

## ELSŐ FEJEZET

# Elméleti áttekintés

Mielőtt az Olvasót bevezetném az egyes adatbázis-kezelők üzemeltetésének rejtelseibe, illetve az általuk elfogadott SQL-parancsok részleteibe, a későbbi fejezetekben foglalt ismeretek könnyebb és hatékonyabb elsajátítása érdekében szükség van némi előkészítésre, mind elméleti, mind gyakorlati síkon. Kezdetnek röviden összefoglalom a relációs technológia alapelveit, valamint előnyeit és hátrányait a többi lehetséges adattárolási modellhez (hierarchikus, objektumorientált, kulcs-érték párok stb.) viszonyítva, hiszen bármilyen általános és sokrétű legyen is, nem minden feladat megoldására ez a legmegfelelőbb eszköz. Egyáltalán nem törekszem a kérdéskör kimerítő tárgyalására, amelyre véleményem szerint egy fejlesztőnek egyébként sincs szüksége, sem ideje; a gyakorlatok lelkiismeretes elvégzésével úgyis hamar „rá lehet érezni” a relációs rendszerek által megkövetelt sajátos gondolkodásmódra.

A számítástechnika hajnalán, az 1950-es években a számítógépeket szinte kizárólag matematikai problémák megoldására alkalmazták, azonban alig pár évre volt csak szükség arra, hogy a kutatókon és kormányzati szerveken kívül a vállalatok is felismerjék az automatikus adatfeldolgozásban rejlő óriási lehetőséget. Kezdetben igen egyszerű tevékenységekről (mással a mai számológépekkel összemérhető teljesítményű berendezések amúgy sem lettek volna képesek megbirkózni) volt szó: lyukkártyára vagy mágnesszalagra rögzített adatok összegzéséről, sorba rendezéséről, szétválogatásáról és így tovább. 1960. elején megjelent a COBOL (*Common Business Oriented Language: általános üzleti célú nyelv*) programozási nyelv, a fejlesztőknek így egyrészt már nem kellett a sokszor meglehetősen nehézkes gépközi nyelvekben dolgozniuk, másrészt az elkészült programok elvben hordozhatóvá váltak a különböző operációs rendszerek között. Az „adatbázis” azonban még mindig nem volt több, mint egyszerű rekordsorokat tartalmazó, a fejlesztő által nyilván- és karbantartott állományok összessége; ráadásul minden egyes adatfeldolgozási feladathoz külön alkalmazást kellett írni.

A mind bonyolultabb és átfogóbb szoftverrendszerek megvalósítása iránti igény olyan eszköz létrehozását tette szükségessé, amely teljes egészében magára vállalja az adatok szervezésével, tárolásával és előhívásával kapcsolatos teendőket. Ezáltal egyrészt a feldolgozást végző alkalmazás jelentős terhektől szabadul meg, másrészt azt nem kell időigényes felülvizsgálatnak és javításnak alávetni minden alkalommal, amikor a rögzített adatok köre vagy típusa valamilyen okból módosításra szorul. Az első ilyen, immár teljes joggal adatbázis-kezelőnek tekinthető megoldások egyike volt az IBM által 1968-ban piacra dobott, egyelőre még hierarchikus modellt követő IMS/360 (*Information*

*Management System: információkezelő rendszer*). A következő évben jelent meg E. F. Coddnak a relációs modellt bevezető, rendkívül nagy hatású cikke; az IBM-nél azonnal hozzáfogtak az elmélet megvalósításához, és ennek eredményeképpen 1977-re megszületett a SEQUEL (*Structured English Query Language: strukturált angol lekérdező nyelv*) nyelven programozható, System R elnevezésű prototípus. Ám a termék továbbfejlesztett, kereskedelmi változáttal túlságosan sokat késlekedtek: az Oracle (akkoriban Relational Software) számottevő előnyre tett szert azáltal, hogy már 1980-ban kiadta a System R mintájára készült adatbázis-kezelőjét.

A történet inentől kezdve valószínűleg ismerős. Nemsokára szinte mindegyik operációs rendszeren és számítógéptípuson elérhetővé váltak a különböző szoftvergyártók által készített, hol egyszerűbb, hol bonyolultabb relációs adatbázis-kezelő rendszerek (*Relational Database Management System, RDBMS*). Legtöbbjük mára feledésbe merült, csak néhányuknak sikerült mindvégig megtartaniuk vezető szerepüket, vagy valamilyen más termékbe olvadva továbbélniük. 1986-ban megtörtént a jogi okokból SQL-re (*Structured Query Language*) átkeresztelt SEQUEL nyelv szabványosítása, azonban ahogyan a relációs technológia egyre szélesebb körű felhasználása mind nagyobb teljesítményű és kifinomultabb rendszerek létrehozását tette szükségessé, a lefektetett keretek szűknek bizonyultak. A szabvány 1989-ben kiegészült a hivatkozási integritás megőrzését célzó eszközökkel, 1992-ben pedig, többek között, a sémákat kezelő parancsokkal, a táblák külső és belső illesztésének lehetőségével, új adattípusokkal, valamint a lekérdezések eredményein végezhető halmazműveletekkel bővült. A fejlődés természetesen itt korántsem állt meg: az 1999-es kiadásban egyrészt igyekeztek minél nagyobb mértékben bevezetni a programozásban jól bevált objektumorientált szemléletmódot, másrészt folytatódott a „hagyományos” képességek kiterjesztése. Az utóbbihoz tartoznak például a nagyméretű bináris és szöveges értékek, a logikai típus, vagy a jogosultságok szabályozását megkönnyítő szerepek megjelenése.

Az SQL jelenleg érvényben levő, 2003. évi változatában leírt újdonságokat felsorolni is hosszú lenne: mezőként tárolható értékhalmozok kezelése, többdimenziós adatok elemzését segítő OLAP- (*On-Line Analytical Processing: online elemző feldolgozás*) függvények, XML (*Extensible Markup Language: kiterjeszhető jelölőnyelv*) formátumú tartalom rögzítése és teljes, illetve részleges előhívása stb. A szabványban leírt szolgáltatások egy részét korábban már több adatbázis-kezelő megvalósította, sokukat azonban egyelőre még a legnagyobb rendszerekben sem találjuk meg.

Ne ijedjen meg az Olvasó, ha az előbbieken említett elnevezések egyikéről-másikáról sohasem hallott. Noha a könyvben szereplő adatbázis-kezelők bonyolultságban nem vehetik fel a versenyt az Oracle vagy az IBM termékeivel (ami a tanulás szempontjából tagadhatatlanul előny), jó néhány képesség, ha néhol igen kezdetleges formában is, de megjelenik bennük. A későbbi fejezetekben tehát az XML, az OLAP és társaik ismét felbukkannak majd, immár jelentőségük és jelentőségük részletes magyarázatával együtt.

A rövid történeti áttekintés után legfőbb ideje közelebbről is megismerkednünk magával a relációs modellel: hogyan történik a tények formalizálása, ténylegesen milyen szerkezetben történik az adatok tárolása a relációs modell követelményei szerint, illetve általában milyen szolgáltatások jellemzők a relációs adatbázis-kezelőkre.

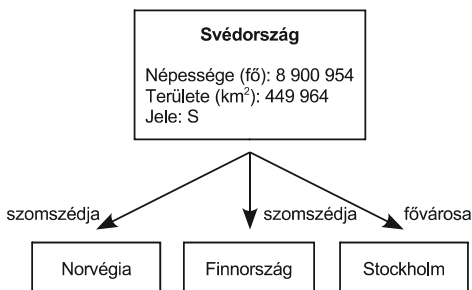
## 1.1. Adatmodellezés

Adatbázis létrehozásakor legelőször azt kell meghatároznunk, hogy a körülötünk levő világnak tulajdonképpen melyik részletét kívánjuk rögzíteni, és ezáltal későbbi előkeresésre, feldolgozásra alkalmassá tenni. Egy üzlethálózat vezetősége számára a napokra és boltokra lebontott eladási adatok lényegesek, a középiskola adminisztrátorai a diákok különböző tantárgyakból kapott osztályzataira kíváncsiak, a könyvtárosok a gondjaikra bízott könyveket és kölcsönzéseiket szeretnék nyilvántartani stb. Miután kijelöltük az adatbázis által lefedett tudásterület (*knowledge domain*) határait, az abban foglalt tényanyagot formálisan, a számítógép számára megfogható alakban kell megfogalmaznunk – ezt a tevékenységet nevezzük adatmodellezésnek. Ennek során a tudásterületet háromfajta összetevő segítségével kell felépítenünk: egyedekkel (*entity*), egyedekhez rendelt tulajdonságokkal (*attribute*), valamint az egyedek között fennálló kapcsolatokkal (*relation*). A rendelkezésünkre álló eszköztár első pillantásra túlságosan szegényesnek tűnhet, ám mégis elegendő a mindennapi gyakorlat során adódó feladatok túlnyomó többségének megoldásához, ha nem is mindig vezet az elképzelhető legtömörebb kifejezőmódhoz. Az adatmodellezés nem kötődik a relációs elvhez, bármelyik más technológián alapuló adatbázis (objektumorientált, hierarchikus) megvalósításának előkészítésében is hasznos.

A következőkben bemutatásra kerülő adatmodellezési eljárás, hogy jobban illeszkedjen a valós feladatokhoz és a relációs modellhez, bizonyos pontokon eltér a szakirodalomban ismertett módszerektől. A leírás egyáltalán nem tudományos igényű és valószínűleg nem is teljes. Amennyiben az Olvasó komolyabban el kíván mélyülni ebben a témakörben, mindenképpen ajánlott egy kifejezetten ezzel foglalkozó könyvet áttanulmányoznia.

