód arra, hogy egy vagy több táblából adatok egy részhalmazát nyerjük ki. Természetesen hozzá kell rendelnünk egy korrelációs nevet, hogy a beágyazott lekérdezés végrehajtásából kapott eredményhalmaznak legyen neve. A 8.6. ábrán látható, hogyan kell a JOIN záradékba SELECT utasítást beágyazni.

Vegyük észre az ábrán, hogy a SELECT utasítás tartalmazhatja bármelyik lekérdezési záradékot, kivéve az ORDER BY-t. Ezenkívül az INNER JOIN kulcsszavak bármelyik oldalán használhatunk SELECT utasításokat vagy táblákat, vegyesen is.

Vessünk ismét egy pillantást a Recipes és a Recipe_Classes táblákra! Feltételezzük, hogy a feladat továbbra is csak a főételek és desszertek lekérdezése. Íme a lekérdezés új változata, amelyben a Recipe_Classes táblát egy olyan SELECT utasításon vezetjük keresztül, amely része az INNER JOIN záradéknak:

SQL
```
SELECT R.RecipeTitle, R.Preparation,
   RCFiltered.ClassName
FROM (SELECT RecipeClassID,
   RecipeClassDescription AS ClassName
   FROM Recipe_Classes AS RC
   WHERE RC.ClassName = 'Main course' OR
   RC.ClassName = 'Dessert') AS RCFiltered
INNER JOIN Recipes AS R
ON RCFiltered.RecipeClassID = R.RecipeClassID
```



8.6. ábra

A táblanevek SELECT-re cserélése egy JOIN-on belül

Megjegyzés *Néhány adatbázis-kezelő rendszer nem támogatja a SELECT utasítás FROM záradékba való beágyazását. Ha a rendszerünk nem támogatja ezt a szolgáltatást, akkor általában megoldja a problémát, ha a belső SELECT-et egy nézettáblába mentjük, és a nézettábla nevét használjuk a SELECT utasítás helyett.*

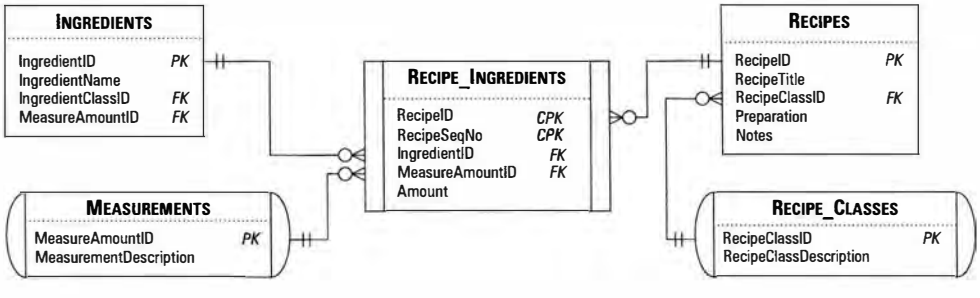
A CD-mellékleten található mintaadatbázisok egyikét a MySQL adatbázis-kezelő rendszerrel hoztuk létre, amely nem támogatja a SELECT utasítások beágyazását. Ha megnézzük ezeket a MySQL adatbázisokat a CD-n, azt találjuk majd, hogy a beágyazott SELECT-et igénylő lekérdezéseket úgy oldottuk meg, hogy egy nézettáblát mentettünk, majd a végső lekérdezésben a nézettábla nevét használtuk.

Figyelem! Először is, ha úgy döntünk, hogy egy táblanév helyére egy beágyazott SELECT utasítást teszünk, győződjünk meg róla, hogy nem csak azokat az oszlopokat szerepeltetjük, amelyeket a végeredményben szeretnénk majd látni, hanem a JOIN-ban az összekapcsoláshoz használt egyéb oszlopokat is. Ezért szerepel a beágyazott utasításban mind a RecipeClassID, mind a RecipeClassDescription. A játék kedvéért a RecipeClassDescription a beágyazott utasításban a ClassName (Osztálynév) álnevet kapta, ezért a SELECT záradék RecipeClassDescription helyett ClassName-re hivatkozik. Vegyük észre, hogy az ON záradék most a beágyazott SELECT utasítás korrelációs nevére – RCFiltered – hivatkozik a tábla eredeti neve vagy a táblához a beágyazott SELECT utasításban hozzárendelt korrelációs név helyett.

Ha az adatbázis-kezelő rendszerünk elég okosan tud optimalizálni, akkor a fenti módon megfogalmazott lekérdezéseknek ugyanolyan gyorsnak kell lenniük, mint a korábbi változat esetében, ahol a RecipeClassDescription szerinti szűrőt egy WHERE záradékkal adtuk meg a JOIN *után*. Sejthetjük, hogy az adatbázis-kezelő a lekérdezés leghatékonyabb teljesítése érdekében először megszűri a RecipeClass tábla sorait, mielőtt megkezdené az összehasonlítást a Recipes tábla soraival. Sokkal lassabb lehet ***először*** kapcsolni össze a Recipe_Classes összes sorát a Recipes tábla megfelelő soraival és *azután* alkalmazni a szűrőt. Ha úgy találjuk, hogy ez a lekérdezés több időt vesz igénybe, mint indokolt, akkor a WHERE záradékot egy a JOIN-ba ágyazott SELECT utasításba átrakva megpróbálhatjuk rávenni az adatbázis-kezelőt, hogy először végezze el a Recipe_Classes szerinti szűrést.

JOIN beágyazása JOIN-ba

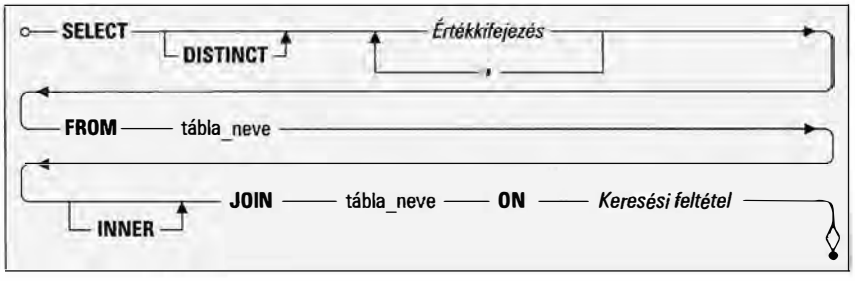
Habár sok feladat megoldható két tábla összekapcsolásával, az összes kívánt adat kinyeréséhez gyakran lesz szükségünk három, négy vagy több tábla összekapcsolására. Például megpróbálhatjuk a receptekhez kapcsolódó összes lényeges információt – a recept típusa, a recept neve és a recept hozzávalóinak listája – egyetlen lekérdezéssel megkapni. Az ezt a feladatot megoldó lekérdezéshez szükséges táblákat a 8.7. ábrán láthatjuk.



8.7. ábra

A receptek összes adatának lekérdezéséhez szükséges táblák a Recipes mintaadatbázisból

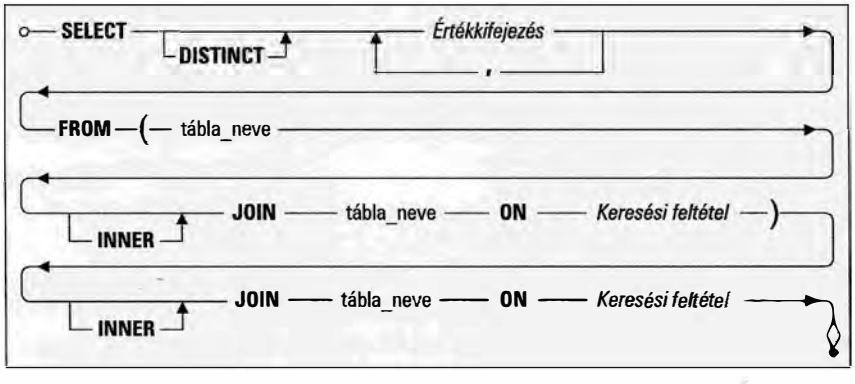
Úgy néz ki, *öt* különböző táblából lesz szükségünk adatokra! Ne aggódjunk – a feladatot meg tudjuk oldani egy bonyolultabb FROM záradékkal, amelyben a JOIN záradékokba más JOIN-okat ágyazunk. A trükk a következő: mindenhol, ahol szabályosan használható egy tábla neve, ott használhatunk egy zárójelek közé ágyazott teljes JOIN záradékot is. (A korrelációs nevet tartalmazó záradékokat itt elhagytuk, és az ON záradékot választottuk, hogy egy egyszerű, két tábla közötti JOIN-t adjunk meg.)



8.8. ábra

Egy egyszerű, két táblát összekapcsoló INNER JOIN

Most adjuk hozzá a harmadik táblát a koktélhoz, egyszerűen tegyünk egy nyitó zárójelet az első táblanév elé, egy bezárójelet a keresési feltétel után, majd írjunk a kifejezés végére egy INNER JOIN-t, egy táblanevet, az ON kulcsszót és még egy keresési feltételt. Az eredmény a 8.9. ábrán látható.



8.9 ábra

Egy egyszerű INNER JOIN három tábla között

Ha belegondolunk, a zárójelek közötti INNER JOIN, ami az első két táblát összekapcsolja, egy logikai táblát alkot, egy belső, átmeneti eredményhalmazt. Ez az eredményhalmaz veszi át az első példában, a 8.8. ábrán szereplő egyszerű táblanév helyét. Ezt az eljárást –

egy adott teljes JOIN záradék zárójelek közé helyezését és egy következő „JOIN kulcsszó, táblanév, ON kulcsszó és keresési feltétel" hozzáadását – tovább folytathatjuk, amíg megkapjuk azt a teljes eredményhalmazt, amit szerettünk volna. Írjunk egy olyan lekérdezést, amelyhez a 8.7. ábrán látható összes táblából kellenek majd adatok, és lássuk a megoldást!

„Írd ki a receptek típusát, nevét, elkészítésének menetét, a hozzávalók nevét, a hozzávalók sorszámát, a hozzávalók mennyiségét és a hozzávalók mennyiségének mértékegységét a receptek adatbázisából, sorszám szerint rendezve!"

Fordítás Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount,and measurement description from the recipe classes table joined with the recipes table on recipe class ID in the recipe classes table matching recipe class ID in the recipes table, then joined with the recipe ingredients table on recipe ID in the recipes table matching recipe ID in the recipe ingredients table, then joined with the ingredients table on ingredient ID in the ingredients table matching ingredient ID in the recipe ingredients table, and then finally joined with the measurements table on measurement amount ID in the measurements table matching measurement amount ID in the recipe ingredients table, order by recipe title and recipe sequence number

(Válaszd ki a receptosztály leírását, a recept címét, az elkészítés menetét, a hozzávaló nevét, sorszámát, mennyiségét és mértékegységét a receptosztályok táblájából, összekapcsolva a receptek táblájával a receptosztályok táblájában szereplő receptosztály-azonosító és a receptek táblájában levő receptosztály-azonosító szerint, majd ezt összekapcsolva a recepthozzávalók táblájával a receptek táblájában szereplő receptazonosító és a recepthozzávalók táblájában levő receptazonosító szerint, majd ezt összekapcsolva a hozzávalók táblájával a hozzávalók táblájában szereplő hozzávaló-azonosító és a recepthozzávalók táblájában levő hozzávaló-azonosító szerint, és végül ezt összekapcsolva a mennyiségek táblájával a recepthozzávalók táblájában szereplő mennyiségazonosító és a mennyiségek táblájában levő mennyiségazonosító szerint, a recept címe és sorszáma szerint rendezve.)

Tisztázás Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
inner ~~joined with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipe classes table matching~~ = recipes.recipe class ID ~~in the recipes table, then~~
inner ~~joined with the~~ recipe ingredients ~~table~~

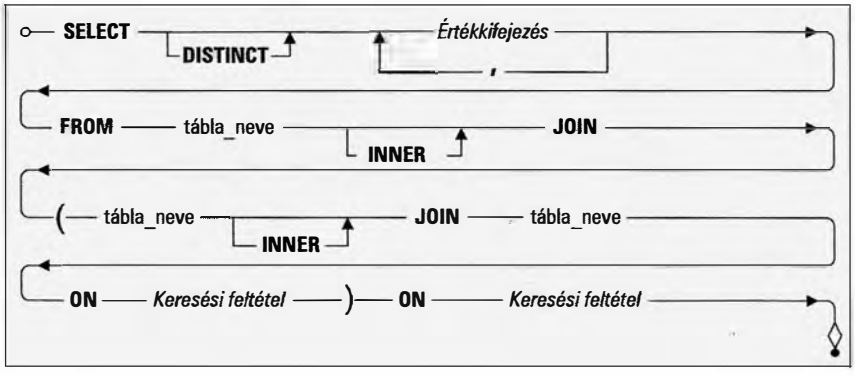
on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ ingredients.recipe ID ~~in the recipe ingredients table,then~~
inner ~~joined with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matching~~
= ingredients.ingredient ID ~~in the recipe ingredients table,and then~~
~~finally~~ inner ~~joined with the~~ measurements ~~table~~
on measurements.measurement amount ID ~~in the measurements table matching~~
= recipe ingredients.measurement amount ID ~~in the recipe ingredients table.~~
order by recipe title ~~and~~ recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
INNER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```

Ez igen! Szeretne valaki nekilátni és hozzáadni egy szűrőt, ami csak a főételek osztályában levő recepteket engedi át? Ha a válaszunk az, hogy csupán egy WHERE záradékra van szükség az ORDER BY záradék előtt, akkor fején találtuk a szöget.

Valójában ahol egyszerű táblanév áll, az bármikor helyettesíthető egy két táblát összekapcsoló JOIN záradékkal. A 8.9. ábráról az olvasható le, hogy először az első és a második táblát kapcsoljuk össze egymással, majd az eredményt a harmadik táblával. De összekapcsolhatjuk előbb a második és a harmadik táblát is (legalábbis ha a harmadik tábla a másodikkal van kapcsolatban, nem az elsővel) és azután az eredményt az első táblával. A 8.10 ábrán ez a változat látható.



8.10. ábra

Kettőnél több tábla összekapcsolása, más sorrendben

Megvilágíthatjuk a problémát egy festészeti hasonlattal. Ha ki szeretnénk keverni egy pasztellzöld színt, nem számít a keverés sorrendje. Ha előbb a fehér festéket a kékkel keverjük össze, majd az így kapott pasztellkékbe sárgát keverünk, a végeredmény ugyanaz a szín lesz, mint ha előbb a kék és a sárga festéket keverjük össze, majd a kapott zöldhöz keverünk fehéret. Az imént mutatott, öt táblát használó lekérdezést megvalósíthattuk volna a következő SQL kóddal is:

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
Recipes.RecipeTitle, Recipes.Preparation,
Ingredients.IngredientName,
Recipe_Ingredients.RecipeSeqNo,
Recipe_Ingredients.Amount,
Measurements.MeasurementDescription
FROM Recipe_Classes
INNER JOIN (((Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
Recipe_Ingredients.MeasureAmountID)
ON Recipe_Classes.RecipeClassID =
Recipes.RecipeClassID
ORDER BY RecipeTitle, RecipeSeqNo
```

Jó, ha tisztában vagyunk ezzel a lehetőséggel is, mivel bármikor találkozhatunk ilyen megvalósítással mások által írt lekérdezésekben vagy a Query By Example programmal létrehozott SQL kódban. Ezenkívül némelyik adatbázis-kezelő belső optimalizálója érzékeny a JOIN záradékok sorrendjére. Ha egy sok JOIN-t használó lekérdezés végrehajtását túl lassúnak találjuk egy nagy adatbázison, lehet, hogy gyorsíthatunk rajta a JOIN-ok sorrendjének megváltoztatásával az SQL-utasításban. Az egyszerűség kedvéért a fejezet során használt lekérdezésekben általában sorban, balról jobbra és fentről lefelé haladva követik egymást a JOIN-ok, a B függelékben található diagramoknak megfelelően.

Ellenőrizzük a kapcsolatokat!

Idáig eljutva látható, hogy a tábláink közötti kapcsolatok alapvető fontosságúak. Ha azt látjuk, hogy a számunkra szükséges adatok különböző táblákban helyezkednek el, akkor előfordulhat, hogy az előző példához hasonló bonyolultságú FROM záradékot kell létrehoznunk ahhoz, hogy az összes információmorzsát értelmes módon ki tudjuk gyűjteni. Ha nem vagyunk tisztában a tábláink közötti kapcsolatokkal és az összekötő oszlopokkal, amelyeken keresztül megvalósulnak ezek a kapcsolatok, akkor könnyen csapdába kerülhetünk.

Sokszor számos kapcsolatot kell végigkövetnünk, hogy megkapjuk a kívánt adatokat. Egyszerűsítsük például az előző lekérdezést, és kérjük csak a receptneveket és a hozzávalók neveit:

„Add meg az összes receptem nevét és az ezekhez a receptekhez tartozó összes hozzávaló nevét!"

Fordítás
Select the recipe title and the ingredient name from the
recipes table joined with the recipe ingredients table
on recipe ID in the recipes table matching
recipe ID in the recipe ingredients table,
and then joined with the ingredients table
on ingredient ID in the ingredients table matching
ingredient ID in the recipe ingredients table
(Válaszd ki a recept címét és a hozzávaló nevét a receptek táblájából, összekapcsolva a recepthozzávalók táblájával a recepthozzávalók táblájában levő receptazonosító és a receptek táblájában található receptazonosító szerint, majd ezt egyesítve a hozzávalók táblájával a hozzávalók táblájában szereplő hozzávaló-azonosító és a recepthozzávalók táblájában levő hozzávaló-azonosító szerint.)

Tisztázás
Select ~~the~~ recipe title ~~and the~~ ingredient name
from ~~the~~ recipes ~~table~~
inner ~~joined with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~

```
            and then inner joined with the ingredients table
            on ingredients.ingredient ID in the ingredients table matching
            = recipe_ingredients.ingredient ID in the recipe ingredients
            table
SQL         SELECT Recipes.RecipeTitle,
               Ingredients.IngredientName
            FROM (Recipes
            INNER JOIN Recipe_Ingredients
            ON Recipes.RecipeID =
               Recipe_Ingredients.RecipeID)
            INNER JOIN Ingredients
            ON Ingredients.IngredientID =
               Recipe_Ingredients.IngredientID
```

Feltűnt, hogy bár a Recipe_Ingredients (Recepthozzávalók) táblából *egyetlen oszlopra sem* volt szükségünk, a lekérdezésben mégis használnunk kellett? Ennek az az oka, hogy a Recipes és az Ingredients táblák csak a Recipe_Ingredients táblán *keresztül* vannak kapcsolatban.

Az INNER JOIN alkalmazási lehetőségei

Most, hogy már ismerjük az INNER JOIN használatának alapjait, nézzünk meg néhány feladatfajtát, amelyek megoldhatók a segítségével!

Kapcsolódó sorok keresése

Amint azt tudjuk, az INNER JOIN alkalmazása leggyakrabban táblák összekapcsolása abból a célból, hogy különböző, egymással kapcsolatban álló táblákból kérhessünk le adatokat. Álljon itt egy felsorolás, amely olyan lekérdezésekre mutat példákat a mintaadatbázisokból, amelyek megoldhatók az INNER JOIN-nal:

„Add meg a beszállítók listáját és a termékeket, amelyeket szállítanak nekünk!"
„Írd ki az alkalmazottakat és a vásárlókat, akiktől rendelést vettek fel!"
„Mutasd meg az ügynököket és az általuk lekötött rendezvényeket!"
„Sorold fel a megrendelőket és az általuk lekötött előadókat!"
„Keresd meg azokat az előadókat, akik felléptek Berg vagy Hallmark nevű megrendelőnk rendezvényein!"
„Sorold fel az épületeket és minden épület összes osztálytermét!"
„Írd ki a tanszék dolgozóit és az egyes tanárok óráit!"
„Mutasd meg az összes tekecsapatot és mindegyik csapat kapitányának a nevét!"
„Adj listát az összes tekecsapatról és a csapatok összes tagjáról!"
„Mutasd meg azokat a recepteket, amelyek marhahúst vagy fokhagymát tartalmaznak!"
„Sorold fel az összes hozzávalót azokhoz a receptekhez, amelyek tartalmaznak répát!"

A fejezet *Példák* című részében megmutatjuk, hogyan írhatók meg az ilyen és ehhez hasonló feladatokat megoldó lekérdezések.

Egyező értékek keresése

Az INNER JOIN egzotikusabb használata olyan sorok megkeresése kettő vagy több eredményhalmazban, amelyeknek egy vagy több olyan értéke egyezik meg, amelyek *nem* kapcsolódó kulcsértékek. Ha emlékszünk még, a 7. fejezetben azt olvashattuk, hogy létezik egy az INTERSECT-tel egyenértékű megoldás, ami az INNER JOIN-t használja. Következzen némi ízelítő olyan típusú feladatokból, amelyek megoldhatók ezzel a módszerrel:

„Sorold fel az ugyanolyan nevű vásárlókat és alkalmazottakat!"
„Írd ki azokat a vásárlókat és alkalmazottakat, akik ugyanabban a városban laknak!"
„Keresd meg az összes vásárlót, aki biciklit és bukósisakot is rendelt!"
„Keresd meg az összes ügynököt és előadót, akiknek ugyanaz az irányítószáma!"
„Sorold fel azokat az előadókat, akik felléptek Bonnicksen és Rosales megrendelőnk rendezvényein!
„Adj listát az ugyanolyan keresztnevű nevű hallgatókról és tanárokról!"
„Adj listát azokról a hallgatókról, akiknek rajzból legalább 85% az értékelése, és emellett számítástechnikából is legalább 85%-t értek el!"
„Keresd meg azokat a tekejátékosokat, akiknek ugyanaz az irányítószámuk!"
„Keresd meg azokat a tekejátékosokat, akiknek a tiszta pontszáma legalább 155, mind a Thunderbird Lanes, mind a Bolero Lanes pályán!"
„Keresd meg azokat a hozzávalókat, amelyekből ugyanakkora mennyiségre van szükség!"
„Mutasd meg azokat a recepteket, amelyek marhahúst és fokhagymát is tartalmaznak!"

A következő részben megmutatjuk, hogyan oldhatók meg az ehhez hasonló feladatok.

Példák

Már tudjuk, hogyan lehet INNER JOIN-ok segítségével lekérdezéseket írni, és láttunk pár példát olyan feladatokra, amelyek egy INNER JOIN-nal oldhatók meg. Most vessünk egy pillantást egy meglehetősen átfogó példahalmazra, amelynek példái mind INNER JOIN-t használnak. A példák között szerepel az összes mintaadatbázis, és a segítségükkel megtanulhatjuk, hogyan használható az INNER JOIN adatok kinyerésére két táblából, kettőnél több táblából, illetve hogyan oldhatunk meg feladatokat értékek egyezése alapján.

Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név ugyanaz, mint a CD-mellékleten található megfelelő SQL-lekérdezés neve. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH08"-cal kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Mivel a példák nagy része összetett JOIN-okat használ, előfordulhat, hogy az adatbázis-kezelőnk kicsit máshogy dolgozza fel ezeket a lekérdezéseket. Ezért lehet, hogy az eredményhalmazból általunk itt felsorolt első néhány sor nem pontosan egyezik meg azzal a válasszal, amit az otthoni futtatás során kapunk, de az összes sor száma a válaszban ugyanannyi kell, hogy legyen. Az egyszerűség kedvéért a továbbiakban a Fordítás és Tisztázás részeket összevonjuk.*

Két tábla

Induljunk ki az alapszínekből! Először olyan lekérdezéspéldákat mutatunk, amelyek csupán két tábla közötti INNER JOIN-t igényelnek.

Sales Orders adatbázis

„Jelenítsd meg az összes terméket a termékcsoportjukkal együtt!"

Fordítás/ Select category description ~~and~~ product name
Tisztázás from ~~the~~ categories ~~table~~
inner ~~joined with the~~ products ~~table~~
on categories.category ID ~~in the categories table matching~~
= products.category ID ~~in the products table~~
(Válaszd ki a termékcsoportot és a terméknevet a termékcsoportok táblájával összekapcsolt terméktáblából, ahol az összekapcsolás a termékcsoportok táblájában szereplő termékcsoport-azonosító és a termékek táblájában levő termékcsoport-azonosító szerint történik.)

SQL
```
SELECT Categories.CategoryDescription,
   Products.ProductName
FROM Categories
INNER JOIN Products
   ON Categories.CategoryID = Products.CategoryID
```

CH08_Products_And_Categories (40 sor)

CategoryDescription	ProductName
Accessories	Dog Ear Cyclecomputer
Accessories	Dog Ear Helmet Mount Mirrors
Accessories	Viscount C-500 Wireless Bike Computer
Accessories	Kryptonite Advanced 2000 U-Lock
Accessories	Nikoma Lok-Tight U-Lock
Accessories	Viscount Microshell Helmet
Accessories	Viscount CardioSport Sport Watch
Accessories	Viscount Tru-Beat Heart Transmitter
Accessories	Dog Ear Monster Grip Gloves
<< *további sorok* >>	

Entertainment Agency adatbázis

„Sorold fel az előadókat, a szerződésük kezdeti és befejező dátumát és a szerződés összegét!"

Fordítás/ Select entertainer stage name, start date, end date,
Tisztázás ~~and~~ contract price from ~~the~~ entertainers ~~table~~
inner join~~ed with the~~ engagements ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matching~~
= engagements.entertainer ID ~~in the engagements table~~
(Válaszd ki az előadó művésznevét, a kezdő dátumot, a befejező dátumot és a szerződés összegét az előadók táblájával INNER JOIN-nal összekapcsolt rendezvénytáblából, ahol az összekapcsolás az előadók táblájának előadóazonosítója és a rendezvények táblájának előadóazonosítója szerint történik.)

SQL
```
SELECT Entertainers.EntStageName,
   Engagements.StartDate, Engagements.EndDate,
   Engagements.ContractPrice
FROM Entertainers
INNER JOIN Engagements
   ON Entertainers.EntertainerID =
      Engagements.EntertainerID
```

CH08Entertainers_And_Contracts (111 sor)

EntStageName	StartDate	EndDate	ContractPrice
Carol Peacock Trio	2007-10-21	2007-10-21	$140.00
Carol Peacock Trio	2007-11-13	2007-11-19	$680.00
Carol Peacock Trio	2007-12-23	2007-12-26	$410.00
Carol Peacock Trio	2007-12-29	2008-01-07	$1,400.00
Carol Peacock Trio	2008-01-08	2008-01-08	$320.00
Carol Peacock Trio	2008-01-22	2008-01-30	$1,670.00
Carol Peacock Trio	2008-02-11	2008-02-19	$1,670.00
Carol Peacock Trio	2008-02-25	2008-02-28	$770.00
Carol Peacock Trio	2007-09-11	2007-09-18	$770.00
<< további sorok >>			

School Scheduling adatbázis

„Írd ki azokat a tantárgyakat, amelyekből van oktatás szerdán!"

Fordítás/ Select subject name
Tisztázás from ~~the~~ subjects ~~table~~
inner join~~ed with the~~ classes ~~table~~

on subjects.subject ID ~~in the subjects table matching~~
= classes.subject ID ~~in the classes table~~
where Wednesday schedule ~~is~~ = true
(Válaszd ki azokat a tantárgyneveket a tantárgyak táblájával INNER JOIN-nal összekapcsolt óratáblából – ahol az összekapcsolás a tantárgyak táblájában levő tárgyazonosító és az órák táblájában szereplő tárgyazonosító szerint történt –, ahol a szerdai beosztás értéke 'igaz'.)

SQL
```
SELECT DISTINCT Subjects.SubjectName
FROM Subjects
INNER JOIN Classes
   ON Subjects.SubjectID
   = Classes.SubjectID
WHERE Classes.WednesdaySchedule = -1
```

Megjegyzés *Mivel ugyanolyan órából több is lehet egy nap, a DISTINCT kulcsszót használtuk, ami kiszűri az ismétlődő sorokat. Egyes adatbázis-kezelő rendszerek támogatják a TRUE kulcsszót, de mi egy általánosabb, „minden bitjén egy értékű egész" értéket (–1) használtunk. Ha az adatbázis-kezelő rendszerünk az igaz-hamis értékeket egy biten tárolja, akkor az igaz érték ellenőrzéséhez 1-gyel is végezhetjük az összehasonlítást. A hamis értéket mindig a 0 jelenti.*

CH08_Subjects_On_Wednesday (45 sor)

SubjectName
Advanced English Grammar
Art History
Biological Principles
Business Tax Accounting
Chemistry
Composition–Fundamentals
Composition–Intermediate
Computer Art
Database Management
<< *további sorok* >>

Bowling League adatbázis

„Sorold fel a csapatokat és az egyes csapatok kapitányának nevét!"

Fordítás/ Tisztázás
Select team name ~~and~~ captain full name
from ~~the~~ teams ~~table~~
inner ~~joined with the~~ bowlers ~~table~~
on team captain ID ~~equals~~ = bowler ID

(Válaszd ki a csapatnevet és a csapatkapitány teljes nevét a csapatok táblájából, amelyet INNER JOIN-nal összekapcsoltunk a játékosok táblájával, a csapatkapitány-azonosító és a játékosazonosító szerint.)

SQL
```
SELECT Teams.TeamName, (Bowlers.BowlerLastName
   || ', ' || Bowlers.BowlerFirstName) AS CaptainName
FROM Teams
INNER JOIN Bowlers
   ON Teams.CaptainID = Bowlers.BowlerID
```

CH08_Teams_And_Captains (10 sor)

TeamName	CaptainName
Marlins	Fournier,David
Sharks	Patterson,Ann
Terrapins	Viescas, Carol
Barracudas	Sheskey, Richard
Dolphins	Viescas, Suzanne
Orcas	Thompson, Sarah
Manatees	Viescas, Michael
Swordfish	Rosales, Joe
Huckleberrys	Viescas,David
MintJuleps	Hallmark, Alaina

Recipes adatbázis

„Mutasd meg azokat a recepteket, amelyek marhahúst és fokhagymát is tartalmaznak!"

Fordítás/ Tisztázás
Select ~~unique~~ distinct recipe title
from ~~the~~ recipes ~~table~~
joined ~~with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table~~
where ingredient ID ~~is~~ in ~~the list of beef and garlic~~ IDs (1, 9)
(Válaszd ki azokat az egyedi receptcímeket a receptek táblájából, amelyet összekapcsoltunk a recepthozzávalók táblájával a recepttáblabeli receptazonosító és a recepthozzávalók táblájában levő receptazonosító szerint, ahol a hozzávaló azonosítója a marhahús és a fokhagyma alkotta listából kerül ki.)

SQL
```
SELECT DISTINCT Recipes.RecipeTitle
FROM Recipes
INNER JOIN Recipe_Ingredients
   ON Recipes.RecipeID = Recipe_Ingredients.RecipeID
WHERE Recipe_Ingredients.IngredientID IN (1, 9)
```

Megjegyzés *Mivel némelyik recept tartalmazhat mind marhahúst, mind fokhagymát, a DISTINCT kulcsszót alkalmaztuk, ami kiszűri az ismétlődő sorokat.*

CH08_Beef_Or_Garlic_Recipes (5 sor)

RecipeTitle
Asparagus
Garlic Green Beans
Irish Stew
Pollo Picoso
Roast Beef

Kettőnél több tábla

Most tegyük a dolgokat kicsit izgalmasabbá olyan kívánságokkal, amelyek kettőnél több tábla összekapcsolásával teljesíthetők!

Sales Orders adatbázis

„Keresd meg az összes vásárlót, aki valaha is rendelt bukósisakot!"

Fordítás/ Tisztázás
Select customer first name, customer last name
from ~~the~~ customers ~~table~~
inner ~~joined with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table matching~~
= orders.customer ID ~~in the orders table,~~
~~then~~ inner ~~joined with the~~ order details ~~table~~
on orders.order number ~~in the orders table matching~~
= order_details.order number ~~in the order details table, then~~
inner ~~joined with the~~ products ~~table~~
on products.product number ~~in the products table matching~~
= order_ details.product number ~~in the order details table~~
where product name ~~contains~~ LIKE '%Helmet%'

(Válaszd ki azoknak a vásárlóknak a keresztnevét és vezetéknevét a vásárlók táblájából, amelyet INNER JOIN-nak összekapcsoltunk a megrendelések táblájával a vásárlók táblájában levő vásárlóazonosító és a megrendelések táblájában található vásárlóazonosító szerint, majd ezt összekapcsoltuk egy INNER JOIN-nal a megrendelések részleteit tartalmazó táblával a megrendelések táblájának megrendelési száma és a megrendelés részleteit tartalmazó tábla megrendelési száma szerint, majd ezt összekapcsoltuk egy INNER JOIN-nal a termékek táblájával a termékek táblájának termékszáma és a megrendelés részleteit tartalmazó tábla termékszáma szerint, ahol a terméknév tartalmazza a 'Helmet' karakterláncot.)

SQL
```
SELECT DISTINCT Customers.CustFirstName,
   Customers.CustLastName
FROM ((Customers
INNER JOIN Orders
   ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
   ON Orders.OrderNumber =
      Order_Details.OrderNumber)
INNER JOIN Products
   ON Products.ProductNumber =
      Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%Helmet%'
```

Figyelem! *Ha az adatbázis-kezelő rendszerünk megkülönbözteti a kis- és nagybetűket a karakter típusú mezőkben, akkor erre oda kell figyelnünk, amikor megadjuk a keresési feltételt. Például sok adatbázis-kezelőben a 'helmet' (bukósisak) és a 'Helmet' különbözőnek számít.*

Megjegyzés *Lehet, hogy némelyik vásárló több alkalommal is rendelt bukósisakot, ezért a DISTINCT kulcsszót alkalmaztuk, ami kiszűri az ismétlődő sorokat.*

CH08_Customers_Who_Ordered_Helmets (25 sor)

CustFirstName	CustLastName
Andrew	Cencini
Angel	Kennedy
Caleb	Viescas
Darren	Gehring
David	Smith
Dean	McCrae
Estella	Pundt
Gary	Hallmark
Jim	Wilson
John	Viescas
<< *további sorok* >>	

Entertainment Agency adatbázis

„Találd meg az összes előadót, aki Berg vagy Hallmark nevű megrendelőink rendezvényein játszott!"

Fordítás/ Select ~~unique~~ distinct entertainer stage name
Tisztázás from ~~the~~ entertainers ~~table~~

inner joined with the engagements table
on entertainers.entertainer ID
in the entertainers table matching
= engagements.entertainer ID in the engagements table,
then inner joined with the customers table
on customers.customer ID in the customers table matching
= engagements.customer ID in the engagements table
where the customer last name is = 'Berg'
or the customer last name is = 'Hallmark'
(Válaszd ki azokat egyedi művészneveket az előadók táblájával INNER JOIN-nal összekapcsolt rendezvények táblájából – ahol az összekapcsolás az előadók táblájában levő előadóazonosító és a rendezvények táblájában található előadóazonosító szerint történt, majd ezt összekapcsoltuk a megrendelők táblájával a megrendelők táblájában levő megrendelőazonosító és a rendezvények táblájában található megrendelőazonosító szerint –, amelyeknél a megrendelő vezetékneve = 'Berg' vagy 'Hallmark'.)

SQL

```
SELECT DISTINCT Entertainers.EntStageName
FROM (Entertainers
INNER JOIN Engagements
   ON Entertainers.EntertainerID =
      Engagements.EntertainerID)
INNER JOIN Customers
   ON Customers.CustomerID =
      Engagements.CustomerID
WHERE Customers.CustLastName = 'Berg'
OR Customers.CustLastName = 'Hallmark'
```

CH08_Entertainers_For_Berg_OR_Hallmark (8 sor)

EntStageName
Carol Peacock Trio
Coldwater Cattle Company
Country Feeling
Jim Glynn
JV & the Deep Six
Modern Dance
Susan McLain
Topazz

Bowling League adatbázis

„Sorold fel az összes tornát, a tornák egyes mérkőzéseit és a játékok végeredményét!"

Fordítás/ Tisztázás

Select tourney ID, tourney location, match ID, lanes, odd lane team, even lane team, game number, game winner
from the tournaments table
inner joined with ~~the~~ tourney matches ~~table~~
on tournaments.tourney ID ~~in the tournaments table matching~~
= tourney_matches.tourney ID ~~in the tourney matches table,~~
~~then~~ inner joined with the teams ~~table aliased~~ as odd team
on oddteam.team ID ~~in the odd team table matches~~
= teams.odd lane team ID ~~in the teams table,~~
~~then~~ inner joined with the teams ~~table aliased~~ as even team
on eventeam.team ID ~~in the even team table matches~~
= tourney_matches.even lane team ID
~~in the tourney matches table,~~
~~then~~ inner joined with the match games ~~table~~
on match_games.match ID ~~in the match games table matches~~
= tourney_matches.match ID ~~in the tourney matches table,~~
~~then~~ inner joined with the teams ~~table aliased~~ as winner
on winner.team ID ~~in the winner table matches~~
= match_games.winning team ID ~~in the match games table~~
(Válaszd ki a torna azonosítóját, a torna helyszínét, a mérkőzés azonosítóját, a pályákat, a páros pályán levő csapatokat, a páratlan pályán levő csapatokat, a játék sorszámát, és a játék nyertesét a torna táblájából, amelyet INNER JOIN-nal összekapcsoltunk a mérkőzések táblájával a torna táblájában levő tornaazonosító és a mérkőzések táblájában található tornaazonosító szerint, majd ezt INNER JOIN-nal összekapcsoltuk a csapatok táblájával odd team (páratlan csapat) álnéven a páratlan csapatok mérkőzéseit tartalmazó tábla csapatazonosítója és csapatok táblájában levő páratlancsapat-azonosító szerint, majd ezt INNER JOIN-nal összekapcsoltuk a csapatok táblájával even team (páros csapat) álnéven a páros csapatok mérkőzéseit tartalmazó tábla csapatazonosítója és a torna mérkőzéseit tartalmazó tábla pároscsapat-azonosítója szerint, majd ezt INNER JOIN-nal összekapcsoltuk a játékok táblájával a játékok táblájának mérkőzésazonosítója és a mérkőzések táblájának mérkőzésazonosítója szerint, majd ezt INNER JOIN-nal összekapcsoltuk a csapatok táblájával winner (nyertes) álnéven a tábla csapatazonosítója és a játékok táblájának nyertescsapat-azonosítója szerint.)

SQL

```
SELECT Tournaments.TourneyID AS Tourney,
Tournaments.TourneyLocation AS Location,
Tourney_Matches.MatchID, Tourney_Matches.Lanes,
OddTeam.TeamName AS OddLaneTeam,
EvenTeam.TeamName AS EvenLaneTeam,
```

```
Match_Games.GameNumber AS GameNo, Winner.TeamName AS
Winner
FROM Tournaments
INNER JOIN Tourney_Matches
ON Tournaments.TourneyID
= Tourney_Matches.TourneyID
INNER JOIN Teams AS OddTeam
ON OddTeam.TeamID
= Tourney_Matches.OddLaneTeamID
INNER JOIN Teams AS EvenTeam
ON EvenTeam.TeamID
= Tourney_Matches.EvenLaneTeamID
INNER JOIN Match_Games
ON Match_Games.MatchID
= Tourney_Matches.MatchID
INNER JOIN Teams AS Winner
ON Winner.TeamID
= Match_Games.WinningTeamID
```

Megjegyzés *Ez egy igazán izgalmas lekérdezés, mivel egyetlen tábla (Teams) három példányára van szükség a megoldáshoz. A táblák közül legalább kettőnek korrelációs nevet kellett adnunk, hogy szabályos legyen a lekérdezés, de inkább mind a háromnak adtunk álnevet, ami tükrözi a lekérdezésben betöltött szerepüket. Ezenkívül az SQL rész megírásakor nem követtük pontosan a Fordítás/Tisztázás részt. Így szemléletesen látszik, hogy az egymásba ágyazott JOIN-okat úgy határozhatjuk meg, ahogyan jólesik, mindaddig, amíg nem keverjük össze a kapcsolódásokat.*

CH08_Tournament_Match_Game_Results (168 sor)

Tourney	Location	MatchID	Lanes	OddLaneTeam	EvenLaneTeam	GameNo	Winner
1	Red Rooster Lanes	1	01-02	Marlins	Sharks	1	Marlins
1	Red Rooster Lanes	1	01-02	Marlins	Sharks	2	Sharks
1	Red Rooster Lanes	1	01-02	Marlins	Sharks	3	Marlins
1	Red Rooster Lanes	2	03-04	Terrapins	Barracudas	1	Terrapins
1	Red Rooster Lanes	2	03-04	Terrapins	Barracudas	2	Barracudas
1	Red Rooster Lanes	2	03-04	Terrapins	Barracudas	3	Terrapins
1	Red Rooster Lanes	3	05-06	Dolphins	Orcas	1	Dolphins
1	Red Rooster Lanes	3	05-06	Dolphins	Orcas	2	Orcas
1	Red Rooster Lanes	3	05-06	Dolphins	Orcas	3	Dolphins
<< további sorok >>							

Megjegyzés *Habár úgy tűnik, hogy a sorok tornák és mérkőzések szerint rendezettek, ez csupán az a sorrend, amelyben az általunk használt adatbázis-kezelő (jelen esetben a Microsoft Office Access) visszaadta az eredmény sorait. Ha biztosak akarunk lenni benne, hogy a kapott sorok bizonyos szempont szerint rendezettek, a lekérdezést ki kell egészítenünk egy ORDER BY záradékkal.*

Recipes adatbázis

„Mutasd meg a főételek receptjeit és a hozzávalóik listáját!"

Fordítás/ Tisztázás

Select recipe title, ingredient name,
measurement description, ~~and~~ amount
from ~~the~~ recipe classes ~~table~~
inner ~~joined with the~~ recipes ~~table~~
on recipes.recipe class ID ~~in the recipes table matches~~
= recipe_classes.recipe class ID ~~in the recipe classes table,~~
~~then~~ inner ~~joined with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
~~then~~ inner ~~joined with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table,~~
~~and finally~~ inner ~~joined with the~~ measurements ~~table~~
on measurements.measure amount ID
~~in the measurements table matches~~
= recipe_ingredients.measure amount ID
~~in the recipe ingredients table,~~
where recipe class description ~~is~~ = 'Main course'
(Válaszd ki a recept címét, a hozzávaló nevét, a mennyiség leírását és mértékegységét a receptosztályok táblájával INNER JOIN-nal összekapcsolt recepttáblából, ahol az összekapcsolás a receptosztályok táblájában levő receptosztály-azonosító és a receptek táblájának receptosztály-azonosítója szerint történt, majd ezt INNER JOIN-nal összekapcsoltuk a recepthozzávalók táblájával a recepthozzávalók táblájának receptazonosítója és a receptek táblájában levő receptazonosító szerint, majd ezt INNER JOIN-nal összekapcsoltuk a hozzávalók táblájával a hozzávalók táblájának hozzávaló-azonosítója és a recepthozzávalók táblájában levő hozzávaló-azonosító szerint, majd ezt INNER JOIN-nal összekapcsoltuk a mennyiségek táblájával a mennyiségtábla mennyiségazonosítója és a recepthozzávalók táblájának mennyiségazonosítója szerint, ha a receptosztály leírása 'Main course', vagyis főétel.)

SQL

```
SELECT Recipes.RecipeTitle,
   Ingredients.IngredientName,
   Measurements.MeasurementDescription,
   Recipe_Ingredients.Amount
FROM (((Recipe_Classes
```

```
INNER JOIN Recipes
ON Recipes.RecipeClassID =
   Recipe_Classes.RecipeClassID)
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID
WHERE Recipe_Classes.RecipeClassDescription =
   'Main course'
```

Figyelem! *A MeasureAmountID (mennyiségazonosító) mind az Ingredients (Hozzávalók) táblában, mind a Recipe_Ingredients (Recepthozzávalók) táblában szerepel. Ha az utolsó JOIN-t úgy határozzuk meg, hogy a Recipe_Ingredients tábla helyett az Ingredients tábla MeasureAmountID-ját használja, akkor az adott hozzávaló alapértelmezett mennyiségét fogjuk visszakapni, ahelyett, ami a receptben szerepel.*

CH08_Main_Course_Ingredients (53 sor)

RecipeTitle	IngredientName	MeasurementDescription	Amount
Irish Stew	Beef	Pound	1
Irish Stew	Onion	Whole	2
Irish Stew	Potato	Whole	4
Irish Stew	Carrot	Whole	6
Irish Stew	Water	Quarts	4
Irish Stew	Guinness Beer	Ounce	12
Fettuccini Alfredo	Fettuccini Pasta	Ounce	16
Fettuccini Alfredo	Vegetable Oil	Tablespoon	1
Fettuccini Alfredo	Salt	Teaspoon	3
<< további sorok >>			

Egyező értékek keresése

Végül bővítsük a lehetőségeket még egy dimenzióval: az utolsó példahalmazban olyan lekérdezéseket találunk, amelyek két vagy több eredményhalmaz vagy tábla azonos értékeire alkalmaznak JOIN-t. (Ha az adatbázis-kezelő rendszerünk támogatja az INTERSECT kulcsszót, akkor az alábbi problémák közül sokat megoldhatunk az eredményhalmazok metszetével is.)

Sales Orders adatbázis

„Keresd meg az összes vásárlót, aki kerékpárt és bukósisakot is rendelt!"

Ez egy elég egyszerű feladatnak tűnik – talán túlságosan is egyszerűnek. Tegyük fel a kérdést most más módon, úgy, hogy világosabb legyen, mit is várunk el az adatbázis-kezelőtől:

„Találd meg az összes vásárlót, aki rendelt kerékpárt, majd keresd meg az összes vásárlót, aki rendelt bukósisakot, és végül írd ki a két listából a közös vásárlókat, hogy megkapjuk, kik rendeltek kerékpárt és bukósisakot is!"

1. fordítás Select customer first name and customer last name from those common to the set of customers who ordered bicycles and the set of customers who ordered helmets
(Válaszd ki azoknak a vásárlóknak a keresztnevét és a vezetéknevét, akik szerepelnek mind a kerékpárt rendelt vásárlók között, mind a bukósisakot rendelt vásárlók között.)

2. fordítás/ tisztázás
Select customer first name ~~and~~ customer last name from
(Select ~~unique~~ distinct customer name,
customer first name, customer last name
from ~~the~~ customers ~~table~~
inner joined ~~with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~then~~ inner joined ~~with the~~ order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_ details.order number ~~in the order details table,~~
~~then~~ inner joined ~~with the~~ products ~~table~~
on products.product number ~~in the products table matches~~
= order_details.product number ~~in the order details table~~
where product name ~~contains~~ LIKE '%Bike') as cust bikes
inner joined ~~with~~
(Select ~~unique~~ distinct customer ID
from ~~the~~ customers ~~table~~
inner joined ~~with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~then~~ inner joined ~~with the~~ order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_ details.order number ~~in the order details table,~~
~~then~~ joined ~~with the~~ products ~~table~~
on products.product number ~~in the products table matches~~
= order_details.product number ~~in the order details table~~
where product name ~~contains~~ LIKE '%Helmet') as cust helmets
on cust bikes.customer ID ~~in the cust bikes table matches~~
= cust helmets.customer ID ~~in the cust helmets table~~

SQL

```
SELECT CustBikes.CustFirstName,
   CustBikes.CustLastName
FROM
   (SELECT DISTINCT Customers.CustomerID,
      Customers.CustFirstName,
      Customers.CustLastName
    FROM ((Customers
    INNER JOIN Orders
    ON Customers.CustomerID
      = Orders.CustomerID)
    INNER JOIN Order_Details
    ON Orders.OrderNumber =
      Order_Details.OrderNumber)
    INNER JOIN Products
    ON Products.ProductNumber =
      Order_Details.ProductNumber
    WHERE Products.ProductName LIKE '%Bike')
  AS CustBikes
INNER JOIN
   (SELECT DISTINCT Customers.CustomerID
    FROM ((Customers
    INNER JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID)
    INNER JOIN Order_Details
    ON Orders.OrderNumber =
      Order_Details.OrderNumber)
    INNER JOIN Products
    ON Products.ProductNumber =
      Order_Details.ProductNumber
    WHERE Products.ProductName LIKE '%Helmet')
  AS CustHelmets
ON CustBikes.CustomerID =
   CustHelmets.CustomerID
```

Megjegyzés *A második beágyazott SELECT utasítást egyszerűsítettük, hogy csak a vásárlóazonosítót adja vissza, mert a két halmaz INNER JOIN-nal történő összekapcsolásához csak erre az oszlopra van szükség. Valójában a Customers táblával való összekapcsolást el is hagyhattuk volna, és a vásárlóazonosítót kiolvashattuk volna az Orders táblából. Emlékezzünk vissza, hogy a FROM záradékokba beágyazott SELECT utasításokra „logikai táblákként" gondolhatunk, és minden utasításhoz egyedi nevet rendelhetünk, hogy megírjuk a végső ON záradékot.*

A fenti problémát a két halmaz metszetével is megoldhatjuk, de az INTERSECT használata esetén a kimenet összes oszlopát bele kell foglalnunk mindkét eredményhalmazba, amelyből metszetet képzünk. Őszintén szólva ez nem biztos, hogy a legjobb módja a probléma megoldásának. A 11. fejezetben bemutatjuk, hogyan érhetünk célt hatékonyabban allekérdezések segítségével.

CH08_Customers_Both_Bikes_And_Helmets (21 sor)

CustFirstName	CustLastName
William	Thompson
Robert	Brown
Dean	McCrae
John	Viescas
Mariya	Sergienko
Neil	Patterson
Andrew	Cencini
Angel	Kennedy
Liz	Keyser
Rachel	Patterson
<< *további sorok* >>	

Entertainment Agency adatbázis

„Sorold fel azokat az előadókat, akik a Berg és a Hallmark nevű megrendelőknek is játszottak!"

Amint korábban láttuk, a Berg *vagy* Hallmark megoldása könnyű. Fogalmazzuk meg máshogy a feladatot, hogy egyértelműbb legyen, mit várunk el az adatbázis-kezelőtől:

„Találd meg az összes előadót, aki játszott Berg egy rendezvényén, majd keresd meg az összes előadót, aki játszott Hallmark egy rendezvényén, és végül mutasd meg a közös előadókat, hogy megkapjuk, kik játszottak mindkét megrendelő rendezvényén!"

1. fordítás Select entertainer stage name from those common to the set of entertainers who played for Berg and the set of entertainers who played for Hallmark
(Válaszd ki azoknak az előadóknak a művésznevét, akik szerepelnek mind a Berg rendezvényén fellépett előadók között, mind a Hallmark rendezvényen fellépett előadók között.)

2. fordítás/ Tisztázás Select entertainer stage name
from (Select ~~unique~~ distinct entertainer stage name
from ~~the~~ entertainers ~~table~~
inner joined ~~with the~~ engagements ~~table~~

on entertainers.entertainer ID ~~in the entertainers table matches~~
= engagements.entertainer ID ~~in the engagements table,~~
~~then~~ inner join~~ed with the~~ customers ~~table~~
on customers.customer ID ~~in the customers table matches~~
= engagements.customer ID ~~in the engagements table~~
where customer last name ~~is~~ = 'Berg') as entberg
inner join~~ed with~~
(Select ~~unique~~ distinct entertainer stage names
from ~~the~~ entertainers ~~table~~
inner join~~ed with the~~ engagements ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= engagements.entertainer ID ~~in the engagements table, then~~
join~~ed with the~~ customers ~~table~~
oncustomers.customer ID ~~in the customers table matches~~
= engagements.customer ID ~~in the engagements table~~
where customer last name ~~is~~ = 'Hallmark') as enthallmark
on entberg.entertainer ID ~~in the entberg table matches~~
= enthallmark.entertainer ID ~~in the enthallmark table~~

SQL

```
SELECT EntBerg.EntStageName
FROM
   (SELECT DISTINCT Entertainers.EntertainerID,
      Entertainers.EntStageName
    FROM (Entertainers
    INNER JOIN Engagements
    ON Entertainers.EntertainerID =
      Engagements.EntertainerID)
    INNER JOIN Customers
    ON Customers.CustomerID =
      Engagements.CustomerID
    WHERE Customers.CustLastName = 'Berg')
  AS EntBerg
INNER JOIN
   (SELECT DISTINCT Entertainers.EntertainerID,
      Entertainers.EntStageName
    FROM (Entertainers
    INNER JOIN Engagements
    ON Entertainers.EntertainerID =
      Engagements.EntertainerID)
    INNER JOIN Customers
    ON Customers.CustomerID =
      Engagements.CustomerID
    WHERE Customers.CustLastName = 'Hallmark')
```

```
    AS EntHallmark
ON EntBerg.EntainerID =
   EntHallmark.EntainerID
```

CH08_Entertainers_Berg_AND_Hallmark(4 sor)

EntStageName
Carol Peacock Trio
JV & the Deep Six
Modern Dance
Country Feeling

Megjegyzés *Ez egy újabb példa egy olyan feladatra, amit az INTERSECT használatával is megoldhatunk. Allekérdezésekkel még hatékonyabban célt érhetünk; ezekről a 11. fejezetben olvashatunk.*

School Scheduling adatbázis

„Mutasd meg azokat a hallgatókat és tanárokat, akiknek ugyanaz a keresztnevük!"

Fordítás/ Tisztázás: Select student full name ~~and~~ staff full name
from ~~the~~ students ~~table~~
inner ~~joined with the~~ staff ~~table~~
on students.first name ~~in the students table matches~~
= staff.first name ~~in the staff table~~
(Válaszd ki azoknak a hallgatóknak és tanároknak a teljes nevét a hallgatók és a tanárok táblájából, amelyeket a két tábla keresztnév mezői szerint összekapcsoltunk egy INNER JOIN-nal, akiknél a keresztnév megegyezik.)

SQL

```
SELECT (Students.StudFirstName || ' ' ||
   Students.StudLastName) AS StudFullName,
   (Staff.StfFirstName || ' ' ||
   Staff.StfLastName) AS StfFullName
FROM Students
INNER JOIN Staff
ON Students.StudFirstName = Staff.StfFirstName
```

CH08_Students_Staff_Same_FirstName (2 sor)

StudFullName	StfFullName
Michael Viescas	Michael Hernandez
David Hamilton	David Smith

Bowling League adatbázis

„Keresd meg azokat a játékosokat, akiknek a tiszta pontszámuk legalább 170 mind a Thunderbird Lanes, mind a Bolero Lanes pályákon!"

Valóban, ez egy újabb „csináljunk metszetet JOIN-nal" feladat. Tegyük fel a kérdést másképp, hogy egyértelműbb legyen, mit várunk el az adatbázis-kezelőtől:

„Keresd meg az összes játékost, akinek a tiszta pontszáma legalább 170 a Thunderbird Lanes pályán, majd keresd meg az összes játékost, akinek a tiszta pontszáma legalább 170 a Bolero Lanes pályán, és végül írd ki a közös játékosokat, hogy megkapjuk, kinek volt jó eredménye mindkét tekepályán!"

1. fordítás Select bowler full name from those common to the set of bowlers who have a score of 170 or better at Thunderbird Lanes and the set of bowlers who have a score of 170 or better at Bolero Lanes
(Válaszd ki a játékos teljes nevét azok közül, akik szerepelnek mind a Thunderbird Lane pályán legalább 170 pontot elértek között, mind a Bolero Lane pályán legalább 170 pontot elértek között.)

2. fordítás/ Tisztázás
Select bowler full name
from (Select ~~unique~~ distinct bowler ID ~~and~~ bowler full name
from ~~the~~ bowlers ~~table~~
inner joined ~~with the~~ bowler scores ~~table~~
on bowlers.bowler ID ~~in the bowlers table matches~~
= bowler_scores.bowler ID ~~in the bowler scores table,~~
~~then~~ inner joined ~~with the~~ tourney matches ~~table~~
on tourney_matches.match ID
~~in the tourney matches table matches~~
= bowler_scores.match ID ~~in the bowler scores table,~~
~~and finally~~ inner joined ~~with the~~ tournaments ~~table~~
on tournaments.tourney ID ~~in the tournaments table matches~~
= tourney_matches.tourney ID ~~in the tourney matches table~~
where tourney location ~~is~~ = 'Thunderbird Lanes'
and raw score ~~is greater than or equal to~~ >= 170) as bowlertbird
inner joined ~~with~~
(Select ~~unique~~ distinct bowler ID ~~and~~ bowler full name
from ~~the~~ bowlers ~~table~~
inner joined ~~with the~~ bowler scores ~~table~~
on bowlers.bowler ID ~~in the bowlers table matches~~
= bower_scores.bowler ID ~~in the bowler scores table,~~
~~then~~ inner joined ~~with the~~ tourney matches ~~table~~
on tourney_matches.match ID
~~in the tourney matches table matches~~
= bowler_scores.match ID ~~in the bowler scores table,~~

and finally inner joined with the tournaments table
on tournaments.tourney ID in the tournaments table matches
= tourney_matches.tourney ID in the tourney matches table
where tourney location is = 'Bolero Lanes'
and raw score is greater than or equal to >= 170) as bowlerbolero
on bowlertbird.bowler ID in the bowlertbird table matches
= bowlerbolero.bowler ID in the bowlerbolero table

SQL

```
SELECT BowlerTbird.BowlerFullName
FROM
   (SELECT DISTINCT Bowlers.BowlerID,
      (Bowlers.BowlerLastName || ', ' ||
      Bowlers.BowlerFirstName) AS BowlerFullName
    FROM ((Bowlers
    INNER JOIN Bowler_Scores
    ON Bowlers.BowlerID = Bowler_Scores.BoWlerID)
    INNER JOIN Tourney_Matches
    ON Tourney_Matches.MatchID =
       Bowler_Scores.MatchID)
    INNER JOIN Tournaments
    ON Tournaments.TourneyID =
       Tourney_Matches.TourneyID
    WHERE Tournaments.TourneyLocation =
       'Thunderbird Lanes'
    AND Bowler_Scores.RawScore >= 170)
  AS BowlerTbird
INNER JOIN
   (SELECT DISTINCT Bowlers.BowlerID,
       (Bowlers.BowlerLastName || ', ' ||
       Bowlers.BowlerFirstName) AS BowlerFullName
    FROM ((Bowlers
    INNER JOIN Bowler_Scores
    ON Bowlers.BowlerID = Bowler_Scores.BowlerID)
    INNER JOIN Tourney_Matches
    ON Tourney_Matches.MatchID =
       Bowler_Scores MatchID)
    INNER JOIN Tournaments
    ON Tournaments.TourneyID =
       Tourney_Matches.TourneyID
    WHERE Tournaments.TourneyLocation =
       'Bolero Lanes'
    AND Bowler_Scores.RawScore >= 170)
  AS BowlerBolero
ON BowlerTbird.BowlerID = BowlerBolero.BowlerID
```

Megjegyzés *Mivel egy tekejátékosnak bármelyik pályán lehetett egynél többször is nagy pontszáma, a DISTINCT kulcsszót használtuk, amely kiszűri az ismétlődő sorokat. Ez is egy olyan feladat, amit könnyebb allekérdezésekkel megoldani, amelyekről a 11. fejezetben olvashatunk.*

CH08_Good_Bowlers_TBird_And_Bolero (11 sor)

BowlerFullName
Kennedy, John
Patterson, Neil
Kennedy,Angel
Patterson, Kathryn
Viescas, John
Viescas, Caleb
Thompson, Sarah
Thompson, Mary
Thompson,William
Patterson, Rachel
Pundt, Steve

Recipes adatbázis

„Mutasd meg az összes hozzávalót azokhoz a receptekhez, amelyek tartalmaznak répát!"

Fordítás/ Select recipe ID, recipe title, ~~and~~ ingredient name
Tisztázás from ~~the~~ recipes ~~table~~
inner ~~joined with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
inner ~~joined with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table,~~
~~then finally~~ inner ~~joined with~~
(Select recipe ID from ~~the~~ ingredients ~~table~~
inner ~~joined with the~~ recipe ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table~~
where ingredient name ~~is~~ = 'Carrot') as carrots
on recipes.recipe ID ~~in the recipes table matches~~
= carrots.recipe ID ~~in the carrots table~~

SQL
```
SELECT Recipes.RecipeID, Recipes.RecipeTitle,
```

```
   Ingredients.IngredientName
FROM ((Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN
   (SELECT Recipe_Ingredients.RecipeID
    FROM Ingredients
    INNER JOIN Recipe_Ingredients
    ON Ingredients.IngredientID =
        Recipe_Ingredients.IngredientID
    WHERE Ingredients.IngredientName = 'Carrot')
  AS Carrots
ON Recipes.RecipeID = Carrots.RecipeID
```

CH08_Recipes_Containing_Carrots (16 sor)

RecipeID	RecipeTitle	IngredientName
1	Irish Stew	Beef
1	Irish Stew	Onion
1	Irish Stew	Potato
1	Irish Stew	Carrot
1	Irish Stew	Water
1	Irish Stew	Guinness Beer
14	Salmon Filets in Parchment Paper	Salmon
14	Salmon Filets in Parchment Paper	Carrot
14	Salmon Filets in Parchment Paper	Leek
14	Salmon Filets in Parchment Paper	Red Bell Pepper
14	Salmon Filets in Parchment Paper	Butter
<< további sorok >>		

Megjegyzés *Ez a kérelem egyszerűbben is megfogalmazható egy allekérdezéssel. Ennek a menetét a 11. fejezetben láthatjuk majd.*

Összefoglalás

A fejezet során alaposan kitárgyaltuk, hogy milyen módon lehet két vagy több táblát vagy eredményhalmazt összekapcsolni egyező mezőkön keresztül. A JOIN fogalmának meghatározásával kezdtük, majd feltártuk az INNER JOIN-ok készítésének részleteit. Megmutattuk, mik a JOIN-nál alkalmazható „megengedett" feltételek, és felhívtuk a figyelmet az értelmetlen JOIN-ok veszélyére.

Egy egyszerű példával, két tábla összekapcsolásával kezdtünk. Ez után megmutattuk, hogyan lehet a FROM záradékban a táblákhoz korrelációs (ál-) nevet rendelni. Az álnevek használatának lehetnek kényelmi okai, de vannak olyan esetek is, amikor az alkalmazásuk elkerülhetetlen – például amikor ugyanaz a tábla egyszerre több helyen szerepel, vagy ha beágyazott SELECT-et használunk.

Megmutattuk, hogyan helyettesíthető egy táblahivatkozás a FROM záradékban egy megfelelő beágyazott SELECT utasítással. Ez után új lehetőségeket feltárva megmutattuk, hogyan lehet kettőnél több táblát vagy eredményhalmazt összekapcsolni. Az INNER JOIN utasításformája tárgyalásának befejezéseként az átgondolt adatbázisszerkezet fontosságát hangsúlyoztuk, illetve azt, hogy tisztában kell lennünk a tábláink közötti kapcsolatokkal.

Felsoroltuk az okait – egy sor példával szemléltetve –, hogy miért hasznos az INNER JOIN. A fejezet további részében több mint egy tucat, INNER JOIN-t alkalmazó példát mutattunk be. A példákat három részre osztottuk: két tábla közötti JOIN-ok, kettőnél több tábla közötti JOIN-ok és egyező értékek szerinti JOIN-ok. A következő fejezetben bemutatjuk a JOIN egy másik változatát – ez lesz az OUTER JOIN –, fejezetünk végén pedig olyan kérelmeket találunk, amelyeket önállóan dolgozhatunk ki.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Sorold fel a vásárlókat és a dátumokat, amikor rendelést adtak fel, a dátum szerint rendezve!"*
   (Tipp: a megoldáshoz két tábla közötti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Customers_And_OrderDates (944 sor).
2. *„Írd ki a dolgozókat azokkal a vásárlókkal együtt, akiknek a részére rendelést vettek fel!"*

(Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Employees_And_Customers (211 sor).

3. *„Jelenítsd meg az összes rendelést, az egyes rendelésekhez tartozó termékeket, az egyes termékekért járó összeget, a rendelési szám szerint rendezve!"*
(Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Orders_With_Products (3975 sor).
4. *„Mutasd meg a beszállítókat és azokat a termékeket, amelyeket 100 dolláros ár alatt szállítanak nekünk!"*
(Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Vendors_And_Products_Less_Than_100 (66 sor).
5. *„Keresd meg azokat a vásárlókat és alkalmazottakat, akiknek ugyanaz a vezetéknevük!"*
(Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Customers_And_Employees_Same_LastName (16 sor).
6. *„Keresd meg azokat a vásárlókat és alkalmazottakat, akik ugyanabban a városban laknak!"*
(Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Customers_Employees_Same_City (10 sor).

Entertainment Agency adatbázis

1. *„Jelenítsd meg az előadókat és a lekötött fellépéseket a lekötés kezdő dátuma szerint rendezve!"*
(Tipp: a megoldáshoz két tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Agents_Booked Dates (111 sor).
2. *„Írd ki a megrendelőket és az általuk lekötött előadókat!"*
(Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Customers_Booked_Entertainers (75 sor).
3. *„Keresd meg azokat az ügynököket és előadókat, akiknek ugyanaz az irányítószámuk!"*
(Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Agents_Entertainers_Same_Postal (10 sor).

School Scheduling adatbázis

1. *„Jelenítsd meg az épületeket és az egyes épületekben levő osztálytermeket!"*
(Tipp: a megoldáshoz két tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Buildings_Classrooms (44 sor).
2. *„Sorold fel a hallgatókat és az összes órát, amit felvettek!"*
(Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
A megoldás itt található: CH08_Student_Enrollments (33 sor).

3. *„Sorold fel a tanszék dolgozóit és hogy ki milyen tárgyat oktat!"*
   (Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Staff_Subjects (110 sor).
4. *„Keresd meg azokat a hallgatókat, akiknek legalább 85%-os az értékelésük rajzból, és emellett valamilyen számítástechnikai tárgyból is legalább 85%-ot értek el!"*
   (Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Good_Art_CS_Students (1 sor).

Bowling League adatbázis

1. *„Sorold fel a csapatokat és az összes csapattagot!"*
   (Tipp: a megoldáshoz két tábla közötti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Teams_And_Bowlers (32 sor).
2. *„Jelenítsd meg a játékosokat, hogy mely mérkőzéseken játszottak, valamint az egyes játékokban elért pontszámokat!"*
   (Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Bowler_Game_Scores (1344 sor).
3. *„Keresd meg azokat a játékosokat, akiknek ugyanaz az irányítószámuk!"*
   (Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Bowlers_Same_ZipCode (92 sor).

Recipes adatbázis

1. *„Sorold fel az összes salátareceptet!"*
   (Tipp: a megoldáshoz két tábla közötti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Salads (1 sor).
2. *„Sorold fel az összes receptet, amely tartalmaz tejterméket!"*
   (Tipp: a megoldáshoz kettőnél több tábla közötti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Recipes_Containing_Dairy (2 sor).
3. *„Keresd meg azokat a hozzávalókat, amelyekből az alapértelmezett mennyiségre van szükség!"*
   (Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Ingredients_Same_Measure (628 sor).
4. *„Mutasd meg azokat a recepteket, amelyek tartalmaznak marhahúst és fokhagymát!"*
   (Tipp: a megoldáshoz egyező értékek szerinti JOIN-ra lesz szükség.)
   A megoldás itt található: CH08_Beef_And_Garlic_Recipes (1 sor).

9

Külső összekapcsolás

„A probléma és a megoldás közötti egyetlen különbség, hogy a megoldást az ember megérti."
– Charles Franklin Kettering, feltaláló, 1876-1958

A fejezet témakörei

- Mi az OUTER JOIN?
- A LEFT és a RIGHT OUTER JOIN
- A FULL OUTER JOIN
- Példák
- Összefoglalás
- Önálló feladatok

Az előző fejezetben az INNER JOIN műveleteket tárgyaltuk, vagyis azt, hogy miként kapcsolhatunk egymáshoz két vagy több táblát, illetve eredményhalmazt, az egyező sorokat keresve. Eljött az ideje az OUTER JOIN műveletek tárgyalásának: ezúttal úgy kapcsolunk össze táblákat, hogy az egyező sorokon túl az eltérőeket is megtaláljuk.

Mi az OUTER JOIN?

Ahogy az előző fejezetben elmondtuk, az SQL-szabvány két vagy több tábla, illetve eredményhalmaz összekapcsolásához a JOIN műveletek több típusát határozza meg. Az OUTER JOIN (külső összekapcsolás) segítségével nem csak az általunk megadott feltételeknek megfelelő sorokat nyerhetjük ki egy adatbázisból, hanem az összekapcsolt adathalmazok egyikéből vagy másikából – esetleg mindkettőből – azokat is, amelyek nem felelnek meg ezeknek a feltételeknek.

Tételezzük fel példának okáért, hogy a School Scheduling adatbázisból akarunk a hallgatókról és az általuk felvett órákról információt nyerni. Ahogy az előző fejezetben megtanultuk, az INNER JOIN csak a legalább egy órára feliratkozott hallgatókat, illetve a legalább egy hallgató által felvett órákat mutatja meg. Nem mutatja meg azokat a hallgatókat,

akik felvételt nyertek az iskolába, de még nem iratkoztak fel egyetlen órára sem, és azokat az órákat sem, amelyek ugyan felvehetők, de egyetlen hallgató sem mutatott érdeklődést irántuk.

Mi a teendő akkor, ha fel akarjuk soroltatni az összes hallgatót, és ha vettek már fel órákat, akkor azokat is? Ehhez hasonló probléma, ha ki akarjuk íratni az összes órát és azokat a hallgatókat is, akik felvették az órákat – ha vannak már ilyenek. Az efféle feladatok megoldására valók az OUTER JOIN műveletek.

A 9.1. ábra halmazdiagram segítségével mutatja be hallgatók és az órák egy lehetséges kapcsolatát. Ahogy láthatjuk, néhány hallgató még nem jelentkezett egy órára sem, és néhány órát még egyetlen hallgató sem vett fel.

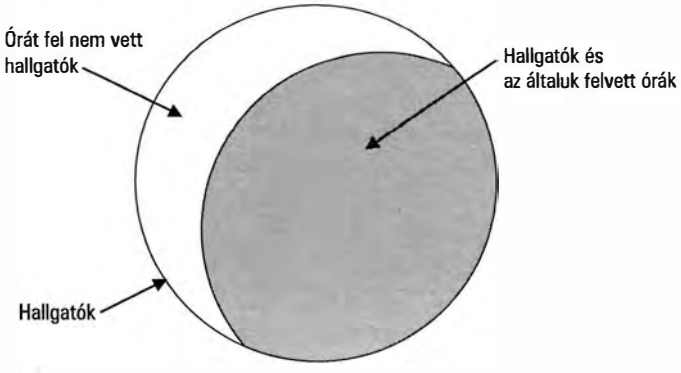
9.1. ábra

Hallgatók és órák egy lehetséges kapcsolata

Ha az összes hallgató és az általuk felvett órák listájára vagyunk kíváncsiak, a 9.2. ábrán alapuló eredményt fogjuk kapni.

Felmerülhet a kérdés: „Mit fogunk látni azoknál a hallgatóknál, akik egy órát sem vettek fel?" Ha eszünkbe jut a Null érték, más szóval a „semmi" fogalma, amelyet az 5. fejezetben tárgyaltunk, akkor már tudjuk is, hogy mit fogunk látni. Ha az adatbázisrendszer egy lekérdezés során, amelyben a hallgatókra és a hozzájuk kapcsolt órákra vagyunk kíváncsiak, olyan hallgatóhoz ér, aki még egyetlen órát sem vett fel, a Classes (Órák) tábla minden oszlopára Null értéket fog visszaadni. A két halmaz különbségének fogalmára gondolva, amit a 7. fejezetben tárgyaltunk, a Classes tábla Null értéket tartalmazó sorai az összes hallgató halmaza, illetve a legalább egy órát felvett hallgatók halmaza közötti különbséget jelentik.

Ehhez hasonlóan, az összes órát, illetve a legalább egy órát felvett hallgatókat kiíratő lekérdezés esetében a Students tábla Null értéket tartalmazó üres sorai az összes óra halmaza és a legalább egy hallgató által felvett órák halmaza közötti különbséget jelentik. Ahogy ígértük, az OUTER JOIN-nal Null értékekre keresve egy másik megoldást kapunk arra, hogy megtaláljuk két halmaz különbségét. Az EXCEPT művelettel szemben, amely két halmaz teljes soraira illeszkedik, a JOIN művelettel az illesztés néhány oszlopra (rendszerint az elsődleges és az idegen kulcsra) korlátozható.



9.2. ábra

Az összes hallgató és az általuk felvett órák

A LEFT és a RIGHT OUTER JOIN

Az OUTER JOIN utasítást általában olyan esetekben használjuk, ahol látni akarjuk az egyik tábla vagy eredményhalmaz összes sorát és a másik tábla vagy eredményhalmaz illeszkedő sorait. Ehhez a LEFT OUTER JOIN vagy a RIGHT OUTER JOIN utasítás a megfelelő eszköz.

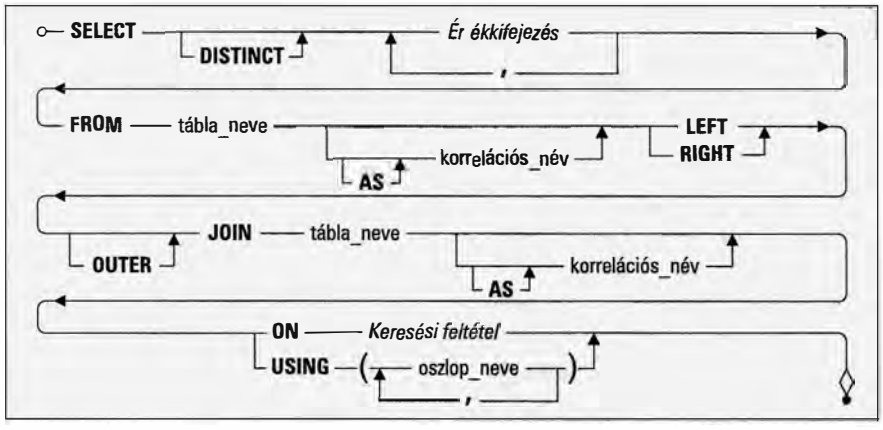
Mi a különbség LEFT (bal) és RIGHT (jobb) között? Idézzük fel az előző fejezetből, hogy az INNER JOIN művelet két táblára alkalmazható, mégpedig a következőképpen: megnevezzük az első táblát, beírjuk a JOIN kulcsszót, majd megnevezzük a második táblát. Ha a lekérdezéshez OUTER JOIN utasítást használunk, akkor az SQL-szabvány az elsőként megnevezett táblát tekinti a bal oldalon lévőnek, a másodikként megnevezettet pedig a jobb oldalon lévőnek. Így amennyiben látni szeretnénk az első tábla minden sorát és a második tábla illeszkedő sorait, a LEFT OUTER JOIN utasítást kell használnunk. Ha pedig a második tábla minden sorát és az első tábla illeszkedő sorait szeretnénk látni, akkor a RIGHT OUTER JOIN utasítást kell megadnunk.

Utasításforma

Vizsgáljuk meg a LEFT vagy RIGHT OUTER JOIN műveletet tartalmazó utasítások felépítésekor használt utasításformát!

Táblák használata

Kezdjük egy egyszerű, táblákat használó OUTER JOIN-t tartalmazó utasítás felépítésével. A két tábla felhasználásával készült, OUTER JOIN utasítást tartalmazó lekérdezés felépítéséhez szükséges szintaxisdiagramot a 9.3. ábra mutatja be.



9.3. ábra

OUTER JOIN meghatározása két táblára

Az INNER JOIN-hoz hasonlóan (amelyet a 8. fejezetben tárgyaltunk) minden művelet a FROM záradékban történik. (A dolgok egyszerűsítése végett a WHERE és az ORDER BY záradékokat itt elhagytuk.) Egyetlen táblanév megadása helyett két táblanevet adunk meg, a JOIN kulcsszóval összekapcsolva. Ha nem adjuk meg a kívánt JOIN típusát, az adatbázisrendszer INNER JOIN-t feltételez. Ez esetben, lévén, hogy OUTER JOIN-t szeretnénk, meg kell adnunk, hogy LEFT vagy RIGHT JOIN-ra gondolunk. Az OUTER kulcsszó elhagyható.

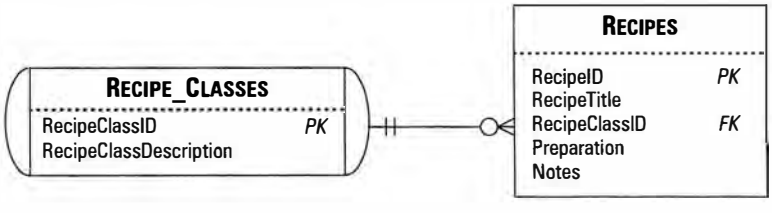
Megjegyzés *Azoknak, akik az „A" függelékben található teljes szintaxisdiagramokat is követik, érdemes tudniuk, hogy a megfelelő részeket (Select utasítás, Táblahivatkozás, Kapcsolt tábla) leegyszerűsítettük, hogy a tárgyalt utasításformára összpontosíthassunk.*

Minden JOIN legfontosabb része a második táblanevet követő ON vagy USING záradék, ami azt tudatja az adatbázisrendszerünkkel, hogy a JOIN hogyan hajtandó végre. A JOIN kiértékelésekor az adatbázisrendszer logikailag kombinálja az első tábla minden sorát a második tábla minden sorával (az ilyen kombinációt, amelynek során az első tábla minden sorát kombináljuk a második tábla minden sorával, *Descartes-szorzatnak* vagy *direkt szorzatnak* nevezzük), majd az ON vagy USING záradékban lévő feltételek vizsgálatával megkeresi a visszaadandó illeszkedő sorokat. Tekintve, hogy OUTER JOIN műveletet kértünk, az adatbázisrendszer visszaadja a LEFT (bal oldali) vagy RIGHT (jobb oldali) tábla nem illeszkedő sorait is.

Már tanultunk a keresési feltételek WHERE záradékká alakításáról a 6. fejezetben. A JOIN utáni ON záradékban használt keresési feltétellel logikai vizsgálatot végeztethetünk el, amelynek igaz eredményt kell adnia, hogy az adatbázisrendszer a két összekapcsolt sort visszaadja. Csak olyan keresési feltételt van értelme írni, amely az első táblának legalább egy oszlopát a második tábla legalább egy oszlopához hasonlítja. Bár meglehetősen összetett keresési feltétel írható, általában elvégeztethetünk egy egyszerű egyenlőségvizsgálatot az egyik tábla elsődleges kulcsa, illetve a második tábla idegen kulcsa között.

Hogy ne bonyolítsuk a dolgokat, kezdjük az utolsó fejezetben használt, receptosztályokról és receptekről szóló példával. Ne feledjük, hogy egy jól megtervezett adatbázisban az összetett osztályneveket el kell különíteni egy második táblába, amelyből a neveket aztán egy egyszerű kulcsértékkel kapcsoljuk az első táblához. A Recipes (Receptek) mintaadatbázisban a receptosztályok és a receptek külön táblákban vannak. A Recipe_Classes (Receptosztályok) és a Recipes táblák kapcsolatát a 9.4. ábra mutatja be.

Amikor először végiggondoltuk, hogy a tárolandó receptjeink milyen osztályokba sorolhatók, lehet, hogy az összes eszünkbe jutó receptosztály nevét gépre vittük. Most, pár recept begépelése után jó volna megtudni, hogy mely csoportok nem tartalmaznak még recepteket. Ezen felül jó lenne kiíratni az összes receptosztályt, osztályonként a már beírt receptek címeivel. Mindkét probléma megoldható egy OUTER JOIN utasítással.



9.4. ábra

A receptosztályok és a receptek külön táblákban vannak

Megjegyzés *A fejezet során a 4. fejezetben bevezetett „Kérelem – Fordítás – Tisztázás – SQL" módszert követjük.*

„Mutasd meg az adatbázis összes recepttípusát a hozzájuk tartozó receptekkel!"

Fordítás Select recipe class description and recipe title from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matching recipe class ID in the recipes table

(Válaszd ki a receptosztályok meghatározását és a receptek címét a receptosztályok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a receptek táblájával, a receptosztályok táblájában lévő receptosztály-azonosítót a receptek táblájában található receptosztály-azonosítónak megfeleltetve.)

Tisztázás Select recipe class description ~~and~~ recipe title
from ~~the~~ recipe classes ~~table~~
left outer joined ~~with the~~ recipes ~~table~~
on recipe_ classes.recipe class ID ~~in the recipe classes table~~
~~matching~~ = recipes.recipe class ID ~~in the recipes table~~

SQL
```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
```

Ha a FROM záradékban több táblát is megadunk, ne feledkezzünk meg arról, hogy minden oszlopnevet minden előforduláskor a táblanévvel együtt adjunk meg, hogy teljesen egyértelmű legyen, hogy melyik tábla melyik oszlopára utalunk. Vegyük észre, hogy a RecipeClassID (Receptosztály-azonosító) oszlop nevét a táblanévvel együtt *kellett* megadnunk, hiszen két RecipeClassID nevű oszloppal dolgoztunk – a Recipes táblában lévővel, illetve a Recipe_Classes táblában találhatóval.

Megjegyzés *Bár az SQL legtöbb kereskedelmi megvalósítása támogatja az OUTER JOIN-t, van néhány, amelyik nem. Ha az adatbázisunk nem támogatja az OUTER JOIN-t, akkor úgy oldhatjuk meg a problémát, hogy minden szükséges táblát felsorolunk a FROM záradékban, majd az ON záradékban szereplő keresési feltételeket a WHERE záradékba tesszük. Az adatbázisunk által megkívánt – a szabványostól eltérő – OUTER JOIN utasítást helyettesítő utasításformáról az adatbázisunk dokumentációjából tudhatunk meg bővebbet. Például a Microsoft SQL Server korai változatai az alábbi utasításformát fogadják el (figyeljünk a WHERE záradékban lévő csillagra!):*

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID *=
   Recipes.RecipeClassID
```

Ha az Oracle-t használjuk, a lehetséges utasításforma a következő (figyeljünk a WHERE záradékban szereplő pluszjelre!):

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID(+)
```

Őszintén szólva, ezeket a furcsa utasításformákat azoknak az adatbázisrendszereknek a készítői találták ki, amelyekben az OUTER JOIN művelet biztosította képességet már jóval azelőtt meg akarták valósítani, hogy az utasítás bekerült volna az SQL-szabványba. Szerencsére az SQL-szabványban meghatározott utasításforma lehetővé teszi a számunkra, hogy a végső eredményhalmaz forrását teljes egészében a FROM záradékon belül adjuk meg. Gondoljunk úgy a FROM záradékra, mint egy olyan kapcsolt eredményhalmaz teljes megadására, amelyből az adatbázisrendszer kinyeri a válaszokat. Az SQL-szabvány szerint a WHERE záradékot csak arra használjuk, hogy a FROM záradékkal meghatározott eredményhalmazból sorokat szűrjünk ki. Tekintve, hogy a WHERE záradék használatával létrehozott OUTER JOIN-szerű működés adatbázisrendszerenként változó nyelvtant jelent, többféle, a szabványtól eltérő rendszer használata esetén többféle utasításforma megtanulására lehet szükségünk.

Ha a mintául szolgáló lekérdezést a Recipes példaadatbázison futtatjuk, 16 sort kell visszakapnunk. Minthogy levesreceptet nem vittünk be az adatbázisba, Null értéket kell kapnunk a RecipeTitle (Receptcím) oszlopnak abban a sorában, ahol a RecipeClassDescription (Receptosztály leírása) oszlopban „Soup" (leves) szerepel. Ennek a sornak a megtalálásához a következőképp járjunk el:

„Sorold fel azokat a receptosztályokat, amelyekben még nincs recept!"

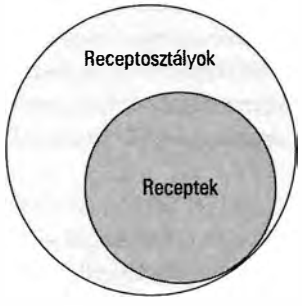
Fordítás Select recipe class description from the recipe classes table
left outer joined with the recipes table on recipe class ID where
recipe ID is empty
(Válaszd ki azoknak a receptosztályoknak a meghatározását a receptosztályok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a receptek táblájával a receptosztály-azonosítók szerint, amelyeknél a receptazonosító üres.)

Tisztázás Select recipe class description
from ~~the~~ recipe classes ~~table~~
left outer ~~joined with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipes table matches~~
= recipes.recipe class ID ~~in the recipes table~~
where recipe ID is ~~empty~~ NULL

SQL

```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
WHERE Recipes.RecipeID IS NULL
```

Ha végiggondoljuk, most éppen egy különbségképzést, vagyis egy EXCEPT műveletet végeztük el (lásd a 7. fejezetet), de JOIN utasítást használva. Kicsit olyan ez, mintha annyit mondanánk, hogy *„Mutasd meg az összes receptosztályt, kivéve azokat, amelyek a receptek táblájában is előfordulnak."* A 9.5. ábrán látható halmazdiagram segít elképzelni, hogy mi is történik.



9.5. ábra

Receptosztályok és receptek egy lehetséges kapcsolata

A 9.5. ábrán minden recepthez tartozik receptosztály, de vannak olyan receptek, amelyekhez még nem határoztunk meg receptosztályt. Az IS NULL vizsgálattal azokat a világosabb, külső körben lévő sorokat kérjük el a rendszertől, amelyekhez nem tartozik semmi a belső, sötétebb körrel jelölt receptek közül.

Vegyük észre, hogy az OUTER JOIN utasításnak a 9.3. ábrán bemutatott szintaxisdiagramján látható egy elhagyható USING záradék is. Ha a vizsgált oszlopoknak a két táblában azonos a nevük, és egyenlőségvizsgálatot akarunk végezni rajtuk, használhatjuk a USING záradékot is, az oszlopnevek megjelölésével. Oldjuk meg újra az előző problémát, ezúttal a USING használatával!

„Jelenítsd meg azokat a receptosztályokat, amelyekben még nincs recept!"

Fordítás Select recipe class description from the recipe classes table
left outer joined with the recipes table using recipe class ID where
recipe ID is empty
(Válaszd ki azoknak a receptosztályoknak a meghatározását a receptosztályok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a receptek táblájával a receptosztály-azonosítók szerint, amelyeknél a receptazonosító üres.)

Tisztázás Select recipe class description
from ~~the~~ recipe classes ~~table~~
left outer ~~joined with the~~ recipes ~~table~~ using recipe class ID
where recipe ID is ~~empty~~ NULL

```
SQL     SELECT Recipe_Classes.RecipeClassDescription
        FROM Recipe_Classes
        LEFT OUTER JOIN Recipes
        USING (RecipeClassID)
        WHERE Recipes.RecipeID IS NULL
```

Az utasítás alakja lényegesen egyszerűbb a USING használatával, igaz? Van azonban egy aprócska csapda a dologban: a USING záradékban szereplő minden oszlop elveszti a táblájával a kapcsolatot, ugyanis az SQL-szabvány kimondja, hogy az adatbázisrendszernek „egybe kell olvasztania" a két oszlopot. Így aztán ebben a példában eredményként egyetlen RecipeClassID oszlopot kapunk, amelyre nem hivatkozhatunk a SELECT vagy más záradékokban, sem Recipes.RecipeClassID-ként, sem Recipe_Classes.RecipeClassID-ként.

Figyelnünk kell arra is, hogy néhány adatbázisrendszer még nem támogatja a USING záradékot. Ha ezzel a problémával kerülünk szembe az adatbázisrendszerünk használata során, akkor mindig helyettesíthetjük a USING záradékot egyenlőséget vizsgáló ON záradékkal.

Megjegyzés *Az SQL-szabvány meghatároz még egy további JOIN típusú műveletet, a NATURAL JOIN-t (természetes összekapcsolás). A NATURAL JOIN utasítás két táblát az összes azonos nevű oszlop összehasonlításával kapcsol össze. Ha az adatbázisrendszerünk ismeri a NATURAL JOIN-t, és csak a vizsgált oszlopok közösek, akkor az előző példafeladat így is megoldható:*

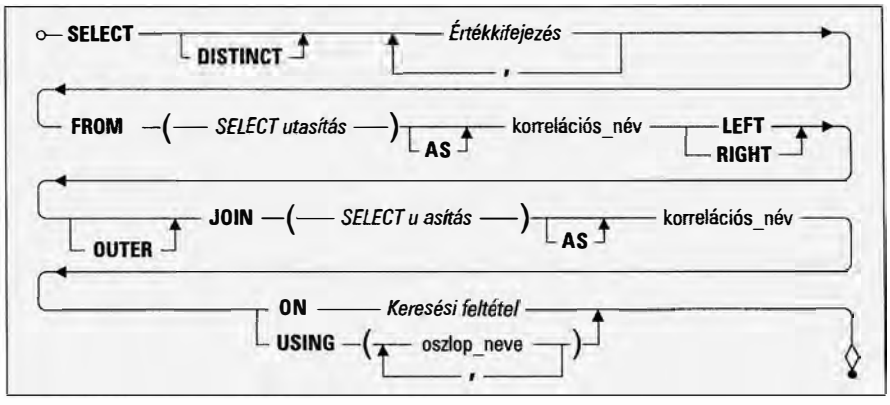
```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
NATURAL LEFT OUTER JOIN Recipes
WHERE Recipes.RecipeID IS NULL
```

NATURAL JOIN használatakor ne adjunk meg ON vagy USING záradékot!

SELECT utasítás beágyazása

Ahogy a 8. fejezetben láthattuk, a legtöbb SQL-megvalósítás megengedi, hogy a FROM záradék bármely táblanevét egy teljes SELECT utasítással helyettesítsük. Persze ilyenkor meg kell adnunk egy korrelációs nevet (az álnevek hozzárendelésével kapcsolatban lásd a 8. fejezetet), hogy a beágyazott lekérdezés kiértékelésekor kapott eredményre hivatkozni tudjunk. Az OUTER JOIN beágyazott SELECT utasítással történő használatát a 9.6. ábra mutatja be.

Vegyük észre, hogy a SELECT-tet tartalmazó kifejezések – eltekintve az ORDER BY záradéktól – az összes lekérdezésben használható záradékot tartalmazhatják. Az OUTER JOIN kulcsszavak bármely oldalán a táblanevekkel azonos módon használhatunk SELECT utasításokat.



9.6. ábra

OUTER JOIN beágyazott SELECT utasításokkal

Lássuk ismét a Recipes és a Recipe_Classes táblákat. Ebben a példában tételezzük fel, hogy csak a saláták (Salads), a levesek (Soups) és a főételek (Main courses) érdekesek a számunkra. Íme a SELECT-tel megszűrt Recipe_Classes tábla, amelyet LEFT OUTER JOIN művelettel összekapcsoltunk a Recipes táblával:

```
SQL     SELECT RCFiltered.ClassName, R.RecipeTitle
        FROM
           (SELECT RecipeClassID,
               RecipeClassDescription AS ClassName
            FROM Recipe_Classes AS RC
            WHERE RC.ClassName = 'Salads'
               OR RC.ClassName = 'Soup'
               OR RC.ClassName = 'Main Course')
          AS RCFiltered
        LEFT OUTER JOIN Recipes AS R
        ON RCFiltered.RecipeClassID = R.RecipeClassID
```

Ha a SELECT utasítást FROM záradékban használjuk, óvatosnak kell lennünk. Ha úgy döntünk, hogy egy táblanevet SELECT utasításra cserélünk, akkor figyeljünk oda, hogy ne csak a végeredményben látható oszlopokat vegyük fel a SELECT paraméterei közé, hanem azokat is, amelyek a JOIN művelethez kellenek. Ezért látjuk a beágyazott kifejezésben a RecipeClassDescription oszlopot és a RecipeClassID oszlopot is. Pusztán a szórakozás kedvéért a beágyazott utasításban a RecipeClassDescription oszlopnak a ClassName (Osztálynév) álnevet adtuk. Ennek eredményeképpen a külső SELECT utasítás a RecipeClassDescription oszlop helyett a ClassName oszloppal dolgozik. Észrevehetjük, hogy az ON záradék itt már a beágyazott SELECT utasításban szereplő RCFiltered korrelációs névre hivatkozik az eredeti táblanév vagy ahelyett a korrelációs név helyett, amelyet a beágyazott SELECT utasítás belsejében rendeltünk a táblához.

Tekintve, hogy a Recipes mintaadatbázisunkban egyetlen levesrecept sem szerepel, a lekérdezés futtatása után egyetlen sort fogunk látni, amelyben a RecipeClassDescription oszlop értéke leves (Soup), és amely mellett a RecipeTitle oszlopban Null érték szerepel. Ugyanilyen egyszerű dolog olyan lekérdezést írni, ahol a Recipes táblán futtatott SELECT utasítás az OUTER JOIN művelet jobb oldalán áll. Példának okáért az alábbi módon kerestethettük volna ki a „beef" szót a címükben tartalmazó recepteket:

SQL
```
SELECT RCFiltered.ClassName, R.RecipeTitle
FROM
   (SELECT RecipeClassID,
      RecipeClassDescription AS ClassName
    FROM Recipe_Classes AS RC
    WHERE RC.ClassName = 'Salads'
      OR RC.ClassName = 'Soup'
      OR RC.ClassName = 'Main Course')
  AS RCFiltered
LEFT OUTER JOIN
   (SELECT Recipes.RecipeClassID, Recipes.Recipe
      Title
    FROM Recipes
    WHERE Recipes.RecipeTitle LIKE '%beef%')
  AS R
ON RCFiltered.RecipeClassID = R.RecipeClassID
```

Ne feledjük, hogy a LEFT OUTER JOIN utasítás a JOIN művelet bal oldalán álló eredményhalmaz vagy tábla összes sorában keres, nem törődve azzal, hogy a jobb oldalon áll-e illeszkedő sor. Az előző lekérdezés, azon felül, hogy visszaad egy olyan sort, amelyben a Soup mellett a RecipeTitle oszlop Null értéket tartalmaz, visszaad egy hasonlóan Null értékkel bíró Salad sort is. Úgy vélhetnénk, hogy az adatbázisban nincsenek salátareceptek, holott *vannak*, ámde egyik salátarecept címe sem tartalmazza a „beef" szót.

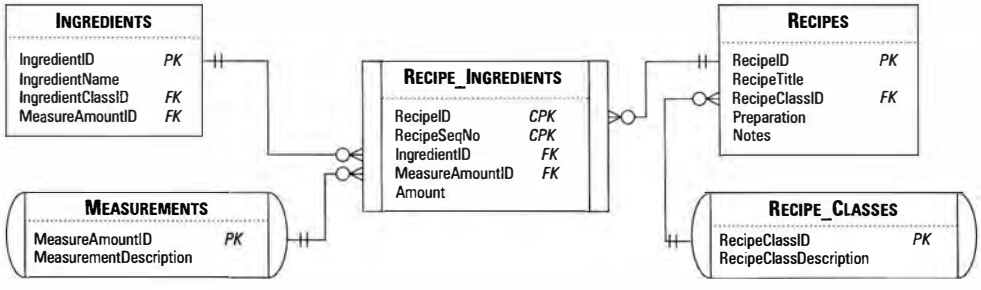
Megjegyzés *Talán feltűnt, hogy a JOIN utasítás utáni ON záradékba akár teljes keresési feltételt is beírhatunk. Ez teljesen igaz, példafeladatunk következő megoldása az SQL-szabvány szerint tökéletesen szabályos:*

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
AND
   (Recipe_Classes.RecipeClassDescription = 'Salads'
OR Recipe_Classes.RecipeClassDescription = 'Soup'
OR Recipe_Classes.RecipeClassDescription =
   'Main Course')
AND Recipes.RecipeTitle LIKE '%beef%'
```

Vizsgálódásaink szerint néhány jelentős SQL-megvalósítás sajnos a fenti kódot hibásan futtatja, vagy egyáltalán el sem fogadja, így hát azt tanácsoljuk, hogy az ON záradék keresési feltételében mindig két táblát vagy eredményhalmazt összehasonlító feltételt adjunk meg. Ha a szóban forgó táblák sorait szűrni akarjuk, akkor tegyük ezt egy beágyazott SELECT utasítás WHERE záradékaként megfogalmazott keresési feltétellel.

