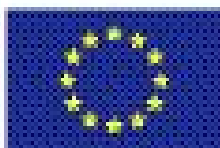


Magyarország célba ér



A projekt az Európai Unió
társfinanszírozásával, az Európa terv
keretében valósul meg.

Quittner Pál

Baksa-Haskó Gabriella

ADATBÁZISOK, ADATBÁZIS-KEZELŐ RENDSZEREK



HEFOP 3.3.1–P.-2004-06-0071/1.0

**Ez a kiadvány a
„Gyakorlatorientált képzési rendszerek kialakítása
és minőségi fejlesztése az agrár-felsőoktatásban”
című program keretében készült**

Quittner Pál

Baksa-Haskó Gabriella

ADATBÁZISOK, ADATBÁZIS-KEZELŐ RENDSZEREK

Szerző:

Quittner Pál
Budapesti Corvinus Egyetem

Baksa-Haskó Gabriella
Budapesti Corvinus Egyetem

Lektor:

Kormos János
Debreceni Egyetem

Magó Zsolt
Debreceni Egyetem

© DE AMTC AVK 2007

ISBN 978-963-9732-66-7

**E tankönyv teljes mértékben megegyezik a Debreceni Egyetem honlapján,
a <http://odin.agr.unideb.hu/hefop/> elérési úton megtalálható, azonos című tankönyvvel.**

Első kiadás

A kiadvány szerzői jogvédelem alatt áll. A kiadványt, illetve annak részeit másolni, reprodukálni, adatrögzítő rendszerben tárolni bármilyen formában és bármilyen eszközzel – elektronikus úton vagy más módon – a kiadó és a szerzők előzetes írásbeli engedélye nélkül tilos.

Kiadó:

Debreceni Egyetem Agrár- és Műszaki Tudományok Centruma
Agrárgazdasági és Vidékfejlesztési Kar

Debrecen, 2007.

TARTALOMJEGYZÉK

TARTALOMJEGYZÉK	i
BEVEZETÉS	vii
1. ADATBÁZIS-KEZELŐ RENDSZEREK	1-1
1.1. Információs rendszerek	1-1
1.1.1. Adatok	1-1
1.1.2. Hardver	1-2
1.1.3. Szoftver	1-2
1.1.4. Felhasználók	1-4
1.2. Adatbázis architektúra	1-6
1.2.1. Adatfüggetlenség	1-7
1.2.2. Az architektúra három szintje	1-8
1.2.2.1. Példa a háromszintű architektúrára	1-8
1.2.2.2. Külső szint	1-12
1.2.2.3. Koncepcionális szint	1-12
1.2.2.4. Belső szint	1-12
1.3. Az adatbázis-kezelő rendszer komponensei	1-13
1.3.1. Adatleíró és adatkezelő nyelv	1-13
1.3.1.1. Adatleíró nyelv	1-13
1.3.1.2. Adatkezelő nyelv	1-14
1.3.1.3. Vezérlésellenőrző nyelv	1-14
1.3.1.4. Illesztés a programozási nyelvekhez	1-14
1.3.2. Metaadatok	1-15
1.3.3. Grafikus felhasználói interfész	1-15
1.3.4. Szolgáltató programok	1-16
1.4. Adat- és adatbázis-felügyelő	1-16
1.4.1. Adat-felügyelő	1-17
1.4.2. Adatbázis-felügyelő	1-17
1.5. Az integrált adatbázis előnyei (és néhány hátránya)	1-18
1.5.1. Előnyök	1-18
1.5.2. Néhány hátrány	1-19
1.6. Adatbázis-kezelő rendszerek csoportosítása	1-20
1.6.1. Egy felhasználós rendszerek	1-20
1.6.2. Kliens-szerver architektúra	1-21
1.6.3. Osztott adatbázisok	1-22
1.6.3.1. Osztott adatbázisok architektúrája	1-24
1.6.3.2. Adat többszörözés	1-25
1.6.3.3. Horizontális adatmegosztás	1-25
1.6.3.4. Vertikális adatmegosztás	1-26
1.6.3.5. Kombinált adatmegosztás	1-26
1.6.3.6. Adatelosztási stratégiák	1-27
1.7. Ellenőrző kérdések	1-27
2. ADATTÁROLÁS ÉS ADATSZERVEZÉS	2-1
2.1. Adattárolók	2-2
2.1.1. Tárolási szempontok	2-2
2.1.2. Lemeztárolók	2-3
2.1.2.1. A lemez felépítése	2-3

2.1.2.2.	Hozzáférési idő.....	2-5
2.2.	Adatszervezési és adatelérési módok	2-7
2.3.	Lineáris tárolási struktúrák.....	2-8
2.3.1.	Soros szervezés és elérés.....	2-8
2.3.2.	Szekvenciális szervezés és elérés.....	2-8
2.4.	Közvetlen elérés	2-11
2.4.1.	Indexelés.....	2-11
2.4.1.1.	Index táblázat	2-11
2.4.1.2.	Szelektív indexek, a B ⁺ fa.....	2-13
2.4.1.3.	Nem szelektív indexek, bit-térképes index	2-16
2.4.1.4.	Join-index	2-18
2.4.2.	Hashing.....	2-19
2.5.	Adatszervezési és elérési módok összehasonlítása	2-21
2.5.1.	Sűrűsödés	2-21
2.5.2.	Elérési módok összehasonlítása	2-22
2.6.	Keresés több kulcs szerint.....	2-24
2.7.	Optimális elérési út meghatározása.....	2-25
2.8.	Adattömörítés	2-27
2.8.1.	Változó hosszúságú mezők	2-27
2.8.2.	Kódtáblázat.....	2-27
2.9.	Rendezés.....	2-28
2.10.	Ellenőrző kérdések	2-29
3.	ADATMODELLEK.....	3-1
3.1.	Az adatmodellezés célja	3-1
3.2.	Az adatmodellek fejlődése	3-2
3.3.	Főbb modellezési szempontok	3-3
3.4.	Egyed típusok és tulajdonságok	3-4
3.5.	Kapcsolatok.....	3-5
3.6.	Egyed típus – kapcsolat (Entity – Relationship) diagram	3-6
3.7.	Példa adatmodell létrehozására	3-10
3.8.	Ellenőrző kérdések	3-13
4.	RELÁCIÓS ADATBÁZIS-KEZELŐ RENDSZEREK.....	4-1
4.1.	A relációk tulajdonságai.....	4-1
4.2.	Példa relációkra	4-3
4.3.	Relációk az adatbázisban	4-5
4.4.	Kulcsok.....	4-6
4.4.1.	Az adatbázis integritása.....	4-6
4.4.2.	Elsődleges kulcs	4-7
4.4.3.	Relációk közti kapcsolatok.....	4-8
4.4.3.1.	Idegen kulcs.....	4-8
4.4.3.2.	Idegen kulcsot érintő műveletek	4-11
4.4.4.	További kulcsok	4-12
4.4.4.1.	Elhelyezési kulcs	4-12
4.4.4.2.	Keresési kulcs.....	4-13
4.5.	Relációs műveletek	4-13
4.5.1.	A legfontosabb relációs műveletek	4-13
4.5.2.	Átnevezés (RENAME).....	4-14
4.5.3.	Korlátozás (RESTRICT)	4-15
4.5.4.	Vetület (PROJECT).....	4-15
4.5.5.	Kereszt-Descartes szorzat (TIMES).....	4-16

4.5.6.	Egyesítés (JOIN)	4-17
4.5.6.1.	Természetes join.....	4-17
4.5.6.2.	Külső join	4-20
4.5.7.	Unió (UNION)	4-21
4.5.8.	Metszet (INTERSECT) és különbség (DIFFERENCE)	4-22
4.5.9.	Bővítés (EXTEND)	4-23
4.5.10.	Csoportképzés (SUMMARIZE).....	4-23
4.5.11.	Kijelölés	4-24
4.6.	Normalizálás.....	4-25
4.6.1.	Függőségek.....	4-26
4.6.2.	Első normál forma (1NF)	4-28
4.6.3.	Második normál forma (2NF)	4-31
4.6.4.	Harmadik normál forma (3NF)	4-35
4.6.5.	A dekomponálás más lehetőségei	4-38
4.6.6.	Nézetek definiálása normalizált relációkból	4-39
4.6.7.	Magasabb normál formák.....	4-40
4.6.7.1.	Boyce-Codd normál forma.....	4-40
4.6.7.2.	Negyedik normál forma	4-42
4.6.8.	A normál formák összefoglalása	4-44
4.7.	A relációs modellen alapuló adatbázisok összefoglalása	4-44
4.7.1.	Alapkövetelmények.....	4-45
4.7.2.	Főbb lépések.....	4-46
4.7.3.	Egy adatbázis megtervezése.....	4-46
4.7.3.1.	Feladatleírás	4-47
4.7.3.2.	A tervezési feladat megoldása.....	4-47
4.8.	Ellenőrző kérdések	4-52
5.	AZ SQL NYELV	5-1
5.1.	SQL szabvány	5-1
5.2.	Szintaxis jelölés.....	5-2
5.3.	A nyelv elemei	5-3
5.3.1.	Fenntartott szavak	5-4
5.3.2.	Azonosítók	5-4
5.3.3.	Állandók (literálok).....	5-6
5.3.4.	Operátorok és feltételek	5-7
5.3.4.1.	Aritmetikai operátorok	5-7
5.3.4.2.	Összehasonlító operátorok	5-8
5.3.4.3.	Logikai operátorok	5-11
5.3.4.4.	Halmaz operátorok	5-12
5.3.4.5.	Feltételek	5-13
5.3.5.	Határoló jelek	5-13
5.3.6.	Objektumok.....	5-13
5.3.6.1.	Adatbázis (Database).....	5-14
5.3.6.2.	Séma (Schema).....	5-14
5.3.7.	NULL érték	5-16
5.3.8.	SQL kifejezések	5-17
5.4.	Adattípusok	5-18
5.4.1.	Karakter adattípus	5-19
5.4.2.	Numerikus adattípus.....	5-20
5.4.3.	Dátum/időpont adattípus (DATE/TIME).....	5-21
5.4.4.	Sorazonosító (ROWID).....	5-22

5.4.5.	Hosszú adatok	5-22
5.5.	A legfontosabb SQL függvények	5-23
5.5.1.	Csoport függvények	5-23
5.5.2.	Oszlop függvények	5-24
5.5.2.1.	Karakter függvények	5-25
5.5.2.2.	Numerikus függvények	5-25
5.5.2.3.	Dátum/időpont függvények	5-26
5.5.2.4.	Konvertáló függvények	5-26
5.5.2.5.	Egyéb függvények	5-27
5.6.	SQL utasítások	5-29
5.6.1.	SQL utasítások összefoglalása	5-29
5.6.2.	Adatleíró utasítások	5-30
5.6.2.1.	Objektumok létrehozása (CREATE ...)	5-30
5.6.2.2.	Objektumok megszüntetése (DROP ...)	5-41
5.6.2.3.	Objektumok módosítása (ALTER ...)	5-42
5.6.2.4.	Jogosultságok kezelése (GRANT, REVOKE)	5-47
5.6.3.	Adatkezelő utasítások	5-50
5.6.3.1.	Az adatkezelő utasítások közös jellemzői	5-50
5.6.3.2.	Sorok kiválasztása (SELECT)	5-51
5.6.3.3.	Sorok törlése (DELETE)	5-64
5.6.3.4.	Sorok módosítása (UPDATE)	5-65
5.6.3.5.	Sorok bevitele (INSERT)	5-68
5.6.4.	Tranzakcióvezérlő utasítások (COMMIT, ROLLBACK)	5-71
5.7.	SQL utasítások beépítése programokba	5-73
5.7.1.	SQL utasítások csoportosítása	5-73
5.7.2.	SQL programok elkészítése	5-75
5.7.3.	Host változók	5-79
5.7.4.	Kurzor használata	5-81
5.8.	Ellenőrző kérdések	5-86
6.	ADATBÁZIS-KEZELŐ RENDSZEREK A GYAKORLATBAN	6-1
6.1.	MS ACCESS	6-1
6.1.1.	Szerkezet, felület, kezelés	6-1
6.1.2.	Adatbázis tervezés	6-2
6.1.2.1.	Táblák	6-3
6.1.2.2.	Feltételek	6-8
6.1.2.3.	Operátorok és függvények	6-11
6.1.2.4.	Táblák importálása	6-13
6.1.2.5.	Indexek	6-14
6.1.2.6.	Kapcsolatok	6-15
6.1.3.	Adatkezelés	6-17
6.1.3.1.	Adatbevitel	6-17
6.1.3.2.	Adatmódosítás	6-18
6.1.3.3.	Törlés	6-18
6.1.3.4.	Adatok megjelenítése, formázás, rendezés, szűrés	6-18
6.1.4.	Lekérdezések	6-20
6.1.4.1.	Választó lekérdezések létrehozása grafikusán	6-21
6.1.4.2.	Egyéb lekérdezések létrehozása grafikusán	6-28
6.1.4.3.	ACCESS SQL	6-32
6.1.5.	Űrlap	6-34
6.1.5.1.	Űrlap készítése gyorsvarázslóval	6-35

6.1.5.2.	Űrlap létrehozása varázslóval.....	6-36
6.1.5.3.	Űrlap tervének módosítása	6-38
6.1.6.	Jelentés	6-40
6.1.7.	Biztonság, adatvédelem.....	6-45
6.1.8.	Több felhasználós kezelés.....	6-46
6.2.	Internet-alapú adatbázis-kezelési technikák: PHP-MySQL.....	6-47
6.2.1.	Néhány szó a PHP nyelvről.....	6-47
6.2.2.	Adatbáziskezelés a PHP nyelvben	6-49
6.2.3.	A mySQL nyelv sajátosságai	6-50
6.2.4.	Kurzor használata PHP mySQL-ben.....	6-52
6.3.	Ellenőrző kérdések	6-53
7.	KATALÓGUS, ADATSZÓTÁR.....	7-1
7.1.	Katalógus táblázatok	7-1
7.1.1.	Katalógus relációk az objektumokról és összetevőikről	7-2
7.1.1.1.	Táblázat katalógus.....	7-3
7.1.1.2.	Oszlop katalógus	7-3
7.1.1.3.	Index és indexkomponens katalógus.....	7-4
7.1.1.4.	Objektumokkal kapcsolatos egyéb katalógusok	7-4
7.1.2.	Katalógus a hozzáférési jogokról	7-5
7.1.2.1.	Általános rendszer jogosultságok katalógusa.....	7-5
7.1.2.2.	Katalógus az objektumokra való jogosultságokról	7-5
7.1.2.3.	Katalógus a programok használati jogáról	7-6
7.1.3.	Katalógus a függőségekről	7-6
7.2.	Katalógus osztott adatbázisokban	7-7
7.3.	Statisztikák	7-8
7.4.	Ellenőrző kérdések	7-9
8.	GYAKORLATI PROBLÉMÁK ÉS MEGOLDÁSUK	8-1
8.1.	Tranzakciók.....	8-1
8.2.	Problémák párhuzamos feldolgozásnál.....	8-2
8.2.1.	Elveszett módosítás	8-2
8.2.2.	Nem véglegesített adatok feldolgozása	8-3
8.2.3.	Munka inkonzisztens adatokkal	8-3
8.3.	Zárak.....	8-4
8.3.1.	A zárok mérete	8-4
8.3.2.	A zárok erőssége.....	8-5
8.3.3.	A zárok időtartama	8-6
8.3.4.	A zár mechanizmus működése	8-7
8.4.	Patthelyzet	8-8
8.5.	Problémák és megoldásuk	8-11
8.5.1.	Be kellett a rácsos kaput zárni.....	8-11
8.5.2.	Utolsó pár, előre fuss!	8-13
8.5.3.	Hülye-biztos programokkal csak hülyék dolgoznak	8-13
8.5.4.	Nincs optimális optimalizáló.....	8-14
8.6.	Ellenőrző kérdések	8-15
9.	ADATTÁRHÁZAK.....	9-1
9.1.	Az adattárház jellemzői.....	9-1
9.2.	A hagyományos adatbázisok és az adattárházak összehasonlítása.	9-2
9.3.	Adattárház architektúra	9-3
9.4.	Adatszerkezet	9-4
9.4.1.	Csak beírható táblázatok	9-5

9.4.2.	Csillag elrendezés	9-8
9.5.	Adatbányászat	9-11
9.6.	Ellenőrző kérdések	9-11
10.	IRODALOMJEGYZÉK	10-1
11.	TÁRGYMUTATÓ	11-1

BEVEZETÉS

Amióta a Vízművek felépítette önálló adatbázisát, már a vízcsapból is adatbázis folyik. Saját adatbázisából szolgáltatja a Központi Adatfeldolgozó és Nyilvántartó Hivatal az állampolgárok, az APEH az adófizetők és nem fizetők adatait. Adatbázisukból állapítják meg az orvosok, hogy melyik betegüket milyen tünetekkel, mely orvosságokkal és mennyi ideig kezelték. Szerencsés esetben ugyanebből, rosszabb esetben egy másik adatbázisból (!) ellenőrzik az Országos Egészségügyi Pénztár szigorú revizorai, hogy mindez mennyibe került. Sőt, Kovács Pistike 1/b oszt. tan. is ünnepélyesen kijelentette apukája számítógépe mellett állva, hogy mihelyst – most már bukás nélkül – olyan magas osztályba fog jutni, ahol már az új oktatási törvény alapján is osztályzatokat kap, akkor egy adatbázisban fogja tárolni a jegyeit. Igaz, tette hozzá az érdeklődő újságírók és rokonok kérdéseire válaszolva, hogy az ahhoz való hozzáférést korlátozni fogja. Szüleinek csak a jeles és talán még a jó osztályzatok megtekintését fogja engedélyezni.

Valóban annyira fontosak-e számunkra az adatbázisok, annyira befolyásolják, megváltoztatják életünket, ahogy azt a médiában naponta megjelenő szalagcímek és a sok tucatnyi kis színes történet és híradás sugallja? Új tudományos, technikai eredmény, módszer, ami jelentős hatással bír a jelenünkre és a jövőnkre? Vagy csak egy a média által fölkapott divathóbortok közül, amelyik idővel lecseng és bekerül a tényleges fontosságának megfelelő helyre? Lehet-e, hogy az adatbázisok révén az Orwelli Nagy Testvér, az állam vagy más szervezetek, személyek minden tevékenységünkről, minden tulajdonságunkról tudomást szerezhetnek? Kell-e, hogy ezért személyiségi jogaink védelmében ugyanúgy harcoljunk a nagy adatbázisok létrehozása ellen, ahogy a XIX. század elején a vasút ellenzői azt követelték, hogy a mozdony előtt mindig haladjon egy lovas ember, aki figyelmezteti az arra haladókat a közelgő veszélyre? Vagy ahogy napjaikban egyes környezetvédők minden eszközt bevetnek az autópályák építése vagy az olcsó energia ellátást biztosító atomerőművek építése ellen.

Az adatbázisok létrehozása, használata a XX. század második felében lezajlott információs forradalom, az információs technológia világméretű elterjedésének a következménye. Mint szinte mindent, az adatbázisokat is lehet jól és rosszul, jóra és rosszra is használni. A tudósoknak, mérnöknek, törvényhozónak nem az a feladata, hogy az adatbázisok elterjedését, széleskörű alkalmazását gátolja, mert azok rossz célokra is felhasználhatóak. Az a kötelességük, hogy segítsék a szabályozott működtetésüket. Technikai és jogi eszközökkel biztosítsák, hogy a bennük levő információk megszerzése és felhasználása csakis a jól megfogalmazott törvényeknek megfelelő módon történhessen.

A személyi számítógépek széleskörű elterjedése, és a különböző számítógépek távközlési hálózattal való összekapcsolása lehetővé tette a háztartás vezetéséhez szükséges adatok otthoni számítógépes tárolásától kezdve az egész világot átfogó bankhálózatok egységes információs rendszerének gyors és megbízható elérését az arra jogosultak számára. Könyvünk ifjabb olvasói el sem tudják képzelni, mit jelent a mai utazó számára, aki a '70-es évek közepén még a három évente igényelhető 70 dolláros valuta keretéből gazdálkodhatott, hogy a PIN kódja megadásával Londonban vagy Párizsban azonnal fölvehet a magyarországi bankkártyájával akár 500 Eurót is (feltéve persze, hogy van a számláján annyi pénz vagy akkora hitelkeret). Az Internettel elérhető Web-es adatbázisok tovább bővítették az adatbázisokat használók körét. A milliányi napi feladat megoldását ellátó, úgynevezett tranzakciós adatbázisok mellett egyre inkább tért hódítanak a vezetői döntést támogató adatbázisok, az adattárházak is.

Ebben a könyvben a korszerű adatbázisok és adatbázis-kezelő rendszerek jellemzőit kívánjuk megmutatni. Továbbá azt, hogy miként kell az adatokat tárolni, milyen kapcsolatokat létesíthetünk közöttük, milyen szoftver módszerek állnak a rendelkezésünkre, hogy az adatokat naprakészen tarthassuk, az adatbázisból a kívánt információkat gyorsan megkaphassuk, és ugyanakkor az adatok biztonságát, ellentmondás mentességét is szavatolni tudjuk.

A témát elsősorban gyakorlati oldalról, a felhasználónak, az adatbázis létrehozójának és üzemeltetőjének a szempontjából tárgyaljuk.

Az 1. fejezetben általános összefoglalást adunk az adatbázisokról, az adatbázis-kezelő rendszerekről, felhasználási lehetőségeikről.

A felhasználói követelmények meghatározása után a 2. fejezetben ismertetjük, miként lehet ezeket megvalósítani a mai, modern adattároló eszközökön. Külön hangsúlyt helyezünk arra, hogy ne csak az elvi lehetőségekről kapjon tájékoztatást az olvasó, hanem arról is, miként lehet az itt felsorolt módszereket a gyakorlatban is hatékonyan megvalósítani.

Az élet és a számítógépes leképezés közti kapcsolatot az adatmodellek határozzák meg. (3. fejezet) Ezek közül napjaink szinte egyeduralkodóvá vált modelljét, a relációs modellt tárgyaljuk bővebben a 4. fejezetben. Részletesen ismertetjük az adatbázis-tervezésre szolgáló Entity – Relationship (Egyedtípus – Kapcsolat) módszert és a hozzákapcsolódó normalizációs eljárást. A szigorú matematikai formalizmus helyett szemléletes leírást használunk, mely lehetővé teszi a módszerek világos megértését, és a gyakorlatban való alkalmazását is.

Nagyon nagy hangsúlyt helyezünk az 5. fejezetben a ma már szabványnak tekintett adatbázis kezelő nyelv, az SQL ismertetésére. Napjainkban minden adatbázis-kezelő rendszer ezt, illetve ennek valamilyen részalmazát használja. Az itt leírtakat a gyakorlatban előforduló feladatok legnagyobb részében minden adatbázis-kezelő rendszerben egy az egyben alkalmazni lehet. Ezért ezt a fejezetet úgy állítottuk össze, hogy a tanulás mellett kézikönyvként is használható legyen. A megértést segíti a referencia könyvként használható szöveg mellett a sok alkalmazási példa.

A 6. fejezetben is a gyakorlati felhasználóknak szeretnénk segítséget nyújtani. Két Magyarországon is elterjedt rendszer, az ACCESS és a MySQL használatát tárgyaljuk olyan mélységben, hogy annak alapján az olvasó képes lesz önállóan is dolgozni ezekkel a rendszerekkel.

Az adatbázisnak az adatok mellett tartalmaznia kell az adatok definícióit, kapcsolatait, tárolási és használati módjuknak a leírását is. Az erre szolgáló adatszótárt, katalógust ismertetjük a 7. fejezetben.

A relációs modell megvalósításakor számos gyakorlati problémával kerülünk szembe. Ezek részben tervezési, részben működtetési, üzemeltetési problémák. Ezekre hívjuk fel a figyelmet és javasolunk megoldást a 8. fejezetben. Ez a témakör az adatbázisokkal foglalkozó könyvekben gyakran csak elnagyolva szerepel, holott mindennapi munkánkban gyakran szembekerülünk vele.

A 9. fejezetben az elsősorban döntéstámogatásra kifejlesztett adattárházak felépítését és működésük alapjait foglaljuk össze.

A ma működő adatbázisokat jellegük szerint két nagy csoportba, ezeken belül az elsőt három alcsoportba oszthatjuk.

- A hagyományos adatbázisok elsősorban a szervezet napi feladatait támogatják. A szervezet működéséhez szükséges *tranzakciók* regisztrálását, folyamatos feldolgozását végzik el. Ezeket hívjuk *tranzakció orientált adatbázisoknak*. Ezeken belül megkülönböztethetünk

- önálló személyi számítógépeken megvalósított egy személy vagy egy csoport információs igényeit kielégítő adatbázisokat, melyekhez egy időben *csak egyetlen egy felhasználó* férhet hozzá,
- központi nagy számítógépre telepített *centralizált, integrált adatbázisokat*, melyekhez hálózaton keresztül *többen több helyről* is hozzáférhetnek és több személy és csoport egymástól független összetett igényeit kell egy időben kielégíteniük,
- nagy és/vagy személyi számítógépek hálózatára alapozott *tartalmában integrált, de helyileg megosztott adatbázisokat*, melyben a leginkább helyileg igényelt adatok a felhasználás helyén vannak tárolva, ahol a hálózat bármelyik pontjáról bármikor elérhetőek. A mindenki által használt adatok tárolási helyének nyilvántartása egységesen történik.
- A vezetői döntések támogatására szolgáló igen nagy, statikus adatbázisok, az adattárházak, melyeket lekérdezések és elemzések végrehajtására terveztek. Különböző tranzakciós adatokból származtatott adatokat tárolnak egységes formában időbeli változásuk és felhasználási területük szerint csoportosítva.

A könyvben tárgyalt példák és módszerek elsősorban a centralizált, integrált adatbázisokra vonatkoznak. Ennek részben az az oka, hogy a működő adatbázisok zöme ebbe a csoportba tartozik. A másik ok az, hogy az itteni megoldások a többi típusnál is alkalmazhatóak. Az egy felhasználós rendszereknél könnyebbé teszi a dolgunkat az, hogy az adatszerkezet, az adatok közti kapcsolat általában egyszerűbb, és az egyidejű hozzáférésekből adódó problémákkal nem kell törődnünk. Az osztott adatbázisoknál figyelembe kell ugyan vennünk, hogy az információk megtalálásához és összeállításához igénybe vesszük az adatátviteli hálózatot is, de ez a megfelelő szoftver közbeiktatásával az adatbázis felhasználója előtt rejtve marad. Az adattárházakban is használhatjuk a hagyományos adatbázisok teljes eszköztárát, ami még néhány speciális eljárással kiegészül, hogy az adatokból kapható rejtett információkat is megtaláljuk. Ezek a különbségek az adatbázis-kezeléssel kapcsolatos feladataink lényegét nem befolyásolják.

A fő hangsúlyt olyan adatbázisok tervezésének, létrehozásának és működtetésének az ismertetésére helyezzük, melyekben

- igen sok, akár több száz különböző típusú rekord van,
- a különböző típusú rekordok között igen sokféle kapcsolat áll fenn,
- a rekordok száma több milliótól milliárdokig is terjedhet,
- akár több ezer műveletet (lekérdezés, módosítás, beírás, törlés) is végeznek óránként az adatbázisban,
- egyszerre ezer terminálról is elérhető és használható interaktívan az adatbázis,
- az adatok integritását, védelmét, biztonságát, helyreállíthatóságát a rendszer biztosítja.

Természetesen azok is hasznosan forgathatják a könyvet, akik a fent felsoroltaknál kisebb méretű, egyszerűbb felépítésű adatbázisokkal dolgoznak. Legfőljebb egyes problémákkal kevésbé élesen vagy később szembesülnek. Mindegy, hogy milyen gépen, milyen szoftverrel dolgozunk, általános érvényű szabály, hogy bizonyos adatmennyiség és/vagy tranzakció gyakoriság alatt nem kell törődnünk a hatékonysággal. A rendszer mindenképpen használható. Egy bizonyos küszöb fölött azonban nem árt, ha gondolunk rá, egy még magasabb szint fölött pedig már okvetlenül törődnünk kell azzal, hogyan szervezzük az adatainkat, hogyan fogalmazzuk meg a feladatainkat, hogy a rendszer hatékonyan, vagy akár egyáltalában működjön. Ne kelljen feleslegesen különleges hardverre pénzt kiadnunk és mégis kellő gyorsasággal kapjuk meg a kért információkat.

Az, hogy ezek a szintek 1 vagy 100 millió rekordnál, 10 vagy 100 tranzakció/percnél vannak, az a rendelkezésünkre álló hardver/szoftver kombináció mellett döntő mértékben

függ az adataink szervezésétől, tárolási módjától. Hogy ezt jól tudjuk megvalósítani, abban kívánunk ezzel a könyvvel segítséget nyújtani.

1. ADATBÁZIS-KEZELŐ RENDSZEREK

A hétköznapi szóhasználatban adatbázison egy adatfeldolgozó környezetben használt adatok összességét értjük. Ez a megfogalmazás azonban túl általános. Eszerint például a barátaink címeit és telefonszámait tartalmazó fájlt ugyanúgy adatbázisnak tekinthetjük, mint egy vállalat teljes integrált információs rendszerének alapjául szolgáló adatokat és a működtetését biztosító számítógépes szolgáltatásokat. *Mai értelmezésünk szerint az adatbázis logikailag kapcsolatban álló adatok szervezett összessége. Emellett azonban az adatokon kívül tartalmazza az adatok definícióját, értelmezését, és a különböző adatok között fennálló összefüggéseket is. Ezen kívül működni kell az adatbázison egy olyan adatbázis-kezelő rendszernek, amely lehetővé teszi az adatokból a bennük tárolt információ előállítását.* A későbbiekben az adatbázisnak ezt a definícióját még tovább fogjuk pontosítani.

1.1. Információs rendszerek

Az információs rendszerek, mint azt a nevük is mutatja, információkat szolgáltatnak. A korszerű információs technológia, az információs rendszerek alapját az adatbázisok képezik. Az információs technológia alapvető célja az, hogy növelje a szervezetben dolgozó emberek teljesítményét. Ebbe beletartozik, az egyes tranzakciók (pl. átutalások, megrendelések egyetemi vizsgák) regisztrálása, a technikai folyamatok dokumentálása és a döntésekhez szükséges információk szolgáltatása.

Minden információs rendszer négy fő részből tevődik össze. Ezek az

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

A köztük lévő kapcsolatot úgy fogalmazhatjuk meg, hogy *az információs rendszer adatokat tárol a hardveren, melyből az eszközök és a szoftverek segítségével a felhasználók információkat kaphatnak.*

1.1.1. Adatok

Az információ és az adat szavakat gyakran szinonimaként használjuk. Sokszor ez nem okoz problémát, de a pontos számítástechnikai terminológiában célszerű különbséget tennünk a kettő között. Az információ valamely jelenségre vonatkozó értelmes közlést jelent, melynek általában az új (legalábbis akkor számára új, vagy meglevő ismeretet megerősítő) ismerettartalma fontos a felhasználó számára. A számítógépes rendszerekben azonban az információkat nem közvetlenül tároljuk. Azok csak bináris jelsorozatként, értékekként jelennek meg. A számítógépes ábrázolásmódnak az értéke az adat, hogy ez mit jelent, az az információ. A 07/04/2007 jelsorozat lehet egy kétszeres osztás (aminek értéke 0,00087), de lehet egy dátum is, ami egy angol számára a 2007. év április hetedikét, míg egy amerikaiak ugyanezen év július 4-ét, a függetlenség napját jelenti. Hogy a fenti interpretációk közül melyik az igazi (vagy egy negyedik értelmezés a helyes), ahhoz tudnunk kell, hogy *mit* tartalmaz az adat (pl. dátumot) és azt is, hogy *milyen formában*. Ahhoz, hogy igazi információt adjon, még azt is tudnunk kell, hogy *minek a dátuma*.

Az adaton és a hozzákapcsolódó információn régebben tényeket rögzítő értékeket (pl. egy személy neve, lakcíme) értettünk. A mai adatbázisok már olyan objektumok tárolására is alkalmasak, mint például teljes dokumentumok, képek, hang- és videofelvételek. A

továbbiakban az adat fogalmába ezeket is beleértjük, bár a példáinkban többnyire csak az egyszerű elemi tényadatok szerepelnek.

Az adatok lehetnek

- elemi tényadatok,
- adatként tekinthető objektumok,
- elemi tényadatokból előállított származtatott adatok,
- az adatok leírását, értelmezését és formátumát tartalmazó metaadatok.

Az *elemi tényadatok* valami vagy valaki valamilyen jellemzőjének, tulajdonságának a konkrét értékét adják meg (Pl. Gipsz Jakab születési időpontja, hajszíne, utolsó havi fizetése).

A *származtatott adatok* elemi tényadatokból vagy származtatott adatokból állíthatók elő különféle műveletek segítségével. Származtatott adat lehet például az önálló elemi adatokként tárolt vezetéknév és a keresztnév egybevonásával (konkatenációjával) előállított név, a fizetés és az egyéb juttatások összegeként előállított teljes jövedelem. Külön csoportot képeznek a származtatott adatok között a különböző csoportosítások eredményeként létrejövő *aggregátumok*, mint például a naponként eladott áruk összértéke, vagy a tantárgyanként vizsgázók száma és átlagosztályzatuk. A napi műveleteket támogató tranzakció orientált adatbázisok a redundancia minimalizálása érdekében (néhány, a hatékonyság érdekében tett kivételtől eltekintve) nem tartalmaznak származtatott adatokat, míg a vezetői döntéseket támogató adattárházakban (ld. 9. fejezet) igen sok származtatott adat található.

Az adatbázisban a logikailag állandóan összetartozó adatokat nevezzük *rekordnak*. Ezek, mint azt az adatmodelleknél majd tárgyalni fogjuk (ld. 3. fejezet) a valós világ egyes egyed típusait (pl. hallgatók, tantárgyak, bankfiókok, átutalások) ábrázolják az adatbázisban. Ezeknek egyes tulajdonságai az elemi (vagy származtatott) adatok. Ezek a *rekord mezői*. Azt, hogy miként ábrázoljuk az egyes adatokat, mi a jelentésük, az információ tartalmuk, azt az adatbázisban tárolt *metaadatok* segítségével határozhatjuk meg. Ezeket az adatbázisnak egy önálló része, az *adatszótár*, más nevén a *katalógus* tartalmazza. A katalógust a 7. fejezetben ismertetjük részletesen.

1.1.2. Hardver

Minden adatbázis valamilyen konkrét hardveren valósul meg. Ennek főbb részei a processzorok, amelyeken az adatokat feldolgozzuk, az operatív memória és a lemezek, melyeken tároljuk, valamint a terminálok és a telekommunikációs rendszer. A hardverrel ezen könyvben csak érintőlegesen foglalkozunk, mivel ez szinte teljesen rejtve marad a felhasználók előtt. Csupán az adatok tárolására szolgáló lemezek felépítését és működését ismertetjük a 2.1. alfejezetben, mivel ezek hatékonyságát nagymértékben tudjuk befolyásolni az adatszerkezési és adatelérési módok (ld. 2.2. alfejezet) helyes használatával.

1.1.3. Szoftver

A hardveren fizikailag tárolt adatok és a felhasználó közti kapcsolatot a szoftver hozza létre. Ez három rétegből áll.

- az operációs rendszer,
- az adatbázis-kezelő rendszer,
- alkalmazási programok.

Az *operációs rendszer* vezérli és ellenőrzi a számítógépeken futó programok végrehajtását és kezeli a perifériákat (lemezek, terminálok, adatátviteli vonalak, stb.). Ebből kifolyólag az *adatbázis-kezelő 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

adatbázis-kezelő rendszernek nem kell az egyes perifériák speciális tulajdonságaival egyedileg törődnie, elegendő az input/output parancsokat az operációs rendszerek parancsainak szintjéig részletezni. Ez jelentősen lecsökkenti az adatbázis-kezelő rendszer elkészítéséhez szükséges munkát.

Az *alkalmazási program* határozza meg, hogy milyen adatokból milyen információkat kell összeállítani. Ennek logikája a konkrét alkalmazásától függ, és így kívül esik könyvünk témakörén. Az adatbázissal való kapcsolattartás technikai megvalósítását részletesen ismertetjük az 5. fejezetben, az SQL nyelvénél.

Az *adatbázis-kezelő rendszerekkel* a felhasználók az alkalmazási programokon keresztül kétféle módon léphetnek kapcsolatba.

- interaktívan, párbeszédes módon,
- programba beépíthető adatbázist kezelő, létrehozó és vezérlő utasításokkal.

Az interaktívan használható *önálló rendszereknek* (self-contained systems) saját, egyszerűen elsajátítható nyelve van. A probléma megoldását ezen a nyelven kell megfogalmazni. Ezek a nyelvek természetesen jóval szegényesebbek, mint a magas szintű programozási nyelvek és általában több gépidőt igénylő programokat hoznak létre. Bonyolult feladatok megoldására csak korlátozottan alkalmazhatóak. Használatuk azonban sokkal egyszerűbb és kényelmesebb, megtanulásuk is könnyebb. Nagy előnyük, hogy segítségükkel nem csupán előre megírt programok eljárásai és eredményei használhatók fel, hanem ad hoc igények is kielégíthetők az ember-gép párbeszéd révén.

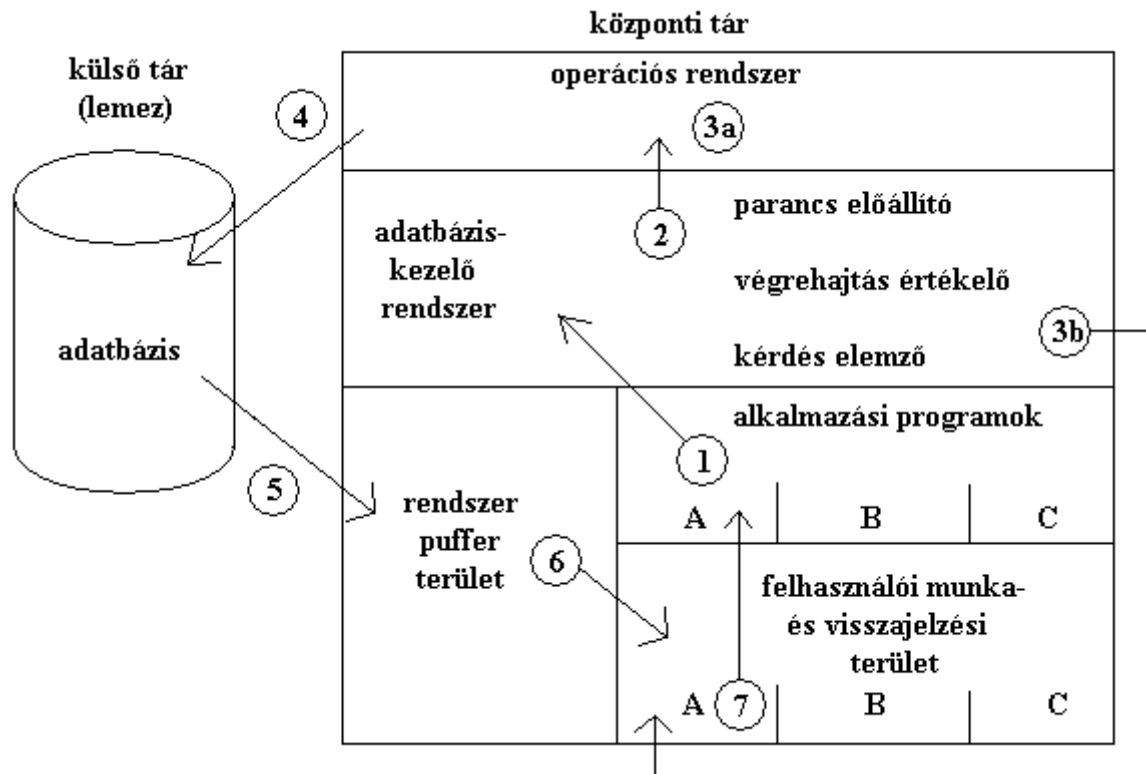
A *programba beépíthető rendszerek* (host language systems) önmagukban nem használhatóak. Csupán az adatbázis leírására, a beviteli-kiviteli (input-output) és egyéb az adatbázissal kapcsolatos műveletek megfogalmazására alkalmasak. Az adatokkal kapcsolatos további műveleteket valamilyen magas szintű programozási nyelven (pl. COBOL, Java, C++) kell megfogalmazni, melyből egy az SQL utasítás által generált szubrutinhívás hívja meg az adatbázis-kezelő rendszert. Az adatbázis-kezelő rendszer utasításainak programba való beépítését az 5.7. alfejezetben tárgyaljuk.

A kereskedelmi forgalomban kapható adatbázis-kezelő rendszerek többségének létezik önállóan használható és programba beépíthető változata is. Az alkalmazás igényeitől függ, hogy mikor melyikkel dolgozunk.

Az adatbázis-kezelő szoftver működésének elve az 1.1. ábrán látható. A példa egy adat beolvasásának folyamatát mutatja be.

1. Az *A* alkalmazási program adato(ka)t kér az adatbázisból. (Egyszerre több program is futhat, és ezek bármelyike fordulhat az adatbázis-kezelő rendszerhez.)
2. Az adatbázis-kezelő rendszer elemzi és értelmezi az utasítást. Megállapítja, hogy formailag (szintaktikailag) helyes-e, léteznek-e a program által kért objektumok és azok részei, és hogy a felhasználónak van-e jogosultsága az azokon kért műveletek elvégzésére.
3. Ha az elemzés eredménye pozitív, akkor az adatbázisra vonatkozó utasítást átalakítja az operációs rendszer által értelmezhető parancsokká és továbbítja ezeket az operációs rendszer felé. (3a ág). Egy adatbázis-művelet kérés általában egy egész parancssorozatot generál.
Ha a kérés bármilyen okból nem hajtható végre (az utasítás szintaktikailag hibás, nem létezik az adatbázisban olyan elem, amire az utasítás vonatkozik, vagy a kérő nem dolgozhat vele), akkor visszajelzést ad a felhasználó munka- és visszajelzési területére, megjelölve a visszautasítás okát is (3b ág).
4. Az operációs rendszer létrehozza a kapcsolatot a külső tárolóval. (ha ez nem sikerül, visszajelzést ad).

5. Az operációs rendszer a kért adato(ka)t behozza a rendszer munkaterületére (rendszer puffer).



1.1. ábra: Az adatbázis-kezelő szoftver működésének elve.

6. Az operációs rendszer átadja a kért adato(ka)t a programnak. Jelzi a művelet eredményességét, vagy (részleges) eredménytelenségének okát (pl. nincs az adatbázisban a kért adat), esetleg valamilyen különleges eseménynek az észlelését.
7. A felhasználói program átveszi az adato(ka)t, (ha kapott), értékeli a visszajelzéseket, és folytatja a feldolgozást.

Lényegében hasonló módon történik egy új adat beírása, meglévő törlése vagy módosítása, vagy az adatbázissal kapcsolatos bármilyen más művelet végrehajtása is.

Az egyszerűség kedvéért feltételeztük, hogy az alkalmazási program utasításai bent vannak a központi tárolóban. A gyakorlatban ez sokszor nem így van. Lehet, hogy a program a felhasználó kliens-gépén van (ld. 1.6.2. pont), sőt, az utasítás is ezen, interaktívan, a kérés kiadásakor készül el. Ekkor a kapcsolatba lép még a telekommunikációs program is. Ez azonban a fent elmondottak lényegét nem befolyásolja.

Könyvünk olvasói szempontjából a legfontosabb szoftver komponens az adatbázis-kezelő rendszer (Data Base Management System). Ezzel külön foglalkozunk az 1.3. alfejezetben.

1.1.4. Felhasználók

A felhasználás célját tekintve az adatbázisokkal dolgozók két nagy csoportját különböztetjük meg:

- adminisztratív célú felhasználók
- döntési információkat kérő felhasználók.

Ezen belül mindkét csoport a felhasználás módjától függően az alábbi alcsoportokra bontható:

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

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

A fentiekén kívül vannak még olyan személyek, akik az egész adatbázis működtetését irányítják, ellenőrzik. Ezek feladatait az 1.4.2. pontban, az *adatbázis felügyelőnél* ismertetjük.

Az *adminisztratív célú felhasználókra* általában jellemző, hogy *nagyszámú tranzakciót bonyolítanak le*. A tranzakció *jól definiált eljárás alapján* történik. A hatékonyság többé-kevésbé jól mérhető az időegység alatt végrehajtott tranzakciók számával, a feladat megválaszolásához szükséges idővel és az ehhez szükséges számítógépes erőforrások mértékével.

A *döntési információkat igénylő felhasználók* csak ritkán bonyolítanak le előre definiálható tranzakciókat. Feladataik változatosak, gyakran ad hoc jellegűek. A megoldás módja és az eredmények kiértékelése is sokféle lehet. Az adatbázisból nyert információk alapján döntések születnek. A rendszer hatékonyságát azzal mérhetjük, hogy mennyire segítik a szervezet fejlődését, működését ezek a döntések. Ez persze sokkal kevésbé pontosan definiálható, mint az előző csoportnál.

Az alkalmazás módja szerinti csoportosításnál az *alkalmazási programozók* a saját, vagy a végfelhasználói igények alapján készítik el a konkrét feladatok megoldását biztosító alkalmazási programokat. Az adatbázis-kezelő rendszert elsősorban a megfelelő adatoknak az adatbázisból való kikeresésére illetve az adatok változtatására használják fel. Ezeket a műveleteket építik be az adatbázissal dolgozó felhasználói programokba. Az adatbázist ily módon csak gyakorlott programozó tudja használni. Az általa elkészített programokkal dolgozó felhasználónak már nincs szüksége különleges számítógépes ismeretekre. De az adatbázissal csak a programban meghatározott módon léphet kapcsolatba, csak az ott beprogramozott módon használhatja. Az ilyen programokat alkalmazó tevékenysége lényegében ugyanaz, mint a következő bekezdésben ismertetendő paraméteres felhasználóké.

A programkészítésnél legtöbbször nem a program elkészítésének a gyorsasága, hanem a program rugalmassága, a menet közben változó igényekhez való adaptálhatósága és/vagy a futtatás hatékonysága a döntő.

A felhasználók többsége azonban nem programozó. Semmiféle, vagy csak minimális számítástechnikai ismeretekkel rendelkezik. Attól függően, hogy igényeik rögtönzöttek-e, formájukat és tartalmukat mindig a pillanatnyi helyzetnek megfelelően kell megfogalmazni, és ezáltal előre nem, vagy csak nagyon korlátozottan programozhatók be, vagy pedig néhány jól meghatározható, előre programozható témakörre és kérdéscsoportra korlátozódnak, megkülönböztethetünk *nem programozó alkalmazókat* és *menük alapján dolgozó felhasználókat*. Az előbbieknek a feladataik megoldásához egy általános, adatkezelésre alkalmas, nem számítógépes szakemberek által is könnyen megtanulható nyelvre van szükségük, ami beépül egy önállóan használható adatbázis-kezelő rendszerbe. Ennek segítségével maguk definiálhatják az adatokat és az adatkapcsolatokat és az adatokkal elvégzendő műveleteket. Egy ilyen rendszert, a Microsoft ACCESS-t a 6.1. alfejezetben tárgyalunk részletesen. A menü-orientált felhasználóknak még egy ilyen egyszerű szoftverre sincs szükségük. A szabványos, előre beprogramozott kérdéstípusok megválaszolásához szükséges paramétereket egy képernyőn megjelenített menürendszer aktuális bemenő adataiként adják meg. Ezekből, az előre elkészített programok alapján a rendszer szolgáltatja

az információkat. Tipikusan ezen alkalmazások közé tartoznak a banki befizető/kifizető /átutaló/lekérdező, vagy a repülőgépes helyfoglaló rendszerek.

A számítástechnika elterjedése, a felhasználóbarát rendszerek kifejlesztése eredményeként egyre növekszik azon nem informatikus szakemberek száma, akik saját feladataikat, információs igényeiket meg tudják úgy fogalmazni, hogy azt az adatbázis-kezelő rendszer segítségével közvetlenül végre is tudják hajtani. Az ilyen szakembereknek – a kezdeti elindítás után – már nincs vagy csak nagyon ritkán van szükségük számítógépes szakemberre, informatikusra a munkájukhoz, azon kívül, hogy azok a szükséges informatikai infrastruktúrát a rendelkezésükre bocsátják és megbízható működését biztosítják. Ez nagymértékben megnöveli az adatbázisok és az adatbázis-kezelő rendszerek rendszeres felhasználóinak számát és a felhasználások lehetőségét.

1.2. Adatbázis architektúra

Tágabb értelemben véve kétféle adatbázis típust különböztetünk meg. A *tény adatbázisok* meghatározott formában tárolt adatokat tartalmaznak, melynek egyedei között különféle kapcsolatok állnak fenn. Az adatbázis-kezelő rendszer a formázott adatok jelentése, értéke és a kapcsolatok alapján állítja elő a kívánt információkat. A tény adatbázisokon belül megkülönböztetjük az elsősorban a napi feladatok elvégzésére szolgáló *tranzakció orientált* adatbázisokat, melyek fő célja nagyszámú, előre pontosan meghatározott feladat, tranzakció (pl. raktárkészlet folyamatos nyilvántartása, bérszámfejtés) feldolgozása, és a vezetői döntéseket elősegítő *adattárházakat*. A másik nagy csoport, a *szöveges információ-visszakereső rendszerek* dokumentumokról tartalmaznak nem formázott, szöveges (képben, hangban) megadott információt. A rendszer azokat a dokumentumokat szolgáltatja, amelyekben egy adott szövegrész (kép-, hangrészlet) meghatározott körülmények között és formában előfordul.

A határvonal a két típus között egyre inkább elmosódik. Ma már minden szöveges információ-visszakereső rendszerben tárolunk tény adatokat is, és minden tény adatbázis-kezelő rendszerben lehet nem formázott adatokat, szövegmintákat keresni. Ezért a továbbiakban adatbázison a tény adatbázisokat értjük, ezeken belül pedig a produkciós adatbázisokat, és ezekkel is foglalkozunk. Az adattárházak speciális feladatait és jellemzőit külön ismertetjük a 9. fejezetben.

Az adatbázis-kezelő rendszereknek biztosítani kell

- különféle felhasználói igények hatékony kielégítését,
- adatfüggetlenséget,
- az adatok közötti komplex kapcsolatok ábrázolását,
- redundancia mentességet, illetve annak ellenőrzését,
- egyszerű használatot,
- az adatok védelmét, nehogy illetéktelenek hozzáférhessenek,
- az adatok integritását, hogy lehetőleg a hozzáférésre jogosultak se ronthassák el az adatbázist,
- helyreállíthatóságot, hogy bármilyen hiba esetén az eredeti állapotot vissza lehessen állítani,
- több felhasználós adatbázisnál az egyidejű hozzáférést,
- osztott adatbázisnál az adatok fizikai szétosztását, logikai összevonását és a duplikátumok konzisztenciáját.

A továbbiakban ezen feladatok közül azokat, melyek tartalma nem magától értetődő részletesen elemezzük.

A fentiek mellett tudomásul kell vennünk, hogy 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.

1.2.1. Adatfüggetlenség

Mivel ugyanazon adatokat több program és több felhasználó is használhatja, az adatokat és a programokat, amennyire lehet, függetleníteni kell egymástól. Ha például az egyik programban a feldolgozási szempontok módosulása miatt a rekordokat egy új mezővel bővítjük, akkor ez a változtatás lehetőleg csak ennek az egy programnak a változtatását tegye szükségessé. Az összes többi program, amelyik ugyanezen rekordokat, vagy ezek egy részét továbbra is a változatlan formában használja, változatlan maradjon. Ennek megvalósítása érdekében külön kell választanunk az adatok fizikai leírását a programok által látott adatszerkezettől. Ezzel azt is elérhetjük, hogy ugyanazon numerikus adatokkal az egyik programban bináris, a másokban decimális számokként dolgozhatunk, és ha az adatbázisban a belső ábrázolásmódot lebegőpontosra módosítanánk, akkor sem kellene egyik programban sem változtatni. Ugyanígy lehetséges, hogy ugyanaz a dátum az egyik programban 2007.3.15, míg egy másikban MARCH-15-2007 formában jelenik meg és kerül felhasználásra. Jó adatbázisoknál a nagy félelemmel várt „YEAR 2000 katasztrófa”, a régi, esetleg kétjegyű évszámokról való átállás nem okozott semmiféle komolyabb problémát. A különböző belső adatábrázolási módok közötti konverziót elvégezte az adatbázis-kezelő rendszer.

Egy rendszer *fizikailag adatfüggetlen*, ha az adataival dolgozó felhasználói programok és a felhasználók ad hoc információkérései gyakorlatilag függetlenek az adatok tárolási és elérési módjától. Ez azt jelenti, hogy ha például az adatokat egy régebbi, lassabb mágneslemezről áthelyezik egy korszerűbb, gyorsabb lemezre, akkor ebből a felhasználók semmit sem vesznek észre, hacsak azt nem, hogy az adatbázissal gyorsabban tudnak kommunikálni.

A *logikai adatfüggetlenség* viszont azt jelenti, hogy az adatbázis logikai szerkezetében létrehozott változások az adatbázist felhasználó programokat nem, vagy csak minimális mértékben befolyásolják, mint ezt az előbbi példákkal illusztráltuk.

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

Természetesen a tökéletes adatfüggetlenség nem valósítható meg. Ezért szerepelnek a meghatározásokban a „gyakorlatilag” illetve a „minimális” szavak. Hiszen a hardver módosításával, vagy az adatelérési stratégia megváltoztatásával nagymértékben befolyásolhatjuk a rendszer működési sebességét. A felhasználó szintjén azonban ezek a változások nem abban jelentkezik, hogy új programokat kell írnia, másképpen kell a kérdéseit megfogalmaznia, hanem úgy, hogy az eredményt, a választ gyorsabban (vagy esetleg más felhasználók igényeinek hatékonyabb kielégítése miatt éppenséggel lassabban), drágábban vagy olcsóbban fogja megkapni.

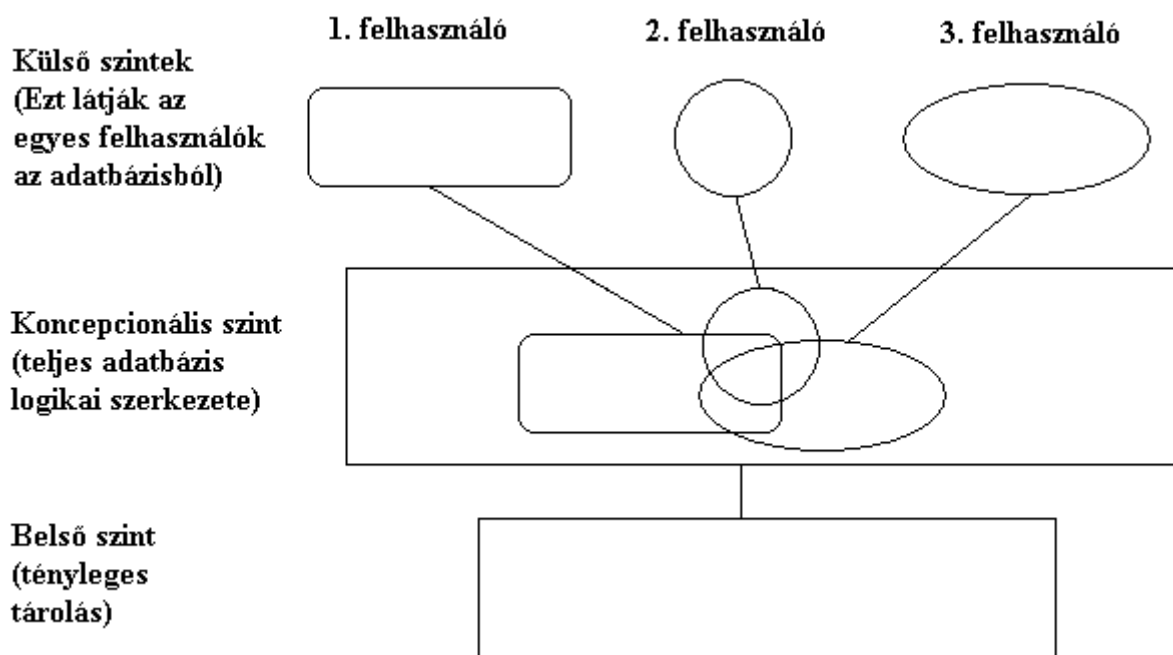
1.2.2. Az architektúra három szintje

Az adatmodell a valós világ objektumait, azok tulajdonságait és kapcsolatait írja le. *Az adatmodell nem a konkrét adatokkal, azok értékeivel foglalkozik, hanem azok típusaival, kapcsolataival.* A feladatmegoldás első lépése a megfelelő adatmodell kialakítása. Ez a valós világot tükrözi híven, a kívánt részletességgel, de emellett alkalmas arra, hogy az általa reprezentált adatszerkezeten a valóság mozgásai a rendelkezésre álló számítástechnikai eszközökkel nyomon követhetők legyenek. Az adatmodelleket és elkészítésüket a 3. és 4. fejezetben ismertetjük részletesen.

Az adatfüggetlenség biztosítása érdekében az adatmodell szerkezetileg az 1.2. ábrán látható három szintre bomlik.

- *Belső* vagy más néven *fizikai szint*, amely az adatoknak a tárolón való fizikai elhelyezését és fizikai elérési lehetőségeit írja le.
- *Külső* vagy más néven *logikai szintek*, melyek azt írják le, miként látják az egyes felhasználók az adatbázist.
- *Koncepcionális szint*, amely azt írja le, hogy logikailag egységbe vonva hogyan néz ki ténylegesen a teljes adatbázis. Ennek látják különböző vetületeit a külső szinteken a felhasználók, és ez képződik le egyértelműen tárolási struktúráként és elérési módokként a belső szinten.