Szemléltetésként nézzük meg a következő megállapítást: Svédország szomszédjai Norvégia és Finnország, fővárosa Stockholm, lakossága 8 900 964 fő, területe 449 904 km<sup>2</sup>, jele S. A mondat tartalmát lefordítva a számítógép fogalomvilágába a következőket kapjuk: négy egyed, nevezetesen három országot és egy várost; a Svédország egyedhez tartozó három tulajdonságot, melyek közül a népessége és a területe számszerű, a jele pedig szöveges; végül a Svédország és Stockholm egyedeket összekötő fővárosa kapcsolatot (lásd a mellékelt ábrát). Szigorúan véve az eredmény hiányos, hiszen egyrészt nem derül ki belőle, milyen mértékegységben értendők a 8 900 964 és 449 904 ér-

tékek, másrészt rejtve marad a Stockholm város, illetve a másik három egyed ország mivolta. Megtehetnénk ugyan, hogy az egyedekhez hozzárendelünk egy típusa mezőt (benne egy rövid ismertetéssel vagy egyezményes jelöléssel), és hogy a tulajdonságokat párosával kezeljük, az egyik bejegyzés hordozza az értéket, a másik a hozzá tartozó mértékegységet. Azonban sokkal egyszerűbben célt érünk, ha a fenti adatokat nem kifejezetten, hanem csupán hallgatólagosan jelenítjük meg, vagyis néhány, amúgy is értelemszerű szabály betartásával azokat kikövetkeztethetővé tesszük.



#### Kijelentés formalizált változata

A legfontosabb, hogy minden egyedat besoroljunk valamilyen típusba (város, személy, számlatétel, lakcím stb.), valamint hogy gondoskodjunk arról, az adott típushoz tartozó összes egyedat pontosan ugyanolyan tulajdonságokkal ruházzuk fel, illetve mindig ugyanolyan jellegű kapcsolatokat, azon belül ugyanazt a szerepkört alkalmazzuk rájuk. Amennyiben például az adatbázis tervezésekor elhatároztuk, hogy az országokról területüket és fővárosukat rögzítjük, akkor ezt kivétel nélkül mindegyik ország típusú egyedre meg kell tennünk. Ugyanakkor nyilvánvalóan értelmetlen lenne az országokra valamely más típusra értelmezett tulajdonságot vagy kapcsolatot (életkor, anyja) megadni, esetleg a fővárosa kapcsolatot fordított irányban felvenni (Stockholm fővárosa Svédország). A másik szabály: a különböző tulajdonságok és kapcsolatok a típus egyedeinél pontosan ugyanazt jelentsék. Ha a területe értéket az egyik országnál  $\text{km}^2$ -ben írjuk be, a másikonál ne hektárt vagy  $\text{m}^2$ -t használjunk; hasonlóképpen, ha a neme megadása bizonyos személynél az F és N jelekkel történt, a többiekénél ugyanitt ne a Férfi és Nő szöveg legyen olvasható. A két szabály áthágásával az adatbázis egységessége és következetessége sérül, jelentősen nehezítve a későbbi feldolgozást. Célszerű tehát minél hamarabb részletesen lefektetni a megengedett típusok, tulajdonságok és kapcsolatok körét az alkalmazásukra vonatkozó irányelvekkel együtt, majd annak alapján folyamatosan felügyelni a megvalósítás menetét.

Az adatmodellezésről festett kép még korántsem teljes. Ami a tulajdonságokat illeti, egyrészt lehetnek közöttük olyanok, amelyek kitöltése nem kötelező – akár mert bizonyos egyedeknél a kérdéses tulajdonság értelmetlen lenne (férfi személyi adatainál a leánykori név), akár mert a szükséges információ valamilyen ok miatt egyszerűen nem hozzáférhető. Másrészt egyáltalán nem til-

tott, hogy különböző típusok egy vagy több, egyforma nevű tulajdonsággal rendelkezzenek, hiszen például a népessége és a területe értékek országokra, megyékre, városokra egyaránt érvényesek. Amennyiben két, valóban hasonló információt tartalmazó tulajdonságnak választunk azonos nevet, könnyebbé tehetjük a felhasználók és a programozók eligazodását az adatbázisban; máskülönben zavart okozunk, rosszabb esetben súlyos következményekkel járó tévedéseket idézünk elő. Az eltérő típusú egyedekhez tartozó, azonos megnevezésű tulajdonságainak értékeit éppúgy érdemes egységes mértékegységben, illetve jelölésrendszer szerint tárolni, mintha egyfajta típusú egyedekről lenne szó.

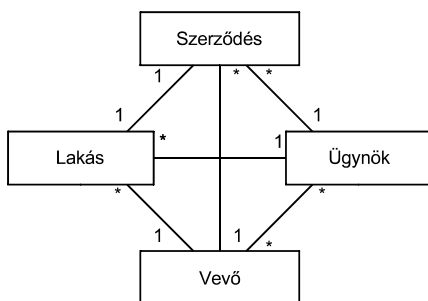
A kapcsolatokról elmondottak szintén némi kiegészítésre szorulnak. Az eddig bemutatott példákkal ellentétben a kapcsolatok nem csupán két, hanem tetszőleges számú (de természetesen legalább egy) egyedre vonatkozhatnak. Elvileg elképzelhető olyan megfigyelés kapcsolat, mely összefüggésbe hozza a betegséget, az alkalmazott gyógymódot, a végeredményt, valamint az esetleges káros mellékhatást; és létezhet olyan elavult nevű „kapcsolat” is, amellyel a forgalomból kivont cikkeket jelöljük ki, bármifajta járulékos információ megadása nélkül. A gyakorlatban a két tagnál kevesebbet vagy többet tartalmazó kapcsolatok rendkívül ritkák, az előbbieket tulajdonságként, az utóbbiakat egyedekként szokták megvalósítani (lásd az alfejezet későbbi részében). Előfordulhat, hogy a kapcsolatban több egyező típusú egyed is jelen van, ám ebből egyáltalán nem következik az, hogy helyzetük egyenrangú. Az országokra értelmezett szomszédja két egyede valóban büntetlenül felcserélhető, a személyek között fennálló szülője, vagy a megfigyelés eredményt és mellékhatást jelképező szerepkörei viszont nem. Az azonos nevet viselő kapcsolatok jelentése egy jól megtervezett adatbázisban pontosan ugyanaz, ennek megfelelően ideális esetben mindig ugyanannyi, meghatározott típusokhoz tartozó egyed tartalmaznak. Időnként mégis szükség lehet bizonyos szerepkörök – például a semmiféle változást nem tapasztaló megfigyelésekben az eredmény és a mellékhatás – kihagyására; a tulajdonságokhoz hasonlóan ezeket a szerepköröket előzetesen mint nem kötelezően kitöltendőket kell megjelölnünk.

Valamely egyed korlátlan számú különböző kapcsolatnak lehet résztvevője, Kiss János személyére nemcsak a megrendelések, hanem a múltbeli reklamációk, a leghűségesebb vevőknek járó különleges kedvezmények, vagy a fizetési mód kapcsán is hivatkozhatunk. Mi a helyzet azonban az egyező típusú kapcsolatokkal? Előfordulhat, hogy ugyanarra az egyedre, ugyanabban a szerepkörben két vagy több azonos megnevezésű kapcsolat hivatkozik? Az egyszerűség kedvéért egyelőre szorítkozzunk a pontosan két tagból álló kapcsolatokra, amelyek három csoportba sorolhatók be:

- Az egy-egy típusú kapcsolatok, mint amilyen például a fővárosa vagy a házastársa, ugyanarra az egyedre csupán egyszer (vagy egyszer sem, amennyiben a kapcsolat megadása nem kötelező) vonatkozhatnak, függetlenül attól, hogy az egyed melyik szerepkört tölti be. Egyetlen olyan ország van, amelynek Budapest a fővárosa; és fordítva, Magyarországnak egy fővárosa van.

- Az egy-több jellegű kapcsolatoknál az egyik szerepkörben álló egyed kizárólag egyszer, a másik viszont akármennyi alkalommal megjelenhet az adatbázisban. Ide tartozhat a lakcíme és a megyéje kapcsolat: mindenkinek egy lakcíme van, ám ugyanott többen is lakhatnak; illetve egy ország több megyéből állhat, míg ugyanaz a megye nem fordulhat elő két különböző országban belül.
- A több-több kapcsolatok esetében mindkét szerepkörben megismétlődhetnek egyedek, a három csoport közül tehát ez a legkövetlenebb. Példaként említhető a vásárlókat és a termékeket összekötő megrendelő kapcsolat, hiszen egyrészt adott személy tetszőleges számú cikket rendelhet, másrészt ugyanazt a cikket (feltéve, hogy nem egyedi készítésű árukról van szó) sokan megrendelhetik.

Amennyiben a kapcsolat egyetlen szerepkört foglal magában, azt nyilvánvalóan értelmetlen lenne többszörösen alkalmazni egy egyedre, vagyis jellege ekkor mindenképpen „egy-egy”. Ha a kapcsolat által összefogott egyedek száma kettőnél nagyobb, a fent említett viszonyok pusztán két-két szerepkör között értelmezendők, és nem feltétlenül egységesek az egész kapcsolatra. Ennek szemléltetéséhez képzeljünk el egy számos ügynököt foglalkoztató ingatlanforgalmazó céget, ahol minden ügynök saját körzettel rendelkezik, s kizárólag az ott található lakások közvetítéséért felelős. A sikeres eladások adatait a szerződések kapcsolat segítségével fogjuk össze, amely hivatkozik a vevőre, az ügynökre, a lakásra és az eladás feltételeit tartalmazó, elektronikusan aláírt szerződésre. A lakás és a szerződés között egy-egy, az ügynök és a lakás között egy-több, végül a vevő és az ügynök között több-több jellegű összefüggés (mivel ugyanaz a vevő több ügynöktől vásárolhat, illetve egyazon ügynök több vevőt kiszolgálhat) áll fenn. Ebből már könnyen kikövetkeztethető a fennmaradó három szereppár kapcsolatának jellege, ahogyan azt az Olvasó a mellékelt ábrán is láthatja.



**Szerepkörök közötti viszonyok egy négy szerepkörből álló kapcsolatban**

Az egyszeres jelenlétet „1”, a többszöröst „\*” jelzi, azaz például az egy-több kapcsolatot jelképező egyenes két végén rendre az „1” és a „\*” szimbólum helyezkedik el.

Adatbázisok tervezésekor az Olvasó könnyen olyan helyzetben találhatja magát, hogy el kell döntenie, adott tény formalizálásához a rendelkezésre álló lehetőségek közül melyiket válassza. Éppen úgy, mint valamilyen algoritmus kódolásakor, az egyes megvalósítási változatok elméletben egyenrangúak lehetnek ugyan, de a későbbi feldolgozás egyszerűsége, áttekinthetősége, teljesítménye szempontjából jelentősen eltérhetnek egymástól. Ezért a következőkben ismertetem azokat a legfontosabb irányelveket, amelyek betartásával elkerülhetők a későbbi kellemetlen meglepetések, amikor már valószínűleg túlságosan késő vagy költséges módosítani az adatbázis szerkezetét.