JOIN utasítások egymásba ágyazása

Bár sok probléma megoldható mindössze két tábla összekapcsolásával, kérdéseink egy részére csak három, négy, vagy esetleg ennél is több tábla összekapcsolásával kaphatunk választ. Példának okáért tegyük fel, hogy a receptekről minden fontos információt – a recept típusát, a recept címét és az összes hozzávalót – egyetlen lekérdezéssel akarjuk megkapni. Most, hogy értjük, hogy az OUTER JOIN művelet mire való, akár kiírathatjuk az *összes* receptosztályt – azokat is, amelyekhez még nem adtunk meg recepteket – és az összes receptet, minden részletükkel együtt. A 9.7. ábra minden, a kérdés megválaszolásához szükséges táblát feltüntet.

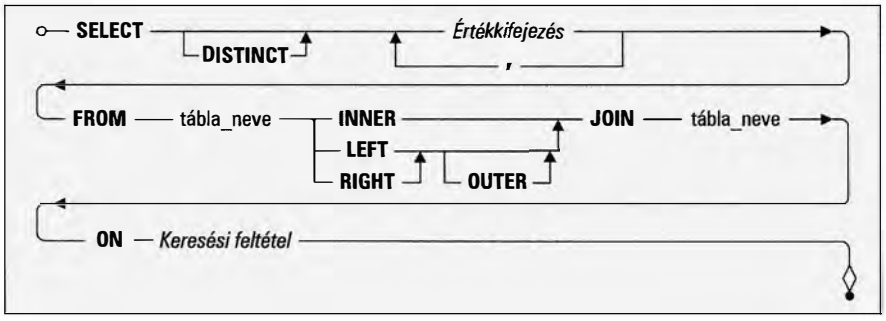


9.7. ábra

A receptekről elérhető összes információ kinyeréséhez szükséges táblák a Recipes mintaadatbázisban

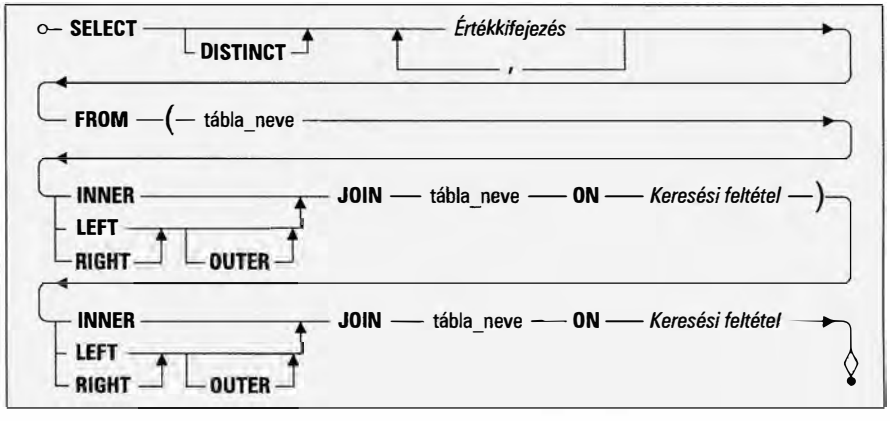
Úgy néz ki, hogy *öt* különböző tábla adataira van szükségünk! Ahogy azonban a 8. fejezetben láttuk, a feladat megoldható egy összetettebb – JOIN záradékokba ágyazott JOIN záradékokat tartalmazó – FROM záradék megalkotásával. Íme a trükk: mindenhol, ahol megadhatunk táblanevet, megadhatunk teljes, zárójelek közötti JOIN záradékot is. A 9.8. ábra két tábla összekapcsolásának egyszerűsített változatát mutatja be (elhagyva a korrelációsnév-záradékokat és az ON záradékot választva a két tábla INNER vagy OUTER JOIN-nal történő összekapcsolásához).

A harmadik táblát úgy tudjuk az eddigiekhez kapcsolni, hogy az első tábla neve előtt zárójelet nyitunk, amit a keresési feltételt követően bezárunk, utána következhet a második JOIN kulcsszó, a harmadik tábla neve, az ON kulcsszó és a második keresési feltétel. A teendőinket a 9.9. ábra mutatja be.



9.8. ábra

Két tábla egyszerű összekapcsolása a JOIN utasítással



9.9. ábra

Három tábla egyszerű összekapcsolása a JOIN utasítással

Ha belegondolunk, a zárójelen belüli, JOIN utasítással összekapcsolt táblák egy logikai táblát, ha úgy tetszik, belső eredményhalmazt adnak. Ez az eredményhalmaz veszi át a 9.8. ábra első, egyszerű táblájának a helyét. A műveletet folytathatjuk: az egész eddigi JOIN záradékot zárójelek közé tehetjük, újabb JOIN kulcsszót írhatunk az utasításhoz, újabb táblanévvel, amelyet újabb ON kulcsszó követ, újabb keresési feltétellel, egészen addig, amíg előáll a kívánt eredményhalmaz. Írjunk olyan lekérdezést, amely a 9.7. ábrán látható összes táblából használ adatokat, és lássuk mi történik! (Ezt a lekérdezéstípust egy olyan jelentéshez használhatjuk, amely az összes receptosztály összes receptjének adatait kiírja.)

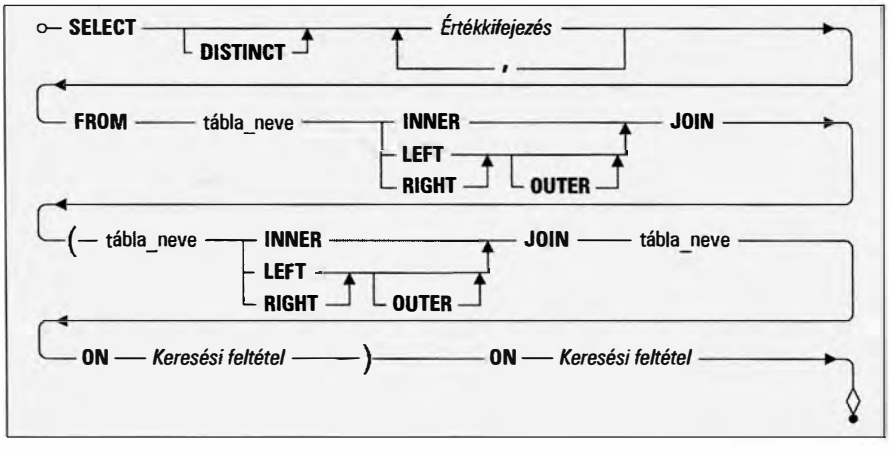
> *„Mutasd meg a recepteket tartalmazó adatbázis minden recepttípusát, a bennük lévő receptek címét, az elkészítés menetét, a hozzávalók nevét, a hozzávalók sorrendjét, a hozzávalók mennyiségét és a hozzávalók megadásánál használt mértékegységeket, a receptek címe és sorszáma szerint rendezve!"*

Fordítás Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matching recipe ID in the recipes table, then joined with the recipe ingredients table on recipe ID in the recipes table matching recipe ID in the recipe ingredients table, then joined with the ingredients table on ingredient ID in the ingredients table matching ingredient ID in the recipe ingredients table, and then finally joined with the measurements table on measurement amount ID in the measurements table matching measurement amount ID in the recipe ingredients table,order by recipe title and recipe sequence number
(Válaszd ki a receptosztályok leírását, a receptek címét, az elkészítési utasításokat, a hozzávalók nevét, a receptek sorszámát, a mennyiségeket és a mértékegységeket a receptosztályok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a receptek táblájával, a receptosztályok táblájának receptosztály-azonosítóit megfeleltetve a receptek táblájában levő receptazonosítóknak, majd ezt összekapcsoltuk a recepthozzávalók táblájával a recepttábla receptazonosítóit megfeleltetve a recepthozzávalók táblájának receptazonosítóinak, majd ezt összekapcsoltuk a hozzávalók táblájával a hozzávalók táblájának hozzávaló-azonosítóit megfeleltetve a recepthozzávalók táblájának hozzávaló-azonosítóinak, és végül ezt összekapcsoltuk a mértékegységek táblájával a tábla mennyiségazonosítóit megfeleltetve a recepthozzávalók táblájában levő mennyiségazonosítóknak, receptcím és receptsorszám szerint rendezve.)

Tisztázás Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
left outer joined ~~with the~~ recipes ~~table~~
on recipe_classes .recipe class ID ~~in the recipe classes table matching~~ = recipes .recipe class ID ~~in the recipes table, then~~ inner joined ~~with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,then~~
inner joined ~~with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matching~~
= recipe_ingredients.ingredient ID
~~in the recipe ingredients table, and then~~
~~finally~~ inner joined ~~with the~~ measurements ~~table~~
on measurements.measurement amount ID ~~in the~~

measurements table matching
= recipe_ingredients.measurement amount ID
in the recipe ingredients table,
order by recipe title, and recipe sequence number

SQL
```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```



9.10. ábra
Kettőnél több tábla egyesítése más sorrendben

Valójában bárhová, ahová egyébként egy táblanevet tennénk, tehetjük két tábla JOIN művelettel képzett összekapcsolását is. A 9.9. ábra azt sugallja, hogy először az első tábla egyesítendő a másodikkal, majd a kapott eredmény a harmadikkal. Ugyanakkor az is járható út, ha először a második és a harmadik táblát kapcsoljuk össze (már amennyiben a harmadik tábla a másodikkal áll logikai kapcsolatban, nem pedig az elsővel), és csak ez után hajtjuk végre az utolsó összekapcsolást az első táblával. A 9.10. ábra ez utóbbi módszert mutatja be.

Az előzőekben vizsgált, öt táblát használó lekérdezés SQL kódját a következőképpen is felírhattuk volna:

```
SQL       SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM Recipe_Classes
LEFT OUTER JOIN
   (((Recipes
   INNER JOIN Recipe_Ingredients
   ON Recipes.RecipeID = Recipe_Ingredients.RecipeID)
   INNER JOIN Ingredients
   ON Ingredients.IngredientID =
      Recipe_Ingredients.IngredientID)
   INNER JOIN Measurements
   ON Measurements.MeasureAmountID =
      Recipe_Ingredients.MeasureAmountID)
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
ORDER BY RecipeTitle, RecipeSeqNo
```

Ne feledjük, hogy néhány adatbázisrendszer optimalizálója érzékeny a JOIN utasítások sorrendjére. Ha egy sok JOIN műveletet tartalmazó lekérdezés futtatása egy nagy adatbázison hosszú időt vesz igénybe, az SQL kódban szereplő JOIN utasítások sorrendjének megváltoztatása a futás felgyorsulását eredményezheti.

Észrevehettük, hogy mindössze egyetlen OUTER JOIN utasítást használtunk az előző, több JOIN utasítást tartalmazó példákban. Lehet, hogy azon törjük a fejünket, hogy lehetséges és értelmes-e több OUTER JOIN utasítást használni egy összetett JOIN kifejezésben. Tegyük fel, hogy azon túl, hogy vannak recepteket még nem tartalmazó receptosztályaink, rendelkezünk pár olyan recepttel is, amelyekhez eleddig nem adtunk meg hozzávalókat. Az előző példában nem találkozunk a Recipes tábla egyetlen olyan sorával sem, amelynek ne lenne a Recipe_Ingredients táblában megfelelő sora. Ennek az az oka, hogy az INNER JOIN utasítás az ilyeneket elhagyja. Akkor hát jelenítsük meg most az összes receptet!

„Sorold fel a recepteket tartalmazó adatbázis minden recepttípusát, a bennük lévő receptek címét, az elkészítés menetét, a hozzávalók nevét, a hozzávalók sorrendjét, a hozzávalók mennyiségét és a hozzávalók megadásánál használt mértékegységeket, a receptek címe és sorszáma szerint rendezve!"

Fordítás Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matching recipe class ID in the recipes table, then left outer joined with the recipe ingredients table on recipe ID in the recipes table matching recipe ID in the recipe ingredients table, then joined with the ingredients table on ingredient ID in the ingredients table matching ingredient ID in the recipe ingredients table, and then finally joined with the measurements table on measurement amount ID in the measurements table matching measurement amount ID in the recipe ingredients table, order by recipe title and recipe sequence number

(Válaszd ki a receptosztályok leírását, a receptek címét, az elkészítési utasításokat, a hozzávalók nevét, a receptek sorszámát, a mennyiségeket és a mértékegységeket a receptosztályok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a receptek táblájával, a receptosztályok receptosztály-azonosítója és a receptek tábla receptosztály-azonosítója szerint, majd ezt szintén LEFT OUTER JOIN-nal összekapcsoltuk a recepthozzávalók táblájával, a receptek táblájában levő receptazonosítók és a recepthozzávalók táblájában levő receptazonosítók megfeleltetésével, majd ezt összekapcsoltuk a hozzávalók táblájával, a hozzávalók táblájának hozzávaló-azonosítóit megfeleltetve a recepthozzávalók táblájában levő hozzávaló-azonosítóknak, és végül ezt összekapcsoltuk a mértékegységek táblájával, a mértékegységek táblájának mennyiségazonosítói és a recepthozzávalók táblájának mennyiségazonosítói szerint, receptcím és receptsorszám szerint rendezve.)

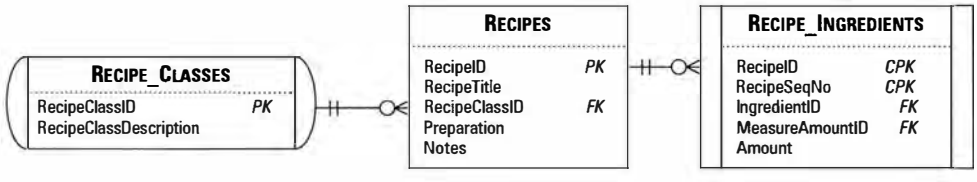
Tisztázás Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
left outer join~~ed with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipe classes table matching~~ = recipes.recipe class ID ~~in the recipes table, then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,then~~
inner join~~ed with the~~ ingredients ~~table~~

on ingredients.ingredient ID ~~in the ingredients table matching~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table,~~
~~and then finally~~ inner joined ~~with the~~ measurements ~~table~~
on measurement.measurement amount ID
~~in the measurements table matching~~
= recipe_ingredients.measurement amount ID
~~in the recipe ingredients table,~~
order by recipe title ~~and~~ recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
LEFT OUTER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```

Legyünk óvatosak! Az efféle, több OUTER JOIN utasítást tartalmazó kód csak azokban az esetekben működik helyesen, ha követjük az egy-a-többhöz kapcsolatok által kijelölt utat. Vessünk megint egy pillantást a 9.11. ábrán látható, Recipe_Classes, Recipes és a Recipe_Ingredients táblák közötti kapcsolatra:



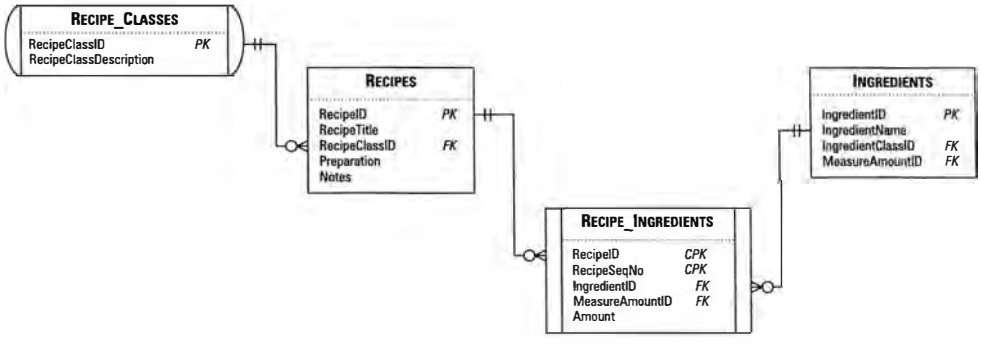
9.11. ábra

A Recipe_Classes, Recipes és a Recipe_Ingredients táblák közötti kapcsolat

Az egy-a-többhöz kapcsolatot néha *szülő-gyermek kapcsolatnak* nevezik. Minden szülősornak (ezek vannak a kapcsolat „egy" oldalán) nulla vagy ennél több gyermeksora (ezek vannak a kapcsolat „több" oldalán) lehet. Ha nincsenek árva sorok a „több" oldalon (például a Recipes táblának lehet olyan sora, ahol a RecipeClassID oszlop Null értékkel bír), akkor a gyermektábla *minden* sorának rendelkeznie kell megfelelő sorral a szülőtáblában. Így hát van értelme a `Recipe_Classes LEFT JOIN Recipes` kifejezésnek, amellyel kigyűjthetők a Recipe_Classes táblának a Recipes táblában gyermeksorral még nem rendelkező sorai. A `Recipe_Classes RIGHT JOIN Recipes` kifejezésnek pedig (árva sorok hiánya esetén) az INNER JOIN utasítással megegyező eredményt kell adnia.

Ehhez hasonlóan a `Recipes LEFT JOIN Recipe_Ingredients` kifejezés is értelmes, hiszen lehetnek olyan receptjeink, amelyeknek eleddig nem jegyeztük fel a hozzávalóit. A `Recipes RIGHT JOIN Recipe_Ingredients` kifejezés ellenben nem működik, mert a Recipe_Ingredients táblában lévő kapcsolóoszlop (RecipeID) a tábla összetett elsődleges kulcsának része. Így biztosra vehető, hogy a Recipe_Ingredients táblában nincsenek árva sorok, lévén az elsődleges kulcsot alkotó táblák egyike sem tartalmazhat Null értéket.

Lépjünk hát egy lépéssel tovább, a mindezidáig egyetlen receptben sem szereplő hozzávalók (ingredients) lekérdezéséhez. Először is nézzük meg figyelmesen a 9.12. ábrát, amelyen a táblák, köztük az Ingredients tábla kapcsolata látható.



9.12. ábra

A Recipe_Classes, Recipes, Recipe_Ingredients és az Ingredients táblák közötti kapcsolat

Próbáljuk ki a következő lekérdezést (vigyázat: buktató van a dologban!):

> *„Mutasd meg az adatbázisban tárolt összes recepttípust, valamint az összes recept címét, az elkészítés menetét, a hozzájuk tartozó hozzávaló-sorszámokat, a hozzávalók mennyiségét, a hozzávalók mértékegységeit és végül az összes hozzávalónevet, a receptek címe és sorszáma szerint rendezve!"*
>
> Fordítás Select the recipe class description, recipe title,preparation instructions,ingredient name,recipe sequence number,

amount,and measurement description from the recipe classes
table left outer joined with the recipes table on recipe class ID
in the recipe classes table matches class ID in the recipes table,
then left outer joined with the recipe ingredients table on recipe ID
in the recipes table matches recipe ID in the recipe ingredients table,
then joined with the measurements table on measurement amount ID
in the measurements table matches measurement amount ID
in the measurements table,and then finally right outer joined
with the ingredients table on ingredient ID in the ingredients table
matches ingredient ID in the recipe ingredients table,
order by recipe title and recipe sequence number
(Válaszd ki a receptosztályok leírását, a receptek címét, az elkészítési utasításokat, a hozzávalók nevét, a receptek sorszámát, a mennyiségeket és a mértékegységeket a receptosztályok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a receptek táblájával, a receptosztályok táblájának receptosztály-azonosítóit megfeleltetve a recepttábla osztályazonosítóinak, majd ezt szintén LEFT OUTER JOIN-nal összekapcsoltuk a recepthozzávalók táblájával, a receptek táblájában levő receptazonosítók és a recepthozzávalók táblájának receptazonosítói szerint, majd ezt összekapcsoltuk a mértékegységek táblájával, a mértékegységtábla mértékegység-azonosítóit megfeleltetve a recepthozzávalók táblájában levő mértékegység-azonosítóknak, és végül ezt RIGHT OUTER JOIN-nal összekapcsoltuk az összetevők táblájával, az összetevők táblájának hozzávaló-azonosítóit megfeleltetve a recepthozzávalók táblájában levő hozzávaló-azonosítóknak, receptcím és receptsorszám szerint rendezve.)

Tisztázás

Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number,
amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~ left outer joined ~~with the~~ recipes ~~table~~
on recipe_classes .recipe class ID ~~in the recipe classes table matches~~
= recipes .class ID ~~in the recipes table,~~
~~then~~ left outer joined ~~with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
~~then~~ inner joined ~~with the~~ measurements ~~table~~
on measurements.measurement amount ID
~~in the measurements table matches~~
= measurements.measurement amount ID
~~in the measurements table,~~
~~and then finally~~ right outer joined ~~with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table,~~
order by recipe title, ~~and~~ recipe sequence number

SQL
```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
LEFT OUTER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID)
RIGHT OUTER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
ORDER BY RecipeTitle, RecipeSeqNo
```

Úgy véljük, hogy ez így működni fog? Hát, a válasz egy messze visszhangzó NEM! A legtöbb adatbázisrendszer a teljes FROM záradék elemzése után próbálja megállapítani a táblák kapcsolatainak kiépítéséhez vezető leghatásosabb módszert. Mindenesetre most tételezzük fel, hogy az adatbázisrendszer úgy dönt, hogy a JOIN utasításaink zárójelbe foglalásának mikéntjét tökéletesen követni fogja. Ez annyit tesz, hogy az adatbázisrendszer a legbelső JOIN utasítással (a Recipe_Classes tábla összekapcsolása a Recipes táblával) kezdi a munkát, és innen halad kifelé.

A Recipe_Classes táblának van néhány olyan sora, amelyeknek nincs megfelelője a Recipes táblában, vagyis az első JOIN utasítás olyan sorokat is vissza fog adni, ahol a RecipeClassID oszlop Null értékű. Ha ismét a 9.12. ábrára pillantunk, láthatjuk, hogy a RecipeClasses és a Recipes táblák között egy-a-többhöz kapcsolat áll fenn, ezért az összes receptet visszakapjuk, hacsak nincsenek receptosztályhoz nem rendelt receptjeink. A következő összekapcsolás a Recipe_Ingredients táblával úgyszintén LEFT OUTER JOIN művelettel valósul meg. Látni akarunk a Null értékekre való tekintet nélkül minden sort az előző (a Recipe_Classes és a Recipes tábla közötti) összekapcsolásból, illetve a nekik megfelelő sorokat a Recipe_Ingredients táblából. Az előző esethez hasonlóan, minthogy a Recipe_Classes tábla pár sorának nincs megfelelője a Recipes táblában, és a Recipes tábla pár sorának nincs megfelelője a Recipe_Ingredients táblában, jónéhány sor kaphat Null értéket a Recipe_Ingredients táblából származó IngredientID oszlopban. Annyit tettünk, hogy bejártuk a Recipe_Classes és Recipes táblák, illetve a Recipes és Recipe_Ingredients táblák közötti egy-a-többhöz kapcsolatokat. Eddig minden rendben volt. (Apropó, az utolsó, a Measurements táblával végzett INNER JOIN művelet okafogyott: tudjuk, hogy minden hozzávalóhoz tartozik MeasurementAmountID érték.)

Most jön a probléma. Az utolsó RIGHT OUTER JOIN utasítás veszi az Ingredients tábla összes sorát, és az *összes megfelelő* sort az előző JOIN műveletek eredményéből. Az 5. fejezetből még emlékszünk, hogy a Null különleges érték – nem egyenlő egyetlen másik értékkel, még egy másik Null értékkel sem. Null-tól különböző érték szerepel az Ingredients tábla *összes* sorának IngredientID oszlopában. Egyetlen olyan sor sem fog illeszkedni rájuk az előző JOIN műveletek eredményéből, amelynek IngredientID oszlopában Null érték van, így ezeket az utolsó JOIN művelet el fogja hagyni! Megjelenik az összes, egyetlen receptben sem használt hozzávaló, de nem fogjuk látni a recept nélküli receptosztályokat, sem a hozzávalók nélküli recepteket.

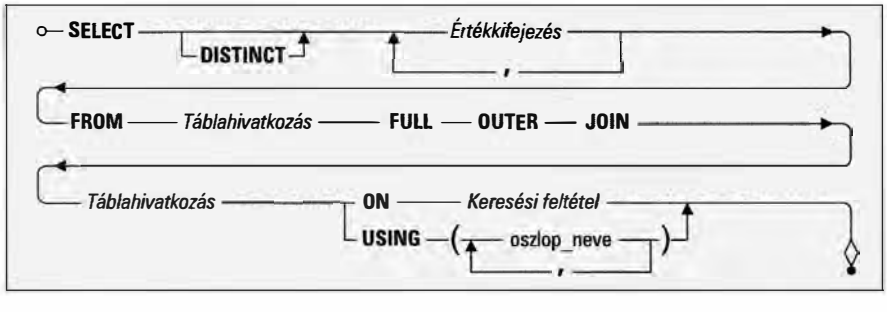
Ha az adatbázisrendszerünk a JOIN műveletek eltérő sorrendben történő elvégzése mellett dönt, a Null érték illeszthetetlensége miatt megjelenhetnek recept nélküli receptosztályok, hozzávaló nélküli receptek, ellenben nem fogjuk megtalálni a mindezidáig recept nélkül maradt hozzávalókat. Néhány adatbázisrendszer felismerheti ezt a logikai problémát, és a lekérdezés lefuttatását elvetheti valamiféle „félreérthető OUTER JOIN műveletek" hibaüzenettel. A baj, amivel szembesülünk, abból ered, hogy visszafelé próbáltunk haladni az egy-a-többhöz kapcsolat mentén. A hegyről lefelé egyszerű az út, felfelé azonban esetleg különleges eszközök kellenek. Mi hát a probléma megoldása? Olvassunk tovább, és kiderül!

A FULL OUTER JOIN

A FULL (teljes) OUTER JOIN se nem LEFT (bal), se nem RIGHT (jobb) – hanem mindkettő. Megjeleníti a JOIN műveletben lévő mindkét tábla vagy eredményhalmaz összes sorát. Ha a JOIN művelet bal oldalán lévő soroknak nincs megfelelőjük, a jobb oldali eredményhalmazból Null érték kerül melléjük. Hasonlóan, ha a JOIN művelet jobb oldalán lévő soroknak nincs megfelelőjük, Null értékek kerülnek melléjük a bal oldali eredményhalmazból.

Utasításforma

Most, hogy már JOIN műveletekkel dolgozunk egy ideje, a FULL OUTER JOIN utasításformája magától értetődő lesz. A 9.13. ábrán áttanulmányozhatjuk a FULL OUTER JOIN szintaxisdiagramját.



9.13. ábra

A FULL UTER JOIN utasítás szintaxisdiagramja

A dolgok egyszerűsítése végett a táblanév, a SELECT utasítás, illetve egy másik JOIN művelet eredményhalmazának helyén a *táblahivatkozás* kifejezést használjuk. Vessünk egy újabb pillantást az előző rész végén felvetett problémára, amelyet a FULL OUTER JOIN-nal már helyesen tudunk megoldani:

„Mutasd meg az adatbázisban tárolt összes recepttípust, valamint az összes recept címét, az elkészítés menetét, a hozzájuk tartozó hozzávaló-sorszámokat, a hozzávalók mennyiségét, a hozzávalók mértékegységeit és végül az összes hozzávalónevet, a receptek címe és sorszáma szerint rendezve!"

Fordítás Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table full outer joined with the recipes table on recipe class ID in the recipe classes table matches recipe class ID in the recipes table, then left outer joined with the recipe ingredients table on recipe ID in the recipes table matches recipe ID in the recipe ingredients table, then joined with the measurements table on measurement amount ID in the measurements table matches measurement amount ID in the recipe ingredients table, and then finally full outer joined with the ingredients table on ingredient ID in the ingredients table matches ingredient ID in the recipe ingredients table, order by recipe title and recipe sequence number
(Válaszd ki a receptosztályok leírását, a receptek címét, az elkészítési utasításokat, a hozzávalók nevét, a receptek sorszámát, a mennyiségeket és a mértékegységeket a receptosztályok táblájából, amelyet FULL OUTER JOIN-nal összekapcsoltunk a receptek táblájával, a receptosztályok táblájának receptosztály-azonosítóit megfeleltetve a recepttábla receptosztály-azonosítóinak, majd ezt LEFT OUTER JOIN-nal összekapcsoltuk a recepthozzávalók táblájával, a mértékegységek táblájának mértékegység-azonosítóit megfeleltetve a recepthozzávalók táblájában levő mértékegység-azonosítóknak, végül ezt FULL OUTER JOIN-nal összekapcsoltunk a hozzávalók táblájával, a hozzávalók táblájának hozzávaló-azonosítóit megfeleltetve a recepthozzávalók táblájában levő hozzávaló-azonosítóknak, a receptek címe és a receptek sorszáma szerint rendezve.)

Tisztázás Select ~~the~~ recipe class description, recipe title, preparation ~~instructions,~~ ingredient name, recipe sequence number, amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
full outer joined ~~with the~~ recipes ~~table~~
on recipe_classes .recipe class ID ~~in the recipe classes table matches~~
= recipes .recipe class ID ~~in the recipes table,~~

then left outer joined with the recipe ingredients table
on recipes.recipe ID in the recipes table matches
= recipe_ingredients.recipe ID in the recipe ingredients table,
then inner joined with the measurements table
on measurements.measurement amount ID
in the measurements table matches
= recipe_ingredients.measurement amount ID
in the recipe ingredients table,
and then finally full outer joined with the ingredients table
on ingredients.ingredient ID in the ingredients table matches
= recipe_ingredients .ingredient ID in the recipe ingredients table,
order by recipe title and recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
FULL OUTER JOIN Recipes
  ON Recipe_Classes.RecipeClassID =
     Recipes.RecipeClassID)
  LEFT OUTER JOIN Recipe_Ingredients
  ON Recipes.RecipeID =
     Recipe_Ingredients.RecipeID)
  INNER JOIN Measurements
  ON Measurements.MeasureAmountID =
     Recipe_Ingredients.MeasureAmountID)
  FULL OUTER JOIN Ingredients
  ON Ingredients.IngredientID =
     Recipe_Ingredients.IngredientID
ORDER BY RecipeTitle, RecipeSeqNo
```

Ezúttal az első és az utolsó JOIN utasítás veszi a JOIN utasítás mindkét oldalán lévő összes sort, így a Null értékek illeszthetetlenségéből származó probléma megoldódik. Most már nem csak a recept nélküli receptosztályok és a hozzávalók nélküli receptek látszanak, hanem az eleddig egyetlen receptben szereplő hozzávalók is. Lehet, hogy működik a lekérdezés akkor is, ha az első JOIN utasítás egy LEFT OUTER JOIN, de minthogy nem tudhatjuk előre, hogy az adatbázisrendszerünk milyen sorrendben dolgozza fel a JOIN utasításokat, a helyes választ biztosítandó használjunk FULL OUTER JOIN-t elöl is, hátul is.

Megjegyzés *Ahogy arra számíthattunk, az SQL-szabvány LEFT OUTER JOIN és RIGHT OUTER JOIN utasításait nem támogató adatbázisrendszerek a FULL OUTER JOIN utasítás helyett is eltérő utasításokat használnak. Az adatbázisunk által megkívánt, a szabványostól eltérő, az OUTER JOIN utasítást helyettesítő utasításformáról az adatbázisunk dokumentációjából tudhatunk meg bővebbet. Például a Microsoft SQL Server korai változatai az alábbi utasításformát fogadják el (figyeljünk a WHERE záradékban lévő csillagokra!):*

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID *=*
   Recipes.RecipeClassID
```

A FULL OUTER JOIN utasítást nem, de a LEFT és a RIGHT OUTER JOIN utasítást támogató termékek egyenértékű eredményhez a LEFT OUTER JOIN és a RIGHT OUTER JOIN műveleteken végzett UNION művelettel jutnak el. A UNION utasítást a következő fejezetben részletekbe menően tárgyaljuk. Minthogy a FULL OUTER JOIN utasításhoz megadott WHERE záradék pontos formája termékenként változó, többféle, a szabványtól eltérő rendszer használata esetén többféle nyelvtan megtanulására lehet szükség.

Nem kulcsmezőkön végzett FULL OUTER JOIN művelet

Mindezidáig az OUTER JOIN utasítások tárgyalása során csak olyan eseteket vizsgáltunk, ahol az OUTER JOIN-nal a táblákat vagy eredményhalmazokat egymással kapcsolatban lévő kulcsmezők alapján kapcsoltuk egymáshoz. Ugyanakkor néhány érdekes probléma megoldása során az OUTER JOIN műveletet nem kulcsmezőkön hajtjuk végre. Például az előző fejezetben láthattuk, hogy miként lehet a School Scheduling adatbázis azonos keresztnevű hallgatóit és tanárait megtalálni. Tételezzük fel, hogy ezúttal ki akarjuk íratni az összes hallgatót és az összes tanárt, megmutatva azokat, akiknek azonos a keresztnevük. Ez a feladat a FULL OUTER JOIN utasítással végezhető el.

„Mutasd meg az összes hallgatót, az összes tanárt, és írd ki azokat, akiknek azonos a keresztnevük!"

Fordítás Select student full name and staff full name from the students table full outer joined with the staff table on first name in the students table matches first name in the staff table
(Válaszd ki a hallgatók teljes nevét és a tanárok teljes nevét a hallgatók adatbázisából, amelyet FULL OUTER JOIN-nal összekapcsoltunk a tanárok táblájával, a hallgatók vezetéknevét megfeleltetve a tanárok vezetéknevének.)

Tisztázás Select student full name ~~and~~ staff full name from ~~the~~ students ~~table~~

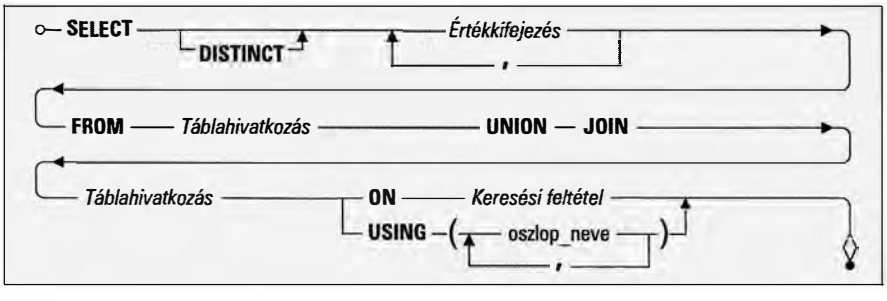
full outer joined with the staff table
on students.first name in the students table matches
= staff.first name in the staff table

SQL
```
SELECT (Students.StudFirstName || ' ' ||
   Students.StudLastName) AS StudFullName,
   (Staff.StfFirstName || ' ' ||
   Staff.StfLastName) AS StfFullName
FROM Students
FULL OUTER JOIN Staff
ON Students.StudFirstName =
   Staff.StfFirstName
```

A UNION JOIN

Az OUTER JOIN utasítások tárgyalása nem lenne teljes, ha a UNION JOIN utasítást legalább egyszer nem említenénk meg. Az SQL-szabvány szerint a UNION JOIN utasítás tulajdonképpen egy olyan FULL OUTER JOIN, amelynek az eredményéből a megfeleltetett sorokat elhagyjuk. Az utasításformát a 9.14. ábra mutatja.

Mint ahogy arra számítani lehet, nem sok kereskedelmi SQL-megvalósítás támogatja a UNION JOIN utasítást. Hogy őszinték legyünk, megoldhatatlan probléma volt számunkra a UNION JOIN utasítás használatára megfelelő indokot találni.



9.14. ábra

A UNION JOIN utasítás formája az SQL-ben

Az OUTER JOIN használatának területei

Minthogy az OUTER JOIN utasítások nem csak azokat a sorokat mutatják meg, amelyeknek megtalálják a megfelelőjét, hanem azokat is, amelyeknek nem, jól használhatóak azoknak a soroknak a fellelésére, amelyeknek nincs megfeleltethető párjuk a másik táblában – már ha egyáltalán van ilyen sor. Jók olyankor is, amikor a keresett soroknak van ugyan megfelelőjük néhány egyéb oszlopban, de nem mindben. Ezen felül hasznunkra vannak akkor, amikor olyan jelentéshez gyűjtünk anyagot, amelyben meg szeretnénk jeleníteni az összes termékcsoportot (arra való tekintet nélkül, hogy léteznek-e ezekhez kapcsolódó sorok

a többi táblában), vagy az összes vásárlót (arra való tekintet nélkül, hogy a vásárló valaha is adott-e fel rendelést). Következzék pár példa az OUTER JOIN utasítások valamelyikével megoldható esetekre.

Hiányzó értékek megtalálása

Előfordul, hogy arra vagyunk kíváncsiak, hogy mi hiányzik. Úgy találhatjuk meg a hiányzó adatokat, hogy egy OUTER JOIN utasítással a Null értékre keresünk. Íme néhány hiányzó értékekkel kapcsolatos gyakorlófeladat:

„Melyek azok a termékek, amelyekre soha nem érkezett rendelés?"
„Mutasd meg azokat a vásárlókat, akik sosem rendeltek bukósisakot!"
„Sorold fel azokat az előadókat, akiket még sehová sem hívtak szerepelni!"
„Jelenítsd meg azokat az ügynököket, akik még egy előadónknak sem szereztek munkát!"
„Mutasd meg azokat a tornákat, amelyen még nem játszottak mérkőzést!"
„Írd ki a kar azon tagjait, akik nem végeznek tanítási tevékenységet!"
„Jelenítsd meg azokat a hallgatókat, akik még sohasem adtak le órát!"
„Mutasd meg azokat az órákat, amelyeket senki nem vett fel!"
„Sorold fel az eddig egy receptben sem említett hozzávalókat!"
„Jelenítsd meg a hiányzó recepttípusokat!"

Részlegesen megfeleltetett információ megtalálása

Főleg jelentések készítésekor jöhet jól egy vagy több tábla összes sorának kiíratása a kapcsolódó táblák megfeleltetett soraival együtt. Íme néhány részlegesen megfeleltetett információval kapcsolatos, OUTER JOIN utasítással megoldható feladat:

„Sorolj fel minden rendelést, az időpontjával és a termékkel együtt!"
„Jelenítsd meg az összes vásárlót és minden kerékpárrendelést!"
„Mutasd meg az összes művészeti stílust a stílust kedvelő megrendelőkkel!
„Írd ki az összes előadót és minden lekötött fellépésüket!"
„Írd ki az összes játékost és azokat a meccseket, ahol 160-nál többet gurítottak!"
„Jelenítsd meg az összes tornát és minden lejátszott mérkőzést!"
„Mutasd meg az összes tantárgycsoportot és az összes tárgy minden óráját!"
„Sorold fel az összes hallgatót és minden órát, amit eddig felvettek!"
„Jelenítsd meg az összes kart és az általuk tartott órákat!"
„Sorold fel az összes recepttípust és az érintett hozzávalókat!"
„Mutasd meg az összes hozzávalót és minden receptet, amelyhez kellenek!"

Példák

Mostanra már ismerjük az OUTER JOIN utasítást tartalmazó lekérdezések megalkotásának mikéntjét, és láttunk néhány OUTER JOIN-nal megválaszolható kérdéstípust is. Tekintsünk át néhány jellemző példát, amelyek mindegyikében OUTER JOIN utasítást használunk. Ezekben a gyakorlatokban mintaadatbázisaink mindegyike előkerül, miközben hiányzó vagy részlegesen megfeleltetett értékeket igyekszünk fellelni.

Az SQL kód után rögtön bemutatjuk a kód által visszaadott eredményhalmaz-mintát. Az a név, amely megelőzi az eredményhalmazt, megegyezik a CD-mellékleten lévő mintaadatokon lefuttatható lekérdezésével. Minden lekérdezést a hozzá tartozó mintaadatbázisban tároltunk (ahogy a példában jeleztük), és az ehhez a fejezethez tartozó lekérdezések neve elé CH09-et írtunk. A könyv bevezetőjében található utasításokat követve betölthetjük és kipróbálhatjuk a mintákat.

Megjegyzés *Lévén, hogy a következő példák közül sok összetett JOIN utasításokkal dolgozik, az adatbázisrendszerünk optimalizálója különbözőképpen dolgozhatja fel a lekérdezéseket. Ebből kifolyólag az eredmények első néhány sora eltérhet a megadottól, ugyanakkor a sorok számának meg kell egyeznie vele. A szokásos „Fordítás" és „Tisztázás" részeket az egyszerűséget szem előtt tartva a következő példák mindegyikénél egyesítettük.*

Sales Orders adatbázis

„Melyek azok a termékek, amelyekre soha nem érkezett rendelés?"

Fordítás/ Tisztázás
Select product number ~~and~~ product name from ~~the~~ products table left outer joined with the order details table on products.product number ~~in the products table matches~~ = order_details .product number ~~in the order details table~~ where ~~the~~ order detail order number is null

SQL

```
SELECT Products.ProductNumber,
   Products.ProductName
FROM Products LEFT OUTER JOIN Order_Details
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Order_Details.OrderNumber IS NULL
```

CH09_Products_Never_Ordered (2 sor)

ProductNumber	ProductName
4	Victoria Pro All Weather Tires
23	Ultra-Pro Rain Jacket

„Jelenítsd meg az összes vásárlót és minden kerékpárrendelést!"

Fordítás 1. Select customer full name, order date, product name, quantity ordered, and quoted price from the customers table left outer joined with the orders table on customer ID, then joined with the order details table on order number, then joined with the products table on product number, then finally joined with the categories table on category ID where category description is 'Bikes'

(Válaszd ki azoknak a vásárlóknak a teljes nevét, a rendelésük dátumát, a termék nevét, a megrendelt mennyiséget és az árat a vásárlók táblájából – amelyet LEFT OUTER JOIN-nal összekapcsoltunk a rendelések táblájával a vásárlóazonosító alapján, majd ezt összekapcsoltuk a rendelések részleteit tartalmazó táblával a rendelésszám alapján, majd ezt összekapcsoltuk a termékek táblájával a termékszám alapján, és végül ezt összekapcsoltuk a termékcsoportok táblájával a termékcsoport-azonosító alapján –, akiknél a termékcsoport leírása „kerékpárok".)

Fordítás 2/ Tisztázás Select customer full name, order date, product name, quantity ordered, ~~and~~ quoted price
from ~~the~~ customers ~~table~~ left outer joined ~~with~~
(Select customer ID, order date, product name,
quantity ordered, ~~and~~ quoted price
from ~~the~~ orders ~~table~~
inner joined with the order details table
on orders .order number ~~in the orders table matches~~
= order_details .order number in the order details table,
~~then~~ joined ~~with the~~ products ~~table~~
on order_details.product number ~~in the order details table matches~~ = products.product number ~~in the products table, then finally~~ joined ~~with the~~ categories ~~table~~
on categories.category ID ~~in the categories table matches~~
= products.category ID ~~in the products table~~
where category description ~~is~~ = 'Bikes') as rd
on customers.customer ID in the customers table matches
= rd.customerID ~~in the embedded SELECT statement~~

Megjegyzés *Minthogy csak bizonyos rendelésekre (a kerékpárokra) vagyunk kíváncsiak, a fordítási folyamatot két részre szedtük, így kiemelve azt, hogy a rendeléseket az OUTER JOIN művelet alkalmazása előtt kell szűrnünk.*

SQL
```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   RD.OrderDate, RD.ProductName,
   RD.QuantityOrdered, RD.QuotedPrice
```

```
FROM Customers
LEFT OUTER JOIN
  (SELECT Orders.CustomerID, Orders.OrderDate,
      Products.ProductName,
      Order_Details.QuantityOrdered,
      Order_Details.QuotedPrice
    FROM ((Orders
    INNER JOIN Order_Details
    ON Orders.OrderNumber =
       Order_Details.OrderNumber)
    INNER JOIN Products
    ON Order_Details.ProductNumber =
       Products.ProductNumber)
    INNER JOIN Categories
    ON Categories.CategoryID =
       Products.CategoryID
    WHERE Categories.CategoryDescription =
       'Bikes')
  AS RD
ON Customers.CustomerID = RD.CustomerID
```

CH09_All_Customers_And_Any_Bike_Orders (913 sor)

CustFullName	OrderDate	ProductName	QuantityOrdered	QuotedPrice
Suzanne Viescas				
William Thompson	2007-12-23	Trek 9000 Mountain Bike	5	$1,164.00
William Thompson	2008-01-15	Trek 9000 Mountain Bike	6	$1,164.00
William Thompson	2007-10-11	Viscount Mountain Bike	2	$635.00
William Thompson	2007-10-05	Viscount Mountain Bike	5	$615.95
William Thompson	2008-01-15	Trek 9000 Mountain Bike	4	$1,200.00
William Thompson	2007-10-11	Trek 9000 Mountain Bike	3	$1,200.00
William Thompson	2008-01-07	Trek 9000 Mountain Bike	2	$1,200.00
<< további sorok >>				

Megjegyzés *Ez a lekérdezés meglehetősen cseles, hiszen az összes vásárlóra szeretnénk az OUTER JOIN műveletet alkalmazni a kerékpárrendelések tekintetében. Ha a „Fordítás 1”-ből írnánk SQL kódot, akkor nem találnánk meg azokat a vásárlókat, akik nem rendeltek kerékpárt. A Customers és az Orders táblákon elvégzett OUTER JOIN visszaadja az összes vásárlót és minden rendelést. Amikor azonban kibővítjük a lekérdezést a kerékpárrendelésekre vonatkozó szűrőfeltétellel, már csak a kerékpárt rendelő vásárlókat fogjuk megkapni.*

A „Fordítás 2" bemutatja a helyes módszert: létrehozunk egy belső eredményhalmazt – ami csak a kerékpárrendeléseket tartalmazza –, majd ezt egyesítjük egy OUTER JOIN utasítást használva a Customers táblával, hogy kinyerjük a végső eredményt.

(Úgy látszik, hogy William Thompson igazán remek ügyfél!)

Entertainment Agency adatbázis

„Sorold fel azokat az előadókat, akiket még sehová sem hívtak szerepelni!"

Fordítás/ Tisztázás
Select entertainer ID ~~and~~ entertainer stage name
from ~~the~~ entertainers ~~table~~
left outer join~~ed with the~~ engagements ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= engagements.entertainer ID ~~in the engagements table~~
where engagement number is null

SQL

```
SELECT Entertainers.EntertainerID,
   Entertainers.EntStageName
FROM Entertainers
LEFT OUTER JOIN Engagements
ON Entertainers.EntertainerID =
   Engagements.EntertainerID
WHERE Engagements.EngagementNumber IS NULL
```

CH09_Entertainers_Never_Booked (1 sor)

ProductNumber	ProductName
EntertainerID	Katherine Ehrlich

„Mutasd meg az összes művészeti stílust a stílust kedvelő megrendelőkkel!

Fordítás/ Tisztázás
Select style ID, style name, customer ID, customer first name,
~~and~~ customer last name
from ~~the~~ musical styles ~~table~~
left outer join~~ed with~~
(~~the~~ musical preferences ~~table~~ inner join~~ed~~
~~with the~~ customers ~~table~~
on musical_preferences.customer ID
~~in the musical preferences table matches~~
= customers.customer ID ~~in the customers table~~)
on musical_styles.style ID ~~in the musical styles table matches~~
= musical_preferences.style ID ~~in the musical preferences table~~

SQL

```
SELECT Musical_Styles.StyleID,
   Musical_Styles.StyleName,
   Customers.CustomerID,
   Customers.CustFirstName,
```

```
   Customers.CustLastName
FROM Musical_Styles
LEFT OUTER JOIN (Musical_Preferences
   INNER JOIN Customers
   ON Musical_Preferences.CustomerID =
      Customers.CustomerID)
ON Musical_Styles.StyleID =
   Musical_Preferences.StyleID
```

CH09_All_Styles_And_Any_Customers (41 sor)

StyleID	StyleName	CustomerID	CustFirstName	CustLastName
1	40s Ballroom Music	10015	Carol	Viescas
1	40s Ballroom Music	10011	Joyce	Bonnicksen
2	50s Music			
3	60s Music	10002	Deb	Waldal
4	70s Music	10007	Liz	Keyser
5	80s Music	10014	Mark	Rosales
6	Country	10009	Sarah	Thompson
7	Classical	10005	Elizabeth	Hallmark
<< további sorok >>				

(Ezek szerint senki nem szereti az ötvenes évek zenéjét...)

Megjegyzés *A FROM záradékot meglehetős odafigyeléssel fogalmaztuk meg. Igyekeztünk úgy befolyásolni az adatbázisrendszert, hogy először hajtsa végre a Musical_Preferences és a Customers táblák közötti INNER JOIN műveletet, és csak ezt követően jöjjön a kapott eredmény és a Musical_Styles tábla közötti OUTER JOIN. Ha az adatbázisrendszerünk hajlamos a JOIN utasításokat balról jobbra haladva feldolgozni, esetleg érdemes az INNER JOIN utasítást tartalmazó FROM záradékot elsőként a Musical_Styles táblával végzett RIGHT OUTER JOIN műveletnek követnie. A Microsoft Office Accessben a helyes választ csak akkor kapjuk meg, ha az INNER JOIN utasítást beágyazott SELECT-ként használjuk.*

School Scheduling adatbázis

„Írd ki a kar azon tagjait, akik nem végeznek tanítási tevékenységet!"

Fordítás/ Tisztázás

Select staff first name ~~and~~ staff last name
from ~~the~~ staff ~~table~~ left outer joined with the faculty classes ~~table~~
on staff.staff ID ~~in the staff table matches~~
= faculty_classes.staff ID ~~in the faculty classes table~~
where class ID is null

SQL
```
SELECT Staff.StfFirstName, Staff.StfLastName,
FROM Staff
LEFT OUTER JOIN Faculty_Classes
ON Staff.StaffID = Faculty_Classes.StaffID
WHERE Faculty_Classes.ClassID IS NULL
```

CH09_Staff_Not_Teaching (4 sor)

StfFirstName	StfLastName
Jeffrey	SmithEhrlich
Tim	Smith
Kathryn	Patterson
Joe	Rosales III

„Jelenítsd meg azokat a hallgatókat, akik még sohasem adtak le órát!"

Fordítás/ Tisztázás
Select student full name
from ~~the~~ students ~~table~~ left outer joined ~~with~~
(Select student ID from ~~the~~ student schedules ~~table~~
inner ~~joined with the~~ student class status ~~table~~
on student_class_status.class status
~~in the student class status table matches~~
= student_schedules.class status ~~in the student schedules table~~
where class status description ~~is~~ = 'withdrew') as withdrew
on students.student ID ~~in the students table matches~~
= withdrew.student ID ~~in the embedded SELECT statement~~
where ~~the~~ student_schedules.student ID ~~in the~~
~~student schedules table~~ is null

SQL
```
SELECT Students.StudLastName || ', ' ||
   Students.StudFirstName AS StudFullName
FROM Students
LEFT OUTER JOIN
   (SELECT Student_Schedules.StudentID
    FROM Student_Class_Status
    INNER JOIN Student_Schedules
    ON Student_Class_Status.ClassStatus =
       Student_Schedules.ClassStatus
    WHERE Student_Class_Status.ClassStatus
       Description = 'withdrew')
  AS Withdrew
ON Students.StudentID = Withdrew.StudentID
WHERE Withdrew.StudentID IS NULL
```

CH09_Students_Never_Withdrawn (15 sor)

StudFullName
Hamilton,David
Stadick, Betsy
Galvin, Janice
Hartwig, Doris
Bishop, Scott
Hallmark, Elizabeth
Sheskey, Sara
Wier, Marianne
<< további sorok >>

„Mutasd meg az összes tárgycsoportot és az összes tárgy minden óráját!"

Fordítás/ Tisztázás
Select category description, subject name, classroom ID,
start time, ~~and~~ duration
from ~~the~~ categories ~~table~~
left outer ~~joined with the~~ subjects ~~table~~
on categories.category ID ~~in the categories table matches~~
= subjects.category ID ~~in the subjects table,~~
~~then~~ left outer ~~joined with the~~ classes ~~table~~
on subjects .subject ID ~~in the subjects table matches~~
= classes.subject ID ~~in the classes table~~

SQL

```
SELECT Categories.CategoryDescription,
   Subjects.SubjectName, Classes.ClassroomID,
   Classes.StartTime, Classes.Duration
FROM (Categories
LEFT OUTER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
LEFT OUTER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID
```

Megjegyzés *Az egymásba ágyazások sorrendjének megállapításakor most is igen figyelmesen jártunk el, hogy a várt eredményt kapjuk.*

Az eredményhalmaz további részében látni fogjuk, hogy a *Developing a Feasibility Plan*, a *Computer Programming*, és az *American Government* tárgyakhoz egyelőre nem tartozik óra. Azt is megfigyelhetjük, hogy a *Psychology*, a *French* és a *German* tárgycsoportokban még nincsenek tárgyak.

CH09_All_Categories_All_Subjects_Any_Classes (82 sor)

CategoryDescription	SubjectName	ClassroomID	StartTime	Duration
Accounting	Financial Accounting Fundamentals I	3313	9:00	50
Accounting	Financial Accounting Fundamentals I	3313	13:00	50
Accounting	Financial Accounting Fundamentals II	3415	8:00	50
Accounting	Fundamentals of Managerial Accounting	3415	10:00	50
Accounting	Intermediate Accounting	3315	11:00	50
Accounting	Business Tax Accounting	3313	14:00	50
Art	Introduction to Art	1231	10:00	50
Art	Design	1619	15:30	110
<< *további sorok* >>				

Bowling League adatbázis

„Mutasd meg a le nem játszott tornákat!"

Fordítás/ Select tourney ID, tourney date, and tourney location
Tisztázás from ~~the~~ tournaments ~~table~~
left outer ~~joined with the~~ tourney matches ~~table~~
on tournaments.tourney ID ~~in the tournaments table matches~~
= tourney_matches.tourney ID ~~in the tourney matches table~~
where match ID is null

SQL
```
SELECT Tournaments.TourneyID,
   Tournaments.TourneyDate,
   Tournaments.TourneyLocation
FROM Tournaments
LEFT OUTER JOIN Tourney_Matches
ON Tournaments.TourneyID =
   Tourney_Matches.TourneyID
WHERE Tourney_Matches.MatchID IS NULL
```

CH09_Tourney_Not_Yet_Played (6 sor)

TourneyID	TourneyDate	TourneyLocation
15	2008-07-11	Red Rooster Lanes
16	2008-07-18	Thunderbird Lanes
17	2008-07-25	Bolero Lanes
18	2008-08-01	Sports World Lanes
19	2008-08-08	Imperial Lanes
20	2008-08-15	Totem Lanes

„Sorold fel az összes játékost, és azokat a játékokat, ahol 180 felett gurítottak!"

Fordítás 1. Select bowler name, tourney date, tourney location,match ID, and raw score from the bowlers table left outer joined with the bowler scores table on bowler ID, then inner joined with the tourney matches table on match ID, then finally inner joined with the tournaments table on tournament ID where raw score in the bowler scores table is greater than 180
(Válaszd ki a játékosok nevét, a torna dátumát, a torna helyét, a mérkőzésazonosítót és a tiszta pontszámot a játékosok táblájából, amelyet LEFT OUTER JOIN-nal összekapcsoltunk a játékosok eredményeit tartalmazó táblával a játékosazonosító alapján, majd ezt INNER JOIN-nal összekapcsoltuk a tornák mérkőzéseit tartalmazó táblával a mérkőzésazonosító alapján, végül ezt INNER JOIN-nal összekapcsoltuk a tornák táblájával a tornaazonosító alapján, ahol a játékosok eredményeit tároló táblában a tiszta pontszám 180-nál magasabb.)

Látjuk, hogy az előző fordítás miért nem fog működni? A LEFT JOIN jobb oldalán lévő táblák egyikén kell szűrést végezni, így aztán a szűrést egy beágyazott SELECT utasításban kell elhelyezni. Gondoljuk át újra a fordítást, aztán következhet a tisztázás és végül a probléma megoldása.

Fordítás 2/ Select bowler name, tourney date, tourney location,
Tisztázás match ID, ~~and~~ raw score
from ~~the~~ bowlers ~~table~~ left outer joined ~~with~~
(Select tourney date, tourney location, match ID,
bowler ID, ~~and~~ raw score
from ~~the~~ bowler scores ~~table~~
inner joined ~~with the~~ tourney matches ~~table~~
on bowler_scores .match ID ~~in the bowler scores table matches~~
= tourney_ matches.match ID ~~in the tourney matches table,~~
~~then~~ inner joined ~~with the~~ tournaments ~~table~~
on tournaments.tournament ID ~~in the tournaments table matches~~
= tourney_ matches.tournament ID ~~in the tourney matches table~~
where raw score ~~is greater than~~ > 180) as ti
on bowlers.bowler ID ~~in the bowlers table matches~~
= ti.bowler ID ~~in the embedded SELECT statement~~

SQL

```
SELECT Bowlers.BowlerLastName || ', ' ||
   Bowlers.BowlerFirstName AS BowlerName,
   TI.TourneyDate, TI.TourneyLocation,
   TI.MatchID, TI.RawScore
FROM Bowlers
LEFT OUTER JOIN
```

```
    (SELECT Tournaments.TourneyDate,
       Tournaments.TourneyLocation,
       Bowler_Scores.MatchID,
       Bowler_Scores.BowlerID,
       Bowler_Scores.RawScore
     FROM (Bowler_Scores
     INNER JOIN Tourney_Matches
     ON Bowler_Scores.MatchID =
       Tourney_Matches.MatchID)
     INNER JOIN Tournaments
     ON Tournaments.TourneyID =
       Tourney_Matches.TourneyID
     WHERE Bowler_Scores.RawScore > 180)
   AS TI
ON Bowlers.BowlerID = TI.BowlerID
```

CH09_All_Bowlers_And_Scores_Over_180 (106 sor)

BowlerName	TourneyDate	TourneyLocation	MatchID	RawScore
Black, Alastair				
Cunningham,David				
Ehrlich, Zachary				
Foumier, Barbara				
Fournier,David				
Hallmark, Alaina				
Hallmark, Bailey				
Hallmark, Elizabeth				
Hallmark, Gary				
Hernandez,Kendra				
Hernandez, Michael				
Kennedy,Angel	2007-11-20	Sports World Lanes	46	185
Kennedy,Angel	2007-10-09	Totem Lanes	22	182
<< további sorok >>				

Megjegyzés *Igen, ez ismét olyan példa, ahol először kell megalkotni a megszűrt, INNER JOIN műveletet kívánó eredményhalmazt, és csak ez után következhet az OUTER JOIN művelet, amely majd elvégzi az összekapcsolást azzal a táblával, amelyikből az „összes" sor kell.*

Recipes adatbázis

„Mutasd meg az eddig egy receptben sem említett hozzávalókat!"

Fordítás/ Tisztázás

Select ingredient name from ~~the~~ ingredients ~~table~~
left outer ~~joined with the~~ recipe ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table~~
where recipe ID is null

SQL

```
SELECT Ingredients.IngredientName
FROM Ingredients
LEFT OUTER JOIN Recipe_Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
WHERE Recipe_Ingredients.RecipeID IS NULL
```

CH09_Ingredients_Not_Used (20 sor)

IngredientName
Halibut
Chicken, Fryer
Bacon
Iceberg Lettuce
Butterhead Lettuce
Scallop
Vinegar
Red Wine
<< további sorok >>

„Sorold fel az adatbázisban tárolt összes recepttípust, valamint az összes recept címét, a hozzájuk tartozó hozzávaló-sorszámokat, a hozzávalók mennyiségét, a hozzávalók mértékegységeit és végül az összes hozzávalónevet!"

Fordítás/ Tisztázás

Select ~~the~~ recipe class description, recipe title,
ingredient name,recipe sequence number,
amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
full outer ~~joined with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipe classes table matches~~
= recipes.recipe class ID ~~in the recipes table,~~
~~then~~ left outer ~~joined with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
~~then~~ inner ~~joined with the~~ measurements ~~table~~

on measurements.measurement amount ID
in the measurements table matches
= recipe_ ingredients.measurement amount ID
in the recipe ingredients table,
and then finally full outer joined with the ingredients table
on ingredients.ingredient ID in the ingredients table matches
= recipe_ ingredients.ingredient ID in the recipe ingredients table,

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
FULL OUTER JOIN Recipes
  ON Recipe_Classes.RecipeClassID =
     Recipes.RecipeClassID)
  LEFT OUTER JOIN Recipe_Ingredients
  ON Recipes.RecipeID =
     Recipe_Ingredients.RecipeID)
  INNER JOIN Measurements
  ON Measurements.MeasureAmountID =
     Recipe_Ingredients.MeasureAmountID)
  FULL OUTER JOIN Ingredients
  ON Ingredients.IngredientID =
     Recipe_Ingredients.IngredientID
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
```

Megjegyzés *Ez volt az a feladat, amelyet a FULL OUTER JOIN utasításról szóló részben már megoldottunk. Azt a döntést, hogy itt is szerepeltetjük, az indokolja, hogy így láthatjuk az eredményt is. A lekérdezést nem találjuk meg a CD-n lévő mintaadatbázisok között, sem a Microsoft Access, sem a MySQL résznél, lévén egyik termék sem támogatja a FULL OUTER JOIN műveletet. A FULL OUTER JOIN-on alapuló megoldás helyett a CD-re a két OUTER JOIN műveleten, illetve ezek UNION utasítással történő összekapcsolásán alapuló, az előzővel azonos eredményt hozó megoldás került. A UNION használatáról a következő fejezetben fogunk tanulni. Az itt bemutatott eredmény a lekérdezés Microsoft SQL Serveren történő futtatásából származik.*

CH09_All_Recipe_Classes_All_Recipes (109 sor)

RecipeClassDescription	RecipeTitle	IngredientName	RecipeSeqNo	Amount	MeasurementDescription
Starch	Yorkshire Pudding	Flour	1	1.5	Cup
Starch	Yorkshire Pudding	Water	2	1	Cup
Starch	Yorkshire Pudding	Eggs	3	2	Piece
Starch	Yorkshire Pudding	Salt	4	0.5	Teaspoon
Starch	Yorkshire Pudding	Milk	5	0.5	Cup
Starch	Yorkshire Pudding	Beef drippings	6	4	Teaspoon
Dessert	Trifle	Sponge Cake	1	1	Package
Dessert	Trifle	Raspberry Jello	2	1	Package
Dessert	Trifle	Bird's Custard Powder	3	1	Package
Dessert	Trifle	Raspberry Jam	4	1	Jar
<< további sorok >>					

Összefoglalás

Ebben a fejezetben áttekintettük az OUTER JOIN utasítások világát. Kezdetnek megadtunk egy OUTER JOIN-t tartalmazó műveletet, majd összehasonlítottuk a 8. fejezetben megismert INNER JOIN-nal. Ezt követően megtanultuk, hogy miként lehet egyszerű, két táblát használó LEFT, illetve RIGHT OUTER JOIN utasítást tartalmazó lekérdezéseket és bonyolultabb, beágyazott SELECT utasításokat alkotni, hogy aztán több JOIN műveletet felhasználó kifejezésekkel folytathassuk. Megmutattuk, hogy a Null értékre történő kereséssel párosított OUTER JOIN azonos eredményt ad a 7. fejezetben megismert, különbségképzésre használatos EXCEPT művelettel. Megvizsgáltunk néhány olyan nehézséget, amellyel a több OUTER JOIN utasítást tartalmazó lekérdezések írása során találkozhatunk. A LEFT és a RIGHT OUTER JOIN utasítások megismerését egy olyan, több OUTER JOIN műveletet igénylő probléma felvetésével zártuk, ami nem oldható meg, ha csak a LEFT vagy a RIGHT OUTER JOIN-t használhatjuk.

A FULL OUTER JOIN utasítást tárgyalva megmutattuk, hogy ezt a összekapcsolási típust további INNER JOIN és OUTER JOIN utasításokkal kiegészítve miként kaphatunk helyes választ az előző problémára. Röviden kitértünk a FULL OUTER JOIN egy változatára, a UNION JOIN utasításra, és néhány, ilyen művelettel megválaszolható kérdés segítségével bemutattuk a felhasználási lehetőségeit. A fejezet fennmaradó részében közel egy tucat példát láthattunk az OUTER JOIN utasítás használatára. Adtunk pár példát minden mintaadatbázisunkkal kapcsolatban, hogy lássuk a feltett kérdések megválaszolására alkalmas kifejezések mögött megbúvó gondolatmenetet.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Mutasd meg azokat a vásárlókat, akik sohasem rendeltek bukósisakot!"*
   (Tipp: ez megint olyan kérdés, amelyhez előbb egy INNER JOIN művelet szükséges, amivel megtaláljuk az összes bukósisak-rendelést, majd OUTER JOIN műveletet hajtunk végre a vásárlókra vonatkozólag.)
   A megoldás itt található: CH09_Customers_No_Helmets (2 sor).
2. *„Jelenítsd meg azokat a vásárlókat, akikkel azonos irányítószám alatt nincs értékesítőnk!*
   A megoldás itt található: CH09_Customers_No_Rep_Same_Zip (18 sor).
3. *„Sorolj fel minden rendelést, a dátumával és a termékkel együtt!"*
   A megoldás itt található: CH09_All_Products_Any_Order_Dates (2682 sor).

Entertainment Agency adatbázis

1. *„Jelenítsd meg azokat az ügynököket, akik még egy előadónknak sem szereztek munkát!"*
   A megoldás itt található: CH09_ Agents_No_Contracts (1 sor).
2. *„Mutasd meg azokat a megrendelőket, akik még nem kötöttek le fellépést!"*
   A megoldás itt található: CH09_ Customers_No_Bookings (2 sor).
3. *„Sorold fel az összes előadót és minden lekötött fellépésüket!"*
   A megoldás itt található: CH09_ All_Entertainers_And_Any_Engagements (112 sor).

School Scheduling adatbázis

1. *„Mutasd meg azokat az órákat, amelyeket senki nem vett fel!"*
   (Tipp: csak azok a sorok kellenek hozzá a Student_Classes táblából, amelyekben „enrolled", vagyis „feliratkozott" szerepel, a „completed" és a „withdrew" – „elvégezte" vagy „leadta" – sorok érdektelenek.)
   A megoldás itt található: CH09_ Classes_No_Students_Enrolled (63 sor)
2. *„Jelenítsd meg azokat a tárgyakat, amelyekhez nem tartozik szak!"*
   A megoldás itt található: CH09_ Subjects_No_Faculty (1 sor)

3. *„Jelenítsd meg az összes olyan hallgatót, aki még nem vett fel egyetlen tárgyat sem!"* (Tipp: először azokat a hallgatókat kell megkeresnünk, akik feliratkoztak („enrolled") valamilyen tárgyra, majd azokat a hallgatókat, akik nincsenek benne ebben az eredményhalmazban.)
   A megoldás itt található: CH09_ Students_Not_Currently_Enrolled (2 sor)
4. *„Jelenítsd meg az összes kart, és az általuk kínált tárgyakat!*
   A megoldás itt található: CH09_All_Faculty_And_Any_Classes (79 sor)

Bowling League adatbázis

1. *„Jelenítsd meg azokat a mérkőzéseket, amelyekről még nincs adat!"*
   A megoldás itt található: CH09_ Matches_Not_Played_Yet (1 sor)
2. *„Jelenítsd meg a tornákat és a lejátszott mérkőzéseket!"*
   A megoldás itt található: CH09_ All_Tourneys_Match_Results (174 sor)

Recipes adatbázis

1. *„Jelenítsd meg a hiányzó recepttípusokat!"*
   A megoldás itt található: CH09_Recipe_Classes_No_Recipes (1 sor)
2. *„Mutasd meg az összes hozzávalót, és minden receptet, amelyikhez kellenek!"*
   A megoldás itt található: CH09_All_Ingredients_Any_Recipes (108 sor)
3. *„Írd ki a salad, a soup és a main courses kategóriákat és a bennük lévő recepteket!"*
   A megoldás itt található: CH09_Salad_Soup_Main_Courses (9 sor)
4. *„Jelenítsd meg az összes receptosztályt és a bennük lévő recepteket!"*
   A megoldás itt található: CH09_ All_RecipesClasses_And_Matching_Recipes (16 sor)

10

Unió

„Könyörgöm azoknak, akiknek a jámborsága lehetővé teszi, hogy tiltakozzanak, hogy imádkozzanak az unióért..."
– Sam Houston, Texas állam hőse

A fejezet témakörei

- Mi az a UNION?
- UNION utasítást tartalmazó lekérdezések
- A UNION utasítás használatának területei
- Példák
- Összefoglalás
- Önálló feladatok

A 7. fejezetben bevezettünk három alapvető halmazműveletet: a metszetképzést, a különbségképzést és az uniót. A 8. fejezet megmutatta, hogy a metszetképzéssel azonos értékű művelet úgy is végezhető, hogy a kulcsmezőkön alapuló INNER JOIN művelettel kapcsoljuk össze az eredményhalmazokat. A 9. fejezet a halmazok különbségének képzését tárgyalta az OUTER JOIN művelet, illetve a Null értékek megkeresésének segítségével. Ez a fejezet a harmadik műveletről, az unióról szól.

Mi az a UNION?

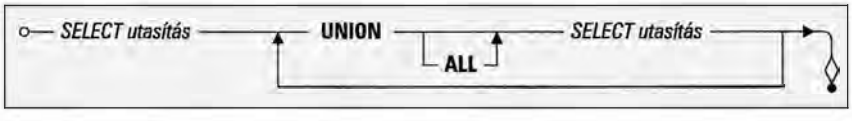
A UNION (unió) utasítás lehetővé teszi, hogy két vagy több hasonló eredményhalmazból kiválasztott sorokat egyetlen eredményhalmazzá egyesítsünk. Hangsúlyozzuk, hogy nem „oszlopokról", hanem „sorokról" van szó. A 8. és a 9. fejezetben megtanultuk két vagy több eredményhalmaz oszlopainak JOIN művelettel történő összeboronálását. A JOIN utasítás az eredményhalmaz oszlopait egymás mellé helyezi. Példának okáért, ha JOIN utasítással egyesítjük a Recipe_Classes (Receptosztályok) tábla RecipeClassDescription (Receptosztály leírása) mezőjét és a Recipes (Receptek) tábla RecipeTitle (Receptcím) mezőjét, az eredmény a 10.1. ábrához lesz hasonló.

RecipeClassDescription	RecipeTitle
Main course	Irish Stew
Main course	Fettuccini Alfredo
Main course	Pollo Picoso
Main course	Roast Beef
Main course	Huachinango Veracruzana (Red Snapper, Veracruz style)
Main course	Tourtière (French-Canadian Pork Pie)
Main course	Salmon Filets in Parchment Paper
Vegetable	Garlic Green Beans
<< *további sorok* >>	

10.1. ábra

Adatok kinyerése két táblából JOIN utasítással

Először is vessünk egy gyors pillantást egy egyszerű, UNION-t tartalmazó utasítás formájára, amit a 10.2. ábrán mutatunk be.



10.2. ábra

Egyszerű, UNION-t tartalmazó utasítás szintaxisdiagramja

A UNION utasítás egybeilleszti az egyik eredményhalmazból származó sorokat a másikból származókkal. Mindkét eredményhalmaz létrehozása olyan SELECT utasítással történik, amely azon felül, hogy a FROM záradékban összetett JOIN utasítást tartalmazhat, rendelkezhet WHERE, HAVING, illetve GROUP BY záradékokkal is. Az elkészült eredményhalmazokat végül a UNION kulcsszó használatával kapcsoljuk össze. (A GROUP BY záradékkal a 13., a HAVING záradékkal pedig a 14. fejezetben foglalkozunk.) Ha a Recipe_Classes és a RecipeTitle táblák unióján lefuttatjuk a RecipeClassDescription mezőben tárolt értékeket megjelenítő lekérdezést, akkor a 10.3. ábrán láthatóhoz hasonló választ kapunk.

Vegyük észre, hogy az eredményül kapott halmaz egyetlen oszlopból áll. Az oszlop a nevét a SELECT utasításban szereplők közül az elsőtől örökli, de tartalmaz adatot a RecipeTitle oszlopból (Asparagus) csakúgy, mint a RecipeClassesDescription oszlopból (Dessert). A két oszlop adatai azonban nem egymás mellett, hanem egyetlen oszlopban, egymás alatt jelennek meg.

RecipeClassDescription
Asparagus
Coupe Colonel
Dessert
Fettuccini Alfredo
Garlic Green Beans
Hors d'oeuvres
Huachinango Veracruzana (Red Snapper, Veracruz style)
Irish Stew
<< további sorok >>

10.3. ábra

Adatok kinyerése két táblából UINON utasítással

A 10.2. ábrát tanulmányozva esetleg elgondolkodtunk az ALL kulcsszó céljáról, amelyet az ábra szerint nem kötelező használnunk. Ezt a kulcsszót elhagyva az adatbázisrendszer a többször előforduló értékeket csak egyszer jeleníti meg. Például ha a RecipeClass-Description oszlopban és a RecipeTitle oszlopban egyaránt található Dessert, akkor a végső eredményhalmazba csak egyetlen Dessert értéket tartalmazó sor kerül. Amennyiben viszont beírjuk az utasításba az ALL kulcsszót, az összes Dessert megjelenik. Ha belegondolunk, a UNION ALL forma lényegesen hatékonyabb lehet, mivel az adatbázisrendszer nem végez többletmunkát a többször szereplő értékeket keresve. Ha bizonyosak vagyunk afelől, hogy a UNION utasítással egyesített lekérdezéseink nem tartalmaznak többször előforduló sorokat (vagy ezek nem zavarnak bennünket), akkor használjuk mindig az ALL kulcsszót.

Ahhoz, hogy a UNION művelet megtörténhessen, a bemeneti eredményhalmazoknak meg kell felelniük néhány követelménynek. Először is a UNION utasítással összekapcsolt két SELECT utasításban a SELECT utasítás után ugyanannyi oszlopnevet kell megadnunk, így az eredményhalmaz is ugyanannyi oszloppal bír majd. Másodszor, a megjelölt oszlopoknak az SQL-szabvány szerint „összehasonlíthatónak" kell lenniük.

Megjegyzés *A teljes SQL:2003-szabvány engedélyezi a UNION művelet egymástól eltérő oszlopokon való végrehajtását is, a legtöbb kereskedelmi SQL-megvalósítás azonban csak az általunk is tárgyalt alapvető, más szóval belépő szintű szabványnak felel meg. Előfordulhat, hogy a UNION utasításnak ennél kreatívabb felhasználását is támogatja az adatbázisrendszerünk.*

Ahogy a 6. fejezetben tárgyaltuk, szöveges értékeket csak szöveges értékekkel, számokat csak számokkal és dátumokat, illetve időértékeket is csak egyező formátumú adatokkal érdemes összehasonlítanunk. Bár néhány adatbázisrendszer megengedi az adattípusok

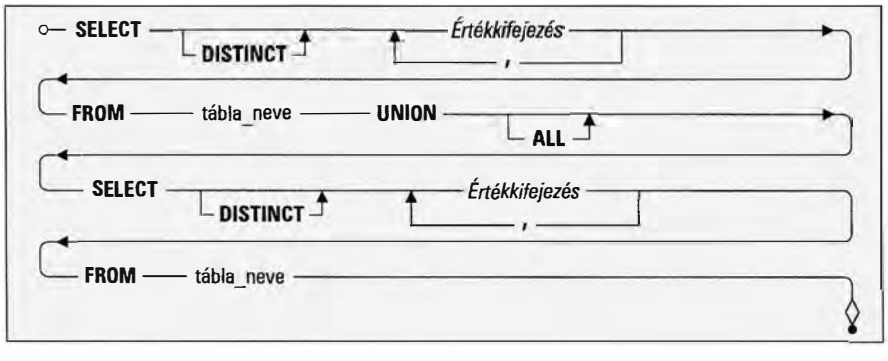
keverését az összehasonlításokon belül, nem sok értelme van egy szöveges értéket – például a János szót – egy számértékkel – például az 55-tel – összehasonlítani. Összehasonlíthatónak akkor nevezünk két oszlopot, ha van értelme a két oszlopot WHERE záradékban összehasonlítani. Ezt jelenti az, hogy az SQL-szabvány megköveteli, hogy a UNION művelet egyik bemeneti értékhalmazának összehasonlítható adattípusúnak kell lennie a másik bemeneti értékhalmazzal.

UNION utasítást tartalmazó lekérdezések

Az INNER JOIN-ról, illetve az OUTER JOIN-ról szóló előző fejezetekben azt tanulmányoztuk, hogy miként alkothatunk SELECT utasításokat a SELECT, FROM és WHERE záradékok felhasználásával. A fejezetek középpontjába a FROM záradékon belüli összetett JOIN utasítások kerültek. UNION műveletet tartalmazó utasítás írásakor olyan *SELECT kifejezést* adunk meg, amely két vagy több SELECT utasítást egy UNION művelettel köt össze. Minden SELECT utasításnak lehet egyszerű vagy összetett FROM záradéka, ahogy azt az elvégzendő feladat megkívánja.

Egyszerű SELECT utasítások használata

Kezdjük azzal, hogy előállítjuk két egyszerű, a FROM záradékban egyetlen táblát használó SELECT utasítás unióját. A 10.4. ábra két egyszerű SELECT utasításon végzett UNION művelet szintaxisdiagramját mutatja be.



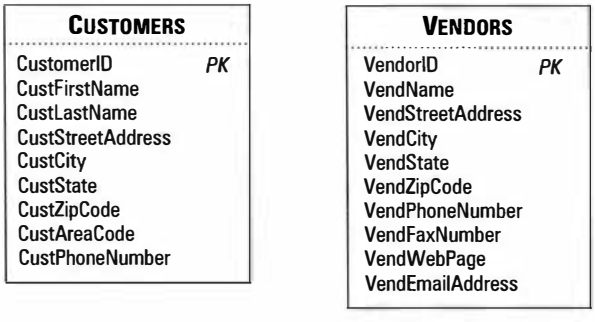
10.4. ábra

UNION művelet használata két egyszerű SELECT utasítás összekapcsolására

A JOIN utasítástól eltérően itt minden a két SELECT utasítást összekapcsoló UNION műveleten belül történik. Ahogy korábban már szóltunk róla, ha az ALL kulcsszót elhagyjuk, az adatbázisrendszer minden többször előforduló sorból csak az elsőt hagyja meg. Ez azzal járhat, hogy az eredményhalmaz esetleg a UNION utasításban szereplő bemeneti értékhalmazok sorainak összegénél kevesebb sorral fog bírni. Ha viszont az ALL kulcsszót használjuk, az eredményhalmaz sorainak száma meg fog egyezni a UNION utasításban szereplő bemeneti értékhalmazok sorainak összegével.

Megjegyzés *Az SQL-szabvány meghatároz egy CORRESPONDING nevű záradékot is, amelyet a UNION kulcsszó után helyezve jelezhetjük, hogy a UNION műveletet a bemeneti értékhalmazok egyforma nevű oszlopait összehasonlítva kívánjuk elvégeztetni. Az összehasonlítást tovább finomíthatjuk, ha a CORRESPONDING kulcsszó után felsoroljuk az összehasonlítandó oszlopok neveit. Bár a nagyobb kereskedelmi SQL-megvalósítások között nem leltünk olyat, amely végre tudná hajtani ezt a feladatot, a használatunkban lévő termék későbbi változataiba esetleg bekerül majd a támogatása.*

Hozzunk létre egy egyszerű uniót – olyan, a vásárlók és a beszállítók címeit tartalmazó listát, amely a Sales Orders mintaadatbázison alapul majd. A 10.5. ábra bemutatja a két szükséges táblát.



10.5. ábra

A Sales Orders mintaadatbázis Customers és Vendors táblái

Vegyük észre, hogy nincs „természetes" kapcsolat a két tábla között, viszont mindkettőben vannak hasonló értelmű és adattípusú oszlopok. A címlistához a név, az utca és házszám, a város neve, az állam neve, illetve az irányítószám oszlopok kellenek. Minthogy ezek a mezők mindkét táblában összehasonlítható módon, szöveges adatként tárolódnak, az adattípusok miatt nem kell aggódnunk. (Néhány adatbázis-tervező az irányítószámokat számként tárolná, de ez nem jelent gondot, már amennyiben a két tábla irányítószámot tartalmazó oszlopai összehasonlítható adattípusúak.)

Van azért egy gondunk: míg a neveket a Vendors (Beszállítók) táblában egy oszlopban tároljuk, addig a Customers (Vásárlók) táblában már két mezőre vannak bontva – a CustFirstName, illetve CustLastName (vagyis keresztnév és vezetéknév) oszlopokra. Annak érdekében, hogy a két táblából azonos számú oszlopot tudjunk használni, olyan kifejezést kell alkotnunk, amely a Customers tábla két oszlopát egyesítve, egy oszlopként adja át a UNION műveletnek a Vendors tábla szintén egyetlen oszlopával együtt. Alkossuk meg ezt a lekérdezést!

