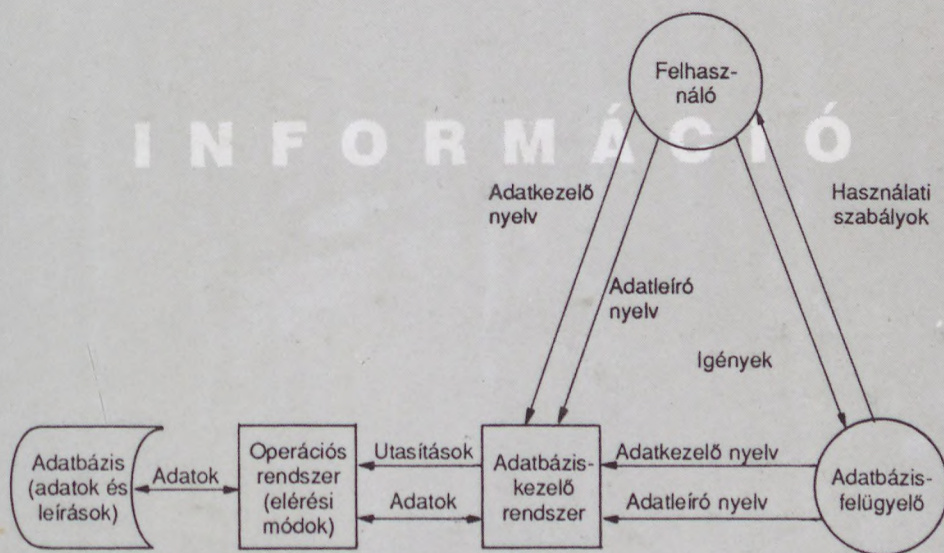


QUITTNER PÁL

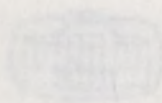
ADATBÁZIS-KEZELÉS A GYAKORLATBAN



AKADÉMIAI KIADÓ, BUDAPEST

2296
ADATBÁZIS-KEZELÉS A GYAKORLATBAN

ADATBÁZIS-KEZELÉS A GYAKORLATBAN



MAJTER KÖNYV- ÉS IRATVÁSÁRLÓ, BUDAPEST 1993

ES8802

22961

QUITTNER PÁL

Tartalomjegyzék

ADATBÁZIS-KEZELÉS A GYAKORLATBAN

MTAK



AKADÉMIAI KIADÓ, BUDAPEST 1993

503823

MAGYAR
TUDOMÁNYOS AKADÉMIA
KÖNYVTÁRA

ADATBÁZIS-KEZELÉS A GYAKORLATBAN

ISBN 963 05 6587 0

Kiadja az Akadémiai Kiadó, Budapest

© Quittner Pál, 1993

Minden jog fenntartva, beleértve a sokszorosítás, a nyilvános előadás, a rádió-
és televízióadás, valamint a fordítás jogát, az egyes fejezeteket illetően is.

Printed in Hungary

Tartalomjegyzék

Bevezetés

I. Információs rendszerek és adatbázisok

1. Információs rendszerek	17
1.1 Információs infrastruktúra	17
1.2 Az információs rendszerek célja	18
1.3 Az információs rendszerek részei	19
1.3.1 Adatok	19
1.3.2 Hardver	20
1.3.3 Szoftver	20
1.3.4 Felhasználók	25
1.3.5 Informatikai Központ	27
2. Adatbázisok	29
2.1 Különbéféle igények hatékony kielégítése	30
2.2 Adatfüggetlenség	30
2.3 Az adatbázis architektúra három szintje	31
2.3.1 Külső szint	33
2.3.2 Koncepcionális szint	33
2.3.3 Belső szint	33
2.4 Adatbázis-kezelő rendszer komponensei	34
2.4.1 Adatleíró nyelv	34
2.4.2 Adatkezelő nyelv	35
2.4.3 Illesztés a programozási nyelvekhez	36
2.4.4 Jelentéskészítő (Report generátor)	37
2.4.5 Képernyő-tervező	37
2.4.6 Szolgáltató program	38
2.4.7 Adatszótár (katalógus)	38
2.4.8 Adatbiztonság	39
2.5 Az integrált adatbázis előnyei	39
2.6 Adatbázis-felügyelő	40
2.7 Néhány hátrány	44
2.8 Személyi számítógépek adatbázis-kezelő rendszerei	45
2.9 Osztott adatbázisok	46
2.9.1 Osztott rendszerek	46

2.9.2 Osztott adatbázisok jellemzői	51
2.9.3 Előnyök és problémák	52
2.9.4 Az adatok elhelyezése	54
2.9.5 A konzisztencia biztosítása	55

II. Adatmodellek

3. Általános leírás	59
3.1 Az adatmodellek osztályozása	59
3.2 Főbb modellkészítési szempontok	60
3.3 Kapcsolattípusok	62
4. Fizikai tárolási és elérési mód	65
4.1 Általános szempontok	65
4.2 Mágneslemez	67
4.2.1 Fizikai jellemzők	67
4.2.2 Hozzáférési idő	68
4.3 Lapszerkezet	69
4.4 Elérési módok	73
4.4.1 Címzés	73
4.4.2 Alapvető elérési utak	73
4.5 Soros elérés	74
4.6 Szekvenciális szervezés és elérés	75
4.6.1 Listaszerkezet	76
4.6.2 A lista és a fizikai soros szerkezet összehasonlítása	80
4.7 Indexelés	81
4.7.1 Példa indexekre	82
4.7.2 Indexek	83
4.7.3 Duplikált indexek	85
4.7.4 Keresés indextáblában	89
4.7.5 B^+ fa	93
4.7.6 Keresés több kulcs szerint	97
4.8 Hashing	99
4.9 Adattömörítés és titkosítás	100
4.9.1 Adattömörítési módszerek	100
4.9.2 Titkosítási algoritmus	101
4.10 Rendezés	102
4.10.1 Automatikus rendezés	103
4.10.2 Rendezési módszerek	104
5. A hierarchikus modell	109
5.1 Hierarchikus rendszerek	109
5.2 Faszerkezet	109
5.3 Megvalósítás	114
5.3.1 Szegmensek	115
5.3.2 A hierarchikus adatbázis leírása	115
5.3.3 Műveletek hierarchikus adatbázisban	116
5.4 Példa hierarchikus adatbázisra	117
6. A hálós modell	123
6.1 Hálós adatszerkezet	123
6.2 Kapcsolórekordok	125

6.3 A DBTG javaslat	127
6.4 Sémaleírás	128
6.5 A set koncepciója	131
6.6 Alsémaleírás	134
6.7 Adatkezelő nyelv	135
6.8 Hatékonysági kérdések	139
7. A relációs modell	143
7.1 Általános ismertetés	143
7.2 A relációk tulajdonságai	144
7.3 Táblázatok, nézetek	147
7.4 Kulcsok	149
7.4.1 Az adatbázis integritása	149
7.4.2 Elsődleges kulcs	150
7.4.3 Idegen kulcsok	151
7.4.4 Indexek	156
7.5 Műveletek relációkon	157
7.5.1 A műveletek célja	157
7.5.2 Relációs algebra és kalkulus	157
7.5.3 Relációs műveletek	158
7.6 A legfontosabb műveletek definíciója	159
7.6.1 Átnevezés (RENAME)	159
7.6.2 Korlátozás (RESTRICT)	159
7.6.3 Vetület (PROJECT)	160
7.6.4 Keresztszorzat (TIMES)	161
7.6.5 Unió (UNION)	162
7.6.6 Metszet (INTERSECT) és különbség (DIFFERENCE)	164
7.6.7 Egyesítés (JOIN)	164
7.6.8 Egyesítés alapértelmezéssel (OUTER JOIN)	167
7.6.9 Bővítés (EXTEND)	167
7.6.10 Csoportfüggvény képzés (SUMMARIZE)	168
7.6.11 Kijelölés	168
7.7 Példák relációs műveletekre	169
7.7.1 Példa korlátozásra és vetületre	169
7.7.2 Példa egyesítésre, korlátozásra és vetületre	170
7.7.3 Példa csoportosításra, csoportfüggvényre	172
7.8 Elérési út optimalizálása	172
7.9 Normalizáció	176
7.9.1 A normalizáció fogalma	176
7.9.2 Függőségek	176
7.9.3 Első normál forma (1NF)	179
7.9.4 Második normál forma (2NF)	181
7.9.5 Harmadik normál forma (3NF)	184
7.9.6 Normalizálás utáni problémák, magasabb normalizációs formák	188
7.9.7 Nézetek definiálása a normalizált relációkból	191
7.10 Katalógus	193
7.10.1 A katalógus szerepe	193
7.10.2 Katalógus-táblázatok	193
7.10.3 Katalógus objektumokról	194
7.10.4 Katalógus hozzáférési jogokról	195
7.10.5 Függőségi kapcsolatok	196
7.10.6 Általános rendszerinformációk	197

7.10.7 Katalógus osztott adatbázisokban	197
7.11 A relációs modell összefoglalása	199
8. Szöveges információ-visszakereső rendszerek	201
III. Az SQL nyelv	
9. Az SQL nyelv általános ismertetése	209
9.1 Az SQL, mint szabvány	209
9.2 Szintaktikai jelölés	210
9.2.1 Jelölési mód	210
9.2.2 A diagram értelmezése	211
9.3 Az SQL utasítások csoportosítása	213
10. Adatleíró utasítások	215
10.1 Általános jellemzők	215
10.2 Táblázatok létrehozása (CREATE TABLE utasítás)	216
10.3 Nézetek létrehozása (CREATE VIEW utasítás)	219
10.3.1 Nézetek definiálása	219
10.3.2 Példák nézetek létrehozására	221
10.3.3 Korlátozások	223
10.4 Index létrehozása (CREATE INDEX utasítás)	224
10.5 Táblázat, nézet, index megszüntetése (DROP utasítás)	225
10.6 Táblázat módosítása (ALTER TABLE utasítás)	226
11. Adatkezelő utasítások	229
11.1 Általános ismertetés	229
11.2 Szelektálás (SELECT utasítás)	231
11.2.1 A teljes- és a szubszelekt funkciója	231
11.2.2 A SELECT utasítás alapformája (szubszelekt)	232
11.2.3 A teljes szelekt	235
11.2.4 Példák szelekcióra	236
11.2.5 SQL függvények	238
11.2.6 További kiválasztási feltételek	240
11.2.7 Megjegyzések a kiválasztási feltételekhez	245
11.3 Adatváltoztató utasítások	246
11.3.1 Az adatváltoztató utasítások közös jellemzői	246
11.3.2 Törlés (DELETE utasítás)	246
11.3.3 Adatbevitel (INSERT utasítás)	248
11.3.4 Módosítás (UPDATE utasítás)	249
12. Vezérlő utasítások	251
12.1 Általános ismertetés	251
12.2 Jogosultság megadása (GRANT utasítás)	251
12.3 Jogosultság visszavonása (REVOKE utasítás)	253
12.4 Objektumok ideiglenes elzárása	253
12.4.1 Zárak kezelése	253
12.4.2 Pathhelyzet	254
12.5 Változtatások véglegesítése (COMMIT utasítás)	255
12.6 Utasítás előtti állapot visszaállítása (ROLLBACK utasítás)	256
13. További SQL utasítások	257
13.1 Általános ismertetés	257

13.2 Operátori parancsok	257
13.3 Utasítások tárolása és újrafuttatása	258
13.4 Megjelenítési formátum változtatása (FORMAT utasítás)	259
13.5 Saját működési környezet beállítása (SET utasítás)	260
13.6 Optimális elérési út	260
13.6.1 Generált elérési út leírása (EXPLAIN utasítás)	261
13.6.2 Megjegyzések a hatékonysághoz	262
13.6.3 Indexek automatikus használata	263
13.7 Egyéb utasítások	265
14. SQL utasítások beépítése programokba	267
14.1 SQL programok elkészítése	267
14.2 SQL utasítások csoportosítása	268
14.3 SQL utasítások felismerése	270
14.4 Host változók	271
14.5 Kommunikációs terület (SQLCA)	273
14.6 Különleges események kezelése (WHENEVER utasítás)	274
14.7 Statikus SQL utasítások	276
14.7.1 Általános jellemzők	276
14.7.2 Egy sor kiválasztása (SELECT INTO utasítás)	277
14.7.3 Kurzor használata	277
14.7.4 Változtatás a kurzor helyzete alapján	280
14.8 Dinamikus SQL utasítások programban	281
14.8.1 Dinamikus utasítások jellemzői	281
14.8.2 Dinamikus utasítások azonnali végrehajtása (EXECUTE IMMEDIATE utasítás)	282
14.8.3 Dinamikus utasítások előkészítése és végrehajtása (PREPARE és EXECUTE utasítás)	284
14.9 Jogosság ellenőrzés, elérési út	286

Függelék

A) Ajánlott irodalom	289
B) Tárgymutató	291

Bevezetés

A számítógépek térhódítása már eddig is jelentősen befolyásolta mindennapi tevékenységeinket. Alkalmazásuk mindennapossá vált a gazdasági, a műszaki és a tudományos élet minden területén. Az információs forradalom ugyanúgy megváltoztatta életünket, ugyanolyan új lehetőségeket nyitott meg az emberiség számára, mint az ipari forradalom néhány évszázaddal ezelőtt.

Mondhatjuk, hogy a számítógépes adatbázisok és információs rendszerek, köznap szóhasználattal az adatbankok elterjedésének jelentősége az emberi ismeretek, a tudás és az információk terjedésében és rendszerezésében a könyvnyomtatásával mérhető össze. Ahogy a nagy munkával elkészített, kézzel rajzolt, ritka, csak kevesek által elérhető és olvasható kódexeket felváltották a nagy tömegben, gyorsan előállítható, mindenki által megvásárolható nyomtatott könyvek, újságok, úgy szorítják ki, illetve már szorították ki a kézzel összeválogatott kartonlapos, fűzőlapos irattárolókat a számítógépes információs rendszerek. (Külön érdekesség, hogy a számítástechnika az utóbbi évtizedben magát a nyomdatechnikát is teljesen átalakította.)

A személyi számítógépek elterjedése, és a különböző méretű és teljesítményű számítógépeknek távközlési hálózattal való összekapcsolása lehetővé teszi a hálózatos vezetéséhez szükséges adatok otthoni számítógépes tárolásától kezdve az egész világot átfogó bankhálózatok egységes információs rendszerének kialakítását, gyors és megbízható elérését az arra jogosultak számára.

Minden valamirevaló számítógéphez forgalmaznak valamiféle „adatbáziskezelő rendszert”. Ezek ára, a személyi számítógépekhez készült néhány száz dolláros, lényegében egyszerű állománykezelési funkciókat ellátó programtól a különböző mainframe nagygépeken futtatható, komplex információs rendszerek kezelésére alkalmas, igazi adatbázis-kezelési funkciókat ellátó több százezer dolláros szoftvertermékekig terjed. Sok helyen már a hivatali presztizs is megköveteli, hogy a cégnek saját „adatbázisa” legyen, még akkor is, ha ez nem más, mint egy egyszerű állomány-nyilvántartás. Más intézményeknél pedig azt nem értik, hogy miért tart egy komplex, sok igényt egyidejűleg kielégítő adatbázis felépítése olyan hosszú ideig, és miért kellene ehhez és működtetéséhez olyan drága szoftver eszközök.

Ebben a könyvben korszerű adatbázisok és adatbáziskezelő rendszerek jellemzőit kívánjuk megmutatni. Továbbá azt, miként kell az adatokat tárolni, milyen kapcsolatokat létesíthetünk köztük, milyen szoftver módszerek állnak rendelkezésünkre, hogy az adatokat naprakészen tarthassuk, az adatbázisból a kívánt in-

formációkat gyorsan megkaphassuk, és ugyanakkor az adatok biztonságát, helyességét, ellentmondás mentességét is szavatolni tudjuk.

A könyv a témát elsősorban gyakorlati oldalról, a felhasználónak, az adatbázis létrehozójának és üzemeltetőjének szempontjából tárgyalja. Az adatbázis modellek leírásánál és az elérési módok hatékonyságának vizsgálatánál szemléletes leírást használunk, mely lehetővé teszi a módszerek világos megértését, és a gyakorlatban való alkalmazását. A részletesen tárgyalt relációs modell megvalósításánál számos olyan gyakorlati problémára hívjuk fel a figyelmet és javasolunk módszereket megoldásukra, melyek a témakörrel foglalkozó elméleti könyvekben gyakran hiányoznak, vagy csak egy megjegyzés erejéig szerepelnek, ugyanakkor a mindennapi munkánkban rendszeresen szembekerülünk velük. Éppen ezért—ellentétben az elsősorban referenciakönyvként használható kézikönyvekkel—a szöveg közben nem adunk irodalmi hivatkozásokat. A könyvben ismertetett módszereket rendszerint több szerző anyagának elemző összesítésével, valamint a szerző saját gyakorlati tapasztalatai alapján választottuk ki és foglaltuk össze. Természetesen nem gondoljuk azt, hogy egyetlen egy könyv áttanulmányozásával az olvasó ennek a komplex témának minden fontos területét a kívánt mértékben megismerheti. Ezért a könyv végén az irodalomjegyzékben felsorolunk néhány olyan művet, amelyek a téma egésze, vagy annak egyes speciális részei iránt jobban érdeklődő olvasók számára további hasznos tudnivalókat tartalmazhatnak.

A kilencvenes évek adatbázisait jellegük szerint három nagy csoportba oszthatjuk:

- személyi számítógépen megvalósított, egy személy vagy csoport információs igényeit kielégítő adatbázis, melyhez egy időben csak egy felhasználó férhet hozzá
- nagy számítógépen (mainframe), vagy megfelelő hozzáférési védelemmel rendelkező személyi számítógép hálózaton levő, egy központi helyre telepített integrált adatbázis, melynek a legkülönbözőbb személyek és csoportok különféle igényeit kell egyidejűleg, és egymástól függetlenül kielégítenie
- nagy- és/vagy személyi számítógépek hálózatára alapozott integrált, de megosztott adatbázis, melyben a leginkább helyileg igényelt adatok a felhasználás helyén vannak, ahol a hálózat bármelyik másik pontjáról is elérhetők. A mindenki által használt adatok tárolása, és annak nyilvántartása, hogy az elosztott adatok melyik adatbázisban található, viszont centralizáltan, vagy legalább is központi felügyelet alatt történik.

A könyvben tárgyalt példák és módszerek elsősorban a második típusba tartozó adatbázisokra vonatkoznak. Ennek részben az a magyarázata, hogy ezek az eljárások az elmúlt évtizedben már kialakultak, sőt bizonyos mértékig szabványosnak is tekinthetők. A másik ok az, hogy az itteni megoldások lényegében a másik két típusnál is alkalmazhatók. Az egyfelhasználós rendszereknél könnyebbé teszi dolgunkat az, hogy az adatszerkezet, az adatok közötti kapcsolat általában egyszerűbb és az egyidejű hozzáférésekből adódó problémákkal nem kell törődnünk. Az osztott adatbázisoknál viszont figyelembe kell vennünk, hogy az információk megtalálásához és összeállításához igénybe vesszük az adatátviteli hálózatot is, amely azonban megfelelő szoftver közbeiktatásával az adatbázis felhasználója

előtt rejtve marad. Ezek a különbségek azonban az adatbáziskezeléssel kapcsolatos feladataink lényegét nem befolyásolják.

A főhangsúlyt olyan adatbázisok létrehozásának és működtetésének ismertetésére helyezzük, melyekben

- igen sok (több száz) különböző típusú rekord van,
- a különböző típusú rekordok között igen sokféle kapcsolat áll fenn,
- a rekordok össz száma milliótól több száz millióig is terjedhet,
- naponta akár több ezer műveletet (lekérdezés, módosítás) is végeznek az adatbázisban,
- egyszerre több tucat, sőt több száz terminálról is elérhető és használható az adatbázis interaktív módon,
- az adatok védelmét, biztonságát, helyreállíthatóságát a rendszer biztosítja.

Természetesen azok a felhasználók is hasznosan forgathatják a könyvet, akik a fent felsoroltaknál kisebb méretű, egyszerűbb szerkezetű és igényű adatbázisokkal dolgoznak. Legfőlőbb egyes problémákkal kevésbé élesen vagy később kerülnek szembe. Mindegy, hogy milyen gépen, milyen szoftverrel dolgozunk, általános érvényű a szabály, hogy bizonyos adatmennyiségig, és/vagy tranzakciószámig nem kell törődnünk a hatékonysággal; egy adott küszöb fölött viszont már nem árt, ha gondolunk rá; míg egy meghatározott szint fölött már okvetlenül tekintetbe kell vennünk, ha működőképes rendszert akarunk létrehozni.

Az már, hogy ezek a szintek ezer vagy százezer rekordnál, 1 vagy 100 tranzakció/percnél vannak, a rendelkezésünkre álló hardvertől, az alkalmazott operációs rendszertől és nem csekély mértékben az adatbázis-kezelő szoftvertől, az adatbázis-kezelő rendszer hatékonyságától függ.

Információs rendszerek

1.1 Információs infrastruktúra

Az információs rendszereknek nem csak tárolni kell az adatokat, hanem biztosítani kell azt is, hogy azok a kellő időben, a megfelelő formában és a szükséges példányszámban eljussanak az őket igénylő személyekhez, és az újonnan keletkező adatok bejussanak és tárolódjanak a rendszerben, illetve azonnal elérjék azokat, akikhez szólnak. Ennek érdekében a hatvanas években kialakult az információs rendszerek infrastruktúrája.

Ez a következő összetevőkből állt:

1. Adatfeldolgozás, vezetői informálás

Ezt a feladatot a számítóközpont látta el. Kezdetben tipikus alkalmazási terület volt például a bérszámfejtés, számlázás, repülőgépi helyfoglalás, csekk feldolgozás. A hetvenes évektől kezdve már egyre inkább teret nyert a vezetői döntésekhez való információk szolgáltatása is.

2. Telekommunikáció

Az információ más, általában távolabbi helyre való eljuttatására szolgál. A kezdeti időben a telefon és a telex, majd később a telefax is a számítógéptől függetlenül működött. (Legfeljebb számítógép által szolgáltatott adatokat továbbított, melyet esetleg más számítógépek újra felhasználtak.) Később azonban ezek a berendezések egyre inkább összeépültek a számítógépekkel, és ma már lehetővé teszik a közvetlen ember-gép és gép-gép kapcsolatot, elvileg akármilyen nagy távolságból is. Így a közös hálózatba kötött gépeknél, és az ehhez kapcsolódó felhasználók számára (elvileg) közömbös, hogy az adat helyileg hol keletkezik, hol tárolódik, és hová kell eljutnia.

3. Központi és egyedi (részben) automatizált irodai szolgáltatások

Ide tartoznak a központi leírók, postabontók, mikrofilm laboratóriumok, írógépek majd szövegszerkesztők, másolók. Ezek közül sok ma is önállóan működik (ugyan úgy, ahogy a telefonhasználat zöme is független a szoros értelemben vett információs rendszerektől), de jelentős részük egyúttal része valamilyen szűkebb vagy tágabb értelemben vett információs rendszernek is. A szövegszerkesztő, írógépként használt személyi számítógép például egyidejűleg szol-

gálhat egy központi adatbázis lekérdezésére is, vagy az elektronikus levelező rendszeren keresztül üzeneteket továbbíthat egy másik felhasználóhoz. Napjainkban, bár a fenti területek önállósága más felhasználásoknál megmaradt, feladataik az információs rendszereknél annyira összefonódtak, hogy egyaránt szolgálnak az informatikai infrastruktúra alapjául. A korszerű információs rendszerek egységbe ötvözve magukban foglalják a hagyományos tranzakció orientált adatfeldolgozást, a döntéseket elősegítő vezetői információs rendszerek szolgáltatásait, a telekommunikációt és az iroda-automatizálást. Természetesen ez nem jelenti azt, hogy ugyanazzal a szoftverrel kell a szövegeket leírni, mint amelyikkel adatokat módosítunk a központi adatbázisban, vagy amelyek az osztott adatbázisok közti adat- és utasításkommunikációt megoldja. A fent említett területek feladatainak megoldását az egységes informatikai rendszereken belül is más szakemberek végzik különböző, a konkrét feladat megoldásához illeszkedő specifikus berendezésekkel és módszerekkel.

Ebben a könyvben csupán a tágabb értelemben vett adatfeldolgozáshoz, valamint a vezetői döntések támogatásához szükséges adatok és információk előállításával, az adatbázisokkal és az adatbáziskezelő rendszerekkel¹ foglalkozunk. A többi témakört csak annyiban érintjük, amennyire a fent vázolt téma ismertetéséhez szükségünk van.

1.2 Az információs rendszerek célja

A hatvanas évek „információs rendszerei” valójában adatállományok tranzakció-orientált feldolgozását végző óriási „papírgyárak” voltak. Kimutatások, listák tömegét gyártották. Ezek egy részét sikerrel hasznosították, például fizetési listák, jegyzékek készítésére, számlák, szállítólevelek kiállítására. Másrészüik viszont azonnal a papírkosárba került, — legfőljebb az volt a kérdés, hogy melyik vezetői szinten. Egyes rossz májű ellendrukkerek szerint a papírdaraboló iratmegsemmisítőt akár közvetlenül a számítógép nyomtatója után is elhelyezhették volna. Ez természetesen túlzás, de tény, hogy az akkori számítógépes rendszerek jelentős része sokkal több adatot ömlesztett ki, mint amennyit célszerűen fel lehetett használni. A rendszer hatékonyságát a számítógép áteresztőképességével, a naponta feldolgozott tranzakciók számával, vagy az átfutási idővel, a feldolgozás megkezdésétől a végeredmény elkészültéig eltelt idővel mérték. A rendszer készítőinek teljesítményét pedig jobb híján abból próbálták megállapítani, hány hibátlan programutasítást írtak le naponta.

A hetvenes években ez a szemlélet módosult, elterjedtek a vezetői információs rendszerek. Ezek célja az volt, hogy információt adjon a vezetésnek a döntéshez, felhívja a figyelmet a szokásostól eltérő, illetve rendkívüli esetekre, eseményekre, melyek különleges vezetői beavatkozást igényeltek. A divatos jelszó akkor ez volt: a megfelelő információt a megfelelő személynek, a megfelelő időpontban.

A fenti értékelési szempontok ma is helytállóak, de az emberi munka egyre fokozódó felértékelődésével tágabban kell értelmeznünk ezeket. Azt mondhatjuk,

¹ Az angol Data Base Management System elnevezése alapján a szokásos rövidítése DBMS.

hogy a korszerű információs rendszerek alapvető célja az, hogy az információs technológia alkalmazásának segítségével növelje a szervezetben dolgozó emberek teljesítményét. Az információs technológia alatt ezen könyvben számítógépek, számítógépes szoftverek, gép által olvasható információk és telekommunikációs módszerek széleskörű alkalmazását értjük.

A fenti definíció magában foglalja azt a tényt, hogy akinek a döntés pillanatában kellő és aktuális információ áll rendelkezésére, az valószínűleg jól, vagy legalább is jobban fog dönteni (feltéve persze, hogy megvannak a döntéshez szükséges egyéb ismeretei, valamint a döntés végrehajtásához vagy hajtathatásához szükséges eszközei és hatalma is), és azt is, hogy például a gép által készített szállítási lista részletekbe menő pontossággal tartalmazza, hogy mit mivel csomagoljanak össze és azt hová, mikor, mivel, milyen útvonalon szállítsák el.

1.3 Az információs rendszer részei

Minden információs rendszer négy főrészből tevődik össze.

Ezek:

- adatok,
- hardver (eszközök),
- szoftverek,
- felhasználók.

A köztük lévő kapcsolatot tömören úgy fogalmazhatjuk meg, hogy az információs rendszer adatokat tárol a hardveren, melyből az eszközök és a szoftver(ek) segítségével a felhasználók információkat kaphatnak.

1.3.1 Adatok

Az információ és az adat szavakat gyakran szinonimaként használják. Sokszor ez nem okoz problémát, de a pontos számítástechnikai terminológiában célszerű különbséget tenni köztük.

Az információ valamely jelenségre vonatkozó értelmes közlést jelent, melynek általában az új (legalábbis akkor számára új) ismereteket szolgáltató része fontos a felhasználó számára. A legtöbb számítógépes rendszerben azonban az információk nem kereshetők közvetlenül vissza, sőt az információk többsége nem is áll rendelkezésre a kívánt formában. Egy vezetői információs rendszerben különböző információkra van szükség. Minél magasabb a vezető szintje, annál inkább igényli az egész rendszer áttekintését, de annál kevésbé kell terhelni a részletekkel. Eme általános eltérések mellett az információs igények is állandóan változnak, így az összes elképzelhető igényre kész választ tárolni szinte lehetetlen. Éppen ezért, a végleges információ közlése előtt a számítógépes rendszerben tárolt információkat a felhasználó által kívánt formára kell hozni, össze kell vonni bizonyos részleteket, műveleteket kell velük elvégezni stb. Ilyenkor nem vesszük figyelembe az információ tényleges jelentését, hogy milyen jelenséget írnak le, csak az információt tartalmazó jelsorozatokkal, adatokkal dolgozunk. Az *adat* az információnak a számítógépes rendszerben való konkrét megjelenési formája. A feldolgozásban az

információk helyett az adatokkal végezzük el a kívánt műveleteket. Például egy egész országra kiterjedő, megyékre bontott statisztika készítésekor ugyanazokat a műveleteket fogjuk elvégezni minden azonos típusú adaton, függetlenül attól, hogy honnan származnak. A gépi feldolgozás szempontjából csak az a fontos, hogy egyértelműen azonosítani tudjuk eredetüket, hovatartozásukat. Ehhez nem szükséges tudnunk az információk jelentését, elegendő az információkat tartalmazó adatok ismerete, például az, hogy valaminek az értéke 893.

Természetesen ugyanazokat az adatokat ugyanazon információra is különbözőképpen értelmezhetjük. A 07.04.1992 dátum típusú adat az amerikai számára július 4-ét, a nemzeti ünnep napját, míg az angolnak csupán április 7-ét jelenti. Ugyanezt a magyar számítógépes dátumellenőrzés hibásnak minősítené (hisz sem 19. hónap, sem 92. nap nincs), ha nem tudnánk, milyen formátumban (pl. nap-hó-év) adták meg a dátumot. Hasonlóképpen a nagyszámítógépeken szokásos EBCDIC kódban egy byte-ként ábrázolt 11010010 jelsorozat is jelenthet -2-t, vagy K-t, attól függően, hogy az adott helyen előjeles számot, vagy alfabetikus karaktert ábrázolunk. Eppen ezért, hogy az adatból információt kaphassunk, nem csak azt kell tudnunk, hogy mit jelent, hanem azt is, hogy azt milyen formában tartalmazza.

Ezekkel a kérdésekkel, és hogy ezt miként oldják meg az adatbáziskezelő rendszerek, a 10. fejezetben, az adatbázis leírásánál fogunk részletesen foglalkozni.

1.3.2 Hardver

Minden rendszer valamilyen konkrét hardveren valósul meg. Ennek része a számítógép (vagy osztott adatbázisoknál a számítógépek), melyen az adatokat tároljuk, valamint a terminálok és a telekommunikációs rendszer, mivel általában ezeken keresztül lépünk kapcsolatba a géppel. A hardverrel ezen könyvben csak érintőlegesen foglalkozunk. Csupán az adatok fizikai tárolására szolgáló mágneslemezek felépítését és működését, valamint az adatoknak operatív tárból való ideiglenes tárolására szolgáló puffertechnikát ismertetjük röviden a 4.2 és 4.3 pontokban, mivel ezek hatékonyságát jelentős mértékben tudjuk befolyásolni az adat-szervezési, adatelérési és pufferalási módok helyes használatával.

1.3.3 Szoftver

A hardveren fizikailag tárolt adatok és a felhasználó közti kapcsolatot a szoftver hozza létre.

Ez három részből tevődik össze:

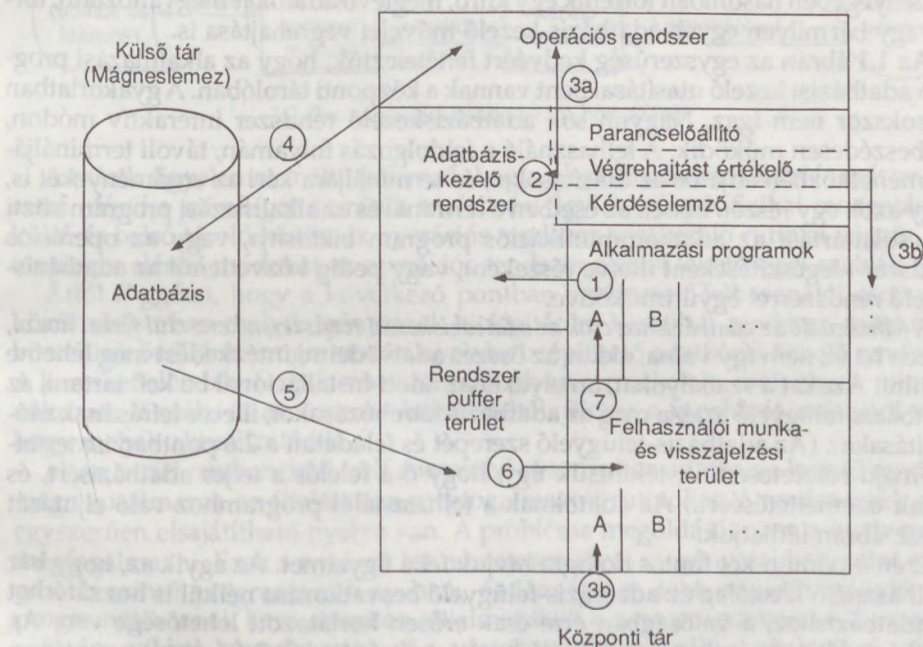
- operációs rendszer,
- adatbáziskezelő rendszer,
- alkalmazási programok.

Az operációs rendszer vezérli és ellenőrzi a számítógépeken futó programok végrehajtását, és kezeli a perifériákat (mágneslemezek, terminálok, adatátviteli vonalak stb.). Ebből kifolyólag az adatbáziskezelő rendszer is az operációs rendszer felügyelete alatt fut és az operációs rendszeren keresztül lép kapcsolatba a perifériákkal, hogy onnan adatokat hozzon be vagy oda adatokat írjon ki. Ennek a szétválasztásnak az az

előnye is megvan, hogy az adatbáziskezelő rendszernek nem kell az egyes perifériák speciális tulajdonságaival egyedileg törődnie, elegendő az input/output parancsokat az operációs rendszerek parancsainak szintjéig részletezni. Ez jelentős mértékben lecsökkenti az adatbáziskezelő rendszer elkészítéséhez szükséges programozási munkát. Ugyanakkor lehetővé teszi ugyanannak a rendszernek sokkal szélesebb körű alkalmazását, mivel az operációs rendszer szintjén teljesen azonosak lehetnek a parancsok különböző berendezésekre, központi egységekre vagy operációs rendszerekre.

Az alkalmazási program határozza meg, hogy milyen adatokból milyen információkat kell összeállítani. Ezzel a II. részben, az adatmodelleknél foglalkozunk röviden, inkább elvi szinten, majd a 11. fejezetben az adatkezelő nyelvnél tárgyaljuk részletesen.

Az 1.1 ábrán bemutatjuk az adatbáziskezelő szoftver működésének elvét egy adat beolvasásakor.



1.1 ábra Adatbázis-kezelő szoftver működésének elve

1. Tegyük fel, hogy az alkalmazási program egy adatot kér az adatbázisból. (Egyszerre több alkalmazási program is futhat a gépen, és ezek bármelyike fordulhat az adatbáziskezelő rendszerhez. A kérések általában sorbaállítódnak, feldolgozásuk egymás után történik.)
2. Az adatbáziskezelő rendszer értelmezi és elemzi az utasítást. Megállapítja, hogy végrehajtható-e (szintaktikailag helyes-e, léteznek-e és hozzáférhetőek-e pillanatnyilag a kért objektumok, és hogy a felhasználóknak van-e jogosultsága a kért művelet(ek) elvégzésére).

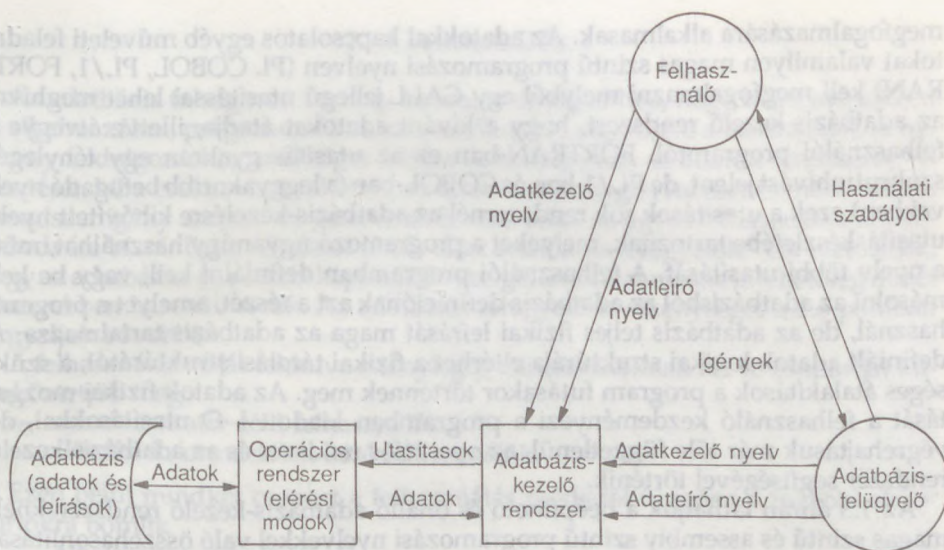
- 3.a Ha az elemzés eredménye pozitív, akkor az adatbázisra vonatkozó utasításokat átalakítja az operációs rendszer számára értelmezhető parancsokká, és továbbítja ezeket az operációs rendszer felé. Egy adatbázisművelet kérés rendszerint egy egész parancssorozatot generál.
- 3.b Ha a kérés bármilyen okból nem hajtható végre, visszajelzést ad a felhasználói munka- és visszajelzési területre, megjelölve általában a visszautasítás okát is.
4. Az operációs rendszer létrehozza a kapcsolatot a külső tárolóval. (Ha ez nem sikerül, visszajelzést ad.)
5. A kért adatokat behozza a rendszer puffertérületére.
6. Átadja a felhasználói programnak az adatokat. Jelzi a művelet eredményességét, vagy (részleges) eredménytelenségének okát, esetleg valamilyen különleges eseménynek az észlelését.
7. A felhasználói program átveszi az adatokat (ha kapott), és értékeli a visszajelzéseket.

Lényegében hasonlóan történik egy kiíró, meglévő adatokat megváltoztató, törölő vagy bármilyen egyéb adatbázis-kezelő művelet végrehajtása is.

Az 1.1 ábrán az egyszerűség kedvéért feltételeztük, hogy az alkalmazási program adatbázist kezelő utasításai bent vannak a központi tárolóban. A gyakorlatban ez sokszor nem igaz. Nagyon sok adatbáziskezelő rendszer interaktív módon, párbeszédesen működik. A felhasználó a feldolgozás folyamán, távoli termináljáról menetközben adja be az utasításokat, és termináljára kéri az eredményeket is, vagy azok egy részét. Ebben az esetben a terminál és az alkalmazási program közti kapcsolattartást az adatkommunikációs program biztosítja, vagy az operációs rendszer kiegészítéseként illetve részeként, vagy pedig közvetlenül az adatbázis-kezelő rendszerrel együttműködve.

A felhasználó az adatbázishoz csak az adatbáziskezelő rendszeren keresztül férhet hozzá, hiszen ha ez nem így volna, akkor az összes adatvédelmi intézkedést meg lehetne kerülni. Azokat a szabályokat, amelyeket az adott installációnál be kell tartani, az adatbázis-felügyelő szabja meg az adatbázis létrehozásakor, illetve leírásának módosításakor. (Az adatbázis-felügyelő szerepét és feladatait a 2.6 pontban ismertetjük majd részletesen. Itt tekintjük úgy, hogy ő a felelős a teljes adatbázisért, és annak üzemeltetéséért.) Az adatoknak a felhasználói programhoz való eljutását az 1.2. ábrán láthatjuk.

Ezen a ponton két fontos dologra hívjuk fel a figyelmet. Az egyik az, hogy bár a felhasználó látszólag az adatbázis-felügyelő beavatkozása nélkül is hozzáférhet az adatbázishoz, a valóságban erre csak erősen korlátozott lehetősége van. Az adatbázis létrehozásakor az adatleíró nyelv segítségével (l. 2.4.1 pont) ugyanis az adatbázis-felügyelő egyértelműen beépíti az adatbázis-kezelő rendszerbe azokat az ellenőrzési módokat, melyek a jogtalan beavatkozást megakadályozzák. Természetesen, ha egy felhasználónak joga van pl. adatokat módosítani, akkor azt az adatbázis-felügyelő általában már nem tudja ellenőrizni, hogy az új adatok érdemi szempontból helyes értékeket tartalmaznak-e. (Bizonyos felülvizsgálatot azonban itt is végezhet, pl. ha az adatoknak meghatározott értéktartományok között kell mozognia.) Azt azonban már megakadályozza, hogy a felhasználó olyan adatokhoz férhessen hozzá, vagy olyanokat módosíthasson, melyek számára nincsenek engedélyezve.



1.2 ábra Az adatbázishoz való hozzáférés

A másik figyelemre méltó jellemző, hogy a legtöbb adatbázis-kezelő rendszer szorosán ráépül a számítógép operációs rendszerére. Így az adatok fizikai mozgatását a külső és belső tároló között az operációs rendszer adatkezelő rutinjai végzik, és a tényleges elérési módokat az operációs rendszer elérési lehetőségei szabják meg.

Attól függően, hogy a következő pontban ismertetett felhasználói csoportok közül elsősorban melyik igényeinek kielégítésére készült a rendszer, megkülönböztetünk önálló és programozási nyelvbe beépíthető adatbázis-kezelő rendszert. A kereskedelembe forgalomban vannak olyan termékek is, melyeknek mindkét változata létezik, és az üzembe helyezésétől, illetve a meghívás módjától függ, hogy melyik aktualizálódik.

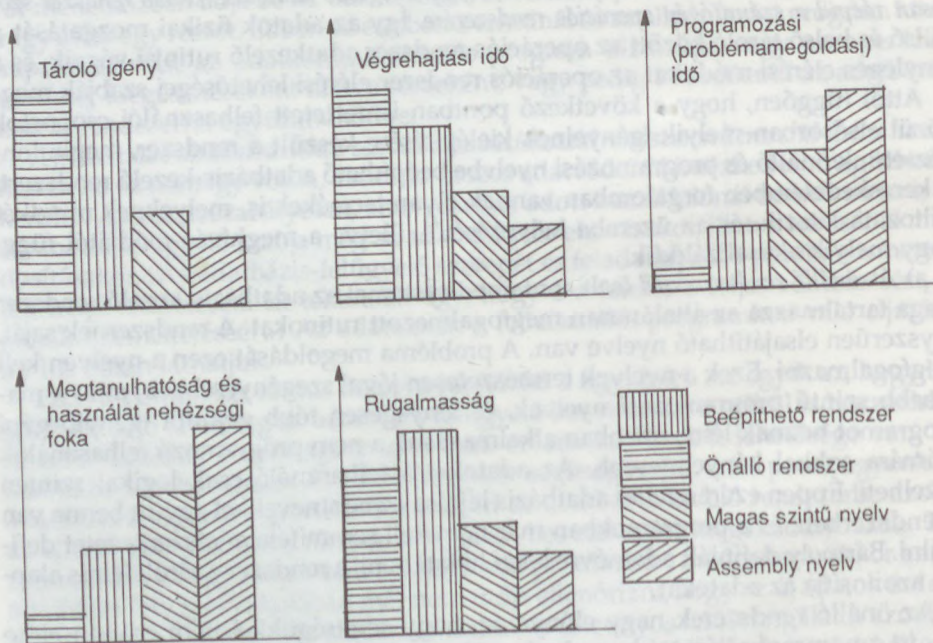
a) Az önálló rendszereknél (self contained systems) az adatbázis-kezelő rendszer maga tartalmazza az általánosan megfogalmazott rutinokat. A rendszernek saját, egyszerűen elsajátítható nyelve van. A probléma megoldását ezen a nyelven kell megfogalmazni. Ezek a nyelvek természetesen jóval szegényesebbek, mint a magasabb szintű programozási nyelvek, és lényegesen több gépidőt igénylő gépi programot hoznak létre, azonban alkalmazásuk a nem programozó felhasználók számára sokkal kényelmesebb. Az adatokat a felhasználó csak logikai szinten kezelheti. Éppen ezért a teljes adatbázis leírása az adatnevekkel együtt benne van a rendszerben, és a programokban már nem kell semmiféle adatszerkezetet definiálni. Bármely definiált adatnévre lehet hivatkozni, a rendszer a tárolt leírás alapján azonosítja az adatokat.

Az önálló rendszerek nagy előnye az, hogy segítségükkel nem csupán előre megírt programok eljárásai és eredményei használhatók fel, hanem ad hoc igények is kielégíthetők az ember-gép párbeszéd révén. A személyi számítógépekre készített adatbázis-kezelő rendszerek csaknem kizárólag ebbe a kategóriába tartoznak.

b) A beépíthető rendszerek (host language systems) önmagukban nem használhatók. Csupán az adatbázis leírásaira és a beviteli-kiviteli (input-output) műveletek

megfogalmazására alkalmasak. Az adatokkal kapcsolatos egyéb műveleti feladatokat valamilyen magas szintű programozási nyelven (pl. COBOL, PL/1, FORTRAN) kell megfogalmazni melyből egy CALL jellegű utasítással lehet meghívni az adatbázis-kezelő rendszert, hogy a kívánt adatokat átadja, illetve átvegye a felhasználói programtól. FORTRAN-ban ez az utasítás gyakran egy tényleges szubrutinhívást jelent, de PL/1-ben és COBOL-ban (a leggyakoribb befogadó nyelvekben) ezek a utasítások sok rendszernél az adatbázis-kezelésre kibővített nyelv utasításkészletébe tartoznak, melyeket a programozó ugyanúgy használhat, mint a nyelv többi utasítását. A felhasználói programban definiálni kell, vagy be kell másolni az adatbázisból az adatbázis definíciónak azt a részét, amelyet a program használ, de az adatbázis teljes fizikai leírását maga az adatbázis tartalmazza. A definiált adatok logikai struktúrája eltérhet a fizikai tárolási struktúrától, a szükséges átalakítások a program futásakor történnek meg. Az adatok fizikai mozgását a felhasználó kezdeményezi a programban kiadott I-O utasításokkal, de végrehajtásuk már tőle függetlenül, az operációs rendszer és az adatbázis-kezelő rendszer segítségével történik.

Az 1.3 ábrán láthatjuk a beépíthető és önálló adatbázis-kezelő rendszereknek magas szintű és assembly szintű programozási nyelvekkel való összehasonlítását különféle szempontok szerint. (Az ábra csak kvalitatív tájékoztatásra szolgál, pl. gyors, közepes, lassú.)



1.3 ábra Beépíthető és önálló adatbázis-kezelő rendszerek összehasonlítása magasszintű és assembly szintű programozási nyelvekkel különböző szempontok alapján (kvalitatív ábrázolás)

1.3.4 Felhasználók

Az adatbázis tulságosan komoly és fontos dolog ahhoz, hogy teljes mértékben az alkalmazókra, programozókra, szervezőkre lehessen bízni a használatát. A hatékony feldolgozáshoz szükséges feltételek kialakítását, az adatok rendszeres ki-mentését, hiba esetén visszatöltését, az adatmodell megtervezését és módosítását, a rendszer igény szerinti újraszervezését nem lehet az egyes felhasználók kényére-kedvére bízni. Ezt a feladatot látják el az adatbázis-felügyelők.² Ők biztosítják, hogy az adatbázist és a hozzákapcsolódó szolgáltatásokat mások is lehetőleg minél könnyebben használhassák. Az adatbázis felügyelő tevékenységet a 2.6 pontban tárgyaljuk bővebben.

A felhasználás célját tekintve a tényleges felhasználóknak két fő csoportját különböztetjük meg:

- adminisztratív célú felhasználók,
- döntési információkat kérő felhasználók.

Ezen belül mindkét csoport a felhasználás módjától függően a további alcsoportokra bomlik

- alkalmazási programozók,
- nem programozó alkalmazók,
- menü alapján dolgozó felhasználók.

Az egyes csoportok és alcsoportok között az elkülönülés nem éles. Ugyanaz a személy, feladattól függően egyszer az egyik, máskor a másik módon használja az adatbázist.

Az első csoportban levőkre, az *adminisztratív célú felhasználókra*, általában jellemző, hogy *nagyszámú tranzakciót bonyolítanak le*, és az egyes tranzakciók jelentősége viszonylag csekély (kivéve persze, ha pl. igen nagy összeg átutalásáról van szó). A tranzakció *jól definiált eljárás* alapján történik. A hatékonyság többé-kevésbé egyértelműen mérhető az időegység alatt lebonyolított tranzakciók számának és az ehhez használt számítógépes erőforrásoknak a viszonyával.

A második csoportot, a *döntési információkat igénylő felhasználókat* az jellemzi, hogy lényegesen kevesebb tranzakciót bonyolítanak le. Ezek különféle módokon hajthatók végre, és különböző módon értékelhetők. Az adatbázisból nyert információk eredményeként döntések születhetnek, és a rendszer hatékonyságát ebben az esetben végső soron az dönti el, mennyire segítik elő a szervezet fejlődését, működését ezek a döntések. Ez persze sokkal kevésbé pontosan definiálható, mint az első csoportnál.

Tipikus példa a kétféle feldolgozásra, amikor az adminisztratív típusú feldolgozásnál évekre visszamenőleg kigyűjtik egy hitelt kérelmezőnek a folyószámláján történő pénzmozgásokat, korábbi hitelkérelmeit, azok visszafizetését, és egyéb pénzügyi információkat. Ezekből, valamint a bank pénzügyi helyzetét tartalmazó adatokból a döntési helyzetben levő felhasználó olyan információkat válogat ki,

² Az angol „data base administrator” elnevezés után szokás ezt a feladatkört ellátó személyt (vagy csoportot) adatbázis adminisztrátornak is nevezni. Mivel azonban a magyarban az adminisztrátort általában nem tekintik érdemi ügyintézőnek, célszerűbbnek tartjuk az adatbázis-felügyelő kifejezést.

olyan elemzést készít, melynek alapján megállapíthatja, hogy a kérelmező megkaphatja-e a kért hitelt és ha igen, milyen feltételek mellett.

Az alkalmazás módját tekintve, mint azt már említettük, megkülönböztethetünk alkalmazási programozókat, nem programozó, de önálló feladatokat megfogalmazó alkalmazókat és menüorientált felhasználókat.

Az *alkalmazási programozók* a saját vagy a végső felhasználó által megadott igények alapján készítik el a konkrét alkalmazások programjait. Az adatbázis-kezelő rendszert elsősorban csak a megfelelő adatok kikeresésére, illetve a külső tárolóra való kiírásra használják fel. Rendszerint nagy tömegű adatra támaszkodva viszonylag sok, de azonos jellegű adattal dolgoznak, melyek az adatbázis általában jól körülhatárolható részének a feladat által kívánt rekordokba való csoportosításával definiálhatók. A programozáskor legtöbbször nem a programkészítés gyorsasága, hanem a futtatás hatékonysága a döntő. Az adatbázist ily módon csak gyakorlott programozó tudja használni. A nem programozó felhasználó csak a már megírt programokon keresztül léphet kapcsolatba a rendszerrel, tevékenysége a kész programok által meghatározott lehetőségekre korlátozódik, így lényegében szerepe ugyanaz, mint a következő bekezdésben ismertetett paraméteres felhasználóké.

Az alkalmazók döntő többsége azonban nem programozó. Semmiféle, vagy csak minimális programozási ismeretekkel rendelkeznek. Attól függően, hogy igényeik ad hoc jellegűek-e, és formájukat, tartalmukat a pillanatnyi helyzetnek megfelelően kell-e mindig megfogalmazni, és ezáltal előre nem programozhatók, vagy pedig az információigények néhány jól körülhatárolt, előre programozható témakörre és kérdéscsoportra korlátozódnak, megkülönböztethetünk *nem programozó alkalmazókat és menük alapján dolgozó felhasználókat*. Az előbbieknél feladataik megoldásához egy általánosított adatkezelésre alkalmas, nem számítástechnikai szakember által is könnyen megtanulható, elsősorban egyszerű logikai kapcsolatokat és matematikai alpműveleteket leíró programozási nyelvre van szükségük. Ennek a nyelvnek a segítségével a felhasználók maguk definiálhatják az általuk igényelt adatkapcsolatokat és az adatokon végrehajtandó műveleteket. A menüorientált felhasználóknak még erre a viszonylag egyszerű szintaktikájú nyelvre sincs szükségük kérdéseik megfogalmazásához. Néhány szabványos, előre beprogramozott kérdéstípus aktuális paramétereit megadva kaphatják meg a szükséges információkat (pl. banki lekérdező rendszerek, helyfoglaló rendszerek).

A számítástechnika elterjedése, a felhasználóbarát rendszerek kifejlesztése lehetővé tette, hogy egyre növekedjen azoknak a száma, akik a megadott alkalmazási területnek szakemberei. Feladataikat, információs igényeiket logikailag meg tudják fogalmazni, és rendelkeznek olyan számítógépes ismeretekkel, hogy az adatbáziskezelő rendszer által nyújtott lehetőségek segítségével ezeket a gép által is közvetlenül értelmezhető formába öntsék. Ez nagy mértékben megnöveli az adatbázis kezelő rendszerek gyakorlati alkalmazóinak számát és a felhasználások lehetőségét.

1.3.5 Informatikai központ

Az elinduláshoz és működéshez szükséges magas költségek, valamint az eredmények, az információk fontossága miatt az adatok gyűjtésével, tárolásával, rendszerezésével, információk szolgáltatásával a szervezeteknek egy önálló részlege³ foglalkozik. A klasszikus adatfeldolgozás korában ezt számítóközpontnak nevezték. Ma, amikor sokkal nagyobb súlyt fektetnek a szolgáltatott információk értékére (mivel az sokkal értékesebb a szervezet számára, mint a hardver és a szoftver, amellyel a rendszer működik) az elnevezés is tükrözi ezt a változást. A sokféle lehetőség közül, mi az egyszerűség kedvéért *Informatikai Központ*nak nevezzük a szervezetek információs és számítástechnikai rendszerével kapcsolatos teendőket ellátó részlegét. Ennek vezetője célszerűen a szervezeti hierarchia legfelső szintjén levő vezetők valamelyikének van közvetlenül alárendelve.

Az Informatikai Központ az alábbi főbb részlegekből áll, melyeknek létszáma igen eltérő, de mindegyikre szükség van a normális üzemmenet biztosításához.

A legnépesebb, és a napi feladatok ellátásához a legfontosabb a *Számítóközpont*, amit ma inkább *Információ Szolgáltató Részleg*nek neveznénk. Itt végzik el a szokásos adatfeldolgozási feladatokat, beleértve az egész vállalatra kiterjedő hagyományos rendszerek kifejlesztését is.

A Számítóközpont biztosítja a technikai szolgáltatásokat is. Ezek közé tartozik többek között az adatbázis felügyelet, a rendszer szoftver támogatás, a számítógépek közötti kapcsolat megteremtése és általában a telekommunikációs szolgáltatások.

Természetesen nagy szervezetben emellett lehetnek az egyes részlegeknek is önálló számítóközpontjai. Ezek önállóan szolgáltathatnak információkat, fejleszthetnek ki rendszereket a Központi Számítóközpont által meghatározott egységes elvek, szempontok és eljárások figyelembevételével és betartásával.

Napjaink számítóközpontjainak szervezeti felépítése nemigen különbözik a 10–15 évvel ezelőtől. Nagy vonalakban tekintve, a szervezet egy számítógépet üzemeltető, egy aktuális alkalmazási rendszereket fejlesztő (és karbantartó), egy rendszer szoftverrel kapcsolatos technikai szolgáltatást nyújtó és egy telekommunikációt biztosító részlegből áll. Míg korábban lényegében ez jelentette a teljes információ-szolgáltató részleget, most ez az Informatikai Központnak csak egyik része.

A napi munkát a viszonylag nagyobb létszámú számítóközpont végzi el. A jövő szempontjából azonban nélkülözhetetlen az általában csak néhány főt számláló *információs erőforrások tervezését és felhasználását irányító részleg*. Ennek munkatársai tervezik az intézményi szintű információs rendszereket és segítséget nyújtanak a szervezet egyes részlegeinek saját információs terveiknek az összintézményi tervekhez való illesztéséhez, egymás közti egyeztetéséhez. Főbb tervezési és koordinációs területeik: hálózatok, iroda automatizálás, végfelhasználói számítógépes alkalmazások, adatbázis technológia, rendszerfejlesztési eljárások szab-

³ Mivel a szervezet mérete (és bizonyos mértékig a vezetők címkorsága) szabja meg, hogy egy adott funkciót igazgatóság, főosztály, osztály vagy csoport lát el, az egyszerűség kedvéért általában a részleg elnevezését fogjuk használni.

ványosítása és automatizálása, valamint természetesen a szervezet profiljába eső feladatokra alkalmazási rendszerek távlati tervezése.

A harmadik részleg nyújt segítséget a szervezeten belüli számítógépes felhasználók számára.

Külön részleg foglalkozik ezen belül az általános szoftverekkel kapcsolatos felhasználói tanácsadással. Ez lehet a központi gép szoftvere, de lehetnek olyan PC-s rendszerek, melyeket helyileg használnak.

Az alkalmazási feladatok fejlesztésében nyújt segítséget ezen belül egy másik részleg. Ez történhet a központ keretén belül (pl. egy helyi célra készült alkalmazást átalakítanak mindenki számára felhasználható formába), de igen hatékony lehet, ha egy feladaton a helyi fejlesztőkkel dolgoznak együtt a probléma megoldása érdekében.

Ugyanitt egy harmadik, önálló részlegnek a feladata a végfelhasználók képzése a már meglevő és az újonnan bevezetett információs technológiai módszerek alkalmazására.

Adatbázisok

A hétköznapi szóhasználatban adatbázison egy adatfeldolgozó környezetben használt adatok összességét értjük. Ez a megfogalmazás azonban túl általános. Eszerint a hagyományos adatfeldolgozó rendszerek állományainak összességét is adatbázisnak, az ezeket kezelő programokat pedig adatbázis-kezelő rendszernek tekinthetnénk. A valódi adatbázis azonban lényegesen több ennél. Az adatokon kívül tartalmazza a különböző típusú adatok közti összefüggéseket (pl. a hallgatók, osztályok, tantárgyak és oktatók egyedi adatain kívül azokat a kapcsolatokat is, melyek segítségével megállapíthatjuk, melyik hallgató melyik osztályba jár, kik és mire tanítják). Ezeken kívül az adatbázisokon működni kell olyan adatbázis-kezelő rendszernek, mely lehetővé teszi az adatokból a bennük tárolt információk előállítását.

Tágabb értelemben véve kétféle adatbázis típust különböztethetünk meg. A *tény-adatbázisok* meghatározott formában tárolt adatokat tartalmaznak, melyeknek egyedei között különféle kapcsolatok állnak fenn. Az adatbázis-kezelő rendszer a formázott adatok értéke és a kapcsolatok alapján állítja elő a kért információt. A *dokumentum típusú* vagy más néven *szöveges információ-visszakereső rendszerek* dokumentumokról tartalmaznak nem formázott, szöveges leírás formájában megadott információt. A rendszer azokat a rekordokat szolgáltatja, melyekben egy adott szövegrész meghatározott körülmények között és formában előfordul a szövegben.

A gyakorlatban adatbázisnak általában a tény-adatbázisokat szokták tekinteni, és a dokumentum típust külön jelzik. Könyvünkben is elsősorban ezekkel foglalkozunk és a szöveges információ-visszakereső rendszerek néhány jellegzetességét a 8. fejezetben külön ismertetjük.

Megjegyezzük, hogy az elkülönülés a kétféle típus között nem teljesen éles. Minden szöveges információ-visszakereső rendszerben tárolhatunk formázott adatokat is, és csaknem minden tény-adatbázis kezelő rendszerben lehet valamilyen mértékben nem formázott szövegrészeket is keresni.

A (tény)-adatbázis kezelő rendszereknek biztosítani kell:

- különféle igények hatékony kielégítését,
- adatfüggetlenséget,
- adatok közti komplex kapcsolatok ábrázolását,
- redundancia mentességet illetve annak ellenőrzését,

- egyszerű használatot,
- az adatok védelmét, nehogy illetéktelenek hozzáférhessenek,
- az adatok integritását, hogy a hozzáférésre jogosultak se ronthassák el lehetőleg az adatbázist,
- helyreállíthatóságot, hogy bármilyen esetleges hiba esetén az eredeti állapotot vissza lehessen állítani,
- több felhasználó adatbázisoknál az egyidejű hozzáférést, illetve annak ellenőrzését,
- osztott adatbázisoknál ezen felül lehetővé kell tenni az adatok szétosztását és megtalálását a különböző helyekről, az egyidejű hozzáférés ellenőrzését és az adatforgalom optimalizációját is.

A továbbiakban ezeket a feladatokat részletesebben is elemezzük.

2.1 Különféle igények hatékony kielégítése

Bármiféle információszolgáltatásnak csak akkor van értelme, ha a kapott információ

- pontos,
- kielégítő részletességű,
- érthető,
- időben érkezik meg,
- könnyen hozzáférhető,
- nem túl drága,
- felhasználásra is kerül.

Ezért biztosítani kell, hogy az adatbázisban tárolt adatokból a különféle felhasználók által kívánt információk az igényeknek megfelelő időn belül rendelkezésre álljanak. Az adatokat oly módon lehessen redundanciamentesen (vagy tudatosan meghagyott minimális redundanciával) rendszerezni, hogy kiválasztásuk, csoportosításuk többféle szempont szerint is hatékonyan történhessen meg. A felhasználónak csak az adatok logikai szerkezetét kelljen megismernie, fizikai tárolási módját nem szükséges tudnia.

2.2 Adatfüggetlenség

Mivel ugyanazon adatokat több felhasználó és több program is használhatja, az adatokat és programokat, amennyire lehetséges, függetleníteni kell egymástól. Ha például valamelyik programban a feldolgozási szempontok módosulása miatt a rekordokat egy új mezővel kibővítjük, akkor ez a változtatás lehetőleg csak ennek az egy programnak a módosítását tegye szükségessé. Az összes többi program, amelyik ugyanezen rekordokat vagy ezek egy részét továbbra is a módosítatlan formában használja, változatlan maradjon. Ennek megvalósítása érdekében külön kell választani az adatok fizikai leírását a program által látott adatszerkezettől. Az adatbázisokkal foglalkozó szakemberek ezt *adatfüggetlenségnek* nevezik. Ennek valójában két fokozata van, a fizikai és a logikai adatfüggetlenség.

Egy rendszer *fizikailag adatfüggetlen*, ha az adataival dolgozó felhasználó programok vagy a felhasználók ad hoc információkérései gyakorlatilag függetlenek az adatok tárolási és elérési módjától.

A *logikai adatfüggetlenség* viszont azt jelenti, hogy az adatbázis logikai szerkezetében létrehozott változások az adatbázist felhasználó programokat nem befolyásolják jelentős mértékben.

A tényleges adatfüggetlenség a fenti értelmezésben a fizikai és logikai adatfüggetlenség együttes megvalósulása. *Egy rendszer tehát akkor adatfüggetlen, ha az adatbázisban tárolt adatokat az adatbázis-kezelő rendszer közvetítésével felhasználó programok és ad hoc jellegű lekérdezések gyakorlatilag függetlenek mind az adatok tárolási és elérési módjától, mind pedig az adatbázis logikai szerkezetétől.*

Természetesen teljes adatfüggetlenség nem valósítható meg (ezért is szerepel a meghatározásokban az enyhítő „gyakorlati” szó), hiszen például az elérési mód meghatározásával, amint azt a 4. fejezetben részletesen megmutatjuk, nagymértékben befolyásolhatjuk a rendszer hatékonyságát. (Pl. pointer lánc helyett részletes indexátlázzal való visszakeresés.) A felhasználó szintjén azonban ezek a változások nem úgy jelentkeznek, hogy új programokat kell írnia, más módon kell a kérdéseit feltennie, hanem csak abban, hogy az eredményeket, a választ hamarabb vagy olcsóbban, vagy esetleg más felhasználók igényeinek hatékonyabb kielégítése céljából történt átszervezés miatt éppenséggel lassabban vagy drágábban fogja megkapni.

2.3 Az adatbázis architektúra három szintje

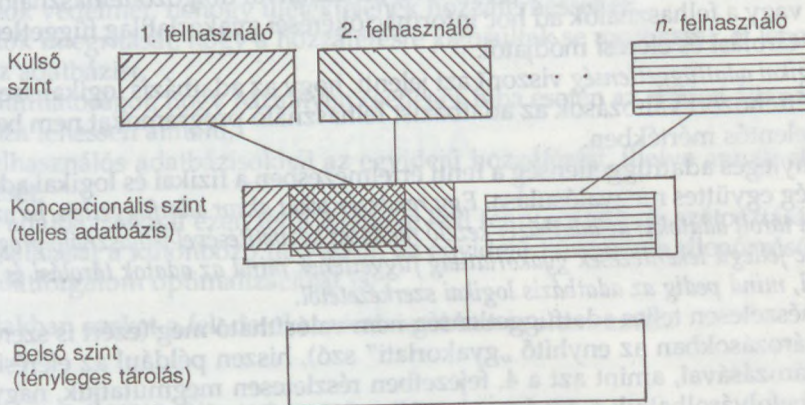
Az információs rendszer a valóság objektumainak, ezek jellemzőinek, kapcsolatainak összefüggéseit ábrázolja modell formájában. *Az adatmodell nem a konkrét adatokkal, azok értékeivel foglalkozik, hanem azok típusaival, kapcsolataival.* A feladatmegoldásnak első lépése mindig a megfelelő adatmodell kialakítása, mely nem csupán az ábrázolni kívánt világot tükrözi híven, hanem alkalmas arra, hogy az általa reprezentált adatszerkezeten a valóság mozgásai a rendelkezésre álló számítástechnikai eszközökkel nyomon követhetők legyenek.

Az adatfüggetlenség biztosítása érdekében az adatmodell szerkezetileg három szintre bomlik (2.1 ábra)

- *belső* vagy más néven *fizikai szint*, mely az adatok fizikai elhelyezését és fizikai elérési módját írja le
- *külső szintek*, melyek azt írják le, miként látják az egyes felhasználók az adatbázist
- *koncepcionális szint*, mely azt írja le, hogy logikailag egységbe vonva hogyan néz ki ténylegesen az adatbázis. Ennek látják különböző vetületeit a külső szinten a felhasználók, és ez képződik le egyértelműen tárolási és elérési struktúráként a belső szinten.

Az adatfüggetlenség teljes, ha a három szint egymástól teljesen független.

A három szinten megfogalmazható objektumokat, tulajdonságaikat és kapcsolataikat egy egyszerű példán mutatjuk be.



2.1 ábra Az adatbázis architektúra három szintje

A koncepcionális szint az alábbi adatokat (objektumokat és tulajdonságait) tartalmazza.

Hallgató (neve 20 karakter; melyik osztályba jár (osztálykód), 4 karakter; születési helye 20 karakter, és időpontja 8 karakter)

Osztály (osztály kódja 4 karakter; osztályfőnök neve 20 karakter; órarend 7×12×8 karakter)

Ezen kívül a koncepcionális szintbe még azt a megszorítást is be vesszük, hogy a hallgató rekordban csak olyan osztálykód szerepelhet, amelyiknek az osztályrekordok között is van megfelelője, és az osztályrekordok között két azonos osztálykód nem fordulhat elő.

Ez azt jelenti, hogy az adatbázis-kezelő rendszer automatikusan megakadályozza, ha úgy akarunk módosítani, vagy olyan rekordot akarunk bevinni, amelyekre ezen feltételek nem teljesülnek.

A külső szint az 1. felhasználónál az alábbi:

Hallgató (név 15 karakter; osztálykód 4 karakter)
(Ebben az esetben előfordulhat, hogy a hallgató neve csonkítva jelenik meg.)

A 2. felhasználó külső szintje:

Hallgató (név 20 karakter; osztályfőnök neve 20 karakter)

Itt az osztályfőnök nevét az adatbázis-kezelő rendszer automatikusan szolgáltatja, mivel a koncepcionális szint ezt a kapcsolatot az osztálykódon keresztül biztosítja.⁴

A belső szint leírja, hogy a hallgatók adatait tartalmazó rekordok melyik lemez mely cilinderjein és sávjain vannak, pl. osztályonként és ezen belül név szerint

⁴Ez a kapcsolatléírás például a 7. fejezetben ismertetendő relációs modell ábrázolási módjának felel meg.

sorrendben tárolva, hosszuk 52 karakter, melyből az első 20 a hallgató neve, a következő 4 az osztálya, következő 20 a születési helye, majd időpontja év–hó–nap formában, és hogy bármelyik hallgató rekordját név szerint közvetlenül megkapjuk indexek (l. 4.7 pont) alapján.

Hasonló leírást készítünk az osztályrekordra is.

A külső és a koncepcionális szinten ugyanazt az adatmodellt használjuk, a belső ábrázolás viszont adatmodelltől független.

2.3.1 Külső szint

A külső szint az, ahogy az *egy felhasználók látják az adatbázist*. A koncepcionális szinten tárolt objektumokból, azok tulajdonságaiból és a köztük levő kapcsolatokból az adatleíró nyelv segítségével definiálják az ezen a szinten előforduló rekordtípusokat és azok tulajdonságait. A külső szint rekordjai megegyezhetnek a koncepcionális és a belső szint rekordjaival is, de tartalmazhatják annak egy részét (a példában az 1. felhasználó), de az is lehet, hogy több rekord részeiből tevődnek össze meghatározott szempontok alapján történő összeválogatás alapján (2. felhasználó).

2.3.2 Koncepcionális szint

A koncepcionális szint írja le, *miként néznének ki mindenki számára az adatok*, ha mindenki mindent láthatna belőlük. Ezt a leírást szokták az *adatbázis sémájának* is nevezni (míg az ennek valamely részét képező külső szinteket alsémának). Ez tartalmazza a koncepcionális szint összes rekordtípusának tartalmi leírását (milyen tulajdonságai vannak, és ezeket milyen formában tároljuk), de az adatfüggelenség érdekében nem tartalmaz utalást arra, hogy hol vannak ezek az adatok és konkrétan hogyan férhetünk hozzájuk.

A koncepcionális szinten adjuk meg azt is, hogy a különböző rekordtípusokhoz, illetve egy adott rekordtípuson belül különféle rekordokhoz, vagy azok mezőihöz kik és milyen célból férhetnek hozzá, azaz itt határozzuk meg az adatbázis védelmét.

Ugyancsak a koncepcionális szinten írjuk le, milyen kapcsolatok vannak az egyes rekordtípusok között, aminek egy lehetőségét a hallgató-osztály példán bemutattuk. A kapcsolat leírás módja függ az adatmodelltől, ezért ezt ott tárgyaljuk majd részletesebben.

2.3.3 Belső szint

Leírja a tárolás és az elérés módját.

A *tárolási mód* megadja, melyik lemezen, mely cilindereken, mely sávokon és blokkokban (lapokban) helyezkednek el az adott rekordtípus egyes rekordjai, ezen belül a rekordok egyes mezői, milyen konverziót kell elvégezni a fizikailag tárolt adatokon, hogy a felhasználóknak megfelelő formában álljanak rendelkezésre.

Az *elérési mód*ban megadjuk hogyan, milyen rendben helyezkednek el a rekordok, hogyan érhetjük el őket. Tipikus elérési módok:

- fizikai sorrendben,
- valamilyen mező szerint, logikai sorrendben,
- közvetlenül indexek alapján.

A belső szinttel, miután ennek meghatározása az adatbázis működését gyakorlati szempontból igen erősen befolyásolja, a 4. fejezetben részletesen foglalkozunk.

2.4 Adatbázis-kezelő rendszer komponensei

Minden adatbázis-kezelő rendszer tartalmaz egy vezérlő programot, mely összehangolja az egyes komponensek működését.

Ennek részei a következők:

- *Kapcsolattartó*. Ez teszi lehetővé, hogy az egyes alkalmazások kapcsolatba tudjanak lépni az adatbázissal.
- *Feladat átkonvertáló*. Az adatbázis-kezelő rendszer nyelvén megfogalmazott utasításokat értékeli, elemzi és ha végrehajthatók, meghatározza a végrehajtás várhatóan leghatékonyabb módját.
- *Elérés meghatározó*. Az objektumok elérésére meghatározott módszert átfordítja az operációs rendszer számára érthető parancsokká. Elindítja, majd értékeli ezen parancsok végrehajtását.

Az alábbiakban röviden ismertetendő komponensek mind szükségesek egy jól működő, nagy számítógépes adatbázis-kezelő rendszerben. Ennek ellenére, nem mindegyik rendszer biztosítja az itt felvázolt lehetőségeket, illetve lehet, hogy csak külön kiegészítésként, esetleg máshonnan beszerezett szoftver termék hozzáillesztésével lehet a kívánt funkciót megvalósítani.

Egy korszerű adatbázis-kezelő szoftvertől megkövetelhető, hogy a következő szolgáltatásokat megfelelő szinten biztosítsa:

- valamennyire tudják használni csekély számítástechnikai ismeretekkel rendelkezők is,
- legyen felhasználóbarát lekérdező nyelve és jelentéskészítője (report generátor),
- biztosítsa a rendszer legfontosabb jellemzőinek automatikus dokumentálását,
- képernyőhasználat egyszerű legyen,
- párbeszédesen (interaktív módon) lehessen használni,
- egyszerűen lehessen az adatbáziskezelő utasításokból (magasszintű) programnyelvi utasításokat generálni.

2.4.1 Adatleíró nyelv

Az adatmodellnek, a leírásnak a további számítógépes feldolgozásra is alkalmas tartalmi és formai megfogalmazását *sémának* nevezzük. Ennek szabatos megfogalmazására szolgál az *adatleíró nyelv* (Data Definition Language, DDL). A legtöbb

adatbázis-kezelő rendszerben ehhez kapcsolódnak azok a leírások, *alsémák* (külső szintek), melyek megmutatják, hogy az egyes alkalmazási programok az adatbázist hogyan látják. Minden adatbázishoz egyetlenegy séma tartozik, amely pontosan meghatározza az adatszerkezetet és a tárolási struktúrát, valamint az egyes adatelemek között fennálló logikai kapcsolatokat. Ezenkívül a fizikai struktúrák és kapcsolatok leírása is részben ezen a szinten történik. Egy adatbázis sémához akárhány alséma illeszthető, melyekben az adatokat újra el lehet nevezni, típusuk, csoportosításuk eltérhet a sémában meghatározott szerkezettől. Az alsémában meghatározott kapcsolatokat az adatbázis-kezelő rendszer alakítja át a sémának megfelelő formába, és a szükséges konverziókat is elvégzi. Az átalakítás folyamata, az adatbázisnak az alsémával nem definiált része, sőt az alsémának a fizikai szerkezete is rejtve marad a felhasználó előtt. Az adatleíró nyelvnek tehát három különböző szinten kell leírnia az adatokat:

- a) A felhasználó szintjén, amely a tényleges adatbázis valamilyen alsémáját, az adatmodell egy külső szintjét jelenti. Ez történhet egy külön erre a célra szolgáló adatleíró nyelven is, de igen gyakran megadható az alkalmazási program nyelvén is.
- b) Az adatok logikai kapcsolatának szintjén, mely a teljes logikai séma, a koncepcionális modell leírását tartalmazza.
- c) Az adatok fizikai tárolásának meghatározása szintjén, mely az egyes adatok fizikai elérhetőségét, tárolási módját és a köztük fennálló fizikai kapcsolatokat (pointerek, indexek, hashing rutinok stb.) írja le. E két utóbbi funkciót rendszerint egy önálló nyelv segítségével valósítják meg.

Az adatbázisnak nemcsak az adatokat, hanem a köztük fennálló logikai kapcsolatokat is tartalmaznia kell. Így például egy személyzeti nyilvántartásban az osztályok és a dolgozók is egyaránt adatok, de ugyanakkor kapcsolat áll fenn köztük, tudniillik az, hogy az osztályokhoz tartoznak a dolgozók, és az egyes dolgozók valamelyik osztálynak tagjai vagy vezetői. Ezek a kapcsolatok ugyanúgy részei az adatbázisnak, mint az adatok, bár ábrázolásuk történhet más módon is (pl. pointerek segítségével). Az adatleíró nyelvnek ezen kapcsolatok leírására is alkalmasnak kell lennie.

Amíg azonban az egyes rekordok leírása a különböző adatmodellekben lényegében hasonló módon történik a modell mindhárom szintjén, addig a rekordok közti kapcsolatok leírásának módja erősen modellfüggő, mint azt a II. részben látni fogjuk.

2.4.2 Adatkezelő nyelv

A feldolgozás céljaira szükségünk van egy adatkezelő nyelvre (Data Manipulation Language, DML), melynek segítségével a felhasználó az adatleíró nyelvvel definiált adatbázison műveleteket végezhet. A műveletek az adatbázis adatainak előállítására, módosítására vagy lekérdezésére vonatkoznak, és módot adnak az adatbázis (alsémában meghatározott részének) megnyitására és lezárására, a benne levő rekordok kulcs vagy relatív cím (következő, előző, első, utolsó) alapján történő megkeresésére, a megtalált rekord beolvasására, módosított rekord kiírására, új

rekord kulcs vagy cím szerinti beszűrésére, rekord teljes vagy részleges törlésére, rekord egyes mezőinek módosítására.

Amennyiben ezek a változtatások olyan mezőket is érintenek, melyhez a fizikai szinten valamilyen elérési struktúra is hozzátartozik (pl. indexek), akkor az adatbázis-kezelő rendszer ennek karbantartását is automatikusan elvégzi.

Az adatkezelő nyelv lehet önálló nyelv (Self-Contained System), de beépülhet valamilyen magas szintű programozási nyelvbe is (Host Language System), mint azt az 1.3.3 pontban ismertettük.

A régebbi adatbázis-kezelő rendszerek adatkezelő nyelve eljárás orientált volt. A felhasználónak le kellett írnia, hogyan, milyen úton akarja elérni a kívánt adatokat az adatbázisban. Ehhez jól kellett ismernie az adatbázis szerkezetét, azt, hogy milyen módon haladhat a kívánt cél felé a rekordok közti kapcsolatok alapján. Ezt úgy is mondták annak idején, hogy navigál az adatbázisban. Aki jól ismerte a különböző utakat, nagyon hatékonyan navigálhatott. Aki viszont nem volt kellő mértékben tisztában ezekkel, az, ha nem is hajózta kétszer körbe a Földet, bizony gyakran fölöslegesen vizsgált végig több ezer rekordot, míg eljutott az Ígéret Földjére.

Az óceánokat átszelő hajókat felváltják a transzkontinentális repülőgépek. A „navigare necesse est” mondás új értelmet kap. Napjaink korszerű relációs adatbázis-kezelő rendszerei automatizálták az adatbázisban való útkeresést. A felhasználó csak azt mondja meg, mi a célja. Az optimális utat a rendszer maga határozza meg.

Természetesen ehhez sok olyan információt kell a rendszerben saját magáról naprakészen tárolni, mely ezt az optimalizálást lehetővé teszi. Ez tovább növeli a rendszer saját adminisztrációját. Ezen kívül speciális feladatok megoldásánál a szakértő alkalmazónak lehetnek olyan ismeretei, amelyeket a rendszer automatikusan nem tudhat. Ez fordulhat elő, ha egy adott összefüggésben egy általában nem túl szelektív feltétel igen szelektívvé válhat. Például egy kórházi nyilvántartásban egyéb feltételek teljesülése mellett csak valamely igen ritka vércsoportba tartozó egyedek érdekesek. Ekkor célszerűbb lehet a szóbajöhethető adatokat először ezen vércsoport szerint kiválasztani, mert csak néhányat kell a további feltételek szempontjából is megvizsgálni. A gyakori vércsoportoknál ez az út több ezer rekord felesleges beolvasását és kiértékelését jelentené. Ezért a rendszer automatikus elérési út generálója nem a vércsoport szerint kezdi meg a kiválasztást, holott ez az adott esetben nagyságrendekkel gyorsabb lehet.

Körülbelül úgy jellemezhetjük ezt, mint az automata sebességváltót. A városi csúcsforgalomban araszoló autós számára az automata sebességváltó sokkal egyszerűbb, kényelmesebb és általában gazdaságosabb, mint a kézi. De az üres országúton vagy a versenypályán száguldozó hivatásos vezető jobban ki tudja használni a jármű adottságait, ha saját maga dönti el, mikor váltson sebességet.

2.4.3 Illesztés a programozási nyelvekhez

Ha az adatbázis-kezelő rendszer utasításait önálló nyelven interaktív módon adják ki, akkor ez a komponens teszi lehetővé, hogy a felhasználó a termináljáról kiad hassa ezeket az utasításokat, és ezek eredményét a terminálján nyomon követhesse.

se. Ha az adatbázis-kezelő nyelv programozási nyelvbe építhető be, akkor ennek segítségével tudja a megfelelő fordítóprogram értelmezni az adatleíró és adatkezelő utasításokat. Általában ezeket az utasításokat—a befogadó programnyelv és az adatbázis-kezelő nyelv szintaktikájának megfelelően—a programozók beírják a programba, amiket egy preprocesszor átalakít az adott programnyelvnek megfelelő utasításokká.

Ennek a preprocesszálnak vagy másnéven prekompilálásnak az előnyei a következők:

- A programokat abban a nyelvben lehet írni, amelyik a legjobban illeszkedik a feladathoz, vagy amelyiket az installációnál általánosan használnak, vagy amelyiket a programozó a legjobban ismer. (Az első az elvileg helyes, a másik kettő a gyakorlatban megvalósuló megoldás.)
- A programozóknak csak az adatkezelő (esetleg még az adatleíró) nyelvet kell külön megtanulniuk az adatbázis használatához.

A nagygépes adatbázis-kezelő rendszereknek általában van párbeszédés és beépíthetős változata is. Az utóbbiak általában COBOL-ba, PL/I-be, Assemblerbe illeszthetők be, de sok közülük FORTRAN-ba és C-be, és további nyelvekbe is beépíthető.

A személyi számítógépekre készült rendszerek többségénél nincs ilyen csatoló a befogadó nyelvekhez.

2.4.4 Jelentéskészítő (Report generátor)

Ha a kérdések eredménye nem néhány rekord, akkor azt célszerű megfelelően formált jelentés (report) alakjában kiíratni. Ez nem csak az egyes eredmény sorok szép elrendezését, fejlécet, lapváltást jelenti, hanem csoportváltást, megfelelő numerikus adatokra csoportösszegek, totál összegek képzését, sőt még akár grafikonok készítését is.

A jelentéskészítő program utasításait rendszerint az angolhoz közelálló speciális nyelven fogalmazzák meg. Ezek bonyolultsága és hatékonysága rendszertől függően igen széles határok között változhat. Vannak olyan, elsősorban programozók számára készült generátorok, melyeknek lehetőségei (és bonyolultsága is) csaknem megegyeznek egy programozási nyelvvel, és vannak olyanok, melyek lényegében csak egyszerű formatálást tudnak elvégezni, és a nem programozó alkalmazók számára nyújtanak segítséget.

Akárhogy is nézzük, egy jó report generátor—ha már egyszer megtanultuk a használatát—mindenképpen áttekinthetőbbé teszi az eredményeket és időt takarít meg mind a programozóknak, mind a végfelhasználóknak.

2.4.5 Képernyő-tervező

A felhasználóbarát környezethez hozzátartozik az is, hogy az adatokat illetve az adatbázis-kezelő utasításokat könnyen lehessen bevinni a terminálról. Az ehhez szükséges képernyő formátum megtervezése és elkészítése meglehetősen bonyo-

lult. Soronként és oszloponként kiszámolva meg kell határoznunk minden egyes felirat és mező helyét, be kell állítanunk rájuk a megfelelő képernyő atributeumokat.

Ezt a feladatot könnyítik meg a képernyő tervezők. A képernyőre „felrajzoljuk”, mit, hol és milyen formában kívánunk látni, illetve bevinni. Ezt a tervező „lefordítja” a képernyő-formátumot generáló utasításokká.

A személyi számítógépre készült adatbázis-kezelő rendszerek szinte mind rendelkeznek képernyő-tervezővel.

2.4.6 Szolgáltató program

Rendszeresen előforduló feladatok szabványos, paraméterezhető megoldására szolgálnak.

Leggyakoribb felhasználási területük:

- formázott adatok betöltése új adatbázis létrehozásakor, nagy tömegű adat bevitelekor, kimentett adatok helyreállításakor,
- adatbázis adatok formázott kimentése biztonsági okokból, vagy más rendszerbe való átvitelhez,
- adatbázis adatainak újraszervezése (kimentés és adott módon való újratöltés) az optimális hatékonysági szempontok figyelembevételével,
- főlössleges adatok tömeges törlése,
- tömeges, nem formázott kimentés biztonsági okokból és hiba esetén helyreállítási lehetőség,
- adatbázis tárolókivonat (dump) készítése, hogy az adatokat nem a logikai szerkezetükben, hanem a tényleges fizikai tárolási formában láthassuk.

2.4.7 Adatszótár (katalógus)

Lehetővé teszi az adatbázis-felügyelő, a programozók, rendszerszervezők és a végfelhasználók számára, hogy az adatbázist és az ahhoz kapcsolódó alkalmazásait egységesen, minden érdekelt számára hozzáférhető módon dokumentálják. Az adatszótár használata nagy mértékben lecsökkenti az adatbázissal és az alkalmazási rendszerekkel kapcsolatos papírmunkát.

Legegyszerűbb formájában az adatszótár, vagy más elnevezéssel a rendszerkatalógus az adatdefiniációkat, a rekordok és más objektumok közötti kapcsolatokat, az egyes objektumokra vonatkozó hozzáférési és műveleti jogokat, jelentésformátumokat, a külső szinten definiált nézeteket tartalmazza.

Fejlettebb változatai lehetővé teszik a meglevő bejegyzések alapján új objektumok generálását, különféle változtatások hatásának vizsgálatát a rendszerben és azok ellentmondásmentes végrehajtását, illetve logikai ellentmondás vagy hibalehetőség esetén felhívják a figyelmet annak következményeire, esetleg megakadályozzák a változtatás végrehajtását. Ez utóbbira tipikus példa, amikor a koncepciónális szinten törölni akarunk egy olyan rekord típust, amelyik része, vagy megegyezik valamely külső szint létező rekordtípusával. A rendszertől függ, hogy ezt a törlést automatikusan végrehajtja-e, figyelmeztet-e erre a függőségre, és csak ha ezután megerősítjük a kérést, akkor végzi el a törlést, vagy egyáltalában

nem hajlandó addig törölni, amíg vannak élő hivatkozások. Mint azt a katalógusról szóló részben tárgyalni fogjuk, mindhárom módszernek megvannak az előnyei és hátrányai is. Természetesen az automatikus törlésnél az adatszótár segítségével meg kell tudnunk nézni, hogy az esetleges törlés milyen más objektumokat érint-het.

Sok adatszótár tartalmazhat információkat olyan állományokról is, melyek nem részei az adatbázisnak és a lefordított programokról is, melyek az adatbázissal dolgoznak.

2.4.8 Adatbiztonság

Fokozott gondot kell fordítani az adatbiztonságra és az adatok sérthetetlenségére, mivel egyidejűleg vagy egymás után több program is használja ugyanazokat az adatokat. Szigorúan szabályozni kell, hogy ki jogosult egyes adattípusok olvasására és módosítására, és ezt a rendszernek automatikusan kell ellenőriznie. Védni kell az adatbázist az illetéktelen hozzáféréstől, mely lehet szándékos, de véletlen is (például hibás program miatt). Ezenkívül biztosítani kell azt is, hogy a módosítások minden szinten következetesen végrehajtásra kerüljenek, a különböző programok konzisztens adatokkal dolgozzanak, és esetleges hiba esetén az adatbázis a hiba előtti állapotba visszaállítható legyen. Mivel egyre több intézmény teljes információs rendszere csak számítógépen lévő adatokra és azokat kezelő programokra épül fel, biztosítani kell ezeket is szándékos vagy véletlen emberi beavatkozásból, elemi csapásoktól bekövetkező meghibásodás vagy megsemmisülés ellen.

2.5 Az integrált adatbázis előnyei

A fentieket összefoglalva az adatbázisra és az adatbázis-kezelő rendszerekre a következő meghatározást adhatjuk:

Az integrált adatbázis olyan egymással kapcsolatban álló adatok összessége, melyeket különféle felhasználók különböző csoportosításban használhatnak. Az adatok fizikai elhelyezése központilag, minimális redundanciával történik. Ugyancsak központilag ellenőrzött az adatok védelme, új adatok bevitele és meglevő adatok módosítása is. Az adatbázis-kezelő rendszer lehetővé teszi az arra jogosult felhasználóknak az adatbázishoz való hozzáférést úgy, hogy a felhasználói programok az adatok tárolási módjától függetlenül lehessenek.

Az integrált adatbázis előnyeit a következőkben foglalhatjuk össze:

- az adatbázist használó intézmény centralizáltan ellenőrizheti a működéséhez szükséges összes adatot,
- az adatok többszörös tárolása megszüntethető, ezáltal a különböző helyeken tárolt, de azonos adatok aktuális értéke közötti esetleges ellentmondás kiküszöbölhető,
- az adatok védelme illetéktelen hozzáféréstől központilag jobban megvalósítható,
- az adatok közti logikai ellentmondások jobban kiszűrhetők,

- a különböző kérdéseket különféle formában lehet megfogalmazni, az egyes adatok között fennálló kapcsolatok jobban értékelhetők,
- a teljes intézmény szempontjából a feldolgozás hatékonyabb lehet.

Azért, hogy az adatbázis mindezen követelményeknek eleget tegyen, leírását, létrehozását, karbantartását, elérési módjának szabályozását, az igényeknek megfelelő időnkénti átszervezését központilag kell végezni. Ezért a tevékenységéért az *adatbázis-felügyelő* a felelős, akinek feladatait a következő pontban tárgyaljuk részletesebben.

A hetvenes évek végéig a logikailag és fizikailag is integrált adatbázisnak, azaz annak, hogy az adatokat fizikailag is egy helyen tárolják, az előnyei nyilvánvalóak voltak. A nyolcvanas évek elejére azonban kiderült a fizikai centralizáltság néhány szűk keresztmetszete is. Ezek közül a leglényegesebb abból adódott, hogy bizonyos adatcsoportoknak a felhasználására legtöbbször csak egy adott helyről volt szükség, de az adatoknak mégis mindig végig kellett haladniuk az adatátviteli hálózaton. Ezáltal a lényegesen nagyobb, mindent mindig elérhető módon tárolni képes jóval drágább központi gépen kívül jelentősen megnövekedtek az adatátviteli hálózat méretei és költségei is. Ennek kiküszöbölésére születtek meg az *osztott adatbázisok*, melyek lehetővé teszik a fizikailag különböző helyeken levő adatbázisok logikailag egységes kezelését és az adatoknak az egyes adatbázisok között központilag ellenőrzött módon történő áramlását. Az osztott adatbázisok legfontosabb tulajdonságait a 2.9 pontban külön tárgyaljuk.

2.6 Adatbázis-felügyelő

Az adatbázis technológia bevezetése egy intézménynél, vállalatnál maga után vonja a szervezet és a szervezési módszerek megváltoztatását is. Az integrált információs rendszer módosítja az emberek feladatait, munkáját, és ami még ennél is fontosabb, a gondolkodásmódjukat. Ha ezek nem következnek be, akkor a cég nem tudja kihasználni az adatbázis-technológia, az adatbázis-kezelő rendszer alkalmazásának előnyeit, sőt, sokkal drágábban, esetleg sokkal később fog a hiányosabb információkhoz hozzájutni.

Ahhoz, hogy az adatbázist megfelelő módon alakítsák ki és használják, egy új funkciót kellett létrehozni, az adatbázis-felügyelő feladatkörét.

Ezt ma már nagy intézményeknél a legtöbb esetben nem egy személy, hanem egy csoport látja el.

A 70-es, 80-as évek informatikában mindent tudó adatbázis-felügyelőjének feladata egyre inkább kétfelé osztódik.

Az elsősorban a számítógépes hardverrel és szoftverrel kapcsolatos technikai feladatok irányítását és ellenőrzését végzi az adatbázis-felügyelő.

Más irányú képességek és szakértelem szükséges az érdemi információs igényeket meghatározó felhasználókkal való kapcsolattartáshoz, ezen igények egyeztetéséhez, kielégítésük prioritásának meghatározásához. Régebben ezeket a feladatokat is az adatbázis-felügyelő látta el. Ma már ezt a személyt inkább az *informatikai erőforrások felügyelőjének* nevezzük, aki a szervezetet ismeri kiválóan, de azért jó számítástechnikai szakember. Így szót tud érteni az adatbázis-felügyelővel, aki viszont elsőrangú számítástechnikai szakember az adatbázis-technológia

területén, de azért jól ismeri a szervezet feladatait is, hogy megértse és megfelelően meg is tudja valósítani azt, amit partnere, az erőforrás-felügyelő szeretne.

A feladatkör eme szétesztása ellenére a továbbiakban egységesen az adatbázis-felügyelő feladatairól fogunk beszélni. A munka jellege szabja meg, hogy ebből a „kettős én”-ből melyikük teendőire gondolunk. A 2.1 pont elején felsorolt követelmények kielégítésében különböző irányokból elindulva közösen kell munkálkodniuk és csak az utolsó pont, mely szerint az információ csak akkor ér valamit, ha azt jól fel is használják, esik kizárólagosan az erőforrás-felügyelő hatáskörébe.

Az adatbázis-kezelő rendszerek állandó fejlődése nem vonja automatikusan maga után a felhasználók adatfeldolgozói munkájának javulását is. Egy jó adatbázis-kezelő rendszer önmagában még nem elegendő, azt megfelelő módon kell használni egy célszerűen felépített adatbázison. Ha az adatbázist egyszerűen a már meglevő állományok egyesítésével hozzuk létre, akkor a programozók ugyanúgy fogják használni a megszokott adatneveket a duplikált adatokra, mint régen. Így lényegében semmi sem változik azon kívül, hogy a korábbi egyszerűbb módszerek helyett egy jelentősen bonyolultabb (és drágább!) eljárást használunk, melynek mind a hardverkonfigurációja, mind a gépidőigénye nagyobb. Az adatok teljes redundanciamentes tárolása igen nagy mértékben megnövelheti a hozzáférési időt, de ugyanehhez a rossz eredményhez juthatunk célszerűen kialakított redundancia esetén is, ha a több név alatt többszörösen tárolt adatokra nem a megfelelő névvel hivatkozunk. A gyakorlatban ilyen hibák a legtöbbször az alábbi okok miatt következnek be:

- nyílt logikai átfedés: ugyanaz az adattípus azonos névvel több állományban is szerepel,
- rejtett logikai átfedés: ugyanaz az adattípus különböző elnevezéssel bár, de több állományban szerepel,
- látszólagos logikai átfedés: ugyanaz az elnevezés különböző adatállományokban más adattípust jelent,
- fizikai átfedés: ugyanazon adatokat többszörösen tároljuk,
- logikai átfedés hiánya: a különböző adatállományok között a megfelelő logikai kapcsolatot nem tudjuk létrehozni. (Pl. az árucikkekről és a raktárakról is egyedenként külön-külön minden adatunk megvan, de azt nem tudjuk ebből megállapítani, hogy melyik raktárban milyen cikkek vannak.),
- eltérő kódolás: ugyanannak a fogalomnak az azonosítására többfajta kódrendszert használunk.

A hatékony feldolgozáshoz szükséges feltételek kialakítását, a változások konzisztens keresztülvitelét, az adatbázis tartalmának rendszeres kimentését, a hozzáférési jog szabályozását nem lehet az egyes felhasználók kényére-kedvére bízni. Ezeket a feladatokat központilag kell irányítani. Ezt a munkát látja el az adatbázis-felügyelő vagy más néven az adatbázis-felelős.

Az adatbázis-felügyelő fő feladatai a következők:

- az adatbázis megszervezése. Ez rendszerint magában foglalja az adatmodell kialakítását, az adatbázis leírását, a definiált adatok elnevezését, amennyiben mód van rá, a keresési stratégiák megválasztását, a hozzáférési jogok meghatározását és az adatbázis betöltését,
- az adatbázist használók segítése,

- az adatbázisba bevitt adatok helyességének ellenőrzése, illetve segítségnyújtás az erre kijelölt felhasználóknak az ellenőrzés elvégzéséhez,
- az adatbázis adatainak rendszeres kimentése, hogy esetleges hibák után az eredeti helyes állapotot vissza tudja állítani,
- az adatbázis használatának állandó figyelése, a hatásfok jelentős csökkenése esetén az adatbázis újraszervezése,
- a felhasználók igényei alapján az adatbázis struktúrájának módosítása.

Az adatbázis-felügyelő munkája azért is igen bonyolult, mert az adatbázis létrehozásakor és karbantartásakor igen sok, egymással gyakran ellentétes szempontot és érdeket kell figyelembe venni. Ráadásul az egyes szempontok rangsorolása sem mindig egyértelmű, és az idők folyamán változhat is.

2.1 táblázat

Adatbázisok létrehozásakor és használatakor elérendő legfontosabb célok

	Cél	A cél eléréséhez rendelkezésre álló módszer
E l s ő d l e g e s c é l ő k	Több felhasználó különféle igényeinek kielégítése Kompatibilitás a régi programokkal és adatokkal, bővítési lehetőség a várható igényekre Változtatási költségek minimalizálása Működtetési költségek minimalizálása Adatok pontossága, ellentmondásmentessége, biztonsága, titkossága	Fizikai és logikai adatfüggetlenség, adatkezelő nyelv(ek) Fizikai és logikai adatfüggetlenség, adatleíró nyelv, séma és alsémák létrehozása Fizikai és logikai adatfüggetlenség, adatleíró és adatkezelő nyelv, séma és alsémák létrehozása Ellenőrzött minimális redundancia, hatékony keresési és elérési módok, adatok standardizálása Felhasználók központi ellenőrzése, minimális redundancia, automatikus védelem illetéktelen felhasználók ellen, az adatbázis központi helyreállításának lehetősége
T o v á b b i f o n t o s c é l ő k	Új adatgyűjtések lecsökkentése Egyszerű használat Egyszerű változtatási lehetőségek Igény szerinti rendelkezésre állás Központi ellenőrzési lehetőségek Hibák gyors kiküszöbölése	Meglevő adatok sokoldalú fölhasználásának biztosítása, adatleíró nyelv, új alsémák létrehozása Magas szintű nyelvekbe való beépíthetőség, egyszerű lekérdező nyelvek, távfeldolgozási lehetőségek Fizikai és logikai adatfüggetlenség, adatleíró nyelv, új alsémák létrehozása Hatékony keresési és elérési módok, on-line hozzáférés Hozzáférést, változtatást, hatékonyságot automatikusan ellenőrző programok Változtatásokat automatikusan rögzítő és hiba után a rendszert helyreállító és újraindító programok

A 2.1 táblázatban összefoglaltuk ezek közül a legfontosabbakat. Ismételten hangsúlyozni kívánjuk azonban, hogy a fontossági besorolás csak relatív, és első-sorban az adott információs rendszer igényeitől függően változhat. Az adatbázisok gyakorlati alkalmazásakor az adatbázis-felügyelő legnehezebb feladata az információs igények felmérése és ennek alapján az információs szerkezet megtervezése, valamint ennek olyan fizikai tárolási struktúrába való elrendezése, melyet az adatbázis-kezelő rendszer kezelni tud.

A tervezés első lépése az adatmodell különböző szintjeinek elkészítése. Ennek érdekében pontosan meg kell határozni a felhasználók számára fontos adatokat és a közöttük fennálló kapcsolatokat. Ez általában igen nehéz feladat, mert ehhez ismerni kell az adatbázist használni kívánó szervezetnek vagy intézménynek nemcsak a jelenlegi felépítését és információáramlási folyamatát, hanem annak várható jövőbeni növekedését, változását is.

A koncepcionális adatmodell-tervezési folyamat végére az adatbázis-felügyelőnek meg kell határoznia és el kell neveznie az adatbázisba kerülő összes elemet, azok tulajdonságait és kapcsolataikat. Meg kell állapítania, melyek azok a tulajdonságok, melyeknek értékei azonosítják az egyes elemeket (kulcsok), és melyek azok, amelyek más tulajdonságaikat jellemezik.

A következő lépés a fizikai adatbázis megtervezése, a rekordok létrehozása, elhelyezése, a keresési és elérési módok meghatározása. Ezeket az adatleíró nyelven megfogalmazott feltételeket az adatbázis-kezelő rendszer táblázatokká, rutinokká fordítja le magának, és a konkrét használatkor e táblázatokból veszi ki a szükséges információkat.

Egyes adatbázis-kezelő módszerek a logikai leírás alapján automatikusan állítják elő a tárolási struktúrát. Ezeket lényegesen egyszerűbb használni, de az így létrejött rendszer gyakran nem működik elég hatékonyan. Ezért a legtöbb rendszer bizonyos szabadságot ad az adatbázis-felügyelőnek. Az adatbázis-felügyelő dönti el (ha a rendszer választási lehetőséget nyújt), mely kulcsokat lehet indexek alapján, melyeket pedig hashing-gel elérni, milyen algoritmussal történjen a hashing (lásd 4.8 alpontot), a szekvencialitás milyen kulcs alapján valósuljon meg fizikailag, melyekre pointerrek segítségével, a pointerlánc egy- vagy kétirányú legyen, hierarchikus szevezéskor (lásd a 4.6 és 5.3.2 pontot) milyen bejárési utat biztosítson stb.

Ahhoz, hogy az adatbázis-kezelő rendszer által biztosított lehetőségekből az összes felhasználó súlyozott igényeit figyelembe vevő leghatékonyabb módszert tudja kiválasztani, az adatbázis-felügyelőnek jól kell ismernie nemcsak az adatbázis-kezelő rendszert és az adatkezelő szoftvert, hanem a felhasználói igényeket is. Mégis, bármennyire sokoldalú is egy nagyobb adatbázis-kezelő rendszer, előfordulhat, hogy bizonyos logikai kapcsolatokat közvetlenül nem képes kezelni. Például az egyszerű hierarchikus rendszerekben (lásd az 5. fejezetet) $M:N$ kapcsolat nem valósítható meg. Sok relációs rendszer nem támogatja a táblázatok közti kereszthivatkozásnál az idegen kulcsokat (l. 7.4.3 pontot). Ebben az esetben az adatbázis-felügyelő feladata az is, hogy olyan további rutinokkal egészítse ki a rendszert, melyek lehetővé teszik ezeknek az összefüggéseknek az ábrázolását, vagy úgy módosítsa a logikai információrendszert, hogy az illeszkedjék a meglévő adatbázis-kezelő rendszer lehetőségeihez.

Miután az adatbázis-felügyelő létrehozta az adatbázis koncepcionális és a fizikai szintjét, kialakítja az adatmodellnek az egyes felhasználók igényeinek megfelelő külső szintjeit is.

2.7 Néhány hátrány

Az eddigiekben kizárólag az adatbázisok és az adatbázis-kezelő rendszerek használatának előnyeit ecseteltük. Mára azonban nyilvánvalóvá vált, hogy az élet minden területére érvényes az eredetileg közgazdászok által fölfedezett alapigazság, mely angolul így hangzik: „There are no free lunches”. Ez hétköznapi magyar nyelvre lefordítva azt jelenti: „Nincs potya kaja”. Az adatbázis használatának előnyeiért valamilyen más módon fizetnünk kell. Legfőljebb azt mondhatjuk, hogy a legtöbb esetben az előnyök kompenzálják a hátrányokat, így végső fokon mégis érdemes az adatbázis kezelő rendszer mellett döntenünk.

Az alábbiakban felsoroljuk a legfontosabb érveket, melyek egy ilyen rendszer ellen, vagy legalábbis nem egyértelműen mellette szólnak.

Az első, és talán a legfontosabb érv, hogy nincs olyan szervezet, amelyik minden általa használt adatot egy egységes integrált adatbázisban, vagy akár annak korszerűbb, osztott formájában tárolna. Murphy törvénye alapján, ha létezne egy olyan adatbázis, amely egy (akármilyen kicsi) vállalat összes adatát, ezek kapcsolatait, a hozzáférési jogosultságokat mind tartalmazná, ennek kezelése olyan bonyolult lenne, hogy senki sem használná. Nem beszélve még arról a „csekély-ségről”, hogy egy ilyen adatbázisnak a megtervezése, létrehozása, élő adatokkal való feltöltése és állandó naprakész állapotba való tartása annyi pénzt, időt és emberi munkát emésztene fel, hogy a vállalat alapvető tevékenységéhez már nem is maradna semmi ezekből az erőforrásokból. Így az a vállalat, amelyik csupán egyetlen integrált adatbázisra építené az összes tevékenységét, 9 hónapon belül csődbe jutna. (A szerző kiegészítése Murphy előbb idézett alaptételéhez.)

Mindig lesznek olyan—esetleg csak helyi—alkalmazások, melyek saját adatokkal, saját adatbázisokkal, sőt, az alkalmazás egyszerűsége és kifejlesztésének gyorsasága miatt, egyszerű állományokkal dolgoznak. A legtöbb adatbázis-kezelő rendszer az ilyen „közönséges” állományokat nem is tudja kezelni, csak akkor, ha ezek adatait előbb betöltik az adatbázisba. Ezzel pedig az adatbázisok szintjén állítottuk elő a „kétszer tároljuk, kétszer kell karbantartani, hogyan biztosítsuk a kétféle adatállomány konzisztenciáját” klasszikus problémáját. Kevés olyan rendszer van, mint az IBM AS/400 gépein futó SQL/400 relációs adatbázis-kezelő, mely teljesen transzparens az adatszerkezetre. Ez a hagyományos állományokon ugyanúgy tud dolgozni az adatbázis-kezelő nyelv utasításaival, mint az adatbázis táblázatain, és az állománykezelő utasításokkal is szabadon dolgozhatunk az adatbázis objektumain.