A kapcsolatok és a tulajdonságok nagyon hasonlóak egymáshoz, hiszen lényegében mindkét esetben két adatot hozunk összefüggésbe egymással. Ország fővárosát mind a korábbiakban látott fővárosa nevű kapcsolattal, mind pedig a város nevét tartalmazó tulajdonsággal megadhatjuk – melyiket érdemes előnyben részesíteni? Elméleti síkon kijelenthető: a kérdéses ismeretet egyedként kell kezelni, ha a tudásterület valamilyen központi fogalmának (például amelyik előfordul a tudásterület rövid szöveges leírásában) felel meg, egyéb esetben pusztán tulajdonságként. A gyakorlatban három dolog szabja meg, hogyan célszerű eljárni: az ismeret felbontható-e további tulajdonságokra, hány egyedtípussal hozható kapcsolatba, végül pedig milyen jellegűek ezek a kapcsolatok. Az Olvasó elkerülheti ugyan ezeket a megfontolásokat, és minden létező ismeretet egyedként ábrázolhat, ám ez annyira súlyos tárkapacitás- és teljesítménygondokhoz vezet, amelyek kétségbe vonhatják az adatbázis megvalósíthatóságát.

Amennyiben az ismeret valamilyen egyszerű értéket (dátum, igen/nem jelölés, pénzösszeg stb.) takar, ritkán ismétlődik, vagy csupán egyetlen elemtípushoz kapcsolódik – ráadásul mindenkor változatlan szerepkörben –, akkor azt az adott elemtípus tulajdonságaként érdemes ábrázolnunk. Semmit sem nyernénk a vásárlók nemének vagy lakcímének önálló egyedde alakításából, hiszen az első esetben a neme kapcsolat példányai jóval több tárterületet igényelnének, mint maga az F/N érték; míg a második esetben a lakcímre nagy valószínűséggel úgyis csak egyetlen egyednél hivatkozunk. A tudásterületen önálló jelentéssel nem bíró, kevésbé jelentős tények egyedekké történő „előléptetésével” feleslegesen bonyolódik az adatmodell, és nehezkesebbé válik a későbbi adatkezelés, ezért ezt lehetőleg el kell kerülni. Ellenben, ha a szóban forgó ismeret egyrészt összetett tartalmú, másrészt vagy több egyedhez köthető, vagy egyetlenegyhez, ám ahhoz több különböző szerepkörben, úgy azt helyesebb egyedként leképeznünk az adatmodellezés során. Nyilvánvalóan ide sorolhatók a vásárlók, hiszen őket nemcsak a megrendelések, hanem a reklamációk, kedvezmények és eladási statisztikák kapcsán is megemlítjük az adatbázisban; valamint a vállalat saját munkatársai, akikre bár kizárólag csupán a rendelésekben utalunk, ám ott akár mint ügynökökre, akár mint ügyintézőkre.

Különleges helyzetet jelent, mikor egy egyedtípus valamely jellemzője többszörös, mondjuk előadás időpontjai és az azokhoz tartozó helyszínei, vagy ország hivatalos nyelvei. Összetett jellemzőnél, mint amilyen az első példa is, nincs min gondolkodni: kénytelenek vagyunk felvenni az előadás-példány típusú helyszíni és időpont tulajdonságokkal, valamint az ilyen egyedeket a megfelelő előadásokhoz kötő példánya kapcsolatot. Mindezt annak dacára, hogy az adatbázisban a helyszíni-időpont párosok önállóan nem értelmezhetőek, és nincs olyan kiemelt szerepük, mint az előadásokat, résztvevőket vagy szervezőket jelképező egyedeknek. Egyszerű jellemzőknél viszont, mint a hivatalos nyelv, az Olvasó számára első pillantásra talán jó ötletnek tűnik, hogy tárolásukra használjunk annyi tulajdonságot, ahány nyelv előfordulhat egyetlen országgal kapcsolatban. A tulajdonságok elnevezései legyenek `nyelv1`, `nyelv2`, `nyelv3` stb.; közülük csak az első megadását tegyük kötelezővé, a többieké maradjon választható. Gondoljunk azonban bele: később annak megállapításához, hogy az egyes országoknak hány hivatalos nyelve van, illetve mely országok hivatalos nyelve az angol, az összes `nyelv*` tulajdonságra hivatkoznunk kellene majd a feltételekben! Mi több, az ország típusú egyedek nagyobb részében üres értékekre pazarolnánk el az értékes tárkapacitást! Bármilyen furcsának tűnik, ismét be kell vezetnünk egy új egyedtípust (ezúttal egyetlen tulajdonsággal, a nyelv megnevezésével), illetve emellett persze a hozzá tartozó, egy-több jellegű hivatalos nyelve kapcsolatot.

Nemcsak a kapcsolatok és a tulajdonságok, hanem a kapcsolatok és az egyedek is hasonlóak: mindkettő tetszőleges számú, egymással egyenrangú ismeret között teremt összefüggést, illetve osztályozni képes azokat (amennyiben egyetlen adatról van szó). Az irányelvek itt a következők: ha az összekapcsolandó ismeretek nem mindegyike egyed, vagy az adatbázis más helyein közvetlenül hivatkozni kívánunk magára az összekapcsolásra, utóbbit egyedként valósítsuk meg. Termékrendeléseknél például teljesül a fenti két feltétel: egyrészt a rendelésben valószínűleg szerepel a kért mennyiség, amelyet egyszerűsége miatt a legjobb indulattal sem tekinthetünk egyednek, csupán tulajdonságnak; másrészt a rendelésre utalnak a számlákat, a kifizetéseket és a szállítmányokat képviselő egyedek. A szülője kapcsolatot ezzel szemben nem célszerű egyedre alakítani: két szerepkörében kizárólag egyedeket találunk, ráadásul kevéssé hihető, hogy bárhol a szülő vagy a gyermek személye helyett kifejezetten a leszármazás tényét kívánnánk megemlíteni (kivéve egy olyan genealógiai adatbázisban, ahol bizonyítékként meg kell adni a leszármazást alátámasztó forrásmunkát).

Kevésbé egyértelmű, hogy egyedek osztályozásánál mikor érdemes tulajdonsághoz, és mikor egyetlen tagból álló kapcsolathoz folyamodnunk, hiszen ez nagyban függ a leképezendő tudásterület általános jellegétől és a kérdéses osztály hatálya alá tartozó egyedek számától. Egyértelmű, hogy személyek nemét inkább egy `nem` nevű, `F` vagy `N` értéket tartalmazó tulajdonsággal célszerű megvalósítani, mint a férfi és nő kapcsolatokkal.

Azonban a vállalat munkatársainak vezetőkre és alárendeltre való szétválasztásánál a választás már korántsem ennyire könnyű. Jobbnak tűnik az igen/nem értékű vezető tulajdonság, amikor a munkatársakat több más szempont szerint is be kell sorolnunk – teljes- és részmunkaidőben foglalkoztatottak, külsősök és belsősök, szellemi és fizikai dolgozók stb. Máskülönb hatékonyabb a vezetőség kapcsolat, különösen, ha a vezetők aránya igen csekély, így ugyanis a munkatársaknak megfelelő egyedeket nem kell terhelnünk egy előfordulásainak túlnyomó többségében „nem” értéket hordozó újabb tulajdonsággal.

## 1.2. Táblák, oszlopok és sorok

Miután a kiválasztott tudásterület tényanyagát sikerült egyedek, tulajdonságok és kapcsolatok révén ábrázolnunk, az adatbázis tervezésének legnehezebb részén már túl vagyunk. Mivel azonban a relációs rendszerek az iménti fogalmakat közvetlenül nem ismerik, ehelyett mindenfajta információt (beleértve az adatbázis üzemeltetésére vonatkozó adatokat is) kizárólag táblák segítségével tárolnak, még egy leképezésre van szükség. Mielőtt ennek részleteiben elmélyülhetnénk, természetesen közelebbről meg kell ismerkednünk a táblák fogalmával és felépítésével.

A relációs modellben az információ legkisebb egysége a mező (*field*), amely típusának megfelelően egész vagy valós számot, szöveget, dátumot, időpontot stb. hordozhat. Az egymással szoros kapcsolatban, matematikai szóhasználattal valamilyen relációban (innen a modell elnevezése) álló mezőket sorokba (*row*) foglalhatjuk: így például a lakcímet alkotó utca, házszám, emelet, város és irányítószám értékeket. Az azonos szerkezetű, pontosabban fogalmazva adott számú és típusú mezőket megszabott sorrendben tartalmazó sorokból táblák (*table*) építhetők fel. Amennyiben minden egyes sorból kiválasztjuk az ugyanazon szerepkört betöltő, ennél fogva egyforma típusú mezőket, megkapjuk a tábla oszlopait (*column*). Az oszlopok némelyikébe tartozó mezők kitöltése kötelező, másoké nem; az esetlegesen hiányzó értékek helyén mindig a NULL áll, függetlenül a kérdéses mező típusától. A táblák tehát legfeljebb annyiban térnek el a hagyományos papírra írt táblázatoktól, hogy egyrészt lehetetlen oda nem illő adatokat bejegyezni (házszámként nevet vagy dátumként számot), másrészt egyáltalán nem okoz gondot több ezer, akár millió sor rögzítése. Szemléltetésként tekintsünk egy lakcímetek tároló, öt sorból és öt oszlopból álló táblát, mégpedig a következők szerint:

UTCA	HÁZSZÁM	EMELET	VÁROS	IRÁNYÍTÓSZÁM
MACSKA U.	17	NULL	PÉCS	9945
KUTYA T.	3	3	BUDAPEST	1338
GALAMB U.	6	NULL	MISKOLC	7609
EGÉR KRT.	21	0	SZEGED	4211
KESELYŰ T.	105	1	VÁC	2765

A táblában minden sor egy-egy lakcímet képvisel, melyekben az UTCA, a HÁZSZÁM, a VÁROS és az IRÁNYÍTÓSZÁM oszlopokhoz tartozó mezők kötelezően kitöltendők, az EMELET oszlop mezői közül viszont bármelyik üresen hagyható (ezt mutatják a NULL értékek), amennyiben az adott cím például egyszintes épületet vagy egyalakásos házat jelöl. Az UTCA és a VÁROS szöveges, a többi három oszlop numerikus tartalmú, noha valószínűleg sohasem fogunk semmifajta aritmetikai műveletet elvégezni rajtuk. A legtöbb adatbázis-kezelőnél azonban az egész számok tárolása, feldolgozása és külső programok felé való esetleges továbbítása jóval hatékonyabban történik, mint a szöveges típusú adatoké; így, ha szabadon választhatunk a kétféle ábrázolási mód között, előbbit részesítsük előnyben.