Megjegyzés *A fejezet során a 4. fejezetben bevezetett „Kérelem – Fordítás – Tisztázás – SQL" módszert használjuk.*

„Készíts egy címlistát, amely a vásárlók nevét, címét, városát, államát és irányítószámát, valamint a beszállítók nevét, címét, városát, államát és irányítószámát tartalmazza!"

Fordítás Select customer full name, customer address, customer city, customer state, and customer ZIP Code from the customers table combined with vendor name, vendor address, vendor city, vendor state, and vendor ZIP Code from the vendors table (Válaszd ki a vásárlók teljes nevét, címét, városát, államát és irányítószámát a vásárlók táblájából, és ezt egyesítsd a beszállítók nevével, címével, városával, államával és irányítószámával a beszállítók táblájából.)

Tisztázás Select customer full name, customer address, customer city, customer state, ~~and~~ customer ZIP Code from ~~the~~ customers ~~table~~ ~~combined with~~ union Select vendor name, vendor address, vendor city, vendor state, ~~and~~ vendor ZIP Code from ~~the~~ vendors ~~table~~

SQL
```
SELECT Customers.CustLastName || ', ' ||
   Customers.CustFirstName AS MailingName,
   Customers.CustStreetAddress, Customers.CustCity,
   Customers.CustState, Customers.CustZipCode
FROM Customers UNION SELECT Vendors.VendName,
   Vendors.VendStreetAddress, Vendors.VendCity,
   Vendors.VendState, Vendors.VendZipCode
FROM Vendors
```

Figyeljük meg, hogy mindkét SELECT utasítás öt oszlopot állít elő, de a Customers tábla esetében külön kifejezést kellett használnunk a két nevet tartalmazó oszlop egyesítésére. Mindkét SELECT utasítás összes oszlopa szöveges adat, tehát az összehasonlíthatóságukat illetőleg nem ütköztünk problémába.

Esetleg elgondolkodunk azon, hogy mi lesz a lekérdezés futtatása után kapott oszlopok neve. Jó kérdés! Az SQL-szabvány meghatározza, hogy amennyiben a megfelelő oszlopok nevei azonosak (példának okáért az első SELECT utasítás negyedik oszlopának neve megegyezik a második SELECT utasítás negyedik oszlopával), akkor a kimeneti oszlop neve egyezzen meg velük. Amennyiben az oszlopnevek eltérnek (mint az előzőekben megalkotott példánkban), akkor az SQL-szabvány a következőképp rendelkezik: „Ha a lekérdező kifejezés azonnali UNION vagy INTERSECT műveletet tartalmaz, és a megnyitott táblák megfelelő oszlopnevei nem egyezőek, akkor az eredményül kapott oszlop nevét a megvalósítás határozza meg."

Magyarán, az adatbázisrendszerünk dönti el, hogy milyen nevet rendel a kimeneti oszlophoz. Az adatbázisrendszerünk akkor követi az SQL-szabványt, ha a választott név nem szerepel az UNION művelet más bemeneti oszlopának neveként. A legtöbb kereskedelmi

adatbázisrendszer alapértelmezés szerint az első SQL-utasításban szereplő nevet adja az eredményoszlopnak. Az előző példa esetében tehát MailingName, CustStreetAddress, CustCity, CustState, és CustZipCode oszlopokat fogunk látni.

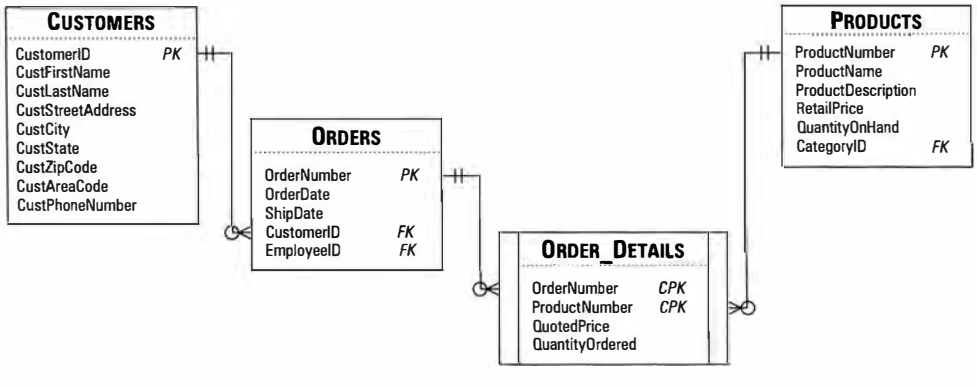
Észrevehetjük, hogy az ALL kulcsszót nem használtuk a UNION utasításban. Bár valószínűtlen, hogy egy vásárló vezeték- és keresztneve megegyezik egy beszállító nevével (a címről, városról, államról és irányítószámról most nem beszélünk), el akartuk kerülni a címek többszöri előfordulását. Ha biztosak vagyunk abban, hogy nem lesznek többször előforduló értékek a UNION művelet bemeneti oszlopaiban, akkor használhatjuk az ALL kulcsszót. Az ALL kulcsszó minden valószínűség szerint felgyorsítja a lekérdezés lefutását, hiszen ilyenkor az adatbázisrendszer nem végez többletmunkát a többszöri előfordulások kiküszöbölése érdekében.

Összetett SELECT utasítások egyesítése

Ahogy gondolhatjuk, a UNION művelettel összekapcsolt SELECT utasítások olyan összetettek lehetnek, amennyire csak a feladat elvégzéséhez szükséges. Az egyetlen megkötés, hogy a SELECT utasításoknak végső soron azonos számú oszlopot kell előállítaniuk, illetve a megfelelő oszlopoknak összehasonítható adattípusúnak kell lenniük.

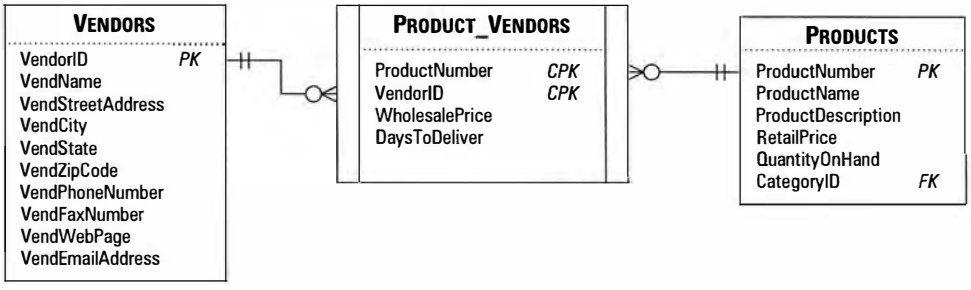
Tételezzük fel, hogy egy olyan listát szeretnénk, amelyben benne van az összes vásárlónk, az általuk rendelt kerékpárok, mindez egyesítve az összes beszállítóval, illetve az általuk szállított kerékpárokkal. Először is keressük ki a szükséges táblákat! A 10.6. ábra a vásárlók és a termékek egymáshoz rendeléséhez szükséges táblákat mutatja.

Úgy tűnik, hogy négy táblát kell JOIN művelettel összekapcsolnunk. Ha látni szeretnénk a beszállítókat és az általuk árult kerékpárokat, a 10.7. ábrán látható táblákra van szükségünk.



10.6. ábra

A vásárlók és az általuk rendelt termékek egymáshoz rendeléséhez szükséges kapcsolatok a táblák között



10.7. ábra

A beszállítók és az általuk árult termékek egymáshoz rendeléséhez szükséges kapcsolatok a táblák között

Ahogy a 8. fejezetben tárgyaltuk, több JOIN záradékot is egymásba ágyazhatunk, így összekapcsolhatjuk a tábláinkat egy összetett probléma megoldásához szükséges adatok összegyűjtéséhez. Ha ismétlésre van szükségünk, a 10.8. tábla bemutatja a három tábla egymásba ágyazásakor használt utasításformát.

Immár rendelkezésünkre áll az összes ismeret a kérdés megválaszolásához. Megalkothatjuk az összetett INNER JOIN műveletet, amely megadja a vásárlók adatait, beszúrhatjuk a UNION kulcsszót, majd végül megalkothatjuk a beszállítók adatainak kinyeréséhez szükséges összetett INNER JOIN műveletet is.

„Sorold fel a vásárlókat és az általuk rendelt kerékpárokat, egyesítve a beszállítókkal és az általuk szállított kerékpárokkal!"

Fordítás Select customer full name and product name from the customers table joined with the orders table on customer ID in the customers table matches customer ID in the orders table, then joined with the order details table on order number in the orders table matches order number in the order details table, and then joined with the products table on product number in the products table matches product number in the order details table where product name contains 'bike', combined with select vendor name and product name from the vendors table joined with the product vendors table on vendor ID in the vendors table matches vendor ID in the product vendors table, and then joined with the products table on product number in the products table matches product number in the product vendors table where product name contains 'bike'
(Válaszd ki a vásárlók teljes nevét a vásárlók táblájából, amelyet összekapcsoltunk a rendelések táblájával a vásárlóazonosító alapján, megfeleltetve a vásárlók táblájának vásárlóazonosítóit a rendelések táblájában levő vásárló-

azonosítóknak, majd ezt összekapcsoltuk a rendelések részleteit tartalmazó táblával a rendelésszám alapján, a rendelések táblájának rendelésszámait megfeleltetve a rendelések részleteit tartalmazó tábla rendelésszámainak, majd ezt összekapcsoltuk a termékek táblájával a termékszám alapján, a terméktábla termékszámait megfeleltetve a rendelések részleteit tartalmazó tábla termékszámainak, ahol a termék neve tartalmazza a „kerékpár" szót, majd mindennek vedd az unióját a beszállító nevét és a termék nevét a beszállítók táblájából kiválasztó utasítással, amelyet összekapcsoltunk a termékbeszállítók táblájával a beszállítóazonosító alapján, a beszállítók táblájának beszállítóazonosítóit megfeleltetve a termékbeszállítók táblájában levő beszállítóazonosítóknak, majd ezt összekapcsoltuk a termékek táblájával a termékszám alapján, a termékek táblájának termékszámait megfeleltetve a termékbeszállítók táblájában levő termékszámoknak, ha a termék neve tartalmazza a „kerékpár" szót.)

Tisztázás

Select customer full name ~~and~~ product name
from ~~the~~ customers ~~table~~ joined with the orders ~~table~~
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~then~~ joined with the order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_details.order number ~~in the order details table,~~
~~and then~~ joined with the products ~~table~~
on products.product number ~~in the products table matches~~
= order_details.product number ~~in the order details table~~
where product name ~~contains~~ like '%bike%'~~,~~
~~combined with~~ union select vendor name ~~and~~ product name
from ~~the~~ vendors table joined with the product vendors ~~table~~
on vendors.vendor ID ~~in the vendors table matches~~
= product_vendors.vendor ID ~~in the product vendors table,~~
~~and then~~ joined with the products ~~table~~
on products.product number ~~in the products table matches~~
= product_vendors.product number ~~in the product vendors table~~
where product name ~~contains~~ like '%bike%'

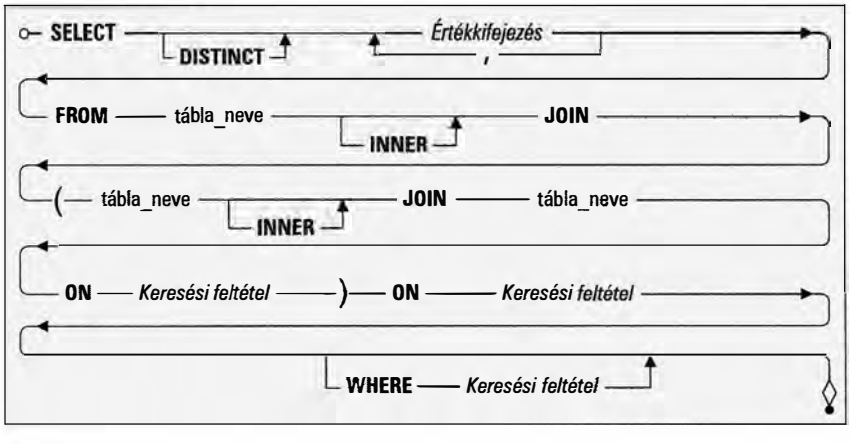
SQL

```
SELECT Customers.CustLastName || ', ' ||
   Customers.CustFirstName AS FullName,
   Products.ProductName, 'Customer' AS RowID
FROM ((Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
```

```
   Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%bike%'
UNION
SELECT Vendors.VendName, Products.ProductName,
   'Vendor' AS RowID
FROM (Vendors INNER JOIN Product_Vendors
ON Vendors.VendorID = Product_Vendors.VendorID)
INNER JOIN Products
ON Products.ProductNumber =
   Product_Vendors.ProductNumber
WHERE Products.ProductName LIKE '%bike%'
```

Hát, ez nem sokkal rövidebb a Ráktérítőnél, de legalább elvégzi a teendőt! Figyeljük meg, hogy bevezettünk egy RowID (sorazonosító) nevű azonosítót mindkét SELECT utasításban, hogy könnyen elkülöníthessük a vásárlók, illetve a beszállítók közül származó sorokat. Késztetést érezhetünk a DISTINCT kulcsszó beszúrására az első SELECT utasításba, mondván, hogy az igazán jó ügyfél egynél többször is megrendelheti ugyanazt a kerékpármodellt. Minthogy nem használtuk az ALL kulcsszót az UNION utasításban, a lekérdezés eleve eltünteti a többszöri előfordulásokat, így ha beszúrjuk a DISTINCT kulcsszót, arra kérjük az adatbázisrendszert, hogy a többszöri előfordulásokat kiszűrő többletmunkát kétszer is végezze el.

Ha UNION utasítást tartalmazó lekérdezést kell alkotnunk, akkor érdemes először az egyes SELECT utasításokat megírnunk. Ennek végeztével könnyedén átmásolhatjuk a SELECT utasításokat az új lekérdezésbe, ahol a UNION kulcsszóval választjuk el azokat egymástól.



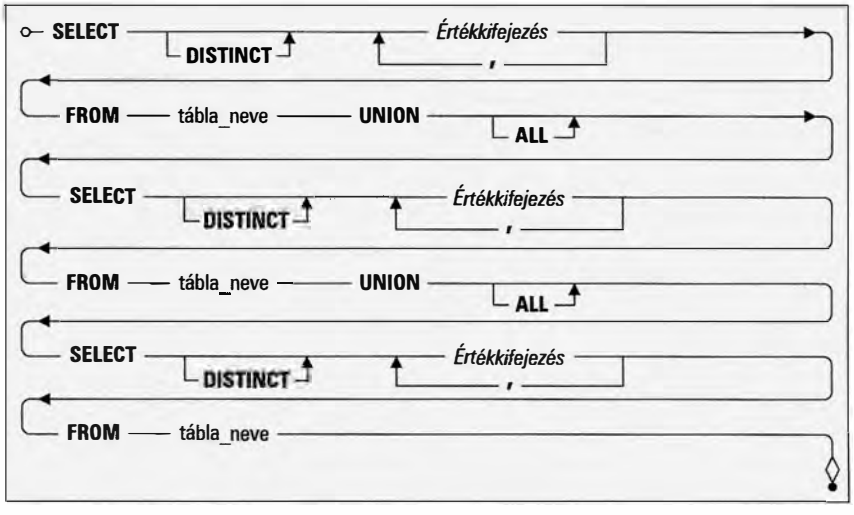
10.8. ábra

Három tábla összekapcsolása JOIN utasítással

Több UNION művelet használata

Mindezidáig csak annyit mutattunk meg, hogy két értékhalmaz miként egyesíthető a UNION utasítás segítségével. Igazság szerint a második SELECT utasítást újabb UNION kulcsszó követheti, majd ezt újabb SELECT utasítás. Bár néhány SQL-megvalósítás korlátozza a UNION műveletekkel egyesíthető értékhalmazok számát, elméletben addig bővíthetjük a lekérdezésünket újabb UNION-okkal és SELECT-ekkel, ameddig csak kedvünk tartja.

Tételezzük fel, hogy három különböző táblából – a Customers, az Employees és a Vendors táblákból – kell címlistát készítenünk, mondjuk karácsonyi üdvözlőlapok címzéséhez. Az ennek a listának az elkészítéséhez használatos szintaxisdiagramot a 10.9. ábra mutatja be.



10.9. ábra

Három tábla egyesítése UNION utasítással

Látható, hogy külön SELECT utasítással nyerjük ki a neveket és a címeket a Customers táblából, ezt UNION utasítással egyesítjük az Employees tábla hasonló adataival, és ezt is UNION művelettel egyesítjük a Vendors táblából származó nevekkel és címekkel. (A folyamat egyszerűsítése végett a Fordítás és a Tisztázás részeket ebben a példában összevontuk.)

„Készítsd el a vásárlók, alkalmazottak és beszállítók címlistáját!"

Fordítás/ Tisztázás

```
Select customer full name, customer street address, customer
city, customer state, ~~and~~ customer ZIP Code
from ~~the~~ customers ~~table~~
~~combined with~~ union
select employee full name, employee street address,
employee city, employee state, ~~and~~ employee ZIP Code
```

```
from the employees table
combined with union
select vendor name, vendor street address, vendor city,
vendor state, and vendor ZIP Code
from the vendors table
```

SQL

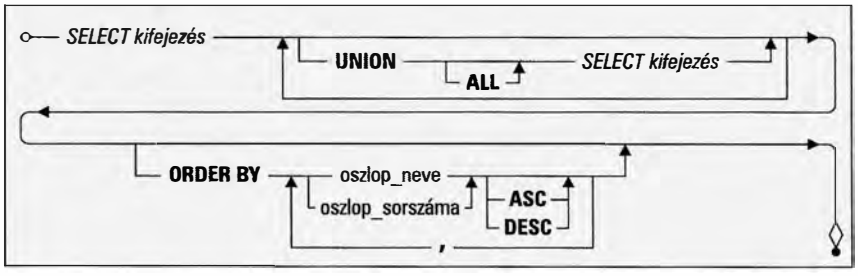
```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   Customers.CustStreetAddress, Customers.CustCity,
   Customers.CustState, Customers.CustZipCode
FROM Customers
UNION
SELECT Employees.EmpFirstName || ' ' ||
   Employees.EmpLastName AS EmpFullName,
   Employees.EmpStreetAddress, Employees.EmpCity,
   Employees.EmpState, Employees.EmpZipCode
FROM Employees
UNION
SELECT Vendors.VendName, Vendors.VendStreetAddress,
   Vendors.VendCity, Vendors.VendState,
   Vendors.VendZipCode
FROM Vendors
```

Természetesen ha a címlistát egy bizonyos városra, államra, vagy irányítószám-tartományra szűrni szeretnénk, megtehetjük, mégpedig úgy, hogy megadunk egy WHERE záradékot, bármely vagy akár az összes SELECT utasítás részeként. Példának okáért, ha a vásárlók, alkalmazottak és beszállítók listáját csak egy bizonyos államra vonatkozólag akarjuk elkészíteni, akkor WHERE záradékot kell adnunk az *összes* beágyazott SELECT utasításhoz. Alkalmazhatunk szűrést egyetlen SELECT utasításon is: például akkor, amikor a texasi beszállítók, az összes vásárló és az összes alkalmazott listáját készítjük el.

Uniók rendezése

Hogyan történik a UNION utasítás eredményeként kapottak rendezése? Sok adatbázis-rendszer esetében azt tapasztalhatjuk, hogy az eredményhalmazban a kimeneti oszlopok egyszerűen a bemeneti oszlopok rendezését követve jelennek meg, balról jobbra. Például az iménti példánkban szereplő, három tábla unióját megvalósító lekérdezésnél a sorokban a rendezés alapja a nevek, két azonos névnél az utcák és a házszámok, és így tovább.

Hogy a postások se szomorkodjanak (és esetleg kedvezményt kapjunk a tömeges levélfeladáskor), rendezzük a sorokat irányítószám szerint. Ez elvégezhető egy ORDER BY záradék hozzáadásával, ha odafigyelünk arra, hogy a záradéknak az utolsó SELECT utasítás legvégén kell állnia. Az ORDER BY záradék ugyanis a UNION művelettel képzett eredményhalmazra, nem pedig a SELECT utasítás kimenetére vonatkozik. A 10.10. ábra bemutatja, hogy miként hajtható végre egy ilyen rendezés.



10.10. ábra

Rendezési feltétel megadása UNION utasítást tartalmazó lekérdezésben

Ahogy a diagram mutatja, annyi UNION és SELECT utasítást alkalmazunk az egyesítendő eredményhalmazok kinyerése végett, amennyit csak akarunk, de az ORDER BY záradéknak az utasítás végén kell következnie. Felmerülhet a kérdés: „Milyen oszlopnevet vagy oszlopszámot használjunk az ORDER BY záradékban?" Ne feledjük, hogy a lekérdezés előző részeinek együttes rendezéséről van szó. Amint azt már tárgyaltuk, az oszlopnevek ugyan megvalósításfüggőek, a legtöbb adatbázisrendszer mégis az első SELECT utasítás során megadott oszlopneveket használja.

Megadhatjuk az oszlop sorszámát is; ilyenkor az első kimeneti oszlop kapja az 1-es számot. Abban a lekérdezésben, amely a neveket, címeket, városokat, államokat és irányítószámokat adja meg, az 5-ös oszlopszám (az irányítószámok az ötödik oszlopban vannak) megadásával rendezhetjük a kimenetet irányítószámok szerint.

Rendezzük a címlistát előállító lekérdezés eredményét mindkét módon. Íme az oszlop neve szerinti rendezés helyes utasításformája:

SQL
```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   Customers.CustStreetAddress, Customers.CustCity,
   Customers.CustState, Customers.CustZipCode
FROM Customers
UNION
SELECT Employees.EmpFirstName || ' ' ||
   Employees.EmpLastName AS EmpFullName,
   Employees.EmpStreetAddress, Employees.EmpCity,
   Employees.EmpState, Employees.EmpZipCode
FROM Employees
UNION
SELECT Vendors.VendName, Vendors.VendStreetAddress,
   Vendors.VendCity, Vendors.VendState,
   Vendors.VendZipCode
FROM Vendors
ORDER BY CustZipCode
```

Természetesen abból indulunk ki, hogy a kimenetben annak az oszlopnak a neve, amely alapján rendezni akarunk, megegyezik az első SELECT utasításban megadott oszlopnévvel. Az oszlop sorszáma alapján megadott rendezési feltétel használatát a következő példa szemlélteti:

SQL
```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   Customers.CustStreetAddress, Customers.CustCity,
   Customers.CustState, Customers.CustZipCode
FROM Customers
UNION
SELECT Employees.EmpFirstName || ' ' ||
   Employees.EmpLastName AS EmpFullName,
   Employees.EmpStreetAddress, Employees.EmpCity,
   Employees.EmpState, Employees.EmpZipCode
FROM Employees
UNION
SELECT Vendors.VendName, Vendors.VendStreetAddress,
   Vendors.VendCity, Vendors.VendState,
   Vendors.VendZipCode
FROM Vendors
ORDER BY 5
```

A UNION utasítás használatának területei

A UNION utasítást aligha fogjuk olyan sokszor használni, mint az INNER JOIN-t vagy az OUTER JOIN-t. A UNION utasításnak leginkább különböző táblákból származó, két vagy több hasonló eredményhalmaz egyesítésekor fogjuk hasznát látni. Bár a UNION utasítást *lehet* egyazon táblából származó két eredményhalmaz egyesítésére is használni, az ilyen esetek rendszerint megoldhatók egy összetettebb WHERE záradékot tartalmazó SELECT utasítással is. A *Példák* című részben felvetünk pár ilyen problémát, hogy megmutassuk, hogyan oldható meg a feladat hatékonyabban a WHERE záradékkal.

Következzen néhány olyan, a mintaadatbázisainkra vonatkozó kérelem, amely a UNION utasítással fogalmazható meg:

„Mutasd meg az összes vásárló és beszállító nevét és címét!"
„Készítsd el a kerékpárt rendelő és a bukósisakot rendelő vásárlók egyesített listáját!"
(Ez is azoknak a kérelmeknek az egyike, amelyet megfogalmazhatunk egy SELECT utasítással és egy összetett WHERE záradékkal.)
„Állítsd elő a vásárlók és a beszállítók címjegyzékét!"
„Írd ki a kerékpárt rendelő vásárlók és a kerékpárt szállító beszállítók közös listáját!"
„Készítd el az ügynököket és az előadókat egyesítő listát!"
„Állítsd elő a kortárs zenét kedvelő megrendelők és a kortárs zenét játszó előadók közös listáját!"

„Mutasd meg a rajzból 85 vagy több pontos átlagot elért hallgatók és a 9-es vagy jobb minősítésű, rajzot tanító kari oktatók közös listáját!"
„Találd meg azokat a játékosokat, akik a Thunderbird Lanes pályán legalább 155-ös tiszta pontszámot értek el, egyesítve a listát a Bolero Lanes pályán legalább 140-es tiszta pontszámot elérőkkel! (Ez is olyan eset, ami megoldható egy SELECT utasítással és egy összetett WHERE záradékkal.)
„Sorold fel közös listán a torna páratlan pályán induló mérkőzéseit, a csapatok neveit, a csapatkapitányokat, valamint a páros pályán induló mérkőzéseket, a csapatok neveit és a csapatkapitányokat!"
„Állíts össze szójegyzéket az összes receptnévből és hozzávalóból!"
„Jeleníts meg közös listát az összes hozzávalóval és az alapértelmezett mértékegységeikkel, illetve a receptekben felhasznált hozzávalókkal és az alapértelmezett mértékegységeikkel!"

Példák

Mostanra már ismerjük a UNION utasítást tartalmazó lekérdezések megalkotásának mikéntjét, és láttunk néhány UNION-nal megválaszolható kérdéstípust is. Tekintsünk át néhány jellemző példát, amelyek mindegyikében UNION utasítást használunk, és amelyekben mintaadatbázisaink mindegyike előkerül! A példák azt szemléltetik, hogy miként egyesíthetjük UNION művelettel két eredményhalmaz sorait.

Az SQL kód után rögtön bemutatjuk a kód által visszaadott eredményhalmaz-mintát. Az a név, amely megelőzi az eredményhalmazt, megegyezik a könyv CD-mellékletén lévő mintaadatokon lefuttatható lekérdezés nevével. Minden lekérdezést a hozzá tartozó mintaadatbázisban tároltunk (ahogy a példában jeleztük), és az ehhez a fejezethez tartozó lekérdezések neve elé CH10-et írtunk. A könyv bevezetőjében található utasításokat követve a számítógépünkbe tölthetjük és kipróbálhatjuk a mintákat.

Megjegyzés *A szokásos „Fordítás" és „Tisztázás" részeket az egyszerűség kedvéért a következő példák mindegyikénél összevontuk.*

Sales Orders adatbázis

„Mutasd meg az összes vásárló és alkalmazott nevét és címét, minden előfordulásukat beleértve, irányítószám szerint rendezve!"

Fordítás/ Tisztázás

```
Select customer first name, customer last name,
customer street address, customer city,
customer state, and customer ZIP Code
from the customers table
combined with union all
Select employee first name, employee last name,
employee street address, employee city,
```

employee state, ~~and~~ employee ZIP Code
from ~~the~~ employees ~~table,~~
order by ZIP Code

SQL

```
SELECT Customers.CustFirstName,
   Customers.CustLastName,
   Customers.CustStreetAddress, Customers.CustCity,
   Customers.CustState, Customers.CustZipCode
FROM Customers
UNION ALL
SELECT Employees.EmpFirstName,
   Employees.EmpLastName,
   Employees.EmpStreetAddress, Employees.EmpCity,
   Employees.EmpState, Employees.EmpZipCode
FROM Employees
ORDER BY CustZipCode
```

CH10_Customers_UNION_ALL_Employees (35 sor)

CustFirstName	CustLastName	CustStreetAddress	CustCity	CustState	CustZipCode
Estella	Pundt	2500 Rosales Lane	Dallas	TX	75260
Robert	Brown	672 Lamont Ave	Houston	TX	77201
Kirk	DeGrasse	455 West Palm Ave	San Antonio	TX	78284
Kirk	DeGrasse	455 West Palm Ave	San Antonio	TX	78284
Angel	Kennedy	667 Red River Road	Austin	TX	78710
Maria	Patterson	3445 Cheyenne Road	El Paso	TX	79915
Mark	Rosales	323 Advocate Lane	El Paso	TX	79915
Caleb	Viescas	4501 Wetland Road	Long Beach	CA	90809
<< további sorok >>					

(Kirk DeGrasse bizonyára vásárló és alkalmazott egy személyben.)

„Készítsd el a kerékpárt rendelő és a bukósisakot rendelő vásárlók egyesített listáját!"

Fordítás/ Tisztázás

Select customer first name, customer last name,
~~and the constant~~ 'Bike'
from ~~the~~ customers ~~table~~
join~~ed with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~then~~ join~~ed with the~~ order details ~~table~~
on orders.order number ~~in the orders table matches~~

= order_details.order number ~~in the order details table,~~
~~and then~~ joined ~~with the~~ products ~~table~~
on product number ~~in the products table matches~~
= order_details.product number ~~in the order details table~~
where product name ~~contains~~ like '%bike%,'
~~combined with~~ union
Select customer first name, customer last name,
~~and the constant~~ 'Helmet'
from ~~the~~ customers ~~table~~ joined ~~with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~then~~ joined ~~with the~~ order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_details.order number ~~in the order details table,~~
~~and then~~ joined ~~with the~~ products ~~table~~
on product number ~~in the products table matches~~
= order_details.product number ~~in the order details table~~
where product name ~~contains~~ like '%helmet%'

SQL

```
SELECT Customers.CustFirstName,
   Customers.CustLastName, 'Bike' AS ProdType
FROM ((Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%bike%'
UNION
SELECT Customers.CustFirstName,
   Customers.CustLastName, 'Helmet' AS ProdType
FROM ((Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%helmet%'
```

CH10_Customer_Order_Bikes_UNION_Customer_Order_Helmets (52 rows)

CustFirstName	CustLastName	ProdType
Alaina	Hallmark	Bike
Andrew	Cencini	Bike
Andrew	Cencini	Helmet
Angel	Kennedy	Bike
Angel	Kennedy	Helmet
Caleb	Viescas	Bike
Caleb	Viescas	Helmet
Darren	Gehring	Bike
<< további sorok >>		

Figyeljük meg, hogy ez is azoknak a feladatoknak az egyike, amelyek megoldhatók egy SELECT utasítással és egy lényegesen összetettebb WHERE záradékkal. A UNION utasítás használatának egyetlen előnye, hogy lényegesen egyszerűbb egy további „halmazazonosító" oszlop hozzáadása (esetünkben a ProdType oszlopról van szó) az eredményhalmazokhoz, amiből megállapíthatjuk, hogy melyik vásárló melyik bemeneti halmazból került az eredményoszlopba. Ugyanakkor fontos lehet tudnunk, hogy a legtöbb adatbázisrendszer a UNION műveletnél sokkal gyorsabban értékeli ki a WHERE záradékokat – még azokban az esetekben is, amikor ez utóbbi összetett feltételeket tartalmaz. Lássuk ugyanennek a feladatnak a megoldását WHERE záradékkal:

SQL

```
SELECT DISTINCT Customers.CustFirstName,
   Customers.CustLastName
FROM
   ((Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%bike%'
OR Products.ProductName LIKE '%helmet%'
```

Megjegyzés *Láthatjuk, hogy a DISTINCT kulcsszó szükséges a többször előforduló sorok kiküszöböléséhez, amennyiben nem UNION utasítást használunk. Ne feledjük, hogy a UNION utasítás automatikusan eltávolítja a többszöri előfordulásokat, hacsak nem UNION ALL formában adjuk meg. A DISTINCT kulcsszó használata megengedett a UNION utasítással együtt is, de ilyenkor az adatbázisrendszerünkkel a szükségesnél több munkát végeztetünk el.*

CH10_Customers_Bikes_Or_Helmets (27 sor)

CustFirstName	CustLastName
Alaina	Hallmark
Andrew	Cencini
Angel	Kennedy
Caleb	Viescas
Darren	Gehring
David	Smith
Dean	McCrae
Estella	Pundt
<< *további sorok* >>	

Entertainment Agency adatbázis

„Készíts egy az ügynököket és előadókat egyesítő listát!"

Fordítás/ Tisztázás
Select agent full name, ~~and the constant~~ 'Agent'
from ~~the~~ agents ~~table~~
~~combined with~~ union
Select entertainer stage name, ~~and the constant~~ 'Entertainer'
from ~~the~~ entertainers ~~table~~

SQL

```
SELECT Agents.AgtLastName || ', ' ||
   Agents.AgtFirstName AS Name, 'Agent' AS Type
FROM Agents
UNION
SELECT Entertainers.EntStageName,
   'Entertainer' AS Type
FROM Entertainers
```

CH10_Agents_UNION_Entertainers (22 sor)

Name	Type
Bishop, Scott	Agent
Carol Peacock Trio	Entertainer
Caroline Coie Cuartet	Entertainer
Coldwater Cattle Company	Entertainer
Country Feeling	Entertainer
Dumbwit, Daffy	Agent
Jazz Persuasion	Entertainer
Jim Glynn	Entertainer
<< *további sorok* >>	

School Scheduling adatbázis

„Mutasd meg a rajzból 85 vagy több pontos átlagot elért hallgatók és a 9-es vagy jobb minősítésű, rajzot tanító kari oktatók közös listáját!"

Fordítás/ Tisztázás

Select student first name ~~aliased~~ as FirstName, student last name ~~aliased~~ as LastName, ~~and~~ grade ~~aliased~~ as Score
from ~~the~~ students ~~table~~
~~joined with the~~ student schedules ~~table~~
on students.student ID ~~in the students table matches~~
= student_schedules.student ID ~~in the student schedules table,~~
~~then~~ ~~joined with the~~ student class status ~~table~~
on student_class_status.class status
~~in the student class status table matches~~
= student_schedules.class status ~~in the student schedules table,~~
~~then~~ ~~joined with the~~ classes ~~table~~
on classes.class ID ~~in the classes table matches~~
= student_schedules.class ID ~~in the student schedules table,~~
and then joined with the subjects table
on subjects.subject ID ~~in the subjects table matches~~
= classes.subject ID ~~in the classes table~~
where class status description ~~is~~ = 'completed'
and grade ~~is greater than or equal to~~ >= 85
and category ID ~~is~~ = 'ART' ~~combined with~~
union Select staff first name, staff last name,
~~and~~ proficiency rating ~~aliased~~ as Score
from ~~the~~ staff ~~table~~ ~~joined with the~~ faculty subjects ~~table~~
on staff.staff ID ~~in the staff table matches~~
= faculty_subjects.staff ID ~~in the faculty subjects table,~~
~~and then~~ ~~joined with the~~ subjects ~~table~~
on subjects.subject ID ~~in the subjects table matches~~
= faculty_subjects.subject ID ~~in the faculty subjects table~~
where proficiency rating ~~is greater than~~ > 8
and category ID ~~is~~ = 'ART'

SQL

```
SELECT Students.StudFirstName AS FirstName,
   Students.StudLastName AS LastName,
   Student_Schedules.Grade AS Score,
   'Student' AS Type
FROM (((Students INNER JOIN Student_Schedules
ON Students.StudentID =
   Student_Schedules.StudentID)
INNER JOIN Student_Class_Status
ON Student_Class_Status.ClassStatus =
   Student_Schedules.ClassStatus)
INNER JOIN Classes
```

```
ON Classes.ClassID = Student_Schedules.ClassID)
INNER JOIN Subjects
ON Subjects.SubjectID = Classes.SubjectID
WHERE Student_Class_Status.ClassStatusDescription =
   'Completed'
AND Student_Schedules.Grade >= 85
AND Subjects.CategoryID = 'ART'
UNION
SELECT Staff.StfFirstName, Staff.StfLastName,
   Faculty_Subjects.ProficiencyRating AS Score,
   'Faculty' AS Type
FROM (Staff INNER JOIN Faculty_Subjects
ON Staff.StaffID = Faculty_Subjects.StaffID)
INNER JOIN Subjects
ON Subjects.SubjectID = Faculty_Subjects.SubjectID
WHERE Faculty_Subjects.ProficiencyRating > 8
AND Subjects.CategoryID = 'ART'
```

CH10_Good_Art_Students_And_Faculty(16 sor)

FirstName	LastName	Score	Type
Alaina	Hallmark	10	Faculty
Elizabeth	Hallmark	93.27	Student
Kendra	Bonnicksen	88.27	Student
Kendra	Bonnicksen	91.56	Student
Kendra	Bonnicksen	92.05	Student
Liz	Keyser	10	Faculty
Mariya	Sergienko	9	Faculty
Michael	Hernandez	10	Faculty
<< *további sorok* >>			

Bowling League adatbázis

„Sorold fel közös listán a torna páratlan pályán induló mérkőzéseit, a csapatok neveit, a csapatkapitányokat, valamint a páros pályán induló mérkőzéseket, a csapatok neveit és a csapatkapitányokat!"

Fordítás/ Tisztázás

Select tourney location, tourney date, match ID, team name, captain name ~~and the constant~~ 'Odd Lane'
from ~~the~~ tournaments ~~table~~
join~~ed with the~~ tourney matches ~~table~~
on tournaments.tourney ID ~~in the tournaments table equals~~
= tourney_matches.tourney ID ~~in the tourney matches table,~~

then joined with the teams table
on tourney_matches.odd lane team ID in the tourney matches
table equals = teams.team ID in the teams table,
and then joined with the bowlers table
on teams.captain ID in the teams table
equals = bowlers.bowler ID in the bowlers table,
combined with union all
Select tourney location, tourney date, match ID, team name,
captain name and the constant 'Even Lane'
from the tournaments table
joined with the tourney matches table
on tournaments.tourney ID in the tournaments table equals
= tourney_matches.tourney ID in the tourney matches table,
then joined with the teams table
on tourney_matches.even lane team ID
in the tourney matches table
equals = teams.team ID in the teams table,
and then joined with the bowlers table
on teams.captain ID in the teams table
equals = bowlers.bowler ID in the bowlers table,
order by tourney date 2, and match ID 3

SQL

```
SELECT Tournaments.TourneyLocation,
   Tournaments.TourneyDate,
   Tourney_Matches.MatchID, Teams.TeamName,
   Bowlers.BowlerLastName || ', ' ||
   Bowlers.BowlerFirstName AS Captain,
   'Odd Lane' AS Lane
FROM ((Tournaments INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
   Tourney_Matches.TourneyID)
INNER JOIN Teams
ON Teams.TeamID =
   Tourney_Matches.OddLaneTeamID)
INNER JOIN Bowlers
ON Bowlers.BowlerID = Teams.CaptainID
UNION ALL
SELECT Tournaments.TourneyLocation,
   Tournaments.TourneyDate,
   Tourney_Matches.MatchID, Teams.TeamName,
   Bowlers.BowlerLastName || ', ' ||
   Bowlers.BowlerFirstName AS Captain,
   'Even Lane' AS Lane
```

```
FROM ((Tournaments INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
   Tourney_Matches.TourneyID)
INNER JOIN Teams
ON Teams.TeamID =
   Tourney_Matches.EvenLaneTeamID)
INNER JOIN Bowlers
ON Bowlers.BowlerID = Teams.CaptainID
ORDER BY 2, 3
```

Figyeljük meg, hogy a két SELECT utasítás szinte teljesen megegyezik! Az egyetlen különbség abban áll, hogy míg az első SELECT utasítás a Tourney_Matches táblát a Teams táblával az OddLaneTeamID mező alapján egyesíti, addig a másik az egyesítést EvenLaneTeamID mező alapján végzi. Azt is figyeljük meg, hogy a megoldás során úgy döntöttünk, hogy az oszlopnevek (TourneyDate, illetve MatchID) helyett inkább az oszlopok sorszáma (második, illetve harmadik oszlop) alapján végezzük a rendezést. Végül megemlítjük, hogy a UNION ALL forma használata azért jogos, mert egyik csapat sem játszik önmaga ellen.

CH10_Bowling_Schedule (114 sor)

TourneyLocation	TourneyDate	MatchID	TeamName	Captain	Lane
Red Rooster Lanes	2007-09-04	1	Marlins	Fournier,David	Odd Lane
Red Rooster Lanes	2007-09-0	1	Sharks	Patterson, Ann	Even Lane
Red Rooster Lanes	2007-09-04	2	Barracudas	Sheskey, Richard	Even Lane
Red Rooster Lanes	2007-09-04	2	Terrapins	Viescas, Carol	Odd Lane
Red Rooster Lanes	2007-09-04	3	Dolphins	Viescas, Suzanne	Odd Lane
Red Rooster Lanes	2007-09-04	3	Orcas	Thompson, Sarah	Even Lane
Red Rooster Lanes	2007-09-04	4	Manatees	Viescas, Michael	Odd Lane
Red Rooster Lanes	2007-09-04	4	Swordfish	Rosales, Joe	Even Lane
Thunderbird Lanes	2007-09-11	5	Marlins	Fournier,David	Even Lane
Thunderbird Lanes	2007-09-11	5	Terrapins	Viescas, Carol	Odd Lane
<< további sorok >>					

Recipes adatbázis

„Állíts össze szójegyzéket az összes receptnévből és hozzávalóból!"

Fordítás/ Tisztázás

```
Select recipe class description, ~~and the constant~~ 'Recipe Class'
from ~~the~~ recipe classes ~~table~~
~~combined with~~ union
Select recipe title, ~~and the constant~~ 'Recipe'
```

from ~~the~~ recipes ~~table~~
~~combined with~~ union
Select ingredient name, ~~and the constant~~ 'Ingredient'
from ~~the~~ ingredients ~~table~~

SQL

```
SELECT Recipe_Classes.RecipeClassDescription
   AS IndexName, 'Recipe Class' AS Type
FROM Recipe_Classes
UNION
SELECT Recipes.RecipeTitle, 'Recipe' AS Type
FROM Recipes
UNION
SELECT Ingredients.IngredientName,
   'Ingredient' AS Type
FROM Ingredients
```

CH10_Classes_Recipes_Ingredients (101 sor)

IndexName	Type
Asparagus	Ingredient
Asparagus	Recipe
Bacon	Ingredient
Balsamic vinaigrette dressing	Ingredient
Beef	Ingredient
Beef drippings	Ingredient
Bird's custard powder	Ingredient
Black olives	Ingredient
<< további sorok >>	

Összefoglalás

A fejezetet az unió művelet értelmezésével kezdtük, és megmutattuk a különbséget két tábla JOIN utasítással történő összekapcsolása, illetve az eredményhalmazok UNION utasítással való egyesítése között. Ez után megtanultuk, hogy miként alkothatjuk meg két, azonos táblából oszlopokat megjelenítő SELECT utasítás unióját. Megismertük az ALL kulcsszó jelentőségét, megjegyezve, hogy vagy azokban az esetekben érdemes használni, amikor tudjuk, hogy a lekérdezéseinkben nem lesznek többször előforduló értékek, vagy akkor, amikor nem törődünk a többször előforduló értékekkel. Ezt követően továbbléptünk az olyan összetett SELECT utasítások egyesítése felé, amelyek mindegyike JOIN művelettel kapcsolt össze több táblát, majd ismertettük, hogy miként lehet egyesíteni kettőnél több eredményhalmazt a UNION utasítással. A UNION kulcsszót tartalmazó utasítások formájának tárgyalását az eredmény rendezésének bemutatásával zártuk.

Megvizsgáltuk, hogy milyen esetekben van hasznunkra a UNION művelet, és felvetettünk egy sor kérdést, amelyek megválaszolhatók a UNION utasítás segítségével. A *Példák* rész néhány példával szemléltette a UNION utasítás használatát a mintaadatbázisokon, hogy lássuk a feltett kérdések megválaszolására alkalmas kifejezések mögött megbúvó gondolatmenetet.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Sorold fel közös listában azokat a vásárlókat, akik rendeltek bukósisakot, azokkal a beszállítókkal együtt, akik szállítanak bukósisakot!"*
   (Tipp: két összetett, JOIN műveletet tartalmazó SELECT utasítást kell a UNION kulcszóval összekapcsolnunk.)
   A megoldás itt található: CH10_Customer_Helmets_Vendor_Helmets (91 sor).

Entertainment Agency adatbázis

1. *„Készíts egy a vásárlókat és az előadókat egyesítő listát!"*
   (Tipp: ahhoz, hogy mindkét SELECT utasításban azonos számú oszlop legyen, az egyik névhez külön kifejezést kell rendelnünk.)
   A megoldás itt található: CH10_Customers_UNION_Entertainers (28 sor).
2. *„Állítsd elő a kortárs zenét kedvelő megrendelők és a kortárs zenét játszó előadók közös listáját!"*
   (Tipp: két összetett, JOIN műveletet tartalmazó SELECT utasítást kell a UNION kulcsszóval összekapcsolnunk)
   A megoldás itt található: CH10_Customers_Entertainers_Contemporary (5 sor).

Scool Scheduling adatbázis

1. *„Készítsd el a hallgatók és a tanárok címlistáját, irányítószám szerint rendezve!"*
   (Tipp: próbáljuk oszlopszámmal megadni a rendezési feltételt.)
   A megoldás itt található: CH10_Students_Staff_Mailing_List (45 sor)

Bowling League adatbázis

1. *„Találd meg azokat a játékosokat, akik a Thunderbird Lanes pályán legalább 155-ös tiszta pontszámot értek el, egyesítve a listát a Bolero Lanes pályán legalább 140-es tiszta pontszámot elérőkkel!"*
   (Tipp: ez is olyan feladat, amely megoldható egy SELECT utasítással és egy összetett WHERE záradékkal.)
   A UNION utasítást használó megoldás itt található:
   CH10_Good_Bowlers_TBird_Bolero_UNION (129 sor)
   A WHERE utasítást használó megoldás itt található:
   CH10_Good_Bowlers_TBird_Bolero_WHERE (135 sor)
   Próbáljuk megindokolni, hogy miért tér el az eredmény sorainak a száma az előző kérdés kétféle megoldásánál!
   (Tipp: az első lekérdezésben próbálkozzunk a UNION ALL utasítás használatával!)

Recipes adatbázis

1. *„Jeleníts meg közös listát az összes hozzávalóval és az alapértelmezett mértékegységeikkel, illetve a receptekben felhasznált hozzávalókkal és az alapértelmezett mértékegységeikkel!"*
   (Tipp: a megoldáshoz egy egyszerű és egy összetett JOIN utasításra lesz szükség.)
   A megoldás itt található: CH10_Ingredient_Recipe_Measurements (144 sor).

11

Allekérdezések

„Nem oldhatunk meg problémákat ugyanazzal a gondolkodásmóddal, mint amivel létrehoztuk őket."
– Albert Einstein

A fejezet témakörei

- Mi az az allekérdezés?
- Allekérdezések oszlopkifejezésként
- Szűrés allekérdezésekkel
- Az allekérdezések használati területei
- Példák
- Összefoglalás
- Önálló feladatok

Az előző három fejezetben megmutattuk, hogyan dolgozzunk egynél több táblából származó adatokkal. Az összes idáig vett módszer az információk részhalmazának összekapcsolására összpontosított – egy vagy több oszlop és egy vagy több sor egy teljes táblából, vagy egy a FROM záradékba ágyazott lekérdezés. Ezenkívül felfedeztük a halmazok összekapcsolását is a UNION művelettel. Ebben a fejezetben megmutatjuk, hogyan lehet egy táblából vagy lekérdezésből egy oszlopot kinyerni és azt egy SELECT vagy WHERE értékkifejezéseként felhasználni.

Két fontos dolgot is meg kell tanulnunk ebben a fejezetben:

1. Az SQL-ben mindig több módja van annak, hogy egy adott problémát megoldjunk. Olyannyira, hogy ez a fejezet korábban már tárgyalt feladatokra mutat újabb megoldásokat.
2. Olyan összetett szűrőket alkothatunk, amelyek nem a FROM záradékban megadott táblákra támaszkodnak. Ez nagyon fontos, mert a WHERE záradékban használt allekérdezések jelentik az egyetlen módját annak, hogy helyes számú válaszsort kapjunk, ha kapcsolódó táblák szűrt tartalma alapján akarjuk egy tábla sorait megkapni.

Megjegyzés *Ez a fejezet haladó témaköröket tárgyal, és feltételezi, hogy elolvastuk és megértettük a 7., 8. és 9. fejezetet.*

Mi az az allekérdezés?

Egyszerűen fogalmazva, az *allekérdezés* egy olyan SELECT kifejezés, amelyet egy SELECT utasítás valamelyik részébe ágyazunk be, így kapva meg a teljes lekérdezést. Ebben a fejezetben pontosabban is meghatározzuk az allekérdezésket, és megmutatjuk, hogyan lehet azokat a FROM záradékon kívül használni. Az SQL-szabvány az allekérdezések három típusát határozza meg:

1. Sorallekérdezés – olyan beágyazott SELECT kifejezés, amely egynél több oszlopot és legfeljebb egy sort ad vissza.
2. Tábla-allekérdezés – olyan beágyazott SELECT kifejezés, amely egy vagy több oszlopot és nulla vagy több sort ad vissza.
3. Skaláris allekérdezés – olyan beágyazott SELECT kifejezés, amely pontosan egy oszlopot és legfeljebb egy sort ad vissza.

Sorallekérdezések

Alkottunk már olyan lekérdezéseket, amelyek egy SELECT utasítást ágyaztak be egy FROM záradékba, ami lehetővé tette a sorok szűrését még azelőtt, hogy az eredményt más táblákkal vagy lekérdezésekkel egyesítettük volna. (Ezt hívják tábla-allekérdezésnek, ahogy azt hamarosan látni fogjuk). A *sorallekérdezés* a SELECT utasítás egy olyan különleges formája, ami egynél több oszlopot, de csak egy sort ad vissza.

Az SQL-szabvány szerint a sorallekérdezéssel egy a szabványban *sorértékkonstruktornak* nevezett dolgot hozhatunk létre. A WHERE záradék létrehozásakor általában olyan keresési feltételt adunk meg, amely az egyik táblánk valamelyik oszlopát hasonlítja össze vagy egy másik oszloppal, vagy egy adott literális értékkel. Az SQL-szabvány ugyanakkor megengedi, hogy olyan keresési feltételt alkossunk, ami több értéket egy logikai sorként hasonlít össze egy másik, szintén egy logikai sorként kezelt értékhalmazzal (két sorértékkonstruktort). Az összehasonlítandó értékek listáját vagy zárójelezett listaként adhatjuk meg, vagy egy sorallekérdezéssel nyerünk ki egy sort valamelyik táblából. Sajnos azonban ezt az utasításformát nem sok kereskedelmi forgalomban lévő adatbáziskezelő támogatja.

Miért is lehet hasznos a sorallekérdezés? Példaként vegyünk egy Products (Termékek) nevű táblát, amelyben két külön mezőben tárolunk egy összetett alkatrész-azonosítót. Az azonosító első része (SKUClass) mondjuk az alkatrész jellegére utaló karaktereket tartalmaz, ami egy számítógép-alkatrészeket gyártó ügyfél esetében például CPU vagy DSK lehet, az azonosító második része (SKUNumber) pedig a termék azonosítója a termékcsoporton belül. Tegyük fel, hogy az összes DSK09775 vagy annál magasabb összetett azonosítójú alkatrészre vagyunk kíváncsiak. Íme egy példa, ami egy sorértékkonstruktort használó WHERE záradékkal oldja meg a feladatot:

```
SQL     SELECT SKUClass, SKUNumber, ProductName
        FROM Products
        WHERE
           (SKUClass, SKUNumber)
           >= ('DSK', 9775)
```

Az előző WHERE záradék olyan sorokat kér le, amelyekben az SKUClass és az SKUNumber együttese nagyobb, mint a DSK és a 9775 együttese. Az alábbi kérelem ezzel egyenértékű:

```
SQL     SELECT SKUClass, SKUNumber, ProductName
        FROM Products
        WHERE (SKUClass > 'DSK')
        OR ((SKUClass = 'DSK')
        AND (SKUNumber >= 9775))
```

Az összehasonlítás második felét lecserélhetnénk egy olyan SELECT utasításra, amely egy két oszlopból álló sort ad vissza – vagyis egy sorallekérdezésre (akár a WHERE záradékkal leszűkítve a választ egy sorra). A legtöbb adatbázis-kezelő azonban nem támogatja sem a sorértékkonstruktorokat, sem a sorallekérdezéseket, ezért ebben a fejezetben többé nem lesz róluk szó.

Tábla-allekérdezések

Álljunk meg egy pillanatra! Az előző három fejezetben nem volt már szó róla, hogyan ágyazhatunk be egy több sort és oszlopot visszaadó SELECT kifejezést egy FROM záradékba? De bizony volt, becsempésztük! Már az előző fejezetekben is használtunk tábla-allekérdezéseket, hogy egy lekérdezés FROM záradékába összetett eredményt rakhassunk. Most meglátjuk, hogyan lehet egy tábla-allekérdezést az IN állítás (predikátum) összehasonlító értékeinek forrásaként használni – ennek alapjait a 6. fejezetben tanultuk meg. Emellett megtanulunk még egy pár olyan új összehasonlító kulcsszót, amelyeket csak tábla-allekérdezésekben használunk.

Skaláris allekérdezések

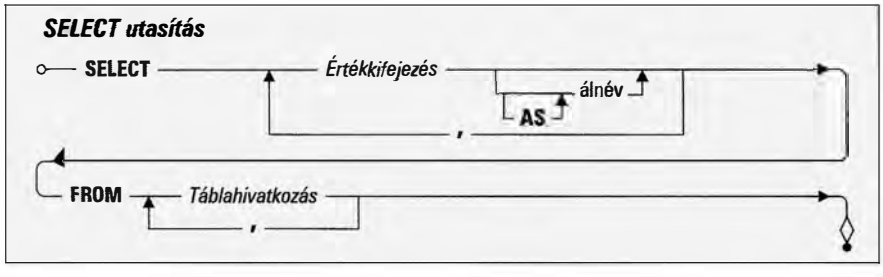
Arról is szó lesz ebben a fejezetben, hogyan használhatunk skaláris allekérdezéseket bárhol, ahol egyébként értékkifejezést használnánk. A skaláris allekérdezések segítségével egy oszlopot vagy számított kifejezést kaphatunk egy olyan táblából, amely nem feltétlenül szerepel a fő lekérdezés FROM záradékában. A skaláris allekérdezés által visszaadott értéket a SELECT záradéknak megadott oszloplistában vagy egy WHERE záradék összehasonlító értékeként is felhasználhatjuk.

Allekérdezések oszlopkifejezésként

Az 5. fejezetben sokat tanultunk arról, hogyan használhatunk kifejezéseket egy lekérdezés számított oszlopainak létrehozásához. Akkor nem mondtuk el, hogy a SELECT utasítás egy különleges fajtáját, az allekérdezést is használhatjuk arra, hogy egy másik táblából adatokat szerezzünk, még akkor is, ha az a tábla nem szerepel a FROM záradékban.

Utasításforma

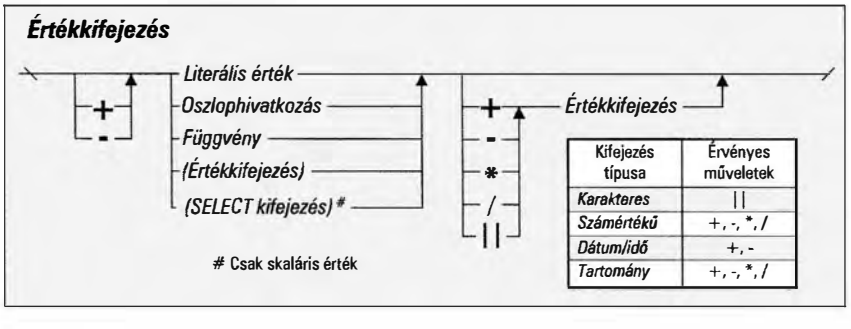
Térjünk vissza az alapokhoz, és nézzük meg a 11.1. ábrán a SELECT utasítás egyszerű alakját:



11.1. ábra

Egy egyszerű SELECT utasítás szintaxisdiagramja

Egyszerűnek tűnik, de valójában nem az, mert az értékkifejezés akár igen összetett is lehet. Az értékkifejezések összes lehetséges alkotóelemét a 11.2. ábrán láthatjuk.



11.2. ábra

Az értékkifejezések szintaxisdiagramja

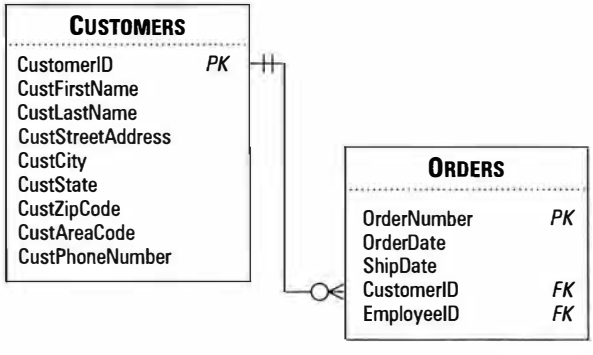
Az 5. fejezetben megmutattuk, hogyan lehet egyszerű értékkifejezéseket képezni literális értékekkel, oszlophivatkozásokkal és függvényekkel. Vegyük észre, hogy már a *SELECT kifejezés* is szerepel a listán. Eszerint a közvetlenül a SELECT kulcsszó után álló kifejezések listájában skaláris allekérdezést is használhatunk. Mint már említettük, a skaláris

allekérdezés olyan SELECT kifejezés, amely pontosan egy oszlopot és nem több mint egy sort ad vissza. Ez így is van rendjén, hiszen olyan helyre helyettesítjük be, ahol általában egy oszlopnevet vagy egy egyetlen oszloppal visszatérő kifejezést használnánk.

Ezen a ponton felmerülhet a kérdés: mire jó ez? Az így használt allekérdezéssel egy másik táblából vagy lekérdezésből foglalhatunk bele egy értéket a lekérdezésünk kimenetébe. Az allekérdezés FROM záradékának adatforrásaként használt táblának vagy lekérdezésnek nem kell szerepelnie a befoglaló lekérdezés FROM záradékában. Legtöbbször az allekérdezés WHERE záradékának megadott feltételekkel kell biztosítanunk, hogy az legfeljebb egy sort adjon vissza. Az allekérdezés ezen feltétele akár a külső lekérdezés által visszaadott értékre is hivatkozhat, hogy az adott sorhoz tartozó adatot kinyerje.

Nézzünk pár példát a Sales Orders mintaadatbázis Customers (Vásárlók) és Orders (Rendelések) tábláinak felhasználásával. A két tábla kapcsolatát a 11.3. ábra mutatja.

Alkossunk lekérdezést, amely kiírja egy adott nap összes rendelését, és egy allekérdezéssel kigyűjti a megfelelő vásárlók vezetékneveit a Customers táblából!



11.3. ábra
A Customers és Orders táblák

Megjegyzés *A fejezetben végig a 4. fejezetben bemutatott „Kérelem – Fordítás – Tisztázás – SQL" módszert alkalmazzuk, továbbá zárójelekkel jelöljük a Tisztázás részben az allekérdezéseket, és ahol lehet, behúzzuk őket, hogy jobban látszódjanak.*

„Mutasd meg az összes 2007. október 3-án leszállított rendelést és mindegyik rendeléshez a megfelelő vásárló vezetéknevét!"

Fordítás Select order number, order date, ship date, and also select
the related customer last name out of the customers table from
the orders table where ship date is October 3, 2007
(Válaszd ki a rendelésszámot, a rendelés dátumát, a szállítás dátumát és
az ezekhez tartozó vásárló vezetéknevét a vásárlók táblájából, ha a szállítás
dátuma 2007. október 3.)

Tisztázás
```
Select order number, order date, ship date,
and also (select the related customer last name out of the
from customers table)
from the orders table
where ship date is = October 3, 2007 '2007-10-03'
```

SQL
```
SELECT Orders.OrderNumber, Orders.OrderDate,
   Orders.ShipDate,
   (SELECT Customers.CustLastName
   FROM Customers
   WHERE Customers.CustomerID =
      Orders.CustomerID)
FROM Orders
WHERE Orders.ShipDate = '2007-10-03'
```

Vegyük észre, hogy az allekérdezésben a CustomerID (vásárlóazonosító) értékét korlátozni kellett az Orders táblából vett sorok CustomerID értékére, máskülönben az allekérdezésben a Customers tábla *összes* sorát megkaptuk volna. Ne felejtsük el, hogy itt egy skaláris allekérdezésnek – vagyis egy olyan lekérdezésnek, ami csak egy értéket ad vissza egy sorból – kell szerepelnie, ezért tennünk kell valamit, hogy legfeljebb egy sort kapjunk vissza. Mivel a CustomerID a Customers tábla elsődleges kulcsa, biztosak lehetünk benne, hogy az Orders tábla CustomerID oszlopával való összevetés pontosan egy sort eredményez.

Akik a 8. fejezetben tárgyalt INNER JOIN működési elvét megértették, most valószínűleg azon gondolkodnak, miért akarná bárki a fent leírt módon megoldani ezt a feladatot ahelyett, hogy a külső lekérdezés FROM záradékában egy JOIN-nal összekapcsolná az Orders és Customers táblákat. A jelenlegi célunk azonban az, hogy egy egyszerű példán keresztül megmutassuk a kimeneti oszlopok allekérdezéssel való előállításának ***elvét***. Természetesen ezt a feladatot az alábbi, INNER JOIN-t használó lekérdezéssel célszerű megoldani:

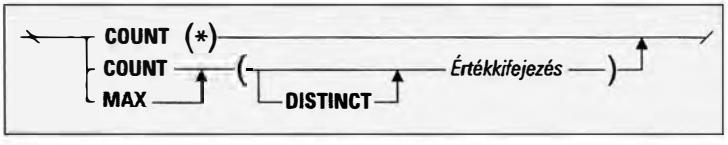
SQL
```
SELECT Orders.OrderNumber, Orders.OrderDate,
   Orders.ShipDate, Customers.CustLastName
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.OrderID
WHERE Orders.ShipDate = '2007-10-03'
```

Bevezetés az összesítő függvények használatába: a COUNT és a MAX

Most, hogy már értjük a kimeneti oszlopok allekérdezésekkel való előállításának lényegét, szélesítsük ki a látókörünket, és nézzük meg, hogyan lehet ez a lehetőség a hasznunkra. Először is vessünk egy pillantást egy pár összesítő függvényre (részletesen a következő fejezetben lesz róluk szó)!

Az SQL-szabvány számos olyan függvényt határoz meg, amelyek egy lekérdezésben értékeket számítanak ki. Ezek egy része – az összesítő függvények – egy eredményhalmazon belüli sorok egy csoportjából számít ki egyetlen értéket. Összesítő függvényekkel megszámlálhatjuk például a sorokat, megkereshetjük a legnagyobb és legkisebb értéket sorok egy csoportján belül, vagy kiszámíthatjuk az eredményhalmaz értékeinek vagy kifejezéseinek az átlagát, illetve összegét.

Nézzünk meg pár ilyen függvényt és hogy miként hasznosíthatók egy allekérdezésen belül. A 11.4. ábra a COUNT és a MAX függvényt mutatja, amelyek képesek egy SELECT záradékban kimeneti oszlopot előállítani.



11.4. ábra

A COUNT és MAX összesítő függvények használata

A COUNT-tal egy eredményhalmaz sorainak, illetve nem Null értékeinek a számát kapjuk meg. A sorok számának kiderítésére a COUNT(*) használatos. Ha a COUNT(*oszlop_neve*) segítségével egy bizonyos oszlopot adunk meg az eredményhalmazban, az adatbázis-kezelő rendszer a nem Null értékű sorokat számlálja meg, de a DISTINCT kulcsszó hozzáadásával az egyedi értékeket is megszámláltathatjuk. Hasonlóképp, a MAX egy oszlopon belüli legnagyobb érték visszaadására szolgál. Ha az értékkifejezés számot jelöl, a legmagasabb értéket kapjuk az oszlopból vagy a megadott kifejezésből. Ha az értékkifejezés karakteres adattípust ad vissza, a legnagyobb érték az adatbázis-kezelő jelsorrendjétől függ.

Oldjunk meg egy pár érdekes feladatot ezeknek a függvényeknek egy allekérdezésben való használatával:

„Írd ki az összes vásárló nevét és a rendeléseik számát!"

Fordítás Select customer first name, customer last name, and also select the count of orders from the orders table for this customer from the customers table (Válaszd ki a vásárló keresztnevét és vezetéknevét, és válaszd ki a rendelések számát a rendelések táblájából ehhez a vásárlóhoz a vásárlók táblájából.)

Tisztázás Select customer first name, customer last name, ~~and also~~
(select ~~the~~ count ~~of orders~~ (*) from ~~the~~ orders ~~table~~
~~for this customer~~ where orders.customer ID =
customers.customer ID)
from ~~the~~ customers ~~table~~

SQL
```
SELECT Customers.CustFirstName,
   Customers.CustLastName,
   (SELECT COUNT(*)
   FROM Orders
   WHERE Orders.CustomerID =
      Customers.CustomerID)
AS CountOfOrders
FROM Customers
```

A kimeneti oszlopként használt allekérdezések kezdenek érdekesek lenni! A IV. részben majd még többet tudunk meg az összesítő függvények kreatív használatáról, de ha nem akarunk többet, mint a kapcsolódó sorokat megszámlálni, az allekérdezés jó módszer. Sőt, ha semmi mást nem szeretnénk, mint a vásárló nevét és a rendelések számát, akkor nagyjából ez az egyetlen megoldás. Ha az Orders táblát hozzáadjuk a fő lekérdezés FROM záradékához (`FROM Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID`), több sort kapunk minden olyan vásárló esetében, akinek egynél több megrendelése volt. A 13. fejezetben majd tanulunk egy másik módszert, amelyhez a sorokat a vásárló neve szerint csoportosítjuk.

Nézzünk egy másik érdekes feladatot, amelyhez egy másik összesítő függvényt, a MAX-ot használjuk fel:

„Mutasd meg a vásárlók listáját és az utolsó napot, amikor rendelést adtak fel!"

Fordítás Select customer first name, customer last name, and also select the highest order date from the orders table for this customer from the customers table (Válaszd ki a vásárlók keresztnevét és vezetéknevét a vásárlók táblájából, illetve a hozzájuk tartozó legnagyobb rendelési dátumot a rendelések táblájából.)

Tisztázás Select customer first name, customer last name, ~~and also~~
(select ~~the highest~~ max(order date)
from ~~the~~ orders ~~table~~
~~for this customer~~ where orders.customer ID =
customers.customer ID)
from ~~the~~ customers ~~table~~

SQL
```
SELECT Customers.CustFirstName,
   Customers.CustLastName,
   (SELECT MAX(OrderDate)
   FROM Orders
   WHERE Orders.CustomerID =
      Customers.CustomerID)
AS LastOrderDate
FROM Customers
```

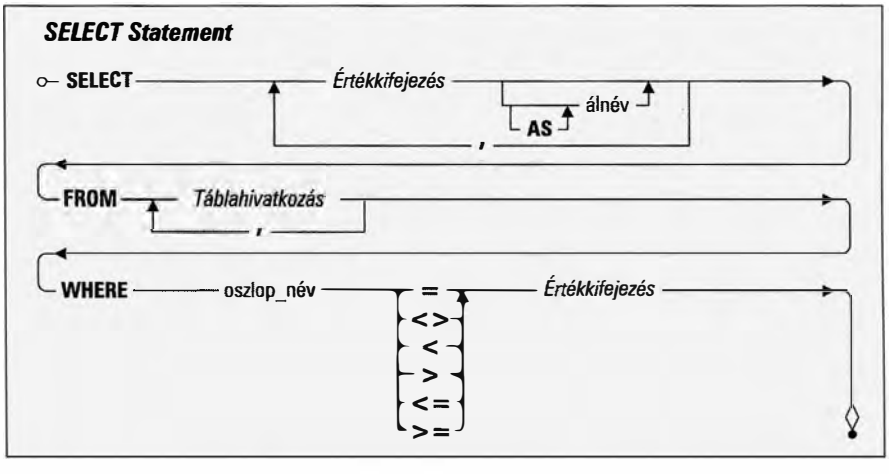
Mint azt sejtjük, a MAX ily módon használva remekül alkalmazható bármely kapcsolódó tábla legmagasabb értékének megkeresésére. Később, a fejezet *Példák* részében ezeknek a függvényeknek más alkalmazásait is megmutatjuk.

Szűrés allekérdezésekkel

A 6. fejezetben megtanultuk, hogyan szűrjük a kapott adatokat egy WHERE záradék hozzáadásával. Arról is volt szó, hogyan lehet egyszerű vagy összetett összehasonlításokkal csak a kívánt sorokat megkapni az eredményhalmazban. Most ezekre a képességekre építve megmutatjuk, miként lehet az allekérdezéseket összehasonlítások tagjaiként használva még összetettebb szűrést végezni.

Utasításforma

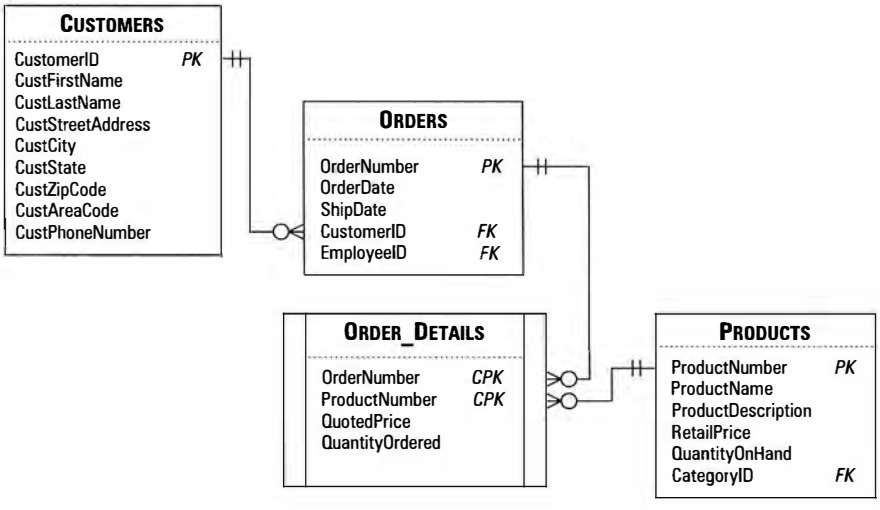
Térjünk vissza a 11.1. ábra SELECT utasításához, és nézzük meg azt az utasításformát, amelyet egy egyszerű összehasonlító állítást tartalmazó WHERE záradékkal ellátott lekérdezés felépítéséhez használhatunk! Az egyszerűsített diagram a 11.5. ábrán látható.



11.5. ábra

Eredmények szűrése egyszerű összehasonlító állítással

A 11.2. ábráról emlékezhetünk rá, hogy az értékkifejezés allekérdezés is lehet. A 11.5. ábra egyszerű példájában az értékkifejezést egy oszloppal hasonlítjuk össze, így az értékkifejezésnek egy értéke lehet – azaz egy skaláris allekérdezést kapunk, ami pontosan egy oszlopot és legfeljebb egy sort ad vissza. Oldjunk meg egy olyan egyszerűbb feladatot, ami egy allekérdezés által visszaadott értékkel való összehasonlítást tesz szükségessé. Ebben a példában a vásárlói rendelések összes részletére leszünk kíváncsiak – pontosabban minden vásárlónak csak az *utolsó* rendelésére. A szükséges táblákat a 11.6. ábra mutatja.



11.6. ábra

A rendelések összes részletének kigyűjtéséhez szükséges táblák

„Írd ki a vásárlókat és a legutolsó rendelésük minden részletét!"

Fordítás Select customer first name, customer last name, order number, order date, product number,product name, and quantity ordered from the customers table joined with the orders table on customer ID in the customers table equals customer ID in the orders table, then joined with the order details table on order number in the orders table equals order number in the order details table, and then joined with the products table on product number in the products table equals product number in the order details table where the order date equals the maximum order date from the orders table for this customer

(Válaszd ki a vásárló keresztnevét és vezetéknevét, a rendelés számát és dátumát, a termék számát és nevét, valamint a rendelt mennyiséget a vásárlók táblájából, amelyet összekapcsoltunk a rendelések táblájával a rendelések táblájában levő vásárlóazonosító és a vásárlók táblájában található vásárlóazonosító szerint, majd ezt összekapcsoltuk a rendelések részleteit tartalmazó táblával a rendelések táblájában levő rendelésszám és a rendelések részleteit tartalmazó tábla rendelésszámai alapján, majd ezt összekapcsoltuk a termékek táblájával a termékek táblájában levő termékszám és a rendelések részleteit tartalmazó tábla termékszámainak egyezése alapján, ha a rendelés dátuma egyenlő a rendelési dátumok maximumával a rendelések táblájában az adott vásárló esetében.)

Tisztázás

Select customer first name, customer last name,
order number, order date,product number,
product name,~~and~~ quantity ordered
from ~~the~~ customers ~~table~~
inner ~~joined with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table equals~~
= orders.customer ID ~~in the orders table, then~~
inner ~~joined with the~~ order details ~~table~~
on orders.order number ~~in the orders table equals~~
= order_details.order number ~~in the order details table,~~
~~and then~~ inner ~~joined with the~~ products ~~table~~
on products.product number ~~in the products table equals~~
= order_details.product number ~~in the order details table~~
where ~~the~~ order date ~~equals~~ =
(select ~~the~~ max~~imum~~ (order date) from ~~the~~ orders ~~table~~
~~for this customer~~ where orders.customer ID
= customers.customer ID)

SQL

```
SELECT Customers.CustFirstName,
   Customers.CustLastName, Orders.OrderNumber,
   Orders.OrderDate,
   Order_Details.ProductNumber,
   Products.ProductName,
   Order_Details.QuantityOrdered
FROM ((Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Orders.OrderDate =
   (SELECT MAX(OrderDate)
   FROM Orders AS O2
   WHERE O2.CustomerID = Customers.CustomerID)
```

Feltűnt, hogy az Orders táblára való második hivatkozásnak (az allekérdezésben szereplő Orders táblának) álnevet adtunk? Még ha ki is hagytuk volna az álnevet, sok adatbázis-kezelő rendszer akkor is felismerte volna, hogy az allekérdezésben szereplő Orders táblára gondolunk. Valójában az SQL-szabvány úgy rendelkezik, hogy a nem minősített hivatkozásokat a legbelső lekérdezéstől kezdve kell feloldani. Ennek ellenére mi megadtunk egy álnevet, hogy teljesen világos legyen, hogy az allekérdezés WHERE záradékában hivatkozott Orders tábla valójában az allekérdezés FROM záradékában szereplő tábla. Ha ezt a gyakor-

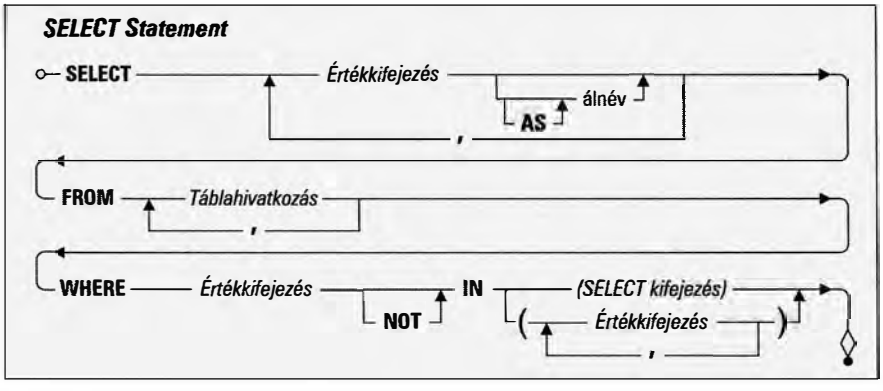
latot követjük, a lekérdezéseink sokkal érthetőbbek lesznek – akár magunknak, amikor hónapok múltán újra visszatérünk hozzájuk, akár másnak, akinek meg kell értenie, hogy mit csinál a lekérdezés.

Az allekérdezések állításaiban használt különleges kulcsszavak

Az SQL-szabvány meghatároz néhány, az allekérdezések WHERE záradékában használható kulcsszót.

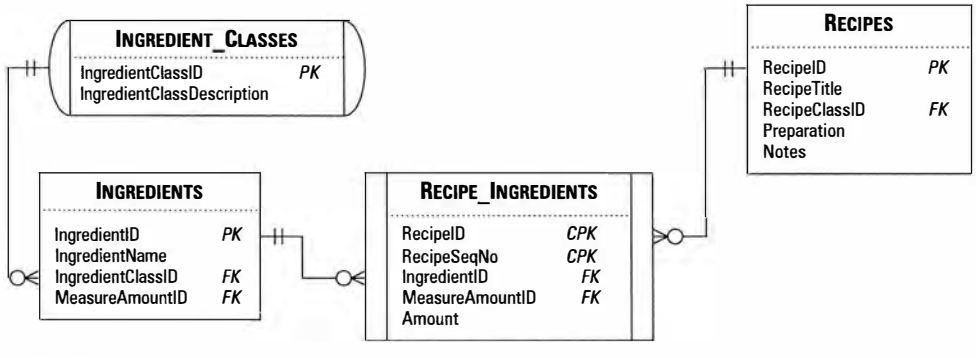
Halmaztagság: IN

A 6. fejezetben megtanultuk, hogyan hasonlíthatunk össze egy a WHERE záradékban elhelyezett IN kulcsszóval egy oszlopot vagy kifejezést és értékek egy listáját. Most már tudjuk, hogy az IN listájában szereplő értékkifejezés *lehet* skaláris allekérdezés is. Mi lenne, ha a teljes listát allekérdezéssel hoznánk létre? Ahogy a 11.7. ábra mutatja, ennek nincs akadálya.



11.7. ábra

Allekérdezés használata IN állításban



11.8. ábra

A receptek és hozzávalók kiírásához szükséges táblák

Ebben az esetben egy tábla-allekérdezést használhatunk, ami egy oszlopot és annyi sort ad vissza, amennyi a listához szükséges. Használjuk fel a Recipes mintaadatbázist egy példához; a szükséges táblákat a 11.8. ábra mutatja.

Tegyük fel, hogy olyasvalakit látunk vendégül vacsorára, aki imádja a tengeri étkeket. Habár tudjuk, hogy van egy pár tengeri ételreceptünk, nem ismerjük az adatbázisban szereplő összes hozzávaló nevét. Azt viszont tudjuk, hogy van egy Seafood (Tengeri hozzávalók) értékű IngredientClassDescription (Hozzávalóosztály-leírás), ezért összekapcsolhatjuk az összes táblát, és szűrhetünk az IngredientClassDescription-re – vagy lehetünk kreatívak, és használhatjuk az allekérdezéseket és az IN állítást:

„Írd ki az összes olyan receptet, amelynek van tengeri hozzávalója!"

Fordítás Select recipe title from the recipes table where the recipe ID is in the selection of recipe IDs from the recipe ingredients table where the ingredient ID is in the selection of ingredient IDs from the ingredients table joined with the ingredient classes table on ingredient class ID in the ingredients table matches ingredient class ID in the ingredient classes table where ingredient class description is 'seafood'
(Válaszd ki azoknak a recepteknek a címét a receptek táblájából, amelyeknek a receptazonosítója szerepel a hozzávalók táblájából kiválasztott receptazonosítók között, ha a hozzávaló azonosítója szerepel a hozzávalók táblájából kiválasztott hozzávaló-azonosítók között, mely táblát összekapcsoltuk a hozzávalóosztályok táblájával a hozzávalók táblájában levő hozzávaló-azonosítók és a hozzávalóosztályok táblájában tárolt hozzávaló-azonosítók egyezése alapján, ha a hozzávalóosztály leírása „seafood", vagyis „tengeri hozzávaló".)

Tisztázás Select recipe title from ~~the~~ recipes ~~table~~
where ~~the~~ recipe ID ~~is~~ in ~~the~~
(select~~ion of~~ recipe ID~~s~~
from ~~the~~ recipe ingredients ~~table~~
where ~~the~~ ingredient ID ~~is~~ in ~~the~~
(select~~ion of~~ ingredient ID~~s~~
from ~~the~~ ingredients ~~table~~
inner join~~ed with the~~ ingredient classes ~~table~~
on ingredients.ingredient class ID
~~in the ingredients table matches~~
= ingredient_classes.ingredient class ID
~~in the ingredient classes table~~
where ingredient class description ~~is~~ = 'seafood'))

SQL
```
SELECT RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID IN
```

```
    (SELECT RecipeID
    FROM Recipe_Ingredients
    WHERE Recipe_Ingredients.IngredientID IN
       (SELECT IngredientID
       FROM Ingredients
       INNER JOIN Ingredient_Classes
       ON Ingredients.IngredientClassID =
          Ingredient_Classes.IngredientClassID
       WHERE
       Ingredient_Classes.IngredientClassDescription
          = 'Seafood'))
```

Gondoltuk volna, hogy allekérdezést is lehet allekérdezésbe tenni? Még egy szinttel tovább mehettünk volna, ha az INNER JOIN-t kiszedjük a második allekérdezésből. A második allekérdezést az alábbi módon is megfogalmazhattuk volna:

SQL

```
(SELECT IngredientID
FROM Ingredients
WHERE Ingredients.IngredientClassID IN
   (SELECT IngredientClassID
   FROM Ingredient_Classes
   WHERE
   Ingredient_Classes.IngredientClassDescription
      = 'Seafood'))
```

Ez azonban már olyan, mintha ágyúval lőnénk verébre, mert az IN záradék IN záradékba ágyazása nehezebben olvashatóvá teszi a lekérdezést. Az előző példában csak azért tettük, hogy megmutassuk, meg *lehet* csinálni. Meg kell ismételnünk azonban, hogy csak azért mert *lehet*, még nem biztos, hogy *kell*. Megegyezhetünk abban, hogy könnyebben látjuk, mi folyik, ha csak egy IN állítás és egy összetettebb JOIN van az allekérdezésben. Itt egy másik megoldás, ami ezt a módszert alkalmazza:

SQL

```
SELECT RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID IN
   (SELECT RecipeID
   FROM (Recipe_Ingredients
   INNER JOIN Ingredients
   ON Recipe_Ingredients.IngredientID =
      Ingredients.IngredientID)
   INNER JOIN Ingredient_Classes
   ON Ingredients.IngredientClassID =
      Ingredient_Classes.IngredientClassID
```

```
WHERE
Ingredient_Classes.IngredientClassDescription
   = 'Seafood')
```

Felmerülhet a kérdés: „Miért szenvedünk ezzel? Miért nem csinálunk egy összetett JOIN-t a külső lekérdezésben és kész?" Azért, mert rossz választ kapnánk! A visszakapott sorok tulajdonképpen mind a Recipes tábla sorai a tengeri ételes receptekre, de egy sort többször is visszakaphatunk. Próbáljuk meg az allekérdezés nélkül megoldani, hogy lássuk, miért kapunk többször szereplő sorokat:

SQL

```
SELECT RecipeTitle
FROM ((Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Recipe_Ingredients.IngredientID =
   Ingredients.IngredientID)
INNER JOIN Ingredient_Classes
ON Ingredients.IngredientClassID =
Ingredient_Classes.IngredientClassID
WHERE
Ingredient_Classes.IngredientClassDescription
   = 'Seafood')
```

Ha újból megnézzük a 11.8. ábrát, láthatjuk, hogy a Recipe_Ingredients (Recepthozzávalók) táblában a Recipes tábla minden sorához több sor is tartozhat. A FROM záradék által megadott eredményhalmaz legalább annyi sort tartalmaz, mint amennyi a Recipe_Ingredients-ben van, a RecipeTitle (Receptcím) oszlop értékeinek sokszoros ismétlése mellett. Még ha hozzá is adunk egy szűrőt, ami az eredményeket a Seafood fajtára szűkíti, akkor is több sort kapunk az összes olyan receptre, amelynek egynél több tengeri összetevője van.

Igen, használhatnánk a DISTINCT kulcsszót, de ekkor az adatbázis-kezelő kényszerül többletmunkára, hogy eltávolítsa az ismétlődéseket. Ha ezt a DISTINCT-et használó lekérdezést nézettáblaként mentjük, majd frissíteni próbáljuk a nézettábla adatait, azt tapasztaljuk, hogy az nem frissíthető, mert a DISTINCT elfedi a sorok egyediségét, és az adatbázis-kezelő nem tudja eldönteni, hogy melyik sort frissítse.

Az allekérdezések eme módszere akkor válik igazán fontossá, ha a receptek nevénél többet akarunk kiíratni. Tegyük fel például, hogy az *összes* hozzávalóját ki szeretnénk íratni *bármely* olyan receptnek, amelynek van tengeri hozzávalója. Ha összetett JOIN-t hasz-

nálunk a külső lekérdezésben és a Seafood hozzávalóosztályra szűrünk, csak a tengeri hozzávalókat kapjuk meg, a receptek többi hozzávalóját nem. Nézzünk egy valamivel bonyolultabb kérelmet:

„Sorold fel azokat a recepteket a hozzávalóikkal együtt, amelyekhez kell tengeri hozzávaló!"

Fordítás Select recipe title and ingredient name from the recipes table joined with the recipe ingredients table on recipe ID in the recipes table equals recipe ID in the recipe ingredients table,and then joined with the ingredients table on ingredient ID in the ingredients table equals ingredient ID in the recipe ingredients table where the recipe ID is in the selection of recipe IDs from the recipe ingredients table joined with the ingredients table on ingredient ID in the recipe ingredients table equals ingredient ID in the ingredients table, and then joined with the ingredient classes table on ingredient class ID in the ingredients table equals ingredient class ID in the ingredient classes table where ingredient class description is 'seafood'
(Válaszd ki azoknak a recepteknek a nevét és a hozzávalóik nevét a receptek táblájából és az azzal a receptazonosítók szerint összekapcsolt recepthozzávalók táblájából, amelyet összekapcsoltunk a hozzávalók táblájával a hozzávalók táblájában levő hozzávaló-azonosító és a recepthozzávalók táblájában levő hozzávaló-azonosító szerint, ha a recept azonosítója szerepel a receptazonosítók között a recepthozzávalók táblájában, amelyet összekapcsoltunk a hozzávalók táblájával a hozzávaló-azonosító szerint, majd ezt összekapcsoltuk a hozzávalóosztályok táblájával a hozzávalók táblájában levő hozzávaló-azonosítók és a hozzávalóosztályok táblájában tárolt hozzávaló-azonosítók szerint, ha a hozzávalóosztály leírása „seafood".)

Tisztázás Select recipe title, ~~and~~ ingredient name
from ~~the~~ recipes ~~table~~
inner ~~joined with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table equals~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table, and then~~ inner ~~joined with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table equals~~
= recipe_ingredients.ingredient ID
~~in the recipe ingredients table~~
where ~~the~~ recipe ID ~~is~~ in ~~the~~
(select~~ion of~~ recipe ID~~s~~
from ~~the~~ recipe ingredients ~~table~~
inner ~~joined with the~~ ingredients ~~table~~
on recipe_ingredients.ingredient ID

in the recipe ingredients table equals
= ingredients.ingredient ID in the ingredients table, and then
inner joined with the ingredient classes table
on ingredients.ingredient class ID
in the ingredients table equals
= ingredient_classes.ingredient class ID
in the ingredient classes table
where ingredient class description is = 'seafood')

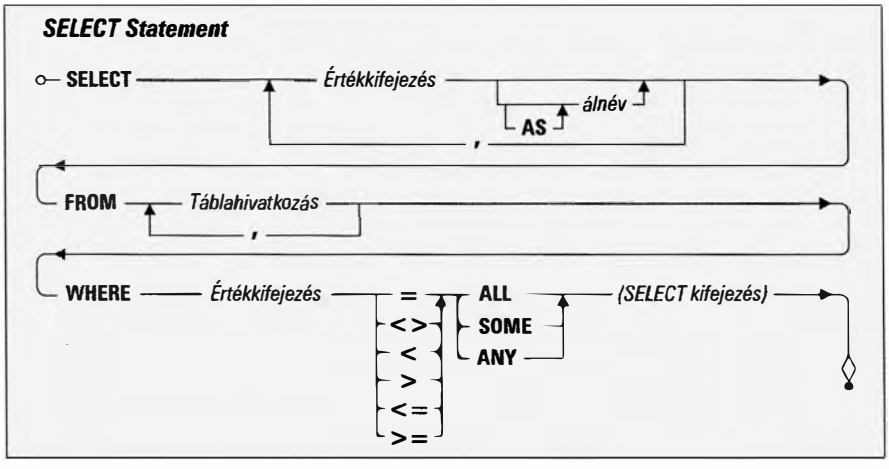
SQL

```
SELECT Recipes.RecipeTitle,
   Ingredients.IngredientName
FROM (Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
WHERE Recipes.RecipeID IN
   (SELECT RecipeID
   FROM (Recipe_Ingredients
INNER JOIN Ingredients
ON Recipe_Ingredients.IngredientID =
   Ingredients.IngredientID)
INNER JOIN Ingredient_Classes
ON Ingredients.IngredientClassID =
   Ingredient_Classes.IngredientClassID
WHERE
Ingredient_Classes.IngredientClassDescription
   = 'Seafood')
```

A dolog nyitja, hogy az összetett INNER JOIN a lekérdezés fő részében az *összes* hozzávalót kigyűjti a kiválasztott receptekhez, az összetett allekérdezés pedig csak a tengeri receptekhez tartozó receptazonosítót adja vissza. Úgy fest, mintha kétszer hajtanánk végre egy összetett JOIN-t – őrültség, de van benne rendszer!