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



1.2. ábra: Az adatbázis architektúra három szintje.

1.2.2.1. Példa a háromszintű architektúrára

Az adatmodell háromszintű felépítését egy egyszerű példán mutatjuk be. Ezt a példát különböző formában a könyv további részeiben is rendszeresen használni fogjuk.

Modellünk egy egyszerűsített hallgatói nyilvántartást ír le.

A koncepcionális szint az alábbi egyed típusokat (objektumokat) és ezeknek a következő tulajdonságait (jellemzőit) tartalmazza:

HALLGATÓ: A hallgató vezeték- és keresztnéve, egyetemi azonosító kódja, születési időpontja, évfolyama.

TANTÁRGY: A tantárgy azonosító kódja, megnevezése, oktatója (csak egyet, a vezető oktatót adhatjuk meg), heti óraszám, és hogy kötelező-e vizsgázni belőle.

MITHALLGAT: Ki, milyen tantárgyat vett fel. Ebből hány gyakorlati pontot kapott és hogyan vizsgázott. A „ki”-t a hallgató egyetemi azonosító kódja, a „milyen tantárgyat” a tantárgy azonosító kódja határozza meg.

A koncepcionális szintbe még azt is bele vesszük, hogy a hallgatót és a tantárgyat is *egyértelműen* azonosítja az adatbázison belül a megfelelő azonosító kód, egy hallgató ugyanazt a tárgyat csak egyszer veheti fel, nem lehet nem létező tantárgyat felvenni, és nem hallgathat egy tantárgyat sem nem létező hallgató. Ez azt jelenti, hogy az adatbázis-kezelő rendszer automatikusan megakadályozza, hogy két azonos kódú hallgatót vagy tantárgyat vegyünk fel az adatbázisba, vagy egy MITHALLGAT adatrekordban nem létező hallgató vagy tantárgykód szerepeljen, vagy egy hallgató többször vegye fel ugyanazt a tárgyat¹. Feltételként kikötjük még azt is, hogy egy hallgató adatait csak akkor vesszük fel az adatbázisba, ha legalább a hallgatói azonosító kódját és vezetéknevét ismerjük, egy tantárgyat, ha a tantárgyi kódját és megnevezését ismerjük. Ugyancsak ismernünk kell egy MITHALLGAT rekord felvételekor mind a hallgató, mind a tantárgy kódját.²

A fizikai szint leírásához megadjuk, hogy a vezetéknev, keresztnév, oktató, a tantárgy megnevezése 30 karakter, a hallgatói azonosító 5, a tantárgy azonosító 4 karakter, az évfolyam, az óraszám és az osztályzat egyjegyű szám, a gyakorlati pontszám háromjegyű decimális szám és a kötelező vizsgát az „I”, a nem kötelezőt az „N” karakter jelenti. Ezen kívül azt is előírjuk, hogy a hallgatókat azonosítójuk és nevük alapján, a tantárgyakat azonosítójuk, megnevezésük és oktatójuk alapján azonnal meg akarjuk találni, és ezért az adatbázis-kezelő rendszer hozza létre az ennek megvalósításához szükséges adatszerkezetet (ld. 2.4.1. pont). Végül – általában az operációs rendszer közvetítésével – azt is meghatározzuk, hogy az egyes objektumok adatai melyik lemez melyik területén legyenek.

A végső adatmodellben mindig csak egy koncepcionális és egy belső, fizikai szint van.³ Ezzel szemben akárhány különböző külső, logikai szint lehet, melyek közül az egyik megfelelhet teljesen a koncepcionális szintnek. Ugyanazzal a logikai szinttel több felhasználó is dolgozhat.

Példánkban két felhasználó számára definiálunk önálló külső szintet:

1. Felhasználó külső szintjének egyedtípusai és tulajdonságai:

HALLGATÓ: Azonos a koncepcionális szint HALLGATÓ objektumával

¹ Természetesen modellünk a valóságot kissé leegyszerűsítve ábrázolja, hiszen egy hallgató egy tárgyból többször is vizsgázhat (pl. megbukott), többször is felveheti. Ezekkel a problémákkal most nem törődünk. Ha azt mondjuk, hogy a modell csak a legutolsó állapotot tükrözi, az előzményeket nem tartalmazza, akkor ábrázolásmódunk már teljesen megfelelő.

² Ezek a feltételek azt jelentik, hogy egy új rekord felvételekor nem kell annak minden elemét ismernünk. Elfogadjuk hiányos rekordokat is (lehet olyan adat, amit a felvételkor nem is tudhatunk, például tanév elején az osztályzat.) De megszabjuk, mi az a minimális tartalom, ami szükséges egy rekord beírásához.

³ Az, hogy egy adatbázisnak csak egy koncepcionális és csak egy fizikai szintje lehet, nem jelenti azt, hogy ezek az igények változásával nem változhatnak. Az egyediség mindig csupán egy adott időpontra vonatkozik.

MITOKTAT: Tartalmazza a koncepcionális szint TANTÁRGY objektumából a tantárgy megnevezését és az oktató nevét, de mindkettőt 15 karakterre lerövidített formában

Ez a felhasználó a koncepcionális szint egyik egyedtypusát (HALLGATÓ) teljes mértékben, egy másiknak (TANTÁRGY) csak egy részét láthatja. Az adatbázis bizonyos részeihez egyáltalában nem férhet hozzá. Így például a vizsgaeredményekből semmit sem láthat, sőt azt sem tudja, hogy az egyes hallgatók milyen tantárgyakat vettek fel.

2. Felhasználó külső szintjének egyedtypusai és tulajdonságai:

VIZSGAEREDMÉNY: Tartalmazza a hallgató vezeték és keresztnévét egy tulajdonságként, névként összevonva, a tantárgy kódját és a hallgatónak ebből a tantárgyból elért osztályzatát. A hallgató nevét az adatbázis-kezelő rendszer állapítja meg. A MITHALLGAT minden tantárgykód - osztályzat kettőséhez tartozó hallgatói azonosító kódhoz kikeresi HALLGATÓ-ból az ehhez a kód értékhez tartozó nevet.

Ez a felhasználó a hallgatók adataiból a neveken, és a tantárgyakból elért osztályzataikon kívül semmit sem láthat.

Az 1.3. ábrán bemutatjuk a példában szereplő adatbázis koncepcionális, az 1.4. ábrán a külső szintjeit. A különböző objektumokra fennálló feltételeket ezen az ábrán nem tüntettük fel. Az 1.5. ábrán ugyanebben az adatbázisban az egyes egyedtypusok konkrét előfordulásai, az adatokkal feltöltött adatbázis egy része látható.

KONCEPCIONÁLIS SZINT

HALLGATÓ

vezetéknév	30 karakter
keresznév	30 karakter
egyetemi azonosító kód	5 karakter
születési dátum	dátum
évvolyam	1 jegyű szám

TANTÁRGY

tantárgykód	4 karakter
tantárgy megnevezése	30 karakter
oktató	30 karakter
óraszám	1 jegyű szám
vizsga	1 karakter

MITHALLGAT

hallgató azonosító kódja	5 karakter
tantárgy kódja	4 karakter
gyakorlati pontszám	3 jegyű szám
osztályzat	1 jegyű szám

1.3. ábra: Hallgatói adatbázis koncepcionális szintje.

1. KÜLSŐ SZINT	
MITOKTAT	
tantárgy megnevezése	15 karakter
oktató neve	15 karakter

HALLGATÓ	
vezetéknév	30 karakter
keresztnev	30 karakter
egyetemi azonosító kód	5 karakter
születési dátum	dátum
évvolyam	1 jegyű szám

2. KÜLSŐ SZINT	
VIZSGAEREDMÉNY	
név (vezeték+keresztnev)	60 karakter
tantárgykód	4 karakter
osztályzat	1 jegyű szám

1.4. ábra: Hallgatói adatbázis külső szintjei.

HALLGATÓ		<i>koncepcionális és 1. külső szint</i>		
Ajkai	István	00001	1990-02-11	4
Bárcy	Benő	00002	1992-11-21	2
Zlatkó	Péter	00003	1991-03-13	3
Kővágóhegyi Ritter	Aladár Nándor	00004	1992-12-31	3

TANTÁRGY		<i>koncepcionális szint</i>		
SQL1	Adatbázis-1	Quittner Pál	4	I
AB00	Adatbázis-2	Baksa-Haskó Gabriella	3	I
ELFS	Elméleti fizika	Albert Zweistein	2	N

MITHALLGAT		<i>koncepcionális szint</i>		
00001	SQL1	70	4	
00004	SQL1	10	1	
00002	AB00	60	3	
00002	ELFS	100	5	
00003	SQL1	15	1	

MITOKTAT		<i>1. külső szint</i>	
Adatbázis-1	Quittner Pál		
Adatbázis-2	Baksa-Haskó Gab		
Elméleti Fizika	Albert Zweistei		

VIZSGAEREDMÉNY		<i>2. külső szint</i>	
Ajkai István	SQL1	4	
Kővágóhegyi Ritter Aladár Nándor	SQL1	1	
Bárcy Benő	AB00	3	
Bárcy Benő	ELFS	5	
Zlatkó Péter	SQL1	1	

1.5. ábra: Hallgatói adatbázis koncepcionális szintjén tárolt és a külső szintekről látható adatai.

1.2.2.2. Külső szint

A külső, más néven logikai szint az az ábrázolási mód, ahogyan az *egyes felhasználók látják az adatbázist*. Leírásához ugyanazt az adatmodellt használjuk, mint a koncepcionális szinthez. Egyed típusait (objektumait) és azok tulajdonságait a koncepcionális szinten definiált egyed típusokból, azok tulajdonságaiból és a köztük lévő kapcsolatokból definiáljuk az adatleíró nyelv (ld. 1.3.1.1. pont) segítségével. A külső szint egyed típusai megegyezhetnek a koncepcionális szint egyed típusaival és a belső szint rekordjaival, mint például az 1. felhasználó HALLGATÓ objektuma. De lehet annak csupán egy része, mint az 1. felhasználó MITOKTAT objektuma, vagy összeállhat több egyed típus adatainak meghatározott szempontok alapján történő összeválogatása alapján is (2. felhasználónál a VIZSGAEREDMÉNY).

1.2.2.3. Koncepcionális szint

A koncepcionális szint írja le a teljes logikai adatszerkezetet. *Miként néznének ki mindenki számára az adatok*, ha mindenki mindent láthatna belőlük. Tartalmazza az összes egyed típust, ezek adatbázisbeli leképezéseinek, a rekordoknak a leírását. Pontosan meghatározza minden egyed típus tartalmi leírását, és tulajdonságait. Például van egy HALLGATÓ egyed típusunk (rekord típusunk), annak van egy VEZETÉKNÉV tulajdonsága (oszlopa, mezője), melynek egy konkrét értéke egy meghatározott hallgató vezetéknévét jelenti. De az adatfüggetlenség érdekében ezen a szinten közvetlenül nem utalunk arra, hogy ez karaktereket tartalmaz 30 bájt hosszúságban, hol találhatók ezek az adatok és hogyan férhetünk hozzájuk. Ezen információkat az adatbázis szerkezetét leíró metaadatok, a katalógus tartalmazza, és a fizikai szinten használjuk fel őket.

A koncepcionális szinten írjuk le azt is, hogy milyen feltételek, kapcsolatok vannak az egyes rekord típusokon belül és a különböző rekord típusok között, mint azt az 1.2.2.1. pont példájában bemutattuk.

Bár elvileg nem tartozik az adatmodellhez, gyakorlatilag itt adjuk meg azt is, hogy a különböző rekord típusokhoz, vagy azok egyes részeihez kik és milyen célból férhetnek hozzá. Itt határozzuk meg az adatbázis védelmének logikáját.

1.2.2.4. Belső szint

A belső, fizikai szint leírja az *adatok tárolásának, szervezésének és elérésének módját*.

A *tárolási mód* megadja, melyik lemezeken, annak melyik cilinderein, mely sávjaiban és blokkjaiban helyezkednek el az adott rekord típus egyes rekordjai. Ezen belül hol vannak az egyes rekordok, a rekordokon belül azok mezői, milyen konverziót kell elvégezni a fizikailag tárolt adatokon, hogy a felhasználóknak a megfelelő formában álljanak rendelkezésre.

A *szervezési mód* megadja, hogyan, milyen rendben helyezkednek el a rekordok, milyen struktúrákat építettünk fel rájuk, hogy lehetővé tegyük egyszerű és gyors elérésüket.

Az *elérési mód* az a tényleges hozzáférési út, ahogy az alkalmazásban a különböző szervezési módok közül kiválasztjuk azt, amellyikkel az adott esetben elérjük az adatokat.

Tipikus szervezési és elérési módok:

- A tárolón való elhelyezkedésnek megfelelő *fizikai sorrend*.
- Valamelyik mező(k) értékei szerinti *logikai sorrend*.
- Valamelyik mező(k) értéke szerint *közvetlen hozzáférés*.

A belső szinttel, miután ennek meghatározása az adatbázis működését gyakorlati szempontból igen erősen befolyásolja a 2. fejezetben részletesen foglalkozunk.

1.3. Az 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 elemzi, értelmezi, és ha végrehajthatóak, meghatározza a végrehajtás várhatóan leghatékonyabb módját.
- *Elérés meghatározó.* Az adatok elérésére meghatározott módszert átfordítja az operációs rendszer által értelmezhető parancsokká. Elindítja, majd értékeli ezen parancsok végrehajtását.

A fenti komponensek együttműködését az 1.1.3. pontban ismertettük és az 1.1. ábrán mutattuk be.

A korszerű adatbázis-kezelő rendszerektől megköveteljük, hogy az alábbi szolgáltatásokat nyújtsák:

- valamennyire tudják használni csekély számítástechnikai, informatikai ismeretekkel rendelkezők is,
- legyen felhasználóbarát nyelve az adatbázisban való műveletek elvégzésére,
- lehessen használni párbeszédes (interaktív) üzemmódban,
- egyszerűen lehessen az adatbázis-kezelő utasításokból programnyelvi utasításokat generálni,
- biztosítsa a rendszer meghatározó jellemzőinek automatikus dokumentálását,
- létezzen hozzá grafikus interfész,
- nagy adatmennyiségek rutin feldolgozásához szolgáltatson hatékony eljárásokat.

A kereskedelmi forgalomban kapható rendszerek többé-kevésbé teljesítik a fenti követelményeket, bár bizonyos esetekben ez csak külön szoftver termékek megvásárlásával biztosítható.

1.3.1. Adatleíró és adatkezelő nyelv

Napjainkban minden adatbázis-kezelő rendszer az adatleíró, adatkezelő és az adatbázissal kapcsolatos egyéb feladatok megfogalmazására az SQL nyelvet, illetve ennek valamely részhalmazát használja. A grafikus felületeken kommunikáló rendszerek is erre a nyelvre fordítják át a kijelölt feladatokat. Az SQL nyelvet részletesen ismertetjük az 5. fejezetben.

1.3.1.1. Adatleíró nyelv

Az adatmodellnek a számítógépes feldolgozás számára alkalmas meghatározását az *adatleíró nyelven (Data Definition Language, DDL)* fogalmazzuk meg. Ezzel definiáljuk a koncepcionális és a külső szintet és a fizikai struktúrák és kapcsolatok leírása is részben ezen a szinten történik.

Az adatleíró nyelvnek három különböző szinten kell leírni az adatbázis különböző elemeit:

- A felhasználó szintjén, amely az adatbázis valamelyik logikai szintjét jelenti.
- Az adatbázis összes egyedtípusának, kapcsolatának és tulajdonságának, az adatbázis rekordjainak és mezőinek szintjén. Ez a koncepcionális modell. A leírás alapján realizált struktúra maga az adatbázis.

- Az adatok fizikai tárolásának és szervezésének szintjén. Ez az egyes adatok fizikai tárolásának helyét, tárolási formáját, hozzáférési lehetőségeit definiálja.

Az adateleíró nyelv az adatbázis objektumainak létrehozása mellett alkalmas meglévő objektumok megszüntetésére és módosítására is. Ennek SQL nyelvi megvalósítását az 5.6.2. pontban tárgyaljuk.

Az adatbázis felhasználói – technikai ismereteiktől és az adatbázisban való jogosultságaiktól függően – dolgozhatnak az adateleíró nyelv különböző szintjeivel. Az átlagos felhasználó az első vagy legfőbb az első és második szintet használhatja, míg a harmadik szinttel csaknem kizárólag az adatbázis-felügyelő (ld. 1.4.2. pont) dolgozhat.

1.3.1.2. Adatkezelő nyelv

A feldolgozás céljaira szükségünk van egy *adatkezelő nyelvre* (*Data Manipulation Language, DML*). Ennek segítségével a felhasználó az *adateleíró nyelvvel definiált adatbázison műveleteket végezhet*. Ezek a műveletek az adatbázis adatait létrehozhatják, kiválaszthatják, módosíthatják, vagy törölhetik. Amennyiben a változtatások olyan mezőket is érintenek, melyhez a fizikai szinten valamilyen elérési struktúra is tartozik, akkor az adatbázis-kezelő rendszer ennek aktualizálását is elvégzi. Az SQL nyelv adatkezelő műveleteit az 5.6.3. pontban ismertetjük.

A régebbi adatbázis-kezelő rendszerek adateleíró 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 mondták annak idején, hogy navigál az adatbázisban. Aki jól ismerte a különböző utakat, 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.

Napjaink relációs adatbázis-kezelő rendszereiben automatizálták az adatbázisban való útkeresést. A felhasználó csak azt mondja meg, mi a célja. Az ehhez vezető utat a rendszer határozza meg. Ezáltal az adatbázis-kezelő rendszerek használata könnyebbé vált. Hatékony alkalmazásukhoz a gyakorlatban előforduló egyszerűbb esetek többségében nem kellenek komolyabb informatikai ismeretek. Többek között ez is oka volt annak, hogy a relációs rendszerek egyeduralkodóvá váltak.

1.3.1.3. Vezérlésellenőrző nyelv

Az adatbázisban történő műveletek vezérlésére, ellenőrzésére szolgál a *vezérlésellenőrző nyelv* (*Data Control Language, DCL*).¹ Ennek segítségével szabályozható, hogy ki, min, milyen műveleteket végezhet el, meddig lehet az elvégzett műveletek eredményeit hatálytalanítani, az eredeti állapotot visszaállítani, illetve mikortól válnak azok véglegessé, visszavonhatatlanná. A hozzáférést szabályozó SQL utasításokat az 5.6.2.4. pontban, a műveletek eredményét vezérlő utasításokat pedig az 5.6.4. pontban tárgyaljuk.

1.3.1.4. Illesztés a programozási nyelvekhez

Az adatkezelő és az adateleíró nyelvet használhatjuk önállóan, interaktív módon (ld. 1.1.3. pont). Ekkor a felhasználó a termináljáról adja ki az utasításait, és itt kapja meg az eredményeket is. Az ilyen nyelvek műveleti lehetőségei korlátozottak. Ha az adatokon

¹ A Data Control Language-nek nincs elfogadott magyar megfelelője. Az általunk használt vezérlésellenőrző nyelv jobban kifejezi a nyelv funkcióját, mint a szószерinti adatellenőrző vagy adatvezérlő nyelv.

bonyolult algoritmus szerint kell műveleteket elvégezni, akkor ehhez az adatokat programból kell közvetlenül kezelni. Ezt teszik lehetővé a programba beépíthető adatbázis-kezelő rendszerek. A programozó az adatbázis-kezelő utasításokat ugyanúgy beírhatja a programjába, mint a programnyelv többi utasítását. Ezek végrehajtásakor az adatbázis és a program közti adatcserét az adatbázis-kezelő rendszer végzi el. A kívánt adatok bekerülnek a programba, ahol megtörténhet a további feldolgozásuk, illetve a programban feldolgozott adatok kiíródnak az adatbázisba.

Az 5.7. alfejezetben ismertetjük, miként lehet az SQL nyelv utasításait beépíteni programokba.

Az adatbázis-kezelés programból történő megvalósításának előnyei:

- Bonyolultabb, összetett műveleteket is el lehet végezni az adatokon.
- A programokat abban a nyelvben lehet írni, amelyik a legjobban illeszkedik a feladathoz, vagy amelyiket az installációnál általánosan használnak, vagy amelyiket a programozó (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 van párbeszédés és beépíthető változata is. Az utóbbiak beépíthetők többek között COBOL-ba, JAVA-ba, PHP-ba, C és C++-ba is.

1.3.2. Metaadatok

A *metaadatok* az adatbázis szerkezetét, objektumainak, adatainak elnevezését, tulajdonságait, szerkezetét írják le. Tárolásuk az adatbázis egy önálló részében, a 7.fejezetben ismertetendő *katalógusban*, más néven *adatszótárban* történik. Az adatbázis felépítésének (nem a konkrét adatoknak!) bármilyen megváltozása automatikusan aktualizálja a katalógust is. Ezáltal a katalógus mindig naprakészen tükrözi az adatbázis állapotát. Az adatszótár használata nagymértékben lecsökkenti az adatbázis dokumentálásával kapcsolatos munkákat.

Legegyszerűbb formájában a katalógus az adat, rekord és objektum definíciókat, a köztük fennálló kapcsolatokat, a hozzáférési és műveleti jogokat, az adatbázis architektúra mindhárom szintjének pontos, naprakész definícióját tartalmazza.

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

1.3.3. Grafikus felhasználói interfész

Az adatbázissal kapcsolatos napi munkák egyik legkritikusabb eleme a tranzakciók végrehajtásához szükséges adatok bevitele. Nagymértékben megkönnyíti a felhasználók munkáját, ha az adatokat egy jól megtervezett beviteli képernyőről egyszerűen lehet bevinni az adatbázisba. Az adatbázis-kezelő rendszerhez tartozó, vagy külön kiegészítésként megvásárolható *képernyő tervezővel* kijelölhetjük a képernyőn, hová, milyen adatot kell beírni, a beírt adat plauzibilitását azonnal ellenőrizhetjük, hiba esetén a bevittelt a hiba okára utaló hibaüzenet kíséretében visszautasíthatjuk.

Ha az adatbázisból való lekérdezés eredménye nem néhány sor, akkor azt célszerű megfelelően formázott jelentés (report) alakjában kiíratni. Ez nem csak az egyes eredménysorok szép elrendezését, fejlécet, lapváltást jelent, hanem csoportváltás beillesztését, kiválasztott numerikus adatokra csoport és totál összegek képzését, sőt akár még grafikonok készítését is. Különösen fontosak ezek a *jelentéskészítők (report generátor)* az adattárházakból nyert eredmények interpretálásánál.

Az adatbázis széleskörű informatikai ismeretekkel rendelkező felhasználóinak is egyszerűbbé teszi a munkáját, ha a különböző utasítások kötelező és választható paramétereit nem kell mindig egyenként, betűnként leírva megadni, hanem egy listából, egérrel választhatjuk ki őket. Ilyen módon gyorsabban, gyakorlatilag szintaktikai hibák nélkül tudjuk megfogalmazni utasításainkat. Különösen hasznos a grafikus interfész a lekérdezések összeállításában. Az igazság kedvéért azonban hozzá kell tennünk, hogy komplex, bonyolult lekérdezéseket a grafikus interfészen keresztül nem tudunk megfogalmazni. Ilyenkor vissza kell térnünk az SQL utasítás pontos formátumának elemenkénti megadására.

Az adatbázis-kezelő rendszerek grafikus interfészére mutatunk be példát a 6.1. alfejezetben. Ebben ismertetjük az ACCESS adatbáziskezelő-rendszer képernyő tervezőjét, Report Generátorának használatát, grafikus interfészen keresztül való lekérdezések megfogalmazását és adatbázis objektumok létrehozását.

1.3.4. Szolgáltató programok

Az adatbázis-kezelő rendszerek a rendszeresen előforduló, nagy tömegű adat rutinszerű feldolgozására speciális programokkal rendelkeznek. Ezeket *szolgáltató programoknak* hívjuk.

Leggyakoribb felhasználási területeik:

- Formázott adatok betöltése új adatbázis, vagy új adatbázis objektum létrehozásakor, nagy tömegű adat bevitelkor, kimentett adatok helyreállításakor.
- Adatbázis adatok formázott kimentése biztonsági okokból, vagy más rendszerbe való átvitelhez.
- Adatbázis adatainak újraszervezése (kimentés és adott sorrendben való visszatöltés) az optimális hatékonysági szempontok igénybevételével.
- Fölösleges adatok tömeges törlése.
- Tömeges, nem formázott kimentés biztonsági okokból.
- Meghibásodott adatbázis(rész) helyreállítása biztonsági mentésből.
- Adatbázis tárolókivonat készítése, hogy az adatokat nem a formázott alakban, hanem tényleges, fizikailag tárolt formájukban láthassuk.
- Az optimális elérési utak meghatározására szolgáló statisztikák készítése az adatok aktuális elhelyezkedéséről.

1.4. Adat- és adatbázis-felügyelő

Az információs technológia bevezetése egy intézménynél, vállalatnál maga után vonja a szervezet és a szervezési módszerek megváltoztatását. Az adatbázis az intézmény közös erőforrása, fejlesztését, működését központilag kell biztosítani. Ennek érdekében alakult ki a vállalati hierarchiában az informatikai munkákért felelős magas szinten lévő vezetői pozíció, akinek a munkáját két személy, az *adat-felügyelő* és az *adatbázis-felügyelő* segíti.¹ Nagyobb intézményeknél ezek általában egy-egy erre a célra specializált csoport munkatársai és a felügyelő ennek a csoportnak vezetője. Kisebb helyeken esetleg ugyanaz a személy töltheti be mind a két funkciót.

¹ Az angol szakirodalom a Data Administrator és a Data Base Administrator szavakat használja, aminek a magyar tükörfordítása az "adminisztrátor" lenne. Ehhez a szóhoz azonban a magyar nyelvben jóval kisebb jelentőségű funkció társul, mint az angolban (ott például a kormányzat is "Administration"). Ezért célszerűbb a magyar értelmezéshez közelebb álló "adat-felügyelő" és "adatbázis-felügyelő" kifejezéseket használni.

1.4.1. Adat-felügyelő

Az adat-felügyelő felügyeli az intézmény, vállalat informatikai erőforrásait. Kiválóan kell ismernie a vállalat belső szervezetét, de azért jó számítástechnikai szakember is. Így szót tud érteni az adatbázis-felügyelővel, aki viszont elsőrendű számítástechnikai szakember az adatbázis-technológia területén. De jól ismeri a szervezet feladatait is, hogy megértse és megfelelően meg is tudja valósítani azt, amit partnere, az adat-felügyelő szeretne.

Az adat-felügyelő fő feladatai az intézményen, vállalaton belül:

- az informatikai politika kialakítása,
- az adatbázis megtervezése,
- az adatbázis használatának beillesztése a környezetbe,
- a különböző részlegek ellentétes igényeinek egyeztetése,
- a hozzáférési jogok meghatározása,
- felhasználók kiképzése, továbbképzése,
- az informatikai környezet fejlesztése.

Ezen feladatok végrehajtásakor szorosan együttműködik a felhasználókkal, akiknek a gyakran megvalósíthatatlan és/vagy egymással ellentétes igényeit kell megpróbálnia kielégíteni, és az adatbázis-felügyelővel, aki a gyakorlatban implementálja az általa kidolgozott terveket és feltételeket.

A fent felsorolt feladatok közül a könyv olvasói számára legfontosabb az adatbázis megtervezése. Ezzel részletesen foglalkozunk majd a 3.7. alfejezetben és a 4.7.3. pontban.

1.4.2. Adatbázis-felügyelő

Az adatbázis-felügyelő elsősorban a számítógépes hardverrel és szoftverrel kapcsolatos technikai tevékenységek irányítását és ellenőrzését látja el.

Az adatbázis-kezelő rendszerek állandó fejlődése nem vonja automatikusan maga után az adatbázisokat használók munkájának javulását is. Ha az adatbázist egyszerűen a már meglévő állományok egyesítésével hozzuk létre, vagy hozzácsatolásával bővítjük, akkor a programozók és az interaktív felhasználók ugyanúgy fogják használni a régi, megszokott neveket a duplikált adatokra, mint azelőtt. Így lényegében semmi sem változik azon kívül, hogy a korábbi, egyszerűbb módszerek helyett egy jelentősen bonyolultabb eljárást használunk, melynek mind a hardver-konfigurációja, mind a gépidő-igénye nagyobb. Az adatok száz százalékosan redundancia-mentes tárolása nagymértékben megnövelheti a hozzáférési időt. De ugyanez bekövetkezhet redundáns tárolásnál is, ha a több helyen tárolt adatok közül nem a megfelelőre hivatkozunk. Ugyancsak a hatékonyságot rontja, ha a folyamatos módosítások következtében a fizikai adatszerkezet nem fog megfelelni az adatfeldolgozás logikájának és ugyanazokat az adatokat csak sokkal több lépésben érhetjük el.

A hatékony feldolgozáshoz szükséges feltételek kialakítását, a változtatások konzisztens keresztülvitelét, az adatbázis tartalmának rendszeres biztonsági kimentését nem lehet az egyes felhasználók kényére-kedvére bízni. Ezt a munkát végzi el központilag az adatbázis-felügyelő. Az adatbázis-felügyelő fő feladatai a következők:

- Közreműködés az adatbázis megtervezésében, megszervezésében. Ez magában foglalja az adatmodell kialakítását, az adatbázis szintjeinek leírását, a definiált adatok elnevezését, a hozzáférési jogok meghatározását.
- Az adatbázis fizikai létrehozása. Ebbe beletartozik a fizikai, a koncepcionális és a logikai szintek megvalósítása, a keresési stratégiák megválasztása, a hozzáférési jogok ellenőrzésének megvalósítása.
- Az adatbázis induló adatainak betöltése. Nagy adatmennyiségek hozzáadása.

- Az adatbázisba bevitt adatok integritásának, ellentmondás mentességének ellenőrzésére szolgáló technikai feltételek megteremtése.
- Az adatbázis adatainak rendszeres kimentése, és esetleges hiba után az eredeti állapot visszaállítása.
- Adatbázissal kapcsolatos munkák ütemezése.
- Az adatbázis használatához szükséges szabványok kidolgozása, azok betartatása.
- Az adatbázis használatának állandó figyelése. A hatékonyság jelentős csökkenése esetén az adatbázis újraszervezése.
- A felhasználók igényei alapján az adatbázis szerkezetének a módosítása.
- Az adatbázissal dogozó felhasználók segítése.

Ahhoz, hogy az adatbázis-kezelő rendszer által biztosított lehetőségekből az összes felhasználó fontosság és felhasználási gyakoriság szerint súlyozott igényeit figyelembe vevő leghatékonyabb módszert tudja kiválasztani, az adatbázis-felügyelőnek jól kell ismernie nemcsak az adatbázis-kezelő rendszert, hanem a felhasználói igényeket is. Hiba a rendszert egy olyan feldolgozásra optimalizálni, amelyik havonta egyszer fut le, ha emiatt a naponta ismétlődő alkalmazások lelassulnak. Hasonlóképpen, ha egy valós idejű irányító rendszerben egy válasznak egy adott időn belül meg kell érkeznie, akkor az adatokat úgy kell szervezni, hogy ez mindenképpen megvalósuljon, akár más feladatok feldolgozási hatékonyságának rovására is.

1.5. Az integrált adatbázis előnyei (és néhány hátránya)

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

Az integrált adatbázis olyan egymással kapcsolatban álló adatok összessége, melyeket különféle felhasználók különböző csoportosításban használhatnak. Az adatok fizikai elhelyezése, központilag, redundancia mentesen, vagy minimális, ellenőrzött redundanciával történik. Ugyancsak központilag ellenőrzött az adatok védelme, új adatok bevitele és meglévő adatok módosítása is. Az adatbázis része az adatbázis elemeinek leírását, dokumentálását tartalmazó katalógus. Az adatbázison működik egy adatbázis-kezelő rendszer. Ez lehetővé teszi az arra jogosult felhasználóknak az adatbázishoz való hozzáférést úgy, hogy a felhasználói programok és az adatok tárolási módja egymástól független legyen.

1.5.1. Előnyök

Az integrált adatbázisok legfőbb előnyeit az alábbiakban foglalhatjuk össze:

- Az adatbázist használó intézmény centralizáltan ellenőrizheti a működéséhez szükséges összes adatot.
- Az egységes tervezés következtében az adatok közti logikai ellentmondások jobban kiszűrhetők.
- Az adatok többszörös tárolása, a redundancia megszüntethető. Ezáltal a különböző helyeken használt azonos tartalommal bíró adatok értéke minden alkalmazásban ugyanaz lesz. Nem fordulhat elő, hogy a Humánpolitikai és a Bér Osztály egymásnak ellentmondó személyi adatokkal dolgozhasson. Ugyanannak a személynek a mindkét nyilvántartásban szereplő adatai mindig egyformák.
- Az adatok védelme illetéktelen hozzáféréstől központilag jobban megvalósítható.
- Az adatok központilag szervezett mentése és hiba esetén a helyreállítása jelentősen lecsökkenti az üzemkiesés valószínűségét, illetve időtartamát.
- Az adatbázisba beépíthető feltételek révén nagymértékben csökkenthetjük a hibalehetőségeket és a hibás adatok számát. Olyan adat, amelyik nem felel meg az

adatbázisba beépített feltételeknek, semmiképpen sem kerülhet be. Az ellenőrzést nem kell minden alkalmazásba külön beépíteni.

- Az adatbázisban lévő adatleírásoknak a programokba való kötelező beépítésével biztosítható az adatoknak az egységes felhasználása. Az adatok fizikai tárolásának megváltozása nem befolyásolja a felhasználói programokat.
- A különböző kérdéseket különböző formában lehet megfogalmazni, az egyes adatok közti kapcsolatok jobban értékelhetők.
- Az adatbázis-kezelő rendszer sok olyan beépített segédeszközt tartalmaz, amely megkönnyíti a felhasználói programok megírását.
- A teljes intézmény szempontjából a feldolgozás hatékonyabb lehet.

Azért, hogy az adatbázis fent felsorolt előnyeit kihasználhassuk, létrehozását, karbantartását, az igényekhez illeszkedő időnkénti átszervezését, használatának szabályozását mind a fejlesztői, mind az üzemeltetési környezetben központilag irányítja az adatbázis-felügyelő.

1.5.2. Néhány hátrány

Az eddigiekben kizárólag az adatbázisok és az adatbázis-kezelő rendszerek használatának előnyeit ecseteltük. Nyilvánvaló azonban, hogy erre a területre is igaz az eredetileg közgazdászok által fölfedezett alapigazság, mely eredeti formájában 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. Legföljebb azt mondhatjuk, hogy a gyakorlatban a legtöbb esetben az előnyök kompenzálják a hátrányokat. Ezért végül érdemes az adatbázis használata mellett döntenünk.

Az egységes adatbázis bevezetése és kötelező használata ellen elsősorban az szólhat, hogy nincs olyan szervezet, amelyik képes lenne minden általa használt adatát egy integrált vagy akár egy osztott adatbázisba összehozni. Murphy törvénye alapján, ha létezne olyan adatbázis, amelyik egy (akármilyen kicsi) vállalat összes adatát és azok kapcsolatait tartalmazná, akkor annak kezelése olyan bonyolult lenne, hogy senki sem használná. Nem beszélve még arról a „csekélyiségről”, hogy egy ilyen adatbázisnak a megtervezése, élő adatokkal való feltöltése és naprakészen tartása annyi pénzt, időt és emberi erőforrást igényelne, hogy az a szervezet, amelyik egyetlen egy adatbázisra építené összes tevékenységét rövid időn belül csődbe jutna. (A szerzők kiegészítése Murphy törvényéhez.) Mindig lesznek olyan helyi alkalmazások, melyek saját adatokkal, saját fájlokkal, saját kisebb, önálló adatbázisokkal dolgoznak. Nem véletlen, hogy a legújabb adatbázis típusokban, az adattárházakban az adatbázisok adatainak átvételekor az egyik legfontosabb lépés az adatok helyességének ellenőrzése és egységes formába való átalakítása (ld. 9.3. alfejezet).

Az adatbázis-kezelő rendszerek eleve azzal a céllal készülnek, hogy különféle adatstruktúrákon, sokféle alkalmazásra lehessen őket felhasználni. Így mindenkinek nyújtanak valamit. De éppen általános jellegükből következik, hogy kifejezetten egy feladat megoldására készült, hagyományos fájlokkal, vagy az adott célra optimalizált egyedi adatbázissal dolgozó célrendszerek hatékonyabbak lehetnek egyes feladatok elvégzésére. Ez tovább erősíti a kísértést, hogy az integrált adatbázistól függetlenül dolgozzunk. Különösen igaz ez akkor, ha nagy erőforrás igényű feladatról van szó, mert ilyenkor a látszólagos megtakarítás nagy lehet. (Valójában, amit a réven nyerünk, elveszítjük a vámon. Az adatok konzisztenciájának biztosításakor, a későbbi, okvetlenül elvégzendő módosításokkor fellépő problémák miatt ez az előny csaknem mindig időleges csupán.)

További hátrány, hogy az adatbázis működtetéséhez ahhoz értő, speciális személyzet szükséges. Mivel ez a szakterület rohamosan fejlődik, változik, ezen emberek rendszeres továbbképzéséről is gondoskodni kell.

Ma már kevés olyan hely van, ahol nulláról indul az adatbázis létrehozása. Mégis gyakran előfordul, hogy a korábbi rendszerben ellentmondásmentesnek és hibátlanak bizonyult adatok között az új rendszer szigorúbb követelményei és jobb ellenőrzési lehetőségei inkonzisztenciákat, ellentmondásokat tárnak fel. Ezeket általában egyedileg kell kijavítani. Más adatbázis-kezelő rendszerre való áttéréskor gyakori, hogy a korábbi programokat újra kell írni, vagy legalábbis módosítani kell. Egy új adatbázisra, adatbázis-kezelő rendszerre való átállás mindenképpen hosszabb ideig tart és költséges. Az induló befektetés (hardver, szoftver, emberek) mindenképpen magas.

Mivel (szinte) minden információt az adatbázisból nyerünk, rendkívül nagy gondot kell fordítanunk annak állandó rendelkezésre állására, üzembiztos, megszakítás nélküli működtetésére.

További gondot jelent, hogy a legtöbb cég bizonyos konkrét feladatok megoldására készen vásárolt szoftvert használ. Nem biztos, hogy ez a szoftver közvetlenül együtt tud működni az adatbázis-kezelő rendszerrel. Ilyenkor esetleg egy drágább, rosszabb paraméterekkel rendelkező szoftvert vagyunk kénytelenek választani, vagy el kell készítenünk a két rendszer közti kompatibilitást biztosító interfészt.

Az integrált rendszer néhány további hátrányát, melyek az osztott adatbázisok kialakításához vezettek az 1.6.3. pontban, az osztott adatbázisoknál tárgyaljuk.

1.6. Adatbázis-kezelő rendszerek csoportosítása

Az adatbázis-kezelő rendszereket csoportosíthatjuk aszerint, hogy

- egy vagy több felhasználó dolgozhat-e vele egyidejűleg,
- ugyanazon a gépen van-e az adatbázis és az adatbázis-kezelő rendszer,
- az adatbázist egy vagy több gépen tároljuk.

1.6.1. Egy felhasználós rendszerek

Egy felhasználós adatbázis-kezelő rendszerek ma már kizárólag személyi számítógépeken futnak. Ezek, az eddig ismertett többfelhasználós rendszerektől az alábbiakban különböznek:

- Általában csak egyszerűbb adatszerkezetet tudnak kezelni.
- A relációs modell nyújtotta lehetőségeket csak részben támogatják.
- Adatvédelmük lényegesen gyengébb. Gyakran nincs is önálló, vagy csak nagyon korlátozott védelme van az adatbázisnak magának.
- Biztonsági másolatok készítése, helyreállítás hiba esetén nem vagy csak nagyon korlátozottan automatizált.
- Igen jó grafikus felhasználói interfésszel rendelkeznek.
- Egyidejű hozzáférésekből adódó problémákra – jellegéből kifolyólag – nincs fölkészülve. Ezért az adatbázis többfelhasználós kibővítése gondot okozhat.
- Az adatbázis-kezelő rendszer beszerzésének és működtetésének költségei nem csak abszolút értelemben, hanem relatív, az egész konfiguráció, vagy a teljes szoftver költségeihez képest is sokkal alacsonyabbak. Ennek oka, hogy egy bonyolult, összetett, nagy rendszert kezelő szoftvert esetleg csak néhány száz példányban lehet eladni, míg egy sikeres PC-s adatbázis-kezelő rendszert akár százezres példányszámban is értékesíthetnek.

- Nem, vagy csak korlátozottan kell figyelembe venni más felhasználók informatikai igényeit. Legfőljebb azzal kell törődnünk, hogy milyen adatokat veszünk át másoktól, és milyen adatokat igényelnek tőlünk.
- Igen nagy adatmennyiségek tárolására és feldolgozására nem alkalmasak. (Ez a technikai határ folyamatosan tolódik fölfelé. A mai PC-k tárolási lehetőségei, műveleti sebességük több nagyságrenddel meghaladják a '70-es évek szuper-számítógépeinek hasonló jellemzőit.)

1.6.2. Kliens-szerver architektúra

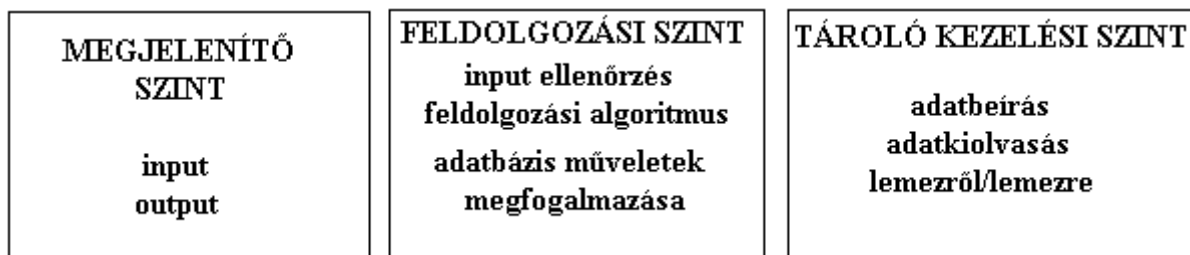
A '80-as évekig az adatbázis és az adatbázis-kezelő rendszer ugyanazon a számítógépen volt. Csupán a felhasználóval való közvetlen kommunikáció került ki terminálokra, munkaállomásokra, majd személyi számítógépekre. Igen sok, régről itt maradt, nagy biztonságot igénylő alkalmazás még ma is így működik. Természetesen ezeknél a központi gép egy igen nagy teljesítményű mainframe.

A nagy rendszerek többségében azonban, főleg az újabban létrejöttekben, a feldolgozás és az adatbázis helye elválik egymástól.

A kliens-szerver architektúrában az adatbázis feldolgozási munkákat az *1.6. ábrán* látható három szintre bontjuk.

A *megjelenítő szinten* keresztül lépünk közvetlen kapcsolatba az adatbázissal. Ezen keresztül írjuk be, illetve olvassuk ki az adatokat. Ehhez hozzátartozik az input adatok formai ellenőrzése és az eredmények formázott megjelenítése is. A hardver általában a rendszer többi részével hálózaton keresztül kommunikáló munkaállomás vagy személyi számítógép. Szokásos elnevezése *kliens gép*, vagy *front-end processzor*. Egy adatbázishoz nagyon sok kliens gép csatlakozhat.

Az adatbázishoz közvetlenül kapcsolódó utasítások, a fizikai műveletek végrehajtása egy önálló gépen, az *adatbázis szerveren*, más elnevezéssel a *back-end processzoron* történik. (Gyakran ezt nevezik röviden szerver gépnek.) Ezen fut az adatbázis-kezelő rendszer és ehhez a géphez kapcsolódik a lemezeken tárolt adatbázis. Ez a *tároló kezelési szint*.



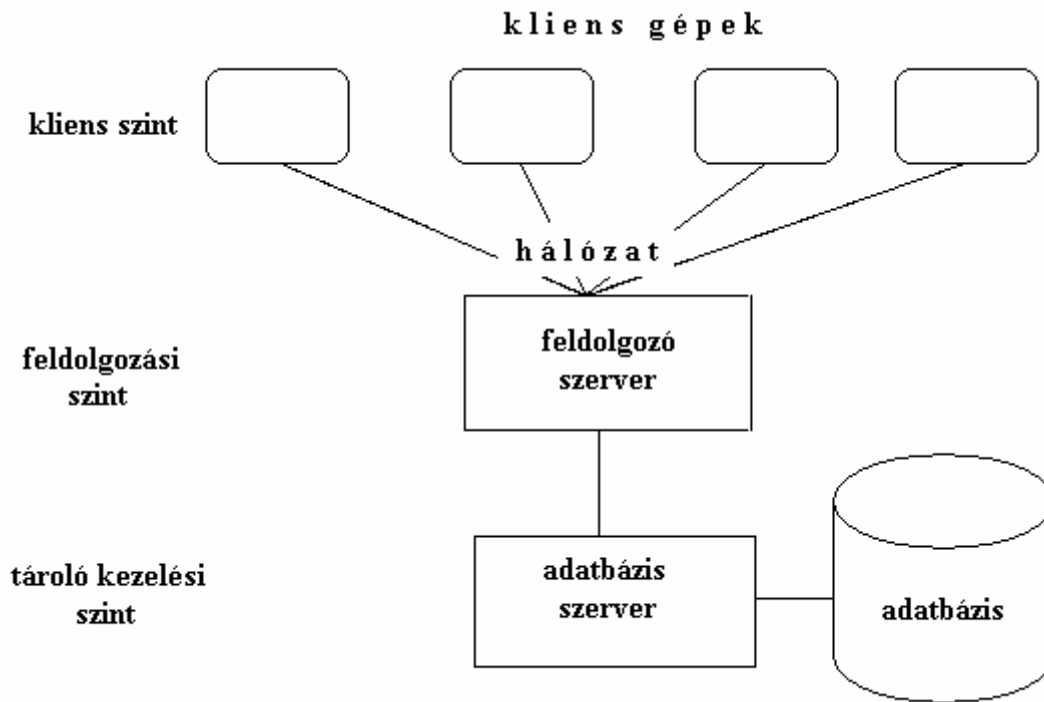
1.6. ábra: **Kliens-szerver feldolgozás szintjei.**

A feldolgozás érdemi része, a felhasználó azonosítása, az input adatok tartalmi ellenőrzése, az alkalmazás logikájából adódó műveletek végrehajtása, és az adatbázissal kapcsolatos műveletek meghatározása a *feldolgozási szinten* történik. Ez megvalósulhat egy önálló gépen, a *feldolgozó szerveren* (Application Server), vagy megosztva a kliens és az adatbázis szerver gép között. A megosztás mértéke a kliens és a szerver gépek teljesítményétől, a hálózat paramétereitől és/vagy a feldolgozás logikájától függ.

Speciális formája a megosztásnak, amikor az adatbázis szerver gépen tároljuk az adatbázis műveletet, de az ott tárolt *eljárást* (stored procedure) a kliens gépről kiadott utasítással indítjuk el. Ez a módszer megnöveli az SQL műveletek hatékonyságát és lecsökkenti az adatforgalmat.

Osztott adatbázisokban (ld. 1.6.3. pont) több feldolgozó és adatbázis szerver van.

A háromszintű kliens-szerver architektúrát az 1.7. ábra mutatja.



1.7. ábra: A háromszintű kliens-szerver architektúra.

A feladatoknak ez a szétosztása lehetővé teszi a kliens gépeknek a felhasználóval való közvetlen kommunikációra, a szerver gépnek pedig az adatkezelő műveletekre optimalizálását. A három különböző szinten futó programokat egymástól függetlenül változtathatjuk, ami nagyfokú rugalmasságot eredményez a feldolgozásban.

1.6.3. Osztott adatbázisok

Az integrált adatbázisok széleskörű elterjedése több problémát is kihozott. Ezek közül néhányat az 1.5.2. pontban ismertettünk. Az ott felsoroltakon kívül komoly gondokat okozott az hogy az integrált adatbázisoknál

- a központ meghibásodásakor minden információszolgáltatás leáll,
- igen nagymértékben megnövekszik az adatátviteli vonalak terhelése,
- az adatoktól távol lévő felhasználók kevésbé érzik „sajátjuknak” a központban tárolt adatokat, kisebb erőfeszítéseket tesznek azok állandó naprakész állapotban való tartására.

Ezen problémák kiküszöbölésére alakították ki az *osztott adatbázisokat*.

Az osztott adatbázis egy olyan logikai adatbázis, amelyik fizikailag különböző helyeken lévő, egymással adatátviteli hálózattal összekapcsolt helyi adatbázisokból épül fel. Mindegyik adatbázis adatai bármelyik adatbázisból elérhetőek, dolgozni lehet velük. A felhasználónak nem kell tudnia, hogy az egyes adatok melyik helyi adatbázisban vannak tárolva. A helyi adatbázisokat nevezzük csomópontoknak is. Ezek a legkülönbözőbb számítógépeken lehetnek és különböző adatbázis-kezelő rendszerek alatt futhatnak. Irányításuk, addig, amíg csak a helyi adatokkal dolgoznak, megegyezik az eddig ismertetett módokkal.

A felhasználó szemszögéből nézve az osztott adatbázisoknak négy fontos jellemzőjét emeljük ki:

- A felhasználó az adatkezelő műveletekkel az osztott adatbázisban ugyanúgy dolgozhat, mint egy integrált adatbázisban. Az adatoknak a különböző helyekről való összeszedését az adatbázis-kezelő szoftver automatikusan elvégzi helyette. Ezt hívják *helyfüggetlenségnek* (local transparency).
- Ha ugyanaz a rekord (adat) több példányban is megtalálható az adatbázisban, akkor a felhasználó számára közömbös, hogy melyik példánnyal dolgozik. Ezt hívják *másolatfüggetlenségnek* (replication transparency). Természetesen az adatbázis-kezelő rendszernek mindig a legkönnyebben elérhető példányt, ha van, akkor a helyi másolatot kell rendelkezésre bocsátania.
- Az osztott adatbázis minden egyes csomópontjában működik egy helyi adatbázis-kezelő rendszer. Ha csupán a helyben tárolt adatokkal kell dolgozni, akkor ez ugyanúgy működik, mint egy centralizált rendszer.
- Mindegyik csomópont azonos értékű abból a szempontból, hogy bármelyik bármelyikkel azonos módon léphet kapcsolatba.

Az osztott adatbázisok létrehozása az alábbi előnyökkel jár:

- Egy csomópont meghibásodása csak a kizárólagosan ott tárolt adatokat teszi elérhetetlenné. A rendszer többi része működőképes marad.
- Jelentősen lecsökken az adatátviteli vonalak terhelése, miáltal a kommunikációs költségek jóval kisebbek lesznek.
- Az adatok a felhasználók „közelében” maradnak. Ezáltal jobban motiváltak azok naprakészen tartásában.
- A teljes adatbázis modulárisan építhető fel. Az egyes csomópontok „menet közben” cserélhetők, jelentősen módosíthatók, a rendszer leállítása nélkül.

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

- A rendszer és adminisztrációja igen bonyolulttá válhat.
- Inhomogén adatbázisoknál adatátvitelkor problémák adódhatnak.
- Egyetlen csomópont sem „zárhat be” következmények nélkül a többi előtt. Ha például a csomópont egy raktár adatbázisa és ott lejárt a dolgozók munkaideje, a helyi adatbázist mégsem lehet bezárni. A többi felhasználónak szüksége lehet olyan adatokra, amelyeket csak ott tárolnak.
- Ha minden adatot csak egy helyen tárolunk, akkor megnövekszik a hibaveszély és az adatforgalom. Ha redundánsan, több helyi adatbázisban is tároljuk, akkor meg kell oldanunk ezek változtatásának szinkronizációját, hogy a gyakorlatilag fontos esetekben mindenhol mindig ugyanaz az adat álljon mindenkinek a rendelkezésére. Ennek megfelelő megoldása az osztott adatbázisok egyik legnagyobb problémája. A gyakorlatban alkalmazott módszerek:
 - Az adatok többszörözése, több helyen való tárolása. Ez a *replikáció*.
 - Az egyedtípus egyes egyedeit különböző helyeken tárolhatják, de egy egyed-előfordulás (egy rekord) minden eleme egy helyen található. Ez a *horizontális adatmegosztás*.
 - Az egyedtípus egyes mezőit különböző helyeken tárolják. A teljes (logikai) rekordot több helyi adatbázisból kell összeállítani. Ez a *vertikális adatmegosztás*.
 - A fentiek valamilyen kombinációja.

Ezekkel az 1.6.3.2. - 1.6.3.5. alpontokban külön is foglalkozunk.

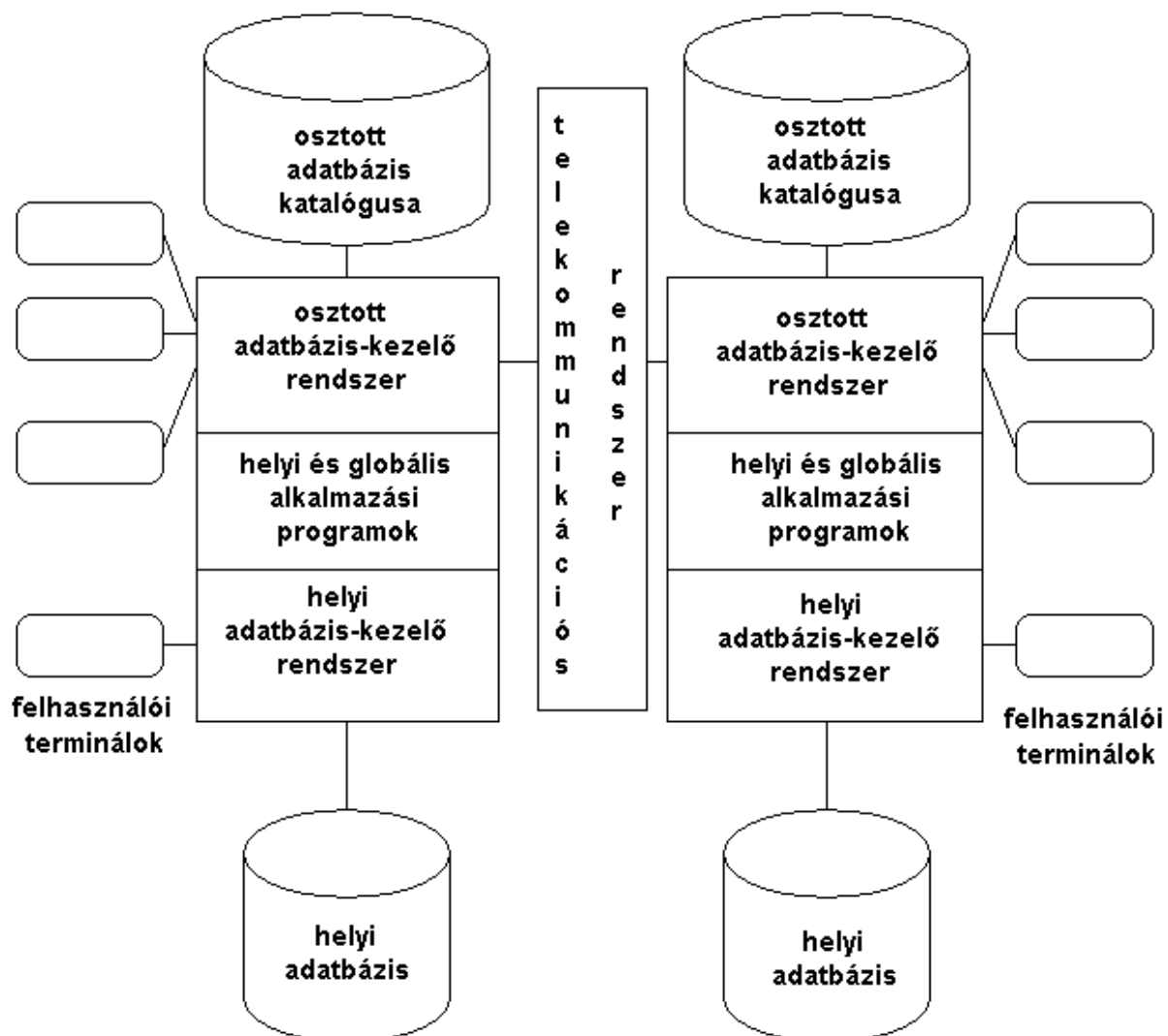
1.6.3.1. Osztott adatbázisok architektúrája

Minden csomópont tartalmazza

- A helyi adatbázis-kezelő rendszert,
- az osztott adatbázis-kezelő rendszer egy helyi példányát,
- a teljes rendszer szerkezetét leíró globális katalógus (ld. 7.2. alfejezet) egy helyi példányát,
- a helyi alkalmazási programokat (melyek dolgozhatnak más helyeken lévő adatokkal is).

Egy jól felépített osztott adatbázisban a hozzáférések zömében a felhasználó kizárólag a helyi adatbázis adataival dolgozik. Ha más adatbázisban lévő adat(ok)hoz kell hozzáférnie, akkor azok fizikai helyét a globális katalógusból (ld. 7.2. alfejezet) keresi ki a rendszer. Az ilyen *távoli hozzáférésnél* (remote access) a helyi gép a távoli adatbázis-gép kliensként viselkedik.

Az osztott adatbázisok leggyakoribb architektúráját az 1.8. ábrán láthatjuk.



1.8. ábra: Osztott adatbázis architektúrája.