A tábláknak tulajdonképpen két „arca” van. A szerkezetet, mely megszabja az oszlopok számát, nevét, típusát, illetve kitöltésük kötelező vagy választható jellegét, ideális esetben az adatbázis tervezésekor véglegesen rögzítjük. Ezzel szemben a tartalom, vagyis a táblához tartozó sorok összessége általában folyamatosan változik az adatbázis élettartama alatt, mind az egyes mezők értékeit, mind pedig az adatok mennyiségét tekintve. Az Olvasó látni fogja, mennyire élesen szétválnak a szerkezetre és a tartalomra vonatkozó SQL-parancsok – egyrészt mert az adatok megjelenítése, illetve feldolgozása során szinte kizárólag utóbbiakra szorítkozunk, másrészt mivel a relációs rendszereknél az előbbiek szintaxisára és képességeire vonatkozóan sokkal jelentősebb eltérésekre kell felkészülnünk. Megemlítendő még, hogy bár minden adatbázis-kezelő esetében korlátos a különböző típusok által kezelhető érték-tartomány (szöveges mezők maximális hossza, megengedett legnagyobb egész szám stb.), a táblák oszlopainak és sorainak számával kapcsolatban vagy nincs megkötés, vagy olyannyira magasra állították a határt, hogy gyakorlatilag végtelennek tekinthető.

Az elmondottak alapján az Olvasó bizonyára már sejti, hogyan ábrázolhatjuk táblák segítségével az adatmodellezésnél kialakított egyedeket. Nincs más teendőnk, mint hogy minden egyes egyedtípus számára külön-külön táblát hozzunk létre, amelyben az oszlopok a tulajdonságokat, a sorok pedig a típushoz tartozó egyedeket képviselik. Ebből következően az oszlopokat a tulajdonságokról, a táblát a hozzárendelt egyedtípusról célszerű elnevezni, így a későbbi SQL-parancsokban mindkettőre könnyen felismerhető formában tudunk hivatkozni. Miként lehet azonban magukra az egyedekre utalni, például amikor egy új kapcsolatot kívánunk bejegyezni, esetleg módosítani vagy törölni akarjuk valamelyik egyedet? Amennyiben az egyedek valóban egyediek, azaz nem létezhet két, egyazon típushoz tartozó és minden tulajdonságában azonos egyed, és ennek következtében a táblákban sosem fordulhat elő két, egymással teljesen egyező sor, az egyértelmű kiválasztás elvi feltételei adottak. Amire a gyakorlatban szükség van, az az úgynevezett elsődleges kulcsok (*primary key*) meghatározása.

Habár a relációs rendszerek egyáltalán nem tiltják, hogy egy táblában ugyanaz a sor többször szerepeljen, jól megtervezett adatbázisokban sosem találkozunk ilyesmivel. Sor megismétlése jelentős többletinformációt nem hordoz, amennyiben pedig tulajdonképpen az egyedek számát akarjuk így jelezni (mondjuk egy bolt árukészletét felsoroló táblában), azt jóval hatékonyabb módon megtehetjük egy darabszám vagy mennyiség nevű tulajdonság hozzáadásával.

Kulcsnak tekintendő a tábla oszlopainak bármely olyan együttese, amelynek összetett értéke alkalmas a sorok egyedi azonosítására. Ha a vizsgált táblában nincs ismétlődő sor (amit jelen alfejezet hátralevő részében mindig hallgatólagosan feltételezni fogok), összes oszlopa nyilvánvalóan kulcsot alkot; legtöbbször azonban ennél szűkebb, akár egyetlen oszlopból álló kulcsokat is találhatunk. Földrajzi adatbázisunkban az ország típusú egyedeket felsoroló táblában kulcs az ország nevét, betűjelét, vagy akár területét hordozó oszlop; utóbbinál persze elegendően nagy pontossággal kell rögzítenünk a számot, nehogy két országnál ugyanaz az érték szerepeljen. A városok listájánál viszont a városnév önmagában még nem használható kulcsként – hiszen például számos európai településnek van megfelelője az Egyesült Államokban –, csupán az országnévvel vagy -jellel kiegészítve. Ugyanakkor felesleges az online áruház beszállítóinak táblájában a vállalat nevén felül az adószámot is bevonni a kulcsba, amikor kettejük közül akármelyik egyedül ugyanolyan jól megállja a helyét.

A tábla kulcsai közül azt érdemes elsődleges kulcsnak választani, amelyel a leghatékonyabb módon lehet az egyedekre hivatkozni: álljon minél kevesebb oszlopból, ezek az oszlopok hordozzanak minél kisebb tárigényű értékeket, valamint legyen magától értetődő éppúgy a fejlesztők, mint a felhasználók számára. Az elsődleges kulcs egyszerűsége, illetve rövidege kedvező hatással van a relációs rendszer teljesítményére, hiszen egyrészt a tábla sorait főként ezen keresztül érjük el az adatfeldolgozás során, másrészt (emiatt) az adatbázis-kezelő az elsődleges kulcsokat egy különálló listában is nyilvántartja, amelyet a tábla módosításának megfelelően természetesen folyamatosan karban kell tartania. Gyakran az ábrázolt tudásterület jellegéből fakadóan az egyedek már eleve rendelkeznek az elsődleges kulcs szerepének betöltésére kitűnően alkalmas tulajdonsággal: személyeknél a személyi igazolvány száma, gépkocsiknál a rendszám, megrendeléseknél az iktatási sorszám és így tovább. Amennyiben viszont az egyediséget hosszabb szöveges tulajdonsággal, vagy több tulajdonság kombinációjával tudnánk csupán biztosítani, célszerű saját magunknak gondoskodnunk az egyedek beszámozásáról, egy kifejezetten erre a célra szolgáló új oszlop hozzáadásával.

Az elsődleges kulcs kijelölése nem kötelező, ám ekkor számítanunk kell arra, hogy jelentős mértékben lelassulnak a táblával kapcsolatos műveletek, ráadásul az adatbázis-kezelő új sorok hozzáadásakor nem ellenőrzi az egyediségre vonatkozó feltételek teljesülését. Márpedig ezeknek a feltételeknek a megsértése nehezen felderíthető hibákhoz vezethet, és akár rendkívül súlyos anyagi károkat is okozhat. Gondoljunk csak el, mi történe akkor, ha ugyanazt a vásárlót véletlenül két különböző laccímmel jegyeznénk be! A reklámánya-

gok kétszeres kiküldése a legkevesebb; amikor viszont a megrendelt áru kétszer kerül kiszállításra, két különböző számla kíséretében, számottevő erőfeszítésbe kerülhet a vevő bizalmának helyreállítása és a felesleges szállítmány visszavétele. Bizonyos körülmények között indokoltá válhat az elsődleges kulcs ideiglenes eltávolítása: például külső forrásból származó adatok beolvasásakor, amelyekben előfordulhatnak ismétlődő vagy hiányos bejegyzések; vagy nagy mennyiségű információ egyszerre történő betöltésekor, ahol ezáltal csökkenthető a végrehajtás ideje. Ezek azonban többnyire nem részei az adatbázis üzemszerű működésének, és mindig az adminisztrátor vagy más, felelős szakember felügyelete mellett zajlanak.

A relációs rendszerek nem tesznek lényeges megkötést arra vonatkozóan, milyen típusú oszlopok szerepelhetnek a kulcsokban, és ennek következtében az elsődleges kulcsokban. Néhol effajta célra egyáltalán nem, vagy csupán korlátozott módon vehetők igénybe nagyméretű bináris és szöveges tartalom befogadására képes (*Large Object, LOB*) oszlopok; sok helyen követelmény, hogy mindegyik oszlop kötelezően kitöltendő legyen. Elsősorban szöveg vagy egész szám típusú értékekből tevődik össze a kulcs; dátum, időpont, logikai vagy felhasználó által kialakított felsorolás (titulusok, a hét napjai, színek stb.) típusúakkal ritkábban találkozunk, és általában sohasem állnak önmagukban. Valós számok alkalmazása egyáltalán nem javasolt, a számítógép ugyanis teljes pontossággal nem tudja tárolni őket, ezért időnként előfordulhat, hogy matematikailag elvileg egyenrangú számítások a gyakorlatban egymáshoz nagyon közeli, de eltérő bináris mintával ábrázolt eredményekhez vezetnek, ami gondot okozhat a sorok kulcs alapján történő beazonosításában.

A kapcsolatok leképzése leheletnyivel bonyolultabb, mint az egyedeké, a követendő módszer azok jellegétől függ. Egy-egy esetben, mondjuk amikor országokat és fővárosaikat csatlakoztatjuk össze egymással, még elég egyszerű a dolgunk: az egyik oldalnál (tipikusan ahonnan vélhetőleg gyakrabban „indulnak ki” majd későbbi lekérdezéseink) járulékos tulajdonság gyanánt átmutatunk a másik fél megfelelő elemeire, mégpedig a rájuk érvényes elsődleges kulcs révén. Miután a táblába újonnan bejegyzett kulcs nem a saját, hanem a hivatkozott egyedek azonosítására szolgál, így idegen kulcsnak (*foreign key*) nevezik. Persze az idegen kulcsot pontosan ugyanannyi és ugyanolyan típusú oszlop ábrázolja, mint amennyi és amilyenek a „távoli” táblában az elsődlegest. Az oszlopok elnevezéseit, akármilyen terjedelmessé váljanak is emiatt, lehetőség szerint úgy válasszuk meg, hogy első pillantásra kiderüljön szerepük. Például, ha a vásárlók általános adatai mellett utalni szeretnénk a számlájukhoz kötődő információkra, utóbbi a bank megjelölésével és a bank által önkényesen meghatározott kóddal behatárolva, akkor a SZÁMLA\_BANK, SZÁMLA\_SZÁM kielégítő megoldás lehet. Valódi rekordokkal a következőképpen lehet szemléltetni a két tábla felépítését, illetve viszonyát:

ÜGYFÉL:				BANKSZÁMLA:		
VÁROS	NÉV	SZÁMLA_BANK	SZÁMLA_SZÁM	BANK	SZÁM	LEVONÁSI-LIMIT
SZEGED	KIS ANTAL	ALFA	994035	ALFA	994035	250000
BUDAPEST	NAGY JENŐ	BETA	45661	BETA	45661	100000
BUDAPEST	SZABÓ PÁL	ALFA	781053	ALFA	781053	20000

Bár elvileg kivitelezhető, hogy a fenti szerkezetben mindkét fél utaljon a másiknak, tehát nemcsak az ügyfél a számlaszámra, hanem fordítva szintén, a számlaszám az ügyfélre, a gyakorlatban csupán ritkán találkozunk ilyesmivel. Meglehetősen kényelmetlen két helyen is folyamatosan ellenőrizni, illetve karbantartani a kapcsolatokat, ráadásul az egyedtípusok között majdnem mindig megállapítható alá- és fölérendelt viszony, emiatt igazándiból sohasem fogjuk zavarban érezni magunkat, melyikükhöz toldjuk az idegen kulcsot.