Mennyiségi állítások: ALL, SOME és ANY

Amint azt az imént láttuk, az IN állítással egy oszlopot vagy kifejezést hasonlíthatunk össze egy listával, hogy megtudjuk, benne van-e a listában az adott oszlop vagy kifejezés, más szóval az oszlop vagy kifejezés *egyenlő-e* a lista egy tagjával. Ha arra vagyunk kíváncsiak, hogy az oszlop vagy kifejezés nagyobb vagy kisebb-e, mint a lista elemei közül bármelyik, az összes, vagy néhány, a ***mennyiségi állításokat*** használhatjuk erre a célra. Az utasításformát a 11.9. ábra mutatja.



11.9. ábra

Mennyiségi állítások használata egy SELECT utasításban

Ebben az esetben a SELECT kifejezés egy tábla-allekérdezés kell legyen, amely pontosan egy oszlopot és nulla vagy több sort ad vissza. Ha az allekérdezés egynél több sort ad, a sorok értékei egy listát alkotnak. Mint látható, ez az állítás egy összehasonlító műveletet egyesít egy kulcsszóval, ami megmondja az adatbázis-kezelőnek, hogyan alkalmazza a műveletet a lista elemeire. Az ALL kulcsszó használatakor az összehasonlításnak az allekérdezés által visszaadott összes értékre igaznak kell lennie. A SOME vagy ANY kulcsszó használata esetén elég, ha a lista egy elemére igaz az összehasonlítás.

Ha belegondolunk, ha az allekérdezés több sort ad vissza, az = ALL mindig hamis lesz, kivéve ha az allekérdezéstől kapott összes érték egyforma, és az összehasonlítás bal oldalán álló értékkifejezés az összessel megegyezik. Ugyanezen logika alapján azt gondolhatnánk, hogy a <> ANY mindig hamis lesz, ha a bal oldali értékkifejezés ténylegesen *egyenlő* bármelyik értékkel. Valójában az SQL-szabvány ugyanúgy kezeli a SOME-ot és az ANY-t. Ha tehát azt mondjuk, hogy <> SOME vagy <> ANY, akkor az állítás igaz, ha a bal oldali értékkifejezés nem egyenlő legalább egy a listában szereplő értékkel. Egy másik zavarba ejtő pont, hogy ha az allekérdezés nem ad vissza sorokat, akkor bármely összehasonlítás igaz, amelyben az ALL található, és bármelyik hamis, amelyben a SOME vagy az ANY szerepel.

Nézzünk meg néhány kérelmet, hogy működés közben lássuk a mennyiségi állításokat. Először oldjunk meg egy feladatot a Recipes adatbázisban (a 11.8. ábra segít felidézni a használt táblákat):

„Mutasd meg a marhahúst vagy fokhagymát tartalmazó recepteket!"

Fordítás Select recipe title from the recipes table where recipe ID is in the selection of recipe IDs from the recipe ingredients table

where ingredient ID equals any of the selection of ingredient
IDs from the ingredients table where ingredient name is 'beef' or 'garlic'
(Válaszd ki azoknak a recepteknek a nevét a receptek táblájából, amelyeknél a recept azonosítója szerepel a recepthozzávalók táblájából kiválasztott receptazonosítók között, ha a hozzávaló azonosítója megegyezik bármelyik kiválasztott hozzávaló azonosítójával a hozzávalók táblájából, ahol a hozzávaló neve „beef" vagy „garlic", vagyis „marhahús" vagy „fokhagyma".)

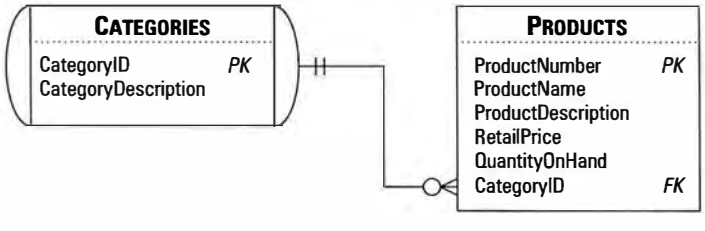
Tisztázás

Select recipe title from ~~the~~ recipes ~~table~~
where recipe ID ~~is~~ in ~~the~~
(selection ~~of~~ recipe IDs from ~~the~~ recipe ingredients ~~table~~
where ingredient ID ~~equals~~ = any of ~~the~~
(selection ~~of~~ ingredient ID~~s~~ from ~~the~~ ingredients ~~table~~
where ingredient name ~~is~~ in 'beef' ~~or~~ 'garlic'))

SQL

```
SELECT Recipes.RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID IN
   (SELECT Recipe_Ingredients.RecipeID
   FROM Recipe_Ingredients
WHERE Recipe_Ingredients.IngredientID = ANY
   (SELECT Ingredients.IngredientID
   FROM Ingredients
   WHERE Ingredients.IngredientName
   IN ('Beef', 'Garlic')))
```

Ha olyan érzésünk támadna, hogy az = ANY helyett az IN-t is használhatnánk, igazunk van. Egy JOIN-t is létrehozhattunk volna a Recipe_Ingredients és az Ingredients között az első allekérdezésben, hogy megkapjuk a RecipeID-k listáját. Ahogy a fejezet elején állítottuk, majdnem minden feladatot többféleképpen is meg lehet oldani az SQL-ben, a mennyiségi állítások azonban néha olvashatóbbá teszik a kérelmet.



11.10. ábra

A Categories és a Products táblák kapcsolata

Oldjunk most meg egy bonyolultabb feladatot, hogy lássuk a mennyiségi állítások valódi erejét. Ez a példa a Sales Orders mintaadatbázist használja, a 11.10. ábra pedig a felhasznált táblákat mutatja meg.

„Keresd meg az összes kiegészítőt, ami drágább, mint bármelyik ruhadarab!"

Fordítás Select product name and retail price from the products table joined with the categories table on category ID in the products table matches category ID in the categories table where category description is 'accessories' and retail price is greater than all the selection of retail price from the products table joined with the categories table on category ID in the products table matches category ID in the categories table where category name is 'clothing'
(Válaszd ki a termék nevét és fogyasztói árát a termékek táblájából, amelyet összekapcsoltunk a termékcsoportok táblájával a termékcsoport-azonosítók szerint, ha a termékcsoport leírása „accessories", vagyis „kiegészítő", és a fogyasztói ára nagyobb, mint az összes fogyasztói ár, amit kiválasztottunk a termékek táblájából, amelyet összekapcsoltunk a termékcsoportok táblájával a termékcsoport-azonosítók szerint, ha a termékcsoport neve „clothing", vagyis „ruhadarab".)

Tisztázás Select product name ~~and~~ retail price
from ~~the~~ products ~~table~~
inner ~~joined with the~~ categories ~~table~~
on products.category ID ~~in the products table~~
~~matches~~ = categories.category ID ~~in the categories table~~
where category description ~~is~~ = 'accessories'
and retail price ~~is greater than~~ > all ~~the~~
(select~~ion of~~ retail price
from ~~the~~ products ~~table~~
inner ~~joined with the~~ categories ~~table~~
on products.category ID ~~in the products table~~
~~matches~~ = categories.category ID ~~in the categories table~~
where category name ~~is~~ = 'clothing')

SQL
```
SELECT Products.ProductName,
   Products.RetailPrice
FROM Products
INNER JOIN Categories
ON Products.CategoryID
   = Categories.CategoryID
WHERE Categories.CategoryDescription =
   'Accessories'
AND Products.RetailPrice > ALL
   (SELECT Products.RetailPrice
```

```
FROM Products
INNER JOIN Categories
ON Products.CategoryID =
   Categories.CategoryID
WHERE Categories.CategoryDescription =
   'Clothing')
```

Mi történik itt? Az allekérdezés kinyeri az összes ruhadarab árát, majd a külső lekérdezés kiírja az összes olyan kiegészítőt, amelynek magasabb az ára, mint *bármelyik* ár a ruhadarabok allekérdezésében. Vegyük észre, hogy úgy is megoldhatnánk a feladatot, ha azt a RetailPrice-t (Fogyasztói ár) keresnénk, ami magasabb, mint az allekérdezésben talált árak MAX értéke, de a lényeg itt az ALL használatának bemutatása volt.

Létezést kifejező állítások: EXISTS

A halmaztagsági (IN) és a mennyiségi (SOME, ALL, ANY) állítások egy értékkifejezéssel való összehasonlítást végeznek – általában egy külső lekérdezés FROM záradékában megadott oszloppal. Néha hasznos egyszerűen azt tudni, hogy létezik-e az allekérdezés által visszaadott eredményhalmazhoz kapcsolódó sor. A 8. fejezetben megmutattuk az AND-problémák összetett INNER JOIN-nal való megoldását. Az EXISTS is használható ugyanezen problémák megoldására. Nézzünk meg egy olyan példát, amit már megoldottunk a 8. fejezetben:

„Keresd meg az összes vásárlót, aki biciklit és bukósisakot is rendelt!"

Fordítás Select customer ID, customer first name, and customer last name from the customers table where there exists some row from the orders table joined with the order details table on order ID in the orders table equals order ID in the order details table, and then joined with the products table on product ID in the products table equals product ID in the order details table where product name contains 'Bike' and the orders table customer ID equals the customers table customer ID, and there also exists some row from the orders table joined with the order details table on order ID in the orders table equals order ID in the order details table, and then joined with the products table on product ID in the products table equals product ID in the order details table where product name contains 'Helmet' and the orders table customer ID equals the customers table customer ID
(Válaszd ki a vásárló azonosítóját, keresztnevét és vezetéknevét a vásárlók táblájából, ha létezik hozzá sor a rendelések táblájában, amelyet összekapcsoltunk a rendelések részleteit tartalmazó táblával a rendelésazonosítók szerint, majd összekapcsoltuk a termékek táblájával a termékazonosítók szerint, ha a termék neve tartalmazza a „Bike", vagyis „Kerékpár" szót, és a rendelések táblájában levő vásárlóazonosító egyenlő a vásárlók táblája-

ban levő vásárlóazonosítóval, és ha létezik hozzá sor a rendelések táblájában, amelyet összekapcsoltunk a rendelések részleteit tartalmazó táblával a rendelésazonosítók alapján, majd összekapcsoltuk a termékek táblájával a termékazonosítók szerint, ha a termék neve tartalmazza a „Helmet", vagyis „Bukósisak" szót, és a rendelések táblájában levő vásárlóazonosító egyenlő a vásárlók táblájának vásárlóazonosítójával.)

Tisztázás

Select customer ID, customer first name,~~and~~ customer last name
from ~~the~~ customers ~~table~~
where ~~there~~ exists ~~some row~~
(select * from ~~the~~ orders ~~table~~
inner ~~joined with the~~ order details ~~table~~
on orders.order ID ~~in the orders table equals~~
= order_details.order ID ~~in the order details table, and then~~
inner ~~joined with the~~ products ~~table~~
on products.product ID ~~in the products table equals~~
= order_details.product ID ~~in the order details table~~
where product name ~~contains~~ LIKE '%Bike'
and ~~the~~ orders ~~table~~ customer ID ~~equals~~ =
~~the~~ customers ~~table~~ customer ID)~~;~~
and ~~there also~~ exists ~~some row~~
(select * from ~~the~~ orders ~~table~~
inner ~~joined with the~~ order details ~~table~~
on orders.order ID ~~in the orders table equals~~
= order_details.order ID ~~in the order details table, and then~~
inner ~~joined with the~~ products ~~table~~
on products.product ID ~~in the products table equals~~
= order_details.product ID ~~in the order details table~~
where product name ~~contains~~ LIKE '%Helmet'
and ~~the~~ orders ~~table~~ customer ID ~~equals~~ =
~~the~~ customers ~~table~~ customer ID)

SQL

```
SELECT Customers.CustomerID,
   Customers.CustFirstName,
   Customers.CustLastName
FROM Customers
WHERE EXISTS
   (SELECT *
   FROM (Orders
   INNER JOIN Order_Details
   ON Orders.OrderNumber =
      Order_Details.OrderNumber)
   INNER JOIN Products
   ON Products.ProductNumber =
      Order_Details.ProductNumber
```

```
    WHERE Products.ProductName LIKE '%Bike'
    AND Orders.CustomerID =
       Customers.CustomerID)
AND EXISTS
    (SELECT *
    FROM (Orders
    INNER JOIN Order_Details
    ON Orders.OrderNumber =
       Order_Details.OrderNumber)
    INNER JOIN Products
    ON Products.ProductNumber =
       Order_Details.ProductNumber
    WHERE Products.ProductName LIKE '%Helmet'
    AND Orders.CustomerID =
       Customers.CustomerID)
```

Megjegyzés *Tudjuk, hogy a mintaadatbázisban az összes kerékpár megnevezése tartalmazza a terméknév végén a Bike szót. Van az adatbázisban egy Viscount C-500 Wireless Bike Computer megnevezésű termék is, amely azonban nem kerékpár. Hasonlóképp tudjuk, hogy a mintaadatbázisban szereplő összes bukósisaknak a Helmet szó van a megnevezése végén, és egy másik név, a Dog Ear Helmet Mount Mirrors, középen tartalmazza a Helmet szót, viszont nem bukósisak.*

Azért választottuk a LIKE '%Bike' és a LIKE '%Helmet' használatát a LIKE '%Bike%' és a LIKE '%Helmet%' helyett, hogy biztosan a helyes termékeket válasszuk ki. Az igazság az, hogy a táblát kellett volna úgy tervezni, hogy a könnyebb szűrés érdekében rendelkezzék egy termékcsoport mezővel. Ahogy a 2. fejezetben már említettük, az ilyen apró tervezési hibák megnehezíthetik a feladat helyes megoldását.

Vegyük észre, hogy a FROM záradékban szereplő bármelyik tábla bármely oszlopának nevét használhatjuk az allekérdezés SELECT záradékában keresett oszlopként. Mi az összes oszlopot jelentő „*" rövidítést használtuk. Máshogy fogalmazva a lekérdezés így szól: „Add meg azokat a vásárlókat, akikhez létezik valamilyen kerékpárra szóló rendeléssor a rendelések részletei között, és akikhez szintén létezik valamilyen bukósisakra szóló rendeléssor a rendelések részletei között!". Mivel nem az OrderID (rendelésazonosító) oszloppal végeztünk összehasonlítást, nem érdekes, hogy a vásárló külön rendelésben rendelt-e biciklit és bukósisakot, vagy sem.

Megjegyzés *Ez a lekérdezés annyira érdekes, hogy ezt a megoldást Cust_Bikes_And_Helmets_EXISTS néven a mintaadatbázisba menteltük. Az eredeti, INNER JOIN-os megoldást Cust_Bikes_And_Helmets_JOIN néven találhatjuk meg.*

Az allekérdezések használati területei

Ezen a ponton már elég jól kell értenünk, hogyan használjuk az allekérdezéseket kimeneti oszlopok előállítására vagy egy WHERE záradékban végrehajtandó összetett összehasonlításra. Az allekérdezések széles körű felhasználásának bemutatására a legjobb mód, hogy mutatunk pár mintafeladatot, amelyeket allekérdezésekkel lehet megoldani, majd pedig adunk egy jó adag példát az *Önálló feladatok* részben.

Allekérdezések felépítése oszlopkifejezésként

Ahogy ebben a fejezetben már említettük, a kapcsolódó táblákból értékeket kiolvasó allekérdezéseknél feltehetőleg hatékonyabb a JOIN használata. Ha azonban az összesítő függvényekre gondolunk, egy függvény számítási eredményeit allekérdezésekkel megkapni mindjárt sokkal izgalmasabb. Az összesítő függvények ily módon való felhasználását a következő fejezetben tárgyaljuk. Addig is itt van pár feladat, amelyeket kimeneti oszlopot előállító allekérdezésekkel oldhatunk meg:

„Sorold fel a beszállítókat és az általuk nekünk eladott termékek számát!"
„Jelenítsd meg a termékeket és a dátumot, amikor a termékeket utoljára megrendelték!"
„Sorold fel az előadókat és minden előadó fellépéseinek a számát!"
„Jelenítsd meg az összes megrendelőt és mindegyikük utolsó lekötött rendezvényének a dátumát!"
„Sorold fel az összes tanárt és az általuk tartott órák számát!"
„Jelenítsd meg az összes tantárgyat és az egyes tárgyak hétfői óráinak a számát!"
„Jelenítsd meg az összes tekejátékost és az általuk lejátszott mérkőzések számát!"
„Jelenítsd meg az összes tekejátékost és az általuk elért legmagasabb pontszámot!"
„Írd ki az összes húsfélét és hogy hány receptben szerepelnek!"
„Mutasd meg a receptek fajtáit és hogy hány recept van az adott fajtából!"

Allekérdezések használata szűrőként

Most, hogy már ismerjük az allekérdezéseket, bővíthetjük az összetett lekérdezéseket megoldó eszköztárunkat. Ebben a fejezetben sok érdekes módját jártuk körül annak, hogyan használjunk egy WHERE záradékban allekérdezéseket szűrőként. A 14. fejezetben megmutatjuk, hogyan lehet az allekérdezéseket a HAVING záradékban adatcsoportok szűrésére használni.

Az alábbiakban néhány olyan feladatot láthatunk, amelyeket egy WHERE záradékban sorok szűrésére használt allekérdezések segítségével oldhatunk meg. Vegyük észre, hogy ezekből a feladatokból jó párat már megoldottunk a korábbi fejezetekben – most arra kell rájönnünk, hogy miként oldhatjuk meg őket másképp, allekérdezésekkel.

Megjegyzés *Segítségként a feladatok után zárójelben megadtuk a megoldáshoz használható kulcsszavakat.*

„Sorold fel azokat a vásárlókat, akik kerékpárt rendeltek!" (IN)
„Jelenítsd meg azokat a vásárlókat, akik ruhaneműt vagy kiegészítőket rendeltek!" (=-SOME)
„Keresd meg azokat a vásárlókat, akik kerékpáros bukósisakot rendeltek!" (IN)
„Keresd meg azokat a vásárlókat, akik kerékpárt rendeltek, de nem rendeltek bukósisakot!" (NOT EXISTS)
„Melyek azok a termékek, amelyekből még soha nem rendeltek?" (NOT IN)
„Írd ki azokat a megrendelőket, akik olyan előadók fellépéseit kötötték le, akik country-t vagy country rock-ot játszanak!" (IN)
„Keresd meg azokat az előadókat, akik játszottak a Bonnicksen vagy a Rosales nevű megrendelőnél!" (= SOME)
„Jelenítsd meg azokat az ügynököket, akik nem kötöttek le egyetlen előadót sem!" (NOT IN)
„Sorold fel azokat az előadókat, akik Bonnicksen és Rosales nevű megrendelőinknél is játszottak!" (EXISTS)
„Jelenítsd meg a keddi órákra feliratkozott tanulókat!" (IN)
„Mutasd meg azokat a tanulókat, akiknek 85 vagy annál több a rajzból elért átlaguk, és 85 vagy több a számítástechnikai átlaguk!" (EXISTS)
„Jelenítsd meg azokat a tanulókat, akik soha nem adtak le órát!" (NOT IN)
„Írd ki a szerdán tartott órákat!" (IN)
„Jelenítsd meg azokat a csapatkapitányokat, akiknek magasabb az átlaguk, mint a csapatuk bármely tagjának!" (> ALL)
„Mutasd meg a még le nem játszott tornákat!" (NOT IN)
„Keresd meg azokat a játékosokat, akiknek 170 vagy több volt a tiszta pontszámuk a Thunderbird Lanes és a Bolero Lanes pályán!" (EXISTS)
„Írd ki azokat a játékosokat, akiknek alacsonyabb az átlaguk, mint a csapatuk többi tagjának!" (< ALL)
„Mutasd meg a marhahúst és fokhagymát tartalmazó recepteket!" (EXISTS)
„Jelenítsd meg a répát tartalmazó receptek hozzávalóit!" (IN)
„Sorold fel azokat a hozzávalókat, amelyeknek a valamelyik receptben felhasznált mennyisége eltér az alapértelmezett mennyiségtől!" (<> SOME)
„Írd ki a még egyetlen receptben sem használt hozzávalókat!" (NOT IN)

Példák

Most már tudjuk, hogyan kell allekérdezéseket tartalmazó lekérdezéseket létrehozni, és láttunk pár kérelmet, amit allekérdezések segítségével fogalmazhatunk meg. Nézzünk meg egy egészséges adag példát, amelyek mindegyike egy vagy több allekérdezést használ! A példák a mintaadatbázisokra támaszkodnak, és a kimenetet biztosító vagy szűrő allekérdezések használatát mutatják be. Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név ugyanaz, mint a CD-mellékleten található megfelelő SQL-lekérdezés neve.

Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH11"-gyel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne felejtsük el, hogy a példáinkban használt oszlop- és táblanevek a B függelékben található mintaadatbázisok szerkezetéből származnak. Mivel sok példa összetett JOIN-okat használ, a használt adatbázis-kezelő esetleg más módon oldja fel a lekérdezéseket, ezért előfordulhat, hogy az első pár sor nem egyezik az általunk kapott eredménnyel, de a sorok végleges száma ugyanannyi kell legyen. Az egyszerűség kedvéért a következő példákban összevontuk a Fordítás és a Tisztázás lépéseit.*

Allekérdezések kifejezésekben

Sales Orders adatbázis

„Sorold fel a beszállítókat és az általuk nekünk eladott termékek számát!"

Fordítás/ Tisztázás
Select vendor name ~~and also~~
(select ~~the~~ count(*) ~~of products~~
from ~~the~~ product vendors ~~table~~
where ~~the~~ product vendor ~~table~~ vendor ID
~~equals~~ = ~~the~~ vendors ~~table~~ vendor ID)
from ~~the~~ vendors ~~table~~

SQL

```
SELECT VendName,
   (SELECT COUNT(*)
   FROM Product_Vendors
   WHERE Product_Vendors.VendorID =
      Vendors.VendorID)
AS VendProductCount
FROM Vendors
```

CH11_Vendors_Product_Count (10 sor)

VendName	VendProductCount
Shinoman, Incorporated	3
Viscount	6
Nikoma of America	5
ProFormance	3
Kona, Incorporated	1
Big Sky Mountain Bikes	22
Dog Ear	9
Sun Sports Suppliers	5
Lone Star Bike Supply	30
Armadillo Brand	6

Megjegyzés *A SELECT záradékban levő allekérdezéshez álnevet rendeltünk, hogy a kimenet értelmes nevet adjon. Ha ezt nem tesszük meg, az adatbázis-kezelő alkot egy nevet (például Expr1 vagy valami hasonló).*

Entertainment Agency adatbázis

„Jelenítsd meg az összes megrendelőt és mindegyikük utolsó lekötött rendezvényének a dátumát!"

Fordítás/ Tisztázás

Select customer first name, customer last name, ~~and also~~
(select ~~the highest~~ MAX(start date)
from ~~the~~ engagements ~~table~~
where ~~the~~ engagements ~~table~~ customer ID
~~equals~~ = ~~the~~ customers ~~table~~ customer ID)
from ~~the~~ customers ~~table~~

SQL

```
SELECT Customers.CustFirstName,
   Customers.CustLastName,
   (SELECT MAX(StartDate)
   FROM Engagements
   WHERE Engagements.CustomerID =
      Customers.CustomerID)
AS LastBooking
FROM Customers
```

Megjegyzés *Néhány megrendelőnél üres (Null) a LastBooking sor, mert nekik nincs lekötött rendezvényük.*

CH11_Customers_Last_Booking (15 sor)

CustFirstName	CustLastName	LastBooking
Doris	Hartwig	2008-02-23
Deb	Waldal	2008-02-17
Peter	Brehm	2008-02-26
Dean	McCrae	2008-02-24
Elizabeth	Hallmark	2008-02-19
Matt	Berg	2008-02-23
Liz	Keyser	2008-02-19
Darren	Gehring	
Sarah	Thompson	2008-02-24
<< további sorok >>		

School Scheduling adatbázis

„Jelenítsd meg az összes tantárgyat és az egyes tárgyak hétfői óráinak a számát!"

Fordítás/ Select subject name ~~and also~~
Tisztázás (select ~~the~~ count(*) ~~of classes~~
from ~~the~~ classes ~~table~~
where Monday schedule ~~is~~ = true
and ~~the~~ classes ~~table~~ subject ID ~~equals~~ = ~~the~~ subjects ~~table~~
subject ID)
from ~~the~~ subjects ~~table~~

SQL
```
SELECT Subjects.SubjectName,
   (SELECT COUNT(*)
   FROM Classes
   WHERE MondaySchedule = 1
   AND Classes.SubjectID = Subjects.SubjectID)
AS MondayCount
FROM Subjects
```

Megjegyzés *Győződjünk meg arról, hogy az adatbázis-kezelőnk által támogatott „igaz" értékre vizsgálunk. Emlékezzünk vissza, hogy néhány adatbázisrendszer a TRUE kulcsszóval vagy a –1 értékkel való összehasonlítást követel meg.*

CH11_Subjects_Monday_Count (56 sor)

SubjectName	MondayCount
Financial Accounting Fundamentals I	2
Financial Accounting Fundamentals II	1
Fundamentals of Managerial Accounting	1
Intermediate Accounting	1
Business Tax Accounting	1
Introduction to Business	0
Developing A Feasibility Plan	0
Introduction to Entrepreneurship	1
<< további sorok >>	

Megjegyzés *Ha nincsenek sorok, a COUNT összesítő függvény Null helyett 0-t ad vissza.*

Bowling League adatbázis

„Jelenítsd meg az összes tekejátékost és az általuk elért legmagasabb pontszámot!"

Fordítás/ Select bowler first name, bowler last name~~, and also~~
Tisztázás (select ~~the highest~~ MAX(raw score)
from ~~the~~ bowler scores ~~table~~

where the bowler scores table bowler ID
equals = the bowlers table bowler ID)
from the bowlers table

```
SQL     SELECT Bowlers.BowlerFirstName,
           Bowlers.BowlerLastName,
           (SELECT MAX(RawScore)
           FROM Bowler_Scores
           WHERE Bowler_Scores.BowlerID =
           Bowlers.BowlerID)
        AS HighScore
        FROM Bowlers
```

CH11_Bowler_High_Score (32 sor)

BowlerFirstName	BowlerLastName	HighScore
Barbara	Fournier	164
David	Fournier	178
John	Kennedy	191
Sara	Sheskey	149
Ann	Patterson	165
Neil	Patterson	179
David	Viescas	195
Stephanie	Viescas	150
<< *további sorok* >>		

Recipes adatbázis

„Írd ki az összes húsfélét és hogy hány receptben szerepelnek!"

Fordítás/ Select ingredient class description, ingredient name, and also
Tisztázás (select the count(*) of rows
from the recipe ingredients table
where the recipe ingredients table ingredient ID equals =
the ingredients table ingredient ID)
from the ingredient classes table
inner joined with the ingredients table
on ingredient_classes.ingredient class ID
in the ingredients classes table
matches = ingredients.ingredient class ID in the ingredients table
where ingredient class description is = 'meat'

```
SQL     SELECT Ingredient_Classes.IngredientClassDescription,
           Ingredients.IngredientName,
```

```
   (SELECT COUNT(*)
   FROM Recipe_Ingredients
   WHERE Recipe_Ingredients.IngredientID =
      Ingredients.IngredientID)
AS RecipeCount
FROM Ingredient_Classes
INNER JOIN Ingredients
ON Ingredient_Classes.IngredientClassID =
   Ingredients.IngredientClassID
WHERE
   Ingredient_Classes.IngredientClassDescription
   = 'Meat'
```

CH11_Meat_Ingredient_Recipe_Count (11 sor)

IngredientClassDescription	IngredientName	RecipeCount
Meat	Beef	2
Meat	Chicken, Fryer	0
Meat	Bacon	0
Meat	Chicken, Pre-cut	0
Meat	T-bone Steak	0
Meat	Chicken Breast	0
Meat	Chicken Leg	1
Meat	Chicken Wing	0
Meat	Chicken Thigh	1
Meat	New York Steak	0
Meat	Ground Pork	1

Allekérdezések szűrőkben

Sales Orders adatbázis

„Jelenítsd meg azokat a vásárlókat, akik ruhaneműt vagy kiegészítőket rendeltek!"

Fordítás/ Tisztázás

Select customer ID, customer first name, customer last name
from ~~the~~ customers ~~table~~
where customer ID ~~is equal to~~ = any ~~of the~~
(selec~~tion of~~ customer ID
from ~~the~~ orders ~~table~~
inner join~~ed with the~~ order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_details.order number ~~in the order details table, then~~
inner join~~ed with the~~ products ~~table~~

on products.product number ~~in the products table matches~~
= order_details.product number ~~in the order details table,~~
~~and then~~ inner join~~ed with the~~ categories ~~table~~
on categories.category ID ~~in the categories table matches~~
= products.category ID ~~in the products table~~
where category description ~~is~~ = 'clothing'
or category description ~~is~~ = 'accessories')

SQL

```
SELECT Customers.CustomerID,
   Customers.CustFirstName,
   Customers.CustLastName
FROM Customers
WHERE Customers.CustomerID = ANY
   (SELECT Orders.CustomerID
   FROM ((Orders
   INNER JOIN Order_Details
   ON Orders.OrderNumber =
      Order_Details.OrderNumber)
   INNER JOIN Products
   ON Products.ProductNumber =
      Order_Details.ProductNumber)
   INNER JOIN Categories
   ON Categories.CategoryID =
      Products.CategoryID
   WHERE Categories.CategoryDescription
      = 'Clothing'
   OR Categories.CategoryDescription
      = 'Accessories')
```

CH11_Customers_Clothing_OR_Accessories (27 sor)

CustomerID	CustFirstName	CustLastName
1001	Suzanne	Viescas
1002	William	Thompson
1003	Gary	Hallmark
1004	Robert	Brown
1005	Dean	McCrae
1006	John	Viescas
1007	Mariya	Sergienko
1008	Neil	Patterson
<< további sorok >>		

Megjegyzés *A móka kedvéért ezt a lekérdezést az = ANY használatával oldottuk meg. Ki tudunk-e gondolni egy IN-t vagy EXISTS-t használó megoldást? A megoldásokat CH11_Customers_Clothing_OR_Accessories_IN és CH11_Customers_Clothing_OR_Accessories_EXISTS néven megtalálhatjuk a mintaadatbázisban.*

Entertainment Agency adatbázis

„Keresd meg azokat az előadókat, akik játszottak Berg és Hallmark nevű megrendelőinknél!"

Megjegyzés *Ezt a feladatot a 8. fejezetben két összetett tábla-alekérdezés összekapcsolásával oldottuk meg. Ezúttal az EXISTS kulcsszót használjuk.*

Fordítás/ Tisztázás

Select entertainer ID, ~~and~~ entertainer stage name from ~~the~~ entertainers ~~table~~
where ~~there~~ exists
(select * ~~some row~~
from ~~the~~ customers ~~table~~
inner ~~joined with the~~ engagements ~~table~~
on customers.customer ID ~~in the customers table matches~~
= engagements.customer ID ~~in the engagements table~~
where customer last name ~~is~~ = 'Berg'
and ~~the~~ engagements ~~table~~ entertainer ID
~~equals~~ = ~~the~~ entertainers ~~table~~ entertainer ID)~~;~~
and ~~there also~~ exists
(select * ~~some row~~
from ~~the~~ customers ~~table~~
inner ~~joined with the~~ engagements ~~table~~
on customers.customer ID ~~in the customers table matches~~
= engagements.customer ID ~~in the engagements table~~
where customer last name ~~is~~ = 'Hallmark'
and ~~the~~ engagements ~~table~~ entertainer ID
~~equals~~ = ~~the~~ entertainers ~~table~~ entertainer ID)

SQL

```
SELECT Entertainers.EntertainerID,
   Entertainers.EntStageName
FROM Entertainers
WHERE EXISTS
   (SELECT *
   FROM Customers
   INNER JOIN Engagements
   ON Customers.CustomerID =
      Engagements.CustomerID
   WHERE Customers.CustLastName = 'Berg'
```

```
   AND Engagements.EntertainerID =
      Entertainers.EntertainerID)
AND EXISTS
   (SELECT *
   FROM Customers
   INNER JOIN Engagements
   ON Customers.CustomerID =
      Engagements.CustomerID
   WHERE Customers.CustLastName = 'Hallmark'
   AND Engagements.EntertainerID =
      Entertainers.EntertainerID)
```

CH11_Entertainers_Berg_AND_Hallmark_EXISTS (4 sor)

EntertainerID	EntStageName
1001	Carol Peacock Trio
1003	JV & the Deep Six
1006	Modern Dance
1008	Country Feeling

School Scheduling adatbázis

„Jelenítsd meg azokat a tanulókat, akik soha nem adtak le órát!"

Fordítás/ Tisztázás

Select student ID, student first name, ~~and~~ student last name
from ~~the~~ students ~~table~~
where ~~the~~ student ID ~~is~~ not in ~~the~~
(select~~ion of~~ student ID
from ~~the~~ student schedules ~~table~~
inner join~~ed with the~~ student class status ~~table~~
on student_schedules.class status
~~in the student schedules table matches~~
= student_class_status.class status
~~in the student class status table~~
where class status description ~~is~~ = 'withdrew')

SQL

```
SELECT Students.StudentID,
   Students.StudFirstName,
   Students.StudLastName
FROM Students
WHERE Students.StudentID NOT IN
   (SELECT Student_Schedules.StudentID
   FROM Student_Schedules
   INNER JOIN Student_Class_Status
   ON Student_Schedules.ClassStatus =
```

```
    Student_Class_Status.ClassStatus
WHERE
Student_Class_Status.ClassStatusDescription
    = 'Withdrew')
```

Megjegyzés *Ez elég egyszerű lekérdezés, amely egy allekérdezésben megkeresi az összes tanulót, aki valaha is adott le órát, aztán lekéri az összes tanulót, aki nincs ebben a listában (NOT IN). Hogyan oldhatnánk meg ezt egy OUTER JOIN használatával?*

CH11_Students_Never_Withdrawn (15 sor)

StudentID	StudFirstName	StudLastName
1002	David	Hamilton
1003	Betsey	Stadick
1004	Janice	Galvin
1005	Doris	Hartwig
1006	Scott	Bishop
1007	Elizabeth	Hallmark
1008	Sara	Sheskey
1010	Marianne	Wier
<< *további sorok* >>		

Bowling League adatbázis

„Jelenítsd meg azokat a csapatkapitányokat, akiknek a büntetőpontszáma magasabb, mint a csapatuk bármelyik tagjának!"

Fordítás/ Tisztázás

Select team name,bowler ID,bowler first name,
bowler last name,~~and~~ handicap score
from ~~the~~ bowlers ~~table~~
inner ~~joined with the~~ teams ~~table~~
on bowlers.bowler ID ~~in the bowlers table matches~~
= teams.captain ID ~~in the teams table~~
inner ~~joined with the~~ bowler scores ~~table~~
on bowlers.bowler ID ~~in the bowlers table matches~~
= bowler_scores.bowler ID ~~in the bowler scores table~~
where ~~the~~ handicap score ~~is greater than~~ > all ~~the~~
(select~~ion of~~ handicap score
from bowlers as B2
inner ~~joined with the~~ bowler scores ~~table~~ as BS2
on B2.bowler ID ~~in the B2 table matches~~
= BS2.bowler ID ~~in the BS2 table~~
where ~~the~~ B2 ~~table~~ bowler ID ~~is not equal~~ <> ~~the~~ bowlers

table bowler ID
and the B2 table team ID is equal = to the bowlers table team ID)

SQL

```
SELECT Teams.TeamName, Bowlers.BowlerID,
   Bowlers.BowlerFirstName,
   Bowlers.BowlerLastName,
   Bowler_Scores.HandiCapScore
FROM (Bowlers
INNER JOIN Teams
ON Bowlers.BowlerID = Teams.CaptainID)
INNER JOIN Bowler_Scores
ON Bowlers.BowlerID = Bowler_Scores.BowlerID
WHERE Bowler_Scores.HandiCapScore > All
  (SELECT BS2.HandiCapScore
   FROM Bowlers AS B2
   INNER JOIN Bowler_Scores AS BS2
      ON B2.BowlerID = BS2.BowlerID
   WHERE B2.BowlerID <> Bowlers.BowlerID
      AND B2.TeamID = Bowlers.TeamID)
```

Megjegyzés *Azért adtunk kifejezetten álneveket a Bowlers és a Bowler_Scores tábla másodpéldányainak az allekérdezésben, hogy pontosan látszódjék, mi is folyik. Nem az adott játékos pontjaival akarunk összehasonlítani, mert úgy az > ALL állítás hibás lenne. Hasonlóképp, csak az adott csapatban játszó többi játékos pontszámával akarunk összehasonlítást végezni.*

CH11_Team_Captains_High_Score (1 sor)

TeamName	BowlerID	BowlerFirstName	BowlerLastName	HandiCapScore
Huckleberrys	7	David	Viescas	224

Recipes adatbázis

„Jelenítsd meg a répát tartalmazó receptek hozzávalóit!"

Megjegyzés *A 8. fejezetben megígértük, hogy megmutatjuk, miként oldhatjuk meg a feladatot allekérdezéssel. Betartottuk az ígéretünket!*

Fordítás/ Tisztázás

Select recipe title and ingredient name
from the recipes table
inner joined with the recipe ingredients table
on recipes.recipe ID in the recipes table matches
= recipe_ingredients.recipe ID in the recipe ingredients table,
and then inner joined with the ingredients table
on ingredients.ingredient ID in the ingredients table matches

= recipe_ingredients.ingredient ID
in the recipe ingredients table
where recipe ID is in the
(selection of recipe ID
from the ingredients table
inner joined with the recipe ingredients table
on ingredients.ingredient ID in the ingredients table matches
= recipe_ingredients.ingredient ID
in the recipe ingredients table
where ingredient name is = 'carrot')

SQL

```
SELECT Recipes.RecipeTitle,
   Ingredients.IngredientName
FROM (Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
WHERE Recipes.RecipeID
IN
   (SELECT Recipe_Ingredients.RecipeID
   FROM Ingredients
   INNER JOIN Recipe_Ingredients
   ON Ingredients.IngredientID =
      Recipe_Ingredients.IngredientID
   WHERE Ingredients.IngredientName = 'carrot')
```

CH11_Recipes_Ingredients_With_Carrots (16 sor)

RecipeTitle	IngredientName
Irish Stew	Beef
Irish Stew	Onion
Irish Stew	Potato
Irish Stew	Carrot
Irish Stew	Water
Irish Stew	Guinness Beer
Salmon Filets in Parchment Paper	Salmon
Salmon Filets in Parchment Paper	Carrot
Salmon Filets in Parchment Paper	Leek
<< további sorok >>	

Megjegyzés *Ha a „carrot"-ra (répa) történő szűrést a külső lekérdezésbe tesszük, csak a répa hozzávalókat fogjuk látni a kimenetben. Ebben a feladatban azonban az összes hozzávalót látni szeretnénk az összes olyan recepthez, ami répát tartalmaz, ezért az allekérdezés használata jó megoldás. Az eredmény látszólag a receptek címe szerinti sorrendben érkezik, noha nem adtuk meg az ORDER BY záradékot. Ha minden adatbázis-kezelő rendszerben biztosítani szeretnénk ezt a sorrendet, mindenképp adjuk meg az ORDER BY záradékot.*

Összefoglalás

A fejezetet az SQL-szabvány által megadott három allekérdezés-típus – sor-, tábla- és skaláris allekérdezés – meghatározásával kezdtük, és felidéztük, hogy már foglalkoztunk a tábla-allekérdezések FROM záradékban való használatával. Röviden leírtuk a sorallekérdezések használatát, és elmondtuk, hogy ezt még nem sok kereskedelmi adatbázis-kezelő támogatja.

Utána megmutattuk, hogyan használjuk az allekérdezéseket egy SELECT záradék oszlopkifejezéseként. Megvizsgáltunk egy egyszerű példát, majd bemutattunk két összesítő függvényt, amelyeket egy másik táblából származó összesített információ megszerzésére használhatunk (a következő fejezetben részletesen foglalkozunk az összes összesítő függvénnyel).

Ez után azt tárgyaltuk, hogy miként adhatunk meg összetett szűrőket allekérdezések segítségével a WHERE záradékban. Először egyszerű összehasonlításokkal foglalkoztunk, aztán bemutattuk azokat az összehasonlító kulcsszavakat – IN, SOME, ANY, ALL és EXISTS –, amelyek az allekérdezésekből képzett állítások megalkotásakor hasznosak.

Összefoglaltuk, miért hasznosak az allekérdezések, és adtunk pár allekérdezéssel megoldható mintapéldát. A fejezet további része az allekérdezések használatát mutatta be példákon keresztül. A példákat két csoportra bontottuk: allekérdezések használata oszlopkifejezésekben és allekérdezések használata szűrőkben.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Jelenítsd meg a termékeket és a dátumot, amikor a termékeket utoljára megrendelték!"*
   (Tipp: használjuk a MAX összesítő függvényt.)
   A megoldás itt található: CH11_Products_Last_Date (40 sor).
2. *„Sorold fel azokat a vásárlókat, akik kerékpárt rendeltek!"*
   (Tipp: alkossunk szűrőt az IN felhasználásával.)
   A megoldás itt található: CH11_Customers_Order_Bikes (23 sor).
3. *„Keresd meg azokat a vásárlókat, akik kerékpárt rendeltek, de nem rendeltek bukósisakot!"*
   (Tipp: induljunk ki az előbbi lekérdezésből, és adjunk hozzá egy NOT EXISTS-et használó szűrőt.)
   A megoldás itt található: CH11_Customer_Bikes_No_Helmets (2 sor).
4. *„Melyek azok a termékek, amelyekből még soha nem rendeltek?"*
   (Tipp: szűrjünk a NOT IN használatával.)
   A megoldás itt található: CH11_Products_Not_Ordered (2 sor).

Entertainment Agency adatbázis

1. *„Sorold fel az előadókat és minden előadó fellépéseinek a számát!"*
   (Tipp: használjuk a COUNT összesítő függvényt.)
   A megoldás itt található: CH11_Entertainers_Engagement_Count (13 sor).
2. *„Írd ki azokat a megrendelőket, akik olyan előadók fellépéseit kötötték le, akik country-t vagy country rock-ot játszanak!"*
   (Tipp: alkossunk szűrőt az IN felhasználásával.)
   A megoldás itt található: CH11_Customers_Who_Like_Country (13 sor).
3. *„Keresd meg azokat az előadókat, akik játszottak Berg vagy Hallmark nevű megrendelőinknél!"*
   (Tipp: szűrjünk az = SOME segítségével.)
   A megoldás itt található: CH11_Entertainers_Berg_OR_Hallmark_SOME (8 sor).
4. *„Jelenítsd meg azokat az ügynököket, akik nem kötöttek le egyetlen előadót sem!"*
   (Tipp: A NOT IN-t használva szűrjünk.)
   A megoldás itt található: CH11_Bad_Agents (1 sor).

School Scheduling adatbázis

1. *„Sorold fel az összes tanárt és az általuk tartott órák számát!"*
   (Tipp: használjuk a COUNT összesítő függvényt.)
   A megoldás itt található: CH11_Staff_Class_Count (27 sor).
2. *„Jelenítsd meg a keddi órákra feliratkozott tanulókat!"*
   (Tipp: alkossunk szűrőt az IN felhasználásával.)
   A megoldás itt található: CH11_Students_In_Class_Tuesdays (18 sor).
3. *„Mutasd meg azokat a tanulókat, akiknek 85 vagy annál több a rajzból elért átlaguk, és 85 vagy több a számítástechnikai átlaguk!"*

(Tipp: alkossunk szűrőt az EXISTS felhasználásával.)
A megoldás itt található: CH11_Good_Art_CS_Students_EXISTS (1 sor).

4. *„Írd ki a szerdán tartott órákat!"*
(Tipp: alkossunk szűrőt az IN felhasználásával.)
A megoldás itt található: CH11_Subjects_On_Wednesday (45 sor).

Bowling League adatbázis

1. *„Jelenítsd meg az összes tekejátékost és az általuk lejátszott mérkőzések számát!"*
(Tipp: használjuk a COUNT összesítő függvényt.)
A megoldás itt található: CH11_Bowlers_And_Count_Games (32 sor).
2. *„Mutasd meg a még le nem játszott tornákat!"*
(Tipp: használjuk a NOT IN szűrőt.)
A megoldás itt található: CH11_Tourneys_Not_Played (6 sor).
3. *„Keresd meg azokat a játékosokat, akiknek 170 vagy több volt a tiszta pontszámuk a Thunderbird Lanes és a Bolero Lanes pályán!"*
(Tipp: alkossunk szűrőt az EXISTS felhasználásával.)
A megoldás itt található: CH11_Good_Bowlers_TBird_And_Bolero_EXISTS (11 sor).
4. *„Írd ki azokat a játékosokat, akiknek alacsonyabb a tiszta pontszámuk, mint a csapatuk többi tagjának!"*
(Tipp: építsünk szűrőt az < ALL segítségével, és használjuk a DISTINCT-et is arra az esetre, ha egy játékos többször ugyanolyan alacsony pontszámmal játszott.)
A megoldás itt található: CH11_Bowlers_Low_Score (3 sor).

Recipes adatbázis

1. *„Mutasd meg a receptek fajtáit és hogy hány recept van az adott fajtából!"*
(Tipp: használjuk a COUNT összesítő függvényt.)
A megoldás itt található: CH11_Count_Of_Recipe_Types (7 sor).
2. *„Mutasd meg a marhahúst és fokhagymát tartalmazó recepteket!"*
(Tipp: alkossunk szűrőt az EXISTS felhasználásával.)
A megoldás itt található: CH11_Recipes_Beef_And_Garlic (1 sor).
3. *„Sorold fel azokat a hozzávalókat, amelyeknek a valamelyik receptben felhasznált mennyisége eltér az alapértelmezett mennyiségtől!"*
(Tipp: alkossunk szűrőt a <> SOME felhasználásával.)
A megoldás itt található: CH11_Ingredients_Using_NonStandard_Measure (21 sor).
4. *„Írd ki a még egyetlen receptben sem használt hozzávalókat!"*
(Tipp: alkossunk szűrőt a NOT IN felhasználásával.)
A megoldás itt található: CH11_Ingredients_No_Recipe (20 sor).

IV

Adatok összesítése és csoportosítása

12

Egyszerű összesítések

„Kétféle statisztika létezik:
az egyiket kikeresi az ember,
a másikat kitalálja."
– Rex Stout
Nero Wolfe – Egy szajha halála

A fejezet témakörei

- Összesítő függvények
- Összesítő függvények használata a szűrőkben
- Példák
- Összefoglalás
- Önálló feladatok

Már tudjuk, hogyan válasszuk ki egy adott kérelemhez a szükséges oszlopokat, hogyan határozzuk meg a részleteket megadó kifejezéseket, hogyan kapcsoljuk össze a szükséges oszlopokat tartalmazó táblákat, és azt is, hogyan határozzuk meg az eredményhalmazba kerülő adatokat meghatározó szűrőfeltételeket. Ezeket a módszereket azért ismertettük, hogy megtanuljuk, hogyan nyerhetünk ki részletes adatokat az adatbázis egy vagy több táblájából. Ebben és a következő két fejezetben egy lépéssel távolabbról, tágabb perspektívából szemléljük az adatokat, vagyis megpróbálunk „átfogó képet" adni.

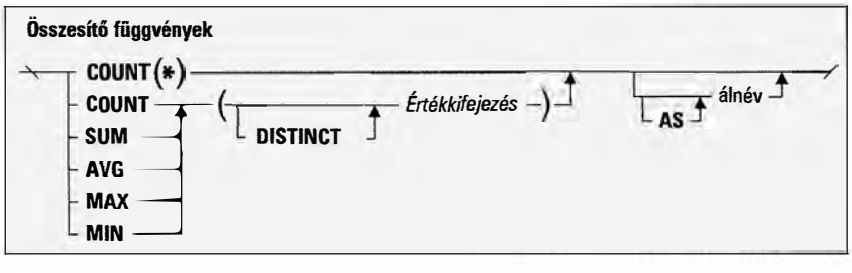
Ebben a fejezetben elmondjuk, hogyan használjuk az összesítő függvényeket információk egyszerű összegzéséhez, a 13. fejezetben azt mutatjuk be, hogyan szervezhetjük az adatokat csoportokba a SELECT kifejezés GROUP BY záradéka segítségével, végül a 14. fejezetben különféle szűrési módszereket ismertetünk, amelyeket az adatok csoportosítása után alkalmazhatunk.

Összesítő függvények

Azok a kérelmek, amelyekkel eddig találkoztunk, a FROM és WHERE záradék által visszaadott sorokból származó, egyedi oszlopértékeket magukba foglaló válaszokat igényeltek. Gyakran találkozunk azonban az alábbiakhoz hasonló kérelmekkel, ahol csupán a sorokból kiszámított értékekre van szükség a válaszhoz.

- *„Hány vásárlónk él Seattle-ben?"*
- *„Mi a legalacsonyabb és mi a legmagasabb ár, amit a készletünkben szereplő tételekhez rendeltünk?"*
- *„Hány tárgyat tanít Mike Hernandez?"*
- *„Hány órakor kezdődik a legkorábbi óránk?"*
- *„Milyen hosszú átlagosan egy óra?"*
- *„Hány tételből áll a 12. számú rendelés?"*

Az SQL-szabvány többféle összesítő függvényt határoz meg, amelyekkel egyetlen értéket számíthatunk ki az eredményhalmaz soraiból vagy egy értékkifejezés visszaadott értékéből. A függvényeket alkalmazhatjuk az összes sorra vagy értékre, vagy a WHERE záradék felhasználásával sorok vagy értékek meghatározott részhalmazára. Összesítő függvények alkalmazásával meghatározhatjuk például egy értékkifejezés legkisebb vagy legnagyobb értékeit, megszámolhatjuk az eredményhalmaz sorait, vagy kiszámíthatjuk egy értékkifejezés egyedi értékeinek az összegét. A 12.1. ábrán látható a minden adatbázis-kezelő rendszer által támogatott, alapvető összesítő függvények utasításformája.



12.1. ábra

Az összesítő függvények szintaxisdiagramja

Láthatjuk, hogy az összesítő függvények nyelvtana igazán egyszerű és lényegre törő. Az előző fejezetben két összesítő függvényt már használtunk allekérdezésekben – ahhoz, hogy visszaadjunk egy SELECT záradékban kiszámított értéket, és ahhoz, hogy lekérdezzünk egy WHERE záradék állításában felhasználható értéket. Ezekre a felhasználási módokra mutatunk még néhány példát a fejezet hátralévő részében.

Megjegyzés *A 2003-as SQL-szabvány tucatnyinál is több további összesítő függvényt határoz meg, de ezek között sok olyan van, amelyeket egyetlen fontosabb kereskedelmi adatbázis-kezelő sem valósít meg. Ebben a fejezetben azokra az alapvető összesítő függvényekre összpontosítunk, amelyeket minden fontosabb rendszer támogat. Miután megtanultuk, hogyan használjuk őket, nézzünk utána adatbázisunk dokumentációjában annak, hogy milyen további függvényeket használhatunk az SQL-kifejezésekben.*

Valamennyi összesítő függvény egyetlen értéket ad vissza, tekintet nélkül arra, hogy egy eredményhalmaz sorait dolgozza fel, vagy egy értékkifejezést. A COUNT(*) kivételével mindegyik összesítő függvény automatikusan figyelmen kívül hagyja a NULL értékeket. Egyidejüleg több összesítő függvényt is használhatunk az értéket visszaadó kifejezések listájában, közvetlenül a SELECT kulcsszó után, sőt az összesítő függvényeket tartalmazó és a literális értékeket használó értékkifejezéseket együtt, keverve is használhatjuk. De valahányszor összesítő kifejezéseket használunk, járjunk el körültekintően.

Amikor egy összesítő kifejezést építünk be, arra kérjük az adatbázis-kezelőt, hogy egy sorcsoport alapján számítson ki egy értéket. A következő fejezetben megtanuljuk, hogyan adjuk meg a célnak megfelelő csoportokat a GROUP BY záradék segítségével, de ebben a fejezetben még csak olyan egyszerű lekérdezéseket vizsgálunk, amelyek kifejezett módon nem határoznak meg csoportokat. Csoport-meghatározás hiányában az adatbáziskezelő az összesítő kifejezések kiszámításához csoport gyanánt a FROM és a WHERE záradék által visszaadott összes sort használja.

Ha kicsit elgondolkodunk, rájöhetünk, hogy nem sok értelme lenne egy tábla valamelyik oszlopát tartalmazó értékkifejezést használni összesítő függvényeken kívül. Emlékszünk arra, amikor a COUNT és a MAX összesítő függvényeket vezettük be a 11. fejezetben? Vizsgáljuk meg a következő SQL kódot:

```
SQL     SELECT LastName, COUNT(*) As CountOfStudents
        FROM Students
```

A COUNT függvény csoport-meghatározás nélküli beépítése arra kéri az adatbázis-kezelőt, hogy számolja meg a FROM záradékból visszakapott eredményhalmaz sorait. A COUNT(*) egyetlen értéket ad vissza – a Students (Hallgatók) tábla sorainak számát –, ezért a lekérdezés csak egyetlen sort eredményezhet. Melyik LastName (Keresztnév) mezőt kell megjelenítenie az adatbázis-kezelőnek? A helyes válasz: az adatbázis-kezelő ezt nem tudja eldönteni, ezért a fenti utasítás hibás.

Literális kifejezést viszont beépíthetünk, hogy javítsunk a kimeneten. Ezt azért tehetjük meg, mert a literális kifejezések valójában olyan állandók, amelyeknek minden sorra nézve azonos az értékük. Azaz a következő SQL kód tökéletesen helyes:

```
SQL     SELECT 'The number of students is: ', COUNT(*)
        As CountOfStudents
        FROM Students
```

Ez egyetlen sort ad vissza:

```
The number of students is:        18
```

Miután túljutottunk ezen a kis figyelmeztetésen, nézzük meg egyenként az összesítő függvényeket, hogyan lehetnek segítségünkre egy-egy kérdés megválaszolásában!

Sorok és értékek megszámlálása a COUNT segítségével

Az SQL-szabvány a COUNT függvény két változatát határozza meg. A COUNT(*) eredményhalmazok sorait dolgozza fel, a COUNT(*értékkifejezés*) pedig értékkifejezések visszaadott értékeit.

Valamennyi sor összeszámlálása

A COUNT(*) segítségével meghatározhatjuk, hogy hány sort tartalmaz az eredményhalmazunk. A COUNT(*) függvény az eredményhalmaz *összes* sorát számba veszi, ideértve az ismétlődő és a NULL értékeket tartalmazó sorokat is. Következzen egy egyszerű példa arra a kérdéstípusra, amit ezzel a függvénnyel megválaszolhatunk:

Megjegyzés *Ebben a fejezetben végig a 4. fejezetben bevezetett „Kérelem – Fordítás – Tisztázás – SQL" módszert használjuk. Valamennyi példa azt feltételezi, hogy alaposan tanulmányoztuk és megértettük az előző fejezetekben tárgyalt fogalmakat, különösen a JOIN-ra és az allekérdezésekre vonatkozó fejezeteket.*

„Írd ki a cég alkalmazottainak a számát!"

```
Fordítás    Select the count of employees from the employees table
            (Válaszd ki az alkalmazottak számát az alkalmazottak táblájából.)
Tisztázás   Select ~~the~~ count ~~of employees~~ (*)
            from ~~the~~ employees ~~table~~
SQL         SELECT COUNT(*)
            FROM Employees
```

Figyeljük meg, hogy a tisztázásban a (*)-ot használtuk, hogy jelezzük, az Employees (Alkalmazottak) tábla *valamennyi* sorát meg akarjuk számolni. Használjunk mi is a csillag jelölést a tisztázás során, amikor ilyen típusú kérelmekkel dolgozunk, mert segít a megfelelő COUNT függvény kiválasztásában. A példabeli SELECT utasítás egyetlen egyoszlopos sorból álló halmazt eredményez, az Employees tábla összes sorának számát képviselő számértékkel.

Nincs megkötés arra, hogy a COUNT(*) függvény hány sort dolgozhat fel. Azt, hogy a COUNT(*) mely sorokat vegye figyelembe, a WHERE záradékkal jelezhetjük. A következő példában egy olyan SELECT utasítást adunk meg, amely az Employees táblában található, Washington államban élő dolgozók sorait számolja meg:

```
SQL     SELECT COUNT(*)
        FROM Employees
        WHERE EmpState = 'WA'
```

A fejezet anyagát feldolgozva meglátjuk, hogyan szűrhetjük a WHERE záradék segítségével egy tetszőleges összesítő függvény feldolgozott sorait vagy értékeit.

Amikor összesítő függvény használunk egy SELECT utasításban, a függvény eredményhalmazában esetenként egy oszlopnevet is láthatunk. Egyes adatbázis-kezelők visszaadnak egy alapértelmezett oszlopnevet is, míg mások nem. A függvény AS paraméterét használva értelmes oszlopnevet adhatunk meg az eredményhalmazban. Az előző példa esetében az itt látható módon tehetjük ezt meg:

```
SQL     SELECT COUNT(*) AS TotalWashingtonEmployees
        FROM Employees
        WHERE EmpState = 'WA'
```

Az eredményhalmazba most már bekerül a TotalWashingtonEmployees (Összes washingtoni alkalmazott) oszlop is; ez tartalmazza a COUNT(*) függvény visszatérési értékét. Ahogy a 12.1. ábra szintaxisdiagramja mutatja, ezt a módszert mindegyik összesítő függvény esetében alkalmazhatjuk.

Értékek összeszámlálása oszlopokban és kifejezésekben

A COUNT(*értékkifejezés*) függvényt arra használjuk, hogy megszámláljuk, hogy egy értékkifejezés összesen hány nem NULL értéket ad vissza. (A kifejezés közismert neve COUNT – mi is ezt használjuk a könyv hátralévő részében.) A COUNT összeszámolja az értékkifejezés valamennyi visszaadott értékét, tekintet nélkül arra, hogy egyediek vagy többször szerepelnek, de a végső összesítésből automatikusan kihagyja a NULL értékekeket. A COUNT segítségével a következő típusú kérdésre adhatunk választ:

„Hány vásárló tüntette fel, hogy melyik megyében él?"

Itt azt kell megmondanunk, hogy hány valódi érték van a megye oszlopában. Emlékezzünk vissza, hogy a COUNT(*) a NULL értékeket is figyelembe veszi, tehát nem adna helyes választ. Ezért helyette a COUNT függvényt használjuk, és a következőképpen fordítjuk le a kérelmet:

Fordítás Select the count of non-Null county values
as NumberOfKnownCounties from the customers table
(Válaszd ki a nem NULL megyeértékeket NumberOfKnownCounties néven
a vásárlók táblájából.)

Tisztázás Select ~~the~~ count ~~of non-Null~~ (county) ~~values~~
as NumberOfKnownCounties
from ~~the~~ customers ~~table~~

SQL
```
SELECT COUNT(CustCounty)
   AS NumberOfKnownCounties
FROM Customers
```

Figyeljük meg, hogy a fordításban és a tisztázásban hangsúlyozottan nem NULL értékeket kérünk. Bár már tudjuk, hogy ez a függvény csak nem NULL értékeket dolgoz fel, helyes gyakorlatot követünk, ha mindkét utasítást kibővítjük a „nem NULL" kitétellel, így biztosan a megfelelő függvényt fogjuk választani. Az itt megadott SELECT utasítás egyetlen sort eredményez, amely a CustCountry oszlop nem NULL értékű megyeneveinek a számát tartalmazza.

Emlékezzünk vissza, hogy a COUNT függvény a többször szereplő megyeneveket egyedinek tekinti, és mindegyiket figyelembe veszi a végső összesítésnél. Szerencsére a többször előforduló értékeket a DISTINCT módosító segítségével figyelmen kívül hagyhatjuk. Lássunk egy példát, hogyan alkalmazhatjuk ezt egy adott kérelemre:

„Hány egyedi megyenév található a customers táblában?"

Fordítás Select the count of unique non-Null county names as
NumberOfUniqueCounties from the customers table
(Válaszd ki az egyedi nem NULL megyeneveket NumberOfUniqueCounties
néven a vásárlók táblájából.)

Tisztázás Select ~~the~~ count ~~of unique non-Null~~ (distinct county) ~~names~~
as NumberOfUniqueCounties
from ~~the~~ customers ~~table~~

SQL
```
SELECT COUNT(DISTINCT CustCounty)
   AS NumberOfUniqueCounties
FROM Customers
```

Amikor a DISTINCT módosítót használjuk, az adatbázis-kezelő minden nem NULL értéket lekér a megye oszlopából, kiküszöböli a többször szereplő értékeket, és *csak ezután* számlálja összse a megmaradt értékeket. A rendszer akkor is hasonlóan jár el, amikor a DISTINCT paramétert a SUM, AVG, MIN és MAX függvényekkel használjuk.

A következő példában kicsit más változatban fogalmazzuk meg az előző kérdést, hogy bemutassuk egy szűrő és a COUNT függvény együttes használatát:

„Hány Oregon állambeli, egyedi megyenév van a vásárlók táblájában?"

Fordítás Select the count of unique non-Null county names as NumberOfUniqueOregonCounties from the customers table where the state is 'OR'
(Válaszd ki azokat az egyedi nem NULL megyeneveket NumberOfUniqueOregonCounties néven a vásárlók táblájából, amelyeknél az állam 'OR'.)

Tisztázás Select ~~the~~ count ~~of unique non-Null~~ (distinct county) ~~names~~ as NumberOfUniqueOregonCounties from ~~the~~ customers ~~table~~ where ~~the~~ state ~~is~~ = 'OR'

SQL
```
SELECT COUNT(DISTINCT CustCounty)
   AS NumberOfUniqueOregonCounties
FROM Customers
WHERE CustState = 'OR'
```

Meg kell jegyezzük, hogy a DISTINCT módosítót *nem használhatjuk* a COUNT(*) függvénnyel együtt. Ennek a megszorításnak jó oka van: a COUNT(*) a tábla *minden* oszlopát figyelembe veszi, és nem tud mit kezdeni azzal, ha bármelyik ismétlődik, vagy NULL értékeket tartalmaz.

Összeg kiszámítása a SUM segítségével

Egy számértékű kifejezés összegét a SUM függvény segítségével számíthatjuk ki, amely az értékkifejezés minden nem NULL értékét feldolgozza, végül visszaadja a kiszámított összeget az eredményhalmazban. Ha az értékkifejezés minden sora NULL, vagy ha a FROM és a WHERE záradék kiértékelésének eredménye egy üres halmaz, akkor a SUM NULL-t ad vissza. Lássunk egy példát a SUM által megválaszolható kérdésekre:

„Összesen mennyit fizetünk ki bérként a dolgozóinknak Kaliforniában?"

Fordítás Select the sum of salary as TotalSalaryAmount from the employees table where the state is 'CA'
(Válaszd ki a fizetés összegét TotalSalaryAmount néven az alkalmazottak táblájából, ha az állam 'CA'.)

Tisztázás Select ~~the~~ sum ~~of~~ (salary) as TotalSalaryAmount from ~~the~~ employees ~~table~~ where ~~the~~ state ~~is~~ = 'CA'

SQL
```
SELECT SUM(Salary) AS TotalSalaryAmount
FROM Employees
WHERE EmpState = 'CA'
```

Az itt használt értékkifejezések egyszerű oszlophivatkozások voltak. De használhattunk volna számértékeket tartalmazó értékkifejezéseket is, mint a következő példában:

„Mennyit ér az aktuális raktárkészletünk?"

Fordítás Select the sum of wholesale price times quantity on hand as TotalInventoryValue from the products table
(Válaszd ki a nagykereskedelmi ár és a készletmennyiség szorzatát TotalInventoryValue néven a termékek táblájából.)

Tisztázás Select ~~the~~ sum ~~of~~ (wholesale price ~~times~~ * quantity on hand) as TotalInventoryValue from ~~the~~ products ~~table~~

SQL
```
SELECT SUM(WholesalePrice * QuantityOnHand)
   AS TotalInventoryValue
FROM Products
```

Mint tudjuk, egy sornak tényleges értékeket kell tartalmaznia ahhoz, hogy a SUM függvény feldolgozza. Ebben az esetben az adatbázis-kezelő a Products (Termékek) tábla minden megfelelő sorát feldolgozza, a SUM függvénnyel összegzi az eredményt, végül a végösszeget visszaadja az eredményhalmazban.

A következő példa azt mutatja be, hogyan számolhatjuk ki egyedi számértékek összegét a SUM segítségével:

„Számold ki a termékeink nagykereskedelmi árának összegét!"

Fordítás Select the sum of unique wholesale costs as SumOfUniqueWholesaleCosts from the products table
(Válaszd ki az egyedi nagykereskedelmi árak összegét SumOfUniqueWholesaleCosts néven a termékek táblájából.)

Tisztázás Select ~~the~~ sum ~~of unique~~ (distinct wholesale costs) as SumOfUniqueWholesaleCosts from ~~the~~ products ~~table~~

SQL
```
SELECT SUM(DISTINCT WholesaleCost)
   AS SumOfUniqueWholesaleCosts
FROM Products
```

Átlag kiszámítása az AVG függvénnyel

Egy újabb számértékeken használható függvény az AVG, amely az értékkifejezések által visszaadott nem NULL értékek számtani közepét számítja ki. Az AVG segítségével megválaszolhatjuk például a következő kérdést:

„Mekkora az átlagos szerződési összeg az 10014-es számú beszállító esetében?"

Fordítás Select the average of contract price as AverageContractPrice from the vendor contracts table where the vendor ID is 10014
(Válaszd ki azoknak a szerződési összegeknek az átlagát AverageContractPrice néven a beszállítói szerződések táblájából, amelyeknél a beszállítóazonosító 10014.)

Tisztázás Select ~~the average of~~ avg (contract price)
as AverageContractPrice
from ~~the~~ vendor contracts table
where the vendor ID ~~is~~ = 10014

SQL
```
SELECT AVG(ContractPrice)
   AS AverageContractPrice
FROM Vendor_Contracts
WHERE VendorID = 10014
```

Amikor a tisztázást készítjük, ne felejtsük el kihúzni az „average" (átlag) szót és „avg"-t írni a helyére. Ügyeljünk, hogy ne az „average" szót használjuk a SELECT után, mert az „average" nem érvényes SQL-kulcsszó, tehát az SQL-utasítás érvénytelen lesz, ha használni próbáljuk.

Az AVG függvényt számérték feldolgozására is használhatjuk, ugyanúgy, mint a SUM függvényt. Jegyezzük meg, hogy az AVG függvényt nem használhatjuk nem számértékű értékkifejezésekkel. A legtöbb adatbázis-kezelő hibát jelez, ha ezeket a függvényeket karakterlánc vagy dátum típusú adatokkal próbáljuk használni.

„Mekkora az átlagos tételösszeg a 64-es azonosítójú rendelésnél?"

Fordítás Select the average of price times quantity ordered as
AverageItemTotal from the order details table where order ID is 64
(Válaszd ki az ár és a rendelt mennyiség szorzatát AverageItemTotal néven
a rendelések részleteit tartalmazó táblából, ahol a rendelésazonosító 64.)

Tisztázás Select ~~the average of~~ avg (price ~~times~~ * quantity ordered)
as AverageItemTotal
from ~~the~~ order details ~~table~~
where order ID ~~is~~ = 64

SQL
```
SELECT AVG(Price * QuantityOrdered)
   AS AverageItemTotal
FROM Order_Details
WHERE OrderID = 64
```

Ne feledkezzünk meg róla, hogy a soroknak érvényes értékeket kell tartalmazniuk a Price (Ár) és a QuantityOrdered (Rendelt mennyiség) oszlopokban ahhoz, hogy az AVG fel tudja dolgozni őket. Ellenkező esetben a számértékű kifejezések NULL értékre értékelődnek ki, és az AVG függvény a teljes sort figyelmen kívül hagyja. Ahogy a SUM esetében is, ha az értékkifejezés minden sora NULL, vagy ha a FROM és WHERE záradék eredménye egy üres halmaz, akkor az AVG NULL értékkel tér vissza.

Következő példánkban a DISTINCT módosítót használjuk egy egyedi számértékeket tartalmazó halmaz átlagának kiszámításához:

„Számítsd ki az összes egyedi termékár átlagát!"

Fordítás Select the average of unique prices as UniqueProductPrices
from the products table
(Válaszd ki az egyedi árak átlagát UniqueProductPrices néven a termékek táblájából.)

Tisztázás Select ~~the average of unique~~ avg (distinct price~~s~~)
as UniqueProductPrices
from ~~the~~ products ~~table~~

SQL
```
SELECT AVG(DISTINCT Price)
   AS UniqueProductPrices
FROM Products
```

A legnagyobb érték meghatározása a MAX függvénnyel

Egy értékkifejezés visszaadott értékei közül a *legnagyobbat* a MAX függvénnyel választhatjuk ki. A MAX függvény bármilyen adattípust fel tud dolgozni, a visszaadott érték pedig a feldolgozott adatok típusától függ:

KARAKTERLÁNCOK A MAX által visszaadott értéket ekkor az adatbázis-kezelő vagy a számítógép által használt jelsorrend határozza meg. Ha az adatbázis-kezelő például az ASCII karakterkészleten alapuló, a kis- és nagybetűket nem megkülönböztető összehasonlítást használ, a cégneveket a következőképpen fogja rendezni: „...4th Dimension Productions...Al's Auto Shop...allegheny & associates...Zercon Productions...zorn credit services." Ebben az esetben a MAX a „zorn credit services"-t adja vissza legnagyobb értékként.

SZÁMOK A MAX függvény a legnagyobb számot adja vissza.

DATETIME A MAX időrend szerint kiértékeli a dátumokat és az időpontokat, és a legutóbbi (vagy legkésőbbi) dátummal vagy időponttal tér vissza.

Lássunk néhány példát arra, hogy miként használhatjuk a MAX függvényt kérdések megválaszolására:

„Mekkora a legnagyobb összeg, amit egy szerződés keretein belül kifizetünk?"

Fordítás Select the maximum contract price as LargestContractPrice
from the engagements table
(Válaszd ki a legnagyobb szerződési összeget LargestContractPrice néven a rendezvények táblájából.)

Tisztázás Select ~~the~~ max~~imum~~ (contract price)
as LargestContractPrice
from ~~the~~ engagements ~~table~~

SQL
```
SELECT MAX(ContractPrice)
   AS LargestContractPrice
FROM Engagements
```

„Mekkora volt a legnagyobb sortételösszeg a 3314. rendelésnél?"

Fordítás Select the maximum price times quantity ordered as LargestItemTotal from the order details table where the order ID is 3314
(Válaszd ki a legnagyobb árat szorozva a mennyiséggel LargestItemTotal néven a rendelések részleteit tartalmazó táblából, ha a rendelésazonosító 3314.)

Tisztázás Select ~~the~~ max~~imum~~ (price ~~times~~ * quantity ordered)
as LargestItemTotal
from ~~the~~ order details ~~table~~
where ~~the~~ order ID ~~is~~ = 3314

SQL
```
SELECT MAX(Price * QuantityOrdered)
   AS LargestItemTotal
FROM Order_Details
WHERE OrderID = 3314
```

Késztetést érezhetünk, hogy a DISTINCT paramétert használjuk a legnagyobb vagy a jelenlegi értékek egyedi példányainak lekérdezésére. Bár az SQL-szabvány engedélyezi a DISTINCT módosítót a MAX függvény használatakor, a DISTINCT *semmilyen hatással nem bír* a MAX függvény esetében. Csak egyetlen legnagyobb érték létezhet, tekintet nélkül arra, hogy ez az érték egyedi-e. Ha például a legutóbbi felvételi időpontot keressük az Agents (Ügynökök) táblában, a következő két kifejezés ugyanazt az értéket fogja visszaadni:

```
SELECT MAX(DateHired) FROM Agents
SELECT MAX(DISTINCT DateHired) FROM Agents
```

Mivel jelenleg az SQL-szabvány részét képezik, ismertettük a függvény mindkét változatát, de azt javasoljuk, hogy a MAX függvényt a DISTINCT módosító nélkül használjuk. Amikor a DISTINCT módosítót használjuk, arra kérjük az adatbázis-kezelőt, hogy először keresse meg az egyedi értékeket, és utána döntse el, hogy melyik közülük a legnagyobb vagy a legutóbbi, vagyis felesleges munkát végeztetünk vele.

A legkisebb érték meghatározása a MIN függvénnyel

A MIN függvény segítségével egy értékkifejezés visszaadott értékei közül a legkisebbet határozhatjuk meg. Hasonlít a MAX függvényre, de a másik szélsőértéket adja vissza: az első karakterláncot (összehasonlítási módszertől függően), a legkisebb számot, és a legkorábbi dátumot vagy időpontot. A MIN függvénnyel a következőhöz hasonló kérdésekre felelhetünk:

„Mekkora a legkisebb összeg, amit egy termékért kérünk?"

Fordítás Select the minimum price as LowestProductPrice from the products table
(Válaszd ki a legalacsonyabb árat LowestProductPrice néven a termékek táblájából.)

Tisztázás Select ~~the~~ mini~~mum~~ (price) as LowestProductPrice
from ~~the~~ products ~~table~~

```
SQL     SELECT MIN(Price) AS LowestProductPrice
        FROM Products
```

„Mekkora volt a legkisebb sortételösszeg a 3314. rendelésnél?"

Fordítás Select the minimum price times quantity ordered as LowestItemTotal from the order details table where the order ID is 3314
(Válaszd ki a legalacsonyabb árat szorozva a mennyiséggel LowestItemTotal néven a rendelések részleteit tartalmazó táblából, ha a rendelésazonosító 3314.)

Tisztázás Select ~~the~~ mini~~mum~~ (price ~~times~~ * quantity ordered) as LowestItemTotal
from ~~the~~ order details ~~table~~
where ~~the~~ order ID ~~is~~ = 3314

```
SQL     SELECT MIN(Price * QuantityOrdered)
           AS LowestItemTotal
        FROM Order_Details
        WHERE OrderID = 3314
```

Jegyezzük meg, hogy a DISTINCT módosítónak *semmilyen hatása nincs* a MIN függvényre. (Már tudjuk, hogy ez volt a helyzet a MAX esetében is.) Csak egyetlen minimumérték létezik, függetlenül attól, hogy ez az érték egyedi-e. A következő két példakifejezés ugyanazt az értéket fogja visszaadni:

```
SELECT MIN(DateHired) FROM Agents
SELECT MIN(DISTINCT DateHired) FROM Agents
```

Mivel a jelenlegi SQL-szabvány részét képezik, ismertettük a függvény mindkét változatát, de azt javasoljuk, hogy a MIN függvényt a DISTINCT módosító nélkül használjuk. Amikor a DISTINCT módosítót használjuk, arra kérjük az adatbázis-kezelőt, hogy először keresse meg az egyedi értékeket, és utána döntse el, hogy melyik közülük a legkisebb vagy a legrégebbi, vagyis felesleges munkát végeztetünk vele.

Több függvény használata

Ahogy ennek a résznek az elején már említettük, egyidejűleg több összesítő függvényt is használhatunk, így mindössze egyetlen SELECT utasítással összehasonlíthatunk adatokat. Például a MIN és a MAX függvényekkel megjeleníthetjük egy vásárlónk legrégebbi és legújabb rendeléseinek dátumát, vagy a MAX, MIN, és AVG függvényekkel megjeleníthetjük egy tanuló legalacsonyabb, legmagasabb és átlagos osztályzatait. Következzen néhány példa arra, hogy miként használhatunk kettő vagy több összesítő függvényt!

„Mutasd meg a hirdetési részleg dolgozóinak legrégebbi és a legújabb vizsgálati dátumait!"

Fordítás Select the minimum review date as EarliestReviewDate and the maximum review date as RecentReviewDate from the employees table where the department is 'Advertising'
(Válaszd ki a legkorábbi vizsgálati időpontot EarliestReviewDate néven és a legkésőbbi vizsgálati időpontot RecentReviewDate néven az alkalmazottak táblájából, ha a részleg 'Advertising'.)

Tisztázás Select ~~the~~ min~~imum~~ review date as EarliestReviewDate, ~~and the~~ max~~imum~~ review date as RecentReviewDate from ~~the~~ employees ~~table~~ where ~~the~~ department ~~is~~ = 'Advertising'

SQL
```
SELECT MIN(ReviewDate) AS EarliestReviewDate,
   MAX(ReviewDate) AS RecentReviewDate
FROM Employees
WHERE Department = 'Advertising'
```

„Hány különböző terméket tartalmazott az 553. rendelés, és mennyi volt a teljes rendelési érték?"

Fordítás Select the count of product ID as TotalProductsPurchased and the sum of price times quantity ordered as OrderAmount from the order details table where the order number is 553
(Válaszd ki azoknak a termékazonosítóknak a számát TotalProductsPurchased néven és az árak összegét szorozva a rendelt mennyiséggel OrderAmount néven a rendelések részleteit tartalmazó táblából, amelyeknek a rendelési száma 553.)

Tisztázás Select ~~the~~ count ~~of~~ (product ID) as TotalProductsPurchased,~~and~~ ~~the~~ sum ~~of~~ (price ~~times~~ * quantity ordered) as OrderAmount from ~~the~~ order details ~~table~~ where ~~the~~ order number ~~is~~ = 553

SQL
```
SELECT COUNT(ProductID) AS TotalProductsPurchased,
   SUM(Price * QuantityOrdered) AS OrderAmount
FROM Order_Details
WHERE OrderNumber = 553
```

Ne felejtsük el, hogy milyen megkötéseket kell figyelembe vennünk, amikor két vagy több összesítő függvényt használunk. Először is, az összesítő függvényeket nem ágyazhatjuk egymásba. Ez a korlátozás érvénytelenné teszi az alábbi kifejezést:

```
SUM(AVG(LineItemTotal))
```

Másodszor, összesítő függvényben értékkifejezésként nem használhatunk allekérdezéseket.

Ez a megkötés érvénytelenné teszi az alábbi kifejezést:

```
AVG((SELECT Price FROM PRODUCTS WHERE Category = 'Bikes'))
```

A megkötések ellenére láttuk, mennyire egyszerűen használhatók az összesítő függvények a SELECT záradékában viszonylag összetett statisztikai adatok kikeresésére. Most lássuk, hogyan szűrhetjük egy eredményhalmaz adatait az összesítő függvények segítségével!

Összesítő függvények használata szűrőkben

Mivel az összesítő függvények egyetlen értéket adnak vissza, keresési feltételek összehasonlító állításaiban is használhatjuk őket. Ehhez azonban az összesítő függvényünket egy allekérdezésbe kell helyeznünk, és az allekérdezést kell az összehasonlító állítás részeként használnunk. Ha mindez ismerősen cseng, eláruljuk, hogy ez nem véletlen. A 11. fejezetben már megtanultuk, hogyan használhatunk a WHERE záradékban keresési feltételek részeként allekérdezéseket, illetve összesítő függvényeket az allekérdezéseken belül. Tehát – legalábbis általános értelemben véve – már tudjuk, hogyan használhatjuk az összesítő függvényeket az eredményhalmazban kapott adatok szűrésére. Bővítsük most tovább ezt a tudást!

Az összesítő függvények összehasonlító állításokban való használatával értékkifejezések értékét hasonlíthatjuk össze egyetlen statisztikai értékkel. Bár literális értékeket is használhatnánk erre a célra, az allekérdezések rugalmasabb megoldást jelentenek, és dinamikusabb jelleget kölcsönöznek a feltételnek. Tegyük fel, hogy a következő kérelmet intézzük az adatbázis-kezelőhöz:

> *„Sorold fel azoknak a rendezvényeknek a szerződésszámát, amelyeknek a szerződési összege nagyobb, mint az összesített átlagos szerződési összeg, vagy egyenlő azzal!"*

Az egyik módszer, amivel megválaszolhatjuk ezt a kérdést, az, hogy saját magunk kiszámítjuk az összesített átlagos szerződési összeget, és a tényleges értéket beillesztjük az összehasonlító állításba:

Fordítás	Select the engagement number from the engagements table where the contract price is greater than or equal to $24,887.00 (Válaszd ki azoknak a rendezvényeknek a szerződésszámát a rendezvények táblájából, amelyeknél a szerződési összeg nagyobb, mint 24 887 dollár, vagy egyenlő azzal.)
Tisztázás	Select ~~the~~ engagement number from ~~the~~ engagements ~~table~~ where ~~the~~ contract price ~~is greater than or equal to~~ >= ~~$~~24,887.00

SQL	`SELECT EngagementNumber`
	`FROM Engagements`
	`WHERE ContractPrice >= 24887.00`

De miért dolgoznánk többet, mint amennyit szükséges? Használjuk a megfelelő összesítő függvényt egy allekérdezésben, és hagyjuk a munka nehezét az adatbázis-kezelőre!

Fordítás Select the engagement number from the engagements table where the contract price is greater than or equal to the overall average contract price in the engagements table
(Válaszd ki azoknak a rendezvényeknek a szerződésszámát a rendezvények táblájából, amelyeknél a rendezvények táblájában a szerződési összeg nagyobb, mint az összesített átlagos szerződési összeg, vagy egyenlő azzal.)

Tisztázás Select ~~the~~ engagement number
from ~~the~~ engagements ~~table~~
where ~~the~~ contract price ~~is greater than or equal to the~~ >=
~~overall~~ (select ~~average~~ avg contract price ~~in the~~
from engagements ~~table~~)

SQL
```
SELECT EngagementNumber
FROM Engagements
WHERE ContractPrice >=
    (SELECT AVG(ContractPrice)
    FROM Engagements)
```

Nyilvánvaló, hogy akkor választjuk a legjobb utat, ha összesítő függvényt használó allekérdezést építünk be. Ha literális értéket használnánk, mindig ellenőriznünk kellene, hogy újraszámoltuk-e az átlagos szerződési összeget, mielőtt végrehajtanánk a SELECT utasítást, hátha megváltoztattuk valamelyik szerződési összeget. Ráadásul arról is meg kellene győződnünk, hogy helyesen írtuk-e be az értéket az összehasonlító állításba. Mindezzel nem kell törődnünk, ha allekérdezést használunk. Az AVG függvény a SELECT minden végrehajtásakor kiértékelődik, és mindig helyes eredményt ad, akár módosítottuk a szerződési összegeket, akár nem. (Ez igaz bármelyik, allekérdezésben használt összesítő függvényre.)

Az összesítő függvények által kiértékelt sorok számát is korlátozhatjuk az allekérdezés WHERE záradékában. Így leszűkíthetjük az összesítő függvény által szolgáltatott statisztikai értékek körét. A 11. fejezet óta már tudjuk, hogyan alkalmazzuk a WHERE záradékot allekérdezésekben, nézzünk tehát egy példát a módszer alkalmazására:

„Sorold fel az összes olyan rendezvény szerződésszámát és szerződési összegét, amelyeknél a szerződési összeg nagyobb, mint a szerződési összegek teljes összege 2007 szeptember hónapjának egészére nézve!"

Fordítás Select engagement number and contract price from the engagements table where the contract price is greater than the sum of all contract prices of engagements dated between September 1, 2007, and September 30, 2007

(Válaszd ki azokat a rendezvényszámokat és szerződési összegeket a rendezvények táblájából, ahol a szerződési összeg nagyobb, mint a 2007. szeptember 1. és 2007. szeptember 30. közötti rendezvények szerződési összegeinek összege.)

Tisztázás

Select engagement number, ~~and~~ contract price
from ~~the~~ engagements ~~table~~
where ~~the~~ contract price ~~is greater than~~ > ~~the~~
(select sum ~~of all~~ (contract prices)
from engagements
where ~~dated~~ start date between ~~September 1, 2007~~,
'2007-09-01'
and ~~September 30, 2007~~ '2007-09-30')

SQL

```
SELECT EngagementNumber, ContractPrice
FROM Engagements
WHERE ContractPrice >
   (SELECT SUM(ContractPrice) FROM Engagements
      WHERE StartDate BETWEEN '2007-09-01'
      AND '2007-09-30')
```

Azt tapasztalhatjuk, hogy a szűrőkben nem sűrűn van szükség összesítő függvényekre, de hasznunkra lehetnek, amikor egyes ritkán felbukkanó, rendhagyó kérdésekre kell felelnünk.

Példák

Ebben a fejezetben megtanultuk, hogyan használjuk az összesítő függvényeket a SELECT záradékban és allekérdezéseken belül, összehasonlító állítások részeként. Most a mintaadatbázisok tábláinak segítségével bemutatunk néhány további példát az összesítő függvények használatára. Példáink az összesítő függvények használatát kimeneti oszlopokban és allekérdezésekben mutatják be.

Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név ugyanaz, mint a CD-mellékleten található megfelelő SQL-lekérdezés neve. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH12"-vel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne felejtsük el, hogy a példáinkban használt oszlop- és táblanevek a B függelékben található mintaadatbázisok szerkezetéből származnak. Az egyszerűség kedvéért a következő példákban összevontuk a Fordítás és a Tisztázás lépéseit.*

Sales Orders adatbázis

„Hány vásárlónk van Kalifornia államban?"

Fordítás/ Select ~~the~~ count(*) as NumberOfCACustomers
Tisztázás ~~of all customers~~ from ~~the~~ customers ~~table~~
where ~~the~~ state ~~is~~ = 'CA'

SQL
```
SELECT COUNT(*) AS NumberOfCACustomers
FROM Customers
WHERE CustState = 'CA'
```

CH12_Number_Of_California_Customers (1 sor)

NumberOfCACustomers
7

„Sorold fel azokat a termékneveket és számokat, ahol a jegyzett ár nagyobb, mint az összesített átlagos fogyasztói ár a termékek táblájában, vagy egyenlő azzal!"

Fordítás/ Select ~~the~~ product name, ~~and the~~ product number
Tisztázás from ~~the~~ products ~~table~~ inner ~~joined with the~~ order details ~~table~~
on products.product number ~~in the products table matches~~
= order_details.product number ~~in the order details table~~
where ~~the~~ quoted price ~~is greater than or equal to~~ >=
(select ~~the average~~ avg(retail price) ~~in the~~ from products ~~table~~)

SQL
```
SELECT DISTINCT Products.ProductName,
   Products.ProductNumber
FROM Products
INNER JOIN Order_Details
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Order_Details.QuotedPrice >=
   (SELECT AVG(RetailPrice)
   FROM Products)
```

Megjegyzés *Azért döntöttünk a DISTINCT használata mellett, mert (legalábbis reméljük) egy termékből egynél többet is rendelhettek, és minden terméknevet és számot csak egyszer szeretnénk látni.*

CH12_Quoted_Price_vs_Average_Retail_Price (4 sor)

ProductName	ProductNumber
Eagle FS-3 Mountain Bike	2
GT RTS-2 Mountain Bike	11
Trek 9000 Mountain Bike	1
Viscount Mountain Bike	6

Entertainment Agency adatbázis

„Sorold fel a legkorábbi időpontú rendezvények szerződésszámát és szerződési összegét!"

Fordítás/ Tisztázás
Select engagement number, ~~and~~ contract price
from ~~the~~ engagements ~~table~~
where ~~the~~ start date ~~is equal to the~~ =
~~earliest~~ (select min(start date) ~~in the~~ from engagements ~~table~~)

SQL
```
SELECT EngagementNumber, ContractPrice
FROM Engagements
WHERE StartDate =
   (SELECT MIN(StartDate) FROM Engagements)
```

CH12_Earliest_Contracts (1 sor)

EngagementNumber	ContractPrice
2	$200.00

„Mennyi volt a 2007 októberében lekötött szerződések teljes értéke?"