További komoly hátrány, hogy az adatbázis-kezelő rendszer drága, számítógépes erőforrás igénye nagy, és a file kezeléshez viszonyítva sokszor bonyolult is, megtanulása hosszabb időt igényel.

Az állományok használatáról az adatbázis technológiára való áttérés mindig sok pénzbe és időbe kerül. Meg kell tanulni az új technikát és ehhez rendszerint

külső szakértőket is segítségül kell hívni, ami, ha hasznos tanácsokat kívánunk kapni, drága mulatság.

Ezeket a problémákat nem azért soroljuk fel, hogy bárkit lebeszéljünk az adatbázisok használatáról, csupán az adatbázisok mindenhatóságába vetett irreális elvárásokat és tévhiteket szeretnénk eloszlatni.

Tovább növeli a költségeket a régi adatoknak az adatbázisba való átvétele. Ilyenkor derül ugyanis ki, hogy a korábban ellentmondásmentesnek és hibátlan-nak tartott adatok között az új rendszer szigorúbb követelményei és jobb ellenőrzési lehetőségei mennyi inkonzisztenciát, ellentmondást tárnak föl. Ezek kiküszöbölése természetesen pontosabbá, jobbá teszi információinkat, de a dolog jellegéből következően általában csak egyedileg, az egyes adatok konkrét megvizsgálásával és módosításával hajtható végre. Ezért ez igen hosszadalmas és sok emberi munkát is igénylő, tehát drága folyamat.

Ehhez még hozzájárul, hogy az eddigi programokat át kell írni, hogy használni tudják az adatbázist. Amíg az új programok megbízható ellenőrzése nem történt meg, a régi és az új rendszert párhuzamosan kell futtatni.

Az adatbázis-kezelő rendszerek eleve azzal a céllal készülnek, hogy különféle adatstruktúrákon, sokféle alkalmazásra lehessen őket felhasználni. Így mindenkinek nyújtanak valamit, de általános céljukból következik, hogy a kifejezetten egy feladat megoldására készült, annak speciális tulajdonságait figyelembe vevő hagyományos állományokkal dolgozó célrendszer arra a feladatra rendszerint hatékonyabb. Hangsúlyozott szerepet kap ez az ellenérv, ha az alkalmazás nem túl komplex, nagyszámú tranzakciót jelent, és a szervezet fő profiljába tartozó feladat megoldására szolgál.

Adatbázis-kezelő rendszerrel dolgozva az adatok és az alkalmazás között a kapcsolat mindig több szoftver-rétegen keresztül valósul meg, ami nagyobb tárgényben és főleg hosszabb futási időkben jelentkezik. Jelentős ez a probléma nagy adatmennyiségnél és/vagy nagy tranzakció számnál, amikor a hardver erőforrások úgyszólván erősen leterheltek.

További gondot jelent a gyakorlatban, hogy a legtöbb cég bizonyos konkrét feladatok megoldására készen vásárolt szoftvert használ. Nem biztos, hogy ez a szoftver együtt tud működni a központi adatbázis-kezelő rendszerrel. Gyakran a két szoftver még adat-szinten sem kompatibilis. Egyik sem tudja elolvasni a másik által készített adatokat. Ha már üzemel egy adatbázis rendszerünk, akkor egy új feladat megoldására a többféle lehetőség közül sokszor egy drágább, vagy rosszabb paraméterekkel rendelkező szoftver mellett vagyunk kénytelenek dönteni, mert csak ez tud együttműködni az adatbázis-kezelő rendszerünkkel.

2.8 Személyi számítógépek adatbázis-kezelő rendszerei

Eddigi megfontolásaink elsősorban sokfelhasználós integrált adatbázisokra érvényesek. Ezeket egy vagy több nagy (mainframe) számítógépen, illetve szerver géppel rendelkező PC hálózatokon installálták.

Az önállóan üzemeltetett személyi számítógépek adatbázis-kezelő rendszerei némileg különböznek nagygépes, gazdagabb rokonaiktól. A főbb eltérések a következők:

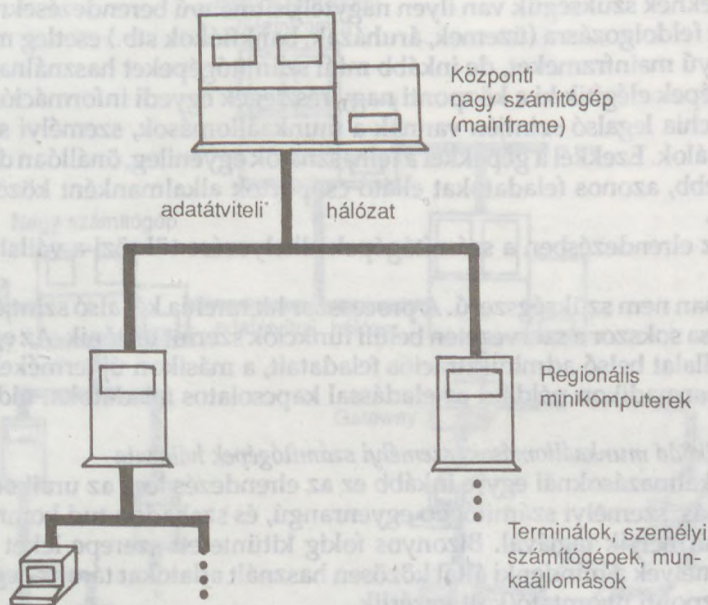
- A személyi számítógépek adatbázis-kezelő rendszerei általában csak egyszerűbb és kevesebbféle adatszerkezetet tudnak kezelni. Kizárólag a relációs adatmodellen alapulnak, de az ott javasolt funkcióknak is csak egy részét valósítják meg.
- Adatvédelmük, a hozzáférési jogok részletes szabályozása és ellenőrzése gyengébb.
- Biztonsági másolat készítése, hiba esetén az eredeti állapot visszaállítása nem, vagy csak kevésbé automatizált. Holott egy kis cég számára két 10 000 forintos megrendelés elvesztése számítógépes rendszerhiba miatt (viszonylag) ugyanakkora kárt okozhat, mint egy nagy cég nagy rendszerében több száz hasonló rekord eltűnése.
- Mivel a géppel egyszerre csak egy felhasználó dolgozik, az egyidejű hozzáférésből adódó problémákra nincsen védelem (zárak). Ezért aztán az ilyen rendszereket később nehéz egy hálózatba beilleszteni. Az élet és a piac azonban ezt sokszor kikényszeríti. Az így, utólag megbütykölt rendszerekben a záruk kezelése sokkal nehezkesebb és jóval több problémát és mérget okoz a felhasználóknak, mint a nagygépes rendszereknél, melyeket már eleve ezen szempontok figyelembevételével terveztek.
- Az adatbázis-kezelő rendszer beszerzési és a működéséhez szükséges további extra hardver/szoftver költségei nem csak abszolút értelemben sokkal kisebbek, hanem relatíve is. (Pl. a teljes számítógépes rendszer vagy az adatbázis-kezelő rendszer által szolgáltatott információk értékéhez viszonyítva.) Különösen érvényes ez a szoftverre, hiszen egy nagygépes rendszert esetleg csak néhány száz helyre tudnak eladni, míg egy sikeres PC-s szoftvert akár tíz- sőt százezres példányban is megvásárolnak.
- A cég szervezetére, szervezeti felépítésére az adatbázis technológia bevezetése általában nincs, vagy csak jóval kisebb hatással van.
- A rendszer (az árral arányosan) jóval hamarabb „telítődik”. Míg a jól működő nagygépes rendszerek csak igen nagy adatmennyiség feldolgozásakor lassulnak le számottevően, a PC-s gépeknél összetettebb adatszerkezeteknél ez már néhány ezer rekordnál is bekövetkezhet.

2.9 Osztott adatbázisok

2.9.1 Osztott rendszerek

Ahogy a 60-as, 70-es években a számítástechnikai eszközök koncentrációja jelentette a haladás irányát, úgy a nyolcvanas évektől egyre inkább előtérbe került az osztott rendszerek és ezeken belül az osztott adatbázisok létrehozása.

Az osztott rendszereket elsősorban azzal szokták jellemezni, miként helyezkednek el térbelileg a hozzátartozó számítógépek. A gyakorlatban azonban ez csak



2.2 ábra Processzor hierarchia

az egyik jellemzőjük. A legfontosabb feladatok, melyeket az osztott rendszer tervezésénél és működésénél figyelembe kell vennünk, az alábbiak:

- Hol történik az adatok számítógépes feldolgozása (hol vannak helyileg a processzorok)?
- Milyen kommunikációs kapcsolatok vannak közöttük?
- Hol tárolják a feldolgozáshoz szükséges adatokat?
- Milyen szabályok, szabványok érvényesek a feldolgozásra, az adat- és információ továbbításra?

Technikai szempontból az osztott rendszereknek négy fő típusát különböztetjük meg.

1. Hierarchikusan elhelyezkedő processzorok

Az osztott erőforrások szokásos háromszintű modelljében legfőleg találjuk a központi, nagyteljesítményű számítógépet (mainframe). Ehhez kapcsolódnak a helyi minikomputerek és ezekhez kapcsolódnak a végfelhasználók munkaállomásai, személyi számítógépei, termináljai (2.2 ábra).

Az adatok tárolása és feldolgozása a hierarchia bármelyik szintjén lehetséges. Általában egy adott típusú feldolgozás mindig ugyanazon a szinten történik. Megjegyezzük továbbá, hogy nem kötelező a háromszintű hierarchia sem. A szintek száma lehet ennél több, vagy kevesebb is.

A processzor hierarchia rendszerint illeszkedik a szervezeti hierarchiához is. A nagy számítógépek általában a szervezet középpontjában, helyi központjában találhatók. (Természetesen itt most olyan nagy vállalatokra, intézményekre gondo-

lunk, melyeknek szükségük van ilyen nagyteljesítményű berendezésekre.) A helyi centralizált feldolgozásra (üzemek, áruházak, bankfiókok stb.) esetleg még kisebb teljesítményű mainframeket, de inkább mini számítógépeket használnak. Ugyancsak minigépek elégítik ki a központi nagy részlegek egyedi információs igényeit.

A hierarchia legalsó szintjén vannak a munkaállomások, személyi számítógépek, terminálok. Ezekkel a gépekkel a felhasználók egyénileg, önállóan dolgoznak, esetleg kisebb, azonos feladatokat ellátó csoportok alkalmanként közösen használják őket.

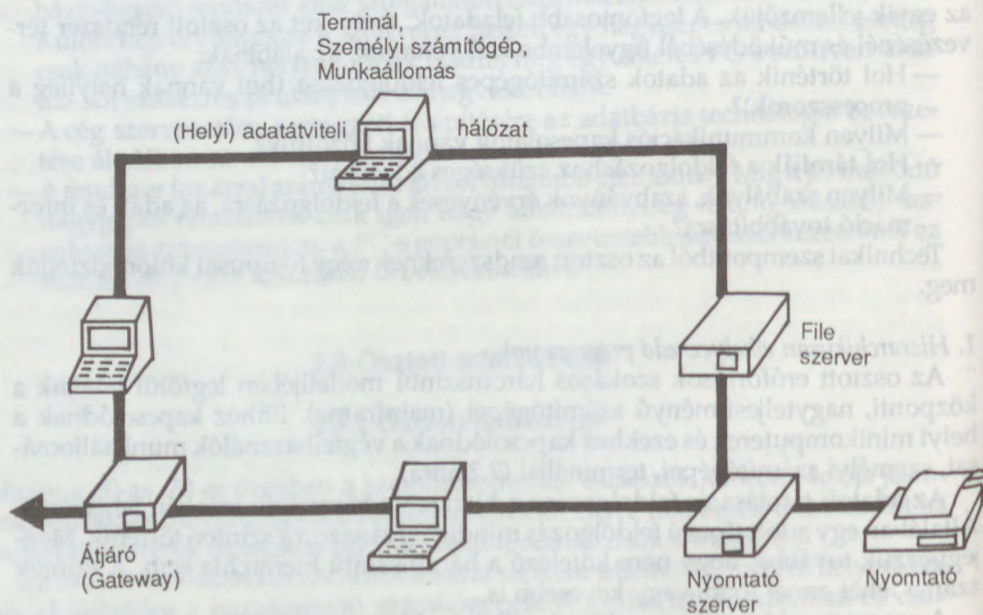
Ebben az elrendezésben a számítógépek elhelyezése tükrözi a vállalati hierarchiát.

Ez azonban nem szükségszerű. A processzor hierarchia két alsó szintjén a gépek felhasználása sokszor a szervezeten belüli funkciók szerint történik. Az egyik rendszeren a vállalat belső adminisztrációs feladatait, a másikon új termékek tervezését, míg a harmadikon például az eladással kapcsolatos feladatokat oldják meg.

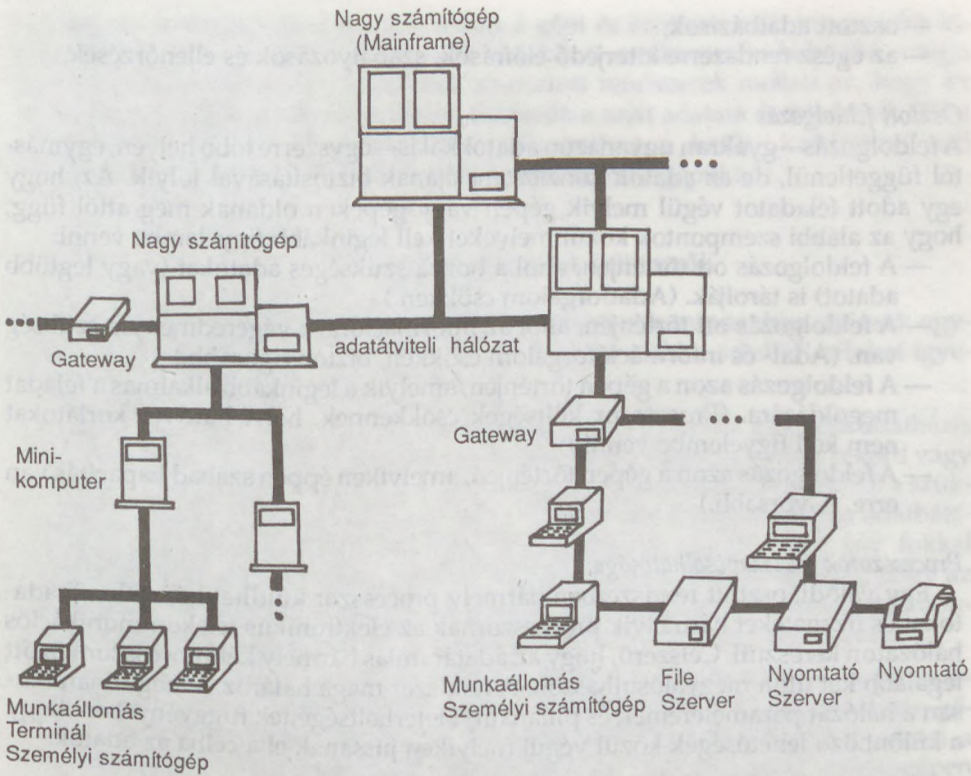
2. Együttműködő munkaállomások, személyi számítógépek hálózata

Irodai alkalmazásoknál egyre inkább ez az elrendezés lesz az uralkodó. Mind-egyik állomás, személyi számítógép egyenrangú, és szabadon tud kommunikálni a hálózat bármelyik tagjával. Bizonyos fokig kitüntetett szerepe lehet a szerver gépeknek, melyek a mindenki által közösen használt adatokat tárolják egy helyen, illetve a központi nyomtató(ka)t vezérlik.

Nagyon sokszor van a hálózatnak egy kimenő kapuja is (gateway). Ez lehetővé teszi a más hálózatokhoz való kapcsolódást. (2.3 ábra)



2.3 ábra Együttműködő munkaállomások hálózata



2.4 ábra Komplex osztott rendszer

3. Önálló egyedi gépek

Elvileg megvalósíthatunk egyedülálló, önálló gépeken is osztott feldolgozást. Az adatok konzisztenciáját manuálisan, illetve adathordozók cseréjével biztosítjuk. Ebben a formában ma már nem is tekintenénk osztott feldolgozásnak, ha nem léteznének olyan, az 1. vagy 2. csoportba tartozó rendszerek, melyek önmagukban osztottnak tekinthetők, és bár részei egy nagyobb információs rendszernek, egymással nincsen közvetlen kapcsolatuk. (Pl. egy több telephelyes vállalatnak mindegyik telephelyén van önálló, helyi hálózata, vagy munkaállomásokkal ellátott központi gépe, de ezek egymással nincsenek kapcsolatban.)

4. Az előzőek tetszés szerinti kombinációja

Ez magában foglalja a hierarchikus processzorrendszerek tetején álló processzorok, vagy különböző helyi hálózatok összekapcsolását, helyi hálózatok kapcsolódását a processzor hierarchia különböző szintjeihez stb. (2.4 ábra).

Ahhoz azonban, hogy a fenti elrendezésben működő berendezések valóban osztott rendszert alkossanak, az alábbi feltételeknek is teljesülni kell:

- osztott feldolgozás lehetősége,
- processzorok összekapcsolhatósága,

- osztott adatbázisok,
- az egész rendszerre kiterjedő előírások, szabályozások és ellenőrzések.

Osztott feldolgozás

A feldolgozás—gyakran ugyanazon adatokkal is—egyszerre több helyen, egymástól függetlenül, de az adatok konzisztenciájának biztosításával folyik. Az, hogy egy adott feladatot végül melyik gépen vagy gépeken oldanak meg attól függ, hogy az alábbi szempontok közül melyeket kell leginkább figyelembe venni:

- A feldolgozás ott történjen, ahol a hozzá szükséges adatokat (vagy legtöbb adatot) is tárolják. (Adatforgalom csökken.)
- A feldolgozás ott történjen, ahol az információra, a végeredményre szükség van. (Adat- és információforgalom csökken, biztonságosabb.)
- A feldolgozás azon a gépen történjen, amelyik a leginkább alkalmas a feladat megoldására. (Processzor költségek csökkennek, helyi hardver korlátokat nem kell figyelembe venni.)
- A feldolgozás azon a gépen történjen, amelyiken éppen szabad kapacitás van erre. (Gyorsabb.)

Processzorok összekapcsolhatósága

Egy valódi, osztott rendszerben bármely processzor küldhet közvetlenül adatokat és üzeneteket bármelyik processzornak az elektronikus telekommunikációs hálózaton keresztül. Célszerű, hogy az adatáramlás bármely két processzor között legalább két úton megvalósulhasson. A rendszer maga határozza meg dinamikusan a hálózat paramétereinek és pillanatnyi leterheltségének függvényében, hogy a különböző lehetőségek közül végül melyiken jussanak el a célba az adatok.

Osztott adatbázisok

A fogalom lényege az, hogy a felhasználó—ha jogosult rá—bármely helyről elérhet, módosíthat bármilyen, bárhol lévő adatot anélkül, hogy törődnie kellene azzal, hogy fizikailag hol és hogyan találja meg azokat.

Ez történhet úgy, hogy az adatokat duplikáljuk, és mindenhol, ahol a rendszerben használjuk, tároljuk egy másolatukat. A rendszer gondoskodik arról, hogy a másolatok mindig és mindenhol azonosak legyenek, illetve amíg a változás történik, senki ne férhessen hozzájuk.

A másik lehetőség, hogy az adatokat csak egyszer tároljuk, de mindenhol elérhető egy adatarányító, amelyik megmondja, melyik adat hol található. Ez az irányító lehet egy központi rendszeren vagy a helyi rendszeren. Lehet ezek kombinációja is, amikor csak a helyileg leggyakrabban használt és a helyileg tárolt adatok címe van bent a helyi irányítóban.

Az egész rendszerre kiterjedő előírások, szabályok és ellenőrzések

Ezek kiterjednek az alábbi területekre:

- csomópontok (processzorok, munkaállomások) közti kommunikáció előírásai,
- adatvédelmi és biztonsági ellenőrzések,
- program és adatátvitel szabályai,
- általános működési feltételek.

Osztott rendszerek létrehozását sokan a gépi és szoftver erőforrások jobb kihasználásával indokolják. Bár ez—jól kialakított rendszernél—helytálló, mégis úgy gondoljuk, sokkal fontosabb érv az osztott rendszerek mellett az, hogy az adatok felhasználói ismét magukénak érezhetik a saját adataik és feladataik fölött az ellenőrzést. Nem a távoli, centralizált központhoz, a „kaffkai kastélyhoz” kell a helyi információért fordulni, hanem a saját számítógépükhöz.

2.9.2 Osztott adatbázisok jellemzői

Bár az eddig megvalósított osztott adatbázisok sok mindenben eltérnek egymástól, mindannyiuk közös jellemzője az, hogy az alábbi „ideális” célokat igyekezzenek minél jobban megközelíteni.

Térbeli transzparencia: a felhasználóknak úgy tűnik, mintha az egész adatbázis egy helyen lenne tárolva. Nem kell azzal törődniük, hogy egy adott rekord vagy annak egyes részei hol vannak. Az adatbázis-kezelő rendszer összeszedi a szükséges adatokat, és egyben adja át őket. Lényegében a hagyományos adatbázisoknak a külső és a belső szint közti leképezése valósul meg egy fokkal magasabb szinten. Ott csak azzal nem kell a felhasználóknak törődni, hogy az adott adatbázisban hogyan és hol tárolódnak az adatok, az osztott adatbázisoknál még azzal sem, hogy helyileg melyik gépen és adatbázisban vannak.

Adatfragmentálás: logikailag összefüggő adatok, pl. táblázatok, rekordok hatékonysági vagy biztonsági szempontból fizikailag is külön helyeken, külön gépeken lehetnek. Így például az összes Budapestről induló légijárat adatai a budapesti, a Bécsből indulók a bécsi, a müncheniek az ottani számítógépen vannak stb. Ugyanakkor a közös könyvelés történhet a berlini, az utasbiztosítás nyilvántartása pedig a londoni rendszeren. Az adatok többségét helyileg használják, és itt felelnek helyességükért, biztonságukért. Az ilyen helyi megoszlás meggyorsítja a helyben megtalálható adatokkal való műveleteket és csökkenti az adatátviteli hálózat terhelését.

Az adatfragmentációnak két típusa van.

Horizontális: egy táblázatot (állományt) úgy osztanak meg, hogy az egyes rekordok egy helyen maradnak. A rekord valamelyik mezőjének értéke (pl. a járat induló helye) szabja meg, hogy a teljes rekord melyik rendszeren kerül tárolásra;

Vertikális: a rekord különböző mezői fizikailag más-más rendszeren vannak. Minden egyes rekordot fizikailag több számítógépből kell összeszedni. Nem a mező értéke, hanem típusa határozza meg a mező tárolási helyét.

További, a fragmentációval kapcsolatos jellemzők még:

Fragmentációs transzparencia: a felhasználónak nem kell törődni sem a fragmentált rekordok elhelyezésével, sem a több helyről történő összeszedésével.

Adatmásolatok kezelése: a rendszer automatikusan tudja kezelni a logikai kapcsolatok biztosítása, vagy a hatékonyság érdekében többszörösen tárolt mezők, rekordok konzisztenciáját. Pl. a törzsutasok adatait az összes helyen tárolják,

így bárhol gyorsan elérhetőek. Egy ilyen rekord módosítását is bárholonnan lehet kezdeményezni. A rendszer biztosítja, hogy a módosítás ideje alatt a rekord egyetlen példányához se férhessen más hozzá, és hogy a módosítás végrehajtása után minden helyen ugyanaz az új változat álljon rendelkezésre.

Másolat-függetlenség: ha ugyanaz az adat több példányban fordul elő a rendszerben, akkor a felhasználónak mindegy legyen, hogy melyikkel dolgozik. A rendszer feladata, hogy azt a példányt bocsájtja rendelkezésre, amelyikhez a leghamarabb hozzáférhet.

Ha az osztott adatbázis teljesíti a fent felsorolt feltételeket, akkor a felhasználó számára ugyanúgy viselkedik, mint egy centralizált adatbázis. Az osztottságból adódó problémákat a rendszer saját maga oldja meg. Kifelé ebből a logikai szinten semmi sem jelentkezik. A koncepcionális és a külső szintet ugyanúgy kell megtervezni, mint centralizált rendszernél, ugyanúgy lehet programnyelveket használni.

Természetesen a rendszer belső adminisztrációja lényegesen bonyolultabb lesz. Emiatt az osztott adatbázisoknak nem csak előnyei, hanem hátrányai is vannak, amint azt a következő pontban fogjunk tárgyalni.

2.9.3 Előnyök és problémák

Az osztott adatbázisok létrehozása az alábbi előnyökkel jár:

- a kevesebb kommunikáció miatt kisebb lesz az üzemeltetési költség,
- az adatok a felhasználó közelében maradnak,
- az adatbázis-kezelő rendszer modulárisan építhető fel,
- az egyes csomópontok „menet közben” cserélhetők a rendszer leállítása nélkül,
- egy csomópont meghibásodása csak a kizárólagosan ott tárolt adatokat teszi elérhetetlenné, a rendszer többi része működőképes marad.

Ugyanakkor nyilvánvaló, hogy számolnunk kell bizonyos hátrányokkal, veszélyekkel, s felmerülnek újabb, megoldásra váró kérdések is:

- a rendszer igen bonyolulttá válhat,
- az inhomogén rendszereknél transzlációs problémák adódhatnak,
- egyetlen csomópont sem „zárhat be” a többi előtt, még akkor sem, ha ennek a csomópontnak nincs szüksége a rendszerre. Ha pl. a csomópont egy raktár, ahol lejárt a dolgozók munkaideje, s így be lehetne zárni, az adatbázisnak (a többi felhasználónak) szüksége lehet azokra az adatokra, melyeket kizárólag ez a csomópont tárol.

A legtöbb gondot azonban a rendszer konzisztenciájának (egyértelműségének) biztosítása okozza. Egy osztott adatbázis-kezelő rendszerben aligha kerülhető el a redundancia (vagyis az, hogy bizonyos adatok több példányban, másolatban is előforduljanak). Gondoljuk meg: gyakorlatilag biztos, hogy lesznek a rendszerben olyan adatok, melyeket több csomópont is magáénak érez, és saját gépén kíván tárolni. Ez az igény indokolt lehet, hisz előfordulhat, hogy az érintett csomópontok közel azonos arányban dolgoznak a kérdéses adattal. Ha most ezt az adatot több

példányban vesszük fel a rendszerbe, azzal kímélhetjük a hálózatot, és költséget is takaríthatunk meg.

Mindezen felül bizonyos—alapvető fontosságú—adatokat azért is célszerű több példányban tárolni, hogy a rendszer valamelyik csomópontjának kiesése ne eredményezze a teljes rendszer lebénulását.

Az előnyökért azonban itt is komoly árat kell fizetni. Nyilvánvalóan megengedhetetlen ugyanis, hogy egy adat két példánya különböző értékű legyen (ebben az esetben egy kérdésre kapott válasz az adatelérés útjától függene). Annak biztosítása pedig, hogy a módosítások még rövid időre se okozzanak inkonzisztenciát, meglehetősen komoly erőfeszítéseket igényel.

Hasonlóan nehéz feladat az adatok gyors megkeresése is. Hiába igyekszünk a felhasználandó adatokat a „helyszínen” tartani, minden adatot minden csomópontban nem tárolhatunk. (Ez a módszer nemcsak a tárolók kihasználása szempontjából lenne katasztrofális. Az esetleges módosítások olyan adatátviteli igényt jelentenének, ami értelmetlenné tenné az ilyen „osztott” szervezést.) Így, ha ritkábban is, szükség lehet a más csomópontokon tárolt adatok használatára. Ha csak kiolvasásra van szükség, akkor is meg kell keresni az adat egy példányát (lehetőleg a legkönnyebben elérhetőt). Ha pedig az adatot módosítani kell, információra van szükség az összes másolat hollétéről. (Ez akkor is nélkülözhetetlen, ha egy példány éppen ott található.) Az összes csomópont végigvizsgálása túl nehézkes és költséges megoldás lenne. Ennek elkerülése érdekében valamiféle tartalomjegyzéket célszerű készíteni arról, hogy az egyes adatok hol találhatók. Ezzel azonban összes gondjaink újra kezdődnek, hiszen a tartalomjegyzék (melyet útmutatónak is szokás nevezni) ugyancsak kiolvasandó és időnként módosítandó adatokat tartalmaz. Ha ezeket egy központi helyen tartjuk, akkor minden tranzakció előtt fel kell keresnünk ezt a központot, növelve a kommunikáció mennyiségét. Vagyis ebben az esetben a központ felkeresésének sűrűségét illetően semmit sem csökkentettünk a kommunikáción a központosított szervezéshez képest. Az átvitt információ mennyisége esetleg kevesebb lehet, de kérdés, hogy ezért érdemes-e vállalni az osztott adatbázis működésével járó bonyodalmakat.

A másik végtet az, hogy ezt az útmutatót minden csomópontban elhelyezzük. Elégge statikus adatbázisoknál (tehát ahol az adatokat inkább csak kiolvasni kell, s a változtatás nagyon ritka) ez a módszer megfelelő lehet. Ha azonban az adatokat gyakran kell változtatnunk (s itt a változatlan tartalom melletti helyváltoztatásra is gondolnunk kell), ez az út sem járható, hisz minden alkalommal minden csomópontot fel kell keresnünk az útmutató aktualizálásához. A megoldás az útmutatók megosztása. A *globális* (hálózati) útmutató—melyre minden csomópontnak szüksége van—az elvi adatmodellt írja le, kiegészítve azt az egyed típusok tárolási helyének megadásával (voltaképpen ezzel biztosíthatjuk az adatbázis megkívánt logikai egységét). A *helyi* útmutatók egy-egy csomópont adatait írják le részletesen. Erre a kérdésre az osztott adatbázisok katalógusának tárgyalásakor (l. 7.10.7 pont) még visszatérünk.

Mivel az osztott adatbázisok használóinak számára is fontos az adatok titkosága, itt is elvárható a rendszertől a különféle jogosultsági szintek megkülönböztetése, és annak biztosítása, hogy az egyes felhasználók a számukra engedélyezett határokat ne léphessék át. Kérdés, hogy az ehhez szükséges adatokat tartalmazó

táblázatok hol helyezkedjenek el, és a megfelelő ellenőrzések hol történjenek. Meglehetősen elterjedt az a megoldás, hogy a jogosultsági viszonyokat is a globális útmutató részeként tárolják.

2.9.4 Az adatok elhelyezése

Az adatok elhelyezésének kérdését korábban érintettük már. A téma jelentősége azonban megköveteli a részletesebb tárgyalást. Az osztott adatbázisok hatékonysága tudniillik nagymértékben múlik azon, hogy az adatok elosztása mennyire illeszkedik a feldolgozáshoz.

Az adatok elosztása lehet *statikus*, amikor az adatbázis-felügyelő jelöli ki a másolatok példányszámát és helyét. (Jóllehet a statikus jelző itt is megkérdőjelezhető, hiszen ezt a kijelölést időről-időre felül lehet és felül is kell vizsgálni.) *Dinamikus* eloszlásnál a rendszer dönti el—megfelelő statisztikák kiértékelése alapján—, hogy mikor szükséges egy új másolatot készíteni, vagy éppen mikor célszerű egyet megszüntetni (ügyelve természetesen arra, hogy az utolsó előfordulást semmiféle statisztikai adat eredményeként se törölje ki).

Az adatok elhelyezéséhez elsősorban az alábbi szempontokat célszerű figyelembe venni:

- a) Hol használják legtöbbet az érintett adatot? A vizsgálatnak természetesen a használat módjára is ki kell terjednie, legfőképpen arra, hogy az illető csomópontban a kérdéses adatot olvassák vagy változtatják-e (létrehozzák), illetve hogy mindkét használati forma előfordul-e. Meg kell vizsgálni azt is, hogy az adat összes felhasználását tekintve, egy-egy módosításra hány olvasás jut. [Gondoljunk pl. egy olyan konstrukcióra, amelyben egy raktárból A-ból és B-ből is bonyolítanak eladásokat, a raktárkészlet kiértékelését azonban csak C-ből végzik. Első látásra úgy tűnik, hogy mivel kiértékelésre az adatoknak végül C-be kell jutniuk, ott és csak ott célszerű azokat tárolni. Ha azonban elég sok eladás (adatmódosítás) esik egy egy leltárkészítésre (olvasásra), valószínűleg hasznosabb az adatokat A-ban és/ vagy B-ben elhelyezni.]
- b) Az érintett adat mennyire nélkülözhetetlen az egész adatbázis használatához? Már korábban utaltunk arra, hogy az olyan adatokat, melyek nélkül az adatbázis használhatatlanná válik, célszerű több példányban tárolni. Lehetséges—és szokás is—az adatokat osztályozni a „nélkülözhetetlenség foka” szerint, és az egyes osztályokhoz egy minimális másolatszámot rendelni.
- c) A tárolás, elérés költségei. Különösen inhomogén rendszereknél fordulhat elő gyakran, hogy egy adat tárolása, elérése az egyik csomópontban más költségekkel jár, mint a másokban, de ezt eredményezheti pl. egy csomópont eszközeinek túlterheltsége is.

A felsorolt szempontok mindegyikéről elmondható, hogy menetközben változnak azok a feltételek, melyeket megvizsgáltunk. Mindig szem előtt kell tartanunk azt a tényt, hogy egy adatbázis tervezése olyan tevékenység, melynek vége csak akkor van, ha az adatbázist senki sem használja többé.

2.9.5 A konzisztencia biztosítása

Utaltunk már arra, hogy milyen fontos és nehéz feladat az adatbázis *konzisztenciájának* biztosítása. A konzisztenciának két szintje különböztethető meg:

- a) *erős* konzisztencia, ha egy adat másolatain minden változtatás egyidőben megy végbe,
- b) *gyenge* konzisztencia esetén egy hosszabb-rövidebb időintervallum alatt a másolatok különbözhetnek, azaz a konzisztencia csak bizonyos idő elteltével áll helyre.

Nyilvánvaló, hogy vagy erős konzisztenciára van szükség, vagy valamilyen módon biztosítani kell, hogy—gyenge konzisztencia esetén—a kritikus időintervallumban a még nem módosított adatok más feldolgozások számára elérhetetlenek legyenek.

Mindezt az ún. szinkronizációs protokollok (szabály, illetve eljárásgyűjtemények) segítségével érhetjük el. A szinkronizációs protokollok lehetnek központosítottak vagy osztottak aszerint, hogy a követendő eljárás igényel-e egy kitüntetett szerepű központot, vagy sem. (Ez a központ nem szükségképpen mindig ugyanaz a csomópont, hanem időről időre, vagy tranzakcióról tranzakcióra változhat.)

Könnyen belátható, hogy a szinkronizációs protokollok egyes adatokat hosszabb-rövidebb időre kénytelenek elzárni a nyilvános használat elől, vagy más szóval ezek bizonyos műveletek esetén kizárólagosan (és nem osztottan) használható erőforrásokká válnak.

Az ebből adódó problémákat a centralizált adatbázisoknál sem sikerült teljesen jól megoldani. Ezek közül a legnagyobb veszélyt a patthelyzet jelenti. Ez legegyszerűbb esetben akkor alakul ki, amikor két folyamat ugyanazt a két adatot vagy adatsortot akarja kizárólagos módon használni, és ebből az egyik elzárta az egyiket, míg a másik a másikat. A továbblépéshez mindkettőnek szüksége van a másik által lefoglalt adatokra, de ezek nem szabadulnak föl, mert egyik folyamat sem tud továbblépni.

Sajnos, a patthelyzetek megelőzésére, elkerülésére—a központosított adatbázisoknál leírtakhoz hasonlóan—itt sem rendelkezünk hatékony módszerekkel. Általában meg kell elégednünk azzal, hogy a patthelyzetet felismerjük, és feloldjuk azáltal, hogy az abban résztvevő tranzakciókat egy kivételével leállítjuk, és a leállítottak eddigi hatását is visszamenőleg megszüntetjük (rollback).

Tovább növeli a gondokat, hogy az integrált rendszerekben az elzárások viszonylag rövid ideig tartanak, általában addig, amíg a tárban a módosítások és a szóbanforgó lemezen az író/olvasó (I/O) műveletek lezajlanak.

Osztott rendszerben a lezárási kérelmeknek és a felszabadítási engedélyeknek keresztül kell menni a hálózaton, ami már önmagában lényegesen meghosszabbítja az elzárási időket. Ezt tovább növeli, ha a módosított rekordokból másolatokat is tárolunk a rendszerben. Ilyenkor a zárat csak akkor oldja fel az adatbázis kezelő rendszer, ha az összes másolat módosítása sikeresen befejeződött.

A 2.6 alfejezetben már láttuk, milyen fontos szerepe van az adatbázis-felügyelőnek. Azt is megjegyeztük, hogy ezt a munkát sokszor nem egy személy, hanem egy csoport látja el. Osztott adatbázisnál a felügyelő szerepe még sokkal bonyolultabb. Ezért—és a földrajzi megosztottság miatt—gyakorlatilag elkerülhetetlen több adatbázis-felügyelő alkalmazása.

Megvizsgálva a helyzetet, azonnal nyilvánvaló, hogy az egyes csomópontokban szükség van helyi adatbázis-felügyelőkre. Ők a helyi adatok elhelyezésével, a helyi adatbázisrész hangolásával stb. foglalkoznak. Ezen kívül kapcsolatot tartanak a helyi felhasználókkal. Ezeket a feladatokat nagyobb távolságról aligha lehet sikeresen megoldani.

Az adatbázis logikai egysége ugyanakkor megköveteli, hogy a sémák, alsémák megadását, módosítását, a védelmi kérdések intézését központilag végezzék, és központilag kell foglalkozni az olyan kérdésekkel is, mint pl. a másodpéldányok aktualizálása.

Általában háromszintű az adatbázis felügyelet. A legfelsőn van az adatbázis-felügyelet vezetője, aki a többi felügyelő munkáját a központból áttekinti, összehangolja. A második szinten—még mindig központilag—három témát különböztetünk meg, s ezekhez rendelünk külön-külön felügyelőt. Egy koordináló, egy kommunikációs és egy védelmi felügyelőt célszerű kijelölni. Végezetül a harmadik szintre tesszük a csomópontok helyi felügyelőit.

Ha konkrét esetekben az adott feladattól és környezettől függően esetleg célszerű is lehet másképpen eljárni, kiindulási alapként indokolt ezt a sémát elfogadni.

ADATMODELLEK

3.1 Az adatmodellek osztályozása

Az adatbázis-kezelő rendszerek mindig valamilyen adatmodellre épülnek. Az adatmodellek lehetnek a különböző típusú adatokat, a közöttük fennálló kapcsolatokait, összefüggéseit és a ezek kapcsolatainak adatbázisbeli elhelyezését. Az adatmodell elkészítése után követelmény az adatbázisnak adatbázisra való feltöltése, és rendszeres használatban tartása. Míg az adatmodellről nem történik az egyes adatok konkrét értékeivel, a használatnál Ötven, amikor a modell bővíthető, illetve módosítható mindig a használati értékekre vagyunk kíváncsiak, azokkal dolgozunk, azaz az adatok csak a kívánt információkat.

Ezt így mondjuk, hogy az adatmodellben az egyedi típusokkal (a konkrét típusokkal) más néven értéktípusokkal foglalkozunk, míg a felolgozásban a rekord-típusok felismerésével foglalkozunk.

Az adatmodellek három függőlen szintje van:

- **Először az, mely az adatok fizikai tárolását és elérését módját adja meg.** Ezt a 4. fejezetben ismertetjük.
- **A koncepcióális szint vagy a séma, mely az adatbázisban lévő összes adat szerkezetét és a közöttük fennálló összes kapcsolatot, az adatokhoz való hozzáférést foglalkoztatja és a hozzáférési jogosultságokat írja le.**
- **A lógikus szint, vagy absztrakció, melyek azt írják le, hogy az egyes felhasználók milyen látják a sémát.**

A könyv második részében adatmodellben szereplő kapcsolati rendszert értjük, amely a koncepcióális és a lógikus szinten írja le az adatbázist.

Ezeknek három fő típusát különböztetjük meg:

- hierarchikus
- hálós
- relációs modell

A gyakorlatban a lógikus és a koncepcióális szintet írja azok egymással gyökérrel függősenek) ugyanazt a modellel írják le.

Az első adatbázis-kezelő rendszerek a **hierarchikus modellel** alapultak. Ezt az 5. fejezetben ismertetjük.

Megvizsgálva a helyzetet, azonnal nyilvánvaló, hogy az egyes csomópontokban szükség van helyi adatbázis-felügyelőkre. Ők a helyi adatok elhelyezésével, a helyi adatbázisról hangoztatással stb. foglalkoznak. Ezen kívül kapcsolatot tartanak a helyi felhasználókkal. Ezeket a feladatokat nagyobb tövisságról aligha lehet sikeresen megoldani.

Az adatbázis logikai csomópontjait a következőképpen lehet felépíteni, hogy a sémák, absztrakciók megadását, módosítását, a csomópontok közötti adatok központi elhelyezését végzik; és központi módon kell foglalkozni az olyan kérdésekkel is, mint pl. a másodpéldányok aktualizálása.

Általában háromszintű az adatbázis-felügyelet. A legfelső van az adatbázis-felügyelet vezetője, aki a többi felügyelő munkáját a központból áttekintheti, összehangolja. A második szinten — még mindig központi módon — három témát különböztetünk meg, a ezekhez rendelünk külön-külön felügyelőt. Egy koordináló, egy kommunikációs és egy vezérlő felügyelővel rendelünk ki főmunkát. Végül a harmadik szintre tesszük a csomópontok helyi felügyelőit.

Ha konkrét esetekben az adott feladatok és a viszonyoktól függően esetleg célszerű is lehet másképpen eljárni, kiegészítésként alapvető indokolt ezt a sémát elfogadni.

Általános leírás

3.1 Az adatmodellek osztályozása

Az adatbázis-kezelő rendszerek mindig valamilyen adatmodellre épülnek. Az adatmodellben leírjuk a különböző típusú adatokat, a köztük fennálló kapcsolatokat, összefüggéseket és a velük kapcsolatos adatvédelmi eljárásokat. Az adatmodell elkészítése után következik az adatbázisnak adatokkal való feltöltése, és rendszeres használatbavétele. Míg az adatmodellnél nem törődünk az egyes adatok konkrét értékével, a használatnál (kivéve, amikor a modellt bővítjük, illetve módosítjuk) mindig a konkrét értékekre vagyunk kíváncsiak, azokkal dolgozunk, azokból állítjuk össze a kívánt információkat.

Ezt úgy mondjuk, hogy az adatmodellben az egyed típusokkal (a rekord típusokkal) más néven entitástípusokkal foglalkozunk, míg a feldolgozásban a rekord típusok konkrét előfordulásaival.

Az adatmodellnek három független szintje van:

- *Belső szint*, mely az adatok fizikai tárolását és elérési módját adja meg. Ezt a 4. fejezetben ismertetjük.
- A *koncepcionális szint* vagy a *séma*, mely az adatbázisban levő összes adat szerkezetét és a köztük fennálló összes kapcsolatot, az adatokhoz való hozzáférés logikáját és a hozzáférési jogosultságokat írja le.
- A *külső szint*, vagy *alsémák*, melyek azt írják le, hogy az egyes felhasználók miként látják a sémát.

A könyv második részében adatmodellel azt a kapcsolati rendszert értjük, amelyik a koncepcionális és a külső szinten írja le az adatbázist.

Ezeknek három fő típusát különböztetjük meg:

- hierarchikus,
- hálós,
- relációs modell.

A gyakorlatban a külső és a koncepcionális szintet (bár azok egymástól gyakorlatilag függetlenek) ugyanazzal a modellel írjuk le.

Az első adatbázis-kezelő rendszerek a *hierarchikus modellen* alapultak. Ezt az 5. fejezetben tárgyaljuk.

A hetvenes évektől egyre jobban elterjedtek a hierarchikusnál lényegesen bonyolultabb adatszerkezetek kezelésére alkalmas *hálós adatmodelleken* alapuló rendszerek. Ezek legnagyobb előnyét ma már inkább hátrányuknak tekintjük. A felhasználónak ugyanis meg kellett adni, milyen úton akar az adatokhoz hozzáférni. Az előny abból adódik, hogy a konkrét adateloszlások ismeretében egy „okos”, az adatokat és kapcsolatokat jól ismerő felhasználó sokkal hatékonyabban tud így dolgozni az adatbázissal. A hátrány is ennek a következménye. Az adatbázis technológia elterjedésével egyre többen akarják ezeket a rendszereket használni, de viszonylag kevesen rendelkeznek olyan mélységű ismeretekkel, hogy a hatékony elérési utakat maguk tudják meghatározni. Ezért az ilyen (potenciális) alkalmazók jelentős részét vagy elriasztotta a felhasználás bonyolultsága, vagy pedig nem tudták kellőképpen kihasználni az adatmodellben rejlő lehetőségeket.

Ezen hátrányok ellenére a hetvenes évek közepétől a nyolcvanas évek elejéig ez az adatmodell volt a legnépszerűbb, és igen sok installációnál még ma is ezt, vagy ezt is használják. A hálós adatmodellt a 6. fejezetben tárgyaljuk.

Matematikai szempontból a legmegalapozottabb és a legjobban kidolgozott a *relációs modell*. Ez biztosítja egyúttal a legnagyobb mérvű adatfüggetlenséget is. További előnye, hogy a mindennapi szemlélethez, az átlagos felhasználó gondolkodásához ez áll a legközelebb, mivel az adatokat kétdimenziós táblázatokban tárolja és szolgáltatja. Alapjait a 70-es években Codd dolgozta ki, de akkor még a gépi megvalósítás nem volt elég hatékony. Ezért egészen a nyolcvanas évek elejéig a modell inkább csak az elméleti szakemberek fantáziáját foglalkoztatta, a gyakorlatban nagy adatbázisokon nemigen alkalmazták.

A személyi számítógépek rohamos elterjedésével kiszélesedett az alkalmazók tábora, és ugyanakkor a kezelendő adatmennyiség mérete legtöbbször már nem volt olyan nagy. Így a relációs modell felhasználó-közelí ábrázolásmódja nagyobb előnyöket jelentett, mint a kevésbé hatékony gépi realizálás hátrányai. A nyolcvanas évek második felétől megjelenő új adatbázis-kezelő rendszerek—a nagyobb számítástechnikai teljesítmény rendelkezésre állása miatt is—már szinte kivétel nélkül a relációs modellre épülnek, illetve sok hálós rendszernek megjelent a relációs változata is (pl. IDMS). A relációs modellt a 7. fejezetben tárgyaljuk.

Mint azt már említettük, megkülönböztetünk tényadat és dokumentum formátumú adatbázisokat. Az eddig felsorolt adatmodellek mind a tényadat formátumú adatokat és kapcsolataikat írják le. A dokumentumokat kezelő szöveges információ visszakereső rendszerek legfontosabb tulajdonságait a 8. fejezetben foglaljuk össze.

3.2 Főbb modellkészítési szempontok

Az adatmodellnek mindig a valós világot, pontosabban annak a teljes feladat megoldásához számunkra szükséges részét kell ábrázolnia.

Ugyanazt a rendszert, ugyanazt az állapotot azonban többféle módon írhatjuk le. Szemléletünkötől, megszokásunktól, és a későbbi felhasználás egyszerűségétől, hatékonyságától függ, hogy a különböző ábrázolási módok közül melyiket választjuk.

Az „ez a borospohár félig tele van”, és az „ez a pohár félig üres”, ugyanazt a valóságtartalmat fejezi ki, és csak optimista vagy pesszimista szemléletüinktől függ, mit mondunk a pohár láttán. Ha azonban később olyan kérdésre kell válaszolnunk, „mennyit ihatunk még a pohárból?” vagy „mennyit ittunk már ki, illetve mennyi hiányzik belőle?”, és még az sem biztos, mekkora pontosan a pohár, és hogy mindig pont a felénél lesz a választóvonal, akkor célszerűbb lesz azt mondanunk: „ennyi bor van benne”, vagy „ennyi bor hiányzik belőle”.

Mivel a valós világ legkisebb részlete is igen bonyolult, a modellben csak valamilyen mélységig vizsgálhatjuk azt. Az egyszerűsítést úgy kell megvalósítanunk, hogy a modell áttekinthető, számítástechnikailag kezelhető maradjon, de ugyanakkor a számunkra fontos összes adatot, teendőt, eseményt pontosan tükrözze. Egyszerű példa erre egy vállalat készpénz nyilvántartása, ahol fölösleges azt a modellbe bevenni, hogy hány db 5 000, 1 000, 500 stb. forintost fizettek ki vagy kaptak. Ugyanezen vállalatnál azonban a részlegekben a fizetéskor igenis fontos a címletezés, hogy mindenkinek a fizetési borítékjába tudják tenni a pontos összeget. Az, hogy a teljes modell meddig egyszerűsíthető, illetve egyes részeiben meddig lehet elmenni az egyszerűsítésben, igen nagy mértékben függ attól, hogy mire akarjuk használni.

Az egyszerűség és az áttekinthetőség mellett figyelembe kell vennünk azt is, hogy az igények változnak. Az adatmodellnek olyannak kell lenni, hogy a felmerülő új igényeket is lehetőleg ki lehessen a segítségével elégíteni, illetve, ha szükséges, akkor a modellt úgy lehessen könnyen bővíteni, hogy az addigi feladatok megoldása a bővítés után is lehetőleg ugyanúgy történjen, mint korábban.

Elvileg helytelenítendő, de gyakorlatban mindig kénytelenek vagyunk figyelembe venni azt a tényt, hogy az adatmodell komplexitását, az adatok mennyiségét és ezzel együtt a részletek egyedi tárolását mindenképpen korlátozza a rendelkezésünkre álló hardver és szoftver választék. Van, amikor ez tényleges fizikai korlátként jelenik meg: ez van, csak ezen kereteken belül lehet dolgoznunk.

Más esetekben ez inkább pénzügyi korlátként jelentkezik. Meg van szabva, mekkora költségből kell az információs rendszert létrehozunk és ebből mennyit költhetünk a megvalósításhoz szükséges szoftverre és hardverre.

A fentiekén kívül még a rendszer válaszadási ideje is beléphet korlátozó tényezőként. A legtöbb információs rendszer részben vagy teljesen on-line módon működik. Ezért meghatározott időn belül választ kell adnia, információt kell szolgáltatnia. Túl sok összefüggést tartalmazó, a részleteket is túlságosan figyelembevevő modellnél ez a válaszidő olyan nagy lehet, hogy a gyakorlatban már semmit sem ér a kapott, tökéletes pontosságú, de már elavult információ. Egy városi, automatikus forgalomirányítási rendszer, amelyik a jelzőlámpák váltásánál sok minden mellett még figyelembe veszi, hogy melyik irányból jövő gépkocsik szennyezik több ólommal a levegőt, és még azt is, hogy hány ember ül összesen ezekben a kocsikban, de a teljes kiértékeléssel csak két perc alatt végez, a gyakorlatban használhatatlan. Ennél sokkal jobb az, amelyik egyszerűen aszerint váltja a lámpákat, hogy melyik irányba akar több kocsni továbbhaladni (kiegészítve azzal a feltétellel, hogy meghatározott idő után, rövid időre mindenképpen szabad utat kap egy állandóan kisebbségben lévő irány, pl. egy főútvonalra betorkolló mellékutca forgalma is).

Ezen általános elvi szempontok mellett a tényleges megvalósításnál figyelembe kell venni a konkrét felhasználási igényeket és szokásokat is.

Ezeket bővebben a 4.1. pontban soroljuk fel.

3.3 Kapcsolattípusok

Minden mindennel összefügg—szokták mondani elkeseredetten egy új adatbázis tervezői, miután fölmérték a jogos és illuzorikus elvárásokat, mire legyen jó, mi mindent tudjon majd a vállalat gondjaira gyógyírt hozó, csodálatos új adatbank. Természetesen, ha a fenti elvet az adatbázisban is meg akarnák valósítani, akkor feladatuk eléggé reménytelen lenne. Az előző pontban tárgyalt elvek figyelembevételével megpróbálnak rendet teremteni a kapcsolatok dzsungelében, és kiemelni ezek közül azokat, amelyek a feladat megoldásához nélkülözhetetlenek. Ezek után igyekeznek egy olyan adatbázis-kezelő rendszert és hardvert beszerezni, illetve biztosítani, mely a végső rostán fennmaradt adat- és kapcsolattömeget még a megkövetelt feltételek mellett (beszerzési- és működési költség, tervezési- és elkészítési idő, működési sebesség, további előre látható és előre nem látható igényekre bővítési lehetőség stb.) kezelni tudja. Ha ez a feladat a megadott peremfeltételek mellett nem teljesíthető, akkor a minden szakkönyv által elítélt, de a gyakorlatban mégis sokszor alkalmazott megoldáshoz folyamodnak. Az adatmodellt egyszerűsítik úgy le, hogy számítástechnikai rendszerükkel meg lehessen valósítani.

A valóság valahol a két véglet között van. A modellben bizonyos engedményeket kell tenni (ezért utópia a tökéletes, mindent átfogó integrált rendszer, a Leibnitz-féle világszellem számítástechnikai reinkarnációja), de az új adatbázis létrehozásához és használatához kibővítik vagy hatékonyabbra cserélik a meglévő számítástechnikai erőforrásokat.

Míg a második lehetőségre az adatbázis-felügyelő csak ésszerű javaslatokat tehet, a kompromisszum első feléhez tevékenyen hozzájárulhat. Ebben nyújt segítséget a kapcsolattípusok elemzése és annak ismerete, hogy a különböző adatbázis-kezelő rendszerek milyen típusú kapcsolatokat milyen hatékonysággal tudnak kezelni.

A kapcsolatokat megkülönböztethetjük egyrészt aszerint, hogy az adatbázis két önálló eleme (entitása, rekordtípusa) vagy ugyanazon rekord két mezője között állnak fenn. Ezek kezelése, egyik típusnak a másikba való átalakítása adatmodell-függő. Erre részletes példát mutatunk majd be a 7.9 pontban, amikor a normalizálást tárgyaljuk a relációs modellben.

Az adatmodellek szempontjából azonban sokkal fontosabb az adatbázisban lévő elemek, objektumok közti kapcsolatok alábbi csoportosítása:

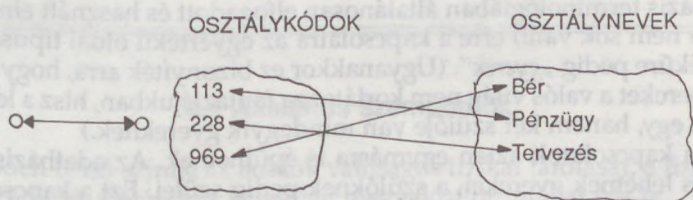
1. *A két elem független egymástól.* Ez a legkedvezőbb, mert akkor az adatbázisban sem kell kapcsolatot teremteni közöttük. Az ilyen kapcsolatokat minden adatbázis-kezelő rendszerben korlátozás nélkül meg tudjuk valósítani. Függetlennek tekinthető entitások lehetnek például egy integrált adatbázisban az általános iskolai hallgatók osztályzatainak és az OTP átutalási betétszámlán lévő összegeknek a rekordjai. (Bár lehet, hogy nagy tudású matematikusok még itt is találnának korrelációt. Jó osztályzatok esetén több jutalmat kapnak a ne-

bulók, megnő a kivét a számlákon. Ez azonban már olyan kapcsolat, amit az adatmodellben biztos nem fogunk tekintetbe venni.)

Ugyanígy függetlennek tekinthetők egymástól a születési dátum év, hó és nap mezői, bár talán erre is lehetne ellenpéldát találni New Yorkban, a nagy áramkimaradást követő kilencedik hónapban.

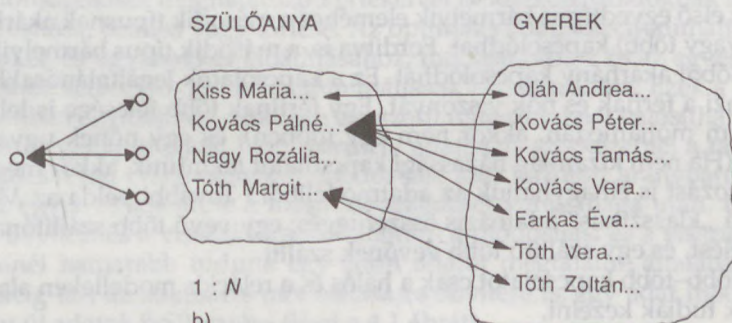
2. A két elem között *kölcsönösen egyértelmű* kapcsolat áll fenn. Ezt az összefüggést 1:1 kapcsolatnak is nevezzük.

Ilyen lehet például magyar állampolgároknál a régi, ma már alkotmányellenes személyi szám, és a majdan bevezetésre kerülő személyi azonosító jel (mely-



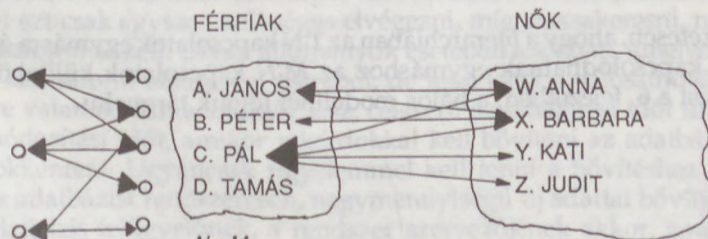
1 : 1

a)



1 : N

b)



N : M

c)

3.1 ábra Kapcsolattípusok. a) 1:1; b) 1:N; c) M:N

nek például első karaktere a feministák nyomására a férfiaknál 0, a nőknél 1, melyet követ a születési időpont, és kiegészít egy négyjegyű szám, amely lehetne a régi személyi szám utolsó négy jegye—esetleg más sorrendben) közti kapcsolat. Ugyancsak kölcsönösen egyértelmű kapcsolat áll fent egy konzisztens, jó vállalati kódrendszerben az osztályok kódja és megnevezése között.

Az 1:1 kapcsolat szintén kezelhető minden adatmodellben.

3. *Egyik irányba egyértelmű, a másikba többértelmű, 1:N kapcsolat.*

A második elemtípus értéke egyértelműen meghatározza az elsőt, de ez fordítva nem igaz. Tipikus példa erre a szülőanya és a gyermekei kapcsolata. Egy anyának több (szigorúan az adatmodellt tekintve akárhány) gyereke lehet, de bármelyik gyereknek csak egy, sőt pontosan egy szülőanyja van. Ez a példa az adatbázissal foglalkozó szakemberek fantáziáját is annyira megragadta, hogy az adatbázis terminológiában általánosan elfogadott és használt elnevezés (pedig ilyen nem sok van!) erre a kapcsolatra az egyértékű oldal típusára „szülő”, a sokértékűre pedig „gyerek”. (Ugyanakkor ez bizonyíték arra, hogy az elméleti szakembereket a valós világ nem korlátozza fantáziájukban, hisz a legtöbb esetben nem egy, hanem két szülője van mindegyik gyereknek.)

Ezek a kapcsolatok aztán egymásra is épülhetnek. Az adatbázisbeli gyerekeknek is lehetnek gyerekei, a szülőknek pedig szülei. Ezt a kapcsolatsort nevezzük hierarchiának vagy egyszerű hierarchiának.

Az 1:N kapcsolatokat minden adatmodellben ábrázolhatjuk.

4. *Több-több, M:N kapcsolat*

Az első egyed típus bármelyik eleméhez a második típusnak akárhány eleme (0; 1 vagy több) kapcsolódhat. Fordítva is, a második típus bármelyik eleméhez az elsőből akárhány kapcsolódhat. Ez a kapcsolatok legáltalánosabb típusa, ez jellemzi a férfiak és nők viszonyát. Egy férfinak több felesége is lehet (persze, ha nem mohamedán, akkor nem egy időben), és egy nőnek ugyanúgy több férje. (Ha nem kizárólag házassági kapcsolatot tekintünk, akkor még az időbeli korlátozást is elhagyhatjuk az adatmodellből.) További példa az M:N kapcsolatra a „klasszikus” adatbázis összefüggés: egy vevő több szállítónak ad megrendelést, és egy szállító több vevőnek szállít.

A több-több kapcsolatot csak a hálós és a relációs modelleken alapuló rendszerek tudják kezelni.

Az egyes kapcsolattípusokat a 3.1 ábrán láthatjuk.

Természetesen, ahogy a hierarchiában az 1:N kapcsolatok egymásra épülhetnek, ugyanúgy kapcsolódhatnak egymáshoz az M:N kapcsolatok különböző elemei. Ezt a kérdést a 6. fejezetben, a hálós modellnél fogjuk tárgyalni.

Fizikai tárolási és elérési mód

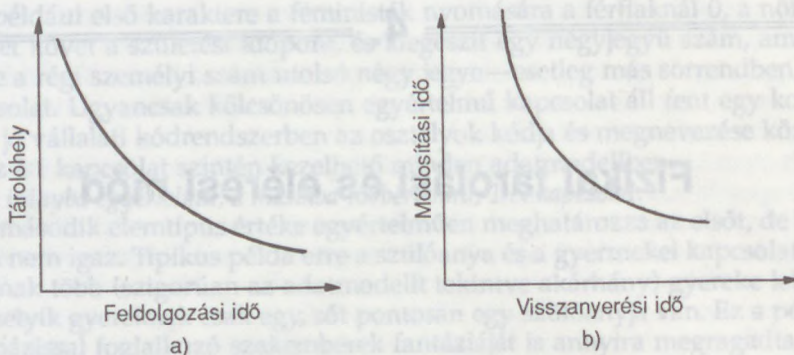
4.1 Általános szempontok

Az adatmodell belső szintje az adatok valóságos fizikai tárolását, a fizikai szerkezetet, és az adatok eléréséhez fizikailag megvalósított utakat írja le.

A belső szint kialakításakor két fő feladatot kell megoldanunk. Az első az adatbázis megfelelő szerkezetének a kialakítása. A második a szerkezet ismeretében adott tulajdonságoknak meghatározott értékeivel rendelkező adatoknak a megtalálása, módosítása, beírása vagy törlése. Az optimális adatbázis minimális tárolóhelyet foglal el, és ugyanakkor előállításához, módosításához és az egyes adatok megtalálásához szükséges időnek is minimálisnak kell maradnia. Ezek a követelmények azonban egyidejűleg csak igen egyszerű rendszereknél valósíthatók meg. Általánosan megállapíthatjuk, hogy bármely adatbázis szerkezetnél a tárolóhely és a feldolgozási idő (visszakeresési+módosítási idő) fordított arányban áll egymással.¹ Az egyik csökkentése a másik növekedését vonja maga után. Ugyanez érvényes a bővítési és a visszakeresési, valamint a módosítási és a visszakeresési időre is; minél hamarabb tudunk egy adott adatot megtalálni, általában annál hosszabb ideig tart az adatbázis módosítása és bővítése is, egy adat módosítása, törlése vagy új adatok beillesztése (lásd a 4.1 ábrát).

Az adatbázis előállításához szükséges időt rendszerint nem szokták minimalizálni, mivel ezt csak egyszer szükséges elvégezni, míg visszakeresni, módosítani igen sok alkalommal kell. Olyan állományok esetében azonban, amelyekben igen sok adatot kell tárolni biztonsági okokból, de igen kicsi a valószínűsége annak, hogy ezekre valamikor újra szükség lesz, célszerű az előállítási időt (illetve azt a speciális módosítási időt, amikor rekordokkal kell bővíteni az adatbázist) minimálisra csökkenteni. Ugyancsak figyelemmel kell lenni a bővítéshez szükséges időre, ha az adatbázist rendszeresen, nagymennyiségű új adattal bővítjük. Éppen ezért az adatbázis-felügyelőnek, a rendszer szervezőjének akkor, amikor végül

¹ A módosítást itt általánosabb értelemben használjuk, beleértjük a meglévő rekordok módosítását, törlését, új rekordok beírását is.



4.1 ábra a) az adatbázis által elfoglalt tárolóhely és feldolgozási idő (módosítási és/vagy visszanyerési idő), b) a módosítási és visszanyerési idő közti kapcsolat jellege

valamilyen adatszerkezet mellett dönt, tisztában kell lenni a következőkkel. Rendszerének hatékonyságát döntő módon befolyásolja, hogy

- milyen információkra van leggyakrabban szüksége a felhasználónak,
- milyen hardvert használ az adatok tárolására,
- mekkora adatmennyiséget kell tárolnia,
- az adatokat, vagy azok egy részét milyen gyakran kell előre meghatározható sorrend alapján elérnie,
- mindig ugyanazon kulcsmezők alapján kell-e rekordokat visszakeresni,
- ha az adatokat nem előre meghatározható sorrend szerint kell visszakeresni, akkor átlagosan hány adatot kell megtalálni egy feldolgozási menetben,
- az adatokat csak olvasni kell, vagy meg is kell változtatni az egyes mezők tartalmát,
- meglévő rekordok hossza megnövekedhet-e,
- hány új adatot kell hozzáadni az adatbázishoz, hány régit kell törölni, és ez milyen gyakran fordul elő.

E szempontok figyelembevételével az adatbázis-felügyelő eldönti, hogy milyen logikai és fizikai adatszerkezetet hozzon létre. Ennek eredményeképpen célszerű kompromisszum alakul ki a tárolóigény, a visszakeresési és a karbantartási idő optimuma között. A tárolási struktúrát megvalósító szoftver-rendszer, az adatbázis-kezelő rendszer ugyanis csak az adatbázis-felügyelő által megadott szempontok alapján tudja automatikusan létrehozni az adatbázis fizikai szintjét, és előkeresni abból a kívánt logikai rekordokat.

Ebben a fejezetben viszonylag hosszasan foglalkozunk a belső ábrázolási szinttel, noha ez az átlagos felhasználó előtt elvileg rejtve marad. Az adatfüggésesség következtében a belső szint változtatásai nem jelentkeznek a felsőbb szinteken, legalább is elméletben. A gyakorlatban azonban a fizikai adatstruktúra és a fizikai elérési utak igen nagy mértékben befolyásolják a rendszer hatékonyságát. Jól megválasztott tárolási struktúrával jelentősen csökkenthetjük az adatbázis által a külső tárolón lefoglalt területet, jól definiált, a tároló szerkezetéhez és a várható felhasználói műveletekhez igazodó elérési utak létrehozásával pedig akár nagyságrendekkel is gyorsabb válaszadási, feldolgozási időt érhetünk el.

Ezért az adatbázis-felügyelőnek mindenképpen tisztában kell lenni azzal, hogy mi történik az adatmodell fizikai szintjén. Ugyanez érvényes azokra a felhasználókra is, akik jogosultak a saját külső szintjük támogatására tényleges fizikai elérési utakat, tárolási struktúrákat létrehozni, vagy logikailag definiálhatnak ilyet a koncepcionális szinten.

4.2 Mágneslemez

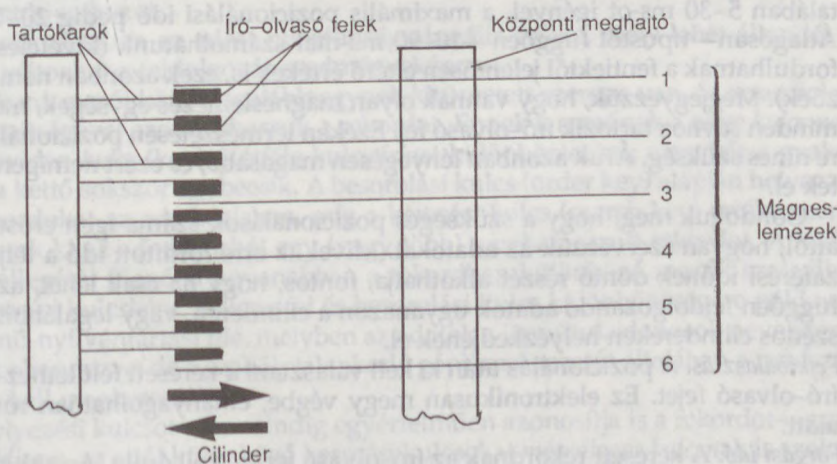
4.2.1 Fizikai jellemzők

A mai adatbázisok kizárólag mágneslemezekon helyezkednek el. Csak a biztonsági okokból kimentett adatokat tárolják esetleg az egyszerűbb és olcsóbb mágnesszalagokon vagy mágneskazettákon.

Ezen kívül legfőleg nagy adatbázisoknál fordulhat elő, hogy nagy tömegű, de igen ritkán használt adatok gazdasági megfontolásokból mágnesszalagon, vagy kazettán vannak, és csak arra az időszakra töltik őket be ideiglenesen mágneslemezre, amikor éppen dolgozni akarnak velük.

A mágneslemeztároló általában több, egymás fölött elhelyezkedő, közös tengely körül forgó merev lemezből áll. Minden felülethez tartozik (legalább) egy író-olvasó fej. Ennek helyzete sugárirányban, a tengellyel összekötő egyenes mentén változtatható. Így az adatok tárolása különböző pályákon (koncentrikus körökön), az ún. *sávokon* (track) történhet.

A különböző felületeken egymás felett elhelyezkedő és így az író-olvasó fej elmozdítása nélkül egyidejűleg elérhető sávok együtt egy *cilindert* alkotnak. (4.2 ábra)



4.2 ábra Mágneslemez tároló felépítése

A mágneslemezcsomag igen sok egységnél cserélhető. A lemezek száma és mérete igen különböző lehet. Kis gépeknél használnak egyfelületű, néhány száz kbyte kapacitású lemezt is, leggyakoribb azonban a 10–20 felület. A kapacitás cserélhető lemezeknél általában néhány Mbyte és 200 Mbyte közé esik, nem cserélhetőknél 20–2500 Mbyte vagy még több.

Míg korábban a tényleges tárolókapacitás növelése érdekében igen elterjedtek a cserélhető mágneslemezek, ma a nagy adatmennyiségek tárolására inkább állandó, nem cserélhető lemezeket használnak, és ezekből csatlakoztatnak többet a géphez. Igaz, hogy ez a megoldás drágább, és az összkapacitás véges (szemben a cserélhető mágneslemezek gyakorlatilag tetszés szerintire megnövelhető tárolókapacitásával), de a szükséges adatok bármikor elérhetőek. A nagyfokú mechanikus igénybevételt jelentő csere kiküszöbölésével a lemezek élettartama megnövekszik, a meghibásodási gyakoriság lecsökken. További előny még, hogy jóval pontosabb írási/olvasási mechanikát lehet alkalmazni. Így ugyanarra a felületre több adatot tudunk felvinni. Ezek a kedvező tulajdonságok kompenzálják a mágneslemez-egységek nagyobb számából adódó árkülönbséget.

4.2.2 Hozzáférési idő

A lemezen levő adatok eléréséhez szükséges idő a következőkből tevődik össze:

- Fejmozgatás.** Írás, illetve olvasás előtt az író–olvasó fejeket a megfelelő cilinderre kell pozicionálni. Ennek ideje attól függ, hogy melyik cilinderen dolgozott utoljára az egység. Természetesen, ha a fejek a kívánt helyen vannak, mozgatási időre nincs szükség, egyébként ez egy konstans (a felgyorsuláshoz szükséges) és egy a távolságtól közelítően lineárisan függő részből adódik. Ha tehát a fejeket egyáltalán mozgatni kell—akárcsak a szomszédos cilinderre is—ez általában 5–30 ms-ot igényel, a maximális pozicionálási idő pedig 20–165 ms. Átlagosan—típustól függően—10–90 ms-mal számolhatunk (kivételesen előfordulhatnak a fentiektől jelentősen eltérő értékek is, ezek azonban nem jellemzőek). Megjegyezzük, hogy vannak olyan mágneslemez-es egységek, melyeken minden sávhoz tartozik író–olvasó fej. Ezeken természetesen pozicionálási időre nincs szükség. Áruk azonban lényegesen magasabb, és ezért nemigen terjedtek el.

Gondoljuk meg, hogy a szükséges pozicionálások száma igen erősen függ attól, hogyan szervezzük az adatokat. Mivel az erre fordított idő a teljes hozzáférési időnek döntő részét alkothatja, fontos, hogy ha csak lehet, az össze-függően feldolgozandó adatok ugyanazon a cilinderen, vagy legalábbis szomszédos cilindereken helyezkedjenek el.

- Fejkiválasztás.** A pozicionálás után ki kell választani a keresett felülethez tartozó író–olvasó fejet. Ez elektronikusan megy végbe, elhanyagolhatóan rövid idő alatt.
- Forgási idő.** A keresett rekordnak az író–olvasó fej elé kell érnie. Az ehhez szükséges idő cilinderenként egy író–olvasó fej esetén nulla és a teljes körülfordulás ideje közé esik. Átlagban a körülfordulási idő felével számolhatunk. Egyes típusoknál, ahol cilinderenként $n \geq 2$ fejkészlet van, a szükséges idő az előbbinek n -ed része.

Mivel a teljes körülfordulás ideje rendszerint 5–25 ms, az átlagos forgási idő a fentieknek megfelelően 1,25–12,5 ms.

d) *Adatátvitel.* A korszerű mágneslemezek adatátviteli sebessége általában 150–3000 kbyte/s között van.

4.3 Lapszerkezet

A korszerű gépek az adatokat a belső operatív tárolóban ún. lapokon tárolják. Ezek mérete általában 1–4 kbyte. A mágneslemezen a tárolási egység a blokk. Ez az a fizikailag összefüggő adatmennyiség, ami egy művelettel beíródik az operatív tárolóba, vagy kiíródik onnan a lemezre. A blokkok mérete lemeztípustól függően rögzített, vagy változtatható, de minden adatbázis-kezelő rendszerben olyan, hogy egy lap egy blokkot foglal le (esetleg egészszámú blokkot, ha a blokkméret rögzített). Így a blokk és a lap között a továbbiakban nem teszünk különbséget, sőt, az egyszerűség kedvéért föltesszük, hogy az adatbázis minden egyes lapja megegyezik egy fizikai blokkal és a kettőt egyértelműen azonosítani tudjuk akár a lap, akár a blokk címével. A hardver szintjén ez azt jelenti, hogy megadjuk, melyik egységen, hányadik cilinderen, ennek melyik lemezoldalhoz tartozó sávjában, hányadik blokkban található.

A fizikai szinten az adatokat rekordokban tároljuk. Rekordokat alkothatnak például egy személyi nyilvántartásban az egyes személyek adatai, melyekhez többek között a következő tulajdonságok tartoznak: név, lakhely, beosztás, születési év stb., például konkrétan az alábbi értékekkel:

Kovács János, Budapest, Rákóczi út 20., matematikus, 1941 stb.

Minden rekord egy vagy több elemi mezőből, adatelemből áll. Az elemi *mező* vagy *adatelem* a legkisebb, önálló tulajdonsággal rendelkező adat, melyre név szerint lehet hivatkozni.

A mező tartalma az adott tulajdonság aktuális értéke, mely lehet állandó, de meg is változhat a feldolgozás eredményeképpen.

A rekord mezői közül legalább egynek kitüntetett szerepe van, és arra szolgál, hogy segítségével azonosíthassuk a rekordot. Ennek a mezőnek a neve *kulcsmező*, vagy röviden *kulcs* (key). Kétféle kulcstípust különböztetünk meg, bár a gyakorlatban a kettő sokszor egybeesik. A besorolási kulcs (order key) alapján helyezzük el a rekordokat az adatbázisban, míg a keresési kulcs (search key) értéke szerint választunk ki az adatbázisból egy (vagy több) meghatározott rekordot. A feldolgozás jellegétől függően ugyanabban a rekordban különböző mezők szolgálhatnak keresési kulcsként. A keresési és besorolási kulcs különbözőségére példa egy gépjármű-nyilvántartási file, melyben az adatok a járműtulajdonosok neve szerint vannak elrendezve, de a szabálytalankodó gépjárművezetőt általában a rendszám alapján kell megkeresni.

Az elhelyezési kulcsot—ha mindig egyértelműen azonosítja is a rekordot—gyakran *elsődleges*, az ettől különböző keresési kulcsokat *másodlagos kulcs*nak is szokták nevezni.

A rekordtípusokon, azok előfordulásain, illetve egyes mezőin elvégezhető műveleteket három fő csoportba sorolhatjuk:

- a) *Szerkesztés jellegű műveletek.* Ide tartozik a rekordtípus létrehozása, megszüntetése, valamint újraszervezése. Az első két művelet tartalma magától értetődő. A harmadikra akkor van szükség, ha az igények módosulása vagy a karbantartási műveletek következtében megváltozott fizikai adattárolási struktúra miatt az eddigi adatszerkezet már nem tudja kellő hatékonysággal kielégíteni a felhasználói igényeket. Ekkor—amennyiben erre mód van—ugyanazon adatok más-fajta fizikai elrendezésével általában vagy a tárolóhely lefoglalás, vagy a hozzáférési idő, vagy mindkettő szempontjából hatékonyabb fizikai adatelrendezést hozunk létre. Az első két művelet típusra az adatleíró nyelvet használjuk, a harmadikra általában az adatbázis-kezelő rendszer szolgáltató programját.
- b) *Karbantartás jellegű műveletek,* melyen új rekordoknak az adatbázisba való behelyezését, azaz az állomány bővítését és meglevő rekordoknak az állományból való törlését értjük. Ide soroljuk még azokat a rekordmódosító műveleteket is, melynek eredményeképpen a rekordnak a besorolási vagy bármelyik keresési kulcsa megváltozik.

Ugyancsak karbantartási jellegű műveletnek tekinthetjük, ha az adatbázist új lapokkal bővítjük.

- c) *Feldolgozás jellegű műveletek.* Ezek közé tartozik adott feltételnek vagy feltételeknek megfelelő rekordok kikeresése és e rekordok tartalmának módosítása. Ez a módosítás azonban nem vonatkozhat kulcsként is szolgáló tulajdonságok értékeire.

Egy adatbázisban többféle rekordtípus lehet. Így például a személyi adatbázis a személyekre jellemző rekordtípus mellett tartalmazhat osztályok és munkakörök jellemzőit leíró rekordtípusokat. A fizikai szinten mindig teljes rekordokkal dolgozunk, és az író/olvasó műveletek azt a lapot (vagy lapokat) mozgatják a lemez és az operatív tár között, amelyik a rekordo(ka)t tartalmazza.

Ha a logikai szintű rekord nem egyezik meg a fizikaival, akkor azt az adatbázis-kezelő rendszer rakja össze a fizikai rekord(ok)ból a felhasználó számára az adatleíró nyelvben megadott szabályok szerint.

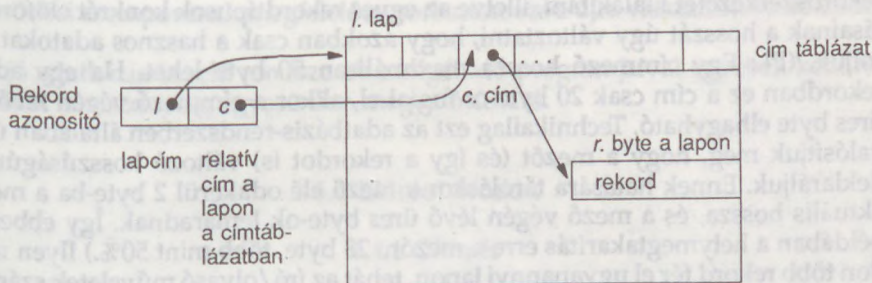
A fizikai rekordot a fizikai szinten egyértelműen azonosítja a címe, amelyik két részből tevődik össze:

- a lap címéből, amelyen a rekord van,
- a rekordnak a lapon belüli címéből, ami rendszerint a rekord első byte-jának a lap kezdetétől való távolságot jelenti.

Tipikus címezési módot láthatunk a 4.3 ábrán.

Ha egy lap bent van az operatív tárból, akkor a rajta levő adatokhoz általában 3–5 nagyságrenddel gyorsabban tudunk hozzáférni, mintha azt a lapot előbb be kellene olvasni a memóriába. Ezért a tárolási struktúrát célszerű úgy kialakítani, hogy az egyszerre vagy közvetlenül egymás után használt adatok lehetőleg ugyanazon a lapon legyenek, mert ezáltal a lassú író/olvasó műveletek számát csökkenteni lehet. Az adatbázis-kezelő rendszerek egyidejűleg általában több lapot tartanak a belső memóriában, és ezekről nyilvántartást is vezetnek. Így, ha egy rekordra szükség van, akkor először megvizsgálják, hogy a megfelelő lap bent van-e a memóriában. Csak ha nincs ott, akkor hozzák be a lemezzel egy másiknak a helyére.

Nem elég azonban a mágneslemez író/olvasó műveletek számát minimalizálni. Az író/olvasó műveletek végrehajtási ideje igen nagy mértékben függ attól, mi-



4.3 ábra Címzés a fizikai szinten. A rekordot azonosító cím két részből áll. Az első rész a lap címére (l), a második rendszerint az adott lapon levő címtáblázatnak arra az elemére mutat (c), amelyik a rekord kezdőbyte-jának a lapon belüli címét (r) tartalmazza

ként helyezkednek el a lemezen egymáshoz viszonyítva az egymás után beolvasandó vagy kiírandó lapok. Ha egymásután vannak ugyanazon a sávon, akkor egy lemezkörülfordulás alatt a sáv összes rekordját be tudjuk olvasni. Nem ilyen jó a helyzet, ha csak ugyanazon a cilinderen helyezkednek el, mert ilyenkor az egyes lapok beolvasása között várakoznunk kell, általában egy $1/2n$ lemezfordulatot, amíg a rekord odaér az n író/olvasófej valamelyike alá. Legrosszabb az az eset, amikor a rekordok különböző cilindereken vannak. Ilyenkor a várakozási idő megnövekszik a pozicionálási idővel is. Ez attól függően, hogy milyen távol van egymástól a két cylinder, rendszerint egy körülfordulási időnek 1 és 5-szöröse között van.

Így például, ha egy cilinderen 20 sáv és egy sávon 5 lap (blokk) van, akkor, ha a lapok beolvasási sorrendje megegyezik azzal a sorrenddel, ahogy a blokkok egymás után elhaladnak az író/olvasó fej előtt, a 100 lap beolvasása 20 körülfordulási időt, egy gyors lemeznél kb. 100 msec-ot vesz igénybe. Ebben az esetben azt mondjuk, hogy az adatok elérési sorrendje megegyezik a tárolási sorrenddel.

Amennyiben ez az egyezés nem áll fenn, akkor a legkedvezőtlenebb esetben minden újabb lap beolvasásánál pozicionálnunk is kell és a 100 lap beolvasása az átlagos elérési idő 100-szorosa lesz. Ez az előzőhöz hasonló jellegű gyors elérésű lemeznél összesen 1–2 sec lesz, vagyis egy nagyságrenddel nagyobb.

A fentieket összefoglalva megállapíthatjuk, hogy az adatok fizikai tárolásának meghatározásakor az alábbi szempontokat kell figyelembe vennünk:

1. A várhatóan egyidejűleg vagy közvetlenül egymás után használt rekordok lehetőleg egy lapon legyenek. Ezáltal minimalizáljuk az író/olvasó műveletek számát.
2. Amennyiben a feldolgozáshoz szükséges rekordok száma és/vagy mérete olyan nagy, hogy egyszerre vagy közvetlenül egymás után több lapot kell beolvasnunk és/vagy kiírnunk, akkor igyekezzünk ezeket a lapokat úgy elhelyezni, hogy a lemezen levő elhelyezkedésük sorrendje megegyezzen a beolvasási/kiírási sorrenddel. Ezáltal az egyes író/olvasó műveletek végrehajtási idejét minimalizáljuk.
3. Az egyes rekordok hosszát minimalizáljuk. Ehhez igyekezzünk redundancia-mentes (a gyakorlatban csak a szükséges minimális redundanciával rendelkező)

rekordszerkezetet kialakítani, illetve az egyes rekordtípusok konkrét előfordulásainak a hosszát úgy változtatni, hogy azokban csak a hasznos adatokat tároljuk. (Pl.: Egy címmező hossza maximálisan 50 byte lehet. Ha egy adott rekordban ez a cím csak 20 byte-ot foglal el, akkor a címmező végén lévő 30 üres byte elhagyható. Technikailag ezt az adatbázis-rendszerben általában úgy valósítjuk meg, hogy a mezőt (és így a rekordot is) változó hosszúságúnak deklaráljuk. Ennek hatására tároláskor a mező elé odakerül 2 byte-ba a mező aktuális hossza, és a mező végén lévő üres byte-ok lemaradnak. Így ebben a példában a helymegtakarítás erre a mezőre 28 byte, több mint 50%.) Ilyen módon több rekord fér el ugyanannyi lapon, tehát az író/olvasó műveletek számát csökkentjük.

Az előzőek ismeretében a fizikai író/olvasó műveletek felgyorsításával kapcsolatban még egy fogalmat kell bevezetnünk, a *sűrűsödést*. Ennek az alapelve, hogy azokat a rekordokat, amelyeket várhatóan együtt, vagy egymás után dolgozunk fel, fizikailag lehetőleg egy lapra, vagy ha ez nem valósítható meg, akkor a feldolgozási sorrendnek megfelelően, közvetlenül egymás utáni lapokra helyezzük el. Ha például egy rekordtípust rendszeresen egy kulcs szerinti sorrendben dolgozunk fel, akkor célszerű ezeket a rekordokat eszerint a kulcs szerint sorbarendezve bevinni az adatbázisba. Így a logikai kulcs szerint egymáshoz közel levő rekordok fizikailag is egymáshoz közel (közvetlenül betöltés után a logikailag egymás utániak fizikailag egymás után) fognak elhelyezkedni. Ekkor azt mondjuk, az adott rekordtípus ezen kulcs szerint sűrűsödik. Így a leggyakoribb feldolgozásnál, a sűrűsödési kulcs szerinti szekvenciális feldolgozásnál, a lapváltások számát alacsony értékre tarthatjuk.

A sűrűsödés nemcsak egy adott rekordtípuson belül lehet előnyös a feldolgozási idő szempontjából. Hierarchikus jellegű feldolgozásnál, pl. amikor minden egyes megrendelésnek van egy megrendelés-fej típusú és sok ehhez kapcsolódó megrendelés-részlet típusú rekordja, és feldolgozáskor először a fej, majd ezután sorban a hozzátartozó részlet típusú rekordokat dolgozzuk fel, célszerű ezeket a különböző típusú rekordokat is ugyanazon a lapon tárolni.

Az adatbázis-kezelő rendszer elő tudja segíteni a sűrűsödést, ha az adatbázis-felügyelő megadja, milyen szempontok szerint kell azt megvalósítani. Az adatok első betöltésénél eleve úgy helyezi el őket, hogy sűrűsödjének. új rekordok beillesztésénél pedig igyekszik azokat, amennyiben mód van rá, olyan lap(ok)ra beszární, hogy a sűrűsödés fennmaradjon.

Egy adott rekordtípusra, vagy egymással kapcsolatban álló rekordtípusokra a sűrűsödés csak egy kulcs, illetve egy kapcsolat szerint valósítható meg. (Véletlenül persze előfordulhat, hogy ez egyben egy másfajta szempont szerint is sűrűsödést jelent.) A hierarchikus modellek egyik fő előnye, hogy a különböző rekordtípusok közti kapcsolaton alapuló sűrűsödést hatékonyan és egyszerűen tudják kezelni.

Természetesen, ahogy egyre újabb és újabb rekordokat szúrunk be a meglevők közé, előfordulhat, hogy a sűrűsödés megszűnik. Az is lehet, hogy a felhasználói szokások, igények változnak meg, és most más kulcs szerinti sorrendben dolgozzák fel leggyakrabban az adatokat. Ez persze az adatbázis működését nem változtatja meg, de a feldolgozási időt jelentősen megnövelheti. Ilyenkor az adatbázis-felügyelő feladata, hogy az adatbázis újraszervezésével a sűrűsödést újra visszaállítsa. Ez azt jelenti, hogy az érintett adatokat először kimentí, majd a

sűrűsödési szempontnak megfelelően sorbarendezve újra visszatölti az adatbázisba.

A jó adatbázis-kezelő rendszerek szolgáltató programjaival egyebek között ez a feladat is viszonylag egyszerűen megvalósítható.

4.4 Elérési módok

4.4.1 Címzés

A korszerű adatbázis-kezelő rendszerekben a fizikai szinten is több szintű címzést és a címzés alapján több lépésből álló elérési utat alkalmaznak. Az adatokat adatbázis lapokon (lemez blokkokban) tárolják, és az egyes rekordok címe két részből, a lapnak (blokknak) a címéből, valamint a rekord ezen lapon belüli címéből tevődik össze. A koncepcionális szintek rekordjait az adatleíró nyelven meghatározott módon képezzük le a fizikai lapokra. Ez azt jelenti, hogy adott típusú rekordokról mindig tudjuk, hogy melyik lapokon található. Sőt, a leképezés módjától függően gyakran azt is tudjuk, hogy egy adott rekordot pontosan melyik lapon, esetleg azon a lapon pontosan melyik helyen célszerű keresni.

A lap címe végső soron mindig valamilyen lemezegység valamilyen cilinderjén egy sávnak egy blokkját jelenti, míg a lapon belüli cím a rekord első byte-jának távolsága a lap kezdetétől. A felhasználói szinten azonban nem ezen fizikai címek alapján keresünk az adatbázisban, hanem adattartalom szerint. Azt akarjuk megtudni, hogy egy adott típusú, vagy ezen belül tovább részletezve, egy adott kulcsú rekordot, mely lapon vagy lapokon, és ezen belül hol keressük. A koncepcionális és a belső szint közötti leképező rendszer ezt a tartalom alapján való keresést fordítja át fizikai címmé.

Ez a címtranszformáció lehet egyértelmű. A rekordot biztosan azon a fizikai címen találjuk meg, ha bent van az adatbázisban, vagy oda kell eltárolnunk. Lehet azonban olyan is, hogy csak egy kiindulási pontot határoz meg, és innen kell meghatározott algoritmus szerint továbblépve megvizsgálni, hogy eljutottunk-e a kívánt adathoz.

4.4.2 Alapvető elérési utak

A most ismertetendő módszerek egy része egyaránt alkalmas mind a lapok, mind a lapon belüli rekordcímek meghatározására. Ezért ezeket egységesen tárgyaljuk. Ilyenkor adatok eléréséről, adatok címzéséről beszélünk. Ahol az eljárást vagy a lapok, vagy a rekordok megtalálására használják csak, ott azt külön megemlítjük. Az adatok szervezési és elérési módja lehet:

- soros, azaz a fizikai egymásutániség szerinti,
- szekvenciális, valamilyen logikai sorrend szerinti,
- közvetlen² (indexelt),

² A közvetlen elérést gyakran direkt, vagy véletlenszerű (random) elérésnek is nevezik.

- közvetlen² (hashing),
- hierarchikus.

A relációs modellek általában csak az első három módot használják, ezért azokat részletesebben ismertetjük. A hierarchikus elrendezésnek és a hashingnek csak a lényegét tárgyaljuk röviden.

Megjegyezzük, hogy ugyanazon adatokat még ugyanazon kulcs szerint is többféle módon szervezhetjük és érhetjük el ugyanabban az adatbázisban.

A fizikai sorrendnek megfelelően bármilyen adatállományt feldolgozhatunk, legfőképpen a hatékonyság lesz az esetek többségében igen alacsony.

Emellett a legtöbb adatbázis-kezelő rendszer lehetővé teszi ugyanazoknak az adatoknak valamely logikai sorrendnek megfelelő, valamilyen kulcs értéke szerinti szekvenciális feldolgozását, és ugyanezen kulcs szerinti direkt, indexek alapján való elérését is, amikor a kulcs értékéből közvetlenül megkapjuk a rekord címét.

4.5 Soros elérés

A fizikai sorrendben való tárolásnál az adatok oly módon helyezkednek el egymás után, hogy sem a kulcsok és a tárolón elfoglalt helyük között, sem az egymás után következő adatok kulcsai, vagy egyéb tulajdonságai között nincs semmilyen logikai kapcsolat. Az adatok folyamatosan (esetleg más típusú adatok közbeékelődése miatt nagyobb csoportokra bontva, csak az egyes csoportokon belül folyamatosan) követik egymást a tárolón. Feldolgozásuk csak sorosan, fizikai elhelyezkedésük alapján történhet.

A soros szervezésű adatállományok létrehozása, karbantartása egyszerű. Az új adatot az állomány végéhez kell hozzáírni. Miután a különböző adatok között nincs logikai kapcsolat, bármelyiket törölhetjük, módosíthatjuk anélkül, hogy ez az állomány szerkezetére hatással lenne. Ha a változtatás az adat hosszát nem növeli meg, akkor az adatot a helyén is minden probléma nélkül megváltoztathatjuk. Ugyanakkor viszont adott tulajdonsággal, meghatározott kulcsértékkel rendelkező rekordoknak a kikeresése igen hosszadalmas, kedvezőtlen esetben akár az egész állományt is végig kell egyesével vizsgálni.

Könnyű belátni, hogy egy soros állományban egy egyedi (az egész állományban csak egy ilyen lehet) kulcsérték meghatározásához átlagosan az adott rekordok felét kell végigvizsgálni. Ha az adott kulcs nem szerepel az állományban, vagy több is lehet belőle és mindegyiket meg kell találni, akkor mindig végig kell nézni a teljes állományt az elejétől a végéig.

Ezért ez a fajta feldolgozás, ha az adatoknak csak egy kis részét akarjuk megkeresni, az összehasonlítások nagy száma miatt meglehetősen lassú. Látni fogjuk, hogy a következő pontokban ismertetendő szervezési módoknál az összehasonlítások száma lényegesen lecsökkenthető, így a keresés sokkal gyorsabb lehet. Ugyanakkor viszont, mivel a lapok soros feldolgozásánál a forgási idő és a pozicionálási idő általában zérus, az egymás utáni adatok elérése sokkal, általában nagyságrenddel gyorsabb. Egy lapon lévő rekordoknál az időmegtakarítás nem ilyen nagy, sőt sokszor elhanyagolható. Ilyenkor is előny viszont a feldolgozás egyszerűsége. A következő rekord helyének megállapítása nem igényel semmilyen

komolyabb számítást, csak egy számláló rögzített értékkel való megnövelését jelenti. Így a kevesebb összehasonlításból származó előny csak akkor jelentkezik valóban a feldolgozási idő csökkenésében, ha nagyságrendileg kevesebb adatot kell nem-sorosan megvizsgálnunk.

Ezért a soros feldolgozást akkor szokták alkalmazni, ha az adatok jelentős részéhez mindig hozzá kell férnünk. Olyankor, amikor minden esetben így dolgozunk, nem is érdemes a soroson kívül más szerkezetet létrehozni az adatokon. Adatbázisok biztonsági kimentése, illetve visszaállítása az így kimentett adatokból leggyakrabban sorosan történik.

Ennek a szervezési és elérési módnak még az az előnye is megvan, hogy tárolóként a mágneslemezeknél jóval olcsóbb mágnesszalagok és kazetták is használhatók.

4.6 Szekvenciális szervezés és elérés

Szekvenciális szervezésnél az egymást követő adatok keresési kulcsa között egyértelmű logikai kapcsolat van. Növekvő sorrendnél mindegyik elem kulcsa nagyobb (vagy egyenlő, ha duplikátumok lehetnek) a közvetlenül előtte lévő kulcsnál. (Csökkenő sorrendnél kisebb, illetve egyenlő.) A szekvenciális úton való hozzá-férésnél bármely adat után csak a közvetlenül utána következőt (néhány rendszernél a közvetlenül előtte lévő is) érhetjük el.

Ahhoz, hogy egy üres szekvenciális állományt feltöltsünk, a betöltendő adatokat a keresési kulcs szerint rendezni kell, és ezt a rendezett állományt kell bevin-nünk. Meglévő állománynál a szekvenciális szerkezet ráépítését, illetve új adatok beillesztését, törlését a 4.6.1 pontban tárgyaljuk.

Szekvenciális elérést ugyanazon adatokon többféle kulcs alapján is létrehozha-tunk. Ez rekordok feldolgozásánál fontos, amikor ugyanazokat a rekordokat fel-dolgozástól függően különböző szempontok szerint akarunk sorba rendezve elérni. (Pl. vezetéknev, születési év vagy munkahely szerint.)

A keresési kulcsok között egyértelműen fennálló logikai kapcsolat azonban nem jelenti azt, hogy a tényleges fizikai tárolási sorrend is megegyezik a logikai sor-renddel. Ha az adatokat egy kulcs szerint gyakran keressük vissza, akkor ezt is célszerű minél nagyobb mértékben megvalósítani, hogy a soros elérés előnyeit is ki tudjuk használni. Természetesen, ha több keresési kulcsra építettünk fel szekvenciális szerkezetet, akkor a fizikai sorrendre tett kikötésünk általában csak egyre teljesülhet.

Lapok szekvenciális elérésénél azért előnyös a fizikai és a logikai sorrend egye-zése, mert az egymás utáni lapok feldolgozása között minimális lesz a forgási és a pozicionálási idő.

Rekordoknál az egyezés teljes, vagy csaknem teljes betartása nem annyira fon-tos. Nagymértékben gyorsítja már a feldolgozást a 4.3 pontban ismertetett sűrű-södés, mely a kétfajta sorrend egyezésének egy enyhébb formája. Ennél a logikailag (szekvenciálisan) egymást követő rekordok lehetőleg ugyanarra a lapra kerülnek. Így szekvenciális feldolgozásnál csak ritkán kell lapot váltani, ami csök-kenti az író/olvasó műveletek, illetve a pufferben tartandó lapok számát.

Természetesen a sűrűsödés is általában csak egy kulcsra valósítható meg egy adott rekordtípusnál. Ez lehetőleg a szekvenciális feldolgozásban leggyakrabban használt kulcs legyen. Ennek megvalósítása érdekében a rekordokat a szekvenciális feldolgozásnak megfelelően logikai sorrendbe rendezik, és így töltik be először az adatbázisba. Ezáltal a szekvenciális feldolgozás jelentősen gyorsabb lesz. Az új rekordokat—amíg van hely a lapon—a sűrűsödési feltételeknek megfelelően próbálja az adatbázis kezelő rendszer elhelyezni. Így a feldolgozás csak akkor lassul le jelentősen, ha olyan sok új rekordot kellett beilleszteni, vagy megfelelő keresési kulcsát módosítani, hogy a sűrűsödés már nem, vagy nem eléggé áll fenn. Ilyenkor az adott rekordtípus kimentésével, majd szekvenciális sorrendben való visszatöltésével a sűrűsödés és a szekvenciális feldolgozás eredeti sebessége visszaállítható.

A legtöbb adatbázis-kezelő rendszerben erre kész szolgáltatóprogramok állnak a felhasználó rendelkezésére.

4.6.1 Listaszerkezet

Az adatbázisban listaszerkezettel logikailag egymás után következő lapokat, adott rekordtípus logikailag egymást követő rekordjait, ugyanannak a rekordnak fizikailag különböző helyen tárolt mezőit, és különböző rekordtípusoknak hierarchikusan egymás alatt levő rekordjait szokták összekapcsolni.

A listaszerkezetben a logikailag egymás után következő elemek egy szekvenciális szerkezet elemei, de a tényleges fizikai elhelyezés és a kulcs között semmiféle kapcsolat nincs. Az adatok közti logikai kapcsolatot mutatók, pointerek határozzák meg, és ezek teszik lehetővé a szekvenciális feldolgozást is. Minden adat-elem, vagy más néven cella, két részből tevődik össze:

- a tényleges adatokat tartalmazó adatrészből és
- a cellához logikailag közvetlenül kapcsolódó cellá(k) címét tartalmazó mutató(k)ból, pointer(ek)ből.

A szokásos szóhasználatban általában nem tesznek különbséget a listaszerkezetben tárolt adatoknál az adatrész és a cella között. Némileg pongyolán, de adatnak nevezik a teljes cellát.

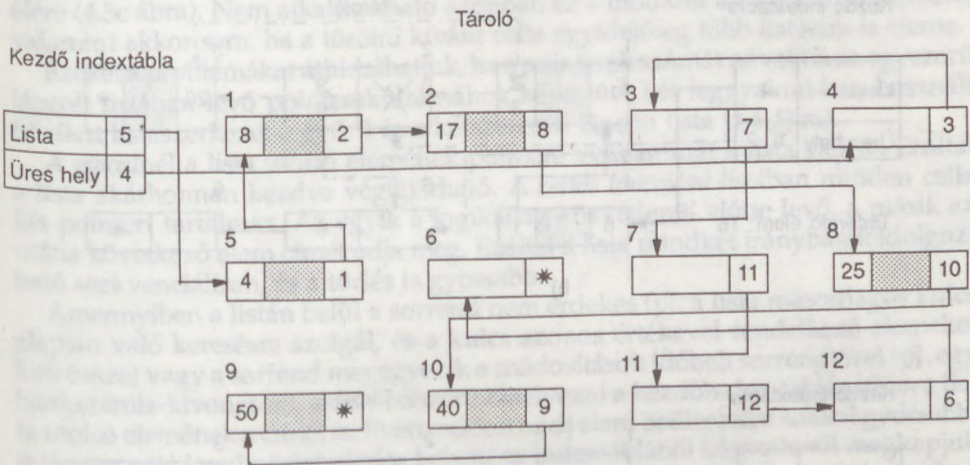
A pointermezőben háromféle formában lehet megadni a következő rekord címét:

- abszolút hardver címként,
- relatív címként, ez adatbázisokban a lapoknál általában a következő lap relatív címe az adatbázis elejétől számolva, rekordoknál pedig a következő rekordot tartalmazó lap relatív címe, és ezen belül a rekord relatív helye a lapon, a 4.3 pontban ismertetett formában,
- szimbólikus cím, mely a logikailag soron következő rekord elhelyezési kulcsát tartalmazza. Ebből a rendszer határozza meg a tényleges fizikai címet.

A ma üzemelő adatbázis-kezelő rendszerek többségénél a második módszert, a relatív címezést alkalmazzák, ha szükség van a listaszerkezet kialakítására. A relatív címet az operációs rendszer alakítja át abszolút címmé.

A továbbiakban az ábrákon a jobb áttekinthetőség érdekében rendszerint relatív vagy szimbólikus pointereket adunk meg. Ez azonban nem jelenti azt, hogy tényleges pointerként is csak ilyen címeket célszerű használni.

A 4.4. ábrán egyszerű listaszervezetet mutatunk be. Minden egyes elem (cella) bal szélső mezője annak a tulajdonságnak az értékét (kulcsot) ábrázolja, mely a szekvenciális szerkezetet meghatározza. A bevonalkázott területen egyéb, a listaszervezet szempontjából közömbös adatok vannak, míg a jobb oldalon álló mező a listapointert, a logikailag soron következő adatelem címét tartalmazza. Ezen pointer segítségével egymás után sorban végig tudunk menni a logikai szekvencialitásnak megfelelő sorrendben a lista elemein, mint egy láncon. Ezért szokásos ezt az adatszerkezetet *láncolt listának* is nevezni. A lánc utolsó elemét a pointermezőben levő speciális jel jelzi (az ábrákon *).

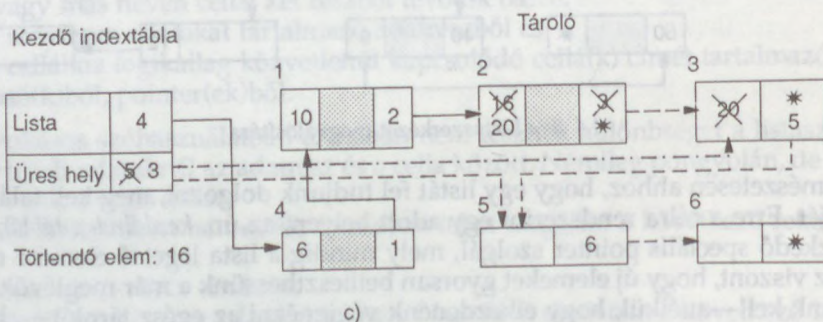
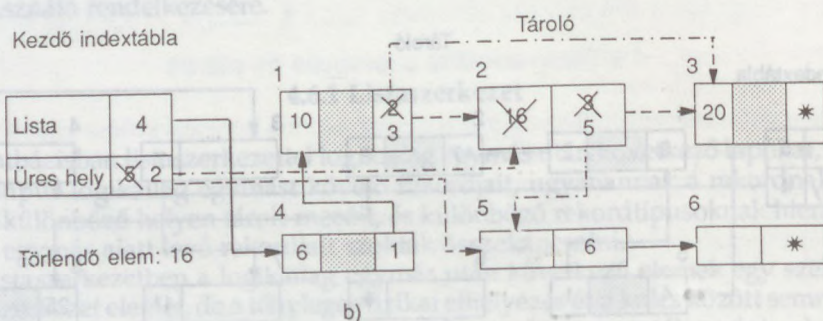
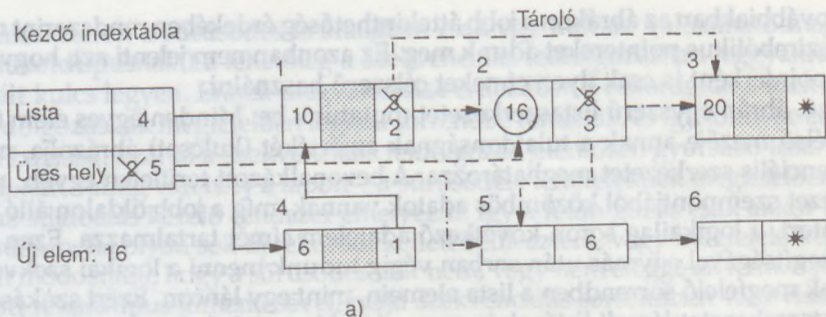


4.4 ábra Listaszervezet megvalósítása

Természetesen ahhoz, hogy egy listát fel tudjunk dolgozni, meg kell találnunk az elejét. Erre a célra rendszerint egy adott helyen, az ún. *kezdőindex-táblában* elhelyezkedő speciális pointer szolgál, mely mindig a lista legelső elemére mutat. Ahhoz viszont, hogy új elemeket gyorsan beilleszthessünk a már meglévők közé, tudnunk kell—anélkül, hogy elkezdenénk végignézni az egész tárolót—, hol találunk szabad helyet, ahová ezt az elemet fizikailag elhelyezhetjük. Ennek érdekében rendszerint létrehozunk egy másik listát, az *üres helyek listáját* is. Ez láncolva tartalmazza az összes pillanatnyilag üres tárolóhely címét.