1.6.3.2. Adat többszörözés

Osztott adatbázist a legegyszerűbben úgy hozhatnánk létre, hogy minden adatot *többszörözzünk*, mindent mindenhol tárolunk. Még ha a megnövekedett tároló igénnyel meg is békélnénk, akkor is a módosításoknak az összes helyen időben való keresztülvezetése elviselhetetlen terhet róna a rendszerre. Az pedig megengedhetetlen, hogy a különböző helyeken tárolt, azonos jelentést tartalmazó adatoknak helyenként más legyen az értéke. Ezért csak olyan adatokat tárolunk minden, vagy több helyi adatbázis rendszerben

- melyek nélkülözhetetlenek a gyors működéshez. Ezek közé tartozik a globális katalógus, és ide tartozhatnak egyes, az alkalmazási programok által igen gyakran használt adatok.
- melyek kritikusak a rendszer működése szempontjából. Meghibásodásuk, időszakos elérhetetlenségük megbénítaná a rendszert. Ezeket biztonsági okokból több példányban, több helyen tároljuk, hogy valamelyikkel mindig lehessen dolgozni.
- melyek igen ritkán változnak (például menetrendek, árukatalógusok, törzsvásárlók, adattárházak adatai).

A harmadik csoport adatainak konzisztenciája viszonylag egyszerűen biztosítható. Az ilyen adatok csak meghatározott időközökben módosíthatóak. Ilyenkor a régi példányokat minden helyen lecserélik az újra, vagy a módosításokat minden példányon végrehajtják.

Az első két csoportban azonban gondokat okoz az adatok konzisztenciája. Bármely, több példányban tárolt adat bármelyikének megváltoztatása maga után vonja az összes duplikátum megváltoztatását is. Amíg a helyi duplikátumok aktualizálása nem zajlik le, addig ezekhez nem, vagy csak korlátozottan lehet hozzáférni. Erről, valamint ugyanazon rekordoknak, adatoknak több helyről indított módosításainak koordinálásáról az osztott adatbázis-kezelő rendszer gondoskodik különféle zár-mechanismusok segítségével (ld. 8.3. alfejezet). Mivel az elzárási idők a módosításoknak az adatátviteli hálózaton való áthaladási ideje miatt lényegesen hosszabbak, mint a centralizált rendszerekénél, ezért a gyakran változó adatoknál a többszörösen tárolt, módosítható adatok száma nem lehet túl nagy.

1.6.3.3. Horizontális adatmegosztás

Horizontális adatmegosztásnál egy egyedtípus (táblázat) bizonyos rekordjait (sorait) az egyik helyen, más rekordjait egy másik helyen tároljuk. Azt, hogy melyik rekordot hol, melyik adatbázisban tároljuk, a rekord valamely mezőjének (mezőinek) értéke határozza meg.

Példa a horizontális adatmegosztásra egy bank, melynek sok országban (városban) vannak fiókjai. Mindegyik ország/város önálló adatbázissal dolgozik, melyek egy osztott adatbázis részei. Az egyes számlatulajdonosok adatait, megbízásait csak egyszer, a számlavezető bank helyi adatbázisában tároljuk, mivel a tulajdonos feltehetően legtöbbször innen kezdeményezi banki műveleteit. Így ezek ott, helyileg kezelhetők. A számlaszám egyértelműen utal a számlavezető fiókra és azon keresztül az adatbázis helyére. Ezért ennek értéke szabja meg a rekordok tárolási helyét.

Ha a feladathoz jól illeszkedő horizontális adatmegosztást alakítunk ki, akkor az adathozzáférések zöme helyileg lebonyolítható. Ez lényegesen gyorsabb feldolgozást biztosít, mintha távoli adatbázisokban is kellene keresni.

A fenti példánál maradva, ha a bank csak törvénytelen módon szerzett, piszkos pénzekkel foglalkozna, és ezért a nyomon követés megnehezítése érdekében mindenki egy másik országból intézné az ügyeit, akkor ezzel a megosztással semmit sem javítanánk a feldolgozáson, sőt, könnyen lehet, hogy rontanánk rajta.

1.6.3.4. Vertikális adatmegosztás

Vertikális adatmegosztásnál az adat tulajdonsága, a rekord mezőjének a neve (és nem a mező tartalma!) határozza meg, hogy hol tároljuk az értékét. Tiszta vertikális megosztásnál a táblázat minden sorát tároljuk mindenhol, ahol az helyi adatokat tartalmaz.. De a soroknak csak azok az oszlopai, a rekordoknak azok a mezői tárolódnak helyileg, melyekre a helyi feldolgozásban szükség van. A teljes (logikai) rekordot a különböző helyeken tárolt rekord-mezőkből a minden helyen tárolt, a rekordot egyértelműen azonosító elsődleges kulcs segítségével kapcsolhatjuk össze. A teljes rekord összeállítása lényegesen időigényesebb, mint a horizontális megosztásnál.

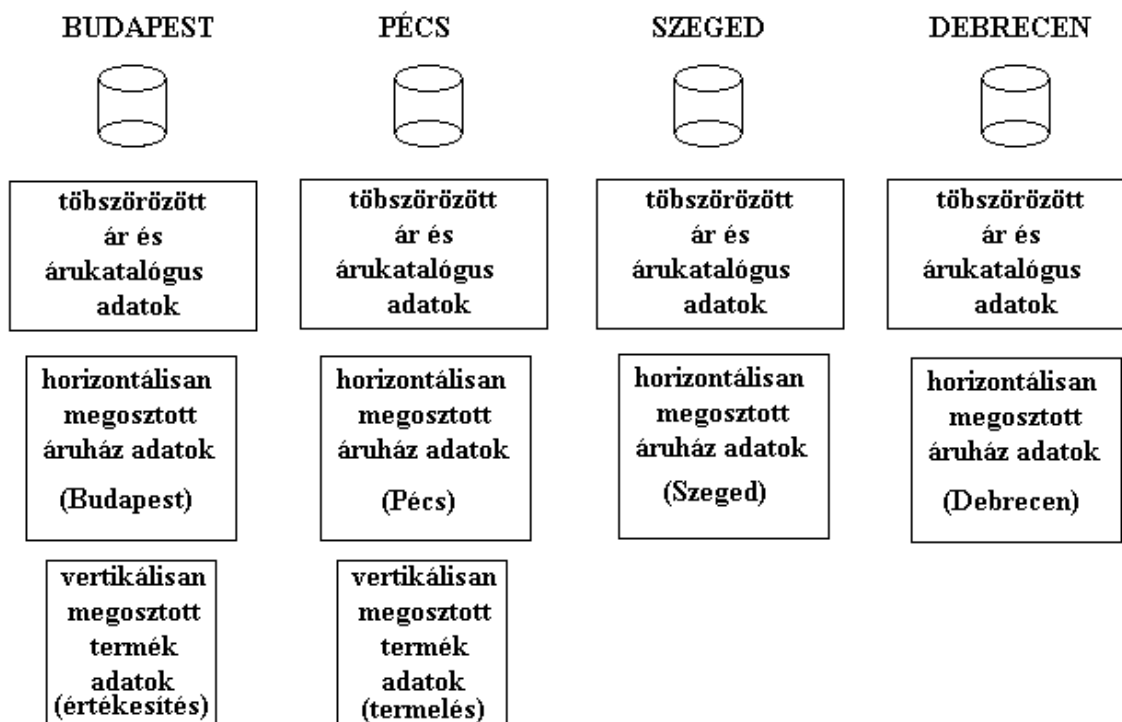
Egyszerű példa a vertikális adatmegosztásra egy olyan gyártó-árusító cég, amelyiknek különböző helyen van a gyártó és az értékesítő részlege. A gyártó helyen lévő adatbázisban tároljuk az egyes termékekre vonatkozó összes gyártási, az értékesítési központban az összes értékesítési adatot. A kettő közti kapcsolatot az egyértelműen azonosító termékszám biztosítja.

1.6.3.5. Kombinált adatmegosztás

Bizonyos esetekben célszerű az előző pontokban ismertetett adatmegosztások kombinációjával dolgozni. Az előző pontban bemutatott cégnek Budapesten van az értékesítési, Pécsen a gyártási központja és ezen két városon kívül még Debrecenben és Szegeden is vannak áruházai. Az osztott adatbázis a négy város helyi adatbázisából épül fel.

Ebben az esetben célszerű lehet az áruházakban tárolt áruknak az eladáshoz szükséges adatait horizontálisan megosztani mind a négy város között. A termékek értékesítésével kapcsolatos ügyviteli adatokat csak Budapesten, a gyártási adatokat csak Pécsen, vertikálisan megosztva tároljuk. Miután mindenhol szükség van rá, és csak ritkán változik, periodikusan aktualizálható az árukatalógus és az árlista, ezeket mindegyik helyen, többszörösen tároljuk.

A kombinált adatmegosztást az 1.9. ábra mutatja.



1.9. ábra: Kombinált adatmegosztás

1.6.3.6. Adatelosztási stratégiák

Az osztott adatbázisok hatékonyságát döntően befolyásolja, miként duplikáljuk, és miként osztjuk szét az adatokat a helyi adatbázisokban. Az első szempont az, hogy az adatokat ott tároljuk, ahol azokat a *leggyakrabban használják*. A második az, hogy tárolhatjuk még *ott is, ahol rendszeresen szükség van rájuk*. Az optimális adatmegosztás érdekében figyelembe kell vennünk:

- Hol használják a legtöbbet az érintett rekordot, adatot. A vizsgálatnak a használat módjára is ki kell terjednie, legfőképpen arra, hogy az adott csomópontban a kérdéses adatot csak olvassák, vagy létrehozzák, törlik, módosítják is. Ha több hely jöhet számításba, és a rekordból, adatból csak egy példány van, akkor azt általában célszerűbb a legintenzívebb változtatási helyen tárolni. (Gondoljunk arra, hogy az *A* helyen lévő raktárból, *B*-ből és *C*-ből árusítunk. Az osztott adatbázis a három helyi adatbázisból épül fel. Első látásra úgy tűnik, hogy mivel a kiértékelésre és az összegzésre az adatoknak végül *A*-ba kell eljutniuk, azokat ott a legcélszerűbb tárolni. Ha azonban elég sok eladás esik egy kiértékelésre, összegzésre, akkor valószínűleg hasznosabb a raktárkészlet adatait *B*-ben és valószínűleg *C*-ben is elhelyezni.
- Az érintett adat mennyire nélkülözhetetlen az adatbázis rendszeres használatához. A nélkülözhetetlen adatokat „nélkülözhetőségük fokától” függően érdemes több példányban, esetleg akár minden helyi adatbázisban tárolni.
- Statikus, vagy csak meghatározott karbantartási időpontokban változó adatok többszörözése gyakran több hasznot hoz az elérési idő megrövidülése révén, mint amennyi hátrányt a megnövekedett tároló igény és a másolatok időszakos felfrissítése jelent.
- A tárolás és az elérés költségei, a hozzáférés gyorsasága. Különösen inhomogén rendszereknél fordulhat elő, hogy egy adat tárolása, elérése más költségekkel jár, hosszabb ideig tart az egyik csomópontban, mint a másikban. A hozzáférési idő növekedését eredményezheti egyes csomópontok túlterhelése is. Ekkor célszerű lehet innen egyes adatokat máshová áttelepíteni vagy duplikálni.

A felsoroltak mindegyikéről elmondhatjuk, hogy működés közben a megvizsgált feltételek megváltozhatnak. Ezért az adatmegosztáson időnként módosítanunk kell. Ez azonban a felhasználói programokat nem érinti. Az egyetlen, amit ebből észrevehetnek az, hogy – ha a feladatoknak megfelelően módosítottuk az adatszerkezetet –, akkor a rendszer gyorsabb lesz.

1.7. Ellenőrző kérdések

1. Milyen összetevőkből épülnek fel az információs rendszerek?
2. Milyen szoftver-rétegeken keresztül lépünk kapcsolatba az adatbázis-kezelő rendszerrel?
3. Milyen szolgáltatásokat kell biztosítani egy korszerű adatbázis-kezelő rendszernek?
4. Milyen szintekből épül fel az adatbázis architektúrája?
5. Mi a különbség az adatleíró (DDL) és az adatkezelő (DML) nyelv között?
6. Mik azok a metaadatok és hol tárolják őket?
7. Milyen grafikus interfészekkel kapcsolódhatunk az adatbázishoz?
8. Milyen feladatokat látnak el a szolgáltató programok?
9. Mi a különbség az adat- és az adatbázis-felügyelő között? Mik a fő feladataik?

10. Mi a kliens-szerver architektúra? Milyen szintjei vannak?
11. Milyen elvek szerint osztják szét az adatokat az osztott adatbázisokban?

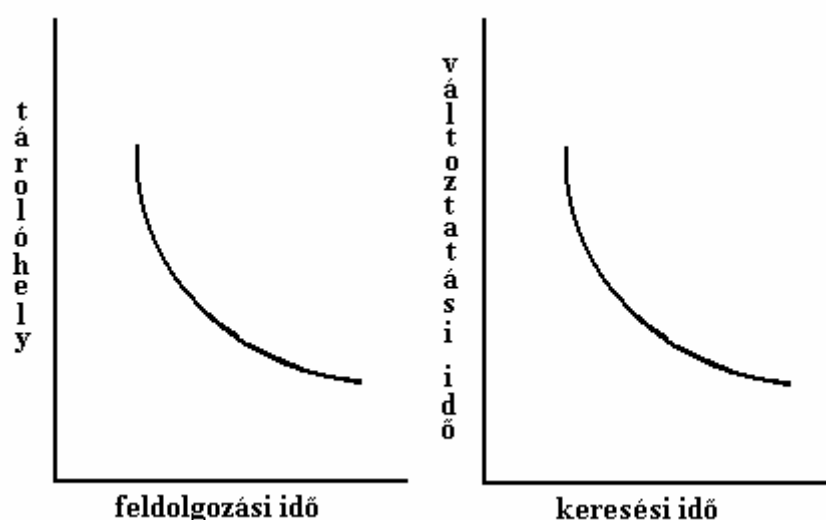
2. ADATTÁROLÁS ÉS ADATSZERVEZÉS

Az adatbázisokkal foglalkozó könyvek ritkán foglalkoznak az adatok fizikai szerkezetével, tényleges tárolási módjával. Ennek az a magyarázata, hogy az adatbázis felhasználói elől ez rejtve marad. A relációs modell, mint olyan, önmagában független annak fizikai realizálásától. Az adatfüggetlenség (ld. 1.2.1. pont) következtében a belső szint jellegzetességei, változásai nem jelentkeznek a felsőbb szinteken, legalább is elméletben. A gyakorlatban azonban a fizikai adatstruktúra és a megvalósítható elérési utak igen nagymértékben befolyásolják a rendszer hatékonyságát. Jól megválasztott tárolási struktúrával jelentősen csökkenthetjük az adatbázis tároló igényét. Jól definiált, a tároló szerkezetéhez és a várható felhasználói műveletekhez igazodó elérési utak létrehozásával pedig akár nagyságrendekkel gyorsabb válaszadási, feldolgozási időket 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 szintjükön fizikai elérési utak, tárolási struktúrák létrehozására, vagy definiálhatnak ilyeneket a koncepcionális szinten.

A belső szint kialakításakor a két fő feladat

- az adatbázis megfelelő szerkezetének a kialakítása,
- a szerkezet ismeretében adott feltételeknek megfelelő rekordok megtalálása, változtatása a lehető legrövidebb időn belül.

Az optimális adatbázis minimális tároló helyet foglal el, és az egyes adatok megtalálásához, változtatásához szükséges időnek is minimálisnak kell lennie. Ezek a követelmények egyidejűleg azonban legfőljebb csak igen egyszerű rendszereknél valósíthatók meg. Általánosan igaz, hogy bármely adatbázis szerkezetnél a tároló hely és a feldolgozási idő fordított arányban áll egymással. Jól megtervezett rendszerben az egyik csökkenése maga után vonja a másik megnövekedését. Ugyanez érvényes a visszakeresési, valamint a változtatási időkre is. Minél hamarabb tudunk egy rekordot megtalálni, annál hosszabb ideig tart egy új rekord beírása, meglévő módosítása, törlése ugyanabban az adatszerkezetben. A fentieket láthatjuk kvalitatívan a 2.1. ábrán.



2.1. ábra: Az adatbázis által elfoglalt tároló hely és a feldolgozási idő, illetve keresési és a változtatási idő közti kapcsolat jellege.

Bár a tárolás fajlagos költsége, 1 Mbájt tárolásának az ára folyamatosan csökken, mégis, a teljes hardver költségeknek 30-40%-át teszik ki az adattárolással kapcsolatos kiadások. Ennek ellenére, mivel több és nagyobb kapacitású tárolókat hozzáilleszthetünk a rendszerhez, de az időkereteket nem bővíthetjük, napjaink adatbázisaiban elsősorban a feldolgozási idő optimalizálására törekszünk.

A fejezet további részében ismertetjük az adatok tárolására szolgáló lemezek legfontosabb jellemzőit. Tárgyaljuk, hogyan tudjuk címezni a tároló különböző helyeit, bemutatjuk a különböző adatszerkezési és elérési módokat, ezek optimalizálását, és példákkal illusztráljuk alkalmazásukat.

2.1. Adattárolók

2.1.1. Tárolási szempontok

Az adatbázis éppen feldolgozás alatt lévő rekordjai mindig a központi memóriában vannak. Ugyancsak itt van a legutóbbi időben használt adatok zöme, és célszerű ide behozni a külső tárolóból a hamarosan feldolgozandó adatokat. Ezek mind ideiglenesen tárolódnak az adatbázis-kezelő rendszer pufferében. Itt minél több olyan adatot kell benntartani, melyek feldolgozásra fognak kerülni, mert innen bármelyik rekordot 10^{-7} - 10^{-9} másodpercen belül el tudunk érni, míg külső tárolónál ez még kedvező esetben is 10^{-3} másodpercet vesz igénybe. Az eltérés 4 - 6 nagyságrend! Ezért az adatbázis működési sebességét nagymértékben felgyorsíthatjuk, ha előre tudunk látni. A feldolgozásra kerülő rekordokat, még mielőtt ténylegesen dolgoznánk velük, behozzuk a pufferba, vagy ha feldolgoztuk, de még újra szükség lehet rájuk, akkor továbbra is benntartjuk őket.

A központi memória nagysága azonban korlátozott. Ezért az adatbázis adatainak zömét a külső tárolókon tartjuk.

- Fizikai felépítésük alapján megkülönböztetünk
 - mágnesszalagos,
 - mágneskazettás,
 - mágneslemezes,
 - optikai lemezes (CD/DVD),
- az elérési lehetőségek szerint
 - soros,
 - közvetlen és soros,
- míg felhasználási jellegük szerint
 - bármikor használható adatok (on-line),
 - az adatbázis helyreállítására szolgáló biztonsági mentések,
 - archivált adatok tárolására szolgáló adathordozókat.

A mágnesszalagokon és -kazettákon tárolt adatokat csak sorosan, felírásuk sorrendjében tudjuk feldolgozni. Ezért ezeket viszonylagos olcsóságuk miatt ma már csak biztonsági mentések és archivált adatok tárolására használjuk. A rendszeres feldolgozáshoz szükséges adatok állandó tárolási helye mindig lemez, ahonnan egy részük a feldolgozás folyamán ideiglenesen bekerül a központi memóriába is.

Bár egyes adatbázis-kezelő rendszerek minden útmutatás nélkül is létrehozzák a koncepcionális modell alapján a tárolási struktúrát, az adatbázis fizikai szintjét, ez a gyakorlatban a legtöbbször nem fog elég hatékonyan működni. Az adathordozók és az adatszerkezet kiválasztásánál figyelembe kell vennünk, hogy

- mekkora adatmennyiséget kell tárolnunk,
- mennyibe kerülhet az adatok tárolása,
- milyen információkra van leggyakrabban szüksége a felhasználóknak,

- milyen szempontok alapján kell az adatokat kiválasztani,
- az adatokat vagy azok bizonyos részeit milyen gyakran kell egy megadott sorrendben elérni,
- ha az adatokat véletlenszerűen, előre nem definiálható sorrendben kell feldolgozni, akkor átlagosan hány rekordot kell megtalálni egy lépésben,
- az adatokat csak olvasni kell, vagy meg kell a tartalmukat is változtatni,
- növekedhet-e módosításkor a már bennlevő rekord hossza,
- hány új adatot kell hozzáadni, régit törölni, és ez milyen gyakorisággal történik,
- vannak-e olyan adatok, amelyeket nagyon gyakran, illetve olyanok, amelyeket csak nagyon ritkán használnak.

E szempontok figyelembevételével az adatbázis-felügyelő dönti el, hogy milyen logikai és fizikai adatszerkezetet hoz létre. Ennek eredményeképpen észszerű kompromisszumot alakít ki a tároló igény, a visszakeresési és a változtatási idők, valamint az adatbázis rugalmas – a változó követelményekhez illeszkedő – használata és továbbfejlesztési lehetőségei között. Olyan egységes adatszerkezet, amelyik minden alkalmazáshoz mindig optimális elérést biztosít, a gyakorlatban nem létezik. Ebből következik, hogy az adatbázis különböző részein más és más adatszerkezetet valósítunk meg, és ezt, ha az alkalmazás módja változik, akkor szükség szerint módosítjuk is.

2.1.2. Lemeztárolók

A mai lemeztárolók vagy mágnesességen alapuló *mágneslemezek*, vagy a fényvisszaverődést felhasználó *optikai lemezek*. Az adatbázis-kezelő rendszer szempontjából azonban teljesen közömbös, hogy az adatokat reprezentáló bitsorozatot a lemezen mágnesezéssel vagy lézerrel állítjuk elő és olvassuk ki. Ugyancsak azonos felhasználói szempontból a kétféle lemezen tárolható adatok szerkezete is. Ezért a továbbiakban egyöntetűen lemezzről fogunk beszélni. A lemezeknek csak azokat a tulajdonságait, jellemzőit fogjuk ismertetni, melyek az adatbázisok hatékony működésének megértéséhez szükségesek. Ebből a szempontból tekintve a következőkben elmondottak minden lemeztípusra érvényesek.

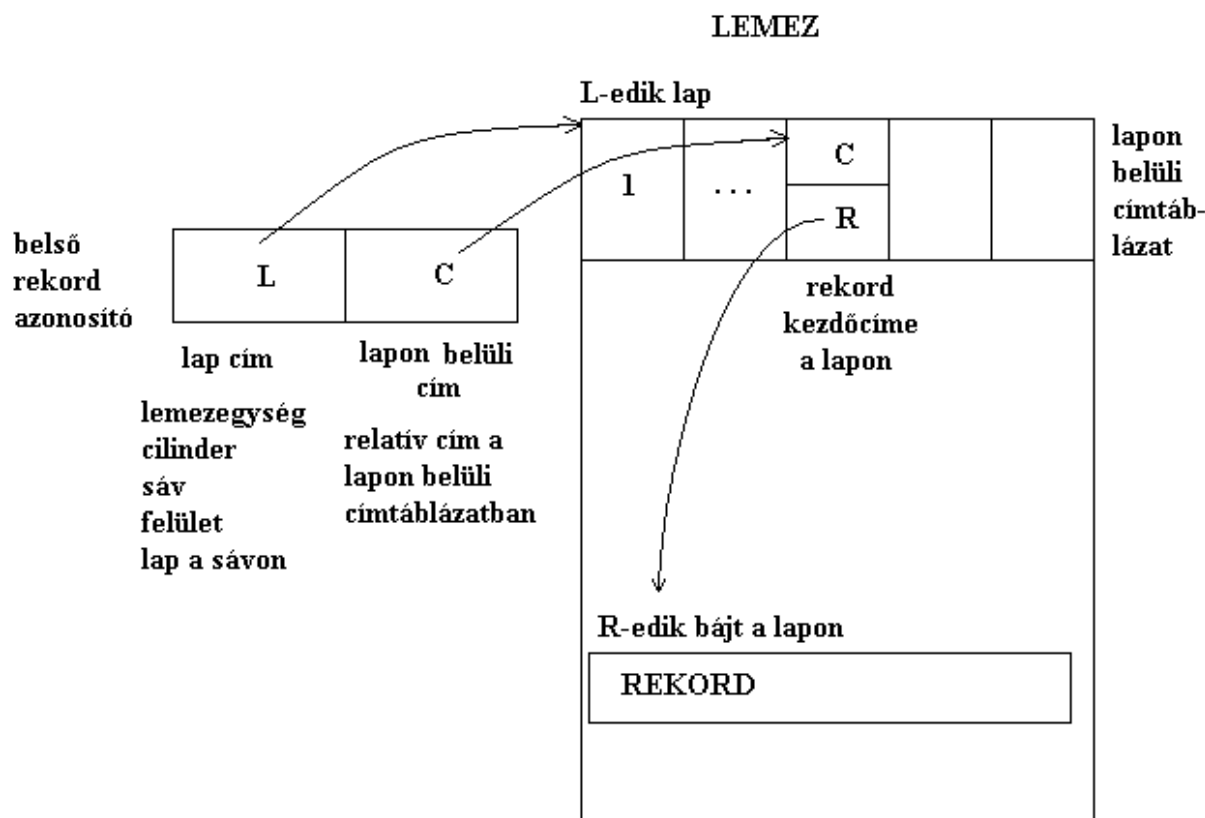
2.1.2.1. A lemez felépítése

A lemez több egymással párhuzamos felületből áll, melyek mindegyikéhez általában egy író-olvasó fej tartozik. (Egyes esetekben csak egy felület van.) Az összes lemezfelület azonos sebességgel forog egy közös tengely körül. Az adatok a felületeken egymás fölötti koncentrikus körökben (optikai lemezen csigavonalban) helyezkednek el. Ezeket a köröket *sávnak* (track) nevezzük. Egy körülfordulás alatt egy felületről egy sáv adatait olvashatjuk be, egy sávot írhatunk ki. A sávokat a forgástengelytől való távolságuk alapján sorszámozzuk. A fej beírásakor/kiolvasáskor minden felületen ugyanazon sorszámú sáv fölött helyezkedik el. A fej egy adott pozíciójából elérhető sávok összességét nevezzük *cilindernek*. A sávokon belül az adatbázis-kezelő rendszer szempontjából az adatok lapokba vannak csoportosítva. Az író/olvasó műveletek mindig egy lapra vonatkoznak. Ez a legkisebb fizikai egység, amit a lemez és a központi tároló között mozgatni tudunk.

Egy adatbázis rekord helyét, tárolási címét meghatározza

- a lemezegység száma,
- ezen belül melyik felületen van,
- melyik sávon van,
- a sáv hányadik lapján található,
- a lapon belül hányadik bájton kezdődik.

Az adatbázis-kezelő rendszer tároláskor minden rekordhoz hozzárendel egy belső rekordazonosítót (ROWID, ld. 5.4.4. pont). Ez rendszerint egy relatív cím. Azt adja meg, hogy a lemezen a rekord az adatok tárolási helyének kezdetétől számítva hányadik lapon található. Mivel a felületek és a sávok száma, valamint a sávok és a lapok kapacitása egy lemezen állandó, ebből a lap helye kiszámítható. A lapon belüli címet a lap elején lévő lapcím táblázat tartalmazza. Ezt a címzési módot mutatjuk be a 2.2. ábrán. A belső rekordazonosító megadja a lap címét (L: lemezegység, cylinder, sáv, lap), és a lapon belüli címtáblázatban annak az elemnek a helyét (C), amelyik a rekordnak a lapon belüli helyére utal (R). Ha a rekord fizikai tárolási helye bármilyen okból (pl. az adatok újraszervezése miatt) megváltozik, akkor a belső azonosító értéke is módosul. Ezért az adatbázis felhasználója számára ez semmiféle azonosítási jelleggel, semmilyen jelentőséggel nem bír, normál körülmények között nem is fogja látni.



2.2. ábra: Fizikai szintű címzés lemezen

A napjainkban leggyakrabban használt mágneslemezek legfontosabb technikai jellemzőinek nagyságrendjét a 2.1. táblázatban foglaltuk össze. (Ebben a személyi számítógépekben még most is használatos floppy diszkeket és a CD lemezeket nem vettük figyelembe.)

Míg korábban a tényleges tárhelykapacitás növelése érdekében igen elterjedtek voltak a cserélhető mágneslemezek, ma a nagy rendszerekben állandó, nem cserélhető lemezeket használnak. Igaz, hogy ez a megoldás drágább és az összkapacitás véges (szemben a cserélhető lemezek gyakorlatilag tetszésszerűen megnövelhető kapacitásával), de a szükséges adatok bármikor azonnal, maximum néhány tized másodpercen belül 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 nagyobb írássűrűséget használhatunk, ugyanarra a felületre több adatot tudunk felvinni.

2.1. táblázat: Mágneslemezek legfontosabb technikai jellemzőinek nagyságrendje.

Felületek száma	20
Cilinderek száma	200 - 400
Sávok kapacitása	40 - 100 KB
Lapméret	1 - 32 KB
Teljes tároló kapacitás	100 MB - 500 GB

2.1.2.2. Hozzáférési idő

Leghamarabb akkor férünk hozzá egy rekordhoz, ha az már bent van az adatbázis-kezelő rendszer pufferében. Ezért a rendszer nyilvántartja a pufferben lévő lapokat. Minden lemezen való keresés előtt megvizsgálja, bent van-e a kért rekordot tartalmazó lap a pufferben, és ha igen, akkor a lemezen való keresésre nincsen szükség.¹ Ezek csak központi tárolón belüli műveletek, így 10^{-6} - 10^{-8} másodperc alatt lezajlik a keresés. Sajnos azonban még a legjobb adatszerkezési eljárásokkal is csak ritkán vagyunk ilyen szerencsések. Legtöbbször a lemezeiről kell behoznunk, vagy oda kell kiírnunk az adatokat, ami 4 - 6 nagyságrenddel hosszabb ideig tart. Az ehhez szükséges időt a továbbiakban *hozzáférési időnek* fogjuk nevezni.

A hozzáférési idő öt tényezőből tevődik össze. Ezek:

- az adatokat tartalmazó lemezegység kiválasztása,
- a lemez felület kiválasztása,
- a megfelelő cilinderre való pozicionálás,
- a rekordot tartalmazó lapnak az író/olvasó fej alá való forgása,
- a lap átvitele a lemez és a központi egység között.

Ezek közül az első kettő elektronikus átkapcsolással valósul meg. Ezért ezek az idők elhanyagolhatóak. A harmadik és a negyedik összetevő – ha a fej nem áll éppen a megfelelő helyen – mindenképpen fizikai mozgással jár. Ezért a *pozicionálási időt* és a *forgási késleltetést* a hozzáférési idő becslésénél mindenképpen tekintetbe kell vennünk. Az utolsó komponens, az *adatátvitel ideje* legtöbbször milliszekundum alatt van. Így, legtöbbször ezt sem kell figyelembe vennünk, mint azt látni fogjuk.

Lemezeire való írás illetve olvasás előtt az író/olvasó fejet az elérendő adatokat tartalmazó cilinderre kell pozicionálni. Ennek ideje attól függ, hogy melyik cilinderen dolgoztunk utoljára, és melyiken fogunk ezután dolgozni. Ha az adatszerkezet jól illeszkedik a feldolgozáshoz, akkor nem kell más cilinderre átállnunk. Ez azonban nem mindig valósul meg. Ilyenkor a pozicionálási idő egy állandó felgyorsulási és lelassulási időből áll, melyhez hozzáadódik még a régi és az új cylinder távolságával arányos elmozdulási idő. Kimutatható, hogy ha véletlenszerűen váltunk a cilinderek között, akkor az átlagos távolság a cilinderek maximális távolságának egy harmada lesz. A kereskedelmi forgalomban lévő fix lemezek az átlagos pozicionálási idő 4 – 10 milliszekundum körül van.

¹ Annak érdekében, hogy nagyobb valószínűséggel legyen bent egy keresett rekord a pufferben, az adatbázisban kiadott írási utasítás csak a pufferben változtatja meg a rekordot. A tényleges fizikai kiírás csak akkor következik be, amikor a puffer betelt, és a következő feldolgozandó lapot csak egy bent lévő lap helyére hozhatjuk be. Ilyenkor sem az összes lapot írjuk ki, hanem felülírunk egy változatlanul maradt lapot (hisz ezt nem kell kiírni a lemezeire, ott is ugyanez a példány van belőle), vagy ha nincs ilyen, akkor például csak a legrégebben használta(ka)t írjuk ki.

A keresett rekord az optimális esetben éppen az író/olvasó fej alatt van. Ilyenkor a forgási késleltetés zérus. Ha az adatszerkezet jól illeszkedik a feldolgozáshoz, akkor a következő feldolgozandó rekord az lesz, amelyik ezután kerül az író/olvasó fej alá. A legrosszabb esetben egy teljes körül fordulást kell várnunk, amíg a rekord a fej alá kerül. Kimutatható, hogy az átlagos forgási késleltetés egy fél körülfordulás idejével egyenlő. Ez a legtöbb fix lemeznél 2 – 8 milliszekundum közé esik.

A lemez és a központi egység közti adatátvitel sebessége 1 - 15 Mbájt/sec körül van. Ezért egy néhány kilobájtos lap átvitele általában 1 milliszekundum alatt van.

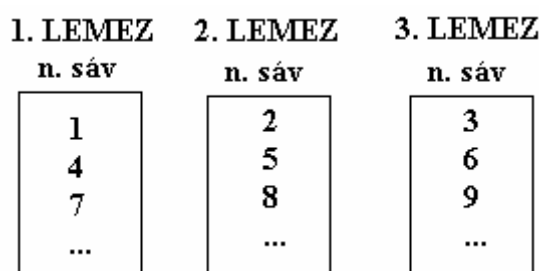
A mai általánosan használt lemezek hozzáférési idejének jellemző összetevőit milliszekundumokban a 2.2. táblázatban foglaltuk össze.

2.2. táblázat: **Általánosan használt lemezek hozzáférési idejének jellemző összetevői milliszekundumokban.**

Pozicionálási idő	Forgási késleltetés	Adatátviteli idő	Átlagos elérési idő
4 - 10	2 - 8	0,1 - 1	6 - 20

Az elmondottakból nyilvánvaló, hogy ha egy feldolgozási lépésben egyszerre több rekordhoz akarunk hozzáférni (pl. az „Adatbázis” tantárgyat felvevő összes hallgató vizsgaeredményéből akarunk átlagot számolni), akkor ezeket a rekordokat célszerű úgy tárolni, hogy ugyanazon a cilinderen egymás után helyezkedjenek el. Hasonlóan, mint ha az Új Magyar Lexikon mind a 19 kötetét időnként használni akarjuk, akkor ezeket mind egy polcra, egymás mellé rakjuk, és nem a betűcikknek vagy a kötet szerkesztők neve szerint rendezett könyvek ABC sorrendjébe. Az adatbázisoknál azonban annyi bonyolultabb a probléma, hogy ott ez az elrendezés előnytelen lehet más feladatok (például tancsoportok szerinti átlagok) kiszámolásánál. Ezekről a kérdésekről a 2.5.1.pontban, a sűrűsödés tárgyalásánál beszélünk majd részletesen.

Ha igen sok fizikailag vagy logikailag egymás után következő rekordot akarunk rendszeresen feldolgozni, akkor ezt jelentősen felgyorsíthatjuk a *RAID technika* (Redundant Array of Inexpensive Disks) alkalmazásával. Ennek egy gyakori alkalmazását a 2.3. ábra segítségével magyarázzuk meg.



2.3. ábra: **Soros/szekvenciális feldolgozás felgyorsítása RAID technika alkalmazásával.**

A rendszeresen egymás után feldolgozandó adatokat több lemezre osztjuk szét (az ábrán háromra), melyeknek forgása szinkronizálva van. Az első lemezre kerül az első, negyedik, hetedik, ... rekord, a másodikra ugyanarra a felületre és sávra a második, ötödik, hetedik, ..., a harmadik lemezre pedig a harmadik, hatodik és a kilencedik rekord. Az egyszerűség kedvéért tételezzük fel, hogy egy sávon 3 rekordot tudunk tárolni. Ebben az esetben, ha az összes rekordot egy lemezen tárolnánk, akkor a 9 rekordot egymás után 3 körülfordulás alatt tudnánk beolvasni, illetve kiírni. Ha a rekordokat a fenti módon szétosztjuk a három szinkronizáltan működő lemezegység között, melyekre/melyekről párhuzamosan

tudunk írni/olvasni, a 9 rekordot egy körülfordulás alatt fel tudjuk dolgozni. N darab lemez alkalmazásával a forgás miatti várakozási időt N -ed részére csökkenthetjük. Beláthatjuk, hogy ugyanez igaz egyes rekordok hozzáférési idejére is.

2.2. Adatszerkezési és adatelérési módok

Az adatbázis-kezelő rendszerek az adatokat a külső tárolókon lapokon tárolják. A lapok mérete rögzített, 1 - 32 KB-ot között van. Az *adatátvitel* a külső tároló és a belső memória puffere(i) között mindig *laponként* történik. *Logikai szinten* az adatbázis-kezelő rendszerek rekordokkal dolgoznak, rekordokat olvasnak be, írnak ki, módosítanak, vagy törölnek. *A rekordok logikailag összetartozó elemi adatokból, mezőkből állnak.* Rekordokat alkothatnak például egy személyi nyilvántartásban az egyes személyek adatai, melyekhez többek között a következő tulajdonságok, *mezők* tartoznak: név, lakhely, beosztás, születési év, vagy egy raktárnyilvántartásban a tárolt áruk jellemzői. Amikor az adatbázisban dolgozunk, akkor mindig a különböző rekordok konkrét előfordulásaival, pl. a „Kovács János, Budapest Lajos utca 49, informatikus, 1980” értékekkel dolgozunk. A mező az adatbázis legkisebb eleme, amelyiknek tartalmára név szerint lehet hivatkozni. A mező tartalma az adott rekordban az adott tulajdonság aktuális értéke. Ez lehet állandó, de meg is változhat a feldolgozás következtében.

A rekord mezői közül legalább egynek kitüntetett szerepe van.¹ Ennek értéke alapján tudjuk az adott rekordtípuson belül egyértelműen azonosítani az egyes rekordokat. Ezt a mezőt/mező kombinációt nevezik *elsődleges kulcsnak* (Primary Key). Ezenkívül léteznek más felhasználást biztosító kulcsmezők is, melyek esetenként megegyezhetnek az elsődleges kulccsal. Az *elhelyezési kulcs* határozza meg a rekord fizikai helyét az adatbázisban, míg a különböző *keresési kulcsok* szerint adott tulajdonságokkal rendelkező rekordokat választhatunk ki az adatbázisból. A mindennapi szóhasználatban a kulcsok között ezt a megkülönböztetést általában elhagyjuk. Egyszerűen kulcsokról beszélünk és ezen rendszerint keresési kulcsot értünk, és csak ha más értelemben használjuk, akkor jelezzük ezt. A különböző kulcsok közti különbségre példa egy gépjármű nyilvántartási állomány, melyben a gépjárműveket rendszámuk alapján azonosítjuk egyértelműen (elsődleges kulcs), az üzembe helyezési időpont szerinti sorrendben helyezük el őket a tárolón (elhelyezési kulcs) és rendszám, üzembe helyezés időpontja és helye, tulajdonos, gyártmány és modell alapján keressük őket (keresési kulcsok).

Az adatszerkezési mód meghatározza, hogyan helyezzük el fizikailag az adatokat, és hogy milyen kiegészítő struktúrákat építünk rájuk, ami különböző hozzáférési módokat tesz lehetővé. Ugyanazon az adatállományon, rekord típuson többféle szervezési módot is megvalósíthatunk. Az elérési mód mutatja meg azt, hogy valamilyen konkrét esetben hogyan férünk hozzá az adatokhoz. Ez az állományon megvalósított szervezési módok egyike lehet csak.

Az adatbázisban az adatok, a rekordok szervezési és elérési módja lehet

- lineáris, ezen belül
 - soros,
 - szekvenciális,
- közvetlen, ezen belül
 - indexelt,
 - hashing,
- hierarchikus,

¹ Bizonyos esetekben ez nem egy mező, hanem több mezőnek a kombinációja, mint például az 1.2.2.1. pontban bevezetett MITHALLGAT rekordnál a "ki hallgatja" és a "mit hallgat" mezők, melyek csak együtt határozzák meg például az osztályzat értékét.

- particionált.

Ezek közül a legfontosabbakat a továbbiakban részletesen ismertetjük, majd összehasonlítjuk, hogy mikor melyiket célszerű használni. Hangsúlyozni kívánjuk, hogy a jó adatbázis-kezelő rendszer az adatokat az optimális módon fogja elérni. De az elérési utat csak azok közül tudja kiválasztani, amelyeket az adatszerkezési mód lehetővé tesz. Hogy milyen adatszerkezési módok valósulnak meg, azt az adatbázis-felügyelő határozza meg.

2.3. Lineáris tárolási struktúrák

Lineáris adatszerkezésnél az adatokhoz kizárólag egymás után férhetünk hozzá. Bármelyik rekord után csak a sorban közvetlenül utána következőt kaphatjuk meg. Ez a sorrend megegyezhet a fizikai tárolási sorrenddel. Ekkor *fizikai soros*, röviden *soros szervezésről és elérésről* beszélünk. Amennyiben a sorrend egy logikai egymásutániságot jelent (pl. ABC sorrend, számoknál nagyság szerinti egymásutániság) akkor logikai soros, más néven *szekvenciális szervezésről és elérésről* beszélünk.

2.3.1. Soros szervezés és elérés

A fizikailag soros tárolásnál, szervezési módnál az adatok, a rekordok helye és keresési kulcsa között semmilyen összefüggés nem áll fenn. A soros elérésnél az adatokat fizikai elhelyezésük alapján, sorban egymás után dolgozzuk fel. Ez a szervezés és elérési mód magából a tárolásból következően nyilvánvalóan minden adatállományon automatikusan megvalósul.

A soros szervezésű adatállományok létrehozása, karbantartása egyszerű. Az új adatot az állomány végéhez kell hozzáírunk. Miután a különböző elemek között nincsen logikai kapcsolat, bármelyiket módosíthatjuk, törölhetjük, mivel ez az adatszerkezetet nem befolyásolja. Kivétel, ha a módosítás megnöveli a rekord hosszát. Ilyenkor, ha szigorúan ragaszkodunk a soros szervezéshez, akkor töröljük a módosítandó rekordot, és új rekordként beírjuk a módosítottat. A törölt rekordok helyén kialakult lyukakat az adatok időnkénti újraszervezésével szüntetjük meg. Ez azt jelenti, hogy az adatokat kimentjük, majd a lyukak kihagyásával folytonosan elhelyezve visszatöltjük az eredeti tároló helyre.

Sok adat beolvasásakor ez az elérés igen gyors, hiszen a lemezen való várakozási idő minimális. Ugyanakkor viszont egy adott tulajdonsággal, meghatározott kulcsértékkel rendelkező rekord megtalálására ez a megoldás igen rossz hatékonyságú. Kedvezőtlen esetben akár az egész állományt végig kell olvasnunk. Egy egyedi kulcsértékkel rendelkező rekord megtalálásához átlagosan a tároló helyek felét végig kell vizsgálnunk. Ha több rekord is rendelkezhet a keresett értékkel és mindegyiket meg kell találnunk, vagy annak megállapítására, hogy az adott kulcsú rekord nincs bent az állományban, mindig végig kell olvasnunk az állomány tárolására szolgáló egész területet.

2.3.2. Szekvenciális szervezés és elérés

Szekvenciális szervezésnél az egymást követő rekordok keresési kulcsa között egyértelmű logikai kapcsolat van. Növekvő sorrendnél mindegyik rekord kulcsa nagyobb (vagy egyenlő, ha duplikátumok is lehetnek) a közvetlenül előtte lévő rekord kulcsánál, csökkenő sorrendnél kisebb, vagy egyenlő. Szekvenciális elérésnél bármely rekord után csak a közvetlenül utána következőt tudjuk elérni (egyes szerkezetekben még a közvetlenül előtte lévő is).

Szekvenciális elérést ugyanazokon az adatokon többféle szempont szerint is létrehozhatunk. Ez teszi lehetővé, hogy az időigényes rendezés (ld. 2.9. alfejezet) nélkül is

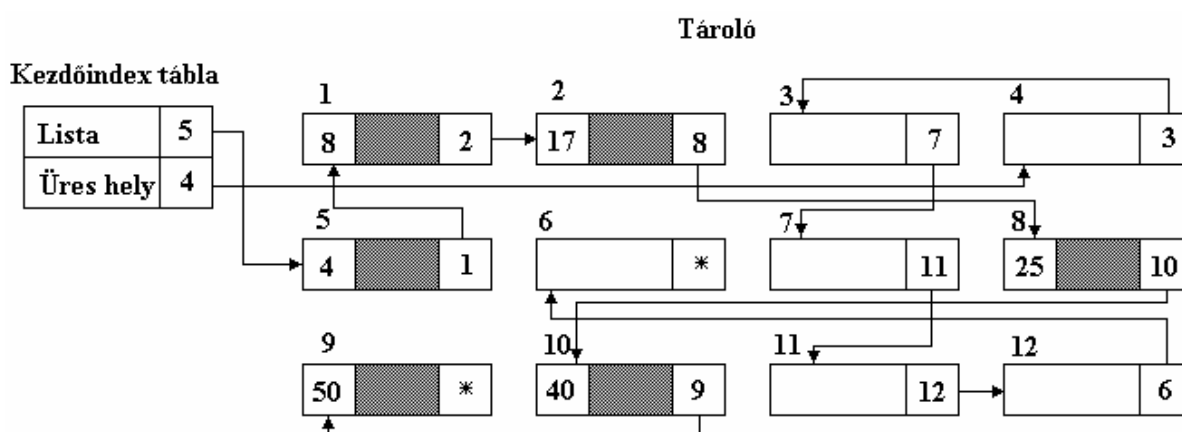
sorba fel tudjuk dolgozni a rekordokat más és más kulcs szerint. (Pl. egyszer névsor, máskor születési időpont, harmadszor fizetés alapján rendezve.)

Ahhoz, hogy egy szekvenciális állományt létrehozzunk, a betöltendő adatokat a keresési kulcs szerint rendezni kell, és ezt a rendezett állományt visszük be az adatbázisba. A legtöbbször azonban már egy meglévő adatállományon kell a szekvenciális szerkezetet létrehozni, vagy meglévő szerkezetbe kell új rekordokat beilleszteni. Ezt a *listaszerkezet* segítségével valósítjuk meg.

A listaszerkezetben a fizikai és a logikai sorrend között semmiféle kapcsolat sem kell legyen.¹ A szekvenciális sorrendben álló, a logikailag egymás után következő rekordokat *pointerek* kapcsolják össze. Minden adatrekord két részből tevődik össze:

- a tényleges adatmezőkből álló adatrészből és
- a pointer mezőből, mely a hozzá logikailag közvetlenül kapcsolódó rekord(ok) címét tartalmazza. Ezt az adatbázis-kezelő rendszer kezeli, a felhasználó számára rejtve van.

A 2.4. ábrán egy egyszerű listaszerkezetet láthatunk. Minden egyes elem (rekord) bal szélső mezője annak a tulajdonságnak az értékét (kulcsot) mutatja, melynek alapján létrehoztuk a szekvenciális szerkezetet. A bevonalkázott területen a rekord további, a lista szerkezet szempontjából közömbös adatai vannak. A jobboldali mező a pointert, a logikailag következő mező címét tartalmazza.²



2.4. ábra: Szekvenciális szervezés megvalósítása listaszerkezet segítségével.

Természetesen ahhoz, hogy egy listát feldolgozhassunk, meg kell találnunk a lista első elemét. Erre a célra egy rögzített helyen, általában a lista legelején, még az adatok előtt elhelyezkedő *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 keresgélni a tárolóban –, hol találunk szabad helyet, ahová ezt elhelyezhetjük. Ezt a gyakorlatban kétféle módon valósítjuk meg.

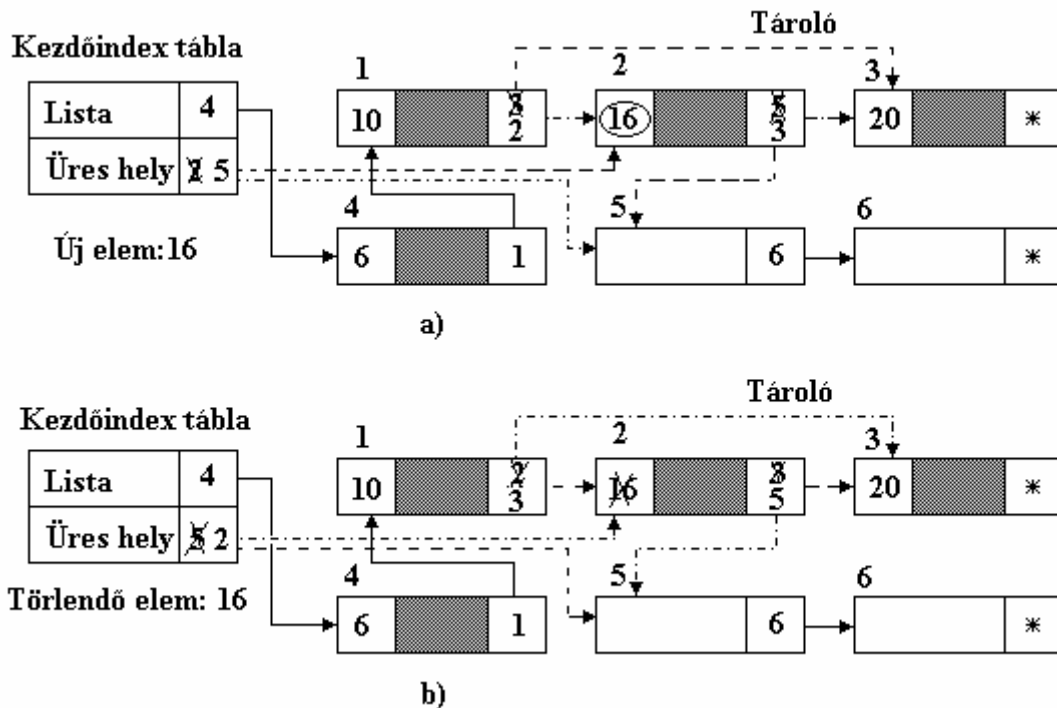
- Létrehozunk egy másik listát, az üres helyek listáját. Ebben vannak összeláncolva a szekvenciálisan szervezett állomány pillanatnyilag üresen álló tároló helyei. A lista első elemének címét szintén a kezdőindex-táblában helyezzük el.

¹ Az elérési idő minimalizálása érdekében azonban célszerű, ha az adatok fizikai elhelyezésének a sorrendje megegyezik, vagy legalább is nem túl sok – a sorból kilógó – rekord kivételével majdnem megegyezik a logikai sorrenddel, mint azt majd a 2.5.2. pontban tárgyaljuk

² Az ábrákon az egyszerűség kedvéért abszolút címeket adtunk meg. A valóságban relatív vagy szimbolikus címeket használnak, melyből az adatbázis-kezelő rendszer számítja ki az abszolút címet. A lista utolsó elemét a pointermezőben egy speciális jel (az ábrákon *) jelzi.

- Az adatokat az állomány elejétől folytonosan töltjük fel. A kezdőindex táblázatban az első üres hely az utolsó adatrekord utáni címre mutat. Az új rekordot ide helyezzük el, és a kezdőcímet egy hellyel tovább toljuk. Ez a módszer egyszerűbb és gyorsabb, mint az előző, de hátránya, hogy ha a meglévő rekordokból törölünk, akkor azok helyét nem tudjuk újra felhasználni. Az állományban „lyukak” jönnek létre, ami sok módosítás után rossz helykihasználást eredményez. Ilyenkor a lyukak eltüntetésére az állományt újra kell szervezni.

A listaszerkezet módosítását, új elem beillesztését, meglévő törlését a 2.5. ábrán mutatjuk be.¹



2.5. ábra: **Listaszerkezet módosítása. A szaggatott vonal a módosítás előtti, a pontozott-szaggatott a módosítás utáni állapotot jelzi.**
a.) Új elem beillesztése. b.) Meglévő törlése.

Ha új elemet kívánunk beilleszteni a listába, akkor ez fizikailag az üres helyek listájának első helyére kerül, és abból a korábbi második hely címe kerül a kezdőcím helyébe. Az új elem logikai helyét a listán való végighaladással állapítjuk meg a kulcsa alapján. Miután ezt meghatároztuk, a logikailag közvetlenül utána lévő rekord pointerére kerül az új elemnek, az új elem címe pedig a logikailag előtte lévőnek a pointermezőjébe.

Hasonlóan, meglévő elem törlésekor a törölt rekord lesz az üres lista első eleme, pointermezőjébe kerül a korábbi első elemnek a címe, és a törölt elem pointerére kerül a logikailag előtte lévő rekord pointermezőjébe.

A gyakorlatban a pointermező nem csak egy pointert tartalmaz. Ugyanarra a kulcsra elhelyezhetünk logikailag előre és visszafelé mutató pointert is. Ezáltal a listán növekvő és csökkenő sorrendben is végighaladhatunk.² Ezenkívül pointer-láncot definiálhatunk más

¹ Bennlévő rekord módosítása, ha az nem a kulcsra vonatkozik nem érinti a listaszerkezetet. A kulcs módosítása a rekord törlését és a módosított kulcsértékkel egy új rekord beillesztését jelenti.

² A két-irányú pointerláncnak törlésnél is jelentősége van. Ha a törölt rekordot nem a listán keresztül értük el, akkor a visszamutató pointer segítségével tudjuk megállapítani, hová kell áthelyezni az előremutató pointerét.

kulcsok alapján való szekvenciális elérésre is. Így például egy személyi nyilvántartásban a személyek rekordjaihoz hozzákapsolhatunk a név szerinti ABC sorrendben, a személyi szám szerinti és a születési időpont szerinti sorrendben való feldolgozást biztosító pointereket. Ezek mindegyike a többiekétől függetlenül használható.

2.4. Közvetlen elérés

Gyakran előfordul, hogy a rekordok közül csak az(oka)t akarjuk kiválasztani, amely(ek)ben egy mező vagy mezőkombináció meghatározott értékkel rendelkezik. Csak azt a hallgatót keressük, akinek az azonosítója '04711', vagy csak azokat, akik a 4. évfolyamra járnak, vagy akik 1986-ban születtek. Ha a kiválasztási feltételnek a rekordoknak csak egy kis része felel meg, akkor még a viszonylag gyors, a fizikai sorrendnek megfelelő soros keresésnek is igen rossz a hatásfoka. Ilyen feladatok hatékony megoldására hozunk létre az adatokon, az adatbázis egyes táblázataiban valamilyen gyors, közvetlen elérést lehetővé tevő adatszerkezetet.

Közvetlen elérésnél a keresett adatokhoz azonnal, gyakorlatilag 1 – 4 lépéssel hozzá tudunk férni. A korszerű adatbázisokban a közvetlen elérést

- *indexek* segítségével, vagy
- *hashing* algoritmussal valósítják meg.

2.4.1. Indexelés

Indexek segítségével igen gyorsan tudunk elérni adott feltételeknek megfelelő rekordokat.

2.4.1.1. Index táblázat

Tekintsük a „klasszikus” adatbázis példát. Vannak szállítóink, áruink és megrendeléseink. Az egyes rekordtípusokat egy-egy táblázatban jelenítjük meg. Ez megfelel a 4. fejezetben ismertetendő relációs modell tárolási szerkezetének is.

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

- SZÁLLÍTÓKÓD. Ez egyértelműen azonosítja az adott megrendelést teljesítő szállítót. A szállító további adatai az ugyanezzel a kódértékkel rendelkező SZÁLLÍTÓ rekordban vannak.
- ÁRUKÓD. Ez egyértelműen azonosítja az adott megrendelésben szereplő árut. Az áru további adatai az ugyanezzel a kódértékkel rendelkező ÁRU rekordban vannak.
- DÁTUM1. A megrendelés dátuma.
- DÁTUM2. A szállítás dátuma.
- MENNYISÉG. A megrendelt és leszállított mennyiség, az árura jellemző (az ÁRUKÓD-hoz tartozó áru rekordban tárolt) mértékegységben.¹

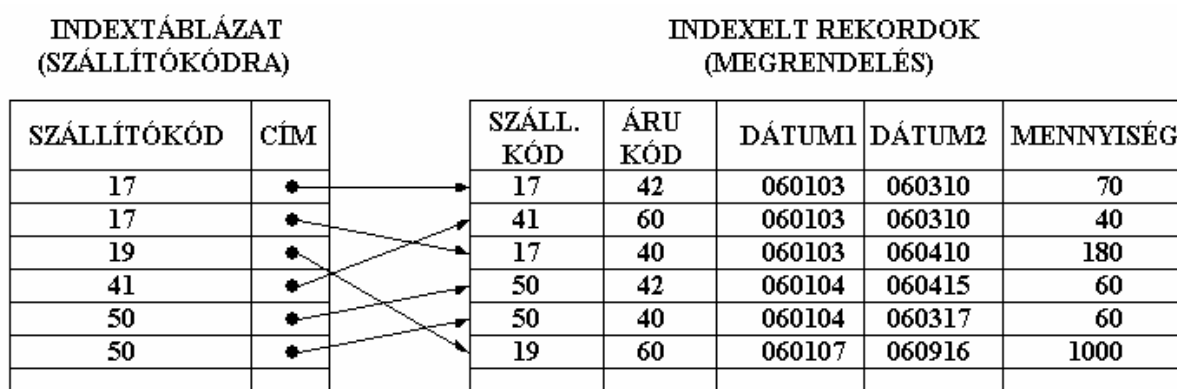
Az adatbázisból rendszeresen választ kell adnunk a megrendelésekkel kapcsolatos következő kérdésekre:

1. Egy adott (kódú) szállítótól milyen megrendeléseink vannak?
2. Egy adott (kódú) áruból milyen megrendeléseink vannak?
3. Egy adott napon milyen megrendeléseket adtunk fel?