Fordítás/ Tisztázás
Select ~~the~~ sum ~~of~~ (contract price) as TotalBookedValue
from ~~the~~ engagements ~~table~~
where ~~the~~ start date ~~is~~ between ~~October 1, 2007~~ '2007-10-01'
and ~~October 31, 2007~~ '2007-10-31'

SQL
```
SELECT SUM(ContractPrice) AS TotalBookedValue
FROM Engagements
WHERE StartDate
   BETWEEN '2007-10-01' AND '2007-10-31'
```

CH12_Total_Booked_Value_For_October_2007 (1 sor)

TotalBookedValue
$27,755.00

School Scheduling adatbázis

„Mekkora a legnagyobb összeg, amit valamelyik alkalmazottnak fizetünk?"

Fordítás/ Tisztázás
Select ~~the~~ max~~imum~~ (salary) as LargestStaffSalary
from ~~the~~ staff ~~table~~

SQL
```
SELECT Max(Salary) AS LargestStaffSalary
FROM Staff
```

CH12_Largest_Staff_Salary (1 sor)

LargestStaffSalary
$60,000.00

„Mekkora a kaliforniai alkalmazottainknak kifizetett teljes összeg?"

Fordítás/ Select ~~the~~ sum ~~of~~ (salary) as TotalAmountPaid
Tisztázás from ~~the~~ staff ~~table~~
~~for all our California staff~~ where state = 'CA'

SQL
```
SELECT SUM(Salary) AS TotalAmountPaid
FROM Staff
WHERE StfState = 'CA'
```

CH12_Total_Salary_Paid_To_California_Staff (1 sor)

TotalAmountPaid
$209,000.00

Bowling League adatbázis

„Hány mérkőzést játszottak a Red Rooster Lanes pályán?"

Fordítás/ Select ~~the~~ count ~~of~~ (tourney location)~~s~~ as NumberOfTournaments
Tisztázás from ~~the~~ tournaments ~~table~~
where ~~the~~ tourney location ~~is~~ = 'Red Rooster Lanes'

SQL
```
SELECT COUNT(TourneyLocation)
   AS NumberOfTournaments
FROM Tournaments
WHERE TourneyLocation = 'Red Rooster Lanes'
```

CH12_Number_Of_Tournaments_At_Red_Rooster_Lanes (1 sor)

NumberOfTournaments
3

„Sorold fel ábécésorrendben azoknak a tekejátékosoknak a vezeték- és keresztnevét, akiknek az átlagos egyéni pontszáma nagyobb, mint az összesített átlagos egyéni pontszám, vagy egyenlő azzal!"

Fordítás/ Select ~~the~~ last name, ~~and~~ first name from ~~the~~ bowlers ~~table~~
Tisztázás where ~~the~~ (select ~~average~~ avg(raw score)
from ~~the~~ bowlers scores ~~table~~ as BS
~~for the current bowler~~ where BS.bowler ID = bowlers.bowler ID)
~~is greater than or equal to the~~ >=
~~overall~~ (select avg(raw score) ~~score in the~~ from bowler scores ~~table~~)
~~sorted~~ order by last name, ~~and~~ first name

SQL
```
SELECT Bowlers.BowlerLastName,
   Bowlers.BowlerFirstName
FROM Bowlers
WHERE (SELECT AVG(RawScore)
FROM Bowler_Scores AS BS
```

```
WHERE BS.BowlerID = Bowlers.BowlerID)
>=(SELECT AVG(RawScore) FROM Bowler_Scores)
ORDER BY Bowlers.BowlerLastName,
         Bowlers.BowlerFirstName
```

CH12_Better_Than_Overall_Average (17 sor)

ProductName	ProductNumber
Cunningham	David
Fournier	David
Hallmark	Alaina
Hallmark	Gary
Hernandez	Michael
Kennedy	Angel
Kennedy	John
Patterson	Kathryn
Patterson	Neil
Patterson	Rachel
Pundt	Steve
Thompson	Mary
Thompson	Sarah
Thompson	William
Viescas	Caleb
Viescas	David
Viescas	John

Megjegyzés *Vegyük észre, hogy a probléma ötletes megoldására a WHERE záradékban két allekérdezést használtunk.*

Recipes adatbázis

„Hány recepthez van szükség marhahúsra?"

Fordítás/ Tisztázás

Select ~~the~~ count (*) ~~of recipes~~ as NumberOfRecipes
from ~~the~~ recipes ~~table~~
where ~~the~~ recipe ID ~~is~~ in ~~the~~
(select~~ion of~~ recipe ID~~s in the~~
from recipe ingredients ~~table~~
inner joined ~~with the~~ ingredients ~~table~~
on recipe_ingredients.ingredient ID
~~in the recipe ingredients table~~
~~matches~~ = ingredients.ingredient ID ~~in the ingredients table~~
where ~~the~~ ingredient name ~~is~~ like 'Beef%')

SQL
```
SELECT COUNT(*) AS NumberOfRecipes
FROM Recipes
WHERE Recipes.RecipeID IN
   (SELECT RecipeID
   FROM Recipe_Ingredients
   INNER JOIN Ingredients ON
      Recipe_Ingredients.IngredientID =
      Ingredients.IngredientID
   WHERE Ingredients.IngredientName
      LIKE 'Beef%')
```

CH12_Recipes_With_Beef_Ingredient (1 sor)

NumberOfRecipes
3

„Hány hozzávalót mérnek csészével?"

Fordítás/ Tisztázás
Select ~~the~~ count (*) ~~of ingredient.~~ as NumberOfIngredients
from ~~the~~ ingredients ~~table~~
inner join~~ed with the~~ measurements ~~table~~
on ingredients.measure amount ID
~~in the ingredients table matches~~
= measurements.measure amount ID
~~in the measurements table~~
where ~~the~~ measurement description ~~is~~ = 'Cup'

SQL
```
SELECT COUNT(*) AS NumberOfIngredients
FROM Ingredients
INNER JOIN Measurements
ON Ingredients.MeasureAmountID =
   Measurements.MeasureAmountID
WHERE MeasurementDescription = 'Cup'
```

CH12_Number_of_Ingredients_Measured_by_the_Cup (1 sor)

NumberOfIngredients
12

Összefoglalás

A fejezetet az összesítő függvények bemutatásával kezdtük. Megtanultuk, hogy hat ilyen függvény van, és azt, hogy miként használhatjuk őket a SELECT utasításban és a SELECT WHERE záradékában. Azt is megtudtuk, hogy – a COUNT(*) kivételével – a feladata végrehajtása közben valamennyi függvény figyelmen kívül hagyja a NULL értékeket.

Ez után bemutattuk az egyes összesítő függvények használatát. Megtanultuk, hogyan számolhatunk sorokat és értékeket a COUNT függvények segítségével, hogyan találhatjuk meg a legkisebb és a legnagyobb értékeket a MIN és a MAX függvényekkel, hogyan számolhatunk középértéket az AVG függvénnyel, és azt is, hogyan összegezhetünk halmazokat a SUM függvénnyel. Mindegyik függvény esetében bemutattuk a DISTINCT módosító használatát, és megemlítettük, hogy ennek a módosítónak nincs hatása a MIN és a MAX függvényre.

A fejezetet az összesítő függvények szűrőkben való használatának bemutatásával zártuk. Most már tudjuk, hogyan használhatunk összesítő függvényeket allekérdezésekben, és hogyan alkalmazhatjuk ezeket az allekérdezéseket szűrők részeként. Arra is fény derült, hogy a szűrőket allekérdezésekben is alkalmazhatjuk, amikor az összesítő függvények értéke egy meghatározott halmaz adatain alapul.

Éppen csak ízelítőt kaptunk abból, hogy mi mindenre lehetünk képesek összesítő függvényekkel. A következő két fejezetben azt mutatjuk be, hogyan szolgáltassunk még kifinomultabb statisztikai adatokat úgy, hogy az összesítő függvényeket csoportosított adatokra alkalmazzuk, és hogyan alkalmazzunk szűrőket összesítő számításokra.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Mekkora a mountain bike kerékpárok átlagos fogyasztói ára?"*
   A megoldás itt található: CH12_Average_Price_Of_A_Mountain_Bike (1 sor).
2. *„Mi volt az utolsó rendelésünk dátuma?"*
   A megoldás itt található: CH12_Most_Recent_Order_Date (1 sor).
3. *„Mekkora volt a 8. számú rendelés végösszege?"*
   A megoldás itt található: CH12_Total_Amount_For_Order_Number_8 (1 sor).

Entertainment Agency adatbázis

1. *„Mekkora a rendezvényszervező ügynökök átlagos fizetése?"*
   A megoldás itt található: CH12_Average_Agent_Salary (1 sor).
2. *„Sorold fel minden olyan rendezvény szerződésszámát, ahol a szerződési összeg nagyobb, mint az összesített átlagos szerződési összeg, vagy egyenlő azzal!"*
   (Tipp: a kérdés megválaszolásához allekérdezés használata szükséges.)
   A megoldás itt található: CH12_Contract_Price_GE_Average_Contract_Price (45 sor).
3. *„Hány előadónk van Bellevue-ben?"*
   A megoldás itt található: CH12_Number_Of_Bellevue_Entertainers (1 sor).

School Scheduling adatbázis

1. *„Milyen hosszú jelenleg az átlagos órahossz?"*
   A megoldás itt található: CH12_Average_Class_Duration (1 sor).
2. *„Sorold fel a tanári kar minden olyan tagjának vezeték- és keresztnevét, akik a legkorábbi felvételi időpont óta velünk vannak!"*
   (Tipp: a megoldáshoz a DateHired oszlopot kiértékelő összesítő függvényt tartalmazó allekérdezésre van szükség).
   A megoldás itt található: CH12_Most_Senior_Staff_Members (3 sor).
3. *„Hány órát tartanak a 3346. teremben?"*
   A megoldás itt található: CH12_Number_Of_Classes_Held_In_Room_3346 (1 sor).

Bowling League adatbázis

1. *„Mekkora a legnagyobb büntetőpontszám, amellyel jelenleg egy tekejátékos rendelkezik?"*
   A megoldás itt található: CH12_Current_Highest_Handicap (1 sor).
2. *„Mely helyszíneken tartottak tornákat a legkorábbi tornaidőpontban?"*
   A megoldás itt található: CH12_Tourney_Locations_For_Earliest_Date (2 sor).
3. *„Mi az időrendben található legkésőbbi torna időpontja?"*
   A megoldás itt található: CH12_Last_Tourney_Date (1 sor).

Recipes adatbázis

1. *„Melyik recepthez kell a legtöbb gerezd fokhagyma?"*
   (Tipp: a megoldáshoz több INNER JOIN és egy allekérdezés alkalmazására lesz szükség.)
   A megoldás itt található: CH12_Recipe_With_Most_Cloves_of_Garlic (1 sor).
2. *„Számold össze a főételek receptjeit!"*
   (Tipp: a megoldáshoz a Recipe_Classes és a Recipes táblák közötti JOIN alkalmazására lesz szükség.)
   A megoldás itt található: CH12_Number_Of_Main_Course_Recipes (1 sor).
3. *„Számold ki, hány teáskanál sóra van szükség az összes recepthez!"*
   A megoldás itt található: CH12_Total_Salt_Used (1 sor).

13

Adatok csoportosítása

„Ne merülj el a részletekben – nézd a teljes képet!"
– Ferdinand Foch marsall, a Franciaországban állomásozó szövetséges csapatok főparancsnoka

A fejezet témakörei

- Miért csoportosítsuk az adatokat?
- A GROUP BY záradék
- „Bizonyos korlátozásokkal"
- Példák
- Összefoglalás
- Önálló feladatok

A 12. fejezet azt magyarázta el, hogy az összesítő függvények (COUNT, MIN, MAX, AVG és SUM) segítségével hogyan kérhetjük az SQL-t arra, hogy számítson ki egy értéket a FROM és WHERE záradékban meghatározott tábla soraiból. Ugyanakkor rámutattunk, hogy ha a SELECT záradékban olyan értékkifejezést használunk, amely összesítő függvényt tartalmaz, akkor vagy *minden* értékkifejezésünknek literális értéknek kell lennie, vagy összesítő függvényt kell tartalmazniuk. Ennek a jellegzetességnek jó hasznát vehetjük, ha csak összesítések egyetlen sorát szeretnénk látni, de mi a helyzet akkor, ha a részösszegekre is kíváncsiak vagyunk? Ebben a fejezetben azt mutatjuk be, hogyan állíthatunk elő ilyen részösszegeket az adatok csoportosítása révén.

Miért csoportosítsuk az adatokat?

Ha a Sales Orders adatbázissal dolgozunk, valóban hasznos lehet, ha ki tudjuk mutatni a rendelések számát (COUNT), az eladások összegét (SUM), az eladások átlagát (AVG), és a legkisebb (MIN) vagy a legnagyobb rendelés (MAX) összegét. Ha ki akarjuk számolni ezen értékek bármelyikét egy vásárlóra, megrendelési dátumra vagy termékre vonatkozóan, bevezethetünk egy szűrőt (WHERE), ami kigyűjti az adott vásárlóra vagy termékre

vonatkozó sorokat. De mi van akkor, ha az *összes* vásárlóra vonatkozó részeredményeket szeretnénk látni, a vásárlók neveivel a részeredmény mellett? Ehhez arra kell kérnünk az adatbázis-kezelőt, hogy *csoportosítsa* a sorokat.

Hasonlóképpen, az Entertainment Agency adatbázisból könnyen készíthetünk kimutatást a rendezvények számáról, összesíthetjük a szerződési összegeket, vagy megkereshetjük a legalacsonyabb és a legmagasabb szerződési összegeket az összes rendezvényre vonatkozóan. Még szűrhetjük is a sorokat, ha ezeket a számításokat egy bizonyos előadóra vagy megrendelőre, esetleg egy adott dátumra vonatkozóan szeretnénk látni. De ha minden egyes megrendelőre vagy előadóra vonatkozóan egy-egy sorösszeget szeretnénk látni, megint csak csoportosítanunk kell a sorokat.

EntStageName	ContractPrice
Carol Peacock Trio	$140.00
Carol Peacock Trio	$1,670.00
Carol Peacock Trio	$770.00
Carol Peacock Trio	$1,670.00
Carol Peacock Trio	$1,670.00
Carol Peacock Trio	$320.00
Carol Peacock Trio	$1,400.00
Carol Peacock Trio	$680.00
Carol Peacock Trio	$410.00
Carol Peacock Trio	$1,940.00
Carol Peacock Trio	$410.00
Caroline Coie Cuartet	$1,250.00
Caroline Coie Cuartet	$2,450.00
Caroline Coie Cuartet	$1,490.00
Caroline Coie Cuartet	$1,370.00
<< további sorok >>	

Kezd világossá válni a dolog? Amikor arra kérjük az adatbázis-kezelőt, hogy oszlopértékek vagy kifejezések alapján csoportosítsa a sorokat, az a sorokból részhalmazokat hoz létre a megegyező értékek alapján. Ezután megkérhetjük az adatbázis-kezelőt, hogy minden csoportra számítsa ki az összesített értékeket. Lássunk egy egyszerű példát az Entertainment Agency adatbázisból! Először is egy olyan lekérdezést kell felépítenünk, amely kiválasztja a lényeges oszlopokat: az előadó nevét és a szerződés árát. Íme az SQL kód:

SQL
```
SELECT Entertainers.EntStageName,
Engagements.ContractPrice
FROM Entertainers
```

```
INNER JOIN Engagements
ON Entertainers.EntertainerID =
Engagements.EntertainerID
ORDER BY EntStageName
```

Az eredmény valami olyasmi lesz, mint ami az alábbi táblázatban látható. (A mintaadatbázisban ezt a kérelmet CH13_Entertainers_And_ContractPrices néven találjuk meg.)

Már tudjuk, hogy megszámolhatjuk az összes sort, vagy megkereshetjük a ContractPrice (Szerződéses ár) oszlop legkisebb, illetve legnagyobb elemét, összegét vagy átlagát – feltéve, hogy kiküszöböljük az EntStageName (Előadó művészneve) oszlopot. De akár meg is tarthatjuk az oszlopot, feltéve, hogy arra kérjük az adatbázis-kezelőt, hogy ennek alapján alkosson csoportokat. Ha az előadó művészneve szerint kérünk csoportosítást, az adatbázis-kezelő létrehoz egy csoportot (vagyis együttest) az első tizenegy sorból („Carol Peacock Trio"), egy másodikat a következő tizenegyből („Caroline Coie Cuartet"), és így tovább, végig az egész táblán. Ezután már kérhetjük a sorok megszámlálását (COUNT), vagy a ContractPrice oszlop SUM, MIN, MAX vagy AVG függvényét, és egyetlen összesített sort fogunk kapni előadónként. Az eredmény a következő táblázatban látható:

EntStageName	NumContracts	TotPrice	MinPrice	MaxPrice	AvgPrice
Carol Peacock Trio	11	$11,080.00	$140.00	$1,940.00	$1,007.27
Caroline Coie Cuartet	11	$15,070.00	$290.00	$2,450.00	$1,370.00
Coldwater Cattle Company	8	$14,875.00	$350.00	$3,800.00	$1,859.38
Country Feeling	15	$34,080.00	$275.00	$14,105.00	$2,272.00
Jazz Persuasion	7	$5,480.00	$500.00	$1,670.00	$782.86
Jim Glynn	9	$3,030.00	$110.00	$770.00	$336.67
Julia Schnebly	8	$4,345.00	$275.00	$875.00	$543.13
JV & the Deep Six	10	$17,150.00	$950.00	$3,650.00	$1,715.00
Modern Dance	10	$14,600.00	$650.00	$2,930.00	$1,460.00
Saturday Revue	9	$11,550.00	$290.00	$2,930.00	$1,283.33
Susan McLain	6	$2,670.00	$230.00	$800.00	$445.00
Topazz	7	$6,620.00	$590.00	$1,550.00	$945.71
<< további sorok >>					

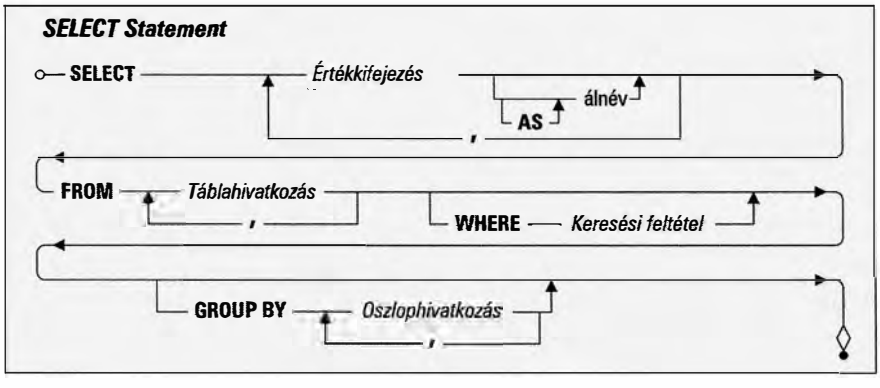
Érdekesnek tűnik, ugye? Fogadjunk, hogy tudni szeretnénk, hogyan csináltuk ezt! Nos, a következő részekben felfedjük a részleteket.

A GROUP BY záradék

Ahogyan a 12. fejezetben felfedeztük, sokféle érdekes információt szerezhetünk az összesítő függvények segítségével. Ugyanakkor megfigyelhettük, hogy mindegyik példánk a FROM és a WHERE záradékok által visszaadott *összes* soron végrehajtotta az összesítő függvényeket. Szűrhettük ugyan az eredményt egyetlen csoportra a WHERE záradékkal, de igazából nem volt módunk egy kérelemben megnézni a többféle csoportból származó eredményeket. Ahhoz, hogy képesek legyünk egy kérelmen belül csoportok szerint összegezni, szükségünk lesz arra, hogy felvegyünk még egy fontos elemet az SQL-szótárunkba – a GROUP BY záradékot.

Utasításforma

Nézzük meg alaposabban a GROUP BY záradékot! A SELECT utasításhoz hozzáadott GROUP BY alapvető utasításformáját a 13.1. ábra mutatja.



13.1. ábra

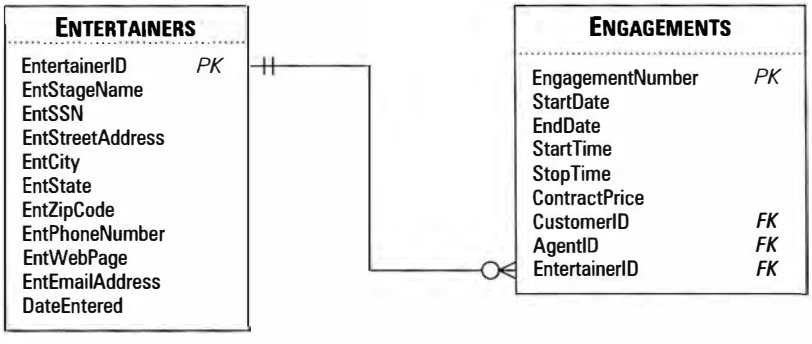
GROUP BY záradékkal ellátott SELECT utasítás szintaxisdiagramja

A korábbi fejezetekből már tudjuk, hogy az adataink forrásául szolgáló táblákat a FROM záradékban határozzuk meg. A FROM záradékban állhat egyetlen tábla neve, de akár összetett, JOIN segítségével többszörösen összekapcsolt táblák is. Ahogy a 8. fejezetben tárgyaltuk, akár egy teljes tábla-allekérdezést (azaz egy SELECT utasítást) is beágyazhatunk táblahivatkozásként. Ez után szükség esetén megadhatunk egy WHERE záradékot is, ahol eldöntjük, hogy eldobjuk vagy megtartjuk a FROM záradék megfelelő eredménysorait. A WHERE záradékkal a 6. fejezetben foglalkoztunk részletesen.

Amikor egy GROUP BY záradékot adunk az utasításhoz, azt írjuk elő az adatbázis-kezelőnek, hogy a FROM és WHERE záradékok által létrehozott logikai tábla mely oszlopait használja a sorcsoportok meghatározásához. Azok a sorok, amelyek az általunk előírt oszlopokban ugyanazt az értéket tartalmazzák, egy csoportot fognak alkotni. A GROUP BY

záradékban felsorolt oszlopokat értékkifejezésekben használhatjuk a SELECT záradékban, és mindegyik csoporton az előző fejezetben tárgyalt bármelyik összesítő függvénnyel végezhetünk számításokat.

Nézzük meg, hogy a GROUP BY záradék alkalmazásával hogyan számíthatjuk ki együttesenként (előadócsoportonként) a szerződési összegekre vonatkozó adatokat – ez az a példa, amivel már korábban kínoztuk magunkat. A 13.2. ábrán látható, hogy mely táblák szükségesek a probléma megoldásához.



13.2. ábra

Az Entertainers és Engagements táblák közötti kapcsolat

Megjegyzés *Ebben a fejezetben végig a 4. fejezetben bevezetett „Kérelem – Fordítás – Tisztázás – SQL" módszert követjük.*

„Írd ki minden együttes nevét, az együttes szerződéseinek számát, a szerződések összesített összegét, a legalacsonyabb és a legmagasabb szerződési összeget és az összes szerződésükre vonatkozó átlagos szerződési összeget!"

(Tipp: amikor olyan kérelemmel van dolgunk, ahol megszámlálunk, összegzünk, legkisebb vagy legnagyobb értéket számítunk, vagy az átlagos szerződési összeget kell meghatároznunk a szerződések[contracts] részletei alapján, de magasabb [entertainers] szinten, akkor a kérelemben összesítő függvényeket és csoportosítást kell alkalmaznunk. Ne felejtsük el, hogy valószínűleg minden előadó több szerződéssel rendelkezik.)

Fordítás Select entertainer name, the count of contracts, the sum of the contract price, the lowest contract price, the highest contract price, and the average contract price from the entertainers table joined with the engagements table on entertainer ID, grouped by entertainer name
(Válaszd ki az előadó nevét, a szerződések számát, az összesített szerződési összeget, a legalacsonyabb szerződési összeget, a legmagasabb szerződési

összeget és az átlagos szerződési összeget az előadók és a rendezvények tábláját az előadóazonosító szerint összekapcsolva, és csoportosítsd az eredményt az előadók neve alapján.)

Tisztázás Select entertainer name, ~~the~~ count ~~of~~ (*) ~~contracts,~~
~~the~~ sum ~~of the~~ (contract price),
~~the lowest~~ min(contract price),
~~the highest~~ max(contract price),
~~and the average~~ avg(contract price)
from ~~the~~ entertainers ~~table~~
inner join~~ed with the~~ engagements ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= engagements.entertainer ID ~~in the engagements table,~~
group~~ed~~ by entertainer name

SQL
```
SELECT Entertainers.EntStageName,
   COUNT(*) AS NumContracts,
   SUM(Engagements.ContractPrice) AS TotPrice,
   MIN(Engagements.ContractPrice) AS MinPrice,
   MAX(Engagements.ContractPrice) AS MaxPrice,
   AVG(Engagements.ContractPrice) AS AvgPrice
FROM Entertainers
INNER JOIN Engagements
ON Entertainers.EntertainerID =
   Engagements.EntertainerID
GROUP BY Entertainers.EntStageName
```

Vegyük észre, hogy a „legalacsonyabb" szót a MIN függvénnyel, a „legmagasabb"-at MAX függvénnyel, az „átlag"-ot pedig az AVG függvénnyel helyettesítettük, ahogyan azt már az előző fejezetben bemutattuk. A COUNT(*) függvényt is alkalmaztuk, mivel meg akartuk számlálni az összes szerződés (contract) sorát, függetlenül az esetleges NULL értékektől. A GROUP BY záradék hozzáadása az, ami lehetővé teszi, hogy az *előadók csoportjain* (együtteseken) hajtsunk végre összesítő számításokat. Az előadók nevét is ennek a záradéknak köszönhetően használhatjuk a SELECT utasításban. (A mintaadatbázisban ezt a kérelmet CH13_Aggregate_Contract_Info_By_Entertainer néven találjuk meg.)

Arra számítottunk, hogy a fenti lekérdezés minden előadóra külön sort ad vissza? Mi a helyzet azokkal az előadókkal, akiknek nincs lekötött fellépésük? Visszagondolva a 9. fejezetben az OUTER JOIN szerkezettel kapcsolatban tanultakra, esetleg a következő megoldással próbálkoznánk:

SQL
```
SELECT Entertainers.EntStageName,
   COUNT(*) AS NumContracts,
   SUM(Engagements.ContractPrice) AS TotPrice,
```

```
    MIN(Engagements.ContractPrice) AS MinPrice,
    MAX(Engagements.ContractPrice) AS MaxPrice,
    AVG(Engagements.ContractPrice) AS AvgPrice
FROM Entertainers
LEFT OUTER JOIN Engagements
ON Entertainers.EntertainerID =
    Engagements.EntertainerID
GROUP BY Entertainers.EntStageName
```

Érdekes dolog az összesítő függvényekkel kapcsolatban, hogy figyelmen kívül hagyják a NULL értékű sorokat. A fenti lekérdezés NULL értéket ad vissza a TotPrice, MinPrice, MaxPrice és AvgPrice értékekre azoknak az előadóknak az esetében, akiknek nincs lekötött fellépésük, de azt tapasztaljuk, hogy a NumContracts 1 értéket vesz fel! Hogy lehetséges ez?

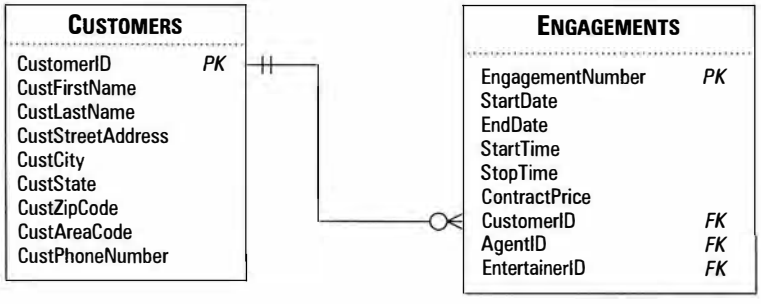
Nos, ez az SQL utasítás a COUNT(*) függvényt használja – azaz az összes visszaadott sort megszámoljuk. Az OUTER JOIN pontosan egy sort ad vissza azoknál az előadóknál, akiknek nincs lekötött fellépésük, ezért a COUNT által visszaadott 1 helyes érték. De ha visszaemlékszünk az előző fejezetre, eszünkbe juthat a COUNT(*értékkifejezés*) függvény is, ami arra utasítja az adatbázis-kezelő rendszert, hogy a megadott értékkifejezés vagy oszlopnév összeszámlálásánál csak a NULL értéktől különböző sorokat vegye figyelembe. Finomítsuk tovább a lekérdezést:

SQL

```
SELECT Entertainers.EntStageName,
    COUNT(Engagements.EntertainerID) AS NumContracts,
    SUM(Engagements.ContractPrice) AS TotPrice,
    MIN(Engagements.ContractPrice) AS MinPrice,
    MAX(Engagements.ContractPrice) AS MaxPrice,
    AVG(Engagements.ContractPrice) AS AvgPrice
FROM Entertainers
LEFT OUTER JOIN Engagements
ON Entertainers.EntertainerID =
    Engagements.EntertainerID
GROUP BY Entertainers.EntStageName
```

Mivel az Engagements (Rendezvények) táblában található EntertainerID (Előadóazonosító) oszlop NULL értékű azoknak az előadóknak az esetében, akiknek nincs lekötött fellépésük, semmit sem kell megszámolni. Ha lefuttatjuk a lekérdezést, látjuk, hogy a NumContracts a helyes, 0 értéket veszi fel, ha az előadónak nincs lekötött fellépése.

Mi a helyzet akkor, ha egynél több érték alapján akarunk (vagy szükséges) csoportosítani? Nézzük még egyszer ugyanezt a problémát, de most a megrendelők, és nem az előadók szemszögéből, és tételezzük fel, hogy az eredményhalmazban fel szeretnénk tüntetni a megrendelő keresztnevét és vezetéknevét is. A szükséges táblákat a 13.3. ábra mutatja.



13.3. ábra

A Customers és Engagements táblák közötti kapcsolat

„Sorold fel minden megrendelő kereszt- és vezetéknevét, a megrendelő szerződéseinek számát és szerződési összegét, a legalacsonyabb és a legmagasabb szerződési összeget, és a megrendelő összes szerződésére vonatkozó átlagos szerződési összeget!"

Fordítás Select customer last name, customer first name, the count of contracts, the sum of the contract price, the lowest contract price, the highest contract price, and the average contract price from the customers table joined with the engagements table on customer ID, grouped by customer last name and customer first name (Válaszd ki a megrendelő keresztnevét, vezetéknevét, a szerződések számát, az összesített szerződési összegeket, a legalacsonyabb szerződési összeget, a legmagasabb szerződési összeget, és az átlagos szerződési összeget a megrendelők és a rendezvények táblájából, amelyet a megrendelő-azonosító alapján összekapcsoltunk, és csoportosítsd az eredményt a megrendelők vezetékneve és keresztneve alapján.)

Tisztázás Select customer last name, customer first name,
~~the~~ count ~~of~~ (*) ~~contracts~~, ~~the~~ sum ~~of the~~ (contract price),
~~the lowest~~ min(contract price), ~~the highest~~ max(contract price),
~~and the average~~ avg(contract price)
from ~~the~~ customers ~~table~~
inner join~~ed with the~~ engagements ~~table~~
on customers.customer ID ~~in the customers table matches~~
= engagements.customer ID ~~in the engagements table,~~
group~~ed~~ by customer last name, ~~and~~ customer first name

SQL
```
SELECT Customers.CustLastName,
   Customers.CustFirstName,
   COUNT(*) AS NumContracts,
   SUM(Engagements.ContractPrice) AS TotPrice,
   MIN(Engagements.ContractPrice) AS MinPrice,
   MAX(Engagements.ContractPrice) AS MaxPrice,
   AVG(Engagements.ContractPrice) AS AvgPrice
```

```
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID =
   Engagements.CustomerID
GROUP BY Customers.CustLastName,
Customers.CustFirstName
```

Az eredmény a következő táblázatban látható (a mintaadatbázisban ezt a kérelmet CH13_Aggregate_Contract_Info_By_Customer néven találjuk meg):

CustLastName	CustFirstName	NumContracts	TotPrice	MinPrice	MaxPrice	AvgPrice
Berg	Matt	9	$13,170.00	$200.00	$2,675.00	$1,463.33
Brehm	Peter	7	$7,250.00	$290.00	$3,800.00	$1,035.71
Ehrlich	Zachary	13	$12,455.00	$230.00	$1,550.00	$958.08
Hallmark	Elizabeth	8	$25,585.00	$410.00	$14,105.00	$3,198.13
Hartwig	Doris	8	$10,795.00	$140.00	$2,750.00	$1,349.38
Keyser	Liz	7	$4,685.00	$200.00	$1,490.00	$669.29
McCrae	Dean	11	$11,800.00	$290.00	$2,570.00	$1,072.73
Patterson	Kerry	7	$6,815.00	$110.00	$2,930.00	$973.57
<< további sorok >>						

Mivel a megrendelő nevének megjelenítéséhez két oszlopra van szükségünk, ***mindkettőt*** fel kellett használjuk a GROUP BY záradékban. Emlékezzünk vissza, hogy ha olyan oszlopot akarunk látni a kimenetben, ami nem összesítő számítás eredménye, akkor az oszlopot a GROUP BY záradékba is be kell építeni. A ContractPrice oszlopot azért nem építettük be a GROUP BY záradékba, mert számos összesítő függvényt tartalmazó kifejezésben felhasználtuk. Ha valóban beépítettük volna a ContractPrice oszlopot, megrendelők és összegek egyedi csoportjait kaptuk volna. A MIN, MAX és AVG függvények mind ezeket a csoportosított összegeket adják vissza. A COUNT csak akkor lesz egynél nagyobb, ha egy adott megrendelőhöz egynél több azonos összegű szerződés tartozik. Ha kicsit jobban elgondolkodunk ezen, rájöhetünk, hogy a megrendelő és összeg szerinti csoportosítás, majd a COUNT függvénnyel való összesítés jó módszer lehet azoknak a megrendelőknek a megkeresésére, akiknek több, azonos összegű szerződésük van.

Azt gondolnánk, hogy ez a lekérdezés azokat a megrendelőket is tartalmazza, akik nem kötöttek le fellépést, de nem így van! Ha valamennyi megrendelő adatait meg szeretnénk kapni, tekintet nélkül arra, hogy kötöttek-e szerződést vagy sem, akkor az OUTER JOIN használatára lesz szükségünk, és körültekintően meg kell számolnunk az Engagements tábla valamelyik oszlopát a COUNT segítségével. A megoldás hasonló ahhoz, amit már korábban ismertettünk az előadókkal és a rendezvényekkel kapcsolatban.

Oszlopok és kifejezések vegyes használata

Tegyük fel, hogy egy oszlopban szeretnénk felsorolni a megrendelő nevét, egy másikban oszlopban a megrendelő címét, a legutóbbi szerződés dátumát, és az összesített szerződési összegeket. A megrendelő neve a következő két oszlopból áll: CustFirstName (Megrendelő keresztneve) és CustLastName (Megrendelő vezetékneve). A teljes címhez szükséges oszlopok a CustStreetAddress (cím), a CustCity (város), a CustState (állam) és a CustZipCode (irányítószám). Lássuk, hogyan foghatunk neki a lekérdezéshez szükséges SQL-utasítás megalkotásának (a kérelmet a mintaadatbázisban CH13_Customers_Last_Booking néven találjuk meg):

„Írd ki minden megrendelő teljes nevét, teljes címét, a legutóbbi szerződés dátumát, és a szerződések teljes összegét!"

Fordítás Select customer last name and customer first name as CustomerFullName; street address, city, state, and ZIP Code as CustomerFullAddress; the latest contract start date; and the sum of the contract price from the customers table joined with the engagements table on customer ID, grouped by customer last name, customer first name, customer street address, customer city, customer state, and customer ZIP Code (Válaszd ki a megrendelő keresztnevét és vezetéknevét CustomerFullName néven; az utcanevet, várost, államot és irányítószámot CustomerFullAddress néven; a legutóbbi szerződés kezdő időpontját; továbbá az összesített szerződési összegeket a megrendelők és a rendezvények tábláiból, amelyeket a megrendelő-azonosító alapján összekapcsoltunk, és csoportosítsd az eredményt a megrendelők vezetékneve, keresztneve, illetve a megrendelő lakcímének utcaneve, városa, állama és irányítószáma alapján.)

Tisztázás Select customer last name ~~and~~ || ',' || customer first name as CustomerFullName, street address~~,~~ || ', ' || city~~,~~ || ', ' || state~~,~~ ~~and~~ || ' ' || ZIP Code as CustomerFullAddress, ~~the latest~~ max(contract start date) as latest date, ~~and the~~ sum ~~of the~~ (contract price) as total contract price from ~~the~~ customers ~~table~~ inner join~~ed with the~~ engagements ~~table~~ on customers.customer ID ~~in the customers table matches~~ = engagements.customer ID ~~in the engagements table~~ group~~ed~~ by customer last name, customer first name, customer street address, customer city, customer state, ~~and~~ customer ZIP Code

SQL

```
SELECT Customers.CustLastName || ', ' ||
   Customers.CustFirstName AS CustomerFullName,
   Customers.CustStreetAddress || ', ' ||
   Customers.CustCity || ', ' ||
```

```
    Customers.CustState || ' ' ||
    Customers.CustZipCode AS CustomerFullAddress
    MAX(Engagements.StartDate) AS LatestDate,
    SUM(Engagements.ContractPrice),
    AS TotalContractPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID =
    Engagements.CustomerID
GROUP BY Customers.CustLastName,
    Customers.CustFirstName,
    Customers.CustStreetAddress,
    Customers.CustCity, Customers.CustState,
    Customers.CustZipCode
```

Figyeljük meg, hogy fel kellett sorolnunk a kimeneti kifejezésben használt minden egyes olyan oszlopot, amelyet nem építettünk be valamelyik összesítő függvénybe. A StartDate (Kezdő dátum) és a ContractPrice (Szerződéses ár) oszlopokat használtuk összesítő kifejezésben, ezért nem kellett őket a GROUP BY záradékban felsorolnunk. Tulajdonképpen nem lett volna túl sok értelme sem a StartDate, sem a ContractPrice alapján csoportosítani, mivel ezeket az oszlopokat a több megrendelőn végrehajtott összesítő számításainkban amúgy is felhasználtuk. Ha például a StartDate alapján csoportosítottunk volna, a MAX(StartDate) a csoportosított értéket adta volna vissza, a SUM(ContractPrice) pedig csak a megrendelő egy adott dátumra vonatkozó szerződéseinek összegét. Nem kaptuk volna meg egynél több szerződés összegét, hacsak a vevőnek – nem túl valószínű módon – egynél több szerződése is lett volna a megadott időpontban.

A GROUP BY záradék használata WHERE záradékban található allekérdezésben

A 11. fejezetben bevezettük a COUNT és a MAX összesítő függvényeket, és bemutattuk, hogyan szűrhetünk sorokat allekérdezésből származó összesített érték segítségével, a 12. fejezetben pedig azt mutattuk meg, hogyan használjuk a MIN, a MAX és az AVG függvényt allekérdezésekben. Nézzünk most egy kérelmet, amelyhez egyaránt szükség van egy összesítő függvényt használó allekérdezésre és az allekérdezésben megadott GROUP BY záradékra:

„Sorold fel azokat a rendezvényszerződéseket, amelyeknek az összege nagyobb, mint bármely más megrendelő szerződéseinek összege!"

Fordítás Select customer first name, customer last name, engagement start date, and engagement contract price from the customers table joined with the engagements table on customer ID where the contract price is greater than the sum of all contract prices from the engagements table for customers other than the current customer, grouped by customer ID

(Válaszd ki a megrendelők keresztnevét és vezetéknevét, valamint a rendezvény kezdő időpontját és szerződési összegét a megrendelők és a rendezvények táblájából, amelyeket összekapcsoltunk a megrendelőazonosítók alapján, ha a szerződéses összeg nagyobb, mint az összesített szerződési összeg a rendezvények táblájából, az összes megrendelőre vonatkoztatva – kivéve az aktuális megrendelőt –, végül az eredményt csoportosítsd a megrendelőazonosító alapján.)

Tisztázás

Select customer first name, customer last name,
engagement start date, ~~and~~ engagement contract price
from ~~the~~ customers ~~table~~
inner ~~joined with the~~ engagements ~~table~~
on customers.customer ID ~~in the customers table matches~~
= engagements.customer ID ~~in the engagements table~~
where ~~the~~ contract price ~~is greater than~~ > ALL
(select ~~the~~ sum ~~of all~~ (contract price~~s~~)
from ~~the~~ engagements ~~table~~ as E2
~~for~~ where E2.customer~~s~~ ID <>
~~other than the current~~ customers.customer ID~~;~~
group~~ed~~ by E2.customer ID)

SQL

```
SELECT Customers.CustFirstName,
   Customers.CustLastName,
   Engagements.StartDate,
   Engagements.ContractPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID =
   Engagements.CustomerID
WHERE Engagements.ContractPrice > ALL
   (Select SUM(ContractPrice)
   FROM Engagements AS E2
   WHERE E2.CustomerID <> Customers.CustomerID
   GROUP BY E2.CustomerID)
```

Elemezzük, mit is csinál az allekérdezés! Minden, a lekérdezés által összekapcsolt Customers és Engagements táblákban megvizsgált rendezvény esetében az allekérdezés a SUM segítségével, az *összes többi* megrendelőre vonatkozóan összegzi a szerződési összegeket, majd a megrendelőazonosító alapján csoportosítja az eredményt. Mivel sok megrendelő található az adatbázisban, az allekérdezés több SUM értékkel fog visszatérni – minden megrendelőhöz eggyel (természetesen az aktuális megrendelőt kivéve) –, ezért egy egyszerű nagyobb, mint (>) összehasonlítás nem lenne elegendő. Szerencsére – ahogy azt már a 11. fejezetben tanultuk – használhatjuk a halmazokon értelmezett nagyobb, mint (> ALL) összehasonlítást

az értékhalmaz ellenőrzésére. Ha végrehajtuk a lekérdezést az Entertainment Agency mintaadatbázison (ennek eredményét a CH13_Biggest_Big_Contract táblába mentettük), úgy találjuk, hogy csak egy szerződés felel meg a kívánalmainknak; ezt láthatjuk lejjebb.

CustFirstName	CustLastName	StartDate	ContractPrice
Elizabeth	Hallmark	2008-01-22	$14,105.00

A SELECT DISTINCT utasítás kiváltása

Elgondolkoztunk már azon, hogy esetleg anélkül használhatnánk a GROUP BY záradékot, hogy akár egyetlen összesítő függvényt tartalmazna a SELECT utasítás? Semmi akadálya! Amikor így teszünk, ugyanazt az eredményt kapjuk, mintha a 4. fejezetben bemutatott DISTINCT kulcsszót használnánk.

Lássunk egy egyszerű kérelmet, amely egyedi értékeket használ, és oldjuk meg mindkét módszerrel:

„Sorold fel a vásárlók táblájában található egyedi városneveket!"

Fordítás 1. Select the unique city names from the customers table
(Válassszuk ki az egyedi városneveket a vásárlók táblájából.)

Tisztázás Select ~~the unique~~ distinct city names
from ~~the~~ customers ~~table~~

SQL
```
SELECT DISTINCT Customers.CustCityName
FROM Customers
```

Fordítás 2. Select city name from the customers table,
grouped by city name
(Válaszd ki a városneveket a vásárlók táblájából, és csoportosítsd az eredményt városnevek szerint.)

Tisztázás Select city name
from ~~the~~ customers ~~table~~,
group~~ed~~ by city name

SQL
```
SELECT Customers.CustCityName
FROM Customers
GROUP BY Customers.CustCityName
```

Ha visszagondolunk, a GROUP BY a sorokat a csoportosító oszlop(ok) alapján csoportosítja, és csoportonként egy sort ad vissza. Ez a módszer kicsit eltér attól, ahogy a DISTINCT kulcsszót használjuk, de az eredmény ugyanaz. De melyik a jobb? Az gondolnánk, hogy a DISTINCT tisztább kifejezése annak, hogy egyedi sorokat szeretnénk kapni, de azt tapasztaljuk, hogy az adatbázis-kezelő gyorsabban megoldja a problémát, ha a GROUP BY alkalmazását választjuk, ráadásul a GROUP BY segítségével több információt kapunk az adatainkról. Nézzük meg a következő lekérdezést:

```
SQL     SELECT Customers.CustCityName, Count(*) as
        CustPerCity
        FROM Customers
        GROUP BY Customers.CustCityName
```

Ezzel a lekérdezéssel nem csak az egyedi városneveket kapjuk meg, de azt is megtudjuk, hogy melyik városban hány megrendelő van. Hát nem nagyszerű?

„Bizonyos korlátozásokkal"

Már említettük, hogy a GROUP BY záradék bizonyos megkötéseket jelent a kérelmek megalkotásakor. Nézzük, melyek ezek a korlátozások, nehogy valamelyik gyakori hiba áldozataivá váljunk!

Oszlopokra vonatkozó megkötések

Amikor a GROUP BY záradékot használjuk, arra kérjük az adatbázis-kezelő rendszert, hogy alkosson egyedi csoportokat a FROM záradékból származó és a WHERE záradékban szűrt sorokból. A SELECT záradékaiban tetszés szerint használhatunk összesítő függvényeket, és ezek a kifejezések a FROM és a WHERE záradékaiban meghatározott oszlopok bármelyikét felhasználhatják. Ahogy egy korábbi példában már rámutattunk, nem lenne túl ésszerű egy oszlopra összesítő kifejezésben hivatkozni, és az oszlopot a csoportosító hivatkozások közé is beépíteni.

Ha olyan kifejezéseket használunk, amelyek oszlophivatkozást tartalmaznak, de összesítő függvényt nem, akkor fel kell sorolnunk az összes ilyen módon használt oszlopot a GROUP BY záradékban. A leggyakoribb hibák közé tartozik, hogy azt hisszük, nyugodtan hivatkozhatunk oszlopokra a nem összesített kifejezésekben, amíg az oszlopok egyedi sorokból származnak. Nézzük például az alábbi, helytelen kérelmet, amely egy elsődleges kulcsértéket tartalmaz – amiről tudjuk, hogy meghatározásából adódóan egyedi:

„Írd ki a megrendelőazonosítót, a megrendelő teljes nevét, és a szerződési összegek teljes összegét!"

Fordítás Select customer ID, customer first name, and customer last name as CustFullName, and the sum of contract prices as TotalPrice from the customers table joined with the engagements table on customer ID, grouped by customer ID
(Válaszd ki a megrendelőazonosítót, a megrendelő keresztnevét és vezetéknevét CustFullName néven és az összesített szerződési összegeket TotalPrice néven a megrendelők és a rendezvények táblájából, amelyeket a megrendelőazonosító alapján összekapcsoltunk, majd az eredményt csoportosítsd a megrendelőazonosító alapján.)

Tisztázás Select customer ID, customer first name
~~and~~ || ' ' || customer last name as CustFullName,

and the sum of (contract price)s as TotalPrice
from the customers table
inner joined with the engagements table
on customers.customer ID in the customers table matches
= engagements.customer ID in the engagements table,
grouped by customer ID

SQL
```
SELECT Customers.CustomerID,
   Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   SUM(Engagements.ContractPrice) AS TotalPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID =
   Engagements.CustomerID
GROUP BY Customers.CustomerID
```

Tudjuk, hogy a CustomerID (Megrendelőazonosító) megrendelőnként egyedi. Elégséges kellene, hogy legyen, ha a CustomerID alapján csoportosítanánk, ahhoz, hogy a megrendelők vezeték- és keresztnevére vonatkozó egyedi információkat kapjunk a CustomerID alapján kialakított csoportokon belül. Sajnos azonban az SQL nyelv a nyelvtani formán (a szintaxison), és nem a jelentésen (azaz a szemantikán) alapul. Más szóval az SQL nem képes figyelembe venni az adatbázis tábláiba „beletervezett" információkat – amibe az is beletartozik, hogy mely oszlopok szolgálnak elsődleges kulcsként. Az SQL megköveteli, hogy a kérelem nyelvtana „tiszta" legyen, azaz le lehessen fordítani az adatbázistáblák szerkezetének ismerete nélkül. Ennek megfelelően a fenti SQL-utasítás hibát okoz az SQL-szabványnak megfelelő adatbázis-kezelő rendszerekben, mivel a SELECT záradékában olyan oszlopokra hivatkoztunk, amelyek sem összesítő függvényben, sem a GROUP BY záradékban nem fordulnak elő (CustFirstName és CustLastName). A helyes SQL-kérelem a következő:

SQL
```
SELECT Customers.CustomerID,
   Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   SUM(Engagements.ContractPrice) AS TotalPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID =
   Engagements.CustomerID
GROUP BY Customers.CustomerID,
   Customers.CustFirstName,
   Customers.CustLastName
```

Lehet, hogy ez túlzásnak tűnik, de ez a helyes módszer!

Megjegyzés *Egyes adatbázis-kezelő rendszerekben a GROUP BY záradékban a SELECT záradékban használt kifejezések pontos másolatát kell megadnunk. Ilyen például az Oracle vagy a Microsoft Office Access. A fenti példánkban ahelyett, hogy az egyes oszlopokat felsoroltuk volna, az SQL-utasítást a következőképpen kellett volna befejeznünk:*

```
GROUP BY Customers.CustomerID,
   Customers.CustFirstName || ' ' ||
   Customers.CustLastName
```

Ez ugyan nem felel meg az SQL-szabványnak, de előfordulhat, hogy csak így tudjuk működésre bírni a kérelmet a rendszerünkön.

Kifejezéseken alapuló csoportosítás

Korábban már mutattunk néhány működő példát olyan kifejezések megalkotására, amelyek nem tartalmaznak összesítő függvényeket. A leggyakoribb hibák egyike, ha a SELECT záradékban létrehozott kifejezés alapján próbálunk csoportosítani, ahelyett, hogy az egyedi oszlopokat vennénk alapul. Ne feledkezzünk meg arról, hogy a GROUP BY záradékban csak a FROM és a WHERE záradékból származó oszlopokra hivatkozhatunk. Nem használhatunk olyan kifejezést, amit a SELECT záradékban hoztunk létre.

Vessünk egy újabb pillantást egy korábban már megoldott példára, hogy lássuk, mit is jelent ez, de ez alkalommal kövessük el a hibát!

„Írd ki valamennyi Washington állambeli megrendelő teljes nevét, teljes címét, a megrendelő legutóbbi szerződésének dátumát, valamint a szerződéseinek a teljes összegét!"

SQL

```
SELECT Customers.CustLastName || ', ' ||
   Customers.CustFirstName AS CustomerFullName,
   Customers.CustStreetAddress || ', ' ||
   Customers.CustCity || ', ' ||
   Customers.CustState || ' ' ||
   Customers.CustZip AS CustomerFullAddress
   MAX(Engagements.StartDate) AS LatestDate,
   SUM(Engagements.ContractPrice)
   AS TotalContractPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID =
   Engagements.CustomerID
WHERE Customers.CustState ='WA'
GROUP BY CustomerFullName,
   CustomerFullAddress
```

Van olyan adatbázis-kezelő rendszer, amelyik eltűri ezt az utasítást, de attól ez az utasítás még helytelen. A CustomerFullName (Megrendelő teljes neve) és a CustomerFullAddress (Megrendelő teljes címe) oszlopok nem léteznek egészen addig, amíg az adatbázis-kezelő

ki nem értékeli a FROM, a WHERE és a GROUP BY záradékokat. A GROUP BY nem fogja megtalálni ezeket az oszlopokat a FROM és a WHERE által visszaadott sorok között, ezért az SQL-szabványt szigorúan értelmező adatbázis-kezelő rendszer nyelvtani hibát fog jelezni.

A korábbiakban már bemutattuk, mi a helyes megoldás: fel kell sorolnunk a CustomerFullName és a CustomerFullAddress kifejezésekben felhasznált valamennyi oszlop nevét. Egy másik helyes megközelítés lehet, hogy a FROM záradékban a megfelelő allekérdezés beépítésével létrehozzuk a kiszámított oszlopokat. Következzen az ezt megvalósító kifejezés:

SQL
```
SELECT CE.CustomerFullName,
   CE.CustomerFullAddress,
   MAX(CE.StartDate) AS LatestDate,
   SUM(CE.ContractPrice)
   AS TotalContractPrice
FROM
  (SELECT Customers.CustLastName || ', ' ||
   Customers.CustFirstName AS CustomerFullName,
      Customers.CustStreetAddress || ', ' ||
      Customers.CustCity || ', ' ||
      Customers.CustState || ' ' ||
      Customers.CustZip AS CustomerFullAddress,
      Engagements.StartDate,
      Engagements.ContractPrice
   FROM Customers
   INNER JOIN Engagements
   ON Customers.CustomerID =
      Engagements.CustomerID
   WHERE Customers.CustState ='WA')
AS CE
GROUP BY CE.CustomerFullName,
   CE.CustomerFullAddress
```

Az utasítás működik, mivel a CustomerFullName és a CustomerFullAddress oszlopokat a FROM záradékban, kimenetként létrehoztuk. El kell ismernünk, hogy ettől a kifejezés meglehetősen összetetté vált. A gyakorlatban jobb, ha csak felsoroljuk az összes nem összetett kifejezésben használni kívánt oszlopot, ahelyett, hogy megpróbálnánk a kifejezéseket a FROM záradékon belül, oszlopként létrehozni.

Mikor használjuk a GROUP BY záradékot?

Ha idáig jutottunk az olvasásban, valószínűleg már elég jól értjük, hogyan kaphatunk részösszegeket adatcsoportokra, összesítő függvények és a GROUP BY záradék alkalmazásával. Úgy találtuk, hogy a GROUP BY sokoldalú felhasználásának bemutatására az a legjobb módszer, ha felsorolunk pár olyan problémát, amelyet megoldhatunk ennek az új záradéknak a segítségével, majd a *Példák* részben további példákat mutatunk be:

„Sorold fel a beszállítókat és beszállítónként a termékek napban mért szállítási idejét!"
„Írd ki minden termék nevét és összesített eladásait!"
„Sorold fel minden vásárló és rendelési időpont esetében a vásárló teljes nevét és valamennyi dátumhoz az adott időpontban rendelt tételek teljes árát!"
„Írd ki minden együttes azonosítóját, az együttes tagjait, és a tagoknak fizetett összeget úgy, hogy a teljes szerződési összeget elosztod az együttes tagjainak a számával!"
„Sorold fel minden ügynök nevét, a lekötött rendezvények összesített szerződési összegét és az ügynök megbízásainak a számát!"
„A teljesített órákat tekintve, írd ki tárgycsoportonként és tanulónként a tárgycsoport nevét, a tanuló nevét és a tanuló átlagos osztályzatát a tárgycsoportban felvett órákra vonatkozóan!"
„Sorold fel tárgycsoportonként a tárgycsoport nevét és a kínált órák számát!"
„Sorold fel a tanárokat és az egyes tanárok által tartott órák számát!"
„Sorold fel minden tornára és mérkőzésre a torna azonosítóját, a torna helyét, a mérkőzés számát, a csapatok nevét és az egyes csapatok összesített büntetőpontszámát!"
„Írd ki minden tekejátékos nevét és átlagos pontszámát!"
„Hány recept létezik hozzávalóosztályonként?"
„Melyik hozzávalóból mennyire lesz szükségünk, ha a szakácskönyv valamennyi receptjét el akarjuk készíteni?"

Példák

Már tudjuk, hogyan alkothatunk lekérdezéseket a GROUP BY záradék segítségével, és láttuk, miféle kérdéseket tudunk megválaszolni. Nézzünk most néhány olyan példát, amelyek az adatok csoportosítását igénylik (minden példa a mintaadatbázisokból származik)! Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név ugyanaz, mint a CD-mellékleten található megfelelő SQL-lekérdezés neve. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH13"-mal kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne felejtsük el, hogy a példáinkban használt oszlop- és táblanevek a B függelékben található mintaadatbázisok szerkezetéből származnak. Az egyszerűség kedvéért a következő példákban összevontuk a Fordítás és a Tisztázás lépéseit.*

A példák feltételezik, hogy elolvastuk és megértettük az előző fejezetekben tárgyalt fogalmakat, különös tekintettel a JOIN-okra és az allekérdezésekre.

Sales Orders adatbázis

„Sorold fel minden vásárló és megrendelési dátum esetében a vásárló teljes nevét és az adott időpontban rendelt tételek teljes összegét!"

Fordítás/ Tisztázás

Select customer first name ~~and~~ || ' ' || customer last name
as CustFullName, order date, ~~and the~~
sum ~~of~~ (quoted price ~~times~~ * quantity ordered) as TotalCost
from ~~the~~ customers ~~table~~
inner joined with the orders table
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~and then inner~~ joined with the order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_details.order number ~~in the order details table,~~
group~~ed~~ by customer first name,
customer last name, ~~and~~ order date

SQL

```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   Orders.OrderDate,
   SUM(Order_Details.QuotedPrice *
   Order_Details.QuantityOrdered) AS TotalCost
FROM (Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber =
   Order_Details.OrderNumber
GROUP BY Customers.CustFirstName,
   Customers.CustLastName, Orders.OrderDate
```

CH13_Order_Totals_By_Customer_And_Date (847

CustFullName	OrderDate	TotalCost
Alaina Hallmark	2007-09-02	$4,699.98
Alaina Hallmark	2007-09-14	$4,433.95
Alaina Hallmark	2007-09-21	$353.25
Alaina Hallmark	2007-09-22	$3,951.90
Alaina Hallmark	2007-09-30	$10,388.68
Alaina Hallmark	2007-10-12	$3,088.00
Alaina Hallmark	2007-10-22	$6,775.06
Alaina Hallmark	2007-10-30	$15,781.10
<< további sorok >>		

Entertainment Agency adatbázis

„Sorold fel minden együttes azonosítóját, az együttes tagjait, és a tagoknak fizetett összeget úgy, hogy a teljes szerződési összeget elosztod az együttes tagjainak a számával!"

Megjegyzés *Ez elég trükkös példa, mivel egy tag több együttesbe is tartozhat. Minden együttes esetében összegeznünk kell a szerződési összegeket, majd el kell osztanunk az összeget a tagok számával (feltételezve, hogy minden tag egyenlő összeget kap). Az összeszámláláshoz az aktuális előadóazonosító alapján szűrt allekérdezésre van szükség, (az együttes és nem a tag azonosítója alapján!), azaz az „entertainer ID" alapján is csoportosítanunk kell! És még valami, ne felejtsük el kiküszöbölni az inaktív tagokat (Status = 3) sem!*

Fordítás/ Tisztázás

Select entertainer ID, member first name, member last name,
~~and the~~ sum ~~of~~ (contract price)~~s divided by~~ / ~~the~~
(select count(*) ~~of active members~~
from entertainer members as EM2 ~~in the current entertainer group~~
where status ~~is not equal to~~ <> ~~not active~~ 3
and ~~the~~ EM2 ~~table~~ entertainer ID ~~equals~~
= ~~the~~ entertainer members ~~table~~ entertainer ID)
from ~~the~~ members ~~table~~
inner join~~ed with the~~ entertainer members ~~table~~
on members.member ID ~~in the members table matches~~
= entertainer_members.member ID
~~in the entertainer members table,~~
~~then~~ inner join~~ed with the~~ entertainers ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= entertainer_members.entertainer ID
~~in the entertainer members table,~~
~~and finally~~ inner join~~ed with the~~ engagements ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= engagements.entertainer ID ~~in the engagements table,~~
where member status ~~is not equal to~~ <> ~~not active~~ 3~~;~~
group~~ed~~ by entertainer ID,
member first name, ~~and~~ member last name,
~~sorted~~ order by member last name

SQL

```
SELECT Entertainers.EntertainerID,
   Members.MbrFirstName, Members.MbrLastName,
   SUM(Engagements.ContractPrice)/
       (SELECT COUNT(*)
        FROM Entertainer_Members AS EM2
        WHERE EM2.Status <> 3
        AND EM2.EntertainerID =
```

```
        Entertainers.EntertainerID)
      AS MemberPay
FROM ((Members
INNER JOIN Entertainer_Members
ON Members.MemberID =
   Entertainer_Members.MemberID)
INNER JOIN Entertainers
ON Entertainers.EntertainerID =
   Entertainer_Members.EntertainerID)
INNER JOIN Engagements
ON Entertainers.EntertainerID =
   Engagements.EntertainerID
WHERE Entertainer_Members.Status<>3
GROUP BY Entertainers.EntertainerID,
   Members.MbrFirstName, Members.MbrLastName
ORDER BY Members.MbrLastName
```

CH13_Member_Pay (39 sor)

EntertainerID	MbrFirstName	MbrLastName	MemberPay
1010	Kendra	Bonnicksen	$2,887.50
1013	Kendra	Bonnicksen	$3,767.50
1007	Robert	Brown	$2,975.00
1008	Robert	Brown	$6,816.00
1008	George	Chavez	$6,816.00
1013	George	Chavez	$3,767.50
1010	Caroline	Coie	$2,887.50
1013	Caroline	Coie	$3,767.50
<< *további sorok* >>			

School Scheduling adatbázis

„A teljesített órákat nézve, sorold fel tárgycsoportonként és tanulónként a tárgycsoport nevét, valamint a tanuló nevét és átlagos osztályzatát a csoportból felvett tárgyak alapján!"

Fordítás/ Tisztázás

Select category description, student first name, student last name,
~~and the average~~ AVG(~~of~~ grade) as AvgOfGrade
from ~~the~~ categories ~~table~~
inner join~~ed with the~~ subjects ~~table~~
on categories.category ID ~~in the categories table matches~~
= subjects.category ID ~~in the subjects table,~~
~~then~~ inner join~~ed with the~~ classes ~~table~~

on subjects.subject ID ~~in the subjects table matches~~
= classes.subject ID ~~in the classes table,~~
~~then~~ inner ~~joined with the~~ student schedules ~~table~~
on classes.class ID ~~in the classes table matches~~
= student_schedules.class ID ~~in the student schedules table,~~
~~then~~ inner ~~joined with the~~ student class status ~~table~~
on student_class_status.class status
~~in the student class status table matches~~
= student_schedules.class status ~~in the student schedules table,~~
~~and finally~~ inner ~~joined with the~~ students ~~table~~
on students.student ID ~~in the students table matches~~
= student_schedules.student ID ~~in the student schedules table~~
where class status description ~~is~~ = 'Completed,'
group~~ed~~ by category description, student first name,
~~and~~ student last name

SQL

```
SELECT Categories.CategoryDescription,
   Students.StudFirstName,
   Students.StudLastName,
   AVG(Student_Schedules.Grade) AS AvgOfGrade
FROM ((((Categories
INNER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
INNER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID)
INNER JOIN Student_Schedules
ON Classes.ClassID = Student_Schedules.ClassID)
```

CH13_Student_GradeAverage_By_Category (47 sor)

CategoryDescription	StudFirstName	StudLastName	AvgOfGrade
Accounting	Betsy	Stadick	90.67
Accounting	David	Hamilton	79.43
Accounting	Elizabeth	Hallmark	90.24
Accounting	John	Kennedy	71.45
Accounting	Michael	Viescas	90.01
Accounting	Sara	Sheskey	89.92
Accounting	Scott	Bishop	87.82
Accounting	Steve	Pundt	84.37
Art	Elizabeth	Hallmark	86.43
Art	George	Chavez	77.45
<< további sorok >>			

```
INNER JOIN Student_Class_Status
ON Student_Class_Status.ClassStatus =
   Student_Schedules.ClassStatus)
INNER JOIN Students
ON Students.StudentID =
   Student_Schedules.StudentID
WHERE Student_Class_Status.ClassStatusDescription =
   'Completed'
GROUP BY Categories.CategoryDescription,
   Students.StudFirstName,
   Students.StudLastName
```

Bowling League adatbázis

„Írd ki minden torna és mérkőzés esetében a tornaazonosítót, a torna helyét, a mérkőzés számát, a csapatok nevét és minden csapatnál a csapat összesített büntetőpontszámát!"

Fordítás/ Tisztázás

Select tourney ID, tourney location, match ID, team name, ~~and~~ ~~the~~ sum ~~of~~ (handicap score) as TotHandiCapScore
from ~~the~~ tournaments ~~table~~
inner join~~ed with the~~ tourney matches ~~table~~
on tournaments.tourney ID ~~in the tournaments table matches~~
= tourney_matches.tourney ID ~~in the tourney matches table,~~
~~then~~ inner join~~ed with the~~ match games ~~table~~
on tourney_matches.match ID
~~in the tourney matches table matches~~
= match_games.match ID ~~in the match games table,~~
~~then~~ inner join~~ed with the~~ bowler scores ~~table~~
on match_games.match ID ~~in the match games table matches~~
= bowler_scores.match ID ~~in the bowler scores table~~
and match_games.game number
~~in the match games table matches~~
= bowler_scores.game number ~~in the bowler scores table,~~
~~then~~ inner join~~ed with the~~ bowlers ~~table~~
on bowlers.bowler ID ~~in the bowlers table matches~~
= bowler_scores.bowler ID ~~in the bowler scores table,~~
~~and finally~~ inner join~~ed with the~~ teams ~~table~~
on teams.team ID ~~in the teams table matches~~
= bowlers.team ID ~~in the bowlers table,~~
group~~ed~~ by tourney ID, tourney location, match ID,
~~and~~ team name

SQL

```
SELECT Tournaments.TourneyID,
   Tournaments.TourneyLocation,
   Tourney_Matches.MatchID, Teams.TeamName,
```

```
    SUM(Bowler_Scores.HandiCapScore)
    AS TotHandiCapScore
FROM ((((Tournaments
INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
    Tourney_Matches.TourneyID)
INNER JOIN Match_Games
ON Tourney_Matches.MatchID =
    Match_Games.MatchID)
INNER JOIN Bowler_Scores
ON (Match_Games.MatchID =
    Bowler_Scores.MatchID) AND
    (Match_Games.GameNumber =
    Bowler_Scores.GameNumber))
INNER JOIN Bowlers
ON Bowlers.BowlerID = Bowler_Scores.BowlerID)
INNER JOIN Teams
ON Teams.TeamID = Bowlers.TeamID
GROUP BY Tournaments.TourneyID,
    Tournaments.TourneyLocation,
    Tourney_Matches.MatchID, Teams.TeamName
```

CH13_Tournament_Match_Team_Results (112 sor)

TourneyID	TourneyLocation	MatchID	TeamName	TotHandiCapScore
1	Red Rooster Lanes	1	Marlins	2351
1	Red Rooster Lanes	1	Sharks	2348
1	Red Rooster Lanes	2	Barracudas	2289
1	Red Rooster Lanes	2	Terrapins	2391
1	Red Rooster Lanes	3	Dolphins	2389
1	Red Rooster Lanes	3	Orcas	2395
1	Red Rooster Lanes	4	Manatees	2292
1	Red Rooster Lanes	4	Swordfish	2353
2	Thunderbird Lanes	5	Marlins	2297
2	Thunderbird Lanes	5	Terrapins	2279
<< további sorok >>				

„Írd ki minden tekejátékos legmagasabb pontszámát!"

Fordítás/ Tisztázás: Select bowler first name, bowler last name, ~~and~~ ~~the~~ max~~imum~~ (raw score) as HighScore from ~~the~~ bowlers ~~table~~ inner join~~ed with the~~ bowler scores ~~table~~

on bowlers.bowler ID ~~in the bowlers table matches~~
= bowler_scores.bowler ID ~~in the bowler scores table,~~
group~~ed~~ by bowler first name, ~~and~~ bowler last name

SQL

```
SELECT Bowlers.BowlerFirstName,
   Bowlers.BowlerLastName,
   MAX(Bowler_Scores.RawScore) AS HighScore
FROM Bowlers
INNER JOIN Bowler_Scores
ON Bowlers.BowlerID = Bowler_Scores.BowlerID
GROUP BY Bowlers.BowlerFirstName,
   Bowlers.BowlerLastName
```

CH13_Bowler_High_Score_Group (32 sor)

BowlerFirstName	BowlerLastName	HighScore
Alaina	Hallmark	180
Aslatair	Black	164
Angel	Kennedy	194
Ann	Patterson	165
Bailey	Hallmark	164
Barbara	Fournier	164
Caleb	Viescas	193
Carol	Viescas	150
David	Cunningham	180
David	Foumier	178
<< *további sorok* >>		

Recipes adatbázis

„Írd ki, hogy hozzávalóosztályonként hány recept létezik!"

Megjegyzés *Itt a kihívás abban áll, hogy receptenként egy bizonyos receptosztályt egynél többször nem akarunk megszámolni.Ha például egy recept többféle növényi hozzávalót vagy tejterméket tartalmaz, akkor osztályonként csak egyszer vehetjük figyelembe. Lehet, hogy eljött az idő a COUNT (DISTINCT értékkifejezés) bevetésére?*

Fordítás/ Tisztázás

Select ingredient class description,
~~and the unique~~ count ~~of~~ (distinct recipe ID) as CountOfRecipeID
from ~~the~~ ingredient classes ~~table~~

inner joined with the ingredients table
on ingredient_classes.ingredient class ID
in the ingredient classes table matches
= ingredients.ingredient class ID in the ingredients table,
and then inner joined with the recipe ingredients table
on ingredients.ingredient ID in the ingredients table matches
= recipe_ingredients.ingredient ID
in the recipe ingredients table,
grouped by ingredient class description

SQL

```
SELECT
   Ingredient_Classes.IngredientClassDescription,
   Count(DISTINCT RecipeID) AS CountOfRecipeID
FROM (Ingredient_Classes
INNER JOIN Ingredients
ON Ingredient_Classes.IngredientClassID =
   Ingredients.IngredientClassID)
INNER JOIN Recipe_Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
GROUP BY
   Ingredient_Classes.IngredientClassDescription
```

CH13_IngredientClass_Distinct_Recipe_Count (19 sor)

IngredientClassDescription	CountOfRecipeID
Bottle	1
Butter	3
Cheese	2
Chips	1
Condiment	2
Dairy	2
Fruit	1
Grain	2
Herb	1
<< további sorok >>	

Megjegyzés *Mivel a Microsoft Access nem támogatja a COUNT DISTINCT parancsot, azt vehetjük észre, hogy az Access mintaadatbázisban talált lekérdezés először a FROM záradékban egy tábla-allekérdezés segítségével kiválasztja a receptazonosító egyedi (DISTINCT) értékeit, majd összeszámolja az eredménysorokat.*

Összefoglalás

A fejezet elején elmagyaráztuk, hogy mi lehet az okunk arra, hogy csoportosítsuk az adatokat, hogy több részösszeget kapjunk egy eredményhalmazból. Miután szembesültünk egy nehéznek tűnő példafeladattal, továbbléptünk, és bemutattuk, hogyan oldjuk meg a problémát – és még sok másikat – a GROUP BY záradék segítségével. Arra is mutattunk példát, hogyan használhatunk együtt oszlopkifejezéseket és összesítő függvényeket.

Ezt követően egy érdekes példát elemeztünk, ahol a GROUP BY záradékot egy olyan allekérdezésben használtuk, amely egy WHERE záradékban szolgált az adatok szűrésére. Később rámutattunk, hogy a GROUP BY segítségével, de összesítő függvények nélkül alkotott lekérdezések ugyanazt eredményezik, mintha a SELECT záradékban a DISTINCT kulcszót használnánk. Ez után kitértünk arra, hogy milyen fontos, hogy a GROUP BY záradékot kellő odafigyeléssel építsük fel, és az oszlopokat építsük be, ne pedig a kifejezéseket.

A GROUP BY záradékról szóló fejtegetésünk végén néhány gyakran elkövetett hibára hívtuk fel a figyelmet. Elmondtuk, hogy az SQL nem tételez fel semmilyen, az elsődleges kulcsokkal kapcsolatos ismeretet. A SELECT záradékban használt oszlopkifejezésekkel kapcsolatos gyakori hibákra is kitértünk.

Összefoglaltuk, miért hasznos a GROUP BY záradék, és példafeladatokat adtunk közre, amelyeket megoldhatunk a segítségével. A fejezet további része arra mutatott példákat, hogyan alkothatunk a GROUP BY felhasználását igénylő kérelmeket.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Írd ki az összes beszállítót és beszállítónként a termékek szállításához szükséges napok számát!"*
   (Tipp: használjuk az AVG összesítő függvényt, és csoportosítsunk a beszállítók szerint.)
   A megoldás itt található: CH13_Vendor_Avg_Delivery (10 sor).

2. *„Írd ki minden termék nevét és összesített eladásait!"*
   (Tipp: használjuk a SUM függvényt, a mennyiség és az árak szorzatát, valamint csoportosítsunk a termékek neve alapján.)
   A megoldás itt található: CH13_Sales_By_Product (38 sor).
3. *„Sorolj fel minden beszállítót és az általuk kínált termékeket!"*
   A megoldás itt található: CH13_Vendor_Product_Count_Group (10 sor).
4. *Kihívás: oldjuk meg a 3. problémát allekérdezés segítségével!*
   A megoldás itt található: CH13_Vendor_Product_Count_Subquery (10 sor).

Entertainment Agency adatbázis

1. *„Írd ki minden ügynök nevét, a lekötött fellépések szerződési értékének összegét és az ügynök összes megbízását!"*
   (Tipp: össze kell szoroznunk az összesített szerződési összegeket és a megbízásokat, és ne felejtsünk el a megbízási ráta alapján csoportosítani sem.)
   A megoldás itt található: CH13_Agent_Sales_And_Commissions (8 sor).

School Scheduling adatbázis

1. *„Írd ki tárgycsoportonként a tárgycsoport nevét és a kínált órák számát!"*
   (Tipp: használjuk a COUNT függvényt, és csoportosítsunk a tárgycsoportok nevei alapján.)
   A megoldás itt található: CH13_Category_Class_Count (16 sor).
2. *„Sorold fel a tanárokat és az általuk tartott órák számát!"*
   (Tipp: használjuk a COUNT függvényt, és csoportosítsunk a tanárok neve alapján.)
   A megoldás itt található: CH13_Staff_Class_Count (23 sor).
3. *Kihívás: oldjuk meg a 2. problémát allekérdezés segítségével!*
   A megoldás itt található: CH13_Staff_Class_Count (27 sor).
4. *Meg tudjuk magyarázni, miért ad vissza az allekérdezést használó megoldás négy sorral többet?*

Bowling League adatbázis

1. *„Írd ki minden tekejátékos nevét és átlagos pontszámát!"*
   (Tipp: használjuk az AVG összesítő függvényt, és csoportosítsunk a játékosok neve alapján.)
   A megoldás itt található: CH13_Bowler_Averages (32 sor).
2. *„Számold ki minden tekejátékos jelenlegi átlagát és büntetőpontszámát!"*
   (Tipp: ez egy „barátságos" liga, ezért a büntetőpontszám kiszámítása úgy történik, hogy kétszázból kivonjuk az átlagot, és ennek a 90 százalékát vesszük. Mielőtt kivonnánk kétszázból, kerekítsük és alakítsuk egésszé az átlagot, majd a végeredményt kerekítsük és vágjuk le. Bár az SQL-szabvány nem határozza meg a ROUND függvényt, a legtöbb kereskedelmi adatbázis-kezelőben megtalálható. A részletekért nézzük meg a termék dokumentációját.)
   A megoldás itt található: CH13_Bowler_Average_Handicap (32 sor).

3. *Kihívás: az „Írd ki minden tekejátékos legmagasabb pontszámát!"feladatot oldjuk meg allekérdezéssel!*
   A megoldás itt található: CH13_Bowler_High_Score_Subquery (32 sor).

Recipes adatbázis

1. *„Melyik hozzávalóból mennyire lenne szükségünk, ha a szakácskönyv összes receptjét el szeretnénk készíteni?"*
   (Tipp: használjuk a SUM függvényt, majd csoportosítsunk a hozzávalók neve és a mértékegység-leírás alapján.)
   A megoldás itt található: CH13_Total_Ingredients_Needed (65 sor).
2. *„Sorold fel az összes húsféle hozzávalót, és azokat a recepteket, amelyek mindegyiket tartalmazzák!"*
   A megoldás itt található: CH13_Meat_Ingredient_Recipe_Count_Group (4 sor).
3. *Kihívás: oldjuk meg a 2. problémát allekérdezés segítségével!*
   A megoldás itt található: CH13_Meat_Ingredient_Recipe_Count_Subquery(11 sor).
4. Meg tudjuk magyarázni, miért ad vissza az allekérdezést használó megoldás héttel több sort?

14

Csoportosított adatok szűrése

„Hadd tömjék fejüket iskolamesterek,
nyelvtannal, badarsággal, leckével
Jó pálinka, szilárdan tartom,
tisztább látást ad a lángésznek."
– Oliver Goldsmith

A fejezet témakörei

- A „célcsoport" új jelentése
- Szűrőkkel minden más
- A HAVING használata
- Példák
- Összefoglalás
- Önálló feladatok

A 12. fejezetben részletesen ismertettük az SQL-szabványban meghatározott összesítő függvényeket. Ezt vittük tovább a 13. fejezetben, amikor leírtuk, hogyan kérhetjük az adatbázis-kezelőtől sorok csoporthalmazait, és hogyan számíthatjuk ki a csoportok összesített értékeit. A csoportosítás egyik nagy előnye az volt, hogy az értékkifejezéseket csoportosított oszlopok alapján is ki tudtuk írni, így azonosítva az egyes csoportokat.

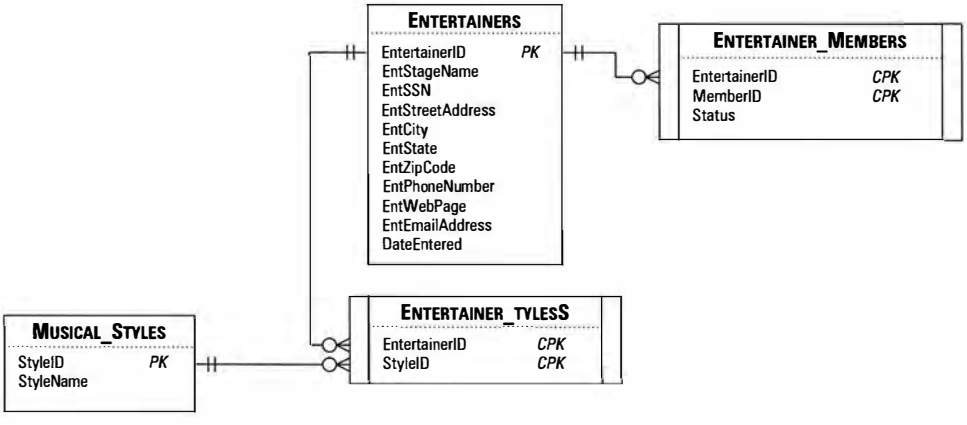
Ebben a fejezetben helyére illesztjük az összesítő és csoportosító kirakósjátek utolsó elemét is. Miután csoportosítottuk a sorokat, és kiszámítottuk az összesített értéküket, gyakran jól jön, ha az összesített számítás eredményét egy állítás (predikátum) segítségével még tovább szűrjük. Ahogy hamarosan meglátjuk, ehhez a kirakósjáték utolsó elemére – a HAVING záradékra – lesz szükség.

A „célcsoport" új jelentése

Tudjuk már, hogy miután sorcsoportokba gyűjtöttük az adatokat, a MIN, a MAX, az AVG, a SUM és a COUNT segítségével minden csoport összesített adatait lekérhetjük. Tegyük fel, hogy tovább szeretnénk finomítani az eredményhalmazt – a csoportok elkülönítése érdekében – valamelyik összesített érték tesztelésével. Nézzünk egy egyszerű lekérdezést:

„Sorold fel azokat az együtteseket, amelyek jazzt játszanak, és háromnál több tagjuk van!"

Ez nem hangzik túlságosan bonyolultan, ugye? A kérelem megoldásához szükséges táblákat a 14.1. ábra mutatja.



14.1. ábra

Táblák a jazz-zenét játszó, háromnál több tagú együttesek megkereséséhez

Megjegyzés *Mint általában, itt is a 4. fejezetben bevezetett „Kérelem-Fordítás-Tisztázás-SQL" módszert követjük, valamint a 8.fejezetben megismert JOIN-okat és allekérdezéseket, illetve a 9. fejezetben tanult OUTER JOIN-t és a 11. fejezetben megismert allekérdezéseket használjuk.*

A HAVING záradék ismerete nélkül valószínűleg késztetést éreznénk, hogy a következő, helytelen módszerrel próbálkozzunk:

Fordítás Select the entertainer stage name and the count of members from the entertainers table joined with the entertainer members table on entertainer ID in the entertainers table matches entertainer ID in the entertainer members table where the entertainer ID is in the selection of entertainer IDs from the

entertainer styles table joined with the musical styles table on style ID in the entertainer styles table matches style ID in the musical styles table where the stylename is 'Jazz' and where the count of the members is greater than 3, grouped by entertainer stage name
(Válaszd ki az előadó művésznevét és a tagok számát az előadók és az együttestagok tábláiból, amelyeket az előadóazonosító alapján összekapcsoltunk, ha az előadóazonosító benne van az előadóazonosítóknak abban a halmazában, amelyet az előadók stílusát és a zenei stílusokat tartalmazó tábláknak a stílusazonosító alapján történő összekapcsolásából nyertünk, és ha az előadók stílusát tartalmazó táblában a stílusazonosító megegyezik a zenei stílusokat tároló tábla stílusazonosítójával, és a stílus neve „Jazz", és ha a tagok száma nagyobb, mint 3, és csoportosítsd az eredményt az előadók művészneve szerint.)

Tisztázás

Select ~~the~~ entertainer stage name
~~and the~~ count(*) ~~of members~~ as CountOfMembers
from ~~the~~ entertainers ~~table~~
inner joined ~~with the~~ entertainer members ~~table~~
on entertainers.entertainer ID ~~in the entertainers table~~
~~matches~~ = entertainer_members.entertainer ID
~~in the entertainer members table~~
where ~~the~~ entertainer ID ~~is~~ in ~~the~~
(select~~ion of~~ entertainer ID~~s~~
from ~~the~~ entertainer styles ~~table~~
inner joined ~~with the~~ musical styles ~~table~~
on entertainer_styles.style ID ~~in the entertainer styles table~~
~~matches~~ = musical_styles.style ID ~~in the musical styles table~~
where ~~the~~ style name ~~is~~ = 'Jazz')
and ~~where the~~ count(*) ~~of the members is greater than~~ > 3~~;~~
group~~ed~~ by entertainer stage name

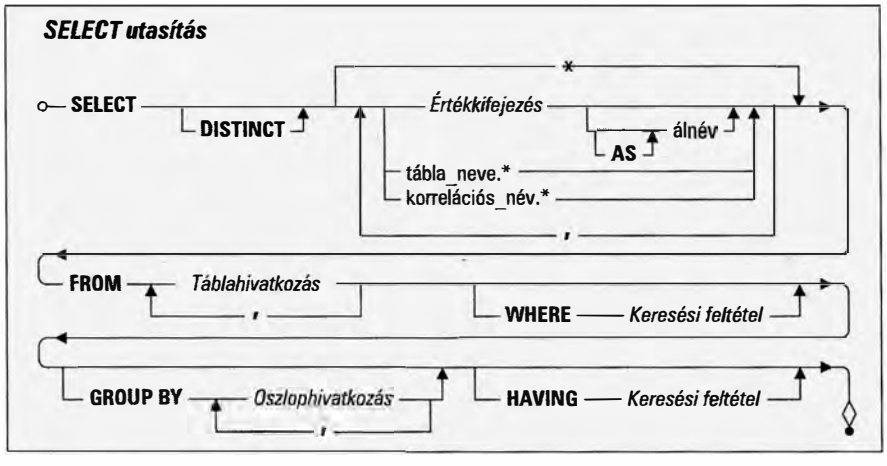
SQL

```
SELECT Entertainers.EntStageName,
   COUNT(*) AS CountOfMembers
FROM Entertainers
INNER JOIN Entertainer_Members
ON Entertainers.EntertainerID =
   Entertainer_Members.EntertainerID
WHERE Entertainers.EntertainerID
IN
  (SELECT Entertainer_Styles.EntertainerID
   FROM Entertainer_Styles
   INNER JOIN Musical_Styles
   ON Entertainer_Styles.StyleID =
      Musical_Styles.StyleID
```

```
    WHERE Musical_Styles.StyleName = 'Jazz')
AND COUNT(*) > 3
GROUP BY Entertainers.EntStageName
```

Mi a baj ezzel az ábrával? Főleg az, hogy minden, a WHERE záradékban (Emlékszünk a 6. fejezetre?) hivatkozott oszlop a FROM záradékban meghatározott táblából ***kell***, hogy származzon. A COUNT(*) a FROM záradékbból származik? Hát persze, hogy nem – sőt a COUNT értékét csak a sorok csoportosítása után tudjuk csak kiszámítani!

Úgy tűnik, a GROUP BY után egy új záradékra lesz szükségünk. A 14.2. ábra a SELECT utasítás teljes utasításformáját bemutatja, az új HAVING záradékkal együtt.



14.1. ábra

A SELECT utasítás és valamennyi lehetséges záradéka

Mivel a HAVING záradék a sorok *csoportosítása után* hajtódik végre, az SQL-szabvány tartalmaz néhány megkötést arra nézve, hogy mely oszlopokra hivatkozhatunk a keresési feltételek állításaiban. Érdemes megjegyeznünk, hogy ha nincs GROUP BY, az adatbáziskezelő a HAVING záradékot a FROM és a WHERE záradékok által visszaadott összes sorra értelmezi, úgy, mintha a visszadott sorok egyetlen csoportot alkotnának.

A megkötések ugyanazok, mint a csoportosított lekérdezések SELECT záradékában hivatkozott oszlopok esetében. Bármilyen, oszlopra való hivatkozás a HAVING záradék keresési feltételében szereplő állításban eleget kell tegyen a következőknek: vagy meg kell nevezznünk egy a GROUP BY záradékban felsorolt oszlopot, vagy összesítő függvényen belül kell elhelyeznünk. Ezeknek a megszorításoknak azért lehet helyük, mert a lehetséges oszlopösszehasonlításoknak a csoportosított sorokat kell felhasználniuk – akár csoportosító értékekről, akár csoportok sorain végrehajtott összesítő számításokról legyen szó.

Most, hogy már tudunk egy keveset a HAVING záradékról, oldjuk meg a korábbi problémát a helyes módszerrel:

„Sorold fel azokat az együtteseket, amelyek jazzt játszanak, és háromnál több tagjuk van!"

Fordítás Select the entertainer stage name and the count of members from the entertainers table joined with the entertainer members table on entertainer ID in the entertainers table matches entertainer ID in the entertainer members table where the entertainer ID is in the selection of entertainer IDs from the entertainer styles table joined with the musical styles table on style ID in the entertainer styles table matches style ID in the musical styles table where the style name is 'Jazz,' grouped by entertainer stage name, and having the count of the members greater than 3

(Válaszd ki azoknak az együtteseknek a művésznevét és a tagok számát az előadók és a tagok tábláiból – amelyeket az előadóazonosító alapján összekapcsoltunk –, amelyeknél az előadók táblájában található előadóazonosító megegyezik a tagok táblájában található előadóazonosítóval, és ahol az előadóazonosító szerepel az előadók stílusait és a zenei stílusokat tároló táblák előadóazonosítói között, mely táblákat összekapcsoltuk a stílusazonosító alapján, ha a stílusnév „Jazz", és mindezt csoportosítsd az előadók művészneve szerint, továbbá követeld meg, hogy a tagok száma 3 vagy több legyen.)