Egy-több kötés megvalósítása, amilyen mondjuk termékek és kategóriáik rögzítése, roppant hasonló az imént tárgyalt egy-egy típuséra: a „több” oldalt megtestesítő táblát (példánkban a termékekét) kiegészítjük az „egy” számosságú egyedekre (a kategóriákra) hivatkozó idegen kulcsokkal. Az ellenkező irány most nyilvánvalóan nem járható, hiszen lehetetlen adott kategória mellé egyetlen vagy pár oszlopban felsorolni a hozzá beosztott összes terméket. Amikor netalán effajta listának éreznénk szükségét, kénytelenek vagyunk azt a termékek rekordjaiból képezni. Figyeljük meg, hogy egy-egy és egy-több csatolásakor voltaképpen egyedek valamelyik tulajdonságát (ország fővárosát, termék kategóriáját stb.) különálló táblába ragadjuk ki, akár mert el kívánjuk kerülni az ismétlődő értékeket, akár miután az egyik típusú információ gyakrabban változik a másiknál, akár mivel az adatbázis-szerkezet áttekinthetőségének megőrzése végett nem engedjük túlságosan nagyra duzzadni a táblánkon belüli oszlopszámot. Lássunk megint, miként fest a dolog a gyakorlatban:

TERMÉK:				KATEGÓRIA:		
KÓD	NÉV	ÁR	KATEGÓRIA_NÉV	NÉV	HASZONKULCS	RAKTÁR
ZT92AF	KERTI ASZTAL	8500	BÚTOR	BÚTOR		15 KÜLSŐ
AN04KL	KERTI SZÉK	4200	BÚTOR	MŰSZAKI		20 BELSŐ
TB11HG	FALI LÁMPA	6350	MŰSZAKI			
DV42JJ	DVD-LEJÁTSZÓ	13000	MŰSZAKI			
EW85PO	GÁZGYÚJTÓ	780	MŰSZAKI			

Végül, több-több kapcsolat ábrázolásához (rendelések és áruk, személyek és általuk hívott telefonszámok, diákok és vizsgaalkalmak) menthetetlenül járulékos táblát kell felvinnünk, amelynek bejegyzései a két oldal egy-egy elemét párosítják, azokra szokás szerint idegen kulcsokkal hivatkozva. Feltétlenül úgy kereszteljük el, hogy látszódjon, nem hagyományos, hanem csatolásokat hordozó táblával van dolgunk, és ami fontosabb, kiderüljön, milyen egyedek milyen viszonyát közli, hiszen könnyen előfordulhat, hogy ugyanazon két egyedtípus többféleképpen is kötődhet, például személyek akár vizsgázóként, akár vizsgáz-

tatóként, akár felügyelőként is jelen lehetnek a vizsgákon. Név gyanánt sokan a kérdéses egyed típusok aláhúzásjel segítségével elválasztott megnevezését tartják megfelelőnek (ÜGYFÉL\_HÍVÁS, FELÜGYELŐ\_VIZSGA), vagy a kapcsolat rövid szövegét (FELHÍVTA, VIZSGÁZIK), esetleg mindkettőt egyidejűleg, noha ez valószínűleg már túlságosan bőbeszédű (HÍVÁS\_ÜGYFÉL\_TELEFONSZÁM). Szemléltetésül:

DIÁK:		DIÁK_VIZSGA:		VIZSGA:	
NÉV	ÁTLAG	DIÁK_NÉV	VIZSGA_NÉV	NÉV	DÁTUM
SZABÓ ANNA	4.3	SZABÓ ANNA	MATEMATIKA	MATEMATIKA	2007-01-03
KISS JÓZSEF	5.0	SZABÓ ANNA	FIZIKA	FIZIKA	2007-01-23
NAGY JUDIT	3.7	KOVÁCS JENŐ	MATEMATIKA	KÉMIA	2007-02-10
KOVÁCS JENŐ	4.8	KOVÁCS JENŐ	KÉMIA	TÖRTÉNELEM	2007-02-11
		KOVÁCS JENŐ	TÖRTÉNELEM	IRODALOM	2007-01-03

Talán sokak számára meglepőnek tűnik a gondolat, de kapcsolati táblában szintén definiálhatunk elsődleges kulcsot, igaz, ez törvényszerűen magában foglalja az összes oszlopot, hiszen önmagában egyik idegen kulcs sem kielégítő a feladatra (ne felejtjük el, itt bármely oldal egyedei tetszőleges számú alkalommal megisméltődhetnek). Elvileg ugyan bizonyos mezőiket kiragadva sajátos azonosítót kotyvaszthatunk, ezt mégsem javaslom, miután egyfelől jelentése valószínűleg nehezen lesz érthető a forráskódot tanulmányozók számára, másfelől egyáltalán nem garantált, az adatbázisban tárolt információk jellegének módosulásával működésképes marad. Megjegyzem, az elsődleges kulcs felvétele nem pusztán esztétikai kérdés, hatására lényegesen felgyorsulhat az egymáshoz rendelt egyedek előkeresése, mondjuk, ha meg akarjuk tudakolni, kik vizsgáznak irodalomból, vagy hogy adott diák mely tantárgyakból fog idén megmérettetni.

Csábító lehet utólag különféle kiegészítő adatokat toldani az idegen kulcsok mellé, például a fenti esetben az illető által a vizsgán szerzett osztályzatot, a feladatok megoldására fordított idejét stb. Így azonban voltaképp új egyed típust hozunk létre, a „levizsgálás”-t, amely egy-egy módozattal kötődik mind a diákokhoz, mind pedig a vizsgaalkalmakhoz, és a korábbi kapcsolati tábla immáron ezen egyedek gyűjtőhelyévé lép elő. Ez nem baj, ám ha ilyen irányban halad tovább az Olvasó, ne mulassza el a táblanév módosítását!

Következzen most pár széljegyzet. Egyrészt megeshet, valamely hivatkozást nem minden egyednél értelmes kitölteni: rendeléseknél a szállítási címet felesleges meghatározni, amennyiben megegyezik a számlázásával; lehet, hogy a termékhez nincs raktár kijelölve, mivel közvetlenül az üzletből elvihető; a telek adatait kizárólag házaknál, nyaralóknál szükséges rögzíteni, lakásoknál nem; és így tovább. Nos, ekkor a szóban forgó idegen kulcshoz nemes egyszerűséggel a különleges NULL értéket szokás beírni (ugyanakkor ügyeljünk, elsődleges kulcsnál tilos effajta jelölést használnunk, kivéve néhány, a hivatalos szabványtól elhajló adatbázis-kezelőt). Ugyanígy járunk el, amennyiben a rekord beillesztésekor még nem ismert a hozzácsatolandó elem, tipikusan azért, mert azt egy következő lépés folyamán fogjuk összeállítani. Való-

jában azonban tetszőleges számú egyed összekapcsolható, nem feltétlenül csupán kettőt. Gondoljunk a rendezvényszervezőhöz beérkező igényekre, amelyekben cégek, helyszínek, témák, időpontok „metszik” egymást, a kapcsolati táblában ilyenkor persze négy utalást kell bevinni.

Harmadrészt az egy-egy és az egy-több típusú csatolások ábrázolásához szintén felvehetünk külön kapcsolati táblát, ami első ránézésre talán butaságnak tűnhet, hiszen növekszik a lefoglalt tárterület és az adminisztrációs teher, de szélsőséges körülményeknél valóban jól jöhet. Tegyük fel, logisztikai vállalat gépkocsiparkjának nyilvántartását építjük fel, a tartálykocsikra meghatározva jelenlegi sofőrjüket, célállomásukat, rakományukat. Miután maguk az egyedek viszonylag ritkán változnak, viszont egymáshoz kötésük annál többször, emiatt a TEHERAUTÓ tábla rekordjainak (pontosabban azon belül a sofőrökre, a városokra, az árutípusokra utaló idegen kulcsok) folytonos módosíthatásánál lényegesen hatékonyabb megoldás a kisebb terjedelmű, ennélfogva akár teljes egészében a memóriában tartható SOFŐRJE, CÉLJA, SZÁLLÍTMÁNYA kapcsolati táblákat birizgálni. Végül nem árt tudnunk, semmi sem akadályozza, hogy idegen kulcs az őt hordozó táblába mutasson, például amikor munkatársakat és helyetteseiket listázzuk:

SORSZÁM	VEZETÉKNÉV	KERESZTNÉV	OSZTÁLY	HELYETTES_KÖD
045	SZABÓ	JÁNOS	BESZERZÉS	NULL
901	KISS	MAGDA	OKTATÁS	437
437	NAGY	ELEMÉR	OKTATÁS	NULL
008	KOVÁCS	BÉLA	PÉNZÜGY	045

Az iménti szerkezet két szempontból tanulságos. Elsődleges kulcs gyanánt a vezeté- és keresztnév együttese helyett, amelyet felületes szemlélő ugyan alkalmasnak ítélt ilyen célra, a biztonság kedvéért mégis inkább saját gyártmányú sorszám bukkan fel. Emellett, szemben a korábbi táblákkal, ahol az idegen kulcsok neveit a hivatkozott egyedtypusból (SZÁMLA, KATEGÓRIA, DIÁK stb.) és táblájukban az elsődleges kulcsot hordozó oszlopok azonosítóiból képződtek (amilyen BANK, SZÁM, NÉV volt), itt az előbbit az egyed szerepének tömör jellemzése váltja fel, hiszen a MUNKATÁRS teljességgel semmitmondó lenne. Természetesen a megoldáshoz nemcsak önrámutatáskor érdemes folyamodni, hanem akkor is, ha ugyanolyan típusú egyedre többször kell utalni, például ingatlan átruházása esetén személyekre, mind eladó, mind vevő minőségében. A jól eltalált címkék roppant fontosak, lévén az adatbázis szerkezetének, belső összefüggéseinek megértésén túl az adatokat lekérdező, illetve megváltoztató SQL-parancsok szövegének áttekinthetőségét szintén előmozdítják.