Ha új elemet kívánunk beilleszteni a listába, akkor az fizikailag az üres lista első elemébe kerül, melynek címét az üres lista kezdetére mutató kezdőindex-táblából azonnal megkaphatjuk.

Természetesen ezáltal az üres lista első elemére mutató pointert meg kell változtatnunk. A kezdőindex-táblába a most betöltött cella pointermezőjében eredetileg levő érték kerül, mely eddig az üres lista második elemére mutatott. A fizikailag így meghatározott új elem logikai helyét a láncon való szekvenciális végighaladással kulcsa alapján határozzuk meg. A beillesztés után a logikailag



4.5 ábra

Lista módosítása (a szaggatott --- vonal a módosítás előtti, a pontozott-szaggatott ---- vonal az azutáni állapotot jelzi).
a) új elem beillesztése; b), c) meglévő elem törlése

közvetlenül előtte levő elem pointermezőjének tartalma lesz az új elem pointerének értéke, míg a régi pointer helyébe az új tag címe kerül. Hasonló módon egy listabeli elem törlése a saját, az előtte levő elem és az üres lista kezdetére mutató pointer értékének megváltoztatását vonja maga után, mivel a törölt tag az üres lista legelejére kerül. A beillesztés és törlés folyamatát a 4.5 ábrán láthatjuk. Ha a

rekordokat az adatkezelő nyelv segítségével illesztjük be az adatbázisba, (vagy töröljük onnan), akkor az adatbázis-kezelő rendszer automatikusan elvégzi ezeknek a pointereknek az aktualizálását.

A gyakorlatban sokszor problémát okoz, ha a cellák csak a logikailag soron következő elem címét tartalmazzák. Ha például a feldolgozást valamilyen okból nem a lista legelejéről kezdjük (illetve folytatjuk tovább), akkor a lánc végéről nem tudunk automatikusan visszatalálni a lista elejére, hogy az ott levő adatokat is feldolgozhassuk. Komolyabb nehézséget jelent, ha a törlendő elemet nem szekvenciálisan, a lánc alapján kerestük ki, mert akkor nem tudjuk, hogy fizikailag hol van a logikailag közvetlenül előtte levő elem, melynek pointerét meg kell változtatni a törlés következményeképpen. Ezen utóbbi problémát is áthidalhatjuk, ha csak egy lánc halad keresztül az adatokon azzal, hogy a törölt cella után következő cella tartalmát átvisszük törlendőbe, és az így kiürült cellát helyezzük az üres lista élére (4.5c ábra). Nem alkalmazható azonban ez a módszer a lista utolsó elemére, valamint akkor sem, ha a törölni kívánt cella egyidejűleg több listának is eleme.

Ezeket a problémákat áthidalhatjuk, ha a pointerek számát növeljük az egyszerű láncolt listában levő pointerek számához képest. A két leggyakrabban használt bővített listaszerkezet a gyűrű és a kétszeresen láncolt lista (4.6 ábra).

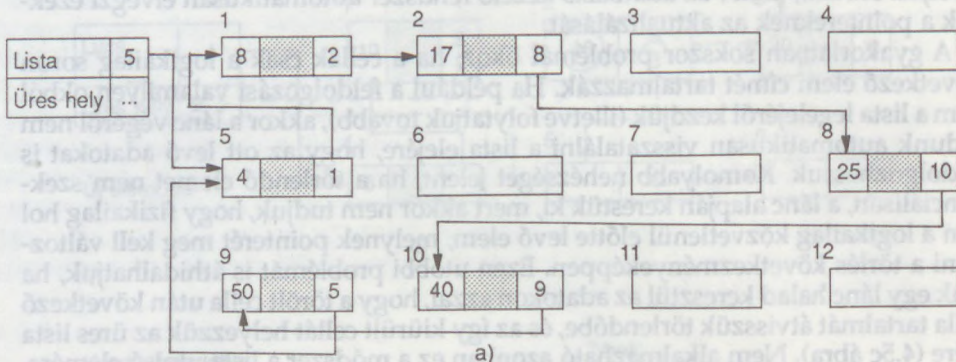
A gyűrűnél a lista utolsó elemének pointeré visszamutat a lista elejére. Ezáltal a lista akárhonnan kezdve végigjárható. A kettős láncolású listában minden cella két pointert tartalmaz. Az egyik a logikailag közvetlenül előtte levő, a másik az utána következő elem címét adja meg. Ezáltal a lista mindkét irányban feldolgozható szekvenciálisan, és a törlés is gyorsabb.

Amennyiben a listán belül a sorrend nem érdekes (pl. a lista másodlagos kulcs alapján való keresésre szolgál, és a kulcs azonos értékével rendelkező elemeket köti össze) vagy a sorrend megegyezik a módosítások időbeli sorrendjével (pl. egy bankszámla-kivonaton), akkor célszerű elhelyezni a kezdőindex-táblázatban a lista utolsó elemének a címét is. Ilyen módon az új elem beillesztése sokkal gyorsabb. A láncon való lassú végighaladás helyett az indestáblából közvetlenül megkapjuk nemcsak azt a helyet, ahova az új rekordot fizikailag el kell helyeznünk (az üres helyek listájának kezdőcíméből), hanem azt az elemet is, amelyikhez logikailag hozzá kell láncolnunk. Ezt a módszert FIFO-nak (First In First Out) is nevezik. Szokták használni a LIFO-t (Last In First Out) is, amikor az új elemet mindig a legelső helyre teszik.³ Ilyenkor az indestáblában nincs szükség a lista végét jelző pointerre.

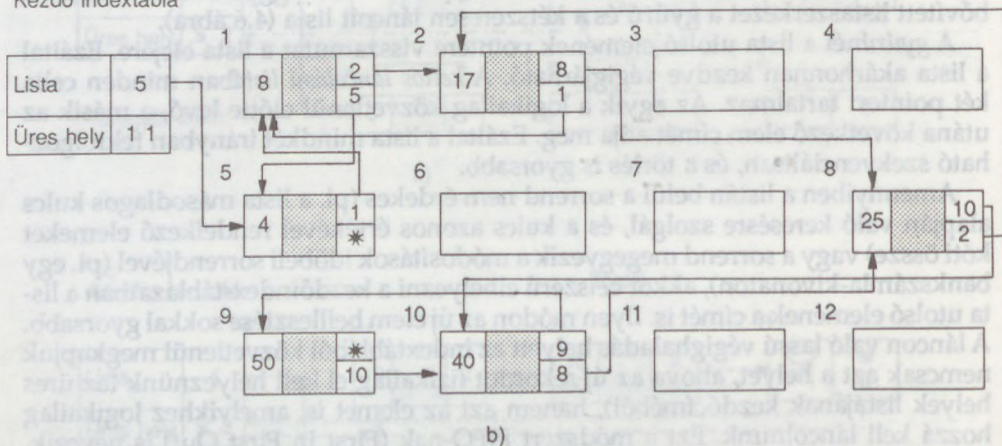
A legtöbb adatbázis-kezelő rendszer ugyanazon lapon különböző hosszúságú rekordokat is tud tárolni. Ezért nem biztos, hogy az újonnan beírandó rekord elfér egy törölt rekord helyére. Ennek megvizsgálása helyett ezek a rendszerek a törölt egy rekord helyét nem használják fel, csak számon tartják, hány van belőlük. Az új rekordot közvetlenül a meglevők mögé írják. Így a szabad helyek listájára nincs szükség, csak az első üres hely címére. Ez nagymértékben fölgyorsítja a törlést, beszúrást, módosítást, de természetesen a tárolókihasználás rovására.

³ Magyarul nemigen használják ezeket a rövidítéseket és kifejezéseket. Leginkább a „legelsőnek bement jön ki legelőször”, és tán az „Utolsó pár előre fuss!” elnevezések fedik le legjobban tartalmukat.

Kezdő indextábla



Kezdő indextábla



4.6 ábra

a) gyűrű, b) kettős láncolású lista (Az üres helyeket összekötő pointereket az áttekinthetőség érdekében nem tüntettük fel az ábrán.)

Amikor a rekordok közti „lyukak” száma meghalad egy szintet, az adatbázis-felügyelő kimentí az adatbázist vagy az erősen „lyukas részt” és most már folytonos helykihasználással visszatölti.

4.6.2 A lista és a fizikai soros szerkezet összehasonlítása

A listaszerkezet előnyei:

- Listaszerkezetben a módosítás, beszúrás egyszerűbb. Az új elemet beszúrhatjuk bármelyik üres helyre, vagy az adatok végére. A pointerekkel a szek-

venciális sorrend megtartható. Soros szervezésnél csak úgy tudjuk a sorrendet megtartani, ha a beszúrás után levő összes rekordot arrább helyezzük.

- Ugyanazon rekordtípuson többféle szekvenciális szerkezetet is létrehozhatunk. A soros szerkezet csak egyféle sorrendet enged meg.
- Szekvenciális szerkezetnél egy adott kulcs megtalálásához (vagy annak megállapításához, hogy nincs ilyen) átlagosan mindig a rekordok felét kell végignézni. Sorosnál ez csak akkor igaz, ha tudjuk, hogy pontosan egy rekordban szerepel a keresett kulcs érték. Minden más esetben az összes rekordot végig kell vizsgálni.
- Listák segítségével más adatszerkezet is megvalósítható ugyanazon az állományon. Ha a pointerlánc nem azonos mezőértékű, hanem azonos hierarchiába tartozó rekordokat köt össze (pl. a vevő törzсреkordját a megrendeléseivel), akkor ez egy hierarchikus elérési utat tesz lehetővé. Ha pedig egy mező minden lehetséges értékére fölépítünk egy olyan pointerláncot, melynek első elemét könnyen megtaláljuk, akkor ez lényegében a 4.7 pontban tárgyalandó indexelt szervezés egyik megvalósítási formája.

A listaszerkezet hátrányai:

- A feldolgozás lassúbb, különösen törlésnél, beszúrásnál. De ha a kulcsok nem sűrűsödnek, akkor a feldolgozási idő olvasásnál is a többszörösére, akár nagyságrendekkel is megnövekedhet.
- Bonyolultabb programmal valósítható csak meg. (Nem igazi hátrány, mert ezeket a programokat már régen megírták a legkülönbözőbb lehetőségekre.)
- A pointerek miatt nagyobb a tárolóigénye.

A szekvenciális feldolgozást nemcsak listaszerkezettel valósíthatjuk meg, hanem indexelt szervezéssel is. Míg a hierarchikus és hálós modellekben előszere-ttel alkalmazták a fizikai szinten a listaszerkezetet, addig a relációs modellek inkább az indexeléssel összekötött szekvenciális elérést részesítik előnyben.

4.7 Indexelés

Az adatoknak rekordokba való csoportosítása és a rekordok elhelyezési kulcsának meghatározása bizonyos fokig szabadon választható. Általában azon mező(k) szerint rendezve helyezzük el a rekordokat, mely szerint a leggyakrabban fogjuk őket rendezve visszakeresni. Ha azonban csak néhány meghatározott kulcsértékkel rendelkező rekordot akarunk kiválasztani, akkor még a viszonylag gyors, a fizikai sorrendnek megfelelő soros olvasásnak is meglehetősen rossz a hatásfoka. Nem beszélve arról, hogy igen gyakran a feladattól függ, hogy melyik mező értéke határozza meg a keresett rekordokat. Ilyenkor pedig a legtöbb esetben a szekvencialitás csak logikai sorrendet jelent. Így lehet, hogy 10 000 rekordot kell szekvencialisan 1000 vagy akár 5000 lapolvasással egybekötve beolvasnunk, hogy megkapjuk a keresett 2–3 rekordot.

Az ilyenfajta feladatok hatékony megoldására használnak *indexeket*.

4.7.1 Példa indexekre

Tekintsük a „klasszikus” adatbázis példát, amikor vannak szállítóink, áruink és megrendeléseink. Ezek mindegyike egy önálló rekordtípus. A rekordtípusokhoz tartozó rekordokat az egyszerűség kedvéért egy-egy táblázatban ábrázoljuk, ami egyébként megfelel a 7. fejezetben ismertetendő relációs modell megjelenítési formájának.

Fordítsuk figyelmünket most a MEGRENDELÉS típusú rekordokra, mely a tartalmát tekintve a következő mezőkből áll:

- SZÁLLÍTÓKÓD, amely egyértelműen azonosítja a szállítót. A szállító további jellemző adatai az ugyanezzel a kódértékkel rendelkező SZÁLLÍTÓ rekordban vannak.
- ÁRUKÓD, ami egyértelműen azonosítja a megrendelt árut. Az áru további jellemzői az ugyanezzel a kódértékkel rendelkező ÁRU rekordban találhatók.
- DÁTUM1, a megrendelés dátuma
- DÁTUM2, a szállítás dátuma
- MENNYISÉG a megrendelt mennyiség az árura jellemző (a megfelelő ÁRU rekordban tárolt) mértékegységben.

Az adatbázisból rendszeresen gyors választ kell adnunk az alábbi jellegű kérdésekre:

1. Egy adott (kódú) szállítótól milyen megrendeléseink vannak?
2. Egy adott (kódú) áruból milyen megrendeléseink vannak?
3. Egy adott napon milyen megrendeléseket adtunk fel?
4. Egy adott időintervallumban milyen megrendelések érkeznek be?

Miután a megrendelési rekordokat betöltöttük az adatbázisba, az 1. kérdésre úgy tudunk gyorsan válaszolni, ha készítünk hozzá egy két mezőből álló kiegészítő táblázatot. Ennek első mezője, és egyben kulcsa a SZÁLLÍTÓKÓD, másik mezője pedig az ehhez a kódhoz tartozó MEGRENDELÉS rekord címét tartalmazza. Ha egy SZÁLLÍTÓKÓD-hoz több MEGRENDELÉS is tartozik, akkor ez a kód annyiszor szerepel a kiegészítő táblázatban, ahányszor a megrendelésekben, és mindegyik címmel egy másik megrendelésre mutat rá (l. 4.7 ábra). (Ha egy-egy SZÁLLÍTÓKÓD-hoz nagyon sok MEGRENDELÉS tartozik, akkor másképpen is szervezhetjük azt a táblát, mint arra majd a 4.7.3 pontban kitérünk.)

A SZÁLLÍTÓKÓD értékét és a MEGRENDELÉS címét tartalmazó táblázatot *index* táblázatnak nevezzük, és azt mondjuk, hogy a megrendelés rekordok, a MEGRENDELÉS rekordtípus (táblázat) *indexelve van a SZÁLLÍTÓKÓD szerint*.

Egy adott szállítótól történő megrendeléseinket ekkor úgy kapjuk meg, hogy kikeressük az indextáblázatból a megfelelő SZÁLLÍTÓKÓD-ot és az ott talált cím(ek) alapján beolvassuk az ehhez tartozó MEGRENDELÉS rekordo(ka)t.

Ez—egész speciális, a gyakorlatban nemigen előforduló esetektől eltekintve—lényegesen gyorsabb, mintha beolvasnánk az összes MEGRENDELÉS rekordot és egyenként vizsgálnánk bennük a SZÁLLÍTÓKÓD értékét. Ez abból következik, hogy az indextáblázat nem csak szekvenciális, hanem a fizikai sorrendje is meg egyezik az olvasási sorrenddel és jóval kisebb, mint a MEGRENDELÉS. Ezen kívül, mint azt a 4.7.4 pontban látni fogjuk, az indextáblázatot nem is kell szekvenciálisan

INDEXTÁBLÁZAT
(SZÁLLÍTÓKÓDRA)INDEXELT REKORDOK
(MEGRENDELÉS)

SZÁLLÍTÓKÓD	CÍM	SZÁLL. KÓD	ÁRU KÓD	DÁTUM 1	DÁTUM 2	MENNYISÉG
17	●	17	42	920103	920310	70
17	●	41	60	920103	920310	40
19	●	17	40	920103	920416	180
41	●	50	42	920104	920215	60
50	●	50	40	920104	920317	60
50	●	19	60	920107	920916	1000

4.7 ábra MEGRENDELÉS rekordok indexelése SZÁLLÍTÓKÓD szerint

végigolvasnunk a keresett szállítókódig. Az indextáblára épített további indexek segítségével néhány lépésben megtaláljuk a keresett szállítókódot és a hozzátartozó cím(ek)et.

A 2., 3. és 4. típusú kérdések hatékony megválasztásához hasonló indexeket építünk fel a MEGRENDELÉS rekordoknak az ÁRUKÓD, a DÁTUM1, és a DÁTUM2 mezőire is.

4.7.2 Indexek

A fentieket általánosítva azt mondhatjuk, hogy az indexelt adatok (rekordtípusok, táblázatok) két részre oszthatók:

- a tényleges adatrészre,
- az indextáblázat(ok)ra, melyek az indexelt mezők minden egyes értékére tartalmazzák egy megfelelő rekord címét.

Egy rekordtípus (táblázat) bármelyik mezőjére, illetve a mezők bármilyen kombinációira építhetünk indexet. Egy rekordtípushoz több indextáblázat tartozhat.

Az elhelyezési, vagy elsődleges kulcsra épített indexet *elsődleges*, a más mezőkre épített indexeket *másodlagos* indexnek nevezzük.

Ha az indextáblázatban bármely kulcs értékhez egy és csakis egy index tartozik, azaz az adott indexértékkel rendelkező rekord legfőljebb egyszer fordulhat elő az indexelt rekordtípusban, akkor az index *egyedi*, ha több is tartozhat, akkor *duplikált*. Az elsődleges kulcsra épített index mindig egyedi.

A példánk logikájából következik, hogy a MEGRENDELÉS rekordtípus SZÁLLÍTÓKÓD, ÁRUKÓD, DÁTUM1, DÁTUM2 mezőire épített indexek mind duplikáltak. Ha viszont a szállítók gyors megtalálására a SZÁLLÍTÓ rekordtípusra is építenénk egy indexet a SZÁLLÍTÓKÓD értéke alapján, akkor ez egyedi lenne.

A MENNYISÉG mezőre fölösleges indexet építenünk, mert sem olyan kérdést, hogy miből rendeltünk 50 darabot, vagy 100 litert, sem olyat, hogy soroljuk fel a megrendeléseinket a megrendelés mennyiségének sorrendjében, várhatóan nemigen fogunk fölteni.⁴

A fentiekből azt is láthatjuk, hogy a logikailag ugyanazt jelentő mezők különböző táblázatokban különbözőképpen is szerepelhetnek indexként.

Ha az indextáblázatban a kulcsok növekvő sorrendben követik egymást, akkor az *index növekvő*, ha csökkenő sorrendben, akkor az *index csökkenő*.

Ha az indextáblázatban az értékeket csak egy mező (kulcs) értéke határozza meg, akkor az *index elemi*, ha több mező együttes értéke, akkor *összetett indexről* beszélünk.

Eddigi példáink mind elemi indexek voltak. Az összetett indexek használatára a 4.7.6 pontban mutatunk be egy jellemző példát.

Az indextáblázat segítségével rendezés nélkül is végigmehetünk szekvenciálisan az indexelt adatokon. Minden olvasás két lépésből áll. Először szekvenciális sorrendben olvasunk az indextáblázatban, majd az ott talált cím alapján kiolvassuk a hozzátartozó indexelt rekordot.

Az adatkezelő nyelvben kiadott szekvenciális olvasó utasításban (l. 5.3.3 és 6.7 pont) természetesen nem kell azzal törődnünk, hogy miként, hány lépésben zajlik le ez a művelet a fizikai szinten. Az adatbázis-kezelő rendszer mindezt elvégzi helyettünk. Természetesen az indexeket csak akkor tudja használni, ha ilyeneket az adatbázis-felügyelő definiált. Ha ezt elmulasztotta, akkor hiába volna az hatékonyabb, a rendszer nem fogja megtenni helyettünk.⁵

Index segítségével nemcsak egyedi kulcs értékekhez tartozó rekordokat kaphatunk meg, hanem például adott kulcstartományba esőket is. A 4.7.1 pont 4. kérdésére például úgy válaszolhatunk, hogy a szállítási határidő indextáblázatában a kezdő időponttól kiindulva szekvenciálisan végighaladunk a végső időpontig, és mindegyik értékhez beolvassuk a hozzátartozó megrendelést.

Bizonyos kérdésekre választ kaphatunk közvetlenül az indextáblázatokból anélkül, hogy magukat az adatokat meg kellene vizsgálnunk. Ezek elsősorban egzisztencia jellegű, vagy csak az adott feltételnek megfelelő rekordok számára vonatkozó kérdések, mint pl. hány megrendelésünk volt egy napon, van-e egy adott kódú áruból egyáltalában megrendelésünk stb. Ilyen jellegű kérdések megválaszolásánál még sokkal jobban jelentkezik az indexelt szervezés hatékonysága.

Indexek felépítése és használata meggyorsítja a keresést. Ennek azonban természetesen ára van. Nézzük meg, mivel fizetünk érte.

⁴ Olyan kérdés elképzelhető, hogy egy adott áruból a megrendelés mennyiségének sorrendjében kell kiírunk a MEGRENDELÉS rekordokat, de ilyenkor először ÁRUKÓD alapján választjuk ki a megfelelő rekordokat, és csak ezen kiválasztottakon belül érdekel a sorrend MENNYISÉG szerint. A csupán a MENNYISÉG mezőre fölépített index viszont ennél nagyságrend(ek)el több rekordra utal, így azt önmagában ennél a feladatnál sem volna értelme használni.

⁵ A legújabb relációs rendszerek közül egyesek még ezt is megcsinálják. Bizonyos esetekben maguktól készítenek ideiglenes indexeket, ha úgy ítélik meg, hogy ez meggyorsítja a feladat megoldását.

Először is az indextáblázatok helyet foglalnak el. Minél több mezőre és mező-kombinációra készítünk indexeket, annál több helyet igényelnek. Közelítésképpen azt mondhatjuk, hogy egy hatékonyan működő adatbázisban az indexterület körülbelül a fele az adatterületnek.

A másik hátrány módosításkor, törléskor, beszúráskor jelentkezik. (Módosításon itt olyan mezőérték megváltoztatását értjük, melyre index épült fel.) Ekkor ugyanis nemcsak az adatrekordon kell a megfelelő műveletet elvégezni, hanem az összes érintett indexet is aktualizálnunk kell.

Ha például egy megrendelésnél a szállítási határidőt 921016-ról 921101-re módosítjuk a MEGRENDELÉS rekordban, akkor az adatbázis-kezelő rendszer

- feljegyzi a módosított rekord címét,
- megkeresi a DÁTUM2 indextáblázatból a 921016 értékhez tartozó sorokat,
- törli ezekből a rekordcímhez tartozó sort,
- beír a DÁTUM2 indextáblázat megfelelő helyére egy új sort, melyben a kulcs értéke 921101, a címé pedig a módosított rekord címe.⁶

4.7.3 Duplikált indexek

A legtöbb másodlagos index duplikált. Megengedett, hogy ugyanazon kulcsérték több rekordban is előfordulhasson. Ha a duplikátumok száma nem túl nagy, akkor a 4.7.1 pontban bemutatott megoldás, mely szerint az indextáblázat minden egyes sorában megismétljük a kulcs értékét, tehát minden egyes indexelt adatrekord egy önálló sor az indextáblázatban, még hatékonyan alkalmazható. Ha azonban nagyon sok duplikátumunk lehet, kulcsértékekként esetleg több száz (pl. a megrendelési vagy a szállítási dátumra, ha nagyon sok szállítóval és áruval dolgozunk), akkor a kulcsok állandó megismétlése fölöslegesen növeli meg az indextáblázatunkat. Ezen kétféle módon szoktak segíteni.

Az egyik az, hogy az indextáblázat minden egyes kulcshoz csak egy címet, az első ezen értékekkel rendelkező rekord címét tartalmazza. Magában az indexelt rekordban bent van pointerként a következő, ugyanezen mezőértékű rekord címe. Az azonos indexértékhez tartozó rekordok egy láncolt listát alkotnak, melynek kezdő címe az indextáblázatban van. Így az indextáblázat csak annyi sorból áll, ahány *különböző* értéke lehet az indexelt mezőnek. Az egyes sorokhoz tartozó láncolt listák hossza megegyezik az adott kulcsértékekkel bíró adatrekordok számával.

Az indextábla méretének csökkenéséért többek között az adatrekordok hosszának és a feldolgozási időnek a növekedésével fizetünk.

A másik módszernél az indextáblázat sorai változó hosszúságúak (vagy rögzített hosszúságúak, de összeláncolhatók, ha egy kulcsértékhez túl sok rekord tartozik). A kulcs mögött annyi cím lesz, ahány adatrekordban ennyi a kulcs értéke.

⁶Vannak olyan rendszerek, melyek csak logikailag illesztik be az indextáblázat megfelelő helyére az új sort és vannak, amelyek—ha mód van rá—fizikailag is a logikai sorrendnek megfelelő helyre teszik. Ez azt jelenti, hogy a mögötte levő indexsorokat odább kell tolni. Az első esetben a feldolgozás sok változtatás után lelassul, a másodiknál az update tart hosszabb ideig.

Itt a helycsökkenésért a változó hosszúságú indexsorok kezeléséből adódó nehézségekkel fizetünk.

A fent elmondottakat a 4.8 és 4.9 ábrán látható példával illusztráljuk.

"Tantárgy" indextábla	Kurzus	Kezdő-cím	Rekordok száma
	Biológia	1400	3
	Fizika	1100	7
	Kémia	1100	6
	Matematika	1000	6
	Történelem	1400	3

"Születési év" indextábla	Születési év	Kezdő-cím	Rekordok száma
	70	1000	3
	71	1900	3
	72	1400	3
	73	2200	2
	74	1100	1
	75	*	0
	76	*	0

"Név" indextábla	Név	Kezdő-cím	Rekordok száma
	Antal Pál	1100	1
	Balogh Géza	1400	1
	Dávid János	1000	1
	Elek Gábor	1900	1
	Fekete András	2200	1
	Gábor Jenő	1700	2
	Horváth Judit	1800	1
	Molnár Éva	1600	1
	Szabó János	1300	3
	Szekeres Anna	1500	1
	Tamás Endre	1200	2
	Vámos Mária	2300	1

TÁROLÓ (adatok)

1000 Dávid János	IBIKIMI 70 ...	1100 Antal Pál	IFIKI 74 ...
1200 Tamás Endre	IFIMI 71 ...	1300 Szabó János	IFIMI 73 ...
1400 Balogh Géza	IBIKITI 72 ...	1500 Szekeres Anna	ITI 71 ...
1600 Molnár Éva	IFIMI 70 ...	1700 Gábor Jenő	IBITI 72 ...
1800 Horváth Judit	IFIMI 72 ...	1900 Elek Gábor	IKI 71 ...
2000 Szabó János (folytatás-1) ...		2100 Tamás Endre (folytatás) ...	
2200 Fekete András	IFIKIMI 73 ...	2300 Vámos Mária	IFIKI 70 ...
2400 Szabó János (folytatás-2) ...		2500 Gábor Jenő (folytatás) ...	

4.8 ábra Duplikált indexek kezelése listaszerkezettel

A 4.8 ábrán egy példát láthatunk több szempont szerint indexelt adatszerkezet-re. Egyetemi hallgatók nyilvántartásából a hallgató neve, születési éve és a kurzusok résztvevői alapján kell hallgatói rekordokat kikeresni és listákat összeállítani. (Pl. azoknak a hallgatóknak a névsora, akik fizikát tanulnak, vagy Szabó János személyi adatai stb.) Az azonos kulccsal rendelkező rekordok láncolva vannak, és az indextáblázat ezen láncolt lista első elemére mutat. Minden fizikai rekord 100 byte hosszú. Mivel az egyes hallgatók adatai ennél hosszabbak lehetnek, egy hallgatói logikai rekordhoz több fizikai rekord is tartozhat.

Minden hallgatói rekord három kötelező kulcsmezővel kezdődik. Az első a hallgató nevét, a második a hallgatott kurzusok rövidítését, míg a harmadik a születési évének utolsó két számjegyét tartalmazza.⁷ Az első és a harmadik mező elemi, míg a második csoportmező, melynek hosszát a rekord valamilyen—számunkra most nem lényeges—formában tartalmazza. Mindegyik kulcsmezőhöz tartozik még egy pointer is, mely a következő azonos kulcsértékű rekordra mutat. A kulcs után különböző adatok (lakcím, korábbi iskolák, osztályzatok stb.) vannak, melyek már nem szolgálnak keresésre.

Az alábbi egyszerű példán bemutatjuk, miként nyerhetünk vissza különféle információkat ebből az adatbázisból.

Ahhoz, hogy a Szabó János nevű hallgatóról minden adatot megkaphassunk, először a rendezett hallgatói névindexet vizsgáljuk végig. Ebből megkapjuk, hogy Szabó János első rekordja az 1300-as pozíciójú tárolóhelyen van. (Az egyszerűség kedvéért a tárolóhelyeket sorszámoztuk. A gyakorlatban ehelyett vagy tényleges cilinder-, sáv- és blokk- vagy rekordszámot adunk meg, vagy valamilyen algoritmussal átszámoljuk tényleges címmé.) Elindulva az 1300-as címtől, Szabó János összes adatát megkaphatjuk, miután a névmezőben elhelyezett pointerek alapján beolvastuk először a 2000, majd a 2400-as helyen levő rekordot (l. 4.8 ábra).

Ha a fizika kurzusra járó hallgatók névsorára vagyunk kíváncsiak, akkor az ehhez tartozó indextáblát keressük ki a tantárgy-indextáblából. Ebből megkapjuk, hogy az első hallgató adatai az 1100-as helyen vannak. Innen megtudhatjuk nevét (Antal) és egyéb adatait is. A fizika kurzuson keresztülhaladó lista alapján eljutunk a kurzus összes többi hallgatójához is. (Az 2200, 1800, 1600, 1300, 1200 és 2300 címtől kezdődő Fekete, Horváth, Molnár, Szabó, Tamás és Vámos.)

Ha az 1972-ben született hallgatók adataira vagyunk kíváncsiak, akkor a „születési év” indexek alapján először az 1400-as helyen levő rekordhoz (Balogh) jutunk. Ehhez csatlakozik a lista alapján az 1700 és 1800-as címen tárolt Gábor és Horváth.

A 4.9 ábra ugyanezen adatokhoz tartozó indextáblázatokat mutat, de a listák kezdete helyett az összes azonos kulcsértékű rekord címe bent van a táblázatban. Ebben az esetben a hallgatói rekordokban felesleges pointereket elhelyezni. Amennyiben valamelyik indextáblázat bejegyzésében már nincs több hely egy új rekord címének beírására, akkor egy új bejegyzést láncolunk a betelthez, és ebbe helyezzük el a további címeket (lásd az ábrán a fizika tantárgy hallgatóinak adatait). A 4.8 ábrán az azonos kulcsértékhez tartozó rekordok a hallgatók nevei sze-

⁷ Megjegyezzük, hogy a relációs modellben ilyen módon közvetlenül nem tudnánk az adatokat ábrázolni.

"TANTÁRGY" INDEXTÁBLA

Kurszus	Rekordcímek						Lánc
							0
Biológia	1000	1400	1700	*			1
Fizika	1100	1200	1300	1600	1800	2200	0
Kémia	1000	1100	1400	1900	2200	2300	0
Matematika	1000	1200	1300	1600	1800	2200	0
Történelem	1400	1500	1700	*			0
Fizika	2300	*					

"NÉV" INDEXTÁBLA

Név	Rekordcímek						Lánc
							0
Antal Pál	1100	*					0
Balogh Géza	1400	*					0
Dávid János	1000	*					0
Elek Gábor	1900	*					0
Fekete András	2200	*					0
Gábor Jenő	1700	2500	*				0
Horváth Judit	1800	*					0
Molnár Éva	1600	*					0
Szabó János	1300	2000	2400	*			0
Szekeres Anna	1500	*					0
Tamás Endre	1200	2100	*				0
Vámos Mária	2300	*					

"SZÜLETÉSI ÉV" INDEXTÁBLA

Születési év	Rekordcímek						Lánc
							0
70	1000	1600	2300	*			0
71	1200	1500	1900	*			0
72	1400	1700	1800	*			0
73	1300	2200	*				0
74	1100	*					0
75	*						0
76	*						

4.9 ábra Duplikált indexek kezelése változó hosszúságú sorokból álló indextáblázatokkal. Az adatok elhelyezését a 4.8 ábrán láthatjuk. A * speciális jel a rekordcímek végét jelzi, a 0-tól különböző érték a lánc oszlopban a következő azonos kulcsértékkel rendelkező rekord relatív címét adja az indextáblázatban.

rint ábécé sorrendben rendezve szerepelnek a láncolt listában, de rendezetlenek a 4.9 ábra indextáblájában. Az indextábla-bejegyzésekben a logikai és a fizikai sorrend mindkét ábrán ugyanaz. Ezek a megkötések azonban minden további nélkül elhagyhatók. Természetesen a rendezettség meggyorsítja az indextáblában való keresést, de ugyanakkor nehezkesebbé teszi új rekordoknak a beillesztését. Éppen ezért az indextáblázat maga is lehet egy láncolt lista, sőt, kevés elem esetén rendezetlen is lehet.

A relációs modellben a fenti duplikátumkezelési módszereket—annak ellenére, hogy nagyszámú duplikátumnál hatékonyak—ritkán alkalmazzák, mivel nehéz őket beilleszteni a modell koncepciójából adódó feltételek közé.

4.7.4 Keresés indextáblában

Az indexek meggyorsítják a rekordok visszakeresését. Azáltal, hogy egy vagy két lépésben kapjuk meg a keresett rekordot tartalmazó lapot, az író/olvasó műveletek számát csökkentjük. Azzal, hogy a memóriába már beolvasott lapon hamarabb találjuk meg a rekordot, a feldolgozáshoz szükséges processzor idő lesz kevesebb. Mivel a két idő közt általában közel 3–4 nagyságrend különbség van, legfontosabb a lapváltási író/olvasó időt minimalizálni. (Igaz, hogy egy multiprogramozási gép ezalatt más feladaton is dolgozhat, de ez csak annak a másik feladatnak a megoldását gyorsítja meg. A mienk futási idejére nincsen döntő hatással.)

Nagy adatbázisokban az indextáblázatban is sok bejegyzés van. Ezért már egy adott indexnek és a hozzátartozó címnek a megtalálása is több lap beolvasását, és ezen belül is hosszabb keresést eredményezhet.

Ezért többféle módszert dolgoztak ki, amelyekkel magában az indextáblázatban tudunk hatékonyan keresni. Ezek közül ebben a pontban az *indexelt szekvenciális szervezést* mutatjuk be, mert ez lehetővé teszi ugyanazon adatoknak mind direkt, mind szekvenciális módon való gyors elérését. Ennek azonban előfeltétele, hogy az elhelyezési kulcs sorrendje megegyezzek, vagy legalábbis nagyjából megegyezzenek a keresési kulcs szerinti sorrenddel. A kiegyensúlyozott fát (B^+ fa), melynél a közel azonos rendezettségnek nem kell teljesülni, tehát bármilyen módon tárolt lemezállományra alkalmazható, a 4.7.5 pontban ismertetjük.

Az indexszekvenciális állomány rekordjai növekvő kulcs szerint rendezve kerülnek fel az állomány létrehozásakor a tárolóra. Az állomány az adatok mellett indexterületeket is tartalmaz, mely a közvetlen elérésre szolgál. Az indexszekvenciális adatszervezési módra egyszerű, de jó példa a telefonkönyv. Az első index a könyv oldalán levő betű, mely megmutatja, hogy az ezzel a betűvel kezdődő nevek melyik oldalon találhatók. Mivel ez jól látható módon elkülönül a tényleges információtól, a nevektől és a telefonszámoktól, azért pl. Tóth Ferenc telefonszámának keresésekor a könyvet azonnal ki tudjuk nyitni a T betűnél, és így legrosszabb esetben a T-vel kezdődő neveket kell sorban végigolvasni. Mivel ezek száma is meglehetősen nagy lehet, ezért a keresés meggyorsítására még egy index, a lap-index szolgál.

Ez is elkülönül a tényleges információtól. Mindig a lap tetején található, és megadja például az azon a lapon levő utolsó nevet. Ezért a keresést a T betűs lapokon célszerűen úgy folytatjuk, hogy a teljes névsor helyett csupán ezt a lap-indexet nézzük, és amikor ennél eljutunk a Tóth névig, vagy túljutunk rajta, akkor kezdjük el az ehhez tartozó oldalon a nevek sorban való végigolvasását. Így az indexszekvenciális állományban közvetlen módon is tudunk keresni.

Miután az új rekordokat nem mindig tudjuk fizikailag beilleszteni a kívánt helyre, számukra, ha szükséges, egy külön területet kell biztosítani. Ezt túlsor-dulási területnek nevezzük. Ennek alapján minden indexszekvenciális állomány

három elkülönülő részből áll, melyek méretét a létrehozásakor állapítjuk meg, és utána már rendszerint csak újraszervezéssel változtatható meg a nagyságuk. Ezek

- az elsődleges adatterület,
- a túlsordulási terület és
- az indexterület.

Az *elsődleges adatterület* tartalmazza az állomány létrehozásakor beírt adatokat növekvő kulcs szerint. Ha új adatok beillesztése nem valószínű, akkor a hatékony helykihasználás érdekében ezt a területet teljesen feltöltjük adatokkal már az állomány létrehozásakor. Ha azonban sok új rekord hozzáfűzése várható, akkor lehetséges, hogy kezdetben ennek a területnek is csak bizonyos részét töltjük fel (pl. minden lapnak csak a 60%-át), és így az új rekordok (legalábbis kezdetben) e területre kerülnek. Ez meggyorsítja a feldolgozást.

A *túlsordulási területek* tartalmazzák azokat a rekordokat, melyek a kulcsuk alapján kijelölt elsődleges adatterületre nem fértek el a már ott levő rekordok miatt. A rekordok láncolt listaszervezésben helyezkednek el. A lista első elemére abból a lapból történik a hivatkozás, melyhez a túlsordult rekordok tartoznak.

Az *indexterületek* teszik lehetővé, hogy a rekordokat gyorsan írhasunk be az adatbázis megfelelő helyére, illetve gyorsan olvashassuk ki.

Az indexeknek az indexszekvenciális szervezésben több szintje lehet.

Az alapvető index a lapindex.⁸ Ennek kulcsértéke az azon a lapon levő legmagasabb értékűvel egyezik meg, pointer része pedig a lap címét adja. Ezeket az indexértékeket növekvő sorrendben tároljuk az indexterület lapjain (4.10 ábra). Az adatterületek szabadon hagyott részei üresek a későbbi bővítések befogadására.⁹ Ha az indextáblázatbeli kulcsok értéke növekvő sorrendben $ik_1, ik_2, \dots, ik_n$, a hozzájuk tartozó lapcímek pedig $c_1, c_2, \dots, c_n$, akkor szekvenciális feldolgozásnál

- a) beolvassuk az indextábla első elemét,
- b) az ott levő cím által megadott lap összes rekordját sorban feldolgozzuk,
- c) amíg az indextábla végére nem értünk, beolvassuk az indextábla következő elemét és a hozzátartozó címen levő lap összes rekordját szekvenciálisan feldolgozzuk.

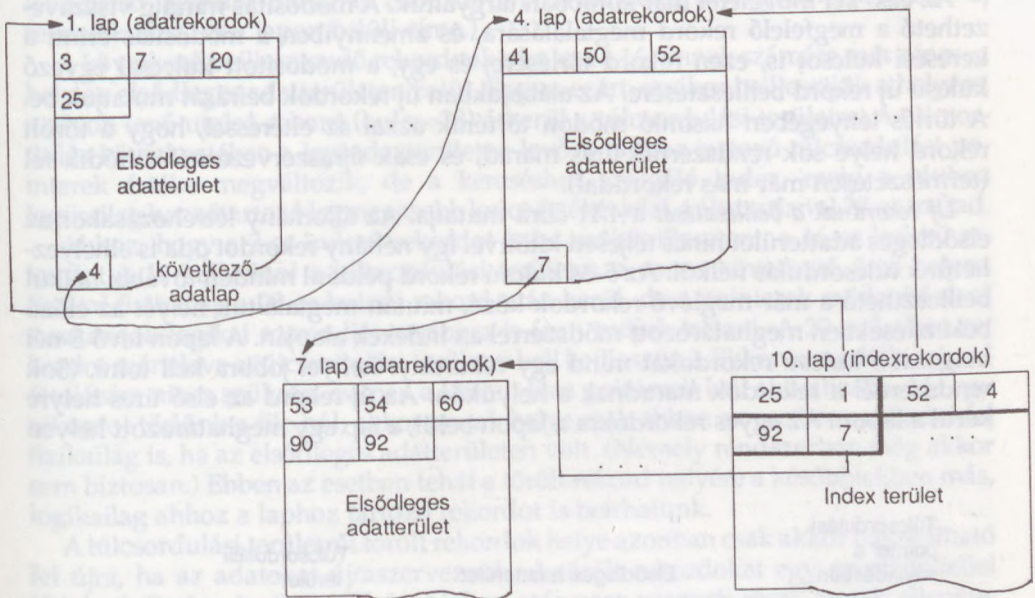
Direkt keresésnél sorban végigolvassuk az indextáblázatot, amíg $ik_{j-1} < k \leq ik_j$, ahol a k a keresett rekord kulcsa. Az a rekord a c_j lapon lesz.¹⁰ Így azt beolvassuk, és azon szekvenciálisan megkeressük a kívánt rekordot. A 4.10 ábrán látható példán mondjuk az 50-es kulcsú rekordot keresve, az indextáblázatból megállapítjuk, hogy ez a kulcs a 25-ös és 52-es indexek közé esik, így az 52-es maximális indexet tartalmazó 4. lapon lesz.

A lemezpozicionális idő csökkentése érdekében az indexlap általában ugyanazon a cilinderen van, mint az indexelt rekordok.

⁸ Régebbi rendszerekben ez lényegében megfelelt a sávindexnek, az előlotti szint pedig a cylinder indexekhez hasonlítható.

⁹ Egyes adatbázis-kezelő rendszerek megengedik, hogy ugyanazon a lapon különböző típusú rekordok is lehessenek.

¹⁰ Vagy ha a bővítések miatt fizikailag túlsordulási területre került, akkor is a pointerekkel ehhez a laphoz kapcsolódik. Ezért azt mondjuk, hogy logikailag az így kiválasztott lapon van.



4.10 ábra Indexszekvenciális szervezés. Az indexek a 10., az adatrekordok a pointerlánccal szekvenciális sorrendbe rendezett 1., 4. és 7. lapon vannak. Minden indexérték a pointermező címén levő lapon tárolt legmagasabb indexszel egyenlő

Ha olyan sok lapindexünk van, hogy azok csak több lapon férnek el, akkor felépíthetünk egy újabb indexszintet, egy lapindex-index táblázatot is. Ezek bejegyzései az egyes indexlapokon lévő legmagasabb indexértékek, és ezen indexlapok címei. Ezt a táblázatot gyakran fő (master) indextáblának is nevezik.

Ha például az indexlapok címei $ic_1, ic_2, \dots, ic_m$, a rajtuk levő legmagasabb indexek rendre

$$ik_{i_1}, ik_{i_2}, \dots, ik_{i_m},$$

akkor a fő index elemei

$$(ik_{i_1}, ic_1); (ik_{i_2}, ic_2); \dots (ik_{i_m}, ic_m).$$

A keresés elve ugyanaz, mint az egyszintű indextáblázatoknál. Csak itt először a főindex-táblából választjuk ki, melyik indextábla lapot olvassuk be, hogy abban megtaláljuk a keresett rekordot tartalmazó lap címét.

Az indexszekvenciális állomány létrehozásakor az adatokat a kulcs szerint töltjük be az adatbázisba. Ekkor a rendszer az adatoknak a lapokon való elhelyezése mellett elkészíti az indextáblázatot is. Ezután a rekordokon az alábbi műveletet végezhetjük:

- kulcs szerinti szekvenciális feldolgozás,
- kulcs szerinti közvetlen keresés és feldolgozás,
- új rekordok beírása, régiak törlése,
- meglévő rekordok módosítása.

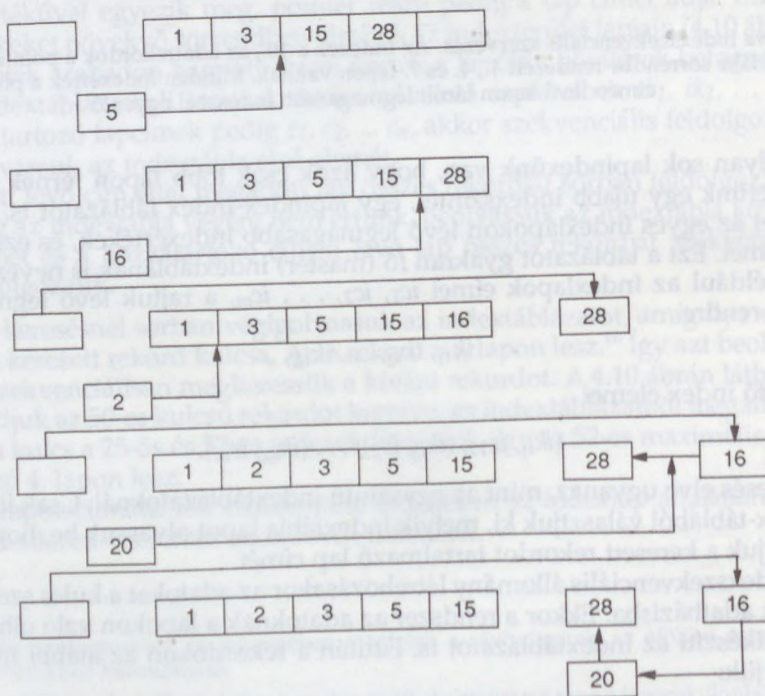
Az első két műveletet már korábban tárgyaltuk. A módosítás mindig visszavezethető a megfelelő rekord megtalálására, és amennyiben a módosítás érinti a keresési kulcsot is, ezen rekord törlésére, és egy, a módosított kulccsal egyező kulcsú új rekord beillesztésére. Az alábbiakban új rekordok beírását mutatjuk be. A törlés lényegében hasonló módon történik azzal az eltéréssel, hogy a törölt rekord helye sok rendszernél üres marad, és csak újraszervezésnél töltődik fel (természetesen már más rekorddal).

Új rekordnak a beillesztését a 4.11 ábra mutatja. Az állomány létrehozásakor az elsődleges adatterület nincs teljesen kitöltve. Így néhány rekordot oda is elhelyezhetünk túlsordulás nélkül. Az 5-ös kulcsú rekord például minden további nélkül beilleszthető a már meglévő rekordok közé, miután megtaláltuk helyét az előző bekezdésekben meghatározott módszerrel az indexek alapján. A lapon levő 5-nél magasabb kulcsú rekordokat mind egy rekordnyi hellyel jobbra kell tolni. (Sok rendszernél a rekordok maradnak a helyükön. Az új rekord az első üres helyre kerül a lapon. Az egyes rekordokra a lapon belül, a lap egy meghatározott helyén

Túlsordulási
pointer a
sávindexben

Elsődleges adatterület

Túlsordulási
terület



4.11 ábra Új rekord beillesztése indexszekvenciális szerkezetbe

levő belső lapindex mutat, és ebbe kerül be a fizikailag is megfelelő helyre az új rekord kulcsa, és a lapon belüli címe.)

A következő beillesztendő rekordnak a kulcsa a 16. Ennek számára már nincsen hely az elsődleges adatterületen belül. Éppen ezért, amikor beillesztjük a helyére, a sávon levő utolsó rekord (kulcs=28) átkerül a túlcordulási területre. A túlcordulás következtében a lapindexterületen levő, e laphoz tartozó túlcordulási pointerek értéke megváltozik, de a kereséshez szolgáló index, mely a sávhoz logikailag hozzátartozó legmagasabb kulcs értékét jelzi, változatlanul 28-as marad.

Ahhoz, hogy a 2-es kulcsú rekordot is be tudjuk illeszteni, a 16-os kulcsú rekordot is ki kell vinni a túlcordulási területre, a soron következő üres helyre. Ezáltal fizikailag a 28-as kulcsú rekord után kerül, de a pointerek módosításával a rekordok logikai sorrendjét továbbra is fenn tudjuk tartani. A 20-as kulcsú rekordot már eleve a túlcordulási területre kell beilleszteni. Ekkor semmiféle rekord áttolására nincs szükség, csupán a láncolt lista pointerait kell aktualizálni. Ha egy rekordot törölünk a file-ból, akkor annak helye csak abban az esetben szabadul fel fizikailag is, ha az elsődleges adatterületen volt. (Némely rendszerben még akkor sem biztosan.) Ebben az esetben tehát a törölt rekord helyére a későbbiekben más, logikailag ahhoz a laphoz tartozó rekordot is beírhatunk.

A túlcordulási területről törölt rekordok helye azonban csak akkor használható fel újra, ha az adatokat újraszervezzük. A törölt rekordokat egy speciális jellel látjuk el. Ezek a logikai feldolgozásban már nem vesznek részt, annak ellenére, hogy fizikailag még léteznek. Figyeljük meg, hogy az indextáblára bevitt keresési kulcsok az állomány újraszervezéséig nem változnak meg.

Ennélfogva, ha például a 29-es kulcsszámú rekordot kell bevinni, akkor az mindenképpen a második lapra kerül, még akkor is, ha az első lapon van fizikailag hely, és a másodikon már nincs.

Megjegyezzük, hogy mind a szekvenciális, mind a direkt feldolgozásnál, ha túlcordulás van, akkor a túlcordult rekordokat a rendszer automatikusan megtalálja. Természetesen ez lényegesen megnöveli a feldolgozási, elérési időt, főleg akkor, ha a túlcordult rekordok más lapra kerülnek. Ezért az adatbázis-kezelő rendszer nyilvántartja a különböző típusú túlcordult rekordok számát.

Az állomány újraszervezésével visszaállíthatjuk az eredeti hozzáférési időket. A rekordokat a kulcs szerint kimentjük, majd ebből a mentésből, mint inputból, újra előállítjuk az indexszekvenciális állományt. Ez most egyetlen túlcordult rekordot sem fog tartalmazni.

4.7.5 B⁺ fa

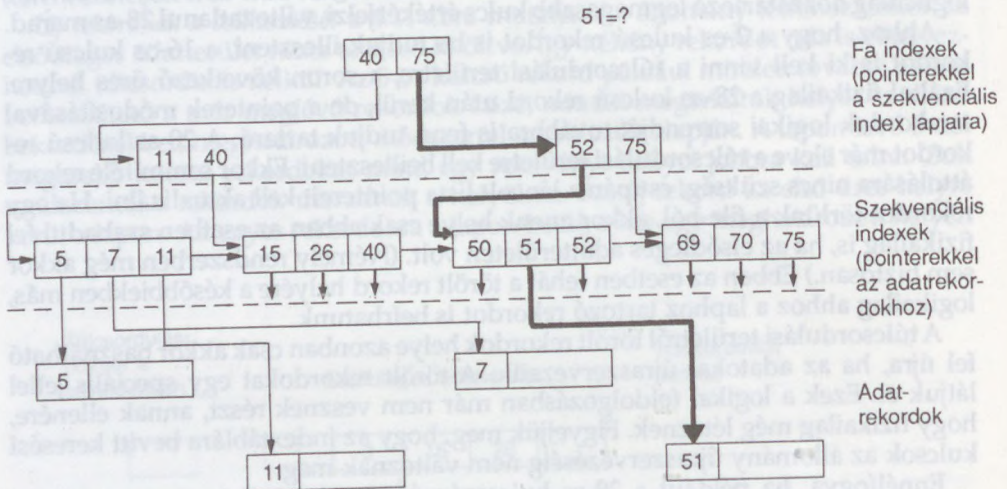
Az indexelt szekvenciális elérést a relációs modellen alapuló rendszerekben leggyakrabban a B⁺ fa segítségével valósítják meg. Ez lehetővé teszi a hatékony szekvenciális és direkt elérést, tehát rendelkezik az indexszekvenciális szervezés előnyeivel, de ugyanakkor a túlcordult rekordok nem rontják a hozzáférés hatékonyságát, és bármilyen fizikai sorrendben elhelyezkedő adatokon alkalmazható.

A B⁺ fa szerkezetnél az indexterület két részre oszlik:

1. Szekvenciális indexek, melyek megfelelnek a leírt indextáblának. Az egyes rekordok kulcsait és címüket tartalmazza, kulcsértékek szerint rendezetten. Szekven-

ciális elérésnél ezeken, illetve az innen kapott lapokon haladunk sorba. A szekvenciális indexeket tartalmazó lapokat a megfelelő sorrendben való végighaladás érdekében láncolt lista köti össze.

2. B^+ fa indexek, melyek a gyors, közvetlen hozzáférést teszik lehetővé. Ez a gyakorlatban egy többszintes faszerkezet, melynek elvi működését a 4.12 ábrán levő példán magyarázunk el.



4.12 ábra B^+ fa szerkezete

Az egyszerűség kedvéért csak egy rekordtípus egy mezőjére vizsgáljuk az indexek felépítését. (A többi rekordtípusra és mezőre az elmondottak egyenként ugyanígy alkalmazhatók.)

A fa legfelső szintjén n_1 indexhely (a példában kettő, 40-es és 75-ös értékkel) van. Az ezekhez tartozó pointerek a fa következő szintjén levő indexhelyekre (általában indexlapokra) mutatnak, melyekben további indexek és címek mutatnak a fa egyre mélyebb szintjeire, míg végül el nem érünk a szekvenciális indexeket tartalmazó lapokra. Az egyes faindex szinteken egy csoportba (általában lapra) $n_1, n_2, n_3, \dots$ indexet és címet helyezhetünk el (általában $n_1=n_2=n_3=\dots$) a szekvenciális indexnél pedig n darabot. (A példában az áttekinthetőség kedvéért $n_1=n_2=2, n=3$. A gyakorlatban azonban sokkal több index fér el egy lapon.) A cím mindig a faszerkezet eggyel alacsonyabb szinten levő lapjára mutat, és az index megadja az alacsonyabb szintű lapon található legmagasabb index értékét.

A példában pl. a második szinten két lap van. Az első lapon két indexet találunk. Ezek értéke 11 illetve 40, mert az első szekvenciális lapon a legnagyobb index 11, a másodikon 40. A második szinten levő indexekhez tartozó címek a szekvenciális indexek első, illetve második lapjára mutatnak. Hasonló módon kerül a második szint második lapjába az 52-es és 75-ös index. Az első szinten levő indexek rendre a második szint lapjainak legmagasabb indexeire utalnak (40-es az első, 75-ös a második lapra).

Az 51-es kulcsú rekord megkeresése az alábbi módon történik:

- 1.) A legfelső szint indexeit megvizsgálva $40 < 51 < 75$, ezért a következő lépésben a 75-höz tartozó címen található indexlapot olvassuk be.
- 2.) Itt $51 < 52$ (már az első indexnél kisebb), ezért az 52-es indexhez tartozó címen levő lapot olvassuk be.
- 3.) Ezzel eljutottunk a szekvenciális indexekhez. A kijelölt indexlapot sorban végigolvasva megtaláljuk az 51-es kulcshoz tartozó címet, ahonnan behozzuk a rekordot. (Amennyiben az 51-es index nincs ezen a lapon, akkor a fenti rekord-típusból ilyen kulcsú rekord nincs az adatbázisban.)

A 4.12 ábrán látható konkrét tárolási struktúra nagyon áttekinthető, de erősen idealizált. A gyakorlatban szinte sohasem tudunk egy fát ilyen szimmetrikusan felépíteni, és egyenletesen feltölteni adatokkal. Példánkban is azonnal bajba kerülünk, ha be kellene illeszteni az adatbázisba, mondjuk a 17-es kulcsú rekordot úgy, hogy az indexszerkezetet is megtartsuk.

Természetesen azért egy ilyen feladat megoldása nem okoz gondot egy jó rendszernek.

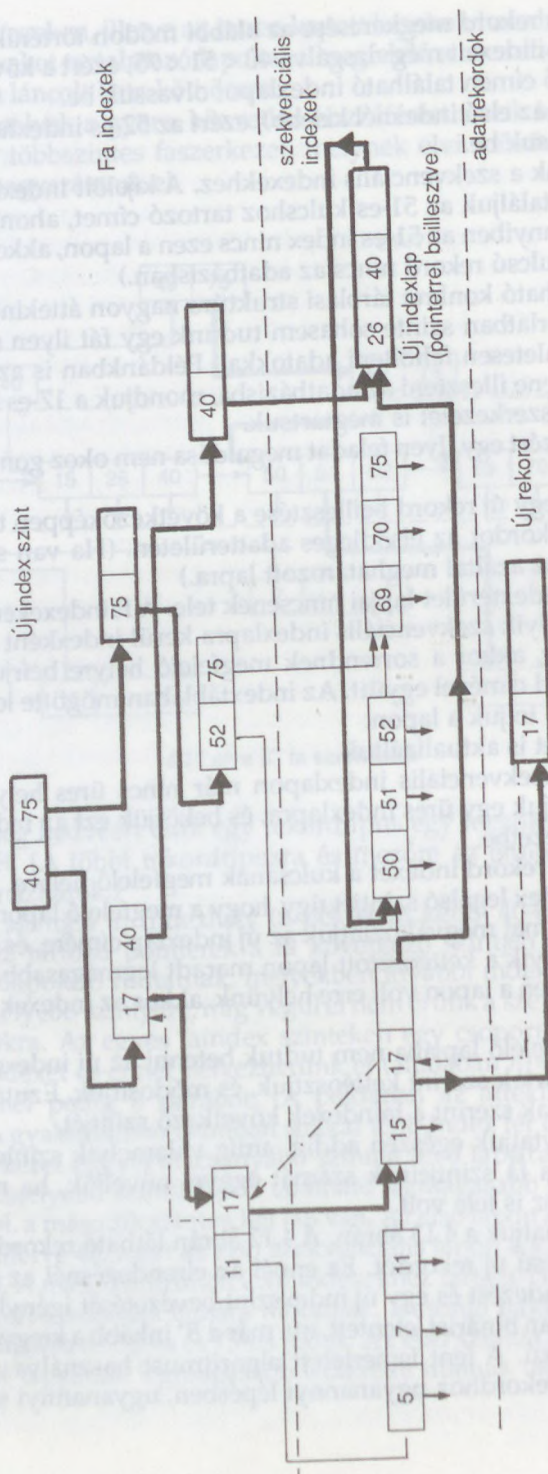
A gyakorlatban egy új rekord beillesztése a következőképpen történik:

1. Elhelyezzük a rekordot az elsődleges adatterületen. (Ha van sűrűsödő kulcs, akkor lehetőleg az azáltal meghatározott lapra.)
2. Legtöbbször az indexterület lapjai nincsenek tele. A faindexeken végighaladva megkeressük, melyik szekvenciális indexlapra kerül indexként az új kulcs. Ha ott van üres hely, akkor a sorrendnek megfelelő helyre beírjuk, az újonnan behelyezett rekord címével együtt. Az indextáblában mögötte levő bejegyzéseket eggyel följebb toljuk a lapon. Ezzel az indexeket is aktualizáltuk.
3. Ha a megfelelő szekvenciális indexlapon már nincs üres hely, akkor annak második felét átírjuk egy üres indexlapra, és bekötjük ezt az indexlapokat sorbakötő listaszerkezetbe.
4. Elhelyezzük az új rekord indexét a kulcsának megfelelő helyre.
5. Módosítjuk a faindex legalsó szintjét úgy, hogy a megfelelő lapon a kettéosztott lap indexében a címet megváltoztatjuk az új indexlap címére, és beteszünk egy új indexet is, amelyik a kettéosztott lapon maradt legmagasabb kulcsot tartalmazza. Amennyiben a lapon volt erre helyünk, akkor az indexek aktualizálását befejeztük.
6. Ha a faindex megfelelő lapjába nem tudtuk betenni az új indexet, akkor azt a 2–4. pontokban leírtak szerint kettéosztjuk, és módosítjuk. Ezután módosítjuk az 5. pontban leírtak szerint a faindexek következő szintjét.

Ezt az eljárást folytatjuk egészen addig, amíg valamelyik szinten üres helyet nem találunk, vagy a fa szintjeinek számát eggyel növeljük, ha már a legfelső szintre jutottunk, és ez is tele volt.

A fenti eljárást láthatjuk a 4.13 ábrán. A 4.12 ábrán látható rekordokhoz be kell illeszteni a 17-es kulcsú új rekordot. Ez ennél az elrendezésnél az indextáblázat minden szintjén átrendezést és egy új indexszint bevezetését igényli.

A fában a B korábban binárist jelentett, ma már a B^+ inkább a kiegyensúlyozottal (balanced) egyenértékű. A fent ismertetett algoritmust használva a fa sohasem borul föl. Bármelyik rekordhoz ugyanannyi lépésben, ugyanannyi szinten végig-



4.13 ábra Új rekord beillesztése B^+ fa szerkezetbe és az indexek aktualizálása.
Az új lapokat és kapcsolatokat vastagítva rajzoltuk be

haladva jutunk el, és az egyes indexlapok telítettsége sem nagyon tér el, mind-egyikük legalább félig tele van.

4.7.6 Keresés több kulcs szerint

Ha több kulcs együttes értéke alapján akarunk kiválasztani rekordokat, akkor erre többféle lehetőségünk van. Amennyiben a kulcsokra indexek vannak, akkor vagy mi választhatunk ezek közül, melyeket használjuk (pl. navigálunk a hálós modellben), vagy az adatbázis-kezelő rendszer maga választja ki az adatokról a katalógusban rendelkezésre álló információk alapján a várható leghatékonyabb utat (relációs módszer). Természetesen ahhoz, hogy a különböző elérési utak között választási lehetőségünk legyen, szükséges, hogy a rekordokra különféle indexeket építsünk fel. Ezeket az indexeket azonban csak akkor hozza létre az adatbázis-kezelő rendszer, ha az adatbázis-felügyelőtől erre az adatleíró nyelvben utasítást kap.

Éppen ezért célszerű megismernünk, milyen lehetőségeink vannak több kulcsra együttesen vonatkozó feltételek teljesülése alapján történő feldolgozás esetén. A módszer lényegét egy egyszerű példán mutatjuk be.

Egy személyi nyilvántartási rendszerben 100 000 rekordunk van, melyben három kulcs (kor, iskolai végzettség, foglalkozás) szerint bármelyik kulcs értéke alapján, valamint a három kulcs együttes értékének minden lehetséges kombinációja alapján keresünk vissza. Az egyszerűség kedvéért tételezzük fel, hogy minden kulcs 10 különféle értéket vehet fel ugyanolyan valószínűséggel. E feltételek mellett bármelyik kulcsértékhez 10 000 adatrekord tartozik, a három kulcs bármely kombinációjához pedig 100. Ahhoz, hogy mind a három tulajdonság alapján önállóan keresni tudjunk, három indextáblázatot kell felállítanunk egyenként 10 kulcs-értékkel és 10 000 hozzá tartozó címmel. Ez összesen 300 000 címet jelent, ami nyilván nem fér el a tárolóban. Ezért az indextáblázatok is lemezen tároljuk.

Tegyük fel továbbá, hogy egy adatbázis lapra 10 adatrekord, vagy 100 indexrekord fér el. Minden indexbejegyzésben egy rekord címe van. Az egyszerűség kedvéért tételezzük fel azt is, hogy ha egy új lapon keresünk valamit, akkor azt a lapot mindig be kell olvasni. (Amennyiben ez nem áll fenn, úgy a lapolvasásokra más értékeket kapnánk, de a különböző módszerek I/O műveleteinek aránya lényegében nem változna.) Ezen kikötések mellett adatbázisunk 10 000 adatlapból, és ha mind a három mezőre van index, akkor 3 000 indexlapból áll.

Keressünk például rekordokat az alábbi három mező értéke alapján: KOR=30 (ez az értelmezés szerint a 30–34 éves korosztályt jelenti), VÉGZETTSÉG=egyetem, és FOGLALKOZÁS=közgazda. Csak azokkal a rekordokkal dolgozzunk tovább, melyekre mind a három feltétel teljesül. Választási lehetőségeink a következők:

1. Ha egyik mezőre sincsen index, akkor végig kell olvasnunk a személyi adatokat tartalmazó összes lapot, és minden egyes rekordot külön meg kell vizsgálnunk, megfelel-e a feltételeknek. Ez összesen 10 000 lapolvasást jelent. (Szerencsére ezeket a lapokat a fizikai sorrendnek megfelelően olvashatjuk.)
2. Egy vagy több keresési kulcshoz tartozik index. Végigmegyünk a sűrűsödő kulcsú indextáblázat megfelelő értéke szerint (pl. a VÉGZETTSÉG=egyetem

index címei alapján). Beolvassuk a rekordokat, és megnézzük egyenként, megfelelnek-e a másik két feltételnek is.

Ebben az esetben a 10 000 lapot elfoglaló 100 000 tételes nyilvántartásunk megnő, ha mindhárom mezőre van index $3 \times 1\,000 = 3\,000$ lappal, ha csak egyre (sűrűsödő), akkor 1 000 lappal az indextáblázatok következtében.

A lapolvasások száma 1100, ami az „egyetem” értékekhez tartozó 100 indexlap ($100 \times 100 = 10\,000$ indexkulcs), és — mivel a kulcsok a beolvasási index szerint sűrűsödnek —, $10\,000/10 = 1\,000$ ezen értékhez tartozó rekord beolvasásából tevődik össze.

Jelen példánkban, mivel minden kulcshoz nagyon sok cím tartozik, ha a keresési kulcs közül egyik sem sűrűsödő, akkor lehet, hogy gyorsabb az elsőként említett soros végigolvasás.

Sűrűsödő, de nem túl szelektív kulcs helyett (minden kulcsértékhez sok rekord tartozik) sokszor célszerűbb lehet egy nem sűrűsödő, de erősen szelektív kulcshoz tartozó index alapján beolvasni egyenként a rekordokat, hogy aztán mindegyikről külön megállapíthassuk, kielégítik-e a többi feltételt is.

Ha a fenti példában az adatbázisban csak 500 közigazda lenne, akkor a FOG-LALKOZÁS-hoz tartozó index szerint olvasva csak 5 index- és legföljebb 500 adatlapot kellene beolvasnunk.

Az, hogy egy index sűrűsödőnek volt-e deklarálva, és hogy a módosítások, bővítések miatt ezt a tulajdonságát meg is tartotta-e, valamint az, hogy mennyire szelektív egy konkrét értékre vagy átlagosan mennyire lehet szelektív, a katalógusból megállapítható vagy legalább is megbecsülhető. Ennek alapján navigáláskor a felhasználó, illetve a relációs modellben a rendszer ki tudja választani a (várhatóan) optimális elérési utat.

3. A három kulcs együttes értéke szerinti kereséshez (pl. annak megállapítására, hogy milyen egyéb adatai vannak a 30–35 év közti, egyetemet végzett közigazdászoknak) ki kell választani a „kor” indextábla „30–35 év között”, az „iskolai végzettség” indextábla „egyetem” és a „foglalkozás” indextábla „közigazda” indexrekordjainak közös elemeit. Amennyiben mindegyik indextáblázatban ugyanazon elv szerint rendezettek az adatrekordok elemei, úgy ehhez elegendő mindegyik indextáblázat megfelelő rekordjait egyszer végigolvasni. Így a felsorolt három feltételt egyidejűleg kielégítő 100 adatrekord megtalálásához $3 \times 100 = 300$ indexlapot, és (legföljebb) 100 adatlapot kell végigolvasnunk, tehát 400-szor kell a lemezhez fordulnunk.

Nagymértékben meggyorsíthatjuk a több feltétel szerinti feldolgozást anélkül, hogy jelentősen megnövekedne az adatbázis tárolószükséglete, ha egy külön indextáblát készítünk a három tulajdonság mindegyik kombinációjához. Ezek száma $10^3 = 1\,000$. Mindegyik bejegyzés 100 adatrekord címét tartalmazza, tehát egy kombináció címei elérnek egy indexlapon.

10 000 adatlap és 3 000 indexlap mellett az 1 000 további indexlap nem jelent túl nagy helynövekedést. Ugyanakkor a többkulcsos visszakereséskor a 100 adatrekord visszanyeréséhez az előző 400 beolvasás helyett csak 101-szer kell a lemezhez nyúlnunk.

Egyszer az indexlaphoz, amelyik a három feltételt együttesen kielégítő rekordok címét tartalmazza, és százszor a megfelelő rekordok beolvasásához. (Feltételeztük, hogy az indextáblázat direkt szervezésű és így egy lépéssel meg-

találjuk a kívánt kulcsértékhez tartozó rekordot, illetve rekordokat. Ez általában igaz, de ha nem így van, akkor is jól szervezett indextáblában néhány lépés után megtaláljuk a kívánt kulcs kombinációhoz tartozó rekordok címét.)

Amennyiben a keresési kulcsok között van olyan, amelyik szerint sűrűsödnek a rekordok, akkor célszerű ezt az összetett kulcsban az első helyre tenni. Ekkor a három kulcsra felépített indextáblázatból eszerint rendezve olvassuk a címeket. Ez azt jelenti, hogy a 100 kiválasztott rekord beolvasásához nem lesz szükség 100 lapváltásra. Jelentősen megnő a valószínűsége annak, hogy egy rekord beolvasása után a következő rekord is még ugyanazon a lapon lesz.

4.8 Hashing

Az indexek alapján való közvetlen elérés viszonylag nagy adminisztrációt jelent az adatbázis-kezelő rendszer számára. Indextáblázatokat kell létrehozni, módosítani, azokban külön kell még keresni, ami mind tárolóhelyet és időt jelent.

Ha csak egy kulcs alapján akarunk adatokat közvetlenül elérni, akkor erre lényegesen alkalmasabb az ún. *hashing módszer*. Itt a kulcs és a megfelelő adat címe között a megfeleltetés „majdnem” kölcsönösen egyértelmű. Valamilyen leképezést alkalmazunk tehát a kulcsok és a címek (illetőleg címtartományok) között, amely a legtöbb esetben különböző kulcsokhoz különféle címeket rendel hozzá. Bizonyos esetekben azonban különböző kulcsokhoz ugyanazt a címet (címtartományt) adja. [Ha a leképezést f -fel jelöljük, akkor lehetőleg $f(K_1) \neq f(K_2)$, ha $K_1 \neq K_2$ de bizonyos—lehetőleg kevés számú—esetben $f(K_1) = f(K_2)$, akkor is, ha $K_1 \neq K_2$.] A leképezés által azonos címhez rendelt, de különböző kulcsú elemeket *szinonimok*nak nevezzük. Nyilvánvaló, hogy ezek elhelyezéséről külön kell gondoskodnunk. Ennek részleteibe itt most nem megyünk bele. Annyi azonban már az eddigiek alapján is világos lehet, hogy a feldolgozás gyorsasága érdekében arra kell törekednünk: minél kevesebb szinonim rekordunk legyen. Azokhoz ugyanis csak bonyolultabb módon, több lépésben, azaz lassabban tudunk hozzáférni.

Másik célunk az kell legyen, hogy a rendelkezésre álló tárolóterülettel jól gazdálkodjunk, azaz ne foglaljunk le a szükségesnél nagyobb területet. Az alkalmazott leképezésnek olyannak kell tehát lennie, hogy a ténylegesen szereplő rekordok között ne legyen túl sok „hézag”, vagyis kihasználatlan cím.

E két törekvés általában nem elégíthető ki maradéktalanul, a gyakorlatban tulajdonképpen ellentmondóak. Az olyan leképezés, amely „sűrűn” adja a címeket, rendszerint sok szinonimra vezet, amelyik „ritkán”, rossz tárolókihasználást eredményez.

Ésszerű kompromisszumot kell tehát kötni. Általában jónak tarthatjuk az olyan leképezést, mely 80–85%-os kitöltöttség mellett nem eredményez 20%-nál több szinonimot.

A legegyszerűbb hashing algoritmus az, amikor a numerikus, vagy a hexadecimális ábrázolásban numerikusnak tekintett kulcsot elosztjuk egy primszámmal, és a maradékot tekintjük az adat címének.

Ez lesz az adat folyamatos elhelyezési sorszáma az adatbázisban, amiből már egyértelműen kiszámítható, melyik lapon és esetleg az is, hogy a lapon belül hol

található. Az osztót úgy választják meg, hogy közelítőleg egyenlő vagy valamivel kisebb legyen a rendelkezésre álló tárolóhelyek számánál.

Ma már ennél lényegesen jobb hashing algoritmusokat is kidolgoztak. Az algoritmus része az adatbázis-kezelő rendszernek (esetleg az adatbázis-felügyelő választhat is köztük), és elhelyezéskor illetve kulcs alapján kereséskor a rendszer határozza meg a fizikai címet, illetve ha szükséges, akkor végignézi a szinonimákat.

Egy adott kulcs szerint való közvetlen feldolgozásra a jó hashing általában hatékonyabb az indexeknél. Ezért a hierarchikus és főleg a hálós modellekre épülő adatbázis-kezelő rendszerekben előszeretettel alkalmazzák ezt a módszert a közvetlen elérési biztosítására. Ugyanakkor viszont a relációs adatbázisokban már nemigen, vagy csak nagyon speciális esetekben alkalmazzák, mivel

- nem teszi lehetővé a szekvenciális elérést még a direkt elérésre szolgáló hashing kulcs szerint sem,
- egy rekordtípuson belül csak egy kulcs szerint lehet segítségével közvetlenül hozzáférni az adatokhoz.

A relációs modellek fő vonzóereje a széleskörű használhatóság, és a hashing az indexekkel ellentétben, éppen ezt nem tudja biztosítani.

4.9 Adattömörítés és titkosítás

4.9.1 Adattömörítési módszerek

A mágneslemezek tárolókapacitásának növekedése és árának csökkenése ellenére is fontos szempont maradt adatbázisok tervezésénél a külső tárolón levő hely jó kihasználása. Ha a hasznos adatok kisebb helyet foglalnak el a tárolón, akkor elsősorban nem a szükséges lemezegységek számát csökkentjük le, hanem több információt tárolhatunk az adatbázisban. Ezen kívül a feldolgozás is gyorsabb lesz két okból:

- egy lapra több adat fér el, ritkább lesz a lapváltások, az író/olvasó műveletek száma,
- ugyanazon információhoz kevesebb byte-ot kell átvinni a kommunikációs hálózaton, gyorsabban jutnak el az adatok a végfelhasználókhoz (és az osztott adatbázisok között).

Adatbázisokban a tömörítést leggyakrabban változó hosszúságú mezők és rekordok alkalmazásával érik el. Ha a mezők értékes adattartalma változó hosszú lehet (pl. nevek, címek), akkor a mezőt olyan hosszúra kell deklarálni, hogy a leghosszabb adat is elférjen benne. Ugyanakkor viszont a legtöbb esetben karaktereknél csak a mező elején levő byte-ok hordoznak értékes információt, a mező végén levő szóközők fölöslegesen foglalnak le helyet. (Számoknál fordított a helyzet, ott a mező elején levő értéktelen nullákat hagyhatjuk el.) A névmezőbe be kell férjen „Szentmihályfalvai Szilveszterné Alsónagybányai Anasztázia” neve is, de lehet, hogy csak „Kő Ede” áll benne.

Még markánsabb a hossz-különbség a 8. fejezetben ismertetendő szöveges információrendszerekben.

A mező végén lévő értéktelen szóközők elhagyása érdekében lehetőség nyílik a fizikai szinten változó hosszúságú mezők deklarálása. Az ilyen mezők elé az adatbázis-kezelő rendszer behelyez egy mezőhosszúság-jelzőt (általában 2 byte). Így a maximálisan N byte hosszúságú, karakteres mező $N-n+2$ byte-ot fog csak lefoglalni a lemezen, ha a végén eredetileg n darab üres karakter, szóköző jel lett volna.

Ha egy rekord változó hosszúságú mező(ke)t tartalmaz, akkor a rekord hossza is változó. Ezért az ilyen típusú rekordok elé a rendszer automatikusan odatesz egy (rendszerint 4 byte hosszú) rekordhosszúság jelzőt is.

Mivel a változó hosszúság jelzése is helyet foglal el, és kezelése, a rekord értékes hosszának megállapítása, illetve visszakereséskor szóközzel való feltöltése is időt igényel, a gyakorlatban csak akkor célszerű ilyen mezőkkel dolgoznunk, ha az átlagos helymegtakarítás mezőnként legalább 12–15 byte.

Szokásos eljárás még (és ha az adatbázisban a tároláskor nem is alkalmazzák, de a telekommunikációs hálózaton való átvitelkor automatikusan megvalósulhat), hogy az egymás után közvetlenül többször ismétlődő karakterek közül csak az elsőt és az ismétlődések számát viszik át a rendszeren. Az ismétlődő karakter előtt egy speciális, másra nem használható bitkombináció jelzi, hogy most egy ismétlődő jelsorozat következik rövidítve. Amennyiben ezt a speciális bitkombinációt kell átvinni, akkor ezt a rendszer átvitel előtt automatikusan megduplázza. Ez jelzi vételkor, hogy nem ismétlődő karakterek érkeztek, hanem a speciális jelsorozat.

Szöveges információ tárolásakor a legnagyobb helymegtakarítás az ún. Huffman-kóddal érhető el. Az egyes karaktereket különböző hosszúságú bitsorozatokkal ábrázolják, a leggyakoribb karakterek állnak a legkevesebb bitből. Bár egyes kísérletekben a helymegtakarítás 30–40%-ot is elért, a módszer mégsem terjedt el a gyakorlatban. Ennek oka elsősorban az átkonvertálás nehézsége és időigényessége.

4.9.2 Titkosítási algoritmus

Az adatbázis-kezelő rendszer ugyan védelmet nyújt az illetéktelen hozzáférések ellen, de az adatok tárolási helyén levő rendszerprogramozók rendelkezhetnek olyan eszközökkel, hogy ezt a védelmet megkerüljék. Méginkább gondot okoz, hogy a biztonsági okokból kimentett másolatokra az adatbázis-kezelő rendszer általában semmiféle védelmet nem nyújt. Aki fizikailag hozzáférhet a mentési szalagokhoz vagy lemezekhez, az el is tudja olvasni, mi van rajtuk. Ugyanez érvényes az adatátviteli vonalakon átmenő adatokra is. Aki azt le tudja hallgatni, az az ott áthaladó információ birtokába kerülhet.

Éppen ezért az igen bizalmas, titkos adatokat nem eredeti formájukban kell tárolni az adatbázisban, hanem rejtjelezve.

A rejtjelezés egy R rejtjelező algoritmus alapján történik, melynek tényleges megvalósítása függ a K_R paraméter(ek)től, a rejtjelezési kulcstól. Ennek hatására az S szövegből előáll az $S_R = R_{K_R}(S)$ szöveg, melyből az eredeti csak a V visszaállító algoritmussal, a K_V paraméter(ek) ismeretében állítható elő, azaz $S = V_{K_V}(S_R)$. A gyakorlatban sokszor $K_R = K_V$, de ez nem feltétlenül szükséges.

A kódolási algoritmusok száma elvileg végtelen, de a számítógépes megvalósítás érdekében az USA Szabványügyi Hivatala (National Bureau of Standards) 1977-ben egy szabványos rejtjelező és visszaállító algoritmust vezetett be, melynek alkalmazására a legtöbb géptípushoz ma már kész programok állnak rendelkezésre. Hiába ismert azonban az algoritmus, a rejtjelezett szöveg megfejtése a kulcs nélkül nem lehetséges. Ennek pedig a javasolt eljárásnál 2^{55} különböző változata lehet, amiknek végigpróbálása és a kapott eredmények kiértékelése még a legnagyobb sebességű szuperszámítógépekkel sem valósítható meg reális időn belül. Ugyanakkor a szöveg átkódolása és a titkosított szöveg visszaállítása a kulcs ismeretében másodpercek vagy legfeljebb néhány perc alatt elvégezhető.

Gondolni kell arra is, hogy a kulcs esetleg illetéktelenek birtokába kerülhet. Ha ennek gyanúja felmerül, akkor a titkos adatokat kimentik, egy új kulcs alapján újra rejtjelezik, a régi adatokat törlik és az átkódoltakat visszatöltik. Az új rejtjelezési kulcs értékét megadják minden arra jogosultnak.

4.10 Rendezés

A számítógépek nagymennyiségű adatot tudnak tárolni és feldolgozni. Ahhoz azonban, hogy ez a feldolgozás hatékony és az ember által áttekinthető legyen, általában szükséges, hogy az adatok bizonyos megfelelően kiválasztott szempont vagy szempontok szerint rendszerezve, rendezve legyenek. Az adatfeldolgozásban az adatok tetszés szerinti rendezettségben való előállítását létrehozó általános rendezőprogramok, azok egyszerű használata és hatékony algoritmusai kezdettől fogva kulcsszerepet játszott. Minden valamirevaló számítógép operációs rendszere tartalmaz rendező/összeválogató¹¹ programgenerátort. Ennek bemenő állománya a rendezendő, kimenete a rendezett adatállomány. Ezen kívül meg kell adni azokat a vezérlő utasításokat is, amelyek leírják a bemenő és a kimenő állományokat, azok rekordjait, a rendezési szempontokat és a helyét a rekordokban.

Az adatbázis technológiában ezeket a rendező/összeválogató programokat közvetlenül nemigen használják. Legfőljebb az adatbázis, vagy annak egyes rekordtípusainak első feltöltésekor rendezik előbb a bemenő állományt. A rekordokat a későbbi feldolgozás hatékonyságának érdekében az elsődleges kulcs szerint eleve rendezve visszük be az adatbázisba.

Az adatbázissal való munka közben azonban közvetve gyakran szükségünk van az adatbázisban levő adatoknak, illetve általában azok egy kiválasztott részének ad hoc jellegű rendezésére. A rendezés rövid ismertetésének fontosságát az indokolja, hogy ez az adatfeldolgozás egyik legidőigényesebb művelete. Ez különösen nagy rekordmennyiségnél jelentkezik, mivel a rendezés időigénye a rendezendő adatok számával nem egyenesen arányos, hanem a rendezési módszertől függően $n \cdot \log(n)$ és n^2 közötti nagyságrenddel arányos (n a rendezendő rekordok száma). Az adatbázis felhasználója sem a rendezőprogrammal, még kevésbé az ezen program által generált rendezési algoritmussal nem kerül közvetlen kapcsolatba. A rendezéshez szükséges műveleteket, amennyiben erre igény van, az adat-

¹¹ Az angol Sort/Merge generátor elnevezést is gyakran használják a szakirodalomban.

bázis-kezelő rendszer automatikusan elvégzi. Ennek ellenére célszerűnek tartjuk a legfontosabb rendezési eljárások rövid ismertetését, hogy a felhasználó érzékelné tudja, milyen terheket ró a rendszerre, azaz általában mennyivel lassítja le működését, ha olyan feladatokat kíván megoldani, amelyek szükségessé teszik az adatok alkalmi rendezését.

Az adatbázis-kezelő rendszerek belső rendezőprogramjai általában nem rendezik fizikai sorrendbe a rendezendő adatokat, hanem csak egy indexállományt hoznak létre. Ez tartalmazza a kiválasztott rendezési kulcsot vagy kulcsokat a kívánt sorrendben, és az adott kulcshoz tartozó rekord címét az adatbázisban.

A rendezett feldolgozás sorrendjét ez az indextáblázat határozza meg.

4.10.1 Automatikus rendezés

A gyakorlatban rendszerint az alábbi esetekben van szükség az adatbázisban utólagos rendezésre:

- új indexek létrehozása meglévő adatokon,
- alkalmilag feldolgozott adatok rendezése olyan kulcs szerint, melyre nincsen index,
- azonos értékkel rendelkező rekordok kiszűrése.

Az alábbiakban röviden tárgyaljuk mindhárom előfordulást.

Új index(ek) létrehozása meglévő adatokon

Utólag rájövünk arra, hogy egy adott mező (vagy mezők) értéke szerint rendszeresen akarunk közvetlenül, vagy növekvő illetve csökkenő sorrendben adatokat feldolgozni. Ebben az esetben a hatékonyságot nagymértékben növelhetjük, ha erre a mezőre utólag egy indexet építünk fel.

Az index felépítéséhez a rekordokat az új index értéke szerint előbb rendezni kell, majd a rendezett kulcsok alapján fel kell építeni az indextáblázatot. Ez rendszerint, mivel nagy adatmennyiségről van szó, viszonylag hosszú időt igénybevevő művelet, de csak egyszer kell elvégezteni. Az adatállomány változásakor az indexek karbantartásához, illetve a sok módosítás miatti újraszervezéshez már nem kell az adatokat újra rendezni.

Ez vagy a 4.6.1 pontban ismertetett listaszervezés vagy a 4.7.5 pontban tárgyalt B⁺-fa módosításával történik.

Természetesen, mint azt már mondtuk, az új index létrehozásához is elegendő, ha a felhasználó az adatleíró nyelven megadja az adatbázis-kezelő rendszernek, hogy

- milyen rekordtípuson,
- ennek mely mezőin,
- milyen rendezettségben (növekvő vagy csökkenő),
- a kulcsérték egyediségét ellenőrizve, vagy duplikátumokat is megengedve

hozza létre az új indexet (l. a 10.4 pontban a CREATE INDEX utasítást).

Eseti műveletek eredményének alkalmi rendezése

Az adatbázisból több rekordot (esetleg több rekordtípusból összeválogatva) választunk ki és ezeket olyan mező(k) szerinti sorrendben akarjuk megjeleníteni

vagy továbbadni feldolgozásra, mely(ek) szerint nincsen index. Ez igen gyakran fordul elő ad hoc jellegű lekérdezéseknél, amikor egyes felhasználásoknak a legkülönbözőbb sorrendben, csoportosításban van szükségük az adatokra.

Mivel a kiválasztott adatok és a rendezési szempontok is állandóan változnak, az összes ilyen esetre nem lehet és nem is lenne célszerű indexeket felépíteni. Ezért, ha ilyen feladatot kell megoldani, akkor az adatbázis-kezelő rendszer mindig a feladat végrehajtásakor generálja a rendezéshez szükséges utasításokat, létrehozza a rendezett ideiglenes állományt, majd a feladat végrehajtása után törli ezeket (l. 11.2 pont, ORDER BY klauzula). A rendezés lefuttatásához természetesen időre, és ideiglenes tárolóhelyre van szükség. Tárolóként a rendszer az adatbázis területén külön erre a célra fenntartott munkaterületet használ, mely az adatbázis méretét növeli meg. A rendezéshez szükséges idő pedig a feldolgozást lassítja le, sokszor esetleg egy nagyságrenddel is.

Duplikátumok kiszűrése

Egy feladat megoldásánál a kiválasztás eredményeként több rekordot is kaphatunk. Ezekből a továbbiakban nem a teljes rekordra, hanem ennek csak néhány mezőjére van szükségünk. Ha a fölösleges oszlopokat elhagyjuk, akkor a kiválasztott és eredetileg különböző rekordokból az eredményben már azonosak is lehetnek. Nekünk viszont ezekből a továbbiakban csak egyre van szükségünk. Az eredményrekordokat (az összes megtartott mező szerint) rendezzük. Az azonos rekordok így közvetlenül egymás mögé kerülnek. A duplikátumok ebből már könnyen kiszűrhetők.

Ilyen feladatra példa lehet egy vállalati nyilvántartás, melyből kérjük azoknak az osztályoknak a nevét (vagy kódját), ahol van havi 10 000 Ft-nál kevesebbet kereső dolgozó. Az egyéni fizetési adatokat is tartalmazó rekordtípusból (táblázatból) kiválasztjuk azokat a rekordokat, ahol a fizetés értéke kisebb 10 000-nél, és csak az osztály nevét (vagy kódját) hagyjuk meg. Ebben az „első közelítésű” eredménylistában minden osztály annyiszor szerepel, ahány 10 000 Ft alatt kereső dolgozója van. Ezt tesszük áttekinthetőbbé, hogy az azonos eredményrekordokból a duplikátumokat elhagyjuk.

4.10.2 Rendezési módszerek

A rendezéskor is, mint minden számítógépes feladat megoldásakor, az idő, az operatív memória, és a külső tárolókapacitás, valamint a beszerzési és üzemeltetési költségek ellentmondásával találkozunk. Gyors rendezéshez nagy operatív memória, illetve igen nagy mennyiségű adatnál nagy ideiglenesen felhasználható lemezterület kell, ami pénzbe kerül. Kisebb memóriánál és tárolókapacitásnál viszont a rendezési idő nő meg. A gépidő és főleg on-line jellegű feldolgozásnál a felhasználó ideje, viszont még drágább.

Az ellentmondás feloldására—mint a számítástechnikában mindig—, célszerű kompromisszum született. Mivel az operatív memóriában a műveleti sebesség nagyságrendekkel gyorsabb, mint a lemezre való írás, illetve onnan az olvasás, ezért az adatokból, amennyit csak lehet, beolvasnak a memóriába és ott rendezik. (Adatokon itt most általában csak az adott rekord rendezési kulcsát és a lemezcímet értjük.) Az így rendezett részhalmazt kiviszik lemezre egy ideiglenes állo-

mányba, és a rendezendő adatok újabb csomagját olvassák be a memóriába, ahol rendezik, majd kiírják lemezre. Ezt addig folytatják, amíg a teljes rendezendő állomány ki nem kerül a lemezre rendezett részhalmazként. Ezekből aztán összeválogatják a teljes rendezett adatállományt.

Kevés adat esetén lehetséges, hogy a teljes adatállományt egyidejűleg az operatív memóriába rendezzük. Nagyobb adatállománynál azonban már külső tárolót, ideiglenes lemezterületeket is fel kell használnunk. Ennek következtében a rendezés hatékonysága két részből tevődik össze:

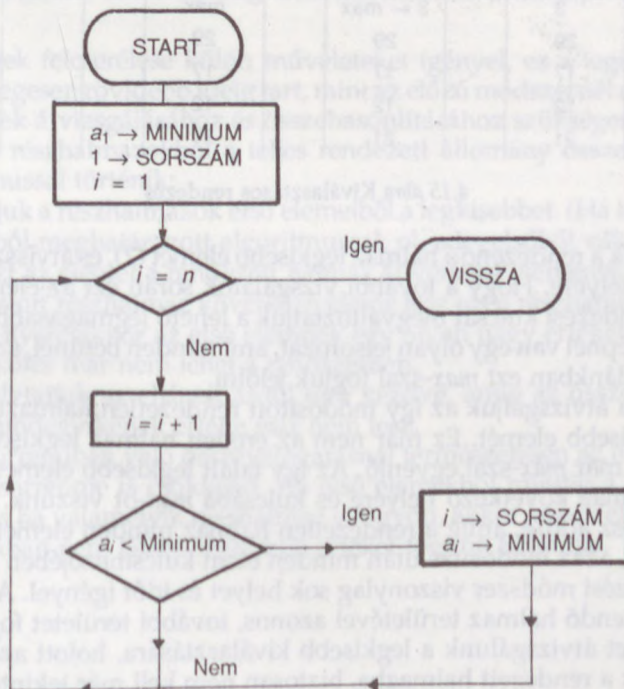
- az operatív memóriában történő belső rendezés és
- a lemezre kivitt, rendezett részhalmazok egy rendezett állományba történő összeválogatásának hatékonyságából.

A belső rendezésre különféle algoritmusokat használunk.

Legtöbbjük azon alapszik, hogy vagy a memóriában levő teljes állományból kiválasztják a legkisebbet (legnagyobbat), és azt az állomány első (utolsó) helyére teszik, vagy két, meghatározott algoritmussal kiválasztott elemből a kisebbet (nagyobbat) teszik folyamatosan előbbre. A végleges sorrendben megfelelő elemet most már kihagyva, ezt az eljárást ismétlik azután addig, amíg minden elem a helyén nem lesz.

Egy ilyen algoritmust az alábbiakban mutatunk be. Ezen a módszer elve jól követhető, de megjegyezzük, hogy vannak ennél hatékonyabb, de bonyolultabb eljárások is. A gyakorlatban ma már inkább ezeket alkalmazzák, hiszen a felhasználó elől az algoritmus úgyis rejtve marad.

Az algoritmus első lépése a legkisebb elem kiválasztása. Ez is többféleképpen hajtható végre. Ennek egyik legegyszerűbb változatát mutatjuk be a 4.14 ábrán.



4.14 ábra Legkisebb elem kikeresésére szolgáló rutin blokkdiagramja

Mivel ezt az eljárást igen sokszor meg kell ismételni, lényeges, hogy ez a rutin jó algoritmust használjon, és hatékony módon legyen programozva. A fent ismertetett eljárás egyszerű és könnyen programozható, de hatékonyság szempontjából azok a módszerek, melyek minden alkalommal kiválasztják a halmaz legkisebb elemét, messze vannak az optimálistól. Éppen ezért nagy tömegű adat rendezésekor sokszor célszerűbb szomszédos vagy egymástól megadott algoritmussal kiszámítható távolságra levő elemeket összehasonlítani, és ily módon a legkisebb elem meghatározása mellett már a rendezés további lépéseit is elő tudjuk készíteni.

A legkisebb elem kiválasztására szolgáló egyszerű módszer ismertetése után most bemutatunk egy olyan áttekinthető eljárást, amelynek ismételt felhasználásával egy halmaz elemeit sorba lehet rendezni.

A módszert kiválasztásos rendezésnek (selection sorting) hívják, és elvét a 4.15 ábra mutatja.

Elem sorszáma	Adatok								
	1. lépés		2. lépés		3. lépés		...	utolsó előtti lépés	
	Rende- zetlen	Ren- de- zett	Rende- zetlen	Ren- de- zett	Ren- de- zetlen	Ren- de- zett	...	Rende- zetlen	Ren- de- zett
1	8	2	8	2	8 ← max	2		max	2
2	16		16	3	16	3		max	3
3	2 ← max		max		max	8		max	8
4	3		3 ← max		max			max	12
5	29		29		29		29 ← max	16	16
6	17		17		17			max	17
7	12		12		12			max	21
8	21		21		21			max	29

4.15 ábra Kiválasztásos rendezés

Kiválasztjuk a rendezendő halmaz legkisebb elemét (2), és átvisszük a rendezett halmaz első helyére. Hogy a további vizsgálatok során ezt az elemet ne vegyük tekintetbe, rendezési kulcsát megváltoztatjuk a lehető legmagasabb értékre. (Minden számítógépnél van egy olyan jelsorozat, ami minden betűnél, számnál és jelnél nagyobb. Példánkban ezt *max*-szal fogjuk jelölni.

Ezután újra átvizsgáljuk az így módosított rendezetlen halmazt és ismét kikeressük a legkisebb elemét. Ez már nem az eredeti halmaz legkisebb eleme, hisz annak kulcsa már *max*-szal egyenlő. Az így talált legkisebb elemet elhelyezzük a rendezett halmaz következő helyére és kulcsába *max*-ot viszünk. Ezt az eljárást folytatjuk egész addig, amíg a rendezetlen halmaz minden elemét át nem vittük a rendezettbe, azaz módosítás után minden elem kulcsmezőjében *max* nem lesz.

Ez a rendezési módszer viszonylag sok helyet és időt igényel. A rendezett halmaz a rendezendő halmaz területével azonos, további területet foglal el. Mindig minden elemet átvizsgálunk a legkisebb kiválasztására, holott azokat, melyeket már kivittünk a rendezett halmazba, biztosan nem kell már tekintetbe venni.

Ezen segít a 4.16 ábrán látható módosítás. Amikor először átvizsgáljuk a teljes rendezetlen halmazt, akkor a legkisebb elemet (2) kicseréljük az első elemmel (8). Második lépésben csak második elemtől nézzük végig a rendezetlen halmazt, és az így talált legkisebb elemet (3) kicseréljük a másodikkal (16). Ezután a harmadik elemtől vizsgálunk, és azt cseréljük ki a legkisebbre. (A példában ez véletlenül most önmagával egyezik.) Utána folytatjuk ezt az eljárást, amíg a halmaz elemeinek végére nem érünk.

Elem sorszáma	Adatok					
	1. lépés	2. lépés	3. lépés	4. lépés	...	Utolsó előtti lépés
1		2	2	2		2
2	16		3	3		3
3		8		8		8
4	3	3	16			12
5	29	29	29	29		16
6	17	17	17	17		17
7	12	12	12	12		21
8	21	21	21	21		

4.16 ábra Kiválasztásos rendezés felcseréléssel
A legkisebb elemet a vastag vonal alatt levők közül választjuk ki

Bár az elemek felcserélése külön műveleteket igényel, ez a legelső elemektől eltekintve lényegesen rövidebb ideig tart, mint az előző módszernél a már *max*-szal megjelölt elemek átvizsgálásához és összehasonlításához szükséges idő.

A rendezett részhalmazokból a teljes rendezett állomány összeválogatása az alábbi algoritmussal történik:

- Kiválasztjuk a részhalmazok első elemeiből a legkisebbet. (Ha több ilyen van, akkor ebből meghatározott algoritmussal, pl. a legelsőből választjuk ki.)
- Beírjuk ezt az elemet a rendezett halmaz következő elemének.
- A kiválasztott részhalmaz következő eleme lesz ezen részhalmaz első eleme. Ha nincs ilyen, akkor az első elemnek egy olyan nagy értéket (*max*) adunk, amilyen kulcs már nem lehet a rendszerben.
- Addig folytatjuk az eljárást a)-tól újra kezdve, amíg az összehasonlítandó elemek mindegyikének értéke *max* nem lesz.

Csökkenő sorrendben való összeválogatásnál természetesen az előre rendezett részhalmazok sorrendje is csökkenő. Az első elemekből mindig a legnagyobbát választjuk ki, és ha valamelyik részhalmaz elfogyott, akkor ennek első elemeként, egy, a rekordokban nem létező legkisebb értéket fogunk tekinteni.

A hierarchikus modell

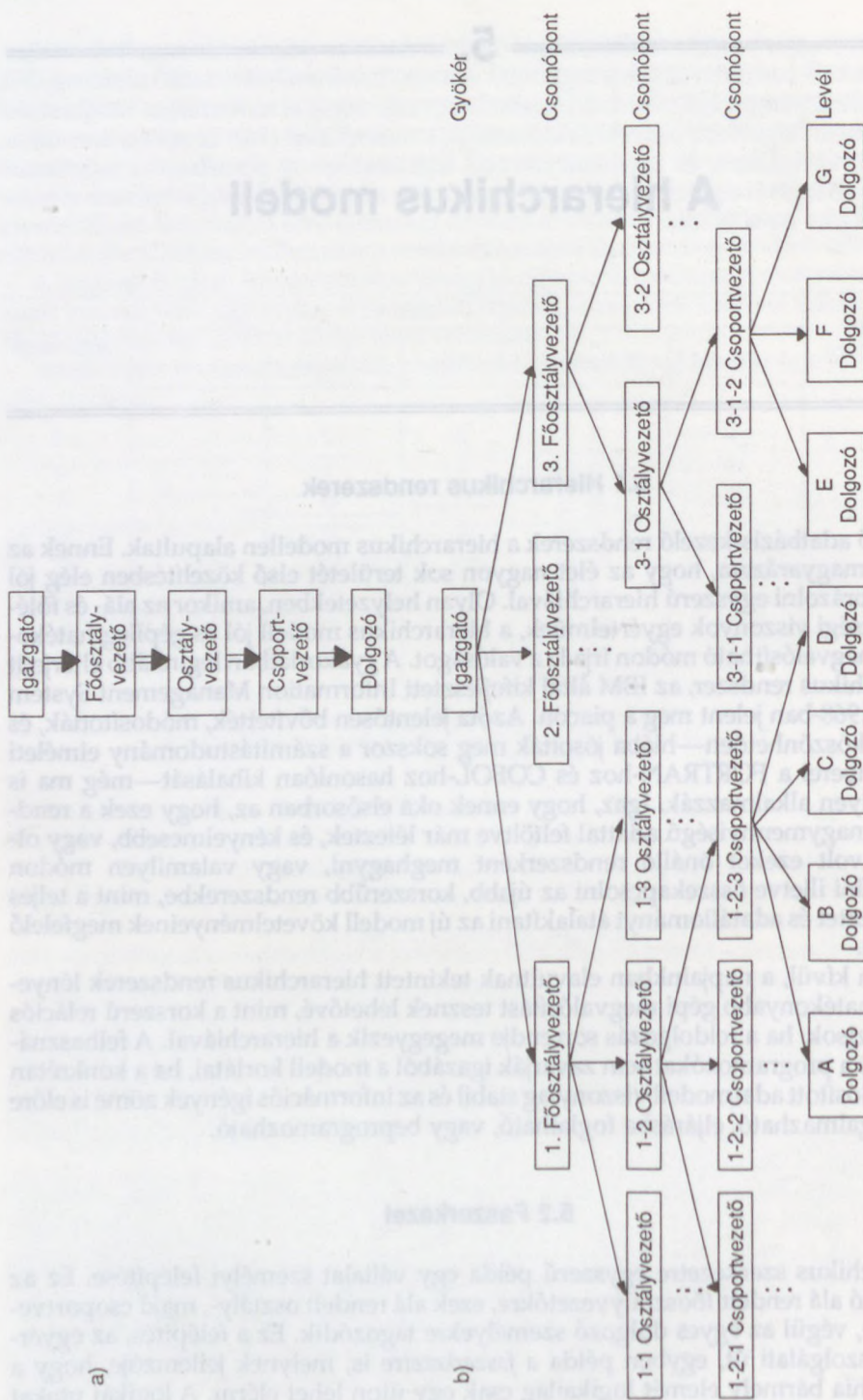
5.1 Hierarchikus rendszerek

Az első adatbázis-kezelő rendszerek a hierarchikus modellen alapultak. Ennek az volt a magyarázata, hogy az élet nagyon sok területét első közelítésben elég jól lehet ábrázolni egyszerű hierarchiával. Olyan helyzetekben, amikor az alá- és fölérendeltségi viszonyok egyértelműek, a hierarchikus modell jól és gépileg hatékonyan megvalósítható módon írja le a valóságot. A gyakorlatban leginkább elterjedt hierarchikus rendszer, az IBM által kifejlesztett Information Management System (IMS) 1968-ban jelent meg a piacon. Azóta jelentősen bővítették, módosították, és ennek köszönhetően—hiába jósolták meg sokszor a számítástudomány elméleti szakemberei a FORTRAN-hoz és COBOL-hoz hasonlóan kihalását—még ma is sok helyen alkalmazzák. Igaz, hogy ennek oka elsősorban az, hogy ezek a rendszerek nagymennyiségű adattal feltöltve már léteztek, és kényelmesebb, vagy olcsóbb volt ezeket önálló rendszerként meghagyni, vagy valamilyen módon integrálni illetve összekapcsolni az újabb, korszerűbb rendszerekbe, mint a teljes szerkezetet és adatállományt átalakítani az új modell követelményeinek megfelelő módon.

Ezen kívül, a napjainkban elavultnak tekintett hierarchikus rendszerek lényegesen hatékonyabb gépi megvalósítást tesznek lehetővé, mint a korszerű relációs adatbázisok, ha a feldolgozás sorrendje megegyezik a hierarchiával. A felhasználókat és a programozókat sem zavarják igazából a modell korlátai, ha a konkrétan megvalósított adatmodell viszonylag stabil és az információs igények zöme is előre megfogalmazható, eljárásba foglalható, vagy beprogramozható.