Tisztázás Select ~~the~~ entertainer stage name
~~and the~~ count(*) ~~of members~~ as CountOfMembers
from ~~the~~ entertainers ~~table~~
inner join~~ed with the~~ entertainer members ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= entertainer_members.entertainer ID
~~in the entertainer members table~~
where ~~the~~ entertainer ID ~~is~~ in ~~the~~
(select~~ion of~~ entertainer ID~~s~~
from ~~the~~ entertainer styles ~~table~~
inner join~~ed with the~~ musical styles ~~table~~
on entertainer_styles.style ID
~~in the entertainer styles table matches~~
= musical_styles.style ID ~~in the musical styles table~~
where ~~the~~ style name ~~is~~ = 'Jazz')~~;~~
group~~ed~~ by entertainer stage name~~;~~
~~and~~ having ~~the~~ count(*) ~~of the members greater than~~ > 3

SQL
```
SELECT Entertainers.EntStageName,
   COUNT(*) AS CountOfMembers
```

```
FROM Entertainers
INNER JOIN Entertainer_Members
ON Entertainers.EntertainerID =
   Entertainer_Members.EntertainerID
WHERE Entertainers.EntertainerID
IN
   (SELECT Entertainer_Styles.EntertainerID
   FROM Entertainer_Styles
   INNER JOIN Musical_Styles
   ON Entertainer_Styles.StyleID =
      Musical_Styles.StyleID
   WHERE Musical_Styles.StyleName = 'Jazz')
GROUP BY Entertainers.EntStageName
HAVING COUNT(*) > 3
```

Bár a kérelem végső kimenetében szerepel a COUNT, a COUNT(*) függvényt enélkül is használhatnánk a HAVING záradékban. Amíg a HAVING záradékban szereplő számított értékek és oszlophivatkozások levezethetők a csoportosított sorokból, nyugodtak lehetünk. Ezt a lekérdezést az Entertainment Agency mintaadatbázisban CH14_Jazz_Entertainers-_More_Than_3 néven találhatjuk meg.

Szűrőkkel minden más

Most már két módszerünk is van a végső eredményhalmaz szűrésére: a WHERE és a HAVING záradék használata. Tudjuk, hogy bizonyos korlátozások vonatkoznak a HAVING záradék keresési feltételeiben alkalmazható állításokra. Egyes esetekben választhatunk, hogy a két záradék közül melyikbe helyezzük a predikátumot. Vizsgáljuk meg, mikor értelmesebb a HAVING helyett a WHERE záradékban elhelyezni a szűrőt!

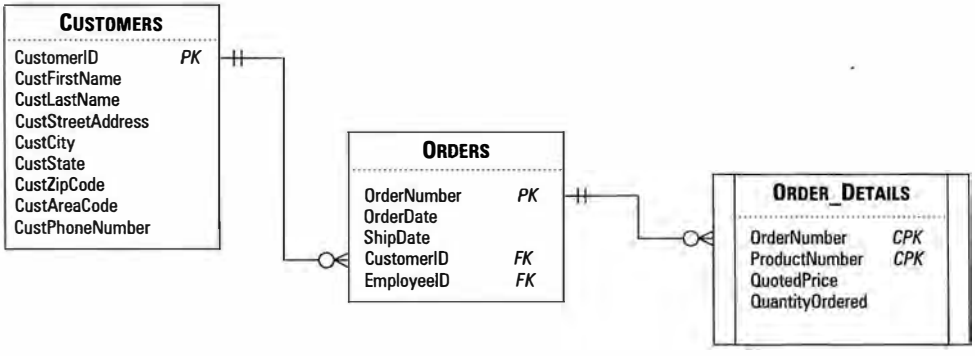
A WHERE vagy a HAVING záradékban szűrjünk?

A 6. fejezetben megismertük a kérelmek FROM záradéka által visszaadott sorok szűrésére felépíthető öt fő állítástípust. Ezek az alábbiak voltak: összehasonlítás (=, <>, <, >, >=, <=), tartomány (BETWEEN), tagság (IN), mintaillesztés (LIKE) és az IS NULL. A 11. fejezetben kiterjesztettük a látókörünket: bemutattuk, hogyan lehet allekérdezéseket összehasonlítási és tagsági állítások paramétereiként használni, és bevezettünk két további predikátum-osztályt – mennyiségi (ANY, SOME, ALL) és létezést kifejező (EXISTS) –, amelyek paraméterként allekérdezéseket igényelnek.

Ne felejtsük el, hogy a WHERE záradékban használt keresési feltétel a sorokat *azelőtt* szűri, mielőtt az adatbázis-kezelő végrehajtaná a csoportosítást. Általában véve azt mondhatjuk, hogy ha a végeredményben csak a sorok valamilyen részhalmazát szeretnénk viszontlátni, jobb, ha a nemkívánatos sorokat minél előbb kiküszöböljük a WHERE záradék segítségével. Tegyük fel, hogy a következő kérelmet szeretnénk megfogalmazni:

„Sorold fel azokat az Egyesült Államok nyugati partján található államokat, ahol az összesített rendelési összeg meghaladja az egymillió dollárt!"

A megoldáshoz szükséges táblákat a 14.3. ábrán láthatjuk.



14.3. ábra

Az államonkénti rendelések összegzéséhez szükséges táblák

Helyénvaló, ha a kérelmet az alábbiak szerint fogalmazzuk meg, a vevő államát meghatározó állítást a HAVING záradékba helyezve:

SQL
```
SELECT Customers.CustState,
   SUM(Order_Details.QuantityOrdered *
   Order_Details.QuotedPrice) AS SumOfOrders
FROM (Customers
   INNER JOIN Orders
   ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber =
   Order_Details.OrderNumber
GROUP BY Customers.CustState
HAVING SUM(Order_Details.QuantityOrdered *
   Order_Details.QuotedPrice) > 1000000
AND CustState IN ('WA', 'OR', 'CA')
```

Mivel az állam oszlopa szerint csoportosítunk, *lehetőségünk van* arra, hogy az oszlopra vonatkozó állítást a HAVING záradékba helyezzük, de ekkor az adatbázis-kezelő rendszert a szükségesnél több munkára kérnénk. Kiderül, hogy a teljes rendelési összeg a texasi vásárlók esetében is meghaladja az egymillió dollárt. Ha a szűrőt a HAVING záradékban helyezzük el, mint ahogy itt látható, az adatbázis-kezelő összegzi a Texasra vonatkozó sorokat is, elraktározza az eredményt, és csak akkor dobja el, amikor kiderül, hogy a Texas csoportra nincs szükség.

Ha az eredményt a vevők államai alapján szeretnénk csoportosítani, de csak a washingtoni, oregoni és kaliforniai vásárlókra vagyunk kíváncsiak, ésszerűbbnek tűnik, hogy a három államot tartalmazó sorokat a WHERE záradékban szűrjük, még mielőtt államonkénti csoportosítást kérnénk a GROUP BY záradékkal. Ha nem így tennénk, a FROM záradék az összes állam vásárlóit visszaadná, és felesleges munkával csoportosítanánk olyan sorokat, amelyekre semmi szükség nincs. Íme egy jobb megoldás a problémára:

Fordítás Select customer state and the sum of quantity ordered times quoted price as SumOfOrders from the customers table joined with the orders table on customer ID in the customers table matches customer ID in the orders table, and then joined with the order details table on order number in the orders table matches order number in the order details table where customer state is in the list 'WA', 'OR', 'CA', grouped by customer state, and having the sum of the orders greater than $1 million
(Válaszd ki a vásárló államát, valamint a rendelt mennyiségek összegének és a jegyzett árnak a szorzatát SumOfOrders néven a vásárlók és a rendelések tábláiból, amelyeket a vásárlóazonosító alapján összekapcsoltunk, ha a vásárlók táblájában található vásárlóazonosító megegyezik a rendelések táblájában található vásárlóazonosítóval, továbbá ezt kapcsold össze a rendelések részleteit tartalmazó táblával a rendelési szám alapján, és válaszd ki azokat az adatokat, ahol a rendelések táblájában található rendelési szám megegyezik a rendelések részleteit tartalmazó táblában található rendelési számmal, ha a vevő állama „WA", „OR"vagy „CA", a vevő állama szerint csoportosítva, és megkövetelve, hogy a rendelési összeg meghaladja az egymillió dollárt.)

Tisztázás Select customer state, ~~and the~~
sum ~~of~~ (quantity ordered ~~times~~ * quoted price) as SumOfOrders
from the customers table
inner join~~ed with the~~ orders ~~table~~
on customers.customer ID ~~in the customers table matches~~
= orders.customer ID ~~in the orders table,~~
~~and then~~ join~~ed with the~~ order details ~~table~~
on orders.order number ~~in the orders table matches~~
= order_details.order number ~~in the order details table~~
where customer state ~~is~~ in ~~the list~~ ('WA', 'OR', 'CA')~~,~~
group~~ed~~ by customer state~~, and~~
having ~~the~~ sum ~~of the orders~~ (quantity ordered * quoted price)
~~greater than~~ > ~~$1 million~~ 1000000

SQL
```
SELECT Customers.CustState,
   SUM(Order_Details.QuantityOrdered *
   Order_Details.QuotedPrice) AS SumOfOrders
FROM (Customers
   INNER JOIN Orders
   ON Customers.CustomerID = Orders.CustomerID)
```

```
INNER JOIN Order_Details
ON Orders.OrderNumber =
   Order_Details.OrderNumber
WHERE Customers.CustState IN ('WA', 'OR', 'CA')
GROUP BY Customers.CustState
HAVING SUM(Order_Details.QuantityOrdered *
   Order_Details.QuotedPrice) > 1000000
```

Ezt a lekérdezést a mintaadatbázisban CH14_West_Coast_Big_Order_States néven találhatjuk meg.

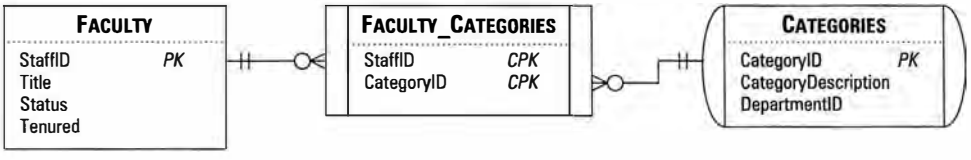
Ne essünk a HAVING COUNT csapdájába!

Sokszor vagyunk kíváncsiak arra, hogy mely tételcsoportoknak van meghatározott számúnál kevesebb tagja. Ilyen eset például, ha azt szeretnénk tudni, hogy melyik együttes áll háromnál kevesebb tagból, melyik recepthez kell hozzávalóként legfeljebb két tejtermék, vagy hogy melyik tárgyat tanítja három vagy kevesebb főállású egyetemi tanár. A trükk ebben az esetben az, hogy azt *is* tudni akarjuk, hogy melyik csoportnak van *nulla* eleme.

Nézzünk egy példát, ami rávilágít, hogy milyen csapdába kerülhetünk:

> *„Sorold fel azokat a tárgycsoportokat, amelyeket háromnál kevesebb főállású egyetemi tanár tanít!"*

A megoldáshoz szükséges táblákat a 14.4. ábra mutatja.



14.4. ábra

A háromnál kevesebb egyetemi tanár által tanított tárgycsoportok megkereséséhez szükséges táblák

Fordítás Select category description and the count of staff ID as ProfCount from the categories table joined with the faculty categories table on category ID in the categories table matches category ID in the faculty categories table, and then joined with the faculty table on staff ID in the faculty table matches staff ID in the faculty categories table where title is 'Professor,' grouped by category description, and having the count of staff ID less than 3
(Válaszd ki a tárgycsoportleírást és a tanárazonosítók számát ProfCount néven abból a táblából, amelyet a tárgycsoportok és a tanszékcsoportok tábláit a csoportazonosító alapján összekapcsolva, majd ezt a tanszékek és a tanszékcsoportok tábláit a tanárazonosító alapján összekapcsoló táblával

összekapcsolva kaptunk, ha a cím „Professor", és az eredményt csoportosítsd tárgycsoportleírás szerint, és követeld meg, hogy a tanárazonosítók száma háromnál kevesebb legyen.)

Tisztázás Select category description
and the count of (staff ID) as ProfCount
from the categories table
inner joined with the faculty categories table
on categories.category ID in the categories table matches
= faculty_categories.category ID in the faculty categories table,
and then inner joined with the faculty table
on faculty.staff ID in the faculty table matches
= faculty_categories.staff ID in the faculty categories table
where title is = 'Professor,'
grouped by category description, and
having the count of (staff ID) less than < 3

SQL
```
SELECT Categories.CategoryDescription,
   COUNT(Faculty_Categories.StaffID) AS
   ProfCount
FROM (Categories
INNER JOIN Faculty_Categories
ON Categories.CategoryID =
   Faculty_Categories.CategoryID)
INNER JOIN Faculty
ON Faculty.StaffID = Faculty_Categories.StaffID
WHERE Faculty.Title = 'Professor'
GROUP BY Categories.CategoryDescription
HAVING COUNT(Faculty_Categories.StaffID) < 3
```

Egész jól fest, nem? Alább láthatjuk a kérelem által visszaadott eredményhalmazt:

CH14_Subjects_Fewer_3_Professors_WRONG

CategoryDescription	ProfCount
Accounting	1
Business	2
Computer Information Systems	1
Economics	1
Geography	1
History	1
Journalism	1
Math	1
Political Science	1

Észrevettük, hogy az eredményhalmazban *nincs* nulla egyetemi tanárt tartalmazó tárgycsoport? Ennek az az oka, hogy a COUNT függvény csak azokat a sorokat számolja meg, amelyek a főállású egyetemi tanárok kiszűrése után maradtak a Faculty_Categories (Tanszékcsoportok) táblában. A várhatóan nullát tartalmazó sorokat a WHERE záradékban eldobtuk.

Csak azért, hogy igazoljuk a gyanúnkat, miszerint léteznek főállású egyetemi tanár nélküli csoportok, állítsunk össze egy olyan lekérdezést, amelynek a segítségével ellenőrizhetjük ezt az elméletünket. Emlékezzünk vissza, hogy COUNT összesítő függvény nullával tér vissza, ha üres halmaz megszámlálására kérjük, továbbá arra, hogy eredményül üres halmazt kaphatunk, ha a kérelmet úgy fogalmazzuk meg, hogy vegye figyelembe, hogy hány sor van egy meghatározott tárgycsoportban. Éppen így fogunk tenni, és arra kényszerítjük az adatbázis-kezelőt, hogy a tárgycsoportokat egyenként nézze végig. A csoportsorokat fogjuk megszámolni, nem a tanszéki tárgyak sorait. Nézzük meg az alábbi SQL-utasítást:

```
SQL     SELECT COUNT(Faculty.StaffID)
        AS BiologyProfessors
        FROM (Faculty
        INNER JOIN Faculty_Categories
        ON Faculty.StaffID =
        Faculty_Categories.StaffID)
        INNER JOIN Categories
        ON Categories.CategoryID =
        Faculty_Categories.CategoryID
        WHERE Categories.CategoryDescription =
        'Biology'
        AND Faculty.Title = 'Professor'
```

BiologyProfessors
0

A lekérdezést a mintaadatbázisban CH14_Count_Of_Biology_Professors néven találhatjuk meg. Mint látjuk, valóban nincsenek olyan főállású egyetemi tanárok a School Scheduling mintaadatbázisban, akik biológiát tanítanak. A lekérdezést úgy fogalmaztuk meg, hogy csak egy tárgycsoportot vegyen figyelembe. Mivel nincsenek olyan sorok, amelyek egyszerre tartalmaznak Professor és Biology értékeket, érvényes üres halmazt kaptunk eredményül. Ennél fogva a COUNT függvény is nullát ad vissza.

Most, hogy mindezt már tudjuk, allekérdezésként beágyazhatjuk ezt a kérelmet egy WHERE záradékba, ami kivonatolja a tárgycsoport-azonosítón alapuló egyezéseket a külső lekérdezésből. Így arra kényszerítjük az adatbázis-kezelőt, hogy a csoportokat egyesével vizsgálja meg, mivel a külső kérelemben a tárgycsoportleírásokat soronként kéri le a Categories táblából. Az SQL kód a következő lesz:

```
SQL     SELECT Categories.CategoryDescription,
          (SELECT COUNT(Faculty.StaffID)
           FROM (Faculty
```

```
    INNER JOIN Faculty_Categories
    ON Faculty.StaffID =
       Faculty_Categories.StaffID)
    INNER JOIN Categories AS C2
    ON C2.CategoryID =
       Faculty_Categories.CategoryID
    WHERE C2.CategoryID = Categories.CategoryID
    AND Faculty.Title = 'Professor')
    AS ProfCount
FROM Categories
WHERE
  (SELECT COUNT(Faculty.StaffID)
    FROM (Faculty
    INNER JOIN Faculty_Categories
    ON Faculty.StaffID =
       Faculty_Categories.StaffID)
    INNER JOIN Categories AS C3
    ON C3.CategoryID =
       Faculty_Categories.CategoryID
    WHERE C3.CategoryID = Categories.CategoryID
    AND Faculty.Title = 'Professor') < 3
```

A kérelmet a mintaadatbázisban CH14_Subjects_Fewer_3_Professors_RIGHT néven találhatjuk meg. Figyeljük meg, hogy az allekérdezés egy másolatát a SELECT záradékban is elhelyeztük, hogy lássuk a tárgycsoportonkénti tényleges számokat. Ebben az esetben ez jól működik, mivel a WHERE záradék allekérdezése érvényes nulla értéket ad vissza azokra a csoportokra, ahol nincs főállású egyetemi tanár. A helyes eredményt lent láthatjuk.

CH14_Subjects_Fewer_3_Professors_RIGHT

CategoryDescription	ProfCount
Accounting	0
Business	2
Computer Information Systems	0
Economics	1
Geography	0
History	1
Journalism	1
Math	1
Political Science	1
Accounting	1
Business	0
Computer Information Systems	1
Economics	0
Geography	0
History	0

Ahogy látjuk is, sok tárgycsoporthoz nincs főállású egyetemi tanár hozzárendelve. Bár ez a végső megoldás egyáltalán nem használja a HAVING záradékot, mégis itt mutatjuk be, rávilágítva, hogy nem mindig a HAVING az egyértelmű eszköz az ilyen feladatok esetében. Mindazonáltal a HAVING záradékot továbbra is használhatjuk számos „...kevesebb, mint..." típusú probléma megoldásához. Ha fel akarjuk sorolni az összes olyan vásárlót, aki havi 500 dollárnál kevesebbet vásárolt, de nem vagyunk kíváncsiak azokra, akik egyáltalán nem vásároltak semmit, akkor a HAVING jó (és valószínűleg gyorsabban végrehajtható) megoldást jelent. Viszont ha azokra a vásárlókra is kíváncsiak vagyunk, akik semmit sem vásároltak, akkor használjuk a fent ismertetett, nem a HAVING záradékot használó megoldást.

A HAVING használati területei

Mostanra valószínűleg már jól átlátjuk, hogyan összegezhetünk csoportokat összesítő függvényekkel és a GROUP BY záradékkal, és tudjuk, hogyan szűrhetjük a csoportosított adatokat a HAVING segítségével. A HAVING sokoldalú használatának bemutatására most felsorolunk néhány problémát, amelyet ezzel az új záradékkal megoldhatunk, utána pedig – a *Példák* részben – bemutatunk néhány példát is.

„Sorold fel a beszállítókat és beszállítónként a termékek átlagos szállítási idejét napban mérve. Azokra a beszállítókra vagyunk kíváncsiak, akiknek az átlagos szállítási ideje több nap, mint az összes beszállítóra számított átlagos szállítási idő!"
„Írd ki minden olyan termék nevét és összeladásait, ahol az összesített eladás meghaladja az összes termékre vonatkozó átlagos eladásokat az adott termékcsoportban!"
„Írd ki minden vásárló és rendelési dátum esetében a vásárló teljes nevét és a rendelt tételek végösszegét, feltéve, hogy az 1000 dollárnál nagyobb volt!"
„Hány rendelés tartalmazott csupán egyetlen terméket?"
„Mely ügynökök kötöttek 3000 dollárt meghaladó értékben üzleteket 2007 decemberében?"
„Sorold fel azokat az előadókat, akiknek kettőnél több egymást átfedő fellépése van!"
„Sorold fel az ügynök nevét, a rendezvényekhez tartozó szerződési összeget, és az ügynök összes megbízását azoknak az ügynököknek az esetében, ahol a megbízások összege meghaladja az 1000 dollárt!"
„Van olyan csapatkapitány, akinek a pontszáma nagyobb, mint a csapata bármelyik tagjának?"
„Sorold fel azokat a tekejátékosokat és átlagos játékpontszámukat, akiknek az átlaga nagyobb, mint 155!"
„Sorold fel azokat a tekejátékosokat, akiknek a legmagasabb pontszáma legalább húsz ponttal meghaladja a jelenlegi átlagukat!"
„A teljesített órákat figyelembe véve, tárgycsoportonként és hallgatónként sorold fel a tárgycsoport nevét, a hallgató nevét, és a hallgatónak az adott tárgycsoportba tartozó összes tárgyra vonatkozó átlagos osztályzatát azoknak a hallgatóknak az esetében, akiknek az átlaga több, mint 90!"

„Sorold fel tárgycsoportonként azoknak a tárgycsoportoknak a nevét és a kínált órák számát, ahol a tárgycsoportba három vagy több óra tartozik!"
„Sorold fel azokat a tanárokat és a tanár által tartott órák számát, ahol a tanár legalább egy, de kettőnél kevesebb tárgyat tanít!"
„Sorold fel azokat a recepteket, amelyek marhahúst és fokhagymát egyaránt tartalmaznak!"
„Összegezd a só mennyiségét receptosztályonként, és írd ki azokat a receptosztályokat, ahol háromnál több teáskanál sóra van szükség!"
„Mely recepttípusokból van kettőnél több?"

Példák

Mostanra elsajátítottuk annak a módszerét, hogy miként alkothatunk a HAVING záradék segítségével lekérdezéseket, és arra is láttunk néhány példát, hogy miféle kérdéseket tudunk a segítségével megválaszolni. Most lássunk néhány olyan példát, ahol az adatok csoportosítására és a csoportok összesített értékeinek szűrésére lesz szükség. Minden példa a mintaadatbázisokból származik.

Ezenkívül készítettünk az eredményhalmazokból is mintákat; ezek közvetlenül az SQL-utasításformák után találhatók. Az eredményhalmaz előtt található név ugyanaz, mint a CD-mellékleten található megfelelő SQL-lekérdezés neve. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH14"-vel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne felejtsük el, hogy a példáinkban használt oszlop- és táblanevek a B függelékben található mintaadatbázisok szerkezetéből származnak. Az egyszerűség kedvéért a következő példákban összevontuk a Fordítás és a Tisztázás lépéseit.*

Sales Orders adatbázis

„Sorold fel minden vásárló és rendelési dátum esetében a vásárló teljes nevét és a rendelt tételek összértékét, ha az az 1000 dollárt meghaladja!"

Fordítás/Tisztázás

```
Select customer first name and || ' ' || customer last name
as CustFullName, order date, and the
sum of (quoted price times * quantity ordered) as TotalCost
from the customers table
inner joined with the orders table
on customers.customer ID in the customers table matches
= orders.customer ID in the orders table;
```

and then inner joined with the order details table
on orders.order number in the orders table matches
= order_details.order number in the order details table,
grouped by customer first name,
customer last name, and order date,
having the sum of (quoted price times * quantity ordered)
greater than > 1000

SQL

```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   Orders.OrderDate,
   SUM(Order_Details.QuotedPrice *
   Order_Details.QuantityOrdered) AS TotalCost
FROM (Customers
INNER JOIN Orders
```

CH13_Order_Totals_By_Customer_And_Date (847 sor)

CustFullName	OrderDate	TotalCost
Alaina Hallmark	2007-09-02	$4,699.98
Alaina Hallmark	2007-09-14	$4,433.95
Alaina Hallmark	2007-09-21	$3,951.90
Alaina Hallmark	2007-09-22	$10,388.68
Alaina Hallmark	2007-09-30	$3,088.00
Alaina Hallmark	2007-10-12	$6,775.06
Alaina Hallmark	2007-10-22	$15,781.10
Alaina Hallmark	2007-10-30	$15,969.50
<< további sorok >>		

```
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber =
   Order_Details.OrderNumber
GROUP BY Customers.CustFirstName,
   Customers.CustLastName, Orders.OrderDate
HAVING SUM(Order_Details.QuotedPrice *
   Order_Details.QuantityOrdered) > 1000
```

Entertainment Agency adatbázis

„Mely ügynökök kötöttek 3000 dollárt meghaladó értékben üzletet 2007 decemberében?"

Fordítás/ Tisztázás

Select ~~the~~ agent first name, agent last name, ~~and the~~
sum ~~of~~ (contract price) as TotalBooked
from ~~the~~ agents ~~table~~
inner join~~ed with the~~ engagements ~~table~~
on agents.agent ID ~~in the agents table matches~~
= engagements.agent ID ~~in the engagements table~~
where ~~the~~ engagement start date ~~is~~
between ~~December 1, 2007,~~ '2007-12-01'
and ~~December 31, 2007,~~ '2007-12-31',
group~~ed~~ by agent first name, ~~and~~ agent last name~~, and~~
having ~~the~~ sum ~~of~~ (contract price) ~~greater than~~ > 3000

SQL

```
SELECT Agents.AgtFirstName, Agents.AgtLastName,
   SUM(Engagements.ContractPrice)
   AS TotalBooked
FROM Agents
INNER JOIN Engagements
ON Agents.AgentID = Engagements.AgentID
WHERE Engagements.StartDate
BETWEEN '2007-12-01' AND '2007-12-31'
GROUP BY Agents.AgtFirstName, Agents.AgtLastName
HAVING SUM(Engagements.ContractPrice) > 3000
```

CH14_Agents_Book_Over_3000_12_2007 (2 sor)

AgtFirstName	AgtLastName	TotalBooked
Marianne	Weir	$6,650.00
William	Thompson	$3,340.00

School Scheduling adatbázis

„A teljesített órákat figyelembe véve sorold fel tárgycsoport és hallgató szerint a tárgycsoport nevét, a tanuló nevét, és a tanuló átlagos osztályzatát a tárgycsoportból felvett valamennyi óra alapján! Azokat a tanulókat szeretnénk látni, akiknek az átlaga meghaladja a 90-et!"

Fordítás/ Tisztázás

Select category description, student first name, student last name,
~~and the average~~ avg(~~of~~ grade) as AvgOfGrade
from ~~the~~ categories ~~table~~

inner joined with the subjects table
on categories.category ID in the categories table matches
= subjects.category ID in the subjects table,
then inner joined with the classes table
on subjects.subject ID in the subjects table matches
= classes.subject ID in the classes table,
then inner joined with the student schedules table
on classes.class ID in the classes table matches
= student_schedules.class ID in the student schedules table,
then inner joined with the student class status table
on student_class_status.class status in the student class status
table matches
= student_schedules.class status in the student schedules table,
and finally inner joined with the students table
on students.student ID in the students table matches
= student_schedules.student ID in the student schedules table
where class status description is = 'Completed,'
grouped by category description, student first name, and
student last name;
and having the average avg(of grade) greater than > 90

SQL

```
SELECT Categories.CategoryDescription,
   Students.StudFirstName,
   Students.StudLastName,
   AVG(Student_Schedules.Grade) AS AvgOfGrade
FROM ((((Categories
INNER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
INNER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID)
INNER JOIN Student_Schedules
ON Classes.ClassID = Student_Schedules.ClassID)
INNER JOIN Student_Class_Status
ON Student_Class_Status.ClassStatus =
   Student_Schedules.ClassStatus)
INNER JOIN Students
ON Students.StudentID =
   Student_Schedules.StudentID
WHERE Student_Class_Status.ClassStatusDescription =
   'Completed'
GROUP BY Categories.CategoryDescription,
   Students.StudFirstName,
```

```
    Students.StudLastName
HAVING AVG(Student_Schedules.Grade) > 90
```

CH14_A_Students (13 sor)

CategoryDescription	StudFirstName	StudLastName	AvgOfGrade
Accounting	Betsey	Stadick	90.67
Accounting	Elizabeth	Hallmark	90.24
Accounting	Michael	Viescas	90.01
Art	Kendra	Bonnicksen	90.63
Art	Sarah	Thompson	91.40
English	Brannon	Jones	93.86
English	Elizabeth	Hallmark	92.90
English	John	Kennedy	93.70
English	Sarah	Thompson	97.39
Music	Brannon	Jones	93.26
Music	Karen	Smith	92.08
Music	Kerry	Patterson	93.28
Music	Marianne	Wier	98.26

„Sorold fel az összes olyan tanárt és az általuk tartott órákat, akik legalább egy, de háromnál kevesebb órát tartanak!"

Megjegyzés *A „HAVING COUNT nulla" problémát úgy előztük meg, hogy kijelentettük, hogy csak azokat a tanárokat szeretnénk látni, akik legalább egy órát tartanak.*

Fordítás/ Tisztázás: Select staff first name, staff last name, ~~and the~~ count ~~of classes~~ (*) as ClassCount from ~~the~~ staff ~~table~~ inner joined ~~with the~~ faculty classes ~~table~~ on staff.staff ID ~~in the staff table matches~~ = faculty_classes.staff ID ~~in the faculty classes table,~~ grouped by staff first name, ~~and~~ staff last name, ~~and~~ having ~~the~~ count ~~of classes~~ (*) ~~less than~~ < 3

SQL:

```
SELECT Staff.StfFirstName, Staff.StfLastName,
   COUNT(*) AS ClassCount
FROM Staff
INNER JOIN Faculty_Classes
ON Staff.StaffID = Faculty_Classes.StaffID
```

```
GROUP BY Staff.StfFirstName, Staff.StfLastName
HAVING COUNT(*) < 3
```

CH14_Staff_Class_Count_1_To_3 (8 sor)

StfFirstName	StfLastName	ClassCount
Jim	Wilson	2
Joyce	Bonnicksen	2
Katherine	Ehrlich	2
Kirk	DeGrasse	2
Luke	Patterson	1
Mariya	Sergienko	2
Peter	Brehm	2
Suzanne	Viescas	2

Bowling League adatbázis

„Sorold fel azokat a tekejátékosokat, akiknek a legmagasabb pontszáma több mint 20 ponttal magasabb, mint a jelenlegi átlaguk!"

Fordítás/ Tisztázás

Select bowler first name, bowler last name,
~~the average~~ avg(raw score) as CurrentAverage,
~~and the~~ max~~imum~~ (raw score) as HighGame
from ~~the~~ bowlers ~~table~~
inner join~~ed with the~~ bowler scores ~~table~~
on bowlers.bowler ID ~~in the bowlers table matches~~
= bowler_scores.bowler ID ~~in the bowler scores table,~~
group~~ed~~ by bowler first name, ~~and~~ bowler last name, ~~and~~
having ~~the~~ max~~imum~~ (raw score)
~~greater than~~ > ~~the average~~ avg(raw score) ~~plus~~ + 20

SQL

```
SELECT Bowlers.BowlerFirstName,
       Bowlers.BowlerLastName,
   AVG(Bowler_Scores.RawScore) AS CurrentAverage,
   MAX(Bowler_Scores.RawScore) AS HighGame
FROM Bowlers INNER JOIN Bowler_Scores
ON Bowlers.BowlerID = Bowler_Scores.BowlerID
GROUP BY Bowlers.BowlerFirstName,
       Bowlers.BowlerLastName
HAVING MAX(Bowler_Scores.RawScore) >
   (AVG(Bowler_Scores.RawScore) + 20)
```

CH14_Bowlers_Big_High_Score (15 sor)

BowlerFirstName	BowlerLastName	CurrentAverage	HighGame
Alaina	Hallmark	158	180
Angel	Kennedy	163	194
Caleb	Viescas	164	193
David	Fournier	157	178
David	Viescas	168	195
Gary	Hallmark	157	179
John	Kennedy	166	191
John	Viescas	168	193
<< további sorok >>			

Recipes adatbázis

„Sorold fel azokat a recepteket, amelyekhez marhahúsra és fokhagymára is szükség van!"

Fordítás/ Tisztázás

Select recipe title from ~~the~~ recipes ~~table~~
where ~~the~~ recipe ID ~~is~~ in ~~the~~
(select~~ion of~~ recipe ID
from ~~the~~ ingredients ~~table~~
inner join~~ed with the~~ recipe ingredients ~~table~~
on recipe_ingredients.ingredient ID
~~in the recipe ingredients table~~
~~matches~~ = ingredients.ingredient ID ~~in the ingredients table~~
where ~~the~~ ingredient name ~~is~~ = 'Beef'
or ~~the~~ ingredient name ~~is~~ = 'Garlic,'
group~~ed~~ by recipe ID ~~and~~
having ~~the~~ count ~~of the values in~~ (recipe ID) ~~equal to~~ = 2)

SQL

```
SELECT Recipes.RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID
IN (SELECT Recipe_Ingredients.RecipeID
   FROM Ingredients
   INNER JOIN Recipe_Ingredients
   ON Ingredients.IngredientID =
      Recipe_Ingredients.IngredientID
   WHERE Ingredients.IngredientName = 'Beef'
   OR Ingredients.IngredientName = 'Garlic'
   GROUP BY Recipe_Ingredients.RecipeID
   HAVING COUNT(Recipe_Ingredients.RecipeID) = 2)
```

CH14_Recipes_Beef_And_Garlic (1 sor)

RecipeTitle
Roast Beef

Megjegyzés *Ez a példa a GROUP BY és a HAVING allekérdezésben való ötletes használatát mutatja be, amikor olyan recepteket keresünk, amelyekben mindkét hozzávaló szerepel. Amikor a recept egyik hozzávalót sem tartalmazza, akkor nem jelenik meg az allekérdezésben. Amikor csak az egyik hozzávalót tartalmazza, akkor a számlálás eredménye 1 lesz, azaz a sort elvetjük. Csak amikor mindkét hozzávaló szerepel, akkor lesz a COUNT által visszaadott érték 2. Vigyázzunk, mert ha egy recept zúzott és egész fokhagymát is tartalmaz, ez a módszer nem működik!*
A COUNT a két fokhagyma-bejegyzésre 2 eredményt ad, és a receptet annak ellenére kiválasztja, hogy az nem tartalmaz marhahúst. Ha esetleg csodálkozunk, miért használtuk az OR műveletet, amikor a marhahús és a fokhagyma jelenlétét egyaránt megköveteljük a receptben, akkor nézzük át az OR használatára vonatkozó részt a 6. fejezetben.
A probléma egy másik lehetséges megoldását a 8. fejezetben közöltük.

Összefoglalás

A fejezetet annak a fejtegetésével kezdtük, hogy miként szűkíthetjük a csoportjainkat, és hogyan szűrhetjük őket a HAVING záradékban használt összesítő számítások segítségével. Bemutattuk a SELECT e végső záradékának utasításformáját, és egy egyszerű példán keresztül el is magyaráztuk.

Ez után arra mutattunk példát, hogy a sorok szűrésére mikor részesítsük előnyben a WHERE záradékot a HAVING záradékkal szemben. Elmagyaráztuk, hogy amikor csak módunkban áll, miért jobb, ha a szűrőt a WHERE záradékban helyezzük el. Mielőtt túlságosan elkényelmesedtünk volna a HAVING használata közben, bemutattunk egy elkerülendő helyzetet, ami az esetlegesen nulla elemű csoportok számlálásakor fordul elő, de arra is muttatunk módszert, hogy miként oldhatjuk meg az ilyen jellegű problémákat.

Végül összefoglaltuk, hogy miért hasznos a HAVING záradék, és a HAVING segítségével megoldandó feladatokat is közöltünk, majd példákat mutattunk be arra, hogy miként alkothatunk a HAVING záradékot felhasználó kérelmeket. A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Sorold fel azokat a beszállítókat, valamint az átlagos szállítási időt napban mérve, ahol a szállítási idő nagyobb, mint az összes beszállítóra vonatkozó átlagos szállítási idő!"*
   (Tipp: az összes beszállítóra vonatkozó átlagos szállítási idő lekérdezéséhez használjunk allekérdezést.)
   A megoldás itt található: CH14_Vendor_Avg_Delivery_GT_Overall_Avg (5 sor).
2. *„Írd ki azoknak a termékeknek a nevét és az összesített eladásait, ahol a termék összesített eladásai meghaladják a termékcsoporton belüli átlagos eladásokat!"*
   (Tipp: az összehasonlítási érték kiszámításához először összegezzük a termékcsoporton belüli termékek eladásait a SUM függvény segítségével, és csoportonként átlagoljuk az összegeket az AVG felhasználásával.)
   A megoldás itt található: CH14_Sales_By_Product_GT_Category_Avg (13 sor).
3. *„Hány rendelés tartalmaz csupán egyetlen terméket?"*
   (Tipp: a FROM záradékban használjunk olyan belső lekérdezést, amely felsorolja a rendelési számokat azokra a rendelésekre, amelyek csupán egyetlen sorból állnak, majd számoljuk meg a sorokat a külső SELECT záradékban.)
   A megoldás itt található: CH14_Single_Item_Order_Count (1 sor).

Entertainment Agency adatbázis

1. *„Sorold fel azokat a előadókat, akiknek kettőnél több egymást átfedő fellépésük van!"*
   (Tipp: alkossunk allekérdezést, és használjuk a HAVING záradékot, illetve a COUNT függvényt.)
   A megoldás itt található: CH14_Entertainers_MoreThan_2_Overlap (1 sor).
2. *„Sorold fel az ügynök nevét, a lekötött rendezvények összesített szerződési összegét és az ügynök összes megbízását azokban az esetekben, ahol az ügynök megbízásainak összértéke meghaladja az 1000 dollárt!"*
   (Tipp: induljunk ki a 13. fejezet hasonló problémájából, és próbáljuk azt HAVING záradékkal kibővíteni.)
   A megoldás itt található: CH14_Agent_Sales_Big_Commissions (4 sor).

School Scheduling adatbázis

1. *„Sorold fel tárgycsoportonként a tárgycsoport nevét és a kínált órák számát azoknak a tárgycsoportoknak az esetében, amelyek három vagy több órát kínálnak!"*
   (Tipp: kapcsoljuk össze a tárgycsoportokat a tárgyakkal, majd az órákkal, a COUNT függvénnyel számoljuk meg a sorokat, és a megfelelő HAVING záradékkal megkapjuk a végeredményt.)
   A megoldás itt található: CH14_Category_Class_Count_3_Or_More (11 sor).
2. *„Sorold fel a tanárokat és azokat az órákat, amelyeket tartanak! Azokra a tanárokra vagyunk kíváncsiak, akik háromnál kevesebb órát tartanak."*
   (Tipp: ez a „HAVING COUNT nulla" helyzet, ezért használjunk allekérdezést.)
   A megoldás itt található: CH14_Staff_Teaching_LessThan_3 (12 sor).

3. *„Számold össze azokat az órákat, amelyeket minden tanár tart!"*
   (Tipp: a probléma megoldásához valójában nem szükséges a HAVING záradék, de esetleg a GROUP BY záradékkal szeretnénk – helytelenül – megoldani.)
   A helyes megoldást a CH14_Staff_Class_Count_Subquery (27 sor) táblában találjuk, a helytelent pedig a CH14_Staff_Class_Count_GROUPED (23 sor) táblában.

Bowling League adatbázis

1. *„Van olyan csapatkapitány, akinek a tiszta pontszáma magasabb, mint bármelyik csapattársáé?"*
   (Tipp: a kapitányok legmagasabb tiszta pontszámát úgy kaphatjuk meg, hogy a JOIN segítségével, a csapatkapitány-azonosító alapján összekapcsoljuk a csapatok tábláját a játékosok táblájával, majd ezt a pontszámok táblájával. A többi csapattag legnagyobb ponszámával egy HAVING záradék segítségével, allekérdezésből végezzük el az összehasonlítást.)
   A megoldás itt található: CH14_Captains_Who_Are_Hotshots (0 sor). (Nincs olyan kapitány, aki jobban tekézik, mint a csapattársai!)
2. *„Sorold fel azoknak a tekejátékosoknak a nevét és pontszámátlagát, akiknek az átlaga meghaladja a 155-öt!"*
   (Tipp: egyszerű HAVING záradékra lesz szükségünk, ahol literális számértékkel hasonlítjuk össze a pontok átlagát az AVG függvény segítségével.)
   A megoldás itt található: CH14_Good_Bowlers (17 sor).
3. *„Sorold fel minden olyan tekejátékos keresztnevét és vezetéknevét, akinek az átlagos pontszáma nagyobb, mint az összesített átlagos pontszám, vagy egyenlő azzal!"*
   (Tipp: a megoldás módszerét a 12. fejezet példái között mutattuk be. Akkor a WHERE záradékban elhelyezett allekérdezéssel oldottuk meg a feladatot – most oldjuk meg a HAVING záradék segítségével!)
   A megoldás itt található: CH14_Better_Than_Overall_Average_HAVING (17 sor).

Recipes adatbázis

1. *„Összegezd a sómennyiséget receptosztályonként, és írd ki azokat a receptosztályokat, amelyekhez három teáskanálnál több só kell!"*
   (Tipp: öt táblából álló összetett JOIN alkalmazására lesz szükségünk, hogy kiszűrjük a sót és a teáskanalat, a SUM segítségével összegezzük az eredményt, majd elvessük a három teáskanálnál többet használó receptosztályokat.)
   A megoldás itt található: CH14_Recipe_Classes_Lots_Of_Salt (1 sor).
2. *„Mely receptosztályokban van kettő vagy több recept?"*
   (Tipp: JOIN segítségével kapcsoljuk össze a receptosztályok és a receptek tábláit, majd a COUNT függvénnyel számoljuk meg az eredményt, és csak azokat az osztályokat tartsuk meg a HAVING záradék segítségével, amelyekben két vagy több recept található.)
   A megoldás itt található: CH14_Recipe_Classes_Two_Or_More (4 sor).

V

Adathalmazok módosítása

15

Adathalmazok frissítése

„A változás, a folyamatos változás,
az elkerülhetetlen változás – ez manapság
a társadalom meghatározó tényezője."
– Isaac Asimov

A fejezet témakörei

- Mi az az UPDATE?
- Az UPDATE utasítás
- Az UPDATE használati területei
- Példák
- Összefoglalás
- Önálló feladatok

Ahogyan az a II., III. és IV. részben kiderült, SELECT utasítással adatokat lekérdezni a táblákból kihívás és szórakozás is egyben. (Na jó, néha sokkal inkább kihívás, mint szórakozás.) Ha a kérdések megválaszolásán kívül mást nem akarunk tenni, akkor a könyv eme utolsó részére nincs is szükségünk. A legtöbb valós alkalmazásban azonban nem csak összetett kérdésekre kell választ adni, hanem lehetővé kell tenni a felhasználóknak az adatok módosítását, törlését, vagy új adatok hozzáadását is. Az SQL-szabvány az adatok lekérdezésére használható SELECT utasítás mellett három olyan utasítást is meghatároz, amely az adatok módosítására használható. Ebben a fejezetben a három közül az elsőről – az UPDATE-ről – lesz szó.

Mi az az UPDATE?

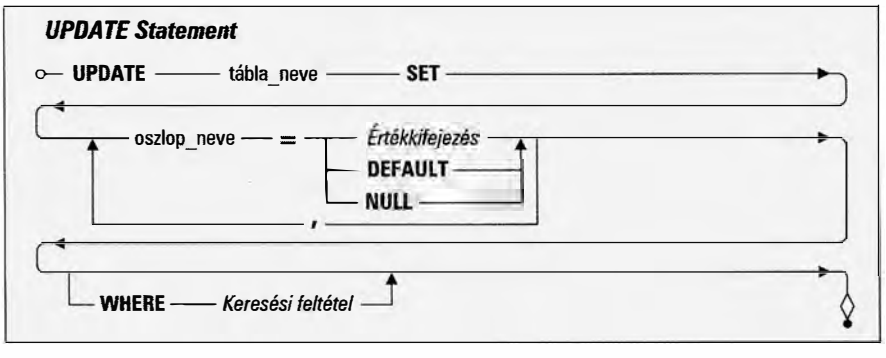
A SELECT utasítás lehetővé teszi adathalmazok lekérdezését a táblákból. Az UPDATE (frissítés) szintén adathalmazokkal dolgozik, de azt teszi lehetővé, hogy *megváltoztassuk* egy vagy több sor egy vagy több adatát. Mostanra már ismerősek számunkra a kifejezések is. Egy oszlop módosításakor egyszerűen egy kifejezést adunk értékül az oszlopnak.

Legyünk azonban óvatosak, mert az UPDATE utasítás hatékonysága veszélyt is jelent! Az esetek döntő többségében csak egy vagy néhány sort akarunk módosítani. Ha nem vagyunk elővigyázatosak, akkor véletlenül egyszerre sorok ezreit módosíthatjuk. Ennek elkerülésére bemutatunk egy módszert, amellyel először kipróbálhatjuk az utasítást.

Megjegyzés *Az összes példát és a feladatok megoldását megtalálhatjuk a megfelelő mintaadatbázisok módosítható („Modify" utótagú) változatában (SalesOrdersModify, EntertainmentAgencyModify, SchoolSchedulingModify, BowlingLeagueModify).*

Az UPDATE utasítás

Az UPDATE utasítás formája sokkal egyszerűbb, mint a SELECT utasításé, amelyről a korábbi fejezetekben tanultunk. Az UPDATE utasításnak csak három záradéka van: az UPDATE, a SET és a nem kötelező WHERE záradék, ahogyan az a 15.1. ábrán látható.



15.1. ábra
Az UPDATE utasítás szintaxisdiagramja

Az UPDATE kulcsszó után a módosítandó tábla nevét adjuk meg, a SET kulcsszó után pedig egy vagy több olyan záradék következik, amely egy-egy oszlophoz új értéket rendel. Legalább egy ilyen záradéknak kötelező lennie, de annyit sorolhatunk fel, amennyire csak szükségünk van több oszlop módosításához. A nem kötelező WHERE záradékkal szabályozhatjuk, hogy a tábla mely sorait módosítsa az utasítás.

Az egyszerű UPDATE utasítás

Lássunk egy példát, amelyben az oszlopnak egyszerűen egy kifejezéssel adunk új értéket!

Megjegyzés *Ebben a fejezetben is a 4. fejezetben ismertetett „Kérelem – Fordítás – Tisztázás – SQL" módszert fogjuk használni.*

„Növeld meg minden termék fogyasztói árát 10 százalékkal!"

No, ez egy kicsit trükkös! Nem könnyű SQL-szerű nyelvre fordítani az ilyen kérelmeket, mert sem az angol, sem a magyar nyelvben nem abban a sorrendben szerepelnek a mondatrészek, mint amelyet az UPDATE utasítás megkövetel. Vegyük szemügyre közelebbről a kérelmet, és próbáljuk meg meghatározni a) a módosítandó tábla, és b) a módosítandó oszlopok nevét. Fogalmazzuk át a kérelmet úgy, hogy a mondatrészek ebben a sorrendben következzenek:

„Módosítsd a termékeket úgy, hogy a fogyasztói ár 10 százalékkal nőjön!"

Fordítás Update the products table by setting the retail price equal to the retail price plus 10 percent of the price
(Módosítsd a termékek tábláját úgy, hogy a fogyasztói ár egyenlő legyen a fogyasztói ár plusz tíz százalékkal.)

Tisztázás Update ~~the~~ products ~~table by~~ ~~setting the~~ retail price ~~equal to~~ = ~~the~~ retail price ~~plus~~ + (~~.10 percent of the~~ * retail price)

SQL
```
UPDATE Products
SET Price = Price + (0.1 * Price)
```

Figyeljük meg: nem írhattuk azt, hogy SET Price + 10%. Az egyenlőségjel bal oldalán szerepelnie kell az oszlopnévnek, a jobb oldalára pedig egy olyan kifejezést kell írnunk, amely megadja az új értéket. Ha az új érték kiszámításához fel kell használnunk az oszlop jelenlegi értékét, akkor szükség szerint szerepeltethetjük az oszlopnevet az egyenlőségjel jobb oldalán is. Ezt a szabályt nagyon világosan fogalmazza meg az SQL-szabvány: az adatbázisrendszernek *előbb* minden értékadó kifejezést ki kell értékelnie, és csak azután kezdhet hozzá a sorok módosításához. Ennek megfelelően az adatbázis, mielőtt bármit is módosítana, feloldja a Price (Ár) oszlopra vonatkozó hivatkozásokat az egyenlőségjel jobb oldalán, azaz behelyettesíti a Price oszlop módosítás előtti értékét.

Ez a fajta értékadó utasítás minden programozási nyelvben azonosan működik. Igaz, úgy tűnik, mintha az oszlophoz önmaga értékét rendelnénk hozzá, valójában azonban elővesszük a módosítás előtti értéket, hozzáadunk tíz százalékot, és az így kapott eredményt használjuk fel az értékadáskor.

Csak a kiválasztott sorok módosítása

Biztos, hogy mindig a tábla összes sorát módosítani akarjuk? Valószínűleg nem. Az UPDATE utasításhoz egy WHERE záradék hozzáadásával korlátozhatjuk, hogy az utasítás mely sorokat módosítsa. Nézzünk egy másik feladatot:

„A ruházati beszállítónk az imént 4 százalékos áremelést jelentett be. Növeld meg a ruházati termékek fogyasztói árát 4 százalékkal!"

Fogalmazzuk át a kérelmet:

„Módosítsd a termékeket úgy, hogy a fogyasztói ár egyenlő legyen a fogyasztói ár plusz négy százalékkal, minden ruházati termék (3-as termékcsoport) esetében!"

Fordítás Update the products table by setting the retail price equal to retail price times 1.04 for all products in category 3
(Módosítsd a termékek tábláját úgy, hogy a fogyasztói ár egyenlő legyen a fogyasztói ár szorozva 1,04-dal azoknak a termékeknek az esetében, amelyeknek a termékcsoportja a 3-as.)

Tisztázás Update ~~the~~ products ~~table by~~ ~~setting the~~ retail price ~~equal to~~ = retail price ~~times~~ * 1.04 ~~for all~~ where ~~products in~~ category ID = 3

SQL
```
UPDATE Products
SET RetailPrice = RetailPrice * 1.04
WHERE CategoryID = 3
```

Megjegyzés *A lekérdezésben egyszerűsítettük a számítást, és inkább 1,04-dal szoroztuk meg az árat, nem pedig hozzáadtuk az eredeti érték 0,04-szorosát. Az eredmény matematikai szempontból azonos, de gyorsabban végrehajtható, mert hatékonyabban, két művelet (ár plusz ár szorozva 0,04-dal) helyett csupán eggyel (ár szorozva 1,04-dal) végzi el a feladatot.*

Azok után, hogy a 11. fejezetben már az allekérdezésekkel is megküzdöttünk, ez igazán egyszerű volt, ugye? De ami késik, nem múlik: az UPDATE utasítások WHERE záradékában gyakran fogunk használni allekérdezéseket – de erre majd csak a fejezet későbbi részében kerül sor.

Első a biztonság: ellenőrizzük, hogy valóban a megfelelő sorokat módosítjuk!

Azt ajánljuk, hogy még a legegyszerűbb UPDATE utasításnál is először mindig ellenőrizzük azt, hogy valóban a megfelelő sorokat fogjuk-e módosítani. De hogyan is kezdjünk hozzá? Amint már említettük, az esetek többségében egy WHERE záradékkal leszűkítjük a módosítandó sorok körét. Miért is ne írhatnánk olyan SELECT utasítást, amely visszaadja azokat a sorokat, amelyeket módosítani akarunk? Folytatva korábbi példánkat, kérjünk le az adatbázisból egy olyan oszlopot, amely alapján azonosítani tudjuk a kérdéses sort, a módosítandó oszlop régi értékét, és annak a kifejezésnek az értékét, amelyet értékül akarunk adni a módosítandó oszlopnak:

„Írd ki azoknak a termékeknek a nevét, a fogyasztói árát és a fogyasztói árat 4 százalékkal megnövelve a termékek tábláját, amelyek a 3-as termékcsoportba tartoznak!"

Fordítás Select product name, retail price, and retail price times 1.04 from the products table for products in category ID 3

(Írd ki a termék nevét, a fogyasztói árat és a fogyasztói árat megszorozva 1,04-dal a termékek táblájából, azokra a termékekre, amelyek termékcsoportja a 3-as.)

Tisztázás Select product name, retail price, ~~and~~ retail price ~~times~~ * 1.04
from ~~the~~ products ~~table~~
~~for~~ where ~~products in~~ category ID = 3

SQL
```
SELECT ProductName, RetailPrice,
   RetailPrice * 1.04 As NewPrice
FROM Products
WHERE CategoryID = 3
```

Az eredményt a 15.2. ábra mutatja.

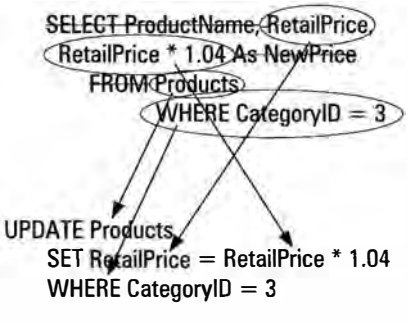
ProductName	RetailPrice	NewPrice
Ultra-Pro Rain Jacket	$85.00	$88.40
StaDry Cycling Pants	$69.00	$71.76
Kool-Breeze Rocket Top Jersey	$32.00	$33.28
Wonder Wool Cycle Socks	$19.00	$19.76

15.2. ábra

A módosítandó sorok ellenőrzése

Figyeljük meg, hogy a lekérdezésben szerepeltettük a termék nevét, így könnyen megállapíthatjuk, hogy mely sorokat fogjuk módosítani. Ha a kívánt eredményt kaptuk, akkor átalakíthatjuk a SELECT utasítást a megfelelő UPDATE utasítássá, törölve a szükségtelen részeket, illetve megcserélve a FROM és SELECT záradékot.

Egyszerűen húzzuk ki a felesleges szavakat, írjuk át a táblanevet az UPDATE záradékba, a céloszlopot és a kifejezést egy egyenlőségjellel elválasztva helyezzük át a SET záradékba, másoljuk át a WHERE záradékot, és már készen is vagyunk.



15.3. ábra

SELECT lekérdezés UPDATE utasítássá alakítása

Egy rövid kitérő: tranzakciók

Mielőtt túl mélyre merülnénk az adatmódosítások tengerében, meg kell ismerkednünk az SQL egy nagyon fontos szolgáltatásával. Az SQL-szabvány meghatároz egy *tranzakciónak* nevezett dolgot, amely arra szolgál, hogy elhatárolja és összefogja a táblákban tárolt adatok módosításainak egy sorozatát. Az SQL tranzakcióira úgy gondolhatunk, mint egy-egy valós vásárlási tranzakcióra, amely akár online, akár egy üzletben történhet. A tranzakciót akkor kezdeményezzük, amikor leadjuk a rendelést, a következő lépés a megrendelt termék kifizetése, a tranzakció pedig akkor fejeződik be, amikor átvesszük az árut. Ha az áru nem érkezik meg, akkor visszakapjuk a pénzünket. Ha az áru nem megfelelő, akkor visszaküldjük, és visszakérjük a pénzünket.

Az SQL-szabvány három utasítást kínál a fenti események modellezésére. A módosítások kezdeményezése előtt kiadott START TRANSACTION paranccsal jelezzük, hogy az ezután következő módosításokat el akarjuk határolni a többitől, és ellenőrizni akarjuk. Gondoljunk erre úgy, mint a rendelés feladására. Elvégezzük az adatok módosítását – rögzítjük a fizetést és az áru átvételét. Ha minden rendben lezajlott, akkor a COMMIT (véglegesítés vagy jóváhagyás) paranccsal véglegesíthetjük a módosításokat. Ha valami hiba csúszott a folyamatba (a kifizetés vagy az áruátvétel meghiúsult), akkor a ROLLBACK (visszagörgetés) paranccsal állíthatjuk vissza az adatainkat abba az állapotba, amelyben a tranzakció megkezdése előtt voltak.

A fenti vásárlás-eladás példa egy kicsit suta, a tranzakciókezelés azonban az SQL nagyon hasznos szolgáltatása, ami különösen akkor használható jól, ha egyszerre több táblát kell módosítanunk. Tranzakciók alkalmazásával biztosíthatjuk, hogy vagy az összes módosítás sikerül, vagy egy sem. A vevő nem szeretné kifizetni az árut átvétel nélkül, de az eladó sem akarja átadni az árut a kifizetés nélkül. Jegyezzük meg, hogy a fentiek nem csak az ebben a fejezetben tárgyalt UPDATE, hanem a következő két fejezetben szereplő INSERT és DELETE parancsokra is vonatkoznak.

Nem minden adatbázis-kezelő valósítja meg a tranzakciókat, és a tranzakciókezelésre használatos utasítások formája is eltérő lehet. Egyes adatbázisrendszerek lehetővé teszik tranzakciók egymásba ágyazását, illetve több jóváhagyási pont létrehozását. Néhány végfelhasználóknak szánt adatbázisrendszer, mint például a Microsoft Office Access, a háttérben mindig új tranzakciót kezd, amikor olyan parancsot adunk ki, ami módosítja az adatokat. Aki már használt Microsoft Accesst, bizonyára emlékszik azokra az üzenetekre, amelyek arról tájékoztatnak, hogy hány sort fogunk módosítani, és lesz-e sikertelen módosítási kísérlet. Ilyenkor vagy elfogadjuk a módosításokat, vagy töröljük azokat (ez utóbbi felel meg a ROLLBACK-nek). Mint ilyen esetekben mindig, azt javasoljuk, hogy nézzünk utána a témának saját adatbázisrendszerünk dokumentációjában.

Több oszlop módosítása

Ahogyan azt az UPDATE utasítás szintaxisdiagramja (15.1 ábra) alapján gondolhatjuk, egyszerre több módosítandó oszlop is megadható, úgy, hogy az egyes értékadásokat vesszővel elválasztva egymás után írjuk. Jegyezzük meg: az adatbázis a felsorolt értékadások mindegyikét végrehajtja a WHERE záradék által meghatározott összes soron. Nézzünk egy példát a School Scheduling adatbázis adatainak frissítésére:

„Módosítsd az órákat az alábbiak szerint: legyen az osztályterem az 1635-ös, és cseréld a beütemezett napokat hétfőről, szerdáról és péntekről keddre, csütörtökre és szombatra minden rajzóra esetében (tantárgy-azonosító: 13)"

Fordítás Update classes and set classroom ID to 1635,Monday schedule to false,Wednesday schedule to false, Friday schedule to false, Tuesday schedule to true,Thursday schedule to true, and Saturday schedule to true for all classes that are subject ID 13
(Módosítsd az órákat: állítsd a tanterem-azonosítót 1635-re, a hétfői alkalmat hamisra, a szerdai alkalmat hamisra, a pénteki alkalmat hamisra, a keddi alkalmat igazra, a csütörtöki alkalmat igazra és a szombati alkalmat igazra minden olyan sorban, ahol a tantárgy-azonosító 13.)

Tisztázás Update classes ~~and~~
set classroom ID ~~to~~ = 1635,Monday schedule ~~to~~ = false,
Wednesday schedule ~~to~~ = false, Friday schedule ~~to~~ = false,
Tuesday schedule ~~to~~ = true,Thursday schedule ~~to~~ = true,
~~and~~ Saturday schedule ~~to~~ = true
~~for all classes that are~~ where subject ID = 13

SQL
```
UPDATE Classes
SET ClassRoomID = 1635, MondaySchedule = 0,
   WednesdaySchedule = 0, FridaySchedule = 0,
   TuesdaySchedule = 1, ThursdaySchedule = 1,
   SaturdaySchedule = 1
WHERE SubjectID = 13
```

Megjegyzés *A legtöbb adatbázisrendszer 0-val jelöli a hamis, és –1-gyel vagy 1-gyel az igaz értéket, de nem árt, ha ezt ellenőrizzük az adatbázisrendszerünk dokumentációjában.*

Ha egészen biztosak akarunk lenni abban, hogy csak a hétfői, szerdai és pénteki napokra tervezett rajzórákat módosítjuk, akkor az alábbiak szerint egészítsük ki a WHERE záradékot:

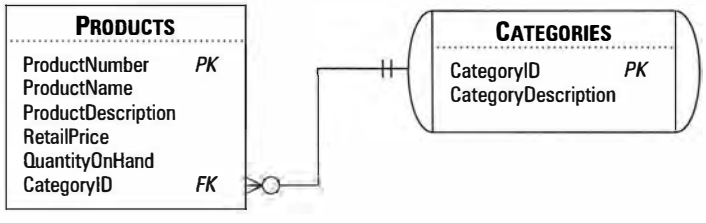
SQL
```
UPDATE Classes
SET ClassRoomID = 1635, MondaySchedule = 0,
   WednesdaySchedule = 0, FridaySchedule = 0,
   TuesdaySchedule = 1, ThursdaySchedule = 1,
   SaturdaySchedule = 1
```

```
WHERE SubjectID = 13
   AND MondaySchedule = 1
   AND WednesdaySchedule = 1
   AND FridaySchedule = 1
```

Figyeljük meg: a szűréskor a módosítás *előtti* értéket vesszük alapul, amikor a feltételeket megfogalmazzuk. Ez a módosított UPDATE utasítás minden olyan sorra hajtódik végre, ahol a subjectID (tantárgy-azonosító) 13, a MondaySchedule, WednesdaySchedule és FridaySchedule mezők értéke pedig igaz (1). Minden olyan sorban, amelyre teljesülnek ezek a feltételek, az UPDATE utasítás módosítja a ClassRoomID (osztályterem-azonosító) és a Schedule (Beütemezett nap) oszlopokat. Ha ugyanezt az utasítást ismételten lefuttatjuk, azt tapasztaljuk, hogy az adatbázis egyetlen sort sem módosít, mivel az első futáskor már megváltoztattunk minden olyan sort, amely kielégítette a szűkítő feltételeket.

Allekérdezés alkalmazása a sorok szűrésére

Az előző részek példáiban a 3-as termékcsoportba tartozó termékeket vagy a 13-as tantárgy-azonosítójú órákat módosítottuk. A valóságban az ehhez hasonló azonosítókódok nem hordoznak túl sok jelentést. Sokkal valószínűbb, hogy azt mondjuk: „ruházati termékek" vagy „rajzórák". A SELECT lekérdezések FROM záradékában JOIN záradékokkal soroljuk fel a kapcsolódó táblákat, és ezután a kapcsolódó táblában szereplő, szemléletesebb jelentéssel bíró értékeket jelenítjük meg. Ahhoz, hogy a kapcsolatokat meg tudjuk határozni, mint mindig, természetesen most is tisztában kell lennünk a táblák közötti összefüggésekkel. A példánkhoz szükséges adatbázistáblákat a 15.4. ábra mutatja.



15.4. ábra

A termékek és a termékcsoport-leírások összekapcsolásához szükséges táblák

Vegyük elő ismét a termékmódosító utasításunk ellenőrzésére szolgáló lekérdezést, de ezúttal tegyük bele a termékcsoportokat tartalmazó Categories táblát is:

SQL
```
SELECT ProductName, RetailPrice,
   RetailPrice * 1.04 As NewPrice
FROM Products
INNER JOIN Categories
ON Products.CategoryID = Categories.CategoryID
WHERE Categories.CategoryDescription = 'Clothing'
```

A ruházat (Clothing) értékre történő szűrés sokkal szemléletesebb, mint a 3-as termékcsoport-azonosító használata. Figyeljük meg azonban, hogy az UPDATE szintaxisdiagramja szerint (15.1. ábra) az UPDATE kulcsszót csak egyetlen táblanév követheti; nem írhatjuk oda a Categories tábla beillesztéséhez szükséges INNER JOIN záradékot. Mi hát a megoldás?

Emlékezzünk vissza a 11. fejezetre, amelyben bemutattuk, hogy a WHERE záradékban hogyan fogalmazhatunk meg olyan feltételt, amely egy másik táblában szereplő érték alapján szűri a sorokat. Oldjuk meg ismét az ármódosításról szóló feladatot, de most úgy, hogy egy allekérdezés segítségével szemléletesebb szűkítő feltételt adunk meg:

„Módosítsd a termékeket úgy, hogy a fogyasztói ár egyenlő legyen a fogyasztói ár plusz négy százalékkal, minden ruházati termék esetében!"

Fordítás Update the products table by setting the retail price equal to retail price times 1.04 for the products whose category ID is equal to the selection of the category ID from the categories table where the category description is clothing
(Módosítsd a termékek tábláját úgy, hogy a fogyasztói ár egyenlő legyen a fogyasztói ár szorozva 1,04-dal, azoknak a termékekeknek az esetében, amelyeknek a termékcsoport-azonosítója egyenlő a termékcsoportok táblájának termékcsoport-azonosítójával abban a sorban, ahol a termékcsoport leírása „ruházat".)

Tisztázás Update ~~the~~ products ~~table by~~ ~~setting the~~ retail price ~~equal to~~ = retail price ~~times~~ * 1.04 for the products whose where category ID is equal to = ~~the~~ (~~selection of the~~ category ID from ~~the~~ categories ~~table~~ where ~~the~~ category description ~~is~~ = 'Clothing')

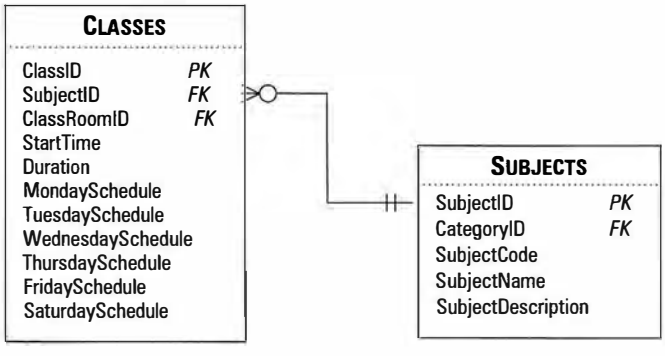
SQL
```
UPDATE Products
SET RetailPrice = RetailPrice * 1.04
WHERE CategoryID =
   (SELECT CategoryID
   FROM Categories
   WHERE CategoryDescription = 'Clothing')
```

Igaz, hogy a fenti forma nem annyira egyértelmű, mint egy egyszerű WHERE záradék, amely egy JOIN záradékkal kapcsolt tábla oszlopára hivatkozik, de a célunkat elérjük vele.

Figyelem! *Figyeljük meg, hogy egyenlőség műveletet használtunk a Products (Termékek) tábla oszlopa és az allekérdezés által visszaadott érték között. Ahogyan ezt a 11. fejezetben megjegyeztük, ha egyenlőségvizsgálatot akarunk végezni egy allekérdezés által visszaadott értékkel, akkor az allekérdezés csak egyetlen sort adhat vissza. Ha több mint egy termékcsoport-leírás (CategoryDescription oszlop)*

tartalmazza a „Clothing" értéket, akkor a lekérdezés meghiúsul. A példánk esetében azonban meglehetősen biztosak lehetünk abban, hogy a „Clothing" leírásra szűkítve csak egyetlen CategoryID értéket kapunk eredményül. Minden olyan esetben azonban, amikor nem biztos, hogy az allekérdezés csak egyetlen értéket ad vissza, az IN összehasonlító műveletet kell használnunk az egyenlőségvizsgálat helyett.

Oldjuk meg a tantervmódosító példát azonos módszerrel, és a nem túl szemléletes SubjectID helyett használjuk a tantárgykódot vagy a tantárgy nevét, amely a Subject tábla SubjectCode, illetve SubjectName mezőjében található. A szükséges táblák szerkezete a 15.5. ábrán látható.



15.5. ábra

A tantárgyak és az órák összefüggéséhez szükséges táblák

Oldjuk meg a feladatot ismét, de most allekérdezés használatával:

„Módosítsd az órákat az alábbiak szerint: legyen az osztályterem az 1635-ös, és cseréld a beütemezett napokat hétfőről, szerdáról és péntekről keddre, csütörtökre, és szombatra minden rajzóra esetében!"

Fordítás Update classes and set classroom ID to 1635,Monday schedule to false,Wednesday schedule to false, Friday schedule to false, Tuesday schedule to true,Thursday schedule to true, and Saturday schedule to true for all classes whose subject ID is in the selection of subject IDs from the subjects table where subject name is 'Drawing'

(Módosítsd az órákat: állítsd a tanterem-azonosítót 1635-re, a hétfői alkalmat hamisra, a szerdai alkalmat hamisra, a pénteki alkalmat hamisra, a keddi alkalmat igazra, a csütörtöki alkalmat igazra és a szombati alkalmat igazra minden olyan sorban, ahol a tantárgy-azonosító egyenlő a tantárgyak táblájában levő tantárgy-azonosítóval abban a sorban, ahol a tantárgy leírása „Rajz".)

Tisztázás Update classes ~~and~~
set classroom ID ~~to~~ = 1635,Monday schedule ~~to~~ = false,
Wednesday schedule ~~to~~ = false, Friday schedule ~~to~~ = false,
Tuesday schedule ~~to~~ = true,Thursday schedule ~~to~~ = true,
~~and~~ Saturday schedule ~~to~~ = true
~~for all classes whose~~ where subject ID ~~is~~ in
~~the~~ (select~~ion of~~ subject ID~~s~~
from ~~the~~ subjects ~~table~~
where subject name ~~is~~ = 'Drawing')

SQL
```
UPDATE Classes
SET ClassRoomID = 1635, MondaySchedule = 0,
   WednesdaySchedule = 0, FridaySchedule = 0,
   TuesdaySchedule = 1, ThursdaySchedule = 1,
   SaturdaySchedule = 1
WHERE SubjectID IN
   (SELECT SubjectID
   FROM Subjects
   WHERE SubjectName = 'Drawing')
```

Figyeljük meg: bár biztosak vagyunk benne, hogy csak egyetlen tantárgy leírása lehet „Drawing", a biztonság kedvéért mégis inkább az IN műveletet használtuk.

Egyes adatbázisrendszerek megengedik a JOIN használatát az UPDATE záradékban

Több adatbázisrendszer, amelyek közül a legismertebbek a Microsoft rendszerei (Microsoft Access és Microsoft SQL Server), lehetővé teszi, hogy az UPDATE utasítás FROM részében JOIN záradékkal kapcsolt táblát is szerepeltessünk. A korlátozás csupán annyi, hogy a JOIN-nak az egyik tábla elsődleges kulcsa és a másik tábla idegen kulcsa közötti kapcsolatra kell hivatkoznia, hogy az adatbázisrendszer ki tudja találni, hogy melyik sort vagy sorokat akarjuk módosítani. Ennek segítségével elkerülhetjük a WHERE záradékban az allekérdezés használatát akkor is, ha egy másik táblában szereplő érték alapján szeretnénk szűrni.

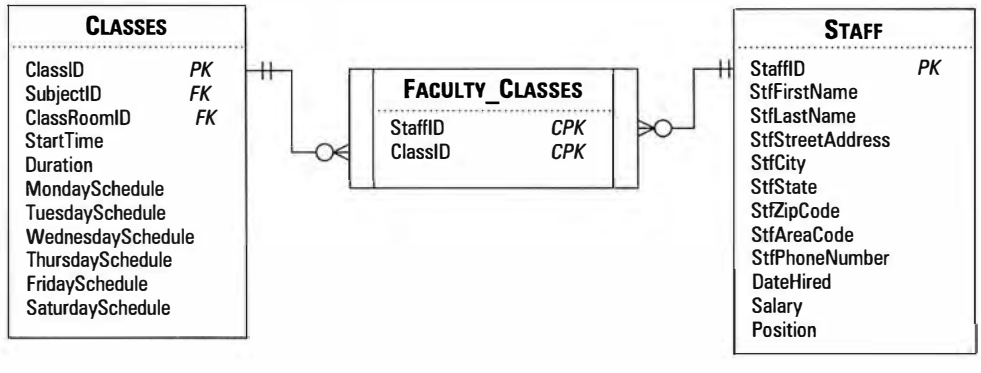
Így – ha adatbázisunk lehetővé teszi – a rajzórák módosításának feladatát megoldhatjuk az alábbi utasítással is:

SQL
```
UPDATE Classes
INNER JOIN Subject
ON Classes.SubjectID = Subjects.SubjectID
SET ClassRoomID = 1635, MondaySchedule = 0,
   WednesdaySchedule = 0, FridaySchedule = 0,
   TuesdaySchedule = 1, ThursdaySchedule = 1,
   SaturdaySchedule = 1
WHERE Subjects.SubjectName = 'Drawing'
```

Amint látható, így elkerültük az allekérdezés használatát, és ez a forma bizonyos szempontból érthetőbb is. A fenti formát arra is felhasználhatjuk, hogy a módosított érték kiszámításakor a SET záradékban alkalmazott allekérdezés helyett egyszerűen felhasználjuk a kapcsolódó táblában szereplő értékeket is. Győződjünk meg saját adatbázisrendszerünk dokumentációjában arról, hogy támogatja-e ezt a megoldást. A mintaadatbázisok Microsoft Access számára készült változatában ezt a módszert alkalmaztuk több példa megoldására.

Ha már itt tartunk: az SQL-szabvány megengedi, hogy az UPDATE utasításban szereplő tábla egy nézettábla legyen, ebből pedig következik a táblák összekapcsolásának lehetősége. A szabvány emellett azonban azt is tartalmazza, hogy a nézettáblák módosíthatóságára vonatkozó megszorításokat az adott megvalósítás határozhatja meg. Ez a kikötés lehetővé teszi az adatbázis-kezelők gyártóinak, hogy minden esetben egy egyszerű táblanevet tegyenek kötelezővé, vagy más módon korlátozzák, hogy milyen nézetet vagy táblakapcsolatokat engednek meg. Mint ilyen esetben mindig, ennek is nézzünk utána saját adatbázisrendszerünk dokumentációjában.

Az allekérdezés bármilyen bonyolult lehet; amilyenre csak szükség van a módosítandó sorok kiszűréséhez. Ha például az egyik egyetemi tanár által tartott összes előadás kezdő időpontját akarjuk megváltoztatni, akkor az allekérdezésbe a Faculty_Classes (Tanszéki órák) és a Staff (Tanárok) táblákat is be kell vonnunk. A résztvevő táblákat a 15.6. ábra mutatja.



15.6. ábra

A tanárok nevei és az órák összefüggéséhez szükséges táblák

Mondjuk, állítsuk be délután két órai kezdésre Kathryn Patterson minden előadását. (A valóságban ilyen módosítást valószínűleg nem végzünk, mert ennek eredményeképp lehet, hogy több előadás fog ugyanabban az időpontban kezdődni, de példának így is épp elég érdekes.) A megoldás valami ilyesmi lehet:

„Módosítsd az órák tábláját úgy, hogy Kathryn Patterson minden órájának kezdete du. 2 óra legyen!"

Fordítás Update the classes table by setting the start time to 2:00 P.M. for all classes whose class ID is in the selection of class IDs of faculty classes joined with staff on staff ID in the faculty classes table matches staff ID in the staff table where the staff first name is 'Kathryn' and the staff last name is 'Patterson' (Módosítsd az órák tábláját: állítsd be a kezdetet du. 2. órára minden olyan óra esetében, amelynek az óraazonosítóját tartalmazza azoknak az óraazonosítóknak a listája, amelyet a tanszéki órák és a tanárok tábláinak olyan összekapcsolásából kapunk, ahol a tanszéki óra tanárazonosítója egyenlő a tanár tanárazonosítójával, és ahol a tanár keresztneve Kathryn, vezetékneve pedig Patterson.)

Tisztázás Update ~~the~~ classes ~~table by~~ ~~setting the~~ start time ~~to~~ = ~~2:00 P.M.~~ '14:00:00' ~~for all classes whose~~ where class ID ~~is~~ in ~~the~~ (select~~ion of~~ class ID~~s of~~ from faculty classes inner join~~ed with~~ staff on faculty_classes.staff ID ~~in the faculty classes table matches~~ = staff.staff ID ~~in the staff table~~ where ~~the~~ staff first name ~~is~~ = 'Kathryn' and ~~the~~ staff last name ~~is~~ = 'Patterson')

SQL

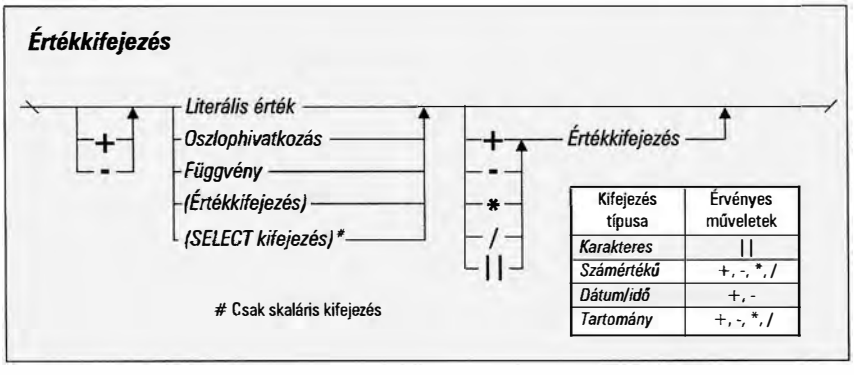
```
UPDATE Classes
SET StartTime = '14:00:00'
WHERE ClassID IN
   (SELECT ClassID
   FROM Faculty_Classes
   INNER JOIN Staff
   ON Faculty_Classes.StaffID = Staff.StaffID
   WHERE StfFirstName = 'Kathryn'
   AND StfLastName = 'Patterson')
```

Tehát a trükk csupán annyi, hogy azonosítjuk és felsoroljuk a módosítandó tábla és minden más, a feladat megoldásához szükséges kapcsolódó tábla közötti összefüggéseket.

Ugyanezt tettük a 8. fejezetben (INNER JOIN) és a 9. fejezetben (OUTER JOIN), amikor a több táblából történő lekérdezések FROM záradékát állítottuk így össze. Az UPDATE utasítás felépítésekor csak a módosítandó tábla kerülhet az UPDATE kulcsszó után, így a többi táblát csak egy olyan allekérdezésbe tehetjük, amelynek az eredményét össze tudjuk kapcsolni a módosítandó táblával.

Allekérdezés használata az UPDATE kifejezésben

Ne gondoljuk azonban, hogy ezzel vége is az allekérdezések alkalmazásának. A 15.1. ábrán megfigyelhetjük, hogy a SET záradékban az értékadás jobb oldalán egy értékkifejezés szerepel. A 15.7. ábra emlékeztetőül bemutatja, hogy miként épül fel egy értékkifejezés.



15.7. ábra

Az értékkifejezések szintaxisdiagramja

A 2. fejezetben azt javasoltuk, hogy számított értékeket ne tároljunk a táblákban. Mint ahogyan a legtöbb szabály esetében, itt is vannak kivételek. Vegyük például az Orders (Rendelések) táblát a Sales Orders mintaadatbázisból. Ha cégünk rendkívül nagy méretű, és több ezer részletező soros megrendeléseket kezel, akkor érdemes megfontolnunk egy „összegzett rendelésérték" oszlop felvételét az Orders táblába. Ennek a számított oszlopnak a használatával elkerülhetjük, hogy az összes rendelés meghatározásához be kelljen olvasni a részletező sorok ezreit. Ha ezt a módszert választjuk, akkor az alkalmazásunknak tartalmaznia kell olyan kódot, amely a részletező sorok módosításakor a számított rendelésérték-oszlopot naprakészen tartja. Mindeddig az értékadáshoz a SET záradékban vagy literális értéket használtunk, vagy egy egyszerű értékkifejezést, amely egy literális értékből, egy műveleti jelből és egy oszlopból állt. Vegyük észre, hogy ilyen módon az adott tábla egy másik oszlopának az értékét is felhasználtuk volna, de ez csak ritkán fordul elő. Érdekesebb eset az, amikor egy olyan SELECT kifejezést (egy allekérdezés eredményét) használjuk az értékadásban, amely egy másik tábla adataiból csak egy értéket ad vissza, például egy összeget. Az allekérdezés WHERE záradékában a sorok szűkítésekor használhatók a módosítandó tábla oszlopai is.

Így tehát, ha egy táblában (Orders) egy összegző oszlopot egy kapcsolódó tábla (Order_Details) valamely oszlopának az összegével akarjuk kitölteni, egy allekérdezést alkalmazó UPDATE utasítást kell összeállítanunk. Az allekérdezésben összegezzük a megrendelt mennyiség és az ajánlati ár szorzatát, majd ezt a számított oszlopban tároljuk.

Azt pedig, hogy az Order_Details (Rendelések részletei) táblából mindig csak az Orders táblához megfelelően kapcsolódó sorokat vegyük figyelembe, WHERE záradékkal biztosítjuk. A kérelem az alábbi lehet:

„Módosítsd a rendelések tábláját úgy, hogy a »rendelés összesen« mező egyenlő legyen az Order_Details tábla kapcsolódó sorai alapján számolt rendelt mennyiség és az ajánlott ár szorzatával!"

Fordítás Update the orders table by setting the order total to the sum of quantity ordered times quoted price from the order details table where the order number matches the order number in the orders table
(Módosítsd a rendelések tábláját úgy, hogy a „rendelés összesen" mező a rendelt mennyiség és az ajánlott ár szorzata legyen a rendelések részleteit tartalmazó tábla azon soraiban, ahol a rendelésszám megegyezik a rendelések táblájában levő rendelésszámmal.)

Tisztázás Update ~~the~~ orders ~~table~~
~~by~~ setting ~~the~~ order total ~~to~~ =
~~the~~ (select sum ~~of~~ (quantity ordered ~~times~~ * quoted price)
from ~~the~~ order details ~~table~~
where ~~the~~ order_details.order number ~~matches the~~ =
orders.order number ~~in the orders table~~)

SQL
```
UPDATE Orders
SET OrderTotal =
   (SELECT SUM(QuantityOrdered * QuotedPrice)
   FROM Order_Details
   WHERE Order_Details.OrderNumber =
   Orders.OrderNumber)
```

Megjegyzés *Ezt a lekérdezést a Sales Order Modify mintaadatbázisban CH15_Update_Order_Totals_Subquery néven találhatjuk meg.*

Vegyük észre, hogy nem alkalmaztunk WHERE záradékot a rendelések szűrésére. Ha egy alkalmazásban akarjuk használni a fenti kódrészletet, akkor valószínűleg érdemes szűkíteni a rendelésszám alapján, hogy az adatbázis csak azt a rendelést módosítsa, amelyikről tudjuk, hogy megváltozott. Egyes adatbázisrendszerek lehetővé teszik, hogy ehhez hasonló számított mezőket hozzunk létre, és meghatározzuk, miként kell az értéküket módosítani. A legtöbb adatbázisrendszer támogatja az úgynevezett *kioldókat* (trigger), amelyeket az adatbázisrendszer automatikusan futtat minden olyan esetben, amikor az adott táblába adatok kerülnek, vagy az ott lévő adatok módosulnak, illetve törlődnek. Azokban a rendszerekben, amelyek támogatják ezeket a szolgáltatásokat, vagy maga a tábla, vagy pedig az adatmódosításkor lefutó kioldó meghatározásában helyezhetjük el a fenti UPDATE utasítást. Mint mindig, a részleteknek nézzünk utána saját adatbázisrendszerünk dokumentációjában.

Az UPDATE használati területei

Ezen a ponton már tudjuk, hogy miként kell egy vagy több sort módosítani egy egyszerű literális értékkel vagy egy allekérdezés eredményeként előálló kifejezéssel. Azt is tudjuk, hogyan kell szűkíteni a módosítandó sorok körét. A legjobban úgy ismerhetjük meg az UPDATE utasítás széles alkalmazási területét, ha felsorolunk néhány olyan problémát, amelyet az UPDATE utasítással oldhatunk meg, illetve megoldunk egy sor példát is:

„Csökkentsd az ajánlott árat 2%-kal minden olyan rendelés esetében, amelyet több mint 30 nappal a megrendelés dátuma után szállítottak ki!"
„Adj 6%-ot minden ügynöki fizetéshez!"
„Cseréld le az összes Sports World Lanes-re tervezett torna helyszínét Oasis Lanes-re!"
„Számold újra minden hallgató tanulmányi átlagát a teljesített órák alapján!"
„Adj 5% kedvezményt minden olyan rendelésre, ahol a vásárló több mint 50 000 dollár értékben vásárolt szeptemberben!"
„Módosítsd a rendezvények szerződési összegét úgy, hogy az előadó napidíját megszorzod a napok számával, és hozzáadsz 15% jutalékot!"
„Frissítsd minden tekejátékos városát és államát az irányítószámot név alapján kikeresve!"
„Módosítsd a területkódot 360-ra minden olyan hallgató és tanár esetében, akinek az irányítószáma 98270 vagy 98271!"
„Biztosítsd, hogy minden kerékpár eladási ára legalább 45% nyereséget tartalmazzon a legalacsonyabb fogyasztói árat alkalmazó beszállító árához képest!"
„Adj 2% kedvezményt minden rendezvényre azoknak a megrendelőknek, akikkel több mint 3000 dollár értékű szerződést kötöttünk 2007 októberében!"
„Módosítsd a Huckleberrys tekecsapat nevét Manta Rays-re!"
„Növeld meg a teljes munkaidős tanárok bérét 5%-kal!"
„A kiegészítők fogyasztói árát állítsd a legmagasabb beszállítói ár plusz 35%-ra!"
„Növeld meg 0,5%-kal minden olyan ügynök részesedését, aki több mint 20 000 dollárnyi üzletet kötött!"
„Számítsd ki újra minden tekejátékosra az összes leütött bábut, a lejátszott mérkőzések számát, az aktuális átlagot, és az aktuális büntetőpontszámot!"

Példák

Ismerjük már az UPDATE utasítások felépítésének módszereit, nézzünk hát néhány példát, amelyben egy-egy tábla egy vagy több oszlopát kell valahogyan módosítanunk! A példákat négy mintaadatbázisból vettük.

A példák mellett közöljük a módosuló tábla adatait az UPDATE lefuttatása előtt és után, illetve a módosított sorok számát is. A módosítás utáni adatokat tartalmazó táblázat címében annak a lekérdezésnek a neve szerepel, amellyel azt a mellékletként adott CD-n szereplő mintaadatbázisokban megtalálhatjuk. Minden lekérdezést a megfelelő adatbázisba

tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH15"-tel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne feledjük, hogy a példákban használt minden tábla- és oszlopnév megfelel a mintaadatbázisok B függelékben található sémaleírásának. A leírások egyszerűsítése céljából a Fordítás és Tisztázás részeket minden példánál összevontuk. A példák feltételezik, hogy az olvasó áttanulmányozta és megértette a korábbi fejezetekben leírt ismereteket, különös tekintettel az allekérdezések témájára.*

Sales Orders adatbázis

„Csökkentsd az ajánlott árat 2%-kal minden olyan rendelés esetében, amelyet több mint 30 nappal a megrendelés dátuma után szállítottak ki!"

Fogalmazzuk újra a problémát úgy, hogy közelebb álljon az SQL nyelvtanához:

„Módosítsd a rendelések részleteit úgy, hogy az ajánlati ár egyenlő legyen az ajánlati ár szorozva 0,98-dal minden olyan rendelés esetében, ahol a kiszállítási dátum több mint 30 nappal későbbi, mint a megrendelési dátum!"

Fordítás/ Update ~~the~~ order details ~~table by~~
Tisztázás ~~setting the~~ quoted price ~~equal to~~ = ~~the~~ quoted price ~~times~~ * 0.98
where ~~the~~ order ID ~~is~~ in
~~the~~ (select~~ion of~~ order ID~~s~~
from ~~the~~ orders ~~table~~
where ship date ~~minus~~ – order date ~~is greater than~~ > 30

SQL
```
UPDATE Order_Details
SET QuotedPrice = QuotedPrice * 0.98
WHERE OrderID IN
   (SELECT OrderID
   FROM Orders
   WHERE (ShipDate - OrderDate) > 30)
```

Megjegyzés *A fenti megoldás feltételezi, hogy adatbázisrendszerünk képes két dátum közötti napok számát meghatározni úgy, hogy az egyik dátumot kivonja a másikból. Nézzünk utána adatbázisrendszerünk dokumentációjában, hogy működik-e ez a módszer.*

Az Order_Details tábla az UPDATE utasítás végrehajtása előtt

OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
291	1	$1,200.00	4
291	14	$139.95	2
291	30	$43.65	6
371	9	$32.01	6
371	22	$79.54	5
371	35	$37.83	6
387	1	$1,200.00	4
387	6	$635.00	4
<< további sorok >>			

Az Order_Details tábla a CH15_Adjust_Late_Order_Prices végrehajtása után (29 sor módosult)

OrderNumber	ProductNumber	QuotedPrice	QuantityOrdered
291	1	$1,176.00	4
291	14	$137.15	2
291	30	$42.78	6
371	9	$31.37	6
371	22	$77.95	5
371	35	$37.07	6
387	1	$1,176.00	4
387	6	$622.30	4
<< további sorok >>			

„Biztosítsd, hogy minden kerékpár fogyasztói ára legalább 45% nyereséget tartalmazzon a legalacsonyabb fogyasztói árat alkalmazó beszállító árához képest!"