A relációs rendszerek egy részénél táblák tartalma megegyező elnevezésű oszlopaikon keresztül roppant kényelmesen összekapcsolhatók (lásd a SELECT parancsról szóló alfejezetben majd ismertetésre kerülő NATURAL záradékot), ám ez véletlenül se ösztökélje az Olvasót arra, hogy a táblára vonatkoztatott idegen kulcsnál megismételje elsődleges kulcsának nevét.

Nyilván gondoskodnunk kell róla, a táblák más táblák rekordjaira utaló oszlopai, illetve oszlopkombinációi (vagyis az idegen kulcsok) érvényesek maradjanak, akármilyen műveleteket foganatosítsunk is az adatokon; ezt hívják hivatkozási épségnek (*referential integrity*). Magyarán amikor valamelyik sort eltávolítjuk, az oda mutató összes idegen kulcsot állítsuk az üres értéket képviselő NULL mennyiségre, esetleg töröljük magukat a hivatkozó rekordokat, miután archiválásuk megtörtént; ha pedig netalántán módosul az elsődleges kulcs, a változtatásokat a vonatkozó idegen kulcsokon szintén elvégezzük. Szerencsénkre az adatbázis-kezelők nagy része automatikus mechanizmusok segítségével támogatják a hivatkozási épség megtartását, jelentős terhet emelve le a fejlesztők válláról. Megjegyzendő, megszorításokat (*constraint*) nem kulcshoz tartozó oszlopokra éppen úgy érvényesíttethetünk. Megkövetelhetjük, hogy a beírt érték megszabott tartományba essen (rendelt termék darabszáma legalább 1), általunk elvárt viszonyban legyen a rekord más mezőivel (bruttó ár 20%-kal nagyobb a nettónál), vagy a tábla egyéb bejegyzéseiben ne ismétlődjön meg (személyi szám olyan táblában, ahol elsődleges kulcsnak a társadalombiztosítási azonosítót választottuk).

Ha az elsődleges kulcs több oszlopot foglal magában, vagy az üzemszerű működés folyamán módosulhat (ilyenek a személynevek), célszerűbb helyettesítő kulcsra (*surrogate key*) cserélni. Helyettesítő kulcsként tipikusan egy automatikusan képzett, a modellezett üzleti folyamat számára semmifajta jelentéssel sem bíró egész számot alkalmaznak. Ennek köszönhetően az idegen kulcsok mérete csökken, rövidebb idő alatt előkereshetők az általuk mutatott rekordok, sőt, miután a korábbi, elsődleges kulcsra foganatosított javítgatások immár közönséges mezőkre vonatkoznak, nem kell törődnünk a kapcsolódó idegen kulcsok velük összhangban történő átírásáról.

Az adatmodellezés és a táblák megtervezése kezdetben talán reménytelenül bonyolultnak látszódhat, ám ahogyan az Olvasó egyre több és több tapasztalatra tesz szert, idővel képessé válik a szükséges adatszerkezeteket közvetlenül a táblák szintjén megfogalmazni, bármiféle tudatos elméleti előkészítés, azaz az egyedtípusok, tulajdonságok és kapcsolatok meghatározása nélkül.

### 1.3. Relációs adatbázis-kezelők

A relációs adatbázisokat az élet számos területén alkalmazzák, gyakran nagyon különböző feladatokra, és (részben ennek következtében) rendkívül eltérő üzemeltetési körülmények között. Nyilvánvalóan más követelményeknek kell eleget tennie a kedvenc World Wide Web címeiket a személyes honlap látogatói számára kilistázó alkalmazásnak, mint egy naponta több százezer tranzakciót lebonyolító banki rendszernek, vagy az ügyfelek vásárlási szokásaiban évtizedekre visszamenőleg mintákat kereső és azokat grafikusán is megjelenítő programnak. A kezelhető adattípusok köre és az SQL-parancsok

választékának gazdagsága csupán egyik oldala az éremnek – legalább ennyire fontos, hogy milyen tekintetben szabályozható az adatbázis működése, van-e lehetőség terhelésmegosztásra, lehet-e a célterület szempontjából hasznos kiegészítő modulokat a relációs rendszerhez kapcsolni, milyen és mennyire könnyen használható adminisztrációs eszközök állnak rendelkezésünkre és így tovább. Az adatfeldolgozás három alaptípusát szokás megkülönböztetni:

- Az OLTP (*On-Line Transaction Processing: online tranzakció-feldolgozás*) jellegű feladatok jelentik a relációs technológia klasszikus és talán leggyakoribb alkalmazási területét; ide sorolhatók például szállodai szobafoglalásokat nyilvántartó, vagy a feljegyzett rendelések teljesítéséért felelős rendszerek. Az adatokon végrehajtandó műveletek (tranzakciók) többnyire egyszerű olvasások vagy módosítások, ám nagyon nagy számú felhasználótól érkeznek, rendkívül sűrűn, ráadásul általában a nap minden órájában, szünet nélkül. Csupán olyan adatbázis-kezelő állja meg a helyét, amely nagy forgalomnál is rövid válaszidőt biztosít, szükség esetén egymással párhuzamosan végrehajtva a műveletek. Ezenfelül gondoskodnia kell róla, hogy hardware vagy software meghibásodásnál tartalék erőforrások bevonásával zavartalanul folytatódjon az adatfeldolgozás, de legalábbis minél hamarabb helyre lehessen állítani az adatbázis épségét, minimális adatvesztéssel.
- Az OLAP-alkalmazások segítségével a vállalat hosszú távú terveinek kidolgozásáért felelős, vagy a korábbi kezdeményezések sikerességét elemző szakértői különféle szempontokból és módszerekkel vizsgálják a több hónapra és évre visszamenőleg rögzített információkat: eladási számokat, forgalmi statisztikákat, gyártás során mért minőségbiztosítási adatokat stb. Az ilyen célú adatbázisok, az úgynevezett adattárházak (*data warehouse*), naponta vagy hetente egy-két alkalommal összegyűjtik és integrálják az OLTP jellegű adatbázisok tartalmát, és csak olvasható formában néhány személy számára hozzáférhetővé teszik. Mivel a számítások sokszor több millió rekordot érintenek, a rövid válaszidő nem követelmény, annál inkább a hatékony adatfeldolgozás, amelyhez egyrészt az elemzést támogató SQL-parancsokra, másrészt több számítógép között felosztható működésre van szükség.

Habár léteznek relációs technológián alapuló OLAP-rendszerek is, az efféle tevékenységhez sokkal inkább multidimenziós adatbázisokat használnak, ahol az elemi információk nem sorok és oszlopok, hanem  $n$ -dimenziós tömbök formájában tárolódnak. Egy bolthálózat eladási adatainál a dimenziók többek között az időnek, helynek, terméknek és vásárlónak fognak megfelelni, a számok bármely tengely mentén összegezhethők, átlagolhatók stb.

- Az OLCP (*On-Line Complex Processing: online összetett feldolgozás*) viszonylag új keletű fogalom olyan összetett adatokon végzett számításookra, mint például a geometriai objektumok, nagyméretű szöveges dokumentumok, fényképek, video- és hangfelvételek. Elektronikus könyvtár, szárazföldi távközlési vonalakat karbantartó vállalat, fotóügynökség – hogy csupán néhányat említsek a számtalan felhasználási lehetőség közül. Az, hogy az adatbázis képes kellő gyorsasággal eltárolni, illetve előhívni a sokszor több millió byte méretű mezőket, önmagában még nem elegendő; rendelkezésre kell bocsátania azokat a műveleteket is, amelyekkel az adatok átalakíthatók, egymással összevetethetők, tulajdonságaik megvizsgálhatók. Egy képarchívumban hasznos szolgáltatás, ha valamelyik képhez témában hasonlókat tudunk keresni, vagy ha meg tudjuk mérni két hálózati csomópont távolságát.

A fejlettebb adatbázis-kezelők a három szerepkör bármelyikébe bele tudnak helykedni működési paramétereik megfelelő beállításával. Ezenkívül eleve tartalmazzák, vagy külön megvásárolható termékeként csatlakoztathatók hozzájuk az egyes szerepkörökre jellemző tevékenységek elvégzését segítő kiegészítő programok, eszközök. Léteznek persze specializálódott, saját területükön igen nagy határfokkal dolgozó rendszerek, azonban ha a megoldandó terület túlságosan kiterjedt ahhoz, hogy lelkiismeret-furdalás nélkül besorolható legyen a típusok egyikébe, válasszuk inkább a lassabb, de univerzálisabb rendszert. Ilyen módon a fejlesztőknek nem kell több SQL „tájszólás”-t, programozási és kezelési felületet elsajátítaniuk, ráadásul valószínűleg kevesebb gondot fog okozni a különböző alrendszerek szoros együttműködésének biztosítása.