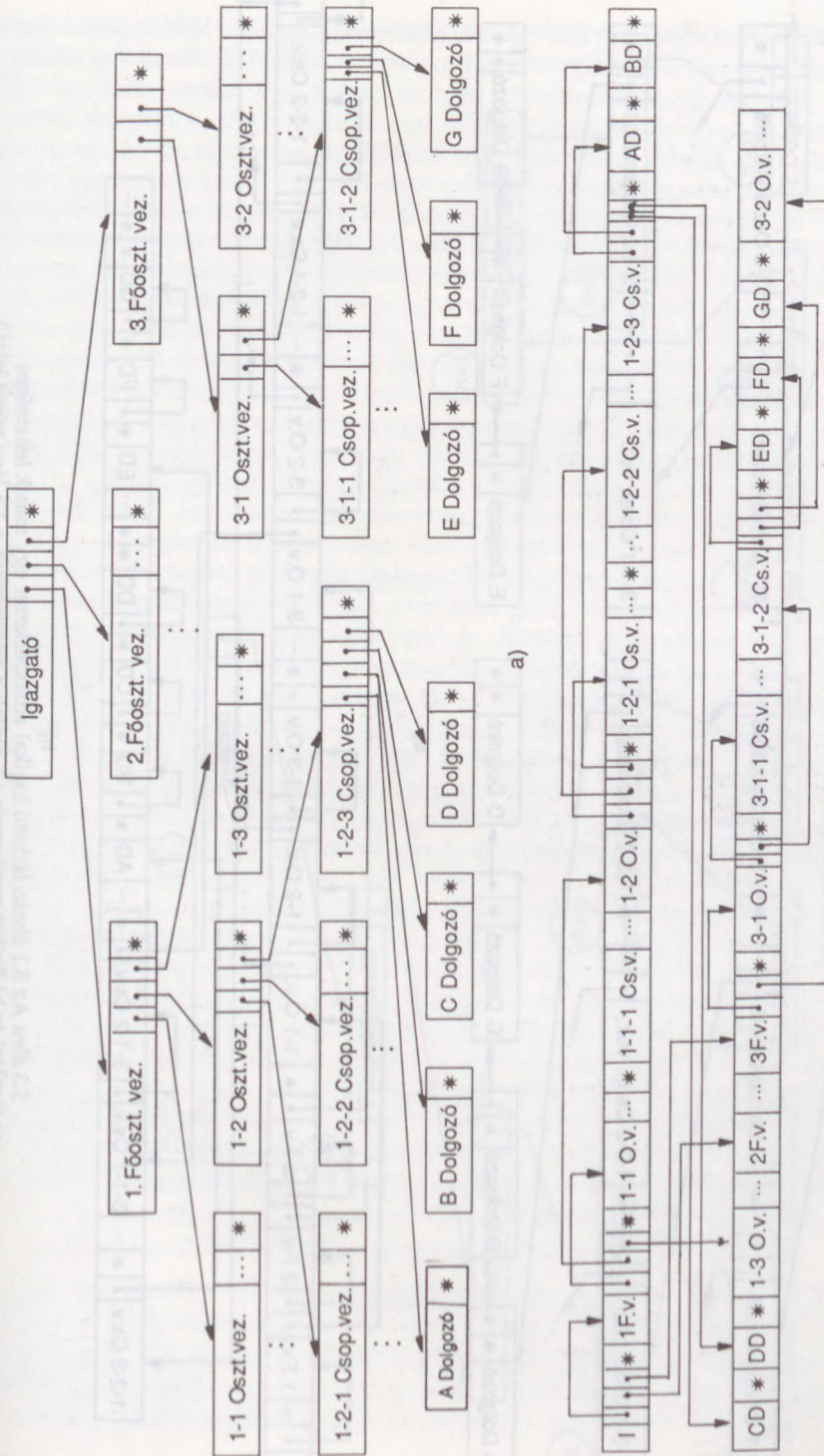
5.2 Faszervezet

Hierarchikus szervezetre egyszerű példa egy vállalat személyi felépítése. Ez az igazgató alá rendelt főosztályvezetőkre, ezek alá rendelt osztály-, majd csoportvezetőkre, végül az egyes dolgozó személyekre tagozódik. Ez a felépítés, az egyértelmű szolgálati út, egyben példa a *faszervezetre* is, melynek jellemzője, hogy a hierarchia bármely elemét logikailag csak egy úton lehet elérni. A logikai utakat

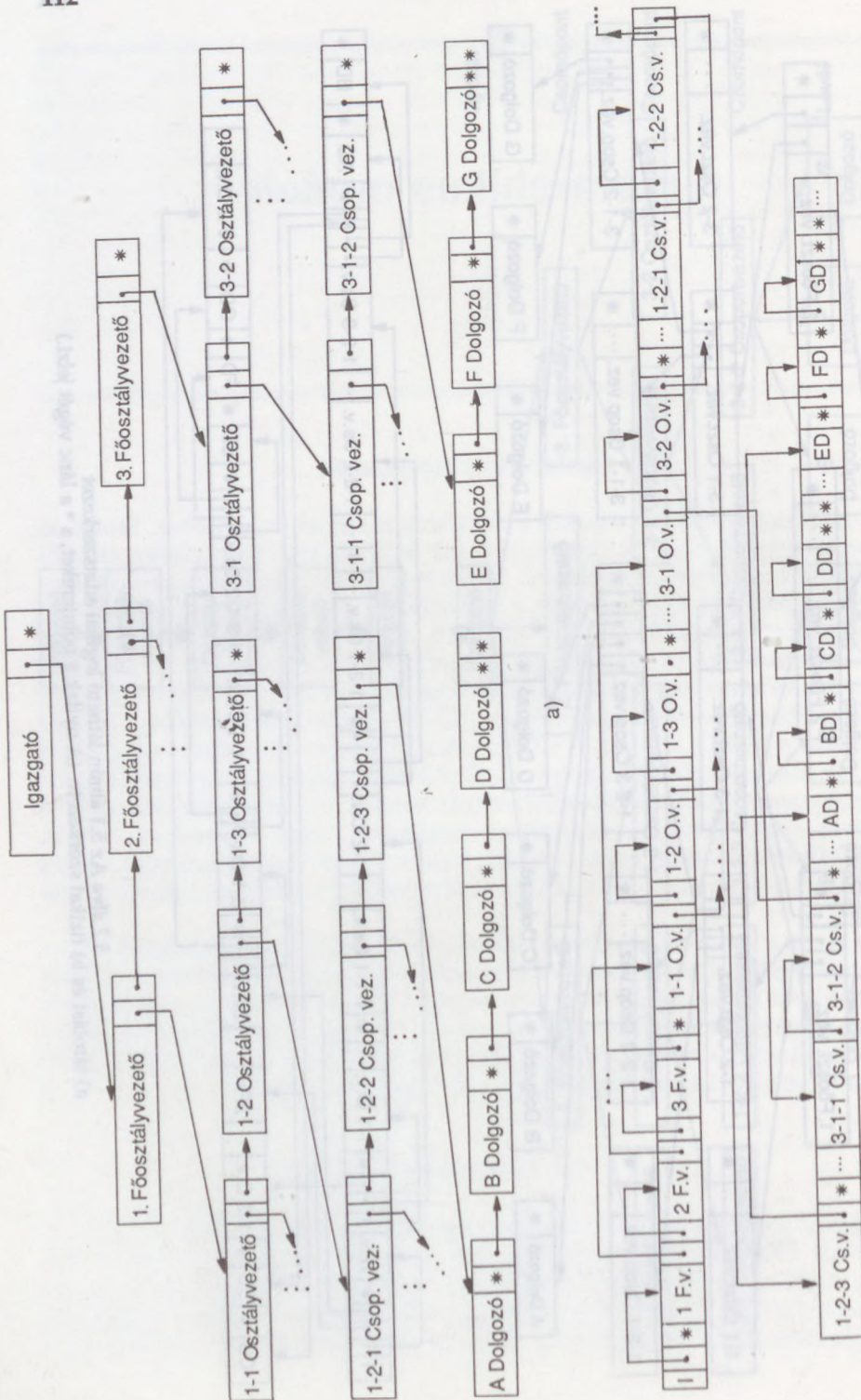


5.1 ábra Példa faszervezetre

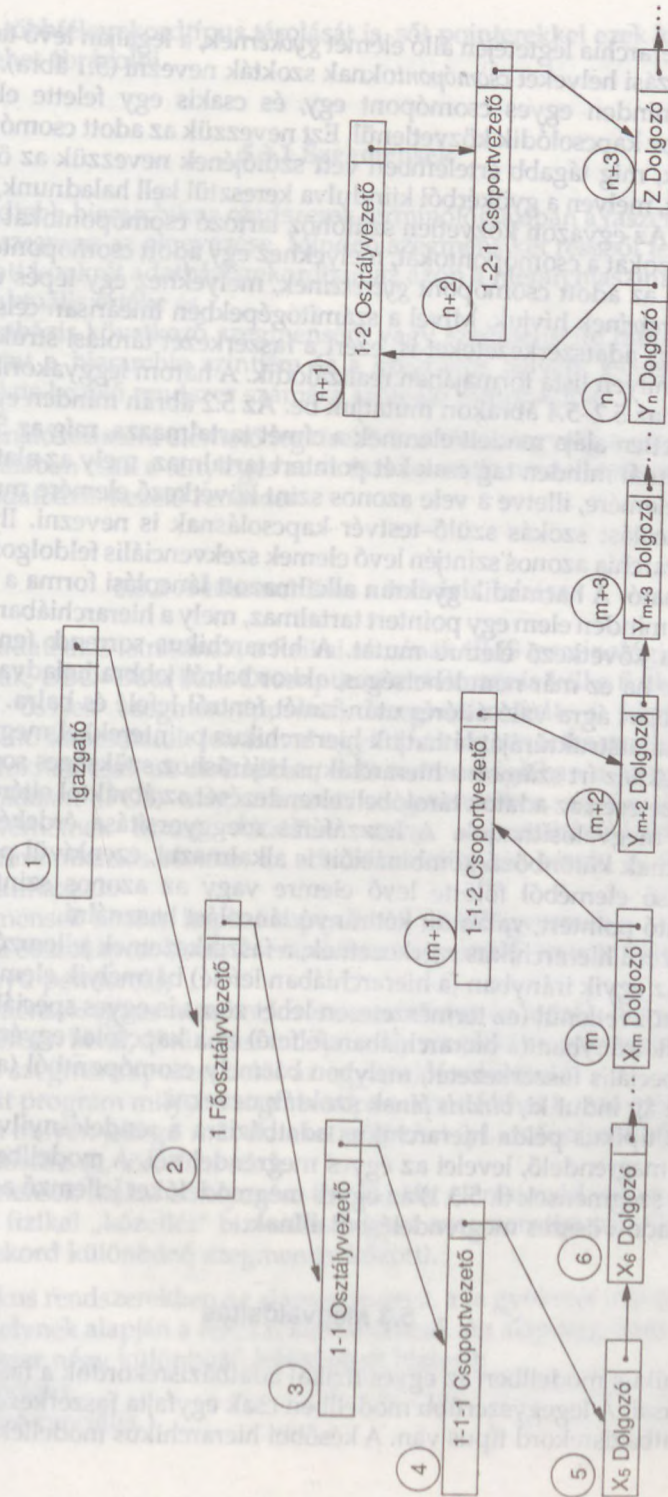
a) általános adatstruktúrával ábrázolva b) részletezve az egyes elemek előfordulásait



5.2 ábra Az 5.1 ábrán látható logikai adatszerkezet a) tárolási és b) fizikai szerkezete. (A nyílak a pointereket, a * a lánc végét jelzi.)



5.3 ábra Az 5.1 ábrán látható logikai adatszerkezet egy másik lehetséges a) tárolási és b) fizikai szerkezete. (A nyílak a pointereket, a * a lánc végét jelzi.)



5.4 ábra Faszervezet hierarchikus pontterekkel összekötve

ágaknak, a hierarchia legtetején álló elemet *gyökérnek*, a legalján levő tagokat *levélnek*, az elágazási helyeket *csomópontoknak* szokták nevezni (5.1 ábra). A gyökereket kivéve minden egyes csomópont egy, és csakis egy felette elhelyezkedő csomóponthoz kapcsolódik közvetlenül. Ezt nevezzük az adott csomópont *közvetlen szülőjének*, míg tágabb értelemben vett szülőjének nevezzük az összes olyan csomópontot, melyen a gyökérből kiindulva keresztül kell haladnunk, hogy eljussunk hozzá. Az egyazon közvetlen szülőhöz tartozó csomópontokat *testvéreknek* nevezzük. Azokat a csomópontokat, melyekhez egy adott csomóponton keresztül juthatunk el, az adott csomópont *gyerekeinek*, melyekhez egy lépés után, azokat *közvetlen gyerekeinek* hívjuk. Mivel a számítógépekben lineárisan célszerű tárolni a hierarchikus adatszerkezeteket is, ezért a faszerkezet tárolási struktúrája rendszerint valamilyen lista formájában realizálódik. A három leggyakoribb megvalósítási módot az 5.2–5.4 ábrákon mutatjuk be. Az 5.2 ábrán minden egyes elem az összes közvetlen alája rendelt elemnek a címét tartalmazza, míg az 5.3 ábrán látható változatnál minden tag csak két pointert tartalmaz, mely az alatta levő hierarchia első elemére, illetve a vele azonos szint következő elemére mutat.

Ezt a láncolást szokás *szülő–testvér* kapcsolásnak is nevezni. Ilyen láncolás esetén a hierarchia azonos szintjén levő elemek szekvenciális feldolgozása gyorsan megvalósítható. A harmadik gyakran alkalmazott láncolási forma a hierarchikus lánc. Ebben minden elem egy pointert tartalmaz, mely a hierarchiában közvetlenül alatta/utána következő elemre mutat. A hierarchikus sorrend: fentről lefelé és balra, illetve ha ez már nem lehetséges, akkor balról jobbra haladva a következő még nem bejárt ágra való áttérés után ismét fentről lefelé és balra. Az 5.4 ábrán az 5.1 ábra adatstruktúráját láthatjuk hierarchikus pointerekkel megvalósítva. Az egyes elemekhez írt számok a hierarchikus bejáráshoz szükséges sorrendet jelentik. Természetesen az adatok tárolóbeli elrendezését az ábráktól eltérő fizikai szerkezettel is megvalósíthatjuk. A hozzáférés meggyorsítása érdekében lehet az alapformáknak különböző kombinációit is alkalmazni, ezenkívül például a hierarchia utolsó eleméből fölfelé levő elemre vagy az azonos szint első tagjára visszamutató pointert, valamint kétirányú láncolást használni.

Az egyszerű hierarchikus szerkezetnek, a faszerkezetnek jellemzője az *egy-több kapcsolat*. Az egyik irányban (a hierarchiában lefelé) bármelyik elemhez több kapcsolódhat közvetlenül (ez természetesen lehet zérus is egyes speciális esetekben), míg a másik irányban (a hierarchiában felfelé) ez a kapcsolat egyértelmű.

Azt a speciális faszerkezetet, melyben bármely csomópontból (a levelek kivételével) két ág indul ki, *bináris fának* szokták nevezni.

További tipikus példa hierarchikus adatbázisra a rendelésnyilvántartás. A fa gyökere a megrendelő, levelei az egyes megrendelések. A modellben egy rekordként tárolt szegmensek (l. 5.3.1) az egyes megrendelőket jellemző adatokból és az általuk föladatazott összes megrendelésből állnak.

5.3 Megvalósítás

A hierarchikus modellben az egyes fizikai adatbázisrekordok a faszerkezet egyes előfordulásai. A legegyszerűbb modellben csak egyfajta faszerkezet, azaz egyféle fizikai adatbázisrekord típus van. A későbbi hierarchikus modellek azonban lehe-

tővé teszik többféle rekordtípus tárolását is, sőt pointerekkel ezek között kapcsolatokat is lehet ábrázolni.

5.3.1 Szegmenssek

A legelterjedtebb hierarchikus rendszerek terminológiájában a faszervezet csomópontjainak *szegmens* az elnevezése. Minden szegmens két részből tevődik össze:

- az adott konkrét adatbázisrekordban az azon a hierarchiaszinten előforduló elem aktuális értéke és
- az adatbázis következő szegmensére vagy szegmenseire mutató pointer, valamint a hierarchia szintjére és a szegmens helyére jellemző egyéb, az adatbázis-kezelő rendszer számára szolgáló információk.

A felhasználó számára elérhető legkisebb önálló adategység a szegmens. Ebből azonban általában csak a tényleges adatrészt bocsájta rendelkezésére a munkaterületen az adatbázis-kezelő rendszer.

5.3.2 A hierarchikus adatbázis leírása

A fizikai adatbázis leírásának, a fizikai sémának (IMS terminológiában az adatbázisleírásnak, DBD=Data Base Description) tartalmaznia kell a fizikai adatbázisrekordtípus összes szegmenstípusának meghatározását a köztük fennálló gyermek–szülő kapcsolatok leírásával együtt. Az egyes felhasználók azonban a teljes rekordoknak csak az alsémában, IMS terminológiában a programkommunikációs blokkban (PCB) meghatározott szegmenseihez, az ún. *érzékeny szegmensek*hez férhetnek hozzá. Általában mind az adatbázis-leírást, mind a programkommunikációs blokkot az adatbázis-felügyelő készíti el. Ezek a következőket tartalmazzák:

- a) a szegmenssek közötti logikai kapcsolatok (szülő–gyermek),
- b) a fizikai összekapcsolásuk (hierarchikus vagy szülő–testvér, illetve egy- vagy kétirányú pointer),
- c) a szegmenssek egyes mezőinek részletes leírása (a mezők hossza és helye a szegmensben belül, ábrázolási módja, mely mező(k) alapján vannak rendezve azonos szegmenstípuson belül az egyes előfordulások),
- d) az adott program milyen szegmenstípusokhoz férhet hozzá egyáltalában, és azokon milyen jellegű műveleteket végezhet (pl. csak olvashatja, olvashatja és módosíthatja, törölheti stb.),
- e) a szegmenssek fizikai elhelyezése (blokkolás, elérési mód, adott szempontok szerint fizikai „közellét” biztosítása egyes szegmenstípusok, illetve ugyanazon rekord különböző szegmensei között).

Hierarchikus rendszerekben az alapszegmenst, a fa gyökerét mindig azonosítja egy kulcs, melynek alapján a teljes rekord elérhető. Az alapszegmens elérésére pl. az IMS rendszer négy különböző lehetőséget biztosít;

- szekvenciális,
- indexszekvenciális,

- közvetlen és
- indexelt–direkt hozzáférési módot.¹

Egyszerűbb rendszerek általában eme elérési módok valamelyikét teszik csupán lehetővé a felhasználó számára.

Az IMS által kezelt szervezési módokban a HSAM-nál az alapszegmensek és ezek mindegyikéhez külön–külön sorrendben kapcsolódva az összes alája rendelt szegmens is egy szekvenciális állományban helyezkedik el. Az egyes szegmensek fizikai sorrendje teljes mértékben megegyezik a hierarchiából adódó sorrenddel.

A HISAM-nál az alapszegmensek tárolása kulcsok alapján egy indexszekvenciális, HDAM-ban egy direkt szervezésű állományban történik. Ez utóbbiban a kulcs és a tárolóhely közti kapcsolatot hashinggel kapjuk meg, HIDAM-ban egy az alapszegmens kulcsára indexelt indextáblázatból állapítjuk meg a rekord tárolási helyét. Mindhárom szervezési módnál az alapszegmensek és az egyes alapszegmensek ugyanazon előfordulásához tartozó gyermekek részben vagy teljesen fizikailag különálló területeken helyezkednek el. A különböző területeken tárolt szegmensek közötti kapcsolatokat az adatbázis-kezelő rendszer által felépített és karbantartott pointerek hozzák létre.

5.3.3 Műveletek hierarchikus adatbázisban

A hierarchikus modell adatkezelő nyelve utasításokat tartalmaz adatok beolvasására, bevitelére, törlésére és módosítására. Ahhoz, hogy egy adott rekordot módosítani, törölni tudjunk, azt először azonosítanunk kell. Ez rendszerint a szegmenst azonosító kulcs értéke alapján történik. A GET UNIQUE jellegű művelet beolvassa az adatbázis első olyan rekordját, amelyik a feltételnek megfelel. A további, a feltételnek megfelelő rekordokat vagy a GET NEXT vagy a GET NEXT WITHIN PARENT típusú utasításokkal találjuk meg. Az előbbi a hierarchia ugyanazon szintjén adja a következő rekordot, az utóbbi pedig ismételve egy adott szülőnek az összes gyermekét szolgáltatja egymás után.

Amennyiben az adott feltételnek megfelelő rekord nincsen, vagy nincs már több, akkor azt a rendszer jelzi a felhasználó felé.

A legtöbb rendszerben ezen kívül lehetőség van másodlagos indexek felépítésére is. Ezek segítségével kereshetünk nem kulcsként definiált értékekre is, például a megrendelés dátuma szerint.

A hierarchikus adatbázis-kezelő rendszerekben leggyakrabban az 5.1 táblázatban összefoglalt műveleteket lehet elvégezni. Mivel a legtöbb rendszer angol nyelvű utasításokkal dolgozik, a táblázatban és a szövegben is az angol elnevezéseket, illetve azok rövidítéseit használjuk. Ezek az utasítások formájukban és funkcióikban megegyeznek az IBM által korábban forgalmazott adatbázis-kezelő rendszerekben felhasznált adatkezelő nyelvnek, a DL/I-nek az utasításaival.

¹ Az angol Hierarchical Sequential, Indexed Sequential, Direct, illetve Indexed Direct Access Method elnevezés rövidítése alapján az egyes elérési módokat HSAM, HISAM, HDAM és HIDAM-nak szokás hívni.

5.1. táblázat

Egyszerű hierarchikus rendszerek adatkezelő nyelvének leggyakrabban használt utasításai.

Utasítás	Rövidítés	Funkció
GET UNIQUE	GU	Meghatározott tulajdonságú szegmens beolvasása
GET NEXT	GN	A hierarchia alapján soron következő szegmens beolvasása ^{1,2}
GET NEXT WITHIN PARENT	GNP	Az ugyanazon közvetlen szülőhöz tartozó soron következő szegmens beolvasása ^{1,2}
GET HOLD UNIQUE	GHU	Ugyanaz, mint GU, de a szegmenst ideiglenesen elzárja más program elől ³
GET HOLD NEXT	GHN	Ugyanaz, mint GN, de a szegmenst ideiglenesen elzárja más program elől ³
REPLACE	REPL	Egy korábbi GET HOLD... típusú utasítást követve visszahelyezi a módosított szegmenst
DELETE	DLET	Egy korábbi GET HOLD... típusú utasítást követve törli a szegmenst
INSERT	ISRT	Egy új szegmenst helyez el az adatbázisba

Megjegyzések

¹ A következő mindig az adatbázisban utoljára elvégzett műveletben szereplő szegmenshez viszonyítva értendő

² Amennyiben nincs már több rekord, úgy ezt megfelelő státuskóddal jelzi a rendszer.

³ Módosításhoz vagy törléshez szükséges az ilyen típusú behozatali utasítást használni. Ez biztosítja, hogy más programok ennek a szegmensnek egyértelműen és nyomon követhetően a változás előtti vagy utáni formájához férjenek hozzá.

Az egyszerű hierarchikus modell legnagyobb előnye az, hogy a hierarchikus felépítésű valóságnak igen gyakran jól követhető, természetes leképezését adja. Akkor, amikor az adatmodell által ábrázolt egyszerű hierarchia megegyezik a feladat jellegéből adódó természetes hierarchiával, a rendszer hatékonysága igen jó.

Ugyanazon adatok között azonban többféle szempont szerint valósíthatunk meg alá-, fölé- és mellérendelési kapcsolatokat. Mihelyt eltérnek a feladatból adódó kapcsolatok a modellben alkalmazott hierarchiától, olyan problémákat kell az adatbázis-kezelő rendszernek vagy magának a felhasználónak megoldania, melyek nem a feladat bonyolultságából, hanem a modell nem megfelelő voltából adódnak. E problémák megoldása pedig elsősorban a rendszer hatékonyságát csökkenti, és egyszerű alkalmazhatóságát nehezíti meg.

5.4 Példa hierarchikus adatbázisra

Az egyszerű hierarchikus modellből adódó problémák illusztrálására tekintsük az alábbi, egyszerű adatmodellt.

Egy gépjármű-nyilvántartásban a gépjárművekről és a tulajdonosokról az alábbi adatokra van szükségünk:

A gépjármű

- rendszáma,
- gyártmánya,
- típusa,
- színe,
- motorjának űrtartalma,
- egyéb jellemzői.

Minden korábbi, vagy a jelenlegi tulajdonosról a

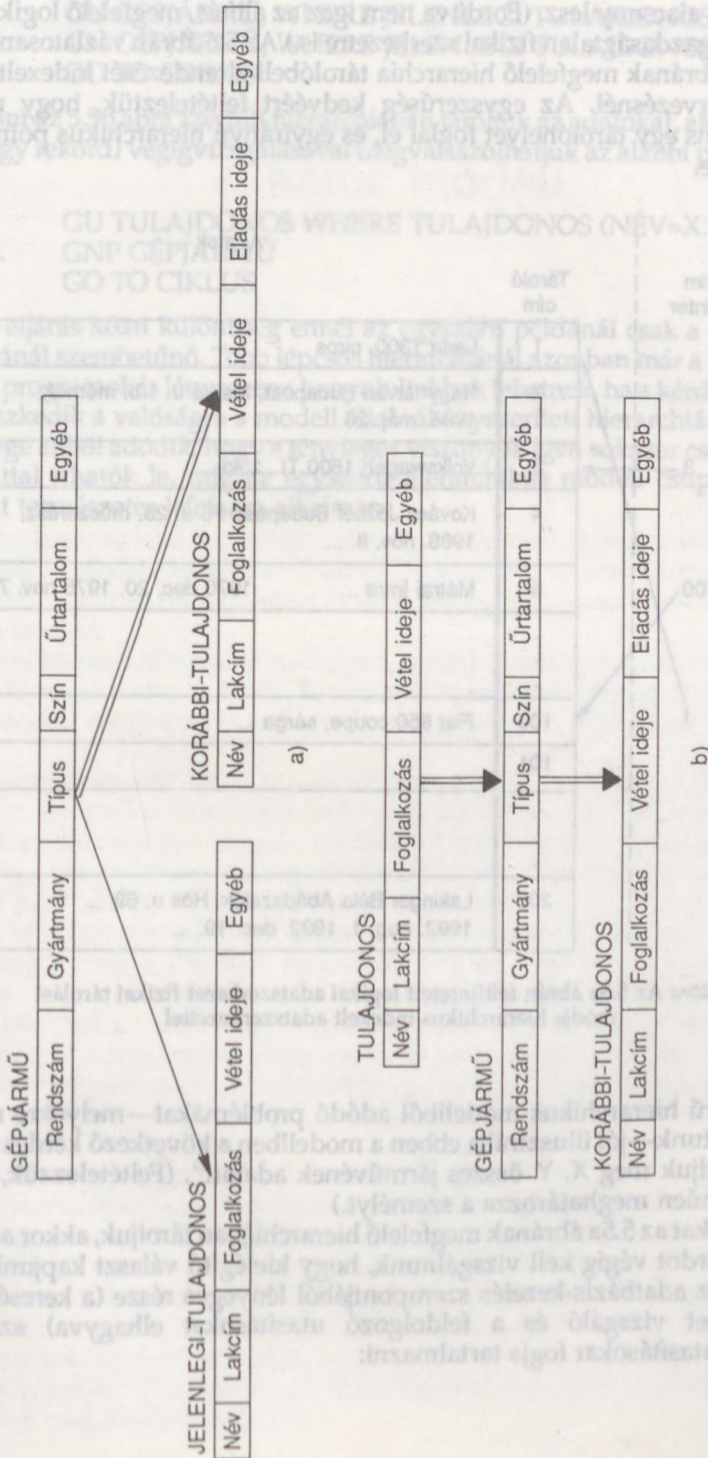
- neve,
- állandó lakcíme,
- jelenlegi foglalkozása,
- ettől-meddig volt a tulajdonában a gépjármű,
- egyéb információk.

(Az egyszerűség kedvéért mind a gépjárműveknél, mind a tulajdonosoknál a példa szempontjából érdektelen adatokat, mint pl. alváz- és motorszám, közlekedési kihágások stb. az „egyéb” gyűjtőfogalomba soroltuk be.)

Ennek a feladatnak két lehetséges hierarchikus ábrázolásmódját láthatjuk az 5.5 ábrán. Az a) ábrán az alapszegmens a GÉPJÁRMŰ szegmens, és ennek közvetlen gyermekei a JELENLEGI-TULAJDONOS és a KORÁBBI-TULAJDONOS szegmensek, míg a b) ábrán az alapszegmens a TULAJDONOS, ennek közvetlen gyereke a GÉPJÁRMŰ szegmens, mely közvetlen szülője a KORÁBBI-TULAJDONOS-nak.² Hogy a különféle lehetséges hierarchikus szervezési és ábrázolási módok közül melyiket célszerű alkalmazni, az a feldolgozás jellegétől függ. Az a) ábrán látható módnál az olyan jellegű kérdéseket tudjuk viszonylag gyorsan megválaszolni, melyeknél a kiválasztás a rendszám alapján történik (pl. adjuk meg egy gépjármű jelenlegi tulajdonosának nevét, vagy az összes korábbi tulajdonosának lakcímét), míg a b)-nél azokat, melyeknél a tulajdonos neve alapján kell megszerezni bizonyos információkat [pl. X. Y.-nak milyen gyártmányú gépkocsija(i) van(nak), vagy kitől vásárolta a gépkocsiját]. Az 5.5 ábrán bemutatott hierarchikus modellek egyikében sem tudunk hatékonyan felelni pl. az alábbi kérdésre: „Soroljuk fel, milyen gépkocsik voltak eddig X. Y. birtokában”. (Természetesen a rendelkezésre álló adatok másféle hierarchiában való elrendezésével, más kérdésfajták lassúbb megválaszolásával erre is hatékonyan tudnánk válaszolni.)

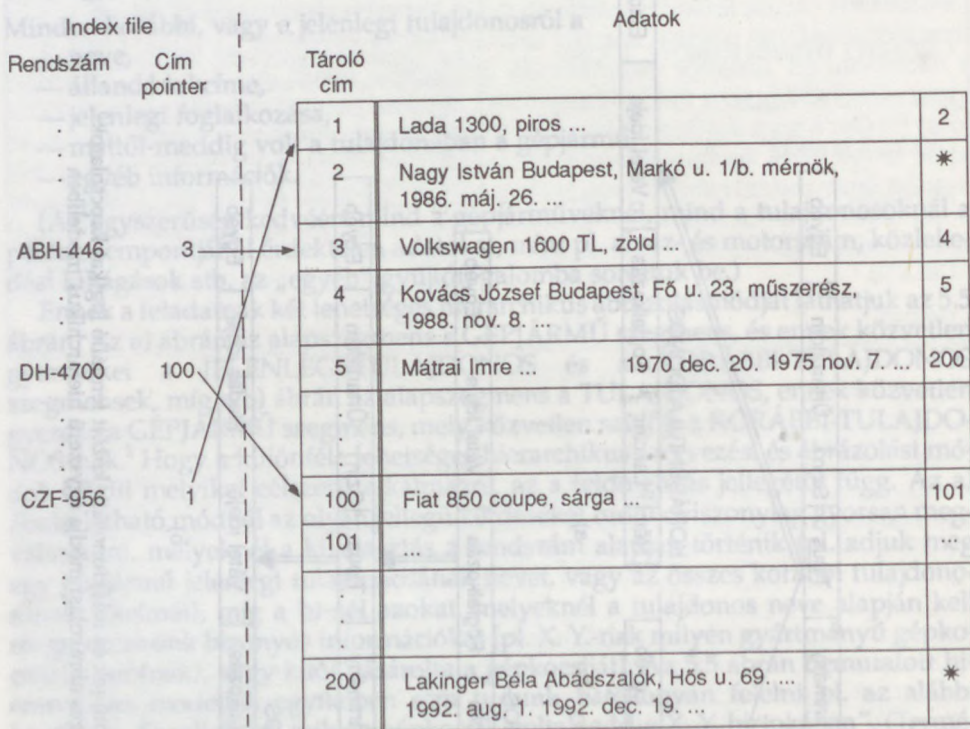
A logikai kapcsolatok végső fokon minden rendszerben fizikai kapcsolatokra épülnek fel, így a visszakeresés hatékonysága nemcsak a logikai szerkezettől függ, hanem a tényleges tárolóbeli elhelyezéstől is. Általában igaz azonban, hogy ha az adatok logikai szerkezete nem illeszkedik a információigényekhez, akkor a rend-

² Mindkét ábrázolásmódnál feltételezzük, hogy a gépkocsinak egy adott időpontban, csak egy tulajdonosa lehet, és a b) ábránál azt is, hogy a név egyértelműen azonosítja a tulajdonost. Az utóbbinál további problémát okozhat az is, hogy egy rekordban a korábbi tulajdonos azonos lehet egy jelenlegi tulajdonossal is, ami redundanciát vagy a faszervezettől való eltérést jelent. Ha azonban a KORÁBBI-TULAJDONOS-okat eleve más típusú elemnek tekintjük, mint a JELENLEGI-TULAJDONOS-okat (hiszen a névtől eltekintve ugyanazon tulajdonságoknak más időpontbeli értékei jellemzők), akkor ez a probléma nem merül fel.



5.5 ábra Gépjárművek és tulajdonosaik között fennálló logikai kapcsolatok két lehetséges ábrázolási módja egyszerű hierarchikus modellben

szer hatásfoka alacsony lesz. (Fordítva nem igaz az állítás, megfelelő logikai szerkezet épülhet gazdaságtalan fizikai szerkezetre is.) Az 5.6 ábrán vázlatosan feltüntettük az 5.5 ábrának megfelelő hierarchia tárolóbeli elrendezését indexelt-direkt (HIDAM) szervezésnél. Az egyszerűség kedvéért feltételeztük, hogy minden egyes szegmens egy tárolóhelyet foglal el, és egyirányú hierarchikus pointerlánc köti őket össze.



5.6 ábra Az 5.5a ábrán feltüntetett logikai adatszerkezet fizikai tárolási módja hierarchikus indexelt adatszerkezettel

Az egyszerű hierarchikus modellből adódó problémákat—melyekre már korábban is utaltunk—jól illusztrálja ebben a modellben a következő kérdés megválaszolása: „Adjuk meg X. Y. összes járművének adatait”. (Feltételezzük, hogy a név egyértelműen meghatározza a személyt.)

Ha adatainkat az 5.5a ábrának megfelelő hierarchiában tároljuk, akkor az összes adatbázisrekordot végig kell vizsgálnunk, hogy kielégítő választ kapjunk. Programunknak az adatbázis-kezelés szempontjából lényeges része (a keresési eredménytelenséget vizsgáló és a feldolgozó utasításokat elhagyva) az alábbi adatkezelési utasításokat fogja tartalmazni:

CIKLUS. GU GÉPJÁRMŰ, WHERE JELENLEGI TULAJDONOS (NÉV=X.Y.).
GN GÉPJÁRMŰ, WHERE JELENLEGI TULAJDONOS (NÉV=X.Y.).
GO TO CIKLUS

Ha viszont az 5.5b ábra szerinti hierarchiában tároljuk az adatokat, akkor a kérdést egy fa (egy rekord) végigvizsgálásával megválaszolhatjuk az alábbi program alapján:

CIKLUS. GU TULAJDONOS WHERE TULAJDONOS (NÉV=X.Y.).
GNP GÉPJÁRMŰ
GO TO CIKLUS

A két eljárás közti különbség ennél az egyszerű példánál csak a keresés hatékonyságánál szembetűnő. Több lépcsős hierarchiánál azonban már a kiválasztásra szolgáló programok is lényegesen bonyolultabbak lehetnek, ha a kérdés szerkezete nem illeszkedik a valóságra a modell által rákényszerített hierarchiától. A probléma lényege abból adódik, hogy a tényleges viszonyok igen sokszor csak több-több kapcsolattal írhatók le, míg az egyszerű hierarchikus modell csupán egy-több kapcsolat természetes leírására alkalmas.

A hálószerkezet az adatstruktúrában olyan általánosított hierarchikus kapcsolatot jelent, ahol az elemek között nemcsak 1:1, hanem bármilyen, így N:M (több-több) kapcsolat is feltehető és nem kell benne legyen sem kiindulási gyök, sem levelek.

A hálós hierarchiában lehetnek olyan elemek, amelyek mindegyik szint helyén, ahol a kapcsolat nemcsak lefelé, hanem felfelé is többértékű lehet, azaz a hierarchikus modell terminológiáját használva egy elemnek több közvetlen szülője is lehet.

A gyakorlatban előforduló kapcsolatok jelentős része nemcsak több-több kapcsolattal is, így ezúton az információtechnológiában hálószerkezetekkel modellezhető.

A hálós adatmodellen alapuló adatbázisok hasonló szempontokkal szembe kell nézünk van a hierarchikus rendszerűkhöz. Valóban az a fő különbség, mint azt a 6.2 pontban látni fogjuk —, viszonyukok több helyen 1:1 hierarchikus kapcsolatra. Mindeket modellekben az adatbázis rekordokhoz tartozó pontokon a segítségével kapjuk meg a kiinduló rekordokhoz tartozó lefelé kapcsolatot. Mindeket modellekben csak akkor tudunk fel-lefelé vizsgálni az adatbázis rekordok között, ha ismerjük az adatbázis szerkezetét. Ebben az esetben viszont, ha a kiinduló pontot mód az adatbázisban előre ismerjük és pontozatokkal definiáltunk, akkor általában hatékonyabban vagy legalábbis ugyanolyan hatékonyan tudunk fel-lefelé vizsgálni az adatbázist, mint ha egy hiányzó modellben ábrázolnánk és vizsgálnánk az adatokat.

A 6.1 ábrán egy egyszerű hálós szerkezetre látunk példát.

A szerkezet olyan, hogy egy rekordnak lehet több szülője (főleg, címjelölés), mindig ugyanaz az adat foglalkozik. Ezzel az a kapcsolat 1:1 típusú.

Ugyanez áll egy rekordnak a megrendeléseire. Az egyik és jellemzően három egyedi (1:1) kapcsolat van. Ugyanez az áll viszont több megrendelésben és több rekordban is szerepelhet, és különböző módokból is megrendelhető. Ezzel tehát a kapcsolatok N:M jellegűek.

Ezzel következőképpen szemlét az ábrán látható a hálós modellben megkapjuk a szükségesnek megfelelően ábrázolni.

A hálós modell

6.1 Hálós adatszerkezet

A hálószerkezet az adatstruktúrában olyan általánosított hierarchikus kapcsolatot jelent, ahol az elemek között nemcsak 1:N (illetve 1:1), hanem bármilyen, így N:M (több-több) kapcsolat is fennállhat és nem kell benne legyen sem kitüntetett gyökér, sem levelek.

A hálós hierarchiában lehetnek olyan szintek (esetleg mindegyik szint ilyen), ahol a kapcsolat nemcsak lefelé, hanem fölfelé is többértékű lehet, azaz a hierarchikus modell terminológiáját használva egy elemnek több közvetlen szülője is lehet.

A gyakorlatban előforduló kapcsolatok jelentős része tartalmaz több-több kapcsolatot is, így ezeket számítástechnikailag hálószerkezettel modellezzük.

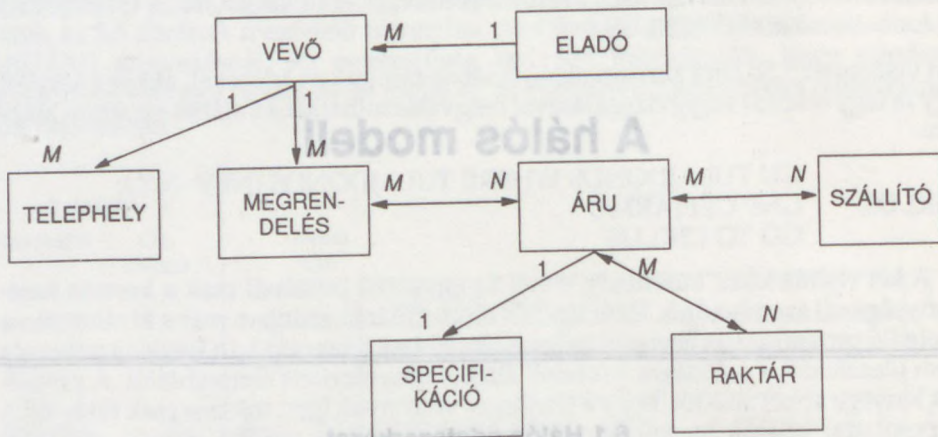
A hálós adatmodellen alapuló adatbázis-kezelő rendszereknek sok közös vonásuk van a hierarchikus rendszerekkel. Valójában a több-több kapcsolatot—mint azt a 6.2 pontban látni fogjuk—, visszavezetik több tiszta 1:N hierarchikus kapcsolatra. Mindkét modellben az adatbázis rekordokba beépített pointerok segítségével kapjuk meg a kiinduló rekordhoz tartozó további rekordokat. Mindkét modellben csak akkor tudunk hatékonyan navigálni az összefüggő rekordok között, ha ismerjük az adatbázis szerkezetét. Ebben az esetben viszont, ha a kívánt elérési mód az adatbázisban előre ismert és pointerokkal definiált volt, akkor általában hatékonyabban, vagy legalább is ugyanolyan hatékonysággal tudunk dolgozni az adatbázissal, mint ahogyha relációs modellben ábrázolnánk és kezelnénk az adatokat.

A 6.1 ábrán egy egyszerű hálós szerkezetre látunk példát.

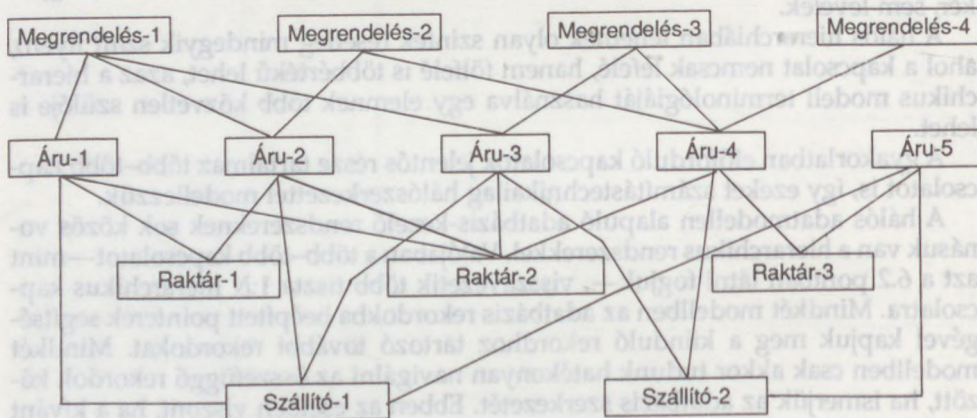
A szervezés olyan, hogy egy vevővel, akinek több telephelye (fiókja, címe) lehet, mindig ugyanaz az eladó foglalkozik. Ezért ez a kapcsolat 1:N típusú.

Ugyanez áll egy vevőnek a megrendeléseire. Az áruk és specifikációk között egyértelmű (1:1) kapcsolat van. Ugyanez az áru viszont több megrendelésben és több raktárban is szerepelhet, és különböző szállítóktól is megrendelhető. Ezért ezek a kapcsolatok N:M jellegűek.

Ennek következtében ezeket az összefüggéseket a hálós modellben tudjuk a valóságnak megfelelően ábrázolni.



6.1 ábra Komplex hálószerkezet az eladók, vevők, megrendelések, áruk, szállítók és raktárak közötti kapcsolatok modellezésére



6.2 ábra A 6.1 ábrán látható adatmodellben a több-több kapcsolattal leírható rekordtípusok konkrét előfordulásainak és a közöttük fennálló kapcsolatoknak ábrázolása

A 6.2 ábrán bemutatjuk ezen modell hálós részének egy aktuális megvalósítását. Itt láthatjuk, hogy például a Raktár-2-ben tárolnak Áru-1, Áru-2, Áru-3 és Áru-4-et is, de az Áru-2-t megtaláljuk a Raktár-1-ben, az Áru-4-et pedig a Raktár-3-ban is. Az ÁRU és RAKTÁR rekordtípusok között tehát több-több kapcsolat áll fenn.

6.2 Kapcsolórekordok

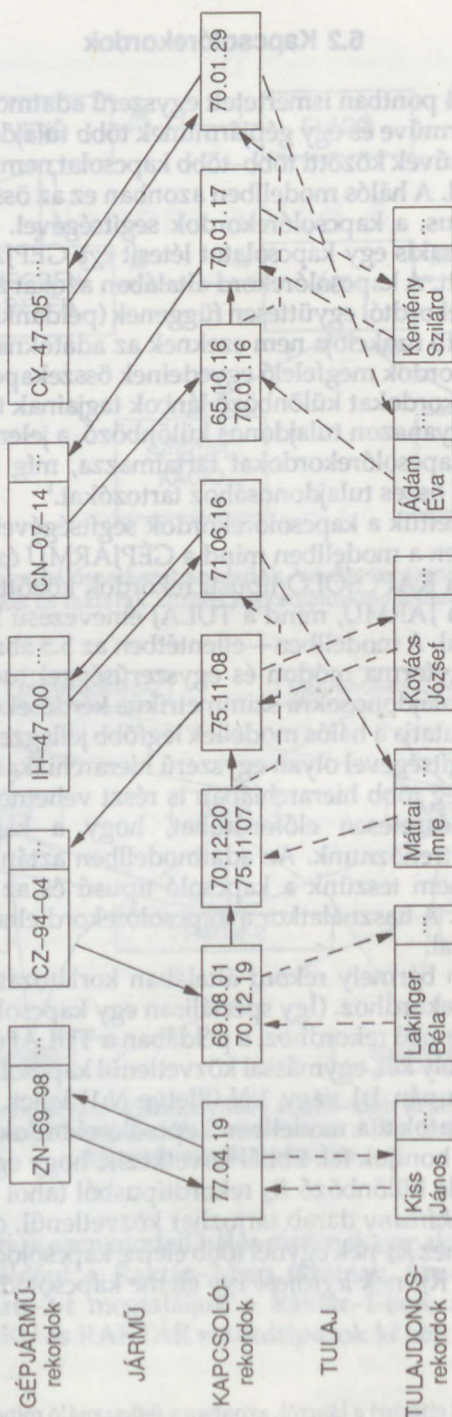
Tekintsük ismét az 5.4 pontban ismertetett egyszerű adatmodellt. Mivel egy tulajdonosnak több gépjárműve és egy gépjárműnek több tulajdonosa lehetett, a tulajdonosok és a gépjárművek közötti több–több kapcsolat nem ábrázolható egyszerű hierarchikus modellel. A hálós modellben azonban ez az összefüggés is kezelhető egy új csomóponttípus, a kapcsolórekordok segítségével. Ennek minden egyes előfordulása egy és csakis egy kapcsolatot létesít egy GÉPJÁRMŰ és egy TULAJDONOS rekord között. A kapcsolórekord általában azokat az adatokat tartalmazza, melyek mindkét rekordtól együttesen függenek (példánkban a tulajdonviszony kezdete és vége). De fő funkciója nem ezeknek az adatoknak a tárolása, hanem a különböző típusú rekordok megfelelő egyedeinek összekapcsolása. Logikai szempontból a kapcsolórekordokat különböző láncok tagjainak tekinthetjük. Az egyik lánc típus (TULAJ) ugyanazon tulajdonos különböző, a jelenlegi és a korábbi gépkocsijaihoz tartozó kapcsolórekordokat tartalmazza, míg a másik (JÁRMŰ) az ugyanazon gépjármű összes tulajdonosához tartozókat.¹

A 6.3 ábrán feltüntettük a kapcsolórekordok segítségével létrehozott hálószerkezet egy részét. Ebben a modellben mind a GÉPJÁRMŰ és a KAPCSOLÓ, mind a TULAJDONOS és a KAPCSOLÓ típusú rekordok között 1:N jellegű kapcsolat áll fenn. Ezért mind a JÁRMŰ, mind a TULAJ elnevezésű kapcsolat ábrázolható egyszerű hierarchiával. A modellben—ellentétben az 5.5 ábrán ábrázolt hierarchikus modellekkel—egyforma módon és egyszerűséggel tudjuk megválaszolni a gépjárművekre és a tulajdonosokra szimmetrikus kérdéseket.

A fenti példa jól mutatja a hálós modellek legfőbb jellegzetességét. A modellben kapcsolórekordok segítségével olyan egyszerű hierarchiákat alakítunk ki, melyeknek elemei egyidejűleg több hierarchiában is részt vehetnek. Bonyolult adatösszefüggéseknél természetesen előfordulhat, hogy a kapcsolórekordokból is többféle típust kell létrehoznunk. Az adatmodellben aztán a továbbiakban semmiféle különbséget nem teszünk a kapcsoló típusú és az eredetileg is fennállt rekordtípusok között. A használatkor a kapcsolórekord elnevezés legfeljebb a rekord keletkezésére utal.

A hálós modellben bármely rekord általában korlátozás nélkül kapcsolódhat több különböző típusú rekordhoz. (Így speciálisan egy kapcsolórekord is kapcsolódhat több különböző típusú rekordhoz, a példában a TULAJ és a JÁRMŰ összefüggés alapján.) De bármely két, egymással közvetlenül kapcsolatban álló rekordtípus között a kapcsolat csupán 1:1 vagy 1:N (illetve N:1) lehet. A valóságban esetleg fennálló $M:N$ kapcsolatokat a modellben kapcsolórekordok közbeiktatásával 1:N jellegű kapcsolatokra bontjuk fel. Ebből következik, hogy egy R_1 rekordtípus bármely tagjához egy tőle különböző R_2 rekordtípusból (ahol R_1 és R_2 is lehet kapcsolórekord) elvileg *akárhány* darab tartozhat közvetlenül, de ha létezik R_1 -nek olyan r_1 eleme, amelyhez R_2 -nek egynél több eleme kapcsolódik közvetlenül, akkor R_2 bármely eleméhez R_1 -nek *legfeljebb egy* eleme kapcsolódhat közvetlenül.

¹ A fizikai tárolási mód eltérhet a lánctól, azonban a felhasználó mindig tekintheti úgy, mintha az láncként valósult volna meg.



6.3 ábra A gépjármű nyilvántartás (egy részének) ábrázolása hálós modellel
(az áttekinthetőség érdekében a láncoknak is csak egy részét tüntettük fel)

6.3 A DBTG javaslat

Az adatbázis-kezelő rendszerek egységesítése céljából a CODASYL-bizottság egy speciális munkacsoportot (Data Base Task Group, DBTG) hozott létre. Ez a csoport tanulmányozta és összehasonlította a már meglevő adatbázis-kezelő rendszereket, majd egy jelentést készített, melyben javaslatot tett arra, hogy milyen tulajdonságokkal kell rendelkeznie az adatbázis-kezelő rendszernek. Ennek első változata 1969-ben került nyilvánosságra, majd ezt követte 1971-ben a végleges jelentés, melyet készítőikről DBTG-reportnak, DBTG-jelentésnek vagy javaslatnak szokás nevezni. A jelentés elkészítése után a DBTG feloszlott, és helyébe két új bizottság lépett, a Data Base Language Task Group és a Data Definition Language Committee, melyeknek feladata volt a javaslat részletes kidolgozása és a COBOL, valamint egyéb programnyelvekbe való beillesztése.

A DBTG célja egy olyan rendszer definiálása volt, amely a következő tulajdonságokkal rendelkezik:

- Minden alkalmazáshoz a legmegfelelőbb struktúrában szolgáltatja az adatokat, függetlenül attól, hogy ugyanazon adatokat vagy azok egy részét máshol is felhasználják. Ez a flexibilitás redundancia nélkül érendő el.
- Több felhasználó, ill. program is hozzáférhet egyidejűleg az adatbázishoz.
- Különbőféle keresési stratégiák használhatók a teljes adatbázisban, illetve annak egyes részein.
- Megvédi az adatbázist illetéktelen felhasználóktól és programoktól.
- Biztosítja az adatok elhelyezésének központi ellenőrzését.
- A programokat függetleníti az adattárolóktól.
- Lehetővé teszi a különféle adatstruktúrák deklarálását, a legegyszerűbb, egy mástól független adatoktól kezdve a bonyolult hálóstruktúráig.
- A felhasználó úgy dolgozhat az adatokkal, hogy nem kell közben törődnie a deklarált struktúrális kapcsolatok karbantartásával.
- A programokat a lehető legnagyobb mértékben függetleníti az adatoktól.
- Az adatok leírása az adatbázisban és a programban függetlenül legyen egymástól.
- Az adatbázis leírása ne korlátozódjék egy programnyelvre.
- Az adatbázis és leírása különböző programnyelvekhez illeszthető legyen.

E célok elérésére három nyelvre tettek javaslatot, ezek:

- a sémaleíró nyelv,
- az alsémaleíró nyelv,
- az adatkezelő nyelv.

- a) A *sémaleíró nyelv* leírja az adatmodellt és fizikai tárolási módját. (Bizonyos fizikai hozzárendeléseket, pl. a perifériákat nyitva hagyja, ezeket futás folyamán lehet ténylegesen meghatározni.) Ez egy önálló szintaktikával rendelkező programnyelv is lehet.
- b) Az *alsémalírás* kapcsolatot létesít a séma és a felhasználói program között, ezért szintaxisának összhangban kell lennie azokkal a programnyelvekkel, melyekből az adatbázis-kezelő rendszer hívható. Mivel a javaslat készítői a legnagyobb felhasználók érdekeit tartották szem előtt, a DBTG-javaslatban specifikált nyelv a COBOL-hoz illeszkedik.

c) Ugyancsak a COBOL-hoz kapcsolódik az általuk részletesen kidolgozott *adatkezelő nyelv* is, melynek utasításait ugyanúgy lehet használni az adatbázis kezeléssel bővített befogadó nyelvben, mint annak eredeti utasításait.²

A hálós modell sémája lényegében megfelel az általános adatmodell terminológiában a koncepcionális szintnek, alsémája pedig az adatmodell külső szintjének (l. 3.1 pont). Ugyanakkor a DBTG által javasolt—és a gyakorlatban használt hálós adatbázis-kezelő rendszerekben implementált—adatleíró nyelv (DDL) már messze nem marad meg ezen a szinten. Mint azt a 6.4 pontban látni fogjuk, ebben a nyelvben igen konkrét útmutatást kell adnunk az adatok tárolási és elérési módjára, tehát komoly mélységig meghatározzuk az adatbázis belső, fizikai szintjét is.

A példákban a DBTG által javasolt szintaxist, illetve az egyik legelterjedtebb hálós adatbáziskezelő rendszer, a Cullinet Software Inc. által forgalmazott IDMS (Integrated Database Management System) utasításait használjuk.

6.4 Sémaleírás

A sémaleírás négy főrészből áll:

- a séma elnevezése,
- a területek leírása,
- a rekordok leírása, valamint
- a set-ek definiálása és leírása.³

Ezek sorrendje bizonyos mértékig kötött. A séma elnevezés mindig az első. A többiekénél az egyes területek, rekordok, set-ek leírásai váltogathatják egymást, de a területleírásnak meg kell előznie mindazon rekordok leírását, melyek ezen a területen helyezkednek el, és bármely set-et csak összes rekordjának definiálása után adhatunk meg.

A 6.4 ábrán az eddigi példáinkban is szerepelt gépjárműnyilvántartás sémájának leírását láthatjuk. A példában a DBTG által eredetileg javasolt szintaxist használjuk. Az egyes adatbázis-kezelő nyelvek utasításai ettől formailag némileg eltérhetnek.

Az 1-es deklaráció a séma nevét (GEPJARMU-NYILVANTARTAS) adja meg, 1a és 1b pedig meghatározza azokat a feltételeket, melyekkel a séma megváltoztatható, illetve átmásolható alsémákba.

Amennyiben a program meg kívánna változtatni a sémát (ez természetesen nem azonos az adatbázis egyes adatainak a megváltoztatásával), akkor annak eleget kell tennie az ELLENORZES nevű eljárásnak, illetve meg kell adnia a LE-

² Természetesen a kereskedelmi forgalomba kerülő hálós adatbázis-kezelő rendszerek többsége később beilleszthető volt más programnyelvekbe is, így pl. PL/I-be vagy Assemblerbe.

³ A „terület”-et a 6.4.3 pontban a DBTG-jelentésben használt *area* kifejezés magyar megfelelőként használjuk. A set helyett azonban nem tartjuk célszerűnek a magyar „halmaz” használatát, mert—mint később látni fogjuk—például az üres set jellemzője, hogy csak egy eleme van. Megfelelő fordítása volna a „kapcsolat” vagy „összefüggés”, azonban a magyar szaknyelvben még egyik sem honosodott meg. Ezért megmaradtunk az angol szónál, de ugyanakkor felhívjuk a figyelmet arra, hogy a DBTG-javaslatban szereplő „set” fogalmat ne tévesszük össze az IBM terminológiában használatos „data set”-tel.

- 1 SCHEMA NAME IS GEPJARMU-NYILVANTARTAS;
- 1a PRIVACY LOCK FOR ALTER IS PROCEDURE ELLENORZES
- 1b PRIVACY LOCK FOR COPY IS 'LEMASOLHAT'.
- 2 AREA NAME IS TULAJDONOS-TER.
- 2a AREA NAME IS GEPJARMU-TER;
- 2b ON OPEN FOR UPDATE CALL MERKUR.
- 3 RECORD NAME IS GEPJARMU;
- 3a LOCATION MODE IS CALC RENDSZAM-HELY USING RENDSZAM
- 3b DUPLICATES ARE NOT ALLOWED;
- 3c WITHIN GEPJARMU-TER.
- 3d 02 RENDSZAM.
 - 03 BETUJEL; PICTURE IS A (2).
 - 03 KOTOJEL1; PICTURE IS X.
 - 03 SZAMJEL.
 - 04 SZAM1; PICTURE IS 9 (2).
 - 04 KOTOJEL2; PICTURE IS X.
 - 04 SZAM2; PICTURE IS 9(2).
- 3e 02 GYARTMANY; PICTURE IS X(12).
- 3f 02 TIPUS; PICTURE IS X(12).
- 3g 02 SZIN; PICTURE IS X(10).
- 3h 02 URTARTALOM; PICTURE IS 9(4).
- 3i 02 EGYEB.
 - 03...
 - :
- 3j RECORD NAME IS TULAJDONOS;
- 3k LOCATION MODE IS CALC NEV-HELY USING NEV
- 3l DUPLICATES ARE NOT ALLOWED;
- 3m WITHIN TULAJDONOS-TER.
 - :
- 3n RECORD NAME IS KAPCSOLAT;
- 3o LOCATION MOD IS VIA TULAJ SET;
- 3p WITHIN TULAJDONOS-TER.
- 3q 02 METTOL.
 - 03 EV; PICTURE 9(4).
 - 03 HO; PICTURE 9(2).
 - 03 NAP; PICTURE 9(2).
- 3u 02 MEDDIG.
 - :
- 4 SET NAME IS JARMU;
- 4a MODE IS CHAIN;
- 4b ORDER IS SORTED;
- 4c OWNER IS GEPJARMU
- 4d MEMBER IS KAPCSOLO
- 4e MANDATORY AUTOMATIC;
- 4f ASCENDING KEY IS METTOL DUPLICATES ARE NOT ALLOWED;
- 4g SET OCCURENCE SELECTION IS THRU LOCATION MODE CF OWNER.
- 4h SET NAME IS JARMU-NYILVANTARTAS;
- 4i MODE IS CHAIN,
- 4j ORDER IS SORTED;
- 4k OWNER IS SYSTEM
- 4l MEMBER IS GEPJARMU
- 4m MANDATORY AUTOMATIC;
- 4n ASCENDING KEY IS RENDSZAM DUPLICATES ARE NOT ALLOWED.
- 4o SET NAME IS TULAJ;
- :

6.4 ábra Sémaleírás a DBTG által javasolt sémaleíró nyelvben

MASOLHAT jelszót, ha a sémát vagy annak részeit át kívánja másolni az alsémába. Mindkettőt az adatbázis-felügyelő adta meg az adatbázis védelme és titkosságának biztosítása érdekében. Hasonló védelmet lehet biztosítani a séma kiírása és az ellenőrzést nyújtó zárok megváltoztatása ellen is.

Az angol nyelv szavait felhasználó adatleíró nyelvben a sémát biztosító záradéknak a teljes szintaxisa:

PRIVACYLOCK FOR cselekedet IS $\left\{ \begin{array}{l} \text{jelsorozat} \\ \text{zárnév} \\ \text{eljárás} \end{array} \right\}$

ahol a cselekedet a séma megváltoztatását (ALTER), átmásolását (COPY), kiírását (DISPLAY) vagy a zárok megváltoztatását (LOCKS) jelenti.

Az adatbázist az adatbázis-felügyelő feloszthatja egy vagy több területre (AREA).

Egy terület lehet pl. egy konvencionális értelemben vett állomány, egy lemez-csomag, egy kijelölt cylinder stb. A területek és a bennük tárolt adatok megfelelő kijelölésével az adatbázis-felügyelő meggyorsíthatja a rendszer működését (gyakran keresett adatok számára gyorsan elérhető területet jelöl ki, együtt keresett adatokat egymáshoz fizikailag is közel helyez el stb.), és nagymértékben megnövelheti biztonságát (pl. fontos adatokat egy területre gyűjt össze, melyeket külön biztonsági intézkedésekkel, jelszóval vagy eljárással véd, gyakran menti ki stb.). Egyes rendszerekben lehetőség van arra is, hogy az adatbázis-felügyelő konkrétan megadhassa, melyik fizikai adatállomány milyen rekordjait foglalják magukban az egyes területek.

A 2-es deklaráció a TULAJDONOS-TER, a 2a a GEPJARMU-TER területet definiálja. (A példánkban kizárólag a lehetőségek bemutatása céljából definiáltunk két különböző területet.) Ez utóbbit a 2b definíció következtében csak azok a programok tudják módosítani (UPDATE), melyek eleget tesznek az adatbázis-felügyelő által meghatározott MERKUR eljárás követelményeinek. Az ellenőrzés a felhasználói program futásakor, a területre való írást megelőző OPEN megnyitó utasítás végrehajtásakor történik.

A rekordleírások (3–3u) megadják az egyes rekordtípusok nevét, elérési, illetve az adatbázisba való behelyezési módját, vagyis hogy melyik területen legyenek, és hogy milyen mezőkből állnak.

A 3-as deklaráció a GEPJARMU elnevezésű rekordot írja le. 3a–3c-ben definiáljuk, hogy az egyes rekordok elérése hashing-gel (lásd 4.8) történik (LOCATION MODE IS CALC) a RENDSZAM-HELY eljárással megadott hashing-algoritmussal a RENDSZAM adatelem aktuális értéke alapján (USING) a RENDSZAM-TER elnevezésű területen belül (WITHIN). Kikötjük még, hogy két különböző kocsinak nem lehet azonos rendszáma (DUPLICATES ARE NOT ALLOWED). Ennek következményeképpen, ha olyan GEPJARMU rekordot akarunk behelyezni az adatbázisba, melynek rendszáma megegyezik egy már bentlevő rekord rendszámával, akkor a rendszer hibát jelez. A további deklarációk (3d–3i) a GEPJARMU rekord egyes mezőit, a rekord egyes adatait írnak le. A PICTURE IS bejegyzés megadja a mező hosszát és a tárolt karakterek jellegét. (A=alfabetikus, 9=numerikus, X=alfanumerikus; az utána zárójelben levő szám, illetve a jelölés megismétlődéseinek

száma adja meg a mező hosszát.) Adatelemeket csoportba is összevonhatunk. Ennek bemutatására pl. a RENDSZAM csoportmezőt több elemi mezőből építettük fel, bár más módon—pl. a nyilvánvalóan felesleges kötőjelek elhagyásával—tömörebben, kisebb helyen tudtuk volna tárolni. A 3j–3m deklarációk ugyanezeket adják meg a TULAJDONOS rekordtípusra. Itt is fenntartottuk azt az egyszerűsítő feltevésünket, hogy a NÉV egyértelműen azonosítja a tulajdonost.

A 3n–3u deklarációk a GEPJARMU és a TULAJDONOS típusú rekordok közti kapcsolatot fenntartó KAPCSOLAT típusú rekordokat írják le, melyeket az adatbázis–felügyelő valamilyen egyéb megfontolás alapján a TULAJDONOS-TER területen kíván elhelyezni (WITHIN) a velük kapcsolatban álló tulajdonosrekord közelében (VIA TULAJ SET, 3p–3o).

6.5 A set koncepciója

A DBTG-javaslat egyik leglényegesebb pontja a *set* koncepciójának megalkotása volt. A *set*-ek segítségével lehet az adatbázisban levő adatok között különféle hierarchikus és hálós kapcsolatokat létrehozni.

Legegyszerűbb formájában a *set* két rekordtípusból épül fel, melyek mindegyike több mezőt tartalmazhat. Az egyik rekordtípus a *tulajdonos*, a másik a *tag*.⁴ A *set* minden egyes előfordulása pontosan egy darab tulajdonos típusú rekordelőfordulást tartalmaz, melyhez tetszés szerinti számú tag típusú rekord előfordulása kapcsolódik. (Ha a tagok száma zérus, akkor üres *set*-ről beszélünk.) A tulajdonos- és a tagrekordok előfordulásai között tehát egy egyszerű hierarchikus viszony áll fenn, melyben a tulajdonosrekord van a hierarchia magasabb szintjén. A hierarchiából következik, hogy *ugyanazon set*-en belül egy tag *csak egy* tulajdonoshoz tartozhat. Ez természetesen nem zárja ki, hogy egy *más típusú set*-ben ugyanaz a rekord-előfordulás ne tartozhasson egy másik tulajdonosrekordhoz. Így például a 6.3 ábrán a JÁRMŰ *set* kapcsolatot létesít a GÉPJÁRMŰ típusú rekordok, mint tulajdonos és a KAPCSOLÓ típusú rekordok, mint tagok között. Az ábrán a *set* 5 előfordulása látható, melyek tulajdonos-rekordjai a ZN–69–98, CZ–94–04, ..., CY–47–05 rendszámú gépkocsik. Az első, harmadik és ötödik *set* előfordulásban csak egy–egy tag van, a másodikban három, a negyedikben kettő.

Ahogy a hierarchia is lehet többszintes, ugyanúgy lehet az is, hogy egy *set*-ben ugyanazon tulajdonostípushoz *többféle típusú tag* tartozik, valamint az, hogy az egyik *set*-ben tagként szereplő rekordok tulajdonosai egy másik *set*-ben egy vagy több más típusú rekordnak. Ebben az értelemben tehát a *set* megfelel az egyszerű hierarchia faszervezetének, ahol a tulajdonos a fa gyökere és a tagok a közvetlen gyerekei. További tulajdonosok az összes nem levél csomópontok, és tagok a gyökér kivételével az összes csomópontok. A szülő és a közvetlen gyereke(i) közötti

⁴ Az angol szakirodalom az „owner” és „member” kifejezést használja.

kapcsolat a set. Így a csomópontok a gyökér és a levelek kivételével egyidejűleg tulajdonosok is és tagok is, de természetesen két különböző set-ben.⁵

A 6.3 ábrán egy másik set-et, a TULAJ set-et is láthatjuk. Ennek tagjai ugyanazok a KAPCSOLÓ rekordok, melyek a GÉPJÁRMŰ rekordban is tagok, de a tulajdonos itt a TULAJDONOS rekordtípus. A közös tagrekordok segítségével a két set valójában egy hálós adatszerkezetet ír le, illetve egy több-több kapcsolatot hoz létre a GÉPJÁRMŰ és a TULAJDONOS típusú rekordok között.

A set-nek egy speciális fajtája a *szinguláris set*, melynek tulajdonosa a rendszer. Minden szinguláris set-nek csak egy előfordulása lehet, de egy adatmodellben akárhány deklarálható. Így például a szokásos értelemben vett szekvenciális állomány egyúttal egy szinguláris set-nek is tekinthető.

A fenti példa két különböző szinguláris set-et tartalmaz, melyeknek tagjai a GÉPJÁRMŰ illetve a TULAJDONOS rekordok.

Az adatbázis-felügyelő a set-ek definiálásával hozza létre a különböző adatok közötti kapcsolatokat. Éppen ezért a sémaleírásnak minden egyes set részletes leírását is tartalmazni kell, mivel az adatbázis-kezelő rendszer az itt megadott logikai összefüggések alapján állítja elő a fizikai kapcsolatokat. A 6.4 ábrán bemutatjuk a DBTG által javasolt sémaleíró nyelven a 6.3 ábrán látható adatbázis deklarációjának set-definiáló részét is. Ez az ott feltüntetett két set-en kívül tartalmazza az összes gépjármű (szekvenciális) feldolgozását elősegítő JARMU-NYILVANTARTAS szinguláris set-et is.

A 4-es utasítás (SET NAME IS) deklarálja a JARMU set-et, 4a, 4b és 4f megadja, hogy az eladási időpont szerint rendezett (ORDER IS SORTED; ASCENDING KEY IS METTOL) tagrekordokat lánc köti össze (MODE IS CHAIN). 4c és 4d definiálja a tulajdonos (OWNER IS) és a tagrekord (MEMBER IS) típusát. A 4e utasítás a tagoknak a set-be való beillesztési és törlési módját határozza meg. Ez a MANDATORY vagy OPTIONAL és az AUTOMATIC vagy MANUAL lehetőségek négyféle kombinációjának valamelyike lehet.

A MANDATORY (kötelező) specifikáció azt jelenti, hogy ha a rekordot egyszer behelyeztük a set-be, akkor onnan csak a rekordnak az adatbázisból való tényleges törlésével (DELETE) távolítható el. (Természetesen ugyanazon set egy másik előfordulásába átvihető egy módosító MODIFY utasítással.) Az OPTIONAL-lal (opcionális) megadott rekordok eltávolíthatók a set-ből a REMOVE utasítással anélkül, hogy az adatbázisból törölni kelljen őket. (Ezek a rekordok tehát más set-eknek továbbra is tagjai vagy tulajdonosai maradhatnak a törlés után is.) A törléskor e szempontok mellett ügyelnünk kell még arra is, hogy egy tulajdonosrekord törlése automatikusan törli az összes hozzá tartozó tagrekordot is.

Az AUTOMATIC specifikációval definiált set-ben, ha egy új tagot helyezünk el az adatbázisban, akkor azt a megfelelő pointerrek előállításával és módosításával a rendszer automatikusan beilleszti a set-be, míg MANUAL megadása esetén csak külön INSERT utasításra végzi ezt el.

⁵Lényeges az, hogy a tulajdonos és a tag rekord nemcsak különbözik egymástól, hanem különböző típusú is. Ugyancsak nem lehet az, hogy ha egy S_1 set-ben az A típusú rekord a tulajdonos és B a tag, akkor egy S_2 set-ben B legyen a tulajdonos és A a tag. Ilyen jellegű szerkezeteket a DBTG adatmodellben csak kapcsolórekordok közbeiktatásával valósíthatunk meg.

Mivel bármely újonnan beillesztett tagrekordot az adatbázis-kezelő rendszernek be kell tudni helyezni megfelelő set-előfordulásba, ezért definiálni kell azt is, hogy ez a kiválasztás miképpen történjen. Erre szolgál a SET OCCURENCE SELECTION IS THRU klauzula. A különféle megadási módok közül itt csak a két legfontosabbat említjük meg. Az egyik az, amikor a tulajdonost határozzuk meg a sémában leírt módon (LOCATION MODE OF OWNER), és az így kapott tulajdonoshoz tartozó set-előfordulásba helyezzük be tagként az új rekordot. Ezt a megoldást választottuk a 6.4. ábra 4g utasításában. A JARMU set-be egy új KAPCSOLO rekordot tehát abba a set-be illeszti bele a rendszer, melynek a tulajdonosául szolgáló GEPJARMU rekordot úgy választja ki, hogy abban a RENDSZAM mező értéke megegyezzen a felhasználó által erre a mezőre megadott aktuális értékkel. A másik gyakran alkalmazott kiválasztási módszernél a rendszer az éppen aktuális set-előfordulásba helyezi el az új rekordot (CURRENT OF SET), azaz abba, amelynek egy tagjára vagy tulajdonosára azon set-típuson belül utoljára hivatkoztunk. A 4f utasításban a DUPLICATES ARE NOT ALLOWED kiegészítés ugyanúgy, mint a rekordleírásban azt jelenti, hogy ugyanazon set-előforduláson belül nem lehet két azonos rendezőmező értékű tag. (Ez nem jelenti azt, hogy két KAPCSOLO rekordban nem lehet ugyanaz a METTOL mező értéke, csupán azt, hogy egy set-előforduláson belül nem lehet azonos. Ez azt az ésszerű feltételezést jelenti, hogy ugyanaz a gépkocsi egy nap nem cserélhet kétszer gazdát!)

A 4h-4n deklarációk a JARMU-NYILVANTARTAS szinguláris set-et (OWNER IS SYSTEM) írják le hasonló módon. Ennek tagjai az egyértelmű RENDSZAM alapján sorba rendezett GEPJARMU rekordok. Ez a set teszi lehetővé a GEPJARMU rekordok soros feldolgozását.

Ugyanígy kell definiálnunk a TULAJ set-et is.

A DBTG által javasolt sémában lehetőség van arra, hogy az adatok fizikai és programbeli ábrázolási módja eltérjen. Erre szolgál a FOR ENCODING CALL eljárás-1, illetve a FOR DECODING CALL eljárás-2 deklaráció. Így például, ha a jobb tárkihasználás érdekében az adatokat tömörítjük (l. 4.9.1), vagy biztonsági okokból rejtjelezve tároljuk (l. 4.9.2), és az eredeti adatokból a KODOLO nevű eljárással állítjuk elő a fizikailag ténylegesen tárolt adatokat, és a DEKODOLO nevű eljárással kapjuk vissza a tároltakból az eredeti formában, akkor a következő

```
02 GYARTMANY; PICTURE IS X (12);
FOR ENCODING ALWAYS CALL KODOLO;
FOR DECODING ALWAYS CALL DEKODOLO;
```

mezőleírás hatására a felhasználó a GYARTMANY mezőt mindig 12 karakteres alfanumerikus mezőnek látja, de tárolt értéke az adatbázisban mindig ennek a jelzorozatnak a KODOLO eljárással előállított leképzése, pl. egy 0-255 közé eső 1 byte hosszúságú bináris szám lesz.

A redundanciát csökkenthetjük a rekordleírásban a VIRTUAL kiegészítéssel is. Az így definiált adato(ka)t a rendszer fizikailag nem tárolja. Értékét vagy a SOURCE-szal megadott adatelemből veszi közvetlenül, vagy pedig a RESULT OF eljárás klauzulában megadott eljárás eredményeként állítja elő. Így például, ha az ATLAGOLAS eljárás az A1, A2, ..., AN változók átlagát állítja elő, akkor a

```
03 ATLAG; IS VIRTUAL RESULT OF ATLAGOLAS...
```


módon deklarált változó az adatbázisban tényleges helyet nem foglal el, de az arra jogosult programok hivatkozhatnak rá, és felhasználhatják értékét, mely az A1, A2, ..., AN változók aktuális értékének átlaga lesz.

A sémaleíró nyelv (DDL=Data Definition Language) teljes szintaxisát, valamint az egyes utasítások részletes magyarázatát a DBTG jelentés, illetve a javaslatuk alapján készített rendszerek kézikönyvei tartalmazzák. A konkrét alkalmazások többsége a DBTG által megengedett lehetőségeknek általában csak egy részét engedi meg.

6.6 Alsémaleírás

Mivel az egyes felhasználók a sémában leírt adatbázisnak csak bizonyos részeit használhatják, az adatbázis-felügyelőnek módjában áll az alsémaleírásban rögzíteni a teljes adatbázisnak azt a részét, melyhez az azzal az alsémával dolgozó program hozzáférhet. Ezenkívül módja van az elnevezések, az ábrázolási mód, a kiválasztási eljárások, biztonsági zárok megváltoztatására is. Egy sémán akárhány különböző alséma deklarálható, akárhány program használhatja ugyanazt az alsémát és különböző alsémák átlapolhatják egymást.

A 6.5 ábrán példaképpen bemutatjuk azt az alsémaleírást, mely csak a gépjármű rendszáma és tulajdonviszony kezdete és vége közötti kapcsolatot engedi meg a 6.4 ábrán definiált sémán.

```

1  SUB-SCHEMA IDENTIFICATION DIVISION.
2  SUB-SCHEMA NAME IS KOCSI OF SCHEMA NAME
   GEPJARMU-NYILVANTARTAS
   2a  PRIVACY KEY FOR COPY IS 'LEMASOLHAT'.
3  SUB-SCHEMA DATA DIVISION.
4  RENAMING SECTION.
4a  RECORD NAME KAPCSOLAT IN SCHEMA IS CHANGED TO TULAJDONJOG.
5  AREA SECTION.
5a  COPY ALL AREAS.
6  RECORD SECTION.
6a  01 GEPJARMU.
6b      02 RENDSZAM; PICTURE IS X(8).
6c  01 TULAJDONJOG.
6d      02 METTOL.
6e      03 EV; PICTURE IS 9(4).
6f      03 HO; PICTURE IS 99.
6g      03 NAP; PICTURE IS 99.
6h      02 MEDDIG.
   :
7  SET SECTION
7a  COPY JARMU, JARMU-NYILVANTARTAS.
```

6.5 ábra Alsémaleírás a DBTG által javasolt alsémaleíró nyelvben

Az 1-es utasítás közli, hogy alsémaleírás következik. A 2-es utasítás megadja az alséma nevét (SUB-SCHEMA NAME IS), mely példánkban KOCSI, és hogy ez a GEPJARMU-NYILVANTARTAS sémának része (OF SCHEMA NAME). A 2a uta-

sítás megadja a GEPJARMU-NYILVANTARTAS séma lemásolásához szükséges jelszót (PRIVACY KEY FOR COPY IS), mely azért szükséges, mert a sémaleírásban a másolhatóságot a LEMÁSOLHAT jelszóhoz kötötte az adatbázis-felügyelő (lásd a 6.4 ábrán 1b utasítást). A 4 és 4a utasítások lehetővé teszik, hogy a sémában KAPCSOLAT néven definiált rekordra az alsémában TULAJDONJOG névvel lehessen hivatkozni. Az 5 és 5a utasítással bemásoltatjuk (COPY) a sématerület leírásait az alsémába. 6–6h utasítások megadják az alsémában definiált rekordok leírását, míg a 7 és 7a utasítás definiálja a JARMU és a JARMU-NYILVANTARTAS seteket, és pedig a sémabeli definícióval megegyező módon (COPY). (Az adatbázis-felügyelő a set-leírásban megváltoztathatja a set kiválasztási módját és védelmét az alsémában, de nem módosíthatja egyéb tulajdonságait.)

6.7 Adatkezelő nyelv

A DBTG által javasolt adatkezelő nyelv utasításait és funkcióit a 6.6 ábrán, ennek konkrét megvalósítását az IDMS rendszerben a 6.7 ábrán foglaltuk össze. Az egyes utasítások részletes ismertetése meghaladná könyvünk kereteit, ezért csak néhány utasítás leglényegesebb vonását mutatjuk be a már korábban is felhozott példán.

Utasítás	Funkció
CLOSE	Felszabadít korábban lefoglalt set-típusokat és -területeket
DELETE	Kitörli az adatbázisból a megadott rekordot
FIND	A rekordkiválasztási feltétel alapján kiválasztja a megfelelő rekordot
FREE	Hatástalanítja a megfelelő KEEP utasítást
GET	Átviszli az aktuális adatokat a felhasználó munkaterületére
IF	Lehetővé teszi az adatbázissal kapcsolatos feltételek kiértékelését
INSERT	A rekord tagja lesz a megadott set(ek)-nek
KEEP	Informálja a rendszert, hogy a program még tovább fel kívánja használni az aktuális rekordot
MODIFY	Módosítja a megadott rekordot
MOVE	Átviszli az aktuális jelzőket
OPEN	Megnyitja a megadott területet a program számára, ellenőrzi a hozzáférési jogot
ORDER	Átrendezi az utasításban megadott set tagjait
REMOVE	Megszünteti az utasításban megadott rekord és set(ek) közti kapcsolatot
STORE	Tárolja a megadott rekordot fizikailag, és beilleszti logikailag a megfelelő setekbe
USE	Hibakezelő eljárást definiál

6.6 ábra A DBTG által javasolt adatkezelő nyelv utasításai

Feladatunk hasonló, mint ami a hierarchikus modellben volt: „Keressük ki további feldolgozás céljaira az összes zöld színű Volkswagen tulajdonosának nevét és lakcímét.” A COBOL befogadó nyelven írt programnak csak azokat a részeit tüntetjük fel a 6.8 ábrán, melyek az adatbázis-kezelő rendszerrel való együttműködés következtében megváltoztak. A program a 6.4 ábrán definiált GEPJARMU-NYILVANTARTAS séma TELJES-ADATBAZIS elnevezésű alsémájával dolgozik, melyet az adatbázis-felügyelő a sémával teljesen azonosnak definiált. Mivel a sémaleírásban csak az UPDATE (megváltoztatás) funkciót biztosítottuk egy ellenőr-

Utasítás	Funkció
BIND	Hozzákapcsolja a megfelelő rekordtípusokat az adatbázison működő programhoz
CONNECT	Beköti a megadott rekordot a megfelelő set-be
DISCONNECT	Megszünteti az utasításban megadott rekord és set közti kapcsolatot
ERASE	Kitörli az adatbázisból a megadott rekordot
FIND	A rekordkiválasztási feltétel alapján kiválasztja a megfelelő rekordot
FINISH	Lezárja az adatbázis-kezelő műveleteket, felszabadítja a korábban lefoglalt rekord- és set-típusokat, területeket
GET	Átviszi az aktuális rekordot a felhasználó munkaterületére
IF	Lehetővé teszi az adatbázissal kapcsolatos feltételek kiértékelését (pl. tagja-e egy adott rekord egy set-típusnak vagy egy konkrét set-előfordulásnak)
MODIFY	Módosítja a megfelelő rekordot az adatbázisban
OBTAIN	A rekordkiválasztási feltétel alapján kiválasztja a megfelelő rekordot, és átviszi a felhasználó munkaterületére
READY	Megnyitja a megadott területeket a felhasználó számára a kívánt feldolgozási módra (pl. kizárólagos használatra, csak olvasásra, módosításra is)
STORE	Tárolja a megadott rekordot fizikailag, és beilleszti a megfelelő set-ekbe

6.7 ábra Az IDMS rendszer adatkezelő utasításai

ző eljárással (6.4 ábrán 2b), csupán olvasni kívánó programunk minden további nélkül hozzáférhet a teljes adatbázishoz.⁶

A COBOL program adatleíró részében (DATA DIVISION) meg kell adnunk, melyik séma alsémájával fog a program dolgozni (1a és 1b). Ezenkívül a WORKING-STORAGE SECTION-ban definiáljuk az ERROR-STATUS változót (2-2a), melynek tartalma zérus, ha az adatbázis-kezelő rendszer a megadott műveletet hibátlanul végrehajtotta. A különböző hibák különféle kódokat adnak, így pl. a FIND (keresd) utasításnál, ha a set végére értünk, akkor az ERROR-STATUS értéke 0307 lesz.

Az eljárás részben (PROCEDURE DIVISION) a DECLARATIVES-eknél megadjuk, mi a teendő, ha az adatkezelő művelet nem zajlott le hibamentesen (4-4h). Erre szolgál a USE utasítás. Külön kezeljük a set vége jelzést, amelyet nem tekintünk hibának, csupán egy feldolgozási lépéssor végének.

A set-ek megnyitása után (5) a FIND utasítással megkeressük a JARMU-NYILVANTARTAS set-ből az első GEPIJARMU rekordot (6).⁷ Figyeljük meg, hogy a rekord megtalálása még nem jelenti annak beolvasását is, így a program csak a GET utasítás után kapja meg a rekordot. Innentől kezdve a program egy ciklust

⁶ Amennyiben nem így volna, úgy az IDENTIFICATION DIVISION-ban meg kellene adni a megfelelő PRIVACY KEY értékeket, illetve eljárásokat.

⁷ A FIND utasítás legfontosabb formái a következők: A FIND FIRST rekordnév RECORD OF set-név SET utasítás a megadott aktuális set első, FIND LAST ... az utolsó, a FIND NEXT ... a soron következő tag rekordját, a FIND OWNER a tulajdonosát keresi meg. A FIND rekordnév USING azonosító közvetlen megtalálást tesz lehetővé az azonosítóban lévő adatbáziskulcs alapján, míg a FIND NEXT DUPLICATE WITHIN rekordnév RECORD a direkt hozzáférésre szolgáló rekordkulcs értéke alapján választja ki a következő, ugyanazon azonosító értékkel rendelkező rekordot (természetesen csak akkor, ha ilyenek létezését megengedtük a sémaleírásban). A FIND CURRENT OF rekordtípus az adott rekordtípus kurrens (utoljára használatba vett) rekordját adja meg, míg a FIND CALC rekordtípus az adott kulcsértékkel rendelkező, közvetlen hozzáféréssel elérhető rekordot.


```

DATA DIVISION.
1  SCHEMA SECTION.
1a  INVOKE SUB-SCHEMA TELJES-ADATBAZIS OF SCHEMA
    GEPJARMU-NYILVANTARTAS.
    :
2  WORKING-STORAGE SECTION
    :
2a  77 ERROR-STATUS PICTURE 9999.
    :
3  PROCEDURE DIVISION.
4  DECLARATIVES.
4a  VART-HIBA SECTION.
4b      USE IF ERROR-STATUS = '0307'
4c  SET-VEGE-KEZELES.
4d      EXIT.
4e  TENYLEGES-HIBA SECTION.
4f      USE IF ERROR-STATUS = '0307'
4g  HIBAKEZELES.
    hibakezelő utasítások
4h  END DECLARATIVES.
    :
5  OPEN ALL.
    :
6  FIND FIRST GEPJARMU RECORD OF JARMU-NYILVANTARTAS SET.
7  SET-VEGE-VIZSGALAT-1.
7a  IF ERROR-STATUS = '0307' PERFORM VEGE
    ELSE PERFORM JARMUKIVALASZTAS.
7b  GO TO SET-VEGE-VIZSGALAT-1.
    :
8  JARMUKIVALASZTAS.
8a  GET GEPJARMU.
8b  IF GYARTMANY = 'VOLKSWAGEN' AND SZIN = 'ZOLD'
    PERFORM TULAJDONOS - KIVALASZTAS THRU NINCS-TOBB.
8c  FIND NEXT GEPJARMU RECORD OF JARMU-NYILVANTARTAS SET.
    :
9  TULAJDONOS-KIVALASZTAS.
9a  FIND FIRST KAPCSOLO RECORD OF JARMU.
10 SET-VEGE-VIZSGALAT-2.
10a  IF ERROR-STATUS = '0307' GO TO NINCS-TOBB.
10b  FIND OWNER RECORD OF TULAJ SET.
10c  GET TULAJDONOS,
    megfelelő adatok feldolgozása és kiírása
10d  FIND NEXT KAPCSOLO RECORD OF JARMU.
10e  GO TO SET-VEGE-VIZSGALAT-2.
10f  NINCS-TOBB. EXIT.
    :
11 VEGE.
11a  CLOSE ALL.

```

6.8 ábra Adatvisszakeresés hálós adatbázis-kezelő rendszer segítségével

hajt végre (7–7b), melyből a 7a alapján akkor lép ki, ha már nincsen több gépiármű rekord.

Ha a FIND utasítással találtunk GEPIJARMU rekordot, akkor végrehajtjuk a JARMUKIVALASZTAS eljárást (8–8c). Beolvassuk az aktuális, azaz a FIND utasítással előzőleg meghatározott GEPIJARMU rekordot (8a). Ha ennek adatai megfelelnek a feltételnek, akkor végrehajtjuk a 8b-ben definiált eljárásorozatot (TULAJDONOS-KIVÁLASZTÁS-tól NINCS TOBB-ig), ha nem, akkor megkeressük a JARMU-NYILVANTARTAS set soronkövetkező rekordját (8c), és folytatjuk a 7–7b ciklust, azaz a GEPIJARMU rekordok szekvenciális beolvasását. Figyeljük meg, hogy az alsémában definiált adatbázisbeli rekordok egyes mezőire a COBOL programból minden további deklaráció nélkül névvel hivatkozhatunk.

Amennyiben a beolvasott GEPIJARMU rekord megfelelt a kiválasztási feltételeknek, akkor megkeressük az aktuális GEPIJARMU rekord első KAPCSOLO rekordját (9a), a hozzá tartozó TULAJDONOS rekordot (10b), melyet beolvassunk (10c) és feldolgozunk. Ezután ugyanezt végrehajtjuk a következő KAPCSOLO rekord tulajdonosával (10d,e, ill. 10b,c) egészen addig, amíg ennek a kocsinak az összes tulajdonosát fel nem dolgoztuk (10a). Mikor ez is megtörtént, akkor megkeressük a következő GEPIJARMU rekordot (8c), és ezzel folytatjuk a feldolgozást. Figyeljük meg, hogy a 10d utasításnál a KAPCSOLO rekordokat a JARMU set alapján járjuk végig.

Természetesen, egy ilyen viszonylag egyszerű példán nem tudtuk bemutatni a DBTG adatkezelő nyelvének teljes hatékonyságát, azonban ha a felhasználó megkísérelné saját maga beprogramozni valamelyik programozási nyelven az itt tárgyalt feladatot, hamar rájönne az adatkezelő nyelv előnyeire.

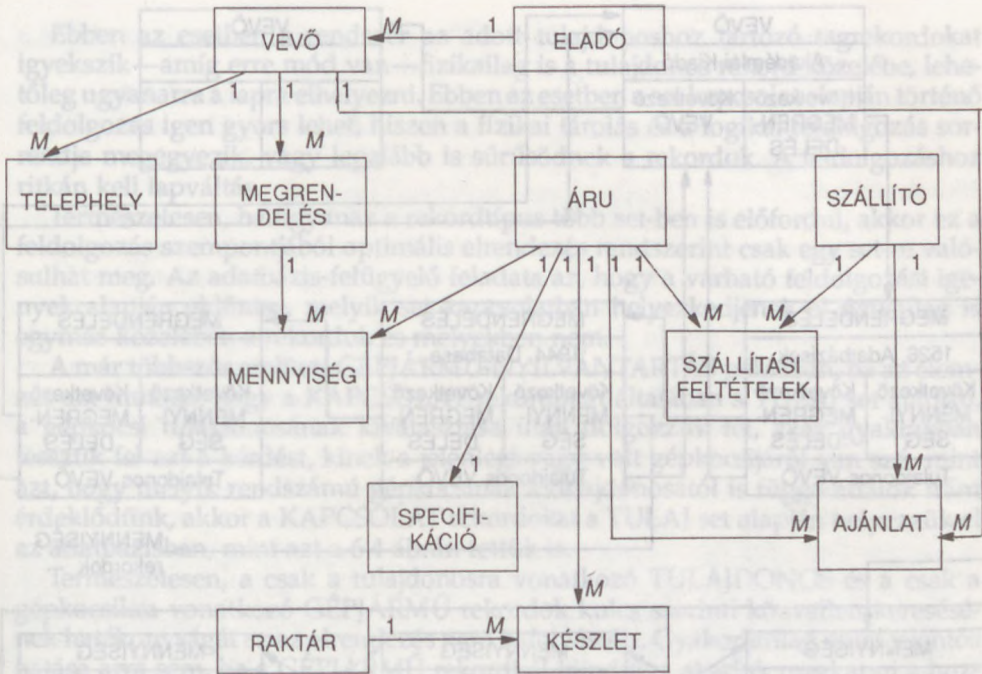
A hálós modell jellemzőinek további illusztrálására a 6.9 ábrán bemutatjuk, hogyan lehet megvalósítani a 6.1 ábrán bemutatott adatszerkezetet.

A MEGRENDELÉS és az ÁRU rekordok közti $M:N$ kapcsolatot a MENNYISÉG kapcsolórekord közbeiktatásával oldjuk fel. Ez tartalmazza, hogy egy adott megrendelésben egy adott áruból mekkora mennyiséget rendeltünk meg. (Ezenkívül tartalmazza pl. az egységárát is, ha az megrendelésenként változhat.) Az, hogy ez a mennyiség melyik árukódra és milyen számú megrendelésre vonatkozik, szerepelhet közvetlenül a MENNYISÉG rekordban is, de lehet, hogy külön művelettel kaphatjuk csak meg a set kapcsolatból a tulajdonosra mutató pointer segítségével.

Az ÁRU és a SZÁLLÍTÓ rekordok között kétféle kapcsolórekord típust hozunk létre. Az egyik, a SZÁLLÍTÁSI-FELTÉTELEK tartalmazza, hogy milyen feltételek mellett lehet az adott árut a szállítótól megrendelni, és azt általában milyen módon szállítja (pl. szállítási határidő, ár, mennyiségi kedvezmények stb.), míg az AJÁNLAT egy konkrét szállításra vonatkozik és függ attól is, hogy melyik VEVŐ-nek teszik.

Az ÁRU és a RAKTÁR rekordokat a KÉSZLET rekordtípus kapcsolja össze. A modell a 6.9 ábrán fentüntetett $1:M$ és $1:1$ set kapcsolatokon kívül öt szinguláris set-et is tartalmaz, melyeknek tagjai rendre a VEVŐ, ÁRU, SZÁLLÍTÓ, RAKTÁR és az ELADÓ.

A 6.10 ábrán feltüntettük, hogyan valósítja meg a rendszer a VEVŐ, MEGRENDELÉS és MENNYISÉG rekordok között a kapcsolatokat pointerok segítségével. Ennél a realizálásnál a MENNYISÉG rekord tartalmazza az áru kódját, így erre nem mutat pointer.



6.9 ábra A 6.1 ábrán megadott logikai modell realizálása kapcsoló rekordok közbeiktatásával

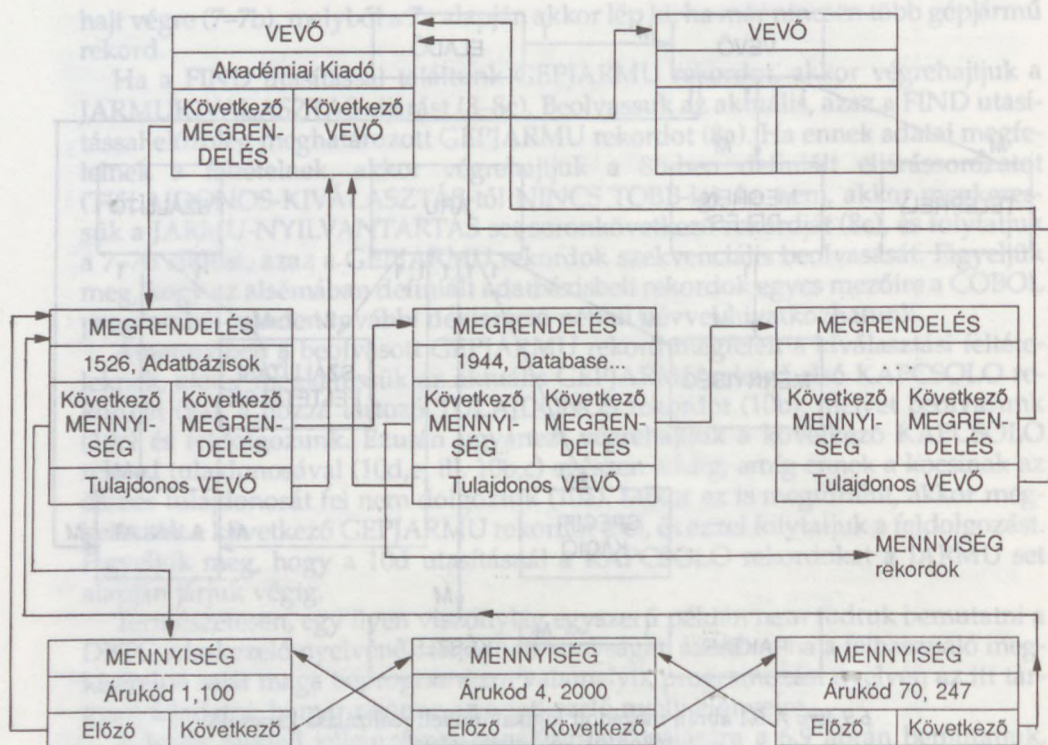
A VEVŐ-MEGRENDELÉS kapcsolatnál a set következő tagjára, valamint a tulajdonos rekordra mutat pointer. Ez utóbbi meggyorsítja a megrendelés száma alapján történő feldolgozást. Az utolsó tagról a set-nek a „következő” pointerre a tulajdonosra, míg a tulajdonosból az első tagra mutat.

6.8 Hatékonysági kérdések

Ahhoz, hogy a felhasználó hatékonyan tudjon dolgozni az adatbázissal, szükséges, hogy

- az adatbázis-felügyelő az adatokat a várható felhasználásoknak megfelelő módon helyezze el fizikailag az adatbázisban,
- a felhasználás logikai sorrendjének megfelelő kapcsolatok definiálva legyenek az adatbázisban,
- a felhasználó ismerje azokat a kapcsolatokat, lehetőségeket, amelyeknek alapján mozoghat, navigálhat az adatbázisban.

Az elhelyezési módot az adatleíró nyelv LOCATION MODE klauzulájában határozzuk meg külön-külön minden egyes rekordtípusra. Ha az adott rekordtípust rendszerint közvetlenül, a kulcs alapján akarjuk elérni, akkor azt a



6.10 ábra A 6.8 ábra egy részének ábrázolása a pointerekkel megvalósított rekordközi kapcsolatok bemutatására

LOCATION MODE IS CALC elérési kulcs

módon definiáljuk.

Ezeket a rekordokat a rendszer a megadott területen belül hashing algoritmus segítségével (l. 4.8) helyezi el és így bármelyiküket a kulcs értéke alapján igen gyorsan, általában egy lépéssel megkaphatjuk.

Ugyanakkor viszont, ha ezeket a rekordokat egy előre meghatározott sorrendben kívánnánk feldolgozni, azt csak akkor tehetnénk meg, ha köztük egy set kapcsolatot definiálnánk ezen feltételnek megfelelően. De a soros feldolgozás hatékonysága ekkor nem optimális, mivel az adatok fizikai elhelyezését és logikai feldolgozásának a sorrendje nem egyezik meg. A logikailag egymást követő rekordokat a set kapcsolat által létrehozott pointerlánc segítségével a legkülönbözőbb fizikai lapokról hozza be a rendszer, ami igen sok lapváltással, azaz viszonylag lassú feldolgozással jár.

Amennyiben tudjuk, hogy a feldolgozás általában a set kapcsolat alapján történik, tehát ha megtaláltuk a tulajdonost, akkor utána szükségünk lesz a set tag rekordjaira is, akkor célszerű, hogy a tag rekordtípus egyedeit fizikailag a set kapcsolat szerint helyezzük el. Ezt biztosítja a

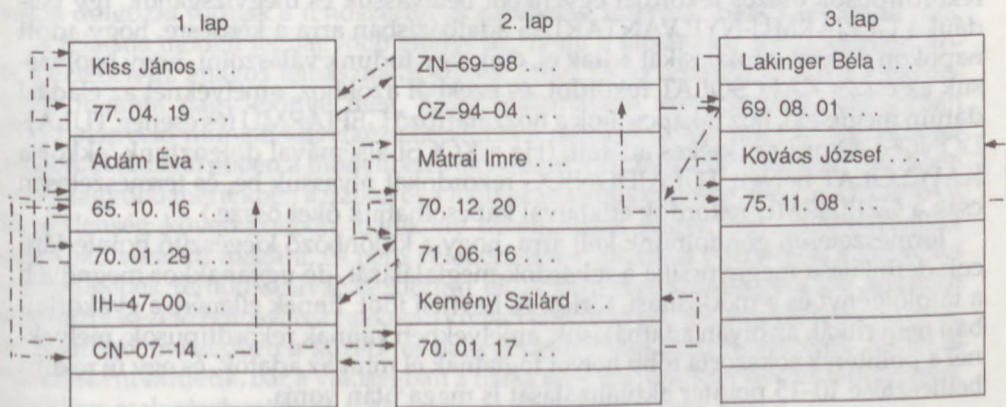
LOCATION MODE IS VIA set kapcsolat.

Ebben az esetben a rendszer az adott tulajdonoshoz tartozó tagrekordokat igyekszik—amíg erre mód van—fizikailag is a tulajdonos rekord közelébe, lehetőleg ugyanarra a lapra elhelyezni. Ebben az esetben a set kapcsolat alapján történő feldolgozás igen gyors lehet, hiszen a fizikai tárolás és a logikai feldolgozás sorrendje megegyezik, vagy legalább is sűrűsödnek a rekordok. A feldolgozáshoz ritkán kell lapváltás.

Természetesen, ha ugyanaz a rekordtípus több set-ben is előfordul, akkor ez a feldolgozás szempontjából optimális elrendezés rendszerint csak egy set-re valószínűleg meg. Az adatbázis-felügyelő feladata az, hogy a várható feldolgozási igények alapján eldöntse, melyik set kapcsolatban helyezkedjenek el fizikailag is egymás közelében a rekordok és melyekben nem.

A már többször említett GÉPJÁRMŰ-NYILVÁNTARTÁS példában, ha az elemzés azt mutatja, hogy a KAPCSOLAT rekordokat általában a TULAJ set alapján, a gépkocsi tulajdonosának kiválasztása után dolgozzuk fel, azaz gyakrabban tesszük fel azt a kérdést, kinek a jelenlegi vagy volt gépkocsijáról van szó, mint azt, hogy melyik rendszámú gépkocsinak a tulajdonosától is függő adatok iránt érdeklődünk, akkor a KAPCSOLAT rekordokat a TULAJ set alapján helyezzük el az adatbázisban, mint azt a 6.4 ábrán tettük is.

Természetesen, a csak a tulajdonosra vonatkozó TULAJDONOS és a csak a gépkocsikra vonatkozó GÉPJÁRMŰ rekordok kulcs szerinti közvetlen keresésének hatékonyságát ez az elrendezés nem befolyásolja. Gyakorlatilag nincs jelentős hatása arra sem, ha a GÉPJÁRMŰ rekordból kiindulva akarjuk megkapni a hozzátartozó TULAJDONOS-ok adatait, vagy a TULAJDONOS-ból a GÉPJÁRMŰ-veket. A különbség csak akkor jelentkezik, amikor a KAPCSOLAT rekordokat akarjuk elérni valamelyik set alapján. Formailag azonban itt sincs különbség, a



6.11 ábra Rekordok fizikai elhelyezése set kapcsolat alapján. A TULAJ kapcsolat pointerét szaggatott (- - -), a JÁRMŰ kapcsolatét pontozott szaggatott (- . - . -) vonal jelzi. A fizikai tárolás a TULAJ set alapján történt, így az egy set előfordulásba tartozó rekordok azonos lapon vannak. Láthatjuk, hogy ugyanazon a lapon a CALC elhelyezéstől függően lehetnek ugyanazon és különböző rekordtípusok előfordulásai is, a set kapcsolatokban definiált elhelyezéstől függetlenül.

logikai út mind a TULAJ, mind a JÁRMŰ set-nél ugyanolyan, csak az előbbi a fizikai elrendezés következtében gyorsabban járható be.

A példához kapcsolódó fizikai tárolási módot a 6.11 ábrán mutatjuk be.

Jelentősen befolyásolják a feldolgozás hatékonyságát a set-ekben definiált pointerláncok is. Egy előre mutató pointerláncot a rendszer minden set-ben automatikusan felépít.

Amennyiben azonban a set egyes tagjaiból akarunk következtetni a tulajdonosra, akkor célszerű a set-ben még egy pointert definiálni, mely minden egyes tagból rámutat a tulajdonosra. Különösen hatékony ez, ha az egyes set előfordulásokban sok tag van, mert ilyenkor a tulajdonos megtalálásához nem kell végigmenni a pointerlánc hátralévő részén (átlagosan az adott előfordulás tagjainak a felén). Ha a tulajdonos-rekordból kiindulva a tagokat rendszerint valamilyen szempont szerint sorbarendezve kívánjuk feldolgozni, akkor célszerű a set pointerláncát erre a mezőre rendezve kialakítani (ORDER IS SORTED). Ekkor nem kell a set-előfordulás rekordjait először beolvasni, ideiglenesen tárolni, majd rendezni. A FIND NEXT utasítás automatikusan a megfelelő sorrendben találja meg a rekordokat.

Gyakran a set egyes tagjait tárolásuk időbeli sorrendjében helyezzük el, és így is akarjuk őket feldolgozni. Ha az egyes előfordulások sok tagból állnak, akkor célszerű lehet a set-ben egy visszafelé mutató pointert is definiálni. Ha ez nincs, akkor az új tag helyét a rendszer csak úgy tudja megállapítani, hogy előbb végigmegy a teljes láncon (vagy ha nem a tulajdonosnál lépünk be, akkor a még hátralévő részén). Ez több száz elemnél elég hosszadalmas lehet. Visszafelé is mutató pointerláncnál a tulajdonos rekordból kiindulva egy lépésben megtalálja az utolsó rekordot, amely mögé most már azonnal be tudja illeszteni az újat.

Sajnos, a hálós rendszerekben dinamikusan nem hozhatunk létre set-kapcsolatokat. Így, ha egy adott logika szerint a rekordok között nem volt set kapcsolat definiálva, akkor eszerint csak úgy végezhetünk feldolgozást, hogy a megfelelő rekordtípusok összes rekordját egyenként beolvassuk és megvizsgáljuk. Így például a GÉPJÁRMŰ-NYILVÁNTARTÁS adatbázisban arra a kérdésre, hogy adott napokon milyen gépkocsikat adtak el, csak úgy tudunk válaszolni, hogy beolvassuk az összes KAPCSOLAT rekordot, és ezekből azokhoz, amelyeknél az eladási dátum megfelelő, hozzákapcsoljuk a hozzátartozó GÉPJÁRMŰ (és esetleg TULAJDONOS) rekord szükséges adatait. (Ha a KOCSI alsémával dolgoztunk, akkor a KAPCSOLAT helyett TULAJDONJOG rekordokat olvassuk be, és természetesen csak a GÉPJÁRMŰ rekordok adataival kapcsolhatjuk őket össze.)

Természetesen gondolnunk kell arra, hogy a különböző kiegészítő pointerláncok definiálása megyorsítja a rekordok megtalálását, de ugyanakkor megnöveli a tárolóigényt és a módosítási, törlési és beviteli időt. Ennek ellenére a gyakorlatban nem ritkák az olyan adatbázisok, amelyekben vannak rekordtípusok, melyeknél a pointerok sokszorta több helyet foglalnak el mint az adatok, és egy új rekord beillesztése 10–15 pointer aktualizálását is maga után vonja.

A relációs modell

7.1 Általános ismertetés

A relációs modellben az adatokat logikailag relációkban, szemléletesen tekintve táblázatokban ábrázoljuk, és a köztük levő különböző kapcsolatokat relációs műveletek segítségével hozzuk létre. A táblázatok az adatok szerkezetének meghatározását, az adatok leírását teszik lehetővé a modell keretében, míg az adatkezelést a relációs műveletekkel valósítjuk meg. A fenti két alapvető feladaton kívül a relációs modell lehetőséget nyújt az adatok integritásának biztosítására, valamint a konkrét feladatok ismeretében a modell alkalmazásának hatékonyságát növelő szerkezetek kialakítására.

Ezen kívül, mint minden más adatbázis-kezelő rendszer tartalmazza magának az adatbázisnak a leírását is, melynek alapján az adatbázis-felügyelő és a felhasználók dolgozni tudnak a rendszerrel és naprakész információkat kaphatnak róla.

A relációs modell legnagyobb előnye az, hogy a táblázatos ábrázolásmód következtében az átlagos felhasználói szemlélethez ez az adatmodell áll a legközelebb. További nagy előnye még, hogy matematikai szempontból—elsősorban Codd alapvető munkái következtében—ezt a modellt dolgozták ki a legrészletesebben. Ráadásul ebben a modellben valósítható meg legjobban, legegyszerűbben és legnagyobb mértékben a fizikai és a logikai adatfüggetlenség. Éppen a nagyfokú függetlenség következtében, hatékony alkalmazása nagyméretű adatbázisokra elég soká váratott magára, és nem véletlen, hogy rohamos elterjedése a személyi számítógépek térhódításával kezdődött el. A személyi számítógépekre adaptált adatbázis-kezelő rendszerek csaknem kivétel nélkül a relációs modellen alapulnak, és az utóbbi évtizedben a közép- és nagygépekre elkészített rendszerek zömét is relációsként hirdetik, bár a valóságban a tiszta relációs rendszer összes feltételének általában csak részben, vagy csak megszorításokkal tesznek eleget.