A kérelem átfogalmazva az alábbi lesz:

„Módosítsd a termékek tábláját úgy, hogy a fogyasztói ár egyenlő legyen 1,45 szorozva annak a beszállítónak a fogyasztói árával, amelyiknél a legalacsonyabb a termék ára, ha a fogyasztói ár még nem egyenlő 1,45 szorozva a fogyasztói árral, és a termékcsoport a 2-es!"

Fordítás/ Update ~~the~~ products ~~table by~~
Tisztázás ~~setting the~~ retail price ~~equal to~~ = 1.45 ~~times~~ *
~~the~~ (~~selection of the unique~~ distinct wholesale price
from ~~the~~ product vendors ~~table~~
where ~~the~~ product vendors ~~table's~~ product number
~~in equal to~~ = ~~the~~ products ~~table's~~ product number

```
        and the wholesale price is-equal-to =
        the (selection-of the minimum (wholesale price)
        from the product vendors table
        where the product vendors table's product number
        is-equal-to = the products table's product number))
        where the retail price is-less-than < 1.45 times
        the (selection-of the-unique distinct wholesale price
        from the product vendors table
        where the product vendors table's product number
        is-equal-to = the products table's product number
        and the wholesale price is-equal-to =
        the (selection-of the minimum (wholesale price)
        from the product vendors table
        where the product vendors table's product number
        is-equal-to = the products table's product number))
        and the category ID is-equal-to = 2
```

SQL

```
UPDATE Products
SET RetailPrice = ROUND(1.45 *
   (SELECT DISTINCT WholeSalePrice
    FROM Product_Vendors
    WHERE Product_Vendors.ProductNumber
    = Products.ProductNumber
    AND WholeSalePrice =
      (SELECT MIN(WholeSalePrice)
       FROM Product_Vendors
       WHERE Product_Vendors.ProductNumber
       = Products.ProductNumber)), 0)
       WHERE RetailPrice < 1.45 *
          (SELECT DISTINCT WholeSalePrice
           FROM Product_Vendors
           WHERE Product_Vendors.ProductNumber
              = Products.ProductNumber
              AND WholeSalePrice =
                 (SELECT MIN(WholeSalePrice)
                  FROM Product_Vendors
                  WHERE Product_Vendors.ProductNumber
                  = Products.ProductNumber))
           AND CategoryID = 2
```

Megjegyzés *A Microsoft Access mintaadatbázisban ezt a feladatot az UPDATE záradékban alkalmazott JOIN kulcsszóval oldottuk meg, mert az Access nem támogatja az allekérdezések használatát a SET záradékban.*

Vegyük észre továbbá, hogy az eredményként kapott árat (0 tizedesjegyre) kerekítjük. A legtöbb kereskedelmi adatbázis-kezelő támogatja a ROUND függvényt, bár azt az SQL-szabvány nem tartalmazza kifejezetten.

A fentieken kívül beilleszthettünk volna még egy további allekérdezést, amely meghatározza a Bikes (Kerékpárok) termékcsoport-leíráshoz tartozó termékcsoport-azonosítót a Categories (Termékcsoportok) tábla segítségével, de úgy gondoltuk, hogy ez az utasítás így is elég bonyolult. Végül egyedi fogyasztói árat választottunk ki a DISTINCT kulcsszó segítségével, mert több beszállítónál is megtalálhatjuk ugyanazt a legalacsonyabb árat, de az összehasonlításhoz egyetlen értékre van szükségünk.

A Products tábla az UPDATE utasítás végrehajtása előtt

ProductNumber	ProductName	RetailPrice	QuantityOnHand	CategoryID
2	Eagle FS-3 Mountain Bike	$1,800.00	8	2

A Products tábla a CH15_Adjust_Bike_Retail_Price végrehajtása után (1 sor módosult)

ProductNumber	ProductName	RetailPrice	QuantityOnHand	CategoryID
2	Eagle FS-3 Mountain Bike	$1,840.00	8	2

Megjegyzés *Ha végignézzük a Products táblában a kerékpárokat (ProductID 1, 2, 6 vagy 11), akkor látjuk, hogy csak a 2-es termék fogyasztói ára kisebb, mint az adott termék bármely beszállító által kínált legalacsonyabb fogyasztói árának 1,45-szorosa. A 2-es kerékpár fogyasztói ára a 6-os beszállítónál 1269 dollár, amelyet 1,45-dal megszorozva 1840,05 dollárt kapunk. Ezt kerekíti az UPDATE utasítás egész számra.*

Entertainment Agency adatbázis

„Adj 6%-ot minden ügynök fizetéséhez!"

A kérelem átfogalmazva az alábbi lesz:

„Módosítsd az ügynökök tábláját, és adj hozzá minden fizetéshez 6%-ot!"

Fordítás/ Update ~~the~~ agents ~~table~~
Tisztázás ~~by~~ set~~ting~~ salary ~~equal to~~ = salary ~~times~~ * 1.06

SQL
```
UPDATE Agents
SET Salary = ROUND(Salary * 1.06, 0)
```

Megjegyzés *Itt ismét a legtöbb kereskedelmi adatbázis-megvalósításban megtalálható ROUND függvényt használtuk, és 0 tizedesjegyre kerekítettünk. A kerekítés megvalósításának részleteit ellenőrizzük saját adatbázisrendszerünk dokumentációjában.*

Az Agents tábla az UPDATE végrehajtása előtt

AgentID	AgtFirstName	AgtLastName	Salary
1	William	Thompson	$35,000.00
2	Scott	Bishop	$27,000.00
3	Carol	Viescas	$30,000.00
4	Karen	Smith	$22,000.00
5	Marianne	Wier	$24,500.00
6	John	Kennedy	$33,000.00
7	Caleb	Viescas	$22,100.00
8	Maria	Patterson	$30,000.00
9	Daffy	Dumbwit	$50.00

Az Agents tábla a CH15_Give_Agents_6Percent_Raise végrehajtása után (9 sor módosult)

AgentID	AgtFirstName	AgtLastName	Salary
1	William	Thompson	$37,100.00
2	Scott	Bishop	$28,620.00
3	Carol	Viescas	$31,800.00
4	Karen	Smith	$23,320.00
5	Marianne	Wier	$25,970.00
6	John	Kennedy	$34,980.00
7	Caleb	Viescas	$23,426.00
8	Maria	Patterson	$31,800.00
9	Daffy	Dumbwit	$53.00

„Módosítsd a rendezvények szerződési összegét úgy, hogy az előadó napidíját megszorzod a szerződött napok számával, és hozzáadsz 15% jutalékot!"

Fogalmazzuk át a kérelmet:

„Módosítsd a rendezvények tábláját úgy, hogy a szerződési összeg 1,15 szorozva a szerződött napok számával szorozva az előadó napidíjával legyen!"

Fordítás/ Tisztázás

Update ~~the~~ engagements ~~table~~
by setting the contract price equal to =
1.15 ~~times~~ * ~~the~~ (end date ~~minus~~ – ~~the~~ start date ~~plus~~ + 1)
~~and then times the~~ *
(selection ~~of the~~ entertainer price per day
from ~~the~~ entertainers ~~table~~
where ~~the~~ entertainers ~~table~~ entertainer ID ~~is equal to~~ =
~~the~~ engagements ~~table~~ entertainer ID

```
SQL     UPDATE Engagements
        SET Engagements.ContractPrice =
        ROUND(1.15 * (EndDate - StartDate + 1) *
           (Select EntPricePerDay
            FROM Entertainers
            WHERE Entertainers.EntertainerID =
            Engagements.EntertainerID), 0)
```

Megjegyzés *A fenti megoldás feltételezi, hogy adatbázisrendszerünk megengedi, hogy a két dátum közötti napok számát megkapjuk úgy, hogy az egyik dátumból kivonjuk a másikat. A részleteket nézzük meg a saját adatbázisunk dokumentációjában.*

A különbséghez hozzáadtunk egyet, hogy a rendezvény tényleges időtartamát megkapjuk, ugyanis a rendezvény első és utolsó napján is van előadás. Egynapos rendezvény esetén nyilvánvaló, hogy így kell eljárni: a kezdő és a végdátum azonos, tehát a különbségük nulla, de a rendezvény pontosan egy napos.

Előadói napidíjak

EntertainerID	EntStageName	EntPricePerDay
1001	Carol Peacock Trio	$175.00
1002	Topazz	$120.00
1003	JV & the Deep Six	$275.00
1004	Jim Glynn	$60.00
1005	Jazz Persuasion	$125.00
1006	Modern Dance	$250.00
1007	Coldwater Cattle Company	$275.00
1008	Country Feeling	$280.00
1009	Katherine Ehrlich	$145.00
1010	Saturday Revue	$250.00
1011	Julia Schnebly	$90.00
1012	Susan McLain	$75.00
1013	Caroline Coie Cuartet	$250.00

Az Engagements tábla az UPDATE utasítás végrehajtása előtt

EngagementNumber	StartDate	EndDate	ContractPrice	CustomerID	AgentID	EntertainerID
2	2007-09-01	2007-09-05	$200.00	10006	4	1004
3	2007-09-10	2007-09-15	$590.00	10001	3	1005
4	2007-09-11	2007-09-17	$470.00	10007	3	1004
5	2007-09-11	2007-09-14	$1,130.00	10006	5	1003
6	2007-09-10	2007-09-14	$2,300.00	10014	7	1008
7	2007-09-11	2007-09-18	$770.00	10004	4	1002
8	2007-09-18	2007-09-25	$1,850.00	10006	3	1007
9	2007-09-18	2007-09-28	$1,370.00	10010	2	1010
<< további sorok >>						

Az Engagements tábla a CH15_Calculate_Entertainment_ContractPrice végrehajtása után (111 sor módosult)

EngagementNumber	StartDate	EndDate	ContractPrice	CustomerID	AgentID	EntertainerID
2	2007-09-01	2007-09-05	$345.00	10006	4	1004
3	2007-09-10	2007-09-15	$862.00	10001	3	1005
4	2007-09-11	2007-09-17	$483.00	10007	3	1004
5	2007-09-11	2007-09-14	$1,265.00	10006	5	1003
6	2007-09-10	2007-09-14	$1,610.00	10014	7	1008
7	2007-09-11	2007-09-18	$1,104.00	10004	4	1002
8	2007-09-18	2007-09-25	$2,530.00	10006	3	1007
9	2007-09-18	2007-09-28	$3,162.00	10010	2	1010
<< további sorok >>						

Megjegyzés *Az Engagements (Rendezvények) táblában eredetileg szereplő szerződési összegek csupán ésszerű határok között véletlenszerűen választott értékek voltak. Az imént megírt UPDATE utasítás természetesen sokkal pontosabb összeget állapít meg az előadók napidíja alapján.*

School Scheduling adatbázis

„Módosítsd a területi kódot 360-ra minden olyan hallgató esetében, akinek az irányítószáma 98270 vagy 98271!"

Az újrafogalmazott kérelem az alábbi lesz:

„Módosítsd a hallgatók tábláját úgy, hogy a területi kód 360 legyen minden olyan hallgató esetében, akinek az irányítószáma 98270 vagy 98271!"

Fordítás/ Update ~~the~~ students ~~table~~

Tisztázás ~~by~~ ~~setting the~~ area code ~~equal to~~ = '360'

where ~~the~~ student ZIP Code ~~is~~ in ~~the list~~ ('98270', ~~and~~ '98271')

SQL

```
UPDATE Students
SET Students.StudAreaCode = '360'
WHERE Students.StudZipCode IN ('98270', '98271')
```

A Students tábla az UPDATE utasítás végrehajtása előtt

StudentID	StudFirstName	StudLastName	StudCity	StudState	StudZipCode	StudAreaCode
<< további sorok >>						
1007	Elizabeth	Hallmark	Marysville	WA	98271	253
1008	Sara	Sheskey	Portland	OR	97208	503
1009	Karen	Smith	Eugene	OR	97401	541
1010	Marianne	Wier	Tacoma	WA	98413	253
1011	John	Kennedy	Portland	OR	97208	503
1012	Sarah	Thompson	Lubbock	TX	79402	806
1013	Michael	Viescas	Redmond	WA	98052	425
1014	Kendra	Bonnicksen	Seattle	WA	98105	206
1015	Brannon	Jones	Long Beach	CA	90809	562
1016	Steve	Pundt	Dallas	TX	75204	972
1017	George	Chavez	Marysville	WA	98270	206
<< további sorok >>						

A Students tábla a CH15_Fix_Student_AreaCode végrehajtása után (2 sor módosult)

StudentID	StudFirstName	StudLastName	StudCity	StudState	StudZipCode	StudAreaCode
<< további sorok >>						
1007	Elizabeth	Hallmark	Marysville	WA	98271	360
1008	Sara	Sheskey	Portland	OR	97208	503
1009	Karen	Smith	Eugene	OR	97401	541
1010	Marianne	Wier	Tacoma	WA	98413	253
1011	John	Kennedy	Portland	OR	97208	503
1012	Sarah	Thompson	Lubbock	TX	79402	806
1013	Michael	Viescas	Redmond	WA	98052	425
1014	Kendra	Bonnicksen	Seattle	WA	98105	206
1015	Brannon	Jones	Long Beach	CA	90809	562
1016	Steve	Pundt	Dallas	TX	75204	972
1017	George	Chavez	Marysville	WA	98270	360
<< további sorok >>						

„Számold újra minden hallgató tanulmányi átlagát a teljesített órák alapján!"

Az átfogalmazott kérelem valami ilyesmi lesz:

„Módosítsd a hallgatók tábláját úgy, hogy a tanulmányi átlag egyenlő legyen a kreditérték és az osztályzat szorzatának összege elosztva a kreditérték összegével!"

Fordítás/ Tisztázás

Update ~~the~~ students ~~table~~
~~by~~ ~~setting the~~ student GPA ~~equal to~~ =
~~the~~ (~~selection of the~~ sum ~~of~~ (credits ~~times~~ * grade)
~~divided by~~ / ~~the~~ sum ~~of~~ (credits)
from ~~the~~ classes ~~table~~
inner join~~ed with the~~ student schedules ~~table~~
on classes.class ID ~~in the classes table matches~~
= student_schedules.class ID ~~in the student schedules table~~
where ~~the~~ class status ~~is~~ = ~~complete~~ 2
and ~~the~~ student schedules ~~table~~ student ID ~~is equal to~~ =
~~the~~ students ~~table~~ student ID)

SQL

```
UPDATE Students
SET Students.StudGPA =
   (SELECT ROUND(SUM(Classes.Credits *
      Student_Schedules.Grade) /
      SUM(Classes.Credits), 3)
    FROM Classes
    INNER JOIN Student_Schedules
    ON Classes.ClassID = Student_Schedules.ClassID
    WHERE (Student_Schedules.ClassStatus = 2)
    AND (Student_Schedules.StudentID =
      Students.StudentID))
```

A Students tábla az UPDATE utasítás végrehajtása előtt

StudentID	StudFirstName	StudLastName	StudGPA
1001	Kerry	Patterson	74.465
1002	David	Hamilton	78.755
1003	Betsy	Stadick	85.235
1004	Janice	Galvin	81
1005	Doris	Hartwig	72.225
1006	Scott	Bishop	72.225
1007	Elizabeth	Hallmark	87.65
1008	Sara	Sheskey	84.625
<< *további sorok* >>			

A Students tábla a CH15_Update_Student_GPA végrehajtása után (18 sor módosult)

StudentID	StudFirstName	StudLastName	StudGPA
1001	Kerry	Patterson	73.875
1002	David	Hamilton	79.346
1003	Betsy	Stadick	87.737
1004	Janice	Galvin	80.78
1005	Doris	Hartwig	73.222
1006	Scott	Bishop	87.603
1007	Elizabeth	Hallmark	89
1008	Sara	Sheskey	85.726
<< *további sorok* >>			

Megjegyzés *Mivel a Microsoft Access nem támogatja az összesítő függvényeket tartalmazó allekérdezések használatát, ott a megoldás egy sor beépített függvény meghívása egy a Student_Schedules és a Classes táblán alapuló előre létrehozott nézettáblán.*

Bowling League adatbázis

„Számítsd ki újra minden tekejátékos számára az összes leütött bábut, a lejátszott mérkőzések számát, az aktuális átlagot és az aktuális büntetőpontszámot!"

Megjegyzés *A 13. fejezet feladatai között szerepelt egy SELECT lekérdezés a büntetőpontszám kiszámítására. Segítséget a Bowling League mintaadatbázisban szereplő CH13_Bowler_Average_Handicap lekérdezésben találhatunk. A büntetőpontszám kiszámításának módja: 200-ból kivonjuk a tekejátékos átlageredményét, és ennek vesszük a 90%-át.*

Fogalmazzuk újra a kérelmet így:

„Módosítsd a tekejátékosok tábláját úgy, hogy kiszámítod az összes leütött bábut, a lejátszott mérkőzések számát, az aktuális átlagot és az aktuális büntetőpontszámot a játékosok pontszámait tartalmazó tábla alapján!"

Fordítás/ Update ~~the~~ bowlers ~~table~~
Tisztázás ~~by~~ setting ~~the~~ total pins ~~equal to~~ =
~~the~~ (selection ~~of the~~ sum ~~of the~~ (raw score)
from ~~the~~ bowler scores ~~table~~
where ~~the~~ bowler scores ~~table~~ bowler ID
~~is equal to~~ = ~~the~~ bowlers ~~table~~ bowler ID),
~~and the~~ games bowled ~~equal to~~ =
~~the~~ (selection ~~of the~~ count ~~of the~~ (raw score)

```
from the bowler scores table
where the bowler scores table bowler ID
is-equal-to = the bowlers table bowler ID),
and-the current average equal-to =
the (selection-of-the-average avg of the (raw score)
from the bowler scores table
where the bowler scores table bowler ID
is-equal-to = the bowlers table bowler ID),
and-the current handicap equal-to =
the (selection of 0.9 times * (200 minus –
the-average avg of the (raw score))
from the bowler scores table
where the bowler scores table bowler ID
is-equal-to = the bowlers table bowler ID)
```

SQL

```
UPDATE Bowlers
SET Bowlers.BowlerTotalPins =
   (SELECT SUM(RawScore)
    FROM Bowler_Scores
    WHERE Bowler_Scores.BowlerID = Bowlers.BowlerID),
   Bowlers.BowlerGamesBowled =
   (SELECT COUNT(Bowler_Scores.RawScore)
    FROM Bowler_Scores
    WHERE Bowler_Scores.BowlerID = Bowlers.BowlerID),
    Bowlers.BowlerCurrentAverage =
       (SELECT ROUND(AVG(Bowler_Scores.RawScore), 0)
        FROM Bowler_Scores
        WHERE Bowler_Scores.BowlerID =
          Bowlers.BowlerID),
```

A Bowlers tábla az UPDATE parancs lefuttatása előtt

BowlerID	BowlerLast	NameBowler First	NameBowler TotalPins	BowlerGames Bowled	BowlerCurrent Average	BowlerCurrent Hcp
1	Fournier	Barbara	5790	39	148	47
2	Fournier	David	6152	39	158	38
3	Kennedy	John	6435	39	165	32
4	Sheskey	Sara	5534	39	142	52
5	Patterson	Ann	5819	39	149	46
6	Patterson	Neil	6150	39	158	38
7	Viescas	David	6607	39	169	28
8	Viescas	Stephanie	5558	39	143	51
<< további sorok >>						

```
Bowlers.BowlerCurrentHcp =
(SELECT ROUND(0.9 *
   (200 - ROUND(AVG(Bowler_Scores.RawScore),
   0)), 0)
FROM Bowler_Scores
WHERE Bowler_Scores.BowlerID = Bowlers.BowlerID)
```

A Bowlers tábla az UPDATE parancs lefuttatása előtt

BowlerID	BowlerLast	NameBowler First	NameBowler TotalPins	BowlerGames Bowled	BowlerCurrent Average	BowlerCurrent Hcp
1	Fournier	Barbara	5790	39	148	47
2	Fournier	David	6152	39	158	38
3	Kennedy	John	6435	39	165	32
4	Sheskey	Sara	5534	39	142	52
5	Patterson	Ann	5819	39	149	46
6	Patterson	Neil	6150	39	158	38
7	Viescas	David	6607	39	169	28
8	Viescas	Stephanie	5558	39	143	51
<< további sorok >>						

„Cseréld le az összes Sports World Lanes-re tervezett torna helyszínét Oasis Lanes-re!"

Fogalmazzuk át a kérelmet így:

„Módosítsd a tornák tábláját úgy, hogy a torna helyszíne Oasis Lane legyen minden olyan torna esetében, amelyet eredetileg a Sports World Lanes-re terveztek!"

Fordítás/ Tisztázás
Update ~~the~~ tournaments ~~table~~
~~by~~ ~~setting the~~ tourney location ~~equal to~~ = 'Oasis Lanes'
where ~~the original~~ tourney location
~~is equal to~~ = 'Sports World Lanes'

SQL

```
UPDATE Tournaments
SET TourneyLocation = 'Oasis Lanes'
WHERE TourneyLocation = 'Sports World Lanes'
```

A Tournaments tábla az UPDATE utasítás végrehajtása előtt

TourneyID	TourneyDate	TourneyLocation
5	2007-10-02	Sports World Lanes
12	2007-11-20	Sports World Lanes
18	2008-08-01	Sports World Lanes

A Tournaments tábla a CH15_Change_Tourney_Location végrehajtása után (3 sor módosul)

TourneyID	TourneyDate	TourneyLocation
5	2007-10-02	Oasis Lanes
12	2007-11-20	Oasis Lanes
18	2008-08-01	Oasis Lanes

Összefoglalás

A fejezet elején röviden bemutattuk az UPDATE utasítást, amely nem az adatok lekérdezésére, hanem azok módosítására alkalmas. Megismerkedtünk az UPDATE utasításformájával, és láttunk egy egyszerű példát, amelyben egy tábla összes sorának egy oszlopát módosítottuk egy kifejezés felhasználásával.

Ezt követően egy újabb példán keresztül bemutattuk, hogy miként használható a WHERE záradék a módosítandó sorok körének leszűkítésére. Ez után egy egyszerű SELECT utasítást írtunk annak ellenőrzésére, hogy valóban a megfelelő sorokat módosítjuk-e, majd a SELECT utasítást átformáltuk a megfelelő UPDATE utasítássá. Ez után elmagyaráztuk, miért is fontosak a tranzakciók, és hogyan lehet őket a hibák elleni védekezésre felhasználni, valamint biztosítani, hogy vagy az összes módosítás végrehajtódjon, vagy egy sem. Folytatásként bemutattuk, hogyan lehet egy tábla több oszlopát módosítani egyetlen UPDATE utasítással.

Ez után beléptünk az allekérdezések birodalmába. Kezdésként a WHERE záradékban helyeztünk el bonyolultabb szűrőfeltételeket. Később azt mutattuk be, hogy a SET záradékban hogyan lehet allekérdezések eredményét értékadásra felhasználni. A fejezet fennmaradó részét az UPDATE utasítások felépítését szemléltető példáknak szenteltük.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Adj 5% kedvezményt minden olyan rendelésre, amelynél a vásárló több mint 50 000 dollár értékben vásárolt 2007 októberében!"*
   (Tipp: két egymásba ágyazott allekérdezésre lesz szükségünk, amelyben előkeressük az összes olyan rendelésszámot, amelynek a vásárlóazonosítója – a CustomerID oszlop – benne van azon vásárlók azonosítóinak a listájában, akik több mint 50 000 dollár értékben rendeltek októberben.)
   A megoldás itt található: CH15_Give_Discount_To_Good_October_Customers (650 sor módosul). Futtassuk a CH15_Update_Order_Totals-ban található UPDATE utasítást is, hogy az Orders táblában is helyesek legyenek az adataink.
2. *„A kiegészítők (1-es termékcsoport) fogyasztói árát állítsd a legmagasabb beszállítói ár plusz 35%-ra!"*
   (Tipp: a példák között a CH15_Adjust_Bike_Retail_Price-ban használtunk olyan módszert, amilyenre itt szükségünk van.)
   A megoldás itt található: CH15_Adjust_Accessory_Retail_Price (11 sor módosul).

Entertainment Agency adatbázis

1. *„Adj 2% kedvezményt minden rendezvényre azoknak a megrendelőknek, akik több mint 3000 dollár értékű rendezvényt kötöttek le 2007 októberében!"*
   (Tipp: egy összesítő allekérdezésre van szükségünk, amely egy HAVING záradékkal találja meg azokat a megrendelőket, akik több mint 3000 dollárnyi értékű rendezvényt kötöttek le 2007 októberére.)
   A megoldás itt található: CH15_Discount_Good_Customers_October (34 sor módosul).
2. *„Növeld meg 0,5%-kal minden olyan ügynök részesedését, aki több mint 20 000 dollárnyi üzletet kötött!"*
   (Tipp: használjunk összesítő allekérdezést, amely egy HAVING záradékkal találja meg azokat az ügynököket, akik több mint 20 000 dollárnyi értékű üzletet kötöttek.)
   A megoldás itt található: CH15_Reward_Good_Agents (3 sor módosul).

School Scheduling adatbázis

1. *„Növeld meg a teljes munkaidőben alkalmazott tanárok bérét 5%-kal!"*
   (Tipp: a WHERE záradékban használjunk egy olyan allekérdezést, amely kikeresi azoknak a tanároknak a tanárazonosítóját (StaffID) a tanszék (Faculty) táblájából, akiknek az állapota (status) teljes munkaidős (full time), és az alkalmazott (tenured) mezőjük igaz értékű. Ez utóbbi az adatbázisrendszerünktől függően 1 vagy –1 lehet.)
   A megoldás itt található: CH15_Give_FullTime_Tenured_Raise (21 sor módosul).
2. *„Módosítsd a területi kódot 360-ra minden olyan tanár esetében, akinek az irányítószáma 98270 vagy 98271!"*
   A megoldás itt található: CH15_Fix_Staff_AreaCode (2 sor módosul).

Bowling League adatbázis

1. *„Módosítsd a Huckleberrys tekecsapat nevét Manta Rays-re!"*
   A megoldás itt található: CH15_Change_Huckleberry_Name (1 sor módosul).
2. *„Az irányítószám alapján módosítsd a várost és az államot minden játékos esetében!"*
   (Tipp: használjunk két allekérdezést – egyet, amelyik kikeresi a várost, és egy másikat, amelyik kikeresi az államot a WAZips táblából.)
   A megoldás itt található: CH15_Update_Bowler_City_State (6 sor módosul).

16

Adathalmazok beszúrása

„Abban a meggyőződésben nőttem fel, hogy az egyetlen dolog, amellyel érdemes foglalkozni: új részletekkel gyarapítani a világban létező pontos ismereteket."
– Margaret Mead

A fejezet témakörei

- Mi az az INSERT?
- Az INSERT utasítás
- Az INSERT használati területei
- Példák
- Összefoglalás
- Önálló feladatok

Eddig arról volt szó, hogy miként kérdezzük le mindenféle kreatív módon a táblákban levő adatokat; az előző fejezetben pedig megtanultuk az adatokat az UPDATE utasítással módosítani. De hogyan kerülnek be az adatok a táblákba? Az biztos, hogy nem varázsütésre pattannak elő a semmiből! Ebben a fejezetben erre a kérdésre adjuk meg a választ: hogyan használhatjuk az INSERT utasítást adatok bevitelére?

Mi az az INSERT?

A legtöbb kereskedelmi adatbázisrendszerhez tartozik valamiféle grafikus alkalmazás, amely megjeleníti az adatokat, és lehetővé teszi velük a munkát. A Microsoft Access-szel például egyszerűen megkereshetjük a táblaobjektumot, és megnyithatjuk. Az Access egy adatlapnak nevezett sorokból és oszlopokból álló táblázatban jeleníti meg az adatainkat. Ha a táblázat aljára görgetünk, akkor ott egy üres sorba egyszerűen elkezdhetjük beírni az adatainkat. Ha végeztünk egy sorral, akkor a következő sorra lépve újabb rekordot adhatunk a táblához. Az Access arra is alkalmas, hogy Microsoft SQL táblákat kezeljen hasonló módon. A MySQL Query Browsere (Lekérdezéstallózó) is hasonlóan működik, illetve az IBM DB2-jéhez és az Oracle adatbázisokhoz is léteznek ilyen eszközök.

De mi is történik, amikor az adatok begépelésének végeztével mentjük a módosításokat? A grafikus alkalmazás valójában SQL-utasításokkal adja hozzá a beírt adatokat a táblához. A parancs, amelyet ilyenkor használ, az INSERT (beszúrás). A mintafájlok között böngészve megtalálhatjuk azokat a parancsfájlokat, amelyeket a mintaadatbázisok adatainak feltöltéséhez használtunk. Például az EntertainmentAgencyData.SQL fájl első néhány sora az alábbi:

```
USE EntertainmentAgencyExample;
INSERT INTO Customers
  VALUES (10001, 'Doris', 'Hartwig', '4726 - 11th Ave. N.E.',
   'Seattle', 'WA', '98105', '555-2671');
INSERT INTO Customers
  VALUES (10002, 'Deb', 'Waldal', '908 W. Capital Way',
   'Tacoma', 'WA', '98413', '555-2496');
INSERT INTO Customers
  VALUES (10003, 'Peter', 'Brehm', '722 Moss Bay Blvd.',
   'Kirkland', 'WA', '98033', '555-2501');
INSERT INTO Customers
  VALUES (10004, 'Dean', 'McCrae', '4110 Old Redmond Rd.',
   'Redmond', 'WA', '98052', '555-2506');
INSERT INTO Customers
  VALUES (10005, 'Elizabeth', 'Hallmark', 'Route 2, Box 203B',
   'Auburn', 'WA', '98002', '555-2521');
```

Az első parancs (USE) megmondja az adatbázis-kezelőnek, hogy melyik adatbázison hajtsa végre az utána következő parancsokat. Minden INSERT parancs arra utasítja az adatbázist, hogy egyetlen sort szúrjon be egy adott táblába. A több ezer sor felvitele egy mintaadatbázisba látszólag munkaigényes feladat, de ha elindítjuk a parancsfájlokat, akkor azt tapasztaljuk, hogy mindegyik lefut néhány másodperc alatt. Néhány egyszerűbb tábla esetében mi is a grafikus felületeket használtuk az adatok közvetlen begépelésére; más táblákhoz azonban programokat írtunk, amelyek előállították és végrehajtották az INSERT utasításokat. Aki otthonosan mozog a Microsoft Accessben és a Visual Basicben, megtalálhatja a Sales Orders mintaadatbázis adatait előállító programot a zfrmSellProducts űrlapban.

Bármilyen alkalmazást írunk – legyen az hagyományos vagy webalkalmazás –, a kódunk biztosan fog olyan részletet tartalmazni, amely INSERT utasításokat állít össze és futtat, amikor a felhasználó új adatokat ír be. A legtöbb esetben az INSERT ... VALUES változatot fogjuk használni. Később egy másik formát is bemutatunk, amellyel egyszerűen másolhatunk adatokat egyik táblából a másikba.

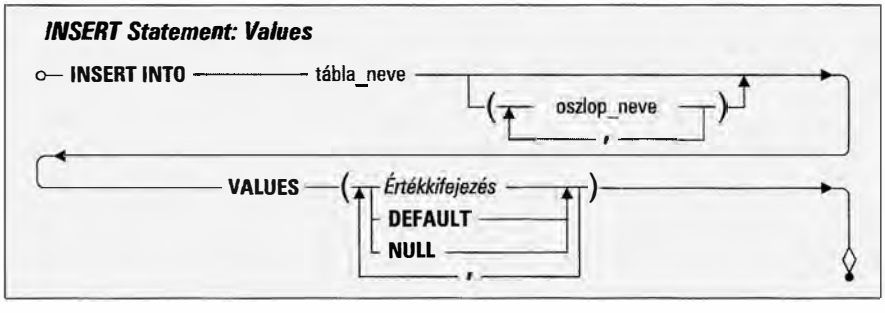
Megjegyzés *Az összes példát és a feladatok megoldását megtalálhatjuk a megfelelő mintaadatbázisok módosítható („modify") változatában (SalesOrdersModify, EntertainmentAgencyModify, SchoolSchedulingModify, BowlingLeagueModify).*

Az INSERT utasítás

Az SQL-ben az INSERT utasításnak két fő formája van. Az első változatban a VALUES kulcsszó után felsoroljuk az adott táblába új sorként beszúrandó értékeket. A második változatban egy SELECT záradékot használva olvassuk be azokat az adatokat, amelyeket a módosítandó táblába írunk. Lássuk először a VALUES kulcsszót alkalmazó változatot!

Értékek beszúrása

Bár az SQL-t elsődlegesen úgy tervezték, hogy adathalmazokon dolgozzon, az INSERT utasítást mégis arra használjuk a leggyakrabban, hogy egyszerre egyetlen sort szúrjunk be egy táblába. Egy táblához új sort hozzáadni legegyszerűbben a VALUES kulcsszót tartalmazó INSERT utasítással lehet. Ennek a formának a szintaxisdiagramját a 16.1. ábra mutatja.



16.1. ábra

Az INSERT utasítás VALUES kulcsszót tartalmazó változatának szintaxisdiagramja

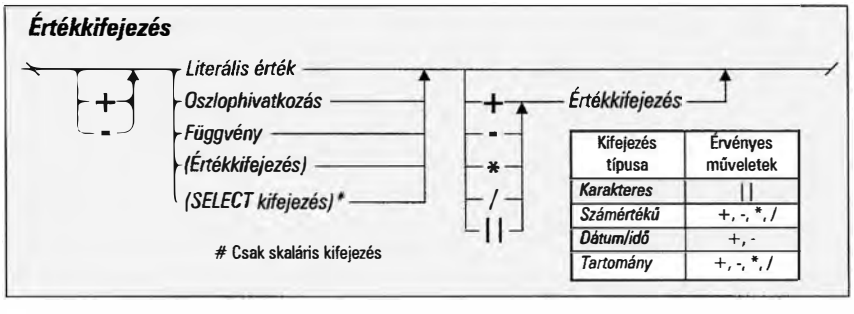
Látható, hogy az utasítás az INSERT INTO kulcsszavakkal kezdődik. Ez után következik annak a táblának a neve, amelyhez az adatokat hozzá akarjuk adni. Ha minden oszlopnak értéket akarunk adni, akkor az oszlopnevek listáját elhagyhatjuk. (Mi például nem írtuk ki az oszlopnevek listáját a mintaadatbázisok adatfeltöltését végző INSERT utasításokban, mert minden oszlop értékét megadtuk.) Ennek ellenére azt javasoljuk, hogy akkor is írjuk ki az oszlopok neveit, ha mindegyikbe akarunk adatot írni. Ha nem így járunk el, akkor később, ha valaki új oszlopot ad a táblához, vagy módosítja az oszlopok sorrendjét, a mi utasításunk végrehajtása meghiúsul. Az oszlopnévlista egy nyitó zárójellel kezdődik, amelyet az oszlopnevek vesszővel elválasztott listája követ (amennyiben egynél több oszlopot adunk meg), majd egy záró zárójellel végződik.

Megjegyzés *Az SQL-szabvány szerint a tábla_neve nézettáblát is takarhat, de kiköti, hogy a nézettáblának lehetővé kell tennie a módosítást és a beszúrást. Számos adatbázisrendszer lehetővé teszi, hogy nézettáblákba szúrjunk be adatokat, de mindegyiknek saját szabályai vannak, amelyek alapján meghatározzák, hogy egy nézettábla módosítható-e, illetve beszúrható-e bele új adat.*

Az esetek többségében egy nézettáblába nem szúrhatunk be adatokat, ha az oszlopok valamelyike kifejezés vagy összesítő függvény eredményét tartalmazza. Egyes adatbázisrendszerek azt is lehetővé teszik, hogy a táblanév helyére a JOIN és ON kulcsszavakkal létrehozott nézetet írjunk. A részleteknek nézzünk utána saját adatbázisrendszerünk dokumentációjában. Ebben a fejezetben csak olyan példákat fogunk bemutatni, amelyek egy egyszerű táblába szúrnak be adatokat.

Végül a VALUES kulcsszó következik, majd egy nyitó zárójel, egymástól vesszővel elválasztott értékkifejezések listája, és egy záró zárójel. Jegyezzük meg, hogy az értékeket ugyanolyan sorrendben kell felsorolnunk, mint amilyen sorrendben az oszlopokat soroltuk fel az oszlopnévlistában. Az első oszlop értéke tehát az első kifejezés lesz, a második oszlop értéke a második kifejezés, és így tovább. Ha minden oszlopnak értéket adunk, és elhagyjuk az oszloplistát, akkor az értékeknek olyan sorrendben kell követniük egymást, mint ahogyan az oszlopok a tábla meghatározásában követik egymást. Ha azt akarjuk, hogy az adatbázis egy adott oszlopba a számára beállított alapértelmezett értéket írja, akkor ezt a DEFAULT kulcsszóval jelezhetjük. (Ha azonban nem határoztunk meg az oszlop számára alapértelmezést, akkor hibát kapunk.) NULL érték beszúrására a NULL kulcsszó használható.

A korábbi fejezetekből emlékezhetünk rá, hogy egy értékkifejezés meglehetősen bonyolult is lehet, és allekérdezéseket is tartalmazhat, amelyek akár a módosítandó táblából, akár más táblákból kérdezhetnek le adatokat. Emlékeztetőül álljon itt a 16.2. ábra, amely bemutatja az értékkifejezések felépítését.

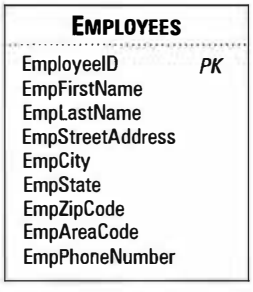


16.2. ábra

A VALUES záradékban az itt bemutatott szintaxisdiagramnak megfelelő értékkifejezéseket használhatunk az egyes oszlopok értékének megadására

Megjegyzés *Nem minden adatbázisrendszer teszi lehetővé SELECT kifejezés használatát az INSERT utasítás VALUES záradékában. A részleteknek nézzünk utána saját adatbázisrendszerünk dokumentációjában.*

Lássuk, hogyan szúrhatunk be egy újabb sort a SalesOrders adatbázis Employees táblájába! Hasonlóan minden más utasítás használatához, itt is ismernünk kell a tábla felépítését. Az Employees tábla felépítését a 16.3. ábrán láthatjuk.



16.3. ábra

Az Employees tábla a SalesOrders adatbázisban

Fogalmazzuk meg a kérelmet!

Megjegyzés *Ebben a fejezetben is a 4. fejezetben bevezetett „Kérelem – Fordítás – Tisztázás – SQL" módszert fogjuk használni.*

„Vedd fel a Susan Metters nevű új alkalmazottat, akinek a címe 16547 NE 132nd St, Woodinville, WA 98072, a területkódja 425, a telefonszáma pedig 555-7825!"

A kérelemben általában nem soroljuk fel tételesen az oszlopokat, de ne feledkezzünk meg róluk a fordítási lépésben. Az új alkalmazott felvételét így fordíthatjuk át:

Fordítás Insert into the employees table in the columns first name, last name, street address, city, state, ZIP Code, area code, and phone number the values Susan, Metters, 16547 NE 132nd St, Woodinville,WA, 98072, 425, and 555-7825
(Írd be az alkalmazottak táblájának keresztnév, vezetéknév, cím, város, állam, irányítószám, területkód és telefonszám oszlopaiba a következő értékeket: Susan, Metters, 16547 NE 132nd St, Woodinville, WA, 98072, 425, és 555-7825.)

Tisztázás Insert into ~~the~~ employees ~~table in the columns~~
(first name, last name, street address, city, state, ZIP Code, area code, ~~and~~ phone number)
~~the~~ values ('Susan', 'Metters', '16547 NE 132nd St','Woodinville', 'WA', '98072', 425, ~~and~~ '555-7825')

SQL
```
INSERT INTO Employees
   (EmpFirstName, EmpLastName,
   EmpStreetAddress, EmpCity, EmpState,
```

```
    EmpZipCode, EmpAreaCode, EmpPhoneNumber)
VALUES ('Susan', 'Metters',
    '16547 NE 132nd St', 'Woodinville', 'WA',
    '98072', 425, '555-7825')
```

A fenti utasítás megtalálható a SalesOrders adatbázis módosítható változatában CH16_Add_Employee néven.

Vajon miért nem foglaltuk bele a lekérdezésbe az elsődleges kulcsot, az EmployeeID (alkalmazottazonosító) oszlopot? Ha kíváncsiak vagyunk a válaszra, olvassunk tovább!

Az elsődleges kulcs következő értékének előállítása

A fenti példalekérdezésbe nem foglaltuk bele az elsődleges kulcsot, az EmployeeID-t. Az elsődleges kulcsnak azonban minden adatbázisrendszerben kötelezően értékkel rendelkeznie. Nem fog az utasítás végrehajtása hibával félbeszakadni?

A válasz: nem, de csak azért, mert kihasználunk egy olyan szolgáltatást, amely majdnem minden kereskedelmi adatbázis-megvalósításban megtalálható. Ha nincs jelentősége az elsődleges kulcs pontos értékének – megelégszünk azzal, hogy valamilyen egyedi értéket tartalmazzon –, akkor létrehozhatjuk az elsődleges kulcsot egy olyan adattípussal, amelyet az adatbázisrendszer minden új sor beírásakor automatikusan megnövel. A Microsoft Accessben ez a típus az AutoNumber. Az AutoNumber valójában egy olyan egész szám, amely különleges jellemzőkkel (attribútumokkal) rendelkezik. A Microsoft SQL Server az Identity adattípust használja (amely szintén egész szám). A MySQL-ben olyan egész számot tartalmazó oszlopot használhatunk, amelynek beállítjuk az AUTO_INCREMENT jellemzőjét. A példánkban használt SQL-utasításforma mindhárom típusú mintaadatbázisban működni fog, mert mindhárom mintaadatbázis módosítható változatában a megfelelő különleges képességet használva hoztuk létre az elsődleges kulcsot tartalmazó oszlopot szinte minden táblában.

Az Oracle adatbázis egy kicsit másképp működik. Nem különleges adattípust kínál, hanem lehetővé teszi egy Sequence (sorozat) típusú áloszlop használatát, amelynek a NEXTVAL jellemzőjére kell hivatkoznunk minden esetben, amikor egyedi azonosítóra van szükségünk egy új sor számára. Tegyük fel, hogy Oracle adatbázisunkban korábban létrehoztunk egy EmpID nevű áloszlopot. Az SQL-utasításunk az alábbi lehet:

SQL

```
INSERT INTO Employees
(EmployeeID, EmpFirstName, EmpLastName,
EmpStreetAddress, EmpCity, EmpState,
EmpZipCode, EmpAreaCode, EmpPhoneNumber)
VALUES (EmpID.NEXTVAL, 'Susan', 'Metters',
'16547 NE 132nd St', 'Woodinville', 'WA',
'98072', 425, '555-7825')
```

Vegyük észre, hogy minden oszlopnak adunk értéket, ráadásul abban a sorrendben, ahogyan a tábla meghatározásában szerepelnek. Így elhagyhatjuk az oszlopnévlistát, tehát az SQL-utasítás lehet az alábbi is:

```
SQL     INSERT INTO Employees
        VALUES (EmpID.NEXTVAL, 'Susan', 'Metters',
        '16547 NE 132nd St', 'Woodinville', 'WA',
        '98072', 425, '555-7825')
```

Ahogyan már korábban is említettük, nem javasoljuk az oszlopnévlista elhagyását, mert ha az adatbázis rendszergazdája új oszlopot ad a táblához, vagy felcseréli az oszlopok sorrendjét, akkor az INSERT utasítás végrehajtása meghiúsul. Ezt a formát csak a teljesség kedvéért idéztük.

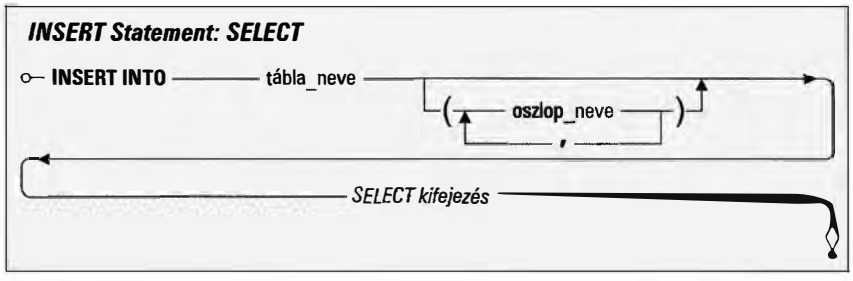
Ha éppen elmélkedős hangulatban vagyunk, eszünkbe juthat, hogy nem lehetne éppen ilyen egyszerűen egy allekérdezés használatával előállítani a következő elsődleges kulcsot? Az SQL-szabvány természetesen ezt is támogatja; ennek megfelelően az INSERT utasítás az alábbi formát is öltheti:

```
SQL     INSERT INTO Employees
        (EmployeeID,
        EmpFirstName, EmpLastName,
        EmpStreetAddress, EmpCity, EmpState,
        EmpZipCode, EmpAreaCode, EmpPhoneNumber)
        VALUES (
        (SELECT MAX(EmployeeID) FROM Employees) + 1,
        'Susan', 'Metters',
        '16547 NE 132nd St', 'Woodinville', 'WA',
        '98072', 425, '555-7825')
```

Sajnos azonban a főbb adatbázisrendszerek közül több nem támogatja az allekérdezések használatát a VALUES záradékban. A részletek tekintetében hagyatkozzunk adatbázisrendszerünk dokumentációjára.

Adatok beszúrása a SELECT utasítás segítségével

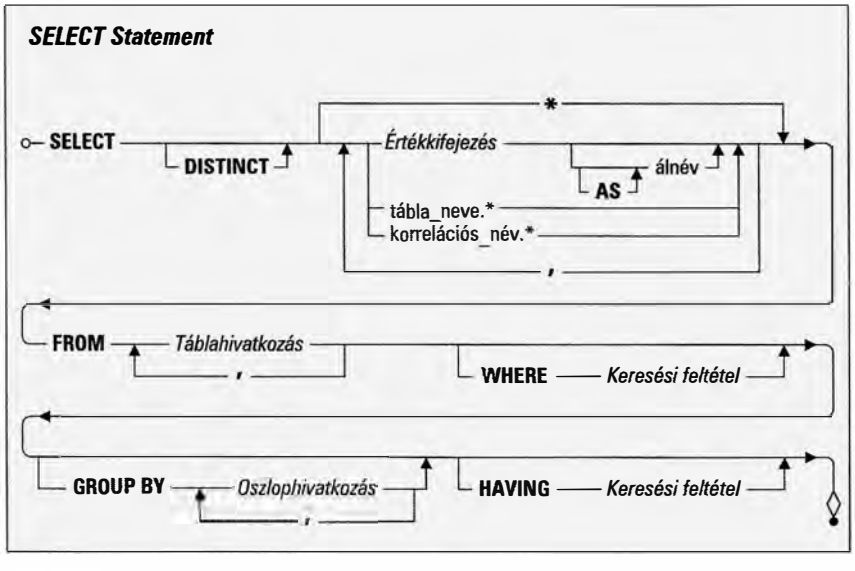
Olvasónk elgondolkodhat, hogy vajon miért adtuk az *Adathalmazok beszúrása* címet a fejezetnek, ha egyszerre csak egy sort szúrunk be? Bizonyos értelemben egy sor több adata is adathalmazt alkot, de adathalmaznak általában több sor együttesét nevezzük. Semmi baj: sorok halmazát is beszúrhatjuk egy táblába, ha SELECT kifejezést alkalmazunk a VALUES záradék helyett. Mivel a SELECT kifejezések egy vagy több táblából olvasnak be sorokat, a SELECT-et használó INSERT-et az adatmásolás hatékony módszerének is tekinthetjük. A SELECT kifejezést tartalmazó INSERT utasítás szintaxisdiagramját a 16.4. ábra mutatja.



16.4. ábra

A SELECT kifejezést tartalmazó INSERT utasítás szintaxisdiagramja

Vegyük észre, hogy az INSERT utasítás e változata ugyanúgy kezdődik, mint az előző: az INSERT INTO kulcsszavak után megadjuk a módosítandó tábla nevét. Ha a SELECT kifejezés ugyanannyi oszlopot ad vissza, ugyanolyan sorrendben, mint ahogyan azok a módosítandó táblában szerepelnek, akkor elhagyhatjuk a nem kötelező oszlopnévlistát. De ahogyan azt korábban is javasoltuk, még ha minden oszlopnak tervezünk is értéket adni, akkor se hagyjuk el az oszlopnévlistát, ha ugyanis elhagyjuk, és valaki új oszlopot ad a táblához, vagy módosítja az oszlopok sorrendjét, akkor az utasításunk végrehajtása meghiúsul.



16.5. ábra

A SELECT utasítás szintaxisdiagramja

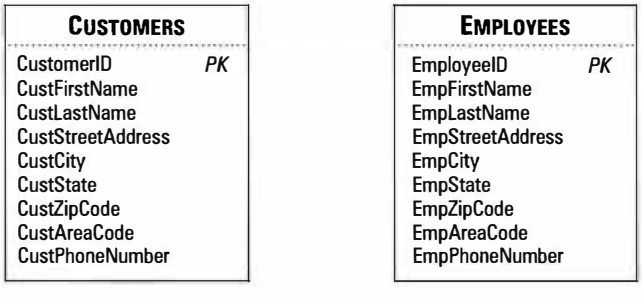
Ha megvizsgáljuk az „A" függelékben közölt szabványos SQL-szintaxisdiagramokat, akkor láthatjuk, hogy a SELECT kifejezés egyszerűen egy SELECT utasítás, amelyet esetlegesen UNION, INTERSECT vagy EXCEPT művelettel más SELECT utasításokhoz kapcsolunk. (Az említett három művelet leírását megtaláljuk a 7. fejezetben, a UNION utasítás részletes magyarázata pedig a 10. fejezetben található.) A SELECT utasítás szintaxisdiagramja a 16.5. ábrán látható.

A korábbi fejezetekből emlékezhetünk, hogy a „táblahivatkozás" lehet egy táblanév, táblanevek vesszővel elválasztott listája, illetve két vagy több tábla összekapcsolása JOIN záradékokkal. A „keresési feltétel" lehet egy oszlop és egy érték egyszerű összehasonlítása, a BETWEEN, IN, LIKE, NULL kulcsszavakat tartalmazó bonyolultabb feltétel, vagy még bonyolultabb allekérdezés. Röviden: a SELECT lekérdezések minden képessége rendelkezésünkre áll a beszúrandó sor adatainak meghatározására.

Ássunk mélyebbre néhány olyan példa segítségével, amelyeknek a megoldásához SELECT kifejezést tartalmazó INSERT utasítást kell használnunk. Íme az első kérelem, amely egy tábla adott sorának lemásolását kéri egy másik táblába:

> *„Épp most vettük fel alkalmazottnak David Smith-t, aki a vásárlónk volt. Másold át David Smith összes adatát a vásárlók táblájából az alkalmazottak táblájába!"*

Mind minden utasítás megírásakor, most is tisztában kell lennünk a részt vevő táblák szerkezetével. A két kérdéses tábla felépítését a 16.6. ábra mutatja.



16.6. ábra

A Sales Orders mintaadatbázis Customers és Employees táblája

Fogalmazzuk át a kérelmet úgy, hogy könnyebb legyen SQL-utasítássá alakítanunk:

> *„Másold át az alkalmazottak táblájába a vásárlók táblájának megfelelő oszlopait a David Smith nevű vásárló esetében!"*
>
> Fordítás Insert into the employees table in the columns first name, last name, street address, city, state, ZIP code, area code, and phone

number the selection of the first name, last name, street address, city, state, ZIP code, area code, and phone number columns from the customers table where the customer first name is 'David' and the customer last name is 'Smith'
(Írd be az alkalmazottak táblájának keresztnév, vezetéknév, cím, város, állam, irányítószám, területkód és telefonszám oszlopaiba annak a lekérdezésnek az eredményét, amely kiválasztja a keresztnevet, vezetéknevet, címet, várost, államot, irányítószámot, területkódot és telefonszámot a vásárlók táblájából, ha a vásárló keresztneve David, a vezetékneve pedig Smith.)

Tisztázás

Insert into ~~the~~ employees ~~table~~
~~in the columns~~ (first name, last name, street address,
city, state, ZIP code,
area code, ~~and~~ phone number)
~~the~~ selection ~~of the~~ first name, last name, street address,
city, state, ZIP code,
area code, ~~and~~ phone number ~~columns~~
from ~~the~~ customers ~~table~~
where ~~the~~ customer first name ~~is~~ = 'David'
and ~~the~~ customer last name ~~is~~ = 'Smith'

SQL

```
INSERT INTO Employees
   (EmpFirstName, EmpLastName, EmpStreetAddress,
      EmpCity, EmpState, EmpZipCode,
      EmpAreaCode, EmpPhoneNumber)
   SELECT Customers.CustFirstName,
      Customers.CustLastName,
      Customers.CustStreetAddress,
      Customers.CustCity,
      Customers.CustState, Customers.CustZipCode,
      Customers.CustAreaCode,
      Customers.CustPhoneNumber
   FROM Customers
   WHERE (Customers.CustFirstName = 'David')
   AND (Customers.CustLastName = 'Smith')
```

Figyeljük meg, hogy nem szerepeltettük az EmployeeID oszlopot, hanem az adatbázisra hagytuk, hogy automatikusan állítsa elő a következő egyedi értéket a beszúrandó új sor(ok) számára. Ezt a lekérdezést CH16_Copy_Customer_To_Employee néven mentettük a Sales Orders mintaadatbázis módosítható változatába.

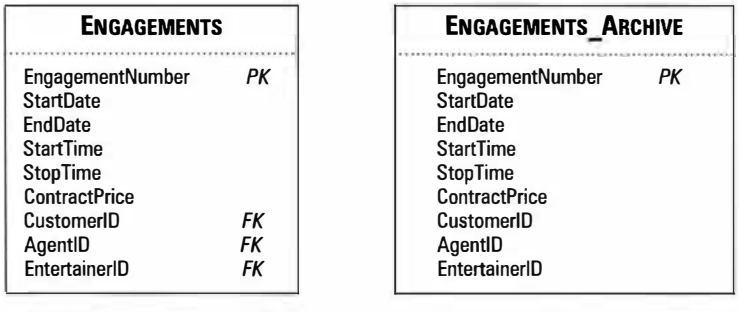
Mivel csak egyetlen David Smith nevű vásárlónk van, a fenti utasítás csak egyetlen sort szúr be az Employees táblába. Ez még mindig nem adathalmaz, de láthatjuk, milyen egyszerű egy SELECT kifejezés segítségével lekérdezni a beszúráshoz szükséges adatokat, ha azok rendelkezésre állnak egy másik táblában.

Lépjünk tovább egy olyan példára, amelyben esetleg akár sorok százait szúrjuk be egyszerre. Minden olyan adatbázis-alkalmazásban, amely időről időre új információkat gyűjt, általában található olyan szolgáltatás, amelynek a segítségével a felhasználók archiválhatják a korábbi adataikat, vagy biztonsági másolatot készíthetnek egy adott időpontnál régebben keletkezett adatokról egy másik táblába. Az ötletnek az az alapja, hogy nem szerencsés régi adatok tömkelegével lassítani az új adatok feldolgozását, mert archiválás nélkül minden lekérdezésnek rég elavult adatok tengerében kell keresnie a megfelelő sorokat.

Írjunk tehát egy olyan INSERT utasítást, amely minden olyan tranzakció adatát, amely egy adott dátumnál korábban keletkezett, átmásolja egy másik táblába. (A következő fejezetben azt is bemutatjuk majd, hogyan törölhetjük az archivált adatokat az aktív táblából.) A kérelem jellemzően így fest:

„Archiválj minden 2008. január 1-je előtti rendezvényt!"

Ebben a különleges esetben az Engagements (Rendezvények) és az Engagements_Archive (Archív rendezvények) táblának azonos a felépítése, ahogyan azt a 16.7. ábra is mutatja.



16.7. ábra

Az Entertainments Agency mintaadatbázis Engagements és Engagements_Archive táblája

Ez például olyan eset, amikor biztonságosan elhagyható az oszloplista. A fordítás igen egyszerű:

Fordítás	Insert into the engagements archive table the selection of all columns from the engagements table where the engagement end date is earlier than January 1, 2008 (Szúrd be az archív rendezvények táblájába annak a lekérdezésnek az eredményét, amely kiválasztja az összes olyan oszlopot a rendezvények táblájából, ahol a végdátum 2008. január 1. előtti.)
Tisztázás	Insert into ~~the~~ engagements archive ~~table~~

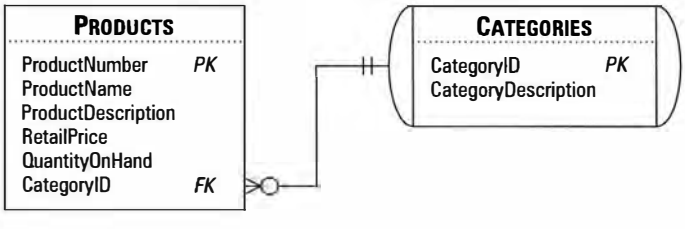
~~the~~ selection of all columns *
from ~~the~~ engagements ~~table~~
where ~~the~~ engagement end date
~~is earlier than~~ < ~~January 1, 2008~~ '2008-01-01'

```
SQL     INSERT INTO Engagements_Archive
           SELECT Engagements.*
           FROM Engagements
           WHERE Engagements.EndDate
              < '2008-01-01'
```

Ez már szinte túl egyszerű! De emlékezzünk vissza: korábban azt javasoltuk, hogy mindig kifejezetten soroljuk fel, hogy mely oszlopoknak akarunk értéket adni. Ha így teszünk, akkor az utasításunk akkor is képes lesz lefutni, ha valaki új oszlopot ad bármelyik táblához, vagy módosítja az oszlopok sorrendjét. Igaz, hogy ez egy kicsit több erőfeszítést igényel, mégis azt javasoljuk, hogy a feladatot megoldó SQL-utasítást inkább így írjuk:

```
SQL     INSERT INTO Engagements_Archive
           (EngagementNumber, StartDate, EndDate,
            StartTime, StopTime, ContractPrice,
            CustomerID, AgentID, EntertainerID)
           SELECT Engagements.EngagementNumber,
              Engagements.StartDate, Engagements.EndDate,
              Engagements.StartTime, Engagements.StopTime,
              Engagements.ContractPrice,
              Engagements.CustomerID,
              Engagements.AgentID, Engagements.EntertainerID
           FROM Engagements
           WHERE Engagements.EndDate < '2008-01-01'
```

A lekérdezést CH16_Archive_Engagements néven találhatjuk meg az Entertainment Agency mintaadatbázis módosítható változatában.



16.8. ábra

A Products és a hozzá kapcsolódó Categories tábla a Sales Orders adatbázisban

Most pedig lássunk egy példát a SELECT kifejezés kreatív használatára:

„Vegyél fel egy új terméket a kerékpárok termékcsoportjába Hot Dog Spinner néven, 895 dolláros fogyasztói áron!"

A megoldáshoz szükséges táblák szerkezete a 16.8. ábrán látható.

Addig világos, hogy a módosítandó tábla a Products (Termékek), de abban a CategoryID (termékcsoport-azonosító) oszlopba egy számértékű azonosítót kell tennünk. A kérelem azt mondja „a kerékpárok termékcsoportjába". Hogyan találjuk ki a szükséges termékcsoport-azonosítót? Természetesen SELECT kifejezéssel! A CategoryID-n kívül a ProductName (Terméknév) és a RetailPrice (Fogyasztói ár) oszlopot is ki akarjuk tölteni, de emlékezzünk vissza: a SELECT utasítás literális értéket is visszaadhat egy – vagy akár az összes – oszlop esetében. Tehát a megoldás az, hogy lekérdezzük a termékcsoport-azonosítót a Categories táblából, a többi értéket pedig literális értékekként adjuk meg. Fogalmazzuk át a kérelmet, és adjunk rá megoldást!

„Adj egy új sort a termékek táblájához, amelyben a termék neve legyen Hot Dog Spinner, a fogyasztói ára 895 dollár, a termékcsoport-azonosító pedig a Bikes leírású termékcsoport azonosítója a termékcsoportok táblájából!"

Fordítás Insert into the products table in the columns product name, retail price, and category ID the selection of 'Hot Dog Spinner' as the product name, 895 as the retail price, and category ID from the categories table where the category description is equal to 'Bikes'
(Írd be a termékek táblájának termék, fogyasztói ár és termékcsoport-azonosító oszlopaiba annak a lekérdezésnek az eredményét, amely kiválasztja a Hot Dog Spinner szöveget, a 985 számot és a termékcsoport-azonosítót a termékcsoportok táblájából, ha a termékcsoport leírása „Bikes".)

Tisztázás Insert into ~~the~~ products ~~table~~ ~~in the columns~~ (product name, retail price, ~~and~~ category ID) ~~the~~ selection ~~of~~ 'Hot Dog Spinner' as ~~the~~ product name, 895 as ~~the~~ retail price, ~~and~~ category ID from ~~the~~ categories ~~table~~ where ~~the~~ category description ~~is equal to~~ = 'Bikes'

SQL

```
INSERT INTO Products
(ProductName, RetailPrice, CategoryID)
SELECT 'Hot Dog Spinner' AS ProductName,
895 AS RetailPrice, CategoryID
FROM Categories
WHERE CategoryDescription = 'Bikes'
```

Korábban azt gondolhattuk volna, hogy a SELECT kifejezések csak arra jók, hogy teljes sorokat másoljunk, de amint látjuk, arra is alkalmasak, hogy a beszúráshoz szükséges összes adat közül csak néhányat határozzunk meg más táblák adatai alapján. A fenti módszer alkalmazására láthatunk néhány érdekes esetet a *Példák* részben.

Az INSERT használati területei

Ha idáig elértünk, akkor már tudjuk, hogy miként lehet egy vagy akár több sort beszúrni egy táblába egyszerű VALUES záradék vagy egy SELECT kifejezés segítségével. Az INSERT utasítás széles felhasználási körének bemutatására az a legjobb, ha felsorolunk néhány olyan problémát, amelyre az INSERT utasítás használata adja a megoldást, majd egy sor példán is bemutatjuk a használatát. Az itt felsorolt problémák csak ízelítőt adnak az INSERT-tel megoldható feladatok tömkelegéből:

„Hozd létre Matthew Patterson tekejátékos rekordját Neil Patterson rekordjának lemásolásával!"
„Vedd fel az Entertainment adatbázisba vásárlóként Kendra Hernandezt, akinek a címe 457 211th St NE, Bothell, WA 98200, a telefonszáma pedig 555-3945!"
„A Sales Orders adatbázisba vedd fel vásárlóként Mary Bakert, akinek a címe 7834 W 32nd Ct, Bothell, WA 98011, a területkódja 425, a telefonszáma pedig 555-9876!"
„Hozz létre egy „Italian" nevű új tantárgycsoportot a Humanities tanszék számára „ITA" tantárgykóddal!"
„Vegyél fel egy új csapatot Aardvarks néven, csapatkapitány nélkül!"
„Rögzíts egy új rendezvényt a Matt Berg nevű megrendelő számára: foglald le a Jazz Persuasiont 2008. augusztus 15-ére és 16-ára, este 7-től 11-ig; az üzletet Karen Smith kötötte!"
„Vedd fel a Hot Dog Bikes nevű új beszállítót, amelynek a címe 1234 Main Street, Chicago, IL 60620, a telefonszáma (773) 555-6543, a faxszáma (773) 555-6542, a honlapjának címe http://www.hotdogbikes.com/ és az e-mail címe Sales@hotdogbikes.com!"
„Vegyél fel egy új órát a 4-es azonosítójú tantárgy (Intermediate Accounting) számára 5 kreditpontért! Az előadást a 3315-ös teremben tartják kedd és csütörtök délutánonként 3-kor, és 80 perc hosszú."
„Archiváld a 2007-es évad minden tornáját, mérkőzését, a játékokat és a játékosok pontszámait!"
„Vedd fel a New Age-et a zenei stílusok listájára!"
„Archiváld a 2008. január 1-je előtt keletkezett rendeléseket a részletező soraikkal együtt!"
„Angel Kennedy hallgatóként akarja bejegyeztetni magát. A férje már korábban beiratkozott. Rögzíts egy új hallgatórekordot Angel számára, felhasználva John rekordjának adatait!"
„Másold le 2007 minden tornáját és mérkőzését 2009 azonos hetére!"
„Az ügynök Marianne Wier le szeretne kötni néhány előadót. Hozz létre a számára megrendelőrekordot az ügynökök tábla adatainak felhasználásával!"

„Liz Keyser nevű vásárlónk szeretné ugyanazokat a termékeket megrendelni, mint amelyeket 2007. december 11-én rendelt. Készítsd el az új rendelését 2008. június 12-i rendelési dátummal; a szállítási dátum legyen 2008. június 15.!"
„Tim Smith, a tanári kar tagja, szeretne beiratkozni hallgatóként. Hozz létre hallgatórekordot Tim számára a tanári rekord adatainak felhasználásával!"
„Doris Hartwig nevű megrendelőnk 2008. augusztus 4-ére szeretné lekötni ugyanazokat az előadókat, akik 2007. december 1-jén játszottak nála."
„Angel Kennedy nevű vásárlónk meg szeretné rendelni ugyanazokat a termékeket, mint amelyeket 2007. november folyamán rendelt. Készítsd el számára a rendelést 2008. június 15-i rendelési és 2008. június 18-i szállítási dátummal!"

Példák

Már ismerjük az INSERT utasítás felépítésének módszerét, lássunk hát egy sor példát, amelyek mindegyikében egy vagy több sort szúrunk be egy táblába! A példák a mintaadatbázisokból származnak. Minden példa mellé mellékeltük azokak a soroknak az adatait, amelyet az adott INSERT utasításnak a táblába kell szúrnia, és megadtuk az újonnan keletkező sorok számát is.

A módosítás utáni adatokat tartalmazó táblázat címében annak a lekérdezésnek a neve szerepel, amellyel azt a mellékletként adott CD-n szereplő mintaadatbázisokban megtalálhatjuk. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH16"-tal kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne feledjük, hogy a példákban használt minden tábla- és oszlopnév megfelel a mintaadatbázisok B függelékben található sémaleírásának. A leírások egyszerűsítése céljából a Fordítás és Tisztázás részeket minden példánál összevontuk. A példák feltételezik, hogy az olvasó áttanulmányozta és megértette a korábbi fejezetekben leírt ismereteket.*

Sales Orders adatbázis

„Vedd fel új a Hot Dog Bikes nevű beszállítót, amelynek a címe 1234 Main Street, Chicago, IL 60620, a telefonszáma (773) 555-6543, a faxszáma (773) 555-6542, a honlapjának címe http://www.hotdogbikes.com/ és az e-mail címe Sales@hotdogbikes.com!"

Fordítás/ Insert into ~~the~~ vendors ~~table~~
Tisztázás ~~in the columns~~ (VendName,VendStreetAddress,VendCity,
VendState,VendZipCode,VendPhoneNumber,
VendFaxNumber,VendWebPage,
~~and~~ VendEMailAddress)
~~the~~ values ('Hot Dog Bikes', '1234 Main Street',

```
'Chicago', 'IL', '60620', '(773) 555-6543',
'(773) 555-6542', 'http://www.hotdogbikes.com/',
and 'Sales@hotdogbikes.com')
```

SQL

```
INSERT INTO Vendors
   (VendName, VendStreetAddress, VendCity,
    VendState, VendZipCode, VendPhoneNumber,
    VendFaxNumber, VendWebPage,
    VendEMailAddress)
VALUES ('Hot Dog Bikes', '1234 Main Street',
'Chicago',
    'IL', '60620', '(773) 555-6543',
    '(773) 555-6542', 'http://www.hotdogbikes.com/',
    'Sales@hotdogbikes.com')
```

A sor, amelyet a CH16_Add_Vendor lekérdezés szúr be a Vendors táblába (1 új sor)

VendName	VendStreet Address	VendCity	Vend State	VendZip Code	VendPhone Number	VendFax Number	VendWeb Page	VendEMail Address
Hot Dog Bikes	1234 Main Street	Chicago	IL	60620	(773) 555-6543	(773) 555-6542	http://www.hotdogbikes.com/	Sales@hotdogbikes.com

„Archiváld a 2008. január 1-je előtt keletkezett rendeléseket és a részletező soraikat!"

Megjegyzés *A rendelés összes adatának archiválásához mind az Orders, mind az Order_Details tábla adatait le kell másolni, tehát két utasításra lesz szükségünk. Először a rendeléseket beszúró INSERT utasítást futtassuk, mert az Order_Details_Archive tábla OrderID oszlopa idegen kulcsot tartalmaz, amely az Orders_Archive azonos nevű oszlopára mutat.*

Ha adatbázisrendszerünk támogatja a tranzakciókat (lásd a 15. fejezetben), akkor érdemes egy új tranzakciót nyitnunk, majd először lefuttatni a rendeléseket, utána pedig a rendelésrészletezőket lemásoló utasítást, és csak akkor véglegesíteni (COMMIT) a módosításokat, ha mindkettő hiba nélkül lefutott. Ha a második INSERT utasítás hibát eredményez, akkor visszagörgethetjük a módosításokat. Nincs értelme az adatoknak csak az egyik felét menteni.

Mivel dátumok alapján kell kiválogatni az archiválandó sorokat, a rendelésrészletezők archiválásakor allekérdezést kell használnunk, amellyel kiválogathatjuk azoknak a megrendeléseknek az azonosítóját, amelyek az adott dátum előtt keletkeztek.

Fordítás 1/ Insert into ~~the~~ orders archive ~~table~~

Tisztázás the selection of order number, order date, ship date,
customer ID, employee ID, and order total
from the orders table
where the order date is earlier than < '2008-01-01'

SQL
```
INSERT INTO Orders_Archive
   SELECT OrderNumber, OrderDate, ShipDate,
      CustomerID, EmployeeID, OrderTotal
   FROM Orders
   WHERE OrderDate < '2008-01-01'
```

Fordítás 2/ Insert into the order details archive table

Tisztázás the selection of order number,product number,
quoted price, and quantity ordered
from the order details table
where the order number is in
the selection of the order number
from the orders table
where the order date is earlier than < '2008-01-01')

SQL
```
INSERT INTO Order_Details_Archive
   SELECT OrderNumber, ProductNumber,
      QuotedPrice, QuantityOrdered
   FROM Order_Details
   WHERE Order_Details.OrderNumber IN
   (SELECT OrderNumber
    FROM Orders
    WHERE Orders.OrderDate < '2008-01-01')
```

A sorok, amelyeket a CH16_Archive_2007_Orders lekérdezés szúr be az Orders_Archive táblába (598 új sor)

OrderNumber	OrderDate	ShipDate	CustomerID	EmployeeID	OrderTotal
1	2007-09-01	2007-09-04	1018	707	$12,751.85
2	2007-09-01	2007-09-03	1001	703	$816.00
3	2007-09-01	2007-09-04	1002	707	$11,912.45
4	2007-09-01	2007-09-03	1009	703	$6,601.73
5	2007-09-01	2007-09-01	1024	708	$5,544.75
6	2007-09-01	2007-09-05	1014	702	$9,820.29
7	2007-09-01	2007-09-04	1001	708	$467.85
8	2007-09-01	2007-09-01	1003	703	$1,492.60
<< *további sorok* >>					

Megjegyzés *Egyik lekérdezés megírásakor sem követtük azt az ajánlásunkat, miszerint mindig írjuk ki az oszlopnévlistát. Ezt a példát azért építettük fel így, hogy olyan esetet is be tudjunk mutatni, ahol nem feltétlenül szükséges az oszlopnévlista.*

Entertainment Agency adatbázis

„Vedd fel az Entertainment adatbázisba vásárlóként Kendra Hernandezt, akinek a címe 457 211th St NE, Bothell, WA 98200, a telefonszáma pedig 555-3945!"

Fordítás/ Tisztázás: Insert into ~~the~~ customers ~~table~~ ~~in the columns~~ (customer first name, customer last name, customer street address, customer city, customer state, customer ZIP Code, ~~and~~ customer phone number) ~~the~~ values ('Kendra', 'Hernandez', '457 211th St NE', 'Bothell','WA', '98200', ~~and~~ '555-3945')

SQL:

```
INSERT INTO Customers
   (CustFirstName, CustLastName,
    CustStreetAddress, CustCity, CustState,
    CustZipCode, CustPhoneNumber)
VALUES ('Kendra', 'Hernandez',
   '457 211th St NE', 'Bothell', 'WA',
   '98200', '555-3945')
```

A sor, amelyet a CH16_Add_Customer lekérdezés szúr be a Customers táblába (1 új sor)

CustFirstName	CustLastName	CustStreetAddress	CustCity	CustState	CustZipCode	CustPhoneNumber
Kendra	Hernandez	457 211th St NE	Bothell	WA	98200	555-3945

„Rögzíts egy új rendezvényt a Matt Berg nevű megrendelő számára: foglald le a Jazz Persuasiont 2008. augusztus 15-ére és 16-ára, este 7-től 11-ig; az üzletet Karen Smith kötötte!"

Megjegyzés *Ha vetünk egy pillantást az Engagements (Rendezvények) táblára, akkor láthatjuk, hogy szükségünk lesz Matt Berg vásárlóazonosítójára a Customers (Megrendelők) táblából, a Jazz Persuasion előadóazonosítójára az Entertainers (Előadók) táblából, és Karen Smith ügynökazonosítójára az Agents (Ügynökök) táblából. Ezeket az értékeket egy SELECT kifejezéssel kérdezhetjük le. A három kérdéses táblát JOIN feltételek nélkül írjuk a FROM záradékba. Ez a módszer csak azért működik, mert csak egy megrendelőnket hívják Matt Bergnek, csak egy ügynököt hívnak Karen Smith-nek, és csak egy Jazz Persuasion nevű előadónk van. Ha valamelyikből is több lenne, akkor több sort szúrnánk be az Engagements táblába. Ne felejtsük el kiszámítani a szerződési összeget az Entertainers táblában szereplő napidíj és a 15%-os haszonkulcs figyelembe vételével!*

Fordítás/ Insert into ~~the~~ engagements ~~table~~
Tisztázás ~~into the~~ (customer ID, entertainer ID, agent ID,
start date, end date,
start time, end time, ~~and~~ contract price) ~~columns~~
~~the~~ selection ~~of~~ customer ID, entertainer ID, agent ID,
~~and the values August 15, 2008~~ '2008-08-15',
~~August 16, 2008~~ '2008-08-16',~~'07:00:00 P.M.'~~ '19:00:00', ~~'11:00:00 P.M.'~~ '23:00:00',
~~and the~~ (entertainer price per day ~~times~~ * 2 ~~times~~ * 1.15)
from ~~the~~ customers, entertainers, ~~and~~ agents ~~tables~~
where ~~the~~ customer first name ~~is~~ = 'Matt'
and ~~the~~ customer last name ~~is~~ = 'Berg'
and ~~the~~ entertainer stage name ~~is~~ = 'Jazz Persuasion'
and ~~the~~ agent first name ~~is~~ = 'Karen'
and ~~the~~ agent last name ~~is~~ = 'Smith'

SQL

```
INSERT INTO Engagements
   (CustomerID, EntertainerID, AgentID,
    StartDate, EndDate,
    StartTime, StopTime,
ContractPrice)
   SELECT Customers.CustomerID,
       Entertainers.EntertainerID, Agents.AgentID,
       '2008-08-15', '2008-08-16',
       '19:00:00', '23:00:00',
       ROUND(EntPricePerDay * 2 * 1.15, 0)
   FROM Customers, Entertainers, Agents
   WHERE (Customers.CustFirstName = 'Matt')
       AND (Customers.CustLastName = 'Berg')
       AND (Entertainers.EntStageName = 'Jazz
       Persuasion')
       AND (Agents.AgtFirstName = 'Karen')
       AND (Agents.AgtLastName = 'Smith')
```

A sor, amelyet a CH16_Add_Engagament lekérdezés szúr be az Engagaments táblába (1 új sor)

CustomerID	EntertainerID	AgentID	StartID	EndID	StartTime	StopTime	ContractPrice
10006	1005	4	08/15/2007	08/16/2007	19:00:00	23:00:00	$288.00

School Scheduling adatbázis

„Hozz létre egy „Italian" nevű új tantárgycsoportot a Humanities tanszék számára „ITA" tantárgykóddal!"

Megjegyzés *Szükségünk lesz a Humanities tanszék azonosítójára, amelyet egy a Departments táblán futtatott SELECT kifejezéssel kaphatunk meg.*

Fordítás/ Insert into ~~the~~ categories ~~table~~
Tisztázás ~~the~~ selection ~~of~~ 'ITA' as ~~the~~ category ID,
'Italian' as ~~the~~ category description,
~~and~~ department ID
from ~~the~~ departments ~~table~~
where department name ~~is~~ = 'Humanities'

SQL
```
INSERT INTO Categories
   SELECT 'ITA' AS CategoryID,
      'Italian' AS CategoryDescription,
      Departments.DepartmentID
   FROM Departments
   WHERE Departments.DeptName = 'Humanities'
```

A sor, amelyet a CH16_Add_Category lekérdezés szúr be a Categories táblába (1 új sor)

CategoryID	CategoryDescription	DepartmentID
ITA	Italian	4

„Vegyél fel egy új órát a 4-es azonosítójú tantárgy (Intermediate Accounting) számára 5 kreditpontért! Az előadást a 3315-ös teremben tartják kedd és csütörtök délutánonként 3-kor, és 80 perc hosszú."

Megjegyzés *Feltételezhetjük, hogy az ütemezett napok alapértelmezett értéke 0 vagy hamis, tehát csak a kedd és a csütörtök számára kell az 1 vagy igaz értéket megadni.*

Fordítás/ Insert into ~~the~~ classes ~~table~~
Tisztázás ~~into the columns~~ (subject ID, classroom ID, credits, start time,
duration,Tuesday schedule, ~~and~~ Thursday schedule)
~~the~~ values (4, 3315, 5, ~~3 PM~~ 15:00:00,
80, 1, ~~and~~ 1)

SQL
```
INSERT INTO Classes
     (SubjectID, ClassRoomID, Credits, StartTime,
     Duration, TuesdaySchedule, ThursdaySchedule)
   VALUES (4, 3315, 5, '15:00:00',
      80, 1, 1)
```

A sor, amelyet a CH16_Add_New_Accounting_Class lekérdezés szúr be a Classes táblába (1 új sor)

SubjectID	ClassRoomID	Credits	StartTime	Duration	TuesdaySchedule	ThursdaySchedule
4	3315	5	15:00:00	80	1	1

Bowling League adatbázis

„Hozd létre Matthew Patterson tekejátékos rekordját Neil Patterson rekordjának lemásolásával!"

Megjegyzés *Az összes leütött bábut, a lejátszott mérkőzések számát, az aktuális átlagot és az aktuális büntetőpontszámot állítsuk 0-ra.*

Fordítás/ Tisztázás

Insert into ~~the~~ bowlers ~~table~~
~~into the columns~~ (bowler last name, bowler first name,
bowler address, bowler city,
bowler state, bowler zip,
bowler phone number, team ID,
bowler total pins, bowler games bowled,
bowler current average, ~~and~~ bowler current handicap)
~~the~~ select~~ion of~~ bowler last name, ~~the value~~ 'Matthew',
bowler address, bowler city,
bowler state, bowler zip,
bowler phone number, team ID,
~~and the values~~ 0, 0,
0, ~~and~~ 0
from ~~the~~ bowlers ~~table~~
where ~~the~~ bowler last name ~~is~~ = 'Patterson'
and ~~the~~ bowler first name ~~is~~ = 'Neil'

SQL

```
INSERT INTO Bowlers
   (BowlerLastName, BowlerFirstName,
    BowlerAddress, BowlerCity,
    BowlerState, BowlerZip,
    BowlerPhoneNumber, TeamID,
    BowlerTotalPins, BowlerGamesBowled,
     BowlerCurrentAverage, BowlerCurrentHcp)
   SELECT Bowlers.BowlerLastName, 'Matthew',
      Bowlers.BowlerAddress, Bowlers.BowlerCity,
      Bowlers.BowlerState, Bowlers.BowlerZip,
      Bowlers.BowlerPhoneNumber, Bowlers.TeamID,
```

A sor, amelyet a CH16_Add_Bowler lekérdezés szúr be a Bowlers táblába (1 új sor)

BowlerLastName	BowlerFirstName	BowlerAddress	BowlerCity	BowlerState	BowlerZip
Patterson	Matthew	16 Maple Lane	Auburn	WA	98002
BowlerPhone Number	**TeamID**	**BowlerTotalPins**	**BowlerGames Bowled**	**BowlerCurrent Average**	**BowlerCurrent Hcp**
(206) 555-3487	2	0	0	0	0

```
    0, 0,
    0, 0
FROM Bowlers
WHERE (Bowlers.BowlerLastName = 'Patterson')
    AND (Bowlers.BowlerFirstName = 'Neil')
```

„Vegyél fel egy új csapatot Aardvarks néven, csapatkapitány nélkül!"

Fordítás/ Insert into ~~the~~ teams ~~table~~
Tisztázás ~~into the columns~~ (team name, ~~and~~ captain ID)
~~the~~ values ('Aardvarks', ~~and~~ Null)

SQL
```
INSERT INTO Teams
      (TeamName, CaptainID)
VALUES ('Aardvarks', NULL)
```

A sor, amelyet a CH16_Add_Team lekérdezés szúr be a Teams táblába (1 új sor)

TeamName	CaptainID
Aardvarks	NULL

Összefoglalás

A fejezetet az új adatok beszúrására használható INSERT utasítás rövid leírásával kezdtük. Bemutattuk az INSERT utasításformáját és egy egyszerű példát, amely egy értéklistával megadott sort szúrt be egy táblába. Ez után megmutattuk azokat a módszereket, amelyeket a különböző adatbázisrendszerek kínálnak az elsődleges kulcs egyedi értékének előállítására. Elmondtuk, hogy a Microsoft SQL Server az Identity adattípust, a Microsoft Access az AutoNumber adattípust, a MySQL pedig az AUTO_INCREMENT jellemzőt kínálja. Röviden elmagyaráztuk az Oracle adatbázisokban használt Sequence áloszlop használatának módját, végül pedig bemutattuk, hogy miként lehet a VALUES záradékban alkalmazott allekérdezéssel meghatározni az elsődleges kulcs létező legnagyobb értékét, és azt eggyel növelni.

Felfedező utunkat a SELECT kifejezéseket alkalmazó INSERT utasításokkal folytattuk, amelyek sorok másolására használhatók. Először áttekintettük a SELECT kifejezés utasításformáját, majd bemutattuk, hogyan lehet egy sort az egyik táblából egy másikba másolni. Következő példánkban egyszerre több régi rekordot másoltunk át egy archiváló táblába, végül pedig bemutattuk, hogyan képes egy SELECT kifejezés egy kapcsolódó táblából egy vagy több adatot lekérdezni és azt az új sor beszúrásakor felhasználni. A fejezet további része példákkal szemléltette az INSERT utasítás használatát.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Liz Keyser nevű vásárlónk szeretné ugyanazokat a termékeket megrendelni, mint amelyeket 2007. december 11-én rendelt. Készítsd el az új rendelését 2008. június 12-i rendelési dátummal; a szállítási dátum legyen 2008. június 15.!"*
   (Tipp: az Orders és az Order Details tábla sorait is át kell másolnunk. Új rendelésazonosítónak megteszi, ha Liz Keyser 2008. december 11-i rendelésének azonosítójához hozzáadunk ezret.)
   A megoldás itt található: CH16_Copy_Dec11_Order_For_Keyser (1 új sor), illetve CH16_Copy_Dec11_OrderDetails_For_Keyser (4 új sor).
2. *„A Sales Orders adatbázisba vedd fel vásárlóként Mary Bakert, akinek a címe 7834 W 32nd Ct, Bothell, WA 98011, a területkódja 425, a telefonszáma pedig 555-9876!"*
   A megoldás itt található: CH16_Add_Customer (1 új sor).
3. *„Angel Kennedy nevű vásárlónk meg szeretné rendelni ugyanazokat a termékeket, mint amelyeket 2007. november folyamán rendelt. Készítsd el számára a rendelést 2008. június 15-i rendelési és 2008. június 18-i szállítási dátummal!"*
   (Tipp: az Orders és az Order Details tábla sorait is át kell másolnunk. Új rendelésazonosítónak megteszi, ha Angel Kennedy novemberi rendeléseinek azonosítóihoz hozzáadunk ezret.)
   A megoldás itt található: CH16_Copy_November_Orders_For_AKennedy (7 új sor), illetve CH16_Copy_November_OrderDetails_For_AKennedy (37 új sor).

Entertainment Agency adatbázis

1. *„Az ügynök Marianne Wier le szeretne kötni néhány előadót. Hozz létre a számára megrendelőrekordot az ügynökök tábla adatainak felhasználásával!"*
   (Tipp: egyszerűen másoljuk át a megfelelő oszlopokat az Agents táblából a Customers táblába.)
   A megoldás itt található: CH16_Copy_Agent_To_Customer (1 új sor).
2. *„Vedd fel a New Age-et a zenei stílusok listájára!"*
   A megoldás itt található: CH16_Add_Style (1 új sor).
3. *„Doris Hartwig nevű megrendelőnk 2008. augusztus 4-ére szeretné lekötni ugyanazokat az előadókat, akik 2007. december 1-jén játszottak nála."*

(Tipp: Használjunk SELECT kifejezést, amely JOIN záradékkal összekapcsolja a Customers és az Engagements táblát, a szükséges dátumokat pedig adjuk meg literális értékekként.)
A megoldás itt található: CH16_Duplicate_Engagement (1 új sor).

School Scheduling adatbázis

1. *„Angel Kennedy hallgatóként akarja bejegyeztetni magát. A férje már korábban beiratkozott. Rögzíts egy új hallgatórekordot Angel számára, felhasználva John rekordjának adatait!"*
   A megoldás itt található: CH16_Add_Student (1 új sor).
2. *„Tim Smith, a tanári kar tagja, szeretne beiratkozni hallgatóként. Hozz létre hallgatórekordot Tim számára a tanári rekord adatainak felhasználásával!"*
   A megoldás itt található: CH16_Enroll_Staff (1 új sor).

Bowling League adatbázis

1. *„Archiváld a 2007-es évad minden tornáját, mérkőzését, a játékokat és a játékosok pontszámait!"*
   (Tipp: négy különálló utasításra lesz szükségünk a Tournaments, a Tourney_Matches, a Match_Games és a Bowler_Scores tábla számára.)
   A megoldás itt található: CH16_Archive_2007_Tournaments_1 (14 új sor), CH16_Archive_2007_Tournaments_2 (57 új sor), CH16_Archive_2007_Tournaments_3 (168 új sor), illetve CH16_Archive_2007_Tournaments_4 (1344 új sor).
2. *„Másold le 2007 minden tornáját és mérkőzését 2009 azonos hetére!"*
   (Tipp: az egyszerűség kedvéért tételezzük fel, hogy az új TourneyID-k értékét megkaphatjuk úgy, hogy a 2007-es tornák azonosítóját 25-tel növeljük. Mind a Tourneys, mind a Tourney_Matches tábla sorait át kell másolnunk.)
   A megoldás itt található: CH16_Copy_2007_Tournaments_1 (14 új sor) és CH16_Copy_2007_Tournaments_2 (57 új sor).

17

Adathalmazok törlése

„Azért jöttem, hogy szeressem a soraimat,
babjaimat, de sokkal többet kaptam,
mint amire vágytam."
– Henry David Thoreau

A fejezet témakörei

- Mi az a DELETE?
- A DELETE utasítás
- A DELETE használati területei
- Példák
- Összefoglalás
- Önálló feladatok

Most már tudjuk, hogyan kell UPDATE utasítással adatokat módosítani. Azt is megtanultuk, hogyan kell INSERT utasítással adatokat hozzáadni egy táblához. De hogyan szabaduljunk meg a nemkívánatos adatoktól? Erre a célra az SQL talán legegyszerűbb, de legveszélyesebb utasítását – a DELETE utasítást használjuk.