Most röviden tekintsük át, milyen szolgáltatásokkal találkozhatunk a relációs rendszereknél! Még ha ezeknek csupán töredéke áll is rendelkezésünkre a nyílt forráskódú adatbázis-kezelőknél, az Olvasó képet alkothat a jelenleg általánosan hozzáférhető technológia szintjéről, és ennek alapján a könyvben bemutatott programok valószínű fejlődési irányáról. A szolgáltatásokat, szerteágazó mivoltuk miatt, kénytelen voltam csoportokba foglalni, mégpedig az alábbiak szerint:

*Adatfeldolgozási szolgáltatások* alatt a felhasználható adattípusok és műveletek körét értem. Szöveget, dátumot és/vagy időpontot, egész számot, lebegőpontos számot, nagyméretű szöveges, illetve bináris értékeket hordozó oszlopok szinte minden rendszerben létrehozhatók. Kevésbé elterjedtek a logikai, XML- és felhasználó által definiálható típusok, valamint a vektorok, tömbök, illetve a tulajdonságokkal és metódusokkal ellátott objektumok mezőként való tárolása. A szokásos aritmetikai, trigonometriai, dátum-, konverziós és szöveges műveleteken kívül sok helyen lehetőségünk van a nagyméretű értékek közvetlen kezelésére (keresés, részlet kiemelése, módosítás stb.). Különösen figyelemre méltóak az úgynevezett OLAP-műveletek: segítségével a tábla sorait tetszőleges szempontból szakaszokra bonthatjuk, majd a szakaszokon belüli adatokat például külön-külön rangsorolhatjuk, legnagyobb és legkisebb értéküket megállapíthatjuk, bennük mozgó átlagot képez-

hetünk. Bizonyos rendszerekben egész táblákat összehasonlíthatunk egyetlen operátorral, halmazokként tekintve őket kiszámíthatjuk metszetüket, különbségüket vagy akár egyesítésüket. Különösen nagyméretű, eltérő szerepkörű és ennek megfelelően többé-kevésbé függetlenül üzemeltetett adatbázist átfogó architektúráknál lehet hasznos, ha ugyanabban az SQL-parancsban több adatbázist is megszólíthatunk.

Az *adatszerkezési szolgáltatások* körébe tartoznak a táblákban szereplő információk könnyebb vagy gyorsabb elérésére, kezelésére irányuló eszközök. Egy közönséges táblában a sorok előhívása akkor lesz a leghatékonyabb, ha a szűrési kritérium elsősorban az elsődleges kulcsban részt vevő oszlopok értékeire támaszkodik. A rendszer ugyanis, mint korábban említettem, ezeket az értékeket a hozzájuk tartozó sor címével együtt egy rendezett listában, az indexben tartja nyilván, amelynek segítségével azután a kért sort vagy sortartományt néhány lépésben, a teljes tábla végigolvasása nélkül meg tudja találni. Indexeket azonban nemcsak az elsődleges kulcsra, hanem bármely más oszlopokra, illetve oszlop csoportra ki lehet alakítani – függetlenül attól, hogy az általuk jelölt értékegyüttesek egyediek-e a sorok között vagy sem. Hasonló célt szolgál, bár gyökeresen eltérő elven alapul, a testet öltött nézet (*materialized view*). Akkor folyamodunk hozzá, mikor valamely, egy vagy több tábla tartalma alapján összeállított adatsort (mondjuk a fővárosi ügyfelek listáját) gyakran használjuk fel a feldolgozás során kiadott SQL-parancsokban, azonban túlságosan időigényes lenne azt minden egyes alkalommal újra kiszámítani. A testet öltött nézet kialakításával voltaképpen egy táblát hozunk létre, automatikusan feltöltve azt a kérdéses adatsorral; amennyiben az adatsor alapját képező információk módosulnak, az adatbázis-kezelő gondoskodik a nézet frissítéséről, levéve ezt a terhet a fejlesztő válláról.

Az elsődleges kulcs, valamint az oszlopokra meghatározott adattípusok csak bizonyos mértékig tudják megakadályozni nem megengedett értékek bekerülését a táblába. Hiába írjuk elő, hogy az EMELET oszlopban csak egész szám lehet, a rendszer nem fog tiltakozni, amikor egy hibásan megírt alkalmazás vagy egy figyelmetlen felhasználó ide 2000-et vagy -10-et jegyez fel. Hiába követeljük meg, hogy a vállalatok listájában a cégneveknek mindenképpen egyedieknek kell lenniük, ha az ABC Kft. hol „ABC”, hol „Abc” megjelöléssel szerepel. Az *adat-ellenőrzési szolgáltatások* éppen ennek a nehézségnek a megoldására törekednek, lehetővé téve további szabályok megfogalmazását, majd pedig szigorúan ügyelve azok betartására. Két eszköz érdemel itt említést. A megszorítás (*constraint*) tulajdonképpen olyan, a kérdéses táblába beillesztendő sor mezőire, vagy az adatbázisban található bármely más tábla oszlopaira vonatkozó feltétel, amely, ha nem teljesül, a sor felvételét az adatbázis-kezelő megtagadja. A triggerek (magyarul talán beavatkozókat mondhatnánk) feladata nem annyira az ellenőrzés, mint inkább az adatok következetességének biztosítása: amennyiben a beszúrandó, a törölendő vagy a módosítandó sorra teljesül a megadott feltétel, adott tevékenységi sor kerül végrehajtásra.

Így egyszerűen megoldható, hogy amikor törölünk egy terméket az árulistából, a rendszer automatikusan eltávolítsa a többi táblából a hozzá kapcsolódó információkat, vagyis erről nem magunknak kell gondoskodnunk az adatfeldolgozást végző program minden egyes, törléssel kapcsolatos pontján.

Az *alkalmazásfejlesztési szolgáltatások* az adatbázisban tárolt adatok feldolgozását végző programok fejlesztésében, tesztelésében és hibajavításában segítenek bennünket. Az egyszerűbb rendszerekben pusztán egy programozási felületet kapunk, amelyekben a dokumentációban megszabott módon és sorrendben kell meghívunk az SQL-parancsok kiadásához és az általuk szolgáltatott eredmények beolvasásához. Az előfeldolgozó (*preprocessor*), amelyben esetünkben egyáltalán hozzáférhető, jelentős gépeléstől és számtalan hiba elkövetésétől menthet meg minket azáltal, hogy a forráskódba közvetlenül beírt SQL-parancsokat lefordítja a szükséges változódeklarációkká és eljárás-hívásokká. Még ennél is kényelmesebb a tárolt eljárások (*stored procedure*) használata: szinte minden SQL-parancs szerepelhet bennük bármifajta hókuszpókusz nélkül, és ráadásul, miután végrehajtásuk magában az adatbázis-kezelőben történik, sebességük felülmúlja a külső eljárásokét. Sok esetben a tárolt eljárásokból gazdag eljáráskönyvtár érhető el, a különleges elbánást igénylő mezők (XML, nagyméretű szöveges értékek stb.) kezeléséhez és a külvilággal való kapcsolattartáshoz.

A *biztonsági szolgáltatások* feladata a táblákban tárolt információkhoz, illetve az adatbázis különféle elemein elvégezhető műveletekhez való hozzáférés szabályozása. Egyrészt a rendszer felhasználóinak engedélyezhetjük vagy megtilthatjuk bizonyos táblák adott oszlopainak olvasását, módosítását, a tábla sorainak törlését, oda új sorok beszúrását. Másrészt meghatározhatjuk, kinek van joga megváltoztatni az adatbázis szerkezetét, azaz ki hozhat létre és törölhet táblákat, nézeteket, indexeket és így tovább. Az adminisztráció egyszerűsítése érdekében általában lehetőségünk van mind a felhasználókat, mind a jogokat csoportokba foglalni, így amikor meg kell határoznunk egy nagyobb részleg munkatársainak jogosultsági profilját, nem kell a személyek minden egyes engedélyét külön SQL-parancsban kérnünk. Néhány rendszerben még azt is korlátozhatjuk, legfeljebb milyen erőforrásigényű parancsokat adhat ki a felhasználó és óránként legfeljebb mennyit, milyen erős titkosítási algoritmus használatával kell hozzákapcsolódnia az adatbázishoz, vagy hogy bejelentkezéskor hogyan kell azonosítania magát.

A *mentési és visszaállítási szolgáltatások* szerepe az, hogy segítségükkel az adminisztrátor rendszeres időközönként másolatot tudjon készíteni az adatbázisról, amelyek egy esetleges későbbi hardver- vagy szoftvermeghibásodásánál – a folyamatosan készített naplózási állományokkal (*log file*) együtt – felhasználhatók az üzemszerű működés helyreállítására. Helyenként a mentéshez (*backup*) a rendszer teljes leállítására van szükség, máshol ez megoldható a felhasználók kiszolgálásával párhuzamosan. Megemlítendő még az archiválás, vagyis a táblákban szereplő elavult, egyre ritkábban használt adatok áthelyezése olcsóbb, és ennél fogva jellemzően lassabb tárolóeszközre (pél-

dául szalagra, CD-ROM-ra); néhány adatbázis-kezelőnél az archiválás egy kisebb teljesítményű „háttér” adatbázisba is történhet, melyben az információk ugyanolyan strukturált módon elérhetőek maradnak. Végül ebbe a csoportba tartozik az adatok exportálása és importálása, ami szolgálhatja más rendszerekkel való kapcsolattartást (adattárházak, statisztikai elemzést végző alkalmazások), vagy egy másik helyen működtetett adatbázis feltöltését (World Wide Web lekérdezőfelület, teszrendszer).

Az *üzemeltetési szolgáltatások* részben az adatbázis karbantartását, hangolását és adminisztrációját támogató, részben annak több számítógépen való üzemeltetését biztosító eszközöket foglalja magában. Az első témakörben sok mondanivalóm nincs, az adatbázis-kezelők távoli felügyelete és a grafikus, akár egy egyszerű World Wide Web böngészővel megnyitható kezelési felületek manapság teljesen hétköznapiak számítanak; sokkal jelentősebb különbségeket tapasztalhatunk a második pontban. Alapvetően két okból merül fel egynél több számítógép, más néven csomópont (*node*) alkalmazása: vagy a terhelést kívánjuk ilyen módon elosztani (*load balancing*), vagy valamilyen szintű hibátűrést (*fault tolerance*) megvalósítani. Előbbinél a csomópontok működtethetik az adatbázis egy-egy teljes példányát, időről időre tájékoztatva a többieket a rajtuk külön-külön végzett módosításokról, de akár fel is oszthatják az adatbázist egymás között, más-más táblákat vagy a táblák más-más sorait kezelve. Utóbbinál az elsődleges csomópont felelős a felhasználók kiszolgálásáért és azért, hogy a többiek tökéletesen tükrözzék tartalmát. Így amennyiben valamiért felmondaná a szolgálatot, a másodlagos csomópontok bármelyike át tudja venni szerepét, elsődlegessé válva ezáltal, mégpedig anélkül, hogy ebből a felhasználók bármit is észrevennének.