7.2 A relációk tulajdonságai

Matematikailag tekintve a relációs modell egy vagy több kétdimenziós, speciális tulajdonságokkal rendelkező táblázatból, más néven relációból áll. Ezek a relációk írják le a valós világ különböző egyedeit, azok tulajdonságait.

A relációnak *oszlopai* és *sorai* vannak. Az oszlopok nevei alkotják a reláció (táblázat) fejlécét, a sorok bejegyzései a reláció adatait. A sorok jellemzik az adott reláció *egyedeit*, *entitásait*, az oszlopok pedig az egyes *tulajdonságokat*, az *attribútumokat*. A klasszikus file-szervezési terminológiában a relációt egy állománynak tekintjük. A bejegyzések, a sorok felelnek meg az állomány egyes rekordjainak, az oszlopok a rekordok elemi mezői, a táblázat egyes adatelemei pedig a rekordok megfelelő mezőinek értékei.

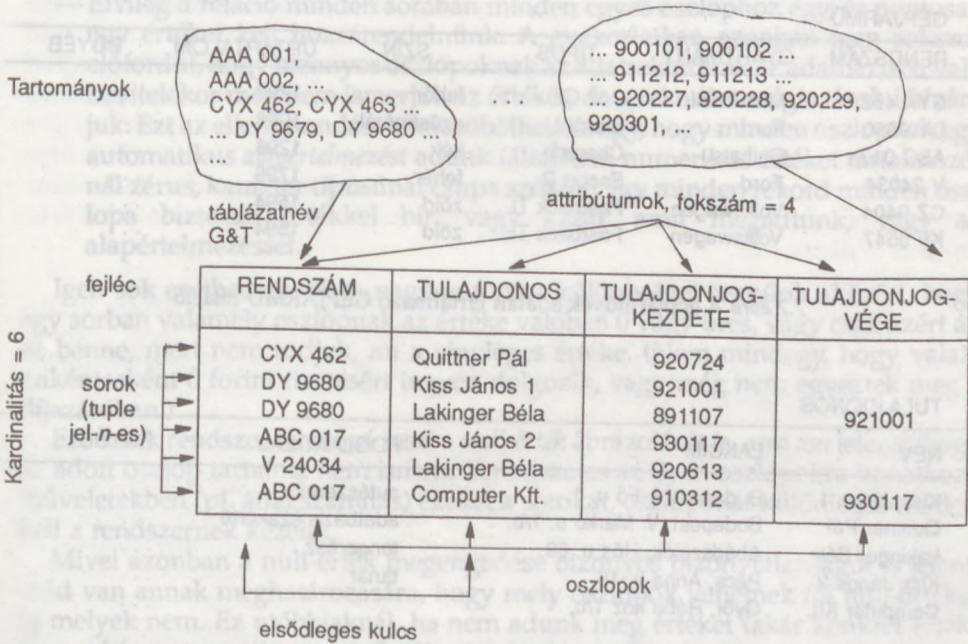
Ahhoz, hogy ezek a táblázatok valóban relációs adatbázist alkossanak, az egyes táblázatokra a következő feltételeknek kell teljesülnie:

- Minden relációnak van egy egyértelmű neve, amelyik azt azonosítja az adatmodellben.
- Adott reláció minden sorában azonos számú oszlop van. Az oszlopok számát (n) szokták a reláció fokának, fokszámának is nevezni, a sorokat pedig n -esnek, vagy az angol kifejezést használva tuple-nak, illetve n -tuplenak is. A reláció foka a modellben állandó. Ha megváltoztatjuk, akkor az valójában az eredeti reláció megszüntetését, és egy új létrehozását jelenti.¹
- Minden oszlopnak az adott reláción belül egyértelmű neve van, amellyel azonosítható.
- Minden oszlop csak meghatározott értékeket (ez lehet folytonos tartományból is) vehet fel. Ezek összességét nevezzük az adott oszlophoz tartozó attribútum értékek tartományának (domain). A tartomány a modell szempontjából azonos típusú elemi adatokat tartalmaz csak. A tartomány mindig alkalmazásfüggő.
- Bármely sorban bármelyik oszlop csak egyetlen egy értéket vehet fel a megengedett értékek közül. Ezt úgy mondjuk, hogy az oszlopértékek elemiek, nem tartalmazhatnak ismétlődő mezőt.

A 7.1 ábrán egy egyszerű gépjárműtulajdonos-nyilvántartó relációt (G&T) láthatunk. Ez csak a tulajdonjog megállapításához minimálisan szükséges adatokat tartalmazza. Feltételezzük—és ebben a fejezetben végig élni fogunk ezzel a feltételezéssel—, hogy a személyeket a nevük egyértelműen azonosítja. (Ha pl. két Kiss János lenne, akkor azokat Kiss János 1 és Kiss János 2-nek neveznénk, így biztosítva a név egyediségét.). Ez a reláció csak akkor ábrázolja a valós világ megfelelő részét, ha a gépjárművek tulajdonjoga nem lehet osztott, egy gépkocsinak egy adott időben egy, és pontosan egy tulajdonosa van. (Ha nincs még eladva, akkor pl. a forgalmazó kereskedő.)

A gépjármű nyilvántartás teljes relációs modelljének kialakításához szükségünk van még további két egyedtípus, a gépjárművek és a tulajdonosok relációként való

¹ Egyes relációs rendszerek a gyakorlatban megengedik a táblázat fokának növelését, új oszlopok hozzáadását (l. ALTER utasítás, 10.6 pont), de ez akkor, mint mondtuk, a modellt tekintve már egy új relációt jelent.



7.1 ábra Reláció ábrázolása

A reláció foksáma 4, a tartományok száma 3.

(A TULAJDONJOG-KEZDETE és a TULAJDONJOG-VÉGE attribútumok ugyanazon tartományra, az érvényes dátumok halmazára épülnek.)

ábrázolására is. Ezt legcélszerűbben két önálló relációval (táblázattal) valósíthatjuk meg.

A GÉPJÁRMŰ táblázat oszlopai (7.2 ábra) a RENDSZÁM, GYÁRTMÁNY, TÍPUS, SZÍN, ÜRTARTALOM és a gépjármű további, csak a gépkocsitól magától függő EGYÉB tulajdonságai (ezeket nem részletezzük most).²

A második, a TULAJDONOS táblázat (7.3 ábra) az egyes tulajdonosoknak a gépkocsiktól független adatait tartalmazza. Oszlopai a tulajdonos (egyedi) neve, lakcíme, foglalkozása és EGYÉB szükséges, szintén már nem részletezett adatai.

A GÉPJÁRMŰ és a G&T táblázat illetve a TULAJDONOS és a G&T táblázat között a RENDSZÁM, illetve a NÉV alapján egy-több, a GÉPJÁRMŰ és TULAJDONOS között pedig több-több kapcsolat áll fenn. (Ez azt a lehetőséget is magában foglalja, hogy pl. lehet olyan gépjármű a GÉPJÁRMŰ táblázatban, amelyiknek rendszáma már van, de tulajdonosa még nincsen. Így a G&T táblázatban ez a jármű még nem szerepel.)

² Mind a GÉPJÁRMŰ, mind az alább ismertetendő TULAJDONOS táblázatban az EGYÉB oszlopba összefoglalt adatokat vagy egyetlen összefüggő, szöveges információt nyújtó magyarázó mezőnek kell tekintenünk, vagy több önálló, de meghatározott számú oszlop (pl. ALVÁZ-SZÁM, MOTOR-SZÁM, TELJESÍTMÉNY, ...) helyett kiírt rövidítő jelölésként.

GÉPJÁRMŰ					
RENDSZÁM	GYÁRTMÁNY	TÍPUS	SZÍN	ÜRTARTALOM	EGYÉB
CYX 462	Mitsubishi	Colt GL	fehér	1298	...
DY 9680	Ford	Sierra L	ezüstmetál	1996	...
ABC 017	Daihatsu	Charade	kék	1298	...
V 24034	Ford	Escort D	fehér	1796	...
CZ 9404	Volkswagen	Fastback TL	zöld	1594	...
KF 9547	Volkswagen	Fastback TL	zöld	1594	...

7.2 ábra A gépjárművek adatait tartalmazó GÉPJÁRMŰ reláció

TULAJDONOS			
NÉV	LAKCÍM	FOGLAKOZÁS	EGYÉB
Kiss János 1	Budapest, I. Fő u. 1.	autószerelő	...
Quittner Pál	Budapest, V. Markó u. 1/b.	adatbázis szakértő	...
Lakinger Béla	Abádszalók, Hős u. 69.	tengerész	...
Kiss János 2	Pécs, Anna u. 16.	tanár	...
Computer Kft.	Győr, Rába köz 1/b.		...

7.3 ábra A gépjármű tulajdonosok adatait tartalmazó TULAJDONOS reláció

- Az oszlopok sorrendje nem befolyásolja az adatmodellt, de ha egyszer egy reláció fejlécében megadtuk, akkor annak a relációnak minden sorában ennek a sorrendnek megfelelően kell az attribútumok értékét megadnunk.³
- Mivel a reláció, a táblázat egyedei az egyes sorok, melyeknek azonosítása a tartalmuk alapján történik, a táblázatnak nem lehet két teljesen azonos sora, két olyan bejegyzése, melyben minden oszlop értéke megegyezik. (Ez a feltevés azt a valós világra vonatkozó reális megkötetést tartalmazza, hogy ha két eleme között semmiféle különbséget nem tudunk tenni, akkor azokat nem tekintjük különbözőnek. Jellemzésük tulajdonságai leírásával történik és legfőképpen—ha szükséges—, megadhatjuk, hány darab van összesen belőlük.)
- A sorok számát nevezzük a reláció *kardinalitásának*. Ez időben, új adatok bevitelével, meglévők törlésével dinamikusan változik. A sorok sorrendje a reláció szempontjából közömbös, az elvi modell szempontjából a soroknak nincsen kitüntetett rendezettsége.⁴

³ A balról jobbra való olvasás miatt azonban megszokott, hogy a „fontosabb” oszlopokat helyezzük balra, előre és a kevésbé fontosakat hátrábbra. Ez azonban csak egy kényelmi elrendezés, melynek, a modellt tekintve nincsen tartalmi jelentősége.

⁴ Gyakorlati szempontból, a hatékonyság növelése miatt azonban a tényleges fizikai tárolási sorrendet igen sokszor meghatározott rendezettség szerint valósítjuk meg.

— Elvileg a reláció minden sorában minden egyes oszlophoz egy, és pontosan egy értéket kell hozzárendelnünk. A gyakorlatban azonban igen sokszor előfordul, hogy bizonyos oszlopoknak az adatrekordnak az adatbázisba való bevitelkor még nem ismerjük az értékét, de a rekordot mégis tárolni kívánjuk. Ezt az ellentmondást kiküszöbölhetjük úgy, hogy minden oszlopnak egy automatikus *alapértelmezést* adunk (általában numerikus értéket tartalmazónál zérus, karakter típusúnál csupa szóköz). Így minden rekord minden oszlopa biztosan értékkel bír: vagy azzal, amit megadtunk, vagy az alapértelmezéssel.

Igen sok esetben azonban nagyon fontos különbséget tennünk aközött, hogy egy sorban valamely oszlopnak az értéke valóban 0 vagy üres, vagy csak azért áll ez benne, mert nem tudjuk, mi a tényleges értéke. (Nem mindegy, hogy valaki önkéntesként 0 forint fizetésért ingyen dolgozik, vagy még nem egyeztek meg a díjazásában.)

Ezért sok rendszer lehetővé teszi a *null-érték* ábrázolását is, ami azt jelenti, hogy az adott oszlop tartalma nem ismert. Természetes az ilyen oszlopokra vonatkozó műveletekben (pl. átlagszámítás) ezeket a sorokat, oszlopokat különleges módon kell a rendszernek kezelni.

Mivel azonban a null-érték megengedése bizonyos bizonytalanságot is jelent, mód van annak meghatározására, hogy mely oszlopok vehetnek föl null-értéket és melyek nem. Ez utóbbiaknál, ha nem adunk meg értéket (akár konkrét érték-ként, akár automatikusan definiált alapértelmezésként), akkor a rendszer nem engedi meg az adott sornak az adatbázisba való bevitelét.

— Minden relációban van az oszlopoknak legalább egy olyan kombinációja, amely egyértelműen meghatározza az adott sor további oszlopainak értékét. Ezt az oszlopot (oszlopkombinációt, vagy ezek tetszőlegesen kiválasztott egyikét) nevezzük *elsődleges kulcsnak*. Mivel egy táblázatnak nem lehet két egyforma sora, ilyen elsődleges kulcs mindig létezik, szélső esetben a reláció összes oszlopa alkotja. A kulcsok a relációs modellben kulcsszerepet töltenek be, ezért a 7.4 pontban részletesebben is foglalkozunk velük.

7.3 Táblázatok, nézetek

Az eddigiekben hallgatólagosan feltételeztük, hogy a relációs modell relációban (táblázataiban) levő adatokat ugyancsak táblázatos formában tároljuk is az adatbázisban. Amennyiben ez valóban mindig így lenne, úgy az adatbázis-kezelő rendszerek egyik legfontosabb előnyét, a redundancia mentességet veszítenénk el, hiszen akkor minden elemi adatot (oszlopértéket) annyiszor tárolnánk, ahány relációban az előfordul. Éppen ennek elkerülésére a relációs modell lehetőséget nyújt a fizikailag ténylegesen tárolt táblázatokból további, virtuális relációk előállítására

is, amelyekkel a legtöbb esetben ugyanúgy dolgozhatunk, mint a fizikailag valóban létező táblázatokkal.⁵

Elvileg a relációs modell az alábbi típusú relációkat engedi meg, bár megjegyezzük, hogy a kereskedelmi forgalomban kapható rendszerek nem mindegyike támogatja még az itt felsorolt lehetőségek mindegyikét.

Alap vagy bázis reláció

Fizikailag ténylegesen létező reláció, amely az adatokat a megadott formátumban ténylegesen tartalmazza. Általában ezt nevezzük a szűkebb, pontosabb értelemben táblázatnak.

Virtuális reláció vagy nézet

A felhasználó vagy az adatbázis-felügyelő által definiált reláció, amely önmagában nem tartalmaz adatokat. A nézet definiálásakor határozzuk meg, hogy az egyes sorok, oszlopok értéke mely táblázatok, nézetek soraiból, oszlopaiból milyen relációs műveletekkel jön létre.

Pillanatfelvétel

Adott táblázat(ok), nézet(ek) meghatározott adatait tartalmazza egy meghatározott pillanatban. Abban különbözik a nézettől, hogy adatai fizikailag is tárolva vannak. Rendszerint csak olvasható, és az adatbázis biztonságát, az adatok visszaállíthatóságát szolgálja. Ilyen pillanatfelvétel lehet például bizonyos táblázatok rendszeres napi kimentése.

Lekérdezések eredménye

Mint a neve is mutatja, célinformációkat szolgáltat a felhasználó számára. Nem adatbázis-reláció a szó szigorúan vett értelmében. Rendszerint pl. nincs önálló neve, esetenként tartalmazhat azonos sorokat, és csak addig él, míg a kapott információ feldolgozásra nem kerül.

Közbenső táblázatok

Bonyolultabb feladatok megoldásához az adatbázis-kezelő rendszernek ideiglenesen tárolnia kell a közbenső eredményeket. (Például a kiválasztott sorokat egy másik szempont szerint kell rendezni vagy az azonos sorokat a végeredményben ki kell szűrni.) Ezek rendszerint valódi, fizikailag is tárolt táblázatok (bár lehet, hogy az adatok egy részére csak pointerekkel hivatkoznak), melyek azonban csak ideiglenesen léteznek az adatbázisban. Általában a felhasználó által kért összetett művelet befejezésével vagy esetenként az adatbázissal való munka lezárásával automatikusan törlődnek a rendszerből.

A relációs modellben a felhasználó, illetve az adatbázis-felügyelő a relációkkal az adatmodell koncepcionális és külső szintjén írja csak le az adatok szerkezetét. Az adatok közti kapcsolatot a relációk, illetve a különböző relációk közötti kapcsolatokat az azonos tartományokra épülő attribútumok értékei közti összefüggések teljesülései (általában az attribútum értékek egyezősége) hozzák létre. Ellentétben a hierarchikus és a hálós modellel, itt nincsenek a felhasználó által definiálható láncok, pointerok, melyek ezeket a kapcsolatokat előállítják a külön-

⁵ A virtuális táblázatokban általában a beírás, módosítás, törlés lehet korlátozott, mert bizonyos esetekben ezek eredménye nem egyértelműen definiált, vagy ellentmondás keletkezhet az eredeti és a definiált adatok között. (Pl. definiáljuk az a és b oszlopokból a virtuális $c=a+b$ oszlopot. Ha c értéket meg akarnánk növelni, nem tudja a rendszer, hogy ehhez a , b -t hogyan kell megváltoztatni, hogy az egyezés továbbra is megmaradjon.)

böző entitások között és amelyeknek ismeretében a felhasználó navigálhat az adatbázisban, valamint ellenőrizheti az adatok integritását. Az adatmodell azonos és különböző típusú egyedei között az integritást a rendszer a következő pontban ismertetendő elsődleges és idegen kulcsok segítségével hozza létre és biztosítja, míg összekapcsolásukat a relációkon elvégezhető különböző műveletek teszik lehetővé. Ezeket a 7.5 és 7.6 pontokban ismertetjük részletesebben.

7.4 Kulcsok

7.4.1 Az adatbázis integritása

Az adatbázisban lévő adatok integritását, a különböző relációk ellentmondásmentességét az adatbázis-kezelő rendszernek biztosítani kell. Ezen szabályok egy része annyira alapvető, hogy teljesülésüket a rendszer automatikusan ellenőrzi és ha a feltételek nem állnak fenn, akkor az ilyen adatok bevitelét, módosítását vagy törlését nem is engedi meg. Az integritás feltételek másik része specifikus, alkalmazásfüggő. Ezek közé tartozik például az az általános érvényű feltétel, hogy minden attribútum csak a megfelelő tartomány értékeit veheti fel. Ennek aztán lehetnek további, alkalmazástól függő speciális esetei, hogy különböző attribútumok vagy attribútumok kombinációi között meghatározott kapcsolatoknak kell teljesülni. (Pl. a G&T táblázat bármelyik sorában mind a TULAJDONJOG-KEZDETE, mind a -VÉGE attribútumok értékét a valós dátumok tartományából vehetjük, de a KEZDETE értéke nem lehet nagyobb, mint a VÉGE.)

Az ilyen speciális feltételek egy részét beépíthetjük az adott adatbázis definíciójába (pl. dátum ellenőrzés, egy oszlop értéke csak numerikus lehet, vagy csak meghatározott értéket vehet fel stb.), más részüket esetenként a felhasználónak kell külön programmal ellenőriznie.

A konkrét adatbázis-kezelő rendszertől függ például, hogy milyen mértékben támogatja az attribútum–tartomány kapcsolatot, és mely összefüggések azok, melyek automatikus megvizsgálására már nincsen lehetőség az adatbázis-kezelő rendszerben. (A legtöbb esetben csak korlátozott mértékű ellenőrzés van. Az IBM SQL/DS korábbi változataiban csak az adattípust ellenőrzi. Az AS/400-nál értékhatárok és korlátozott számban konkrét értékek is megadhatók a tartományra, míg a DB2-ben és az SQL/DS 3. release-ben a felhasználó beépítheti a saját ellenőrző programját, mely minden módosításkor, beíráskor automatikusan aktivizálódik.)

Éppen ezért ebben a pontban csak az általános, minden relációs adatbázisra érvényes integritási szabályokkal foglalkozunk. Ezek közül az első az egyes relációk egyedeinek egyértelmű azonosítására szolgáló *elsődleges kulcs*, amit gyakran egyszerűen—és tévesen—kulcsnak is szoktak nevezni.⁶

⁶Ez a pongyolaság csak akkor engedhető meg, ha a sorok azonosítása és elrendezése mindig csak egyetlenegy szempont szerint történne. Ennek ellenére a mindennapi szóhasználatban gyakran és főleg a marketing célú publikációkban, de még a szakirodalomban is találkozunk vele. Könyvünkben a félreérthetőség elkerülése érdekében, ha az elsődleges kulcsról van szó, mindig kiírjuk elé a jelzőt, kivéve esetleg az olyan szöveggörnyezetet, ahol azt már ismételtlen kitettük és az összefüggésből már egyértelmű, hogy továbbra is az elsődleges kulcs értendő a kulcsra.

Az integritást biztosító kulcsok másik csoportja az *idegen kulcs*, amelynek segítségével a rendszer automatikusan ellenőrizni tudja, hogy különböző relációk azonos tartományra épülő attribútumainak értékei között ne legyen ellentmondás. (Például, ne tölthessünk ki szállítólevelet nem létező megrendelőre vagy árura.)⁷

A fenti két alapvető integritási szabály a valós élet modellezéséből adódik. Ezért az adatbázis-kezelő rendszer csak a valós objektumok, az alap vagy bázisrelációk, a táblázatok integritásának ellenőrzésére használja fel ezeket.

A nézetek és egyéb másodlagos relációk integritása már alkalmazás-specifikus, függ a képzett reláció definíciójától. Ezért ezek betartásáról a felhasználónak vagy az adatbázis-felügyelőnek kell gondoskodnia, ha lehet, úgy, hogy beépíti a képzett reláció definíciójába (így az ellenőrzés automatikus), vagy külön alkalmazási programmal.

7.4.2 Elsődleges kulcs

Mint azt a 7.2 pontban a relációk tulajdonságainak definíciójánál felsoroltuk, egy reláció bármelyik sorát tartalma alapján azonosítjuk, és minden relációban létezik az oszlopoknak legalább egy olyan kombinációja, amelyik az adott relációban mindig egyedi és ezáltal mindig egyértelműen meghatározza a sor többi oszlopának az értékét.⁸

A gyakorlatban előfordulhat, hogy egy reláció sorait nem csak egy oszlopkombináció azonosíthat egyértelműen. Így például egy személyi nyilvántartásban a korábbi személyi szám mellett azonosíthat egy személyt a neve, anyja neve, születési éve és helye oszlopkombináció is, vagy ha a nyilvántartásunk olyan—mint ahogy azt a 7.2 pontban feltételeztük—, hogy *mindig* egyedi neveket tudunk használni, akkor akár a név önmagában is.⁹ Ebben az esetben ezeket az oszlopokat (oszlop-kombinációkat) *lehetséges kulcsoknak* (candidate key) nevezzük, és közülük bármelyiket választhatjuk elsődleges kulcsnak. Ha ez megtörtént, akkor a többi lehetséges kulcsot a reláció *alternatív kulcsának* nevezzük.

Azt, hogy a lehetséges kulcsok közül végül melyiket választjuk ki elsődleges kulcsként a modell szempontjából elvileg közömbös, gyakorlati szempontból azonban általában nem mindegy. A relációk egyes sorait igen sok rendszer az elsődleges kulcs alapján helyezi el fizikailag, így a rekordok eszerint sűrűsödnek. Ez azt jelenti, hogy elsődleges kulcsnak célszerű a lehetséges kulcsok közül a kiválasztásra leggyakrabban használtat kijelölni. Így például, ha az elemek peri-

⁷ Sajnos egyenlőre még sok kereskedelmi forgalomban lévő rendszer sem támogatja az idegen kulcsokat. Így pl. az IBM igen elterjedt AS/400 gépcsaládjára kidolgozott SQL/400 sem teszi lehetővé—legalább is e sorok írásakor—az idegen kulcsok definiálását és ezáltal automatikus ellenőrzésüket.

⁸ A relációs modellben az azonosítás tartalom alapján, asszociatív módon történik. Ez azonban csak a modell logikai szintjén igaz. A modell implementálásához, a fizikai címzéshez, nincs szükség asszociatív memóriájú hardverre. Éppen ellenkezőleg, az eddigi sikeres rendszerek többsége a hagyományos, hely szerinti címzést használja és a hagyományos I/O technikát tartalmazó hardverre és operációs rendszerre épül rá.

⁹ Itt most lényeges, hogy az egyediségnek nem csak az akkor éppen aktuális adatállományra kell teljesülnie, hanem az adatbázis reláció teljes létezésének időtartamára.

ódusos rendszerét tartalmazná egy relációnk, akkor a vegyjelet, az elem neve, és a rendszám (pl. H, hidrogén, 1) mint lehetséges kulcsok közül a rendszámot valószínűleg eleve elvetnénk. Ha fizikusok vagy kémikusok dolgoznának elsősorban a relációval, akkor inkább a vegyjelet, különben az elem nevét definiálnánk elsődleges kulcsnak.

A reláción belül az egyedi elsődleges kulcsnak még az alábbi feltételeket is teljesítenie kell:

— *A kulcs, illetve összetett kulcsnál annak egy komponense sem lehet null-érték.* Ez annak az ésszerű logikai feltételnek az átfogalmazása a relációs modell terminológiájába, hogy csak azzal tudunk valamit azonosítani, amit ismerünk, aminek tudjuk az értékét.

— *Összetett kulcsnál a komponensek száma minimális.* Bármelyik oszlopot elhagynánk a kulcsból, az egyértelműség megszűnne. Az ezen pont elején említett példában, ha olyan neveink vannak, hogy a név mindig egyértelműen azonosít, akkor név, anyja neve, születési év, születési hely kombináció nem elsődleges kulcs, mert nem minimális. Természetesen ezek a feltételek most is nem csak az éppen aktuális adatokra vonatkoznak. Hiszen ha a név nem biztosan egyértelmű, akkor is elég nagy valószínűséggel előfordulhat, hogy nincs két azonos nevű személy a relációban. De ettől azért ebben az esetben a nevet nem tekinthetjük elsődleges kulcsnak, mert bármikor felbukkanhat egy olyan személy, akit fel kell vennünk a nyilvántartásba, holott ilyen névvel már egy másik ember benne van.

Az elsődleges kulcsok kijelölése az adatbázis-tervezés egyik legfontosabb feladata. A konkrét adatmodell azonban a legtöbbször nyilvánvalóvá teszi, hogy mi legyen egy reláció elsődleges kulcsa. Amennyiben azonban további alternatív kulcsok is léteznek, azok egyediségét külön kell közölnünk az adatbázis leírásakor, hogy ezt a rendszer automatikusan biztosítani tudja. Ez az alternatív kulcsra definiált egyedi indexszel (l. 4.7 és CREATE UNIQUE INDEX, 10.4) valósítható meg.

A példában szereplő G&T táblázat elsődleges kulcsa összetett kulcs, a RENDSZÁM és a TULAJDONOS oszlopok alkotják.¹⁰

7.4.3 Idegen kulcsok

Ahhoz, hogy az adatbázisunk ellentmondásmentes legyen, bizonyos relációk között meghatározott összefüggéseknek állandóan érvényesnek kell lenni. Ezek közül a legfontosabb az, amikor az egyik relációban, a *hivatkozó* relációban, egy oszlop vagy oszlopkombináció csak olyan értéket vehet fel, amelyik egy másik, a *hivatkozott* reláció egy elsődleges kulcsának értékével egyenlő, vagy legfeljebb a null-értéket veheti fel. Ezt az oszlopot (kombinációt) nevezik a *hivatkozó* relációban *idegen kulcsnak*. (Az egymással állandó kapcsolatban álló relációk szokásos elnevezése még a *függő* illetve *szülő* reláció vagy táblázat is.)

¹⁰ Így ebben a modellben ugyanaz a személy ugyanazta gépjárművet kétszer nem vásárolhatja meg. Az egyszerűsítés kedvéért most ezt a korlátozást elfogadjuk.

Ilyen idegen kulcs a G&T táblázatban a RENDSZÁM és a TULAJDONOS, mely a GÉPJÁRMŰ illetve a TULAJDONOS táblázatok elsődleges kulcsára hivatkozik. Látjuk, hogy az egyes oszlopok nevei különbözőek is lehetnek a hivatkozó és a hivatkozott táblázatban, de természetesen tartalmilag azonos típusú adatokra vonatkoznak. (Általában ugyanarra a tartományra épülnek, esetleg a hivatkozott táblázat megfelelő tartományának egy részére.) Az sem okoz gondot, hogy egy táblázat egyik oszlopának neve megegyezik egy másik táblázat nevével. A fenti példában az idegen kulcsok egyben részei a hivatkozó táblázat elsődleges kulcsának is. Ez azonban szintén nem kötelező. Az idegen kulcs kapcsolat akkor is érvényes lenne, ha az idegen kulcs nem volna megegyező az elsődleges vagy bármely alternatív kulccsal, vagy annak valamelyik összetevőjével a hivatkozó táblázatban. Ez utóbbira példa lehet egy személyi nyilvántartás, ahol az ALKALMAZOTTAK táblázat tartalmazza az alkalmazottak személyi adatait és köztük az őket foglalkoztató osztály kódját is (l. 7.9 pontnál a 7.21 ábrát). A reláció elsődleges kulcsa az alkalmazott azonosító száma (a vezetéknev-keresztnev kombináció egyedi névképzés esetén alternatív kulcs), az osztálykód pedig idegen kulcs.

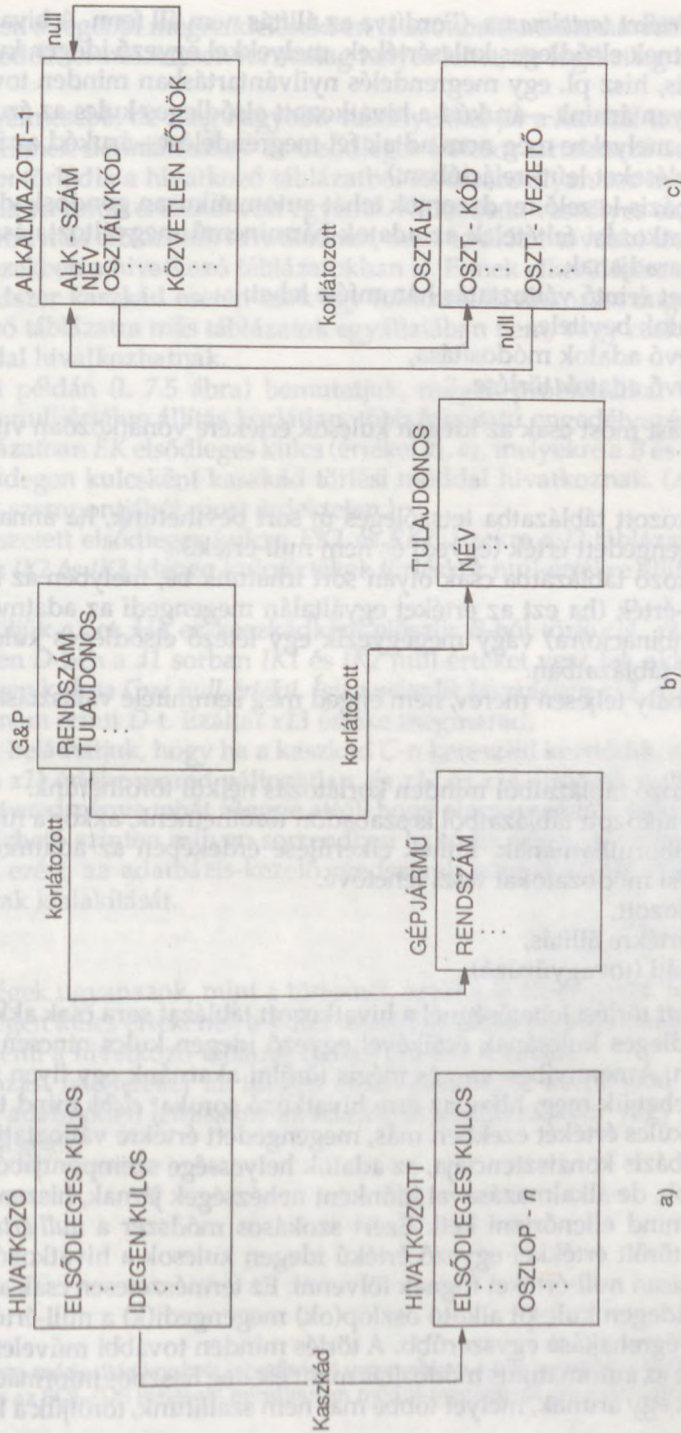
Megengedett az is, hogy ugyanaz legyen a hivatkozó és a hivatkozott reláció. Erre példa lehetne a 7.21 ábra ALKALMAZOTTAK táblázatának egy olyan kibővítése, mely tartalmazza az alkalmazott közvetlen főnökének (feltételezve, hogy csak egy lehet) az azonosító számát. Ez az adatbázis konzisztenciája miatt természetesen csak olyan érték lehet, amelyik már szerepel az alkalmazotti számok között. A fenti egyben jó példa arra is, hogy miért kell megengedni idegen kulcsnál a nulla értéket is. A vállalat legelső emberének (elnök, vezérigazgató) nem lehet már főnöke, ezért ebben a sorban ennek az attribútumnak az értéke csak a nulla érték lehet.

A relációk között idegen kulcs alapján definiált kapcsolatok egyik szokásos ábrázolási módját a 7.4 ábrán mutatjuk be. A nyíl a hivatkozó táblázatból mutat a hivatkozottra, és általában tartalmazza a kulcs módosíthatóságának, törlésének módját is (l. később ebben a pontban). Az idegen kulcs nevét vagy a relációban, vagy a kapcsolat-nyíl mellett tüntetjük fel.¹¹

Elvileg a hivatkozó táblázat is lehet további vonatkozásban hivatkozott táblázat, és az is előfordulhat, hogy egy hivatkozási ciklus több táblázat érintésével záródik (7.4c ábra). Itt az ALKALMAZOTT-1 táblázatban csak olyan dolgozók lehetnek, akiknél a KÖZVETLEN-FŐNÖK száma szerepel ugyanezen táblázat egy másik sorában ALK-SZÁM-ként vagy null-érték (önmagára való hivatkozás), OSZTÁLYKÓD-ja null-érték vagy az OSZTÁLY táblázatban már létezik (egyszerű hivatkozás). Az OSZTÁLY táblázatban csak olyan sorok lehetnek, melyekben az OSZT-VEZETŐ értéke null-érték, vagy az érvényes ALK-SZÁM az ALKALMAZOTT-1 táblázatban (egyszerű hivatkozás és egyben hivatkozási ciklus létrehozása).

Az idegen kulcsok definiálásával feltételeket szabunk különböző relációkra. Ezek összességét *hivatkozási feltételeknek* vagy *függőségi kapcsolatoknak* is nevezzük. Ezek lényegét röviden úgy foglалhatjuk össze: az adatbázisban nem lehet olyan idegen kulcs, melynek értéke nem egyezik meg egy hozzákapcsolódó elsődleges kulcs értékével vagy

¹¹ Lehet—és szokásos—a nyilat a hivatkozó táblázat felé irányítani. Ez ellentmond a hivatkozási logikának, de megfelel a hivatkozás által létrejövő 1:N kapcsolat ábrázolási módjának.



7.4 ábra Relációk közti kapcsolatok ábrázolása
 a) általános formában b) a GÉPJÁRMŰ NYILVANTARTÁS modellben c) hivatkozási ciklus

nem a null-értéket tartalmazza. (Fordítva az állítás nem áll fenn. A hivatkozott relációban lehetnek elsődleges kulcsértékek, melyekkel egyező idegen kulcs nincsen. Ez érthető is, hisz pl. egy megrendelés nyilvántartásban minden további nélkül lehetnek olyan áruink—árukód a hivatkozott elsődleges kulcs az árukat leíró relációban—, melyekre még nem adtak fel megrendelést—árukód az idegen kulcs a megrendeléseket leíró relációban.)

Az adatbázis-kezelő rendszernek tehát automatikusan gondoskodni kell arról, hogy a hivatkozási feltételek az adatok bármilyen megváltoztatása után is érvényben maradjanak.

A feltételeket érintő változtatás háromféle lehet:

- új adatok bevitele,
- meglevő adatok módosítása,
- meglevő adatok törlése.

(A korlátozást most csak az idegen kulcsok értékére vonatkozóan vizsgáljuk.)

Bevitel

A hivatkozott táblázatba tetszőleges új sort bevihetünk, ha annak elsődleges kulcsa megengedett érték (egyedi és nem null-érték.)

A hivatkozó táblázatba csak olyan sort írhatunk be, melyben az idegen kulcs értéke null-érték (ha ezt az értéket egyáltalán megengedi az adatmodell erre az oszlop(kombináció)ra) vagy megegyezik egy létező elsődleges kulcs értékkel a hivatkozott táblázatban.

Ez a szabály teljesen merev, nem enged meg semmiféle választási lehetőséget.

Törlés

A hivatkozó táblázatból minden korlátozás nélkül törölhetünk.

Ha a hivatkozott táblázatból is szabadon törölhetnénk, akkor a függőségi kapcsolatok felborulhatnának. Ennek elkerülése érdekében az adatmodell csak az alábbi törlési módokat teszi lehetővé:

- korlátozott,
- null-értékre állítás,
- kaszkád (tovagyűrűző).

Korlátozott törlési lehetőségénél a hivatkozott táblázat sora csak akkor törölhető, ha az elsődleges kulcsának értékével egyező idegen kulcs nincsen a hivatkozó táblázatban. Amennyiben van és mégis törölni akarnánk egy ilyen rekordot, azt csak úgy tehetjük meg, hogy az erre hivatkozó sorokat *előbb* mind töröljük vagy az idegen kulcs értékét ezekben más, megengedett értékre változtatjuk.

Az adatbázis konzisztenciája, az adatok helyessége szempontjából ez a mód a legbiztosabb, de alkalmazásával időnként nehézségek járnak, hiszen a függő táblázatokat mind ellenőrizni kell. Ezért szokásos módszer a *null-értékre állítás* is, melynél a törölt értékkel egyező értékű idegen kulcsok a hivatkozó táblázatban automatikusan null-értéket fognak fölvenni. Ez természetesen csak akkor lehetséges, ha az idegen kulcsot alkotó oszlop(ok) megengedi(k) a null-értéket.

Ennek végrehajtása egyszerűbb. A törlés minden további művelet nélkül elvégezhető, de az automatikus módosítás miatt esetleg hasznos információk is elveszhetnek. (Pl. egy árunak, melyet többé már nem szállítunk, töröljük a kódját, minek

következtében a régebbi megrendelésekben is azonosíthatatlanná válik ez az áru. Egy ilyen modell természetesen formailag helyes lehet, de logikailag mindenképpen hibás!)

Még kényelmesebb, de még nagyobb veszélyekkel jár a *kaszkád* (tovagyűrűző) változtatás. Ennek definiálásakor az elsődleges kulcsot tartalmazó sorral együtt *automatikusan* törlődik a hivatkozó táblázatból az összes olyan sor is, amelyikben az idegen kulcs a törölt értékkel volt egyenlő. Különösen veszélyes ez, ha a hivatkozó táblázatra más táblázatok hivatkoznak, mert ez akkor további törléseket von maga után ezekben a hivatkozó táblázatokban is. Ennek elkerülésére a gyakorlatban sok rendszer kaszkád esetén csak egy fokozatú hivatkozást enged meg. Az így hivatkozó táblázatra más táblázatok egyáltalában nem, vagy csak korlátozott törlési móddal hivatkozhatnak.

Az alábbi példán (l. 7.5 ábra) bemutatjuk, milyen problémákat vethet fel a kaszkád és a null-értékre állítás korlátlan, több fokozatú engedélyezése.

Az *A* táblázatban *EK* elsődleges kulcs (értékei a_1, a_2 , melyekre a *B* és *C* táblázat *IK* oszlopai idegen kulcsként kaszkád törlési móddal hivatkoznak. (A többi oszlopa a példa szempontjából most érdektelen.)

B és *C* összetett elsődleges kulcsa *EK1* és *EK2*. Ezekre a *D* táblázatból az *IK1*, *IK2* illetve az *IK2* és *IK3* idegen kulcsértékek törléskor null-értékre állítással hivatkoznak.

A-ból töröljük a_1 -et. Ha ez kaszkádként először *B*-ből törli x_{11} , x_{12} -t, aminek következtében *D*-ben a d_1 sorban *IK1* és *IK2* null-értéket vesz fel, akkor *IK2* null miatt d_1 idegen kulcsa *C*-re null-értékű. Így a második kaszkádban x_{12} , x_{13} -nak *C*-ből való törlése nem érinti *D*-t. Ezáltal x_{13} értéke megmarad.

Ugyanígy beláthatjuk, hogy ha a kaszkád *C*-n keresztül kezdődik, akkor *D*-nek a d_1 sorában x_{11} értéke marad változatlan, és x_{12} és x_{13} állítódik null-értékre.

A művelet eredménye tehát függne attól, hogy alacsonyabb, a felhasználó által nem ellenőrizhető szinten milyen sorrendben hajtódik végre. Ez nem engedhető meg. Éppen ezért, az adatbázis-kezelő rendszerek nem is teszik lehetővé ilyen függőségi utak kialakítását.

Módosítás

A lehetőségek ugyanazok, mint a törlésnél, azzal a különbséggel, hogy a kaszkád itt az idegen kulcs értékének a kulcs új értékére történő automatikus megváltoztatását jelenti a hivatkozó táblázat összes érintett sorában.¹²

A hivatkozási feltételeket az idegen és az elsődleges kulcsokkal együtt kell definiálni az adatmodell leírásakor az adatbázis létrehozásakor vagy szerkezetének módosításakor.

¹² A törlés és a módosítás konkrét lehetőségei ugyanabban a rendszerben is különbözhetnek. Így pl. a DB2 és az SQL/DS törlésnél mindhárom módot megengedi, de módosítani csak korlátozottan lehet.

vetlenül elérhető, szekvenciálisan hatékonyan feldolgozhatóak és egyediségük gyorsan ellenőrizhető.

Ellentétben az elsődleges és idegen kulcsokkal, melyek minden relációs modellen alapuló adatbázisnak alapvető jellemzői, az indexek mindig alkalmazásfüggők. A megoldandó feladat dönti el, hogy milyen táblázatokra, milyen oszlopokra és milyen típusú indexet kell létrehozni az optimális feldolgozás érdekében.

A modell rugalmasságát mutatja, hogy a relációs adatbázis-kezelő rendszerek egyszerű módszerekkel lehetővé teszik már meglévő adatbázisoknál is új indexek, elsődleges és idegen kulcsok definiálását, a meglévők törlését. Ez természetesen csak akkor hajtható végre, ha a meglévő adatok továbbra is elegendő tesznek a kulcsok által definiált integritási feltételeknek.

7.5 Műveletek relációkon

7.5.1 A műveletek célja

A relációs modell legfőbb előnye az, hogy a különféle feltételeknek megfelelő, fizikailag is létező egyedeit vagy ezekből logikai úton előállítható, vagy csak ideiglenes egyedeit könnyen, egyszerűen meg tudjuk határozni, azonosíthatjuk, kiválaszthatjuk, definiálhatjuk őket. A relációs műveleteket az alábbi feladatok elvégzésére használjuk:

- A kereséskor kiválasztandó adatok meghatározása;
- A módosítandó, törlendő adatok kiválasztása;
- A beviendő adatok meghatározása;
- Virtuális adatok definiálása. Ezek lehetnek az adatmodellnek állandó, névvel azonosított relációi (nézetek, pillanatfelvételek) vagy csak ideiglenesen létezők, melyek az adott feladat megoldásához szükségesek és utána automatikusan megszűnnek.

A fentiekén kívül felhasználjuk még a relációs műveletek az adatbázishoz való hozzáférési jogok meghatározásánál és integritási feltételek megadásánál is, amikor is ezek segítségével jelöljük ki a védelemmel rendelkező, lezárandó, illetve egymással kapcsolatban álló objektumok, relációk körét.

Mivel a műveletekkel leggyakrabban adatokat választunk ki, a továbbiakban általában kiválasztásról, kiválasztási műveletről fogunk beszélni, de ezalatt nem csak a szűkebb értelemben vett kiválasztást értjük majd, hanem a fent felsorolt funkciók bármelyikét.

7.5.2 Relációs algebra és kalkulus

A kiválasztási feltételeket a *relációs algebra* vagy a *relációs kalkulus* nyelvén fogalmazhatjuk meg. A relációs algebra nyelve eljárás-orientált, pontosan megadja, hogyan, milyen lépésekkel oldjuk meg a feladatot. A relációs kalkulus a feladat megfogalmazására összpontosít, a hogyan már a nyelvet értelmező szoftverre bízta. A két nyelv egymással egyenértékű. Ugyanaz a feladat megfogalmazható a relációs kalkulus műveleteivel és a relációs algebra formalizmusa és szabályai

szerint is. A gyakorlati alkalmazásoknál a nyelvek szintaxisa önálló vagy illeszkedik a befogadó programnyelvhez.

Az információs technológiában ma már kvázi-szabványként elfogadott SQL nyelv—melyet a III. részben fogunk részletesen tárgyalni—lényegében a relációs kalkuluson alapszik. A hangsúlyt a *mit*-re teszi. A *hogyan* legfőljebb azért lehet érdekes az alkalmazó számára, mert a konkrét feladatok megoldásának hatékonyságát jelentős mértékben befolyásolhatja.

7.5.3 Relációs műveletek

Bármelyik nyelven is fogalmazzuk meg a feladatokat, azok végül relációkon elvégezhető különféle műveletekre vezethetők vissza. Ezek között vannak elemi műveletek és vannak összetettek, melyeket gyakori előfordulásuk miatt célszerű önálló műveletként definiálni, de visszavezethetők az alpműveletekre.

Az alábbiakban ismertetjük a gyakorlatban leginkább használt relációs műveleteket. A lista nem teljes, de nem is minimális, azaz a felsoroltak között vannak—mint azt már említettük—nem elemi műveletek is. Hangsúlyozni kívánjuk azonban, hogy a rendszer ezen műveletek mindegyikére *zárt*. Ez azt jelenti, hogy ha bármelyik relációra vagy relációkra alkalmazzuk ezen műveletek akármelyikét, eredményül újra egy relációt kapunk, melyre ezek a műveletek ismételten alkalmazhatók.

Az általunk alapvető fontosságúnak tekintett és ezért önállóan ismertetett relációs műveletek a következők:

- átnevezés (RENAME),
- korlátozás vagy restrikció (RESTRICT),
- vetület vagy projekció (PROJECT),
- keresztszorzat (TIMES),
- unio (UNION),
- metszet (INTERSECT),
- különbség (DIFFERENCE),
- egyesítés vagy összekapcsolás (JOIN),
- egyesítés alapértelmezéssel (külső vagy OUTER JOIN)
- bővítés (EXTEND),
- csoportfüggvény képzés (SUMMARIZE),
- kijelölés.

A relációs műveleteket csoportosíthatjuk aszerint is, hogy egy vagy két-operandusra van-e hatásuk.

Az előzőekből következik, hogy a két-operandusos műveletek érvényessége kiterjeszthető akárhány operandusra. Az ismételten alkalmazott művelet egyik operandusa lesz az előző művelet eredménye, a másik pedig egyenként a harmadiknak, ... stb. tekinthető további operandusok.

A több-operandusos műveletek zöménél az operandusok felcserélhetők. A különböző műveletek sorrendje is általában tetszőleges. A relációs műveleteknek a tényleges fizikai író/olvasó utasításokra való átfordításakor van ezeknek a tulajdonságoknak jelentősége. A műveleteket az optimális vagy hibás sorrendben el-

végezve nagyságrendekkel is eltérő válaszütemeket kaphatunk. Azokra a műveletekre, melyekre a fenti feltételek nem állnak fenn, külön utalni fogunk.

7.6 A legfontosabb műveletek definíciója

7.6.1 Átnevezés (RENAME)

Az egy-relációs műveletek közül a legegyszerűbb elemi művelet az *átnevezés* (RENAME). Ez egyszerűen átnevezi az R reláció egy oszlopát.

Formája:

$$R \text{ RENAME } (r_1, r_2)$$

ahol r_1 az eredeti, r_2 pedig az új oszlopnév.

Ugyanazon reláció több oszlopának átnevezését vagy a RENAME elemi művelet ismételt alkalmazásával végezhetjük el, vagy az utasításnak $R \text{ RENAME } (r_{11}, r_{12}; r_{21}, r_{22}; \dots, r_{m1}, r_{m2})$ -re való bővített definiálásával, ahol r_{i1}, r_{i2} az összetartozó régi és új oszlopnév párok.

Erre a műveletre a zártság miatt van szükségünk. Enélkül pl. a két-relációs műveletek eredménye olyan „reláció” lehetne, mely azonos nevű oszlopokat tartalmazna.

7.6.2 Korlátozás (RESTRICT)

Egy-relációs művelet a *korlátozás* vagy *restrikció* (RESTRICT).¹³ Az eredmény az R relációnak azokat és csak azokat a sorait tartalmazza, melyeknél az r_1 oszlop értékére teljesül az F_1 korlátozó feltétel. Ez általában az $<, >, =, \dots$ stb. összehasonlítást jelenti, ahol az összehasonlítandó egy literál, a sor egy másik oszlopának az értéke vagy egy relációs művelettel kapott skalár érték.

Általános formája:

$$\text{RESTRICT } R \text{ WHERE } r_1:F_1$$

Az eredmény reláció oszlopainak neve, és elsődleges kulcsa ugyanaz, mint a kiinduló relációé. A különböző összehasonlítási lehetőségeket részletesen tárgyaljuk majd a 11.2 pontban a SELECT SQL utasítás WHERE klauzulájában.

Alapértelmezésben az F_1 korlátozási feltétel általában egy egyszerű logikai feltétel.

Pl. $\text{RESTRICT GÉPJÁRMŰ WHERE GYÁRTMÁNY} = \text{'Ford'}$

a GÉPJÁRMŰ táblázat összes 'Ford' gyártmányú gépkocsiját adja eredményül.

A gyakorlatban azonban igen sokszor összetett feltételek alapján kell kiválasztanunk a korlátozás eredményeként kapott sorokat. Ezek a feltételek vonatkozhat-

¹³ Gyakran kiválasztásnak, szelekciónak (SELECT) is nevezik. Ez azonban félreértésekre vezethet, mert az SQL nyelv kiválasztó utasításának is SELECT az elnevezése, mely azonban ennél sokkal összetettebb. Alkalmazásával az összes relációs művelet elvégezhető.

nak ugyanarra az oszlopra vagy különböző oszlopokra, kapcsolódhatnak logikai ÉS-sel (AND) vagy VAGY-gyal (OR) és tartalmazhatnak negációt (NOT) és zárójeleket is.

Pl. az 1600 cm³ úrtartalomnál nagyobb Ford Sierra és Ford Escort gépkocsikat az alábbi korlátozások bármelyikével (és más hasonlókkal) kiválaszthatjuk:

```
RESTRICT GÉPJÁRMŰ WHERE GYÁRTMÁNY='Ford'
AND ÚRTARTALOM>1600
AND (TÍPUS='Sierra' OR TÍPUS='Escort')
```

vagy

```
RESTRICT GÉPJÁRMŰ WHERE
(GYÁRTMÁNY='Ford' AND TÍPUS='Escort' AND
ÚRTARTALOM>1600)
OR
(GYÁRTMÁNY='Ford' AND TÍPUS='Sierra' AND
ÚRTARTALOM>1600)
```

(Amennyiben a kérdés feltevésekor eleve tudjuk, hogy „Sierra” és „Escort” típusú gépkocsikat csak a „Ford” gyárt, akkor a GYÁRTMÁNY-ra vonatkozó feltételt el is hagyhatjuk a korlátozásból, azt egyszerűbbé tehetjük. Az ilyenfajta egyszerűsítési lehetőség azonban csak a konkrét adatmodell tulajdonságaiból adódik, ezért ezekkel sem itt, sem a relációs műveletek további tárgyalásánál nem foglalkozunk.)

A legtöbb adatbázis-kezelő rendszer elemi relációs műveletként kezeli a logikai kifejezést tartalmazó korlátozást is. Definiálható azonban az összetett feltétel relációs megvalósítása úgy is, hogy a korlátozás műveletében csak elemi feltétel szerepelhet. Az összetett feltétel eredménye az egyedi elemi korlátozás eredményeképpen kapott relációkból állítható elő további, két-operandusos relációs műveletek eredményeképpen. (l. majd az unió és a metszet műveleteket.)

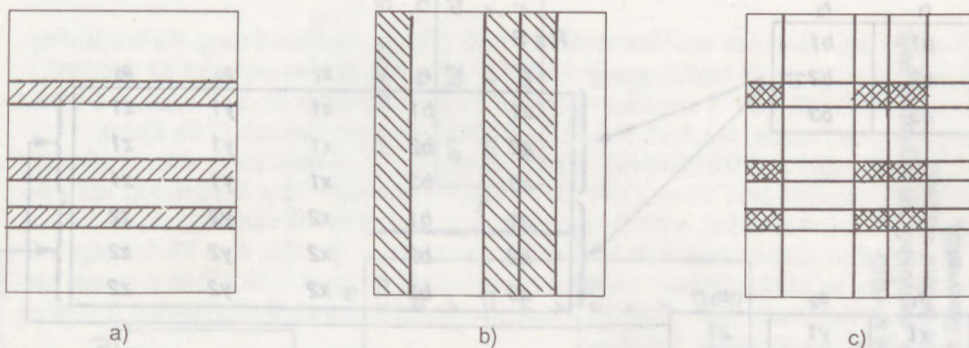
7.6.3 Vetület (PROJECT)

A harmadik egy-operandusos alapl művelet a vetület vagy projekció (PROJECT). Ez az R relációból csak a vetületben megadott oszlopokat tartja meg ugyanazon névvel azonosítva:

$R \text{ PROJECT } [r_1, r_2, \dots, r_n]$

ahol $r_1, r_2, \dots, r_n$ az eredeti R relációnak az eredményben is megmaradó oszlopai. Sorrendjük különbözhet az eredeti relációban lévő sorrendtől. Mivel a relációk halmaza a műveletekre zárt, kell hogy a projekció eredménye is teljesítse a relációkra vonatkozó 7.2 feltételeket. Ezért a fenti definíciót még ki kell egészítenünk azzal, hogy amennyiben az oszlopok elhagyásával azonos sorok keletkeztek (pl. a GÉPJÁRMŰ reláció 7.2 ábrán látható adatainak a GYÁRTMÁNY és ÚRTARTALOM oszlopokra való projekciójánál), akkor ezekből egyet tartunk csak meg. Ha a projekció tartalmazza az eredeti reláció elsődleges kulcsát, akkor az eredménynek is ez lesz az elsődleges kulcsa, különben a kulcsot külön meg kell határozni.

A projekciót egy reláció oszlopainak felecserelésére is használhatjuk. A 7.6 ábrán láthatjuk a korlátozás és a projekció hatását egy adott relációra.



7.6 ábra Egy-relációs műveletek

a) korlátozás b) projekció c) korlátozás és projekció
 (a két művelet elvégzésének sorrendje tetszőleges)

7.6.4 Keresztszorzat (TIMES)

A két-relációs műveletek közül a legalapvetőbb elemi művelet a keresztszorzat. Az $R(r_1, r_2, \dots, r_m)$ és az $S(s_1, s_2, \dots, s_n)$ m illetve n foksámú reláció keresztszorzata $R \times S$ vagy $R \text{ TIMES } S$, az az $m+n$ foksámú reláció, melynek oszlopait úgy kapjuk, hogy az R reláció oszlopai mögé írjuk S összes oszlopát, sorait pedig úgy, hogy R minden egyes sora mögé hozzáírjuk S minden egyes sorát. Ha az eredeti relációk sorainak száma, kardinalitása i , ill. j volt, akkor a keresztszorzaté $i \cdot j$ lesz. (l. 7.7 ábra.)

A szorzat elsődleges kulcsa a két tényező reláció elsődleges kulcsának egyesítése, konkatenációja lesz.

A szorzat oszlopainak a neve megegyezik az egyes tényezők oszlopainak a nevével. A zártságból következik, hogy a művelet csak akkor hajtható végre, ha a tényezők *szorzatkompatibilisek*, azaz nincsenek bennük azonos nevű oszlopok. Amennyiben ez nem állna fenn, akkor az átnevező relációs művelettel (RENAME) előbb át kell neveznünk az R és/vagy az S reláció megfelelő oszlopait.

A szokásos eljárás az (és a III. részben ismertetendő SQL nyelv automatikusan ezt is alkalmazza), hogy minden oszlopnevet automatikusan minősítenek a reláció nevével. Így, ha az oszlopnev nem egyértelmű, akkor ezáltal tehető és teendő egyértelművé:

relációnév.oszlopnev

(Például: GÉPJÁRMŰ.RENDSZÁM vagy G&T.RENDSZÁM)

Ez az automatizmus csak akkor nem alkalmazható, ha egy relációnak önmagával képezzük a keresztszorzatát. (Ez minden további nélkül megengedett, sőt, a gyakorlatban is előforduló művelet.) Ebben az esetben saját magunknak kell egyértelműsíteni az oszlopneveket.

R

r_1	r_2
a1	b1
a2	b2
a3	b3

S

s_1	s_2	s_3
x1	y1	z1
x2	y2	z2

 $R \times S$

r_1	r_2	s_1	s_2	s_3
a1	b1	x1	y1	z1
a2	b2	x1	y1	z1
a3	b3	x1	y1	z1
a1	b1	x2	y2	z2
a2	b2	x2	y2	z2
a3	b3	x2	y2	z2

7.7 ábra Két reláció (R és S) keresztszorzata
Az oszlopok nevei a két relációban különböznek, így a szorzás előtt nincs szükség az átnevezés művelet elvégzésére.

Megjegyezzük, hogy bár a keresztszorzat elvileg igen fontos alapművelet, a gyakorlati alkalmazásoknál, amennyire csak lehetséges, igyekeznek elkerülni korlátozás nélküli alkalmazását. Ennek az az egyszerű oka, hogy amíg pl. 10 000 sorból álló relációk kezelése önmagában általában még nem vet fel hatékonysági problémákat, addig ezek keresztszorzata már $10\,000 \times 10\,000 = 10^8$ sort tartalmaz. Ennek tárolása, az ebben való soros keresés vagy a közvetlen hozzáféréshez ideiglenes indexek felépítése már komoly tárolási és válaszidő problémákat vehet föl.

7.6.5 Unió (UNION)

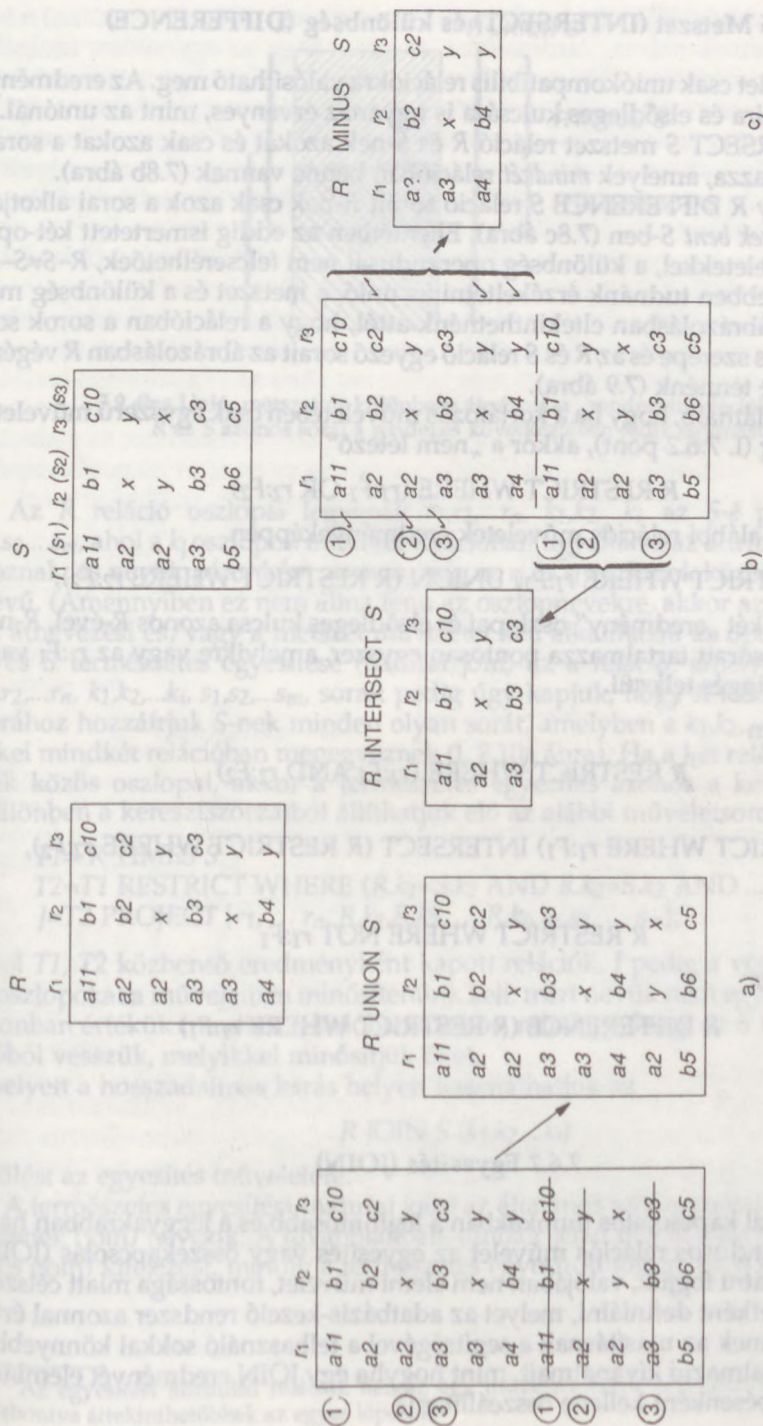
Két reláció unióját $R \cup S$ -t akkor tudjuk képezni, ha azok unió-kompatibilisek, azaz mindkét relációban ugyanazok az oszlopok vannak. Ebben az esetben az unió azon sorokat tartalmazza, amelyek vagy az R vagy az S relációban bent vannak, de azokat, amelyek mindkettőben szerepelnek, csak egyszer (l. 7.8a ábra).

Az eredményrelációnak mind az oszlopai, mind az elsődleges kulcsa megegyezik az operandus relációk (közös) oszlopaival illetve kulcsával.

Az R és S relációk csak akkor hozhatók uniokompatibilis formába az átnevezés relációs művelettel, ha

- fokszámuk megegyezik és
- a páronként azonos névre átnevezhető oszlopok azonos tartományra vagy annak részére hivatkoznak.

Amennyiben ezek a feltételek nem teljesülnek, de mégis szükségünk van a két reláció valamilyen részleges formában megvalósítható uniójára, úgy az R és S relációból projekcióval és/vagy bővítéssel olyan R' , S' relációkat állítunk elő, melyekre az uniokompatibilitás már érvényesül. Természetesen azt, hogy az így kapott esetleg szűkebb relációkra van-e még értelme az uniónak, az adott feladat határozza meg.



7.8 ábra Két-operandusos relációs műveletek

- a) unió: A két reláció sorait egymás mögé írjuk, és az azonosakból az egyiket töröljük
- b) metszet: A két reláció sorait egymás mögé írjuk és csak azokból a sorokból hagyunk meg egyet, melyekből két egyforma volt.
- c) különbség: Az első reláció soraiból kihúzzuk azokat, amelyek a második relációban is bent vannak. Az R és S relációt uniókompatibilis formába hoztuk, S oszlopait a művelet előtt átneveztük.

7.6.6 Metszet (INTERSECT) és különbség (DIFFERENCE)

Mindkét művelet csak uniókompatibilis relációkra valósítható meg. Az eredmény-reláció oszlopaira és elsődleges kulcsára is ugyanaz érvényes, mint az uniónál.

Az $R \text{ INTERSECT } S$ metszet reláció R és S -nek azokat és csak azokat a sorait (tuple) tartalmazza, amelyek *mindkét* relációban benne vannak (7.8b ábra).

Az $R-S$ vagy $R \text{ DIFFERENCE } S$ reláció sorait R -nek csak azok a sorai alkotják, melyek *nincsenek bent* S -ben (7.8c ábra). Ellentétben az eddig ismertetett két-operandusos műveletekkel, a különbség operandusai nem felcserélhetőek, $R-S \neq S-R$.

Szemléletesebben tudnánk érzékeltetni az unió, a metszet és a különbség műveletét, ha az ábrázolásban eltekinthetnénk attól, hogy a relációban a sorok sorrendjének nincs szerepe és az R és S reláció egyező sorait az ábrázolásban R végére, illetve S elejére tennénk (7.9 ábra).

Könnyen belátható, hogy ha a korlátozás műveletében csak egyszerű műveletet engedünk meg (l. 7.6.2 pont), akkor a „nem létező”

$R \text{ RESTRICT WHERE } (r_1:F_1 \text{ OR } r_2:F_2)$

előállítható az alábbi relációs műveletek eredményeképpen

$(R \text{ RESTRICT WHERE } r_1:F_1) \text{ UNION } (R \text{ RESTRICT WHERE } r_2:F_2),$

hiszen mind a két „eredmény” oszlopai és elsődleges kulcsa azonos R -ével, R -nek mindazokat a sorait tartalmazza pontosan egyszer, amelyekre vagy az $r_1:F_1$ vagy az $r_2:F_2$ összefüggés teljesül.

Hasonlóképpen

$R \text{ RESTRICT WHERE } (r_1:F_1 \text{ AND } r_2:F_2)$

azonos az

$(R \text{ RESTRICT WHERE } r_1:F_1) \text{ INTERSECT } (R \text{ RESTRICT WHERE } r_2:F_2),$

és

$R \text{ RESTRICT WHERE NOT } r_1:F_1$

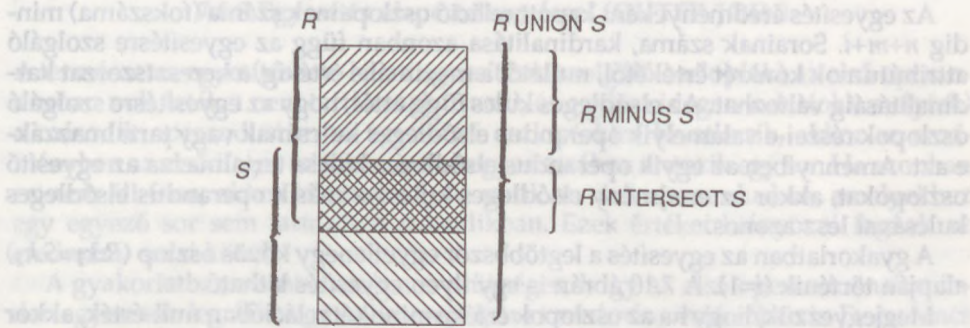
egyenértékű az

$R \text{ DIFFERENCE } (R \text{ RESTRICT WHERE } r_1:F_1)$

műveletekkel.

7.6.7 Egyesítés (JOIN)

Adatbázisokkal kapcsolatos munkákban a legfontosabb és a leggyakrabban használt két-operandusos relációs művelet az egyesítés vagy összekapcsolás (JOIN). Bár, mint azt látni fogjuk, valójában nem elemi művelet, fontossága miatt célszerű önálló műveletként definiálni, melyet az adatbázis-kezelő rendszer azonnal értelmezni tud. Ennek az utasításnak a segítségével a felhasználó sokkal könnyebben meg tudja fogalmazni kívánalmait, mint hogyha egy JOIN eredményét elemi utasításokból, lépésenként kellene összeállítania.



7.9 ábra Unió, metszet és különbség ábrázolása „rendezett relációkkal”.
 R és S azonos sorai a rendezés következtében fedik egymást.

Az R reláció oszlopai legyenek $r_1, r_2, \dots, r_n$, $k_1, k_2, \dots, k_i$, az S -é pedig $k_1, k_2, \dots, k_i$, $s_1, s_2, \dots, s_m$, ahol a k_j oszlopok mindkét relációban ugyanarra az attribútumra vonatkoznak, és nevük páronként azonos, míg az r_j és s_l oszlopok között nincs azonos nevű. (Amennyiben ez nem állna fenn az oszlopnevekre, akkor az egyesítés előtt az átnevezési és/vagy a metszet műveletet kell alkalmazni az operandus(ok)ra.) R és S természetes egyesítése (natural join) az a reláció, amelyiknek oszlopai $r_1, r_2, \dots, r_n$, $k_1, k_2, \dots, k_i$, $s_1, s_2, \dots, s_m$, sorait pedig úgy kapjuk, hogy R -nek minden egyes sorához hozzáírjuk S -nek minden olyan sorát, amelyben a $k_1, k_2, \dots, k_i$ oszlopok értékei mindkét relációban megegyeznek (l. 7.10a ábra). Ha a két relációnak nincsenek közös oszlopai, akkor a természetes egyesítés azonos a keresztszorzattal. Különben a keresztszorzatból állíthatjuk elő az alábbi műveletsorozattal:

$T1 = R \text{ TIMES } S$

$T2 = T1 \text{ RESTRICT WHERE } (R.k_1 = S.k_1 \text{ AND } R.k_2 = S.k_2 \text{ AND } \dots R.k_i = S.k_i)$

$J = T2 \text{ PROJECT } [r_1, \dots, r_n, R.k_1, R.k_2, \dots, R.k_i, s_1, s_2, \dots, s_m]$,

ahol $T1$, $T2$ közbenső eredményként kapott relációk, J pedig a végeredmény.¹⁴ A k_j oszlopokat a műveletben minősítenünk kell, mert nevük nem egyértelmű. Mivel azonban értékük mindkét relációban azonos, mindegy, hogy az S vagy az R relációból vesszük, melyikkel minősítjük őket.

Ehelyett a hosszadalmas leírás helyett használhatjuk az

$R \text{ JOIN } S (k_1, k_2, \dots, k_i)$

jelölést az egyesítés műveletére.

A természetes egyesítést (natural join) az általános szóhasználatban csak egyesítésnek (join) nevezik. A továbbiakban, amikor jelző nélkül használjuk az egyesítés (join) kifejezést, mindig a természetes egyesítést fogjuk ezen érteni.

¹⁴ Az egyesítést közbülső relációk nélkül, egy műveletsorként is felírhattuk volna, de így szétbontva áttekinthetőbbek az egyes lépések.

Az egyesítés eredményeként kapott reláció oszlopainak száma (fokszáma) mindig $n+m+i$. Sorainak száma, kardinalitása azonban függ az egyesítésre szolgáló attribútumok konkrét értékétől, nullától a maximális értékig, a keresztszorzat kardinalitásáig változhat. Az elsődleges kulcs függ attól, hogy az egyesítésre szolgáló oszlopok részei-e valamelyik operandus elsődleges kulcsának vagy tartalmazzák-e azt. Amennyiben az egyik operandus elsődleges kulcsa tartalmazza az egyesítő oszlopokat, akkor az eredmény elsődleges kulcsa a másik operandus elsődleges kulcsával lesz azonos.

A gyakorlatban az egyesítés a legtöbbször egyetlenegy közös oszlop ($R.k_1=S.k_1$) alapján történik ($i=1$). A 7.10 ábrán is egy ilyen egyesítés látható.

Megjegyezzük, hogy ha az oszlopok értéke mindkét relációban null-érték, akkor ezek nem azonosak. A join oszlop(ok)ban null-értéket tartalmazó sorok nem szerepelnek az eredmény relációban.

R			S		
r_1	r_2	k_1	k_1	s_1	s_2
a1	a2	x	x	f1	f2
a1	b1	x	z	f1	f2
c1	c2	y	a	g1	g2
d1	d2	z	x	g1	g2
x1	x2	z			
x1	x2	y			

R JOIN S					
r_1	r_2	k_1	s_1	s_2	
a1	a2	x	f1	f2	
a1	a2	x	g1	g2	
a1	b1	x	f1	f2	
a1	b1	x	g1	g2	
d1	d2	z	f1	f2	
x1	x2	z	f1	f2	

R OUTER JOIN S					
r_1	r_2	k_1	s_1	s_2	
a1	a2	x	f1	f2	
a1	a2	x	g1	g2	
a1	b1	x	f1	f2	
a1	b1	x	g1	g2	
c1	c2	y	0	0	
d1	d2	z	f1	f2	
x1	x2	z	f1	f2	
x1	x2	y	0	0	

a)
b)

7.10 ábra Egyesítés (JOIN) művelete.

Az egyesítés alapjául a mindkét relációban közös k_1 oszlop szolgál

a) természetes egyesítés (natural join)

b) egyesítés alapértelmezéssel (külső, outer join)

Az s_1 és s_2 oszlopok értékének alapértelmezése: 0

7.6.8 Egyesítés alapértelmezéssel (OUTER JOIN)

A természetes egyesítésben, ha az egyesítésre szolgáló oszlop(ok) az első operandusban null értéket nem tartalmazhat(nak), és egyben idegen kulcs(ok), mely(ek) a második operandusban az egyesítési oszlop(ok)ra hivatkozik, akkor az eredményben az elsődleges reláció minden egyes sorához tartozik egy sor. Ha azonban ez nem áll fenn, akkor lehetnek olyan sorok az első operandusban, amelyekhez egy egyező sor sem tartozik a másodikban. Ezek értékei hiányozni fognak az eredmény sorai közül.

A gyakorlatban azonban sokszor szükséges, hogy az első operandusnak minden egyes sora legalább egyszer szerepeljen az eredmény-relációban is. De ha nincs egyezés az egyesítő oszlopban, nem tudjuk, hogy a második operandus melyik sorával egészítsük ki az első operandus nem csatlakoztatható sorait. Az alapértelmezéssel történő egyesítés (OUTER JOIN) úgy oldja meg ezt a problémát, hogy ezekben az esetekben—jobb híján—a második operandus mindegyik oszlopának alapértelmezését írja be az eredmény relációba (l. 7.10b ábra).

Ilyen módon az alapértelmezéses egyesítés eredményének c_{oj} kardinalitása

$$c_r \leq c_{oj} \leq c_r \cdot c_s,$$

ahol c_r az első és c_s a második operandus kardinalitása.

A külső (outer) join nem alapművelet. Előállítható természetes egyesítések uniójaként. Az operandusok sem cserélhetők fel, az első operandusnak kitüntetett szerepe van.

7.6.9 Bővítés (EXTEND)

Létező relációból új relációt képezhetünk azáltal, hogy új oszlopot adunk hozzá. Általános formája

EXTEND R TO R' ADD oszlopképzés módja AS r_{uj} ,

ahol r_{uj} az R' új reláció új oszlopa. (Definiálhatnánk a műveletet úgy is, hogy a kibővített reláció neve ugyanaz maradjon, és ezzel egyidejűleg az eredeti reláció megszűnik. Fizikai táblázatoknál az ALTER TABLE utasítást (l. 10.6 pont) így is értelmezik.

Az új oszlopot gyakran a reláció meglévő oszlopaiból képezzük aritmetikai műveletekkel, pl. SZÁZALÉK-ÉRTÉK=ÉRTÉK/ALAP*100. Ilyen formában R' csak virtuális reláció (nézet, lekérdezés eredménye) lehet, mert az adatbázis-kezelő rendszer nem rendelkezik olyan automatizmussal, amely biztosítaná ténylegesen tárolt adatoknál ugyanazon sor oszlopai között ilyen összefüggések betartását.

Kényelmesebbé teszi a gyakorlati munkát, ha módunk van egy fizikai táblázatnak teljesen új, az eddigi adatoktól független oszloppal való bővítésére anélkül, hogy a meglévő adatokat át kellene vinnünk az új, bővített táblázatba, majd a régi táblázatot törölve az újat átnevezzük a régi nevére.

Ez az előbbi bővítés speciális formája, amikor $R=R'$ és az oszlopképzés módja az oszloptartomány definiációját adja csak meg.

Természetesen az új oszlopnak alapértelmezést is kell adnunk (vagy meg kell engednünk benne null-értéket), és a táblázat már eddig is meglévő soraiban ez lesz az új oszlop értéke.

7.6.10 Csoportfüggvény képzés (SUMMARIZE)

A gyakorlati munkában igen sokszor előfordul, hogy a kiválasztott adatok egy csoportjára akarunk egy, az egész csoportra jellemző értéket megadni.

Ilyen lehet például egy személyi nyilvántartásban a dolgozók fizetésének összege, átlaga, maximális vagy minimális értéke, vagy ugyanezen értékek a dolgozók egyes csoportjaira, például az azonos osztályokon vagy azonos munkakörben dolgozókra vonatkoztatva.

Ez az összetett művelet a bővítés általánosítása, ahol az új táblázat sorai az eredeti táblázat soraiból csak a csoportra jellemző értékeket, illetve a képzett értékeket (összeg, minimum stb.) tartalmazzák.

Formája:

SUMMARIZE R TO R' GROUP BY csoportosítás módja ADD $f_1(r_1)$ AS r_1

Itt $f_1(r_1)$ a csoportra jellemző érték, pl. az azonos csoportba tartozó sorokban az r_1 oszlopértékek összege $SUM(r_1)$, vagy minimuma, $MIN(r_1)$.

Ha a csoportosítás módját elhagyjuk, akkor az aggregátumot az egész relációra vonatkoztatjuk. Ekkor R' -nek csak egyetlenegy sora lesz, különben annyi, ahány különböző csoportot tudunk belőle képezni.

Hasonlóan a többi relációs művelethez, ezt is definiálhatjuk úgy, hogy egy műveletben több csoportképzést is megengedünk, pl. az új reláció tartalmazza osztályonként és ezen belül beosztásonként (csoportképzési szempontok) az egyes csoportok átlagos, maximális és minimális fizetését, valamint az adott csoportban levő egyének számát is (COUNT).

Megjegyezzük, hogy a csoportosítás műveletét csak bizonyos korlátozással tekinthetjük igazi relációs műveletnek. A legtöbb megvalósításnál, ha ezt alkalmazzuk egy relációra, akkor ennek eredménye nem biztos, hogy egy újabb reláció lesz (azaz nem biztos, hogy teljesíteni fogja a 7.2 pontban meghatározott feltételeket). Így használatát a gyakorlatban általában korlátozzák. Nem használható például bizonyos nézetek definiálására szolgáló műveletekben (l. CREATE VIEW, a 10.3.1 pontban). A csoportképzés további lehetőségeivel a III. részben, az oszlopfüggvényeknél (11.2.5 pont) és a SELECT utasítás GROUP BY klauzulájánál (11.2.2 pont) foglalkozunk.

7.6.11 Kijelölés

Célszerű eddigi műveleteinket kiegészíteni még egy továbbiával. Ennek segítségével az ismertett műveletek által létrehozott relációnak egy nevet tudunk adni. Ezáltal egy új, általában virtuális objektumot hozunk létre az adatbázisban, amelylyel a továbbiakban már egyszerűen dolgozhatunk, függetlenül attól, hogy előállítás a valós adatokból milyen bonyolultan, hány lépésben történik. Ezen utasítás hasznosságát a nézetekkel való munkánál fogjuk igazán értékelni (l. 10.3 pont).

7.7 Példák relációs műveletekre

A könnyebb megérthetőség kedvéért a fenti műveletekre néhány példát is bemutatunk. Ezekben a 7.1, 7.2, 7.3 ábrákon látható GÉPJÁRMŰ-NYILVÁNTARTÁS adatbázisból a GÉPJÁRMŰ, TULAJDONOS és G&T táblázatok adatait használjuk fel.

A megfogalmazási mód természetesen nem kötődik konkrét adatbázis kezelő nyelvhez, sőt a III. részben ismertetendő, ma már szinte szabványként tekinthető SQL nyelvben általában még ennyire sem kell részleteznünk a feladatot. A felhasználó által alkalmazható nyelv sokkal inkább megfelel a relációs kalkulus formalizmusának, mint a relációs algebraénak, de ez természetesen a legkülső réteg.

7.7.1 Példa korlátozásra és vetületre

Adjuk meg azoknak a személyeknek a nevét, akik 1992. december 31. után adtak el vagy vásároltak gépkocsit.

Minden adat rendelkezésünkre áll a G&T relációban. Ennek kell a metszetét és megfelelő feltételekkel a korlátozását elvégezni. A két művelet sorrendje tetszőleges.

((RESTRICT G&T WHERE (TULAJDONJOG-KEZDETE > 921231 OR
TULAJDONJOG-VÉGE > 921231)) PROJECT [TULAJDONOS])

Az eredmény a 7.11 ábrán látható.

((RESTRICT G&T WHERE (TULAJDONJOG-KEZDETE > 921231 OR
TULAJDONJOG-VÉGE > 921231)) PROJECT [TULAJDONOS])

G&T

RENDSZÁM	TULAJDONOS	TULAJDONJOG-KEZDETE	TULAJDONJOG-VÉGE
CYX 462	Quittner Pál	92 07 24	
DY 9680	Kiss János 1	92 10 01	
DY 9680	Lakinger Béla	89 11 07	92 10 01
ABC 017	Kiss János 2	93 01 17	
V 24034	Lakinger Béla	92 06 16	
ABC 017	Computer Kft.	91 03 12	93 01 17

7.11 ábra Az 1992. december 31 után vásárolt vagy eladott gépkocsik tulajdonosainak nevei. (G&T táblázatból, l. 7.1 ábra) Mivel a dátumot év-hónap-nap formában számként tároltuk, az összehasonlításban 921231-et kell írunk.

7.7.2 Példa egyesítésre, korlátozásra és vetületre

Adjuk meg, hogy az egyes személyeknek milyen gyártmányú és típusú gépkocsik vannak a tulajdonában.

Az adatokat a G&T és a GÉPJÁRMŰ táblázatokból kell összeválogatnunk, mivel a kívánt adatok közül az első csak a tulajdonos nevét tartalmazza, a második pedig csak a gyártmányt és a típust. A két különböző táblázat között a kapcsolatot az azonos rendszám (egyesítési oszlop) hozza létre.

Az aktuális tulajdonosnál a G&T táblázatban a TULAJDONJOG-VÉGE értéke ismeretlen, ezért ezeket a null-érték korlátozó feltétellel kaphatjuk meg.

```
(RESTRICT G&T WHERE TULAJDONJOG-VÉGE IS NULL)
JOIN GÉPJÁRMŰ (RENSZÁM)
PROJECT [TULAJDONOS,GYÁRTMÁNY,TÍPUS]
```

A művelet eredményét a 7.12 ábrán láthatjuk.

```
(G&T JOIN (RESTRICT GÉPJÁRMŰ WHERE TULAJDONJOG-VÉGE IS NULL)
(RENSZÁM)) PROJECT [TULAJDONOS, GYÁRTMÁNY, TÍPUS]
```

G&T		GÉPJÁRMŰ		
TULAJDONOS	RENSZÁM	RENSZÁM	GYÁRTMÁNY	TÍPUS
Quittner Pál	CYX 462	CYX 462	Mitsubishi	Colt
Kiss János 1	DY 9680	DY 9680	Ford	Sierra
Kiss János 2	ABC 017	ABC 017	Daihatsu	Charade
Lakinger Béla	V 24034	V 24034	Ford	Escort

7.12 ábra Az egyes személyek tulajdonában levő gépkocsik gyártmánya és típusa (GÉPJÁRMŰ és G&T táblázatokból, l. 7.1 és 7.2 ábra)

Hasonlóan az előző példához, az egyes műveletek sorrendje itt is tetszőleges. Ezért ugyanerre az eredményre jutunk például az alábbi műveletssorral is:

```
RESTRICT(G&T JOIN GÉPJÁRMŰ (RENSZÁM))
WHERE TULAJDONJOG-VÉGE IS NULL
PROJECT [TULAJDONOS,GYÁRTMÁNY,TÍPUS]
```

Ennek végrehajtása valószínűleg jóval hosszabb lesz, mivel az egyesítésnél a G&T táblázat összes sorát figyelembe kell vennünk.

A fenti problémák elkerülésére, hogy a felhasználónak—legalábbis első megközelítésben—ne kelljen a feladat megfogalmazásánál a hatékony megvalósíthatóság módjával is törődnie, a relációs adatbázis-kezelő rendszerek nyelve rendszerint nem kéri az elérési procedúra lépésenkénti leírását. A műveleti sorrendet (amennyiben az többféle lehet) a rendszer optimalizálja a konkrét adatokról rendelkezésre álló információk alapján.

SUMMARIZE GÉPJÁRMŰ ADD
GYÁRTMÁNY (GYÁRTMÁNY) AS GYÁRTMÁNY,
MAX (ÚRTARTALOM) AS ÚRTARTALOM-MAX
MIN (ÚRTARTALOM) AS ÚRTARTALOM-MIN

GROUP BY GYÁRTMÁNY

RENDSZÁM	GYÁRTMÁNY	TÍPUS	SZÍN	ÚRTARTALOM	EGYÉB	ÚRTART-MAX	ÚRTART-MIN
CYX 462	Mitsubishi	Colt	fehér	1298	...	1298	1298
DY 9680	Ford	Sierra	ezüsmetál	1996	...	1996	1596
V 24034	Ford	Escort	fehér	1596	...		
ABC 017	Daihatsu	Charade	kék	996	...	996	996
CZ 9404	Volkswagen	Fastback	zöld	1594	...	1594	1594
KF 9547	Volkswagen	Fastback	zöld	1594	...		

7.13 ábra

A különböző gyártmányú gépkocsik maximális és minimális hengerűrtartalma (GÉPJÁRMŰ táblázatból, l. 7.2 ábra)

Ezek az információk a rendszerkatalógusban (l. 7.10 pont) találhatóak. Az elérés optimalizálásáról a 7.8 pontban és az EXPLAIN utasításnál (l. 13.6.1 pont) beszélünk bővebben.

7.7.3 Példa csoportosításra, csoportfüggvényre

Határozzuk meg minden egyes gyártmány maximális és minimális hengerűrtartalmát. Ehhez a GÉPJÁRMŰ táblázat sorait kell csoportosítanunk GYÁRTMÁNY szerint, majd az egyes csoportokból a MAXIMUM illetve MINIMUM csoportképző függvénnyel ki kell választanunk az ŰRTARTALOM oszlop legnagyobb és legkisebb értékeit.

```
SUMMARIZE GÉPJÁRMŰ TO MAX-MIN-ŰRTARTALOM
GROUP BY GYÁRTMÁNY
ADD GYÁRTMÁNY (GYÁRTMÁNY) AS GYÁRTMÁNY,
MAXIMUM (ŰRTARTALOM) AS MAX-ŰRTARTALOM,
MINIMUM (ŰRTARTALOM) AS MIN-ŰRTARTALOM
```

Itt a GYÁRTMÁNY(GYÁRTMÁNY) jelöléssel azt mutatjuk, hogy az új táblázat tartalmazza GYÁRTMÁNY oszlopnévvel minden egyes maximum és minimum érték mellett az ahhoz a csoporthoz tartozó GYÁRTMÁNY oszlop értékét is. A fenti művelet eredményét a 7.13 ábrán láthatjuk.

Ha ezekre az aggregátumokra rendszeresen szükségünk van, akkor célszerű az adatbázisban a MAX-MIN-ŰRTARTALOM relációt egy állandó virtuális relációként (nézet) definiálni és a későbbiekben ezzel közvetlenül is dolgozhatunk.

Ahhoz például, hogy milyen űrtartalom határok között mozognak a 'Ford' gyártmányú gépkocsik, nem szükséges a GÉPJÁRMŰ táblázatra kiadni egy csoportfüggvényképző relációt, elegendő a

```
RESTRICT MAX-MIN-ŰRTARTALOM
WHERE GYÁRTMÁNY = 'Ford'
```

művelet is. (Ezt a nézetre vonatkozó utasítást természetesen a rendszer tovább bontja a fizikai táblázatra vonatkozó csoportfüggvényképző műveletekre.)

Hasonlóképpen, ha a 7.7.1 és 7.7.2 példákban bemutatott relációkra nem csak ad hoc van szükségünk, hanem rendszeresen kívánunk velük dolgozni, akkor az eredményt célszerű állandó relációként, nézetként definiálni.

7.8 Elérési út optimalizálása

A relációs modell alapján működő adatbázis-kezelő rendszerek hatékonyságát sokan vizsgálták már. Az első matematikai számításokat az összevonás hatékonyságára és absztrakt gépen a különböző operátorok hatékonyságára vonatkozóan végezték, azonban ezekben éppen a legjelentősebb tényezőt, a külső tárolóhoz való fordulás költségeit nem vették tekintetbe. A későbbiekben már ezt is figyelembe vették, és a legújabb rendszerek nem a matematikai műveletek optimalizá-

lása, hanem az adatok fizikai tárolási szerkezete alapján határozzák meg a relációs modell adatfüggetlen kérdéséből a tényleges hozzáférési algoritmust.

Tekintsük az alábbi egyszerű relációkat:

ALKATRÉS Z (ALK-SZÁM, SZÁLL-SZÁM, RAKTÁRKÉSZLET, ÁR)
SZÁLLÍTÓ (SZÁLL-SZÁM, NÉV, CÍM, MINŐSÍTÉS),

melyekben az alkatrészek és a szállítók között több-több kapcsolat áll fent, tehát bármely alkatrész beszerezhető több szállítótól is, és bármely szállító szállíthat többféle alkatrészt. A következő kérdés:

„Adjuk meg azon ALK-SZÁM-okat és az ezeket szállítók SZÁLL-SZÁM-át és CÍM-ét, melyeknek ÁR-a 100-nál több, és a szállító MINŐSÍTÉS-e 5-nél kisebb” a relációs modellben algebrailag például a következőképpen fogalmazható meg:

```
[[ALKATRÉS Z RESTRICT WHERE ÁR > 100]
JOIN
[SZÁLLÍTÓ RESTRICT WHERE MINŐSÍTÉS > 5]
(SZÁLL-SZÁM)]
PROJECT
[ALK-SZÁM, SZÁLL-SZÁM, CÍM]
```

(Az áttekinthetőség érdekében különböző zárójelekkel jelöltük a logikailag összetartozó műveleteket.)

Az eredmény tehát egy olyan reláció, mely az ALKATRÉS Z és a SZÁLLÍTÓ relációknak a SZÁLL-SZÁM alapján történő természetes összekapcsolásából, az ÁR és MINŐSÍTÉS tartományokra való korlátozásból és az ALK-SZÁM, SZÁLL-SZÁM, CÍM-re történő projekció révén áll elő.

Mivel a relációs modell és algebra semmiféle megkötést nem tartalmaz az adatok fizikai elérésére, a keresett eredményreláció több úton is előállítható. Ezek közül gyakorlati szempontból az alábbi négyet érdemes megvizsgálni. Az egyszerűbb jelölés és az általánosítás érdekében azonban a fenti konkrét kérdést a következő általános formába írjuk át:

$$R = ([R_1 \text{ RESTRICT WHERE } r_1:F_1] \text{ JOIN } [R_2 \text{ RESTRICT WHERE } r_2:F_2]) \text{ (e)} \\ \text{PROJECT } [p]$$

ahol r_1 , r_2 , e , p a relációk több oszlopát is jelenthetik. Az e egyesítési oszlop(ok) az E tartomány(ok)on van(nak) értelmezve. Eszerint a műveletek eredménye az R_i ($i=1,2$) relációk $r_i:F_i$ feltételekkel korlátozott e oszlopra vonatkoztatott egyesítésének lesz a p oszlop(ok)ra történő projekciója.

Tételezzük fel, hogy R_1 és R_2 fizikailag létező táblázatok, melyeknek sorait egy fizikai rekordból kapjuk meg, azaz egy író/olvasó művelettel mindig behozzuk a lemezzről a teljes sort. Ekkor a projekció elvégzésének ideje a lemezműveletek számát közvetlenül nem befolyásolja. Természetesen a közbülső lépésenként adódó ideiglenes relációkban annál kevesebb fölösleges adatot kell tárolnunk a sorokból, minél hamarabb elvégeztük a projekciót, a közbülső illetve a végső eredményhez szükségtelen oszlopok kiszűrését. Ezért általános szabályként elfogadjuk, hogy azon oszlopokat, melyekre sem a végeredményben, sem a közbülső lépésekben nincsen szükségünk, azonnal elhagyjuk, míg azokat, amelyeket csak

valamilyen közbenső lépésben használunk fel (pl. ezekre történik az egyesítés) csak addig tartunk meg, amíg kellenek.

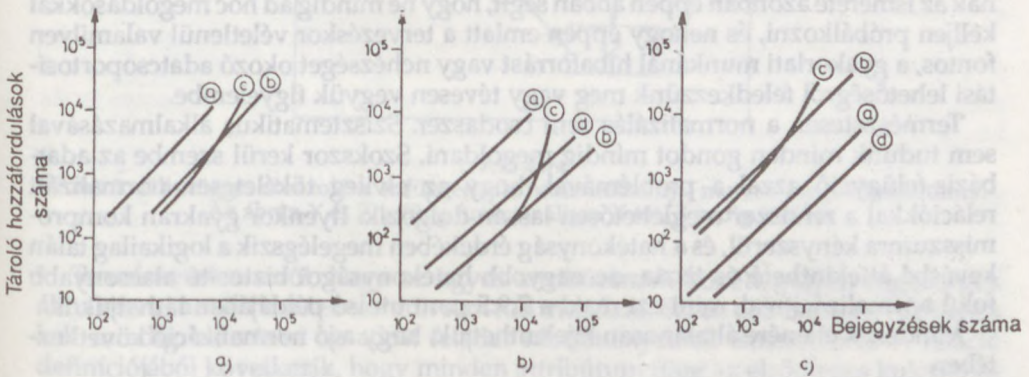
Mivel a relációs műveletek semmiféle kitéltet nem tartalmaznak arra, miként érjük el ténylegesen az adatokat, a fenti feladatot többféle módon is megoldhatja a rendszer.

- a) *Indexvizsgálat az egyesítésre szolgáló tartományokra.* Amennyiben az adattárolási mód olyan, hogy mindkét reláció indexelt az egyesítésre szolgáló tartományokra, úgy végigvizsgáljuk mindkét reláció indexeit. Az indexekből egyenként kiválasztjuk azokat az értékpárokat, melyekre $e_{1,n}=e_{2,m}$, ahol $e_{1,n}$ illetve $e_{2,m}$ R_1 , illetve R_2 valamelyik bejegyzésében az E_1 , illetve E_2 tartomány értéke. Ezután pl. az e_1 indexhez tartozó címek alapján megvizsgáljuk, hogy van-e R_1 -ben egyáltalában olyan bejegyzés, melyre teljesül az $r_1:F_1$ feltétel. Ha nincs egy sem, akkor áttérünk a következő $e_{1,n}=e_{2,m}$ értékpár vizsgálatára. Ha van akkor az első megtalálása után áttérünk az R_2 relációban az e_2 indexhez tartozó címeken levő bejegyzésekre, és egy munkaterületen ideiglenesen tároljuk mindazoknak P -re való projekcióját, melyekre teljesül $r_2:F_2$ feltétel. Ezután ismét visszatérünk R_1 -re. Az e_1 index címein szereplő relációk közül azoknak a P -re való projekcióját, melyekre $r_1:F_1$ teljesül, egyenként egyesítjük (konkatenáljuk) a munkaterületeken ideiglenesen tárolt összes részbejegyzéssel. Az így kapott új bejegyzések rendre részei az eredménynek, ezért átvisszük őket az adatbázis-kezelő rendszer kimenő vagy a felhasználó munkaterületére.
A fenti műveleteket elvégezzük az összes megegyező $e_{1,n}=e_{2,m}$ értékpárra.
- b) *Egyesítés előrendezéssel.* A tárolás által meghatározott úton előbb R_1 -en, majd R_2 -n végighaladva előállítjuk a $W_i (i=1,2)$ segédállományokat. Ezek R_i olyan bejegyzéseinek P_i -re való projekcióját tartalmazzák, melyekre teljesül az $R_i:F_i$ korlátozás. Mindkét segédállományt rendezzük az egyesítésre szolgáló tartomány szerint. Majd a két rendezett állományt a relációs összevonás szabályainak megfelelően az $E_1=E_2$ értékek egyezése alapján egy menetben egyesítjük.
- c) *Többmenetes kiválasztás.* Rendezett segédállományok helyett R_1 -nek a feltételeknek megfelelő projektált bejegyzéseit több menetben behozzuk a tárolóba, ott E_1 szerint rendezzük, és egyenként egyesítjük R_2 megfelelő elemeivel.
- d) *Címegeztetés.* Amennyiben mind az egyesítésre, mind a korlátozásra szolgáló tartományokra van index, akkor az utóbbiak alapján előállítjuk a $W'_i (i=1,2)$ rendezett segédállományokat, melyek a korlátozó feltételeknek eleget tevő bejegyzések címeit tartalmazzák. Az a) módszerhez hasonlóan minden $e_{1,n}=e_{2,m}$ párhoz tartozó címet megvizsgálunk, és ha ugyanaz a cím bent van W'_1 -ben és W'_2 -ben is, akkor a címek által meghatározott bejegyzéseknek vesszük a projekciót, és egyesítjük őket.

Az a) eljárás vitathatatlanul a leghatékonyabb, ha a korlátozásra szolgáló tartományokra a relációk nem indexeltek, és ha az egyesítésre szolgáló tartományokban az egyes relációk bejegyzései az index szerint sűrűsödnek.

Igen előnytelen viszont, ha E_1 és E_2 indexei körül nem sűrűsödnek, de valamely másik S_1 , ill. S_2 indexelt tartomány körül [pl. a fizikai elhelyezés szekvenciálisan vagy indexszekvenciálisan történik S_i ($i=1,2$ szerint), és az R_i relációk összes bejegyzéseihez hozzáférhetünk S_i szerint. Ebben az esetben a b) vagy c) eljárás sokkal előnyösebb, A c) eljárás főleg akkor hatékony, ha a munkaterület elég nagy ahhoz, hogy R_1 -ből a feltételeknek elegendő összes bejegyzés projekciója beleérjen egy menetben, azaz, ha a relációk nem túl nagyok vagy ha a korlátozó feltételek igen szigorúak. Ha viszont mind az egyesítésre, mind a korlátozásokra szolgáló tartományokra indexeltek a relációk, akkor a d) módszert célszerű alkalmazni, különösen, ha a relációk sűrűsödnek az egyesítendő tartomány indexei szerint.

A 7.14 ábrán konkrét numerikus eredményeket láthatunk a fent ismertetett algoritmusok hatékonyságára: miként függ a külső tárolóhoz való fordulások száma a relációk méretétől, a kiválasztástól és a fizikai adatszerkezettől. A számításoknál feltételezték, hogy a kiválasztási feltétel a sorok 10%-át hagyja meg.



7.14 ábra Különböző elérési utak hatékonysága

Két reláció egyesítéséből választunk ki korlátozással sorokat. A korlátozások mindkét relációban a sorok 10 %-ra teljesülnek. a) Az egyesítésre szolgáló tartományra mindkét reláció indexelt, de nem sűrűsödő b) Mind az egyesítési, mind a korlátozó tartományokra indexeltek a relációk, de egyik szerint sem sűrűsödnek. Mindkét reláció sűrűsödik azonban egy másik tartományra. c) Mind az egyesítési, mind a korlátozó tartományra indexeltek a relációk és az egyesítési tartományra mindkét relációban sűrűsödnek is. d) Címegyeztetés, amikor mind az egyesítési, mind a korlátozó tartományra indexeltek a relációk.

A görbék mellé írt jelzés a szövegben ismertetett elérési útra utal.

Bár a relációs modellben, éppen a nagyfokú adatfüggetlenség következtében a felhasználónak nincsen lehetősége a konkrét adatelérési út meghatározására, azt, hogy az végül hogyan valósul meg, a jobb rendszerekben nyomon követheti és becslést is kaphat a hozzáfordulás erőforrás igényére, gyakorlatilag az átlagosan várható elérési időre is (l. az EXPLAIN utasítást a 13.6 pontban). Ennek alapján nagy adatbázisokban összetett feladatok megoldásánál a kérdés átfogalmazásával esetleg hatékonyabbá lehet tenni az aktuális adatokhoz való hozzáférést.

7.9 Normalizáció

7.9.1 A normalizáció fogalma

Az adatbázis-tervezés első lépése az, hogy összegyűjtjük, melyek azok az adatok, amelyekre a különböző feldolgozásoknak szükségük van (vagy a jövőben szükségük lehet). A következő lépés az, hogy az így összeállt „káoszt” megpróbáljuk áttekinthető, jól kezelhető egységekbe szervezni. Ez a két lépés adatmodelltől függetlenül az adatbázis-szervezés alfája és omegája. A relációs modell alkalmazásának egyik nagy előnye az, hogy az adatok ésszerű, a gyakorlatban jól használható csoportosításához, a relációk kialakításához elméletileg megalapozott, formálisan is algoritmizálható (így később, ötödik generációs szoftvereknél már gépileg automatikusan generálható) eljárást ad. Ezt az eljárást nevezzük *normalizációnak*.

A normalizáció mindig az adatok logikai szintjén történik. Általában több lépésből áll, melyek jelentős részét gyakorlott adatbázis-tervező minden elmélet nélkül, kizárólag a józan észre hallgatva is elvégezné. A normalizációs szabályoknak az ismerete azonban éppen abban segít, hogy ne mindig ad hoc megoldásokkal kelljen próbálkozni, és nehogy éppen emiatt a tervezéskor véletlenül valamilyen fontos, a gyakorlati munkánál hibaforrást vagy nehézséget okozó adatszoportosítási lehetőségről feledkezzünk meg vagy tévesen vegyük figyelembe.

Természetesen a normalizálás sem csodaszer. Szisztematikus alkalmazásával sem tudunk minden gondot mindig megoldani. Szokszor kerül szembe az adatbázis-felügyelő azzal a problémával, hogy az elvileg tökéletesen normalizált relációkkal a rendszer meglehetősen lassan dolgozik. Ilyenkor gyakran kompromisszumra kényszerül, és a hatékonyság érdekében megelégszik a logikailag talán kevésbé áttekinthető és tiszta, de nagyobb hatékonyságot biztosító alacsonyabb fokú normalizációval, mint ezt majd a 7.9.5 pont utolsó példájában láthatjuk is.

Mindezek ellenére általánosan kijelenthetjük, hogy a jó normalizáció következtében

- az adatok tárolóigénye kisebb lesz,
- az elemi adatokat gyorsabban és kevesebb hibalehetőséggel változtathatjuk meg,
- az adatbázis logikailag áttekinthetőbb lesz.

A könnyebb áttekinthetőség kedvéért a normalizáció folyamatát és problémáit egy példán mutatjuk be.

Mielőtt azonban erre rátérnénk, egy új fogalmat, a *függőség* fogalmát kell bevezetnünk. A normalizációs elmélet és eljárás lényegében ezen alapszik.

7.9.2 Függőségek

Bármely relációban egyes attribútumok értékei függnék más attribútumok értékeitől.

Ha az R relációban az egyik attribútum, (X) , a *független változó* értéke egyértelműen meghatározza egy másik attribútum (Y) , a *függő változó* értékét, akkor azt mondjuk, hogy Y *funkcionálisan függ* X -től az R relációban. Természetesen ez az

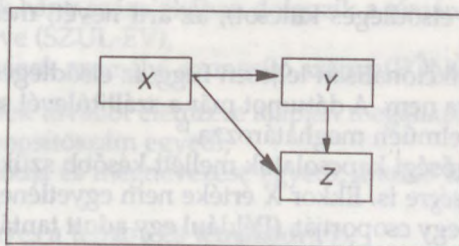
egyértelmű kapcsolat nemcsak R éppen aktuális tartalmára érvényes, hanem időtől független, az egész adatbázis létezésének időtartamára fennálló megszorítás.

Mind az X , mind az Y attribútum lehet összetett, azaz állhat több oszlopból is. A funkcionális függőség szokásos jelölése

$$R.X \rightarrow R.Y.$$

Lehet ezt még funkcionális diagrammban, másnével függőségi ábrában is ábrázolni (7.15 ábra).

R



7.15 ábra

Funkcionális függés ábrázolása. A független attribútumból nyíl mutat a függő attribútumra. Az ábrán Y és Z is függ funkcionálisan X -től, Z pedig Y -től is.

Természetesen arról, hogy az egyes attribútumok között milyen függőségek állnak fenn, az elvi modell semmit sem tud mondani. Ezt az adatbázis tervezőjének kell mindig kideríteni az adott feladat körülményeinek ismeretében. A reláció definíciójából következik, hogy minden attribútum függ az elsődleges kulcstól és nyilvánvalóan—ha van—akkor az összes alternatív kulcstól is.

A gyakorlatban szemléletesebb a funkcionális függésnek az előzőekben megadottal egyenértékű definíciója.

Egy adott R relációban az Y oszlop akkor és csak akkor függ X -től, ha bármely két olyan sorában, amelyekben X értéke azonos, az Y értékek is mindig azonosak.

Fordítva természetesen az állítás nem igaz. Azonos Y értékekhez tartozhatnak különböző X -ek. Például a 7.2 ábra GÉPJÁRMŰ táblázatában (GYÁRTMÁNY, TÍPUS) $\rightarrow$ ŰRTARTALOM, de ugyanazon ŰRTARTALOM értékkel több, elvileg akárhány GYÁRTMÁNY, TÍPUS és (GYÁRTMÁNY, TÍPUS) kombinációjú gépkocsi rendelkezhet.

A funkcionális függőség egy 1:N kapcsolatot jelent a reláción belül. Ennek speciális esete az 1:1 kapcsolat, amikor az R relációban nem csak az X oszlop értéke határozza meg egyértelműen Y értékét, hanem fordítva is, Y értéke is meghatározza egyértelműen X -et. Ennek szokásos jelölése

$$R.X \rightarrow R.Y$$

Példa a kölcsönös funkcionális függőségre egy egyértelmű árukód, vagy osztálykód rendszerben az áru, illetve az osztály kódja és a megnevezése közti köl-

csönösen egyértelmű kapcsolat, vagy bármely relációban az elsődleges és az alternatív kulcsok viszonya.

Az előző példában a független attribútum összetett volt, és a függő csak a kettő kombinációjától együttesen függött. Ezt a fajta függőséget *teljes funkcionális függőségnek* nevezzük.