Mi az a DELETE?

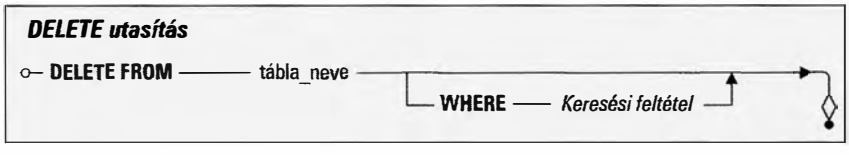
Az előző fejezetből megtudtuk, hogy táblákhoz egészen egyszerűen tudunk adatokat hozzáadni. A VALUES záradékkal egyszerre egy sort szúrhatunk be, míg egy SELECT kifejezéssel több sort is átmásolhatunk. De mit tegyünk, ha tévedésből adtunk hozzá egy sort egy táblához? Hogyan távolítsunk el olyan sorokat, amelyeket archív táblákba másoltunk? Hogyan távolítsuk el egy vásárló nevét, aki nem rendel semmit? Hogyan töröljük egy hallgató nevét, aki felvételizett, de egyetlen órára sem iratkozott fel? Hogyan távolítsunk el minden sort, ha üres táblákat akarunk kapni? A válasz mindegyik kérdésre a következő: használjuk a DELETE (törlés) utasítást.

Mint minden más utasítás az SQL-ben, a DELETE is sorhalmazokkal dolgozik. Ahogy azt majd az alábbi fejezetből megtudjuk, a legegyszerűbb DELETE utasítás az összes sort törli az általunk megadott táblából, legtöbbször azonban megadjuk a törlendő sorok részhalmazát. Teljes mértékben igazunk van, ha úgy gondoljuk, hogy ehhez egy WHERE záradékra van szükség.

Megjegyzés *Az összes példát és a feladatok megoldását megtalálhatjuk a megfelelő mintaadatbázisok módosítható („modify") változatában (SalesOrdersModify, EntertainmentAgencyModify, SchoolSchedulingModify, BowlingLeagueModify).*

A DELETE utasítás

A DELETE utasításnak csak három kulcsszava van: a DELETE, a FROM és a WHERE. A DELETE utasítás diagramja a 17.1. ábrán látható.



17.1. ábra

A DELETE utasítás szintaxisdiagramja

Azt állítottuk, hogy a DELETE utasítás az SQL talán legegyszerűbb utasítása, és természetesen nem vicceltünk – viszont ez a legveszélyesebb végrehajtható utasítás. Ha az utasításban nem szerepel a WHERE záradék, a DELETE a megadott tábla összes sorát törli. Ez egy új alkalmazás tesztelésénél lehet hasznos, például amikor egy létező tábla összes sorát törölni akarjuk, anélkül, hogy a tábla felépítése megváltozna. Tervezhetünk olyan alkalmazást is, amelyben munka- vagy ideiglenes táblákat töltünk fel olyan adatokkal, amelyek egy bizonyos feladatot hajtanak végre. Például általánosan elterjedt az INSERT utasítás használata, ha több sort akarunk átmásolni egy nagyon összetett SELECT kifejezésből egy olyan táblába, amelyet utána statikus jelentések készítéséhez fogunk használni. A WHERE záradék nélküli DELETE utasítás abban az esetben kényelmes megoldás, ha egy új jelentéssor futtatása előtt törölni akarjuk a régi sorokat.

Megjegyzés *Az SQL-szabvány szerint a tábla_neve egy lekérdezés (vagy nézettábla) neve is lehet, de a táblának, amire a lekérdezés neve vonatkozik, „frissíthetőnek" (updatable) kell lennie. Sok adatbázisrendszer támogatja sorok törlését a nézettáblákból, és minden adatbázisrendszernek megvannak a maga szabályai azzal kapcsolatban, hogy mitől is lesz egy tábla frissíthető nézettábla.*

Néhány adatbázisrendszer a nézettábla (az SQL-szabvány szóhasználatában „származtatott tábla") meghatározására a tábla_neve helyett a JOIN és ON kulcsszavak használatát támogatja. Azoknak a rendszereknek az esetében, amelyek támogatják a származtatott táblák használatát, közvetlenül a FROM kulcsszó után, tábla_neve. formátumban meg kell adni annak a táblának a nevét, amely a JOIN-nal végzett törlés célpontja. A részleteknek nézzünk utána az adatbázisrendszerünk dokumentációjában. Fejezetünkben kizárólag egyetlen táblát használunk a DELETE utasítás célpontjaként.*

Minden sor törlése

Talán veszélyesen is könnyű dolog minden sort törölni. A Bowling League mintaadatbázisban található Bowlers (Tekejátékosok) tábla alapján szerkesszünk egy DELETE utasítást!

Megjegyzés *Ebben a fejezetben is a 4. fejezetben bevezetett „Kérelem – Fordítás – Tisztázás – SQL" módszert fogjuk használni.*

„Törölj minden tekejátékost!"

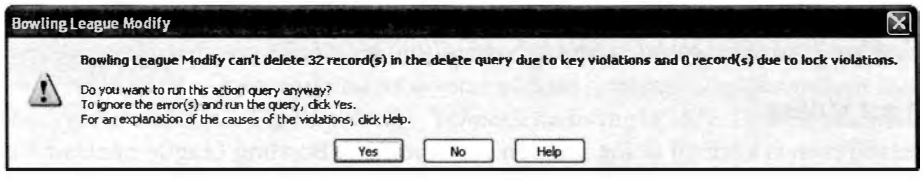
Fordítás	Delete all rows from the bowlers table (Törölj minden sort a tekejátékosok táblájából.)
Tisztázás	Delete ~~all rows~~ from ~~the~~ bowlers ~~table~~
SQL	`DELETE` `FROM Bowlers`

Ha a mintaadatbázisban végrehajtjuk ezt az utasítást, vajon tényleg töröl minden sort? Valójában nem, mert megadtunk egy megszorítást (egy hivatkozási épségi szabályt, amelyet a 2. fejezetben tárgyaltunk) a Bowlers és a Bowlers_Scores (Játékosok pontszámai) táblák közt. Ha egy adott játékoshoz egyetlen sor is tartozik a Bowlers_Scores táblában, az adatbázisrendszerünknek nem szabad megengednie az adott sor törlését a Bowlers táblából.

A Bowling League mintaadatbázis „modify" változatában szereplő két tekejátékos nem ért el egyetlen pontot sem, ezért egy egyszerű DELETE utasítással törölhetjük a rájuk vonatkozó bejegyzéseket. Még ha egyetlen sort sem akartunk igazán törölni, ez a két sor végérvényesen törlődik. Vagy mégsem? Először is, sok adatbázis-kezelő működtet tranzakciónaplót, amelyben rögzíti a táblákkal kapcsolatos módosításainkat, a rendszernaplóból pedig időnként vissza lehet állítani az elveszett adatokat. Emlékezzünk csak a 15. fejezetben röviden tárgyalt tranzakciókra. Ha új tranzakciót kezdünk (vagy a rendszer elindít egyet), hiba esetén visszafordíthatjuk a függőben lévő változásokat.

Amint azt már említettük, ne feledjük, hogy a Microsoft Office Access olyan adatbázisrendszer, amely automatikusan elindít egy tranzakciót, amikor végrehajtunk egy lekérdezést a felhasználói felületen. Ha a fenti lekérdezést megpróbáljuk a Microsoft Accessben futtatni,

a program először figyelmeztetést küld, és megadja, hogy hány sort készül törölni. Ha észrevesszük, hogy az adatbázis-kezelő a tábla összes sorát törölni fogja, leállíthatjuk a törlést. Ha hagyjuk, hogy a rendszer az első figyelmeztetés után folytassa a tranzakciót, a 17.2. ábrán látható hibaüzenet jelenik meg.



17.2. ábra

Néhány adatbázisrendszer figyelmeztetést küld, ha a DELETE utasítás végrehajtása hibát fog okozni

Láthatjuk, hogy „kulcs mégsértése" (key violations) miatt a táblában szereplő 34 rekord közül 32 nem törölhető. Mintha a program ezt közölné velünk: „Hé, fajankó, a Bowlers_Scores táblában még szerepelnek azokra a tekejátékosokra vonatkozó sorok, akiket törölni akarsz!" Ha most a No (Nem) gombra kattintunk, és az adatbázisrendszer a nevünkben végrehajtja a ROLLBACK utasítást, egyetlen sor sem törlődik. A Yes (Igen) gombra kattintva az adatbázisrendszer végrehajtja a COMMIT utasítást, hogy végleg törölje az egyetlen ponttal sem rendelkező tekejátékosokra vonatkozó két sort.

A fejezet későbbi, *Példák* című részében bemutatjuk, hogyan lehet kétféleképpen biztonságosan törölni azokat a tekejátékosokat, akik egyetlen játékban sem vettek részt – ha tényleg ezt akarjuk tenni.

Csak bizonyos sorok törlése

Általában korlátozni kívánjuk a törlendő sorok számát. Ezt egy WHERE záradék hozzáadásával tehetjük meg, ami kiszűri a kifejezetten törölni kívánt sorokat. A WHERE záradékok olyan egyszerűek, illetve összetettek lehetnek, mint amilyenekről már szó esett a SELECT vagy az UPDATE utasítások kapcsán.

Egyszerű WHERE záradék alkalmazása

Kezdjük valami egyszerűvel! Tegyük fel, hogy a Sales Orders adatbázisból törölni akarjuk azokat a rendeléseket, amelyekhez nem tartozik rendelési végösszeg. A kérelem a következőképpen fog kinézni:

„Töröld azokat a rendeléseket, amelyeknek a végösszege nulla!"

Fordítás Delete from the orders table
where the order total is zero
(Töröld azokat a rendeléseket, amelyeknél a rendelési végösszeg nulla.)

Tisztázás Delete from ~~the~~ orders ~~table~~
where ~~the~~ order total ~~is~~ = ~~zero~~ 0

SQL
```
DELETE FROM Orders
WHERE OrderTotal = 0
```

A WHERE záradék egy egyszerű összehasonlító állítás segítségével keresi meg azokat a sorokat, amelyeknek a rendelési végösszege nullával egyenlő. Ha a mintaadatbázisban végrehajtjuk ezt a lekérdezést, azt tapasztaljuk, hogy 11 sor törlődik. A lekérdezés mentett változatát CH17_Delete_Zero_OrdersA néven találhatjuk meg.

Első a biztonság: a megfelelő sorok törlésének biztosítása

Még az egyszerű DELETE lekérdezések esetében is nagyon ajánlatos ellenőrizni, hogy a megfelelő sorok fognak-e törlődni – de hogyan? Mint már említettük, a törölni kívánt sorok részhalmazának kijelölése a legtöbb esetben egy WHERE záradékkal történik. Miért nem szerkesztünk először egy SELECT lekérdezést, hogy visszaadjuk az eltávolítani kívánt sorokat?

„Sorold fel az Orders tábla minden oszlopát azoknak a rendeléseknek az esetében, amelyeknek a rendelési végösszege nulla!"

Fordítás
Select all columns
from the orders table
where the order total is zero
(Válaszd ki minden oszlopot a rendelések táblájából, ha a rendelés végösszege nulla.)

Tisztázás
Select ~~all columns~~ *
from ~~the~~ orders ~~table~~
where ~~the~~ order total ~~is~~ = ~~zero~~ 0

SQL
```
SELECT *
FROM Orders
WHERE OrderTotal = 0
```

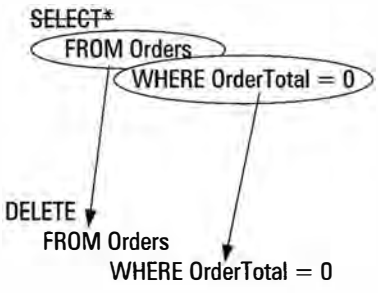
OrderNumbe	OrderDate	ShipDate	CustomerID	EmployeeID	OrderTotal
198	2007-10-07	2007-10-09	1002	703	$0.00
216	2007-10-11	2007-10-11	1016	707	$0.00
305	2007-10-31	2007-11-04	1013	708	$0.00
361	2007-11-11	2007-11-12	1016	706	$0.00
484	2007-12-08	2007-12-09	1021	707	$0.00
523	2007-12-14	2007-12-16	1003	704	$0.00
629	2008-01-07	2008-01-11	1014	704	$0.00
632	2008-01-07	2008-01-11	1001	706	$0.00
689	2008-01-14	2008-01-15	1015	705	$0.00
753	2008-01-27	2008-01-29	1013	701	$0.00
816	2008-02-08	2008-02-11	1011	701	$0.00

17.3. ábra

A törölni kívánt sorok ellenőrzése

Ha a Sales Orders mintaadatbázison futtatjuk ezt a lekérdezést, az eredmény a 17.3. ábrához hasonlóan fog kinézni.

Figyeljük meg, hogy a * helyettesítő karakterrel jeleztük, hogy látni akarjuk az összes oszlopot. Ha az eredményhalmaz tartalmazza az összes törölni kívánt sort, a SELECT utasítást úgy alakítjuk át a megfelelő DELETE utasítássá, hogy a SELECT *-ot egyszerűen DELETE-re cseréljük. A 17.4. ábra megmutatja, hogyan alakítsuk át a SELECT utasítást megfelelő formájú DELETE utasítássá.



17.4. ábra

Ellenőrző SELECT lekérdezés átalakítása DELETE utasítássá

Ez az átalakítás annyira egyszerű, hogy badarság lenne először nem létrehozni a SELECT utasítást, hogy meggyőződhessünk róla, hogy tényleg az általunk kívánt sorok törlődnek majd. Ne feledjük, hogy amennyiben a DELETE utasítást nem védtük meg egy tranzakcióba helyezve, az utasítás végrehajtása után a sorok végleg eltűnnek.

Allekérdezés alkalmazása

Az előző részben tárgyalt lekérdezés, amellyel törölni lehet az összes nulla rendelési végösszeggel rendelkező rendelést, meglehetősen egyszerűnek tűnik. Ne tévesszük viszont szem elől, hogy az OrderTotal oszlop számított értékeket tartalmaz. (A 15. fejezetben már bemutattuk, hogyan kell az UPDATE utasítás segítségével kiszámítani és beállítani a végösszeget.) Mi történne akkor, ha a felhasználó vagy az alkalmazás nem futtatná a frissítést, miután a rendelés részleteire vonatkozó egy vagy több sort felvette, törölte vagy módosította? Lehet, hogy az egyszerű lekérdezés megpróbál törölni egy olyan rendelést, amelyhez még tartoznak sorok az Order_Details (Rendelések részletei) táblában, és az is lehet, hogy kihagy néhány rendelést, amelynek minden részletét eltávolította, de a végösszegét nem frissítette.

Ha biztonságosabb módon akarjuk garantálni, hogy valóban olyan rendeléseket törlünk, amelyekhez nem tartoznak rendelési részletek, egy allekérdezéssel ellenőrizhetjük, hogy vannak-e illeszkedő sorok az Order_Details táblában. A kérelem a következőképpen fog kinézni.

„Törölj minden rendelést, amelynél nem szerepel rendelési tétel!"

Fordítás Delete the rows from the orders table
where the order number is not in
the selection of the order number
from the order details table
(Töröld azokat a sorokat a rendelések táblájából, ahol a rendelési szám nem szerepel a rendelések részleteit tartalmazó tábla kiválasztott rendelési számai között.)

Tisztázás Delete ~~the rows~~ from ~~the~~ orders ~~table~~
where ~~the~~ order number ~~is~~ not in
~~the~~ (selection of ~~the~~ order number
from ~~the~~ order details) ~~table~~

SQL
```
DELETE FROM Orders
WHERE OrderNumber NOT IN
(SELECT OrderNumber
FROM Order_Details)
```

Ez egy kicsit bonyolultabb, mint egy egyszerű összehasonlítás a nullával egyenlő rendelési végösszeg esetében, de biztosítja, hogy csak azok a rendelések törlődjenek, amelyekhez nem tartoznak sorok az Orders_Details táblában. A lekérdezés mentett változatát a mintaadatbázisban CH17_Delete_Zero_OrdersB néven találhatjuk meg. Ez az összetettebb lekérdezés lehet, hogy talál és töröl olyan sorokat, amelyeknek a nullától különböző rendelési végösszege nem lett helyesen módosítva, amikor az utolsó rendelési tétel is törlődött.

Ha WHERE záradékot szerkesztünk egy DELETE lekérdezéshez, valószínűleg elég gyakran fogjuk használni az IN, a NOT IN, az EXISTS vagy a NOT EXISTS kulcsszavakat. (Ha fel szeretnénk frissíteni az emékeinket, olvassuk el újra a 11. fejezetet.) Vegyünk szemügyre még egy példát, amelyben a törölni kívánt sorok kiszűréséhez összetett WHERE záradékra van szükség:

„Törölj minden archív táblába másolt, 2008. január 1-je előtt leadott rendelést és rendelési részletet!"

Emlékezzünk vissza a 16. fejezetre, amelyben láttuk, hogyan kell az INSERT utasítással régi sorokat átmásolni egy vagy több archív táblába. Miután átmásoltuk a sorokat, a feldolgozás folyamatát az alkalmazás fő részében gyakran hatékonyabbá tehetjük azáltal, hogy töröljük az archív sorokat. Amint arra a kérelem is utal, két táblából is kell sorokat törölnünk, ezért bontsuk fel a kérelmet két részre. Először az Order_Details táblából kell törölnünk, mert egy konkrét hivatkozási épségi szabály nem engedi, hogy sorokat töröljünk az Orders táblából, ha illeszkedő sorok találhatók az Orders_Details táblában:

„Töröld minden archív táblába másolt, 2008. január 1-je előtt leadott rendelés rendelési részleteit!"

Látjuk, hogy miért veszélyes ez? A probléma egyik megoldása az lenne, ha egyszerűen törölnénk a sorokat azokból a rendelésekből, amelyeket 2008. január 1-je előtt adtak le:

Fordítás
Delete rows from the order details table
where the order number is in
the selection of the order number
from the orders table
where the order date
is earlier than January 1, 2008
(Töröld azokat a sorokat a rendelések részleteit tartalmazó táblából,
amelyeknek a rendelési száma szerepel abban a listában, amelyet
a rendelések táblájából a 2008. január 1-je előtti rendelési dátumú
rendelések kiválasztásával nyertünk.)

Tisztázás
Delete ~~rows~~ from ~~the~~ order details ~~table~~
where ~~the~~ order number ~~is~~ in
~~the~~ (selection ~~of the~~ order number
from ~~the~~ orders ~~table~~
where ~~the~~ order date
~~is earlier than~~ < ~~January 1, 2008~~ '2008-01-01')

SQL
```
DELETE FROM Order_Details
WHERE OrderNumber IN
(SELECT OrderNumber
FROM Orders
WHERE OrderDate
< '2008-01-01')
```

A lekérdezés mentett változatát CH17_Delete_Archived_Order_Details_Unsafe néven találhatjuk meg. Mi a helyzet akkor, ha valaki megígérte, hogy a sorok archiválásának céljából futtatja az INSERT lekérdezést, de ezt nem tette meg? Ha futtatjuk ezt a lekérdezést, minden 2008. január 1-je előtt leadott rendelés részleteit töröljük, tekintet nélkül arra, hogy a sorok valóban léteznek-e az archív táblában. Biztonságosabb megoldás, ha csak azokat a sorokat töröljük, amelyekről meggyőződtünk, hogy megtalálhatók az archív táblában. Próbáljuk meg még egyszer:

Fordítás
Delete rows from the order details table
where the order number is in
the selection of order number
from the order details archive table
(Töröld azokat a sorokat a rendelések részleteit tartalmazó táblából,
amelyeknek a rendelési száma szerepel a rendelési részletek archív
táblájából kiválasztott rendelési számok között.)

Tisztázás
Delete ~~rows~~ from ~~the~~ order details ~~table~~
where ~~the~~ order number ~~is~~ in

~~the~~ (select~~ion of~~ order number
from ~~the~~ order details archive) ~~table~~

SQL
```
DELETE FROM OrderDetails
WHERE OrderNumber IN
(SELECT OrderNumber
FROM Order_Details_Archive)
```

Ennek a lekérdezésnek a mentett változatát CH17_Delete_Archived_Order_Details_OK néven találhatjuk meg. Figyeljük meg, hogy a lekérdezés nem foglalkozik a rendelés dátumával, ugyanakkor ez a megoldás sokkal biztonságosabb, mert csak azokat a sorokat törli a fő táblából, amelyekhez tartozik rendelési szám az archív táblában. Ha biztosak akarunk lenni afelől, hogy azoknak a 2008. január 1-je előtti rendeléseknek a sorait töröljük, amelyek már az archív táblában vannak, a lekérdezésben használjuk mindkét IN műveletet, az AND logikai művelettel párosítva.

A DELETE használati területei

Most már tisztában kell lennünk azzal, hogyan lehet egy vagy több sort törölni egy táblából – akár az összes sort, akár a WHERE záradék által meghatározott sorokat. A legjobb módja annak, hogy megismerkedjünk a DELETE utasítás széles körű alkalmazási területeivel, hogy felsorolunk néhány problémát, amelyeket a fenti utasítással lehet megoldani, majd a *Példák* részben bemutatunk több példát is. Az alábbiakban látható egy kis ízelítő az olyan típusú problémákból, amelyeket a DELETE utasítással lehet megoldani:

„Töröld azokat a termékeket, amelyekre soha nem adtak le rendelést!"
„Törölj minden olyan előadót, akit soha nem szerződtettek le!"
„Töröld azokat a tekejátékosokat, akik egyetlen mérkőzést sem játszottak!"
„Törölj minden olyan hallgatót, aki nem iratkozott fel egyetlen órára sem!"
„Törölj minden olyan termékcsoportot, amelyhez nem tartozik termék!"
„Töröld azokat a megrendelőket, akik még soha nem kötöttek le egyetlen előadót sem!"
„Töröld azokat a csapatokat, amelyekben egyetlen játékos sem található!"
„Törölj minden olyan órát, amelyre nem iratkozott fel egyetlen hallgató sem!"
„Töröld azokat a vásárlókat, akik soha nem adtak le rendelést!"
„Töröld azokat a zenei stílusokat, amelyekben egyetlen előadó sem játszik!"
„Törölj minden olyan tekemérkőzést, amelyet még nem játszottak le!"
„Töröld azokat a tantárgyakat, amelyekhez nem tartozik óra!"
„Törölj minden olyan rendezvényt, amelyet már az archív táblába másoltunk!"
„Törölj minden olyan tornaadatot, amelyet már az archív táblába másoltunk!"
„Törölj minden olyan beszállítót, aki nem kínál egyetlen terméket sem!"
„Töröld azokat a tagokat (előadókat), akik nem tartoznak egyetlen együtteshez (előadócsoporthoz) sem!"
„Töröld azokat az alkalmazottakat, akik semmit sem adtak el!"

Példák

Most már ismerjük a DELETE lekérdezések felépítésének módját, lássunk hát pár példát, amelyek közül mindegyik egy vagy több sor törlését kéri egy táblából. A példák a négy mintaadatbázisból származnak.

A példákhoz azokat az adatokat is mellékeltük, amelyeket a DELETE utasításnak el kell távolítania a céltáblából, és feltüntettük a törlendő sorok számát is. A törlendő sorok számát jelző szám előtt közvetlenül megjelenő név a lekérdezés neve, amelyen az a könyv CD-mellékletén szerepel. Minden lekérdezést a megfelelő adatbázisba tettünk (amint a példánál ezt jelöltük is); az ehhez a fejezethez tartozó lekérdezések neve „CH17"-tel kezdődik. A példákat a könyv elején található bevezetés útmutatását követve tölthetjük be és próbálhatjuk ki.

Megjegyzés *Ne feledjük, hogy a példákban használt minden tábla- és oszlopnév megfelel a mintaadatbázisok B függelékben található sémaleírásának. A leírások egyszerűsítése céljából a Fordítás és Tisztázás részeket minden példánál összevontuk. A példák feltételezik, hogy az olvasó áttanulmányozta és megértette a korábbi fejezetekben leírt ismereteket.*

Sales Orders adatbázis

„Töröld azokat a vásárlókat, akik soha nem adtak le rendelést!"

Fordítás/ Tisztázás
Delete ~~rows~~ from ~~the~~ customers ~~table~~
where ~~the~~ customer ID ~~is~~ not in
~~the~~ (selection of ~~the~~ customer ID
from ~~the~~ orders) ~~table~~

SQL

```
DELETE FROM Customers
WHERE CustomerID NOT IN
(SELECT CustomerID
FROM Orders)
```

Sor törlése a Customers táblából a CH17_Delete_Customers_Never_Ordered lekérdezéssel (1 sor törlése)

CustomerID	CustFirst Name	CustLast Name	CustStreet Address	CustCity	CustState	CustZip Code	CustArea Code	Custphone Number
1028	Jeffrey	Tirekicker	15622 NE 42nd Ct	Redmond	WA	98052	425	555-9999

„Törölj minden olyan beszállítót, aki nem kínál egyetlen terméket sem!"

Fordítás/ Tisztázás
Delete ~~rows~~ from ~~the~~ vendors ~~table~~
where ~~the~~ vendor ID ~~is~~ not in
~~the~~ (selection of vendor ID
from ~~the~~ product vendors) ~~table~~

```
SQL     DELETE FROM Vendors
        WHERE VendorID NOT IN
        (SELECT VendorID
        FROM Product_Vendors)
```

Sor törlése a Vendors táblából a CH17_Delete_Vendors_No_Products lekérdezéssel (1 sor törlése)

VendorID	VendName	VendStreet Address	VendCity	VendState	VendZip Code	<<további oszlopok>>
11	Astro Paper Products	5639 N. Riverside	Chicago	IL	60637	...

Entertainment Agency adatbázis

„Törölj minden olyan előadót, akit soha nem szerződtettek le!"

Fordítás 1/ Delete ~~rows~~ from ~~the~~ entertainer members ~~table~~
Tisztázás where ~~the~~ entertainer ID ~~is~~ not in
~~the~~ (selection ~~of~~ entertainer ID
from ~~the~~ engagements) ~~table~~

```
SQL     DELETE FROM Entertainer_Members
        WHERE EntertainerID NOT IN
        (SELECT EntertainerID
        FROM Engagements)
```

Sor törlése az Entertainer_Members táblából a CH17_Delete_Entertainers_Not_Booked1 lekérdezéssel (1 sor törlése)

EntertainerID	MemberID	Status
1009	121	2

Fordítás 2/ Delete ~~rows~~ from ~~the~~ entertainers ~~table~~
Tisztázás where ~~the~~ entertainer ID ~~is~~ not in
~~the~~ (selection ~~of~~ entertainer ID
from ~~the~~ engagements) ~~table~~

```
SQL     DELETE FROM Entertainers
        WHERE EntertainerID NOT IN
        (SELECT EntertainerID
        FROM Engagements)
```

Sor törlése az Entertainers táblából a CH17_Delete_Entertainers_Not_Booked2 lekérdezéssel (1 sor törlése)

EntertainerID	EntStage Name	EntSSN	EntStreet Address	EntCity	EntState	EntZipCode	<<további oszlopok>>
1009	Katherine Ehrlich	888-61-1103	777 Fenexet Blvd	Woodin-ville	WA	98072	...

„Törölj minden olyan rendezvényt, amelyet már az archív táblába másoltunk!"

Fordítás/ Tisztázás
Delete ~~rows~~ from ~~the~~ engagements ~~table~~
where ~~the~~ engagement ID ~~is~~ in
~~the~~ (select~~ion of~~ engagement ID
from ~~the~~ engagements archive) ~~table~~

SQL

```
DELETE FROM Engagements
WHERE EngagementID IN
(SELECT EngagementID
FROM Engagements_Archive)
```

Megjegyzés *Annak érdekében, hogy megtaláljuk a törölni kívánt sorokat, először futtatnunk kell a CH16_Archive_Engagements lekérdezést, hogy átmásoljuk az adatokat az archív táblába. Az eredeti mintaadatbázisban szereplő archív tábla üres.*

Sorok törlése az Engagements táblából a CH17_Remove_Archived_Engagements lekérdezéssel (56 sor törlődik, ha először lefuttatjuk a CH16_Archive_Engagements lekérdezést)

Engagement Number	StartDate	EndDate	StartTime	StopTime	Contract Price	CustomerID	AgentID	EntertainerID
2	2007-09-01	2007-09-05	13:00	15:00	$200.00	10006	4	1004
3	2007-09-10	2007-09-15	13:00	15:00	$590.00	10001	3	1005
4	2007-09-11	2007-09-17	20:00	0:00	$470.00	10007	3	1004
5	2007-09-11	2007-09-14	16:00	19:00	$1,130.00	10006	5	1003
6	2007-09-10	2007-09-14	16:00	15:00	$2,300.00	10014	7	1008
<< további sorok >>								

School Scheduling adatbázis

„Törölj minden olyan órát, amelyre nem iratkozott fel egyetlen hallgató sem!"

Megjegyzés *A sorokat először a Faculty_Classes, majd a Classes táblából töröljük, mivel az adatbázis egy épségi szabály miatt nem engedi, hogy sorokat töröljünk a Classes táblából, ha illeszkedő sorok találhatók a Faculty_Classes táblában.*

Fordítás 1/ Delete from ~~the~~ faculty classes ~~table~~
Tisztázás where ~~the~~ class ID ~~is~~ not in
~~the~~ (select~~ion of~~ class ID
from ~~the~~ student schedules) ~~table~~

SQL
```
DELETE FROM Faculty_Classes
WHERE ClassID NOT IN
(SELECT ClassID
FROM Student_Classes)
```

Sorok törlése a Faculty_Classes táblából a CH17_Delete_Classes_No_Students_1 lekérdezéssel (60 sor törlése)

ClassID	StaffID
1002	98036
1004	98019
1006	98045
1020	98028
1030	98036
1156	98055
1162	98064
1183	98005
1184	98011
<< további sorok >>	

Fordítás 2/ Delete from ~~the~~ classes ~~table~~
Tisztázás where ~~the~~ class ID ~~is~~ not in
~~the~~ (select~~ion of~~ class ID
from ~~the~~ student schedules) ~~table~~

SQL
```
DELETE FROM Classes
WHERE ClassID NOT IN
(SELECT ClassID
FROM Student_Schedules)
```

Sorok törlése a Classes táblából a CH17_Delete_Classes_No_Students_2 lekérdezéssel (61 sor törlése)

ClassID	SubjectID	ClassRoomID	Credits	StartTime	Duration	MondaySchedule	<< *további oszlopok* >>
1002	12	1619	4	15:30	110	Yes	...
1004	13	1627	4	8:00	50	Yes	...
1006	13	1627	4	9:00	110	Yes	...
1020	15	3404	4	13:00	110	Yes	...
1030	16	1231	5	11:00	50	Yes	...
1156	37	3443	5	8:00	50	Yes	...
<< *további sorok* >>							

Bowling League adatbázis

„Töröld azokat a tekejátékosokat, akik egyetlen mérkőzést sem játszottak!"

Megjegyzés *A fenti lekérdezést úgy oldhatjuk meg, ha töröljük azokat a tekejátékosokat, akik nulla számú mérkőzésen vettek részt, illetve ha töröljük azokat, akikhez nem tartozik sor a Bowler_Scores táblában. A második megoldás biztonságosabb, mivel nem függ a lejátszott mérkőzések számított értékétől, de oldjuk meg a lekérdezést mind a kétféleképpen.*

Fordítás 1/ Delete ~~rows~~ from ~~the~~ bowlers ~~table~~
Tisztázás where ~~the~~ bowler games bowled ~~is~~ = ~~zero~~ 0
SQL

```
DELETE FROM Bowlers
WHERE BowlerGamesBowled = 0
```

Sorok törlése a Bowlers táblából a CH17_Delete_Bowlers_No_Games lekérdezéssel (2 sor törlése)

BowlerID	BowlerLast Name	BowlerFirst Name	<< *további oszlopok* >>	Bowler Games Bowled	Bowler Current Average	Bowler Current Hcp
33	Patterson	Kerry	...	0	0	0
34	Patterson	Maria	...	0	0	0

Fordítás 2/ Delete ~~rows~~ from ~~the~~ bowlers ~~table~~
Tisztázás where ~~the~~ bowler ID ~~is~~ not in
~~the~~ (select~~ion of~~ bowler ID
from ~~the~~ bowler scores) ~~table~~
SQL

```
DELETE FROM Bowlers
WHERE BowlerID NOT IN
(SELECT BowlerID
FROM Bowler_Scores)
```

Sorok törlése a Bowlers táblából a CH17_Delete_Bowlers_No_Games_Safe lekérdezéssel (2 sor törlése)

BowlerID	BowlerLast Name	BowlerFirst Name	<< *további oszlopok* >>	Bowler Games Bowled	Bowler Current Average	Bowler Current Hcp
33	Patterson	Kerry	...	0	0	0
34	Patterson	Maria	...	0	0	0

„Töröld azokat a csapatokat, amelyekben egyetlen játékos sem található!"

Fordítás/ Delete from ~~the~~ teams ~~table~~
Tisztázás where ~~the~~ team ID ~~is~~ not in
~~the~~ (select~~ion of~~ team ID
from ~~the~~ bowlers) ~~table~~

SQL
```
DELETE FROM Teams
WHERE TeamID NOT IN
(SELECT TeamID
FROM Bowlers)
```

Sorok törlése a Bowlers táblából a CH17_Delete_T lekérdezéssel (2 sor törlése)

TeamID	TeamName	CaptainID
9	Huckleberrys	7
10	Never Show Ups	22

Összefoglalás

A fejezetet a táblákból sorok törlésére használható DELETE utasítás rövid ismertetésével kezdtük. Bemutattuk a DELETE utasításformáját, és egy egyszerű példán keresztül elmagyaráztuk, hogyan kell egy táblából minden sort törölni. Röviden áttekintettük a tranzakciókat, és bemutattuk, hogy a Microsoft Access adatbázisrendszer hogyan segít védekezni a hibák ellen az egyes tranzakciók során.

A továbbiakban ismertettük a törölni kívánt sorok számának korlátozására szolgáló WHERE záradék használatát. Elmagyaráztuk, hogyan kell a SELECT utasítást a törlendő sorok ellenőrzésére használni, és hogyan kell ezt az utasítást DELETE utasítássá alakítani.

Végezetül alaposan körüljártuk a törlendő sorok ellenőrzésére szolgáló allekérdezések használatát, más táblákban létező vagy nem létező kapcsolódó sorokra alapozva, majd példákon keresztül bemutattuk a DELETE lekérdezések felépítését.

A fejezet befejező részében néhány olyan kérdést teszünk fel, amelyekre önállóan kell választ keresnünk.

Önálló feladatok

Az alábbiakban a lekérdezésként megfogalmazandó kérdések és utasítások után annak a lekérdezésnek a nevét láthatjuk a mintaadatbázisokból, amelyikben megtaláljuk a megoldást. Ha gyakorolni szeretnénk, kidolgozhatjuk az egyes kérdésekhez szükséges SQL kódot, majd ellenőrizhetjük a megoldást az adott mintaadatbázisokban található lekérdezésekkel. Amíg ugyanazt az eredményt kapjuk, ne aggódjunk, ha a saját SQL-utasításunk nem egyezik pontosan a mintáéval.

Sales Orders adatbázis

1. *„Töröld azokat a termékeket, amelyekre soha nem adtak le rendelést!"*
   (Tipp: először a Product_Vendors, majd a Products táblából kell törölnünk.)
   A megoldás itt található: CH17_Delete_Products_Never_Ordered_1 (4 sor törlése) és CH17_Delete_Products_Never_Ordered_2 (2 sor törlése).
2. *„Töröld azokat az alkalmazottakat, akik semmit sem adtak el!"*
   A megoldás itt található: CH17_Delete_Employees_No_Orders (1 sor törlése).
3. *„Törölj minden olyan termékcsoportot, amelyhez nem tartozik termék!"*
   A megoldás itt található: a CH17_Delete_Categories_No_Products (1 sor törlése).

Entertainment Agency adatbázis

1. *„Töröld azokat a megrendelőket, akik még soha nem kötöttek le egyetlen előadót sem!"*
   A megoldás itt található: CH17_Delete_Customers_Never_Booked (2 sor törlése).
2. *„Töröld azokat a zenei stílusokat, amelyekben egyetlen előadó sem játszik!"*
   A megoldás itt található: CH17_Delete_Styles_No_Entertainer (8 sor törlése).
3. *„Töröld azokat a tagokat (előadókat), akik nem tartoznak egyetlen együtteshez (előadócsoporthoz) sem!"*
   A megoldás itt található: a CH17_Delete_Members_Not_In_Group (nincs sortörlés).

School Scheduling adatbázis

1. *„Törölj minden olyan hallgatót, aki nem iratkozott fel egyetlen órára sem!"*
   A megoldás itt található: CH17_Delete_Students_No_Classes (1 sor törlése).
2. *„Töröld azokat a tantárgyakat, amelyekhez nem tartozik óra!"*
   (Tipp: mind a Faculty_Subjects, mind a Subjects táblából kell sorokat törölnünk.)
   A megoldás itt található: CH17_Delete_Subjects_No_Classes_1 (6 sor törlése) és CH17_Delete_Subjects_No_Classes_2 (3 sor törlése).

Bowling League adatbázis

1. *„Törölj minden olyan tornaadatot, amelyet már az archív táblába másoltunk!"*
   (Tipp: a Bowler_Scores, Match_Games, Tourney_Matches és Tournaments táblákból kell sorokat törölnünk. Egyetlen törlésre kijelölt sort sem szabad találnunk, hacsak nem hajtottuk végre a 16. fejezetben szereplő négy archív lekérdezést.)

A megoldás itt található: CH17_Delete_Archived_2007_Tournaments_1 (1344 sor törlése), CH17_Delete_Archived_2007_Tournaments_2 (168 sor törlése), CH17_Delete_Archived_2007_Tournaments_3 (57 sor törlése) és CH17_Delete_Archived_2007_Tournaments_4 (14 sor törlése).

2. *„Törölj minden olyan tekemérkőzést, amelyet még nem játszottak le!"*
   A megoldás itt található: a CH17_Delete_Matches_Not_played (1 sor törlése).

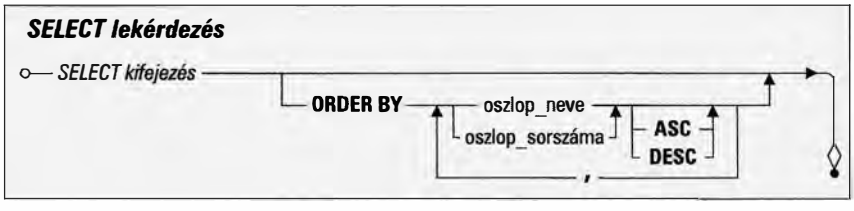
Függelékek

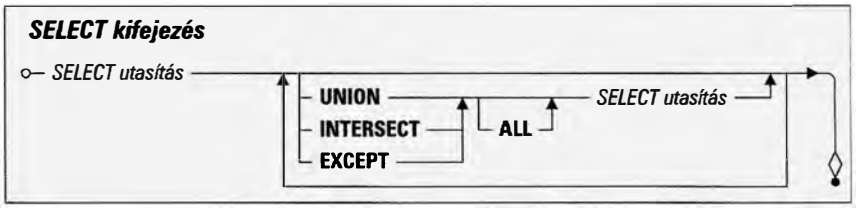
F

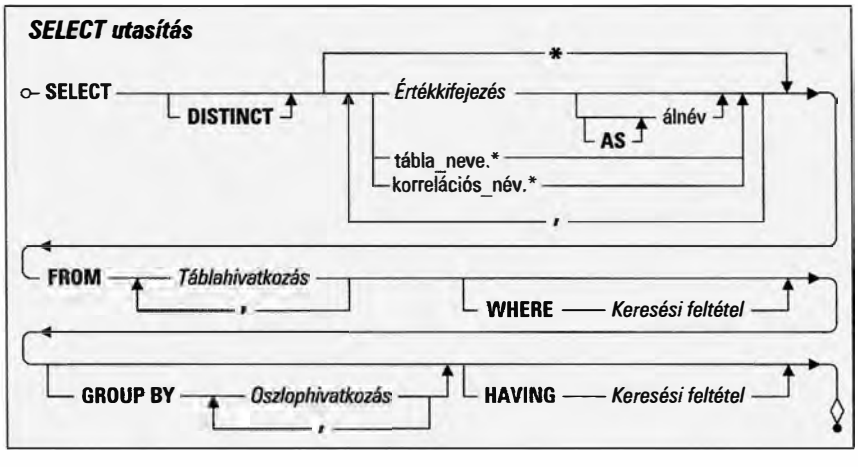
A

Az SQL-szabványnak megfelelő diagramok

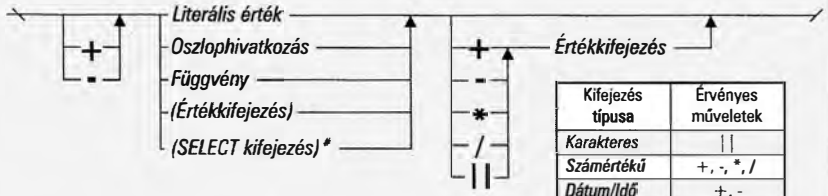
Az alábbiakban megtalálhatjuk a könyvben tárgyalt összes SQL-utasítás teljes szintaxisdiagramját.







Értékkifejezés

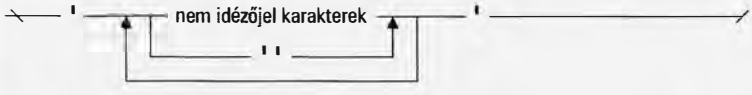


Csak skaláris érték

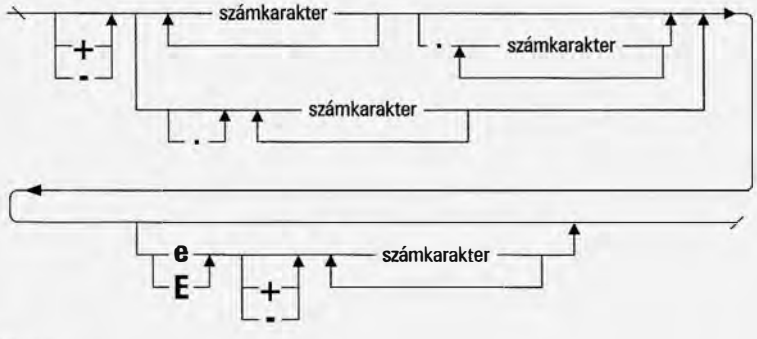
Kifejezés típusa	Érvényes műveletek
Karakteres	||
Számértékű	+, -, *, /
Dátum/Idő	+, -
Tartomány	+, -, *, /

Literális érték

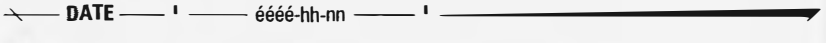
Karakterlánc



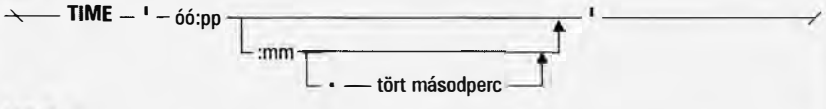
Számérték



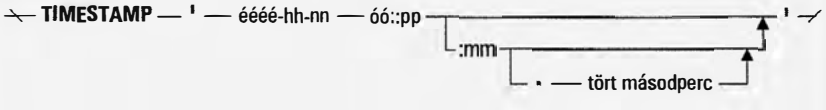
Dátum



Idő

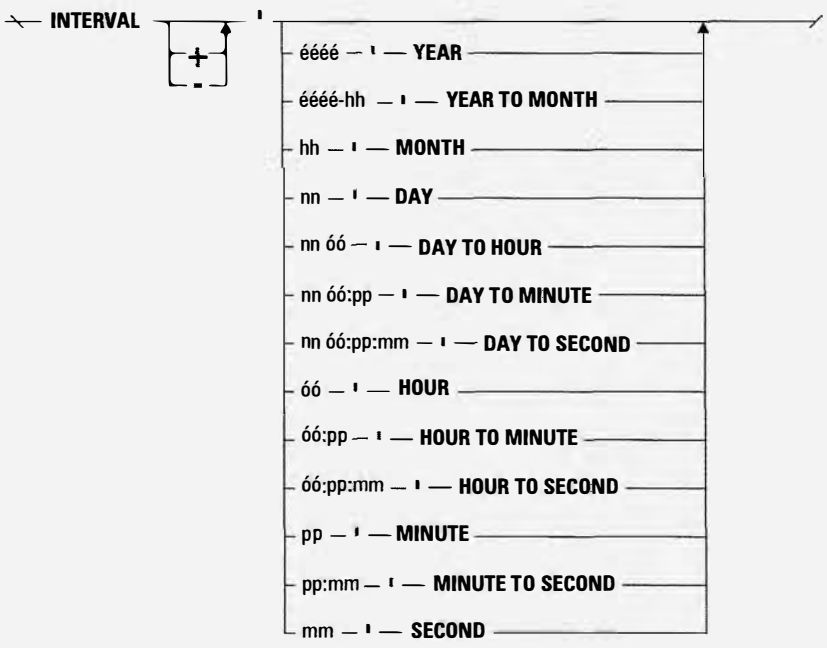


Időbélyeg



Literális érték (folytatás)

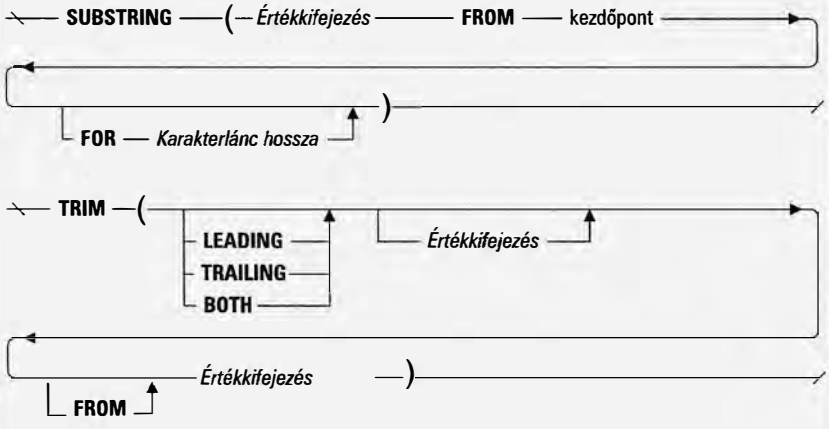
Tartomány



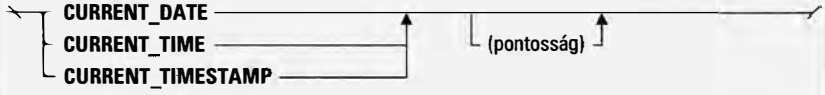
Oszlophivatkozás

Függvények

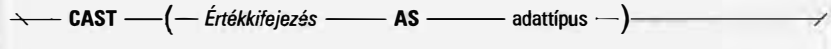
Karakterlánc



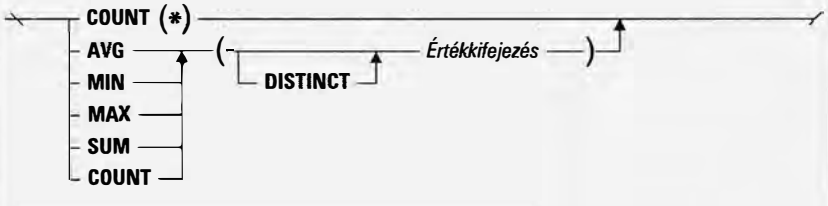
Dátum-idő



Átalakítás

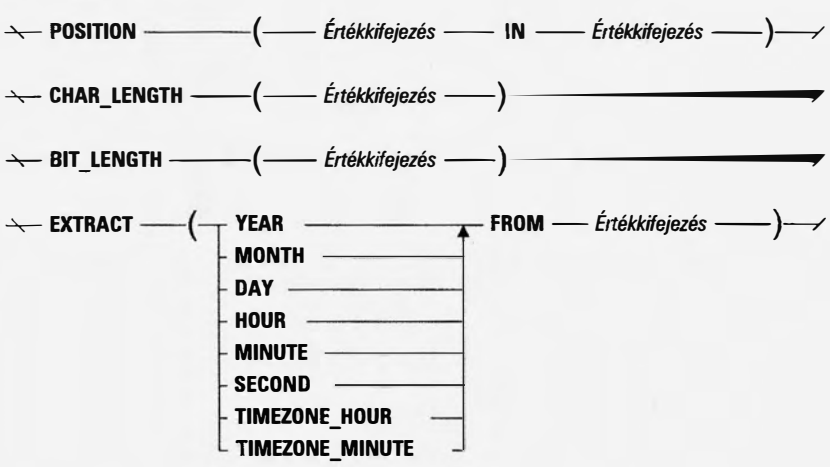


Összesítő

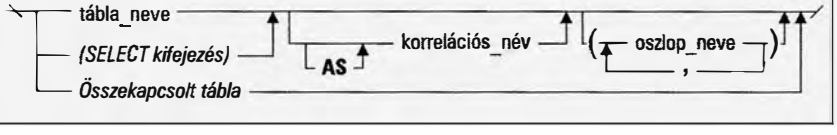


Függvények (folytatás)

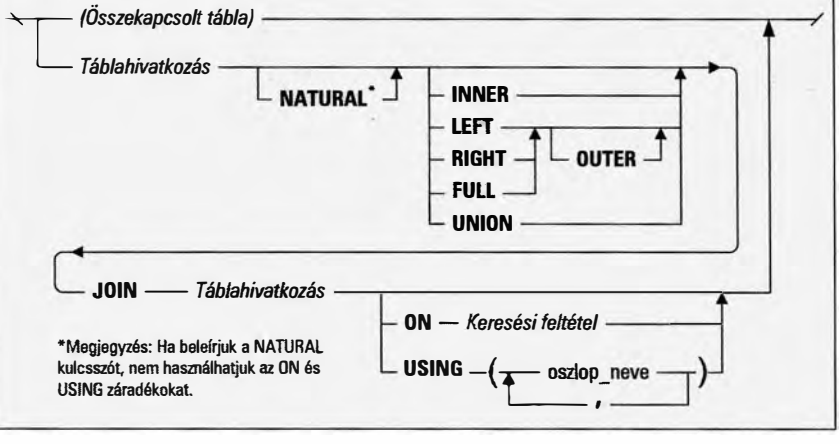
Számértékű



Táblahivatkozás

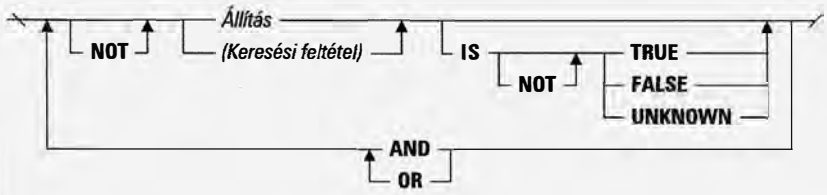


Összekapcsolt tábla



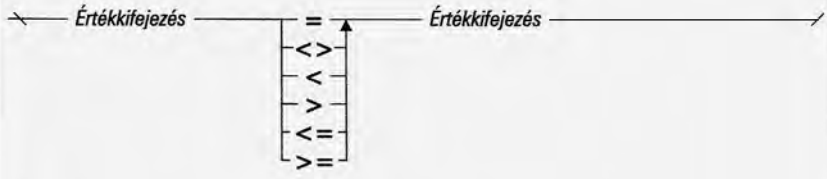
*Megjegyzés: Ha beleírjuk a NATURAL kulcsszót, nem használhatjuk az ON és USING záradékokat.

Keresési feltétel

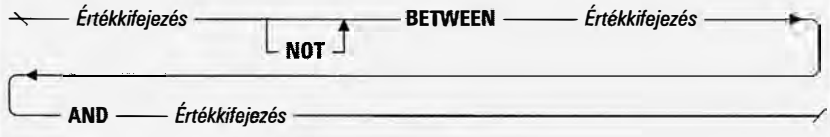


Állítás

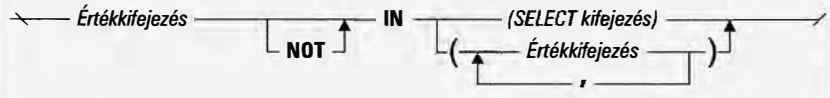
Összehasonlítás



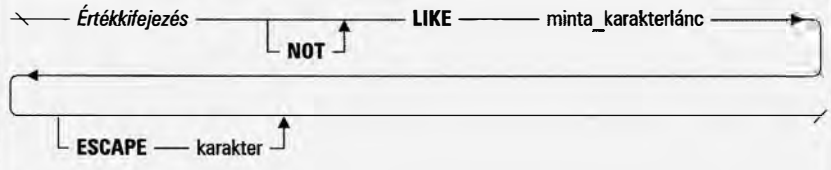
Tartomány



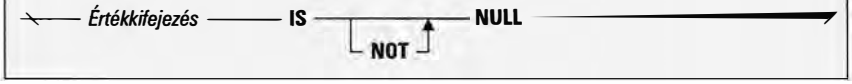
Halmaztagság



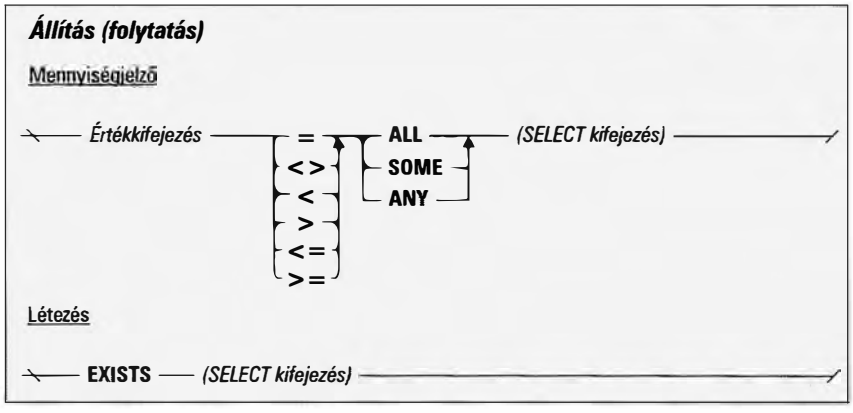
Mintaillesztés



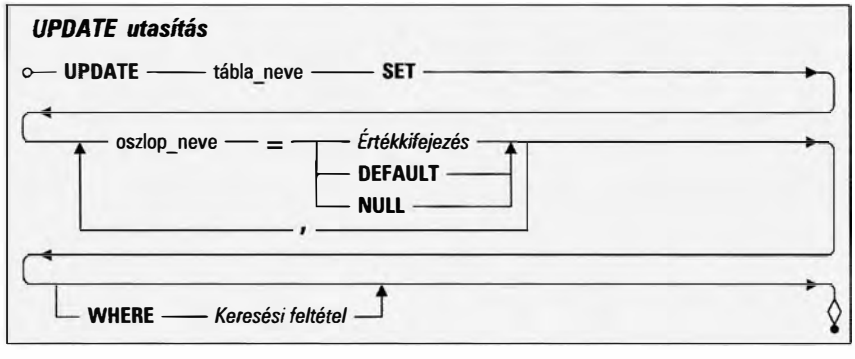
Null



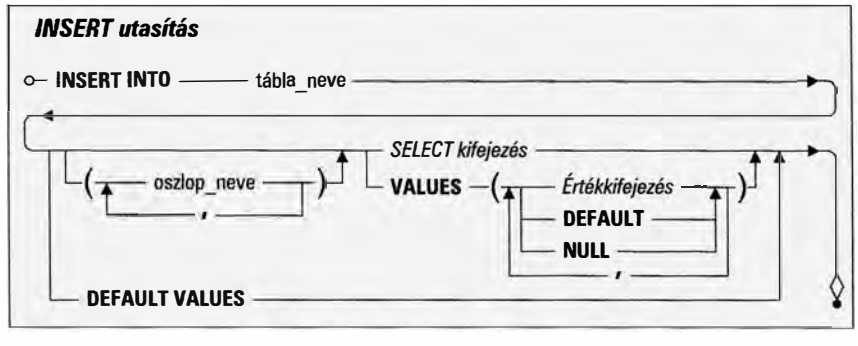
Állítás (folytatás)



UPDATE utasítás



INSERT utasítás

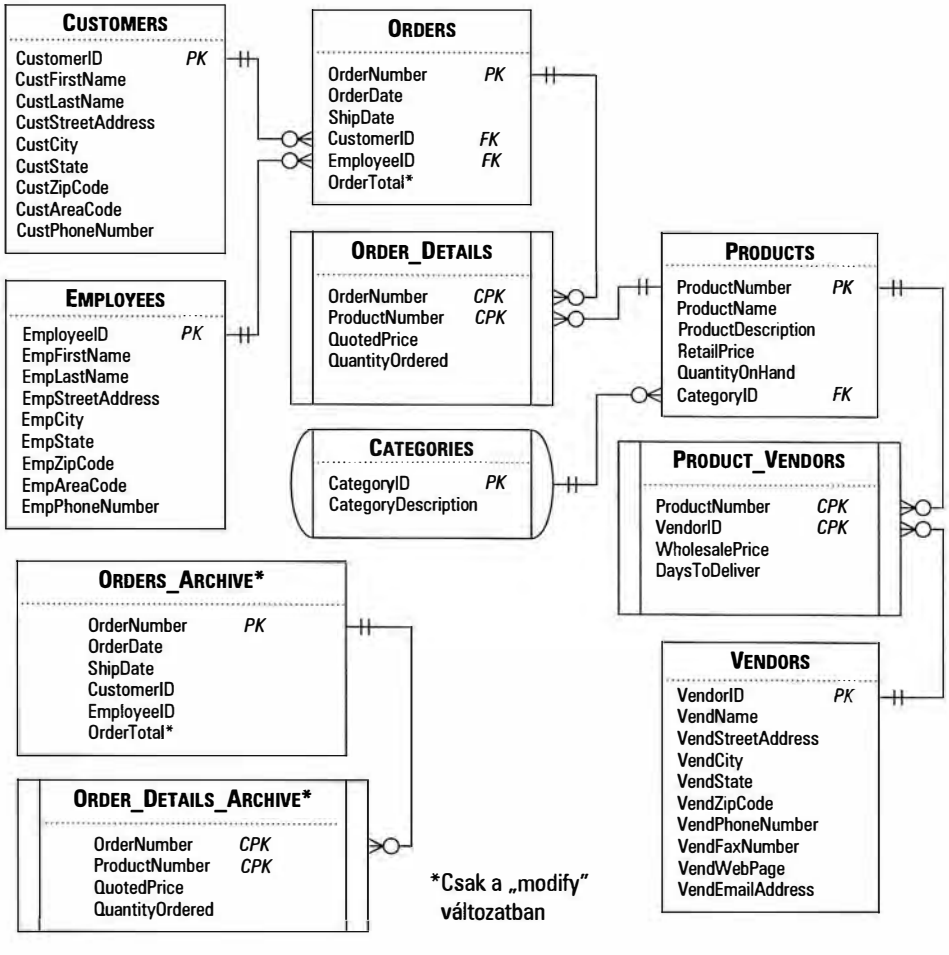


DELETE utasítás

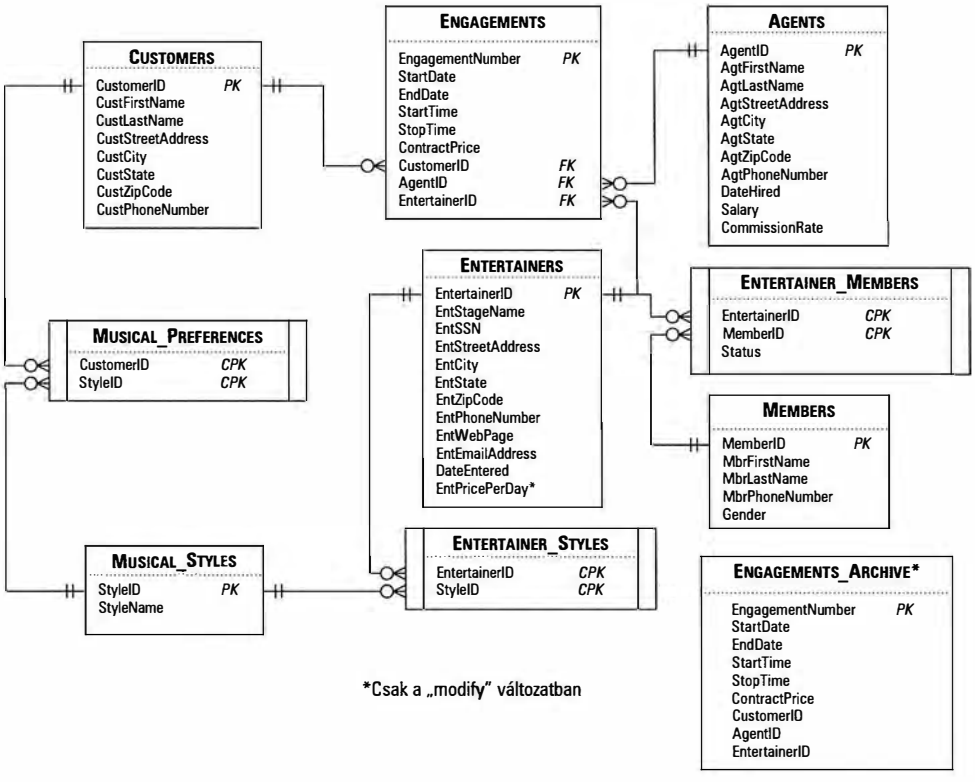
B

A mintaadatbázisok sémája

Sales Orders adatbázis

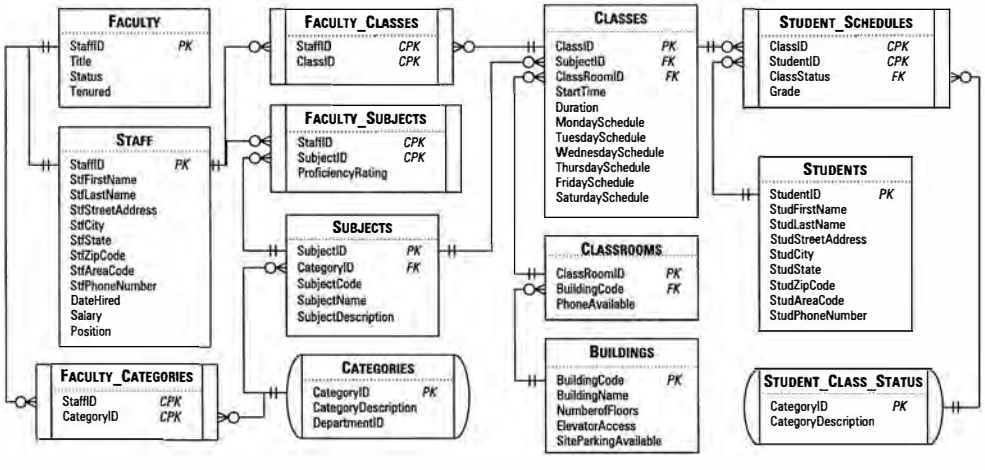


Entertainment Agency adatbázis

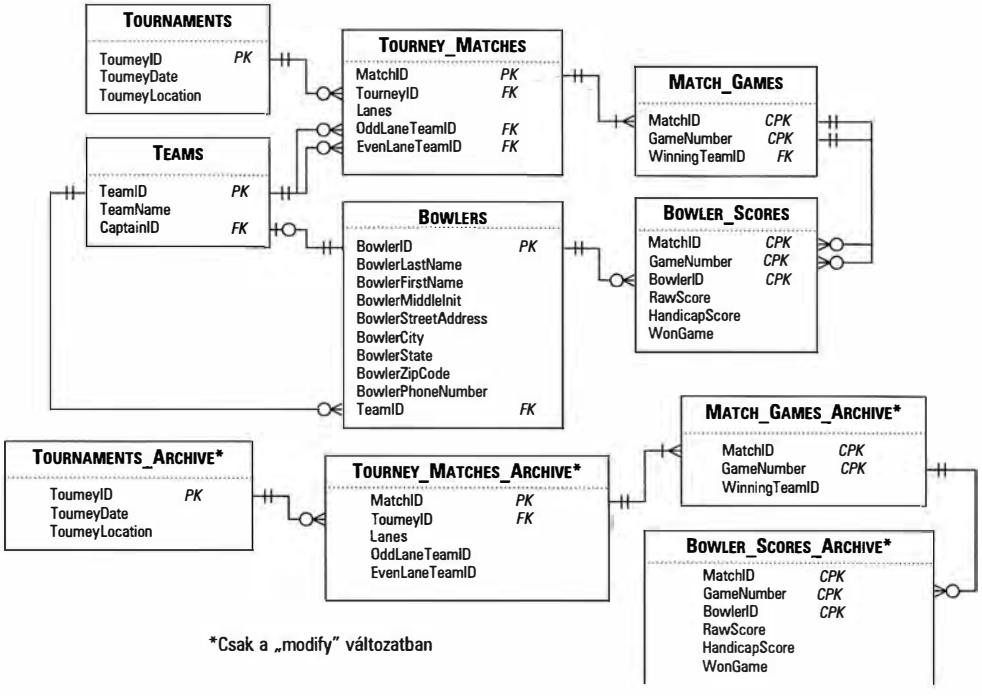


*Csak a „modify" változatban

School Scheduling adatbázis

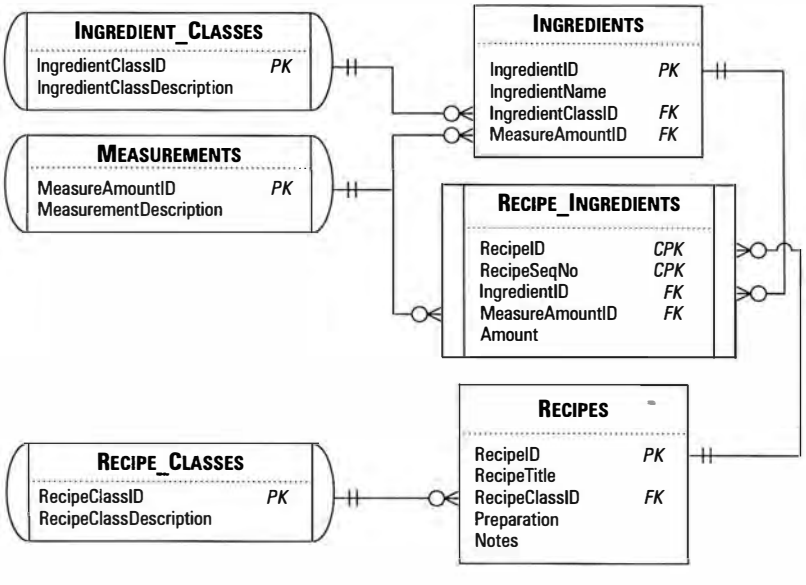


Bowling League adatbázis



*Csak a „modify" változatban

Recipes adatbázis



Dátum- és időfüggvények

Amint azt az 5. fejezetben említettük, az adatbázisrendszerek a legkülönfélébb függvényekkel rendelkeznek, amelyeknek a segítségével dátum- és időértékeket kérdezhetünk le, illetve módosíthatunk. Az SQL-szabvány három függvényt határoz meg: a CURRENT_DATE, CURRENT_TIME és CURRENT TIMESTAMP függvényeket, de nem minden kereskedelmi forgalomban lévő adatbázisrendszer támogatja mind a hármat. Hogy a saját adatbázisrendszerünkben könnyebb legyen dátum- és időértékekkel dolgozni, összeállítottunk egy a főbb adatbázisrendszerekre vonatkozó függvénylistát. A függelékben található listák a függvény nevét és rövid leírását tartalmazzák; az egyes függvények sajátos utasításformáját az adatbázis-kezelőnk dokumentációjából tudhatjuk meg.

IBM DB2

Függvénynév	Leírás
CURRENT DATE	Az aktuális dátumot olvassa ki.
CURRENT TIME	A helyi időzóna aktuális időértéket olvassa ki.
CURRENT TIMESTAMP	Az aktuális dátumot és időt adja vissza a helyi időzónából.
DATE(<*kifejezés*>)	Kiértékeli a kifejezést, és egy dátumot ad vissza.
DAY(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a dátumon, időbélyegen vagy dátumtartományon belüli napok számát.
DAYOFMONTH(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a dátumon vagy időbélyegen belüli napok számát.
DAYOFWEEK(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a nap héten belüli számát , ahol 1 = vasárnap.
DAYOFWEEK_ISO(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a nap héten belüli számát , ahol 1 = hétfő.
DAYOFYEAR(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a nap éven belüli számát.
DAYS(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a napok számát 0001. január 1-től, 1 napot hozzáadva.

Függvénynév	Leírás
HOUR(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az idő vagy időbélyeg óraértékét.
JULIAN_DAY(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a napok számát i.e. 4713. január 1-től a kifejezésben megadott dátumig.
LAST_DAY(<*kifejezés*>)	A kifejezésben szereplő dátum hónapjának legutolsó napját adja vissza.
MICROSECOND(<*kifejezés*>)	Kiszámítja a kifejezésben szereplő időbélyeg vagy időtartam értékét, és visszaadja annak mikroszekundum részét.
MIDNIGHT_SECONDS(<*kifejezés*>)	Kiszámítja az idő vagy időbélyeg értékét, és visszaadja a másodpercek számát éjfél és a kifejezésben megadott időpont között.
MINUTE(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az időpont, időbélyeg vagy időköz perc részét.
MONTH(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a dátumon, időbélyegen vagy dátumtartományon belüli hónap számát.
NEXTDAY(<*kifejezés*>,<*napnév*>)	Kiértékeli a kifejezést, és a kifejezésben szereplő dátumot követően időbélyeg formájában visszaadja a <*napnév*>-ben (ami egy „MON", „TUE" stb. rövidítést tartalmazó karakterlánc) megadott első nap dátumát.
QUARTER(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a kifejezésben szereplő dátumhoz tartozó negyedév számát.
ROUND_TIMESTAMP(<*kifejezés*>, <*formázó karakterlánc* >)	Kiértékeli a kifejezést, és a formázó karakterláncban megadott legközelebbi tartományra kerekíti.
SECOND(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az időpont, időbélyeg vagy időköz másodperc részét.
TIME(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az időpont- vagy időbélyegérték időpont részét.
TIMESTAMP(<*kifejezés1*>,<*kifejezés2*>)	Önálló dátum- (*kifejezés1*) és időpontértékeket (*kifejezés2*) időbélyeggé alakít.
TRUNC_TIMESTAMP(<*kifejezés*>, <*formázó karakterlánc* >)	Kiértékeli a kifejezést, és a formázó karakterláncban megadott legközelebbi tartományra csonkítja.
WEEK(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az értékben szereplő dátumon belüli hét számát. Az első hét január 1-vel kezdődik.

Függvénynév	Leírás
WEEK_ISO(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a kifejezésen belüli dátumban szereplő hét számát, ahol az első hét az év első csütörtökét tartalmazó hét.
YEAR(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a dátum vagy időbélyeg év részét.

Microsoft Office Access

Függvénynév	Leírás
CDate(<*kifejezés*>)	A kifejezést dátumértékké alakítja.
Date	Az aktuális dátumértéket olvassa ki.
DateAdd(<*tartomány*>, <*szám*>,<*kifejezés*>)	*Szám* mennyiségű dátum- vagy időtartományt ad a dátumhoz vagy dátum- és időpont kifejezéshez.
DateDiff(<*tartomány*>, <*kifejezés1*>, <*kifejezés2*>, <*hételsőnapja*>, <*évelsőnapja*>)	A *kifejezés1* és *kifejezés2* dátum- és időértékei közötti megadott számú tartományt adja vissza. Tetszés szerint megadhatjuk a hét első napját (az alapértelmezett a vasárnap), valamint az év első hetét, amely vagy január 1-jén kezdődik, és legalább négy napot tartalmaz, vagy az első teljes hét.
DatePart(<*tartomány*>, <*kifejezés*>, <*hételsőnapja*>, <*évelsőnapja* >)	A *kifejezés*-ből a dátumnak vagy időpontnak a *tartomány* által megadott részét adja vissza. Tetszés szerint megadhatjuk a hét első napját (az alapértelmezett a vasárnap), valamint az év első hetét, amely vagy január 1-jén kezdődik, és legalább négy napot tartalmaz, vagy az első teljes hét.
DateSerial(<*év*>, <*hónap*>, <*nap*>)	A megadott évnek, hónapnak és napnak megfelelő dátumértéket adja vissza.
DateValue(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaad egy dátum- és időértéket.
Day(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum nap részét adja vissza.
Hour(<*kifejezés*>)	Kiértékeli a kifejezést, és az időpont óraértékét adja vissza.
IsDate(<*kifejezés*>)	Kiértékeli a kifejezést, és igazat (True) ad vissza, ha a kifejezés érvényes dátumérték.
Minute(<*kifejezés*>)	Kiértékeli a kifejezést, és az időpont percértékét adja vissza.

Függvénynév	Leírás
Month(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátumértékhez tartozó hónap számát adja vissza.
MonthName(<*kifejezés*>,<*rövidítés*>)	Kiértékeli a kifejezést (amelynek 0 és 12 közötti egész értéknek kell lennie), és a megfelelő hónap nevét adja vissza. A név rövidített, ha a rövidítés argumentuma True.
Now	A helyi időzóna aktuális dátum- és időpontértéket olvassa ki.
Second(<*kifejezés*>)	Kiértékeli a kifejezést, és az időpont másodpercértékét adja vissza.
Time	A helyi időzóna aktuális időpontértéket olvassa ki.
TimeSerial(<*óra*>,<*perc*>,<*másodperc*>)	A megadott órának, percnek és másodpercnek megfelelő időpontértéket adja vissza.
TimeValue(<*kifejezés*>)	Kiértékeli a kifejezést, és az idő részt adja vissza.
WeekDay(<*kifejezés*>,<*hételsőnapja*>)	Kiértékeli a kifejezést, és a nap héten belüli számát adja vissza egész számmal. Az alapértelmezett első nap a vasárnap, de tetszés szerint megadhatjuk a hét első napját.
WeekDayName(<*napokszáma*>, <*rövidítés*>, <*hételsőnapja*>)	A nap héten belüli száma alapján visszaadja a nap nevét. Kérhetjük a név tetszés szerinti rövidítését. Az alapértelmezett első nap a vasárnap, de tetszés szerint megadhatjuk a hét első napját.
Year(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum év részét adja vissza.

Microsoft SQL Server

Függvénynév	Leírás
CURRENT_TIMESTAMP	A helyi időzóna aktuális dátumát és időpontját adja vissza.
DateAdd(<*tartomány*>, <*szám*>,<*kifejezés*>)	A *szám* mennyiségű *tartomány* időegységet adja hozzá a dátumhoz vagy dátum- és idő kifejezéshez.
DateDiff(<*tartomány*>, <*kifejezés1*>,<*kifejezés2*>)	A *kifejezés1* és *kifejezés2* dátum-és időértékek közötti, *tartomány* egységben mért különbségét adja vissza.

Függvénynév	Leírás
DateName(<*tartomány*>, <*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a megadott tartomány nevét tartalmazó karakterláncot. Ha a tartomány hónap vagy a hét egy napja, a nevet adja vissza.
DatePart(<*tartomány*>, <*kifejezés*>)	A kifejezésben szereplő dátum vagy időpont *tartomány* egységben megadott részét adja vissza.
Day(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum nap részét adja vissza.
GetDate()	Az aktuális dátumot adja vissza.
GetUTCDate()	Az egyezményes világidő (UTC) szerinti aktuális dátumot adja vissza.
IsDate(<*kifejezés*>)	Kiértékeli a kifejezést, és igazat (True) ad vissza, ha a kifejezés érvényes dátumérték.
Month(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátumértékhez tartozó hónap számát adja vissza.
Year(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum év részét adja vissza.

MySQL

Függvénynév	Leírás
ADDDATE(<*kifejezés*>, <*napok*>)	A megadott napok számát hozzáadja a kifejezésben szereplő dátumértékhez.
ADDTIME(<*kifejezés*>, <*idő*>)	A megadott időmennyiséget hozzáadja a dátum- és idő vagy dátumkifejezés értékéhez.
CURRENT_DATE	Az aktuális dátumértéket adja vissza.
CURRENT_TIME	A helyi időzóna aktuális időértéket olvassa ki.
CURRENT_TIMESTAMP	A helyi időzóna aktuális dátumát és időpontját olvassa ki.
DATE(<*kifejezés*>)	A dátum- és idő kifejezésből kinyeri a dátumot.
DATEDIFF(<*kifejezés1*>,<*kifejezés2*>)	Az első dátum- és idő kifejezésből kivonja a második dátum- és idő kifejezést, és visszaadja a kettő közötti napok számát.
DATE_ADD(<*kifejezés*>, INTERVAL <*tartomány*> <*mennyiség*>)	A megadott tartományú mennyiséget hozzáadja a kifejezésben szereplő dátum- és idő vagy dátumértékhez.
DATE_SUB(<*kifejezés*>, INTERVAL <*tartomány*> <*mennyiség*>)	A kifejezésben szereplő dátum- és idő vagy dátumértékből kivonja a megadott tartománymennyiséget.

Függvénynév	Leírás
DAY(<*kifejezés*>)	Kiértékeli a kifejezést, majd 1 és 31 közötti szám formájában visszaadja a dátumon belüli nap számát.
DAYNAME(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum vagy dátum- és idő napjának nevét adja vissza.
DAYOFMONTH(<*kifejezés*>)	Kiértékeli a kifejezést, majd 1 és 31 közötti szám formájában visszaadja a dátumon belüli nap számát.
DAYOFWEEK(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a dátumhoz vagy dátum- és időponthoz tartozó nap héten belüli számát, ahol 1 = vasárnap.
DAYOFYEAR(<*kifejezés*>)	Kiértékeli a kifejezést, majd 1 és 366 közötti értékben visszaadja a nap éven belüli számát.
EXTRACT(<*egység*> FROM <*kifejezés*>)	Kiértékeli a kifejezést, és a megadott egységet (például év vagy hónap) adja vissza.
FROM_DAYS(<*szám*>)	Az eltelt napok számának megfelelő dátumot adja vissza (a kezdődátum i. e. 1. december 31., a 366. nap dátuma pedig 01. január 1.).
HOUR(<*kifejezés*>)	Kiértékeli a kifejezést, és az időpont óraértékét adja vissza.
LAST_DAY(<*kifejezés*>)	A kifejezésben szereplő dátum hónapjának legutolsó napját adja vissza.
LOCALTIME, LOCALTIMESTAMP	Lásd a NOW függvényt.
MAKEDATE(<*év*>, <*évnapja*>)	A megadott évnek és az éven belüli napnak (1–366) megfelelő dátumot adja vissza.
MAKETIME(<*óra*>, <*perc*>, <*másodperc*>)	A megadott órának, percnek és másodpercnek megfelelő időpontot adja vissza.
MICROSECOND(<*kifejezés*>)	Kiértékeli a kifejezésben szereplő időbélyeget vagy időtartamot, és visszaadja az abban szereplő mikroszekundum-értéket.
MINUTE(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az időpont vagy dátum és idő perc részét.
MONTH(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátumértékhez tartozó hónap számát adja vissza.
MONTHNAME(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum- és időpont vagy dátumértékhez tartozó hónap nevét adja vissza.
NOW()	Az aktuális dátum- és időpontértéket adja vissza a helyi időzónából.

Függvénynév	Leírás
QUARTER(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja a kifejezésben szereplő dátumhoz tartozó negyedév számát.
SECOND(<*kifejezés*>)	Kiértékeli a kifejezést, és az időpont vagy dátum és idő másodperc részét adja vissza.
STR_TO_DATE(<*kifejezés*>, <*formátum*>)	A megadott formátum alapján kiértékeli a kifejezést, és dátumot, dátum- és időpontot vagy időértéket ad vissza.
SUBDATE(<*kifejezés*>, INTERVAL <*tartomány*>, <*mennyiség*>)	Lásd a DATE_SUB függvényt.
SUBTIME(<*kifejezés1*>,<*kifejezés2*>)	A *kifejezés2*-ben szereplő időpontot kivonja a *kifejezés1*-ben szereplő dátum- és időpontból vagy időpontból, és időpontot vagy dátumot és időpontot ad vissza.
TIME(<*kifejezés*>)	Kiértékeli a kifejezést, és visszaadja az időértékben vagy dátum- és időértékben szereplő időpontot.
TIMEDIFF(<*kifejezés1*>,<*kifejezés2*>)	A *kifejezés2*-ben szereplő időértéket vagy dátum- és időértéket kivonja a *kifejezés1*-ben szereplő időértékből vagy dátum- és időértékből, és visszaadja a különbséget.
TIMESTAMP(<*kifejezés*>)	Kiértékeli a kifejezést, és egy dátum- és időértéket ad vissza.
TIMESTAMP(<*kifejezés1*>,<*kifejezés2*>)	A *kifejezés2*-ben szereplő időpontot hozzáadja a *kifejezés1*-ben szereplő dátumhoz vagy dátum- és időértékhez, és egy dátum- és időértéket ad vissza.
TIMESTAMPADD(<*tartomány*>, <*szám*>, <*kifejezés*>)	A megadott számú tartományt hozzáadja a dátum és idő- vagy dátumkifejezéshez.
TIMESTAMPDIFF(<*tartomány*>, <*kifejezés1*>, <*kifejezés2*>)	Visszaadja a *kifejezés1* és *kifejezés2* dátum- vagy dátum- és időértéke között eltelt számú tartományt.
TIME_TO_SEC(<*kifejezés*>)	Kiértékeli a kifejezésben szereplő időpontot, és az idő értékét másodpercben adja vissza.
TO_DAYS(<*kifejezés*>)	Kiértékeli a kifejezésben szereplő dátumot, és a 0. évtől eltelt napok számát adja vissza.
UTC_DATE	Az egyezményes világidő (UTC) szerinti aktuális dátumot adja vissza.
UTC_TIME	Az egyezményes világidő (UTC) szerinti aktuális időpontot adja vissza.

Függvénynév	Leírás
UTC-TIMESTAMP	Az egyezményes világidő (UTC) szerinti aktuális dátumot és időpontot adja vissza.
WEEK(<*kifejezés*>, <*mód*>)	Kiértékeli a kifejezést, és a megadott módban visszaadja az érték dátum részében szereplő hét számát.
WEEKDAY(<*kifejezés*>)	Kiértékeli a kifejezést, és a héten belüli nap számát egész számmal adja vissza, ahol a 0 a hétfő.
WEEKOFYEAR(<*kifejezés*>)	Kiértékeli a dátumkifejezést, és visszaadja a hét számát (1–53), feltételezve, hogy az első hétnek több mint három napja van.
YEAR(<*kifejezés*>)	Kiértékeli a kifejezést, és a dátum év részét adja vissza.

Oracle

Függvénynév	Leírás
CURRENT_DATE	Az aktuális dátumértéket olvassa ki.
CURRENT_TIMESTAMP	Az aktuális dátumot, időpontot és időbélyeget olvassa ki a helyi időzónából.
DBTIMEZONE	Az adatbázis időzónáját adja vissza.
EXTRACT(<*tartomány*> FROM<*kifejezés*>)	Kiértékeli a kifejezést, és a kért tartományt (év, hónap, nap stb.) adja vissza.
LOCALTIMESTAMP	A helyi időzóna aktuális dátumát és időpontját olvassa ki.
MONTHS_BETWEEN(<*kifejezés1*>, <*kifejezés2*>)	Kiszámolja a *kifejezés1* és *kifejezés2* közötti hónapok, illetve tört hónapok számát.
NEW_TIME(<*kifejezés*>,<*időzóna1*>, <*időzóna2*>)	Úgy értékeli ki a dátumkifejezést, mintha az lenne az első időzóna, és a második időzónában szereplő dátumot adja vissza.
NEXT_DAY(<*kifejezés*>,<*napnév*>)	Kiértékeli a kifejezést, és a kifejezésben szereplő dátumot követően időbélyeg formájában visszaadja a *napnév*-ben (amelyben a „MONDAY", „TUESDAY" stb. szavakat tartalmazó karakterlánc lehet) megadott első nap dátumát.
NUMTODSINTERVAL(<*szám*>,<*egység*>)	A számot a megadott egység szerinti időtartománnyá (NAP, HÓNAP, PERC, MÁSODPERC) alakítja.
NUMTOYMINTERVAL(<*szám*>, <*egység*>)	A számot a megadott egység szerinti időtartománnyá (ÉV, HÓNAP) alakítja.

Függvénynév	Leírás
ROUND(<*kifejezés*>, <*tartomány*>)	A dátumértéket a megadott tartományra kerekíti.
SESSIONTIMEZONE	Az aktuális munkamenet időzónáját olvassa ki.
SYSDATE	Az adatbázis-kiszolgáló aktuális dátumát és időpontját olvassa ki.
SYSTIMESTAMP	Az adatbázis-kiszolgáló aktuális dátumát, időpontját és időzónáját olvassa ki.
TO_DATE(<*kifejezés*>, <*formátum*>)	A megadott formátum alapján a karakterlánc formájú kifejezést dátummá alakítja.
TO_DSINTERVAL(<*kifejezés*>)	A karakterlánc formájú kifejezést nap–másodperc tartománnyá alakítja.
TO_TIMESTAMP(<*kifejezés*>, <*formátum*>)	A megadott formátum alapján a karakterlánc formájú kifejezést időbélyeggé alakítja.
TO_TIMESTAMP_TZ<*kifejezés*>, <*formátum*>)	A megadott formátum alapján a karakterlánc formájú kifejezést időzónában megadott időbélyeggé alakítja.
TO_YMINTERVAL	A karakterlánc formájú kifejezést év–hónap tartománnyá alakítja.
TRUNC(<*kifejezés*>, <*tartomány*>)	A dátumértéket a megadott tartományra csonkítja.

D

Ajánlott irodalom

Ajánlott irodalom

Az alábbi könyveket ajánlatos elolvasni, ha többet szeretnénk megtudni az adatbázis-tervezésről, vagy bővíteni akarjuk az SQL-lel kapcsolatos ismereteinket. Tartsuk szem előtt, hogy szakmaibb jellegénél fogva néhány könyv nagy kihívást jelenthet a számunkra, egyes szerzők pedig igen jelentős háttértudást feltételeznek a számítógépek, az adatbázisok és a programozás terén.

Adatbázisokról szóló könyvek

Connolly, Thomas és Begg, Carolyn: *Database Systems: A Practical Approach to Design, Implementation, and Management* (4. kiadás). Essex, Anglia, Addison-Wesley, 2004.

Date, C. J.: *An Introduction to Database Systems* (8. kiadás). Boston, MA, Addison-Wesley, 2003.

Database in Depth: Relational Theory for Practitioners. Sebastopol, CA: O'Reilly Media, 2005.

Hernandez, Michael J.: *Database Design for Mere Mortals* (2. kiadás). Boston, MA, Addison-Wesley, 2003. (Magyarul: *Adatbázis-tervezés: A relációs adatbázisok alapjairól földi halandóknak*. Kiskapu, 2004.)

Az SQL-ről szóló könyvek

Bowman, Judith S., Emerson, Sandra L. és Darnovsky, Marcy: *The Practical SQL Handbook* (4. kiadás). Boston, MA, Addison-Wesley, 2001.

Celko, Joe.: *Joe Celko's SQL for Smarties: Advanced SQL Programming* (3. kiadás). San Francisco, CA, Morgan Kaufmann Publishers, 2005. (Korábbi kiadás magyarul: *SQL felsőfokon.* Kiskapu, 2002.)

Date, C. J. és Darwen, Hugh: *A Guide to the SQL Standard* (4. kiadás). Reading, MA, Addison-Wesley, 1997.

Gruber, Martin: *SQL Instant Reference* (2. kiadás). Alameda, CA, Sybex Inc., 2000. (Magyarul: *SQL A-tól Z-ig.* Kiskapu, 2003.)

Melton, Jim és Simon, Alan R.: *Understanding the New SQL: A Complete Guide.* San Francisco, CA, Morgan Kaufmann Publishers, 2006.

Tárgymutató

A, Á

B

C, Cs

D

E, É

F

G

H

I

J

K

L

M

N

O, Ö

P

Q

R

S, Sz

T

U, Ü

V

W

X

Z