¹ Az egyszerűség kedvéért feltételeztük, hogy a megrendelt mennyiséget mindig egyben szállítják le.

4. Egy adott intervallumban milyen megrendeléseket szállítottak le?

Természetesen minden kérdésre tudunk válaszolni, ha végigolvassuk az összes MEGRENDELÉS rekordot, és kiválasztjuk közülük a feltételnek megfelelőket. Ez azonban már néhány ezer rekordnál is igen lassú. Az első kérdés megválaszolására sokkal gyorsabban jutunk célhoz, ha készítünk a megrendelés rekordokhoz egy két mezőből álló táblázatot. Ennek első mezője és egyben keresési kulcsa a SZÁLLÍTÓKÓD, második mezője pedig az ehhez a kódhoz tartozó MEGRENDELÉS rekord címét tartalmazza. Ha egy szállítókódhoz több megrendelés is tartozik, akkor ez a kód annyiszor szerepel a kiegészítő táblázatban, ahányszor a megrendelésekben, és mindegyik címmel egy másik megrendelésre mutat rá – mint az a 2.6. ábrán látható –, vagy csak egyszer, és utána egyenként felsoroljuk az ehhez az értékhez tartozó rekordok címeit.



2.6. ábra: MEGRENDELÉS rekordok indexelése SZÁLLÍTÓKÓD szerint.¹

A SZÁLLÍTÓKÓD értékét, és a hozzátartozó MEGRENDELÉS rekord címét tartalmazó táblázatot *indextáblázatnak* nevezzük, és azt mondjuk, hogy a MEGRENDELÉS rekordtípus (táblázat) *indexelve van* a SZÁLLÍTÓKÓD szerint.

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

Mivel az indextáblázat maga is igen nagy lehet, az ebben való keresés felgyorsítására egy fá szerkezetet építünk fel a kulcsokra. Ezt a 2.4.1.2. pontban ismertetjük. Az indextáblázat méretét lecsökkenthetjük (és a több szempont alapján való keresést egyszerűsíthetjük), ha a rekordcímeket az indexelt értékekkel egy bit-térkép segítségével kapcsoljuk össze. Ezt a módszert a 2.4.1.3. pontban mutatjuk be.

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

A fentieket általánosítva azt mondhatjuk, hogy az indexelt adatok (rekordtípusok, táblázatok) használatához szükséges tároló terület két részre osztható:

- a tényleges adatokat tartalmazó *adatterületre*,
- az indexeket és az indexekben való gyors keresést lehetővé tevő egyéb szerkezeteket tartalmazó *indexterületre*.

Az indextáblázat segítségével, mint azt a következő pontban látni fogjuk, rendezés nélkül is végigmehetünk szekvenciálisan az indexelt rekordokon. Ezért az indexeket csoportosíthatjuk az alábbi módon is:

¹ Valójában az adatbázisban a dátumot nem így tároljuk, de értékét (pl.2006-01-03) megjeleníthetjük az ábrán látható módon

- *Növekvő* vagy *csökkenő*, attól függően, hogy az indextáblán milyen rendezettségben tudunk közvetlenül végighaladni.
- *Elemi* vagy *összetett*, attól függően, hogy az indextáblában lévő indexértékek egy mezőből, vagy több mező összevonásából, konkatenációjából (pl. vezetéknév és keresztnév együtt) származnak. Az eddigi példáinkban elemi indexeket láthattunk. Összetett indexre a 2.6. alfejezetben mutatunk be egy példát.
- *Egyedi* vagy *duplikált*, attól függően, hogy az indexelt táblázatban minden indexérték csak egyszer vagy többször is előfordulhat. A rekordok azonosítására szolgáló elsődleges kulcs mindig egyedi.
- *Szelektív* vagy *nem szelektív*, attól függően, hogy az egyes indexelt értékekhez kevés vagy sok rekord tartozik. Az egyedi indexek értelemszerűen mindig szelektívek.

Indexek segítségével nemcsak adott értékkel rendelkező rekordokat kereshetünk meg, hanem adott értéktartományba esőket is. A 4. kérdésre például úgy válaszolhatunk, hogy a szállítási idő (DÁTUM2) indextáblázatában a kezdő időponttól szekvenciálisan végighaladunk a végső időpontig, és mindegyik értékhez beolvassuk a hozzátartozó megrendelést.

Mint azt már többször hangsúlyoztuk, indexek felépítése és helyes használata rendkívül felgyorsítja az adatbázisban a keresést. Ennek azonban természetesen ára van. Nézzük meg, mivel fizetünk érte.

Először is az indextáblázatok helyet foglalnak el. Minél több mezőre és mezőkombinációra készítünk indexet, annál több tároló helyet igényelnek. Ökölzályként azt mondhatjuk, hogy a gyakorlatban az adatbázis területének jó közelítéssel a 2/3-át az adatok, 1/3-át pedig a közvetlen elérést lehetővé tevő indexek foglalják el. Az adatbázis helyigényének megbecsülésekor ezt mindig figyelembe kell vennünk.

A másik hátrány beírásakor, törléskor és az indexelt mező értékének módosításakor jelentkezik. Ekkor ugyanis nemcsak a rekordon kell az adott műveletet elvégezni. Az összes érintett indextáblát is aktualizálni kell.

Ha például a megrendelésekben (ld. 2.6. ábra) minden szállítási dátumot 060310-ről 06313-ra módosítunk, akkor az adatbázis-kezelő rendszer

- feljegyzi a módosított rekord(ok) címét,
- megkeresi a DÁTUM2 indextáblázatból a 060310 értékhez tartozó sorokat,
- törli az indextáblázatból ezeket a sorokat,
- beírja az indextáblázat megfelelő helyére a 060313 kulcsértéket a megváltoztatott rekord(ok) címével együtt.

Ezek a műveletek egy-egy módosítás idejét 3-5-szörösére is megnövelhetik. De ez még mindig megéri a rendszerint sokkal gyakrabban elvégzett keresésnek a 10 -100 –szoros lerövidülése miatt.

2.4.1.2. Szelektív indexek, a B^+ fa

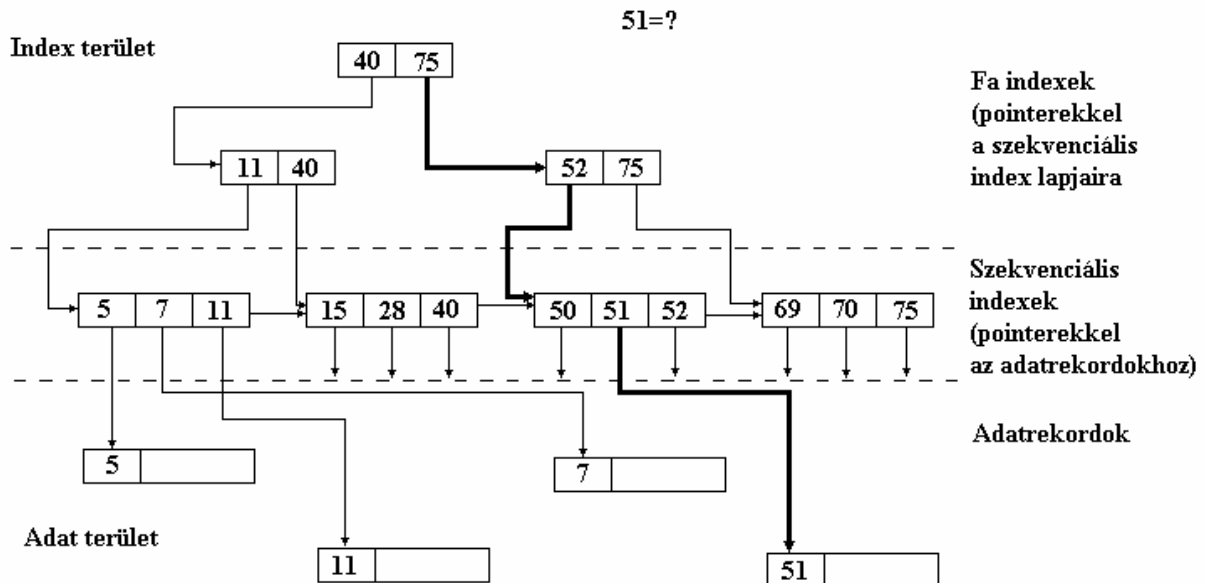
Egyedi és szelektív indexek esetén a keresett rekordokat a B^+ fa elvén felépített indexek segítségével érjük el. Ez lehetővé teszi a rekordok közvetlen és szekvenciális elérését is akárhány különböző keresési kulcs szerint.

A B^+ fa szerkezetnél az indexterület két részre oszlik:

- *Szekvenciális indexek.* Ez az egyes rekordok kulcsait tartalmazza az indexelt mező szerinti szekvenciális sorrendben és az egyes kulcsértékekhez tartozó rekordok címeit. Szekvenciális elérésnél ezeken, illetve az innen kapott rekordcímeken haladunk végig. A szekvenciális indexeket tartalmazó lapokat láncolt lista kapcsolja össze.

- *Fa indexek.* Ezek a gyors, közvetlen hozzáférést teszik lehetővé. Ez a szekvenciális indexekre felépülő kiegyensúlyozott, gyökerével felfelé fordított faszerkezet, melynek kezdőpontjától (a fa gyökerétől) ugyanannyi lépésben érjük el bármelyik szekvenciális index-rekordot, a fa leveleit (ld. 5. feladatot).

A B^+ fa működését a 2.7. ábra alapján érthetjük meg. Az egyszerűség kedvéért a rekordtípusnak csak egy mezőjére vizsgáljuk meg egyedi indexek növekvő sorrendben való felépítését. Duplikált indexek esetén vagy egyszer tüntetjük fel az index értékét, és utána ismétlődő elemként annyi címet sorolunk fel, vagy annyszor ismételjük az indexérték – rekordcím párost, ahány rekord rendelkezik az adott indexértékkel. Különböző mezők alapján való indexelés esetén, minden indexelt mezőre felépítünk egy önálló B^+ fát.



2.7. ábra: B^+ fa szerkezete

A fa legfelső szintjének (gyöker) megfelelő lapon n_1 indexbejegyzés, indexérték és pointer van. (A példában kettő, a 40 és a 75 indexértékekkel. Az ezekhez tartozó pointert a nyíllal jelezzük). A pointerok a fa következő szintjén lévő indexlapokra mutatnak, melyekben további címek és pointerok mutatnak a „megfordított” fa egyre mélyebb szintjének lapjaira, míg végül el nem érünk a szekvenciális indexeket tartalmazó lapokhoz. Az egyes faindex szinteken rendre $n_2, n_3, \dots$, a szekvenciális indexlapokon pedig n bejegyzést helyezhetünk el. (Általában $n=n_1=n_2=n_3 \dots$ a példában az áttekinthetőség kedvéért $n_1=n_2=2$, $n=3$, de a gyakorlatban mindkettő nagyságrendben 100 körül van). A faszerkezet bejegyzéseiben a cím mindig a faszerkezet eggyel mélyebb szintjére mutat, az indexérték pedig megadja az ezen a lapon található legmagasabb indexértéket.

A példában a közvetlenül a szekvenciális (i -ik) szint fölötti $i-1$ -ik fa szinten (az ábrán a második) két lap van. Itt az első lapon két indexbejegyzést találunk. Ezekben a kulcs értéke 11 illetve 40, mert a közvetlenül alatta levő első szekvenciális indexlapon a legnagyobb indexérték 11, a másodikon 40. Hasonló módon kerül az $i-1$ -ik (második) szint második lapjára az 52 és a 75 indexérték. Miután az $i-1$ -ik szinten is több lap van, az ebben való keresésre felépítjük efölé az $i-2$ -ik szintet (az ábrán ez már a legfelső, az első szint, amit gyökernek is nevezünk). Az ebben lévő indexértékek rendre az alatta levő szint lapjainak legmagasabb indexértékeire utalnak (40 az első, 75 a második lapra). Ha ebben megint csak több lapra férnének el az index bejegyzések, akkor erre egy újabb szintet építenénk fel, és ezt folytatnánk addig, amíg el nem jutunk odáig, hogy az összes közvetlenül a szint alatti indexlapokra mutató bejegyzés el nem fér egy lapra, a gyökerre (root). Példánkban a fa három

szintes. Ebben az esetben bármely rekordot 4 lépésben, két fa-index szint + szekvenciális indexlap + rekord elérünk. Általában a lépések száma $i+1$, ahol i a fa szintjeinek a száma.

A fa-szerkezet felépítése mindig alulról felfelé történik. Először a szekvenciális index épül fel. Erre jönnek a fölötté lévő fa-szint lapjai, ha ebből több van, akkor az efölötti szint lapjai, és így tovább, amíg csak el nem érjük az egy lapból álló gyökeret.

Rekord(ok) megkeresését B^+ fa index segítségével a 2.7. ábrán a következő példával mutatjuk be. Keressük az 51 kulcsú rekordot. Ez az alábbi módon történik:

1. A legfelső szint index-bejegyzéseit megvizsgálva $40 < 51 < 75$. Ezért a következő lépésben a 75 indexértékhez tartozó címen található indexlapot olvassuk be.
2. Itt $51 < 52$, a keresett kulcs értéke már az első indexértéknél kisebb. Ezért az 52-es értékhez tartozó címen lévő lapot olvassuk be.
3. Ezzel eljutottunk a szekvenciális indexekhez. A kijelölt indexlapot végigolvasva megtaláljuk az 51-es kulcsú rekord címét. Onnan behozzuk a rekordot. Amennyiben az 51-es kulcshoz nincs bejegyzés ezen a lapon, akkor az adott rekordtípusból ilyen kulcsú rekord nincsen az adatbázisban.

Némileg bonyolultabb a helyzet, ha duplikátumok is lehetnek, és az azonos kulcsértékkel rendelkező rekordok címei csak több lapon férnek el. Ilyenkor az összes olyan indexlapon végig kell mennünk, mely a keresett kulcshoz tartozó címet tartalmaz.

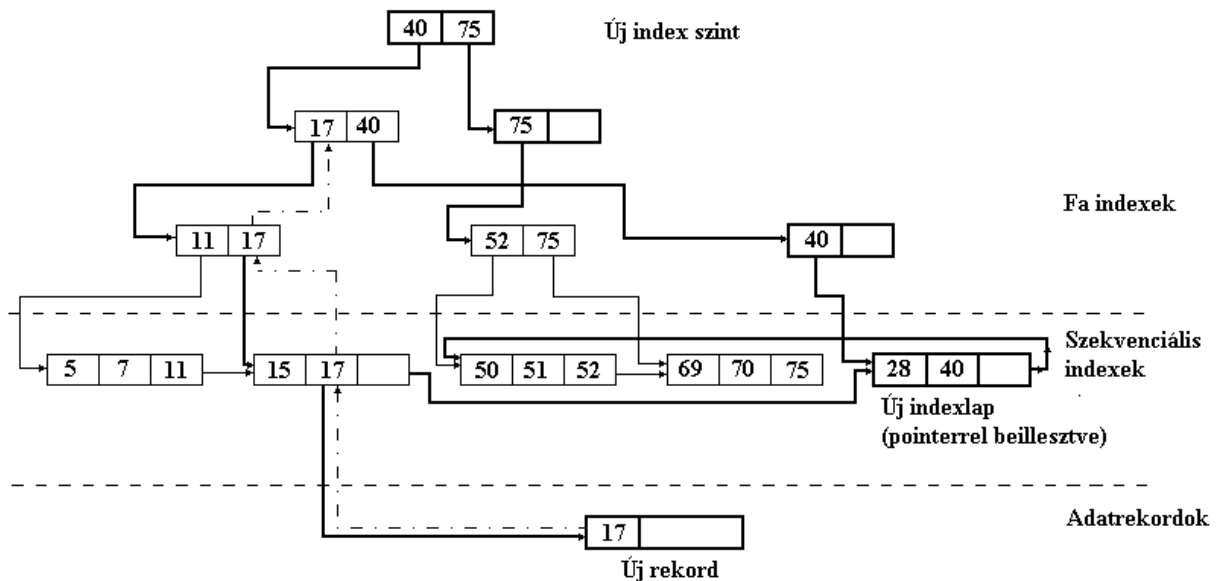
A 2.7. ábrán látható tárolási struktúra nagyon áttekinthető, de erősen idealizált. A gyakorlatban szinte sohasem tudunk egy fát ilyen szimmetrikusan felépíteni és egyenletesen felölteni adatokkal. A példánkban is azonnal bajba kerülünk, ha be akarjuk illeszteni a 17-es kulcsú rekordot úgy, hogy a fa-szerkezetet is megtartsuk.

Természetesen azért egy ilyen feladat megoldása nem okoz igazi gondot. A gyakorlatban egy új rekord beillesztése a következő módon történik:

1. Elhelyezzük a rekordot az adatterületen. Ha van sűrűsödő kulcs (ld. 2.5.1. pont), akkor lehetőleg az azáltal meghatározott lapra.
2. Legtöbbször az index-terület lapjai nincsenek teljesen tele. A fa-szerkezeten végighaladva megkeressük, melyik szekvenciális indexlapra kerül az új kulcshoz tartozó bejegyzés. Ha ott van elegendő üres hely, akkor a címével együtt besúrjuk oda a kulcsok sorrendjének megfelelő helyre.
Ezzel a B^+ fát aktualizáltuk. (Ha a besúrás a szekvenciális indexlap végére kerül, akkor a felette lévő szintet is aktualizálni kell.)
3. Ha a megfelelő szekvenciális indexlapon nincs már üres hely, akkor annak második felét átírjuk egy üres lapra.
4. Az új lapot bekötjük a szekvenciális lapokat összekötő lista-szerkezetbe.
5. Elhelyezzük az új rekord indexbejegyzését a kulcsának megfelelő indexlapra, mivel ott most már biztos van számára elegendő hely.
6. Módosítjuk a faindex közvetlenül fölötté lévő szintjét úgy, hogy a megfelelő lapon a kettéosztott lapra mutató indexbejegyzésben a címet módosítjuk a most besúrt új indexlap címére, és beteszünk egy új bejegyzést is, amelyik a kettéosztott lap első felében maradt legmagasabb kulcsú rekordra utal.
Amennyiben a lapon volt erre helyünk, akkor a B^+ fa aktualizálását befejeztük.
7. Ha a faindex megfelelő lapjába nem tudtuk beilleszteni az új bejegyzést, akkor azt a 3. és 5. lépésekben leírt módon kettéosztjuk és módosítjuk. Ezután módosítjuk a 6. lépésben leírtak szerint a faindexek fölötté lévő szintjét.

8. Ezt az eljárást folytatjuk egészen addig, amíg valamelyik szinten üres helyet nem találunk. Ha már a legfelső szint is tele volt, és a beszúrással ketté kellett osztani, akkor egy új szintet hozunk létre. A fa szintjeinek számát eggyel megnöveljük.

A fentieket követhetjük nyomon a 2.8. ábrán. A 2.7. ábrán látható rekordokhoz illesztjük be a 17-es kulcsú rekordot. Ez az adott elrendezésnél a B^+ fa minden szintjének átrendezését és egy új szint kialakítását is igényli. Ezután viszont például a 16-os kulcsú rekordot egyszerűen, a fa-szerkezet változatlanul hagyásával beilleszthetjük. A második szekvenciális indexlapra beírjuk a 15 és a 17-es bejegyzések közé a hozzá tartozó index-bejegyzést.



2.8. ábra: Új rekord beillesztése és a B^+ fa aktualizálása.
Az új lapokat és kapcsolatokat vastagítva rajzoltuk be.

Összefoglalva, a B^+ fa előnyei a következők:

- Igen gyors, bármely rekordhoz ugyanannyi (maximum 4 -5) lépésből álló elérést tesz lehetővé. Ez a gyakorlatban sokszor csak egy vagy két ténylegesen végrehajtandó lemez műveletre redukálódik, mivel a fa-szerkezet felső szintjei folyamatos működés közben rendszerint bent vannak a rendszer pufferben.
- Ugyanazon rekordtípusban akárhány szempont szerint megvalósítható a keresés.
- Jó az indexterület hely kihasználása. Az indexlapok kitöltöttsége elvileg legalább 50%, de a gyakorlatban 75% körül mozog.

2.4.1.3. Nem szelektív indexek, bit-térképes index

Nem szelektív indexeknél a B^+ fánál jóval hatékonyabb elérést biztosítanak a *bit-térképes* (bitmap) indexek. Ilyenkor egy bit-térképen alapuló index 3 - 4-szer kevesebb helyet foglal el, mint egy B^+ fa, és akár egy nagyságrenddel gyorsabb hozzáférést is lehetővé tesz. Egyedüli hátránya, hogy az indexelt rekordok párhuzamos változtatásakor az indexek aktualizálása miatti várakozási idők lényegesen hosszabbak lehetnek.

A bit-térképes index egy táblázat, melynek első oszlopa az indexelt mezők lehetséges értékeit, fejléce az indexelt rekordok címét (elsődleges kulcs, belső rekordazonosító) tartalmazza. Ha az adott rekordban (a bit-térkép egy oszlopa) az indexelt mező értéke megegyezik a sort jellemző értékkel, akkor ott a táblázat-elem értéke 1, különben 0. Egy ilyen

bit térképet láthatunk az 1.2.2.1. pontban ismertetett HALLGATO táblázat ÉVFOLYAM mezőjére felépítve a 2.9. ábrán.

Rekord azonosító	1	5	3	4	
Indexelt érték (ÉVFOLYAM)	1	2	3	3	
	1	1	0	3	...
	2	2	3	0	
1	0	0	0	1	
2	1	1	0	0	
3	0	0	0	0	...
4	0	0	1	0	
5	0	0	0	0	

2.9. ábra: Index megvalósítása bit-térkép segítségével.

A negyedik sor harmadik oszlopában lévő 1 azt jelenti, hogy a 3. oszlopot jelentő 3303 azonosítójú rekordban az ÉVFOLYAM mező értéke 4. Ha meg akarjuk kapni az összes 2. éves hallgató rekordját, akkor a bit-térképből a 2. évfolyamnak megfelelő sorból kiválasztjuk az 1 értékekhez tartozó fejléc mező értékét. (Az ábrán az első és a második oszlop, az 1112 és az 5212 azonosítójú rekordok).

A bit-térképben való gyors keresés érdekében a térképet magát is felépíthetjük egy B⁺ fával indexelt állományként. A keresési kulcs a bit-térképpel indexelt mező, az adatrekord pedig az ehhez az értékhez tartozó sor. Ezt kiegészíti a fix hosszúságú elemekből álló egy soros rekord-cím táblázat, amelynek *n*-edik eleme a bit-térkép *n*-edik oszlopához tartozó rekordnak a címét tartalmazza.

További előny még, hogy a bináris táblázatokon a műveletek igen gyorsan elvégezhetőek, és maga a térkép viszonylag kicsi. Még 1 millió indexelt rekord 100 különböző indexértéke mellett is csak kb. 13 MB-ot foglal el. Ezért rendszerint az egész térképet állandóan benntarthatjuk a központi tárban, ami még gyorsabbá teszi a feltételeknek megfelelő rekordoknak az index alapján való elérését.

Bit-térképes indexekkel a több kulcs alapján való keresés is igen egyszerű, mivel egy indexelt mezőre minden oszlopban (ha az összes lehetséges értéket figyelembe vettük, akkor pontosan, különben legfőlőbb) csak egy darab 1 állhat. A különböző keresési mezők bit-térképeinek sorait egymás alá írva VAGY kapcsolat esetén az egyes értékeknek megfelelő sorok oszlopainak logikai összeadásával megkapjuk a feltételnek megfelelő rekordokat jelentő oszlopokat. ÉS feltételnél matematikai összeadás után azok az oszlopok felelnek meg, ahol az összeg egyenlő a feltételek számával. Ezt láthatjuk a 2.10. ábrán. Az ÉVFOLYAM = 2 VAGY SZÜLETÉSIÉV = 1998 feltételnek az 1112 és az 5212 rekordok, míg az ÉVFOLYAM = 2 ÉS SZÜLETÉSIÉV = 1998 feltételnek csak az 1112 rekord felel meg.

Indexelt értékek	Rekord azonosító	1	5	3	4	...
		1	2	3	3	
		1	1	0	3	
		2	2	3	0	
(ÉVFOLYAM)	1	0	0	0	1	...
	2	①	①	0	0	
	3	0	0	0	0	
	4	0	0	1	0	
	5	0	0	0	0	
(SZÜLETÉSI ÉV)	1989	0	0	1	0	...
	1988	①	0	0	0	
	1987	0	0	0	1	
	1986	0	0	1	0	
Logikai összeadás		1	1	0	0	...
Matematikai összeadás		2	1	0	0	

2.10. ábra: Két feltétel összekapcsolása bit-térkép segítségével.

Az ÉVFOLYAM = 2 VAGY SZÜLETÉSIÉV = 1998 feltételnek az 1112 és az 5212 rekordok, míg az ÉVFOLYAM = 2 ÉS SZÜLETÉSIÉV = 1998 feltételnek csak az 1112 rekord felel meg.

2.4.1.4. Join-index

Bár az SQL nyelv közvetlenül nem támogatja, igen hasznos lehet, ha gyakran kell több nagy táblázatból összeszedni adatokat, akkor azok egyesítésére (ld. 4.5.6. pont) egy önálló *join-indexet* létrehozni. Különösen célszerű ez akkor, ha az egyesítésre szolgáló oszlop(ok) egyik táblázatban sem elsődleges kulcs(ok). A fentiek megértésére vizsgáljuk meg a 2.11. ábrán látható példát.

Egy cégnek különböző városokban vannak lerakatai és vevői. A szállítási költségek minimalizálása érdekében minden vevőnek a saját városában levő lerakattól kívánnak szállítani. Hogy ezt hatékonyan tudják meghatározni az adatbázisból, mind a vevők, mind a lerakatok rekordjaira felépítettek egy indexet a város mezőre. A két rekordtípust tartalmazó táblázatoknak a város egyezése alapján történő összekapcsolásához azonban ennek minden végrehajtásához (join) össze kell fésülni a két indexet. Ebből tudjuk megállapítani, melyek azok a rekordcímek, amelyek mindkét táblázatban ugyanahhoz a városhoz tartoznak.¹ Ez nagy adatmennyiségnél viszonylag hosszabb időt is igénybe vehet. Ezért, ha erre a műveletre rendszeresen szükségünk van, akkor létrehozunk egy join-indexet. Ez az egyesítésre szolgáló oszlop (város) mellett az egyesítendő egyed típusok (lerakat, vevő) egyértelmű azonosítóját (vevőkód, lerakatkód), vagy az adatbázis-béli belső rekordazonosítóját (ld. 5.4.4. pont) is tartalmazza. A join-indexben való keresés felgyorsítására ennek egyes összetevőire is felépíthetünk indexet. Így ebből bármelyik vevőre közvetlenül megtudhatjuk, honnan szállítsunk neki, illetve melyik lerakattól kiknek célszerű szállítani.

¹ A joint ezen kívül másféleképpen is végrehajthatjuk. A konkrét adateloszlástól függ, hogy egy adott esetben ez, vagy valamilyen más algoritmus vezet gyorsabban eredményre.

Az olvasóra bízunk annak végiggondolását, mi legyen, ha egy városban több lerakat is van, valamint hogyan kezeljék azokat a vevőket, akik olyan városban laknak, ahol a cégnek nincsen lerakata.

VEVŐ	Vevőkód	Vevőnév	Város	...
	1111	Egyes Elek	Budapest	...
	1112	Kiss Anna	Budapest	...
	1114	Nagy Lajos	Pécs	...
	1120	Kő Ede	Miskolc	...
	2100	Gipsz Jakab	Pécs	...
	2222	Erőss Pál	Budapest	...
	2990	Kiss Anna	Szeged	...
		. . .		

LERAKAT	Lerakatkód	Város	Utca	...
	A17	Budapest	Fő u. 1.	...
	B43	Pécs	Mecsek út 7.	...
	B77	Miskolc	Fő tér 5.	...
	C05	Szeged	Tisza köz 6.	...
		. . .		

JOIN- INDEX	Város	Lerakatkód	Vevőkód
	Budapest	A17	1111
	Budapest	A17	1112
	Budapest	A17	2222
	...		
	Miskolc	B77	1120
	...		
	Pécs	B43	1114
	Pécs	B43	2100
	...		
	Szeged	C05	2990
	...		

2.11. ábra: Join index a lerakatok és a szállítási címek városok alapján történő összepárosítására

2.4.2. Hashing

Az indexek alapján való közvetlen eléréshez indextáblázatokat kell létrehozni, azokat karban kell tartani és az indexeken keresztüli elérés is több lépést igényel a gyakorlatban. Mindezeket megtakaríthatjuk a *hashing-módszer* segítségével, feltéve, hogy csak egy kulcs alapján akarunk közvetlen elérést megvalósítani.

A hashing lényege, hogy egy jól kiválasztott algoritmus segítségével a keresési kulcs (ami ebben az esetben csaknem mindig az elsődleges kulcs) és a rekord tárolási címe között egy „majdnem egyértelmű” kapcsolatot hozunk létre. Olyan leképezést alkalmazunk a kulcsok és a címek között, amely különböző kulcsokhoz optimálisan mindig, gyakorlatilag csaknem mindig különböző címeket rendel hozzá. A leképezés által azonos címhez rendelt, de különböző kulcsú elemeket *szinonimáknak* nevezzük.¹ Ezek elhelyezéséről külön kell gondoskodnunk. A szinonimák címét leggyakrabban egy pointerrel, több szinonima esetén egy pointer-lánccal csatoljuk az eredetileg kijelölt címhez.

A megfelelő hash-algoritmus kiválasztásánál két szempontot kell figyelembe vennünk.

Az első az, hogy ne pazaroljunk el túl sok helyet. A leképezésnek tehát olyannak kell lenni, hogy a ténylegesen kiszámított címek között ne legyen túl sok „hézag”, kihasználatlanul maradt cím. Ezért például a numerikus elsődleges kulcsok önmagukban szinte sohasem alkalmasak címként való alkalmazásra is, hiszen például a 11 jegyű magyar személyi számmal nagyjából 10 millió személy rekordját azonosítjuk. Ha a személyi szám egyben a tárolási címet is jelentené, akkor a rendelkezésre álló 10^{11} darab tároló helynek csak 0,01 %-át használnánk ki.

A másik cél az, hogy minél kevesebb szinonim rekordunk legyen. Ha nincsenek szinonimák, akkor bármely rekordot egyetlen egy lépésben megkapunk, ami lényegesen gyorsabb bármilyen index szerinti elérésnél.

A kétféle cél egymásnak ellentmond, mindkettő egyidejűleg szinte sohasem érhető el. Az olyan leképezés, amely „sűrűn” adja a címeket, sok szinonimát is generál. Amelyiknél kevés szinonimát kapunk, az általában rossz tároló kihasználást eredményez. A gyakorlatban észszerű kompromisszumként jónak tartjuk az olyan leképezést, mely 80-85%-os kitöltöttség mellett nem eredményez 20%-nál több szinonimát. Ez azt jelenti, hogy az adatokat durván 1,2 lépéssel tudjuk elérni.

A legegyszerűbb hash-algoritmus az, amikor a numerikus, vagy binárisan ábrázolva numerikusnak tekinthető kulcsot elosztjuk egy olyan prímszámmal, melynek értéke közel van a rendelkezésre álló tároló helyek számához. A kapott maradék jelöli ki a rekord relatív címét, melyből az abszolút cím már meghatározható.

Ahogy telítődik a tárolásra kijelölt hely, úgy növekszik a szinonimák száma. Ha a feldolgozás emiatt lassulni kezd, az adatbázis-felügyelő kimentí az adatokat, nagyobb tároló helyet jelöl ki a számukra, és visszatölti őket az újabb (osztóként pl. más prímszámot használó) hash-algoritmussal. Ebből a felhasználók az adatfüggetlenség (ld. 1.2.1. pont) következtében – az újra visszaállt gyorsabb válaszadási idő kivételével – semmit nem fognak észlelni.

Egy adott kulcs szerint való közvetlen feldolgozásra a jó hashing általában hatékonyabb az indexelésnél. Hátránya azonban a hashingnek, hogy

- hatékonyan csak egyedi kulcsokra lehet alkalmazni,
- szekvenciális elérést nem tesz lehetővé,
- a rekordok nem sűrűsödnek (ld. 2.5.1. pont),
- egy rekordtípuson belül csak egy kulcsra lehet megvalósítani.

Főleg az utolsó érv miatt használják ritkán a relációs adatbázisokban közvetlen elérésre a hashing módszert. Sok adatbázis-kezelő rendszer nem is támogatja, de amelyik ismeri, az is csak akkor alkalmazza, ha ezt explicit, külön kérjük.

¹ Sokszor nem pontos címet, hanem egy címtartományt határozzunk meg, melybe több rekordot is elhelyezhetünk. Ez lehet egy lap, és a rekordot az így kijelölt lapon tároljuk, illetve keressük. Ilyenkor csak azokat a rekordokat tekintjük szinonimáknak, melyek nem fértek rá a hash-algoritmus által kijelölt lapra.

2.5. Adatszerkezési és elérési módok összehasonlítása

Mint azt már említettük, ugyanazt az adatállományt többféle módon szervezhetjük és a megvalósított szervezési módok bármelyike alapján elérhetjük. A következőkben összehasonlítjuk a különböző elérési módokat. Mielőtt azonban ezt megtennénk, egy új fogalommal, a sűrűsödéssel ismertetjük meg az olvasót.

2.5.1. Sűrűsödés

Többször hangsúlyoztuk, hogy az adatokhoz való hozzáférést jelentősen felgyorsíthatjuk, ha az egyes input/output műveletek végrehajtási idejét és a végrehajtandó műveletek számát minimalizáljuk. Ezt úgy tudjuk elérni, hogy azokat a rekordokat, amelyeket várhatóan együtt, vagy egymás után dolgozunk fel, lehetőleg egy lapra, vagy ha ez nem valósítható meg, akkor közvetlenül egymás utáni lapokra helyezzük el. Ennek jellemzésére egy új fogalmat vezetünk be, a *sűrűsödést*. Ennek értéke 0 és 1 (teljes sűrűsödés) között van. Azt mutatja meg, hogy az adott kulcs szerint a rekordok hányad része van a helyén, hányad részénél egyezik meg a fizikai sorrend a feldolgozás logikai sorrendjével. Ez a gyakorlatban azt jelenti, hogy az azonos kulccsal rendelkező rekordok ugyanazon a lapon, vagy ha csak többre férnek el, akkor közvetlen egymás után következő lapokon vannak és ugyanez érvényes az egymás utáni kulcsokkal rendelkező rekordokra is.

Ha például egy rekordtípust leggyakrabban egy kulcs szerinti logikai sorrendben dolgozunk fel, akkor célszerű ezeket az adatokat e szerint a kulcs szerint sorba rendezve bevinni az adatbázisba és ezt a rendezettséget a későbbi módosítások során is megtartani. Így a feldolgozás folyamán egyszerre minimalizáljuk a beolvasandó lapok számát is és az egyes lapok beolvasási idejét is, azaz a feldolgozás idejét (ld. az ellenőrző kérdések 1. feladatát).

Ha a leggyakoribb keresési szempont alapján való közvetlen hozzáférésnél a feltételeknek általában több rekord felel meg, akkor sűrűsödő adatokkal hasonló módon minimalizálhatjuk a hozzáférési időt. Ennek megbecsülésére nézzük meg a következő egyszerű példát.

Tételezzük fel, hogy egy lapon r rekordot tudunk tárolni. Ha a kiválasztási feltételnek több rekord felel meg, akkor teljes sűrűsödés esetén egy fizikai beolvasással r rekord adatait kapjuk meg. Ha a rekordoknak a -ad része lóg ki a sorból (a sűrűsödés $1-a$), az $a*r$ más lapon lévő rekordot jelent. Ha ezeket optimálisan mind egy lapon tudjuk elhelyezni, akkor egy helyett két lap olvasására, ha mind különböző lapokra kerülnének, akkor átlagosan $1 + a*r$, de legalább két lap beolvasására van szükségünk. Ez azt jelenti, hogy minél több rekordot tárolunk egy lapon, annál nagyobb mértékben lassítja a feldolgozást a sűrűsödés „hiánya”. 10% kilógó rekord ($a=0,1$) laponként 40 rekord tárolásánál már 5-szörösére növeli a lemezhez való hozzáférési időt, ami a legtöbb esetben a feldolgozási idő döntő összetevője.

A sűrűsödés nemcsak egy adott rekordtípuson belül lehet előnyös a lemez elérési idő szempontjából. Hierarchikus jellegű feldolgozásoknál, amikor pl. minden egyes szállítónál a szállító rekorddal együtt dolgozzuk fel a hozzátartozó megrendelések rekordjait, akkor célszerű lehet ezeket a különböző típusú rekordokat is ugyanazon a lapon tárolni.

Az adatbázis-kezelő rendszer létre tudja hozni, és fenn is tudja tartani a sűrűsödést, ha az adatbázis-felügyelő megadja, milyen szempontok szerint kell azt megvalósítani. Ha az első betöltéskor rendezve visszük be a rekordokat, akkor eleve sűrűsödve helyezi el őket, és helyet hagy a későbbi beírásokra is. Az új rekordokat igyekszik úgy beilleszteni, hogy a sűrűsödés továbbra is fennmaradjon.

Természetesen, ahogy egyre újabb és újabb rekordokat szúrunk be a meglevők közé, előfordulhat, hogy a bővítésre fenntartott hely betelik. Az új rekordot csak máshová tudjuk elhelyezni. A sűrűsödés mértéke csökkenni fog, a feldolgozás lelassul. Az is előfordulhat, hogy az alkalmazások megváltoznak. Más szempontok, más, jelenleg nem sűrűsödő kulcs

szerint kell az adatokat a leggyakrabban feldolgozni. Ilyenkor az adatbázis-felügyelő feladata, hogy az adatbázis érintett részén a sűrűsödést ismét maximalizálja, illetve a célnak megfelelő másfajta kulcs szerinti sűrűsödést létrehozza. Ez úgy történik, hogy az adatokat kimenti, és a sűrűsödő kulcsnak megfelelő sorrendben – a további bővítésekre ismét helyet hagyva – visszatölti az adatbázisba. Ekkor a sűrűsödés mértéke újra 1 lesz. Ez az újraszervezés az adatbázis-kezelő rendszerek szolgáltató programjaival (ld. 1.3.4. pont) könnyen megvalósítható.

A megfelelő szempont szerint sűrűsödő rekordok segítségével a hatékonyság, a működési sebesség nagymértékben javítható. A probléma az, hogy egy adott rekordtípusra vagy egymással kapcsolatban álló rekordtípusokra a sűrűsödés csak egy kulcs, illetve csak egy kapcsolat szerint valósítható meg. (Bár ez lehet egy több oszlopból álló összetett kulcs is, és ebben az esetben a sűrűsödés az elől álló kulcs(ok)ra is automatikusan fenn áll.) Ilyenkor az alkalmazások gyakorisága, a válaszadási idő korlátjai vagy a feldolgozandó rekordok mennyisége alapján dönti el az adatbázis-felügyelő, hogy melyik keresési kulcs szerint sűrűsödjének a rekordok, melyikre igyekezzen a feldolgozási időt optimalizálni.

2.5.2. Elérési módok összehasonlítása

A tároló helyek egymás után történő megvizsgálásával, *soros eléréssel* minden állomány feldolgozható. Az egymás utáni lapok beolvasása igen gyors, mivel várakozási idő csak az egyik cilinderről a másikra való átálláskor lép fel (ez is csupán mágneslemeznél). Ha az összes rekordot fel kell dolgozni, például adatok biztonsági kimentésekor, akkor egyértelműen a soros feldolgozás a leggyorsabb. Kimutatható, hogy ha az adatoknak viszonylag jelentősebb részét (a cilinderenként tárolt rekordok számától függően 5-15 % fölött) kell elérni, akkor is ez a leggyorsabb, mivel a több lapbeolvasást kompenzálja a rövidebb beolvasási idő. Nem okoz komoly idővesztést a soros feldolgozás akkor sem, ha több rekordot választunk ki, de a teljes állomány csupán néhány lapot foglal el. Ezek a lapok gyorsan beolvashatók, és a pufferekben történő keresés és rendezés milliszekundum alatti ideje elhanyagolható az első lap megtalálási idejéhez viszonyítva.

Más esetekben a soros elérés igen gazdaságtalan. Egy egyedi kulcsértékkel rendelkező rekord megtalálásához átlagosan a tároló helyek felét végig kell vizsgálnunk. Ha több rekord is rendelkezhet a keresett értékkel és mindegyiket meg kell találnunk, vagy annak megállapítására, hogy az adott kulcsú rekord nincs bent az állományban, mindig végig kell olvasnunk az állomány tárolására szolgáló egész területet.

Előnye viszont a soros szervezésnek, hogy új adatok beírása az állomány végére igen egyszerű. Meglévők módosítása sem okoz gondot, ha a módosított rekord elfér a régi helyén. A törlés is egyszerű, ha a törölt rekordok helyét nem akarjuk felhasználni, az így keletkezett üres helyeket alkalmanként végrehajtott újraszervezéssel szüntetjük meg. További előny még, hogy ha a helyben történő módosítástól és törléstől eltekintünk, akkor ez a szervezés és elérés a lemezeknél jóval olcsóbb mágnesszalagokkal és kazettákkal is megvalósítható.

A *szekvenciális szervezés*, a *listaszerkezet* előnyei a soros szerkezettel szemben:

- A módosítás, beszúrás, törlés egyszerűbb.
- Ugyanazon rekordtípuson többféle listaszerkezetet is létrehozhatunk. A soros feldolgozás csak egyféle sorrendet tesz lehetővé.
- A keresett rekordok megtalálásához, vagy annak megállapításához, hogy az nincs az állományban átlagosan csak a rekordok felét kell végigvizsgálnunk.
- Listaszerkezettel másféle, például hierarchikus elérés is megvalósítható ugyanazokon az adatokon. Például pontterek kötik össze egy szállító és a tőle származó szállítások rekordjait.

A *listaszerkezet* hátrányai a soros eléréssel szemben:

- A feldolgozás az egyes rekordok beolvasása közötti hosszabb várakozási idő miatt lényegesen lassabb. Nem áll fenn, vagy csak korlátozott mértékben érvényesül ez a hátrány akkor, ha a listaszerkezetet sűrűsödő kulcsra építjük fel. Ha a sűrűsödés I , akkor teljes mértékben kihasználjuk a soros és a szekvenciális feldolgozás előnyeit, lényegében azok hátrányai nélkül. Sajnos ez egy rekordtípusnál csak egy kulcsra valósítható meg.
- A pointerok miatt nagyobb a tároló igény.
- Módosítás, törlés, új rekord beírása a pointerok karbantartása miatt még akkor is lassabb, ha a fizikai tárolási sorrend teljesen megegyezik a logikai feldolgozás sorrendjével.

Az *indexelt szervezés* az inextáblázaton keresztül általában lehetővé teszi a szekvenciális elérést is. Előnyei az *indexelt elérésnek* még, hogy

- igen hatékony akkor, ha a rekordoknak csak egy kis részéhez akarunk hozzáférni,
- akárhány keresési szempontra megvalósítható,
- lehetővé teszi a keresést több feltétel egyidejű teljesülésének esetére.

Az *indexelt szervezés* hátrányai:

- Az inextáblázat helyet, folyamatos karbantartása időt igényel.

A *hashinggel* megvalósított közvetlen szervezés és elérés előnyei:

- Igen hatékony egyedi rekordok megtalálására.
- Nem igényel külön helyet az inextáblázat számára.

A *hashing* hátrányai:

- Csak egy kulcsra valósítható meg.
- Duplikátumok keresésére nem alkalmas.
- Szekvenciális feldolgozást nem tesz lehetővé.
- A szinonimák számának megnövekedésekor újra kell szervezni.

Az előző oldalon lévő 2.3. táblázatban összefoglaltuk a különböző adatszervezési és elérési módok legfőbb előnyeit és hátrányait.

2.3. táblázat: **Különböző adatszervezési és elérési módok legfőbb előnyei és hátrányai.**

Szempont	Soros	Szekvenciális	Indexelt	Hashing
Tároló hely	Jó	Külön hely kell a pointerok számára	Külön hely kell az inextáblázat és az abban való keresés számára	Elég jó
Szekvenciális feldolgozás	Rendezést igényel	Megvalósítható, sűrűsödő kulcsra igen hatékony	Megvalósítható, sűrűsödő kulcsra igen hatékony	Nem alkalmazható
Nagyon sok rekord keresése	Hatékony	Csak sűrűsödő kulcsra hatékony	Csak sűrűsödő kulcsra hatékony	Nem alkalmazható
Néhány rekord keresése	Rossz	Rossz	Igen hatékony	Nem alkalmazható
Egyedi rekordok keresése	Rossz	Rossz	Igen hatékony	Igen hatékony
Keresés több kulcs alapján	Rossz	Rossz	Hatékony és igen hatékonyra tehető	Nem alkalmazható

Szempon	Soros	Szekvenciális	Indexelt	Hashing
Rekordok törlése	Egyszerű, de időnkénti újraszervezést igényel	Egyszerű de a pointerok automatikus karbantartását igényli és időnként újraszervezést is igényelhet	Egyszerű, de az indexek automatikus karbantartását igényli	Nagyon egyszerű
Rekordok módosítása	Nehézkes	Egyszerű, de a pointerok automatikus karbantartását igényli	Egyszerű, de az indexek automatikus karbantartását igényli	Nagyon egyszerű
Új rekordok beírása	Nagyon egyszerű	Egyszerű, de a pointerok automatikus karbantartását igényli	Egyszerű, de az indexek automatikus karbantartását igényli	Nagyon egyszerű

2.6. Keresés több kulcs szerint

Gyakran keresünk az adatbázisban ilyen jellegű kérdésekre választ: „Kik azok a 25 - 35 év közötti, közgazdász végzettségű személyek, akiknek legalább 15 év vezetői tapasztalatuk van?”

Amennyiben több keresési kulcsra egyidejűleg fennálló feltételek teljesülése alapján akarunk az adatokhoz hozzáférni, akkor az SQL optimalizáló az adatok mennyisége, eloszlása és a rendelkezésre álló indexek alapján meghatározza a – szerinte – optimális utat, melyen keresztül ez várhatóan a leggyorsabban valósulhat meg. Természetesen az optimalizáló is hozott anyagból dolgozik. Csak azokat az adatszerkezeteket, csak azokat az indexeket tudja felhasználni, amelyeket a felhasználók és az adatbázis-felügyelő létrehoztak.

Ha a kiválasztásban szereplő kulcs-kombinációk alkalomszerűen változnak, akkor nem tehetünk túl sokat. Felépítjük a rendszeresen használt keresési kulcsokra az indexeket, és rábízunk a rendszerre, melyeket használja majd ezek közül (ld. 2.7. alfejezet). Sokszor azonban az alkalmazás logikájából eleve tudjuk, hogy bizonyos keresési kulcsok rendszeresen egy adott kombinációban szerepelnek a kiválasztási szempontokban. Jó példa erre, amikor az elsődleges kulcsot két oszlop kombinációja határozza meg. Ez áll elő az 1.2.2.1. pontban ismertetett MITHALLGAT táblázatban is. Ebben sem a hallgató (HID), sem a tantárgy azonosítója (TID) külön-külön nem határozza meg egyértelműen sem az osztályzatot, sem a gyakorlati pontok számát. Csupán a két azonosító együttes kombinációja teszi egyértelművé ezek értékét. Ahhoz, hogy egy új MITHALLGAT rekord beépítésénél, vagy meglévő módosításánál egyszerűen és gyorsan megállapíthassuk, van-e már az új/módosított (HID,TID) kombinációval jellemzett rekord, nem elég önálló indexeket építeni a HID és a TID mezőkre. Külön egyedi indexet kell definiálnunk a két mező kombinációjára is. Ebből azonnal ellenőrizhető, hogy az új rekord beírható-e, illetve végrehajtható-e a kért módosítás.

Ha az összetett indexben az oszlopok sorrendje HID,TID, akkor ez az index fölhasználható a MITHALLGAT táblázatban a hallgatók azonosítója (HID) alapján való közvetlen hozzáférésre is.¹ Fölösleges lenne HID-re egy újabb, duplikált indexet felépíteni. Ez ugyanazokat a címeket tartalmazná, mint az egyedi (HID,TID) index. A különbség csupán annyi, hogy az elsőben egy HID értékhez több cím is tartozik, de ezek mindegyike egyenlő az egyedi (HID,TID) indexértékekhez tartozó egyedi címek valamelyikével.

A 2.4.1.4. pontban láthattuk, milyen hasznos lehet több táblázatnak egy közös oszlop azonos értékei alapján történő rendszeres egyesítéséhez egy önálló join-indexet létrehozni. A

¹ A tantárgy azonosító (TID) alapján való közvetlen hozzáférést ez az index nem teszi lehetővé.

következőkben bemutatjuk, milyen hasznos lehet egy új, összetett index létrehozása, ha az index oszlopai rendszeresen együtt szerepelnek a kiválasztási feltételben.

Tegyük fel, hogy egy személyi nyilvántartó rendszerben rendszeresen keresünk embereket (emberek rekordjait) három szempont (életkor, szakképzettség, szakmai tapasztalat) együttes teljesülése alapján. Ha ezek mindegyikére vannak indexek, akkor bit-térképek esetén azok egybevonásával (ld. 2.4.1.3. pont), B⁺ fa indexek esetén (ld. 2.4.1.2. pont) általában valamelyik index¹ szerint végigmenve, az indexelt értékhez tartozó rekordok egyenkénti beolvasásával, és a másik két feltételnek a beolvasott rekordban való ellenőrzésével, vagy a három indextáblázatból a három értékhez tartozó közös rekordcímek kiválasztásával kaphatjuk meg a keresett rekordokat. Bármelyik módszert választja az optimalizáló, speciális esetektől eltekintve, mindig lényegesen több lapot kell beolvasnunk, mint ahány lapon a keresett rekordok elhelyezkednek.

Jelentősen lecsökkenthetjük a lemez hozzáférések számát, és ez által az elérési időt, ha a három keresési szempont kombinációjára is felépítünk egy önálló indexet. Ez rögtön azoknak a rekordoknak a címét adja meg, amelyekre mindhárom feltétel teljesül. Mivel a három szempont együttesen már elég szelektív, egy adott hármas kombinációhoz tartozó címek általában 1 - 2 lapon elférnek. Ez(eke)t beolvasva a keresett rekordokat a címük alapján egyből megtaláljuk. Az input/output műveletek száma gyakorlatilag megegyezik a feltételnek megfelelő rekordok számával. Ha a hármas, kombinált index alapján kapott címeket a tényleges beolvasás előtt rendezzük, akkor a fizikai lapműveletek száma még ennél is jóval kevesebb lehet, hiszen egy lapon több keresett rekordot is találhatunk.

Az új index a személyi adatok és a hozzátartozó indexek számára szükséges tárterületet legföljebb néhány százalékkal növeli meg. Ezzel szemben a hozzáférési idő egy nagyságrenddel is lecsökkenhet, mert az optimalizáló ennek alapján fogja meghatározni az elérési utat. De ezt csak akkor tudja megtenni, ha ezt az indexet előzőleg létrehoztuk!

2.7. Optimális elérési út meghatározása

Az adatszervezésről és elérésről eddig mondottakat ebben a pontban összefoglaljuk és ismét áttekintjük.

Az adatbázisban a keresett adatokhoz különböző módokon lehet hozzáférni. A lehetséges megoldások közül általában az SQL optimalizáló választja ki a rendelkezésére álló információk alapján az optimális elérési utat. Ehhez a katalógusban lévő statisztikát (ld. 7.3. alfejezet) használja fel. Ez rendszeresen karbantartott adatbázisban lényegében naprakészen mutatja a különféle rekordok számát és a különböző keresési kulcsok szerinti eloszlását. Amennyiben ilyen statisztika nem áll rendelkezésre, akkor az optimalizáló ezeket ésszerű feltételezésekkel helyettesíti, hogy a valószínűleg legjobb elérési utat meg tudja határozni. A generált elérési út részleteit az EXPLAIN SQL parancs segítségével ismerhetjük meg.

Egyes rendszerek legújabb verzióiban az SQL utasítás kiadója is meghatározhatja, hogy az adott utasításban milyen úton kívánja elérni az adatokat. Ezt azonban csak az igen gyakorlott adatbázis-felügyelőknak és felhasználóknak ajánljuk, és kizárólag olyan esetekben, amikor az adatok speciális eloszlásáról olyan információik vannak, melyekről az optimalizáló nem tudhat.

A továbbiakban egy viszonylag egyszerű példán mutatjuk be, hogy a megvalósított adatszerkezetektől, az adatok mennyiségétől és eloszlásától függően milyen módon célszerű megválasztani az elérési utat. Feladatunk az, hogy a gépjárműveket leíró rekordokból ABC

¹ Ez az index vagy a három keresett érték közül a legkevesebb címet tartalmazó indextábla (legkevesebb lapot kell beolvasni), vagy a sűrűsödő index (leggyorsabban lehet a lapokat egymás után beolvasni) lesz.

sorrendben kiválasszuk azoknak az üzembehelyezőknek a nevét és a gépjárművek rendszámát, melyeket 2006-05-05-én állítottak forgalomba és a modell „Ford Focus”.¹

A következő lehetőségeink vannak:

1. Sorosan, fizikai elhelyezésük sorrendjében végigolvassuk az összes gépjármű rekordot. Egyenként ellenőrizve kiválasztjuk belőlük a mindkét feltételnek megfelelő rekordokból az üzembehelyező nevét és a rendszámot és az eredményt rendezzük név szerint.

Ha csak soros adatszerkezet van, akkor ez az egyedüli megoldás.

Más szerkezet megvalósulása esetén is ez lehet az optimális elérési út, ha az együttes feltételeknek relatív sok rekord felel meg (durván > 10%) és egyik keresési szempont és a név szerint sem sűrűsödnek a rekordok.

Gyakorlatilag nem rosszabb a hatékonyság más szervezési mód megléte esetén sem, ha a rekordok száma kevés, egy - két lapon elférnek.

2. Szekvenciális szerkezetet építettünk fel az üzembehelyező nevére. Ha ezen végigmenve választjuk ki a rekordokat, megtakarítjuk a rendezést.

Ha a rekordok sűrűsödnek a név szerint, akkor ez valamivel jobb az 1. megoldásnál. Ha nem sűrűsödnek, és néhány lapnál több helyet foglalnak el, akkor lényegesen rosszabb lesz nála.

3. Index van valamelyik, vagy külön-külön mindkét keresési szempontra.

Az egyik index alapján beolvassuk a feltételnek megfelelő rekordokat (pl. a *Ford Focus*okat), kiválasztjuk közülük a másik feltételnek is megfelelőkből a nevet és a rendszámot, ezeket rendezzük.

Ha az index szerinti kiválasztás már mérsékelten szelektív (néhány százalék alatti), akkor ez egyértelműen jobb az 1. és 2. módszernél. Tovább javíthatunk, ha

- azon az indexen megyünk végig, amelyik kevesebb rekordot választ ki,
- a sűrűsödő indexen megyünk végig (ha van), még akkor is, ha a találatok száma egy nagyságrenddel is több.

4. Index van mindkét keresési szempontra.

Mindkét indextáblázatból kikeressük a feltételnek megfelelő rekordok címét, rendezzük, a közös címekről kiválasztjuk a nevet és a rendszámot, az eredményt név szerint rendezzük.

Nagyjából egyenértékű a 3. módszerrel. Az adatok mennyiségétől és eloszlásától függ, hogy konkrét esetekben éppen melyik a célravezetőbb.

5. Összetett index van a két keresési szempontra.

Ha ezt az indexet használjuk, és a kiválasztott rekordokat rendezzük, akkor jobb az összes eddig felsorolt hozzáférési eljárásnál. Még tovább javíthatunk, ha az összetett kulcs, vagy annak első komponense szerint sűrűsödnek a rekordok.

6. Mesterséges indexet készítünk, melynek összetevői: modell; üzembe helyezés napja, üzembehelyező neve, rendszám.

Ezen index alapján az indextáblázatból azonnal megkapjuk az eredményt. A gépjármű rekordokhoz hozzá sem kell nyúlni.

Ez a módszer a leggyorsabb, nagyságrenddel gyorsabb hozzáférést eredményez még az 5. módszernél is. Ha ez a fajta keresés rendszeres, akkor érdemes az indexterület némi növelése árán ezt a hozzáférési módot lehetővé tenni.

¹ Az erre szolgáló SQL utasítás (ld. 5.6.3.2. pont):

```
select nev,rendszam from gepjarmu where uzembehelyezes='2006-05-05' and modell='Ford Focus'
order by nev.
```

2.8. Adattömörítés

A lemeztárolók kapacitásának növekedése és áruknak csökkenése ellenére is fontos szempont maradt az adatbázisokban a külső tárolón lévő hely jó kihasználása. Ha a hasznos adatok kevesebb helyet foglalnak el a tárolón, akkor

- ugyanazon a hardveren több adatot tárolhatunk.
- egy lapra több adat fér el. Ritkább lesz a lapváltások, a fizikai író/olvasó műveletek száma. Felgyorsul a lemez műveleteket is igénylő feldolgozás.
- kevesebb adatot kell átvinni a hálózaton. Gyorsabb az együttműködés elsősorban az osztott adatbázisokban illetve adatbázisok között.