Általánosan megfogalmazva, az R relációban az Y attribútum funkcionálisan akkor és csak akkor függ teljesen az X (összetett) attribútumtól, ha funkcionálisan függ X -től, de nem függ X -nek egyetlenegy valódi összetevőjétől sem.

Ha X nem összetett, akkor a funkcionális és a teljes funkcionális függőség ugyanazt jelenti.

További példa funkcionális, de nem teljes függőségre egy többteteles szállítólevél, amely többek között tartalmazza a szállítólevél számát, az áru kódját (a kettő együttesen alkotja az elsődleges kulcsot), az áru nevét, mennyiségét, árát, és a szállítási dátumot.

Itt a mennyiség funkcionálisan teljesen függ az elsődleges kulcstól, míg a dátum, az áru neve és ára nem. A dátumot már a szállítólevél száma, a másik kettőt az áru kódja is egyértelműen meghatározza.¹⁵

Ezen alapvető függőségi kapcsolatok mellett később szükségünk lesz a többértékű funkcionális függőségre is. Ekkor X értéke nem egyetlenegy Y értéket határoz meg, hanem azoknak egy csoportját. (Például egy adott tantárgyat az összes tanár közül csak néhány meghatározott személy taníthat.)

Általánosan megfogalmazva, az A, B, C attribútumokkal rendelkező R relációban B akkor és csak akkor van több értékű funkcionális függőségben A -tól, ha bármely adott A és C értékhez tartozó lehetséges B értékek csak A értékétől függenek, C -től függetlenek.

(Ha B mindig csak egyetlenegy értéket vehet fel, akkor ez a szokásos funkcionális függéssel egyenértékű.)

A többértékű funkcionális függés jelölése:

$R.X \quad R.Y$

A különböző relációk közötti függőségeket az adatbázis-kezelő rendszer az idegen kulcsok definiálásával automatikusan szavatolni tudja. Ugyanazon reláció oszlopai közti függőségek betartására azonban ilyen automatizmus nem létezik. Ezt vagy programmal kellene mindig ellenőrizni vagy ellentmondás keletkezhet egy reláción belül.

A normalizációval, mint látni fogjuk, ezeket a függőségeket küszöböljük ki. Így a helyesen normalizált adatbázisba nem kerülhetnek ellentmondásban lévő adatok.

¹⁵ Feltételezzük, hogy az áru ára mindenki számára mindig ugyanaz, legfőképpen egyesek ebből kedvezményeket kaphatnak, ami egy másik attribútum lehet. Amennyiben időben változó árakkal dolgozunk, akkor az áru nem az árukódtól függ funkcionálisan, hanem például az (árukód, szállítási dátum) kombinációtól. Ezeket a feltételeket természetesen a konkrét logikai tervezéskor kell észrevenni.

7.9.3 Első normál forma (1NF)

A normalizálás folyamatának megértésére oldjuk meg az alábbi feladatot:

Dolgozói nyilvántartást kell készítenünk a vállalatnál. Minden dolgozónál szükségünk van a következő adatokra (zárójelben az oszlop neve):

- személyi azonosító szám (SZEMSZ),
- vezetéknév (VEZ-NÉV),
- keresztnév (KER-NÉV),
- osztály kódja, ahol dolgozik (OSZT),
- osztály megnevezése (OSZT-NÉV),
- téma kódja, amelyen dolgozik (TKÓD),
- téma megnevezése (TÉMA),
- munkaidejének hány százalékában dolgozik a témán (MEGOSZL),
- születésének éve (SZÜL-ÉV),
- osztálya főnökének személyi azonosító száma (FŐNÖK).

A vállalati feltételek további elemzése alapján megállapítjuk, hogy

- a személyi azonosítószám egyedi,
- az osztályok kódja és megnevezése egyedi, a kettő között egyértelmű kapcsolat áll fenn,
- ugyanez érvényes a témára és témakódra is,
- egy dolgozó csak egy osztályhoz tartozhat, de több témán is dolgozhat,
- a témák és osztályok között nem kell kapcsolatot teremtenünk,
- nem kell automatikusan ellenőriznünk, hogy a témák összességére fordított idő a dolgozó teljes munkaideje-e (100%).

Az adatokat osztályonként gyűjtik be. Így az induló táblázatunk osztályonként egy-egy rekordot, „sort” tartalmaz, melynek első mezője az osztály kódja, második az osztály neve, utána annyszor jönnek az egyes dolgozók fent felsorolt adatai, ahány dolgozó van az osztályon. Végül az utolsó mező az osztály vezetőjének azonosító száma (7.16. ábra).

Ez a „táblázat” nem reláció, mert ismétlődő mezők vannak benne. Az egyes dolgozókat tartalmazó oszlopkombináció (és ezen belül még a témával kapcsolatos adatok) változó számban fordulnak elő a különböző sorokban. Ahhoz, hogy a relációs modellben dolgozni tudjunk velük, előbb át kell alakítanunk relációvá, meg kell szüntetnünk az ismétlődő mezőket. Ezt úgy érjük el, hogy egy olyan új táblázatot hozunk létre, melynek oszlopai ugyanazok, mint az eredeti táblázat oszlopai, de az ismétlődő mezők mindegyikéből egy új sort állítunk elő, melyek egymástól az eredetileg ismétlődő mező (csoport) tartalmában különböznek csak. Az így kapott táblázat már megfelel a relációk feltételeinek, azt mondjuk, hogy a reláció *első normál formában* (1NF) van (7.17 ábra).

Általánosan azt mondjuk, hogy egy táblázat adatai 1NF-ben vannak, azaz relációt alkotnak, ha teljesülnek a 7.2 pontban részletezett feltételek, azaz

- minden oszlop egy és csak egy attribútumot jelent,
- minden sor különbözik,
- az attribútumok sorrendje minden sorban ugyanaz,
- nincsenek ismétlődő mezők,

OSZT OSZT-NÉV SZEMSZ VEZ-NÉV KER-NÉV TKÓD TÉMA MEGOSZL SZÜL-ÉV FŐNÖK

ismétlődik
annyiszor, ahány dolgozó van az osztályon

a)

P01 Pénzügy	11111 Andrásfai Béla B1 Bér 40 O2 Oktatás 60 1934	Bártfai ...
-------------	---	-------------

... Csilla E1 Ellenőrzés 70 1969	91526 Tarnay Gyula O2 Tervezés 100 1932	91526
----------------------------------	---	-------

T01 Titkárság	01526 Mohácsi Győző T2 Tervezés 100 1971	01993 ...
---------------	--	-----------

Tojásos Kristóf U2 Utazások	80 1966	69690
-----------------------------	---------	-------

b)

7.16 ábra Osztályonként ömlesztett adatok

a) Elvi felépítés b) Tényleges adatok

ALKALMAZOTTAK – INF

OSZT	OSZT-NÉV	SZEMSZ	VEZ-NÉV	KER-NÉV	TKÓD	TÉMA MEGOSZL	SZÜL- ÉV	FŐNÖK
P01	Pénzügy	11111	Andrásfai	Béla	B1	Bér 40	1934	91526
P01	Pénzügy	11111	Andrásfai	Béla	O2	Oktatás 60	1934	91526
P01	Pénzügy	11211	Bártfai	Pál	U2	Utazás 100	1937	91526
P01	Pénzügy	12110	Tóth	Csilla	B1	Bér 30	1971	91526
P01	Pénzügy	12110	Tóth	Csilla	E1	Ellenőrzés 70	1971	91526
P01	Pénzügy	91526	Tarnay	Gyula	T2	Tervezés 100	1932	91526
E01	Eladás	99998	Vigéczi	Jenő	K1	Kereskedelem 100	1945	99998
T01	Titkárság	01526	Mohácsi	Győző	T2	Tervezés 100	1971	01492
T01	Titkárság	69690	Kopó	Levente	E1	Ellenőrzés 100	1957	69690
T01	Titkárság	01492	Tojásos	Kristóf	K1	Kereskedelem 20	1966	69690
T01	Titkárság	01492	Tojásos	Kristóf	U2	Utazás 80	1966	69690

elsődleges kulcs
(SZEMSZ, TKÓD)

7.17 ábra

Az ismétlődő mezők kiküszöbölésével az adatokat az ALKALMAZOTTAK-INF INF relációba hoztuk. Az elsődleges kulcs a SZEMSZ,TKÓD oszlop kombináció.

— minden sorhoz tartozik (legalább) egy egyedi kulcs, melytől az összes többi attribútum funkcionálisan függ.¹⁶

Minden relációs adatbázis-kezelő rendszer megköveteli, hogy az adatok legalább első normalizált formában (1NF) legyenek. Bár ezen a normalizáltsági szinten még sok probléma fölmerülhet, ennél magasabb fok már nem szükséges, de mint a későbbiekben látni fogjuk, általában javasolt.

Ahhoz, hogy az 1NF feltételei valóban teljesüljenek a példában, meg kell vizsgálnunk, van-e a soroknak egyedi kulcsa. A személyi azonosítószám önmagában még nem az, mivel egy személy több témán is dolgozhat, de pl. a témakóddal kombinálva már mindig egyedi marad. Így táblázatunk elsődleges kulcsa a személyi azonosító szám, és a témakód összetett kulcs lesz. (Az azonosító, témanév kombináció alternatív kulcs.)

7.9.4 Második normál forma (2NF)

Nem kell hozzá túl nagy intelligencia, hogy lássuk, a 7.17 ábrán levő 1NF táblázat nagyon sok redundanciát tartalmaz. Azonnal szembeötlik, hogy fölösleges minden osztály és minden téma megnevezését az összes sorban külön feltüntetni. De ezeken a nyilvánvaló redundanciákon kívül más, jobban rejtett hibalehetőségek is vannak ebben a táblázatban, melyeket összefoglalóan *változtatási anomáliáknak*, részletezve beírási, módosítási és törlési anomáliáknak nevezünk.

Nézzük meg mi történik, ha Andrásfai Béla vagy Vigécz Jenő kilép, azaz adatait töröljük a táblázatból. Ebben az esetben automatikusan eltűnik belőle az O2 téma is, illetve az E01 Eladás osztály. (Ez egyben azt is jelenti, hogy ekkor nem tudjuk majd, mi az O2 témának, és mi az E01 osztálynak a megnevezése.) Akaratlanul, de töröltünk az adatbázisból olyan információkat, amelyekre pedig továbbra is szükségünk lenne. Ezt hívjuk *törlési anomáliának*.

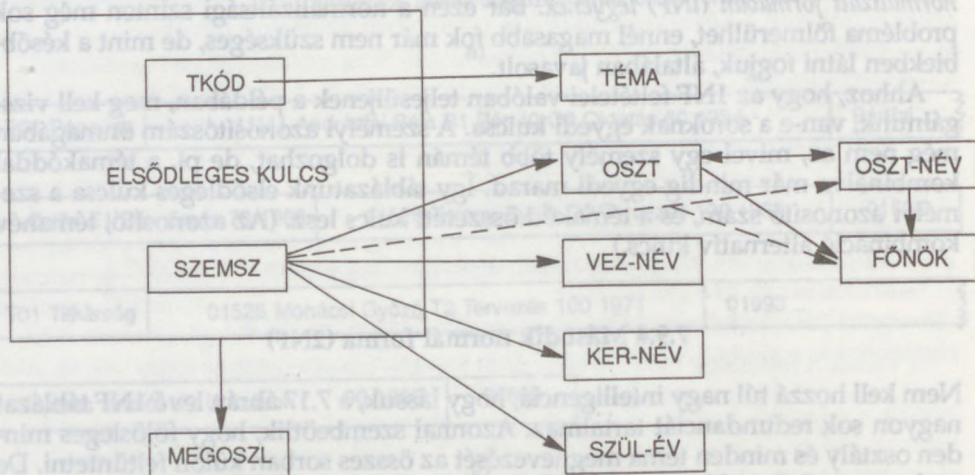
További problémát okozhat, ha Tóth Csilla férjhez megy, vagy pl. a P01 osztály új főnököt kap. Mindkét esetben több sort kell egyidejűleg megváltoztatnunk, hogy adataink konzisztensek maradjanak. Azt a problémát, hogy egy adat megváltoztatásával több helyen kell változtatást eszközölnünk, *módosítási anomáliának* nevezzük. A módosítási anomália is arra vezet, hogy ha a változtatást nem hajtjuk végre konzisztensen, akkor az adatbázisban ellentmondás jöhet létre. (Pl. Bártfai Pálnál a P01 osztályban a főnök kódja 91526 maradna, míg a többinél 11111-re változna.)

Végezetül nézzük meg, milyen problémáink lehetnek új sorok beírásánál, mit jelent a *beírási anomália*.

Ha egy új osztályt hozunk létre a vállalatnál, vagy egy új dolgozót veszünk fel, de még nem tudunk legalább egy témát kijelölni a számára, akkor ezt az információt nem tudjuk bevinni az adatbázisba. Mindkettő olyan sort eredményezne, melyben az elsődleges kulcs (illetve annak egy része) null-érték. Ahhoz, hogy

¹⁶ Igen sok adatbázis-kezelő rendszer formailag nem követeli meg az elsődleges kulcsot. Ennek ellenére a biztonságos munkához ezekben is célszerű definiálni elsődleges kulcsot minden táblázatra.

mindezen problémák gyökerét tisztán lássuk, rajzoljuk fel a 7.17 ábrán levő reláció függőségi diagrammját (7.18 ábra).



7.18 ábra A 7.17 ábrán látható relációval leírt dolgozó nyilvántartás függőségi diagrammja
A $SZEMSZ \rightarrow OSZT-NÉV$ és $SZEMSZ \rightarrow FÖNÖK$ függést, mivel ez az $OSZT$ -on keresztül
úgyis automatikusan megvalósul, az áttekinthetőség kedvéért nem tüntettük fel.

Ebből láthatjuk, hogy a személyi azonosítószám ($SZEMSZ$) és témakód ($TKÓD$) összetett elsődleges kulcstól csak a százalékos megoszlás ($MEGOSZL$) függ funkcionálisan teljesen. Az összes többi attribútum az elsődleges kulcsnak csak egy összetevőjétől függ.

Második normál formában (2NF) akkor és csak akkor van egy reláció, ha 1NF-ben van, és az összes nem kulcs attribútum funkcionálisan teljesen függ az elsődleges kulcstól.

Elemi elsődleges kulcsú 1NF relációk automatikusan 2NF-ben is vannak. Összetett kulcsú relációkat azonban a változtatási anomáliák megszüntetése érdekében 2NF-re kell hoznunk. (Ez nem biztos, hogy megszünteti az összes anomáliát, de jelentősen csökkentheti számukat.) Ezt nevezzük a relációk szétbontásának vagy dekompozíciónak.

A szétbontás úgy történik, hogy az 1NF relációból projekcióval olyan 2NF relációkat állítunk elő, melyeknek elsődleges kulcsai az eredeti reláció elsődleges kulcsa, illetve annak részei, oszlopai pedig azok és csak azok, amelyek az új elsődleges kulcstól funkcionálisan teljesen függenek.

Általánosan megfogalmazva, ha az A, B, C, D oszlopokkal rendelkező relációban A, B kombinációja az elsődleges kulcs, és D funkcionálisan függ A -tól, azaz rövid, szemléletes jelölésben

$R(A, B, C, D)$
ELSÖDLEGES KULCS (A, B)
 $R.A \rightarrow R.D$

akkor a második normál formába történő átalakítás az eredeti R relációnak a következő két új relációval való helyettesítését jelenti:

és

$R_1(A,D)$
ELSŐDLEGES KULCS (A)

$R_2(A,B,C)$
ELSŐDLEGES KULCS (A,B)
IDEGEN KULCS (A), HIVATKOZIK R_1 -RE.

Az idegen kulcs az adatbázis integritása érdekében szükséges. Ezáltal biztosítható, hogy az új R_1 és R_2 relációknak A alapján történő egyesítése, az

$R_1 \text{ JOIN } R_2 (A)$

művelet mindig előállítja az R reláció összes sorát és csak azokat állítja elő.

Amennyiben az összetett kulcsnak nem két, hanem több funkcionálisan teljesen függő felbontási lehetősége van, akkor azok mindegyikére elvégezzük a fent definiált módon a szétbontást.

A példában szereplő relációban a személyi azonosító szám, témakód teljes elsődleges kulcstól csak a százalékos megoszlás függ funkcionálisan teljesen, így dekompozíció után ez lesz az egyik reláció. További két 2NF relációt kapunk, melyeknek elsődleges kulcsa a személyi azonosító szám, illetve a témakód, oszlopai pedig az eredeti 1NF relációnak azon oszlopai, melyek ettől és csak ettől függenek (teljesen) funkcionálisan.

A 2NF-re való szétbontás után az egyes relációk neve, elsődleges és idegen kulcsai

ALKALMAZOTTAK-2NF

ELSŐDLEGES KULCS (SZEMSZ)

MIT CSINÁL?

ELSŐDLEGES KULCS (SZEMSZ,TKÓD)

IDEGEN KULCS (SZEMSZ) HIVATKOZIK ALKALMAZOTTAK-2NF-re

IDEGEN KULCS (TKÓD) HIVATKOZIK TÉMÁK-ra

TÉMÁK

ELSŐDLEGES KULCS (TKÓD)

Az így létrejött 2NF táblázatok oszlopait az 7.19 ábrán láthatjuk.

Ezzel a szétbontással a változtatási anomáliák egy részét kiküszöböltük. Így például Andrásfai Béla kilépésével nem szűnik meg az „Oktatás” téma és az O2 → Oktatás kapcsolat sem (törlési anomália). Tóth Csilla nevét csak egy helyen kell megváltoztatni (módosítási anomália) és új dolgozót is felvehetünk anélkül, hogy témát adnánk neki (beírási anomália). A többi probléma azonban még továbbra is fennmarad. Azokat, mint azt a következő pontban látni fogjuk, a harmadik normál formába való szétbontás küszöböli csak ki.

Ez a szétbontás teljes, azaz az összes olyan információt tartalmazza, amely az eredeti relációban benne volt. De ezen kívül több is van benne, tudniillik az, hogy az O2 témakód megnevezése Oktatás, az E01 osztályé pedig Eladás, míg az eredeti reláció ezeket csak addig tartamazza, amíg van legalább egy olyan dolgozó az adatbázisban, aki ezen a témán, illetve ezen az osztályon dolgozik.

ALKALMAZOTTAK – 2NF

OSZT	OSZT-NÉV	SZEMSZ	VEZ-NÉV	KER-NÉV	SZÜL-ÉV	FŐNÖK
P01	Pénzügy	11111	Andrásfai	Béla	1934	91526
P01	Pénzügy	11211	Bártfai	Pál	1937	91526
P01	Pénzügy	12110	Tóth	Csilla	1971	91526
P01	Pénzügy	91526	Tarnay	Gyula	1932	91526
E01	Eladás	99998	Vigécz	Jenő	1945	99998
T01	Titkárság	01526	Mohácsi	Győző	1971	01492
T01	Titkárság	69690	Kopó	Levente	1957	69690
T01	Titkárság	01492	Tojásos	Kristóf	1966	69690

elsődleges
kulcs
(SZEMSZ)

MIT-CSINAL?

SZEMSZ	TKÓD	MEGOSZL
11111	B1	40
11111	O2	60
11211	U2	100
12110	B1	30
12110	E1	70
91526	T2	100
99998	K1	100
01526	T2	100
69690	E1	100
01492	K1	20
01492	U2	80

elsődleges kulcs
(SZEMSZ, TKÓD)

TÉMÁK

TKÓD	TÉMA
B1	Bér
O2	Oktatás
U2	Utazás
E1	Ellenőrzés
T2	Tervezés
K1	Kereskedelem

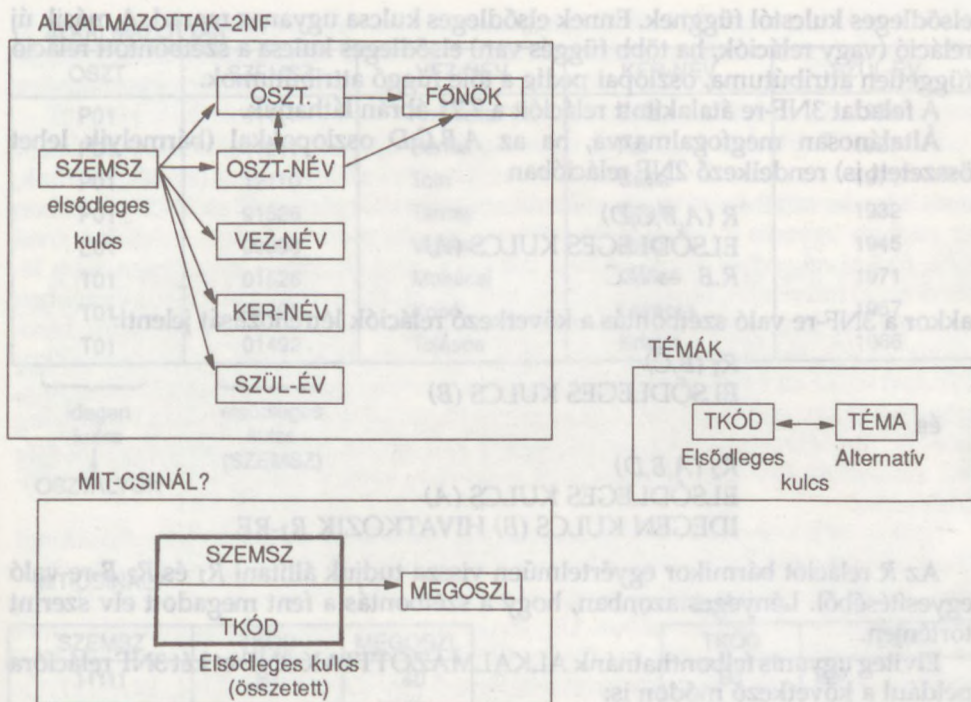
elsődleges kulcs (TKÓD) alternatív kulcs (TÉMA)

7.19 ábra A 7.17 ábrán látható 1NF reláció szétbontása 2NF relációkra.

7.9.5 Harmadik normál forma (3NF)

A 2NF-be hozott relációkban az ALKALMAZOTTAK-2NF-ben nyilvánvalóan redundáns osztálynevek mellett még mindig megmaradtak az alábbi változtatási anomáliák:

— Vigécz Jenő kilépésével (törlésével) megszűnnek az eladási osztályra vonatkozó információink.



7.20 ábra A 7.19 ábrán látható 2NF relációk függőségi diagrammja

- Ha a P01 (vagy a T01) osztálynak új főnöke lesz (módosítás), akkor azt továbbra is több helyen kell végigvezetni.
- Továbbra sem tudunk új osztályt addig fölvenni (beírás), amíg nincsen legalább egy dolgozója.

Ha fölrajzoljuk a 2NF relációk függőségi diagrammját (7.20 ábra), akkor látható, hogy az egyetlen nem elsődleges kulcstól való funkcionális függőség az osztálykódból (osztálynévből) indul ki, és az összes anomália is valamilyen módon ehhez az attribútumhoz kapcsolódik.

Ha új relációk kialakításával ezeket a függőségeket is megszüntetjük, akkor az új relációk mindegyike harmadik normál formában (3NF) lesz. Ekkor, mint látni fogjuk, a még megmaradt változtatási anomáliák is megszűnnek.

Egy reláció akkor és csak akkor van 3NF-ben, ha 2NF-ben van, és az elsődleges kulcshoz nem tartozó attribútumai csak az elsődleges kulcstól függenek funkcionálisan.

Másképpen megfogalmazva a 3NF azt jelenti, hogy funkcionális függés csak az elsődleges és az alternatív kulcsokból indulhat ki.

Az ALKALMAZOTTAK-2NF reláció nincs 3NF-ben, mert pl. az osztály (OSZT) nem elsődleges vagy alternatív kulcs, és más oszlopok (OSZT-NÉV, FŐNÖK) funkcionálisan függenek tőle.

A szétbontás úgy történik, hogy először a 2NF relációnak vesszük azt a projekcióját, amelyik csak azokat az attribútumokat tartalmazza, melyek kizárólag az

elsődleges kulcstól függnnek. Ennek elsődleges kulcsa ugyanaz marad. A másik új reláció (vagy relációk, ha több függés van) elsődleges kulcsa a szétbontott reláció független attribútuma, oszlopai pedig a tőle függő attribútumok.

A feladat 3NF-re átalakított relációit a 7.21 ábrán láthatjuk.

Általánosan megfogalmazva, ha az A, B, C, D oszlopokkal (bármelyik lehet összetett is) rendelkező 2NF relációban

$R(A, B, C, D)$
ELSŐDLEGES KULCS (A)
 $R.B \rightarrow R.C$

akkor a 3NF-re való szétbontás a következő relációk létrehozását jelenti:

$R_1(B, C)$
ELSŐDLEGES KULCS (B)

és

$R_2(A, B, D)$
ELSŐDLEGES KULCS (A)
IDEGEN KULCS (B) HIVATKOZIK R_1 -RE

Az R relációt bármikor egyértelműen vissza tudjuk állítani R_1 és R_2 B -re való egyesítéséből. Lényeges azonban, hogy a szétbontás a fent megadott elv szerint történjen.

Elvileg ugyanis felbonthatnánk ALKALMAZOTTAK-2NF-et két 3NF relációra például a következő módon is:

ALK1-3NF (SZEMSZ, OSZT, VEZ-NÉV, KER-NÉV, SZÜL-ÉV)
ALK2-3NF (SZEMSZ, FŐNÖK)
OSZTÁLY1 (OSZT, OSZT-NÉV),

ahol SZEMSZ mindkét ALK reláció elsődleges kulcsa, OSZTÁLY1 kulcsa pedig OSZT. Az integritást az ALK1-3NF-ben levő OSZT és az ALK2-3NF-ben levő SZEMSZ idegen kulcsok biztosítják.¹⁷

A három 3NF reláció egyesítéséből (ALK1-3NF JOIN ALK2-3NF (SZEMSZ)) JOIN OSZTÁLY1 (OSZT) most is pontosan az ALKALMAZOTTAK-2NF reláció tartalmát kapjuk vissza, de semmiféle automatizmusunk nincs annak biztosítására, hogy ha két dolgozó ugyanazon az osztályon dolgozik, akkor ugyanaz lesz az osztályvezetőjük, főnökük is.

Ha viszont az

ALK3-3NF (SZEMSZ, FŐNÖK, VEZ-NÉV, KER-NÉV, SZÜL-ÉV)
OSZTÁLY3-3NF (OSZT, OSZT-NÉV, FŐNÖK)

felbontást alkalmaznánk és egy személy egyidejűleg több osztályt is vezethet (ami minden további nélkül előfordulhat), akkor a két 3NF reláció FŐNÖK-re való egyesítéséből nemcsak az eredeti ALKALMAZOTT-2NF reláció összes sorát kap-

¹⁷ Amennyiben az OSZT helyett OSZT-NÉV alternatív kulcs lenne az elsődleges kulcs OSZTÁLY1-ben, akkor OSZT-ot nem definiálhatnánk idegen kulcsnak, így az integritás nem maradna meg automatikusan. Ez is példa arra, hogy nem mindegy, hogy a lehetséges kulcsok közül végül melyiket választjuk ki elsődleges kulcsnak.

ALKALMAZOTTAK

OSZT	SZEMSZ	VEZ-NÉV	KER-NÉV	SZÜL-ÉV
P01	11111	Andrásfai	Béla	1934
P01	11211	Bártfai	Pál	1937
P01	12110	Tóth	Csilla	1971
P01	91526	Tarnay	Gyula	1932
E01	99998	Vigécz	Jenő	1945
T01	01526	Mohácsi	Győző	1971
T01	69690	Kopó	Levente	1957
T01	01492	Tojásos	Kristóf	1966

idegen
kulcs
↓
OSZTÁLYOK

elsődleges
kulcs
(SZEMSZ)

MIT-CSINÁL?

TÉMÁK

SZEMSZ	TKÓD	MEGOSZL
11111	B1	40
11111	O2	60
11211	U2	100
12110	B1	30
12110	E1	70
91526	T2	100
99998	K1	100
01526	T2	100
69690	E1	100
01492	K1	20
01492	U2	80

TKÓD	TÉMA
B1	Bér
O2	Oktatás
U2	Utazás
E1	Ellenőrzés
T2	Tervezés
K1	Kereskedelem

elsődleges
kulcs
(TKÓD)

OSZTÁLYOK

OSZT	OSZT-NÉV	FŐNÖK
P01	Pénzügy	91526
E01	Eladás	99998
T01	Titkárság	69690

elsődleges
kulcs
(OSZT)

idegen kulcs
(FŐNÖK)
↓
ALKALMAZOTTAK

7.21 ábra A 7.17 ábrán látható 1NF reláció szétbontása 3NF relációkra.
(Fölhasználva a 7.19 ábrán látható 2NF felbontást.)

nánk meg, hanem további fölösleges, hamis sorokat is. A relációk alapján semmi-fele lehetőségünk nincs ezek kiszűrésére, így ebből az adatbázisból azokról a dolgozókról, akiknek a főnöke több osztályt vezet, nem tudnánk egyértelműen megállapítani, melyik osztályon dolgoznak.

Bár a 3NF relációk logikailag tiszta képet nyújtanak, a gyakorlatban sokszor nem érdemes az adatbázis minden relációját ilyen alakra hozni. (Emlékeztetünk, arra, hogy az adatbázis-kezelő rendszer számára már elegendő az 1NF is!). Ebben az esetben ugyanis a felhasználó számára ténylegesen szükséges adatokat jóval több fizikai rekordból szedi össze a rendszer, ami általában lényegesen több lemezkezelő műveletet, hosszabb elérési időt jelent. Éppen ezért bizonyos esetekben az adatbázis-felügyelő tudatosan nem végzi el ilyen szintig a normalizációt, bizonyos relációk esetében megmarad a 2NF vagy akár az 1NF-nél. Különösen akkor célszerű ez, ha az egymástól függő adatokra rendszeresen együtt van szükségünk, aránylag ritkán változnak meg, és az egyes függő adatok helyfoglalása viszonylag csekély. Természetesen ilyenkor tudatában kell lennie, hogy az esetleges változtatási anomáliák kiküszöböléséről saját magának, vagy a felhasználónak kell gondoskodnia.

Tipikus példa erre egy névjegyzék, mely egyéb adatok mellett az irányítószámot és a pontos címet is tartalmazza. Mivel a pontos cím (ország, megye, város, kerület, utca, házszám) egyértelműen meghatározza az irányítószámot, ez a reláció nincsen harmadik normál formában. Ugyanakkor minden ésszerű alkalmazás a címet az irányítószámmal együtt használja. Ezért fölösleges volna a relációt két 3NF relációra szétbontani és feldolgozást lassítani egy ZIPCODE (CÍM, IRÁNYÍTÓ-SZÁM) relációval való egyesítés műveletével.¹⁸

7.9.6 Normalizálás utáni problémák, magasabb normalizációs formák

A gyakorlatban a legtöbb esetben elegendő az adatokat az előző pontokban ismerttetett algoritmusokkal 3NF, illetve a hatékonyság érdekében tett kompromisszum miatt 2NF vagy 1NF relációkban elrendezni. Az élet azonban igen sokszínű, és előfordulhatnak olyan feltételek, melyeket a jól szétbontott 3NF relációkkal sem tudunk automatikusan biztosítani. Ezek közül most kettőt fogunk bemutatni, melyek közül a második elvezet a még jobban normalizált 4NF relációkhoz. Megjegyezzük, hogy a szakirodalom már ismeri az 5NF relációkat is és lehet, hogy hamarosan még magasabb formák is bevezetésre kerülnek. Ezek jelentősége azonban inkább elméleti. A gyakorlat számára a szisztematikusan létrehozott harmadik normál forma csaknem mindig elegendő, sőt, mint az előző pont példájában láttuk, időnként már túl sok is.

¹⁸ Nem beszélve arról a gyakorlatban egyáltalában nem elhanyagolandó problémáról, hogy ha az adatbázis-felügyelő nem akar minden egyes létező házszámhoz egy külön sort rendelni a ZIPCODE táblázatban, akkor miként oldja meg célszerűen, hogy míg pl. Érden minden utcához ugyanaz az irányítószám tartozik, addig Budapesten egyes utakhoz, mint pl. az Üllői úthoz, attól függően, hogy a házszám páros vagy páratlan vagy egy adott érték alatt, vagy felett van, más és más irányítószám tartozik.

Első példaként tekintsük a HTO relációt, melynek oszlopai a HALLGATÓ, TANTÁRGY és az OKTATÓ. A HTO reláció sorai azt írják le, hogy a hallgatók milyen tantárgyakat, kiktől hallgatnak.

További feltételek, melyek a valós világ megszorításaiból adódnak:

- minden hallgató egy adott tantárgyat csak egy oktatótól hallgat (de sok tantárgyat hallgathat),
- minden oktató csak egy tantárgyat oktat (de egy tárgyat többen is oktathatnak).

A HTO reláció elsődleges kulcsa (HALLGATÓ,OKTATÓ), de nem 3NF, mert az OKTATÓ → TANTÁRGY funkcionális függőség fennáll.

Így, ha ebbe akarnánk egy új sort behelyezni, akkor az adatbázis-kezelő rendszertől függetlenül, külön kellene azt ellenőrizni, hogy az adott oktató valóban tanítja-e a megadott tantárgyat.

Ha ezt most a HTO relációnak az eddig tanultak alapján 3NF-re való szétbontásával akarnánk megoldani, akkor az alábbi relációkra jutnánk:

HO (HALLGATÓ,OKTATÓ)
ELSŐDLEGES KULCS (HALLGATÓ,OKTATÓ)
IDEGEN KULCS(OKTATÓ) HIVATKOZIK OT-RA

és

OT (OKTATÓ,TANTÁRGY)
ELSŐDLEGES KULCS (OKTATÓ).

A két reláció egyesítéséből pontosan HTO-t kapjuk vissza, ennek ellenére a feladatot nem tudtuk tökéletesen megoldani.

Tegyük fel, hogy Kovács János Kiss Pétertől tanul számítástechnikát, de ezt a tárgyat Nagy Endre is oktatja. Semmiféle adatbázis-kezelési automatizmus nem biztosítja azt, hogy HO relációba ne vegyük fel a Kovács János, Nagy Endre sort, holott—mivel a feltételek szerint Nagy Endre csak számítástechnikát tanít—ez ellentmond annak a valós világ modellezésből származó feltételnek, hogy egy hallgató egy tantárgyat csak egy oktatótól hallgat.

Második példának, ahol a 3NF nem vezet megfelelő eredményre, tekintsük a következő feladatot:

Adva vannak

- tantárgyak (melyek közül egy konkrét megtartását kurzusnak nevezzük),
- oktatók,
- könyvek.

További feltételek:

- egy tantárgyat több oktató is taníthat, de egy adott kurzust csak egy tarthat meg,
- egy tantárgyhoz több könyv van előírva és ezek mindegyikét használni kell, bárki is tartja a kurzust,
- minden oktató csak meghatározott tantárgyakat tanít, azaz egy tantárgyat csak bizonyos oktatók tanítanak.

A nem-normalizált adatokat a 7.22 ábrán láthatjuk.

TANTÁRGY-OKTATÓ-KÖNYV

TANTÁRGY	OKTATÓ	KÖNYV
Számítástechnika	{ Kiss Péter } { Nagy Endre }	{ Bevezetés a számítástechnikába } { Számítástechnika rendszerszervezőknek }
Fizika-III.	{ Kiss Péter }	{ Út az atomfizikához } { Radioaktív izotópok alkalmazása } { Fizika III. }
Matematika-I.	{ Millhoffer Hugóné } { Szabó József } { Nagy Endre }	{ Kalkulus }

7.22 ábra Tantárgyak, oktatók, könyvek közti összefüggés nem-normalizált formában

Az adatokat az ismétlődő mezők kiküszöbölésével 1NF-be hozhatjuk. Az így létrejött TOK (TANTÁRGYAK,OKTATÓK,KÖNYV) reláció csak elsődleges kulcsból áll, így egyben 3NF-ben is van. (7.23 ábra) Ennek ellenére ez az elrendezés igen sok redundanciát tartalmaz. (Fölösleges pl. háromszor megadni, hogy Kiss Péter tanítja a Fizika-III-at, vagy kétszer, hogy a Számítástechnikához mik a kötelező könyvek stb.).

TOK

TANTÁRGY	OKTATÓ	KÖNYV
Számítástechnika	Kiss Péter	Bevezetés a számítástechnikába
Számítástechnika	Kiss Péter	Számítástechnika rendszerszervezőknek
Számítástechnika	Nagy Endre	Bevezetés a számítástechnikába
Számítástechnika	Nagy Endre	Számítástechnika rendszerszervezőknek
Fizika-III.	Kiss Péter	Út az atomfizikához
Fizika-III.	Kiss Péter	Radioaktív izotópok alkalmazása
Fizika-III.	Kiss Péter	Fizika III.
Matematika-I.	Millhoffer Hugóné	Kalkulus
Matematika-I.	Szabó József	Kalkulus
Matematika-I.	Nagy Endre	Kalkulus

7.23 ábra A 7.22 ábrán látható adatok elrendezése normalizált relációban.
Az összetett elsődleges kulcs: (TANTÁRGY,OKTATÓ,KÖNYV)

Ráadásul a szokásos változtatási anomáliák is fellépnek. Például egy új Fizika-III oktató belépése három új sor bevitelét jelenti úgy, hogy a KÖNYV csak meghatározott értékeket vehet föl, melyeket azonban nem tudunk integritás feltételekkel automatikusan biztosítani. Hasonlóképpen, egy kötelező könyv törlése vagy módosítása is általában több sort érint.

Mindezek a nehézségek abból adódnak, hogy az elsődleges kulcs egyes oszlopai között többértékű funkcionális függőség áll fenn:

TANTÁRGY	OKTATÓ
TANTÁRGY	KÖNYV

A többértékű funkcionális függőségből adódó problémák kiküszöbölésére vezették be a negyedik normalizált formát (4NF).

Egy R reláció akkor és csak akkor 4NF, ha 3NF és ha van benne több értékű funkcionális függőség, pl. $R.A \twoheadrightarrow R.B$, akkor a reláció összes többi $C, D, \dots$ oszlopa funkcionálisan függ A -tól, azaz $R.A \rightarrow R.C$; $R.A \rightarrow R.D$, ...

A fenti példaként felhozott reláció nyilván nem 4NF, mivel

TANTÁRGY OKTATÓ igaz, de
 $TANTÁRGY \rightarrow KÖNYV$ nem érvényes
 (csak $TANTÁRGY \rightarrow KÖNYV$).

A 4NF-re bontás a többszörös függőségben álló oszlopokat bontja külön relációkra, azaz jelen példánkban

TO (TANTÁRGY, OKTATÓ)
 ELSŐDLEGES KULCS (TANTÁRGY, OKTATÓ)

és

TK (TANTÁRGY, KÖNYV)
 ELSŐDLEGES KULCS (TANTÁRGY, KÖNYV).

Ezzel a szétbontási algoritmussal a redundanciát megszüntettük és az eredeti adatokat minden veszteség nélkül teljesen vissza tudjuk állítani a $TO \Join TK$ ($TANTÁRGY$) művelettel.

Egyetlenegy problémánk azonban maradt, bár ez lehet gyakorlati szempontból előnyös is. Az eredeti TOK relációba csak akkor vehettünk fel egy új tantárgyat, ha legalább egy oktató is és legalább egy könyv is volt már hozzá (ha másképpen nem, akkor null-érték helyett pl. a „NINCS” alapértelmezéssel), míg a 4NF relációkból álló modellbe felvehetünk már akkor is, ha legalább egy oktatót, vagy egy könyvet hozzá tudunk rendelni.

Ha a valós körülmények elemzése többértékű funkcionális függőség fennállására vezet, akkor általában célszerű a relációkat 4NF-re alakítani.

7.9.7 Nézetek definiálása a normalizált relációkból

Mindezideig—egy hatékonyság érdekében tett kivételtől eltekintve—a normalizációnak csak az előnyeit láthattuk. Az átlagos felhasználó számára azonban mindezek az előnyök semmivé zsugorodnának, ha például a 7.21 ábrán látható normalizált relációkból a leggyakoribb kérdést „Egy adott dolgozó milyen nevű osztályon, milyen nevű témákon, és azokon milyen megosztásban dolgozik?” mindig négy táblázat egyesítésének explicit leírásával tudna csak megfogalmazni.

Ebben az esetben a relációs modell használata semmivel sem volna egyszerűbb, mint a hálós adatbázisban való navigálás, azzal súlyosbítva, hogy amíg ott a gyakorlott felhasználó a set-kapcsolatok pontos ismeretében hatékony elérési utakat határozhatott meg, addig itt még erre sincsen lehetőség. Éppen a felhasználók munkájának megkönnyítésére a normalizációt csak a fizikailag is tárolt adatokra, a tényleges táblázatokra végzik el. Ezekből aztán a relációs műveletek segítségével az összes olyan virtuális relációt, nézetet elő lehet állítani, amelyekre a felhasználóknak rendszeresen szüksége lehet. Ezek a nézetek már gyakran csak 1NF relá-

ciók. A felhasználók azonban ezekre a nézetekre hivatkoznak, melyekben az adatok a kívánt módon, látszólag redundánsan, de együtt vannak jelen.

Így például az ezen pont elején megfogalmazott kérdésre definiálhatunk az adatbázisban egy VÁLASZ1 nevű nézetet, mely a 7.22 ábra négy táblázata egyesítésének a projekciójával jött létre és minden dolgozóra az alábbi adatokat tartalmazza:

- személyi azonosító szám,
- vezetéknev,
- keresztnév,
- osztályának megnevezése,
- témájának megnevezése,
- munkaidejének százalékos megoszlása a témán.

Az új, virtuális reláció a már meglévő táblázatokból az alábbi relációs művelet-sorozattal jön létre:

VÁLASZ1= {[(ALKALMAZOTTAK JOIN OSZTÁLYOK (OSZT))
JOIN MIT-CSINÁL? (SZEMSZ)]
JOIN TÉMÁK (TKÓD)]
PROJECT [SZEMSZ,VEZ-NÉV,KER-NÉV,OSZT-NÉV,TÉMA,
MEGOSZL]}

A nézetben minden dolgozóról annyi sor lesz, ahány témán dolgozik. A VÁLASZ1 nézet oszlopai definíciójának ismeretében a felhasználó kérdése az alábbira egyszerűsödik: „kérem a VÁLASZ1 adott dolgozóval azonosított sorait”. Semmi szüksége nincsen arra, hogy tudja, miként jött létre a VÁLASZ1 nézet az eredeti táblázatokból és hogy ezekhez a kérdés megválaszolására milyen utakon fog hozzáférni a rendszer.

A feldolgozás egyszerűbbé tétele mellett gyakran hoznak létre nézeteket adatvédelmi szempontból is. Ha egy felhasználó nem dolgozhat egy reláció minden sorával/oszlopával, akkor egy olyan nézetet hozunk létre korlátozással/projekcióval, amelyik ennek csak az általa feldolgozható részét tartalmazza. A felhasználó a hozzáférési jogosultságot nem a relációra kapja meg, hanem csak erre a nézetre.

Összefoglalva tehát, a relációs adatbázis tervezésének és létrehozásának a lépései következők:

1. A szükséges adatok kiválasztása.
2. Az összetartozó adatok, az entitások meghatározása, adatok csoportosítása.
3. Az entitások közti kapcsolatok meghatározása.
4. A több-több kapcsolatok felbontása új relációk segítségével.
5. A relációk létrehozása, adatok (legalább) első normalizált formákba való csoportosítása.
6. Idegen kulcsok meghatározása a relációk közti kapcsolatok biztosítására.
7. A szükséges magasabb fokú normalizációs elvégzése.
8. Táblázatok létrehozása a normalizált relációk alapján.
9. A várható feladatok adatait együttesen, egyben tartalmazó relációk meghatározása.
10. Az így kapott relációk állandó nézetként való létrehozása.
11. Adatok betöltése, műveletek az adatokkal.

12. Szükség esetén az adatbázis szerkezetének módosítása, új táblázatok, nézetek létrehozása az 1–10-ben leírtak szerint.

A fentiekén kívül természetesen gondoskodni kell az adatok védelméről, a hozzáférési jogok meghatározásáról és betarttatásáról, melyről a 12.2 pontban fogunk részletesebben beszélni.

7.10 Katalógus

7.10.1 A katalógus szerepe

Az adatbázisnak tartalmaznia kell az adatokkal együtt az adatok definiálását, tárolási és használati módjuk leírását. Az adatbázisnak ezt az önálló részét katalógusnak, vagy adatszótárnak nevezzük. A relációs modellben a katalógus maga is relációkból áll, melyekkel elvileg ugyanúgy dolgozhatunk, mint az adatokat tartalmazó táblázatokkal és az ezekből előállított nézetekkel.

A katalógus maga több táblázat és nézet összessége, melyek tartalmazzák a rendszer minden objektumának a leírását. Amennyiben valamely objektumban bármilyen változás következik be, a rendszer automatikusan módosítja a megváltozott objektum adatait nyilvántartó katalógus-relációkat. Így például, ha egy új táblázatot hozunk létre, akkor annak jellemzői automatikusan bekerülnek a táblázatok általános jellemzőit nyilvántartó, az új táblázat oszlopainak adatai pedig az adatbázis összes oszlopát nyilvántartó katalógus-táblázatba.

Az, hogy a katalógusban milyen relációkban, milyen formában, milyen információk vannak, az rendszerfüggő. Olyannyira az, hogy az IBM egységes rendszerarchitektúrájába (SAA) tartozó DB2, SQL/DS és SQL/400 rendszerek is különböző katalógus-táblázatokat tartalmaznak, az azonos típusú objektumokról is különböző információkat tárolnak, sőt, például még a legfontosabb objektumok, a relációk adatait tartalmazó katalógus-reláció nevei is különböznek (SYSIBM.SYSTABLES, SYSTEM.SYSCATALOG, SYSTEM.SYSTABLES). Éppen ezért az alábbiakban konkrét rendszer helyett csak a legfontosabb katalógus-relációk általános tulajdonságait ismertetjük. Megjegyezzük, hogy nagy rendszerek katalógusa 20–30 különböző táblázatot is tartalmazhat.

A katalógus-táblázatok többségét általában mindenki szabadon *lekérdezheti*. Más műveletekre azonban korlátozások állnak fenn, így például *nem lehet*

- katalógus-relációt megszüntetni,
- explicit elzárni más felhasználók elől,
- közvetlen utasítással beírni, törölni, módosítani benne sorokat.

A katalógus sorait a rendszer automatikusan aktualizálja, illetve ha a felhasználó vagy az adatbázis-felügyelő olyan változtatást akar benne végrehajtani, melyre joga van, akkor erre külön katalógus-kezelő utasítások vannak.

7.10.2 Katalógus-táblázatok

A katalógus-táblázatokat, attól függően, hogy miről milyen típusú adatokat tartalmaznak, négy nagy csoportba sorolhatjuk:

- katalógus-táblázatok a rendszer objektumairól, illetve azok részeiről,
- katalógus-táblázatok az egyes objektumokhoz való hozzáférési jogokról,
- katalógus-táblázatok a rendszer különböző objektumai, elemei közti függőségekről,
- a rendszerről való általános információkat tartalmazó katalógus-táblázatok.

7.10.3 Katalógus objektumokról

Az objektumokkal kapcsolatos információk közül általában önálló táblázatban szerepelnek az alábbiak.

Táblázatkatalógus

Ez a rendszer minden egyes táblázatáról és nézetéről (beleértve a katalógus-táblázatokat is) tartalmaz egy sort. Ennek legfontosabb oszlopai

- a táblázat neve,
- a táblázat létrehozója ill. tulajdonosa,¹⁹
- tartalmaz(hat)-e fizikailag is adatokat, vagy csak virtuális reláció (nézet)
- oszlopok száma,
- sorok száma,
- hány lapot foglal le,
- hol található az adatbázisban,
- hány táblázattól függ,
- hány táblázat függ tőle.

A fentiekén kívül tartalmazhat még például információkat a kulcsokról, az eredeti lapokról túlsordult rekordok számáról, átlagos rekordhosszról, sűrűsödésről, valamint szöveges megjegyzést a táblázat funkciójáról.

Oszlop-katalógus

Minden táblázat és nézet minden egyes oszlopáról tartalmaz egy sort. Legfontosabb oszlopai a következők:

- az oszlop neve,
- az oszlopot tartalmazó táblázat/nézet neve,
- a táblázat/nézet létrehozója,
- az oszlop sorszáma a táblázatban/nézetben,
- milyen típusú adatot tartalmaz (karakter, egész, decimális, dátum stb.),
- az oszlop hossza,
- megengedett-e benne null-érték vagy sem.

A fentiekén kívül tartalmazhat még szöveges megjegyzést az oszlop funkciójáról, valamint a rendszer hatékonyságának vizsgálatához olyan adatokat, mint például

¹⁹ Ez általában az a számítógépes azonosító, amellyel az illető bejelentkezik a számítógépnél vagy az adatbázis-kezelő rendszerénél. Olyan rendszerekben, amelyekben a felhasználónak nincsen joga táblázatot fizikailag létrehozni, az adatbázis-felügyelő készíti ezt el helyette. Ebben az esetben a táblázat létrehozója valójában az adatbázis-felügyelő, de ő a tulajdonosi és ezzel együtt a létrehozói jogokat átruházhatja arra a felhasználóra, akinek a részére kialakította az (üres) táblázatot.

dául az adott oszlopban levő különböző értékek száma, ha van rá index, akkor az sűrűsödik-e stb.

Index- és kulcskatalógus

Minden indexről, kulcsról tartalmaznak egy sort. Például az indexekkel kapcsolatos táblázat legfontosabb oszlopai a következők:

- az index neve,
- az index létrehozója,
- a táblázat neve, amelyre az index épült,
- az oszlopok neve, amire az index épül,
- index típus (egyedi, növekvő, elsődleges stb.),
- sűrűsödő-e és ha igen, mennyire,
- különböző indexértékek száma.

Összetett indexeknél a kulcskatalógus hasonló jellegű információkat ad az index egyes oszlopairól is.

Egyéb katalógusok

További katalógusok tartalmazhatják:

- a nézetek definícióját,
- az adatbázis logikai felosztását részekre,
- a lefordított és tárolt felhasználói programoknak az adatbázis elérés módját meghatározó részeinek a leírását,
- szinonimaszótárt, mely lehetővé teszi, hogy minden felhasználó az általa meghatározott névvel hivatkozhasson a rendszer különböző objektumaira.

Megjegyezzük, hogy nem mindegyik rendszer tartalmazza az összes itt felsorolt katalógustáblázatot és az itt ismertetett tartalommal. Ugyanakkor vannak olyanok is, amelyek a fentieknél még részletesebb információt is szolgáltatnak bizonyos objektumokról.

Ugyanez a megjegyzés érvényes az összes további katalógustípusra is.

7.10.4 Katalógus hozzáférési jogokról

Több felhasználó által hozzáférhető adatbázisnál elengedhetetlen, hogy pontosan meg legyen határozva, ki mire jogosult. Ki hozhat létre, vagy törölhet például táblázatokat, ki olvashat, írhat, törölhet vagy módosíthat adatokat egy már meglevő relációban, ki hogyan hivatkozhat egy adott táblázatra stb.

Az általános alapelv az, hogy aki létrehozza a relációt (szokták a reláció tulajdonosának is nevezni), az bármit csinálhat vele, és ezeket a jogokat tovább is adhatja másnak, illetve visszavonhatja bárkitől (l. a GRANT és a REVOKE utasítást a 12.2 és 12.3 pontban). Éppen ezért az objektum leíró katalógusok mindig tartalmazzák az adott objektum létrehozóját is.

Ahhoz azonban, hogy működés közben az adatbázishoz való minden egyes hozzáforduláskor azonnal ellenőrizni tudja a rendszer a jogosultságot, szükséges, hogy ezek az információk is hozzáférhető módon tárolva legyenek az adatbázisban. Ezt a célt szolgálják az alábbi katalógusok:

Általános jogosultság

Az adatbázis használatára jogosultak listája. Ez általában tartalmazza a speciális adatbázis jelszavakat is, ezért ehhez csak az adatbázis-felügyelő férhet hozzá. A többi felhasználó rendszerint ennek csak egy, a jelszót nem tartalmazó nézetét tekintheti meg.

Amennyiben nem mindenki hozhat létre vagy törölhet önálló objektumokat, akkor ennek a táblázatnak oszlopai megmutatják, mire jogosult az adott felhasználó.

Jogosultság táblázatra (nézetre) oszlopra

Azokat a jogosultságokat sorolja fel, amelyekkel az egyes felhasználók rendelkeznek, hogy az adott táblázattal, nézettel—illetve ha ezen belül oszlopra is részletezve van a privilégium, akkor oszloppal—dolgozhassanak.

A táblázatok legfontosabb oszlopai:

- a jogosultságot adó azonosítója,
- a jogosultságot kapó azonosítója,
- az objektum típusa,
- az objektum neve,
- a jogosultság megadásának időpontja,
- mire vonatkozik a jogosultság.

Ez utóbbi vonatkozhat minden műveletre, lehet külön-külön olvasás, beírás, módosítás, törlés, index felépítés vagy idegen kulccsal való hivatkozás engedélyezése is.

A legtöbb rendszerben van egy különleges felhasználó azonosító (Nyilvános/PUBLIC). Ha ez szerepel a katalógusban, akkor az ott engedélyezett műveleteket az objektumon bárki, aki jogosult az adatbázis használatára, elvégezheti.

Alkalmazási program használati joga

Ha az adatbázis-kezelő utasításokat nem párbeszédes módban adják ki, hanem alkalmazási programba építik be, akkor a lefordított program önálló modulként tartalmazza az adatbázis-elérési utasításokat. Amennyiben ezt bárki hozzáfér-csolhatná a saját programjához, futtathatná, akkor az adatbázishoz való hozzáférést nem lehetne ellenőrizni. Éppen ezért külön katalógustáblázat tartalmazza ezeknek az adatbázis-elérési moduloknak a legfontosabb jellemzőit, és azt is, hogy kik jogosultak ezeknek a futtatására.²⁰

7.10.5 Függőségi kapcsolatok

Ez a katalógustáblázat az adatbázis nézeteknek, indexeknek, programoknak a táblázatokról való függőségét tartalmazza. Fontos, mert például egy táblázat megszüntetésével automatikusan megszűnik az összes ráépülő nézet is. Így azok sem

²⁰ A futtatási jogosultság önmagában még nem elegendő. Szükséges az is, hogy a modulban meghatározott konkrét műveletre (pl. az adott táblázatban sorok törlésére) is legyen engedélyünk. Amennyiben az nincs meg, akkor a program futásakor hiba lép fel. A rendszer nem engedi meg a jogosulatlan műveletet.

tudnak tovább dolgozni, akik a táblázathoz hozzá sem férhettek, de a nézetet használhatták. Ugyancsak működésképtelenek lesznek azok a programok is, amelyek egy törölt táblázattal vagy arra épülő nézettel kívánnának dolgozni.

Ha egy nézet több táblázatra épül, vagy egy program több táblázattal dolgozik, akkor a függőséget mindegyikre külön sor jelzi a katalógusban.

A táblázat legfontosabb oszlopai a következők:

- az alapobjektum neve,
- az alapobjektum létrehozója,
- az alapobjektum típusa (nézet, táblázat, index stb.),
- a függő objektum neve,
- a függő objektum típusa (nézet, program),
- a függőség létrehozásának időpontja.

Gyakorlati tanács, hogy ha meg egy objektumot akarunk szüntetni, akkor előbb nézzük meg ebben a katalógusban, hogy az milyen további hatással jár.²¹

7.10.6 Általános rendszerinformációk

Megadják többek között, hogy milyen karakterkészlettel dolgozhatunk, hogyan ábrázoljuk a dátumot és időt (pl: 1992-12-31 vagy 31.12.1992; 1:30 PM stb.), milyen nyelven kommunikál a felhasználóval a rendszer stb.

Ezeket a opciókat általában generáláskor, vagy a rendszer indításakor határozza meg az adatbázis-felügyelő és (alapértelmezésként) a rendszer összes felhasználójára érvényesek.

7.10.7 Katalógus osztott adatbázisokban

Osztott adatbázis katalógusának a centralizált rendszer katalógusa által nyújtott szolgáltatásokon kívül információt kell adni arról is, hogy

- az egyes rekordok fizikailag hol helyezkednek el,
- ha több helyről kell egy logikai rekordot összeszedni, akkor annak összetevői hol találhatók,
- milyen adatokat tárol a rendszer duplikáltan, azok hol találhatók és hogyan kell konzisztenciájukat biztosítani.

Ezek a megfelelően definiált új, az osztott rendszerek igényeit is figyelembe vevő katalógustáblázatokból megkaphatók. Sokkal nagyobb problémát jelent a gyakorlatban az, hogy hol tároljuk magát a katalógust. Alapjában véve a következő lehetőségek közül választhatunk:

1. Központi, centralizált katalógus.
2. A teljes katalógus több példányban van a rendszerben. Minden önálló feldolgozóhely saját, teljes példánnyal rendelkezik.

²¹ Vannak olyan adatbázis-kezelő rendszerek, amelyek nem is engedik meg addig pl. egy táblázat törlését, amíg létezik az adatbázisban olyan nézet, amelyik erre hivatkozik. (Ilyen pl. az IBM AS/400 adatbázis-kezelő rendszere.)

3. Particionált. Minden önálló hely csak az ott tárolt objektumokról vezet nyilvántartást. A teljes katalógus az egyes helyeken levő katalógusok uniója.
4. Minden hely vezeti a saját objektumainak a nyilvántartását és ezen kívül egy teljes központi nyilvántartás is van egy helyen (1 és 3 kombinációja).
5. Az egész rendszerre vonatkozó egységes névkonvenció-használat és katalógus-tartalom.

Mindegyik megoldásnak megvannak az előnyei és a hátrányai. Ha a különböző elemzések alapján végül osztott adatbázis mellett döntöttünk, akkor az 1. típusú megoldásnál ennek előnyeit nagymértékben elveszítjük. A központi katalógus miatt az egyes helyek továbbra is jelentősen függnék a központtól. Ennek mértéke ugyan kisebb, hiszen a katalógus naprakészen tartása nem függ a központ kényekedvtől, automatikus, de a fizikai függés teljes mértékben megmarad.

A 2. módszer gyorsan módosuló rendszereknél szinte kivihetetlen, ha csak nem engedjük meg, hogy az adatok bizonyos ideig inkonzisztensek legyenek. Az összes katalóguspéldány egyidejű aktualizálása olyan bonyodalmas eljárásokhoz vezethet, hogy a tényleges munka megbénulhat.

A 3. módszer csak akkor előnyös, ha a műveletek csaknem kizárólag a helyi adatbázisra vonatkoznak. Ha máshol tárolt adatokra van szükségünk, akkor ezek megtalálásához kedvezőtlen esetben (márpedig Murphy szerint csaknem minden eset ilyen lesz) a rendszer összes katalógusát végig kell vizsgálnunk.

A 4. módszer kiküszöböli ezt a hátrányt, de ismét behozza a központi helytől való függőséget.

A gyakorlatban az 5. módszer kecsegtet a legjobb eredménnyel, ha biztosítjuk az alábbi szabályok betartását:

a) Minden objektumnevet kiegészítünk, minősítünk

- létrehozója azonosítójával,
- létrehozásának helyével,
- első tárolási helyével.

b) Minden helyi katalógusban tároljuk

- az ott használt objektumok szinonímáját,
- az ott tárolt objektumok leírását,
- az ott létrehozott objektumok pillanatnyi tárolási helyét.

Objektumok egyik helyről másikkra való mozgatásánál ezt az adatot a rendszer automatikusan aktualizálja.

Bármilyen objektumra való hivatkozásnál a minősítésből (vagy ha nincs minősítve, akkor a rendszer helyben keresi és itt megtalálja, vagy a szinonima-katalógus alapján kiegészíti) megállapítható, hogy hol hozták létre az objektumot.

Első lépésként a rendszer ott is keresi, és ha nem találja már ott, akkor az ottani katalógusból megkapja a pillanatnyi tárolási helyet.

Látjuk tehát, hogy ezzel a módszerrel bármely objektumot legfőljebb két lépés után megtalálunk. Az adatbázis osztottsága teljes mértékben megmarad és többszörösen csak az objektumok teljes nevét tartalmazó szinonima-katalógust kell tárolni és ennek aktualizálásáról kell gondoskodni.

7.11 A relációs modell összefoglalása

Bár az eddigiekben már részletesen tárgyaltuk, most, éppen amiatt, hogy a korszerű adatbázis-kezelő rendszerek már mind a relációs modellből indultak ki, és várhatóan az elkövetkezendő évtizedben mind jobban teljesíteni fogják a modell feltételeit, összefoglaljuk a relációs modell alapkövetelményeit.

A relációs modell három pilléren nyugszik. Ezek a következők:

- a relációs adatstruktúra megvalósítása,
- az adatok integritásának biztosítása,
- adatkezelési lehetőségek.

A relációs adatstruktúra azt jelenti, hogy minden egyedítípust n -ed fokú relációkban (táblázatokban) ábrázolunk, melyek egyedi előfordulásai (sorai) elemi tartományokra épülő (domain) attribútum (oszlop) értékekből állnak. Egyértelmű azonosításuk a reláción belül az elsődleges kulcs segítségével történik.

Az adatbázison belül minden elem egyértelműen azonosítható a reláció nevével és elsődleges kulcsának értékével.

Az adatstruktúra létrehozásához és módosításához rendelkezni kell egy adatleíró lehetőséggel, melynek segítségével alaprelációk, illetve ezekből relációs műveletekkel származtatott, virtuális relációk létrehozhatók.

Az *adatok integritása* az elsődleges és idegen kulcsok segítségével valósul meg. Az elsődleges kulcs értéke mindig egyedi egy reláción belül és egy összetevője sem lehet null-érték.

Az idegen kulcsok értéke meg kell egyezzen a hivatkozott reláció valamelyik elsődleges kulcsának értékével, vagy a null-értékkel egyenlő.

Az *adatkezelés* a relációs algebra vagy kalkulus műveleteivel lehetővé kell tegye adatok kiválasztását olvasáshoz, módosításhoz, törléshez, beíráshoz, új relációk létrehozásához.

Azokat a rendszereket, amelyek a fenti alapkövetelményeket az alábbiakban még bővebben értelmezett módon minden szempontból kielégítik, *teljes relációs rendszernek* nevezzük. Ilyen jelenleg—a szerző tudomása szerint—a könyv írása-kor kereskedelmi forgalomban még nem kapható.

Azokat a rendszereket, amelyek az adatokat táblázatokkal (és csak táblázatokkal) ábrázolják és legalább a korlátozást, projekciót és a természetes egyesítést megengedik anélkül, hogy ennek végrehajtásához a felhasználónak bármilyen adatelérési utat definiálni kellene, *minimális relációs rendszernek* nevezzük. A személyi számítógép-rendszerekre készített olcsó relációs rendszereknek a többsége ilyen. Ezek alapján viszont nem tekintjük relációs rendszernek például a gyakorlatban igen elterjedt és közkedvelt táblázatkezelő programokat—még akkor sem, ha ezeket adatbázis-kezelő kiegészítésekkel reklámozzák— mivel a legfontosabb relációs műveletet, az egyesítést nem támogatják.

A *relációs szempontból teljes rendszerek* önálló utasításokat tartalmaznak a 7.6 pontban ismertetett relációs műveletek vagy azok többségének elvégzésére és általában valamilyen mértékben biztosítják az adatok integritását is. A kereskedelmi forgalomban levő ismertebb rendszerek közül ezek közé tartozik pl. a DB2, SQL/DS, SQL/400, INGRES, ORACLE stb.

A *teljes relációs rendszereknek* a jelenlegi álláspont szerint az alábbi követelmények mindegyikét ki kell elégíteni:

1. Az adatbázisban minden információt relációkban és csak relációkban tárolnak.
2. Minden információ egyértelműen elérhető a megfelelő reláció, ezen belül pedig a kívánt oszlop nevének és a sor elsődleges kulcsa értékének a megadásával.
3. A null-értékeket egyértelmű és megfelelő módon kezeli a rendszer.
4. Bármikor hozzáférhető, naprakész katalógussal rendelkezik.
5. Van olyan párbeszédés és programba is beépíthető nyelve, amely lehetővé teszi
 - adatok, relációk definiálását,
 - relációs műveletek elvégzését,
 - integritási és adatbiztonsági feltételek definiálását,
 - összetett tranzakciók egységes kezelését (l. COMMIT és ROLLBACK utasításokat a 12.5 és 12.6 pontban).
6. Táblázatok módosíthatók nézeteken keresztül, amennyiben ez egyértelműen, ellentmondásmentesen elvégezhető.
7. Rendelkezik adatok csoportos beírásával, módosításával, törlésével lehetővé tevő műveletekkel.
8. Biztosítja a fizikai adatfüggetlenséget.
9. Biztosítja a logikai adatfüggetlenséget.
10. Az integritási feltételeket maga az adatbázis tartalmazza. Betartásuk független az alkalmazási programoktól. Ezek a feltételek módosíthatók, de az új feltételek az adatbázisban már bennlevő adatokra is érvényben kell legyenek.
11. A fenti szabályoknak érvényben kell maradni függetlenül attól, hogy az adatbázist centralizáltan építik fel, vagy osztott formában.
12. Az integritási és biztonsági korlátozásoknak akkor is érvényben kell maradni, ha alacsonyabb szinten lehetőség van az egyes rekordokhoz való közvetlen hozzáférésre is.

Az előzőekben felsorolt, relációs szempontból teljes rendszerek ezen feltételrendszer jelentős részét már kielégítik, de még nem mindegyiket vagy nem maradéktalanul. Ennek ellenére ezek a mindennapi, átlagos felhasználó számára gyakorlatilag már teljes (vagy csaknem teljes) relációs rendszereknek tekinthetők.

Szöveges információ- visszakereső rendszerek

Az adatbázisok speciális kategóriáját alkotják a szöveges információ-visszakereső rendszerek. Ezekben az információkat nem, vagy csak részben kötött módon tárolják folyamatos, vagy csak bizonyos szempontok szerint részben tagolt szövegeként. A szűkebb értelemben vett adatbázis-terminológiában ezeket a rendszereket nem is szokták igazából adatbázisoknak tekinteni, vagy kiegészítésképpen hozzátesszik a dokumentum formátumú jelzőt. A használatukat biztosító szoftvert is megkülönböztetésképpen szöveges információ-visszakereső rendszernek nevezik.

A gyakorlatban azonban a klasszikus adatbázisok, a formázott adatokat tartalmazó rendszerek, és a dokumentumformátumokat kezelő rendszerek sokszor egybefonódnak. Csaknem minden korszerű relációs adatbázis-kezelő rendszernek van szöveges mezőkben jelsorozat egyezősége alapján való visszakeresést biztosító lehetősége. Ezzel dolgozva—bár korlátozott lehetőséggel és kisebb hatékonysággal—tényadatbázisok is működhetnek szöveges információkereső rendszerként.

Ezért célszerűnek tartjuk, hogy röviden összefoglaljuk ezen rendszerek legfontosabb tulajdonságait és jellemzőit.

A (tényadat formátumú) adatbázisban egy adott kérdésre válaszként kapott rekordok mindegyike egy vagy több különféle relációnak, vagy rekordtípusnak az adataiból tevődik össze, melyeket a relációkon végzett műveletekkel vagy a rekordok között fennálló set-kapcsolatok révén gyűjt össze az adatbázis-kezelő rendszer. A szöveges információ-visszakereső rendszerekben a kérdésre válaszként kapott rekordok megegyeznek az adatállomány azon rekordjaival (illetve annak egy meghatározott részével), melynek mezői részben vagy teljesen tartalmazzák (vagy nem tartalmazzák) a kereséskor megadott jelsorozatokat, és ezek között a kérdés feltevésekor megszabott logikai kapcsolatok is teljesülnek. Tipikus példa erre, amikor cikkekből, könyvekből a tartalmuk rövid leírása alapján, külön tárgyszavazás nélkül akarjuk kiválasztani a bennünket érdeklő anyagok leírását, mondjuk azokat, amelyeknek ismertetésében szerepel a „népszerű” vagy „ismeretterjesztő”, és ezen kívül a „számítástechnika” szó is (pontos meghatározása a jelsorozatnak) vagy, melyek a „népszerű ...” vagy „ismeretterjesztő ...” szavak valamelyikével és a „számítástechnika ...” jelsorozattal kezdődnek. Az előbbi esetben azok a művek, melyeknek ismertetésében ez szerepel: „... népszerűen ismerteti

a számítástechnikának a legújabb eredményeit ..." nem fognak megjelenni az eredmény-rekordok között, az utóbbiban viszont igen.

A szöveges információ-visszakereső rendszerekben a keresés és egyeztetés történhet az egyes dokumentumok soros végigolvasásával (a legtöbb szövegszerkesztőben ez történik), ami nagyobb adatállománynál már igen hosszú időt vesz igénybe, vagy pedig a szöveg minden egyes szavára vagy az anyagnak a rendszerbe való bevitelkor előre megjelölt szavaira felépített indexek segítségével. Ez utóbbi esetben az indextáblák felépítése miatt a tárolóigény lényegesen nagyobb (a gyakorlatban—a keresésre megjelölt szavak számától függően—általában mintegy 2–3-szorosa) lesz. A dokumentumok bevitele, módosítása lényegesen hosszabb időt vesz igénybe, de a visszakeresés sokkal gyorsabb lesz, mint a soros végigolvasással kereső rendszereknél.

A 8.1 ábrán egy televíziós híryanagot tartalmazó visszakereső rendszer egy részének leegyszerűsített változatát látjuk. A dokumentumok az alábbi mezőkből állnak:

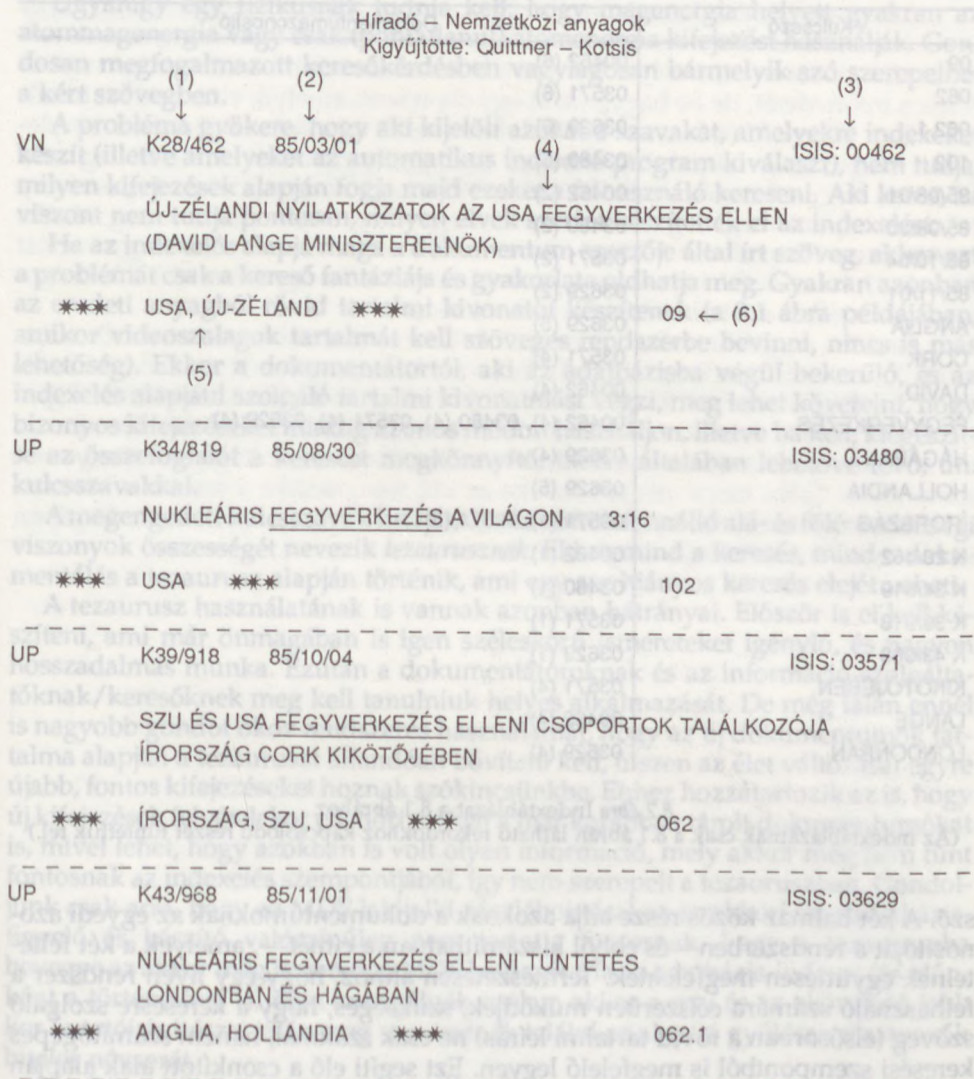
1. Videoszalag tárolási helye (Kxx/xxx).
2. Esemény időpontja (év/hó/nap).
3. Dokumentum azonosítója a számítógépes rendszerben, az adatbázisban (ISIS: xxxx).
4. Rövid tartalmi leírás.
5. Helyszín.
6. Szakkatalógus szám.
7. Megjegyzés (részletesebb leírás stb., az ábrán nem tüntettük fel).

Az ábrán látható anyagokat az 1985-ös híryanagból a „FEGYVERKEZÉS” szóra való keresés eredményeként szolgáltatta a rendszer.

A 8.2 ábra a felsorolt dokumentumokhoz tartozó indextábla egy részét mutatja. Ezt a rendszer az erre kijelölt mezők minden szavára felépíti, kivéve a rövid tartalmi leírás tiltott szavaira (a, az, ellen, elleni, és, melyeknek tételes listáját egy külön állományban, az ún. STOPWORD-FILE-ban tároljuk).

Az indextáblázat kulcsrésze a dokumentumoknak a keresésre szolgáló szavait tartalmazza. A további index bejegyzésekben azoknak a dokumentumoknak az adatbázisbeli azonosítója (címe) és ezen belül annak a mezőnek a sorszáma van, melyekben a kulcs előfordul. Egyes rendszerekben szerepel még a kulcs mezőbeli relatív helye is, mely lehetővé teszi elsősorban összetett keresési feltételeknél azon találatok megkülönböztetését és kiválasztását, melyeknél a keresett jelsorozatok egymáshoz viszonyított helyzete nem közömbös. (Pl. a Péter László nevű személyre való utalást tartalmazó dokumentum keresésekor a kereső kulcsok „Péter” és „László”, de előfordulásuk sorrendje lényeges, a „László Péter”-re hivatkozó anyagokra nincsen szükségünk¹). A gyorsabb keresés és az egyes keresési feltételek közti logikai kapcsolatok gyors megállapítása érdekében mind a kulcsszavak, mind az egyes kulcsszavakhoz tartozó dokumentumazonosítók (címek) rendezve vannak. A dokumentum kiválasztása a kulcsszavak alapján ezeknek az indextáblázatban való kikereséséből és a hozzájuk tartozó címen levő dokumentumok teljes vagy a rendszer által meghatározott formátumú beolvasásából áll.

¹Természetesen, ha a kereséshez az index készítést nem az egyes szavakra, hanem a teljes névre adnánk meg, akkor a fenti probléma nem merülne fel.



8.1 ábra

A televíziós hírányagokat tartalmazó visszakereső rendszer leegyszerűsített változata.
Az ábrán az 1985-ös anyagból a "FEGYVERKEZÉS" szóra történő keresés eredményét
láthatjuk.

Így például az előző ábrákon bemutatott példán a FEGYVERKEZÉS-sel bármilyen formában foglalkozó 1985-ös videoanyagokat (az ábrán már csak ezeket tüntettük fel) úgy választja ki a rendszer, hogy az indextáblázatból kiírja azoknak a rekordoknak a rendszerbeli azonosítóját, melyeknek időpontmezője 85-tel kezdődik, valamint azokét, melyeknek tartalmi leírásában szerepel a FEGYVERKEZÉS

Kulcsszó	Dokumentumazonosító
09	00462 (6)
062	03571 (6)
062.1	03629 (6)
102	03480 (6)
85/03/01	00462 (2)
85/08/30	03480 (2)
85/10/04	03571 (2)
85/11/01	03629 (2)
ANGLIA	03629 (5)
CORK	03571 (4)
DAVID	00462 (4)
FEGYVERKEZÉS	00462 (4), 03480 (4), 03571 (4), 03629 (4)
HÁGÁBAN	03629 (4)
HOLLANDIA	03629 (5)
ÍRORSZÁG	03571 (4), 03571 (5)
K 28/462	00462 (1)
K 34/819	03480 (1)
K 39/918	03571 (1)
K 43/968	03629 (1)
KIKÖTŐJÉBEN	03571 (4)
LANGE	00462 (4)
LONDONBAN	03629 (4)

8.2 ábra Indextáblázat a 8.1 ábrához

(Az indextáblázatnak csak a 8.1 ábrán látható rekordokhoz kapcsolódó részét tüntettük fel.)

szó. A két halmaz közös része adja azoknak a dokumentumoknak az egyedi azonosítóját a rendszerben—és ebből kiszámíthatóan a címét—, amelyek a két feltételnek együttesen megfelelnek. Természetesen ahhoz, hogy egy ilyen rendszer a felhasználó számára célszerűen működjék, szükséges, hogy a keresésre szolgáló szöveg (elsősorban a rövid tartalmi leírás) ne csak szakmai, hanem számítógépes keresési szempontból is megfelelő legyen. Ezt segíti elő a csonkított alak alapján való keresés lehetősége is. Ennél a kulcsszóval való egyeztetés csak egy speciális karakterig (a példaként megadott rendszerben a keresőszóban levő dollárjelig) történik. Ilyen fajta módon ugyanannak a szónak a különböző ragozott alakjaira nem kell külön keresnünk. A FEGYVERKEZÉS\$ szóra történő kereséssel megtaláljuk az összes olyan dokumentumot, melyben a FEGYVERKEZÉS, FEGYVERKEZÉSI, FEGYVERKEZÉSEL ... stb. szavak szerepelnek.

Gondot okozhat keresésnél a megfelelő kulcsszavak kiválasztása. Lehet, hogy a vegyész felhasználó a nátrium bizonyos tulajdonságaival foglalkozó szakirodalomra kíváncsi, de az adott cikkben ez a szó nem szerepel. Helyette a szerző például alkáli fémekről beszél. Ha ezt nem adják meg vaglyagos kereső szóként, akkor ez a cikk nem fog szerepelni a kiválasztottak között.

Ugyanígy egy fizikusnak tudnia kell, hogy magenergia helyett gyakran az atommagenergia vagy csak (pontatlanul) atomenergia kifejezést használják. Gondosan megfogalmazott keresőkérdésben vagylagosan bármelyik szó szerepelhet a kért szövegben.

A probléma gyökere, hogy aki kijelöli azokat a szavakat, amelyekre indexeket készít (illetve amelyeket az automatikus indexelő program kiválaszt), nem tudja, milyen kifejezések alapján fogja majd ezeket a felhasználó kereseni. Aki keres, az viszont nem tudja pontosan, milyen elvek szerint készítették el az indexelést.

Ha az indexelés alapja maga a dokumentum szerzője által írt szöveg, akkor ezt a problémát csak a kereső fantáziája és gyakorlata oldhatja meg. Gyakran azonban az eredeti anyagból rövid tartalmi kivonatot készítenek (a 8.1 ábra példájában, amikor videoszalagok tartalmát kell szöveges rendszerbe bevinni, nincs is más lehetőség). Ekkor a dokumentátortól, aki az adatbázisba végül bekerülő, és az indexelés alapjául szolgáló tartalmi kivonatolást végzi, meg lehet követelni, hogy bizonyos kifejezéseket mindig azonos módon használjon, illetve ha kell, kiegészítse az összefoglalót a keresést megkönnyítő, illetve általában lehetővé tevő, ún. kulcsszavakkal.

A megengedett kulcsszavak listáját, és a közöttük fennálló alá- és fölérendeltségi viszonyok összességét nevezik *tezaurusznak*. Ekkor mind a keresés, mind a dokumentálás a tezaurusz alapján történik, ami egy sor hiányos keresés elejét veheti.

A tezaurusz használatának is vannak azonban hátrányai. Először is el kell készíteni, ami már önmagában is igen széleskörű ismereteket igénylő, és nagyon hosszadalmas munka. Ezután a dokumentátoroknak és az információ-szolgáltatóknak/keresőknek meg kell tanulniuk helyes alkalmazását. De még talán ennél is nagyobb gondot okoz rendszeres használatnál, hogy az új dokumentumok tartalma alapján a tezauruszt állandóan bővíteni kell, hiszen az élet változásai egyre újabb, fontos kifejezéseket hoznak szókincsünkbe. Ehhez hozzátartozik az is, hogy új kifejezések felvételekor ellenőrizni kell a már korábban tárolt dokumentumokat is, mivel lehet, hogy azokban is volt olyan információ, mely akkor még nem tűnt fontosnak az indexelés szempontjából, így nem szerepelt a tezauruszban. Gondoljunk csak arra, hogy az MDF lakitelki zászlóbontásakor az akkori sajtóadatbázis-figyelő és készítő valószínűleg nem tartotta fontosnak, hogy a tezauruszba bevegye az akkor még ismeretlen MDF betűszót. Ha az adatbázis-indexet (és időnként a történelmet is) nem korrigálnák utólag, akkor a mai és az eljövendő jobb kor kutatói és írászei talán meg sem tudnák találni az alapító gyűlésen résztvevők hiteles névsorát.

További segítséget nyújt a felhasználónak a szinonimaszótár. Hasonlóan a tiltott szavak listájához, ez is egy önálló állomány, melynek keresési kulcsa az alapszó (pl. BENELUX), mezői a vele egyenértékű vagy azt helyettesítő szavak (pl. BELGIUM, HOLLANDIA, NÉMETÁLFÖLD, LUXEMBURG). Amennyiben a keresést a megadott szó összes helyettesítőjére, szinonimájára is ki akarjuk terjeszteni, akkor ezt a rendszerrel egy speciális kifejezés (példánkban ANY=bármelyik) megadásával közöljük. Ebben az esetben a rendszer automatikusan végignézi a szó mindegyik szinonimájához, mint kulcshoz tartozó indexet az összes dokumentumhoz tartozó indextáblázatban. A kiválasztási feltételnek megfelelő anyag az itt felsoroltak összessége (logikai VAGY) lesz.

A dokumentálás és a keresés összhangja szempontjából a legfontosabb a helyes *találati arány* kialakítása. Ez azt jelenti, hogy ne válasszunk ki fölöslegesen túl sok olyan dokumentumot, amelyek nem tartalmaznak az adott kérdés szempontjából értékes információt, de ne hagyjunk ki egy olyat sem, amelyik viszont illet tartalmaz. Ez a jellemző két összetevőből áll. Mindkettőt a keresés feltételeinek ténylegesen eleget tevő rekordok számához kell viszonyítanunk. Az első tag arra jellemző, hogy a keresés eredményeként a tényleges megfelelőkhöz képest hány rekordot kapunk meg, míg a második arra, hogy hányat nem találtunk meg. Ideális esetben ezen hányadosok értéke 1 illetve 0. Jól kialakított rendszerben gyakorlott lekérdező, ha az első hányadosra a gyakorlatban 1-nél nagyobb értékkel is dolgozik, a másodikra a 0-t általában biztosítani tudja.

A modern, nagyfelbontású képernyők és lézernyomtatók lehetővé teszik a számítógépben tárolt grafikus információk kiváló képi megjelenítését. Ezért várható, hogy a szöveges információ-visszakereső rendszerek hamarosan kibővülnek a grafikus (vagy képi) visszakereső, majd a hangfelismerő rendszerekkel is. Ennek már ma is vannak fontos gyakorlati alkalmazásai (például a számítógépes ujjlenyomat azonosítás). Talán nincs messze már az az idő sem, amikor a rendőr a körözött személy, vagy a büszke apa az újszülött fényképe helyett annak számítógépen tárolt digitális képét fogja magával vinni.

III.

Az SQL AZ SQL NYELV értelése

2.1 Az SQL mint szabvány

SQL a Structured Query Language, szó szerint Struktúrált Lekérdező Nyelv rövidítése. Valójában azonban sokkal több ennél; egy teljes relációs adatbázis-kezelő nyelv, amelyet ma gyakorlatilag szabványnak vagy majdnem szabványnak tekinthetünk.

Az SQL kifejlesztése és széleskörű elfogadása egyike a számítástechnika azon ritka területeinek, ahol az IBM piac- és gyakorlatorientált fejlesztői és a számítástudomány elméleti tudósai lényegében ugyanabba az irányba indultak el, és hasonló eredményekre jutottak. Így a tyúk vagy a tojás példájához hasonlóan nehéz volna eldönteni, hogy egy SQL típusú adatbázis-kezelő nyelv elméleti megvalósítása segítette-e elő az IBM SQL alapú rendszerréneked a kvázi szabványként való elfogadását, vagy az IBM szíja lényegesen az elméleti és szabványozási szakembereket arra, hogy lényegében ezek a rendszerek szolgáljanak előrelépésként a továbbiak kifejlesztéséhez és illeszkedéséhez. Nem is ez a lényeges, hanem inkább az az öröndetes tény, hogy a relációs adatbázis-kezelő rendszerek korszerűségi fogalomban való megjelenésével csaknem egyidőleg kialakult ezeknek a rendszereknek egy szinte minden fejlesztő és felhasználó által elfogadott nyelvi, melyhez a későbbiekben a többiek már a saját érdekében is kénytelenek voltak alkalmazkodni. Természetesen a különböző cégek SQL-ként foglalkozott termékei között is vannak eltérések. Sőt, például az IBM különböző architektúrákú rendszerekre készített SQL rendszerei (DB2 az MVS; SQL/DS a VM; SQL/VS az OS/400; az OS/400, valamint az OS/2 EEDM az OS/2 környezetben) szinte kétféle „más dialektusban beszélnek”. Utasításkészleteik különböznek, de még az azonos utasítások is eltérő lehetőségekkel rendelkeznek. Eppen ezért az SQL a mai napig nem valamilyen létező SQL rendszer pontos szabványaként és formájában, hanem a nyelv legfontosabb elemeit ismertetik. Ezek számos különböző implementációban lévő változatban megvalósultak már. Ezt azonos megfogalmazású szabvány további pontos lehetőségekkel, amelyek a legfontosabb rendszerekben már ma is léteznek és alkalmazhatók.

Formalizmusát tekintve az SQL a relációs kalkulációhoz, a nem-procedurális nyelvekhez áll közel. A felhasználó lényegében csak azt mondja meg, mit akar. Azt,

Az SQL nyelv általános ismertetése

9.1 Az SQL mint szabvány

SQL a Structured Query Language, szószerint Struktúrált Lekérdező Nyelv rövidítése. Valójában azonban sokkal több ennél; egy teljes relációs adatbázis-kezelő nyelv, amelyet ma gyakorlatilag szabványnak vagy majdnem szabványnak tekinthetünk.

Az SQL kifejlesztése és széleskörű elfogadása egyike a számítástechnika azon ritka területeinek, ahol az IBM piac- és gyakorlatorientált fejlesztői és a számítástudomány elméleti tudósai lényegében ugyanabba az irányba indultak el, és hasonló eredményekre jutottak. Így a tyúk vagy a tojás példájához hasonlóan nehéz volna eldönteni, hogy egy SQL típusú adatbázis-kezelő nyelv elméleti megalapozása segítette-e elő az IBM SQL alapú rendszereinek a kvázi szabványként való elfogadását, vagy az IBM súlya kényszerítette az elméleti és szabványosító szakembereket arra, hogy lényegében ezek a rendszerek szolgáljanak etalonként a továbbiak kifejlesztéséhez és illesztéséhez. Nem is ez a lényeges, hanem inkább az az örvendetes tény, hogy a relációs adatbázis-kezelő rendszerek kereskedelmi forgalomban való megjelenésével csaknem egyidejűleg kialakult ezeknek a rendszereknek egy szinte minden fejlesztő és felhasználó által elfogadott nyelve, melyhez a későbbiekben a többiek már a saját érdekükben is kénytelenek voltak alkalmazkodni. Természetesen a különböző cégek SQL-ként forgalmazott termékei között is vannak eltérések. Sőt, például az IBM különböző operációs rendszerekre készített SQL rendszerei (DB2 az MVS; SQL/DS a VM és VSE, SQL/400 az OS/400, valamint az OS/2 EEDM az OS/2 környezetben) mind kissé „más dialektusban beszélnek”. Utasításkészletük különbözik, de még az azonos utasítások is eltérő lehetőségekkel rendelkeznek. Éppen emiatt ebben a részben nem valamilyen létező SQL rendszer pontos szintaktikáját és leírását adjuk, hanem a nyelv legfontosabb elemeit ismertetjük. Ezek szinte mindegyik forgalomban levő változatban megvalósultak már. Ezt aztán kiegészítjük néhány további fontos lehetőséggel, amelyek a legfejlettebb rendszerekben már ma is léteznek és alkalmazhatóak.

Formalizmusát tekintve az SQL a relációs kalkulushoz, a nem-procedurális nyelvekhez áll közel. A felhasználó lényegében csak azt mondja meg, *mit akar*. Azt,

hogyan, milyen lépések alapján történjen meg, a nyelv értelmezője, fordítóprogramja fogja meghatározni az adatbázis szerkezetének ismeretében. A feladat megoldási algoritmusának meghatározásához jó SQL rendszerek a logikai és fizikai adatszerkezeten kívül figyelembe veszik a tényleges adattartalmat és adatmennyiséget is, hogy az elérési időt optimalizálják. Az ehhez szükséges információkat a rendszer katalógusából merítik.

9.2 Szintaktikai jelölés

Ahhoz, hogy egy nyelvet megismerhessünk, meg kell ismernünk az utasítások szintaktikáját. Ezeket legpontosabban metanyelven, a Bachus-Naur Formában definiálhatnánk. Mivel azonban könyvünkben az SQL nyelv általános jellemzőit ismertetjük és nem konkrét használati utasítást egy adott rendszerhez, célszerűbbnek tartjuk a pontos, de nehezen követhető BNF szintaxis helyett az alábbi, könnyen érthető grafikus szintaktikai leírást használni. A gyakorlat számára ennek precizitása teljesen elegendő. Ráadásul az olvasó, ha valamilyen adatbázis-kezelő rendszerrel kíván majd dolgozni, nagyon valószínű, hogy annak referencia- és felhasználói kézikönyvében is ilyen, vagy ehhez hasonló jellegű szintaktikus definíciókkal fog találkozni.

9.2.1 Jelölési mód

A szintaxis jelölésben megkülönböztetünk

- SQL kulcsszavakat, ezeket a szintaxis-diagrammban nagybetűkkel írjuk.¹

Pl. DELETE

Az SQL kulcsszavakat úgy kell írni, ahogy a szintaxisban megadtuk. Ha nem, vagy nem így adjuk meg őket, akkor az SQL fordító/értelmező szintaktikai hibát jelez. Az SQL kulcsszavakat csak a szintaxis-diagrammban jelölt célra használhatjuk.

- Felhasználó, vagy az adatbázis felügyelő által definiált objektumok, például táblázatok, oszlopok nevei. Ezeknek képzésére nyelvtől függő szabályok vannak (pl. maximális hossz, milyen karakterekből állhat). A szintaktikus jelölésben kisbetűkkel jelezzük

Pl. táblázat_1

- Aritmetikai és logikai operátorok

(+, -, *, /, **, >, <, =, >=, <=, <>).

Ezek rezervált szavaknak tekintendők. Csak ebben az értelemben és pontosan a megadott formában használhatjuk őket.

¹ Az SQL értelmezőjétől függ, hogy a tényleges utasításokban egyaránt használhatunk-e kis- és nagybetűket, és ha igen, akkor azok különbözőeknek vagy azonosaknak tekinthetők-e, vagy pedig az operandusként szolgáló rögzített karaktersorozatok (literálok) kivételével mindent nagybetűkkel kell-e megadnunk az utasításokban. A nagy- és kisbetűk használatának itt ismertetett megkülönböztetése csak az utasítások szintaxisának megadásában, a formális leírásban érvényes.

- A szintaxis-diagram további speciális karaktereit (. ; : stb.) pontosan úgy, és oda kell az utasításba beírni, ahogyan azok a diagramban vannak.
- A kötelező és az opcionális klauzulákat a szintaxisban megadott sorrendben kell megadni.

9.2.2 A diagram értelmezése

Az utasításokat a megadott vonal (út) mentén balról jobbra, illetve sor végétől felülről lefelé a következő sor elejétől olvassuk.

Néhány speciális jelölés:

- ▶▶ — utasítás kezdete,
- ▶ utasítás folytatódik a következő sorban,
- ▶ — előző sor folytatása,
- ▶▶ utasítás vége.

Bizonyos esetekben, amikor egyes utasításoknak csak egy önálló klauzuláját írjuk le, ennek kezdetének és végének jelölésére is az ▶ — és — ▶▶ szimbólumokat használjuk, de a teljes utasításokon belül ezeket külön nem jelezzük így.

Az utasítások írásmódja a következő:

- kötelező elemet a főútvonalra írunk

▶▶ — UTASÍTÁS — kötelező elem — ▶▶,

- opcionális elemet a főútvonal alá írunk

▶▶ — UTASÍTÁS — [opcionális elem] — ▶▶,

- ha több elem közül választhatunk, akkor azokat egymás alá írjuk,
- ha pontosan egyet ki kell választanunk

▶▶ — UTASÍTÁS — [opció-1] [opció-2] — ▶▶,

- ha egy vagy több lehetőség közül lehet, hogy egyet sem választunk, akkor

▶▶ — UTASÍTÁS — [opció-1] [opció-2] — ▶▶,

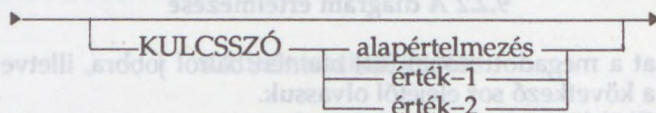
- ismétlődő elem jelölése

▶▶ — UTASÍTÁS — [ismételhető elem] — ▶▶

▶▶ — UTASÍTÁS — [ismételhető opció-1] [ismételhető opció-2] [ism.opció-3] [ism.opció-4] — ▶▶

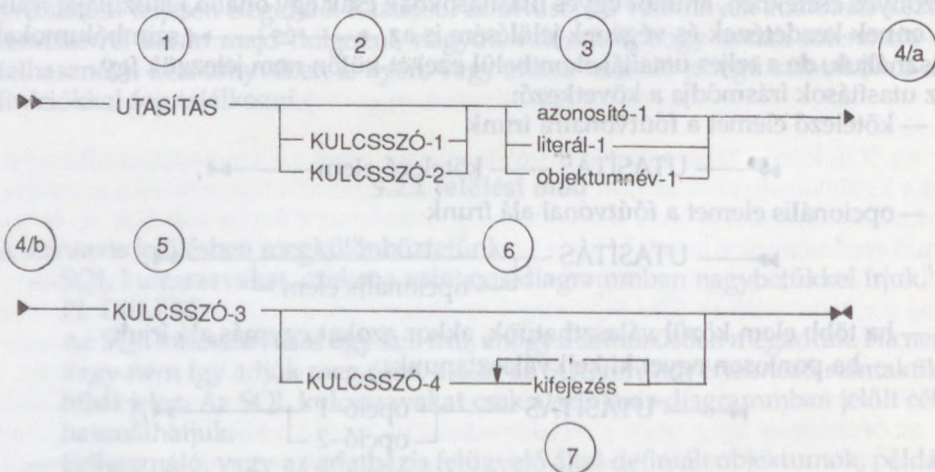
Az első példában az ismétlődő elemet legalább egyszer meg kell adnunk, a másodikban mind az 1, mind a 2 opció elmaradhat, míg a 3 és 4 közül legalább az egyiknek egyszer elő kell fordulni.

Ha egy klauzulát teljesen elhagyunk egy több értékből csak egyet megengedő opciónál és ekkor annak értéke mindig egy meghatározott érték lesz, akkor ezt az alapértelmezést aláhúzással jelöljük, pl.

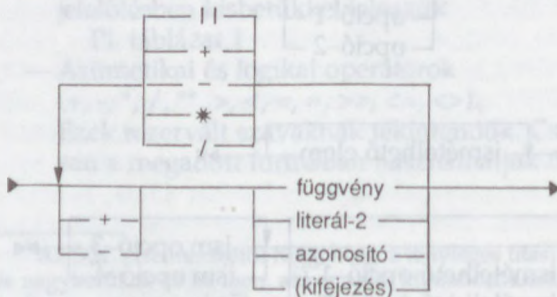


Így tehát a „KULCSSZÓ alapértelmezést” nem kell kiírunk. Ha nem írunk sem „KULCSSZÓ érték-1”-et, sem „KULCSSZÓ érték-2”-t az utasításban, akkor ez egyenértékű az alapértelmezés megtartásával.

Egy utasítás megadásának általános formáját a 9.1 ábrán láthatjuk.



ahol kifejezés:



8

9.1 ábra SQL utasítás általános leírása

Néhány megjegyzés a szintaxis diagram értelmezéséhez:

1. Az utasítás neve. Kulcsszó, megadása kötelező, pontosan így kell írni.
2. Elhagyható kulcsszavak. Egyet sem, vagy legföljebb egyiket adhatjuk csak meg. Ekkor azt pontosan így kell írni.
3. Kötelező operandus. Pontosán egyet kell magadnunk. Ez lehet egy a felhasználó által választható azonosító, literál vagy objektumnév.
- 4/a, 4/b Az utasítás a következő sorban folytatódik.
5. Kötelező kulcsszó, pontosan így kell írni.
6. Az egész klauzula elhagyható, de ha megadjuk, akkor formátumát a főútvonal alatti mellékút határozza meg. Ekkor KULCSSZÓ-4 megadása kötelező.
7. Ha megadjuk a 6. klauzulát, akkor a KULCSSZÓ-4 után a „kifejezés”-t legálább egyszer meg kell adnunk és akárhányszor megismételhetjük. Az egyes kifejezéseket vesszővel választjuk el egymástól.
8. Kifejezés részletes kifejtése. Az áttekinthetőség érdekében ábrázoltuk külön (rész)diagrammon.²

9.3 Az SQL utasítások csoportosítása

Az SQL utasításokat négy nagy csoportba osztjuk:

- adatleíró utasítások,
- adatkezelő utasítások,
- vezérlő utasítások,
- egyéb utasítások.

Az *adatleíró utasítások* közé tartoznak az adatbázis szerkezetét leíró és módosító utasítások. Adatbázis területek kijelölésére, táblázatok, nézetek, indexek létrehozására, módosítására, megszüntetésére szolgálnak.

Legfontosabbak közülük a

CREATE	(létrehoz),
DROP	(megszüntet),
ALTER	(módosít)

típusú utasítások.

Az SQL adatleíró utasításait (a Data Definition Language megfelelőjét) a 10. fejezetben ismertetjük.

Az *adatkezelő utasítások* meglevő táblázatokból, nézetekből választanak ki, törölnek, módosítanak meglevő sorokat, új sorokat írnak be.

² A fenti példa, mely lényegét tekintve egy ismert rendszer kézikönyvéből származik, illusztrálja ennek a jelölésnek nem teljesen pontos definiálási módját. Mereven értelmezve a szintaxis megengedne egy kifejezésében pl. két + jelet is egymás után, aminek nyilván nincs sok értelme. Ilyen esetekben (hibásan, de remélhetőleg nem teljesen alaptalanul) a felhasználó józan eszére van bízva, hogy az értelmetlen, és kizárólag formailag lehetséges változatokat kiszűrje (legrosszabb esetben a rendszer hibaüzenetének megértése után).

Legfontosabbak közülük a

SELECT	(kiválaszt),
DELETE	(töröl),
UPDATE	(módosít),
INSERT	(beír)

típusú utasítások.

Ezek még kiegészülnek a programnyelvbe való beillesztéshez szükséges további utasításokkal.

Az SQL adatkezelő utasításait (a Data Manipulation Language megfelelőjét) a 11. fejezetben tárgyaljuk.

A *vezérlő utasítások* segítségével engedélyezhetjük különböző SQL objektumok használatát, biztosíthatjuk több lépésből álló utasításcsoport konzisztens végrehajtását, vagy bármely hiba esetén a kiindulási állapotba való visszaállítást.

A legfontosabb vezérlő utasítás típusok:

GRANT	(jogosultság megadása)
REVOKE	(jogosultság visszavonása),
COMMIT	(utasítások eredményének véglegesítése),
ROLLBACK	(eredeti állapot visszaállítása),
LOCK	(adatbázis objektumok elzárása),
WHENEVER	(hibakezelés programban),
SET	(rendszer paraméterek beállítása).

Az SQL vezérlőutasításaival a 12. fejezetben foglalkozunk.

Az *egyéb utasítások* igen erősen rendszerfüggőek. Ide soroljuk például a rendszer állapotáról információt adó, de SQL-ben is használható operátori parancsokat, SQL utasítások programbeli előkészítését és konkrét megvalósítását, az adatelérési utakat leíró EXPLAIN utasítást, SQL utasítások tárolására és tárolt utasítások végrehajtására szolgáló utasításokat, az adatbázis leírásának megértéséhez, a katalógusba megjegyzések beírására szolgáló utasításokat stb. Ezeket röviden a 13. fejezetben foglaljuk össze.

Adatleíró utasítások

10.1 Általános jellemzők

Az SQL nyelv adatleíró utasításainak segítségével

- táblázatokat,
- nézeteket,
- táblázatokon indexeket

hozzátunk létre (CREATE TABLE / VIEW / INDEX), törölhetünk (DROP) vagy módosíthatunk (ALTER).

Ezen kívül, attól függően, hogy az adatbázist fizikailag hogyan helyezzük el a lemezeken, és milyen önállóan kezelhető, logikailag összefüggő részekből (táblaterületek, TABLESPACE; adatbázis (rész)területek, DBSPACE; kollekció, COLLECTION)) lehetőség van ezeknek meghatározására is.

Míg táblázatok, nézetek, indexek létrehozására minden SQL-en alapuló adatbázis-kezelő rendszerben¹ vannak megfelelő utasítások, ezek módosítását már nem mindegyik engedi meg.

Az adatbázis kisebb, önálló részeinek definiálása, tárolóhoz való hozzárendelése már igen erősen rendszerfüggő. Ezért ezzel ezen általános ismertetés keretében nem tudunk foglalkozni. Ezeket az utasításokat általában csak az adatbázis-felügyelő adhatja ki.

Bármilyen objektumot csak olyan felhasználó hozhat létre, szüntethet meg, vagy módosíthat, akinek erre jogosultsága van. Az adatbázis-felügyelő mindig rendelkezik minden ilyen jogosultsággal. Ezen kívül kaphatnak erre jogosultságot egyes felhasználók is vagy explicite, tevékenység és név szerint felsorolva, vagy pedig mint egy adott csoport, akár az összes felhasználó csoportjának (PUBLIC) tagjaként, vagy pedig úgy, hogy az adatbázis egy meghatározott része felett teljes joggal ők rendelkeznek.

Általában egy objektum létrehozója jogosult annak megszüntetésére és módosítására is, azokra nézeteket, indexeket, elsődleges és idegen kulcsot is definiálhat.

¹ A továbbiakban ezeket a rövidség kedvéért egyszerűen SQL rendszernek, vagy SQL rendszereknek fogjuk nevezni.

Más felhasználók—az adatbázis-felügyelő kivételével—ezeket csak akkor tehetik meg, ha ezt számukra az objektum tulajdonosa megengedte (vagy név szerint, vagy PUBLIC-ként mindenki számára).

10.2 Táblázatok létrehozása (CREATE TABLE utasítás)

A táblázat létrehozásához meg kell adnunk a táblázat nevét és oszlopainak pontos definícióját. Ezen kívül megadhatjuk, hogy az adatbázis melyik területén helyezkedik el, kit kell a táblázat létrehozójának tekinteni, azaz ki legyen a tulajdonosa (általában a létrehozó és a tulajdonos ugyanaz, de—mint már említettük—az adatbázis-felügyelő létrehozhat más felhasználók számára is táblázatokat). Ennek különösen akkor van jelentősége, ha a táblázatok létrehozásának a jogát korlátoztuk. Az ilyen jogosultsággal nem rendelkezők részére mindenképpen az adatbázis-felügyelőnek kell létrehoznia a táblázatokat, természetesen a felhasználó logikai definíciója alapján. Ugyancsak definiálhatjuk létrehozáskor a táblázat elsődleges kulcsát és idegen kulcsait is. Amennyiben a nem kötelező paraméterek közül valamelyiket nem adjuk meg a CREATE TABLE utasításban, akkor ehelyett a rendszer automatikusan valamilyen ésszerű alapértelmezéssel dolgozik. (Pl. az objektum tulajdonosa a létrehozója lesz, nincsen idegen kulcs stb.)²

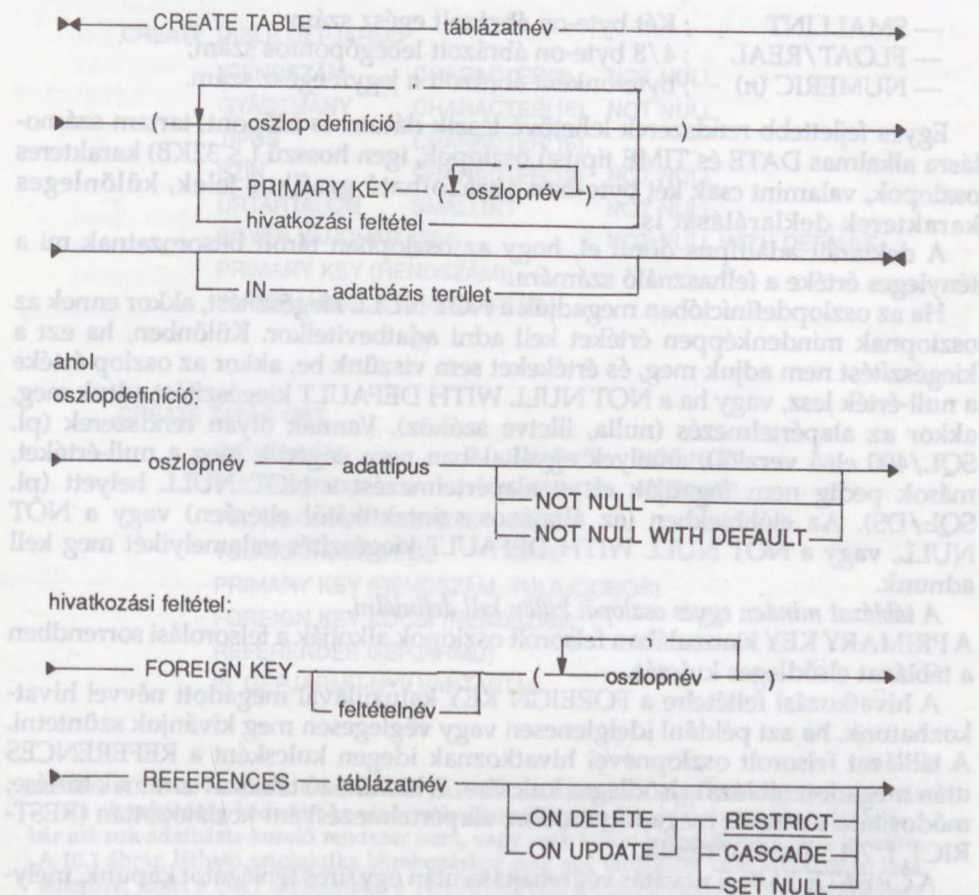
A táblázatok létrehozására szolgáló utasítás általános formája a 10.1 ábrán látható.