Végül a *különleges alkalmazási területekre szabott szolgáltatások* célja, hogy megkönnyítsék a relációs rendszer illesztését valamilyen egyedi feladatra, akár az SQL-parancsokban közvetlenül hozzáférhető új adattípusok, műveletek és függvények bevezetésével, akár különféle segédprogramok vagy kezelési felületek biztosításával. A geometriai információval foglalkozó adatbázisokban például rendkívül hasznos, ha el tudjuk dönteni, vajon az egyik elem teljes mértékben lefedi-e a másikat, vagy ha elő tudjuk keresni az összes olyan elemet, amely egy adott elemtől meghatározott távolságra helyezkedik el. Elektronikus dokumentumok tárolása esetén szükség van az összes szöveges tartalom gyorsan és pontosan keresni képes algoritmusra, amely megfelelően kezeli a ragokat, ezenfelül alkalmas az egyszerű kulcsszavak felsorolásán túlmutató keresési feltételek feldolgozására is. Adattárházaknál elengedhetetlenek az OLTP-adatbázisokból adatokat kinyerő, azokat egységes és integrált formára átalakító, majd pedig tárházba betöltő alkalmazások – ezt a három lépésből álló folyamatot egyébként ETL-nek (*Extract-Transform-Load: kivonatot-átalakít-betölt*) nevezik. Természetesen sok más kiegészítés létezik, amelyek pusztán felsorolása is több oldalt tenne ki, ezért az Olvasónak be kell érnie az iménti rövid ízelítővel.

Az említett szolgáltatások közül azokról, amelyek valamilyen szinten elérhetők a könyvben szereplő adatbázis-kezelők bármelyikében, bővebben beszélni fogok a későbbi fejezetekben. Amennyiben az Olvasó a többi lehetőségbe, azok pontos megvalósításába és használatának részleteibe is betekintést kíván nyerni, tanulmányozza a „nagy” rendszerek (Oracle 10g, IBM DB2, Sybase SQL Anywhere stb.) interneten ingyenesen elérhető kézikönyveit.

## 1.4. Egyéb modellek

A relációs modell korántsem az egyetlen módszer az ismereteinek ábrázolására, még ha jelenleg a leginkább elterjedtnek tekinthető is. Előfordulhatnak olyan feladatok, ahol egy teljes értékű relációs adatbázis-kezelő feleslegesen összetettnek, és ennek következtében túlságosan erőforrás-igényesnek bizonyulna, vagy ahol a feldolgozandó tudásterületet természetéből kifolyólag csupán meglehetősen körülményesen lehetne táblákra, sorokra és mezőkre levetíteni. Sokszor az újonnan bevezetendő relációs rendszernek illeszkednie kell a vállalatnál már meglévő, nem relációs adatbázisokhoz; a fejlesztő számos kellemetlenségtől kímélheti meg magát, ha előzetesen tájékozódik azok szerkezetéről, sajátos logikájáról és programozási módjáról. Az alábbi listában hely hiányában természetesen nem sorolhatom fel a számítástechnika története során kidolgozott összes kidolgozott modellt, csupán közülük a leginkább elterjedteket. Az Olvasó az irodalomjegyzékben megtalálja azoknak a kézikönyveknek a címeit és elérhetőségét, amelyekből részletesebben elsajátíthatja az egyes tárolási modellek, illetve a hozzájuk kapcsolódó adatbázis-kezelők használatát.

- Az adatrögzítés talán leginkább magától értetődő módszere az, amikor a feldolgozni kívánt ismeretanyagot egységes szerkezetű rekordokra bontjuk, és a rekordokat adott sorrendben kiírjuk egy egyszerű állományba (*flat file*). A rekord mezőinek hossza lehet előre megszabott vagy változó. Az utóbbi esetben hatékonyabb ugyan a helykihasználás, hiszen szükségtelen a rövidebb értékek szóközökkel, illetve zérusokkal való „kipárnázása”, azonban gondoskodnunk kell a mezőhatárok egyértelmű megjelöléséről. A rekordokban szereplő mezők száma hasonlóképpen lehet akár állandó, akár változó; amennyiben a kettő közül a másodikat választjuk, az adatokat beolvasó program számára utalnunk kell arra, mely mezőből van jelen több példány, esetleg melyek hiányoznak a rekordból.
- Az egyszerű állományok legnagyobb hátránya, hogy a rekordokat kizárólag kiírásuk sorrendjében lehet feldolgozni, közvetlenül nem érhetjük el őket. Ennek biztosításához a mezők egy részét kulcsként kell kijelölnünk, majd egy önálló állományban, az afféle tartalomjegyzékként szolgáló indexben kell felsorolnunk ennek a kulcsnak a lehetséges értékeit, a hozzájuk tartozó rekordok címeivel együtt. Ha ezen-

felül a rekordok hossza egységes, módosításuk „helyben”, az adatokat hordozó állomány teljes frissítése nélkül is elvégezhető; legfeljebb a jóval kisebb méretű index újrendezésére lehet szükség, amennyiben a változtatás érintette valamelyik kulcsmezőt. Számos programozási nyelv (köztük a COBOL) magára vállalja az indexelt állományok karbantartását, és különleges utasításokkal segíti az azokban tárolt adatok kényelmes kezelését.

- A kulcs-érték párokon (*key-value pairs*) alapuló modell tulajdonképpen az indexelt állományok ötletének szélsőséges leegyszerűsítése: az adatbázisban akárhány tábla, a táblákban akárhány rekord lehet, a rekordok azonban mindig pontosan két, a kulcsot és a hozzárendelt értéket tároló mezőből állnak. Mire használható egy ilyen mértékben korlátozott rendszer? Először is, a fejlesztőnek nem kell feltétlenül egyszerű értéként értelmeznie a két, amúgy bármilyen hosszúra méretezhető mező tartalmát az adatfeldolgozó programokban. Másodszor, éppen a renkívül egyszerű felépítés következtében, az adatbáziskezelő teljesítménye jelentősen felülmúlja relációs társaiét, különösen, ha az index tárolására a rendelkezésre álló mechanizmusok közül a feladathoz legjobban illeszkedőt választjuk. A legismertebb ilyen rendszer a Berkeley DB.
- A hierarchikus modellben (*hierarchical model*) a mezők rekordokba történő szervezésén felül lehetőségünk van magukat a különböző típusú rekordokat is elrendezni egymáshoz képest, mégpedig az elnevezésből adódóan értelemszerűen hierarchikus módon. Tekintsük például egy főiskola szakirányait. A szakirányokhoz tartozó szegmensnek (a hierarchikus modellben így hívják a rekordot) alá vannak rendelve egyrészt az ahhoz tartozó tantárgyak, másrészt az arra feliratkozott diákokat képviselő szegmensek. A tantárgy szegmens maga is rendelkezhet alárendelt szegmensekkel, mondjuk az előadást tartó tanárok, vagy a tárgyalt témához kapcsolódó irodalom felsorolása végett. Egy mező eléréséhez legfelülről elindulva meg kell adnunk a hozzá vezető szegmensek azonosítóinak sorozatát, egy tanár nevének módosításához a szakirány, a tantárgy és az előadás kódját, illetve megnevezését. A modell egyik ismert megvalósítása az IBM IMS terméke.
- A hierarchikus szemlélet csak körülményesen képes leképezni a több-több jellegű összefüggéseket, ezen próbál változtatni a hálózatos modell (*network model*), ahol a rekordok összekapcsolására úgynevezett halmazok szolgálnak. A halmazok egy tulajdonos, valamint egy vagy több résztvevő rekordot jelölnek ki; azt, hogy az első és második szerepkörben milyen típusú rekordok állhatnak, a halmaz típusa szabja meg, erőteljes tehát a hasonlóság a relációs modell egy-több kapcsolataival. Ugyanaz a rekord több, egymással megegyező halmazban is előfordulhat résztvevőként, vagyis tetszőleges számú „szülője” lehet,

ám közülük legfeljebb egyikben lehet tulajdonos. A hálózatos rendszerek (mint amilyen a nyílt forráskódú db.\*) másik előnye, hogy a mezők eléréséhez elég rekordjukat azonosítani, nem kell semmiféle útvonalat bejárjunk.

- Az objektumorientált modell (*object-oriented model*) célja biztosítani az objektumorientált programozási nyelvekben – mint a Java, a C++ vagy a Smalltalk – létrehozott objektumok és környezetük (az objektumok által hivatkozott egyéb objektumok) hatékony mentését, illetve visszaállítását, a fejlesztő számára minél egyszerűbb, természetesebb, észrevétlenebb módon. Számos rendszer (Orient, O2) ennél tovább megy, és tulajdonságaik alapján lekérdezhetővé teszi az objektumokat, általában az SQL-hez igen hasonló szintaxis segítségével. Az utóbbi években elmosódtak a relációs és az objektumorientált technológia határai: egyrészt léteznek már a relációs rendszerekhez objektumorientált szemléletű felületet nyújtó függvénykönyvtárak, másrészt fejlettebb adatbázis-kezelőkben a mezők teljes objektumok tárolására is alkalmasak.
- Az XML-szabvány rohamos terjedésének következtében először a relációs rendszereket tették képessé az ilyen formátumú dokumentumok kezelésére, majd hamarosan megjelentek a kifejezetten XML-re szakosodott, az adatokat fizikailag is ebben az alakban tároló adatbázis-kezelők. Miután az XML-dokumentumok maguk is szabályos szerkezetű, a hierarchikus modellhez hasonló módon felépített adatbázisoknak tekinthetők. Emellett a dokumentumok elemeihez való hozzáféréshez, azok lekérdezéséhez, illetve átalakítására korábban kidolgoztak már társszabványokat, ezekben az XML-alapú rendszerekben jelentős elméleti újítással nem találkozunk. Voltaképpen többnyire egyszerű dokumentumtárakról van szó (mint például a nyílt forráskódú 4Suite), némi hozzáférés-vezérléssel és alapszintű adminisztrációs eszközökkel kiegészítve.

Nagyon nehéz megjósolni, mit hoz a jövő, ám az teljes bizonyossággal állítható, hogy a relációs technológia még hosszú ideig jelentős tényező marad az adatbázisok világában, ha a jövőben is annyira rugalmasan tud igazodni az informatikában zajló változásokhoz, mint eddig. Ugyanakkor várható, hogy az objektumorientált rendszerek egyre inkább tért nyernek, részben az objektumorientált fejlesztési módszertan népszerűségének, részben egyszerű használatuknak köszönhetően, és néhány év elteltével ugyanolyan természetes összetevői lesznek a nyelvi környezetnek, mint a grafikus felület vagy az objektumok távoli elérése.