2.8.1. Változó hosszúságú mezők

A legegyszerűbben változó hosszúságú mezők alkalmazásával rövidíthetjük le a rekordjaink hosszát, tárolhatjuk ugyanazt az információt kevesebb helyen. Ha egy mező értékes adattartalma változó hosszú lehet (pl. nevek, lakcímek), akkor ezt a mezőt olyan hosszúra kell deklarálni, hogy a leghosszabb adat is elférjen benne. Ugyanakkor, a karakteres mezőknél a legtöbb esetben csupán a mező elején lévő karakterek szolgáltatnak érdembeli információt. A mező végén levő szóközök fölöslegesen foglalnak le értékes helyeket. A névmezőbe be kell férjen az 57 karakterből álló „Szentmihályfalvai Szilveszterné Alsónagybányai Anasztázia” név is, de lehet hogy csak „Kő Ede” áll benne 51 utána következő felesleges szóközzel.

A mező végén levő, információt nem tartalmazó szóközöket elhagyhatjuk változó hosszúságú mezők definiálásával (ld. 5.4.1. pont). Ezek hossza a felhasználó felé egyenlő a maximálisan előfordulható adat hosszával. De a fizikailag ténylegesen tárolt mező ennél rövidebb lehet, mivel a szóvégi szóközöket nem tartalmazza. A mező előtt általában 2 bájt szolgál a mező tényleges hosszának jelölésére. Így a maximálisan N bájt hosszúságú, végén n szóközt tartalmazó karakteres mező a valóságban $N+2-n$ bájtot fog elfoglalni a lemezen.

Mivel a változó hosszúság jelzése is helyet foglal el, és kezelése, a mező értékes hosszának megállapítása, illetve szóközzel való feltöltése és az utána következő mezők kezdőhelyének megállapítása időt igényel, a gyakorlatban csak akkor érdemes ezzel dolgoznunk, ha a rekordonkénti átlagos helymegtakarítás mezőnként legalább 10 -15 bájt.

A legtöbb adatbázis-kezelő rendszer numerikus mezőknél is lehetővé teszi, hogy az ábrázolandó értékes jegyeiktől függően két lehetőség között választhassunk. Egy egész- vagy lebegőpontos számot hosszabb vagy rövidebb formában is tárolhatunk.

2.8.2. Kódtáblázat

Olyan szöveges mezőknél, melyek csak meglehetősen korlátozott számú különböző értéket vehetnek fel, hasznos lehet a kódtáblázat használata. Példa erre, ha a lakcímbe, vagy a születési helybe az ország neve helyett annak a gépjárműveken használatos 1 – 3 betűs ország kódját írjuk be. Ezzel helyet takarítunk meg. A rekordban lévő ország kódból viszont meg kell állapítanunk az ország nevét, ami időbe kerül. Mivel a kódtáblázat általában kicsi, teljes egészében állandóan benntartható a központi memóriában. Ilyenkor az átkonvertálás maga nem igényel tényleges lemez műveletet. A kódtáblázat használata elhanyagolható mértékben növeli csak meg a feldolgozási időt. Ha rosszul szervezzük a munkánkat, vagy a kódtáblázat olyan nagy, hogy munka közben lemezen kell tartani, meggondolandó a használata. Különösen igaz ez akkor, ha a megtakarítás mezőnként csak néhány bájt.

Nehezebb a kódtáblázat felépítése, ha nagyon sok különböző érték fordulhat elő. De ilyenkor is hasznos lehet, ha a sok érték között az esetek legnagyobb részében csak néhány száz szerepel. Ilyen lehet az előző példánál maradva a város, amelyik egy átlagos magyar

adatbázisban közel 20 százalékban Budapest, a további 20 – 30 százalékban pedig a legnépesebb 20 -30 magyar város valamelyike lesz. Ilyenkor sokat javít egy olyan, az adatok eloszlásához illeszkedő kódtáblázat létrehozása, melyben a leggyakrabban előforduló elemek kódja a legrövidebb (1 - 2 karakter), míg az igen ritkán előforduló elemek kódja hosszabb, vagy esetleg nincsenek is kódolva. Egy ilyen táblázatban például Budapest kódja lehet *B*, Debrecené, Miskolcé, Pécsé rendre *D*, *M*, *P*, Szolnoké *SZL*, Szombathelyé *SZM*, és az igen ritkán előforduló Jászkarajenő vagy a külföldi város nevek egyáltalában nincsenek kódolva. Az így elérhető helymegtakarítás az adatok eloszlásától és a kódoló ügyességétől függően 75-80 % is lehet.

Egyes adatbázis-kezelő rendszerek, így pl. az IBM UDB/DB2, ha külön kérjük (COMPRESS YES) felépítik helyettünk ezt a kódtáblázatot, és automatikusan használják is.

2.9. Rendezés

Az adatbázisokból nagy mennyiségű információt nyerhetünk. Ahhoz azonban, hogy ez áttekinthető legyen, szükséges, hogy az eredmények bizonyos megfelelően kiválasztott szempont, vagy szempontok szerint rendezve legyenek. Ezen kívül az adatbázisban való munka közben is gyakran szükségünk van a kiválasztott adatoknak vagy azok egy részének rendezett formában való előállítására. A rendezés rövid ismertetését az indokolja, hogy ez az adatfeldolgozás egyik legidőigényesebb művelete. Ez különösen nagy rekordmennyiségnél jelentkezik, mivel a rendezés ideje a rendezendő rekordok számával nem egyenesen arányos, a rendezési módszertől függően $n \cdot \log(n)$ és n^2 közötti nagyságrendben növekszik. (n a rendezendő rekordok száma).

A rendezéshez szükséges műveleteket az adatbázis kezelő rendszer automatikusan elvégzi. Általában nem magukat a rekordokat rendezi fizikai sorrendben, hanem a rendezendő szempont szerint készít egy indextáblázatot és ennek alapján szolgáltatja megfelelő sorrendben a rekordokat.

Rendezésre a következő esetekben van szükség:

- Az eredményeket valamilyen szempont szerinti sorrendben kívánjuk megjeleníteni és a kiválasztás az adatbázisból nem e szerint a szempont szerint történt.
- Új indexet hozunk létre a már az adatbázisban lévő adatokon.
- A feldolgozást az SQL értelmező több lépésre bontja fel, és ezek valamelyikében az ideiglenesen tárolt adatokra rendezett formában van szükség.
- Azonos értékekkel rendelkező adatok kiszűrése, csoportosítása az azonos érték alapján.
- A fentiekén kívül nagy adatmennyiségek bevitelekor általában célszerű az adatokat a beírás előtt rendezni. Ez azonban az adatbázis-kezelő rendszer keretein kívül történik.

Igen gyakran előfordul ad hoc jellegű lekérdezéseknél, hogy egyes felhasználóknak a legkülönbözőbb sorrendben, csoportosításban van szükségük az eredményekre. Mivel a kiválasztott adatok és a rendezési szempontok állandóan változnak, ezek mindegyikére nem érdemes állandó indexet felépíteni.¹ Ha ilyen feladatot kell megoldani, akkor a SELECT utasításban (ld. 5.6.3.2. pont) megadott ORDER BY utasítás kiegészítés hatására az adatbázis-kezelő rendszer generálja a rendezéshez szükséges utasításokat, létrehozza a rendezéshez szükséges ideiglenes állományokat, majd az utasítás végrehajtása után törli ezeket.

Sokszor előfordul, hogy az adatbázis felépítése után jövünk rá arra, hogy olyan mezők értéke alapján akarunk rendszeresen keresni, vagy rendezett sorrendben feldolgozni, melyekre

¹ Az igen gyakran használtakra természetesen célszerű indexeket létrehozni, mint azt ebben a fejezetben is láthattuk.

nincs index. Ebben az esetben a hatékonyságot jelentősen megnövelhetjük, ha ezekre a mezőkre utólag egy indexet építünk fel. Ezt bármikor megtehetjük. De ehhez az adatbázisnak a rekordokat előbb rendezni kell ezek szerint a mezők szerint. Ez sok rekordnál hosszú időt is jelenthet, de csak egyszer kell elvégezni. Utána már élvezhetjük az indexek alapján való közvetlen hozzáférés előnyeit.

Adatok csoportosításánál és az egyes csoportokra jellemző értékek meghatározásánál (pl. évfolyamok szerinti tanulmányi átlag), vagy teljesen azonos eredményeknél a duplikátumok kiszűrésénél is legcélszerűbb az adatokat először rendezni. (A SELECT utasításban GROUP BY utasítás kiegészítés vagy DISTINCT szerepel). A rendezett állományon való szekvenciális végighaladáskor a csoportváltásokat, a duplikált értékeket egyszerűen tudjuk észlelni.

Rendezéskor is, mint minden számítógépes feladat megoldásakor, az idő, a központi memória, és a külső tárolókapacitás, valamint a beszerzési és üzemeltetési költségek ellentmondásával találkozunk. (Hiába, mint azt többször is láttuk „Nincs potya kaja, there are no free lunches”!) Gyors rendezéshez nagy központi memória, de ha ez nem elég, akkor ehhez kiegészítésül még sok nagy, ideiglenesen felhasználható lemezterület kell. Ebből főleg az első sok pénzbe kerül. Kisebb memóriánál és lemezkapacitásnál viszont a rendezési idő nő meg. Ez pedig, főleg párbeszédese munkánál, mivel ez a felhasználó idejét jelenti, még drágább.

Az ellentmondás feloldására – mint a számítástechnikában mindig –, észszerű kompromisszum született. Mivel rendezésnél a rendezendő adatokhoz sokszor kell hozzányúlnunk, ezért ezekből amennyit csak lehet, annyit beolvassunk a memóriába és ott sorba rendezzük. (Adatokon itt most általában nem a teljes rekordot, hanem csak a jóval rövidebb rendezési kulcsot és rekord lemezcímét értjük). A rendezett részhalmazt kiírjuk lemezre egy ideiglenes állományba, és a rendezendő adatok újabb csomagját olvassuk be rendezés céljából a központi memóriába. Rendezés után ezt is kiírjuk egy másik állományba. Ezt az eljárást addig folytatjuk, amíg a teljes rendezendő állomány ki nem kerül rendezett részhalmazokban a lemez(ek)re. Ezután a rendezett részhalmazokat azok és a tárolásukra szolgáló perifériák számától függően egy, vagy általában több lépésben a lemezeken összeválogatjuk egyetlen egy rendezett halmazba.

A fentiek alapján a rendezés hatékonysága két részből tevődik össze. Ezek

- A rendezés hatékonysága a belső memóriában. Ez függ a rendezési módszertől és a rendezésre rendelkezésre álló memória nagyságától, hány rekordot rendezhetünk egy lépésben.
- A lemezeken ideiglenesen tárolt rendezett részhalmazok összeválogatására szolgáló algoritmustól, és a részhalmazok tárolására szolgáló lemezek (önálló lemez területek) számától.

Ezek közül az átlagos felhasználó semmit nem tud befolyásolni. Legfőljebb annyit tehet, hogy ha módja van, akkor igyekszik feladatait úgy megfogalmazni, hogy minél kevesebb esetben legyen sok rekord rendezésére szükség. Az adatbázis-felügyelő is csak annyit javíthat, hogy a központi memóriában való rendezéshez az egyéb igények figyelembevételével maximális tároló helyet biztosít és sok lemezt (lemez területet) jelöl ki az ideiglenes tárolásra.

2.10. Ellenőrző kérdések

1. Becsülje meg, mennyivel lassul le a szekvenciális feldolgozás, ha a rekordok a -ad részénél (sűrűsödési faktor $1-a$; $a < 0,1$) nem egyezik meg a fizikai sorrend a logikaival! A „kilógó” rekordokat ugyanazon a cilindren tároljuk.

2. Mi lehet a magyarázata annak, hogy változó hosszúságú numerikus adatoknál – ellentétben a karakter típusokkal – nem választhatjuk meg szabadon az adatok hosszát, hanem általában csak két lehetőség között választhatunk?
3. Hogyan határozható meg bit-térképénél az $n1, n2, \dots$ pozíciókon lévő I értékekből az adott indexhez tartozó rekordok címe a rekordcímek táblázatából?
4. Hogyan kell kiegészíteni a join-indexet, vagy értelmezni az eredményét a 2.4.1.4. pontban megadott példában, ha egy városban több lerakat is lehet, és ha van olyan város, ahol nincsen lerakatunk?
5. Ha feltételezzük, hogy a B^+ fa egy indexlapján 100 indexbejegyzést tudunk tárolni, akkor hány lépésben tudunk elérni egy adott rekordot, ha a rekordok száma 1 millió, illetve 100 millió?
6. Miért nem lehet a hashing algoritmussal egy rekordon belül többféle mezőre is külön-külön keresni?
7. Felgyorsítja-e a sűrűsödés az elsődleges kulcs alapján való feldolgozást, ha a rekordokat
 - mindig csak közvetlen eléréssel dolgozzuk fel,
 - időnként szekvenciálisan is feldolgozzuk az elsődleges kulcs szerint, de a szekvenciális feldolgozás zömmel más szempont alapján történik,
 - rendszeresen feldolgozzuk szekvenciálisan az elsődleges kulcs szerint?
8. Miért célszerű változó hosszúságú tömörítésnél a leggyakoribb értékekhez a legrövidebb kódot hozzárendelni?

3. ADATMODELLEK

Az adatbázis-fejlesztés első lépése az, hogy meghatározzuk, milyen információkra van szükségünk, milyen adatok állnak ezekhez a rendelkezésünkre, és milyen adatmodell segítségével fogjuk az adatbázist megvalósítani, hogy abban az adatokat tároljuk és abból az információkat előállíthassuk. Ebben a fejezetben az adatmodell kialakításával foglalkozunk. Az adatmodell az adatokat a felhasználó szemszögéből vizsgálja, és – legalábbis elvileg – független attól, hogy megvalósítása milyen módon, milyen technikával történik.

Az adatmodellel mindig az adatbázis koncepcionális szintjének (ld. 1.2.2.3. pont) a logikai szerkezetét írjuk le.

Ahhoz, hogy az adatmodellt elkészíthessük, meg kell állapítanunk, hogy

- mikről, illetve kikről akarunk adatokat tárolni. Ezek lesznek modellünk *egyed típusai, entitásai*.
- az egyes egyed típusokról mit akarunk megtudni, milyen adataikra lesz szükségünk. Ezek lesznek az egyed típusok *tulajdonságai, attribútumai*.
- a különböző egyed típusok között milyen *kapcsolatok* állnak fenn.

Az adatmodell véges számú egyed típusnak és azok egyenként is véges számú tulajdonságainak és kapcsolatainak az összessége. Nem foglalkozik ezen egyed típusok egyedeivel, konkrét előfordulásaikkal (instance), azzal, hogy ezeknek mik az értékei, hogyan kerülnek tárolásra, és miként lehet velük műveleteket végezni. Ez az adatmodell realizált formájának, az adatbázisnak, mely az adatmodell egyed-előfordulásainak szervezett összessége, és az ezen adatokban lévő információk kinyerésére szolgáló adatbázis-kezelő rendszernek a feladata.

Az egyed típusokon és azok kapcsolatain alapuló adatmodellt *egyed típus – kapcsolat*, vagy angol elnevezése alapján *Entity – Relationship*, röviden *E – R modellnek* hívjuk. Ha az egyed típusokat relációkként ábrázoljuk, akkor beszélünk a *relációs modellről*. Napjainkban a működő adatbázisok szinte kizárólag a relációs modellre épülnek.

A modellt kialakíthatjuk felülről lefelé, Top-Down, vagy alulról építkezve, Bottom-Up tervezési módszerrel. A Top-Down módszernél először az intézményi stratégiai információkból kiindulva határozzuk meg az egyes egyed típusokat, tulajdonságaikat és kapcsolataikat. Nagy adatbázisokat általában így terveznek. A Bottom-Up eljárás a felhasználó által igényelt adathalmaznak a kívánt adatstruktúrába való elrendezésére, és a Top-Down eljárással kapott modell további finomítására alkalmas, mint azt majd a 4.6. alfejezet normalizálási példáján bemutatjuk.

3.1. Az adatmodellezés célja

Az adatmodellezés segít abban, hogy megértsük az adatok szerkezetét, értékelni tudjuk tartalmukat. Ez szükséges ahhoz, hogy az adatbázist megtervezzük. Miután a megtervezett adatbázist felépítettük, adatokkal feltöltöttük, segítségével információkat szolgáltatathatunk a napi munkához és a döntéshozatalhoz.

Az adatmodell számítógépes megvalósítása az adatbázis, az adatbázisból való információ szolgáltatást pedig a számítógépes programok teszik lehetővé.

Az adatmodellezés egy módszer, mellyel meghatározzuk, mi kerüljön be az adatbázisba. A cél az, hogy alkalmazásával logikailag csoportosítsuk a valós világnak azt a részét, azokat a tényeket, amiket az adatbázisban tárolnunk kell. Nem törődik azzal, hogy ez miként valósul meg. Ez annak a feladata, aki a modell alapján megvalósítja az adatbázist. Azzal sem foglalkozik, miként fogjuk az adatokat feldolgozni. Ez a rendszer-elemzés, -szervezés területe. Az adatmodellezés célja az, hogy olyan adatmodellt hozzunk létre, amely

a megoldandó feladat szempontjából valóságként, egyértelműen, teljesen és minimálisan, azaz csak a szükséges adatok tárolásával ábrázolja a valós világ adatait és kapcsolatait.

A modell megalkotásában egyenrangú félként vesznek részt a tervezendő (illetve módosítandó) adatbázis felhasználói és az adatmodellező szakemberek.

Az adatmodell létrehozása egy iteratív folyamat. Formája a felhasználók és a tervezők közti együttműködés eredményeként állandóan változik. A végső verziója alig fog hasonlítani az elsőre. Sőt, a már utolsónak tekintett, üzembe vett változata is még több módosításon fog átmenni, részben utólag kiderült tervezési hibák, de főként az élet változásai miatt. Ha papíron dolgozzuk ki a modellt, akkor csak puha ceruzát használjunk. Legyen kéznél egy nagy radír, és a papír közepén kezdjük el a rajzolást, mert nem tudhatjuk, milyen irányba és milyen nagyra fog a modell ábrája kiterjeszkedni. Ha pedig az egyre jobban elterjedő szoftvertervező eszközök valamelyikét használjuk, mindig figyeljük kritikus szemmel a kapott eredményeket. Könnyen lehet, hogy az „általában optimális” modellt szolgáltató szoftver a mi „speciális” esetünkben a gyakorlatban csak nehezen, vagy rossz hatékonyságot eredményező megoldást javasol. Alapelvünk az, hogy aki tervezéskor száz százalékgig rábízta magát a különféle *kész tulokra* (CASE Tool¹), az egy kész tulok!

3.2. Az adatmodellek fejlődése

Az első adatbázis-kezelő rendszereket az 1960-as évek elején helyezték üzembe. Ezek a *hierarchikus adatmodellen* alapultak. Ennek az volt a magyarázata, hogy a valós világ nagyon sok területét jól lehetett hierarchikus szerkezetben ábrázolni egyértelmű alá- és fölérendelési viszonyokkal. Ilyen volt közelítően egy vállalat felépítése, vagy egy gyártmány összeszerelése az alkatrészeiből. A gyakorlatban a '60-as '70-es években számos hierarchikus adatbázist hoztak létre és sok ezen alapuló rendszert fejlesztettek ki. Ezek közül az IBM Information Management System (IMS) rendszere volt a legelterjedtebb. Ez 1968-ban jelent meg a piacon és korszerűsített verzióit régi nagy rendszerekben helyenként még ma is használják.

A hatvanas évek végén alakult meg a CODASYL Bizottság keretében a Data Base Task Group, amelyik célul tűzte ki, hogy rendet teremt az adatbázis technológiában azidőtájt uralkodó dzsungelben. Törekvésüket siker koronázta. 1971-ben jelent meg a végleges jelentésük. Az itt lefektetett alapelvek kiállták az idők próbáját és a mai adatbázis-kezelő rendszerek felépítése is ezek alapján valósul meg (ld. 1.2. alfejezet).

A '70-es évek második felétől egy bő évtizedig a *hálós modellen* alapuló rendszerek voltak a legnépszerűbbek. A hálós szerkezet egy olyan általánosított hierarchiát jelent, ahol az elemek között nem csak 1:N (illetve 1:1) egy-több hierarchikus kapcsolat állhat fenn, hanem tetszőleges N:M (több-több) kapcsolat is (ld. 3.5. pont). Ezzel az élet már teljes mértékben modellezhető, a környezetünk egyedtípusai között fennálló kapcsolatokat ilyen módon mindig le tudjuk írni. Magyarországon a legnépszerűbb hálós rendszer a Culliname Corporation által forgalmazott Integrated Data Management System (IDMS) volt. Mind a hálós, mind a hierarchikus modellben bizonyos alaprekordokat közvetlenül is megkaphattunk, de az azonos logikai szerkezetbe tartozó rekordokat az azokba beépített pointerok segítségével érhattuk csak el. Így például egy szállítólevél fejrészét azonnal, közvetlen eléréssel beolvashattuk, de a hozzátartozó egyedi megrendelés-tételeken csak a rájuk felépített láncon, az úgynevezett *set*-en keresztül mehettünk végig. Az ilyen fajta feldolgozás csak akkor lehet hatékony, ha jól ismerjük az adatbázis szerkezetét, tudjuk, miként lehet és kell az adatok között navigálni.

¹ CASE Tool (ejtsd: kéztul) = Computer Aided Systems Engineering Tool, Számítógéppel támogatott rendszerfejlesztő eszköz.

A személyi számítógépek elterjedése alapjaiban változtatta meg a számítástechnikai alkalmazásokat, így az adatbázisokkal kapcsolatos munkákat is. Az újabb adatmodellek kialakítását és bevezetését elsősorban az indokolta, hogy

- egyre bonyolultabb adatszerkezeteket kellett kezelni,
- egyre nőtt azoknak a felhasználóknak a száma, akik dolgozni akartak az adatbázisokkal, de nem, vagy alig volt számítástechnikai/informatikai képzettségük,
- az alkalmazási rendszerek kifejlesztését és karbantartását egyszerűbbé, gyorsabbá és olcsóbbá akarták tenni.

Ezeknek az elvárásoknak felelt meg a *relációs adatmodell*. Ennek elvi alapjait F. F. Codd már az 1970-es évek elején kidolgozta, de technikai okokból csak a '80-as években terjedtek el a piacon. Mivel napjainkban a relációs modellre épülő relációs adatbázis-kezelő rendszerek illetve azok különböző kibővítései dominálnak, ezekkel a 4. fejezetben külön foglalkozunk.

Az elmúlt évezred utolsó évtizedében egyre nagyobb jelentőségre tettek szert az adattárházak (ld. 9. fejezet), az internet alkalmazások (ld. 6.2. alfejezet) és elterjedt a kliens-szerver architektúra (ld. 1.6.2. pont). Az adatbázisokban nem csupán strukturált adatokat, hanem különféle objektumokat (álló és mozgó képek, hanganyagok, komplex dokumentumok) is tárolni kell. Ezért megjelentek az objektumorientált adatbázis-kezelő rendszerek is, melyek mindezek feldolgozását lehetővé teszik.

Természetesen az egyre hatékonyabban használt adatbázisok mellett megmaradtak azért a hagyományos fájl-kezelő rendszerek is. Részben speciális adatbázis szolgáltatásokat nyújtanak (pl. adatok kimentése), részben különféle egyedi problémák, főleg egy felhasználós személyi számítógépes feladatok megoldására szolgálnak. (Pl. Windows Intéző, Norton Commander).

A könyv további részében dominanciája miatt csak a relációs adatbázis-kezelő rendszerrel foglalkozunk.

3.3. Főbb modellezési szempontok

A jó adatmodell a valós világnak a feladat megoldásához szükséges részét, és csak azt ábrázolja. Az ehhez szükséges kiválasztást és egyszerűsítést úgy kell megvalósítanunk, hogy a modell tartalmazza az összes szükséges adatot és információt, de áttekinthető és számítástechnikailag kezelhető maradjon. Egyszerű példa erre egy vállalat készpénznyilvántartása. A teljes készpénz mennyiséget mindenképpen felvesszük a modellbe. De azt, hogy ez hány 20000, 10000. ..., 5, 2 és 1 forintból áll, akkor érdemes bevenni, ha az egyes részlegeknél készpénzben fizetik ki a dolgozókat, hogy mindenkinek be tudjuk tenni a fizetési borítékjába a pontos összeget. Hogy a teljes modell miként és meddig egyszerűsíthető, igen nagy mértékben attól függ, mire akarjuk használni.

Az egyszerűség és az áttekinthetőség mellett figyelembe kell vennünk azt is, hogy a feladatok és az igények is változnak. Olyan adatmodellt kell kialakítanunk, melybe az új igényeket is könnyen be lehet építeni, úgy lehessen bővíteni, módosítani, hogy a korábbi feladatokat a változtatás után is lehetőleg ugyanúgy hajthassuk végre. Emellett olyan szerkezetet válasszunk, hogy az adatbázis használatakor az esetleges emberi hibákat minél nagyobb valószínűséggel automatikusan kikorrigáljuk, vagy legalábbis megakadályozzuk.

Bár elvileg nem tartozik az adatmodellbe, a gyakorlatban mégis célszerű figyelembe venni a különböző korlátokat. Az adatok mennyiségét, tárolható részletességét és az adatbázis komplexitását mindenképpen behatárolja a rendelkezésünkre álló szoftver és hardver. Felléphet még ezen kívül korlátozó tényezőként a válaszadási idő is. Túl sok összefüggést figyelembevevő, fölöslegesen részletezett adatbázisban ez olyan nagy lehet, hogy a

gyakorlatban már semmit sem ér a tökéletes pontosságú információ. Mire megkapjuk, már elavult.

Figyelembe kell vennünk a modell kialakításánál azt is, hogy csak az aktuális adatokra van-e szükségünk, vagy rendszeresen nyomon kell követnünk azok időbeli változásait is. Annak ellenére, hogy a fejezet elején ennek ellenkezőjét állítottuk, egy jó adatmodell felépítésénél az általános elvi szempontok mellett gondolnunk kell arra is, hogy végül miként fogják használni az adatbázist, hogy az adatmodellel egy hatékonyan működtethető adatbázist lehessen felépíteni.

3.4. Egyed típusok és tulajdonságok

Ahhoz, hogy az adatbázisból információkat nyerhessünk, egyértelműen meg kell határoznunk azt, ahonnan ezt megkaphatjuk. Ez három dolgot jelent:

- Megkülönböztetünk alapadatokat szolgáltató egyed típusokat, más néven entitásokat. Az egyed típus a valós világ olyan elemeinek az összessége, melyek valamilyen közös tulajdonsággal, vagy tulajdonságokkal, jellemzőkkel rendelkeznek.¹ Az egyed típus meghatározása azt jelenti, hogy meghatározzuk, személyekről, egyetemi hallgatókról, gépjárművekről, vagy könyvekről van-e szó. A modellben minden egyed típusra egy a tartalmára utaló egyértelmű névvel hivatkozunk. (Pl. SZEMÉLY, HALLGATÓ, GÉPJÁRMŰ, JÁRMŰ, KÖNYV, OLVASNI_VALÓ). A gyakorlatban erre egyes szám alanyesetben álló főneveket használunk.
- Minden egyed típusoknak rá jellemző *tulajdonságai*, más szóval kifejezve *attribútumai* vannak. Az adott alkalmazástól függ, hogy ezekből miket veszünk be az adatmodellbe. Egy egyetemi személyi nyilvántartásba a SZEMÉLY tulajdonságai között valószínűleg nem fog szerepelni sem a testsúlya, sem a magassága. Ugyanezek az attribútumok viszont nyilvánvalóan szükségesek a körzeti orvos adatbázisában lévő SZEMÉLY egyed típusban. *Minden egyed típus attribútumokból tevődik össze. Az attribútumok az egyed típusoknak azok a jellemzői, melyeknek értékei szükségesek ahhoz, hogy az egyed típusról a kívánt információkat megkaphassuk.* Ökölszabály! Azt, hogy mik lehetnek első kiindulásként az entitásaink és azok tulajdonságai, úgy kaphatjuk meg, hogy végigolvassuk a feladat leírását és kiválasztjuk azokból a főneveket. Ebből általában józan paraszti ésszel kiderül, melyik hová sorolandó. A *szín* például egy gépjármű nyilvántartásban a GÉPJÁRMŰ entitás egy tulajdonsága lesz, míg egy festékgárban, ahol a különböző színű festékek komponenseit adjuk meg, ott SZÍN egy egyed típus lesz, melynek attribútumai a kikeveréséhez szükséges összetevők neve és aránya lesz. A modellben a tulajdonságokra is egy tartalmukra jellemző rövid névvel hivatkozunk. Ha ugyanaz a tulajdonság több entitásban is szerepel, például a gépkocsi rendszáma (RENDSZÁM) tulajdonsága a gépjárműveket nyilvántartó GÉPJÁRMŰ, és a szabálytalankodó gépkocsikról vezetett SZABÁLYSÉRTÉS entitásnak is, akkor mindenütt ugyanúgy hívjuk, és ha nem egyértelmű, akkor adjuk meg, hogy éppen melyik entitás tulajdonságáról beszélünk.
- Különbséget kell tennünk az egyed típus és annak konkrét *előfordulásai* között. *Egy előfordulás az egyed típus konkrét tulajdonságértékekkel tárolt/tárolandó realizálása az adatbázisban.* Minden egyed típusnak nullától (elvileg) végtelen sok előfordulása lehet az adatbázisban. A felső határt az egyed típus számára kijelölt

¹ Mint azt a 4. fejezetben látni fogjuk, a relációs modellben a valós életben meglévő egyed típusok mellett a köztük fennálló kapcsolatokat is egyed típusként ábrázolhatjuk.

tároló terület szabja meg.

Az egyedtípus és az egyedtípus előfordulásai közti különbséget a 3.1 ábrán láthatjuk. A SZEMÉLY egyedtípus egyszer szerepel az adatbázisban. Definícióját, nevét, tulajdonságait adatokat definiáló metaadatként az adatbázis katalógusában (ld. 7. fejezet) tároljuk. Az előfordulásai, az egyes személyek adatai a relációs adatbázisban a SZEMÉLY táblázat egyes sorai.

EGYEDTÍPUS

EGYED ELŐFORDULÁSOK

SZEMÉLY	SZEMÉLY	SZEMÉLY	...
<u>Taj_szá</u>	<u>007611979</u>	<u>008123456</u>	...
Név	Kiss Ferenc	Nagy Ede	...
Szdátum	1990-01-01	1988-02-12	...
Szhely	Budapest	Eger	...
Lakcím_utca	Hold u. 2.	Nap tér 3.	...
Lakcím_város	Budapest	Budapest	...
Irszám	1022	1022	...
Testsúly	99	69	...
Magasság	193	161	...
.	.	.	.
.	.	.	.
.	.	.	.

3.1. ábra: Egyedtípus és egyed előfordulás közti különbség.

- Végül azt is meg kell határoznunk, hogy egy egyedtípus melyik előfordulásával kívánunk dolgozni. Ehhez minden egyedtípushoz hozzárendelünk egy azonosítót. *Az egyedtípusnak azt a tulajdonságát vagy tulajdonság kombinációját, amelyik egyértelműen meghatározza az egyed előfordulást, azaz az egyedtípus minden előfordulására különböző értéket vesz fel, nevezzük az egyedtípus azonosítójának.* Viszonylag ritkán fordul elő, hogy egy egyedtípusnak van egy olyan természetes tulajdonsága, amely bármely elemét egyértelműen tudja azonosítani. Ha van, akkor célszerű ezt azonosításra használni. Ha nincs, akkor vagy egy mesterséges azonosítót vezetünk be (Pl. Adószám, TAJ szám, Rendszám, bár ez utóbbit már szinte természetes azonosítónak érezzük), vagy megpróbáljuk a tulajdonságoknak egy olyan kombinációját kialakítani, melyek együttesen már egyértelműen azonosítani tudják az egyedtípus előfordulásait. (Pl. Név, Anyja neve, Születési hely és idő a személyek azonosítására). Ezzel a kérdéssel a relációs modellben az elsődleges kulcs meghatározásánál (ld. 4.4.2. pont) még bővebben foglalkozunk.

3.5. Kapcsolatok

Minden mindennel összefügg, minden összefügg valamivel. Ha a fenti elvet az adatmodell kialakításánál is alkalmazni kívánnánk, akkor a feladatunk megoldása elég reménytelen lenne. A gyakorlatban ésszerű kompromisszumot kötünk, hogy az egyedtípusok között elképzelhető, elvileg végtelen sok kapcsolat közül melyeket építjük be a modellünkbe.

Az egyes egyedtípusok között a következő alapvető kapcsolatok létezhetnek:

1. *A két egyedtípus független egymástól.* Ez a legegyszerűbb. Ilyenkor az adatmodellben sem kell kapcsolatot létesíteni közöttük. Függetlenek lehetnek például a hallgatók tanulmányi eredményei és a bankszámlák forgalmi adatai.
2. *A két egyedtípus között kölcsönösen egyértelmű kapcsolat áll fenn.* Ezt *egy-egy*, vagy *1:1 kapcsolatnak* nevezzük.
Ilyen kapcsolat áll fenn magyar állampolgároknál a TAJ számhoz, vagy a személyi számhoz vagy az adószámhoz tartozó személyes adatok között¹. Ugyancsak egyértelmű a kapcsolat egy jól tervezett vállalati vagy oktatási modellben a részlegek illetve tantárgyak megnevezése és kódja között.
3. *Az első egyedtípus bármely előfordulásához a másik egyedtípus több, tetszősszerűen számú előfordulása (ebbe a zérus is beleértendő) tartozhat, míg a másodiknak bármelyik előfordulása egyértelműen meghatározza, hogy az elsőből melyik kapcsolódik hozzá.* Ezt nevezzük *egy-több* vagy *1:N kapcsolatnak*.
Tipikus példa erre a szülőanya és a gyerekei közti kapcsolat. Egy anyának akárhány gyereke lehet, míg bármely gyereknek csak egy anyja van. Ezek a kapcsolatok egymásra is épülhetnek. A lány gyerekeknek is lehetnek további gyerekei (lányaik), akiknek ismét lehetnek gyerekei. Ezt a kapcsolatot láncot nevezik hierarchiának.
4. *Az első egyedtípus bármely előfordulásához a másik egyedtípus több előfordulása (ebbe a zérus is beleértendő) tartozhat, és a második bármely előfordulásához is az elsőnek több előfordulása kapcsolódhat.* Ezt nevezzük *több-több* vagy *N:M kapcsolatnak*.
Ez a legáltalánosabb kapcsolattípus. Ez jellemzi egy emancipálódott társadalomban a férfiak és a nők viszonyát. Egy férfi több barátnőt is tarthat, és egy nőnek is több barátja lehet². További példa egy oktatási adatmodellben a hallgatók és a tantárgyak kapcsolata. Egy hallgató több tantárgyat is felvesz, és egy tantárgyat többen hallgathatnak.
A több-több kapcsolattal az adatbázis-kezelő rendszerek közvetlenül nem tudnak dolgozni. Ezért ezeket a megvalósítás alapjául szolgáló végleges modellben fel kell bontanunk egy-több kapcsolatokra³. Ennek algoritmusát a 3.7. alfejezetben mutatjuk be.

Az egyes kapcsolattípusokat a 3.2. ábrán mutatjuk be.

3.6. Egyedtípus – kapcsolat (Entity – Relationship) diagram

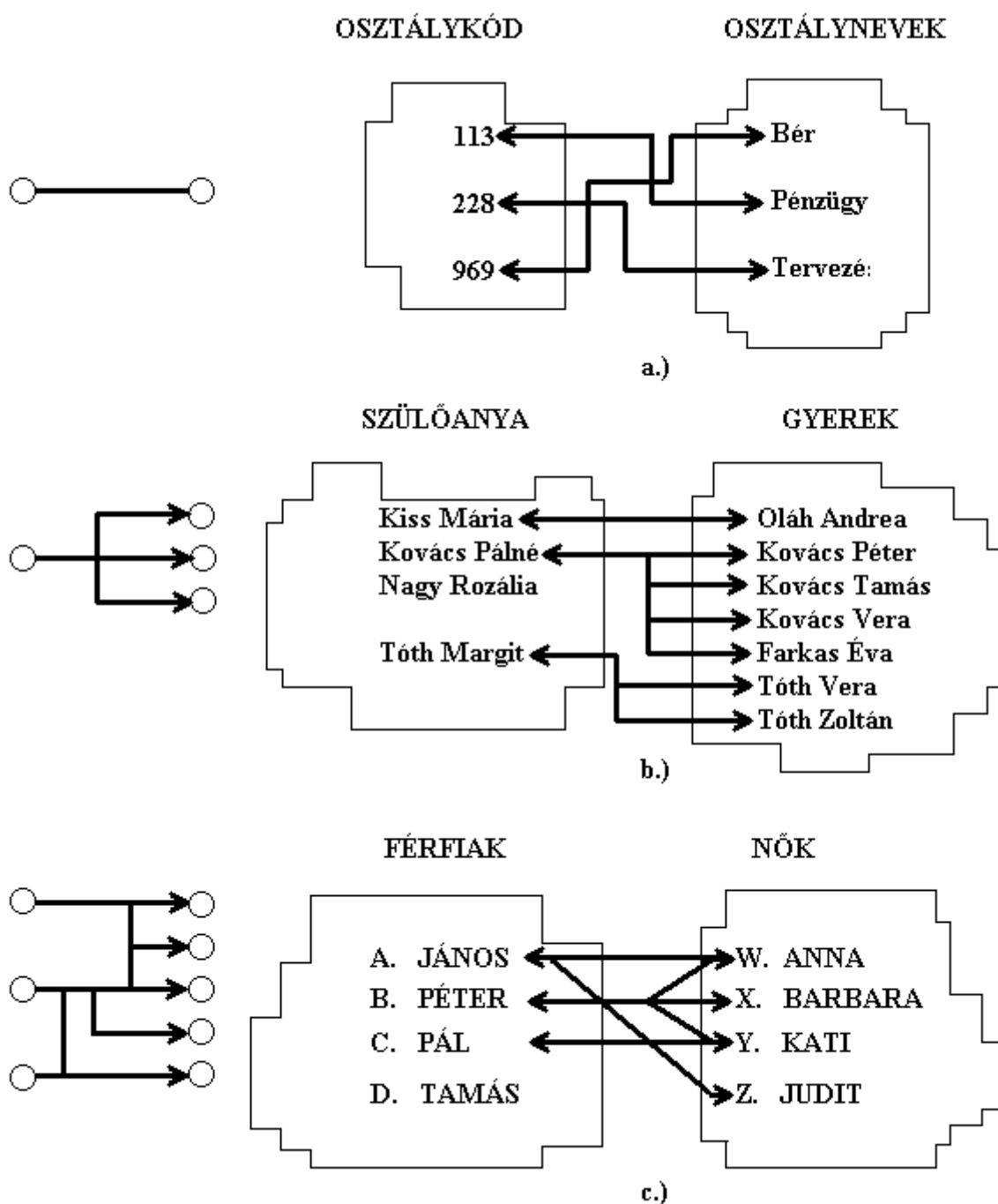
Az egyedtípus – kapcsolat (Entity – Relationship) modell egy szervezet vagy egy alkalmazási terület adatainak részletezett logikai leírása. A modellt rendszerint grafikus formában, az *egyedtípus – kapcsolat*, vagy a magyarban is elfogadott angol nevén *Entity – Relationship diagramban* jelenítjük meg. A diagramban alkalmazott jelölések esetenként eltérőek lehetnek. Mi az egyik leggyakrabban használt formát használjuk. Ebben az

¹ Ezt tovább finomíthatnánk. TAJ- és személyi száma mindenkinek van, így ezek között a kapcsolat mindig, mindenképpen fennáll. Adószáma viszont nincs mindenkinek. Ezért az ezzel megvalósított kapcsolat csak akkor érvényes, ha létezik adószám.

² Tovább finomíthatnánk a kapcsolattípusokat az egyedtípuson belüli kapcsolatokkal is. Ha a férfiak és a nők egyedtípusok helyett csak egy egyedtípussal, a személlyel dolgoznánk, akkor e fenti kapcsolatok ugyanazon egyedtípus elemei között állnának fenn. Ilyen módon modellünkkel tudnánk kezelni a hetero- és a homoszexuális kapcsolatokat is.

³ Egyes adatbázis tervező eszközökben megadhatunk több – több kapcsolatokat is, és ezeket a tervező szoftver bontja szét egy – több kapcsolatokra.

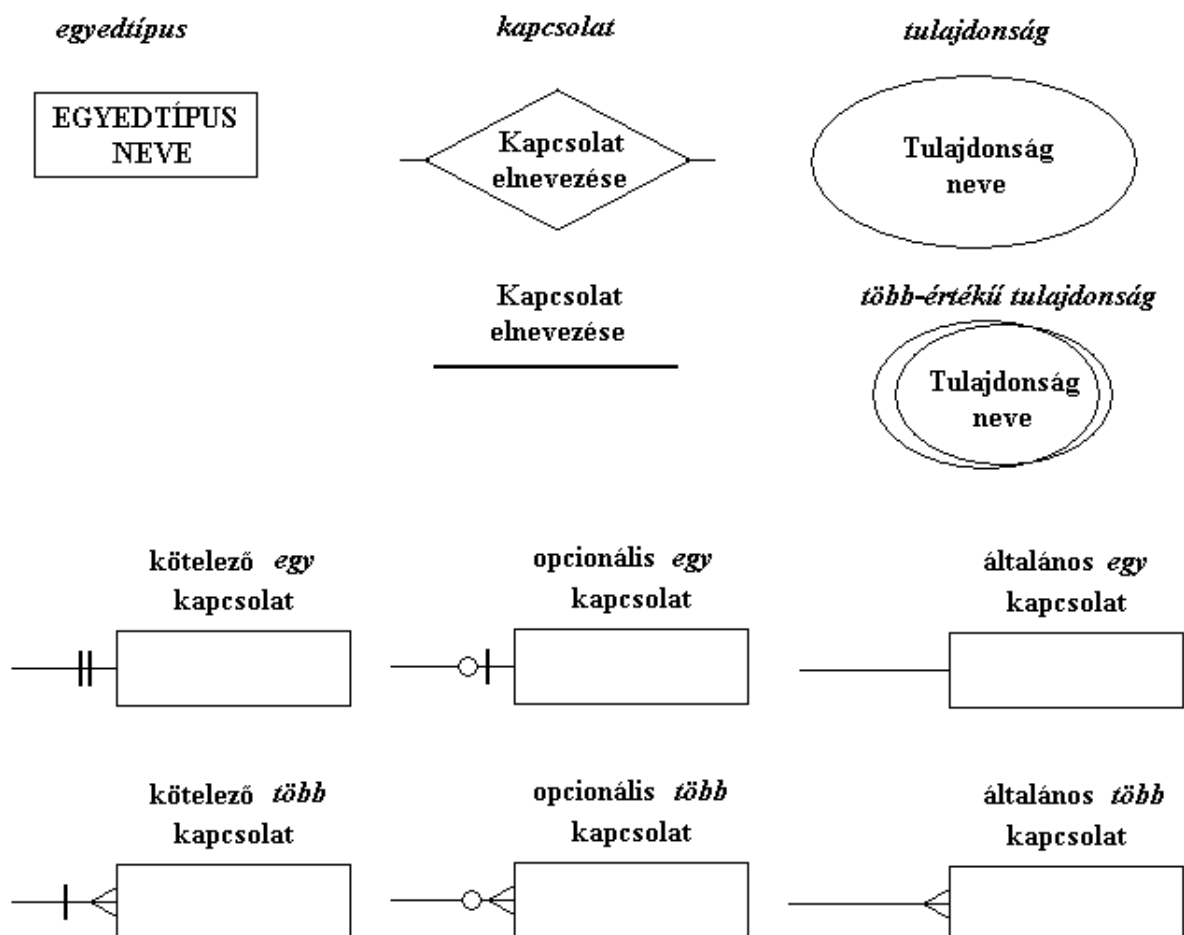
entitásokat vízszintesen álló téglalapokkal, az attribútumokat ellipszisekkel, az entításokat összekötő kapcsolatokat vonallal (és esetleg középük elhelyezett csúcsára állított romboiddal) jelöljük.



3.2. ábra: Kapcsolattípusok. a.) 1:1, b.) 1:N, c.) N:M kapcsolat.

Ha egy tulajdonság több értéket is felvehet az adott entitás egy előfordulásában, akkor kettős ellipszissel ábrázoljuk. Az entítások nevét általában nagy betűkkel, a tulajdonságokét és a kapcsolatokét nagy kezdőbetűvel és kis betűkkel a megfelelő idomba írjuk. Az azonosító

tulajdonság nevét vastagítva vagy aláhúzva adjuk meg¹. Az áttekinthetőség kedvéért a tulajdonságokat gyakran nem tüntetjük fel az ábrán, hanem egyed típusonként külön adjuk meg egy-egy önálló táblában. Ugyancsak elhagyhatjuk a kapcsolatok romboidjait, és a kapcsolat nevét az összekötő vonalra írjuk. A kapcsolatok *több-értékű* oldalán a vonalat három ágra bontjuk. Lehetséges még a kapcsolatok kötelező vagy opcionális volta is utalni.² A különböző jelöléseket a 3.3. ábrán foglaltuk össze.



3.3. ábra: Az Entity – Relationship modell szokásos jelölései.

A 3.4. ábrán egy kereskedő cég beszerzéseinek és megrendeléseinek nagymértékben leegyszerűsített Entity – Relationship diagramját látjuk.

Néhány megjegyzés a diagramhoz:

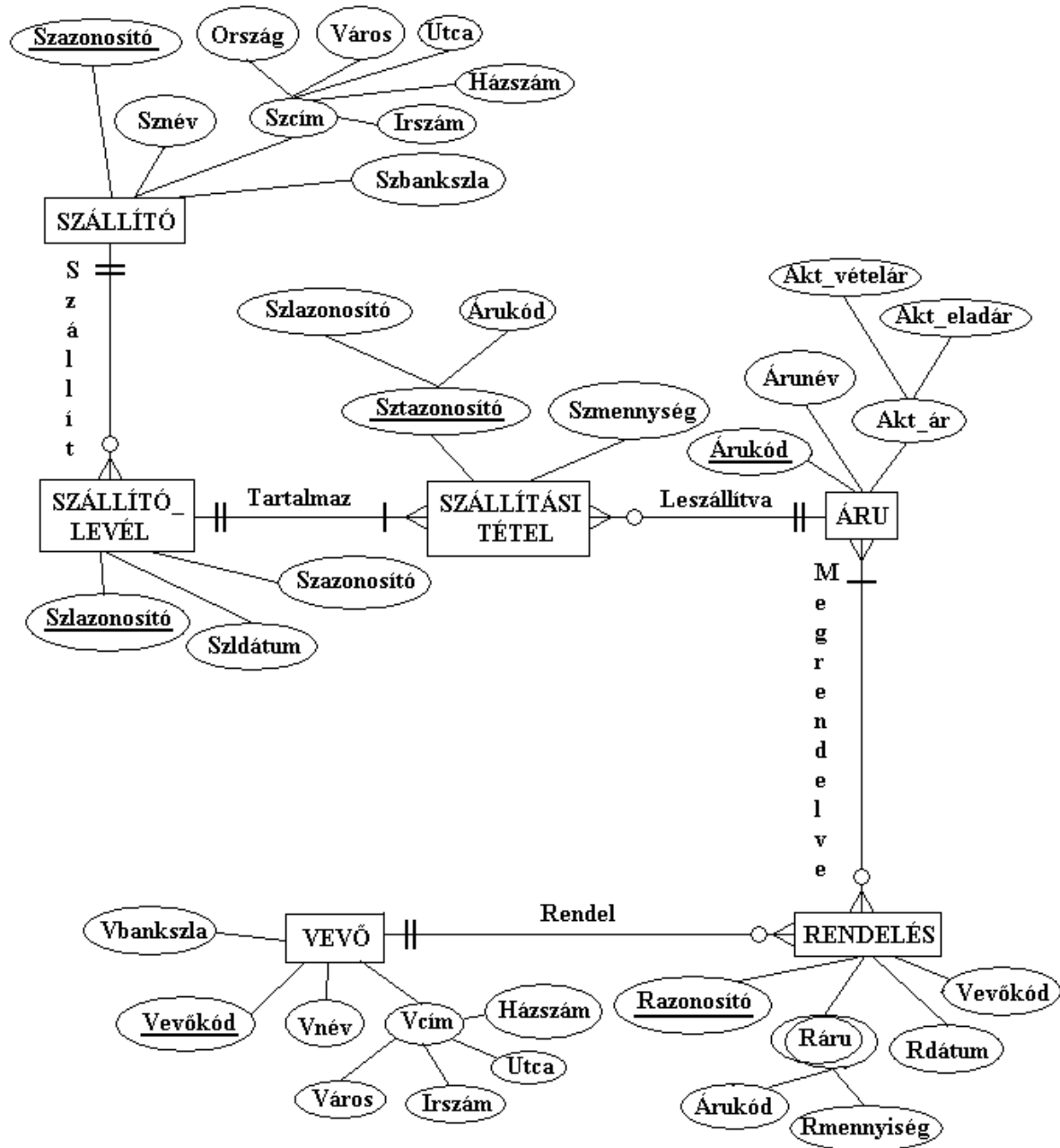
- A modellnek a szállításokat illetve a megrendeléseket tartalmazó részét szándékosan ábrázoltuk különböző módon. A szállításoknál feloldottuk a SZÁLLÍTÓ és ÁRU közti N:M kapcsolatot, míg a megrendeléseknél ezt meghagytuk.³

¹ Ezt a konvenciót csak az adatmodelleknél alkalmazzuk konzekvensen. Az SQL utasításokban tetszés szerint használhatunk kis- és nagybetűket a különböző nevekben. Az adatbázis katalógusában pedig minden név nagybetűsen szerepel.

² Kötelező kapcsolatnál mindenképpen kell legalább egy egyednek létezni a kapcsolat kötelezőként megadott oldalán (Pl. minden szállítás-tételhez kell egy árunak kapcsolódni), míg opcionálisnál ez nem áll fenn (nem minden áruhoz létezik megrendelés).

³ Természetesen, ha itt nem tettük meg, akkor később, az adatbázis létrehozására is alkalmas relációs modell elkészítésekor kell átalakítanunk az N:M kapcsolatot. (ld. az Ellenőrző kérdések 4. feladatát)

- Feltételeztük, hogy minden árunak mindenhol ugyanaz a beszerzési, és mindenhol ugyanaz az eladási ára. Az első, nem igazán életszerű feltétel elhagyására oldjuk meg a 3.8. alfejezet 5. feladatát.
- A szállítólevélben kötelező egy szállítót és legalább egy szállítandó tételt megadni.



3.4. ábra: Egy kereskedő cég beszerzéseinek és szállításainak nagy mértékben leegyszerűsített Entity – Relationship diagramja.

- Nem minden árura van megrendelésünk és lehet olyan is, amelyikre nem kértünk szállítást. (Pl. korábbi készletből megmaradt).
- A rendelést mindig egy vevő adja fel, és legalább egy árut meg kell benne rendelnie.
- Lehetnek olyan vevők, akik még egy rendelést sem adtak fel. (Potenciális vevőként nyilvántartjuk őket).

- A VEVŐ és az ÁRU közötti második kapcsolatot, amelyik a vevő által megrendelt áruk leszállítását írja le, nem vettük be az adatmodellbe. (ld. az Ellenőrző kérdések 7. feladatát.)
- Némi inkonzisztencia van az elnevezésekben, de már nem akarunk/tudunk további módosításokat végrehajtani. (Ez még gyakorlott adatbázis tervezőknél is előfordulhat).
 - Az egyedtípus meghatározó tulajdonságát hol *azonosító*nak (Pl. *Szazonosító*), hol *kódnak* (Pl. *Vevőkód*) nevezzük. Ez azonban, miután mindig tudjuk, miről van szó, soha semmiféle problémát nem fog okozni. Ezért nem törődünk vele.
 - A szállító és a vevő címére más neveket adtunk meg (*szcím*, *vcím*), de a cím részletes felbontásában ugyanazokat az elnevezéseket (*Város*, *Utca*, ...) használjuk. Az adatbázissal való munkában ez nem okoz problémát, mert az adatbázis-kezelő rendszer (az SQL nyelv szabályai szerint) rákényszerít bennünket, hogy tegyük egyértelművé, melyik cím-komponenssel dolgozunk. A modelltől való tárgyalásoknál azonban esetleg félreértést okozhat. Reméljük, nem fog előfordulni. Ha mégis, pontosítunk a szóhasználatban.
 - A vevő részletezett címénél nem adtunk meg *országot*, mert csak belföldre szállítunk. Amennyiben ez megváltozna, akkor az adatmodellt módosítjuk.

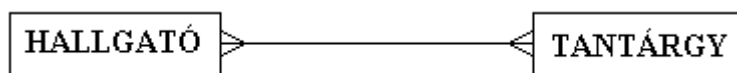
3.7. Példa adatmodell létrehozására

A fentiek megértésére és gyakorlására létrehozuk egy nagymértékben leegyszerűsített *egyetemi nyilvántartás adatmodelljét*. Ezen az adatmodell megalkotásának összes fontos lépését be tudjuk mutatni. A modellnek a valóságot jobban figyelembe vevő bővítését az Ellenőrző kérdések 1. feladata alapján az olvasóra bizzuk.

Első lépésünk a modell egyedtípusainak, entitásainak meghatározása lesz. Nem kell túl nagy képzelőerő ahhoz, hogy első közelítésben ezek az alábbiak legyenek:

- hallgatók
- tantárgyak
- oktatók
- tantermek
- tankönyvek
- tanszékek

Feladatunkat korlátozzuk, hogy a modellben csak a hallgatókkal és a tantárgyakkal törődünk. E két egyedtípus között több-több kapcsolat áll fenn. Egy tantárgyat többen, akárhányan hallgathatnak, és egy hallgató több, elvileg akárhány tantárgyat felvehet. Az ezt leíró adatmodell a 3.5. ábrán látható.



3.5. ábra: Leegyszerűsített egyetemi adatmodell első változata.

Innentől kezdve két módon mehetünk tovább.

1. „Ráérzünk”, hogy a HALLGATÓ és a TANTÁRGY között úgy tudunk kapcsolatot létrehozni, hogy beiktatunk egy új entitást, ami azt tartalmazza, hogy melyik hallgató milyen körülményekkel, milyen tantárgyakat vett fel. Ennek a MITHALLGAT entitásnak az előfordulásai tartalmazzák egy hallgatónak egy tantárggyal kapcsolatos adatait. Az így kapott adatmodellt a 3.6. ábrán láthatjuk.



3.6. ábra: Leegyszerűsített egyetemi adatmodell végső változata.

Végül meghatározzuk az egyes egyed típusok attribútumait (tulajdonságait) és azonosítójukat. Erre ismét több lehetőségünk van. Az egyik véglet az, hogy minden egyed típusnál minden hozzákapcsolható tulajdonságot felsorolunk. Ez entitásonként a következő attribútumokat és elnevezéseket (zárójelben megadva) jelenti¹:

HALLGATÓ (első verzió)

Hallgató-azonosítója (mesterséges, ez azonosítja az egyed típus előfordulásait) (**HID**),

Hallgató vezetéckneve (HVNÉV),

Hallgató keresztnéve (HKNÉV),

Hallgató évfolyama (HÉVF),

Hallgató születési dátuma (HSZDÁTUM),

Minden egyes felvett tantárgyának a tantárgyra jellemző adatai (TANTÁRGY első verziója szerint)

TANTÁRGY (első verzió)

Tantárgy azonosítója (mesterséges, ez azonosítja az egyed típus előfordulásait) (**TID**),

Tantárgy megnevezése (TMEGNEV),

Tantárgy oktatóinak adatai (ONÉV),

Tantárgy óraszám (ÓRASZÁM),

Vizsgakötelezettség (VIZSGA),

Minden egyes, a tantárgyat felvett hallgatónak a személyi adatai (HALLGATÓ első verziója szerint)

MITHALLGAT (első verzió)

A hallgatót és a tantárgyat összekapcsoló azonosító (mesterséges) (**MHID**),

Tantárgy jellemző adatai (TANTÁRGY első verziója szerint),

Minden egyes, a tantárgyat felvett hallgató személyes adatai (HALLGATÓ első verziója szerint),

Az egyes hallgatóknak a tantárggyal kapcsolatos adatai:

Hallgató gyakorlatokon elért pontszáma a tantárgyból (GYAKPONT),

Hallgató osztályzata a tantárgyból (OSZTÁLYZAT),

Tantárgy felvételének féléve (FÉLÉV),

Vizsga időpontja VIZSGA)

Bár a modell ezekkel az egyed típus-definíciókkal elvileg már használható, gyakorlatilag nagyon rossznak bizonyulna. Ennek ellenére, ha tovább finomítanánk a 4.6.

¹ A modellünk egyszerűsítése miatt a személyi adatokból, és a tantárgy jellemzőkből csak a felsoroltakra lesz szükségünk.

alfejezetben ismertetett normalizálási eljárással, végül ugyanarra az eredményre jutnánk, mint amit a b.) eljárással kapunk, amikor minden egyedtípusba csak azokat az attribútumokat vesszük be, melyek kizárólag az egyedtípus azonosítójától függenek és az egyedtípusoknak az így kapott első változatát javítjuk normalizálással.