Az utasítás neve CREATE TABLE. Ezt, valamint a létrehozandó táblázat nevét kötelező megadni. Ez a táblázatrév az egész adatbázisban egyedi kell legyen. A legtöbb olyan rendszer, amelyik több felhasználót is enged egyidejűleg dolgozni, a táblázatok nevét két részből építi fel. Az első rész a táblázat tulajdonosának a neve. Ez az a felhasználó, aki a táblázatot létrehozta, vagy akinek a nevében—általában akkor, ha nem volt jogosultsága—az adatbázis-felügyelő végezte ezt el. Ha ezt nem adjuk meg a CREATE TABLE utasításban, akkor a rendszer alapértelmezésként azt tételezi fel, hogy az lesz a tábla tulajdonosa, aki a CREATE TABLE utasítást kiadta. A név második összetevője az a név, amelyet a létrehozó ad a táblázatnak. A fentiekből következik, hogy akárhány felhasználónak lehet pl. OSZTÁLYOK nevű táblázata, de mindegyiküknek csak egy ilyen lehet.

A táblázat oszlopneveit a felhasználó ismét szabadon választhatja meg. Csak arra kell ügyelnie, hogy egy adott táblázatban ne legyen két azonos nevű oszlop. Az oszlop neve mellett meg kell adnia, hogy az az adatokat milyen formában tartalmazza.

Minden valamirevaló adatbázis-kezelő rendszer támogatja az alábbi adattípusokat (ezek nevét az utasításban általában rövidíthetjük is).

² Megjegyezzük, hogy az SQL nyelv önmagában még az elsődleges kulcsot sem követeli meg. Így létrehozhatnánk olyan relációt is, amelyiknek egyáltalában nincsen elsődleges kulcsa. (A DB2-ben először így is hozzuk létre, és csak miután egyedi indexet definiáltunk a leendő elsődleges kulcsmező(k)re, definiálhatjuk utólag az elsődleges kulcsot.) Természetesen az elsődleges kulcs nélküli relációkat tartalmazó adatmodell csak a nevében relációs. Bár formailag az SQL nyelv segítségével (sajnos) dolgozhatunk rajta, a relációs modell igazi előnyét ez esetben csak korlátozottan használhatjuk ki.



10.1 ábra Táblázat létrehozó utasítás általános formája

- **CHARACTER** (n) ; pontosan n karakterből álló jelsorozat (általában $n < 256$);
- **INTEGER** ; 4 byte-on ábrázolt bináris egész szám;
- **DECIMAL** (p, q) ; összesen p jegyből álló decimális szám. Ezekből q áll a tizedespont/vessző után. Általában $p \leq 15$, (egyes újabb rendszerekben $p \leq 31$).

A fentiekén kívül a legtöbb rendszer ismeri még a további adattípusokat is

- **VARCHAR** (n) ; változó hosszúságú, maximum n karakterből álló jelsorozat. Alkalmazása különösen akkor célszerű, ha az oszlopban tárolandó tényleges szöveges információtartalom hossza igen erősen változik. Általában $n \leq 254$;

- SMALLINT ; Két byte-on ábrázolt egész szám;
- FLOAT/REAL ; 4/8 byte-on ábrázolt lebegőpontos szám;
- NUMERIC (*n*) ; byte-onként ábrázolt *n* jegyű egész szám.

Egyes fejlettebb rendszerek lehetővé teszik dátum és időpont/tartam számolásra alkalmas DATE és TIME típusú oszlopok, igen hosszú ($\leq 32\text{KB}$) karakteres oszlopok, valamint csak két byte-ban ábrázolható grafikus jelek, különleges karakterek deklarálását is.

A deklarált adattípus dönti el, hogy az oszlopban tárolt bitsorozatnak mi a tényleges értéke a felhasználó számára.

Ha az oszlopdefinícióban megadjuk a NOT NULL kiegészítést, akkor ennek az oszlopnak mindenképpen értéket kell adni adatbevitelkor. Különben, ha ezt a kiegészítést nem adjuk meg, és értékeket sem viszünk be, akkor az oszlop értéke a null-érték lesz, vagy ha a NOT NULL WITH DEFAULT kiegészítést adtuk meg, akkor az alapértelmezés (nulla, illetve szóköz). Vannak olyan rendszerek (pl. SQL/400 első verziói), amelyek egyáltalában nem engedik meg a null-értéket, mások pedig nem fogadják el az alapértelmezést a NOT NULL helyett (pl. SQL/DS). Az előbbiekben (az általános szintaktikától eltérően) vagy a NOT NULL, vagy a NOT NULL WITH DEFAULT kiegészítés valamelyikét meg kell adnunk.

A táblázat minden egyes oszlopát külön kell definiálni.

A PRIMARY KEY klauzulában felsorolt oszlopok alkotják a felsorolási sorrendben a táblázat elsődleges kulcsát.

A hivatkozási feltételre a FOREIGN KEY klauzulával megadott névvel hivatkozhatunk, ha azt például ideiglenesen vagy véglegesen meg kívánjuk szüntetni. A táblázat felsorolt oszlopnevei hivatkoznak idegen kulcsként a REFERENCES után megadott táblázat elsődleges kulcsára. A hivatkozó táblázat sorainak törlése, módosítása az utána megadott módon, alapértelmezésként korlátozottan (RESTRICT, l. 7.4.3 pont) történik.

A CREATE TABLE utasítás végrehajtása után egy üres táblázatot kapunk, melybe pl. az INSERT adatkezelő utasítással, vagy a táblázatbetöltő szolgáltató programmal vihetünk be adatokat.

A 10.2 ábrán a 7.2 ábrán bemutatott GÉPJÁRMŰ és G&T relációk egy lehetséges definiálási módját láthatjuk.

Megjegyezzük, hogy az idegen kulcsokat vagy az adatbázis önállóan is kezelhető logikai egységekre bontását és a táblázatoknak ebbe való irányított elhelyezését igen sok rendszer még nem támogatja. Más, fejlettebb rendszerek, mint pl. a DB2 viszont még ennél is több lehetőséget nyújtanak. Ebben lehetőség van egy másik táblázat teljes definíciójának átvételére (LIKE klauzula), egyes oszlopok adatainak átkonvertálására (FIELDPROC programnév), sorok adatainak átalakítására és ellenőrzésére (EDITPROC és VALIDPROC programnév), ahol az átalakító illetve ellenőrző programot a felhasználó saját maga írhatja meg és ez minden, a táblázatra vonatkozó adatkezelő művelet végrehajtásakor automatikusan aktivizálódik. Ezen kívül mód van még annak meghatározására is, hogy a táblázaton végrehajtott műveletek utólagos ellenőrizhetőség céljából auditálásra kerüljenek-e (AUDIT). A CREATE TABLE utasításnak ezen bővített lehetőségeit a DB2 adatbázis-kezelő rendszerben a 10.3 ábrán láthatjuk.

CREATE TABLE GÉPJÁRMŰ

(RENDSZÁM	CHARACTER(6)	NOT NULL,
GYÁRTMÁNY	CHARACTER(16)	NOT NULL,
TÍPUS	CHARACTER(16),	
SZÍN	CHARACTER(12)	NOT NULL,
ÚRTARTALOM	SMALLINT	NOT NULL,
EGYÉB VARCHAR (254)		NOT NULL WITH DEFAULT

PRIMARY KEY (RENDSZÁM))
IN GÉPJÁRMŰ-NYILVÁNTARTÁS.

a)

CREATE TABLE G&T

(RENDSZÁM	CHAR (6)	NOT NULL,
TULAJDONOS	CHAR (30)	NOT NULL,
TULAJDONJOG-KEZDETE	DATE,	
TULAJDONJOG-VÉGE	DATE	

PRIMARY KEY (RENDSZÁM, TULAJDONOS)
FOREIGN KEY KOCSI (RENDSZÁM)
REFERENCES GÉPJÁRMŰ)
IN GÉPJÁRMŰ-NYILVÁNTARTÁS

b)

10.2 ábra A 7.1 és 7.2 ábrák G&T és GÉPJÁRMŰ táblázatainak (egy lehetséges) definiálása.

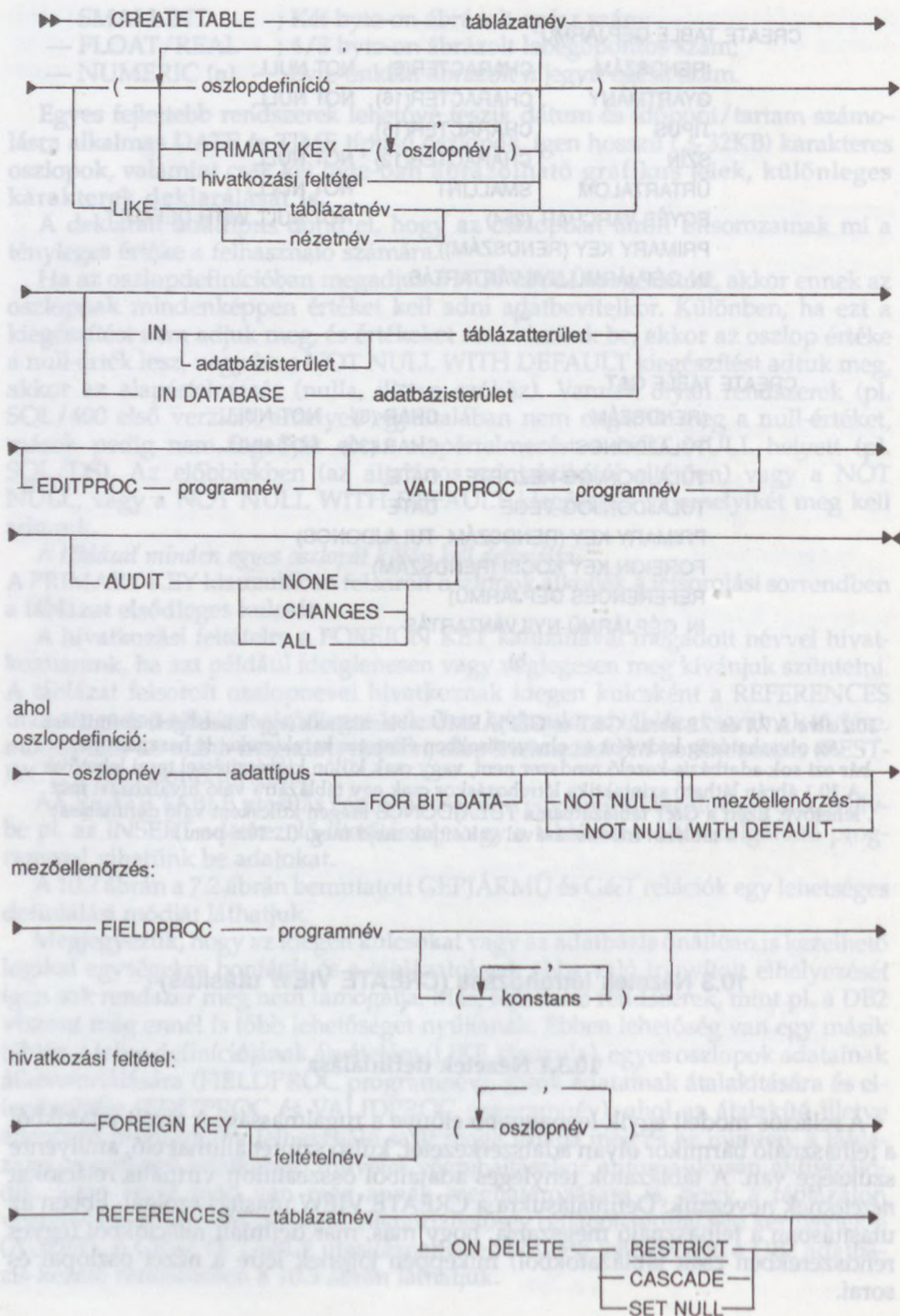
Az olvashatóság kedvéért az elnevezésekben ékezetes karaktereket is használtunk, bár ezt sok adatbázis-kezelő rendszer nem, vagy csak külön kiegészítéssel teszi lehetővé.

A 10.1 ábrán látható szintaktika létrehozáskor csak egy táblázatra való hivatkozást tesz lehetővé. Ezért a G&T táblázatban a TULAJDONOS idegen kulcsként való definiálását a táblázat módosításával valósítjuk majd meg. (L. 10.6 pont)

10.3 Nézetek létrehozása (CREATE VIEW utasítás)

10.3.1 Nézetek definiálása

A relációs modell egyik legnagyobb előnye a rugalmasság. A tárolt adatokból a felhasználó bármikor olyan adatszerkezetet, külső szintet állíthat elő, amilyenre szüksége van. A táblázatok tényleges adataiból összeállított virtuális relációkat *nézeteknek* nevezzük. Definiálásukra a CREATE VIEW utasítás szolgál. Ebben az utasításban a felhasználó megszabja, hogy más, már definiált relációkból (egyes rendszerekben csak táblázatokból) miképpen jöjjenek létre a nézet oszlopai és sorai.

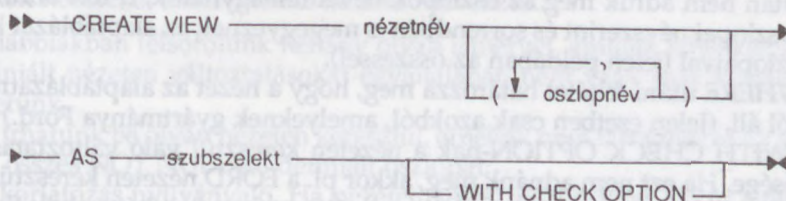


10.3 ábra Táblázatlétrehozó utasítás szintaxis diagramja DB2-ben.

Nézetek definiálása és használata különösen akkor célszerű, ha

- táblázatoknak csak bizonyos részeit, meghatározott sorait és/vagy oszlopait kívánjuk hozzáférhetővé tenni. Ebben az esetben a nézetek az *adatvédelmi funkciókat* segítik;
- egyszerűbbé akarjuk tenni összetett feladatok megfogalmazását. Táblázatok (nézetek) gyakori kombinációira nézetet definiálunk, melyekkel (bizonyos megszorításokkal, l. később) ugyanúgy, ugyanolyan egyszerűen dolgozhatunk, mint egyedi táblázatokkal;
- az adatok fizikailag már mind bent vannak az adatbázisban, de most *más csoportosításban* (is) kellene. Egy új nézet definiálásával— anélkül, hogy az eddigi adatokon és programokon bármit is módosítani kellene—, azonnal létrehozhatjuk az új (logikai) objektumot.

A CREATE VIEW utasítás formátumát a 10.4 ábrán láthatjuk.



10.4 ábra Nézetet létrehozó utasítás szintaxis diagrammja

A nézet definíciójában szereplő szubszelekt határozza meg, milyen táblázatokból, milyen oszlopokat és mely sorokat választunk ki.³

A nézetnek és oszlopainak nevére ugyanazok a feltételek érvényesek, mint a táblázatra.

Természetesen nézetet csak az hozhat létre, aki jogosult a nézetek alapjául szolgáló, a szubszelektben felsorolt összes reláció olvasására.

10.3.2 Példák nézetek létrehozására

Néhány jellemző példa nézet létrehozására.

1. A nézet egy táblázatnak csak meghatározott oszlopait tartalmazza (projekció).

```

CREATE VIEW AUTOMÁRKÁK (GYÁRTMÁNY,TÍPUS)
AS SELECT GYÁRTMÁNY, TÍPUS FROM GÉPJÁRMŰ
  
```

³Didaktikailag éppen ezért talán helyesebb volna, ha a CREATE VIEW utasítást később, a SELECT utasítás után tárgyalnánk. Ezzel viszont megtörnénk az adatleíró utasítások egységét, a táblázatok és nézetek logikai szinten való (csaknem) teljes azonosságát. A most következő példában olyan egyszerű kiválasztási feltételeket adunk meg, melyek a SELECT utasítás pontos ismerete nélkül is követhetők. Amennyiben azonban ez az olvasónak mégis gondot okozna, akkor a 11.2 pont elolvasása után térjen vissza erre a részre.

A nézet (logikailag) a 10.2 ábrán definiált GÉPJÁRMŰ táblázatból csak a GYÁRTMÁNY és TÍPUS adatokat tartalmazza.⁴ Aki tehát ezzel a relációval dolgozik, az ebből láthatja, milyen gyártmányú és típusú gépkocsik vannak az adatbázisban, de hogy ezek motorjának mekkora az úrtartalma, milyen rendszámmal futnak stb. az nem hozzáférhető a számára.

2. A nézet a táblázatnak meghatározott sorait tartalmazza (korlátozás). Például a GÉPJÁRMŰ táblázatból az alábbi nézettel csak a Ford típusú gépkocsikkal dolgozhatunk.

```
CREATE VIEW FORD
AS SELECT * FROM GÉPJÁRMŰ
WHERE GYÁRTMÁNY = 'Ford'
WITH CHECK OPTION
```

A fenti példában a SELECT * jelölés azt jelenti, hogy a GÉPJÁRMŰ táblázat minden oszlopát kiválasztjuk, ezeket mind tartalmazni fogja a nézet. Mivel a nézet neve után nem adtuk meg az oszlopok nevét (elhagyhatók, l. 10.4 ábra), ezért a nézet oszlopoi névszerint és sorrendben is megegyeznek az alaptáblázat kiválasztott oszlopaival (jelen példában az összessel).

A WHERE utáni feltétel határozza meg, hogy a nézet az alaptáblázatnak mely soraiból áll. (Jelen esetben csak azokból, amelyeknek gyártmánya Ford.)

A WITH CHECK OPTION-nak a nézeten keresztül való változtatásnál van jelentősége. Ha ezt nem adnánk meg, akkor pl. a FORD nézeten keresztül történő módosítással minden további nélkül megváltoztathatnánk bármelyik Ford típus gyártmányát mondjuk Opelra. Természetesen az így megváltoztatott sor adatait nem látjuk a FORD nézetben, de az adatbázisban benne van.

Ha a nézetet a WITH CHECK OPTION kiegészítéssel definiáltuk, akkor az adatbázis-kezelő rendszer automatikusan ellenőríz minden változtatást (módosítást, beírást). Azokat a változtatásokat, amelyek olyan sort eredményeznének, amelyik nem felel meg a nézet definíciójának, hibajelzés kíséretében visszautasítja.⁵

3. Hasonlóképpen

- kiválaszthatunk egy nézet definiálásakor egy táblázatból meghatározott oszlopokat (csak ezeket soroljuk fel a SELECT utáni listában) és sorokat (WHERE feltételt is megadunk);
- kiválaszthatunk aggregátumokat, például adott oszlopok csoportonkénti összegét, maximumát stb.;
- vagy több táblázatból egyesítés révén összeállított adatokat.

Ismételten hangsúlyozzuk, hogy a nézetek nem tartalmaznak adatokat. Ezért nem is lehet a nézetre sem elsődleges, sem idegen kulcsot definiálni, sőt indexet sem.

⁴ Fizikailag természetesen egyetlen adat sincsen a nézetben. Az adatokat ténylegesen a nézet alaptáblázata(i), az(ok) a táblázat(ok), amely(ek)re a nézetet definiáltuk, tartalmazzá(k).

⁵ Sajnos igen sok implementációban még nincs meg az a lehetőség (így pl. az SQL/400, vagy az SQL/DS 2. release-ében sem) és ezért a nézeteknek adatvédelmi célokra való felhasználása ezeknél illuzórikus, illetve csak olvasásra való engedéllyel van értelme.

A nézet definíciója bekerül a katalógusba. Minden, a nézetre vonatkozó műveletnél az adatbázis-kezelő rendszer először kiválasztja az alaptáblázat(ok)ból a nézet definíciójának megfelelő adatokat, és ezután ezeken végzi el a kért műveletet.

10.3.3 Korlátozások

Ha a nézetek alapjául szolgáló táblázatokot módosítjuk, akkor ennek eredménye azonnal tükröződik a nézetekben is.

Mi történik azonban, ha egy nézet „adatait” kívánjuk megváltoztatni?

Több eset lehetséges:

- A változtatást végrehajthatjuk. Az alaptáblázat adatai módosulnak.
- A változtatást azért nem hajthatjuk végre, mert annak eredménye ellenkező a nézet definíciójával és a WITH CHECK OPTION-t megadtuk.
- A rendszer semmilyen változtatást nem enged meg a nézeten keresztül. Az ilyen nézeteket *csak olvasható nézet*nek nevezzük.

Az alábbiakban felsorolunk néhány olyan ismérvet, aminek következtében az így definiált nézeten változtatásokat egyáltalában nem vagy csak korlátozottan végezhetünk.

a) Nem írhatunk be olyan nézeten keresztül, amelyik nem tartalmazza az alaptáblázat összes NOT NULL-lal definiált oszlopát.

Ez a korlátozás nyilvánvaló. Ha nézeten keresztül írunk be, akkor a nézet definíciójában nem szereplő oszlopok értéke az alaptáblázatban az új sorokban null-érték, vagy az ahelyetti alapértelmezés (NOT NULL WITH DEFAULT) lesz.

b) Nem lehet változtatni, ha a nézet definíciójában valamilyen aggregátum pl. oszlopértékek összege, SUM (oszlop), vagy maximuma, MAX (oszlop) szerepel. A rendszer nyilvánvalóan nem tudja eldönteni, hogy az alaptáblázat melyik sorát kell módosítani, milyen sort kell beírni, hogy az összeg, vagy a maximum továbbra is helyes maradjon.

c) Két táblázat egyesítésére épülő nézetnél, ha az egyesítésre szolgáló mező nem egyedi *mindkét* táblázatban, akkor nem lehet egyértelműen eldönteni, hogy a nézetből való törléskor vagy módosításkor az alaptáblázatokból melyik sort vagy sorokat kell törölni vagy módosítani.

Például a 10.2 ábra GÉPJÁRMŰ és G&T táblázatainak egyesítésével egy olyan nézetet hozunk létre, amelyik a RENDSZÁM, TULAJDONOS, GYÁRTMÁNY oszlopokat tartalmazza. Ha ebből a nézetből kitörölnének a Lakinger Béla tulajdonosú sorokat, nem egyértelmű, hogy mi történjen a G&T és GÉPJÁRMŰ táblázatok adataival (l. 7.1 és 7.2 ábrán).

G&T-ből törölnének nyilvánvalóan a Lakinger Béla tulajdonosú DY9680 és V24034 rendszámú sorok. De GÉPJÁRMŰ-ből mindkét rendszámot kell-e törölni, vagy csak azt, ahol a jelenlegi tulajdonost töröltük. Ha igen, akkor ezt honnan tudja az adatbázis-kezelő rendszer? Ha mindkettőt, akkor ez tovább változtatja G&T-t, mert itt a Kiss János-1 tulajdonosú sorban a törölt oszlopérték idegen kulcs.

Mivel az ilyen kérdésekre az adatszerkezetből nem lehet egyértelmű választ adni, az ilyen, egyesítésre épülő nézetekben nem lehet módosítani. Technikai okokból a legtöbb SQL rendszer ennél erősebb korlátozással él. Csak olvasha-

tónak tekinti az összes olyan nézetet, amelyik két vagy több táblázat egyesítésére épül fel, valamint azokat is, amelyeknél csoportképzéssel (GROUP BY, l. 11.2.2 pont) történik a kiválasztás.

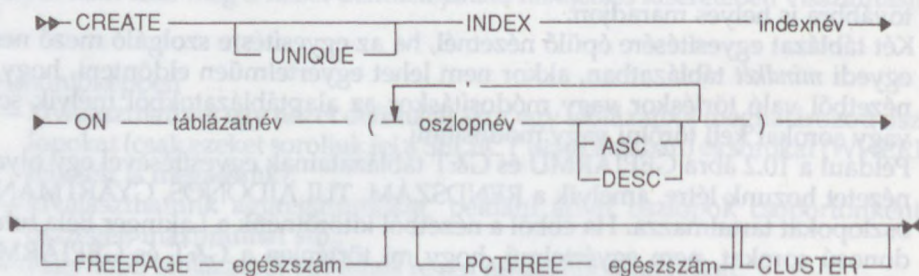
Szokásos korlátozási feltétel még, hogy a sorokat a nézetbe nem lehet rendezve kiválasztani (ez érthető, hiszen a nézet is egy reláció, és ebben definíció szerint nincs a soroknak kitüntetett sorrendje).⁶

10.4 Index létrehozása (CREATE INDEX utasítás)

A relációs adatmodellben az adatok eléréséhez a felhasználó nem adhatja meg az *elérési utat*, sehol nem hivatkozhat indexekre, még akkor sem, ha tudja, hogy ilyenek léteznek. A rendszer maga dönti el, hogy használ-e indexeket és melyeket, természetesen csak akkor, ha vannak.

A felhasználónak és az adatbázis-felügyelőnek csak arra van lehetősége, hogy az olyan táblázatokra, melyeket célszerű lehet indexek alapján feldolgozni, indexeket definiál a megfelelő oszlopokra, oszlopkombinációkra. Amennyiben az elérés nézetén keresztül történik, az SQL a nézet alaptáblázataira épült indexeket veszi igénybe.

Az index létrehozásakor (10.5 ábra) meg kell adnunk az index nevét, valamint azt, melyik táblázat melyik oszlopaira épüljön fel. Ezenkívül megadhatjuk, hogy az index-oszlop(ok) értéke a táblázatban egyedi legyen-e (UNIQUE, alapértelmezésben megengedi a duplikátumot is), növekvő (alapértelmezés, ASC) vagy csökkenő és sűrűsödő (CLUSTER) legyen-e. A FREEPAGE és PCTFREE klauzulákban megadhatjuk, hogy betöltéskor mennyi helyet hagyjon szabadon a rendszer a későbbi bővítések beszurására.



10.5 ábra Indexet létrehozó utasítás szintaxis diagrammja

⁶ A pontosság kedvéért hozzátesszük, hogy a nézet kiválasztási definíciója nem tartalmazhat UNION-t sem. Ez azonban a szintaxisból is következik, a szubszelektékben nem használható az unió művelete (l. 11.2.2 pont).

Az elsődleges kulcs egyben mindig egyedi, növekvő indexként is tekinthető. Egyes rendszerek az elsődleges kulcs definiálásával automatikusan létrehozzák az indexet is, másoknál előbb az indexet kell definiálni.

A 10.2b ábrán definiált G&T táblázatra például a 10.6 ábrán látható indexeket mindenképpen célszerű létrehozni. Az, hogy a TULAJDONVISZONY-KEZDETE és -VÉGE oszlopokra definiálunk-e vagy sem, a várható alkalmazásoktól függ. Az sem baj, ha ebben a kérdésben az adatbázis létrehozásakor nem jól döntöttünk, mert az indexeket utólag is bármikor létrehozhatjuk, vagy ha fölöslegesnek tartjuk, akkor megszüntethetjük.

```
CREATE UNIQUE INDEX RSZ-TUL ON G&T
    (RENDSZÁM, TULAJDONOS)
```

```
CREATE INDEX RSZ ON G&T (RENDSZÁM) CLUSTER
```

```
CREATE INDEX TUL ON G&T (TULAJDONOS)
```

10.6 ábra A G&T táblázatra (10.2b ábra) definiált indexek (Feltételeztük, hogy a G&T táblázathoz leggyakrabban a RENDSZÁM alapján kell majd hozzáférni, ezért eszerint sűrűsödve akarjuk a sorokat tárolni.)

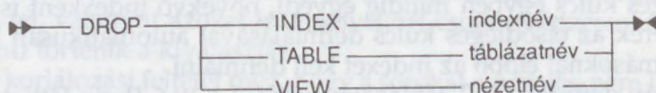
Az index mindig egy táblázat elérésére szolgál. Egy táblázatra akárhány index definiálható. Egy index általában max. 16 oszlopból állhat, és az oszlopok együttes hossza is korlátozott (rendszerint 120–254 byte között). Indexeket bármikor definiálhatunk, így létrehozhatunk utólag is olyan táblázatra, melyben már vannak adatok. Ilyenkor az egyedi index csak akkor készül el, ha a meglevő adatok megfelelnek ennek a követelménynek. Amennyiben lenne két azonos indexértékű sor, a rendszer hibaüzenetet küld, és az indexeket nem hozza létre. Az egyedi index-értékeket menetközben is ellenőrzi minden változtatásnál. Amennyiben egy ilyen művelet duplikált indexértékhez vezetne, azt a rendszer hibaüzenet kíséretében visszautasítja.

Általában olyan oszlopra, amelyik index vagy index része, ne engedjük meg a null értéket (definiáláskor NOT NULL). Ha duplikált kulcsnál mégis megengedjük, akkor a null-értékek növekvő indexnél legelőlről, csökkenőnél a végére kerülnek.

A CREATE INDEX utasítás az utasításban megadott táblázat meglevő értékeire egy B⁺ fát (l. 4.7.5 pont) hoz létre, és olyan további utasításokat generál, melyek a táblázat minden változtatásakor aktualizálják ezt.

10.5 Táblázat, nézet, index megszüntetése (DROP utasítás)

Meglevő táblázatot, nézetet, indexet a DROP utasítással szüntethetünk meg. Az utasítás szintaxis diagramját a 10.7 ábrán láthatjuk. (Megjegyezzük, ez a diagram nem teljes, a DROP utasítással sok minden egyebet, például szinonimákat, táblázat és adatbázis területeket is megszüntethetünk.)



10.7 ábra Táblázat, nézet index megszüntetésére szolgáló utasítás
(A DROP utasítással ezen kívül más objektumokat is megszüntethetünk.
Ezt nem tüntettük föl a diagrammon.)

A DROP utasítás hatására a meglevő objektumok törölődnek a katalógusból. Táblázat megszüntetése automatikusan törli a táblázat összes adatát, a reá hivatkozó nézeteket, indexeket, idegen kulcsokat. Természetesen, azok a korábban lefordított, SQL utasításokat tartalmazó programok, amelyek ezzel a táblázattal dolgoztak, nem fognak működni. Ugyanez vonatkozik a programokra nézetek törlésekor. Ha viszont olyan indexet törölünk, amelyet az fölhasznált az eléréshez, akkor—mivel az elérési út explicit sehol sem szerepel—a legközelebbi futáskor a program más elérési utat alakít ki, és zavartalanul végrehajtott.

Ha viszont egy táblázatra új indexet definiálunk, akkor az ezzel a táblázattal dolgozó, az index elkészítése előtt lefordított programok csak akkor veszik figyelembe az optimális elérési úthoz ezt az új indexet, ha újra lefordítjuk őket. Így csupán új indexek generálásával már meglevő, lefordított SQL utasításokat tartalmazó programok hatékonyságát nem tudjuk javítani. Ehhez a programokat is mind újra le kell fordítanunk.

10.6 Táblázat módosítása (ALTER TABLE utasítás)

Egyes fejlettebb rendszerek megengedik meglevő táblázatok bizonyos mértékű módosítását is. Ez rendszerint egy vagy több új oszlopnak a sorok végére való hozzáírását, elsődleges és idegen kulcsok definiálását, állandó vagy ideiglenes megszüntetését illetve újra aktivizálását jelenti.

Az újonnan hozzáadott oszlop értéke a már meglevő sorokban a null-érték vagy annak alapértelmezése lesz, az új oszlop definíciójától függően.

Meglevő oszlopot megszüntetni, az adattípust megváltoztatni (pl. decimal ábrázolás helyett integer) általában nem lehet.

Elsődleges és idegen kulcsot meglevő adatokra csak akkor lehet létrehozni, ha azok a feltételnek megfelelnek. Ilyen kulcsok—elsősorban az idegen kulcs—ideiglenes felfüggesztésére hivatkozási ciklus esetén lehet szükségünk, ha az idegen kulcs értékét is kötelező megadni. Erre példa az alábbi (minden oszlop definíciója NOT NULL):

ALKALMAZOTT (ALK-KÓD,NÉV,OSZTKÓD)

ELSŐDLEGES KULCS (ALK-KÓD)

IDEGEN KULCS (OSZTKÓD) HIVATKOZIK OSZTÁLY-RA

OSZTÁLY (OSZTKÓD,OSZTNÉV,VEZETŐKÓD)

ELSŐDLEGES KULCS (OSZTKÓD)

IDEGEN KULCS (VEZETŐKÓD) HIVATKOZIK ALKALMAZOTT-RA.

Az így definiált két üres táblázatba egy sort sem tudunk bevinni, hiszen ALKALMAZOTT sort nem írhatunk be, ha nincsen benne olyan OSZTKÓD, amely már benne van az OSZTÁLY táblázatba. Ebbe viszont csak olyan osztályt vehetünk fel, melynek VEZETŐKÓD-ja szerepel az ALKALMAZOTT-ban, amely még üres.

Ezt az ellentmondást úgy oldhatjuk fel, hogy megszüntetjük (legalább is az osztályok betöltésekor) az idegen kulcsot, majd az adatok bevitele után újra definiáljuk. Természetesen, ha az idegen kulcs ellenőrzése nélkül ellentmondó, hibás adatokat töltöttünk volna be, akkor ezeket előbb ki kell javítanunk, addig az idegen kulcsokat nem tudjuk visszaállítani.

Egyes rendszerekben az ALTER utasítással lehetőségünk van az indexek és az adatok közötti bővítésre fenntartott hely nagyságának módosítására, valamint az egyedejű hozzáférést szabályozó záruk (l. 12.4 pont) beállításának módosítására is. Sok egyszerűbb adatházzi beállításokhoz is.

Sok egyszerűbb adatbázis-kezelő rendszer nem ismeri az ALTER utasítást. Ezekben, ha mégis szükség van egy táblázat bővítésére (eredeti nevének megtartása mellett), akkor az adatokat kimentjük, az eredeti táblázatot töröljük (nem lehet két reláció azonos névvel), létrehozzuk a tőle csak az új oszlopban különböző új táblázatot az eredetivel azonos néven, majd az adatokat ebbe visszatöltjük.

Az adatkezelő utasítások segítségével megtehető műveletekkel, nevezetesen választgatunk ki adatokat (SELECT), beszúrunk új sorokat (INSERT), módosítjuk az adatokat (UPDATE) vagy törölhetünk (DELETE) megcélzott sorokat.

Adatkezelő utasítások

11.1 Általános ismertetés

Az adatkezelő utasítások segítségével meglevő táblázatokból, nézetekből választunk ki adatokat (SELECT), írhatunk be új sorokat (INSERT), módosíthatunk (UPDATE) vagy törölhetünk (DELETE) meglevő sorokat.

Egy utasítás mindig egy vagy több, egyértelműen megadott táblázatra vagy nézetre vonatkozó konkrét adatkezelő műveletet jelent (például beírást a GÉPJÁRMŰ táblázatba) és annak az utasítás további klauzuláiban megadott paraméterektől függően 0, 1 vagy több sorát érinti.

Az utasítást csak az hajthatja végre, akinek az abban szereplő táblázat(ok)ra és/vagy nézetre vonatkozóan az adott utasítás elvégzésére jogosultsága van. A jogosultság a legtöbb rendszerben nem csak a táblázatra/nézetre szükséges, hanem az elvégzendő műveletre is. Így például lehetséges, hogy A jogosult a G&T táblázat olvasására (SELECT privilégiuma van), de nem változtathat benne semmit (nincs DELETE, INSERT és UPDATE privilégiuma), míg a GÉPJÁRMŰ táblázat adatain minden műveletet elvégezhet.¹

Az adatbázis-felügyelő általában minden reláción bármely műveletet elvégezhet. (Kivételt képeznek katalógusokban bizonyos közvetlen változtató műveletek.) Ugyancsak jogosult erre bármely táblázathoz/nézethez az adott objektum tulajdonosa. A többi felhasználó (a legegyszerűbb személyi számítógépes rendszerek kivételével) csak olyan műveleteket végezhet egy táblázaton/nézeten, amilyenekre annak tulajdonosától engedélyt kapott. Az engedély megadását és visszavonását a jogosultságnál, a GRANT (l. 12.2 pont) és REVOKE (l. 12.3 pont) utasításoknál tárgyaljuk.

Az adatkezelő utasításokat kiadhatjuk párbeszédesen és programnyelvből is, a fogadó nyelv utasításai közé beépítve. A kétfajta formátum egyes utasításoknál

¹Vannak rendszerek (pl. DB2, SQL/DS), melyekben egyes jogosultságokat (pl. UPDATE) akár oszlopszinten is szabályozni lehet. Ekkor a felhasználó a táblázatnak/nézetnek csak meghatározott oszlopait módosíthatja. Más rendszerekben ezt nézet segítségével oldják meg. A felhasználó nem a táblázatra, hanem csak a módosítható oszlopaiból álló nézetre kap UPDATE privilégiumot.

kissé különböző. Itt most a párbeszédés adatkezelő nyelv utasításait ismertetjük. A programba beépíthető utasításoknak ezektől való eltéréseit a 14. fejezetben foglalkozunk össze.

A szintaktikailag helyes adatkezelő utasításokat a rendszer először átalakítja relációs műveletek sorozatára. Utána ellenőrzi, hogy a kérelmezőnek van-e jogosultsága ezek mindegyikének elvégzésére. Amennyiben igen, úgy a rendelkezésre álló információk alapján megállapítja a (szerinte) optimális műveleti sorrendet és adatelérési utakat, és generálja az operációs rendszer számára az ehhez szükséges utasításokat.

Így például egy olyan utasításnak a végrehajtására, hogy válassza ki a GÉPJÁRMŰ táblázatból az összes Ford gyártmányú gépkocsi adatait és ezeket rendszám szerint növekvő sorrendben szolgáltatassa, az alábbi három módszer bármelyike hatékony lehet. Az adatoktól függ, hogy várhatóan melyik lesz az optimális.

- a) Fizikai sorrendben végigolvassa az összes gépjármű rekordot, kiválasztja a Ford gyártmányúakat és elhelyezi egy ideiglenes táblázatban, majd ezt rendezi rendszám szerint.

Ha sem a gyártmányra, sem a rendszámra nincsen index, akkor ez a módszer a leggyorsabb.

- b) Ha a rendszámra van index, de a gyártmányra nincsen, akkor a rendszámindex szerint olvassa be sorban a rekordokat és ezekből csak a Ford gyártmányúakat adja át.

Ebben az esetben általában ez a leggyorsabb módszer, mivel nincsen szükség sem ideiglenes táblázat létrehozására, sem rendezésre. De ha a táblázatban levő rekordok többsége megfelel a kiválasztási feltételeknek, akkor lehet, hogy az a) módszer célravezetőbb. Az adatok rendezése kevesebb időt igényelhet, mint az indexek szerinti szekvenciális olvasásnál a sok lapváltás. Természetesen, ha a rendszámindex sűrűsödő, akkor mindenképpen a b) megoldás a hatékonyabb.

- c) Ha a gyártmányra van index, de a rendszámra nincsen, akkor ezen index alapján kiválasztja a Fordokat, ideiglenesen tárolja, majd az ideiglenes táblázat sorait rendszám szerint rendezi. Ebben az esetben csaknem mindig ez a megoldás a legjobb. Kivételt képezhet a b) ponthoz hasonlóan az a speciális eset, amikor a kiválasztás nagyon nem szelektív, a járművek többsége Ford.

Amennyiben a rendszámra is és a gyártmányra is van index, akkor az, hogy a b) és a c) módszer közül melyik a hatékonyabb, egyrészt attól függ, hogy melyik index sűrűsödő, másrészt, hogy mennyire szelektív a kiválasztási feltétel. Ha a gyártmány szerinti index sűrűsödő, akkor biztosan a c) utat kell választani, de ha a Fordok aránya meghalad egy bizonyos százalékot, a b) szerinti elérés lesz a jobb. Ha egyik index sem sűrűsödő, akkor általában a c) módszer a legeredményesebb.

Látjuk, hogy már egy ilyen egyszerű műveletnél is milyen problémát jelent a (várhatóan) optimális elérési út meghatározása. Mennyi mindent kellene a felhasználónak figyelembe venni, hogy ezt a feladatot lehetőleg jól oldja meg! A hálós rendszereknél ezt az elemzést valóban el is kellett (volna) végeznie. Természetesen kevés adatnál, kis adatbázisokban az időeltérések (abszolút értékben) jelentéktelenek, de nagy adatállományoknál óriásiak lehetnek. Működésképpessé vagy lehetetlenné tehetik az adatbázist.

Éppen ezért a relációs modell ki is veszi az alkalmazók kezéből az elérési út meghatározását és a rendszerre bízta ezt a feladatot. Az pedig sokkal gyorsabban, és egészen extrém esetektől eltekintve mindig jobban vagy legalább olyan jól oldja meg ezt a problémát, mint a legfelkészültebb adatbázis-felügyelő, az átlagos programozókról, vagy a nem programozó alkalmazókról nem is beszélve.

Az utasítás elvégzésének eredményességéről (vagy eredménytelenségéről) a rendszer értesítést küld a felhasználónak és a kért adatokat megtekintésre megjeleníti, illetve a változtatásokat az adatbázisban végrehajtja.

Többfelhasználós rendszernél a rendszer automatikusan gondoskodik arról is, hogy ha az utasítás végrehajtása alatt más felhasználó is dolgozni kívánna ugyanazokkal az adatokkal, akkor a két felhasználás konzisztens maradjon. Így például ha az egyik felhasználó módosítani akarná egy táblázat összes sorát (ezt megteheti egyetlenegy UPDATE utasítással), akkor másik felhasználó semmit sem módosíthat ugyanebben a táblázatban, amíg az első változtatás sorozat nem ér véget, eredménye nem véglegesítődik az adatbázisban. Ezeket a kérdéseket a zárok kezelésénél, a 12.4 pontban tárgyaljuk majd részletesen.

11.2 Szelektálás (SELECT utasítás)

11.2.1 A teljes- és a szubszelekt funkciója

A SELECT utasítással meghatározott feltételeknek eleget tevő adatokat választunk ki egy vagy több táblázatból/nézetből. Összetett utasítás, melynek segítségével a 7.6 pontban ismertetett relációs műveleteknek és ezek tetszés szerinti (megengedett) kombinációjának csaknem mindegyike elvégezhető egyetlenegy SELECT utasítás kiadásával.

Az utasítás eredménye mindig egy ideiglenes táblázat, mely azonban a 7.2 pont definíciói szerint nem mindig tekinthető relációnak.

Párbeszédese SELECT utasításnál az eredmény táblázatot alapértelmezésben a képernyőn láthatjuk, de kérésre ki is nyomtathatjuk, esetleg adatait átírányíthatjuk egy szekvenciális állományba.

A kiválasztott adatokat a felhasználó tetszőleges sorrendben és formában ábrázolva más, hasonló szubszelekt eredményével uniót képezve megtekintheti. Az uniót és/vagy rendezést tartalmazó SELECT utasítást hívják a *teljes szelekt*nek (full select).

Valamivel korlátozottabbak a szubszelekt (subselect) lehetőségei. Ez egy vagy több relációból kiválasztott sorok megtekintésére, nézet képzésére vagy más relációba történő beíráshoz sorok kiválasztására, egy reláció módosítandó vagy törlendő sorainak kiválasztására, egymásba ágyazott SELECT-eknél a külső SELECT kiválasztandó sorainak meghatározására szolgál. Az utoljára említett formáját *bel-ső kiválasztás*nak is nevezzük.

Az alábbiak, ha csak nem említjük meg külön, hogy az csak a teljes szelektre érvényes, mind a szub-, mind a teljes szelektre vonatkoznak.

11.2..2 A SELECT utasítás alapformája (szubszelekt)

A SELECT utasítás alapformája a *szubszelekt* a következő (11.1 ábra):
 SELECT miként, mit,

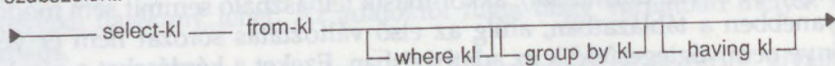
FROM honnan,

WHERE melyre igaz, hogy,

GROUP BY eszerint csoportosítva,

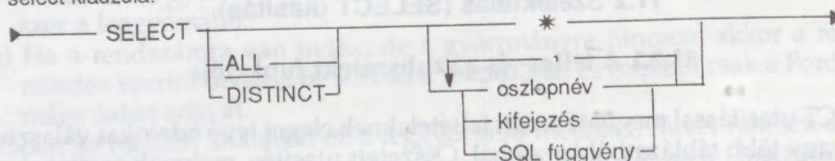
HAVING csoportkiválasztási feltétel.

szubszelekt:

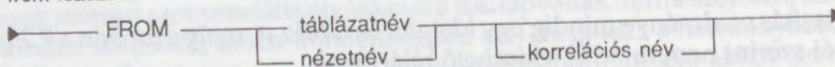


ahol

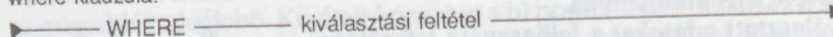
select klauzula:



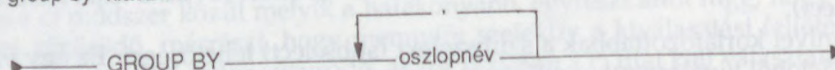
from klauzula:



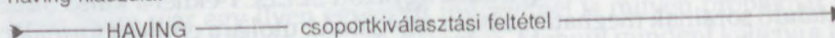
where klauzula:



group-by klauzula:



having klauzula:



11.1 ábra

A SELECT utasítás (szubszelekt) szintaxis diagrammja (kl= klauzula rövidítése)

Az egyes klauzulák főbb jellemzői a következők:²

miként

— ALL, minden, a feltételnek megfelelő sort kiválasztunk. Ez az alapértelmezés.

— DISTINCT, ha a feltételnek megfelelő sorok között azonosak vannak (például azért, mert a SELECT-ben projekció is van, nem minden oszlopot tartunk meg a kiválasztott sorokból), akkor ezek mindegyikéből csak egyet tartunk meg.

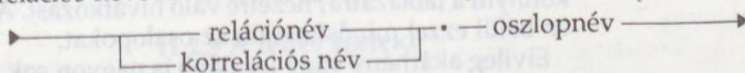
mit

— * A FROM klauzulában megadott táblázatok/nézetek összes oszlopát kiválasztjuk a kiválasztásra szolgáló reláció(k) oszlopainak sorrendjében. Ha több reláció szerepel a FROM klauzulában, akkor a relációk sorrendje az ottani felsorolási sorrend lesz és ezen belül érvényesül az oszlopok sorrendje.

Párbeszédés módon általában így dolgozunk. Még akkor is, ha nincsen minden oszlopra szükségünk. Egyszerűbb egy * jelet kiírni, mint az oszlopokat felsorolni. Programban viszont lehetőleg kerüljük el, mert ha megváltozik a táblázat (pl. ALTER-ral új oszlopa lesz), akkor ez hibát okozhat.

— oszlopnév Az utasítás a projekció műveletét is tartalmazza. Csak az itt felsorolt oszlopokat választjuk ki az eredményül kapott sorokból. Az egyes oszlopok nevét vesszővel kell elválasztani.

Az oszlopnévnek a SELECT utasításon belül egyértelműnek kell lennie. Ezért, ha több táblázatból/nézetből történik a kiválasztás, akkor a nem egyértelmű oszlopneveket egyértelművé kell tenni. Ez általában azt jelenti, hogy az oszlopnevet minősíteni kell a megfelelő táblázat/nézet nevével, vagy a FROM klauzulabeli korrelációs nevével. Ennek szokásos formája:



— kifejezés konstans vagy oszlopokon és/vagy konstansokon végzett megengedett aritmetikai vagy sztring műveletek eredménye. Pl. összeg*percent/100 az adott sorban levő (numerikus) összeg és percent oszlopértékek szorzatának századrészét teszi az eredmény sor ezen oszlopába.

²Először a fogalmakat összefoglalóan ismertetjük, és csak utána hozunk fel példákat, mert így az olvasó áttekintést kap a SELECT utasítás teljes szerkezetéről. Aki azonban úgy érzi, hogy szemléletes példák után könnyebben értené meg ezt a részt, az lapozzon hátrább a 11.2.4 ponthoz, és az ottani példák áttanulmányozása után térjen ide vissza.

SQL függvény A kifejezés egy speciális formája.³ Adott oszlopértéknek más formában való megjelenítését (pl. tizedesjegyek levágását), vagy valamilyen, az oszlopértékre vonatkozó aggregátumot jelent (pl. a kiválasztott sorokban ezen oszlop értékeinek maximumát, MAX(oszlopnev)).

Mind a függvényekben, mind a kifejezésekben a nem egyértelmű oszlopneveket minősíteniünk kell.

A „mit” helyén explicit, vagy a *-gal implicit megadott oszlopokat kiválasztási vagy szelekt listának nevezzük. Az eredménytáblázat oszlopainak sorrendje megegyezik a szelekt lista felsorolási sorrendjével.

honnan

— táblázatnév/nézetnév

Ebből a relációból kell kiválasztani a „mit”-ben megadott oszlopokat.

Több táblázat felsorolásakor ezek keresztszorzatából választjuk ki a sorokat. Valójában a gyakorlatban ez csaknem mindig egyesítést jelent. Az egyesítésre szolgáló oszlopot a WHERE feltételben adjuk meg.

A JOIN kompatibilitásból (l. 7.6.7 pont) az oszlopnevek különbözőségét a minősítéssel biztosítjuk, míg a JOIN oszlopok nevének egyezőségét az egyesítésre szolgáló oszlopoknak a WHERE feltételben való névszerinti felsorolásával valósítjuk meg, amit hallgatólágosan egy közös névre való átnevezésnek tekinthetünk. (L. 11.2.4b. példáját.)

A táblázat/nézetnévnek az egész adatbázisra vonatkozóan egyértelműnek kell lenni. Ezt általában úgy érik el, hogy a relációnevet minősíteni kell a tulajdonosa nevével (felhasználói azonosítójával). Amennyiben a SELECT utasítás kiadója és a táblázat tulajdonosa megegyezik, úgy ez a minősítés elhagyható.

A rendszerint 1–2 karakterből álló rövid korrelációs név megkönnyíti a táblázatra/nézetre való hivatkozást. Az adott SELECT-en belül ezzel minősíthetjük az oszlopokat.

Elvileg akárhány vagy legalább is nagyon sok táblázatot/nézetet egyesíthetünk. Gyakorlatilag azonban háromnál többet nem igen szoktunk megadni a keresztszorzat révén létrejövő igen nagyméretű ideiglenes táblázatok miatt.

³ A szokásos szintaktikában az SQL függvényeket is a kifejezések közé sorolják. Mi azonban a függvények speciális esetére és fontosságára való tekintettel ebben a részben külön tárgyaljuk őket, és általában külön fel is tüntetjük alkalmazhatóságukat a szintaxis-diagramban, illetve a szöveges leírásban.

melyre igaz

A „mit” és „honnan” leírását minden SELECT utasításban meg kell adnunk. Ha a „melyre igaz” WHERE klauzulát nem adjuk meg, akkor a „honnan” reláció(k) minden sorát kiválasztjuk. Különben csak azokat, amelyekre a WHERE kulcsszó után megadott kiválasztási feltétel teljesül. Ez legegyszerűbb formájában egy

oszlopnév–1 operandus összehasonlítandó,

ahol összehasonlítandó egy másik oszlopnév, kifejezés, vagy SQL függvény, operandus pedig általában valamelyik aritmetikai összehasonlító művelet (pl: =, < stb.) lehet.

Ezeket a feltételeket aztán tetszés szerint összekapcsolhatjuk az AND, OR és a NOT logikai műveletekkel.

eszerint csoportosítva

Aggregátumok kiválasztásakor használhatjuk. A feltétel alapján kiválasztott sorokat ezen oszlop(ok) értékei szerint csoportosítjuk, és minden csoportértékhez az ehhez tartozó aggregátum értékét adjuk meg. (Pl. a GÉPJÁRMŰ táblázatból való kiválasztáskor GYÁRTMANY szerint csoportosítunk, és minden gyártmányértékhez megadjuk az adott gyártmányú, de különböző típusú gépjárművek úrtartalmának maximális és minimális értékét, l. 11.2.4 pont c) példáját.)

Ha a szelekt listában aggregátumok szerepelnek, de nincs GROUP BY klauzula, akkor a SELECT eredménye egyetlen sor lesz, melyben az aggregátum az összes kiválasztott sorra vonatkozik.

csoportosítási feltétel

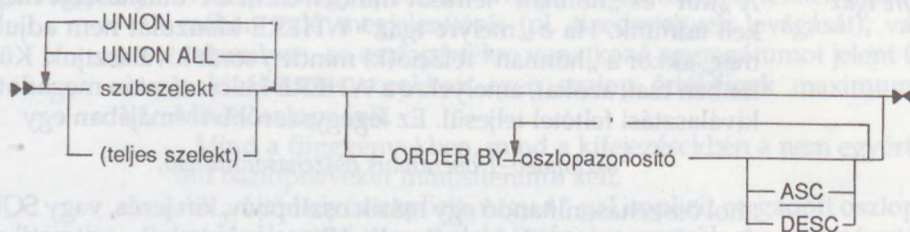
Csak azok a csoportok jelennek meg a SELECT végeredményében, melyekre az itt megadott kiválasztási feltétel teljesül. (Pl. csak akkor kapunk maximumot és minimumot, ha a csoportnak a minimális értéke 1000-nél nagyobb, l. 11.2.4 pont c) példáját.)

11.2.3 A teljes szelekt

A teljes szelektet csak adatok olvasás céljára történő kiválasztására használjuk. A szubszelekt eredményeként kapott sorokat tetszőleges oszlop(ok) értéke szerinti növekvő (ASC, alapértelmezés) vagy csökkenő (DESC) sorrendben kaphatjuk meg, és képezhetjük több szelekció eredményének unióját is.

A teljes szelekt szintaxis diagramját a 11.2 ábrán mutatjuk be.

Az ORDER BY klauzulában az oszlopazonosító a rendezésre szolgáló oszlopnak a SELECT-en belüli egyértelmű (szükség esetén minősített) neve, vagy a pozíciója a szelektlistában, a „mit”-ben felsorolt oszlopok között. Unióképzésnél csak ezt használjuk, mivel az unió-kompatibilitáshoz (l. 7.6.5 pont) az oszlopnevek azonosságát implicit úton, a szubszelektben szereplő oszlopoknak a szelekt lista oszlopaival való sorrendi egyeztetésével valósítjuk meg. Arról természetesen a SELECT utasítást kiadó felhasználónak kell gondoskodnia, hogy az egyes szub-



11.2 ábra A teljes szelekt szintaxis diagrammja

szelektekben ugyanannyi oszlopot és olyan sorrendben válasszon ki, hogy a szelekt listában azonos pozíción álló oszlopok adatait egy attribútum típussal lehesen ábrázolni. (Általában elegendő, hogy az unió minden relációjában vagy karakter vagy numerikus legyen, a különböző hosszúságú karakteres és a különféle típusú numerikus ábrázolásmódok között a rendszer átkonvertál.)

Az UNION esetén, ha az unió eredményeként azonos sorok keletkeznének, akkor azok mindegyikéből csak egy jelenik meg az eredményben, míg UNION ALL megadásakor az összeset megkapjuk.

Szubszelektnél a DISTINCT, teljes szelektnél a UNION megadásával a rendszer kiszűri a duplikált sorokat. Ahhoz, hogy ezt elvégezze, a kiválasztás után rendeznie kell az eredményt akkor is, ha külön rendezési kérést (ORDER BY klauzulát) nem adtunk meg. Ez sok sor esetén elég hosszadalmas lehet. Ezért azonos egyéb feltételek mellett a SELECT ALL illetve a UNION ALL végrehajtása általában jóval hatékonyabb, mint a SELECT DISTINCT illetve a UNION végrehajtása.

Láthatjuk azt is, hogy a SELECT utasításban *nincs is lehetőségünk az elérési út meghatározására*. Ezt a rendszer maga végzi el, és nyilván jóval hatékonyabb utat talál, ha megfelelő indexeket hozhatunk létre.

Ezért, ha a várható szelekciókban egy oszlop gyakran szerepel a WHERE klauzulában, akkor erre rendszerint célszerű indexet definiálni. Ennek segítségével a rendszer gyorsan kiválaszthatja azokat a sorokat, amelyekre az erre az oszlopra vonatkozó feltétel teljesül és a továbbiakban az összes sor helyett csak ezekkel fog dolgozni.

Ugyancsak célszerű indexet definiálni azokra az oszlopokra is, melyek szerint gyakran kérjük rendezetten a SELECT eredményét. Ebben az esetben ugyanis lehetséges, hogy eleve eszerint az index szerint olvassa be a táblázatból a sorokat és ezekből hagyja el a további feltételeknek nem eleget tevőket. Ezáltal a végeredményt már nem kell rendezni.

Ugyanez érvényes, ha a szelekció nézetten keresztül történik, csak ilyenkor az indexet a nézet alatt levő táblázat oszlopára kell felcíptenünk.

11.2.4 Példák szelekcióra

Az alábbiakban néhány egyszerű példát mutatunk be a SELECT utasítás alkalmazására. A példák a relációs modellnél ismertetett 7.1 és 7.2 ábrán bemutatott, és a

10.2 ábra definícióival létrehozott táblázatokra vonatkoznak. Annak érdekében, hogy az olvasónak ne kelljen okvetlenül visszalapozni az eredeti definíciókra, ismét megadjuk a példákban használt két táblázat oszlopait.

G&T (RENDSZÁM,TULAJDONOS,TULAJDONJOG-KEZDETE,
TULAJDONJOG-VÉGE)
GÉPJÁRMŰ (RENDSZÁM,GYÁRTMÁNY,TÍPUS,ÚRTARTALOM,EGYÉB)

- a) Válasszuk ki a G&T táblázatból az 1992–12–31 után vásárolt vagy eladott gépkocsik tulajdonosainak neveit. (A 7.11 ábrán bemutatott relációs művelet megfogalmazása SQL nyelven.)

```
SELECT TULAJDONOS FROM G&T
WHERE TULAJDONJOG-KEZDETE> '1992-12-31'
OR TULAJDONJOG-VÉGE> '1992-12-31'
```

Mivel a TULAJDONJOG-KEZDETE és -VÉGE oszlopokat DATE típusnak definiáltuk (l. 10.2 ábra), az összehasonlításban karaktorsorozatként adhatjuk meg a dátumot. Konstans karaktorsorozatokat az SQL-ben általában aposztrofok (') határolnak.

Az utasításban a kötelező szelekt listán („mit”) és a FROM klauzulán („honnan”) kívül csak a WHERE klauzula szerepel. Ebben a kiválasztási feltétel két OR-ral összekapcsolt összehasonlítás, melynek második operandusa egy kifejezés (konstans).

Ha ugyanaz a személy többször vásárolt vagy adott el gépkocsit a megadott nap után, akkor azt az eredményben is többször kapjuk meg. Ezt, ha akarnánk, a SELECT DISTINCT ... megadásával küszöbölhetnénk ki.

Amennyiben a tulajdonosok neveit növekvő ABC sorrendben kívánnánk megkapni, úgy az utasítást az ORDER BY TULAJDONOS klauzulával kellene kiegészítenünk.

Ugyanerre az eredményre vezetne a

```
SELECT TULAJDONOS FROM G&T
WHERE TULAJDONJOG-KEZDETE> '1992-12-31'
UNION ALL
SELECT TULAJDONOS FROM G&T
WHERE TULAJDONJOG-VÉGE> '1992-12-31'
ORDER BY 1
```

SQL utasítás is (a WHERE korlátozás OR feltétele átalakítható UNION-ná).

Ha UNION ALL helyett UNION-nal kapcsolnánk össze a két szubszelektet, akkor a feltételeknek megfelelő tulajdonosok nevét csak egyszer kapnánk meg az eredményben.

Elképzelhető, hogy a kétféle módon megfogalmazott kérdésre az SQL rendszer különböző elérési utakat generál. (Nem szép, de sajnos a legtöbb rendszerben időnként ez előfordulhat.) Ekkor a válaszdíő függeni fog az utasítás konkrét formájától.

- b) Adjuk meg, hogy jelenleg az egyes személyek tulajdonában milyen gyártmányú és típusú gépkocsik vannak. (A 7.12 ábrán levő relációs művelet

megfogalmazása SQL-ben.) Az eredményt gyártmány, ezen belül típus, ezen belül név szerint akarjuk növekvő ABC sorrendben megkapni.

Mivel a tulajdonos neve a G&T táblázatban, a gyártmány és típus pedig a GÉPJÁRMŰ táblázatban van, a szelekciót a két táblázat egyesítéséből kell elvégezni. Az egyesítés a RENDSZÁM oszlopok alapján történik. Ez a név mindkét táblázatban azonos, ezért minősíteniünk kell.

Az SQL utasítás (egyik lehetséges formája) a következő:

```
SELECT GYÁRTMÁNY,TÍPUS,TULAJDONOS
FROM G&T, GÉPJÁRMŰ
WHERE G&T.RENDSZÁM=GÉPJÁRMŰ.RENDSZÁM
AND TULAJDONJOG-VÉGE IS NULL
ORDER BY GYÁRTMÁNY,TÍPUS, TULAJDONOS
```

Láthatjuk, hogy az egyesítésre szolgáló oszlopnak nem is kell szerepelni a szelekt listában. A két táblázat egyesítése enélkül is végrehajtható. A TULAJDONJOG-VÉGE IS NULL feltétel biztosítja, hogy ugyanazon gépkocsi volt és jelenlegi tulajdonosai közül csak a jelenlegi tulajdonost választjuk ki, hiszen a régebbi tulajdonosoknál ebben az oszlopban szerepel az eladás dátuma, tehát értéke nem null-érték.

- c) Adjuk meg a különböző gyártmányú gépkocsik motorjainak maximális és minimális hengerűrtartalmát. (A 7.13 ábra relációs művelete SQL-be átfordítva.) Az eredményt a maximális űrtartalom értéke szerint csökkenő sorrendben kívánjuk megkapni és kihagyjuk azokat a gyártmányokat, ahol a maximum értéke 1000 cm³-nél kisebb.

Mivel most aggregátumot készítünk és az azonos gyártmányú gépkocsik alkotnak egy-egy csoportot, meg kell adnunk a GROUP BY klauzulát is. A csoportok közül azonban nem mindegyiket kell figyelembevennünk, ezért ezt ki kell egészíteniünk a HAVING klauzulával is. Az összes adat bent van a GÉPJÁRMŰ táblázatban, illetve előállítható belőle. Ezért a FROM klauzulában csak ezt kell megadnunk.

A kérdést az alábbi SQL utasítással válaszolhatjuk meg:

```
SELECT GYÁRTMÁNY,MAX(ŰRTARTALOM),MIN(ŰRTARTALOM)
FROM GÉPJÁRMŰ
GROUP BY GYÁRTMÁNY
HAVING MAX(ŰRTARTALOM) = 1000
ORDER BY 2 DESC
```

11.2.5 SQL függvények

Igen kényelmessé teszik az SQL használatát a rendszerbe beépített függvények. Ezeknek két csoportja van, az

- az oszlopfüggvények és a
- skálárfüggvények.

Az *oszlopfüggvények* aggregátum képzésre szolgálnak. A kiválasztott sorokra, illetve GROUP BY megadásánál az azonos csoportokra adják meg az aggregátumban szereplő oszlop, kifejezés vagy SQL függvény

- átlagát, AVG (argumentum),
- maximumát, MAX (argumentum),
- minimumát, MIN (argumentum),
- összegét, SUM (argumentum),
- a sorok számát, COUNT (*).

Az átlag és az összefüggvény argumentuma csak numerikusként definiált oszlopokat és numerikus konstansokat, numerikus eredményt szolgáltató SQL függvényeket tartalmazhat.

A maximum és minimum meghatározásánál a gép belső ábrázolásmódja szerinti sorrend, általában az EBCDIC kód az érvényes.

Az AVG, SUM és COUNT függvények argumentumát kiegészíthetjük a DISTINCT kulcsszóval. Ekkor az aggregátum képzésben csak a különböző értékeket veszi a rendszer figyelembe. Ilyenkor a COUNT függvélynél is az oszlop nevét kell megadnunk.

Ha egy kiválasztott sorban a függvény argumentumban szereplő bármelyik oszlop értéke null-érték, akkor ez kimarad az aggregátumképzésből. Kivétel ez alól a COUNT(*) függvény, amelynek eredményébe minden kiválasztott sor beleszámít. Éppen ezért vigyáznunk kell, mert ha OSZLOP1 tartalmazhat null-értéket is, akkor nem biztos, hogy AVG(OSZLOP1) kiválasztása ugyanazt adja, mint SUM(OSZLOP1)/COUNT(*).

Ha a WHERE feltételnek egy sor sem tesz eleget, akkor az eredmény oszlop-függvény értéke a null-érték lesz, kivéve a COUNT függvényt, mely ténylegesen a 0 numerikus értéket vesz fel.

Askalár függvények az argumentumban szereplő oszlop, kifejezés, SQL függvény értékét alakítják át más formába. A függvénytől függ, hogy az argumentum milyen adattípust tartalmazhat. Amennyiben az oszlopértéken egy nem megengedett konverziót kívánnánk végrehajtani, az utasítás már az értelmezéskor vagy a végrehajtáskor hibajelzéssel leáll.

A legfontosabb skalárfüggvények a következők:

DECIMAL (argumentum, *p*, *q*)

A numerikus argumentum értékét *p* jegyű számként, *q* tizedesjegggyel adja meg.

DIGITS (argumentum)

Az argumentum numerikus értéket átalakítja karakter formátumba. Csak ennek a függvénynek a felhasználásával lehet karakteresnek és numerikusnak definiált oszlopok tartalmát összehasonlítani.

FLOAT (argumentum), INTEGER (argumentum)

Az argumentum numerikus értéket átkonvertálja lebegőpontos, illetve bináris formába.

SUBSTR (argumentum, *kezdet*, *hossz*)

Az argumentum karaktersorozatból *hossz* hosszúságú szubsztringet készít a *kezdet* kezdőpozíciótól kiindulva.

Azoknál a rendszereknél, amelyek megengedik a DATE és TIME típusú oszlopok deklarálását, igen hasznosak lehetnek az ezekkel való műveletek elvégzésére szolgáló függvények.

Ilyen például a YEAR, MONTH, DAY függvények, melyek a dátum típusú argumentum értékének év, hó illetve nap részét adják vissza, vagy pl. DA-

TUM+2MONTHS típusú kifejezés használata, amelynek értéke DATUM két hónappal megnövelt értéke. (Pl. DATUM='1991-12-31' esetén '1992-02-29', de DATUM='1990-12-31' esetén '1991-02-28'.)

Ha például kíváncsiak vagyunk, hogy a 7.21 ábrán látható ALKALMAZOTTAK táblázatból hányan dolgoznak az egyes osztályokon, akkor erre az alábbi SELECT utasítással kaphatunk választ:

```
SELECT OSZT, COUNT(*) FROM ALKALMAZOTTAK
GROUP BY OSZT
```

Ebből a kérdésből azonban nem kapjuk meg azokat az osztályokat, amelyekben pillanatnyilag senki sem dolgozik, holott COUNT(*)=0-val jó volna, ha ezek is szerepelnének. A 11.2.6 pontnak az EXISTS-nél leírt példájában megmutatjuk, hogyan kell ezt a kérdést teljes szelektté kiegészítenünk, hogy ezek is szerepeljenek az eredményben.

Ugyanezen okból arra a kérdésre, hogy hány osztálya van a vállalatnak, a

```
SELECT COUNT(*) FROM OSZTÁLYOK
```

```
és a SELECT COUNT(OSZT,DISTINCT) FROM ALKALMAZOTTAK
```

nem biztos, hogy ugyanazt az eredményt fogja szolgáltatni.

Az első kérdés a vállalat összes elvben létező osztályát megadja, míg a második csak a ténylegesen működőket, azokat, amelyeknek legalább egy dolgozója van már (vagy még).

11.2.6 További kiválasztási feltételek

A SELECT utasítás igen nagy előnye, hogy a kiválasztási feltételek helyes megfogalmazásával igen bonyolult összefüggéseket is könnyen megadhatunk, melyeknek teljesülni kell a kiválasztott sorokra.

Nem célunk a különböző feltételek teljes választékának a felsorolása. Ez különben is rendszelfüggő. Itt csupán a leggyakrabban használt lehetőségeket ismertetjük, melyek egy része (pl. a BETWEEN és az IN) valójában csak a kiválasztási feltételek egyszerűbb, szemléletesebb leírását teszi lehetővé.

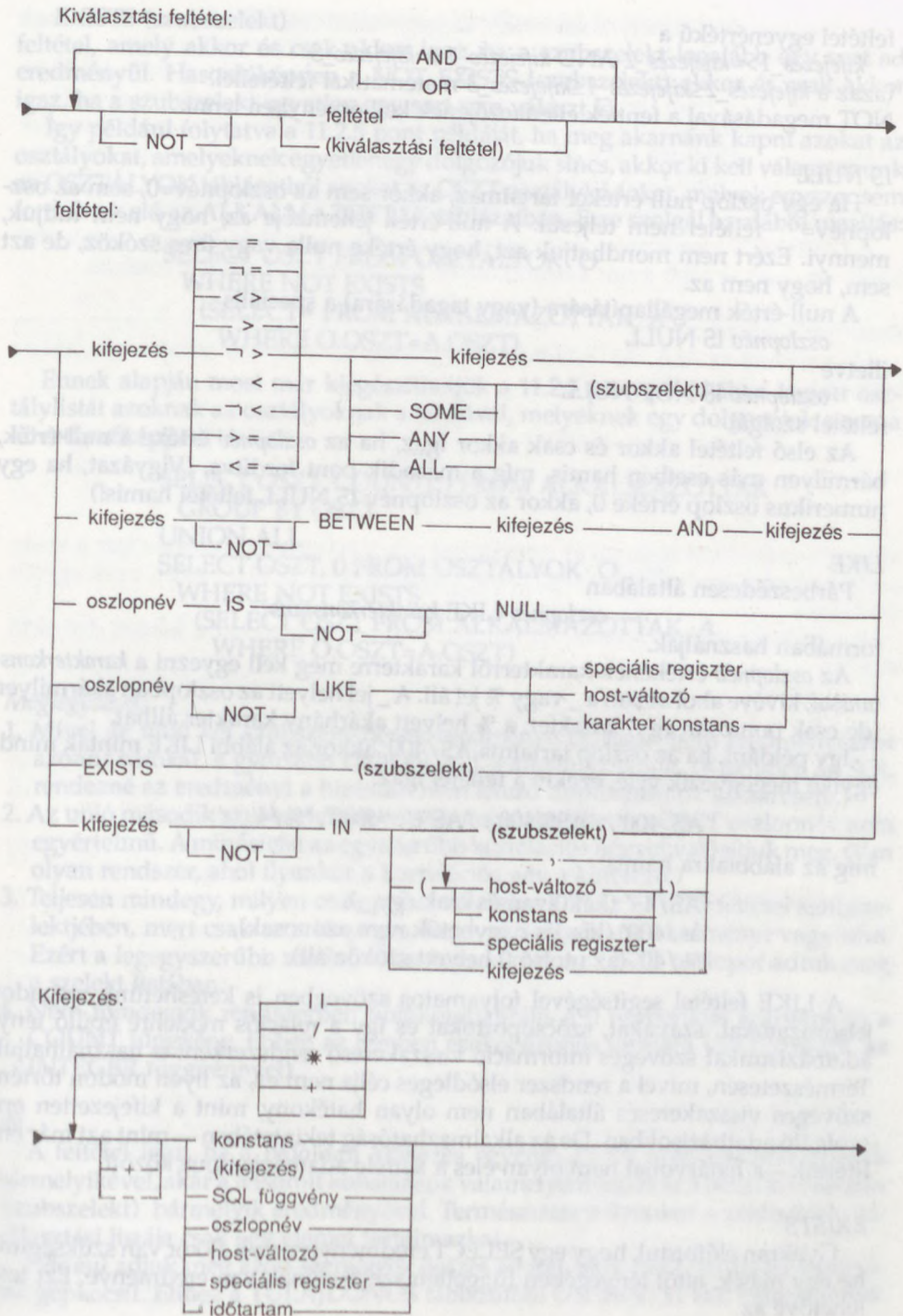
A 11.3 ábrán láthatjuk a WHERE klauzulában megadható kiválasztási feltételeknek a legtöbb rendszer által támogatott formáját. (Egyes rendszerek, pl. DB2, SQL/DS ennél szélesebb választékot is megengednek, mások, főleg a személyi számítógépeken üzemelők még ezek mindegyikének használatát sem teszik lehetővé. Az ábrán látható jelölések közül a *host változót* az SQL utasítások programba való beépítésekor fogjuk definiálni. A *speciális regiszter* értéke leggyakrabban: USER. Ekkor ez az utasítást kiadó felhasználó bejelentkezési azonosítója lesz. *Időtartam* DATE típusú változókat, illetve ezeken elvégzett műveleteket jelent.

A legegyszerűbb feltételt (lényegében a legfölső ágot) a SELECT utasítás általános leírásánál már ismertettük.

BETWEEN

A BETWEEN lehetőség egyszerűbbé és áttekinthetőbbé teszi a feltétel leírását.

A kifejezés_1 BETWEEN kifejezés_2 AND kifejezés_3



11.3 ábra Kiválasztási feltételek szintaxis diagrammja

feltétel egyenértékű a

$kifejezés_1 >= kifejezés_2 \text{ AND } kifejezés_1 <= kifejezés_3$,
(azaz a $kifejezés_2 \leq kifejezés_1 \leq kifejezés_3$ matematikai feltétellel.
NOT megadásával a fentiek ellenkezőjének kell érvényben lenni.

IS NULL

Ha egy oszlop null értéket tartalmaz, akkor sem az $oszlopnév=0$, sem az $oszlopnév=' '$ feltétel nem teljesül. A null-érték jellemzője az, hogy nem tudjuk, mennyi. Ezért nem mondhatjuk azt, hogy értéke nulla vagy üres szóköz, de azt sem, hogy nem az.

A null-érték megállapítására (vagy tagadására) a speciális
oszlopnév IS NULL

illetve

oszlopnév IS NOT NULL

feltétel szolgál.

Az első feltétel akkor és csak akkor igaz, ha az *oszlopnév* értéke a null-érték, bármilyen más esetben hamis, míg a második pont fordítva. (Vigyázat, ha egy numerikus oszlop értéke 0, akkor az *oszlopnév IS NULL* feltétel hamis!)

LIKE

Párbeszédesen általában

oszlopnév LIKE karakterkonstans

formában használják.

Az *oszlopnév* értékének karakterről karakterre meg kell egyeznie a *karakterkonstanssal*, kivéve ahol abban a *_* vagy *%* jel áll. A *_* jel helyett az oszlopban akármilyen (de csak pontosan egy) karakter, a *%* helyett akárhány karakter állhat.

Így például, ha az oszlop tartalma AS/400, akkor az alábbi LIKE minták mindegyike megegyezik vele, ezekre a feltétel igaz:

'AS_400', 'AS%400', 'AS%', '%00', '%/4_'

míg az alábbiakra hamis:

'AS/4_' (két 0 van és csak egy *_*),

'as/400' (kis- és nagybetűk nem azonosak),

'%/40' (az utolsó 0 helyett szóköz áll).

A LIKE feltétel segítségével folyamatos szövegben is kereshetünk megadott jelsorozatokat, szavakat, szócsoportokat és így a relációs modellre épülő tényadatbázisunkat szöveges információ visszakereső rendszerként is használhatjuk. Természetesen, mivel a rendszer elsődleges célja nem ez, az ilyen módon történő szöveges visszakeresés általában nem olyan hatékony, mint a kifejezetten erre szolgáló adatbázisokban. De az alkalmazhatóság tekintetében — mint azt már említettük — a határvonal nem olyan éles a kétféle adatbázis típus között.

EXISTS

Gyakran előfordul, hogy egy SELECT eredményére csak akkor van szükségünk, ha egy másik, ettől lényegében független szelekciónak van eredménye. Ezt teszi lehetővé az

EXISTS (szubszelekt)

feltétel, amely akkor és csak akkor igaz, ha a szubszelekt legalább egy sort ad eredményül. Hasonlóképpen a NOT EXISTS (szubszelekt) akkor és csak akkor igaz, ha a szubszelekt egyetlen egy sort sem választ ki.

Így például folytatva a 11.2.5 pont példáját, ha meg akarnánk kapni azokat az osztályokat, amelyeknek egyetlenegy dolgozójuk sincs, akkor ki kell választanunk az OSZTÁLYOK táblázatból azokat az OSZT osztálykódokat, melyek egyszer sem fordulnak elő az ALKALMAZOTTAK táblázatban. Erre szolgál az alábbi utasítás:

```
SELECT OSZT FROM OSZTÁLYOK O
WHERE NOT EXISTS
  (SELECT * FROM ALKALMAZOTTAK A
   WHERE O.OSZT=A.OSZT)
```

Ennek alapján most már kiegészíthetjük a 11.2.5 pont példájában kapott osztálylistát azoknak az osztályoknak a kódjával, melyeknek egy dolgozójuk sincs, a következőképpen:

```
SELECT OSZT, COUNT(*) FROM ALKALMAZOTTAK
GROUP BY OSZT
UNION ALL
SELECT OSZT, 0 FROM OSZTÁLYOK O
WHERE NOT EXISTS
  (SELECT OSZT FROM ALKALMAZOTTAK A
   WHERE O.OSZT=A.OSZT)
```

Megjegyzések:

1. Mivel az unió két szubszelektjének eredménye biztosan nem eredményezhet azonos sorokat, a gyorsabb UNION ALL-t alkalmazhatjuk. (Különben az SQL rendezné az eredményt a biztosan nem létező duplikátumok kiszűrésére.)
2. Az unió második szubszelektjében a szubszelektben az OSZT oszlopnév nem egyértelmű. A minősítést az egyszerűbb korrelációs névvel valósítjuk meg. (Van olyan rendszer, ahol ilyenkor a korrelációs név a kötelező.)
3. Teljesen mindegy, milyen oszlopokat választunk ki az EXISTS feltétel szubszelektjében, mert csak az a lényeges, hogy az szolgáltat-e eredményt vagy sem. Ezért a legegyszerűbb módot választottuk, csak az OSZT oszlopot adtuk meg a szelekt listában.
4. Nem mindegyik rendszerben uniókompatibilis a 0 numerikus konstans és a COUNT függvény. Ebben az esetben ezeket azonos formára kell hozni (pl. az INTEGER függvénnyel).

IN

A feltétel igaz, ha a baloldali kifejezés egyenlő az IN után felsorolt értékek bármelyikével, akár a felsorolt konstansok valamelyikével, akár a belső kiválasztás (szubszelekt) bármelyik eredményével. Természetesen ilyenkor a szubszelekt kiválasztási listája csak egy elemet tartalmazhat.

Például adjuk meg azon személyek összes adatát, akik 1992-10-16-án vásároltak gépkocsit. Ehhez a TULAJDONOS táblázatból (7.2 ábra) ki kell választanunk

az összes olyan sort, amelyikben a NÉV megegyezik a G&T táblázat olyan sorának a TULAJDONOS-ával, ahol a TULAJDONJOG-KEZDETE='1992-10-16'.

A SELECT utasítás tehát a következő:

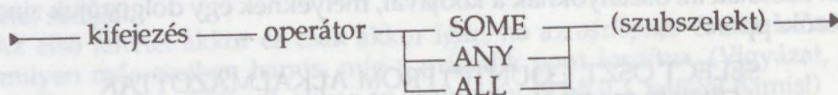
```
SELECT DISTINCT * FROM TULAJDONOS
WHERE NÉV IN
  (SELECT TULAJDONOS FROM G&T
   WHERE TULAJDONJOG-KEZDETE='1992-10-16')
```

A DISTINCT kiegészítést azért adtuk meg, hogy ha valaki (pl. egy autókereskedő) egy napon több autót vásárolt, az ne szerepeljen többször a listában.

Kvantifikált feltétel

Ha a feltétel az ALL, ANY, SOME kulcsszavak bármelyikét tartalmazza, akkor kvantifikált feltételről beszélünk.

A



akkor és csak akkor igaz, ha az operátorral jellemzett összehasonlítás igaz a szubszelekt minden eredményére (ALL) vagy legalább egyre (SOME, ANY, ezek egyenértékűek).

Kvantifikált szubszelektnél, ha nincsen eredmény sor, akkor a feltétel definíció szerint

- igaz ALL-ra,
- hamis ANY-re és SOME-ra.

Így, ha például a T1 táblázat O1 oszlopának értékei az egyes sorokban 1;2;3;4, a T2 táblázat O2 oszlopának pedig 2;3, akkor

a) SELECT O1 FROM T1

WHERE O1 = ANY (SELECT O2 FROM T2)

eredménye két sor az O1=2 ill. 3 értékkel

b) SELECT O1 FROM T1

WHERE O1 > ANY (SELECT O2 FROM T2)

eredménye: 3;4

c) SELECT O1 FROM T1

WHERE O1 > ALL (SELECT O2 FROM T2)

eredménye: 4

d) SELECT O1 FROM T1

WHERE O1 ALL (SELECT O2 FROM T2)
WHERE O2 < 0)

eredménye: 1;2;3;4

(A szubszelekt nem ad eredményt, ezért a feltétel minden O1 értékre igaz lesz.)

11.2.7 Megjegyzések a kiválasztási feltételekhez

a) NULL-érték kezelése

Ha a feltételben akárhol null-érték szerepel akár azért, mert az oszlop vagy a kifejezés eleve null-értéket tartalmazott, akár azért, mert a szubszelekt egyetlenegy sort sem adott eredményül, úgy a feltétel eredménye (az EXISTS és a kvantifikált szubszelekt kivételével) meghatározatlan, amit kérdőjellel (?) jelölünk.

Összetett feltétel ilyen esetekre kibővített igazság táblázatát a 11.4 ábrán láthatjuk.

P	Q	P	AND	Q	P	OR	Q
I	I		I			I	
I	H		H			I	
I	?		?			I	
H	I		H			H	
H	H		H			H	
H	?		H			?	
?	I		?			I	
?	H		H			?	
?	?		?			?	

11.4 ábra Összetett feltételek igazságtáblázata

A két feltétel: P és Q

I = igaz; H = hamis; ? nem meghatározott

P és Q az összetett feltétel két komponense.

Bármelyikük lehet igaz (I), hamis (H), vagy meghatározatlan (?).

Ügyeljünk rá, hogy NOT ?=?, hisz ha valamiről nem tudjuk igaz-e vagy sem, akkor az ellenkezőjéről sem állíthatunk semmit.

b) Eredményoszlopok száma belső kiválasztásnál

A WHERE feltételben szereplő belső kiválasztás az EXISTS kivételével mindig csak egyetlen oszlopot választhat ki.

EXISTS-nél a szelekt lista akárhány oszlopból állhat.

c) Korrelált és nem korrelált belső kiválasztás

Ha a belső kiválasztás eredménye független attól, hogy a feltételben a külső szelekt melyik sorával hasonlítjuk össze, akkor korrelálatlan, különben korrelált szubszelektéről beszélünk. Az első esetben a belső kiválasztást a rendszer csak egyszer értékeli ki. A külső SELECT minden sorának összehasonlításakor ezt az értéket használja. A 11.2.6 pont kvantifikált feltételének példái mind korrelálatlan szubszelektet tartalmaznak.

Korrelált szubszelekt például a 0 dolgozót foglalkoztató osztályok meghatározására szolgáló belső kiválasztás (11.2.6 pont EXISTS példája), mert értéke függ attól, hogy a külső SELECT melyik sorát vizsgáljuk. A korrelált szubszelektet a rendszer a külső SELECT minden új sorára újra elvégzi. Ezért korrelált szubszelektet tartalmazó SELECT utasítások végrehajtása lényegesen lassabb, mint a hasonló jellegű, de korrelálatlan szubszelektet használó utasításoké.

Amikor összetett kérdéseket fogalmazunk meg, akkor ezt a hatékonysági szempontot mindig vegyük figyelembe.

11.3 Adatváltoztató utasítások

11.3.1 Az adatváltoztató utasítások közös jellemzői

A változtató utasítások táblázatok, nézetek *tartalmát* változtatják meg.

Ez jelentheti

- meglevő sor(ok) törlését (DELETE),
- új sor(ok) beírását (INSERT),
- meglevő sor(ok) egy vagy több oszlopának módosítását (UPDATE).

Az utasítást csak az adatbázis-felügyelő, a táblázat/nézet tulajdonosa hajthatja végre, vagy az akinek erre a relációra a tulajdonos ilyen engedélyt adott.

Ha nézetten keresztül változtatunk, akkor ez az alaptáblázat megváltoztatását jelenti.

Ha táblázatban változtatunk, akkor ez automatikusan megjelenik az összes ráhivatkozó nézetben is.

Nem lehet változtatni sem „csak olvasható” nézetten keresztül, sem a rendszerkatalógusban. Ha az adatbázis szerkezetének vagy adatainak változtatása miatt a katalógusban is változtatni kell, az ahhoz szükséges utasításokat a rendszer automatikusan generálja és végrehajtja.

A véglegesítési lehetőséggel (COMMIT, l. 12.5 pont) rendelkező rendszerek a változtatást egy utasításnak tekintik, akárhány sort érint is. Ez azt jelenti, hogy vagy mindegyik kiválasztott sor törlődik, beíródik, módosul vagy egy sem. Ez utóbbi esetben a felhasználó hibajelzést kap. Ez fordulhat elő például kereszthivatkozásokor, ha a korlátozott törléssel definiált idegen kulcs miatt a WHERE-rel kiválasztott sorok bármelyikét nem lehetne törölni, vagy módosítani.

Ugyancsak előfordulhat ez, ha egy INSERT utasítással több sort viszünk be az adatbázisba, de ezek között van olyan, amelyiknek bevitelét például valamelyik kötelezően kitöltendő oszlop értékének hiánya vagy egyedi indexnek definiált oszlopba duplikált érték bevitelére miatt a rendszer nem engedi meg.

A továbbiakban ismertetjük a párbeszédes SQL rendszerből kiadható adatváltoztató utasítások legfontosabb jellemzőit. A programból kiadható, ezektől némileg eltérő változataikat a 14. fejezetben tárgyaljuk.

11.3.2 Törlés (DELETE utasítás)

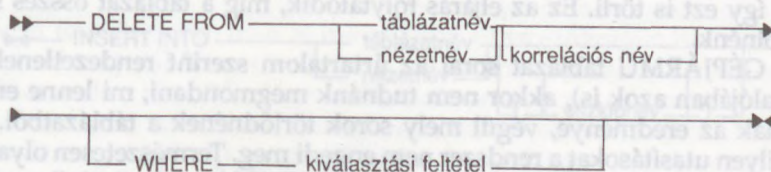
A DELETE utasítás sor(oka)t töröl egy táblázatból vagy nézetből.

Szintaxis diagrammját a 11.5 ábrán láthatjuk. A kiválasztási feltétel a 11.2.6 pontban megadott lehetőségek bármelyike lehet. (Az utasítás egy másik, csak programból használható formáját a 14.7.4 pontban ismertetjük.)

Ha nem adunk meg WHERE klauzulát, akkor az utasítás a táblázat/nézet összes sorát törli, különben csak azokat, amelyekre a WHERE feltétel teljesül.

A táblázat/nézet az összes sor törlése után is megmarad. Ennek megszüntetésére a DROP utasítás szolgál.

Ha a táblázatra idegen kulccsal más táblázat is hivatkozik, akkor annak sorai az idegen kulcs definiálásakor megadott módon változnak meg. A törölt kulccsal egyező értékű idegen kulcsot tartalmazó sorok a hivatkozó táblázatból szintén



11.5 ábra A DELETE utasítás szintaxis diagrammja
(A kiválasztási feltétel részletezését lásd a 11.3 ábrán.)

törölődnek (ON DELETE CASCADE) vagy a null-értéket veszik fel (SET NULL), illetve nem lehet az eredeti DELETE utasítást végrehajtani (RESTRICTED), mert a törlendő értékre korlátozott feltétellel hivatkoznak.

A DELETE FROM G&T

WHERE TULAJDONJOG-VEGE < '1970-01-01'

utasítás törli az összes olyan régi adatot a G&T táblázatból, ahol a gépjárművet 1970. január 1. előtt adták el.

Mivel a G&T táblázatra nem hivatkozik más táblázat, ezek a rekordok mindenképpen törölődnek az adatbázisból, feltéve, ha a DELETE utasítás kiadója jogosult volt a G&T táblázaton ezen művelet elvégzésére.

Ha a kiválasztási feltétel szubszelektet tartalmaz, akkor az nem hivatkozhat a törlendő relációra. Ennek az a magyarázata, hogy az eredmény ekkor függene az egyes sorok törlésének sorrendjétől.

Ezt a következő, hibás DELETE utasításon mutatjuk be. (Ilyen utasítást az SQL értelmező el sem fogad, szintaktikus hibát jelez.)

A GÉPJÁRMŰ táblázatból törölni akarjuk a legkisebb úrtartalmú gépjárműve(ke)t. Ezt a következő, a szintaktikának a fenti kiegészítés nélkül formailag megfelelő utasítással próbáljuk meg:

DELETE FROM GÉPJÁRMŰ

WHERE ÚRTARTALOM =

(SELECT MIN(ÚRTARTALOM) FROM GÉPJÁRMŰ)

Miután a táblázat sorainak rendezettségéről semmit sem tétélezhetünk fel, nem tudjuk, hogy az utasítás milyen sorrendben halad végig a GÉPJÁRMŰ táblázat egyes sorain. Nézzünk két szélsőséges esetet, amikor a végrehajtás

- úrtartalom szerint csökkenő,
- növekvő sorrendben fog lezajlani.

Miután a szubszelekt korrelált, a GÉPJÁRMŰ táblázat minden egyes sorának megvizsgálásához a rendszer újra kiszámolja a szubszelekt által kiválasztott aktuális minimumot, és ennek értéke alapján törli vagy nem törli a sort. Ez az a esetben azt jelenti, hogy egészen addig, amíg a táblázat végére, a legkisebb úrtartalomhoz nem érünk, egy sort sem törölünk. Az utasítás valóban csak a legkisebb úrtartalom értékkel rendelkező sort vagy sorokat törli.

A b) esetben viszont rögtön kitörli az első megvizsgált sort, hisz abban az úrtartalom megegyezik a minimummal. A második sort már az így módosított táblázat minimális úrtartalmával hasonlítja össze, ami természetesen ismét

egyezik, így ezt is törli. Ez az eljárás folytatódik, míg a táblázat összes sorát ki nem törölnénk.

Ha a GÉPJÁRMŰ táblázat sorai az úrtartalom szerint rendezetlenek (mint ahogy valójában azok is), akkor nem tudnánk megmondani, mi lenne ennek az utasításnak az eredménye, végül mely sorok törölődnének a táblázatból. Éppen ezért az ilyen utasításokat a rendszer nem engedi meg. Természetesen olyan szubszelektet, amelyben a kiválasztás nem a törlendő táblázatból történik, megadhatunk.

Így például, ha a GÉPJÁRMŰ táblázatból törölni akarnánk az összes olyan járművet, amelyeknek nincsen tulajdonosa, azt a következő utasítással érhetnénk el:

```
DELETE FROM GÉPJÁRMŰ G
WHERE G.RENSZAM IN
  (SELECT RENDSZAM FROM G&T
   WHERE TULAJDONOS='NINCSEN')
OR NOT EXISTS
  (SELECT RENDSZÁM FROM G&T GT
   WHERE GT.RENSZÁM=G.RENSZÁM
   AND TULAJDONJOG-VEGE IS NULL)
```

(Azt, hogy egy járműnek éppen nincsen tulajdonosa, a G&T táblázatban kétféle módon ábrázolhatjuk. Vagy 'NINCSEN' lesz a TULAJDONOS oszlop értéke, vagy a TULAJDONJOG-VEGE oszlopnak az adott RENDSZÁM minden sorában van értéke.)

11.3.3 Adatbevitel (INSERT utasítás)

Az INSERT utasítás sor(oka)t helyez be egy táblázatba vagy nézetbe.

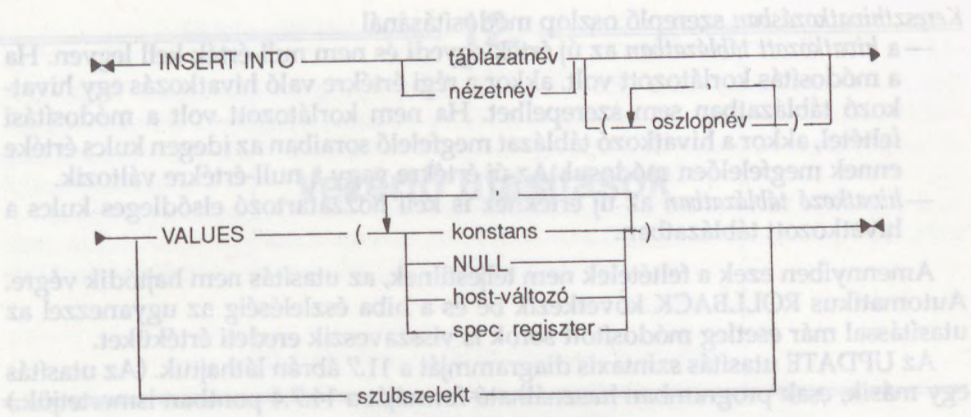
Ha a nézet létrehozásakor nem adtuk meg az ellenőrző klauzulát (WITH CHECK OPTION), akkor a nézetten keresztül olyan sor is beírható az alaptáblázatba, amelyik a nézet feltételeinek nem felel meg.

Beíráskor formátum- és integritás-ellenőrzés történik. Numerikusként definiált oszlopba csak numerikus, karakteresbe csak karakter érték vihető be. (A különböző ábrázolási formák között a rendszer átkonvertál.) Ha a bemenő adat bármelyik mezője nem fér el a megfelelő oszlopban (kivételek a tizedesjegyek), akkor az utasítás nem hajtódik végre; a sor nem kerül be a táblázatba.

Ugyanez érvényes, ha egy oszlopra vagy oszlopkombinációra egyedi indexet (vagy elsődleges kulcsot) definiáltunk és oda egy már létező értéket, vagy kereszt-hivatkozásnál olyan idegen kulcsértéket akarunk bevinni, amelyek nem léteznek a hivatkozott táblázatban.

Az INSERT utasítás szintaxis diagrammját a 11.6 ábrán láthatjuk. A VALUES formátummal egy sort, a szubszelekttel az általa kiválasztott összes sort bevisszük a táblázatba/nézetbe.

Ha nem sorolunk fel oszlopokat, akkor a táblázat/nézet minden oszlopába beírunk adatokat. Ha felsorolunk, akkor csak a felsoroltakba kerülnek adatok. A



11.6 ábra Az INSERT utasítás szintaxis diagrammja
(A kiválasztási feltétel és a kifejezés részletezését lásd a 11.3 ábrán.)

többi oszlop értéke—a táblázat definíciójától függően—a null-érték vagy az alapértelmezés lesz.

A VALUES után megadott értékek sorrendje illetve a szubszelekt kiválasztási listájának oszlopsorrendje meg kell egyezzen az INSERT-ben explicit megadott oszlopsorrenddel, vagy ha ez nincs, akkor a CREATE TABLE utasítás oszlopsorrendjével.

A DELETE-hez hasonló okok miatt a szubszelekt itt sem hivatkozhat arra a táblázatra/nézetre, amelyikbe be akarunk írni.

A G&T táblázatba egy új gépkocsi tulajdonjogát pl. a következő két utasítás bármelyikével bevihetjük:

```
INSERT INTO G&T
```

```
VALUES ('CXX465','Akadémiai Kiadó','1992-12-01', NULL)
```

vagy

```
INSERT INTO G&T(RENSZÁM,TULAJDONOS,TULAJDONJOG-  
KEZDETE) VALUES ('CXX465','Akadémiai Kiadó','1992-12-01')
```

Természetesen, ha a GÉPJÁRMŰ táblázatban nem létezik a 'CXX465' rendszámú gépkocsi vagy a TULAJDONOS-ban az 'Akadémiai Kiadó', vagy ez a kulcs-kombináció már szerepel a G&T táblázatban, akkor ezt a sort nem tudjuk beírni az integritás feltételek miatt.

11.3.4 Módosítás (UPDATE utasítás)

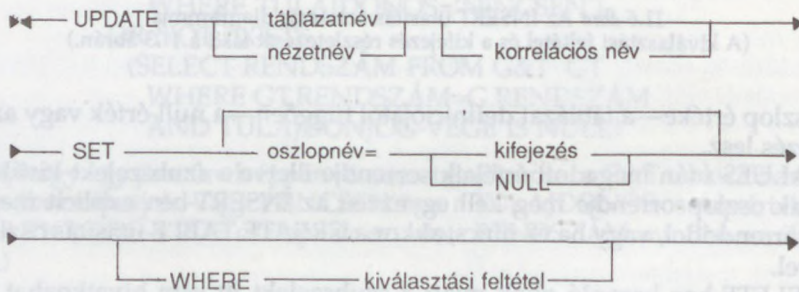
Módosítja a megadott táblázat vagy nézet egy vagy több sorának meghatározott oszlopát vagy oszlopait. A módosítandó sorokat a WHERE feltétellel, az oszlopokat a SET klauzulával választjuk ki. Ha nem adunk meg WHERE klauzulát, akkor minden sort módosítunk, különben csak azokat, amelyekre a feltétel teljesül. Így speciális esetben előfordulhat, hogy egy sort sem módosítunk.

Kereszthivatkozásban szereplő oszlop módosításánál

- a hivatkozott táblázatban az új érték egyedi és nem null-érték kell legyen. Ha a módosítás korlátozott volt, akkor a régi értékre való hivatkozás egy hivatkozó táblázatban sem szerepelhet. Ha nem korlátozott volt a módosítási feltétel, akkor a hivatkozó táblázat megfelelő soraiban az idegen kulcs értéke ennek megfelelően módosul. Az új értékre vagy a null-értékre változik.
- hivatkozó táblázatban az új értékhez is kell hozzátartozó elsődleges kulcs a hivatkozott táblázatban.

Amennyiben ezek a feltételek nem teljesülnek, az utasítás nem hajtódik végre. Automatikus ROLLBACK következik be és a hiba észleléséig az ugyanezzel az utasítással már esetleg módosított sorok is visszaveszik eredeti értéküket.

Az UPDATE utasítás szintaxis diagrammját a 11.7 ábrán láthajtuk. (Az utasítás egy másik, csak programban használható formáját a 14.7.4 pontban ismertetjük.)



11.7 ábra Az UPDATE utasítás szintaxis diagrammja
(A kiválasztási feltétel és a kifejezés részletezését lásd a 11.3 ábrán.)

Így például, ha Lakinger Béla foglalkozása megváltozik, akkor ezt a TULAJDONOS táblázatban így követhetjük:

```

UPDATE TULAJDONOS
SET FOGLALKOZÁS='Nyugdíjas tengerész'
WHERE TULAJDONOS='Lakinger Béla'
  
```

Ha a nézet definíciójakor nem adtuk meg a kötelező ellenőrzést, akkor nézetben keresztüli módosítással olyan adat is bekerülhet a táblázatba, amelyik a módosított nézetben nincsen benn.

Ha az adatbázis-kezelő rendszer beépíthető mezőellenőrzést is lehetővé tesz (pl. DB2, SQL/DS Rel.3), akkor ez minden INSERT és az ellenőrizendő mezőt érintő UPDATE utasításra aktivizálódik. Amennyiben a beírandó ill. módosítandó mező értéke nem felel meg az ebben a rutinban leírt feltételeknek, az utasítás nem hajtódik végre.

Vezérlő utasítások

12.1 Általános ismertetés

A vezérlő utasítások segítségével

- A táblázat/nézet *tulajdonosa* engedélyezheti az objektum különböző módon történő használatát (GRANT);
- Visszavonhatja a korábban megadott jogokat (REVOKE);
- A felhasználó addig, amíg dolgozik velük, elzárhat más felhasználók elől különböző objektumokat (LOCK);
- Több lépésből álló utasításcsoportot jelölhet ki és ennek eredményét az összes utasítás végrehajtása után véglegesítheti (COMMIT), vagy visszaállíthatja a kiinduló állapotot (ROLLBACK);
- A teljesség kedvéért megemlítjük a csak programból használható WHENEVER utasítást is (l. 14.5 pont), melynek segítségével vezérelhetjük, hogy az SQL utasítás eredményétől függően hol folytatódjék a program;
- Ezenkívül— főleg nagy számítógépes SQL rendszerekben— az adatbázis-felügyelő engedélyezheti az egyes felhasználóknak, hogy fizikailag mit változtathatnak az adatbázis szerkezetén (pl. táblázatokat létrehozhatnak-e), vagy visszavonhatja ezeket (GRANT, REVOKE speciális formái).

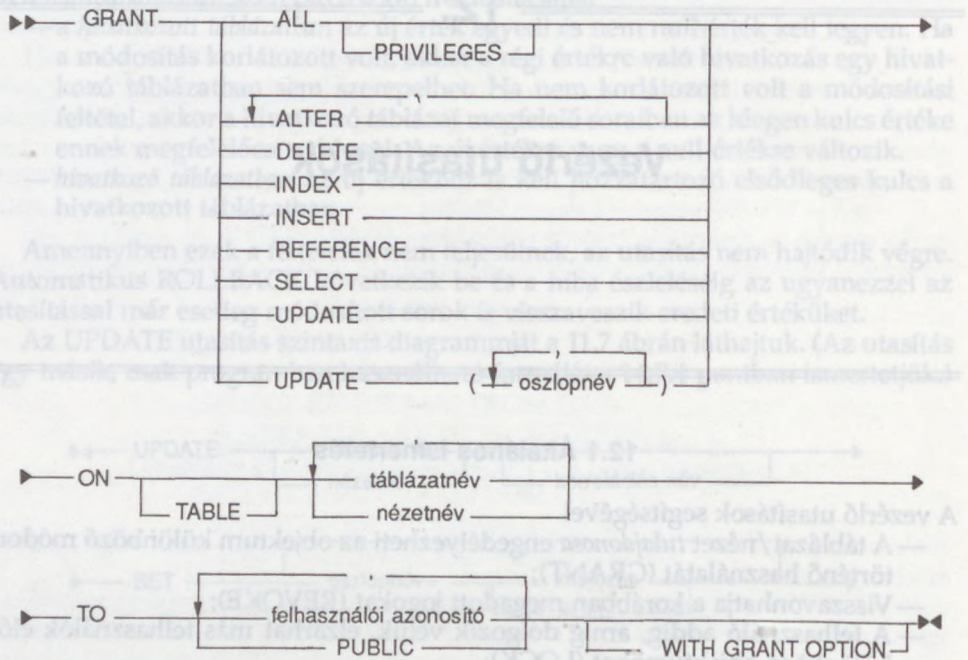
12.2 Jogosultság megadása (GRANT utasítás)

A GRANT utasítás meghatározott műveletekre meghatározott felhasználóknak

- privilégiumokat, jogosultságot ad meghatározott táblázatokon, nézeteken,
- engedélyezi SQL programok futtatását,
- engedélyezi az egész adatbázis állapotának megváltoztatását (pl. önálló adatbázis terület, táblázat létrehozását, megszüntetését stb.). Ez utóbbi erősen rendszerfüggő, ezért ezzel itt nem foglalkozunk.

Az első lehetőséget biztosító GRANT utasítás szintaxis diagrammját a 12.1 ábrán láthatjuk.

Az ALL opció minden jogosultságot megad a felsorolt táblázatokon/nézeteken a névszerint felsorolt felhasználóknak, vagy mindenkinek (PUBLIC). Tételes fel-



12.1 ábra A GRANT utasítás szintaxis diagrammja

sorolásnál kizárólag a felsorolt privilégiumokat adjuk meg. Ha a WITH GRANT OPTION-t is megadtuk, akkor a jogosult az így kapott privilégiumokat bárki másnak tovább adhatja illetve visszavonhatja tőle.

Az adatbázis-felügyelő a tulajdonos engedélye nélkül is rendelkezik a GRANT utasításban megadható összes privilégiummal, de ezeket a tulajdonos helyett általában nem adhatja tovább másoknak, csak ha ő is megkapta explicit a WITH GRANT OPTION kiegészítéssel.

Azt, hogy adatkezelő és az ALTER TABLE utasítást az adatbázis-felügyelőn kívül más csak a tulajdonos engedélyével adhatja ki egy adott táblázatra, nem kell különösképpen indokolni. Ez magától értetődő biztonsági és adatvédelmi alapkövetelmény. Nem olyan nyilvánvaló azonban, hogy miért kell jogosultság index készítésére, vagy idegen kulccsal való hivatkozásra. Gondoljuk meg, mi történik, ha valaki tudtunk nélkül korlátozott törléssel hivatkozik a táblázatunkra vagy egyedi indexet épít fel rá, holott a mi alkalmazásunk ezt nem igényli. Ebben az esetben képtelenek lennénk törölni egy számunkra már fölösleges, de más által hivatkozott sort vagy egy duplikált kulcsú sort bevinni. Ráadásul, jó védelemmel rendelkező rendszerben lehet, hogy meg sem szüntethetnénk a bennünket zavaró indexet, ki sem törölhetnénk a hibát okozó fölösleges sort, mert arra a tulajdonosától nem kaptunk engedélyt.