HALLGATÓ

Hallgató-azonosítója (mesterséges, ez azonosítja az egyedtípus előfordulásait) (**HID**),

Hallgató vezetéckneve (HVNÉV),

Hallgató keresztnéve (HKNÉV),

Hallgató évfolyama (ÉVF),

Hallgató születési dátuma (HSZDÁTUM)

TANTÁRGY

Tantárgy azonosítója (mesterséges, ez azonosítja az egyedtípus előfordulásait) (**TID**),

Tantárgy megnevezése (TMEGNEV),

Tantárgy oktatóinak adatai (ld. pontosítását lejjebb),

Tantárgy óraszám (ÓRASZÁM),

Vizsga kötelezettség (VIZSGA)

MITHALLGAT (második verzió)

A hallgatót és a tantárgyat összekapcsoló azonosító (mesterséges) (**MHTID**),

Hallgató azonosítója (HID),

Tantárgy azonosítója (TID),

Hallgató gyakorlatokon elért pontszáma a tantárgyból (GYAKPONT),

Hallgató osztályzata a tantárgyból (OSZTÁLYZAT),

Tantárgy felvételének féléve (FÉLÉV)

Vizsga időpontja (VDÁTUM)

Tovább egyszerűsítjük a modellünket.

- Egy hallgató egy tantárgyat csak egyszer vehet fel. Az ezzel kapcsolatos adatokból csak a legutolsó, az aktuális értéke érdekel bennünket. Ezért MITHALLGAT második verziójában fölösleges a vizsga időpontja és a félév. A hallgató és a tantárgy azonosítója egyértelműen azonosítja a kettő közti kapcsolatot, így erre sem kell egy új, mesterséges azonosítót bevezetnünk. Ezek figyelembevételével kialakul az egyedtípus végleges szerkezete.

MITHALLGAT

Hallgató azonosítója (a tantárgy azonosítójával együtt azonosítja az egyedeket) (**HID**),

Tantárgy azonosítója (**TID**),

Hallgató pontszáma a tantárgyból (GYAKPONT),

Hallgató osztályzata a tantárgyból (OSZTÁLYZAT)

- A tantárgy oktatói közül csak egynek, a tantárgyért felelős oktatónak a nevét vesszük fel az adatmodellbe. Ezért tantárgyban az *oktatók adatai* helyett a *felelős oktató neve* attribútumot vesszük fel ONÉV elnevezéssel.

Modellünket természetesen kialakíthatjuk az entitások első verziójának más, a két véglet közti definiálásával is. Akármelyik formájából is indulunk ki, a gyakorlatban úgymint relációs adatbázist hozunk majd létre. Ehhez az itt leírtak alapján kialakított adatmodellt, ha szükséges, tovább módosítjuk a 4. fejezetben részletesen tárgyalt módon, hogy megfeleljen a relációs modell követelményeinek is.

2. A 3.5. ábrából kiindulva, létrehozuk a HALLGATO és a TANTÁRGY entitásokat úgy, hogy mindkettőhöz hozzáírjuk az összes hozzákapcsolható attribútumot. Ezzel az a.) módszer HALLGATO és TANTÁRGY első verziójaként definiált egyed típusokat kapjuk. Ezekből a 4.6. alfejezetben leírt normalizálási eljárással alakítjuk ki a végleges adatmodellt.

3.8. Ellenőrző kérdések

1. Bővítsük ki a 3.7. alfejezet adatmodelljét
 - Az oktatók személyi adataival. Egy oktató több tantárgyat is oktathat és több tantárgynak is lehet a felelőse.
 - A tanszékekkel. Egy oktató csak egy tanszékhez tartozhat, és minden tantárgyért egy tanszék felelős.
 - A tantervek jellemzőivel. A modell tartalmazza azt is, hogy melyik tantárgyat melyik teremben és mikor tanítják.
 - A tankönyvek leírásával. A modell tartalmazza azt is, hogy melyik tantárgyat milyen könyvekből tanítanak.
 - Egészítsük ki a modellt, hogy nyomon tudjuk követni a hallgatók eredményeit, ha egy tárgyból többször vizsgáztak, és ha egy tárgyat többször vettek fel.
2. Építsük fel egy egyszerűsített gépkocsi-nyilvántartás adatmodelljét!
3. Készítsük el egy egyszerűsített anyakönyvi-hivatal házasság-nyilvántartásának adatmodelljét!
4. Bontsuk fel a 3.4. ábrán látható adatmodellben az ÁRU és a RENDELÉS közti N:M kapcsolatot! Szüntessük meg RENDELÉS-ben a megrendelt áruk leírására szolgáló *Ráru* tulajdonság többértékűségét!
5. Vegyük be a 3.4. ábrán látható adatmodellbe, hogy ugyanazt az árut különböző szállítóktól eltérő árakon szerezhethetjük be!
6. Egészítsük ki a 3.4. ábrán látható adatmodellt, hogy az árak időbeli változását is nyomon tudjuk követni!
7. Egészítsük ki a 3.4. ábrán látható adatmodellt a megrendelt áruk leszállításával úgy, hogy
 - a.) minden megrendelést egyben szállítanak le,
 - b.) az egyes megrendeléseket több részletben is leszállíthatják.

4. RELÁCIÓS ADATBÁZIS-KEZELŐ RENDSZEREK

A relációs modellben az adatokat és a köztük lévő kapcsolatokat *relációkban*, szemléletesen tekintve táblázatokban, táblákban¹ ábrázoljuk. Ezekből a feladatok megoldásához szükséges adatokat a *relációs műveletek* segítségével kapjuk meg.

A táblákban történő ábrázolásmód következtében ez a modell áll legközelebb az átlagos felhasználói szemlélethez. Ezért, a korábbi adatmodellektől eltérően, lényegének megértéséhez nem szükségesek különleges matematikai ismeretek. Használata könnyen megtanulható. Ugyanakkor – elsősorban Codd alapvető munkái következtében – ennek a modellnek dolgozták ki az elméleti alapjait a legrészletesebben és a legpontosabban. További előny még, hogy a fizikai és a logikai adatfüggetlenség elvileg tökéletesen megvalósítható, a modell lehetőséget nyújt az adatok integritásának biztosítására és a konkrét feladatok ismeretében a hatékony feldolgozást elősegítő adatszerkezetek kialakítására is.

Ezen előnyök következtében napjaink adatbázis-kezelő rendszerei szinte kizárólag a relációs modellen alapulnak.² Elterjedésükhöz nagymértékben hozzájárult az is, hogy gyakorlati alkalmazásukra a '80-as évektől egy egységes, szabványosított nyelv, az SQL áll rendelkezésre, melyet az 5. fejezetben ismertetünk részletesen.

4.1. A relációk tulajdonságai

Matematikailag tekintve a relációs modell két-dimenziós, egymással kapcsolatban álló, speciális tulajdonságokkal rendelkező táblákból áll. Ezek a relációk írják le a valós világ különböző egyed típusait, entitásait, sőt sok esetben a köztük lévő kapcsolatokat is. A relációk tartalma felel meg a valós világ különböző egyedeinek, és különböző tulajdonságaik értékeinek.

A reláció legfontosabb jellemzője a *reláció neve*. Ez kapcsolja össze a valós világ valamely entitásával. Minden reláció *oszlopokból* és *sorokból* áll. A sorok jellemzik az adott relációval ábrázolt egyed típusok előfordulásait, *konkrét egyedeit*, az oszlopok az elemek egyes *attribútumait*, *tulajdonságait*. Az oszlopok nevei, a tábla (képzeltbeli) fejléce határozza meg a tulajdonság nevét, adott sorban lévő értéke pedig az azzal a sorral ábrázolt egyed adott tulajdonságának értékét.

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

- Minden relációnak van egy neve, amelyik azt egyértelműen azonosítja az adatmodellben (és ezáltal az adatbázisban is). Jól tervezett adatbázisban ez megegyezik az általa reprezentált egyed típus elnevezésével, vagy utal arra³.
- Egy adott reláció minden sorában azonos számú oszlop van. Az oszlopok számát nevezik a *reláció fokának*, vagy *fokszámának*.
- Minden oszlopnak az adott reláción belül egyértelmű neve van. Bármely oszlop a reláción belül ezzel a névvel, az adatmodellben – mivel ugyanaz az oszlopnév több relációban is előfordulhat – a reláció és az oszlop nevével együttesen azonosítható egyértelműen. A jól tervezett adatbázisban az oszlop neve megegyezik az általa reprezentált tulajdonság elnevezésével, vagy utal rá.

¹ Mivel a relációs modell alkalmazásánál az SQL nyelvben a *táblázatnak* (*TABLE*) speciális jelentése van, ezért az általános reláció szinonimájaként ebben a részben a *tábla* kifejezést fogjuk használni.

² Újabban megjelentek az objektum-orientált relációs adatbázis-kezelő rendszerek is, melyek azonban alapjukban a relációs rendszerek kiterjesztésének tekinthetők.

³ Ezt a konvenciót alkalmazzuk majd a 4.2 pontban bemutatott minta adatmodellben és a többi példánkban is mind a relációk, mind az oszlopok nevére.

- Minden oszlop csak meghatározott értékeket vehet fel (ez lehet egy folytonos tartományból is). Ezen értékek összességét, az értékészletet nevezzük az adott oszlophoz tartozó attribútum-értékek tartományának, *domainnak*. A tartomány a modell szempontjából mindig azonos típusú elemi adatokat tartalmaz. Ezt nevezzük az oszlop adattípusának. Ez mindig alkalmazásfüggő.
- A relációs adatmodellben leggyakrabban használt alapvető adattípusok
 - karakter,
 - szám,
 - dátum/idő.

Ezeket, és ezek további bontását az 5.4. alfejezetben, az SQL nyelv segítségével történő meghatározásuknál ismertetjük részletesen.

Az adattípus kiválasztásánál a legfőbb szempontok:

- tárolni tudja az összes előforduló értéket (nem csak akkor, amikor indul a rendszer, hanem az adatok mennyiségének, hosszának nagymértékű növekedése után is),
- a tárolóhely minimalizálása,
- tegye lehetővé az adatok integritásának az ellenőrzését,
- az adatokkal kapcsolatos összes műveletet elvégezhetünk benne,
- használatakor ne kelljen konvertálni, vagy ez csak minimális időt követeljen.
- Bármely sorban minden oszlop csak egyetlen értéket vehet fel a megengedett értékek közül. Ez azt jelenti, hogy az oszlopok és értékeik elemiek, *nem tartalmazhatnak ismétlődő mezőt*.
- A fentiek alapján a reláció minden sorában minden egyes oszlophoz egy, és pontosan csak egy értéket kell hozzárendelnünk. A gyakorlatban azonban igen sokszor előfordul, hogy bizonyos oszlopoknak az adatrekordnak (sor) az adatbázisba való bevitelkor még nem ismerjük az értékét, vagy nincs is értéke, de a rekordot mégis tárolni akarjuk (pl. egy személyi nyilvántartásban a személy telefonszáma, vagy ideiglenes lakcíme). Ennek a problémának a kiküszöbölésére két lehetőségünk van.
 - Az oszlopnak automatikus *alapértelmezést* (default) adunk, és ha a bevitelkor nem adunk meg explicit értéket, akkor ezt az értéket veszi fel.
 - Bevezetünk egy speciális értéket, a *NULL értéket*, ami azt jelenti, hogy az oszlop tartalma *nem ismert*. Ez nem tévesztendő össze a 0 értékkel, hiszen nem mindegy, hogy valaki önkéntesként 0 forint fizetésért ingyen dolgozik, vagy még nem egyeztek meg a díjazásában. Mivel a NULL érték megengedése bizonyos bizonytalanságot jelent, meg kell határoznunk, mely oszlopok tartalma vehet fel NULL értéket, és melyekben nem engedjük ezt meg (pl. egy személyi nyilvántartásban az illető neve).
Ha NULL értéket megengedő oszlopnak nem adunk konkrét értéket, akkor annak tartalma automatikusan a NULL érték lesz.
- Az oszlopok sorrendjének nincs jelentősége az adatmodellben. De ha egyszer, a tábla „fejlécében” megadtuk, akkor ezen relációnak minden sorában ennek a sorrendnek megfelelően kell az attribútumok értékét tárolnunk¹.
- Egy reláció sorainak számát nevezzük a reláció *számoosságának*, vagy *kardinalitásának*. Ez az adatbázis használatakor, új adatok bevitelkor, meglévők

¹ A balról jobbra való olvasás és képernyőn való megjelenítés miatt megszokott, hogy a "fontosabb" oszlopokat helyezzük balra, előre, a kevésbé fontosakat hátrébb. Ez azonban csak egy kényelmi elrendezés, melynek a modellt tekintve nincsen tartalmi jelentősége.

törlésével dinamikusan változik. A sorok sorrendje a relációs modell szempontjából közömbös, a relációk sorainak nincsen kitüntetett rendezettsége.¹

- Mivel a reláció egyedei a tábla egyes sorai, melyeknek azonosítása a sorok tartalma alapján történik, egy relációnak nem lehet két teljesen azonos sora, két olyan sora, melyben páronként minden oszlop értéke megegyezik. Ez a feltétel a valós világnak azt a reális megkötöttségét tartalmazza, hogy ha két egyede között semmilyen módon nem tudunk semmiféle különbséget tenni, akkor azokat nem tekintjük különbözőnek. Jellemzésük tulajdonságaik megadásával egyértelmű, és legfeljebb – ha szükséges – új tulajdonságként megadhatjuk, hogy hány darab van összesen belőlük.
- Az előző feltételből következik, hogy minden relációban van az oszlopoknak legalább egy olyan kombinációja, amely egyértelműen azonosítja a sort, meghatározza a sor további oszlopainak az értékét. Ezt az oszlopot (oszlop kombinációt, vagy ezek egyikét), ami megfelel az általános adatmodellben az egyedtípus azonosítójának, nevezzük *elsődleges kulcsnak* (Primary Key). Mivel egy relációnak nem lehet két teljesen egyforma sora, ilyen elsődleges kulcs mindig létezik. Szélsőséges esetben a reláció összes oszlopa alkotja. Az elsődleges kulcsok a relációs modellben kulcsszerepet töltenek be, ezért a 4.4.2. pontban külön is foglalkozunk velük.²

4.2. Példa relációkra

A túloldalon, a 4.1. ábrán bemutatjuk a 3.7. alfejezet kialakított adatmodell megvalósítását a relációs modell alapján. A modell mellett feltüntettük a relációk adatainak egy részét is. A HALLGATÓ relációnak feltüntettük az egyéb jellemzőit is. A többen ezt az áttekinthetőség érdekében elhagytuk. Az adatok integritásának biztosítása érdekében a modellt még ki kell majd egészítenünk a 4.4.3.1. pontban ismertető idegen kulcsokkal.

A három egyedtípusnak megfelelően három relációnk van, HALLGATÓ, TANTÁRGY és MITHALLGAT, melyeknek elsődleges kulcsai rendre a hallgató azonosító száma, HID, a tantárgy kódja, TID, és a kettő együttesen, (HID,TID)³.

A hallgató jellemzői, tulajdonságai, a HALLGATÓ reláció oszlopai, azok adattípusai és értéktartományai:

- A hallgató vezetéckneve, VNÉV. Változó, maximum 20 karakter hosszúságú⁴.
- A hallgató keresztnéve, KNÉV. Változó, maximum 20 karakter hosszúságú.
- Hallgatói azonosító, HID. Mindig 5 karakter hosszúságú. Lehetséges értékei csak 00001 és 99999 között lehetnek.
- A hallgató születésének dátuma, SZDÁTUM. Dátum típusú változó. Lehetséges értéke – a múlt és a jövőre is gondolva, nagy biztonsági ráhagyással – bármilyen

¹ Gyakorlati szempontból azonban a sorok fizikai sorrendjének igen nagy befolyása lehet az adatbázis hatékony működésére. Ezzel a fizikai adatszerkezet ismertetésénél, a 2.5 és a 2.7. alfejezetben részletesen foglalkoztunk.

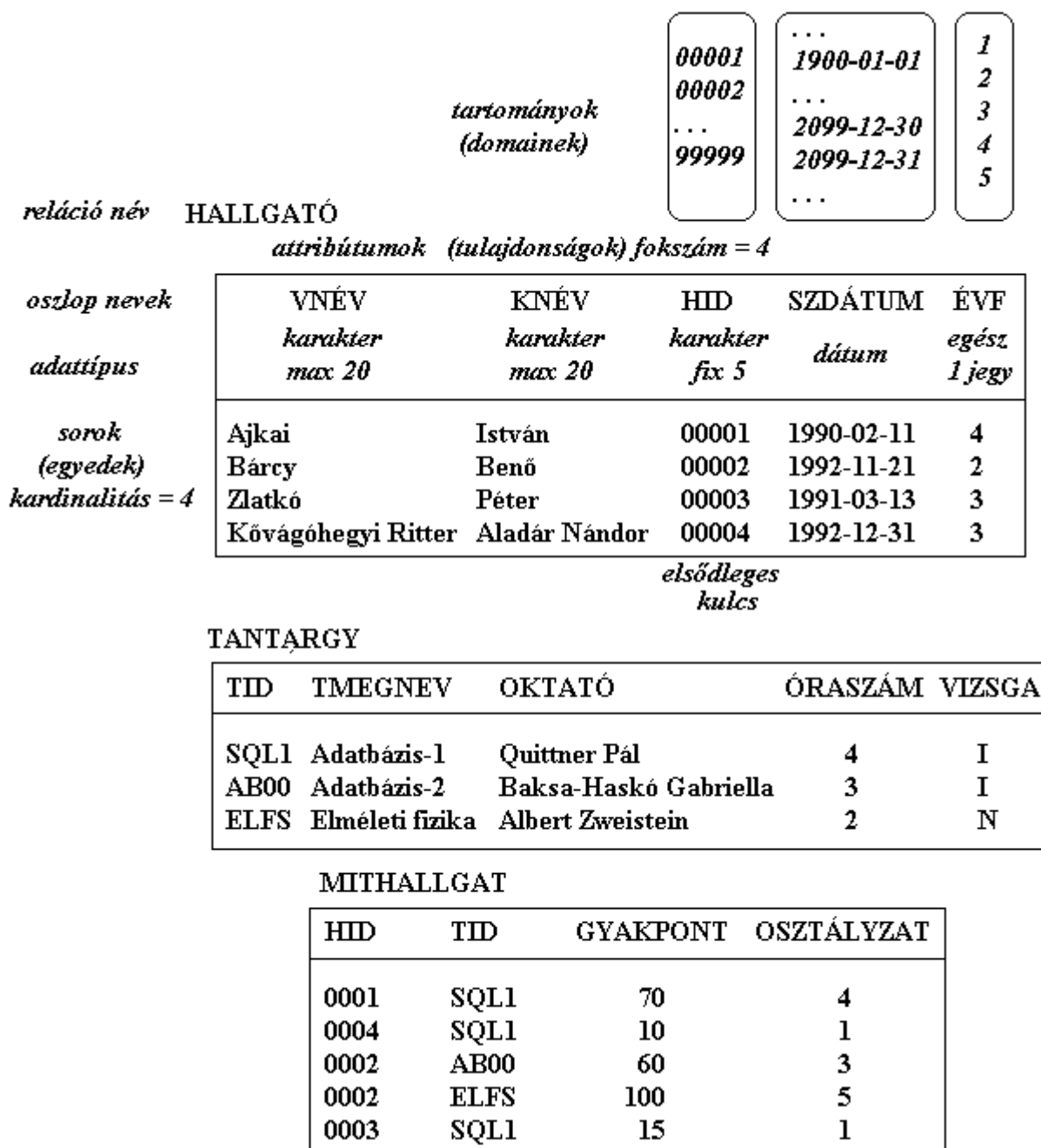
² A gyakorlatban nem mindig vesszük figyelembe ezt a feltételt. Így az adatbázisban lehetnek olyan "relációk" (általában ideiglenes, vagy adatokat nem tartalmazó virtuális relációk, ld. 4.3. alfejezet), melyekben lehetnek azonos sorok is.

³ A három egyedtípus közül csak a HALLGATÓ és a TANTÁRGY igazán létező entitása a valós világnak. A MITHALLGAT a köztük lévő kapcsolatot határozza meg. A relációs modellben azonban az ilyen kapcsolatokat is önálló relációként ábrázoljuk.

⁴ A maximális névhossz meghatározása önkényes. A tervezésnél úgy véljük, hogy 20 karakternél hosszabb nevek nem fordulhatnak elő. Amennyiben ez hamisnak bizonyul, és hosszabb nevet is be kell vinnünk, akkor az adatbázisnak ezt a relációját módosítanunk kell (ld. 5.6.2.3. pont 1. példája). Értéktartományt nem jelölünk ki. Névként elvileg bármilyen karaktersorozat megengedünk.

érvényes dátum lehet. Ha nem ismerjük, akkor is felvesszük a hallgatót az adatbázisba. Ezért ebben az oszlopban a NULL értéket is megengedjük.

- Évfolyam, ÉVF. 1 jegyű, 1 és 5 közti egész szám lehet csak. Alapértelmezése 1.



4.1. ábra: **Hallgatói adatbázis relációs modelljének első formája. A modell mellett feltüntettük a relációk adatainak egy részét is.**

Hasonlóan, a TANTÁRGY reláció oszlopai, azok adattípusai és értéktartományai:

- A tantárgy azonosítókódja, TID. Mindig 4 karakter hosszú.
- A tantárgy megnevezése, TMEGNEV. Változó, maximum 20 karakter hosszúságú.
- A tantárgyért felelős oktató neve, OKTATÓ. Változó, maximum 20 karakter hosszúságú. Nem kötelező megadni.

- A tantárgy heti óraszám, ÓRASZÁM. 1 jegyű, 0 és 9 közti egész szám lehet csak, de NULL értéket is felvehet.
- Kötelező-e belőle a vizsga, VIZSGA. Értéke csak I (igen), vagy N (nem) lehet. Alapértelmezése I.

A MITHALLGAT reláció definíciója:

- A tantárgyat felvett hallgató azonosítója, HID. Mindig 5 karakter hosszúságú, és csak létező hallgató azonosítója lehet¹.
- A hallgató által felvett tantárgy kódja, TID. Mindig 4 karakter hosszúságú, és csak létező tantárgy kódja lehet¹.
- A hallgató által a tantárgyból a gyakorlatokon elért pontok száma, GYAKPONT. 0 és 100 közti, maximum háromjegyű szám lehet. Alapértelmezése 0.
- A hallgatónak a tantárgyból kapott osztályzata, OSZTÁLYZAT. 1 és 5 közötti egy jegyű szám lehet csak. A NULL értéket is megengedjük, hiszen év elején, amikor a hallgató felveszi a tantárgyat, akkor még nem ismerjük az értékét.²

Az így definiált adatmodellnek a tényleges realizálását az 5.14. táblázat: pontban mutatjuk be.

Amennyiben a használat folyamán kiderül, hogy adatmodellünk nem megfelelően írja le a valóságot, akkor a modellt módosítjuk. Amennyiben az adatbázisban már bennlévő adatok megfelelnek az új modell feltételeinek, akkor a módosításokat általában minden nehézség nélkül végrehajthatjuk a fizikailag realizált adatbázisban is az erre szolgáló ALTER típusú SQL utasításokkal (ld. 5.6.2.3. pont).

4.3. Relációk az adatbázisban

Az eddigiekben hallgatólagosan feltételeztük, hogy a relációs modellnek a relációkban lévő adatait ugyanabban a formában, táblaként, fizikailag tároljuk az adatbázisban. Amennyiben ez mindig így lenne, akkor az adatbázis-kezelő rendszerek egyik legfontosabb előnyét, a redundancia mentességet veszítenénk el, hiszen minden elemi adatot (oszlop értéket) annyiszor tárolnánk, ahány sorban, illetve relációban ugyanaz az adat előfordul. Ennek elkerülésére a relációs modell lehetőséget nyújt a fizikailag ténylegesen tárolt relációkból további, virtuális relációk előállítására. Fizikailag az adatoknak ténylegesen csak a minimális mennyiségét tároljuk redundancia mentesen. A feldolgozáshoz valóban szükséges relációkat ezekből állítja elő a rendszer az általunk definiált módon.

Az adatbázisban leggyakrabban előforduló relációtípusok a következők:

- táblázat (Table),
- nézet (View),
- materializált nézet (Materialized View)
- pillanatfelvétel (Snapshot)
- ideiglenes reláció, mely lehet
 - közbenső táblázat
 - lekérdezés eredménye.

A *táblázat* (Table, alap, vagy bázis reláció) fizikailag is ténylegesen tartalmazza az összes adatot a reláció definíciójának megfelelő formában. A táblázatok összessége adja az adatbázis koncepcionális szintjének (ld. 1.2.2.3. pont) az alapját.

¹ Ezt a feltételt a 4.4.3.1. pontban ismertetett idegen kulcs segítségével biztosítjuk majd.

² A leendő alkalmazást ismerve szándékosan tettünk különbséget a GYAKPONT és az OSZTÁLYZAT között a NULL érték tekintetében. GYAKPONT alapértelmezése azért 0, mert a tanév folyamán kapott pontokat mindig hozzáadjuk majd az addig elért pontszámhoz. Ha ez a kezdetkor a NULL érték volna, akkor ezt az első pontszám beírásakor nem tehetnénk meg. NULL érték + bármilyen ismét NULL érték lesz.

A *nézet* (View, virtuális reláció) a felhasználó, vagy az adatbázis-felügyelő által definiált reláció, amelyik fizikailag nem tartalmaz adatokat. Definiálásakor határozzuk meg, hogy milyen táblázatokból, vagy azokra épülő nézetekből, azok mely soraiból és oszlopaiból, milyen feltételek alapján és milyen műveletek segítségével szedje össze az adatbázis-kezelő rendszer a nézet adatait. Ezekkel a nézetekkel lekérdezéskor mindig, módosításkor, beíráskor, törléskor azonban bizonyos esetekben csak korlátozásokkal dolgozhatunk ugyanúgy, mint az adatokat fizikailag tartalmazó táblázatokkal. A nézet használatakor mindig a nézet alapjául szolgáló táblázatok aktuális értékeivel dolgozunk, azokat látjuk, azokat változtatjuk meg¹.

A nézetek lényegében az adatbázis különböző külső szintjeit (ld. 1.2.2.2. pont) vagy azoknak egy részét jelenítik meg.

A nézet adatait az adatbázis-kezelő rendszer az alaptáblázatok aktuális adataiból a nézethez való forduláskor minden egyes alkalommal kiszámolja. Ez időnként, ha például a nézet oszlopai egy nagyon sok sort tartalmazó táblázat oszlopainak átlagát, összegét tartalmazzák igen hosszadalmas lehet. Ha az alap-táblázatok adatai ritkán változnak, akkor a nézet adatait elég egyszer kiszámolni, azt tárolni, és ezzel a tárolt értékkel dolgozni mindaddig, amíg valamelyik alaptáblázat nem módosul. Ekkor a nézet adatait újraszámoljuk, és a továbbiakban ezt tároljuk. Az így definiált, tárolt, és csak a változáskor aktualizált nézetet nevezik *materializált nézetnek* (Materialized View).

A *pillanatfelvétel* (Snapshot) adott táblázat(ok), nézet(ek) pillanatnyi állapotát tartalmazza fizikailag is tárolt formában. Rendszerint csak olvasható. Az adatbázis biztonsága, az adatok helyreállítása érdekében hozzák létre meghatározott időintervallumokban, vagy feladatok elvégzése után.

Bonyolultabb feladatok megoldásához a rendszernek ideiglenesen tárolni kell közbenső eredményeket is. (Például a kiválasztott sorokat más szempont szerint kell rendezni, vagy ki kell szűrni belőlük az azonosakat). Az így létrehozott *közbenső táblázatok* lehetnek valódi, fizikailag is tárolt táblázatok, de hivatkozhatnak csak pointerrel a ténylegesen tárolt adatokra. Ezek a felhasználó számára nem hozzáférhetőek. Csupán addig léteznek az adatbázisban, amíg a felhasználó által kért művelet végrehajtásához szükség van rájuk. Utána automatikusan törlődnek.

A *lekérdezések eredménye* igen gyakran több sor lesz. Ezeket is relációnak tekinthetjük, bár rendszerint nincs önálló nevük, tartalmazhatnak azonos sorokat, és csak addig élnek, amíg a kapott információ feldolgozásra nem kerül.

4.4. Kulcsok

4.4.1. Az adatbázis integritása

Az adatbázisba bevitt és tárolt adatok helyességének ellenőrzésére, az adatbázis integritásának automatikus biztosítására a következő lehetőségeink vannak:

- Relációk egyes oszlopaire vonatkozó ellenőrzés:
 - Megfelelő adattípus deklarálása. Az oszlopba csak a deklarációnak megfelelő adat vihető be. Így például DÁTUM-ként definiált oszlopba csak érvényes dátum, NUMERIKUS-ba csak szám vihető be, KARAKTER-be nem adható meg az oszlop hosszánál több karakter, stb. (ld. 5.4. alfejezet).
 - Alapértelmezést definiálunk. Egyszerűsíti az adatbevitelt és biztosítja, hogy az oszlopnak mindig legyen értéke.

¹ Nem lehet változtatni például egy nézetnek azt az oszlopát, melyet egy táblázat *a* és *b* oszlopának összegeként definiáltunk, mert ebből nem lehet megállapítani, miként kellene az *a* és/vagy a *b* oszlop értékét megváltoztatni, hogy az összegük értéke továbbra is az új oszlop értékével legyen egyenlő.

- NULL érték tiltása vagy megengedése egy oszlopban. Szabályozza, hogy hiányos sor bevihető-e.
- Ellenőrző algoritmus használata. A legtöbb esetben ez az értéktartomány ellenőrzését jelenti (az oszlopba csak a megadott értékhatárok közé eső adatok vihetők be, pl. $1 \leq OSZTÁLYZAT \leq 5$), vagy egyszerű érték ellenőrzést. (az oszlopba csak a megadott értékek vihetők be, pl. VIZSGA értéke csak az I vagy az N értékek valamelyikét veheti fel.)
- Táblázatokra vonatkozó ellenőrzés:
 - Elsődleges kulcs definiálása. Ez biztosítja egy reláció sorainak azonosíthatóságát, a sorok egyediségét. Elsődleges kulccsal rendelkező relációknak nem lehet két azonos sora.
 - Idegen kulcs(ok) definiálása. Ezek segítségével a rendszer automatikusan ellenőrizni tudja, hogy különböző relációk azonos attribútumaira épülő adatai között ne lehessen ellentmondás. (Pl. a MITHALLGAT relációban egy hallgató se vehessen fel nem létező tantárgyat, ne lehessen benn olyan TID, amelyiknek értéke nem szerepel TANTÁRGY azonos oszlopában.)
- Nézetek ellenőrzése
 - A nézeteken keresztül csak olyan adatok vihetők be az alaptáblázatokba, amelyek megfelelnek a nézet definíciójának.

A fenti integritás ellenőrzéseket általában kombináltan használjuk. Megadásukkal biztosítjuk, hogy az adatbázisba *semmiképpen* ne lehessen olyan adatokat bevinni, amelyek ezeknek a feltételeknek nem tesznek eleget.

4.4.2. Elsődleges kulcs

Mint azt a 4.1. alfejezetben a relációk tulajdonságainál megállapítottuk, egy reláció bármely sorát a tartalma alapján azonosítjuk. Minden relációban létezik az oszlopoknak legalább egy olyan kombinációja, amelyik az adott relációban mindig egyedi, és ezáltal egyértelműen meghatározza a sor többi oszlopjának az értékét. Ezt az oszlopkombinációt (amely gyakran csak egy oszlopot jelent) nevezzük a reláció *elsődleges kulcsának* (Primary Key).

Előfordulhat, hogy egy reláció sorait nem csak egy oszlop (kombináció) azonosíthatja egyértelműen. Így egy személyi nyilvántartásban az egyedi azonosító szám (személyi szám, adó szám, TAJ szám) mellett azonosíthat egy személyt a neve, anyja neve, születési helye és ideje is¹, vagy szerepelhet együtt az adó és a TAJ száma is. Ilyenkor ezeket az oszlopokat (oszlopkombinációkat) *lehetséges kulcsoknak* (Candidate Key) nevezzük, és közülük bármelyiket kiválaszthatjuk elsődleges kulcsnak. Ha ez megtörtént, akkor a többi lehetséges kulcsot *alternatív kulcsnak* (Alternative Key) nevezzük.

Azt, hogy a lehetséges kulcsok közül végül melyiket választjuk ki elsődleges kulcsnak, az a modell szempontjából közömbös. Gyakorlati szempontból azonban általában nem mindegy. A relációk egyes sorait gyakran az elsődleges kulcs szerint sűrűsödve helyezük el. Ha a hatékony feldolgozásnál kihasználjuk a sűrűsödést (ld. 2.5.1. pont), akkor az elsődleges kulcsot célszerű eszerint választani. Ha más táblázatból hivatkozunk egy relációra, akkor viszont a hivatkozott oszlopot lehet célszerű elsődleges kulcsnak kijelölni.

Az elsődleges kulcsnak a reláción belül az egyediségen kívül még az alábbi feltételeket is ki kell elégítenie:

¹ Lényeges, hogy az egyediségnek nem csak az éppen aktuális adatállományra kell teljesülnie, hanem a táblázat teljes létezésének az időtartamára. Ilyen szempontból elvileg elképzelhető, hogy a fenti oszlop kombináció két különböző személynél is azonos lehetne, de ennek valószínűségét a gyakorlatban nullának vesszük.

- *Az elsődleges kulcs, illetve összetett kulcsnál annak egy komponense sem lehet NULL érték.* Ez annak az észszerű 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. Hiába van egy adott pillanatban egy személyeket tartalmazó táblázatban, melynek a személy neve, anyja neve, születési helye és ideje az elsődleges kulcsa, csupa különböző nevű személy, a nevet nem vehetjük elsődleges kulcsnak. Ugyanakkor viszont fölösleges – és tilos is – a fenti elsődleges kulcshoz még bármilyen más oszlopot is hozzávenni.

Az elsődleges kulcsok kijelölése az adatbázis-tervezés egyik igen fontos lépése.¹ Definíálása egyben egy egyedi index felépítését is jelenti. Adatbázis-kezelő rendszertől függ, hogy ezt a felhasználónak, vagy az adatbázis-felügyelőnek kell-e külön definiálni, vagy a rendszer maga készíti el automatikusan a kulcs kijelölésének következményeként. Az alternatív kulcsokra nekünk magunknak kell egyedi indexet definiálnunk.

A 4.2. alfejezetben szereplő táblázatok elsődleges kulcsai:

HALLGATÓ (HID)
TANTÁRGY (TID)
MITHALLGAT (HID,TID)

Alternatív kulcs TANTÁRGY-ban a tantárgy megnevezése, TMEGNEV.

4.4.3. Relációk közti kapcsolatok

Az elsődleges kulcs és az oszlopokra meghatározott integritási feltételek egy adott *reláción belül* biztosítják az adatok ellentmondás mentességét. Ahhoz, hogy a különböző relációk között se lehessenek logikai ellentmondások, bizonyos *relációk között* meghatározott összefüggéseknek állandón érvényesnek kell lenni. Erre szolgálnak az idegen kulcsok.

4.4.3.1. Idegen kulcs

A relációk közti kapcsolatok 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 egyenlő². Ezt az oszlopot (kombinációt) nevezik a hivatkozó relációban *idegen kulcsnak* (Foreign Key).

Az egymással ilyen kapcsolatban álló relációk szokásos elnevezése még függő és független, illetve szülő és gyermek relációk.

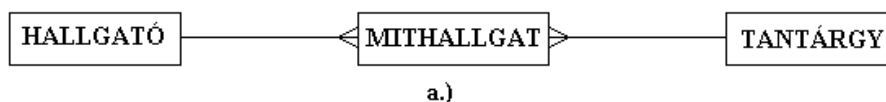
Az idegen kulcsok definiálásával összefüggéseket hozunk létre különböző relációk között. Ezek összességét *hivatkozási feltételeknek*, vagy *függőségi kapcsolatoknak* nevezzük. Ezek lényegét úgy foglalthatjuk össze, hogy *az adatbázisban a hivatkozó relációban nem lehet olyan idegen kulcs, melynek értéke nem egyezik meg a hivatkozott relációban egy hozzákapcsolódó elsődleges kulcs értékével, vagy nem a NULL értéket tartalmazza.* Fordítva az állítás nem áll fenn. A hivatkozott relációban lehetnek olyan kulcs-értékek, melyekkel megegyező idegen kulcs nincsen. Ez érthető is, hisz például lehetnek olyan tantárgyak, amit

¹ Gyakori tervezési hiba, hogy minden egyes relációhoz hozzárendelnek egy mesterséges azonosító számot, melyet egyben elsődleges kulcsnak is kijelölnek. Amennyiben a relációnak van egy természetes azonosítója, például MITHALLGAT-ban (HID,TID), akkor fölösleges ehhez még egy külön azonosító oszlopot is felvenni.

² Egyes rendszerek megengedik az alternatív kulcsra való hivatkozást is.

végül senki sem vesz fel, sőt olyan hallgatók is, akik egy tantárgyat sem vettek fel. (Ez nem csak elvi lehetőség, a tanév elején ez lehet egy ideig az adatbázis természetes állapota.)

Az idegen kulcs valójában két reláció közti egy-több kapcsolat leírását jelenti, ahol az idegen kulcs jelenti a kapcsolatot „több” oldalát. Az Entity – Relationship diagramban feltüntethetjük ennek a kapcsolatnak a nevét és a kulcs-kapcsolatot érintő műveletek módját is.



HALLGATÓ *a hallgatók adatai*

oszlop név	jelentés	adattípus	alapértelmezés	korlátozás
VNÉV	<i>vezetéknév</i>	max 20 karakter		nem lehet NULL
KNÉV	<i>keresztnev</i>	max 20 karakter		nem lehet NULL
HID	<i>hallgató azonosítója</i>	5 karakter		elsődleges kulcs
SZDÁTUM	<i>születési dátum</i>	dátum		
ÉVF	<i>évfolyam</i>	1 jegyű szám	1	>=1 és <=5

TANTÁRGY *a tantárgyak adatai*

oszlop név	jelentés	adattípus	alapértelmezés	korlátozás
TID	<i>tantárgy azonosítója</i>	4 karakter		elsődleges kulcs
TMEGNEV	<i>megnevezés</i>	max 20 karakter		nem lehet NULL alternatív kulcs
OKTATÓ	<i>felelős oktató neve</i>	max 20 karakter		
ÓRASZÁM	<i>heti órák száma</i>	1 jegyű szám		>=0 és <=9
VIZSGA	<i>kötelező-e a vizsga</i>	1 karakter	I	csak I vagy N lehet

MITHALLGAT *a hallgatók és a tantárgyak közötti kapcsolat adatai*

oszlop név	jelentés	adattípus	alapértelmezés	korlátozás
HID	<i>hallgató azonosítója</i>	5 karakter		elsődleges kulcs idegen kulcs HALLGATÓ-ra
TID	<i>tantárgy azonosítója</i>	4 karakter		elsődleges kulcs idegen kulcs TANTÁRGY-ra
GYAKPONT	<i>kapott gyakorlati pontok</i>	3 jegyű szám	0	>=0 és <=100
OSZTÁLYZAT	<i>kapott osztályzat</i>	1 jegyű szám		>=1 és <=5

b.)

4.2. ábra: **Hallgatói adatbázis relációs modelljének végső formája.**

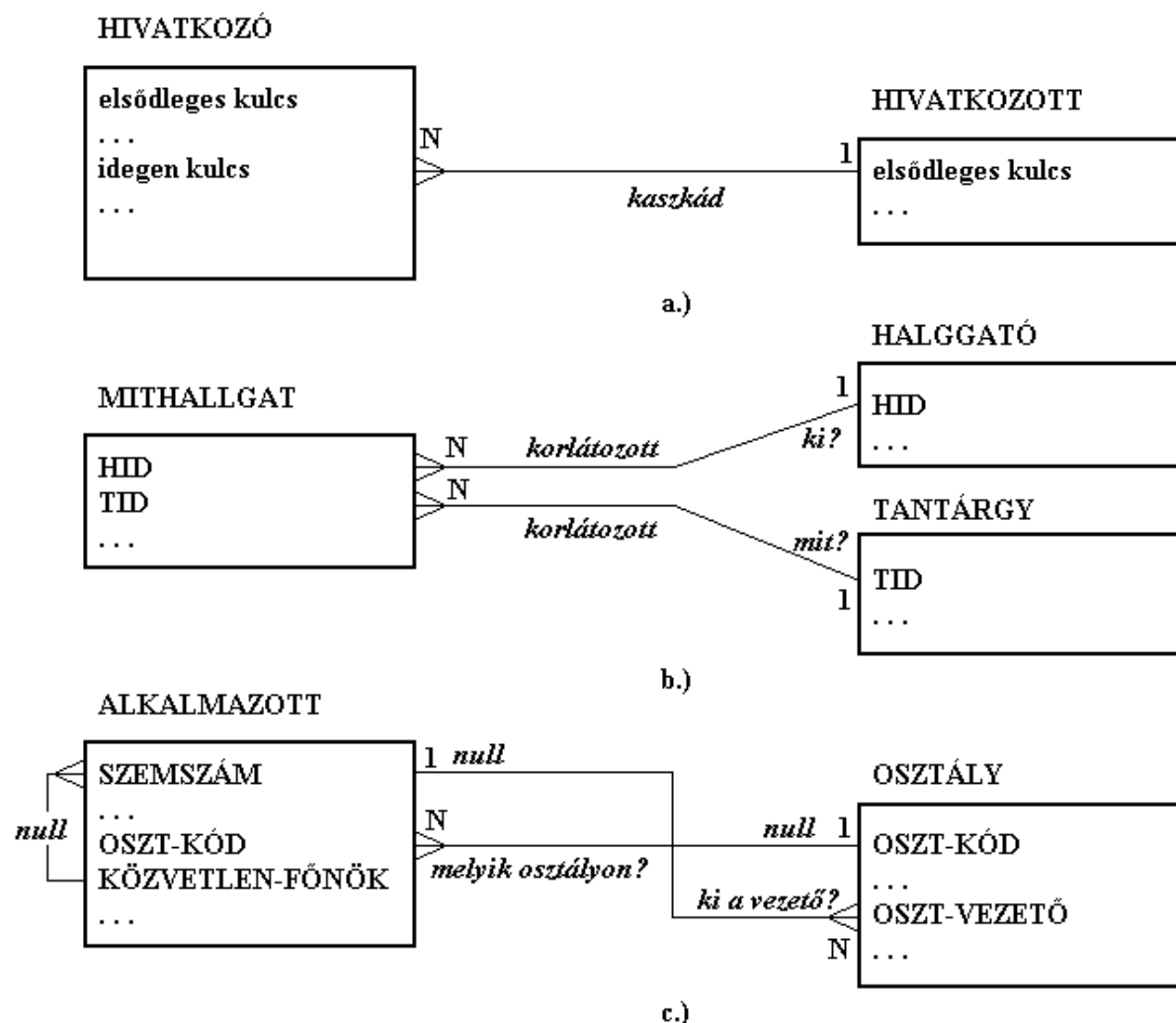
a.) A relációk és kapcsolataik, b.) A relációk attribútumai részletezve.

Ilyen idegen kulcs a MITHALLGAT (hivatkozó) relációban HID és TID, melyek a HALLGATÓ, illetve a TANTÁRGY (hivatkozott) relációk elsődleges kulcsaira hivatkoznak. Ez gátolja meg azt, hogy egy tantárgyat nem létező hallgató vegyen fel, vagy egy hallgató

nem létező tantárgyat hallgathasson. Ezt figyelembe véve a 4.1. ábrán látható adatmodellt kiegészítjük az idegen kulcsokkal is. Így kapjuk meg a 4.2. ábrán látható végleges adatmodellünket.

Az oszlopok típusa a hivatkozó és a hivatkozott relációban azonos. Neveik különbözhetnek, bár egy jól tervezett adatbázisban – mivel ugyanarra a tulajdonságra vonatkoznak – megegyeznek. A fenti példában az idegen kulcsok részei az elsődleges kulcsnak, de ez nem kötelező.

A hivatkozó és a hivatkozott reláció megegyezhet. Erre példa lehet a 4-36. oldalon lévő 4.15. ábra ALKALMAZOTT-3NF táblázatának egy olyan kibővítése, mely tartalmazza az alkalmazott közvetlen főnökének (mely csak egyetlen egy személy lehet) a személyi azonosító számát. Ez – az adatbázis konzisztenciája miatt – természetesen csak olyan értéket vehet fel, amelyik már szerepel az alkalmazottak személyi száma között. Ez egyben jó példa arra is, hogy miért kell egyes esetekben idegen kulcsnál megengedni a NULL értéket is. A vállalat legelső emberének nincsen főnöke. Ezért ebben a sorban ennek az attribútumnak az értéke csak a NULL érték lehet.



4.3. ábra: Idegen kulcsok ábrázolása.

- a.) általános formában b.) az EGYETEMI NYILVÁNTARTÁS modellben
c.) hivatkozási ciklus és önhivatkozás

Elvileg a hivatkozó táblázatra is hivatkozhatnak más táblázatok, és az is előfordulhat, mint azt a 4.3.c. ábrán bemutatjuk, hogy egy hivatkozási ciklus több táblázat bevonása után

záródik. Itt az ALKALMAZOTT táblázatban csak olyan dolgozók szerepelhetnek, akiknél a KÖZVETLEN-FŐNÖK száma szerepel ugyanezen táblázatnak egy másik sorában SZEMSZÁM-ként (önmagára való hivatkozás), vagy NULL érték, OSZT-KÓD-ja létezik az OSZTÁLY táblázatban, vagy NULL érték (egyszerű hivatkozás). Az OSZTÁLY táblázatban csak olyan sorok lehetnek, melyekben az OSZT-VEZETŐ érvényes SZEMSZÁM az ALKALMAZOTT táblázatban, vagy NULL érték (egyszerű hivatkozás, és egyben hivatkozási ciklus létrehozása)¹.

4.4.3.2. Idegen kulcsot érintő műveletek

Az adatbázis-kezelő rendszernek automatikusan biztosítani kell azt, hogy az idegen kulcsokkal definiált hivatkozási feltételek az adatok bármilyen megváltoztatása után is érvényben maradjanak. Az idegen kulccsal összekapcsolt táblázatokban háromféle változtatás lehet, amelyik érintheti a feltételeket:

- új adat bevitele,
- bennlévő adat módosítása,
- bennlévő adat törlése.

Bevitel

A hivatkozott táblázatba bármilyen új sort bevitethetünk.

A hivatkozó táblázatba csak olyan sort vihetünk be, melyben az idegen kulcs értéke megegyezik egy a hivatkozott táblázatban már létező elsődleges kulcs értékével, vagy NULL érték (feltéve, hogy ezt az értéket megengedi az adatmodell).

Ez a szabály teljesen merev, nem enged meg semmiféle választási lehetőséget.

Törlés

A hivatkozó táblázatból minden korlátozás nélkül törölhetünk sorokat.

Ha a hivatkozott táblázatból is szabadon törölhetnénk, akkor a függőségi kapcsolatok felborulhatnak. Ennek megakadályozása érdekében a relációs modell csak az alábbi törlési módokat teszi lehetővé:

- korlátozott (Restricted) törlés,
- törlés NULL értékre állítással (Set Null),
- tovagyrúzó vagy kaszkád (Cascade) törlés.

Azt, hogy a fentiek közül melyik törlési módot engedjük meg, az idegen kulcs definiálásakor kell megadnunk.

Korlátozott törlési lehetőségnél a hivatkozott táblázat sora csak akkor törölhető, ha nincs rá hivatkozás, azaz az elsődleges kulcsával egyező idegen kulcs érték nincsen a hivatkozott táblázatban. Amennyiben van, és mégis törölnünk kell egy ilyen rekordot, ezt csak úgy tehetjük meg, hogy előbb töröljük az összes erre hivatkozó sort, vagy az idegen kulcs értéket ebben más megengedett értékre változtatjuk. Az egyetemi adatbázis példánknál maradvá ez azt jelenti, hogy nem törölhetünk olyan tantárgyat a nyilvántartásból, amit legalább egy hallgató felvett.

Az adatbázis konzisztenciája, az adatok helyessége szempontjából ez a mód a legbiztosabb. Ezért, ha nem adunk meg mást, ez a törlési mód lesz érvényben. Ugyanakkor azonban, alkalmazásával bizonyos nehézségek járhatnak. Törlés előtt a hivatkozó táblázatokat ellenőrizni kell. Ha van hivatkozás, a műveletet nem tudjuk végrehajtani. Hibajelzést kapunk, amit ha nem kezelünk megfelelő módon le, akkor a feldolgozásunk leáll. Ezért szokásos

¹ A NULL értéket OSZT-KÓD-ban vagy OSZT-VEZETŐ-ben egy osztály első személyének (vezetőjének) vagy egy új osztálynak felvételéhez meg kell engednünk.

törlési mód a *NULL értékre állítás* is. Ekkor a törölt értékkel egyező értékű idegen kulcsok a hivatkozó táblázatban mind automatikusan a NULL értéket fogják felvenni. Ez természetesen csak akkor lehetséges, ha az idegen kulcsot alkotó oszlop(ok) megengedi(k) a NULL értéket.

A NULL értékre állítással való törlés végrehajtása egyszerűbb. A törlés minden további kiegészítő művelet nélkül elvégezhető. Ugyanakkor viszont fennáll a veszélye annak, hogy az automatikus módosítás miatt egyes hasznos információk elveszhetnek. (Például egy árunak, amelyiket a továbbiakban már nem szállítunk, töröljük a kódját. Ennek következtében az összes régebbi megrendelésben azonosíthatatlanná válik ez az áru. Egy ilyen adatmodell formailag helyes lehet, de logikailag mindenképpen hibás! Egyetemi modellünk nem is engedné meg a törlést NULL értékre állítással.)

Még kényelmesebb, de még nagyobb veszélyeket rejt magában a *tovagyűrző* vagy *kaszkád* törlés. Ha ezt definiáljuk, akkor a hivatkozott táblázatból az elsődleges kulcsot tartalmazó sorral együtt automatikusan törlődik a hivatkozóból is az összes olyan sor, amelyikben az idegen kulcs a törölt kulcs-értékkel volt egyenlő. Különösen veszélyes lehet ez akkor, ha a hivatkozó táblázatra más táblázatok is hivatkoznak, mert ez további törléseket vonhat maga után¹.

Módosítás

A lehetőségek ugyanazok, mint a törlésnél, azzal a különbséggel, hogy a kaszkád módosítás az idegen kulcs értékének az elsődleges kulcs új értékére történő automatikus megváltoztatását jelenti a hivatkozó táblázat összes érintett sorában.

Természetesen a fent felsorolt feltételeken kívül mind a hivatkozott, mind a hivatkozó táblázatban történő változtatás eredményeként létrejövő új sorra érvényesnek kell lenni a táblázatra definiált összes többi feltételnek is.

4.4.4. További kulcsok

Bár valójában nem tartozik az adatmodellhez, a relációs adatbázisokban az elsődleges és az idegen kulcson kívül még további kulcsoknak is fontos szerepe van. Ezek

- az elhelyezési kulcs,
- a keresési kulcs,
- és a sűrűsödési kulcs, melyet a 2.5.1. pontban már tárgyaltunk.

4.4.4.1. Elhelyezési kulcs

Az adatbázisban minden tároló helynek van egy címe. Az egyes táblázatok sorait a táblázat számára kijelölt tároló hely meghatározott címére helyezzük el. A tárolási hely kiválasztása az alábbi módokon történhet:

- Sorosan, a sort a táblázat számára kijelölt tároló hely első szabad címén tároljuk el.
- A tároló címet a beírandó sor megadott oszlopának (oszlopkombinációjának) aktuális értékéből határozzuk meg. Ezt az oszlopot (illetve oszlopkombinációt) nevezzük a reláció *elhelyezési*, vagy *besorolási kulcsának* (Order Key).

Ez az elhelyezési kulcs igen gyakran egyben az elsődleges kulcs is. Ilyenkor, ha az új rekordok elsődleges kulcsát egy folyamatosan növekvő sorszámgenerátor szolgáltatja, akkor ez egyben sűrűsödési kulcs is, és megfelel a soros beviteli módnak is.

¹ Ennek elkerülésére sok rendszerben a kaszkád törléssel definiált hivatkozó táblázatra más táblázatok egyáltalában nem, vagy csak korlátozott törléssel hivatkozhatnak

Speciális alkalmazásoknál előfordul, hogy a hatékonyság érdekében nem az elsődleges kulcs szerint sűrűsödve akarjuk egy táblázat sorait tárolni. Ekkor a sűrűsödési kulcs lesz egyben az elhelyezési kulcs is.

4.4.4.2. Keresési kulcs

Az adatbázis relációiban különböző szempontok szerint kereshetünk. Azokat az oszlopokat (oszlopkombinációkat), melyek értéke alapján rendszeresen keresünk, nevezzük *keresési kulcsoknak* (Search Key). A keresés meggyorsítására az egyes keresési kulcsokra felépítünk egy indexet (ld. 2.4.1. pont). Az elsődleges kulcs egyben mindig keresési kulcs is. Ezen kívül az alkalmazások igényétől függően akárhány keresési kulcsot, indexet definiálhatunk nem csak az adatbázis megtervezésekor, hanem menetközben, utólag is, ahogy a felhasználói követelmények változnak.

A különféle kulcsokra példaként tekintsük a HALLGATÓ táblázatot. Ezek sorait úgy tároljuk, ahogy a hallgatók adatait sorban felvesszük. A következő új hallgató azonosítója, HID, az utolsó azonosító értékénél eggyel nagyobb szám lesz. HID tehát egyben elhelyezési és elsődleges kulcs is. A táblázatban keresünk az azonosító, a hallgató neve és évfolyama alapján is. Ezek az oszlopok, HID, VNÉV+KNÉV, ÉVF tehát keresési kulcsok lesznek. Az elhelyezés és az azonosító képzésének algoritmusa következtében HID egyben sűrűsödési kulcs is.

4.5. Relációs műveletek

A relációs modell egyik legnagyobb előnye az, hogy a különféle feltételeknek megfelelő, fizikailag is létező, vagy ezekből logikai úton előállítható egyedeit könnyen, egyszerűen meg tudjuk határozni, azonosíthatjuk, kiválaszthatjuk, definiálhatjuk őket a *relációs műveletek* segítségével.

A relációs műveletek az alábbi feladatok elvégzésére szolgálnak:

- a kereséskor kiválasztandó adatok meghatározása,
- a módosítandó, törlendő adatok kiválasztása,
- más reláció(k)ból beírandó adatok kijelölése,
- virtuális adatok definiálása. Ezek lehetnek az adatmodellnek állandó, névvel azonosított relációi (pl. nézetek) vagy csak ideiglenesen létezők, melyek csupán az adott feladat megoldásához szükségesek és annak befejeztével automatikusan megszűnnek.
- az adatbázis egyes részeihez történő hozzáférési jogok érvényességi területének kijelölése.

Mivel a relációs műveletekkel leggyakrabban adatokat választunk ki, a továbbiakban általában kiválasztásról, kiválasztási műveletről fogunk beszélni, de ez alatt nem csak a szűkebb értelemben vett kiválasztást értjük, hanem a fent felsorolt funkciók bármelyikét.

Az ebben a pontban ismertetendő relációs műveleteket és azok különböző kombinációit a gyakorlatban az 5.6. alfejezetben tárgyalandó SQL utasítások segítségével hajtjuk végre az adatbázisban. Ezekben azt fogalmazzuk meg, hogy *mit* akarunk az adatbázisban véghezvinni. Hogy ez *miként* fog megvalósulni, azt általában az adatbázis kezelő rendszerre bízunk.

4.5.1. A legfontosabb relációs műveletek

A relációs műveletek zártak. Relációkon hajtódnak végre és eredményként relációt szolgáltatnak. Aszerint, hogy egy vagy több relációra vonatkoznak, beszélünk *egy- és több-relációs műveletekről*. A zártság következtében a többrelációs műveletek mindig

visszavezethetők egymás után végrehajtott kétoperandusos műveletekre. Az ismételten alkalmazott művelet egyik operandusa lesz az előzőekben végrehajtott művelet eredménye a másik pedig egyenként a harmadik, negyedik, ... és így tovább a következő operandus. Megkülönböztetünk ezen kívül *elemi* és *nem elemi relációs műveleteket* is. Ez utóbbiak különböző elemi műveletek eredményeként állnak elő, de gyakorlati fontosságuk miatt ezekre a műveletsorozatokra önálló névvel is hivatkozhatunk, önálló műveletnek tekintjük.

Az alábbiakban ismertetjük a leggyakrabban használt relációs műveleteket, melyekben vannak elemi és nem elemi műveletek is:

- átnevezés (RENAME),
- korlátozás vagy restrikció (RESTRICT),
- vetület vagy projekció (PROJECT),
- keresztszorzat (TIMES),
- egyesítés vagy összekapcsolás (JOIN),
- egyesítés vagy összekapcsolás alapértelmezéssel (OUTER JOIN),
- unió (UNION),
- metszet (INTERSECT),
- különbség (DIFFERENCE),
- bővítés (EXTEND),
- csoport képzés (SUMMARIZE),
- kijelölés (CREATE).

A továbbiakban részletesen ismertetjük ezeket a műveleteket.

4.5.2. Átnevezés (RENAME)

Az egyrelációs elemi műveletek közül a legegyszerűbb az *átnevezés* (RENAME). Ez egyszerűen átnevezi az R reláció egy oszlopát

RENAME R (r_1, r_2)

ahol r_1 az eredeti, r_2 pedig az új oszlopnév.

Ugyanazon reláció több oszlopának az átnevezését a RENAME elemi utasítás ismételt alkalmazásával végezhetjük el. Ehelyett definiálhatjuk a több oszlop egyszerre történő átnevezését lehetővé tevő RENAME utasítást a

RENAME R ($r_{11}, r_{21}; r_{12}, r_{22}; \dots r_{1n}, r_{2n}$)

módon, ahol r_{1i}, r_{2i} az összetartozó régi és új oszlopnevek.

Erre a műveletre a zártság miatt van szükségünk. E nélkül egy olyan kétrelációs művelet, melyben az operandusokban azonos nevű oszlopok vannak, olyan eredményt adhatna, melyben azonos nevű oszlopok lehetnének.

A több-relációs műveletekben az eredményreláció oszlopneveinek egyértelművé tétele érdekében szokásos az oszlopneveket a reláció nevével minősíteni. Így az R és S relációk azonos nevű k oszlopait $R.k$, illetve $S.k$ -val jelöljük, ahol az előbbi az R , az utóbbi az S k nevű oszlopát jelenti.

4.5.3. Korlátozás (RESTRICT)

Egy relációs elemi művelet a *korlátozás* vagy *restrikció* (RESTRICT)¹. Az eredmény az R relációnak azokat és csak azokat a sorait tartalmazza, melyeknél az r oszlopok értékeire teljesül az F korlátozó feltétel. Ez a feltétel általában az r_i oszlop(ok) egyenkénti összehasonlítását ($=$, $<$, $<=$, ...) jelenti egy másik operandussal, amelyik vagy egy állandó (literál), vagy éppen, vagy egy másik sorban egy másik oszlop értéke, vagy egy relációs művelettel kapott skalár érték, és ezek eredményének AND, OR vagy NOT-tal való összekapcsolását. De mint azt a harmadik példa is mutatja, lehetnek ennél összetettebb feltételek is.²

Általános formája:

RESTRICT R WHERE $r : F$

Az eredmény-relációnak és oszlopainak a neve ugyanaz, mint az eredeti relációnak. Elsődleges kulcsa is megmarad. Az SQL utasításokban a korlátozás megfelel az adatkezelő utasítások (ld. 5.6.3. pont) WHERE klauzulájában megadott feltételnek.

RESTRICT HALLGATÓ WHERE ÉVF = 5

RESTRICT HALLGATÓ WHERE ÉVF = 5 OR SZDATUM $\leq$ '1988-12-31'

RESTRICT HALLGATÓ WHERE ÉVF = 5 AND HID =
{PROJECT [RESTRICT MITHALLGAT WHERE OSZTÁLYZAT = 5] (HID) }

Az első korlátozás kiválasztja a HALLGATÓ relációból az 5. éves hallgatók összes adatát, a második az 5. éveseket, vagy az 1988-ban vagy annál korábban születetteket, míg a harmadik az ötödévesek közül azokat, akiknek van ötös osztályzatuk (a PROJECT értelmezését ld. a következő pontban).

4.5.4. Vetület (PROJECT)

Szintén egy relációs elemi művelet a *vetület* vagy *projekció* (PROJECT). Ez az R relációból csak a vetületben megadott $r_1, r_2, \dots, r_n$ oszlopokat tartja meg ugyanazokkal a nevekkel azonosítva:

PROJECT $R(r_1, r_2, \dots, r_n)$

ahol $r_1, r_2, \dots, r_n$ rendre az eredeti R relációnak a vetületben is megmaradó oszlopai. Sorrendjük különbözhet az eredeti sorrendtől. Ezért a vetület egy reláció oszlopai sorrendjének a felcserélésére is szolgál.

Ha a vetület tartalmazza az elsődleges kulcsot, akkor ennek is ez lesz az elsődleges kulcsa. Ha nem tartalmazza, akkor az első lépésben kapott eredménynek lehetnek azonos sorai. Ezért a fenti definíciót a zártság miatt ki kell egészítenünk azzal, hogy ebben az esetben az azonos sorok közül csak egyet tartunk meg³.

A 4.4. ábrán a korlátozás és a vetület hatását láthatjuk egy adott relációra.

¹ Gyakran kiválasztásnak, szelekciónak (SELECT) is nevezik. Ez azonban félreértésekre vezethet, mert az SQL nyelv kiválasztó utasításának is SELECT az elnevezése, mely azonban ennél lényegesen összetettebb feladatok elvégzésére szolgál.

² Az összehasonlító operátorok és feltételek gyakorlati lehetőségeit az SQL nyelvénél, az 5.3.4. pontban tárgyaljuk.

³ Az SQL nyelvben ennek a SELECT DISTINCT ... kifejezés felel meg.

\\
\\
\\

a.)

\\		\\	\\	
\\		\\	\\	
\\		\\	\\	
\\		\\	\\	
\\		\\	\\	

b.)

x		x	x	
x		x	x	
x		x	x	

c.)

4.4. ábra: Egyrelációs műveletek.

a.) korlátozás, b.) vetület, c.) korlátozás és vetület

Példák a projekcióra:

PROJECT HALLGATÓ (HID, VNÉV, KNÉV)

PROJECT

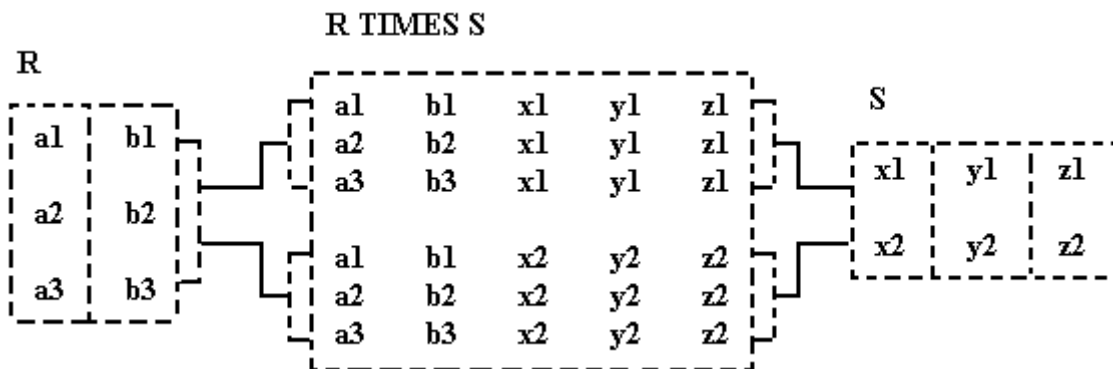
[RESTRICT TANTÁRGY WHERE OKTATÓ = 'QUITTNER'] (TMEGNEV)

Az első vetület kiválasztja a HALLGATÓ táblázat összes sorából az eredetitől eltérő sorrendben a HID, VNÉV, KNÉV oszlopok értékét. A második megadja a TANTÁRGY táblázatból azoknak a tantárgyaknak a megnevezését, melyeket a *QUITTNER* nevű oktató tanít.

Az SQL-ben a vetületet a SELECT utasítás kiválasztási-listájának megadásával állítjuk elő. (ld. 5.6.3.2. pont).

4.5.5. Kereszt-Descartes szorzat (TIMES)

A kétrelációs műveletek közül a legfontosabb a *kereszt-szorzat* (TIMES), amit a matematikában Descartes szorzatnak neveznek. Az $R(r_1, r_2, \dots, r_n)$ és az $S(s_1, s_2, \dots, s_m)$ n illetve m foks számú relációk kereszt-szorzata $R*S$, vagy $R \text{ TIMES } S$ az az $n+m$ foks számú reláció, melynek oszlopait úgy kapjuk meg, hogy az R reláció összes oszlopa mögé hozzáírjuk az S reláció minden oszlopát, sorait pedig úgy, hogy R minden egyes sora mögé egyenként hozzáírjuk S minden sorát. Ha az eredeti relációk sorainak száma, kardinalitása i , illetve j volt, akkor a kereszt-szorzaté $i*j$ lesz (ld. 4.5. ábra).



4.5. ábra: Két reláció, R és S kereszt-szorzata.

Az oszlopok nevei (a, b, x, y, z) a két relációban különböznek, ezért a szorzás elvégzése előtt nincs szükség az átnevezés műveletre.

A szorzat elsődleges kulcsa a két tényező reláció elsődleges kulcsának a konkatenációja lesz.

A szorzat oszlopainak a neve megegyezik az egyes tényezők oszlopainak a nevével. A relációs műveletek zártágából következik, hogy emiatt a művelet csak akkor hajtható vége, ha a két relációban nincsenek azonos nevű oszlopok. Amennyiben ez nem állna fenn, akkor az átnevezés (RENAME) relációs művelettel előbb át kell neveznünk az egyik relációban a másikkal azonos nevű oszlopokat.

Megjegyezzük, hogy bár a keresztszorzat elvileg igen fontos alpművelet, a gyakorlatban szinte mindig a korlátozással együtt alkalmazzák. Ennek az a magyarázata, hogy amíg nagy gépeknél akár 1 millió sorból álló relációk kezelése sem vet fel önmagában hatékonysági problémákat, addig már két 100 000 soros reláció keresztszorzata is 10 milliárd sort tartalmazna. Ennek akár csak ideiglenes tárolása, az ebben való soros keresés, vagy a közvetlen hozzáféréshez ideiglenes indexek felépítése már komoly tárolási és feldolgozási idő problémákat vethet fel.

A speciális korlátozással kiegészített keresztszorzatot, join néven összetett műveletként, önálló relációs műveletként is definiáljuk. Az SQL nyelvben a keresztszorzat műveletét a SELECT utasítás FROM klauzulájában definiáljuk (ld. 5.6.3.2. pont), és vagy rögtön itt, vagy a WHERE klauzulában korlátozzuk azonnal a kiválasztandó sorok számát.

4.5.6. Egyesítés (JOIN)

Az adatbázisokkal kapcsolatos munkákban a legfontosabb és leggyakrabban használt két operandusos művelet az *egyesítés* vagy *összekapcsolás* (JOIN). Bár ez nem elemi művelet, fontossága miatt önálló műveletként definiáljuk, melyet az adatbázis-kezelő rendszer azonnal értelmezni tud. A join utasítás segítségével a felhasználó sokkal könnyebben tudja megfogalmazni kívánalmait, mint hogyha egy join eredményét elemi utasításokból, lépésenként kellene összeállítania.

Az utasításnak két fő formája van:

- a természetes (Natural) vagy belső join,
- a külső (Outer) join, az egyesítés alapértelmezéssel.

A join utasítást és operandusait az SQL nyelvben a SELECT utasítás FROM klauzulájában adhatjuk meg. Az operandusokat összekapcsoló join-oszlopokat vagy itt, vagy a WHERE klauzulában jelölhetjük ki. (ld. 5.6.3.2. pont).

4.5.6.1. Természetes join

A természetes join két relációból egy új relációt hoz létre. Lényegének megértéséhez az alábbi példán mutatjuk be az alkalmazását:

A MITHALLGAT táblázatból megtudhatjuk, hogy a hallgatók milyen tárgyakat hallgatnak, de személyi adataikból csak a hallgatói azonosítót (HID) kaphatjuk meg (ld. a 4-9. oldalon lévő 4.2. ábrát). A hallgatók további adatait a HALLGATÓ táblázatból szerezhethetjük meg. Ehhez a két táblázatot egyesítenünk kell. A MITHALLGAT minden egyes HID értékéhez tartozó rekordhoz hozzá kell kapcsolnunk azt a HALLGATÓ rekordot, amelyikben a hallgató azonosítójának értéke megegyezik (ugyancsak HID, de most ez a HALLGATÓ táblázat egy oszlopa) a MITHALLGAT-ban levő hallgató-azonosító értékével. Ezt a műveletet nevezzük a HALLGATÓ és a MITHALLGAT táblázatok természetes vagy belső egyesítésének vagy összekapcsolásának, vagy a szokásos angol elnevezésével Natural Join-jának, azzal kiegészítve, hogy az egyesítés a HID oszlopok egyenlősége alapján történik. Jelölése

MITHALLGAT JOIN HALLGATÓ (HID)

Ha az egyes relációk adatai a következők:

MITHALLGAT

HID	TID	GYAKPONT	OSZTÁLYZAT
11111	ORA	31	1
11111	DB2	81	4
11111	ADB	91	5
22222	ORA	52	2
22222	DB2	62	3
33333	ADB	63	3
33333	ORA	83	4
33333	ACCS	93	5
44444	DBMS	84	4
44444	ORA	94	5

HALLGATÓ

VNEV	KNEV	HID	SZDÁTUM	EVF
Antal	Alfonz	11111	1980-01-01	1
Balogh	Benedek	22222	1980-02-02	2
Csaba	Cecília	33333	1980-03-03	3
Dani	Demeter	44444	1980-04-04	4
Elek	Etelka	55555	1980-11-11	1

akkor a MITHALLGAT JOIN HALLGATÓ (HID)-t az alábbi ideiglenes relációból kapjuk az összekapcsolást létrehozó, azonos MITHALLGAT.HID és HALLGATÓ.HID oszlopok egyikének elhagyásával.

HID (MITHALLGAT)	TID	GYAKPONT	OSZTÁLYZAT	VNEV	KNEV	HID (HALLGATÓ)	SZDÁTUM	ÉVF
11111	ORA	31	1	Antal	Alfonz	11111	1980-01-01	1
11111	DB2	81	4	Antal	Alfonz	11111	1980-01-01	1
11111	ADB	91	5	Antal	Alfonz	11111	1980-01-01	1
22222	ORA	52	2	Balogh	Benedek	22222	1980-02-02	2
22222	DB2	62	3	Balogh	Benedek	22222	1980-02-02	2
33333	ADB	63	3	Csaba	Cecília	33333	1980-03-03	3
33333	ORA	83	4	Csaba	Cecília	33333	1980-03-03	3
33333	ACCS	93	5	Csaba	Cecília	33333	1980-03-03	3
44444	DBMS	84	4	Dani	Demeter	44444	1980-04-04	4
44444	ORA	94	5	Dani	Demeter	44444	1980-04-04	4

Az 55555 azonosítójú hallgató nem szerepel MITHALLGAT-ban, ezért a join eredményében sem kapjuk meg.

A fenti példa után definiáljuk pontosabban is a műveletet. A természetes join-nak a gyakorlatban használatos definíciója a következő:

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_j oszlopok között nincsen azonos nevű¹. R és S

¹ Az oszlopok sorrendje mindkét relációban tetszőleges lehet. Az azonos attribútumokat jelentő oszlopokat csak az áttekinthetőség kedvéért írtuk az egyik relációnak a végére, a másiknak pedig az elejére. Amennyiben a

természetes egyesítése, összekapcsolása (Natural Join) a $k_1, k_2, \dots, k_i$ oszlopok alapján 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 páronként megegyeznek. (ld. 4.6. ábra). A két reláció közös oszlopaikat nevezzük *join oszlopoknak*. Ha a relációknak nincsenek közös oszlopai, akkor a természetes join megegyezik a keresztszorozattal. Ha a közös join oszlopokban nincsenek azonos értékek, akkor a természetes join eredménye egy üres, egyetlen sort sem tartalmazó reláció lesz.

Az R és S relációk természetes join-ja a $k_1, k_2, \dots, k_i$ join oszlopok alapján, $R \text{ JOIN } S$ ($k_1, k_2, \dots, k_i$) elvileg az alábbi elemi műveletekből tevődik össze¹:

$T1 = R \text{ TIMES } S$

$T2 = \text{RESTRICT } T1 \text{ WHERE } R.k_1 = S.k_1 \text{ AND } R.k_2 = S.k_2 \dots \text{ AND } R.k_i = S.k_i$

$R \text{ JOIN } S(k_1, k_2, \dots, k_i) = \text{PROJECT } T2(r_1, r_2, \dots, r_n, R.k_1, R.k_2, \dots, R.k_i, s_1, s_2, \dots, s_m),$

ahol $T1$ és $T2$ közbenső eredményként kapott relációk.

A természetes egyesítést, összekapcsolást (Natural Join) az általános szóhasználatban jelző nélkül, csak egyesítésnek, összekapcsolásnak vagy joinnak nevezik. A továbbiakban, amikor jelző nélkül használjuk a joint vagy valamelyik szinonimáját, mindig a természetes joint fogjuk ezen érteni.

Az egyesítés eredményeként kapott reláció oszlopainak száma, fokszáma $n+i+m$. Sorainak száma, kardinalitása függ a join-oszlopokban lévő értékektől, nullától elvileg a maximális keresztszorzat kardinalitásáig változhat. A gyakorlatban azonban ennél általában nagyságrenddel kisebb. Elsődleges kulcsa, ha a join oszlopok tartalmazzák valamelyik reláció elsődleges kulcsát, akkor a másiknak az elsődleges kulcsa lesz. Ha nem tartalmazzák, akkor az operandusok elsődleges kulcsainak konkatenációja, esetleg annak valamely része adja az elsődleges kulcsot.

A gyakorlatban az egyesítés legtöbbször egyetlen egy közös oszlop ($R.k_i = S.k_i$) azonos értékeinek összekapcsolása alapján történik ($i=1$). A feladat elején tárgyalt példában és a következő oldalon a 4.6. ábrán is egy ilyen join látható.

Megjegyezzük, hogy ha a join-oszlopban egy sorban mindkét relációban NULL érték van, akkor ezek nem azonosak. Az operandusoknak a join oszlop(ok)ban NULL értéket tartalmazó sorai nem szerepelnek az eredményben.

A természetes join szimmetrikus. A join operandusainak sorrendje lényegtelen.

További példák a join alkalmazására:

$\text{MITHALLGAT JOIN [PROJECT HALLGATÓ (VNÉV, KNÉV, HID)] (HID)}$

$\text{PROJECT \{[TANTÁRGY JOIN MITHALLGAT (TID)] JOIN HALLGATÓ (HID) [VNÉV, KNÉV, OKTATÓ]\}}$

Az első join megadja MITHALLGAT adatai mellé a hallgató nevét is. A második (három reláció összekapcsolása) megadja a hallgatók neve mellé az oktatóinak a nevét is. Azok a hallgatók, akiket senki sem tanít, és azok az oktatók, akiknek egy tanítványa sincs, nem szerepelnek ebben a relációban. Ezeket csak a következő pontban ismertetendő külső join segítségével kaphatjuk meg.

nevekre a feltételek nem teljesülnének, akkor az egyesítés előtt előbb az átnevezés műveletet kell alkalmaznunk legalább az egyik operandusra.

¹ A gyakorlatban a műveletek nem így és nem ebben a sorrendben hajtódnak végre, hisz így mindig létrejönne a keresztszorzat, amit pedig célszerű hatékonysági okok miatt elkerülni.

A jelölésben $R.k_j$ az R , $S.k_j$ az S reláció k_j oszlopát jelenti.

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 (k_1)				
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

4.6. ábra: A join művelet. Az egyesítés alapjául a mindkét relációban szereplő k_1 oszlop szolgál.

4.5.6.2. Külső join

A TANTÁRGY JOIN MITHALLGAT (TID) relációban minden MITHALLGAT sorhoz tartozik egy és pontosan egy TANTÁRGY sor, mert MITHALLGAT-ban TID nem lehet NULL érték, és idegen kulcsként hivatkozik TANTÁRGY elsődleges kulcsára. (ld. 4.4.3.1. pont). Azokat a tantárgyakat azonban, amelyeket senki nem vett fel, a természetes join nem tartalmazza, mivel ezek TID azonosítójához nem kapcsolódik egy MITHALLGAT rekord sem. Hasonlóképpen nem kapjuk meg a HALLGATÓ JOIN MITHALLGAT (HID) relációból azokat a hallgatókat, akik egy tantárgyat sem vettek fel, holott gyakran ezekre is szükségünk lehet.

Az ilyen feladatok egyszerű megoldására definiálták a *külső* vagy *Outer join* műveletet. Ennek eredménye nem csak azokat a sorokat tartalmazza, amelyekben *mindkét* join táblázat soraira érvényes a join feltétel, hanem azokat is, amelyek vagy

- a join bal oldalán levő relációban szerepelnek, de a jobb oldaliban nincs olyan sor, amelyik a join feltételnek megfelel. Az eredményben a jobb oldali operandus nem join oszlopai a NULL értéket veszik fel. Ez a *LEFT OUTER JOIN* vagy röviden *LEFT JOIN*, *egyesítés balról*. A 4.5.6.1. pontban lévő példa a MITHALLGAT és HALLGATÓ táblázatok (ebben a sorrendben) LEFT OUTER JOIN-jának eredménye is lehetne, mivel az idegen kulcs feltétel miatt MITHALLGAT minden sorához tartozik vele azonos HID értékkel rendelkező sor a HALLGATÓ-ban.
- a join jobb oldalán levő relációban szerepelnek, de a bal oldaliban nincs olyan sor, amelyik a join feltételnek megfelel. Az eredményben ennek oszlopai a NULL értéket veszik fel. Ez a *RIGHT OUTER JOIN* vagy röviden *RIGHT JOIN*, *egyesítés jobbról*. Az említett példában a MITHALLGAT és HALLGATÓ táblázatok (ebben a sorrendben) RIGHT OUTER JOIN-jának eredményében szerepel az 55555 azonosítójú hallgató is az alábbi értékekkel:

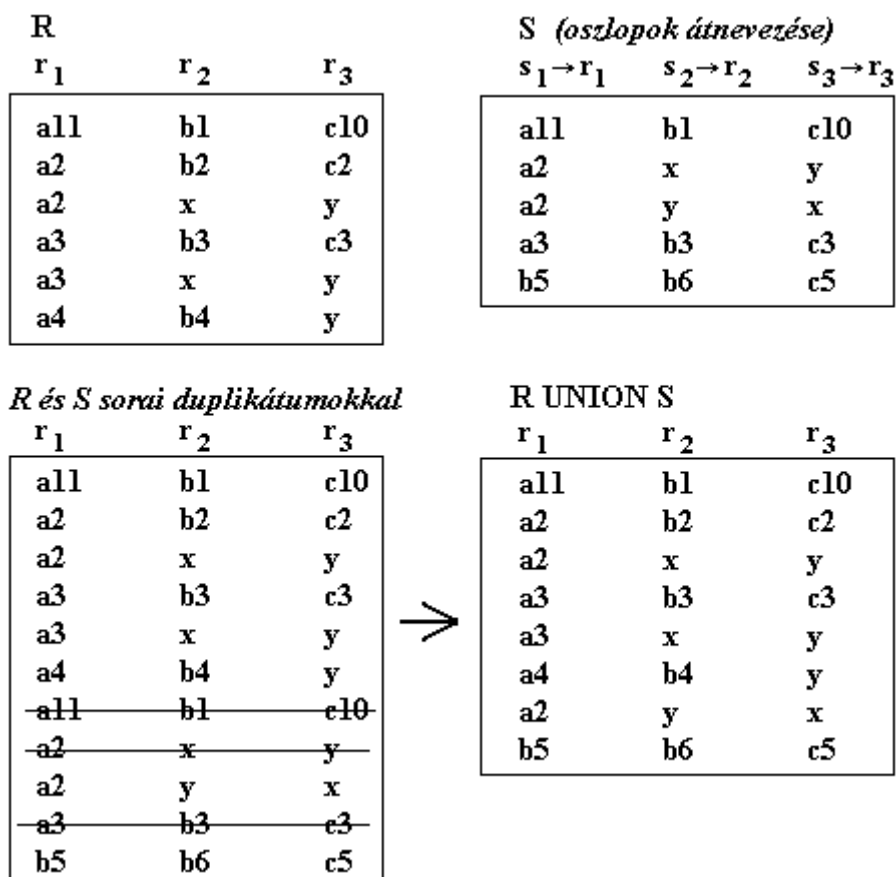
HID	TID	GYAKPONT	OSZTALYZAT	VNEV	KNEV	SZDÁTUM	EVF
55555	NULL érték	NULL érték	NULL érték	Elek	Etelka	1980-11-11	1

- a join bármelyik relációjában szerepelnek. Ez a *FULL OUTER JOIN* vagy röviden *FULL JOIN*, teljes egyesítés. Az eredmény a két join komponens *left outer join*-jának és *right outer join*-jának *unió*-ja (ld. 4.5.7. pont) lesz. Tartalmazza mindazon sorokat, melyek vagy a balról vagy a jobbról való egyesítésben szerepelnek. Azok a sorok, amelyek mindkettőben benne vannak, (a természetes join eredménye) csak egyszer jelennek meg a végeredményben. A példában a két reláció jobboldali és teljes join-ja megegyezik.

A külső join nem alapl művelet. A jobb- és a baloldali join nem is szimmetrikus. Ezekben a relációk sorrendje nem cserélhető fel.

4.5.7. Unió (UNION)

Gyakran szükségünk van arra, hogy azonos egyedtípusokat tartalmazó táblázatok sorait együtt használjuk. Az egyetemi példánknál maradva, tegyük fel, hogy minden egyetem ezen a módon felépíti a saját, önálló adatbázis részét és szükségünk van például minden egyetemi hallgató adataira, azaz az egyes egyetemek HALLGATÓ táblázatainak összességére, az összes HALLGATÓ táblázat minden sorára. Ezt kapjuk meg a különböző egyetemek HALLGATÓ relációinak *uniójaként*¹.



4.7. ábra: Unió művelete

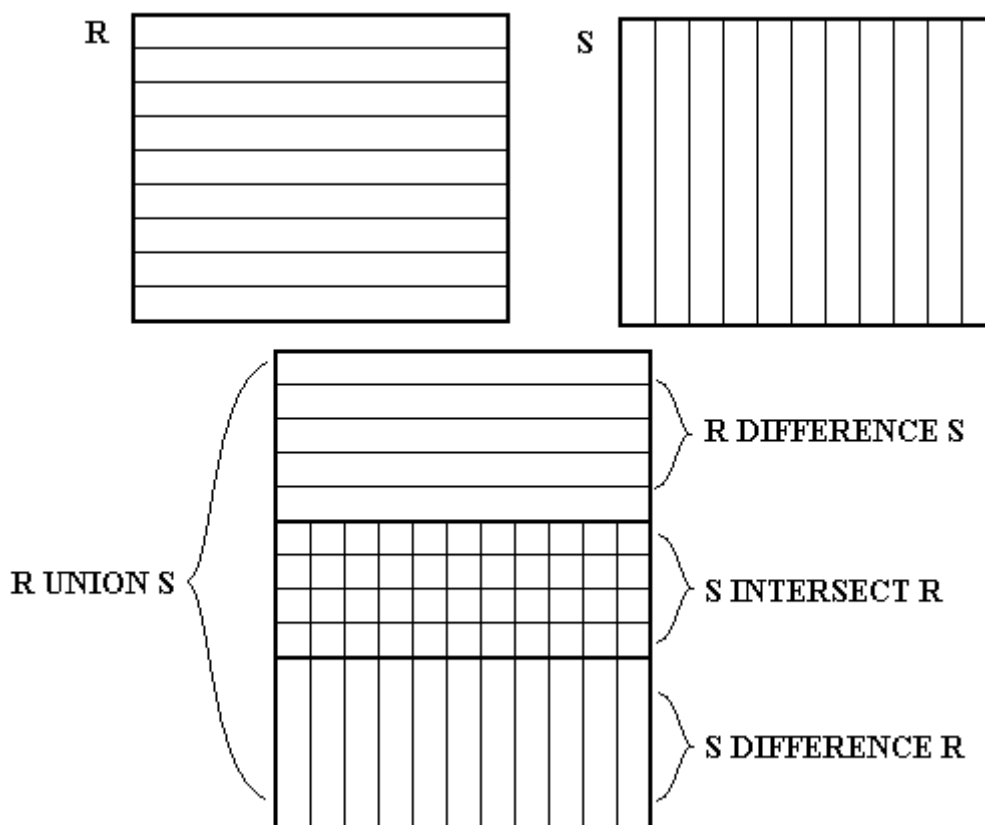
¹ Ha ezek egy adatbázisban szerepelnek, akkor gondoskodnunk kell a reláció nevek egyediségéről. Ezt meg tehetjük úgy, hogy a HALLGATÓ elé tesszük az egyetem nevét, pl. CORVINUS.HALLGATÓ, ELTE.HALLGATÓ.

Az *unió* kétrelációs, elemi művelet. Két reláció unióját, $R \text{ UNION } S$ -t akkor tudjuk képezni, ha azok *unió-kompatibilisek*, azaz mindkét reláció ugyanazon attribútumokat tartalmazza ugyanabban a sorrendben. Ebben az esetben az uniót azok a sorok alkotják, melyek vagy az R , vagy az S relációban bent vannak, de azok, amelyek mindkettőben megtalálhatóak csak egyszer szerepelnek (ld. 4.7. ábra). Az eredmény-relációnak mind az oszlopai, mind az elsődleges kulcsa megegyezik az operandus relációk (közös) oszlopaival illetve elsődleges kulcsával.

Ha az oszlopoknak csak a neve vagy a sorrendje különbözik, akkor a RENAME utasítással unió-kompatibilissé tehetjük a komponenseket. Ha más attribútumokból állnak, formailag akkor is unió-kompatibilissé tehetők új oszlopok definiálásával, melyeknek alapértelmezést, általában NULL értéket adunk.

4.5.8. Metszet (INTERSECT) és különbség (DIFFERENCE)

Mindkét művelet kétrelációs művelet. Csak unió-kompatibilis relációkon valósíthatók meg. Az eredményreláció oszlopai és elsődleges kulcsa – az unióval hasonlóan – megegyeznek az operandus relációk (közös) oszlopaival illetve elsődleges kulcsával.



4.8. ábra: Unió, metszet és különbség ábrázolása „rendezett relációkkal”.

R és S azonos sorai a rendezés következtében lefedik egymást.

Az $R \text{ INTERSECT } S$ reláció R és S azon és csak azon sorait tartalmazza, melyek *mindkét* relációban benne vannak.

Az $R \text{ DIFFERENCE } S$ reláció sorait R -nek azon és csak azon sorai alkotják, amelyek *bent vannak* R -ben, de *nincsenek bent* S -ben. Ebben a műveletben az operandusok nem cserélhetők fel.

Azokat a hallgatókat, akik az ELTE-re és a Corvinus-ra is járnak a

CORVINUS.HALLGATÓ INTERSECT ELTE.HALLGATÓ

reláció szolgáltatja¹.

A műveletek nem elemiek. Könnyen belátható, hogy például az R és S relációk metszete előállítható az alábbi műveletek eredményeként:

$R \text{ JOIN RENAME } S (s_1, r_1; s_2, r_2; \dots, s_n, r_n) (,r_1, r_2, \dots, r_n)$

azaz képezzük a két reláció join-ját az összes oszlop egyezősége alapján.

Szemléletesen úgy tudjuk érzékeltetni az unió, a metszet és a különbség műveletét, ha az ábrázolásban eltekintünk attól, hogy a sorok sorrendjének nincsen szerepe, és a két reláció megegyező sorait R -nek a végére, S -nek pedig az elejére tennénk. (ld. 4.8. ábra)

Az uniót, metszetet és a különbséget az SQL nyelvben a SELECT utasításban különböző SELECT utasítások eredményének összevonásakor (ld. 5.3.4.4. pont) használjuk.

4.5.9. Bővítés (EXTEND)

Létező relációból új relációt képezhetünk azáltal, hogy új oszlopot adunk hozzá. Az utasítás ismételt alkalmazásával akárhány oszloppal bővíthetünk egy relációt.

Általános formája:

$\text{EXTEND } R \text{ TO } R_{ij} \text{ ADD oszlopképzés módja AS } r_{ij},$

ahol r_{ij} a módosítás eredményeként létrejövő R_{ij} reláció új oszlopa.

Az új oszlopot gyakran a reláció meglévő oszlopaiból képezzük aritmetikai műveletekkel, pl. az eredeti relációban szereplő ALAP és ÉRTÉK oszlopokból a SZÁZALÉKOS-ÉRTÉK = $100 \cdot \text{ÉRTÉK} / \text{ALAP}$ oszlopot. Ilyenkor R_{ij} csak virtuális reláció, nézet vagy lekérdezés eredménye lehet. Ezt a műveletet az SQL nyelvben a nézetek létrehozásával, a CREATE VIEW utasítással (ld. 5.14. táblázat: pont) valósítjuk meg.

Ha egy fizikailag is létező táblázatot adatainak, nevének és szerkezetének a megtartásával csak egy új oszloppal akarunk bővíteni ($R = R_{ij}$), akkor az oszlopképzés módja az új oszlop definícióját jelenti. Természetesen ilyenkor az új oszlopnak alapértelmezést kell adnunk, vagy meg kell engednünk benne a NULL értéket. A táblázat már eddig létező soraiban ez lesz az új oszlop értéke. Ezt a műveletet a gyakorlatban az SQL ALTER TABLE utasítással (ld. 5.6.2.3. pont) segítségével hajthatjuk végre. Ez az utasítás lehetőséget ad meglévő oszlop törlésére is, amit a „bővítés” egy speciális formájának is tekinthetünk.

4.5.10. Csoportképzés (SUMMARIZE)

A gyakorlati munkában igen sokszor előfordul, hogy nem közvetlenül a kiválasztott adatokra, hanem azok különböző csoportjaira jellemző adatokra, úgynevezett aggregátumokra akarunk értékeket megkapni. Ilyen lehet például egy személyi nyilvántartásban az összes dolgozó fizetésének az összege, átlaga, a legalacsonyabb, vagy a legmagasabb fizetés, vagy ugyanezek az értékek a dolgozók egyes csoportjaira, például az azonos osztályokon, vagy azonos munkakörben dolgozókra vonatkoztatva.

Az ilyen relációkat tudjuk létrehozni a csoportképzés műveletével.

¹ Ez természetesen csak akkor igaz, ha regisztráláskor a hallgatóknak minden adatát (beleértve az azonosító HID-t is) azonosan veszünk fel mindkét egyetemen.

Ez az összetett művelet a bővítés általánosítása. Az új reláció sorai az eredeti soraiból csak a csoportra jellemző értékeket (pl. maximum), vagy a képzett értéket (pl. csoport átlag) tartalmazzák. Általános formája:

```
SUMMARIZE R TO Rgroup GROUP BY csoportosítás módja
ADD f1(r1) AS rgroup-1, f2(r2) AS rgroup-2, ... fi(ri) AS rgroup-i
```

A *csoportosítás módja* határozza meg, hogy milyen csoportokra (pl. osztályonként, munkakörönként) akarjuk meghatározni az egyes r_j attribútumoknak (pl. fizetés, életkor) az $f_j(r_j)$ csoportképző művelettel (pl. fizetések átlaga, életkor maximuma) a kiválasztott csoportokra jellemző értékét.

Ha a csoportosítás módját elhagyjuk, akkor az eredmény az egész relációra vonatkozik. Ekkor R_{group} -nak csak egyetlen sora lesz. Ha megadjuk, akkor annyi sort kapunk, ahány csoportot tudunk R soraiból képezni (ahány osztályunk, vagy munkakörünk van a személyi nyilvántartásban).

R eredeti oszlopaiból csak azokat tartjuk meg, amelyeknek értékeit a csoportképzés után is értelmezhetjük¹.

```
SUMMARIZE HALLGATÓ TO ÉVFOLYAM-ÉLETKOR
GROUP BY ÉVFOLYAM
ADD ÉVF AS ÉVFOLYAM, MAXIMUM(SZDÁTUM) AS LEGIFJABB,
MINIMUM(SZDÁTUM) AS LEGIDŐSEBB
```

előállítja azt a relációt, amelyik minden évfolyamra egy sort ad, amelyik az évfolyamot, és az évfolyamra járó hallgatók születési dátumából a legnagyobbat és a legkisebbet tartalmazza. A reláció oszlopainak neve: ÉVFOLYAM, LEGIFJABB, LEGIDŐSEBB.

A csoportosítást az SQL-ben a SELECT utasítás (ld. 5.6.3.2. pont) GROUP BY klauzulájával valósítjuk meg.

4.5.11. Kijelölés

Célszerű eddigi műveleteinket kiegészíteni még egy továbbival. Ennek segítségével a relációs műveletek által létrehozott relációnak egy nevet tudunk adni. Ezáltal egy új, általában virtuális objektumot hozunk létre az adatbázisban, mellyel már egyszerűen, a nevére való hivatkozással dolgozhatunk, függetlenül attól, hogy előállítás a valós adatokból milyen bonyolultan, hány lépésben történik.

Formája:

```
CREATE R(r1,r2,...,rn) AS relációs műveletekkel előállított reláció
```

A tetszőleges *relációs műveletek* eredménye az $r_1, r_2, \dots, r_n$ oszlopokból álló R reláció. Az SQL nyelvben ennek a CREATE VIEW utasítás (ld. 5.14. táblázat.: pont) felel meg.

¹ Ha például osztályonként csoportosítunk, akkor az OSZTÁLY oszlopot megtarthatjuk, hiszen ez ugyanúgy, mint a csoport értékek (pl. osztályonkénti átlag fizetés, vagy a maximális fizetés) egyértelműen egy adott osztályhoz kapcsolódik. Az egyes dolgozók nevei viszont nem lehetnek elemei az R_{group} relációnak, mert az osztály sok dolgozója közül nem tudjuk eldönteni, ki legyen az az egy, aki az osztályra jellemző sorban szerepeljen.

A relációs műveletek hasznosságának illusztrálására állítsuk elő azt a relációt, amelyik az egyetemi adatbázisunk *minden* adatát tartalmazza. Ez az alábbi műveletekkel hozható létre:

```
[HALLGATÓ FULL JOIN MITHALLGAT (HID)]  
FULL JOIN TANTÁRGY (TID)
```

A FULL JOIN-ra azért van szükségünk, hogy megkapjuk azokat a hallgatókat is, akik egy tantárgyat sem vettek fel és azokat a tantárgyakat is, melyeket senki sem hallgat.

A továbbiakban, hogy ne kelljen esetleg különböző táblázatokból összeszedni az adatokat, ezzel a relációval, melyben minden információ benne van, akarunk dolgozni. Ezért külön nevet (TELJES-OKTATÁS) adunk neki.

```
CREATE TELJES-OKTATÁS  
(VNÉV, KNÉV, HID, SZDÁTUM, ÉVF, TID, GYAKPONT, OSZTÁLYZAT,  
TMEGNEV, OKTATÓ, ÓRASZÁM, VIZSGA) AS  
[HALLGATÓ FULL JOIN MITHALLGAT (HID)]  
FULL JOIN TANTÁRGY (TID)
```

Ebből azoknak a hallgatóknak a nevét és az azonosítóját, akik egy tantárgyat sem vettek fel a

```
PROJECT [RESTRICT TELJES-OKTATÁS WHERE TID IS NULL]  
(VNÉV, KNÉV, HID)
```

művelettel kaphatjuk meg¹.

Az elsőéves hallgatóknak az egyes tantárgyakból elért átlagát², és a legjobb osztályzatot a

```
SUMMARIZE [RESTRICT TELJES-OKTATÁS WHERE ÉVF = 1]  
GROUP BY TMEGNEV ADD  
TMEGNEV AS TANTÁRGY, ÁTLAG(OSZTÁLYZAT) AS ÁTLAG,  
MAXIMUM(OSZTÁLYZAT) AS LEGJOBB-JEGY
```

relációs műveletsorozat állítja elő.³

Ha nem hoztuk volna létre a TELJES-OKTATÁS (virtuális) relációt, akkor mindkét esetben join-ok segítségével kellene összeszednünk a különböző relációkból az összetartozó adatokat.

4.6. Normalizálás

Az adatbázis tervezés első lépése az, hogy összegyűjtjük, melyek azok az adatok, amelyekre a különböző feldolgozásoknak szükségük van, vagy a belátható jövőben szükségük lehet. A következő lépés az, hogy az így összeállt „ős-káoszt” megpróbáljuk áttekinthető, jól kezelhető kisebb egységekbe szervezni. Ez a két lépés adatmodelltől függetlenül az adatbázis tervezés alfája és ómegája. A relációs modell egyeduralmódóvá válásának egyik oka az volt,

¹ TELJES-OKTATÁSBAN azokhoz a HALLGATÓ azonosítókhoz, melyekhez nem tartozik MITHALLGAT-ban érték (azaz egy tantárgyat sem vettek fel), TID értéke a FULL JOIN következtében NULL érték lesz.

² Azok a sorok, melyekben OSZTÁLYZAT NULL ÉRTÉK nem vesznek részt az átlag képzésében.

³ A szögletes zárójelben lévő RESTRICT kiválasztja az elsős hallgatókat, és a TMEGNEV szerint csoportosítást csak ezekre végezzük el. Az ÁTLAG(OSZTÁLYZAT) és MAXIMUM(OSZTÁLYZAT) jelöli azokat a műveleteket, melyek az egyes csoportok osztályzataira kiszámolják az átlagot, illetve kiválasztják a legmagasabb értékét. Az új relációban a tantárgy nevét a TANTÁRGY oszlop tartalmazza.

hogyan az adatok észszerű, a gyakorlatban jól használható csoportosításához, a relációk kialakításához elméletileg megalapozott, formálisan algoritmizálható eljárást adott. Ezt az eljárást nevezzük *normalizálásnak* vagy *normalizációnak*.

A normalizálás mindig az adatok logikai szintjén történik. Általában több lépésből áll, melyek jelentős részét gyakorlott adatbázis-tervező kizárólag a józan eszére és tapasztalatára alapozva minden elmélet nélkül elvégezne. A normalizálási szabályoknak az ismerete éppen abban segít, hogy ne alkalmi megoldásokkal kelljen kísérletezni, és nehogy éppen emiatt a tervezéskor valamilyen a gyakorlati munkában problémát, hibát okozó adatsortosításokat hozzunk létre.

Természetesen a normalizálás sem csodaszer. Szisztematikus alkalmazásával sem tudunk minden gondot mindig megoldani. Sokszor kerül szembe az adatbázis-felügyelő azzal a problémával, hogy az elvileg tökéletesen normalizált relációkkal a rendszer lassan működik. Ilyenkor gyakran kompromisszumra kényszerül. A hatékonyság érdekében megelégszik a logikailag kevésbé áttekinthető és kevésbé tiszta, de nagyobb hatékonyságot biztosító alacsonyabb fokú normalizálással. Az adattárházak tervezésénél ez általános is.

Mindezek ellenére általános érvénnyel kijelenthetjük, hogy a jó normalizálás következtében

- megszűnik a redundancia, az adatok tároló igénye kisebb lesz,
- az elemi adatokat gyorsabban és kevesebb hibalehetőséggel változtathatjuk meg,
- az adatbázis logikailag áttekinthetőbb lesz.

A könnyebb érthetőség érdekében a normalizálás 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.

4.6.1. Függőségek

Bármely relációban egyes attribútumok értékei függnének más attribútumok értékeitől. Ez a függés lehet

- *Erős*, amikor a függőség kötelező érvényű, mindig fennáll. Ilyen például a hallgató egyetemi azonosítója és neve közti függőség. Minden azonosítóhoz tartozik egy meghatározott név, és minden névhez tartozik azonosító.
- *Gyenge*, amikor a függőség nem kötelező érvényű. Csak akkor áll fenn, ha a függőségi kapcsolatban lévő attribútumoknak vannak értékei. Ilyen például a hallgató egyetemi azonosítója és a sportköri igazolványának száma közti függőség. Nem minden hallgató tagja a sportkörnek, de ha igen, akkor a két attribútum között egyértelmű függőség van.

Hangsúlyozni kívánjuk, hogy a függőségnek az adatbázisnak nem csupán egy adott pillanatban aktuális adataira kell fennállni, hanem mindig, minden időben. Természetesen a függőségi feltételek megváltoztathatók, de ez az adatmodell módosítását is eredményez(het)i.

A normalizálás folyamatánál a függőségeket mindig egy reláción belül vizsgáljuk. Ezeket is a relációk közti függőségekhez hasonlóan, a 3.5. alfejezetben ismertetett kapcsolatokkal jellemezzük.

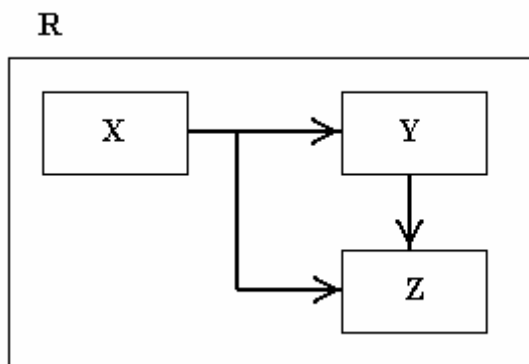
Ha az R relációban az egyik attribútum (X), a független változó értéke egyértelműen meghatározza egy másik attribútum (Y), a függő változó értékét, akkor azt mondjuk, hogy Y az R relációban funkcionálisan függ X-től.

Itt és a további definíciókban is mind X, mind Y lehet összetett, azaz állhat több oszlopból is.

A funkcionális függés jelölése:

$$R.X \rightarrow R.Y$$

Szokásos a függőséget grafikusán, *funkcionális függőségi diagrammban*, röviden *függőségi diagrammban* is ábrázolni. Ezt látjuk a 4.9. ábrán. A független attribútumból nyíl mutat a függőre. Az ábrán Y és Z is függ funkcionálisan X -től, Z pedig Y -től is.



4.9. ábra: Funkcionális függés ábrázolása.

Természetesen arról, hogy az egyes attribútumok között milyen függőségek állnak fenn, az elvi modell semmit sem tud mondani. Ezt az adatbázis tervezőjének kell kiderítenie az adott feladat körülményeinek az ismeretében. Egy árura adott kedvezmény mértéke függhet a vétel időpontjától (mindenkire érvényes akciós ár), a vevő személyétől, vagy a vásárolt mennyiségtől, esetleg mind a háromtól is.

A reláció definíciójából következik, hogy minden attribútum függ az elsődleges kulcstól és nyilvánvalóan – ha van, akkor – az összes alternatív kulcstól is.

A gyakorlatban elterjedtebb a funkcionális függőségnek a fent megadottal egyenértékű definíciója:

Egy R relációban az Y oszlop akkor és csak akkor függ funkcionálisan az X oszloptól, ha bármely két sorban, amelyekben X értéke azonos, azokban az Y értékek is mindig azonosak.

Fordítva természetesen az állítás nem kell, hogy igaz legyen. Azonos Y értékekhez tartozhatnak különböző X értékek. Így például a TANTÁRGY táblázatban (ld. 4.2. alfejezet) a tantárgy megnevezése egyértelműen meghatározza az óraszámot, de ugyanahhoz az óraszámhoz több tantárgynév is tartozhat.

Az X attribútumot, a relációnak azt az oszlopát, amelyik meghatározza más attribútum(ok), oszlop(ok) értékét a reláció *meghatározó elemének* (determinant, determináns) nevezzük.

A funkcionális függés egy 1:N kapcsolatot jelent a reláción belül. Ennek speciális esete az 1:1 kapcsolat, amikor az R relációban nem csak X értéke határozza meg egyértelműen Y értékét, hanem Y értéke is meghatározza egyértelműen X -et. Abban az esetben, ha X elsődleges kulcs, akkor Y alternatív kulcs. A normalizálási eljárásban az alternatív kulcsoknak nem vizsgáljuk a függési viszonyait. Az elsődleges kulccsal azonos funkciójú attribútumnak tekintjük, és azzal azonos módon kezeljük. Az egyedüli eltérés, hogy a dekomponálással (ld. 4.6.3. és 4.6.4. pont) létrehozott új relációkba a kapcsolat biztosítására csak az elsődleges kulcsot vesszük át. Példa kölcsönös függőségre az árukód és az áru egyértelmű megnevezése, vagy a TAJ szám, adószám, személyi szám közti egyértelmű kapcsolat.

Az 1:1 kapcsolatot alternatív kulcsoknál a reláción belül az alternatív kulcsokra létrehozott egyedi index-szel valósíthatjuk meg. Nem alternatív kulcsoknál, illetve 1:N kapcsolatot jelentő függőségnél azonban ennek érvényességét egy reláción belül az adatbázis kezelő rendszer nem tudja automatikusan biztosítani. Ezt csak az alkalmazásokba beépített külön ellenőrző eljárással, programmal tehetnénk meg. A normalizálási algoritmusnak a lényege az, hogy a reláción belüli függőségeket a relációk átalakításával megszüntetjük. A

helyesen normalizált adatbázisba nem kerülhetnek be egymással logikai ellentmondásban lévő adatok.

Hangsúlyozzuk, hogy mind a függő, mind a független attribútum lehet összetett is. Tételezzük fel, hogy az adatbázisunkban a MITHALLGAT táblázatot (ld. 4.2. alfejezet) úgy definiáltuk, hogy belevettük a tantárgy nevét és oktatóját is. Természetesen az elsődleges kulcs (hallgató-azonosító, tantárgy-azonosító) egyértelműen meghatározza az osztályzatot, az oktatót és a tantárgy megnevezését is, de az utóbbi kettőt egyértelműen meghatározza az elsődleges kulcs egy része, a tantárgy azonosítója is. Ezt úgy mondjuk, hogy az osztályzat *funkcionálisan teljesen függ* a (hallgatói-azonosító, tantárgy-azonosító) független oszlop kombinációtól, míg az oktató és a tantárgy neve *funkcionálisan függ*, de csak *funkcionálisan nem teljesen függ* tőlük.

Általánosan megfogalmazva:

Az R relációban az Y attribútum akkor és csak akkor függ funkcionálisan teljesen az X (összetett) attribútumtól, ha funkcionálisan függ X -től, de nem függ X egyetlen valódi összetevőjétől sem.

Ha X nem összetett, akkor a funkcionális és a teljes funkcionális függőség ugyanazt jelenti.

További példa funkcionális, de nem teljes funkcionális függőségre egy többtétéles szállítólevél, amely többek között tartalmazza a szállítólevél számát, az áru kódját (a kettő együtt az elsődleges kulcs), az áru nevét, mennyiségét, árát és a szállítás dátumát. Itt a mennyiség funkcionálisan teljesen függ az elsődleges kulcstól, míg a többi már nem. A dátumot a szállítólevél száma, az áru nevét és árát az árukód, az elsődleges kulcs egyik összetevője már egyértelműen meghatározza.¹

Ezen alapvető függőségi kapcsolatok mellett szükségünk lesz még a *többértékű funkcionális függőség* definiálására is. Ekkor a független változó a függőnek nem egyetlen egy értékét határozza meg, hanem a lehetséges értékeknek egy egyértelműen meghatározott részhalmazát. Például a TANTÁRGY relációban egy tantárgyat nem csak egy oktató taníthat, hanem több is, de nem mindegyik. Adott tárgyat csak meghatározott oktatók taníthatnak.

Az X, Y, Z attribútumokkal rendelkező R relációban Y akkor és csak akkor van többértékű funkcionális függőségben X -től, ha bármely adott X és Z értékhez tartozó lehetséges Y értékek csak X értékétől függenek, Z -től függetlenek.

A többértékű funkcionális függés jelölése

$$R.X \twoheadrightarrow R.Y$$

A továbbiakban megmutatjuk, hogyan tudjuk az adatmodellünket a függőségek ismeretében a normalizálás segítségével logikailag tisztább, gyakorlatilag használhatóbb állapotba hozni.

4.6.2. Első normál forma (1NF)

A normalizálás folyamatának megértésére készítsük el az alábbi feladat adatmodelljét:

Nyilvántartást kell készítenünk a vállalat alkalmazottairól. Minden alkalmazottról szükségünk van az alábbi adatokra (zárójelben az oszlop elnevezése az adatbázisban):

¹ Feltételezzük, hogy az áru itt feltüntetett ára (alapár) mindenkinek ugyanaz, legfőljebb ebből egyesek kedvezmény(ek)e)t kaphatnak, ami(k) további attribútum(ok) lehet(nek). Amennyiben időben változó árakkal kell számolnunk, akkor az ár például az (árukód, szállítási-dátum), de lehet, hogy egy (árukód, rendelési-dátum) kombinációtól függ. Ezeket a feltételeket természetesen a konkrét logikai tervezéskor kell észrevenni és beépíteni az adatmodellbe.

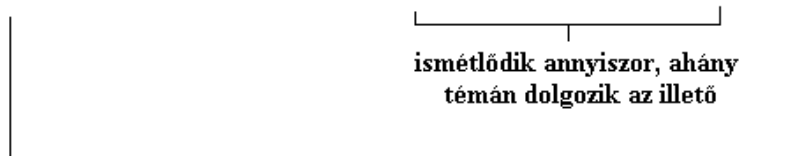
- személyi azonosító szám (SZSZÁM),
- vezetéknév (VEZ-NÉV),
- keresztnév (KER-NÉV),
- osztály kódja, ahol dolgozik (OSZT),
- osztály megnevezése (OSZT-NÉV),
- téma kódja, amelyen dolgozik (TKÓD),
- téma megnevezése (TÉMA),
- munkaidejének hány százalékában dolgozik a témán (MEGOSZL),
- születésének éve (SZÉV),
- 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 az 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 gyűjtjük be. Így az induló táblázatunk osztályonként egy-egy rekordot, „sort” tartalmaz, melynek első mezője az osztály kódja, második az osztály neve, utána annyiszor jönnek az egyes dolgozók fent felsorolt adatai, ahány dolgozó van az osztályon. Végül az utolsó mező az osztály vezetőjének azonosító száma. (ld. 4.10. ábra)

OSZT OSZT-NÉV SZSZÁM VEZ-NÉV KER-NÉV TKÓD TÉMA MEGOSZL SZÉV FŐNÖK



ismétlődik annyiszor, ahány dolgozó van az osztályon

1. rekord

P01 Pénzügy	11111 Andrásfalvi Béla B1 Bér 40 O2 Oktatás 60 1934	11211 Bártfai ...

Tóth Csilla ... E1 Ellenőrzés 70 1969	91526 Tarnay Gyula T2 Tervezés 100 1932	91526

2. rekord

E01 Eladás	99998 Vigécz Jenő K1 Kereskedelem 100 1945	99998

3. rekord

T01 Titkárság	01526 Mohácsi Győző T2 Tervezés 100 1971	...

01492 Tojásos Kristóf ... U2 Utazások 80 1966	69690

4.10. ábra: Az adatbázis felépítéséhez szükséges adatok osztályonként összegyűjtve.

Ez a „táblázat”, melynek sorai az egyes osztályok adatai, nem reláció.¹ Egymásba ágyazott ismétlődő mezők vannak benne. Először ismétlődnek az alkalmazottak, minden osztályon belül annyiszor, ahány alkalmazott dolgozik az osztályon. Ezután az egyes alkalmazottakat tartalmazó adatsorokon belül ismétlődnek a témával kapcsolatos adatok, minden alkalmazottnál annyiszor, ahány témán dolgozik.

Ahhoz, hogy a relációs modellben ennek a táblázatnak az adataival dolgozni tudjunk, át kell ezt alakítanunk „valódi” 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 adatait csak egy oszlop tartalmazza ismétlődés nélkül. Az ismétlődő mezők minden egyes értékéhez egy önálló sort rendelünk hozzá, melynek a többi adata azonos az ismétlődő elem ehhez az értékéhez tartozó többi, nem ismétlődő oszlop értékével.

Az eljárás algoritmus a következő:

Eredeti reláció oszlopai	A	B	C (ismétlődő)	D
Átalakított reláció oszlopai	A	B	C	D
Eredeti sor	a	b	c ₁ , c ₂ , c ₃	d
Átalakított sorok	a	b	c ₁	d
	a	b	c ₂	d
	a	b	c ₃	d

ahol *a*, *b*, *d* a sor *A*, *B* és *D* nem ismétlődő oszlopainak az értékei, *c*₁, *c*₂, *c*₃ pedig ugyanezen sorban a *C* különböző ismétlődő értékei. Az így kapott táblázat már megfelel a relációk legáltalánosabb feltételeinek. Azt mondjuk, hogy a reláció *első normál formában* (1NF) van. A 4.10. ábra első normál formába átalakított adatait a 4.11. ábrán láthatjuk. Az egymásba ágyazott ismétlődést elvileg két lépésben oldottuk fel. Először az alkalmazottak, majd ezután a témák szerinti ismétlődést elimináltuk. Az ábrán a két művelet együttes eredményét láthatjuk.

ALKALMAZOTT – 1NF

OSZT	OSZT-NÉV	SZSZÁM	VEZ-NÉV	KER-NÉV	TKÓD	TÉMA	MEGOSZL	SZÉV	FŐNÖK
P01	Pénzügy	11111	Andrásfalvi	Béla	B1	Bér	40	1934	91526
P01	Pénzügy	11111	Andrásfalvi	Béla	O2	Oktatás	60	1934	91526
P01	Pénzügy	11211	Bártfai	Pál	U2	Utazás	100	1937	91526
P01	Pénzügy	12110	Tóth	Csilla	B1	Bér	30	1969	91526
P01	Pénzügy	12110	Tóth	Csilla	E1	Ellenőrzés	70	1969	91526
P01	Pénzügy	91526	Tarnay	Gyula	T2	Tervezés	100	1932	91526
E01	Eladás	99998	Vigécz	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	Utazások	80	1966	69690

4.11. ábra: Első normál formába hozott reláció.

Az ismétlődő mezők kiküszöbölésével a 4.10. ábra adatait az ALKALMAZOTT-1NF 1NF relációba hoztuk. Az elsődleges kulcs a (SZSZÁM, TKÓD) oszlopkombináció.

¹ Szokás ezt nulladik normál formában lévő (0NF) relációnak is nevezni.

Általánosan azt mondhatjuk, hogy egy táblázat adatai első normál formában vannak, 1NF relációt alkotnak, ha teljesülnek a 4.1. alfejezetben részletezett feltételek, azaz

- minden oszlop egy és csak egy attribútumot jelent,
- minden sor különbözik,
- az attribútumok sorrendje minden oszlopban ugyanaz,
- nincsenek ismétlődő mezők,
- minden sorhoz tartozik (legalább egy) egyedi kulcs, melytől az összes többi oszlop funkcionálisan függ.¹

Minden relációs adatbázis-kezelő rendszer megköveteli, hogy az adatok legalább első normál formában (1NF) legyenek. Bár ezen a normalizáltsági szinten még sok probléma és hiba fölmerülhet, ennél magasabb normalizálás már nem szükséges, de – mint azt a következő pontokban látni fogjuk – általában javasolt.