12.3 Jogosultság visszavonása (REVOKE utasítás)

A GRANT utasítással megadott jogosultságok a REVOKE utasítással vonhatók vissza. Formája lényegében azonos a GRANT utasítás formájával, csak értelem-szerűen a TO „kinek” helyett FROM „kitől” a kötelező kulcsszó a felhasználók felsorolása előtt.

Ha egy jogosultságot valaki több helyről is megkapott (pl. PUBLIC és név szerint), akkor ha az egyiket visszavonják tőle, a másik révén még mindig jogosult a táblázaton/nézeten az engedélyezett műveletre.

12.4 Objektumok ideiglenes elzárása

12.4.1 Zárak kezelése

Nagy adatbázisokon rendszerint több felhasználó is dolgozik egyidejűleg, gyakran ugyanazokkal az adatokkal. Addig, amíg mindenki csak olvasni kívánja ezeket, nincs probléma, hiszen ha többen nézik ugyanazt, attól abban még nem változik semmi.

Amint azonban bármelyik felhasználó változtatni kíván akármit a többiek által nézett adatokon, felmerül a probléma, mit láthatnak a többiek vagy mit változtathatnak, ha ők is változtatni akarnak. Az eredeti adatokat vagy a módosítás utánit? Egyidejű változtatásnál melyik történjen előbb, ugyanarra a sorra párhuzamosan, oszloponként történjen-e a változtatás? Ha a változtatás egy táblázat több sorát érinti, akkor ezalatt egy másik, ezzel egyidejűleg dolgozó felhasználó a sorok egy részénél a régi, más részükénél pedig már az éppen megváltozott adatokat kapná meg, amiből nagy valószínűséggel téves következtetéseket vonna le. Például az egyik utasítás minden alkalmazott fizetését növeli 1000 Ft-tal (UPDATE ALKALMAZOTTAK SET FIZETÉS = FIZETÉS + 1000), egy másik pedig ugyanakkor fizetési statisztikát készítene ugyanerről a táblázatról (SELECT AVG(FIZETÉS), ... FROM ALKALMAZOTTAK GROUP BY ...).

Attól függően, hogy a csoportképzés pillanatában a csoport hány tagjánál történt meg a fizetésemelés, más és más eredményt kapnánk. Ez nyilván nem engedhető meg. Ezért az adatbázis-kezelő rendszer részben automatikus, részben a felhasználó által vezérelhető védelmet nyújt az ilyesfajta, az egyidejű hozzáférésből származó inkonzisztenciák ellen.

Erre a célra szolgálnak a **zárak**.

Kétfajta zárat hozhatunk létre:

— osztott zár (SHARE)

Bárki olvashatja az így elzárt adatokat, de senki nem változtathat rajtuk.

— kizárólagos zár (EXCLUSIVE)

Senki más nem férhet hozzá az így elzárt adatokhoz. Nem olvashatja és nem is változtathatja meg.

Az adatbázis-kezelő rendszer automatikusan

— adatbázis-területeket,

— adatlapokat,

— sorokat zárhat el.

Nyilvánvaló, hogy a finomabb zárolási mechanizmus nagyobb adminisztrációs terhet ró a rendszerre, de kevésbé gátolja a párhuzamos munkát.

Az automatikus zárkezelésen kívül a felhasználónak lehetősége van arra, hogy a LOCK TABLE utasítással az általa használni kívánt táblát akár osztott, akár kizárólagos módon, teljes egészében elzárja más felhasználók elől. Ezt azonban csak akkor használjuk, ha tényleg létfontosságú, hogy addig, amíg mi dolgozunk a táblával, senki más ne olvassa vagy ne változtasson benne.

A felhasználó által kiadott LOCK utasítás általában az általa véghezvitt változtatások véglegesítéséig, a következő COMMIT (l. 12.5 pont) vagy ROLLBACK utasítás (l. 12.6 pont) kiadásáig van érvényben.

A rendszer automatikusan a következő módon zárol a megadott zárszintre (terület, lap, sor):

- Olvasásnál bárki olvashatja. Változtatni csak akkor nem lehet, ha az olvasási kérést úgy adták ki, hogy azt megismételve ugyanazt az eredményt szolgáltatassa (Repeatable Read). Ilyenkor addig, amíg az olvasási utasítást kiadó felhasználó (program) nem fejezi be a kiválasztott adatok megtekintését (programban a feldolgozását), senki más nem változtathat azokon. Ez a mód akkor lehet hasznos, ha a megtekintett adatok alapján valamilyen hosszabb időt igénybe vevő döntést kell hoznunk és biztosak akarunk lenni, hogy az információk megkapása és a döntés meghozatala közti időben a döntéshez szükséges adatok nem változnak meg. (Pl. raktárnyilvántartásnál egy sok tételes rendelés teljesítésénél, vagy egy helyfoglaló rendszerben több átszállást igénylő csoportos helyfoglalás visszaigazolásánál.)

- Változtatáskor kizárólagos zárat helyez el a megfelelő zárszinten. Éppen ezért igyekezzünk a változtató utasítások hatását minél hamarabb véglegesíteni vagy úgy, hogy a COMMIT vagy ROLLBACK utasítást mihelyt lehet kiaduk, vagy pedig úgy, hogy (elsősorban párbeszédes munkáknál) minden utasítás befejezése automatikusan egy COMMIT illetve hiba esetén ROLLBACK kiadását és végrehajtását is jelenti.

Ha olyan objektumhoz akarunk hozzáférni, amelyet más felhasználó éppen elzárt, akkor a rendszertől és működési paraméterezéseitől függően

- várunk, amíg az objektum felszabadul, utána az utasításunk végrehajtódik;
- adott ideig várunk. Ha ezalatt az objektum nem szabadul fel, hibaüzenetet kapunk. Ha felszabadul, akkor utasításunk végrehajtódik;
- azonnal hibaüzenetet kapunk. Az utasítást meg kell ismételni.

12.4.2 Patthelyzet

Az exkluzív zárok alkalmazásával sajnos *patthelyzet* alakulhat ki az adatbázisban. Ekkor két vagy több felhasználó vagy folyamat vár olyan lefoglalt erőforrásokra, amelyek normális üzemmód mellett sohasem lesznek a számukra elérhetőek. Ezáltal úgy tartják fel egymást, hogy egyikük sem tud továbblépni. Tipikus példa erre, amikor két felhasználó dolgozik egyidejűleg, és az egyik exkluzív elzárja a sorok (rekordok) A, a másik a B halmazát. Ha most az elsőnek az A sorok feldolgozásához szüksége volt B-ből is egy vagy több rekordra, míg a másíknak B feldolgozásának befejezéséhez A-beli rekord kell, akkor mindkettőnek várni kell.

Ezáltal egyik felhasználó sem tud tovább dolgozni. A rendszer számukra működésképtelenné vált.

Patthelyzet nem csak adatbázisokban jöhet létre. Kialakulásának megakadályozására többféle módszer ismeretes. Ezeket azonban az adatbázis-kezelő rendszerekben nem igen alkalmazzák, mert a feldolgozást igen lelassítanák. A legtöbb rendszer a pattveszélyt, mint szükséges rosszat tudomásul veszi és észleli, ha megvalósul. Ekkor valamelyik feldolgozás kényszerített befejezésével és egy korábbi állapotába való visszaállítással a továbbhaladáshoz szükséges erőforrásokat felszabadítja. Ezzel a patthelyzetet feloldja.

12.5 Változtatások véglegesítése (COMMIT utasítás)

Amikor egy adatbázisban változtatásokat hajtunk végre, azokat a rendszer mindaddig ideiglenesnek tekint, amíg egy ténylegesen vagy burkoltan kiadott COMMIT utasítással nem véglegesítjük vagy egy ROLLBACK utasítással vissza nem állítjuk a kiindulási állapotot. Ennek az a magyarázata, hogy nincsen értelme például egy hibás sort, amelynek mondjuk egy numerikus adata nem fér el a megfelelő oszlopban, beírni az adatbázisba. Célszerűbb ilyenkor az egész sor bevitelét jó adatokkal megismételni. Ugyanígy, ha egy változtató utasítás több sort érint, és ez az utasítás valamiért nem hajtható teljesen végre, tisztább helyzetet teremtünk, ha nem azt próbáljuk kideríteni, mely sorokat sikerült már jól megváltoztatnunk és melyeket nem, hanem visszaállunk az utasítás előtti, egyértelműen definiált állapotba. Ezután, innen kiindulva előbb megállapítjuk a hiba okát. Kijavítjuk azt és utána egyben, most már (remélhetőleg) hiba nélkül elvégezzük a korábbi utasítást.

Mint már említettük, elsősorban párbeszédes munkánál a rendszer alapműködési módja az, hogy minden egyes utasítás hibátlan végrehajtása után a változtatások véglegesítődnek. Előfordul azonban, hogy a feldolgozásunk eredménye csak akkor véglegesíthető, ha egy több utasításból álló tranzakció minden lépése hibátlanul végrehajtott. Például egy banki átutalás csak akkor tekinthető teljesnek, ha a kifizető számlájáról levették az összeget (egy UPDATE utasítás), ezt jóváírják a címzett számláján (újabb UPDATE) és a pénzmozgást is rögzítették pl. a banki jutalék miatt (egy INSERT utasítás). Ilyen esetekben a felhasználó felfüggesztheti az utasításonkénti automatikus véglegesítést és saját maga mondja meg a COMMIT utasítás kiadásával, mikor kell az utolsó véglegesítési pont (a megelőző COMMIT, ROLLBACK vagy a feldolgozás megkezdése) óta bekövetkezett összes változtatást véglegesíteni. Azt mondjuk ilyenkor, hogy a tranzakció *véglegesítési ellenőrzés*, *Commitment Controll* alatt fut. A fenti példában a két UPDATE és az INSERT utasítás alkot egy logikai munkaegységet. A kezdő COMMIT-ot az első UPDATE előtt, a véglegesítőt az INSERT hibátlan lefutása után kell kiadni.

A Commitment Controll alatt kiválasztott, de még nem véglegesített adatokat más nem módosíthatja, az általunk módosítottakat nem is láthatja. Ezért ezzel a lehetőséggel, hasonlóan a táblázatok elzárásához, csak akkor éljünk, ha okvetlenül szükséges, és akkor is csak annyi időre, amíg az muszáj.

Ahhoz, hogy a COMMIT és a következő pontban ismertetendő ROLLBACK utasítás működni tudjon, a rendszer minden egyes módosításról *naplót* vezet. Eb-

ben följegyzi minden egyes változtatáshoz annak módját, a változás előtti és a változás utáni állapotot. A véglegesítés vagy a kiinduló állapot visszaállítása ebből történik. Ha pedig pont ezalatt lépett fel valami hardver vagy szoftver hiba, amikor ez a véglegesítés/visszaállítás történt, aminek következtében az sem tudott befejeződni, akkor a rendszer újraindításkor a naplózott adatokból automatikusan visszaállítja az eredeti állapotot.

12.6 Utasítás előtti állapot visszaállítása (ROLLBACK utasítás)

Ha egy utasítás folyamán bárminő olyan esemény lépett föl, melynek következtében az utasítás nem hajtható *teljes egészében* végre, úgy az SQL ennek észlelésekor befejezi az utasítás végrehajtását és visszaállítja az eredeti, az utasítás megkezdésekor fennálló állapotot. Ezt hívjuk automatikus ROLLBACK-nek. Ez azért lehetséges, mert a naplóban szerepel minden egyes, az utasítás által megváltoztatott sornak az utasítás megkezdése előtti értéke.

Az automatikus visszaállításon kívül a felhasználónak lehetősége van arra is, hogy ha egy utasítás-sorozat Commitment Control alatt fut, akkor annak bármely pontján explicit kiadja a ROLLBACK utasítást (például észrevette, hogy valamelyik bemenő adatot rossz volt vagy a módosítás nem felel meg a megadott ellenőrzési feltételeknek). Ekkor a rendszer visszaáll a legutolsó COMMIT vagy ROLLBACK pillanatának (vagy a munka kezdetének) állapotába. Ettől kezdve egy új tranzakció naplózását kezdi el.

Megjegyezzük, hogy a személyi számítógépes adatbázis-kezelő rendszerek többsége a folyamatos naplózás tároló és nagy idő-igénye miatt nem rendelkezik ilyen véglegesítési/visszaállítási lehetőséggel. Ezekben a COMMIT és a ROLLBACK utasítást nem lehet használni.

Vannak olyan rendszerek, melyekben az utasításban szükséges még a WORK kulcsszót is megadni.

Ezekben a formátum: COMMIT WORK, illetve ROLLBACK WORK.

További SQL utasítások

13.1 Általános ismertetés

A 10. és 11. fejezetben ismertetett adatleíró és adatkezelő utasítások mindegyik SQL rendszerben megtalálhatók és a 12. fejezet vezérlő utasításait is mind alkalmazhatjuk a nagyobb rendszerekben. Az SQL nyelv és az SQL adatbázisok párbeszédes használatát azonban igen nagymértékben elősegítik az olyan utasítások, melyek lehetővé teszik az egyszer már elkészített utasítások paraméterezhető formában való tárolását és ismételt végrehajtását, a kiválasztás eredményének különféle formában való megjelenítését, információkat szolgáltatnak az adatbázis pillanatnyi üzemeltetési állapotáról, beállítják, illetve kilistázzák a felhasználó saját „adatbázis gépének” aktuális paramétereit, megmutatják, milyen elérési utat generál a rendszer egy adott kiválasztási utasítás végrehajtásához, segítséget adnak az egyes utasítások használatához, leállítják az aktuális utasítást stb.

Az ebben a fejezetben ismertetendő utasítások nem annyira általánosak és szabványosítottak, mint az eddig megismertek. Ezalatt azt értjük, hogy egy részük sok rendszerben (főleg a kisebbekben) nem is található meg vagy ha igen, lehet, hogy nem ezen a néven és/vagy némileg különböző funkciók elvégzésére szolgál. Éppen ezért itt csak röviden ismertetjük, milyen további lehetőségei vannak az SQL alkalmazóknak, ha módjukban áll a kereskedelmi forgalomban kapható nagyobb SQL rendszerek valamelyikével dolgozni. Bár mint mondtuk, igyekszünk az utasításokat lehetőség szerint rendszerfüggetlen módon bemutatni, a példák zöme a „Nagy Kék Világ”, az IBM SQL rendszerein alapszik.

13.2 Operátori parancsok

Az operátori parancsok az egész adatbázis-kezelő rendszer vezérlésére, állapotának lekérdezésére szolgálnak. Egy részük nem egyszerű parancs, hanem parancsok, programhívások sorozata, melyet kizárólag az adatbázis központi üzemeltetője adhat ki. Ezek közé tartozik például az egész rendszer elindítása, leállítása, archiválása, új lemezegységek beillesztése az adatok tárolására stb.

Az SQL rendszer indításakor meg lehet például határozni az egyidejűleg aktív felhasználók maximális számát, a lemezre író, illetve onnan olvasó műveletek eredményének pufferekésére szolgáló tároló területet, a rendszer által használt karakterkészletet. Leállításakor eldöntheti az operátor, hogy azonnal leálljon-e a rendszer (az éppen dolgozó felhasználóknál az egy ROLLBACK-et jelent a folyó utasításra) vagy tiltsa meg újabb műveletek elkezdését és a folyó műveletek befejezése után álljon csak le, archiválja-e az adatbázis vagy a napló pillanatnyi állapotát, vagy sem.

Ezen parancsok mellett vannak olyan operátori parancsok is, amelyeket az adatbázis-felügyelő vagy akár az egyszerű felhasználó is kiadhat. Ezek közül a legfontosabb a SHOW és a COUNTER parancs, amelyek információkat szolgáltatnak a rendszer pillanatnyi állapotáról (SHOW) illetve összefoglalást adnak az eddigi tevékenységekről (COUNTER).

A *SHOW* parancs segítségével megtudhatjuk többek között, hogy

- hányan dolgoznak és éppen mit csinálnak az adatbázisban,
- mik az adatbázis fizikai jellemzői, ebből mi szabad és mi foglalt már,
- milyen zárok vannak éppen érvényben, kik és milyen zárok feloldására várnak,
- mennyire van tele a napló.

A *COUNTER* parancs segítségével megtudhatjuk, hogy a számlálók nullázása óta többek között

- hány SQL utasítással fordultak az alkalmazások az adatbázishoz,
- hány tranzakciót véglegesítettek és hányat állítottak vissza a kiindulási állapotba,
- hányszor kellett felhasználóknak várakozni a zárok miatt, és hányszor alakult ki emiatt patthelyzet,
- hány lap olvasás/írás történt összesen.

13.3 Utasítások tárolása és újrafuttatása

Nagy könnyítést jelent a párbeszédes munkánál, ha ismételten használt utasításokat tárolhatunk, majd egy utasítással elindíthatunk és újra lefuttatunk. Különösen rugalmasá tehetjük a munkánkat, ha ezek az utasítások paraméterezhetők is.

Egyedi utasításokat a STORE utasítással tárolhatunk el, és a START utasítással futtathatunk le újra. Az ERASE utasítással törölhetünk, míg a STORE ... REPLACE opcióval felülírhatunk meglevő utasítást. A LIST SQL utasítás kilistázza az összes (*) vagy a megadott tárolt utasítás szövegét.

Utasítás-sorozatokot egy speciális táblázatban, a ROUTINE táblázatban tárolhatunk el.

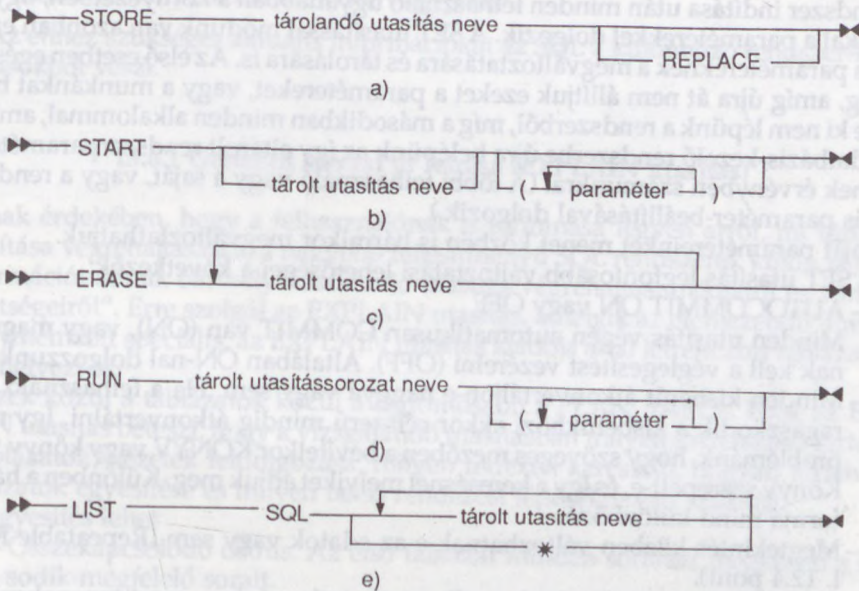
Ennek kötelező oszlopai

- a tárolt utasítássorozat neve (ennek megadásával aktivizálhatjuk),
- a végrehajtandó utasítás pontos megadása,
- az utasítás sorrendje az utasítás-sorozaton belül.

A fentiekén kívül minden utasításhoz megjegyzést is írhatunk.

A ROUTINE táblázatot rendes SQL utasításokkal hozzuk létre, töltjük fel illetve módosíthatjuk tartalmát. Az itt tárolt utasítás-sorozat végrehajtását a RUN utasítással kezdeményezhetjük. Ekkor kell megadni a futáshoz szükséges aktuális paraméterek értékét is.

Az SQL utasítások tárolására és újrafuttatására szolgáló utasítások szintaxis diagrammját a 13.1 ábrán láthatjuk.



13.1 ábra Tárolt SQL utasítások kezelésére szolgáló utasítások szintaxis diagramja
 a) utasítás tárolása; b) tárolt utasítás elindítása és végrehajtása
 c) tárolt utasítás törlése; d) tárolt utasítássorozat elindítása és végrehajtása
 e) tárolt utasítás szövegének megjelenítése

13.4 Megjelenítési formátum változtatása (FORMAT utasítás)

A SELECT utasítás a szelekt listában megadott formátumban jeleníti meg az eredményt. Gyakran azonban ez a formátum nem megfelelő és az utasítás megismétlése nélkül szeretnénk az eredménysorozatokat és a fejléctet más formában, más csoportosításban látni. Erre szolgál a FORMAT utasítás, mely lehetővé teszi, hogy

- a teljes listának megnevezést adjunk,
- az egyes oszlopokban megváltoztassuk a fejléctet, szélességet, kiírt tizedeseket és értéktelen nullákat,
- kihagyjunk, vagy meghagyjunk oszlopokat,
- egyes oszlopokra csoportosítva hogyan ábrázoljunk, csoport- és totálösszeg legyen-e vagy sem,
- mi jelölje a null-értéket,
- mi válassza el az oszlopokat.

Ezen kívül külön utasítások szolgálnak a több képernyős eredményben az előre- és visszalapozásra, hosszú sorok jobbra illetve balra mozgására, a képernyőtartalom kinyomtatására, a megtekintés befejezésére.

13.5 Saját működési környezet beállítása (SET utasítás)

A rendszer indítása után minden felhasználó ugyanabban a környezetben, ugyanazokkal a paraméterekkel dolgozik. A SET utasítással módunk van azonban ezeknek a paramétereknek a megváltoztatására és tárolására is. Az első esetben egészen addig, amíg újra át nem állítjuk ezeket a paramétereket, vagy a munkánkat befejezve ki nem lépünk a rendszerből, míg a másodikban minden alkalommal, amikor az adatbázis-kezelő rendszerbe újra belépünk az így eltárolt rendszerparaméterek lesznek érvényben számunkra. (A többi felhasználó vagy a saját, vagy a rendszer közös paraméter-beállításával dolgozik.)

Saját paramétereinket menet közben is bármikor megváltoztathatjuk.

A SET utasítás legfontosabb változtatási lehetőségei a következők:

— AUTOCOMMIT ON vagy OFF

Minden utasítás végén automatikusan COMMIT van (ON), vagy magunknak kell a véglegesítést vezérelni (OFF). Általában ON-nal dolgozzunk.

— Minden kisbetűt átkonvertáljon-e nagyra vagy sem. Ha a felhasználó nem ragaszkodik a kisbetűkhöz, akkor célszerű mindig átkonvertálni. Így nincs problémánk, hogy szöveges mezőben a bevitelkor KÖNYV vagy könyv vagy Könyv szerepelt-e, és így a keresésnél melyiket adjuk meg. Különben a három forma mind különböző.

— Megtekintés közben változhatnak-e az adatok vagy sem (Repeatable Read, l. 12.4 pont).

Ha lehet, ne ragaszkodjunk a változtathatatlansághoz, mert megnehezíti az egyidejű használatot.

— Tizedes számok standard ábrázolásmódja (az összes és a tizedesjegyek száma).

— Null-érték jelölése az outputban.

— Az oszlopokat elválasztó karakterek megadása.

— Változó hosszúságú oszlopok megjelenítési hossza.

(Az utolsó négy paramétert a FORMAT utasításban is megadhatjuk. Az azonban csak az éppen aktuális SELECT utasítás megjelenítési formáját változtatja meg, míg a SET utasításban megadottak minden olyan SELECT-re érvényesek, amelyekben nem adunk meg ettől eltérő beállítást.)

A paraméterek aktuális beállítását a LIST SET *, ill. a LIST SET paraméter utasítással kaphatjuk meg.

13.6 Optimális elérési út

Az adatok eléréshez az SQL maga állapítja meg a rendelkezésre álló adatokból az optimális elérési utat. Párbeszédés (interaktív) SQL-nél és a következő fejezetben ismertetendő dinamikus SQL-nél ez közvetlenül az utasítás végrehajtása előtt történik, programban levő statikus utasításnál fordításkor.

Az elérési út meghatározását befolyásolják:

- vannak-e indexek (elsődleges kulcs) a rendezési, csoportosítási, kiválasztási feltételekben szereplő oszlopokra;
- hány sorból áll a táblázat;
- hány különböző érték szerepel az indexben (várhatóan mennyire szelektál egy feltétel);
- több táblázatból való kiválasztásnál az egyesítési oszlopokra van-e index;
- szubszelekt(ek) korrelált(ak)-e vagy sem.

Az ehhez szükséges aktuális információkat az SQL a különböző rendszer-katalógusokból veszi.

13.6.1 Generált elérési út leírása (EXPLAIN utasítás)

Annak érdekében, hogy a felhasználónak is tudomása legyen arról, mi történik utasítása végrehajtásakor, a nagyobb teljesítményű SQL rendszerek külön kérésre információt adnak bármely adatkezelő utasítás végrehajtási módjáról és várható „költségeiről”. Erre szolgál az EXPLAIN utasítás, amelyik az elemezendő utasítás kért jellemzőit speciális, az EXPLAIN utasítás kiadója által létrehozott táblázatokban helyezi el.

Ezek közül a táblázatok közül a legfontosabb a PLAN_TABLE. Ebbe az EXPLAIN utasítás beteszi, hogy a vizsgálandó utasításban milyen sorrendben történik a táblázatok/nézetek feldolgozása, milyen indexen keresztül, hogyan történik a táblázatok egyesítése és milyen belső rendezési lépésekre van szükség.

Az egyesítés lehet

- Összekapcsolódó ciklus. Az első táblázat minden sorához megkeresi a második megfelelő sorait.
- Összeválogatás. Az egyesítési oszlopokra rendezett táblázatokot összeválogatja.

Az egyes sorok elérési/kiválasztási módja lehet

- soros, a fizikai elhelyezés szerint,
- indexelt, adott értékek alapján direkt módon,
- indexek alapján szekvenciális.

Automatikus rendezésre lehet szükség az ORDER BY, GROUP BY és DISTINCT megadása miatt.

Az utasítás lebontás után általában több lépcsőből áll. Ezek mindegyikéről egy külön sor kerül be a PLAN_TABLE-be. Ezek az értékek az EXPLAIN utasítás kiadásakor érvényes állapotot tükrözik. Ha közben megváltoznak az elérési utak, akkor párbeszédes vagy dinamikus SQL-ben, valamint újra lefordított programokban lehet, hogy más lesz a feldolgozás rendje.

A feldolgozási sorrenden kívül az EXPLAIN utasítás más táblázatokban információkat helyezhet el

- A feldolgozás várható idejéről. Ez általában csak relatív mérték, különböző kérdések válaszüdejének összehasonlítására használható csak;
- Arról, hogy várhatóan mennyire lesznek szelektívek az egyes oszlopokra a feltételek és hogy használható-e ezekre index;

- Arról, hogy egymásba ágyazott szubszelektnél a belső szubszelekt korrelált-e vagy sem;
- Arról, hogy várhatóan hányszor fognak végrehajtódni a különböző szubszelektek.

Az EXPLAIN utasítás által feltöltött elemző táblázatok adatai alapján bonyolult SQL utasításoknál megállapíthatjuk, hogyan csökkenthetnénk esetleg a végrehajtási időt

- új indexekkel,
- szubszelektek átfogalmazásával,
- speciális esetekben igen szelektív indexek használatának rákényszerítésével (fordításkor a többit töröljük, így a preprocessor kénytelen ezt választani. A visszaállított indexeket ez a program nem használja.),
- összetett SELECT utasítás explicit szétbontásával, az egyes utasítások eredményének ideiglenes táblázatokba való tárolásával, ezekre indexek felépítésével.

13.6.2 Megjegyzések a hatékonysághoz

Az alábbi nagyságrendek közepes- és nagy számítógépre telepített rendszerekre vonatkoznak. Személyi számítógépeknél ezek az értékhatárok egy nagyságrenddel alacsonyabbak lesznek.

Néhány általános szempont, amit nem árt figyelembe vennünk, ha nagyobb adatállományokkal dolgozunk

- Ha a táblázat 10 000 sort tartalmaz, akkor *el kell kezdeni* törődni a hatékonysággal.
 - Ha a táblázat 100 000 sort tartalmaz, akkor már *törődnünk kell* a hatékonysággal.
 - Ha a SELECT után 1000 sort kell rendeznünk, figyelni kell az ORDER BY hatékonyságára.
 - *Egyesítés*kor *kisebb táblázatoknál* is ügyelni kell a SELECT hatékonyságára.
- Néhány további tanács az adatbázis hatékony működtetéséhez.
- Figyeljük, hogy sűrűsödő indexeknél megmarad-e a sűrűsödés. Amennyiben nem, akkor mentsük el a táblázatot és töltsük újra vissza a sűrűsödő indexnek megfelelő sorrendben.
 - A zárák miatti várakozások csökkentése érdekében adatleíró, valamint GRANT, REVOKE és katalógus módosító utasítások után amint lehet, adjunk COMMIT-et.
 - Időnként szervezzük újra a katalógusok indexeit.
 - Ne nézzük a katalógus táblázatokat megismételhető olvasási (Repeatable Read) módban, mert akkor lehet, hogy más nem tud új objektumot létrehozni.
 - Szükség esetén korlátozzuk az adatbázist egyidejűleg használó személyek, programok számát.
 - Nagy táblázatok karbantartását lehetőleg ne csúcsidőben végezzük.

13.6.3 Indexek automatikus használata

Általában indexen keresztül lényegesen gyorsabb a feltételeknek megfelelő sorok kiválasztása. Kivétel ez alól, ha a táblázat jelentős részét kiválasztjuk, és az eredményt nem kell rendezni.

Éppen ezért minden táblázatra legyen legalább egy index és az egyes sorok, az egyedek azonosítására definiáljunk egyedi indexet. (Ezt az elsődleges kulccsal mindenképpen elérjük.)

A táblázatot abban a sorrendben töltsük be, amelyikben leggyakrabban dolgozunk fel, és hagyjunk elegendő helyet a későbbi bővítéseknek is, hogy ne kelljen igen hamar újraszervezni a táblázatot.

Ne készítsünk felesleges indexet, így ha egyéb okokból nem okvetlenül szükséges, ne készítsünk indexet

- gyakran módosított oszlopokra,
- sok duplikátummal (értékeként $> 1\,000$),
- ritkán előforduló keresésekre. (Ha időszakonként mégis nagy keresési aktivitás van egy oszlop szerint, akkor készítsünk erre az időre indexet, melyet utána megszüntetünk.)

Ügyeljünk arra, hogy az indexet az SQL az optimális út generálásakor használni is tudja.

Az indexek kihasználását biztosítjuk, ha

- összehasonlításkor nem kell numerikus konverziót végezni;
- a karakterváltozók hossza megegyezik az összehasonlító állandó jelsorozat (literál) hosszával,
- a LIKE minta nem kezdődik % vagy _ jellel,
- az OR és IN feltételhez ORDER BY-t adunk meg,
- kerüljük azokat a formákat, amikor az SQL eleve nem használ indexeket.

Általában nem használ indexeket az SQL az alábbi esetekben:¹

Numerikus konverzió

Az SQL nem használ indexet, ha az összehasonlítandó pontossága nagyobb, mint az oszlopé.

Pl. ha	A és B SMALLINT típusú,
akkor	... WHERE A < 1.1E1 AND B > 2.0
helyett gyorsabb	... WHERE A < 11 AND B > 2

Stringkészítésnél

Az SQL nem használ indexet, ha az SQL programban a karakter típusú host-változó (l. 14.3 pont) vagy a konstans hossza nagyobb, mint az oszlopé.

Pl. ha	C CHAR(4) és D CHAR(6),
akkor	... WHERE C='0030' AND C<:D

¹Természetesen ez rendszerfüggő. Vannak olyan SQL rendszerek, melyek az itt felsorolt esetek egyikében–másikában felhasználják azért a meglevő indezeket, míg mások még ennél enyhébb feltételek mellett sem tudnak már dolgozni velük.

helyett gyorsabb

```
D1 = D
... WHERE C='0030' AND C:D1,
D1 CHAR(4)
```

ahol

LIKE minta

Ha a LIKE minta % vagy _ jellel kezdődik, az SQL nem tud indexet használni. Különösen lelassítja ez a kiválasztást nagy táblázatokból. Célszerűbb lehet ilyenkor a lehetséges értékeket, ha nem túl sok van belőlük, egy IN (...) feltételben összefoglalni.

OR helyett IN

OR feltételnél az SQL nem használ indexet. Ha ugyanarra az oszlopra vonatkoznak a feltételek, használjunk IN-t, erre tud indexszel dolgozni.

```
Pl.          SELECT * FROM G&T
          WHERE RENDSZÁM='CYX462' OR RENDSZÁM='V24034'
```

helyett jobb

```
          SELECT * FROM G&T
          WHERE RENDSZÁM IN ('CYX462', 'V24034')
```

OR helyett UNION

Ha különböző oszlopokra vonatkoznak az OR-ral összekapcsolt feltételek, célszerűbb UNION-t használni (ha van az oszlopokra index)

```
          SELECT * FROM G&T
          WHERE RENDSZÁM='CYX462' OR TULAJDONOS='Lakinger Béla'
```

helyett jobb

```
          SELECT * FROM G&T
          WHERE RENDSZÁM='CYX462'
          UNION
          SELECT * FROM G&T
          WHERE TULAJDONOS='Lakinger Béla'
```

ORDER BY megadása

Ha egy OR feltételben szereplő oszlopra van index, akkor az SQL index alapján éri el, ha ez a rendezettség (ORDER BY) első oszlopa.

Ha ORDER BY-t adunk, még ha nincs is rá szükségünk, akkor az index alapján választ ki.

Ne használjunk viszont ORDER BY-t, ha nincs ilyen index definiálva, mert akkor az SQL felépít a SELECT-re egy ideiglenes indexet.

13.7 Egyéb utasítások

A teljesség igénye nélkül felsorolunk még néhány hasznos SQL utasítást, melyeket—ha rendelkezésünkre állnak—célszerű rendszeresen használni.

CANCEL

Bármikor, egy másik utasítás futása közben is beadható. Hatástalanítja az aktuális utasítást és automatikus ROLLBACK-et is inicializál.

Különösen akkor hasznos, ha egy hosszú ideig futó utasítás beadása után jöttünk rá, hogy azt egyáltalában nem, vagy másképpen kívánjuk végrehajtani.

HELP

Párbeszédes segítséget nyújt az SQL használatához.

Táblázatként tartalmazza:

- a segítségben elérhető témakörök megnevezését,
- az SQL parancsok szintaktikáját és használati tudnivalókat,
- a párbeszédes SQL (ISQL) parancsok szintaxis diagrammját és használati utasításukat,
- az SQL üzenetek és kódok szövegét.

PRINT

Kinyomtatja a képernyőn megjelent eredményét a SELECT utasításnak.

COMMENT ON

Megjegyzést írhatunk az egyes táblázatokhoz és oszlopokhoz a táblázatok, illetve az oszlopok adatait tartalmazó rendszer-katalógusba.

SQL utasítások beépítése programokba

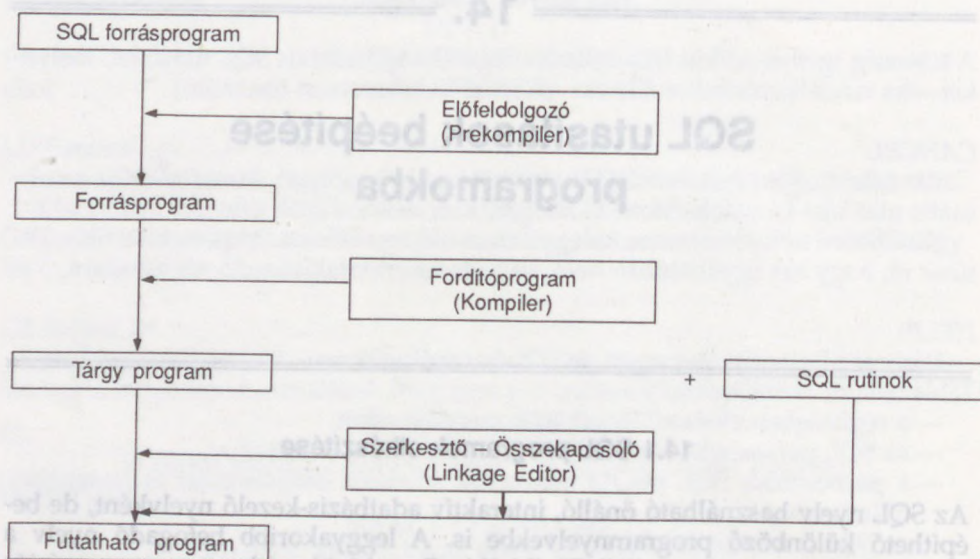
14.1 SQL programok elkészítése

Az SQL nyelv használható önálló, interaktív adatbázis-kezelő nyelvként, de beépíthető különböző programnyelvekbe is. A leggyakoribb befogadó nyelv a COBOL, de gyakran használják PL/I-ből is. Ezen kívül mód van egyes operációs rendszereknél arra, hogy C, RPG, Assembler, sőt akár FORTRAN nyelvű programokba is beépíthessünk SQL utasításokat.

Az SQL utasításokat tartalmazó forrásprogram feldolgozása a következőképpen történik:

1. A megfelelő nyelvű előfeldolgozó (preprocesszor, precompiler) felismeri a forrásnyelvi programokba beépített SQL utasításokat. Szintaktikusan ellenőrzi őket. Ha hibátlanok, akkor átfordítja a befogadó nyelv szintaktikájának megfelelő utasításokká, különben hibaüzenetet küld. Az átfordítás rendszerint egy megjegyzést (COMMENT) jelent a felhasználó informálására, majd egy paraméterezett szubrutin hívását. Maga az utasítás azt szabja meg, hogy melyik szubrutint kell behívni, míg a paramétereket az utasítás paraméterei alapján állítja össze az előfeldolgozó. Ahhoz, hogy az előfeldolgozó a feladatát el tudja végezni, információkat kell a számára szolgáltatni egyrészt magában a forrásprogramban, másrészt a fordítóprogram meghívásakor.
2. A forrásprogramot lefordítják, szerkesztik. A szerkesztőprogram beépíti a futáshoz szükséges SQL rutinokat a betölthető programba.
3. A program futtatásához szükséges a program elkészítőjének engedélye (GRANT RUN) és az, hogy a végrehajtandó lefordított SQL utasításokra vagy a program elkészítőjének, vagy a futtatójának jogosultsága legyen. Amennyiben az SQL utasítás a program futása közben dinamikusan generálódik (l. 14.8 pont), akkor ennek végrehajtásához a program futtatójának a jogosultságát ellenőrzi a rendszer.

Egy SQL program elkészítésének főbb lépéseit a 14.1 ábrán foglaltuk össze.



14.1 ábra SQL program elkészítésének főbb lépései

14.2 SQL utasítások csoportosítása

A 10–13. fejezetben az önálló SQL nyelv legfontosabb utasításait ismertettük. Ebben a fejezetben azokkal az utasításokkal foglalkozunk, amelyeket programból használhatunk. A párbeszédés (interaktív) SQL utasítások az egyéb utasítások kivételével csaknem mind és majdnem változatlan formában kiadhatók programból is. Ennek ellenére a gyakorlatban programokból leginkább csak az adatkezelő utasításokkal és a COMMIT, ROLLBACK, valamint a LOCK utasítással dolgozunk. Az adatleíró utasításokat—mivel ezekre rendszerint csak az objektum létrehozásakor és megszüntetésekor, esetleg megváltoztatásakor van szükségünk—egyszerűbb és kényelmesebb párbeszédéses módban kiadni, mint az egyszeri alkalmazás miatt egy egész programot építeni köréjük. Kivételt ez alól legfőljebb az az eset képez, amikor részeredmények ideiglenes tárolásáról saját magunknak kell a programban gondoskodni és ezért ideiglenes táblázatot hozunk létre. Ezt a táblázatot azonban a program vagy a job befejezése előtt meg kell szüntetnünk (DROP), mert különben a program ismételt futásakor azonos táblázatnév miatt hibajelzést kapunk.

Programból és interaktív SQL-ből is azonosan hívható legfontosabb SQL utasítások a következők:

ALTER...	GRANT
COMMENT	INSERT
COMMIT	LOCK
CREATE ...	REVOKE

DROP... ROLLBACK
EXPLAIN

Programból, és interaktív SQL-ből, de különböző formában is kiadható SQL utasítások az alábbiak:

SELECT illetve SELECT ... INTO
UPDATE illetve UPDATE ... WHERE CURRENT OF CURSOR
DELETE illetve DELETE ... WHERE CURRENT OF CURSOR

Végezetül csak programból használhatók az alábbi SQL utasítások:

BEGIN DECLARE SECTION	FETCH
CLOSE	INCLUDE
DECLARE	OPEN
DESCRIBE	PREPARE
END DECLARE SECTION	PUT
EXECUTE	WHENEVER
EXECUTE IMMEDIATE	

A programba az SQL használatához beépítünk

- SQL utasításokat,
- host változókat. Ezekbe az SQL adatbázisból való beolvasásnál, outputnál értékeket visz be, kiírásnál értékeket vesz ki, melyeket beír az adatbázisba,
- kommunikációs területet. Ebben az utasítások végrehajtásának eredményességét illetve az esetleges hiba okát jelzi,
- utasításokat, melyekkel az SQL utasítások végrehajtásakor bekövetkező előre látott és rendkívüli eseményeket kezelhetjük.

A beépíthető SQL utasításokat többféle módon is csoportosíthatjuk.

Az egyik lehetséges csoportosítás az alábbi:

- *Végrehajtható utasítások.* Ezek tényleges adatbázis-hozzáféréseket generálnak a program fordításakor;
- *Dinamikus,* a program futásakor megadott SQL utasításokat előkészítő és végrehajttató utasítások;
- *Nem végrehajtható utasítások.* Ezek a fordításhoz, vagy nem az adatbázissal közvetlenül kapcsolatos műveletek elvégzéséhez adnak információt.

Az első csoportba tartozó utasításoknak van párbeszédes megfelelője is. A második és harmadik csoport utasításait csak programból adhatjuk ki.

Másik lehetséges felosztás a

- statikus és a
- dinamikus SQL.

Statikus SQL-nél már a program írásakor tudjuk, milyen utasítással dolgozunk. Legfőlegbb egyes paraméterek aktuális értékét kapcsolja végrehajtáskor a rendszer a programbeli host változókhoz. Az adatbázis eléréshez szükséges utat a fordító-program fordításkor hozza létre. Ez—függetlenül az adatbázisban történt változásoktól—egészen addig érvényben marad, amíg a programot újra nem fordítjuk, vagy valamilyen ok miatt (pl. index megszűnt) a generált elérési út nem használható többé.

Dinamikus SQL-nél fordításkor még nem ismerjük az utasítást, vagy olyan fontos paramétere hiányzik (pl. táblázat neve), amely nélkül nem lehet az elérési utat

előre generálni. Az utasítás pontos formája csak futáskor derül ki. Ezért az adatbázis elérési módját a rendszer minden alkalommal dinamikusan, a program futása folyamán állapítja meg.

A harmadik csoportosítási mód az adatkezelő műveletekre vonatkozik aszerint, hogy igényelnek-e kurzort vagy sem.

A hagyományos programnyelvek egyszerre csak egy rekordot tudnak feldolgozni. Az SQL adatkezelő műveletek ugyanakkor egyszerre több sorra, rekordra is vonatkozhatnak. Ebben az esetben szükségünk van egy kurzorra, amelyik megmutatja, hogy az SQL által rendelkezésre bocsájtott rekordok közül most éppen melyikkel dolgozzon a befogadó program. A kurzorral kapcsolatos műveletekkel a 14.7.3 és 14.7.4 pontban foglalkozunk.

14.3 SQL utasítások felismerése

Ahhoz, hogy az előfeldolgozó (preprocesszor) felismerje az SQL utasításokat, a forrásprogramban jelezni kell minden egyes SQL utasítás elejét és végét. Mindent, ami eközött van, SQL utasításnak tekint a precompiler, szintaktikusan ellenőrizi, értelmezi, majd átalakítja a befogadó nyelv utasításává. Ami viszont ezen kívül van, azt változatlanul hagyja. Azt, valamint a precompiler által generált utasításokat a forrásnyelvi fordító fogja feldolgozni.

Ha a precompiler az értelmezés folyamán szintaktikai hibát talál, akkor azt jelzi. Ebben az esetben nem generál forráskódot a nyelvi fordító számára sem. Ugyancsak jelzi azt is, ha olyan SQL műveletet akar a program elvégezni, melyre a program készítőjének nincsen jogosultsága. Ettől azonban a futásra kész program elkészül, mert lehet, hogy a futás idejére már ezt a jogosultságot megkapja vagy nem is a program készítőjének a jogosultságát kell vizsgálni, hanem a futtatójának. Ebben az esetben azonban a lefordított programban van egy jelzés, mely figyelmezteti az adatbázis-kezelő rendszert arra, hogy a program futása közben is ellenőrizni kell minden művelet elvégzésekor a megfelelő jogosultságot.

Az SQL utasítások megjelölésének szintaktikája függ a befogadó nyelvtől. Az utasítás kezdetét COBOL-ban, PL/I-ben (és a legtöbb egyéb befogadó nyelvben is) az

EXEC SQL

jelsorozat jelzi.

Az SQL utasítás végén COBOL-ban az

END-EXEC áll.

(Ha utána újabb SQL utasítás jön, ponttal is lezárva.)

PL/I-ben a véget pontosvessző (;) jelzi.

Ettől, és néhány jelentéktelen eltéréstől az SQL használata minden programnyelvből ugyanaz. Ebben a fejezetben, ha konkrét példákra hivatkozunk, akkor a befogadó nyelv a COBOL lesz, mivel a gyakorlatban ez a legelterjedtebb.

Amennyiben a programban kívülről, már előre megírt SQL utasításokat akarunk bemásolni, arra csak az INCLUDE SQL utasítást használhatjuk. Ezt a következő módon építhetjük be a programba:

EXEC SQL

INCLUDE bemásolandó SQL utasításokat tartalmazó file vagy member neve

END-EXEC.

Az SQL kommunikációs terület leírását (l. 14.5 pont) csaknem mindig így másoljuk be a programba.

14.4 Host változók

Igen rugalmatlan lenne a programunk, ha a statikus SQL utasítások mindig ugyanazokra az adatokra vonatkoznának. Host változók segítségével elérhetjük, hogy a statikusan megadott utasítás a program futása közben meghatározott adatokkal, a host változók aktuális értékével kerüljön végrehajtásra.

Az SQL utasításokban és a program többi részében is használható host változókat a forrásnyelvi programban kell deklarálni a forrásnyelv szintaktikája szerint. A legtöbb implementációban ezt a deklarációt a

BEGIN DECLARE SECTION

és az

END DECLARE SECTION

SQL utasítások közé kell tennünk.

A programban ugyanúgy használjuk őket, mint a többi változókat. Az SQL utasításokon belül azonban kettőspontot (:) kell a host változók neve elé írunk. Ez jelzi az SQL precompilernek, hogy ennek a névnek nincs az adatbázisban megfelelője, a befogadó (host) programban van deklarálva.

Használjuk a host változókat a

- WHERE klauzulában operandusként,
- a SELECT utasítás INTO klauzulájában (l. 14.7.2 pont),
- értéként a szelekt listában,
- az UPDATE utasítás SET klauzulájában,
- az INSERT utasítás VALUE klauzulájában.

Az INSERT, DELETE és UPDATE utasításoknak van a megfelelő párbeszédés utasítással teljesen megegyező formája, mivel ezek az utasítás eredményét az adatbázisba írják ki. A SELECT utasítás az adatbázisból a programba olvas be. Ezen értékek tárolásához mindenképpen kell host változókat deklarálni a programban. Ezért a SELECT utasítás párbeszédés formája kissé különbözik a beágyazottól. A képernyőn megjelenő eredményt az INTO klauzula alapján adja át az SQL a programnak. Ez közvetlenül a szelekt lista után áll. Azoknak a hostváltozóknak a szelekt listával egyező sorrendű felsorolását tartalmazza, melyekbe a lista egyes elemeinek értékei beolvasódnak.

Például az

EXEC SQL

SELECT GYÁRTMÁNY,TÍPUS,ÚRTARTALOM

INTO :P-GYÁRTMÁNY,:P-TÍPUS,:P-ÚRTARTALOM

WHERE RENDSZÁM='CYX462'

END-EXEC.

utasítás a 'CYX462' rendszámú gépkocsi gyártmányát ('Mitsubishi'), típusát ('Colt Gt') és úrtartalmát (1298) viszi be a fölsorolt programváltozókba.¹

Néhány példa host-változóra a párbeszédés SQL-lel teljesen azonos adatkezelő utasításokban:

a) EXEC SQL
 DELETE FROM GÉPJÁRMŰ
 WHERE TÍPUS=:TIP-TÖRL
 END-EXEC.

Ha az utasítás végrehajtásakor a TIP-TÖRL változóban az „Escort” érték volt (ezt olvastuk be oda, vagy bármilyen más módon ezt az értéket vette fel), akkor ez törli a GÉPJÁRMŰ táblázatból az összes „Escort” típusú gépkocsit.

b) EXEC SQL
 UPDATE G&T
 SET TULAJDONJOG-VÉGE=CURRENT DATE
 WHERE RENDSZÁM=:ELADOTT-KOCSI
 AND TULAJDONOS=:ELADÓ
 END-EXEC.

A program az utasítás végrehajtása előtt pl. képernyőről kéri az eladott gépkocsi rendszámát és eddigi tulajdonosának nevét. A CURRENT DATE speciális rezervált SQL szó, amely a napi dátumot tartalmazza.

A host változók általában egy oszlop adataihoz kapcsolódva jelennek meg a programban. Ilyenkor a host változót az oszlop definíciójával kompatibilisen kell deklarálni.

Numerikus adatokat az SQL átkonvertál, de túlszordulásnál természetesen hibát jelez.

Karakteres oszlopba való beírásnál annak tartalma teljesen be kell férjen az oszlopba. Más eseteknél, összehasonlításnál azonos hosszra kiegyenlített (levág) illetve beírásnál szóközökkel feltölt.

Meg kell azonban még oldalunk a programban a *null-értékek* kezelését is. Erre szolgálnak a host változókhoz kapcsolódó *indikátorok*, melyeket a programban két byte-os előjeles egész változóknak (PIC S9999 COMP-4 COBOL-ban, FIXED BINARY (15) PL/I-ben) kell deklarálni.

Az SQL utasításban: host-változó: indikátor formában adjuk meg.

Az indikátor hibakezelésre szolgál. Ha nem adunk meg indikátort is, akkor ha végrehajtáskor hiba lép fel, a művelet végén a kommunikációs területen SQLCODE<0 lesz (l. 14.5 pont). Indikátor esetén bizonyos hibák nem adnak negatív SQLCODE-ot. Ehelyett az indikátor fog a művelet végén 0-tól különbözni. Így például a SELECT INTO utasításnál, ha a beolvasás eredményeképpen a host változó értéke null-érték lenne akkor ezt a -1 indikátorérték jelzi.

¹ A programváltozók neve akármi lehet, ami megfelel a befogadó nyelv szintaktikájának. Az olvashatóság érdekében azonban célszerű utalni a névben arra, hogy melyik oszloppal áll kapcsolatban.

Például az

```
EXEC SQL
```

```
SELECT MAX(ÚRTARTALOM) INTO:P-ÚRT:IND-ÚRT
```

```
FROM GÉPJÁRMŰ
```

```
WHERE TULAJDONJOG-VÉGE='1992-12-24'
```

```
END-EXEC.
```

utasítás végrehajtása után, ha adtak el 1992. december 24-én gépkocsi(ka)t, P-ÚRT értéke az aznap eladott maximális úrtartalom, IND-ÚRT pedig 0 lesz. Ugyancsak 0 lesz a művelet végrehajtása után az annak helyességét jelző SQLCODE értéke is.

Ha viszont egy gépkocsit sem adtak el a kiválasztott napon, akkor a MAX(ÚRTARTALOM) a SELECT-ben null-érték lesz. Ennek az értéknek a programban nincsen közvetlen megfelelője, ezt az utasítás után IND-ÚRT= -1 jelzi. P-ÚRT értékét csak akkor használhatjuk fel a programban, ha ellenőriztük, hogy IND-ÚRT=0.

SQLCODE értéke ebben az esetben is 0, hiszen az indikátor változó kijelölésével gondoskodtunk a null-érték kezeléséről. (Hogy ezt valóban meg is tesszük a programban, azzal a rendszer már nem törődik.)

Természetesen, ha nem adtunk meg indikátort :P-ÚRT után, akkor ha ebbe null-érték kerül, arra az utasítás végrehajtása után hibajelzés, SQLCODE<0 figyelmeztet.

Ha karakteres oszlop beolvasásánál csonkítás történt, akkor az indikátor jelzi az oszlop eredeti hosszát, míg a hostváltozóba a rövidített érték kerül. A programban nem kapunk hibajelzést, csak figyelmeztetést.

14.5 Kommunikációs terület (SQLCA)

A programban deklaráljuk az SQLCA nevű (SQL Communication Area) kommunikációs területet. Ezt célszerűen az INCLUDE SQLCA utasítással másoljuk be.

Az SQLCA terület legfontosabb mezője az SQLCODE változó. Ez minden SQL utasítás végrehajtása után jelzi a művelet eredményességét. Értéke

- 0, normál végrehajtás után,
- negatív, hiba esetén,
- pozitív különleges, de nem (okvetlenül) hibát jelentő esemény észlelése esetén (pl. nem talált olyan rekordot, NOT FOUND feltétel).

A negatív SQLCODE önmagában nincsen hatással a program futására. Ezért, ha nem ellenőrizzük le minden egyes SQL utasítás végrehajtása után, hibásan, hibás értékekkel folytatódhat a program.

További fontos mezők SQLCA-ban:

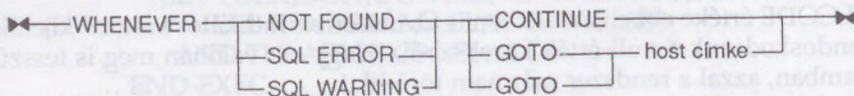
- a hiba természetét leíró rövid szöveg,
- az utasítás által beírt, törölt vagy módosított rekordok száma,
- figyelmeztetések egyes hibát okozható jelenségekre (pl. rövidült a beolvasott oszlop, nem egyezik a szelekt és az INTO lista hossza, nincs WHERE feltétel a DELETE vagy az UPDATE utasításban, null-érték volt AVG, SUM, MAX vagy MIN megállapításánál).

Az egyes mezők neve, és az SQLCA terület pontos felosztása az adatbázis-kezelő rendszertől, sőt ugyanazon rendszernél bizonyos mértékig még a befogadó nyelvtől is függ.

14.6 Különleges események kezelése (WHENEVER utasítás)

Az SQL utasítások végrehajtását ellenőrizhetjük pl. úgy, hogy minden művelet után megvizsgáljuk SQLCODE értékét. Ha ez zérus, akkor folytatjuk a programot, ha nem az, akkor végrehajtjuk a megfelelő hibakezelő rutint (vagy befejezzük a feldolgozást). Ez a módszer azonban nagyon nehézkessé teszi az SQL programok megírását. Egyszerűbb ehelyett a különleges események kezelésére szolgáló WHENEVER SQL utasítás használata.

Az utasítás szintaxis diagrammját a 14.2 ábrán láthatjuk.



14.2 ábra A WHENEVER utasítás szintaxis diagramja

A WHENEVER nem végrehajtható utasítás. Az SQL preprocesszornak és a fordítóprogramnak ad utasítást arra, hogy olyan kódot generáljanak, amelyik a program leírásában (nem futáskor!) minden későbbi SQL utasítás után, annak az eredményétől függően a WHENEVER-ben megadott helytől folytatódjon.

A lehetséges változatok az eredményekre:

- NOT FOUND: Nincs a feltételnek megfelelő sor, vagy kurzorral (l. 14.7.3 pont) való feldolgozásnál nincs már több sor. SQLCODE=100;
- SQLERROR: Bármilyen le nem kezelt hiba. (Pl. nincs jogosultsága a programnak a műveletre, null-érték az eredmény, de nincs indikátor, túlsordulás stb.). SQLCODE<0;
- SQLWARNING: Bármilyen figyelmeztetés, ami nem valószínű, hogy komoly hibát okozna. (Pl. programba való beolvasáskor rövidült egy oszlop, a hibát az indikátor már jelzi stb.). Általában az SQLCA kommunikációs területén jelzi ezt az SQLWARN0='W'értéke, vagy SQLCODE>0 (de≠100, ami a NOT FOUND-ot jelenti).

A meghatározott esemény (NOT FOUND, SQLERROR, SQLWARNING) bekövetkeztekor a program attól függően, hogy az utolsó erre vonatkozó WHENEVER utasításában mi volt érvényben az alábbi helyeken folytatódhat.

- CONTINUE: a következő utasításnál. Ha nem adtunk meg az eredményre WHENEVER-t, akkor ez az alapértelmezés.
- GOTO
vagy GO TO: az ezután megadott címkénél folytatódik a program (a struktúrált programozás kizárólagosságát hirdető, „elméleti programozók” minden átka ellenére!).
- STOP: leáll a program. Ez implicit ROLLBACK-et is maga után von.

Gyakorlati tanácsok:

1. Ha a hibakezelő részben (ide ugrik SQLERROR esetén) is van SQL utasítás, ne felejtjük el ennek elejére megadni a

WHENEVER SQLERROR CONTINUE

utasítást, mert különben ha itt is SQL hiba lépne fel, akkor a program végtelen ciklusba kerül.

2. Ha egészen egyedi hibakezelést igénylő SQL utasításunk van (pl. dinamikus utasításoknál), akkor esetleg célszerű lehet ezeket kivenni a közös SQLERROR

```
EXEC SQL
```

```
WHENEVER NOT FOUND GOTO vege
```

```
END-EXEC.
```

```
EXEC SQL
```

```
WHENEVER SQLERROR GOTO hiba
```

```
END-EXEC.
```

```
...
```

```
EXEC SQL
```

```
... itt NOT FOUND esetén vege-re ugrik
```

```
END-EXEC.
```

```
...
```

```
EXEC SQL
```

```
WHENEVER NOT FOUND GOTO nincs
```

```
END-EXEC.
```

```
EXEC SQL
```

```
... itt NOT FOUND esetén nincs-re ugrik
```

```
END-EXEC.
```

14.3 ábra Példa a WHENEVER utasítás használatára
NOT FOUND esetén, attól függően, hogy az a leírt program melyik részén állt elő,
a *vege* vagy a *nincs* nevű címkére ugrik a program.
SQLERROR (hiba) esetén mindig a *hiba* nevű címkére ágazik el.
SQLWARNING (figyelmeztetés) esetén az alapértelmezéssel,
a következő utasítással folytatódik a program.

ágból. Ezekre külön hibakezelést hajtunk végre az SQLCODE≠0 egyedi levizsgálásával és elemzésével. Ebben az esetben az utasítás előtt adjuk ki a

WHENEVER SQLERROR CONTINUE

utána pedig

WHENEVER SQLERROR GOTO közös hibakezelés

utasításokat

(és ugyanígy az SQLWARNING-ra is).

Ha ezt elfelejtenénk, akkor hiba vagy figyelmeztetés után ennél az utasításnál is a közös hibakezelést hajtánánk végre, illetve az utána jövő, már szabványos hibakezelést igénylő SQL utasításnál is folytatnánk minden hibakezelés nélkül a következő programutasítással a feldolgozást.

A 14.3 ábrán példát láthatunk a WHENEVER utasítás használatára.

Hiba esetén végig a *hiba* nevű címkére ágazik el a program. A kurzor feldolgozás végén, vagy ha nincsen a feltételnek megfelelő sor, akkor attól függően, hogy az a *leírt* program melyik részében levő utasításoknál következik be, a *vége* vagy a *nincs* címkénél folytatódik a program.

Figyelmeztetésre, mivel arra nem adtunk meg WHENEVER utasítást, az alapértelmezés szerinti CONTINUE érvényes.

14.7 Statikus SQL utasítások

14.7.1 Általános jellemzők

A statikus SQL utasításokat akkor használjuk, ha már a program írásakor tudjuk, milyen utasítást, milyen klauzulákkal adunk ki, és ebben milyen táblázatokkal dolgozunk.

Menet közben is változtathatunk azért az utasításon, pl. a SELECT aktuális eredményeivel dolgozunk tovább, mik legyenek az INSERT vagy a WHERE feltétel értékei.

Ezeket a host változók segítségével adjuk meg.

Az adatleíró utasításokat (CREATE, DROP, GRANT stb.) kiadhatjuk statikusan programból is, de mivel a táblázatnév nem lehet változó, sokkal egyszerűbb ezeket interaktívan (esetleg tárolt paranccsal) végrehajtani.

Programban statikusan általában a

- SELECT
- INSERT
- DELETE
- UPDATE

utasításokat használjuk, valamint ha az utasítás több sorra vonatkozhat, akkor az aktuális sor kijelölésére vonatkozó

- kurzor definiáló,
- kurzor használó utasításokat.

Ezek a következők:

- DECLARE CURSOR

- OPEN
- CLOSE
- FETCH
- PUT

14.7.2 Egy sor kiválasztása (SELECT INTO utasítás)

Ha biztosak vagyunk benne, hogy a SELECT utasítás eredménye legfőljebb egy sor lehet, akkor használhatjuk a SELECT INTO formátumot.

Az egyes klauzulákra ugyanaz érvényes, mint a párbeszédés szelektnél (l. 11.2 pont), azzal a különbséggel, hogy a kiválasztott (egyetlen) sor oszlopai rendre az INTO után megadott host változókba kerülnek.

A host változókhoz indikátorokat is megadhatunk.

Az oszlopok és a nekik megfelelő hostváltozók típusa azonos kell legyen.

A megfelelő konverziót, illesztést az SQL automatikusan elvégzi.

Figyelmeztetést kapunk (SQLWARN0)='W'), ha

- karakteres mezőt rövidítettünk. Az indikátor tartalmazza az eredeti hosszt,
- a csoportfüggvénynél a null-érték figyelmen kívül maradt,
- az oszlopok és a hostváltozók száma nem egyezik,
- időaritmetikában szökőév, hóvége miatt módosítás volt, pl. márc 31+1 hó=ápr 30!).

Amennyiben nincs szükségünk kurzorra, akkor a statikus és a párbeszédés utasítások formátuma azonos. Az egyedüli eltérés az, hogy a programbeli statikus utasításban mindenhol, ahol a szintaktika szerint megadható egy kifejezés, ott annak helyén állhat host változó is, míg a párbeszédés utasításokban ez értelem-szerűen nem lehetséges.

14.7.3 Kurzor használata

Kurzorra akkor van szükségünk, ha egy SELECT utasítás több sort adhat eredményül, amelyeket aztán egyenként kell feldolgoznunk.

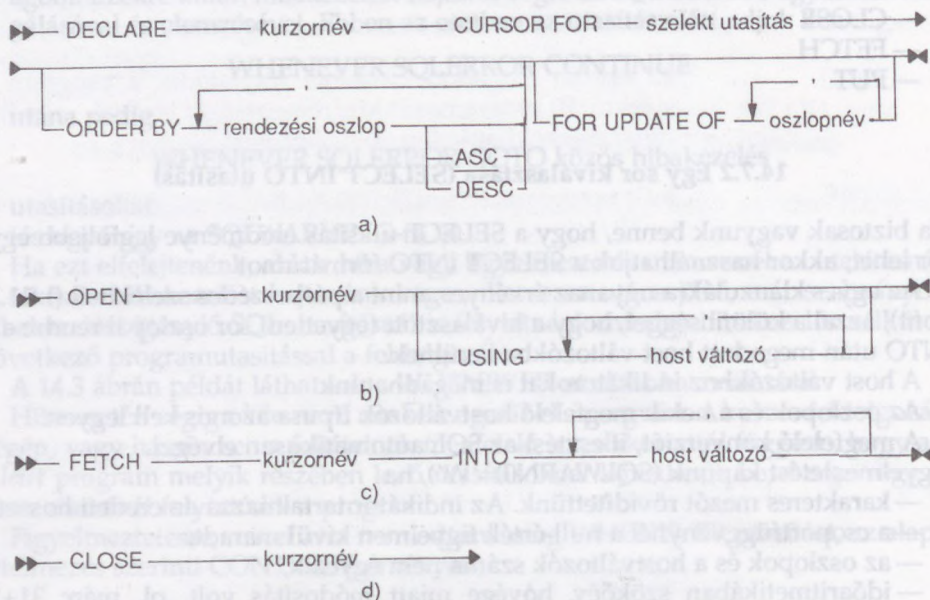
Ahhoz, hogy dolgozhassunk vele,

- a kurzort először definiálni kell (DEFINE CURSOR),
- utána meg kell nyitni (OPEN),
- rá kell állítani a feldolgozandó sorra (FETCH),
- az utolsó sor feldolgozásának befejeztével le kell zárni (CLOSE).

A kurzor, használata előtt a program eljárási részében deklarálni kell a 14.4a ábrán látható módon.

A deklarált kurzor neve egyedi a programban.

Egy programban akárhány kurzor lehet. A kurzor deklarálása nem jelenti az adatok tényleges kiválasztását és beolvasását. Ez az OPEN végrehajtásával, a kurzor megnyitásakor következik be. Ennek hatására a SELECT utasításában kijelölt adatok beolvasódnak egy ideiglenes táblázatba. Ezért a kurzor definíció SELECT utasítása nem tartalmazhat INTO klauzulát.



14.4 ábra Fontosabb kurzorműveletek szintaxis diagrammja

- a) kurzor deklarálása; b) kurzor megnyitása;
c) kurzor mozgatása; e) kurzor lezárása.

Az `OPEN` hatására létrejött ideiglenes táblázat sorain haladunk végig szekvenciálisan a `FETCH` utasítással.²

Ez a kurzort az ideiglenes táblázatban egy sorral előre mozgatja, és az így kijelölt sor kiválasztott oszlopainak értékét beviszi a `FETCH` utasításban megadott host-változóba.

A táblázat végét a `FETCH` utasításra kapott `SQLCODE=100` (WHENEVER-nél `NOT FOUND` feltétel bekövetkezte) jelzi. Ezután a kurzort le kell zárunk (`CLOSE`), hogy az ideiglenes táblázat esetleg elzárt sorait felszabadítsuk.

Ha kurzor alapján akarunk módosítani, azaz azt a sort akarjuk megváltoztatni, amelyiken a kurzor éppen áll, akkor meg kell adnunk a deklaráláskor az `UPDATE FOR` klauzulát is. Ekkor az aktuális sor ebben megadott oszlopoi módosíthatók a kurzort használó `UPDATE` utasítással.

A 14.5 ábrán egy programrészletet látunk, amelyik beolvassa a G&T táblázatból egy adott rendszámú gépkocsi összes sorát, és attól függően dolgozik tovább

²Ezt a táblázatot az SQL automatikusan kezeli. Sem a nevét, sem méretét, sem másféle paraméterét nem kell sem a programban, sem a futási környezetben deklarálnunk. A táblázat, akár mikor is dolgozzuk fel a sorait, mindig a kurzor megnyitáskor fennálló állapotot jelzi. Konkurens feldolgozásnál ezért ha nem akarjuk, hogy a feldolgozásunk közben bárki más megváltoztassa, vagy a még nem teljesen módosított táblázato(ka)t megtekintse, el kell zárunk azokat mások elől a `LOCK TABLE` utasítással, `Repeatable Read` (újraolvasáskor azonos lesz az eredmény) vagy `Commitment Control` (véglegesítés ellenőrzése) megadásával.


```

...
* HOST VÁLTOZÓK DEKLARÁLÁSA
EXEC SQL
    BEGIN DECLARE SECTION.
END-EXEC.
    77 P-RENSZÁM PIC X(6).
    77 P-TULAJDONOS PIC X(30).
    77 P-TULKEZDETE PIC X(10).
    77 P-TULVÉGE PIC X(10).
    77 IND-TULKEZDETE PIC S9(4) COMP-4.
    77 IND-TULVÉGE PIC S9(4) COMP-4.
EXEC SQL
    END DECLARE SECTION.
END-EXEC.

...
* KÜRZOR DEKLARÁLÁSA
EXEC SQL
    DECLARE KOCSI CURSOR FOR
        SELECT TULAJDONOS, TULAJDONJOG-KEZDETE, TULAJDONJOG-VÉGE
        FROM G&T
        WHERE RENDSZÁM = :P-RENSZÁM
END-EXEC.

...
* A KERESENDŐ RENDSZÁM BEOLVASÁSA P-RENSZÁMBA.

...
* SQL HIBAKEZELÉS FELDOLGOZÁSHOZ
EXEC SQL
    WHENEVER SQLERROR GOTO HIBA.
END-EXEC.

* KÜRZOR MEGNYITÁSA
EXEC SQL
    OPEN KOCSI
END-EXEC.

* G&T SORAINAK BEOLVASÁSA
PERFORM BEOLVAS UNTIL SQLCODE=100

* ÖSSZES TULAJDONOS MEGVAN; KÜRZOR LEZÁRÁSA
EXEC SQL
    CLOSE KOCSI
END-EXEC.

...
BEOLVAS.
* G&T SORAINAK FELDOLGOZÁSA
EXEC SQL
    FETCH KOCSI INTO :P-TULAJDONOS, :P-TULKEZDETE
    :IND-TULKEZDETE, :P-TULVÉGE :IND-TULVÉGE
END-EXEC.

* ELÁGAZÁS KORÁBBI VAGY JELENLEGI TULAJDONOS FELDOLGOZÁSÁRA
IF IND-TULVÉGE = -1 THEN PERFORM JELENLEGI
ELSE PERFORM KORÁBBI.

...
JELENLEGI.

...
KORÁBBI.

...
HIBA.
* SQL HIBÁK KEZELÉSE
...

```

14.5 ábra Feldolgozás kurzor segítségével

A G&T táblázatból feldolgozzuk egy adott rendszámú gépkocsi összes sorát. A feldolgozás más a jelenlegi és a korábbi tulajdonosoknál. A kívánt ágra a TULAJDONJOG-VÉGE null-értéket jelző indikátor alapján ágazik el a program.

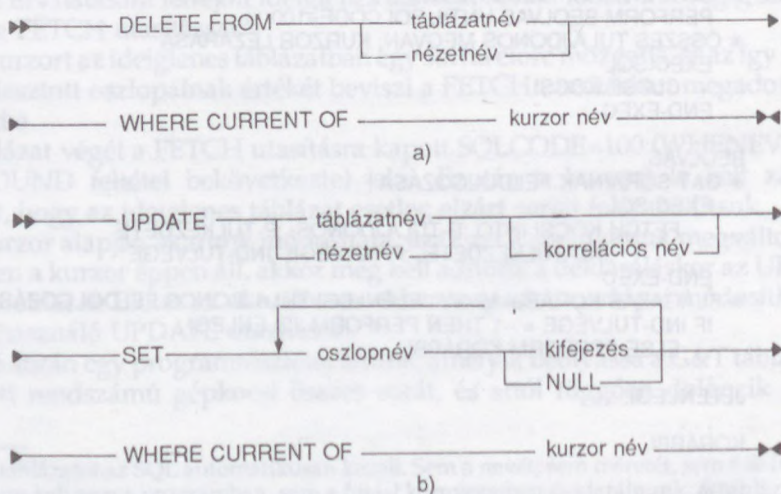
velük, hogy azok a korábbi vagy a jelenlegi tulajdonosra jellemzőek-e. A különbséget a TULAJDONJOG-VÉGE oszlop beolvasásához rendelt indikátor alapján állapítja meg. Ez negatív a jelenlegi tulajdonosnál, mert annál az oszlop értéke null-érték. A TULAJDONOS oszlophoz nem adtunk meg indikátort, mert az eleve nem vehet fel null-értéket (l. a CREATE TABLE G&T utasítást a 10.2 ábrán).

14.7.4 Változtatás a kurzor helyzete alapján

Ha egy kiválasztási feltételnek megfelelő összes sort törölni vagy módosítani akarunk, akkor programból is használhatjuk a párbeszédés DELETE-tel, illetve UPDATE-tal teljesen azonos formájú és hatású programbeli utasítást.

Gyakran előfordul azonban, hogy a kiválasztott sorokkal előbb különféle műveleteket is el kell végeznünk, majd ezek eredményétől függően egyeseket törölünk, másokat különféle módon változtatunk (vagy változatlanul hagyjuk). Erre a célra szolgálnak a csak programból használható pozicionált kurzorral dolgozó DELETE és UPDATE utasítások (14.6 ábra).

Használatukhoz a kurzort először definiálni kell, majd az így kiválasztott sorokat a kurzor megnyitásával beolvassuk egy ideiglenes táblázatba. Ennek sorain haladunk végig a FETCH utasítással ugyanúgy, ahogy azt a 14.7.3 pontban ismertettük. Ha a kurzorral így kijelölt sort törölni, vagy módosítani akarjuk, akkor kiadjuk rá a megfelelő utasítást a WHERE CURRENT OF klauzulával kiegészítve. Ellentétben a normális DELETE és UPDATE-tal, ahol a változtatott sorok száma akármennyi lehet, a pozicionált utasítás mindig csak egy sort töröl vagy módosít.



14.6 ábra Pozicionált kurzort használó utasítások szintaxis diagrammja

a) a kurzorral kijelölt sor törlése

b) a kurzorral kijelölt sor módosítása

Törlés után a kurzort egy újabb FETCH utasítással kell ráállítani a következő sorra. UPDATE végrehajtása után a kurzor továbbra is a módosított sorra mutat.

Az integritási feltételek, záruk kezelése hasonló elvek alapján működik a kurzorral pozicionált változtató utasításoknál is, mint a WHERE feltétel alapján változtatóknál. Ahogyan ott például a szubszelekt alapján való kiválasztásnál bizonyos korlátozások érvényesek (l. 11.3.2 pont), ugyanúgy a kurzor pozíciója szerinti változtatásnál sem adhatjuk meg teljesen szabadon a kurzor definíciójában szereplő SELECT utasítást. Ennek részletes taglalása azonban már meghaladja e könyv kereteit.

Ügyeljünk arra, hogy a legtöbb rendszerben a COMMIT és a ROLLBACK lezárja a kurzort. Így, ha ezt nem a kurzor feldolgozásának a végén, hanem közben adjuk ki, akkor a feldolgozás újra az első sortól fog elkezdődni. Egyes rendszerek ennek elkerülésére ismerik ezeknek az utasításoknak a HOLD kiegészítéssel megadott változatát is. Ekkor a kurzor marad a COMMIT HOLD, vagy a ROLLBACK HOLD kiadásakor érvényes helyén és a feldolgozás ettől, vagy a FETCH utasítás kiadásával az ezután következő sorral folytatódhat.

14.8 Dinamikus SQL utasítások programban

14.8.1 Dinamikus utasítások jellemzői

Előfordulhat, hogy az adatbázissal kapcsolatos igényeinket a program írásakor még nem tudjuk olyan pontossággal megfogalmazni, hogy azok statikus, már a fordítás folyamatakor rögzíthető SQL utasításként beépüljenek a futtatható programba. Ilyenkor használunk dinamikus SQL utasításokat, melyeket a program futásakor fogalmazhatunk meg.

Az adatbázis-kezelő rendszer a program futása közben az SQL nyelven megfogalmazott utasítást

- szintaktikusan ellenőrzi és értelmezi,
- ellenőrzi, hogy a benne szereplő műveletekre a program futtatója jogosult-e,
- ezek után hiba esetén hibaüzenetet küld és visszaadja a vezérlést a programnak.

Különb

- meghatározza az optimális elérési utat,
- létrehozza a végrehajtáshoz szükséges kódot,
- az operációs rendszerrel végrehajtatja a műveleteket,
- a művelet eredményét, beleértve az SQL visszajelzési kódokat is átadja a programnak.

Ezek a műveletek általában nagyobb rendszeradminisztrációt igényelnek, mint a fordítás során már lefordított és a futás idejére végleges formát öltött statikus SQL utasítások végrehajtása. Ebből adódik, hogy a dinamikus SQL utasítások végrehajtása általában lassabb, mint a hasonló feladatot ellátó statikus utasításoké. Ezért dinamikus SQL utasításokat lehetőleg csak akkor használjunk a programban, ha a feladatot statikus utasításokkal nem tudjuk megoldani.

Dinamikus utasítások tipikus alkalmazási területei a következők:

- Ugyanazt a programot használjuk, de különböző táblázatokkal. (Táblázatnév nem lehet host változó!).
- Programfutástól függ, milyen utasítást kell használnunk. Sok a választási lehetőség, nem tudjuk előre beprogramozni.
- A WHERE feltételek operátorát (típusát), és/vagy számát nem ismerjük előre.
- Integritási hiba miatt esetleg nem akarjuk leállítani a futást. Lehetőséget adunk ALTER...KEY vagy DROP...KEY vagy DROP INDEX dinamikus beadásával a folytatásra, vagy más utasítások kiadására.

Nem mindegyik SQL utasítást lehet dinamikusan előállítani. Ezeket csak a pre-kompiler tudja feldolgozni. Az ismertebb SQL utasítások közül nem állítható elő dinamikusan a

BEGIN DECLARE SECTION	EXECUTE IMMEDIATE
CLOSE	FETCH
CONNECT	INCLUDE
DECLARE CURSOR	OPEN
DESCRIBE	PREPARE
END DECLARE SECTION	PUT
EXECUTE	WHENEVER

utasítás.

14.8.2 Dinamikus utasítások azonnali végrehajtása (EXECUTE IMMEDIATE utasítás)

Azokat a dinamikus SQL programutasításokat, amelyek nem tartalmaznak paramétert, azaz az utasítást betű szerint úgy kell azonnal értelmezni, ahogy azt a program megkapja, az EXECUTE IMMEDIATE utasítással hajtathatjuk végre. Ennek formája:

►—— EXECUTE IMMEDIATE —— host-változó ——►

ahol a host változó aktuális tartalma lesz a végrehajtandó SQL utasítás.

Az EXECUTE IMMEDIATE utasítás

- megadott karakterstringből előállítja a végrehajtandó SQL utasítást,
- végrehajtja az utasítást és
- törli a végrehajtott formát.

A host változóhoz nem lehet indikátort megadni. COBOL-ban, C-ben, Assembler-ben változó a hosszúsága, FORTRAN-ban rögzített. PL/I-ben lehet fix- és változó hosszúságúként deklarálni az (SQL) DECLARE SECTION-ban.

A hostváltozóban levő karaktersorozat

- csak dinamikusan is végrehajtható SQL utasításra hivatkozhat,
- nem tartalmazhat utalást host változóra,
- nem kezdődhet EXEC SQL-lel,
- nem végződhet END-EXEC-kel,

- nem tartalmazhat paramétert,
- nem tartalmazhat megjegyzést (comment).

Ha az előállított utasítás hibás, nem kerül végrehajtásra.

Erről az SQL kommunikációs területen (SQLCA) információt kapunk és SQLCODE<0 is jelzi.

Mivel az EXECUTE IMMEDIATE utasítás minden egyes végrehajtásakor újra generálja a host változóban levő SQL utasítás végrehajtásához szükséges kódot, ha ezt többször kell ugyanazzal a host értékkel végrehajtani, akkor célszerűbb lehet ehelyett az SQL utasítást a következő pontban ismertetendő módon egyszer, a program futásának idejére megmaradó formában elkészíteni és ezt a kódsorozatot változtatás nélkül mindig lefuttatni.

Az EXECUTE IMMEDIATE utasítás lehetővé teszi, hogy az alkalmazási program valamelyik ágában nem szelekt típusú interaktív SQL utasításokat adhassunk be akár a program tesztelésére, akár hibás adatok miatt fellépő programhiba elhárítására.

Egy erre vonatkozó példát a 14.7 ábrán láthatunk.

```
EXEC SQL
  BEGIN DECLARE SECTION
END-EXEC
77  UTASÍTÁS      PIC X(80).
```

```
EXEC SQL
  END DECLARE SECTION.
END-EXEC.
```

- * NEM SZELEKT TÍPUSÚ INTERAKTÍV UTASÍTÁSOK VÉGREHAJTÁSA
- * AZ SQL UTASÍTÁS BEOLVASÁSA
 - ACCEPT UTASÍTÁS.
 - PERFORM INT-SQL UNTIL UTASÍTÁS = 'VÉGE'.

```
INT-SQL.
EXEC-SQL
  WHENEVER SQLERROR CONTINUE
END-EXEC.
EXEC SQL
  EXECUTE IMMEDIATE UTASÍTÁS
END-EXEC.
IF SQLCODE NOT = 0 PERFORM EXEC-IMMED-HIBA.
EXEC SQL
  WHENEVER SQLERROR ...
END-EXEC.
```

- * VISSZÁLLÍTJUK A PROGRAM RENDES HIBAKEZELÉSÉT
 - ACCEPT UTASÍTÁS.
 - INTSQL-END.

```
EXEC-IMMED-HIBA.
* HIBA KEZELÉS SQLCODE ALAPJÁN.
  DISPLAY "HIBÁS UTASÍTÁS", SQLCODE.
  DISPLAY UTASÍTÁS.
```

14.7 ábra Interaktív SQL lehetőségek „beépítése” alkalmazási programba

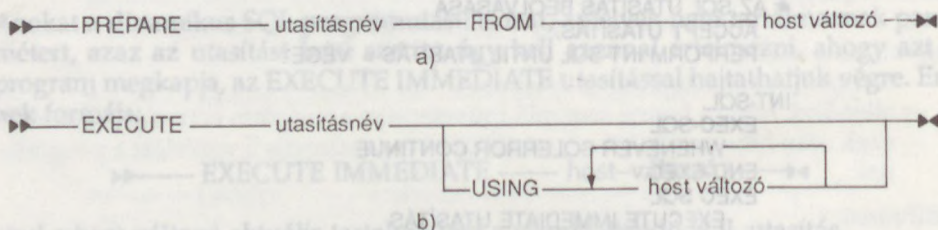
Az UTASÍTÁS nevű host változóba beolvasunk egy nem szelekt típusú SQL utasítást (pl. DELETE FROM TULAJDONOS WHERE NÉV='XYZ'). Ezt végrehajtjuk. Az SQLCODE-ot közvetlenül értékeljük ki. Hibás utasítás megadása esetén meghívjuk az EXEC-IMMED-HIBA nevű eljárást. Ebben a hibakódtól függő, ad hoc megadott interaktív utasításokkal megkeressük a hiba okát és elhárítjuk. Addig dolgozunk párbeszédesen, amíg UTASÍTÁS-ba azt olvassuk be, hogy „VÉGE”.

14.8.3 Dinamikus utasítások előkészítése és végrehajtása (PREPARE és EXECUTE utasítás)

Azokat a dinamikus SQL utasításokat, amelyek paramétert tartalmaznak, azaz az értelmezésnél az utasítás szó szerinti értelmezése helyett egyes adatok értéke futáskor programbeli változókból jön vagy oda kerül, előbb elő kell készíteni a futásra a PREPARE utasítással.³ Az így előkészített utasítást hajtjuk végre az EXECUTE utasítás segítségével. Mivel a PREPARE által megadott kód a futás folyamán megmarad, a paraméterek változtatásával ugyanazt az utasítást más és más értékekkel többször is lefuttathatjuk az SQL kód újra generálása nélkül.

A PREPARE és az EXECUTE utasítás leggyakrabban használt formájának szintaxis-diagrammját a 14.8 ábrán láthatjuk. Az utasításnév változót az SQL maga kezeli, a programban nem szabad deklarálnunk. A PREPARE utasítás végrehajtása után mindig vizsgáljuk le SQLCODE értékét, hogy valóban végrehajtható SQL utasítás jött-e létre!

A paraméter helyét az utasításban kérdőjel (?) jelzi. Végrehajtáskor az utasítás leírásában az n -dik paraméter a feldolgozáskor megadott (USING klauzula) n -dik paraméteréhez kapcsolódik.



14.8 ábra Paramétereket tartalmazó dinamikus SQL utasítások beépítése programba

a) A dinamikus utasítás elkészítésének szintaxis diagrammja

b) Az előkészített utasítás végrehajtásának szintaxis diagrammja

(A szintaxis diagramm nem teljes. Csak a leggyakrabban használt formát tartalmazza.)

A nem szelekt típusú, paramétert tartalmazó dinamikus SQL utasítások használati módja tehát programban a következő:

— Definiáljuk az utasítás szövegét tartalmazó hostváltozót.

³ A PREPARE-rel előkészíthető SQL utasítások lehetőségei korlátozottak. Egyes nyelvek lehetővé teszik ennél szélesebb körű SQL szolgáltatások dinamikus előkészítését a DESCRIBE utasítással. Ekkor például a paraméterek száma is változhat dinamikusan.

- RREPREARE-rel létrehozzuk belőle a dinamikus utasítást (ennek neve tetszőleges, nem szabad deklarálnunk).
- EXECUTE-tal végrehajtjuk az így elnevezett utasítást.
- A paramétereket az utasításban való előfordulásuk sorrendjében rendeljük hozzá a USING klauzulában megadott host változókhoz.
- Mind a PREPARE, mind az EXECUTE utasítás végrehajtása után ellenőrizzük SQLCODE értékét.

Egy ilyen feldolgozási ciklust, ahol mind a feldolgozandó táblázat tulajdonosát, mind a törlendő adat értékét paraméterként adjuk meg, a 14.9 ábrán láthatunk.

Ha a táblázat (tulajdonosa) nem változna, akkor nem volna szükség dinamikus utasításra. A törlendő sor jellemzését statikus utasításban hostváltozóként is megadhatnánk.

```

01 TÖRLÉS
03 FILLER          PIC X(12) VALUE "DELETE FROM".
03 TÁBLÁZAT        PIC X(49).
03 FILLER          PIC X(19) VALUE "WHERE RENDSZÁM=?"

```

a változat: . . .

törlendő táblázat (pl. TOLNA.GÉPJÁRMŰ) nevének beolvasása a TÁBLÁZAT nevű változóba
 PREPARE UTASÍTÁS FROM :TÖRLÉS
 ciklusban:
 rendszám beolvasása RENDSZÁM-ba
 EXECUTE UTASÍTÁS USING :RENSZÁM.

vagy

b változat:

MOVE "?" TO TÁBLÁZAT.
 PREPARE UTASÍTÁS FROM TÖRLÉS
 ciklusban:
 törlendő táblázat és rendszám beolvasása

EXECUTE UTASÍTÁS USING :TÁBLÁZAT, :RENSZÁM

14.9 ábra Dinamikus SQL utasítás előkészítése és végrehajtása.

Az utasításban változik a táblázat tulajdonosa (a nyilvántartó megye neve) és a törlendő gépjármű rendszáma.

Az EXEC SQL, END-EXEC határoló és az SQLCODE ellenőrző utasításokat az áttekinthetőség kedvéért nem tüntettük fel.

A fenti módszert például akkor alkalmazzák, ha több felhasználó dolgozik ugyanazzal a programmal, de mindegyikük csak a *saját táblázatainak* adataival végez különböző műveleteket.

Az ábra a) változatában végig ugyanarra a megyére (a megye a táblázat tulajdonosa, neve minősíti a GÉPJÁRMŰ táblázatot) vonatkozik a feldolgozás, azért elég azt egyszer beolvasnunk. A b) változatban bármelyik megye állományából törölhetünk, ezért az utasításnak két paramétere van.

A szelekt típusú dinamikus utasításokban nem lehet INTO klauzula. A kiválasztott sor(ok) kezelése mindig kurzorral történik.

A felépítés és feldolgozás lépései a következők:

- az input SQL utasítást bevisszük egy hostváltozóba,
- előkészítjük futáshoz (PREPARE),
- kurzort deklarálunk hozzá (DECLARE),
- megnyitjuk a kurzort (OPEN). Ha az utasítás paramétereit tartalmaz, ezek aktuális értékét itt kell hozzárendelni a USING klauzulában megadott host-változókon keresztül,
- ciklusban egyenként behozzuk a sorokat hostváltozó(k)ba (FETCH);
- feldolgozzuk egyenként a sorokat,
- az utolsó sor feldolgozása után lezárjuk a kurzort (CLOSE),
- minden SQL utasítás után kezeljük az SQL return kódokat.

Az utasításban (bizonyos korlátozásokkal) használhatunk paramétereket is.

A 14.10 ábrán egy dinamikus szelekt utasítás előkészítését és feldolgozását láthatjuk. Bár az utasítás nem tartalmaz közvetlenül paramétereket, a szelekt eredménye miatt kapcsolatot kell létesíteni host változókkal, ezért nem lehet előkészítés nélkül, közvetlenül végrehajtani.

```
01 SZELEKT ... VALUE
   "SELECT RENDSZÁM, TÍPUS, GYÁRTMÁNY, TULAJDONOS FROM G&T GT,
- GÉPJÁRMŰ G WHERE G:RENDSZÁM = GT.RENDSZÁM AND
- TULAJDONJOG-VÉGE BETWEEN '1992-12-01' AND '1992-12-31'
```

```
WHENEVER ...
PREPARE KIVÁLASZTÁS FROM :SZELEKT.
DECLARE OLVAS CURSOR FOR KIVÁLASZTÁS.
OPEN OLVAS.
```

```
PERFORM FELDOLGOZÁS UNTIL ...
```

```
CLOSE OLVAS.
```

```
FELDOLGOZÁS SECTION.
```

```
FETCH OLVAS INTO ... (host változók indikátorral)
feldolgozás...
```

14.10 ábra Dinamikus SELECT utasítás előkészítése és végrehajtása

Az utasítás az 1992. dec. 1-én és dec. 31 között eladott gépkocsik adatait hozza be feldolgozásra eladási időpont és ezen belül rendszám szerint rendezve.

Az EXEC SQL, END-EXEC határoló és az SQLCODE ellenőrző utasításokat az áttekinthetőség kedvéért nem tüntettük fel.

14.9 Jogosultság ellenőrzés, elérési út

A programba beépített SQL utasításokhoz is a rendszer generálja a várhatóan optimális elérési utat a katalógusban levő információk alapján. Az így létrejött kód része a programnak.

Statikus SQL utasításoknál az elérési utat fordításkor határozza meg a rendszer, a fordításkor fennálló lehetőségek alapján végezve el az optimalizációt. Ha ezután

olyan változások történnek az adatbázisban, melyek szerint hatékonyabb elérési utat is lehetne találni (pl. új indexeket hoztunk létre, megnövekedett táblázatméretek miatt várhatóan más feltételek szelektívebb kiválasztást biztosítanak), ezeket csak újra fordítás után tudja a program kihasználni. Ha viszont valamelyik régi, a programba beépített elérési mód megszűnik, de az adatok más módon még elérhetők (pl. egy indexet töröltek), akkor a program futtatásakor a rendszer automatikusan új, az akkori feltételek szerint optimális elérési utat generál.

(Természetesen, ha olyan változások állnak be az adatbázisban, amelyek az adatok elérését lehetetlenné teszik (pl. megszüntetünk egy hivatkozott táblázatot), akkor az erre vonatkozó utasítások nem hajtódnak végre. Ezt negatív SQLCODE jelzi. A program hibakezelési eljárásától függ, hogy ilyen esetben miként folytatódik tovább.)

Statikus utasításoknál általában a program írójának kell a művelet elvégzésére jogosultsága legyen, és ő adhatja ezt tovább másoknak a program futtatását engedélyező GRANT RUN utasítással. Vannak rendszerek, amelyek lehetővé teszik a statikus műveletek elvégzését (pl. SQL/400), ha akár a program készítőjének, akár a futtatójának van arra jogosultsága.

Amennyiben a fordításkor a program készítője még nem rendelkezett a megfelelő jogosultsággal, vagy a fordítás óta a program által használt táblázatok/nézetek jogosultságában változás történt, akkor az SQL megjelöli az elérési utat tartalmazó kódot. A program legközelebbi futásakor ellenőrzi a jogosultságokat és ennek eredményeként vagy ismét visszaállítja az eredeti állapotot, vagy továbbra is felfüggeszti és a jogosultsági feltételek teljesítésétől teszi függővé az elérési út felhasználását.

Dinamikus utasításnál hasonlóan, mint a párbeszédés SQL műveleteknél az elérési út generálása és a jogosultság ellenőrzése is mindig a program futásakor, közvetlenül az utasítás végrehajtása előtt történik. A program futtatójának kell jogosultsága legyen a dinamikusan megadott SQL művelet elvégzésére. Amennyiben ezzel nem rendelkezik, akkor a művelet nem hajtódik végre. Ezt negatív SQL kód jelzi, de ettől a program még tovább futhat. Természetesen a felhasználó felelőssége, hogy ilyenkor hogyan dolgozik tovább, hogyan oldja meg az adatvédelmi okokból végre nem hajtott műveletek pótlását, helyettesítését vagy megkezdését.

A fentieket összefoglalva megállapíthatjuk, hogy az adatvédelmi és az integritást biztosító mechanizmusok az adatbázishoz programon keresztül való hozzáférésnél is ugyan olyan teljes mértékű biztonságot nyújtanak, mint párbeszédés működésnél.

Függelék

A) Ajánlott irodalom

Könyvünkben—miután azt általános, összefoglaló ismertetőnek szánjuk adatbázisokkal már gyakorlatban foglalkozó vagy foglalkozni kívánó szakemberek számára—nem adtunk meg konkrét irodalmi utalásokat az egyes témákra, részterületekre. Ugyanakkor, miután a szerző már írt néhány könyvet, és ennél sokkal többet el is olvasott, saját tapasztalatából tudja, egy könyv sem tartalmaz mindent, amit szerzője meg akart írni és amit az olvasó tudni szeretne az adott témáról. Ezért az alábbiakban felsorolunk néhány olyan könyvet, melyek hasznos segítséget nyújthatnak az adatbázisokkal való munka még részletesebb, vagy más szempontok alapján történő megismerésére.

Könyvünk nagy része a relációs modellel foglalkozik. Ebben, mint azt többször is hangsúlyoztuk, a sorok sorrendjének semmiféle kitüntetett szerepe nincsen. Ezt szem előtt tartva, a most következő felsorolási sorrend sem tükröz valamiféle értékelést, csupán a szerzők neve szerint „ORDERED BY” eredmény reláció.

- BERNSTEIN, P.A. — HADZILACOS, V. — GOODMAN, D.:
Concurrency Control and Recovery in Database Systems, Addison-Wesley Publ.Comp., Reading, Mass, 1987
- BURCH, J.G. — GRUDNITSKI, G.:
Information Systems: Theory and Practice, J.Wiley, New York, N.Y. 1986
- CERI, S. — Pelagatti, G.:
Distributed Databases: Principles and Systems, McGraw-Hill, New York, N.Y. 1984
- CODD, E.F.:
The Relational Modell for Database Management, Version 2, Addison-Wesley Publ.Comp., Reading, Mass, 1990
- CURTIS, G.:
Business Information Systems, Addison-Wesley Publ.Comp., Reading, Mass, 1989
- DATE, C.J.:
Relational Database: Selected Writings. Addison-Wesley Publ.Comp., Reading, Mass, 1986
- DATE, C.J.:
Relational Database Writings 1985–89, Addison-Wesley Publ.Comp., Reading, Mass, 1990
- DATE, C.J.:
An Introduction to Database Systems, Fifth Edition, Addison-Wesley Publ.Comp., Reading, Mass, Inc. 1990
- ELMASN, R. — NAVATHE, S.B.:
Fundamentals of Database Systems, Benjamin/Cummings, Redwood City, Calif, 1989
- GAYDASCH, A.:
Effective Database Management, Prentice-Hall, Englewood Cliffs, NJ, 1988
- HALASSY B.:
Adatmodellezés a rendszertervezésben, SZÁMALK, Budapest, 1983

B) Tárgymutató

adat 19
 adatbázis architektúra 31
 adatbázis-felügyelő 22, 25, 40, 55, 66, 72, 80, 115,
 131, 134, 141, 150, 176, 188, 197, 215, 224, 246,
 252
 adatbiztonság 39
 adatelem 69, 76
 adatfragmentáció 51
 adatfüggetlenség 30, 66
 adatkezelő nyelv 35, 116, 128, 135, 214
 adatkezelő utasítások 213, 215
 adatmodell 31, 43, 59, 109, 143
 adatszervezési mód 73
 adatszótár 38, 193, 286
 adattípus 216, 226
 adatvédelem 130, 193, 195, 221
 alkalmazási programozó 26
 ALL feltétel 244
 alséma 33, 35, 59, 128, 134
 ALTER TABLE 226
 alternatív kulcs 150
 ANY feltétel 244
 áteresztőképeség 18
 átnevezés 158
 attribútum 144

B⁺ fa 93, 225
 beépíthető rendszer 23, 36, 267
 belső szint 31, 33, 59, 65, 128
 bevétel 154, 182, 248
 bináris fa 114
 blokk 69, 73
 bővítés 167

CANCEL 265
 cylinder 67
 cím 70, 76
 címzés 73
 COMMENT ON 265
 COMMIT 255, 281
 commitment control 255
 CREATE TABLE 216
 CREATE INDEX 224
 CREATE VIEW 219
 csoportfüggvény 168, 172, 238
 csoportosítás 235, 238

DBTG javaslat 127,
 DLL 34
 dekompozíció 182, 187
 DELETE 246, 280
 dinamikus SQL utasítások 269, 281
 direkt keresés 90, 99
 DML 35
 dokumentum formátum 29, 201

DROP 225
 duplikált index 83, 85, 224

egyedi index 83, 224, 246, 248
 egyesítés 165, 170, 173, 183, 186, 234, 238
 elemi index 84
 elérési mód 33, 73
 elérési út 73, 172, 224, 230, 260, 286
 első normál forma 179
 elsődleges kulcs 69, 147, 150, 182, 186, 218, 226
 elsődleges adatterület 90
 EXECUTE IMMEDIATE 282
 EXECUTE 284
 EXISTS feltétel 242
 EXPLAIN 261

faszerkezet 109
 fejkiválasztás 68
 fejmozgatási idő 68
 felhasználó 25, 224
 FETCH 277, 280
 FIFO 79
 fizikai szint 31, 43, 65, 128
 fizikai sorrend 74, 81
 foksám 144
 FOREIGN KEY klauzula 218
 forgási idő 68,
 FORMAT 259
 fő index 91
 fragmentáció 51
 FROM klauzula 233
 funkcionális függés 177, 182, 191
 funkcionális diagramm 177
 függő reláció 152, 250
 függőség 177
 függőségi kapcsolat 152

GRANT 251, 287
 GROUP BY klauzula 235

hálós modell 60, 123
 hálózat 48
 harmadik normál forma 184
 hashing 99, 140
 hatékonyság 139, 172, 226, 230, 236, 245, 260
 HELP 265
 hierarchikus adatbázis 115
 hierarchikus modell 59, 72, 109
 hivatkozási feltétel 152
 hivatkozó reláció 151, 250
 hivatkozott reláció 151, 250
 host language system 23, 36, 269
 host változó 269, 271, 277, 282
 hozzáférési jog 195, 215, 216, 219, 251, 286
 hozzáférési idő 68

- idegen kulcs 150, 152, 178, 183, 186, 218, 226, 246, 248, 250
- IN feltétel 243, 264
- INCLUDE 270, 273
- index 81, 157, 195, 202, 224, 226, 236, 252, 263
- index-szekvenciális szervezés 89
- indextáblázat 82, 89, 202
- indexterület 90
- indikátor 272, 277
- információ 19
- információs infrastruktúra 17
- információs rendszer összetevői 19
- informatikai központ 27
- INSERT 248
- integrált adatbázis 39, 44
- integritás 149, 157, 183, 249, 281, 282
- INTO klauzula 271, 277
- író-olvasó fej 67
- jelentés készítő 37
- jogosultság 196, 215, 216, 219, 251, 286
- join 164, 170, 173, 183, 186, 234
- kapcsolattípusok 62, 114, 123, 125, 131
- kapcsoló rekord 125
- kardinalitás 146
- katalógus 38, 193, 246, 286
- képernyő tervező 37
- keresztzorzat 161
- kijelölés 168
- kiválasztási feltétel 235, 240
- kiválasztásos rendezés 106
- kizárólagos zár 253
- kommunikációs terület 273, 283
- koncepcionális szint 31, 33, 44, 59, 73, 128
- konzisztencia 55, 231
- korlátozás 159, 169, 222, 237,
- korrelált szubszelekt 245
- közbenső táblázat 148
- közvetlen elérés 89, 99
- kulcs 69, 74, 92, 97, 147
- kurzor 270, 276, 277
- különbség 164
- különleges események kezelése 274
- külső szint 31, 33, 59, 219
- kvantifikált feltétel 244
- lap 69, 73
- lapindex 90
- LIFO 79
- LIKE feltétel 242, 264
- listaszerkezet 76, 81, 85, 114
- LOCK TABLE 254
- mágneslemez 67
- második normál forma 181
- másodlagos kulcs 69
- master index 91
- metszet 164, 169
- mező 69
- módosítás 155, 181, 250, 280
- működési elv 21
- mutató 76, 85, 138
- negyedik normál forma 191
- nézet, csak olvasható 223, 246
- nézet 148, 191, 219, 226, 246, 250
- normál forma 179
- normalizáció 176, 178
- null-érték 147, 151, 155, 166, 170, 218, 223, 225, 226, 238, 239, 242, 245, 249, 273
- operációs rendszer 21
- operátori parancsok 257
- ORDER BY klauzula 235, 264
- oszlop 144, 194, 216, 226, 233
- oszlopfüggvény 238
- osztott feldolgozás 49
- osztott adatbázis 40, 46, 51, 197
- osztott zár 253
- önálló rendszer 23, 36
- összekapcsolás 164, 169, 173, 234, 238
- összetett kulcs 151
- összetett index 84
- patthelyzet 55, 254
- pointer 76, 85, 138, 142
- pozicionálási idő 68, 75
- prekompilier 37, 269, 270
- PREPARE 284
- preprocesszor 36, 269, 270
- PRIMARY KEY klauzula 218
- processzor hierarchia 47
- projekció 161, 169, 173, 182, 185, 234,
- rekord 69, 73
- reláció 144
- relációs modell 60, 143
- relációs műveletek 157
- rendezés 102
- report generátor 37
- REVOKE 253
- ROLLBACK 256, 281
- sáv 67
- SELECT 231, 238
- SELECT INTO 277
- self contained system 23
- séma 33, 34, 59, 128
- set 128, 131, 140
- SET (utasítás) 260
- skalár függvény 238
- SOME feltétel 244
- sor 144
- soros elérés 74, 81

- SQL utasítástípusok 213, 268
- SQL függvény 238
- SQL 209
- SQLCA 273, 283
- SQLCODE 272, 273, 278, 283, 284
- statikus SQL utasítások 269, 271, 276
- sűrűsödés 72, 75, 224, 262

- szegmens 115
- szekvenciális szervezés/elérés 75, 81
- szelekt lista 234, 271
- személyi számítógépek 45
- szervezési mód 73
- szétbontás 182, 185
- szinguláris set 132
- szinonim 99, 205
- szintaxis diagram 210
- szolgáltató program 38, 73
- szöveges információvisszakereső rendszer 29, 201, 242
- szubszelekt 232, 244, 247, 249

- táblázat 143, 147, 194, 216, 226, 234
- tag (member) 131
- találati arány 206
- tárolási struktúra 33, 43, 66
- tárolt SQL utasítás 258
- tény adatbázis 29

- tezausz 205
- titkosítás 101, 130, 133
- tömörítés 100
- törlés 154, 181, 218, 246, 280
- transzparencia 51
- tranzakció 18, 25, 255, 256
- tulajdonos (owner) 131
- túlsordulás 93
- túlsordulási terület 90

- újraszervezés 70, 72, 76, 93
- unió 162
- UNION klauzula 236, 262
- UPDATE 249, 280

- változó hosszúságú mező 72, 101
- változtatási anomália 181, 185, 188
- vetület 160, 169, 221
- vezérlő SQL utasítások 214, 251
- vezetői információs rendszer 18
- virtuális reláció 148, 219
- visszaállítás 256

- WHENEVER 274, 278
- WHERE CURRENT OF klauzula 280
- WHERE klauzula 235, 240

- zár 253, 258



A kiadásért felelős az Akadémiai Kiadó és Nyomda Vállalat igazgatója
A nyomdai munkálatokat az Akadémiai Kiadó és Nyomda Vállalat végezte
Felelős vezető: Zöld Ferenc

Budapest, 1993

Nyomdai táskaszám: 12956

Felelős szerkesztő: Egyed Ildikó
Műszaki szerkesztő: Marton Andor
A kötésterv Marton Andor munkája
Kiadványszám: 4/23
Megjelent 26.45 (A/5) ív terjedelemben

A számítógépeknek napjainkban a fő feladata az információ-szolgáltatás. Információ visszakereső rendszerek szervezése, felépítése, működtetése, a számítógép egyik legfontosabb alkalmazási területe. Ennek megoldásában nyújtanak segítséget az adatbázis-kezelő rendszerek. A mini- és ennél nagyobb számítógépekkel rendelkező számítóközpontoknak csaknem mindegyikében dolgoznak egy vagy több adatbázissal és üzemeltetnek valamilyen adatbázis-kezelő rendszert. A professzionális célokra szolgáló mikrogepek jelentős részén is sok feladatot oldanak meg adatbázis-kezelő szoftver segítségével.

Éppen az adatbázis-kezelő rendszerek elterjedése következtében sok tévhit is kialakult ezen a területen, kezdve attól, hogy a bérosztályon az egyszerű bérfizetési rendszert a „mi adatbázisunknak” tekintik, egészen addig, hogy a legbonyolultabb vezetői döntési problémákra is azonnal megoldást adó csodamódszert várnak egy ilyen szoftver megvételétől.

Az adatoknak és az adatbázis-kezelő rendszerektől való elvárásoknak a rendszerezéséhez kíván segítséget nyújtani ez a könyv mind az adatbázisok szervezőinek, mind felhasználóinak.



508823

QUITTNER PÁL •

ADATBÁZIS-KEZELÉS A GYAKORLATBAN