Egy ilyen hibát rögtön észrevehetünk. Téves adatbevitel, vagy nem az összes adaton konzisztensen végrehajtott módosítás miatt a *Titkárság* főnöke az egyik sorban 01492, míg az összes többiben 69690. Az adatbázisból nem tudjuk eldönteni, melyik a helyes érték.

Ahhoz, hogy az 1NF feltételei valóban teljesülnek-e a példában, meg kell még vizsgálni, van-e az ALKALMAZOTT-1NF-nek elsődleges kulcsa. A személyi azonosítószám (SZSZÁM) önmagában még nem az, mert bár a csak a személyhez kötődő adatokat (VEZ-NÉV, KER-NÉV, SZÉV) és az osztállyal kapcsolatos adatokat (OSZT, OSZT-NÉV, FÖNÖK) egyértelműen meghatározza, a témához kötődőket (TKÓD, TÉMA) és a MEGOSZLást már nem, mivel egy személy több témán is dolgozhat. De az (SZSZÁM, TKÓD) oszlopkombinációtól már minden további oszlop funkcionálisan függ. Ezért ez lesz az elsődleges kulcs. A TKÓD ↔ TÉMA egyértelmű kölcsönös függőség következtében az (SZSZÁM, TÉMA) alternatív kulcs.

Az ismétlődő elemek felismerése általában triviális. Néha azonban nem olyan egyszerű ezt észrevennünk. Ilyenkor az ismétlődő tulajdonság definíciójára támaszkodhatunk, mely szerint *az egyedítípus ismétlődő elemet tartalmaz, ha létezik olyan tulajdonsága, amelyik funkcionálisan független az elsődleges kulcstól.*

A fentiekre példa lehet egy olyan személyi nyilvántartás, melynek egyik oszlopa a legmagasabb iskolai végzettség. Nincs probléma, ha a Zeneakadémiai és a Bölcsészkarai diplomát is egybemosva egyformán egyetemnek tekintjük és az oszlopba csak *egyetemet* írunk be. De ha a diploma típusával is jellemezni akarjuk a legmagasabb végzettséget, melyik a magasabb, a műegyetemi villamosmérnök, az okleveles fizikus, vagy az okleveles közgazda? Mit írunk be ebbe a rovatba, ha valakinek több egyetemi diplomája van? Ez, ha jól akarjuk megoldani, egy burkolt ismétlődő mező, amit ha nem oldunk fel, jövőtehetetlen hibákhoz vezethet. (Nem találjuk meg a legjobb sebész orvost vagy adószakértőt, mert az első diplomája alapján tanárként van nyilvántartva).

4.6.3. Második normál forma (2NF)

Nem kell hozzá túl nagy intelligencia, hogy lássuk, a 4.11. ábrán lévő 1NF reláció nagyon sok redundanciát tartalmaz. Azonnal szembeötlő, hogy fölösleges minden osztály és minden téma megnevezését az összes sorban külön feltüntetni. Elegendő lenne osztályonként, illetve témánként csupán egyszer megadni. De ezeken a nyilvánvaló redundanciákon kívül más, jobban rejtett hibalehetőségek is vannak ebben a relációban, melyek észrevétele már egyáltalában nem triviális.

Nézzük meg, mi történik, ha *Andrásfai Béla*, vagy *Vigécz Jenő* kilép, azaz adatait töröljük a relációból. Ebben az esetben automatikusan eltűnik az *O2, Oktatás* téma, illetve az

¹ Igen sok adatbázis-kezelő rendszer nem követeli meg formailag az elsődleges kulcsot. Ennek ellenére a biztonságos munkához ezekben is célszerű minden táblázatra elsődleges kulcsot definiálni.

E01, Eladás osztály is. Ez egyben azt is jelenti, hogy ha később újra előjön ez a témakód vagy osztálykód, akkor nem tudjuk, hogy ezekhez az *Oktatás*, és az *Eladás* elnevezések tartoznak, illetve ezeknek az elnevezéseknek mi a kódja. 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 például a *Titkárságra* új főnök kerül. Mindkét esetben több sort kell egyidejűleg megváltoztatnunk, hogy adataink konzisztensek maradjanak. Azt a problémát, hogy egy adat módosításakor több helyen kell módosítást eszközölnünk, *módosítási anomáliának* nevezzük. Ez arra vezet, hogy ha a módosítást nem hajtjuk végre konzisztensen, akkor az adatbázisban ellentmondás jöhet létre. Ezt láthatjuk a 4.11. ábrán is, ahol a *Titkárság* főnökét feldolgozási hiba miatt vagy *Mohácsi Győzőnél*, vagy a *Titkárság* többi dolgozójánál nem módosítottuk.

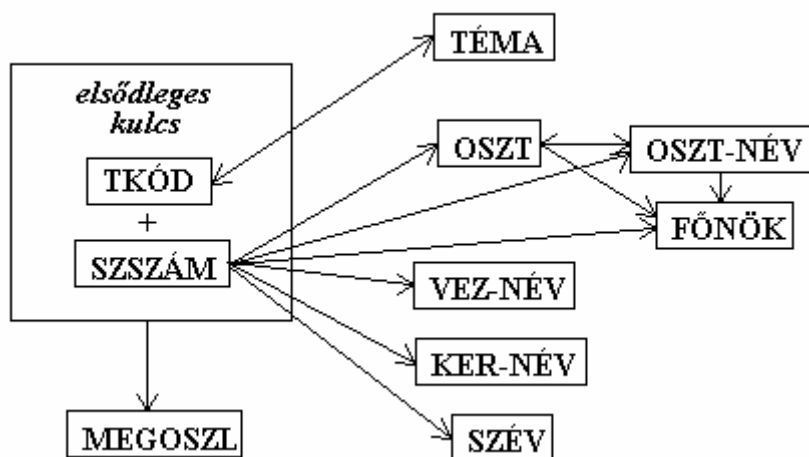
Gondjaink lehetnek új sorok beírásánál is, amit *beírási anomáliának* nevezzük.

Ha egy új osztályt hozunk létre a vállalatnál, amelyiknek még nincs egy dolgozója sem, vagy egy új dolgozót veszünk fel, de még nem tudunk egy témát sem kijelölni a számára, akkor ezt az információt nem tudjuk bevinni az adatbázisba. Mindkettő olyan sort eredményezne az ALKALMAZOTT-INF relációban, melyben az elsődleges kulcs, vagy annak egy része a NULL érték lenne.

A fent ismertetett anomáliákat összefoglaló néven *változtatási anomáliáknak* nevezzük, és ha előfordulnak, akkor sok gondot okozhatnak.

Ahhoz, hogy mindezen problémák gyökerét tisztán lássuk, rajzoljuk fel a 4.11. ábrán látható ALKALMAZOTT-INF reláció függőségi diagrammját (4.12. ábra).

A függőségi diagrammból láthatjuk, hogy a személyi azonosítószám (SZSZÁM) és témakód (TKÓD) összetett elsődleges kulcstól funkcionálisan teljesen csak a százalékos megoszlás (MEGOSZL) függ. Az összes többi attribútum az elsődleges kulcsnak csak egy összetevőjétől (TKÓD illetve SZSZÁM) függ.¹



4.12. ábra: A 4.11. ábrán látható ALKALMAZOTT-INF reláció függőségi diagrammja.

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 elsődleges kulcsú relációkat azonban a változtatási anomáliák megszüntetése érdekében 2NF-

¹ Az ábrán még feltüntettük azt is, hogy az OSZTÁLY és a FŐNÖK nem csak SZSZÁM-tól, hanem az OSZT attribútumtól is függ, ami nem része az elsődleges kulcsnak. Ennek a második normál formába alakításnál nincs jelentősége, a harmadik normál formába hozásnál fogjuk majd figyelembe venni.

re kell hoznunk. (Ez nem biztos, hogy megszünteti az összes anomáliát, de jelentősen csökkentheti a számukat). Ezt a műveletet nevezzük a relációk *szétbontásának*, vagy *dekompozíciónak*.

A dekompozíció reverzibilis művelet. A szétbontott relációkat mindig vissza lehet állítani az eredeti állapotukba. Az 1NF – 2NF – 3NF formák közti dekompozíció mindig veszteségmentes, az átalakítással nem veszünk ismereteket. Az ezeknél magasabb normál formákba (ld. 4.6.7. pont) való átalakításnál előfordulhat, hogy a dekompozíció következtében információt veszíthetünk, melynek megtartásáról más módon, például program segítségével kell gondoskodnunk.

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 1NF R relációban A, B kombinációja az elsődleges kulcs, C funkcionálisan teljesen függ tőle, de D funkcionálisan függ A -tól, azaz

$$R(A, B, C, D) \text{ és } R.A \rightarrow R.D,$$

akkor a második normál formába történő átalakítás az eredeti R relációnak a következő két új relációra való szétbontásával helyettesíthető:

$$R_1(A, D) \text{ és } R_2(A, B, C)$$

ahol $R_2.A$ idegen kulcsként hivatkozik $R_1.A$ -ra. Az idegen kulcs az adatbázis integritása és az átalakítás megfordíthatósága és veszteségmentessége érdekében szükséges. Ez biztosítja, hogy az R_1 és R_2 relációknak A alapján történő összekapcsolása, az

$$R_1 \text{ JOIN } R_2 (A)$$

művelet mindig előállítja az R reláció minden egyes sorát, csak azokat állítja elő, és mindegyik sort pontosan egyszer.¹

Amennyiben R összetett kulcsának több részétől (melyek még maguk is lehetnek összetettek) is függenek oszlopok, akkor ezek mindegyikére elvégezzük ezt a felbontást, mint ezt a példánkban is megtesszük a TKÓD és az SZSZÁM attribútumokra.

Az ALKALMAZOTT-1NF nincs második normál formában. Az (SZSZÁM, TKÓD) összetett elsődleges kulcstól csak MEGOSZL függ funkcionálisan teljesen. Ezért a dekompozíció után ez lesz az egyik reláció. Emellett két új relációt kapunk. Ezeknek a kulcsai SZSZÁM illetve TKÓD, oszlopai pedig az eredeti relációnak azok az oszlopai, melyek ettől funkcionálisan (teljesen) függenek.

A 2NF-re való szétbontás után az egyes relációk neve, elsődleges és idegen kulcsai, oszlopai:

ALKALMAZOTT-2NF
(OSZT, OSZT-NÉV, **SZSZÁM**, VEZ-NÉV, KER-NÉV, SZÉV, FŐNÖK)

MIT-CSINÁL?
(**SZSZÁM**, **TKÓD**, MEGOSZL)

¹ Ez abból következik, hogy az idegen kulcs miatt minden $R_2(A, B, C)$ sorhoz kell találnunk azonos A értékkel rendelkező $R_1(A, D)$ sort, és mivel ebben A elsődleges kulcs, ezért csak egyet találhatunk.

ahol SZSZÁM és TKÓD idegen kulcsként hivatkozik ALKALMAZOTT-2NF-re, illetve VÁLLALATI-TÉMA-ra.

VÁLLALATI-TÉMA
(TKÓD, TÉMA)

Az így létrejött táblázatokat a 4.13. ábrán láthatjuk.

ALKALMAZOTT-2NF

OSZT	OSZT-NÉV	SZSZÁM	VEZ-NÉV	KER-NÉV	SZÉV	FŐNÖK
P01	Pénzügy	11111	Andrásfalvi	Béla	1934	91526
P01	Pénzügy	11211	Bártfai	Pál	1937	91526
P01	Pénzügy	12110	Tóth	Csilla	1969	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

MIT-CSINÁL

SZSZÁM	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

idegen kulcs *idegen kulcs*

VÁLLALATI-TÉMA

TKÓD	TÉMA
B1	Bér
O2	Oktatás
U2	Utazás
E1	Ellenőrzés
T2	Tervezés
K1	Kereskedelem

elsődleges kulcs

alternatív kulcs

4.13. ábra: A 4.11. ábrán látható ALKALMAZOTT-1NF reláció szétbontása 2NF relációkra.

Ezzel a szétbontással a változtatási anomáliák jó részét automatikusan kiküszöböltük. Így például *Andrásfalvi Béla* kilépésével nem szűnik meg sem az *O2* témakód, sem az *Oktatás* téma, sem az *O2* → *Oktatás* kapcsolat (megszűnt törlési anomália). *Tóth Csilla* nevét csak

egy helyen, ALKALMAZOTT-2NF-ben kell megváltoztatni (megszűnt módosítási anomália). Új dolgozót is felvehetünk ebbe a relációba anélkül, hogy témát adnánk neki (megszűnt beírási anomália). A többi probléma azonban még továbbra is fennmarad. Azokat, mint azt a következő pontban látni fogjuk, a harmadik normál formába való átalakítás küszöböli csak ki.

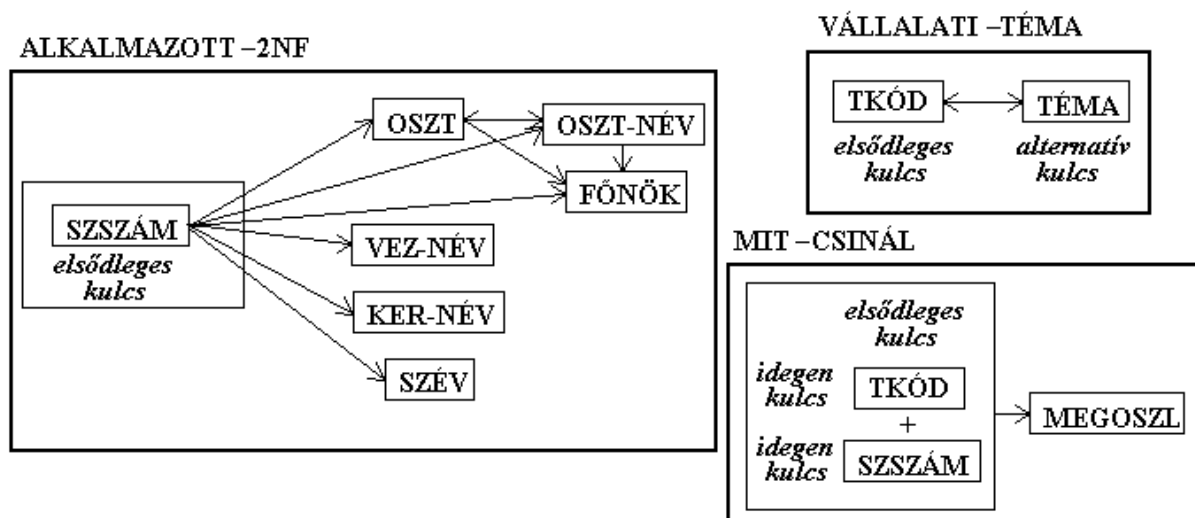
Ez a szétbontás *teljes*, azaz az összes olyan információt tartalmazza, ami az eredeti relációkban benne volt. De ezen kívül több is van benne. Az, hogy létezik *O2* témakód és a megnevezése *Oktatás*, míg az eredeti reláció ezeket csak addig tartalmazza, amíg van legalább egy alkalmazott az adatbázisban, aki ezen a témán dolgozik.

4.6.4. Harmadik normál forma (3NF)

A második normál formába hozott relációkkal még mindig lehetnek gondjaink. Az ALKALMAZOTT-2NF-ben a nyilvánvalóan redundáns osztály nevek és főnök azonosítók mellett még mindig megmaradtak az alábbi változtatási anomáliák:

- *Vigécz Jenő* kilépésével (törlésével) eltűnnek az *Eladás* osztályra vonatkozó információink. (Törlési anomália)
- Ha egy osztálynak új főnöke lesz, akkor azt továbbra is több helyen kell végigvezetni.¹ (Módosítási anomália)
- Továbbra sem tudunk új osztályt felvenni az adatbázisba, amíg nincsen legalább egy dolgozója. (Beírási anomália)

Ha felrajzoljuk a második normál formában lévő adatmodellünk függőségi diagrammját (4.14. ábra), akkor láthatjuk, hogy az összes olyan funkcionális függőség, amelyik nem elsődleges kulcsból indul ki, az osztály kódából (illetve a vele 1:1 függésben álló osztálynévből) indul ki, és az összes anomália is ehhez az attribútumhoz kapcsolódik.



4.14. ábra: A 4.13. ábrán látható ALKALMAZOTTAK-2NF reláció függőségi diagrammja.

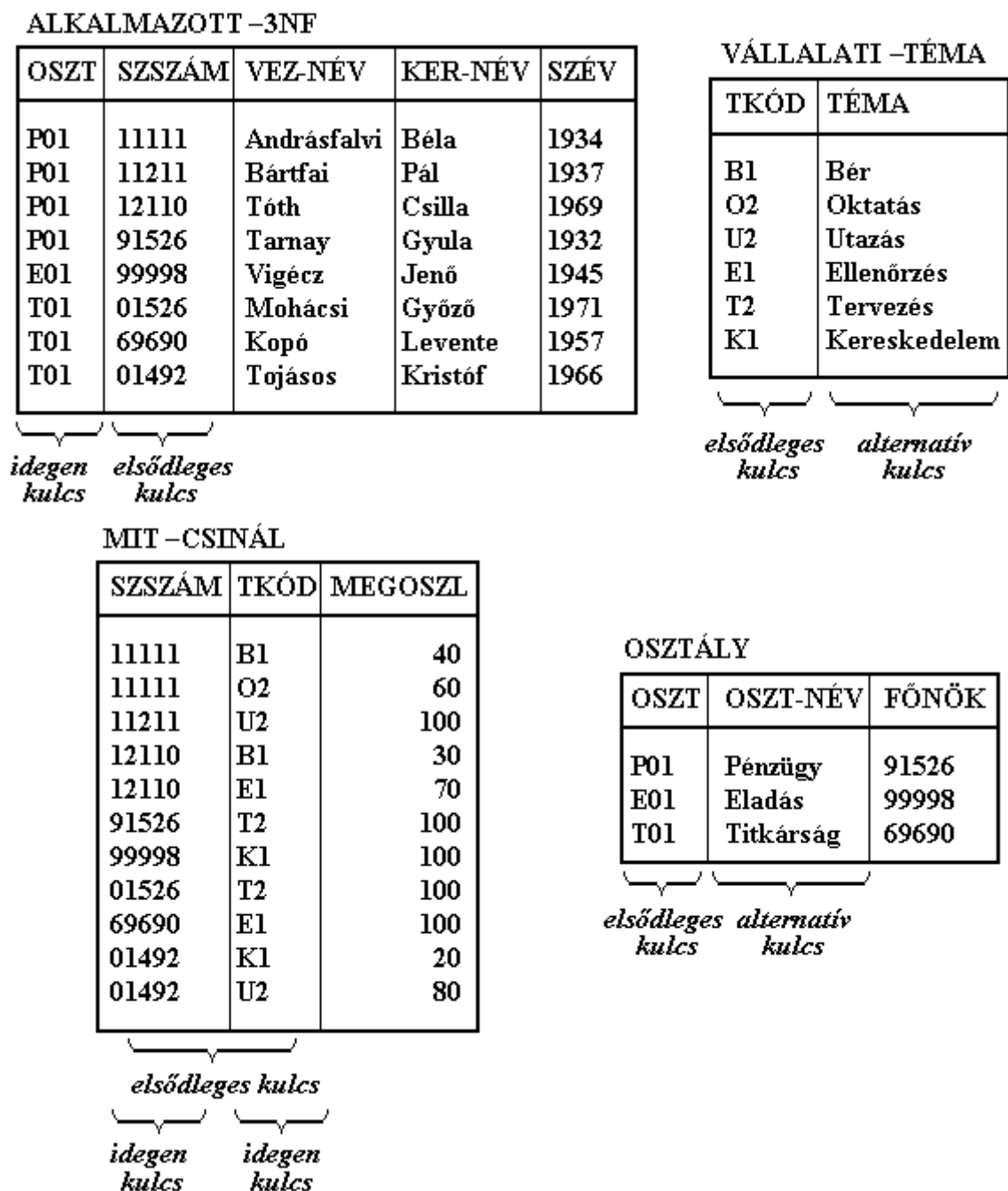
Ha új relációk létrehozásával ezeket a függőségeket is megszüntetjük, akkor a modellünk minden relációja harmadik normál formában (3NF) lesz. Ekkor, mint látni fogjuk, a még megmaradt változtatási anomáliák is megszűnnek.

¹ A már észlelt ellentmondás a *Titkárság* főnökénél, mint látjuk, még mindig megmaradhatott ALKALMAZOTT-2NF-ben.

Egy reláció akkor és csak akkor van harmadik normál formában (3NF), ha második normál formában van, és az elsődleges kulcshoz nem tartozó attribútumai csak az elsődleges kulcstól függenek funkcionálisan.

Másképpen megfogalmazva, a harmadik normál forma azt jelenti, hogy a nem kulcs elemek funkcionális függése csak az elsődleges (és az alternatív) kulcsokból indulhat ki.

Az ALKALMAZOTT-2NF reláció nincs 3NF-ben, mert például az osztály kód (OSZT) nem elsődleges (vagy alternatív) kulcs, és más oszlopok (OSZT-NÉV, FŐNÖK) funkcionálisan függenek tőle. Az ábra másik két relációja már 3NF-ben van.



4.15. ábra: A 4.11. ábrán látható ALKALMAZOTT-1NF reláció szétbontása 3NF relációkra (felhasználva a 4.13. ábrán látható 2NF felbontást).

A szétbontás úgy történik, hogy először vesszük a 2NF relációnak azt a projekcióját, amelyik csak azokat az attribútumokat tartalmazza, melyek kizárólag az elsődleges kulcstól függenek. Ennek elsődleges kulcsa ugyanaz marad. A másik új reláció (vagy relációk, ha több attribútumból is indul ki függés) elsődleges kulcsa a szétbontott reláció független attribútuma, oszlopai pedig a tőle függő attribútumok.

A feladat 3NF-re átalakított relációit a 4.15. ábrán láthatjuk.

Ennél a felbontásnál a 2NF-nél még megmaradó változtatási anomáliák, amiket a pont elején említettünk, automatikusan eltűnnek.¹

Általánosan megfogalmazva, ha az A elsődleges kulcsú és a B, C, D további attribútumokkal (bármelyik lehet összetett is) rendelkező 2NF R relációban C funkcionálisan függ B -től, azaz

$$R(A, B, C, D) \text{ és } R.B \rightarrow R.C$$

akkor a 3NF-be való szétbontás a következő relációk létrehozását jelenti:

$$R_1(B, C) \text{ és } R_2(A, B, D)$$

ahol $R_2.B$ idegen kulcsként hivatkozik $R_1.B$ -re.

Az R relációt egyértelműen vissza tudjuk állítani R_1 és R_2 relációk B alapján történő összekapcsolásával (join). Lényeges azonban, hogy a szétbontás a fenti algoritmus alapján történjen.

Elvileg ugyanis felbonthatnánk az ALKALMAZOTT-2NF relációt például a következő módon is

ALK-ROSSZ-3NF (SZSZÁM, FŐNÖK, VEZ-NÉV, KER-NÉV, SZÉV)
OSZTÁLY (OSZT, OSZT-NÉV, FŐNÖK).

A két reláció FŐNÖK szerint történő összekapcsolásából nem csak az eredeti ALKALMAZOTT-2NF reláció összes sorát kapnánk meg, hanem további, hamis sorokat is, amik akkor keletkeznének, ha ugyanaz a személy több osztályt is vezet, ami elő is fordulhat. A relációk alapján semmiféle lehetőségünk nincs ezek kiszűrésére, így ebből az adatbázisból azokról az alkalmazottakról, akiknek a főnöke több osztályt vezet, nem tudnánk egyértelműen megmondani, melyik osztályon dolgoznak. Természetesen addig, amíg az adataink között nincsen több osztályt is vezető személy, az adatmodellnek ezt a hibáját nem vennénk észre.

A relációs adatmodell realizálásához meg kell még adnunk az egyes relációk oszlopainak jellemzőit, az adattípust (beleértve az adatok hosszát is), esetleges alapértelmezésüket, és a kulcsokon felüli további korlátozásokat is. Ezeket a vállalatnál történő nyilvántartás feltételeinek további elemzése alapján állapíthatjuk meg.

Bár a 3NF relációk logikailag tiszta képet nyújtanak, a gyakorlatban sokszor nem érdemes az adatmodellt, az adatbázis minden relációját harmadik normál formába hozni. (Emlékeztetünk arra, hogy a relációs adatbázis-kezelő rendszereknek elég az 1NF is!) Ebben az esetben ugyanis a felhasználó számára ténylegesen szükséges adatokat sokszor több relációból, több fizikai rekordból szedi össze a rendszer. Ez általában jóval több lemezkezelő műveletet, hosszabb feldolgozási időt jelent. Éppen ezért bizonyos esetekben az adatbázis-felügyelő tudatosan nem végzi el a normalizálást a 3NF szintig, megelégszik a 2NF sőt az 1NF alakkal is. Különösen célszerű ez, ha az egymástól függő adatokra rendszeresen együtt van szükségünk, és az adatok értéke aránylag ritkán változik. Természetesen ilyenkor az

¹ Természetesen azt, hogy a Titkárság főnöke 01492, vagy 69690, azt más információkból kell eldöntenünk.

esetleg felléphető változtatási anomáliák kiküszöböléséről az adatbázis-felügyelőnek, vagy a felhasználónak saját magának kell gondoskodnia.

Tipikus gyakorlati példa a 3NF elhagyására egy személyi nyilvántartás, mely a pontos címben az irányítószámot is tartalmazza. Mivel az ország, város (helység), kerület, utca házszám együttesen egyértelműen meghatározza az irányítószámot (legalábbis egy jól szervezett városban), ez a reláció nem lesz harmadik normál formában. Ennek ellenére minden ésszerű alkalmazásban a cím az irányítószámot is tartalmazza. Elsősorban azért, mert ezek mindig együtt kellene legyenek. Másodsorban azért, mert az irányítószám szinte soha sem változik, tehát csak egyszer kell jól összepárosítani a többi cím adattal. És végül azért, mert ha az értékét mindig ki akarnánk számolni a többi címadatból, azt csak meglehetősen bonyolult és hosszadalmas algoritmussal tudnánk megtenni, és ezt a műveletet minden cím meghatározásakor végre kellene hajtani.

Az adattárházakban (ld. 9. fejezet), melyeknek egyszer beírt sorai nem módosíthatók rendszeresen dolgozunk 1NF és 2NF relációkkal is.

4.6.5. A dekomponálás más lehetőségei

Az első normál formába alakításhoz (ld. 4.6.2. pont) az ismétlődő mezőket más eljárással is kiküszöbölhetjük. Az ismétlődés általában egy N:M, speciális esetben egy 1:N kapcsolatot fed el. Amikor az osztály rekordban feltüntettük az alkalmazottakat, illetve az alkalmazott rekordjában feltüntetjük azt is, hogy milyen témákon dolgozik, akkor valójában az osztályok és alkalmazottak között fennálló egy-több kapcsolatot (egy osztálynak több dolgozója lehet, de egy dolgozó csak egy osztályhoz tartozhat), illetve az alkalmazottak és témák közt fennálló több-több kapcsolatot (egy alkalmazott több témán dolgozhat, és egy témán több alkalmazott dolgozhat) vittünk be egy relációba. Az egy-több kapcsolat megvalósítását idegen kulccsal, a több-több kapcsolat szétbontását kapcsoló rekordok közbeiktatásával a 3. fejezetben ismertettük. Ugyanezt alkalmazhatjuk itt is. Az ismétlődő elemet egy új relációba visszük ki, és idegen kulccsal (1:N kapcsolatot jelentő ismétlődés) vagy kapcsoló rekorddal kapcsoljuk az eredeti rekord ismétlődést nem tartalmazó részéhez (N:M kapcsolatot jelentő ismétlődés). Így a 4.10. ábrán látható, ismétlődő elemeket tartalmazó ömlesztett osztály-sorok oszlopaiból kiemeljük az ismétlődő alkalmazott adatokat (SZSZÁM, VEZ-NÉV, KER-NÉV, SZÉV) és az idegen kulcs kapcsolatot biztosító OSZT attribútumot egy önálló ALKALMAZOTT relációba, mely hivatkozik a nem ismétlődő osztály adatokat tartalmazó OSZTÁLY (OSZT, OSZT-NÉV, FŐNÖK) relációra. Ezután az ALKALMAZOTT-on belül ismétlődő TKOD,TÉMA attribútumokat is kivisszük egy önálló VÁLLALATI-TÉMA relációba amit egy kapcsoló rekorddal (célszerű elnevezése MIT-CSINÁL) kötünk hozzá az alkalmazott nem ismétlődő adatait (SZSZÁM, VEZ-NÉV, KER-NÉV, SZÉV) tartalmazó ALKALMAZOTT relációhoz. Ezzel ugyanahhoz a 3NF adatmodellhez jutottunk, mint az egymás után következő magasabb normál formákba lépésenként történő átalakítással.

Ugyanezt az adatmodellt megkaphatjuk egy jól átgondolt Entity – Relationship Modellből is. A feladat leírás alapján nyilvánvaló, hogy a kiinduló entitásaink:

ALKALMAZOTT (SZSZÁM, VEZ-NÉV, KER-NÉV, SZÉV)
OSZTÁLY (OSZT, OSZTÁLY, FŐNÖK)
VÁLLALATI-TÉMA (TKÓD, TÉMA)

A köztük fennálló 1:N ALKAMAZOTT – OSZTÁLY és az N:M ALKALMAZOTT – VÁLLALATI-TÉMA kapcsolatok alapján beiktatott MIT-CSINÁL kapcsoló relációval ugyanúgy a 4.15. ábrán látható, a normalizálás végeredményeként kapott adatmodellhez jutunk el.

4.6.6. Nézetek definiálása normalizált relációkból

Mindezideig – a hatékonyság érdekében tett esetleges kivételektől eltekintve – a normalizálásnak csak előnyeit láthattuk. Az átlagos felhasználó számára azonban ezek az előnyök semmivé zsugorodnak, ha például a 4.15. ábrán látható normalizált relációkból az egyik leggyakoribb kérdést „Egy adott nevű alkalmazott milyen nevű osztályon, milyen elnevezésű témákon, és azokon milyen megoszlásban, munkaideje hány százalékában dolgozik” mindig csak a négy táblázat összekapcsolásának explicit megadásával tudná csak megfogalmazni.

Éppen ezért a normalizálást a felhasználók munkájának megkönnyítésére az adatbázisokban csak a fizikailag is tárolt adatokra, az SQL táblázatokra (ld. 5.3.6. pont) végzik el. Ezekből aztán a relációs műveletek segítségével az összes olyan virtuális relációt, SQL nézetet (ld. 5.3.6. pont) elő lehet állítani, melyekre a felhasználónak rendszeresen szüksége lehet. Ezek a nézetek már gyakran csak 1NF relációk. A felhasználók azonban ezekre a nézetekre hivatkoznak, melyekben az adatok az általuk kívánt módon, látszólag redundánsan, de egyszerűen kezelhetően, együtt jelennek meg.

Így például az ezen pont elején megfogalmazott kérdésre definiálhatunk egy KI-HOL-MIN-DOLGOZIK nézetet, mely adatmodellünk mind a négy relációjának joinjából projekcióval jött létre és minden dolgozóra az alábbi adatokat tartalmazza:

- Személyi azonosító szám (SZSZÁM),
- vezetéknév (VEZ-NÉV),
- keresztnév (KER-NÉV),
- osztályának megnevezése (OSZT-NÉV),
- azoknak a témáknak a megnevezése, melyeken dolgozik (TÉMA),
- munkaidejének hány százalékában dolgozik az egyes témákon (MEGOSZL).

Az új, virtuális reláció a már meglévő táblázatokból az alábbi relációs műveletekkel jön létre:

```

KI-HOL-MIN-DOLGOZIK =
PROJECT {[[ALKALMAZOTT-3NF JOIN OSZTÁLY (OSZT)]
JOIN MIT-CSINÁL (SZSZÁM)] JOIN VÁLLALATI-TÉMA (TKÓD)}
(SZSZÁM, VEZ-NÉV, KER-NÉV, OSZT-NÉV, TÉMA, MEGOSZL)
```

A nézetben minden dolgozóról annyi sor lesz, ahány témán dolgozik.¹

A KI-HOL-MIN-DOLGOZIK nézet oszlopainak ismeretében, a fent említett kérdés az alábbira egyszerűsödik: Kérem a KI-HOL-MIN-DOLGOZIK adott nevű alkalmazottal jellemzett sorait.

```

PROJECT {RESTRICT KI-MIN-HOL-DOLGOZIK? WHERE
VEZ-NÉV = keresett dolgozó vezetéke
AND KER-NÉV = keresett dolgozó keresztnéve}
(VEZ-NÉV, KER-NÉV, SZSZÁM, OSZT-NÉV, TÉMA, MEGOSZL)
```

A kérdezőnek semmi szüksége nincs arra, hogy tudja, miként jött létre ez a nézet az eredeti táblázatokból, és hogy a kérdés megválaszolásához milyen utakon fog hozzáférni a rendszer.

A feldolgozás egyszerűbbé tétele mellett gyakran hoznak létre nézeteket adatvédelmi szempontokból is. Ha egy felhasználó nem dolgozhat egy reláció minden sorával/oszlopával,

¹ Ha lehet olyan dolgozó, aki egy témán sem dolgozik, vagy nem tartozik osztályhoz, akkor külső joint kellene használnunk.

akkor egy olyan nézetet hozunk létre korlátozással/projekcióval, amelyik ennek csak az általa feldolgozható részét tartalmazza. A felhasználó a jogosultságot nem a relációra kapja, hanem csak erre a nézetre.

Hangsúlyozni kívánjuk, hogy a virtuális relációk, a nézetek létrehozása csak a feladat megfogalmazását könnyíti, gyorsítja meg, az adatokhoz való fizikai hozzáférés nem válik gyorsabbá. Sőt, ha rosszul használjuk őket, például több táblázatra épülő nézetből csak egy táblázat adataira van szükségünk, akkor a nézetre való hivatkozás még lassíthatja is a feldolgozást. Az adatbázis-kezelő rendszer ahelyett, hogy egy táblázatból közvetlenül kiválasztaná a kért adatokat, előbb összeállítja több táblázatból a nézetet, majd ebből veszi ki az ugyanúgy csak egy táblázatra épülő elérési út alapján az adatokat.

4.6.7. Magasabb normál formák

Mint azt már többször megállapítottuk, a gyakorlatban a legtöbb esetben célszerű és elegendő is az adatmodellt harmadik normál formában lévő relációkra alapozni. Az élet azonban sokszínű, és előfordulhatnak olyan feltételek, melyeket jól szétbontott 3NF relációkkal sem tudunk automatikusan biztosítani. Ezek közül most két feladatot mutatunk be, melyek elvezetnek a Boyce-Codd (BCNF), illetve a negyedik (4NF) normál formához.

4.6.7.1. Boyce-Codd normál forma

Egy tanácsadó vállalatnál az ügyfeleknek különböző témákból nyújtanak konzultánsok, tanácsadók segítséget. Minden konzultáns csak egy témakörrel foglalkozik, de ugyanabból a témából más és más ügyfeleknek különböző konzulensek adhatnak tanácsokat. Egy ügyfélnek több témából lehet tanácsadója, de egy adott témában mindig ugyanaz a konzultánsa van. Nyilvántartjuk még minden ügyfélnél a konzultánsoknak az egyes témakörökben történt tanácsadásra fordított munkaidejét is. Az összes információt tartalmazó TANÁCSADÁS reláció a mintaadatokkal a 4.16. ábrán látható.

TANÁCSADÁS

ÜGYFÉL	TÉMA	TANÁCSADÓ	MUNKAIIDŐ
Richter	Gyógyszerkutatás	Ehrlich	100
EGIS	Gyógyszerkutatás	Ehrlich	99
Bayer	Gyógyszerkutatás	Flemming	77
Richter	Piac kutatás	Marco Polo	55
Bayer	Piac kutatás	Marco Polo	2000
Richter	Információs rendszerek	Quittner	11
EGIS	Információs rendszerek	Baksa-Haskó	111
APEH	Adó csalás	Al Capone	10000

4.16. ábra: TANÁCSADÁS reláció

Ennek a relációnak alternatív kulcsai vannak. Elsődleges kulcs lehet (ÜGYFÉL, TANÁCSADÓ) vagy (ÜGYFÉL, TÉMA). Bármelyiket is választjuk is, problémák lépnek fel.

Ha (ÜGYFÉL, TANÁCSADÓ) lesz az elsődleges kulcs, akkor, a reláció nincs még második formában sem. A kulcs egy része, TANÁCSADÓ, meghatározza a nem kulcs komponens TÉMÁ-t. Ezért a korábban megismert változtatási anomáliák ebben a relációban is fellépnek. Néhány példa:

- Ha a gyógyszerkutatásban Ehrlich helyett Flemming adná a tanácsokat, akkor több helyen kellene változtatnunk.

- Nem alkalmazhatunk új tanácsadót, amíg nincs legalább egy ügyfele.
- Ha az APEH nem lesz tovább az ügyfelünk, nem tudjuk, hogy Al Capone szakértő adócsalási ügyekben.

Ha a relációt a normalizálási eljárás szabályai szerint átalakítjuk az alábbi 3NF relációkká:

**TANÁCSADÁS-1 (ÜGYFÉL, TANÁCSADÓ, MUNKAIIDŐ) és
KONZULTÁNS (TANÁCSADÓ, TÉMA)**

ahol TANÁCSADÁS-1.TANÁCSADÓ idegen kulcsként hivatkozik KONZULTÁNS.TANÁCSADÓ-ra, akkor a problémáinkat megoldottuk.

Ha viszont balszerencsénkre (vagy más meggondolásokból) az (ÜGYFÉL, TÉMA) kombinációt választjuk elsődleges kulcsnak, akkor a TANÁCSADÁS (ÜGYFÉL, TÉMA, TANÁCSADÓ, MUNKAIIDŐ) reláció 3NF-ben van, mégis problémák merülnek fel. Nem tudjuk például automatikusan biztosítani, hogy egy témához olyan tanácsadót rendeljünk hozzá, aki valóban azzal a témával foglalkozik, és azt sem, hogy minden tanácsadónak csak egy témája legyen. Az integritási feltételek nem biztosítják, hogy például a *Richternek* adócsalási ügyekben ne *Quittner*t, az *EGIS*nek pedig ne *Baksa-Haskó*t jelöljük ki tanácsadónak, holott mindketten az információs rendszerek szakértői. Ezek a problémák abból adódnak, hogy a nem kulcs TANÁCSADÓ meghatározza a kulcs egy részét, TÉMÁ-t. A relációnak van olyan meghatározó, determináns eleme (ld. 4.6.1. pont), mely nem elsődleges és nem alternatív kulcs. Ez a reláció, bár 3NF, de nincsen Boyce-Codd normál formában (BCNF).

Egy reláció akkor és csak akkor van Boyce-Codd normál formában (BCNF), ha harmadik normál formában van és minden meghatározó eleme elsődleges vagy alternatív kulcs.

Azok a 3NF relációk, melyeknek nincs alternatív kulcsuk automatikusan BCNF-ben is vannak. A TANÁCSADÁS példánknál maradva, a már BCNF-ben lévő

**TANÁCSADÁS-2 (ÜGYFÉL, TÉMA, MUNKAIIDŐ) és
KONZULTÁNS (TANÁCSADÓ, TÉMA)**

átalakítás nem vezet célra. Ezekből ugyanis nem tudjuk megkapni, hogy a téma különböző szakértői közül melyik foglalkozik az adott ügyféllel.

Megoldás lehet a következő, meglehetősen mesterkélt megoldás:

**TANÁCSADÁS-3 (ÜGYFÉL, TÉMA, TANÁCSADÓ, MUNKAIIDŐ) és
KONZULTÁNS (TANÁCSADÓ, TÉMA),**

ahol (TANÁCSADÁS-3.TANÁCSADÓ, TANÁCSADÁS-3.TÉMA) együttesen idegen kulcsként hivatkozik (KONZULTÁNS.TANÁCSADÓ, KONZULTÁNS.TÉMA)-ra.¹

Ehelyett azonban célszerűbb, ha az alternatív kulcsok közül az elsőként javasolt (ÜGYFÉL, TANÁCSADÓ) kombinációt választjuk elsődleges kulcsnak.

¹ Tovább bonyolíthatja a helyzetet, hogy a gyakorlatban sok rendszerben csak elsődleges kulcsra hivatkozhatunk idegen kulccsal. Ezért KONZULTÁNS-ban (TANÁCSADÓ, TÉMA)-t kell elsődleges kulcsnak definiálni, és annak biztosítására, hogy minden tanácsadónak csak egy témája lehessen, külön egyedi indexet kell létrehoznunk KONZULTÁNS.TANÁCSADÓ-ra.

4.6.7.2. Negyedik normál forma

A következő példában sem vezet a harmadik normál formába való átalakítás tökéletes eredményre.

Adva vannak

- Tantárgyak,
- Oktatók,
- Könyvek.

További feltételek

- egy tantárgyat több oktató is taníthat, és egy oktató több tárgyat is taníthat,
- egy tantárgyat több könyvből is taníthatnak és ugyanazt a könyvet több tárgynál is használják,
- bárki is oktatja a tárgyat, ugyanazokat a könyveket kell használnia,
- minden oktató csak meghatározott tantárgyakat tanít, és minden tárgyat csak meghatározott oktatók taníthatnak.

A nem normalizált adatokat a 4.17. ábrán láthatjuk.

TANTÁRGY	OKTATÓ	TANKÖNYV
	<i>ismétlődik TANTÁRGY-ban annyiszor, ahányan tanítják</i>	<i>ismétlődik OKTATÓ-ban annyiszor, ahányan tanítanak belőle</i>
OKTATÁS		
<i>TANTÁRGY</i>	<i>OKTATÓ</i>	<i>TANKÖNYV</i>
Informatika	Quittner Pál Baksa-Haskó Gabriella	Adatbázisok Adatbázis-kezelés
Fizika	Quittner Pál Kiss Dezső	Neutronfizika Út az atomfizikához Gyakorlati fizika
Matematika	Milhoffer Hugóné Kiss Dezső Bolyai János	Kalkulus

4.17. ábra: Tantárgyak, oktatók, tankönyvek közti összefüggés nem normalizált formában.

Az összetartozó (TANTÁRGY, OKTATÓ, TANKÖNYV) adatokat az ismétlődő mezők kiküszöbölésével első normál formába hozhatjuk. Az így létrejött TOK (TANTÁRGY, OKTATÓ, TANKÖNYV) reláció (ld. 4.18. ábra) csak elsődleges kulcsból áll, így egyben BCNF-ben is van. Ennek ellenére ez az elrendezés igen sok redundanciát tartalmaz. (Főleg példánkban háromszor megadni, hogy *Quittner Pál* tanítja a fizikát, kétszer, hogy informatikához mik a kötelező tankönyvek, stb.).

Ráadásul a szokásos módosítási anomáliák mellett még új problémák is jelentkeznek. Például egy fizikát tanító új oktató felvétele három új sor létrehozását jelenti, úgy, hogy ezekben TANKÖNYV meghatározott értékeket (Neutronfizika, Út az atomfizikához, Gyakorlati fizika) vehet csak fel, amit azonban nem tudunk integritási feltételekkel automatikusan biztosítani. Hasonlóan, 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 függés (ld. 4.6.1. pont) áll fenn.

TANTÁRGY $\rightarrow\rightarrow$ OKTATÓ és TANTÁRGY $\rightarrow\rightarrow$ TANKÖNYV

TOK

TANTÁRGY	OKTATÓ	TANKÖNYV
Informatika	Quittner Pál	Adatbázisok
Informatika	Baksa-Haskó Gabriella	Adatbázisok
Informatika	Quittner Pál	Adatbázis-kezelés
Informatika	Baksa-Haskó Gabriella	Adatbázis-kezelés
Fizika	Quittner Pál	Neutronfizika
Fizika	Kiss Dezső	Neutronfizika
Fizika	Quittner Pál	Út az atomfizikához
Fizika	Kiss Dezső	Út az atomfizikához
Fizika	Quittner Pál	Gyakorlati fizika
Fizika	Kiss Dezső	Gyakorlati fizika
Matematika	Millhoffer Hugóné	Kalkulus
Matematika	Kiss Dezső	Kalkulus
Matematika	Bolyai János	Kalkulus

4.18. ábra: A 4.17. ábrán látható adatok elrendezése normalizált relációban.
Az elsődleges kulcs (TANTÁRGY, OKTATÓ, TANKÖNYV).

A többértékű funkcionális függésből adódó problémák kiküszöbölésére vezették be a *negyedik normál formát (4NF)*.

Egy R reláció akkor és csak akkor van *negyedik normál formában*, ha $3NF$ -ben van és csak olyan többértékű funkcionális függőség van benne, melyre igaz, hogy ha $R.A \rightarrow\rightarrow R.B$, akkor a reláció összes többi $C, D, \dots$ oszlopa funkcionálisan függ A -tól.

A fent példában szereplő reláció nyilván nem $4NF$, mivel

TANTÁRGY $\rightarrow\rightarrow$ OKTATÓ igaz, de
TANTÁRGY $\rightarrow$ TANKÖNYV nem érvényes,
(csak TANTÁRGY $\rightarrow\rightarrow$ TANKÖNYV igaz).

A $4NF$ -re bontás a többszörös függőségben álló oszlopokat bontja külön relációkra. Ez példánkban a

TANT-OKT (TANTÁRGY, OKTATÓ) és a
TANT-TKÖNYV (TANTÁRGY, TANKÖNYV)

relációkat jelenti. Ezzel a redundanciát megszüntettük és az eredeti adatokat teljesen vissza tudjuk állítani a TANT-OKT JOIN TANT-TKÖNYV (TANTÁRGY) művelettel.

Egyetlenegy problémánk azért még maradt, bár ez lehet gyakorlati szempontból előnyös is. Az eredeti TOK relációba csak akkor vehettünk fel egy új tantárgyat, ha legalább egy oktató és legalább egy tankönyv volt már hozzá.¹ (Hasonló jellegű korlátozás érvényes az új oktatóra és tankönyvre is). A $4NF$ relációkból álló modellbe már akkor is felvehetünk új tantárgyat, ha legalább egy oktatót vagy egy tankönyvet rendeltünk hozzá.

¹ Ezt megkerülhettük az adatbázis-technikai szempontból nem igen ajánlott 'NINCS' mesterséges oktató és tankönyv értékek bevezetésével.

4.6.8. A normál formák összefoglalása

Az egymás után következő normál formák (1NF, 2NF, 3NF, BCNF, 4NF, 5NF) az adatbázist alkotó relációk „matematikai jóságát” jelzik. Az alacsonyabb fokú relációkból a matematikailag jobb magasabb formákba a relációk tulajdonságai közti különféle függőségek tervszerű megszüntetésével juthatunk el. A normál formák „egymásba skatulyázottak”. Ez azt jelenti, hogy a magasabb fokban levők egyben az alacsonyabb fok kritériumait is teljesítik. Egy 3NF reláció egyben 1NF és 2NF formában is van, egy BCNF pedig egyben 3NF, 2NF és 1NF-ben is. A jóság természetesen csak matematikailag értendő, a logikai érthetőség, az áttekinthetőség, az ellentmondás mentesség fokát jelenti.

Az adatbázis tervezésekor a relációkat általában 3NF-be hozzuk. De a gyakorlatban sokszor hatékonyabban tudunk alacsonyabb normál formában lévő relációkkal dolgozni. Különösen igaz ez az adattárházakra, ahol a lekérdezések felgyorsítása érdekében igen sok relációt első vagy második normál formában tárolunk.

A 3NF-nél magasabb normál formák inkább csak elméleti szempontból érdekesek. Egyedül a BCNF-re való átalakítás fordul néha elő a gyakorlatban.. Amikor erre szükség lehet, akkor is célszerűbb a formális normalizálási műveletek helyett a feladat logikus végiggondolásával a kulcsok közötti függőségekből adódó problémákat megoldani.

A 4.1.táblázatban összefoglaltuk az egyes normál formák reláción belüli jellegzetességeit.

4.1. táblázat: Az egyes normál formák jellegzetességei.

Normál forma típusa	Jellemzői
Első normál forma (1NF)	Nincsenek ismétlődő elemek (csoportok). Minden oszlop csak egy értéket (ez lehet a NULL érték is) vehet fel.
Második normál forma (2NF)	Csak teljes funkcionális függés lehet a <i>kulcs</i> és a <i>nem kulcs</i> ¹ tulajdonságok (attribútumok) között.
Harmadik normál forma (3NF)	Minden nem kulcs tulajdonság csak az elsődleges kulcstól függ és attól funkcionálisan teljesen.
Boyce-Codd normál forma (BCNF)	Minden funkcionális függés a kulcson belül is és a nem kulcs elemek között is ki van küszöbölve.
Negyedik normál forma (4NF)	Nincsenek több értékű függőségek sem.
Ötödik normál forma (5NF)	Semmiféle függőség nincs az egyes tulajdonságok között.

4.7. A relációs modellen alapuló adatbázisok összefoglalása

Ebben a pontban ismétlésként összefoglaljuk a relációs rendszer alapkövetelményeit és az adatbázisok megtervezésének és létrehozásának legfontosabb lépéseit.

¹ *Kulcs*on itt az elsődleges kulcsot, *nem kulcs elemen* pedig az elsődleges és az esetleges alternatív kulcson kívül az összes többi attribútumot értjük.

4.7.1. Alapkövetelmények

A relációs modell három pilléren nyugszik. Ezek a következők:

- a relációs adatstruktúra megvalósítása,
- az adatok integritásának biztosítása,
- adatkezelés relációs műveletek segítségével.

A relációs adatstruktúra azt jelenti, hogy minden egyedtypust n -ed fokú relációkban (táblázatokban) ábrázolunk, melyeknek egyedi előfordulásai (sorai, rekordjai) elemi tartományokra (domain) épülő tulajdonság (attribútum, oszlop) értékekből állnak. Egyértelmű azonosításuk a reláción belül az elsődleges kulcs segítségével történik.

Az adatbázison belül minden sor egyértelműen azonosítható a reláció nevével és elsődleges kulcsának értékével, minden eleme pedig a reláció és az oszlop nevével valamint a sor elsődleges kulcsának az értékével.

A modellben meghatározott adatstruktúra létrehozásához és módosításához rendelkezni kell egy adatleíró lehetőséggel, mellyel az adatokat tartalmazó alaprelációk, illetve ezekből relációs műveletekkel származtatott virtuális relációk létrehozhatók.

Az adatok integritása az elsődleges és az 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 mindig meg kell egyezzen a hivatkozott reláció hivatkozott elemének (általában elsődleges kulcsának) valamelyik aktuális értékével, vagy a NULL értékkel egyenlő.

Az adatkezelés a relációs műveletek segítségével lehetővé kell tegye az adatok kiválasztását olvasáshoz, módosításához, törléséhez, új relációk létrehozásához. Technikailag erre az SQL nyelvet használjuk.

Azokat a rendszereket, amelyek a fenti alapkövetelményeket a továbbiakban még bővebben értelmezett módon minden szempontból kielégítik, *teljes relációs rendszernek* nevezzük. Ezek még az alábbi követelmények mindegyikének is eleget tesznek:

1. Az adatbázisban minden adatot relációkban és csak relációkban tárolnak. Ezekből kapható meg minden információ a relációs műveletek segítségével.
2. Minden adat egyértelműen elérhető a megfelelő reláció, ezen belül pedig a kívánt oszlop nevének és a sor elsődleges kulcsának a megadásával. Ezekből a kívánt információ előállítható.
3. A NULL értéket egyértelműen és megfelelő módon kezeli a rendszer.
4. Bármikor hozzáférhető, naprakész, automatikusan karbantartott katalógussal (adatszótárral) rendelkezik.
5. Van olyan párbeszédes és programba beépíthető nyelve (SQL), amely lehetővé teszi
 - adatok definiálását,
 - relációs műveletek elvégzését,
 - integritási és adatbiztonsági feltételek definiálását,
 - összetett tranzakciók egységes kezelését (ld. a COMMIT és ROLLBACK az 5.6.4. pontban).
6. Táblázatok módosíthatók nézeteken keresztül is, amennyiben ez egyértelműen, ellentmondás-mentesen elvégezhető.
7. Rendelkezik adatok csoportos beírásával, módosításával, törlésével lehetővé tevő műveletekkel.
8. Biztosítja a fizikai és a logikai adatfüggetlenséget.

9. Az integritási feltételeket maga az adatbázis tartalmazza. Betartásuk független az alkalmazási programoktól. Ezek a feltételek módosíthatók, de az új feltételek az adatbázisban már bennlévő adatokra is érvényben kell legyenek.
10. A fenti szabályoknak érvényben kell maradni, függetlenül attól, hogy az adatbázist centralizáltan, vagy osztott formában építik fel.
11. Az integritási és biztonsági korlátozásoknak akkor is érvényben kell maradni, ha alacsonyabb szinten lehetőség van egyes rekordokhoz az adatbázis-kezelő rendszer megkerülésével való hozzáférésre is.

A mai relációs adatbázis-kezelő rendszerek ezen feltételeket általában kielégítik, de vannak olyanok, amelyek nem mindegyiket és/vagy nem maradéktalanul. Ennek ellenére a mindennapi felhasználó számára ezek is gyakorlatilag teljes (vagy majdnem teljes) relációs rendszereknek tekinthetők.

4.7.2. Főbb lépések

A relációs adatbázisok tervezésének és létrehozásának a legfontosabb lépései a következők:

1. A szükséges adatok kiválasztása.
2. Az összetartozó adatok, az entitások (egyedtypusok) meghatározása.
3. Az entitások tulajdonságainak és elsődleges kulcsának meghatározása.
4. Az entitások közötti kapcsolatok meghatározása.
5. A több-több kapcsolatok felbontása új relációk segítségével.
6. A relációk legalább első normál formában való definiálása.
7. Idegen kulcsok definiálása a relációk közti kapcsolatok biztosítására.
8. A szükséges normalizálás elvégzése. A relációk magasabb fokú normál formába történő átalakítása.
9. Az entitások további tulajdonságainak (adattípus, hossz, alapértelmezés, korlátozó feltételek) részletes meghatározása.
10. Táblázatok létrehozása a normalizált relációk alapján.
11. A várható feladatok optimális végrehajtását elősegítő fizikai adatszerkezet kialakítása, indexek létrehozása.
12. A várható feladatok adatait együttesen, egyben tartalmazó virtuális relációk (nézetek) meghatározása, definiálása.
13. Hozzáférési jogok meghatározása, az adatok védelmének biztosítása.
14. Adatok betöltése, műveletek az adatokkal.
15. Szükség esetén az adatbázis szerkezetének a módosítása, új táblázatok, nézetek, indexek létrehozása az 1-14. lépésekben leírtak szerint.

4.7.3. Egy adatbázis megtervezése

Az eddig elmondottak összefoglalásaként

- tervezzük meg a feladat leírásban szereplő csomagküldő szolgálat adatbázisának Entity-Relationship diagrammját,

- határozzuk meg az elsődleges és az idegen kulcsokat,
- adjuk meg az egyes entitások legfontosabb tulajdonságait,
- határozzuk meg, hogy az adatintegritás és a hatékony feldolgozás érdekében mire, milyen típusú indexeket célszerű létrehozunk,
- definiáljuk az adatbázis egyszerűbb használatát elősegítő nézeteket.

4.7.3.1. Feladatléírás

Egy csomagküldő szolgálattól katalógus alapján lehet árukat megrendelni. A katalógus tartalmazza az áruk leírását és az árakat is. Ugyanaz az áru különböző kivitelben (méret, szín) is megrendelhető. A szín és a méret az árat nem befolyásolja. A katalógusban szereplő árakat negyedévenként frissítik fel.

Az árukat különböző raktárakban tárolják. Egy adott áru csak egy raktárban található.

A megrendelők egy megrendelésben akárhány árut akárhány kivitelben megrendelhetnek.

A szállítás egy megrendelőhöz naponta egyszer történik. Egy szállításban akárhány árut akárhány kivitelben kiszállíthatunk. Egy szállítás tételei különböző megrendelésekből is származhatnak, de egy adott rendelési tételt mindig teljes mértékben teljesíteni kell. (Ha egy megrendelésben két kék és két fehér 44-es Benetton ing szerepelt, akkor lehet, hogy egy szállításban csak a két kék inget küldjük el és a két fehéret egy másikban szállítjuk, de az nem lehet, hogy egy szállítmány csak egy fehér és/vagy egy kék inget tartalmazzon. Ugyanakkor leszállíthatjuk ezekkel együtt egy másik rendelés összes ingét is).

A vevők különböző kedvezményekben részesülhetnek. Minden vevőnek van egy saját kialakított, rá jellemző kedvezménye. Ezen kívül minden vevő egyformán részesülhet a távolsági kedvezményben, melynek mértéke minden vevőre és árura azonos és csak attól függ, hogy melyik városban van a vevő. (Ha valakinek több városba is szállítunk, akkor minden városban önálló vevőként tartjuk nyilván). Emellett létezik minden árura egy mennyiségi kedvezmény is, melynek nagysága, és az, hogy hány darab felett adható, az áruként változó. (Azt, hogy a különböző kedvezmények összegzéseként végül ténylegesen mekkora kedvezmény jár az egyes megrendelések után, a feldolgozó program állapítja meg, de az ehhez szükséges adatokat az adatbázisból veszi.)

A vevő egy szállítás minden áruját átveszi. Az esetleges reklamációk kezelését a modellnek nem kell tartalmazni. Az időbeli változásokat az árak kivételével nem kell nyomon követni.

4.7.3.2. A tervezési feladat megoldása

Első lépésként meghatározzuk, hogy milyen kézenfekvő egyed típusokat definiálunk, ezeknek mik lesznek az azonosítóik (elsődleges kulcs, vastagítva jelölve), és milyen tulajdonságaik vannak (milyen adatokat fognak tartalmazni).

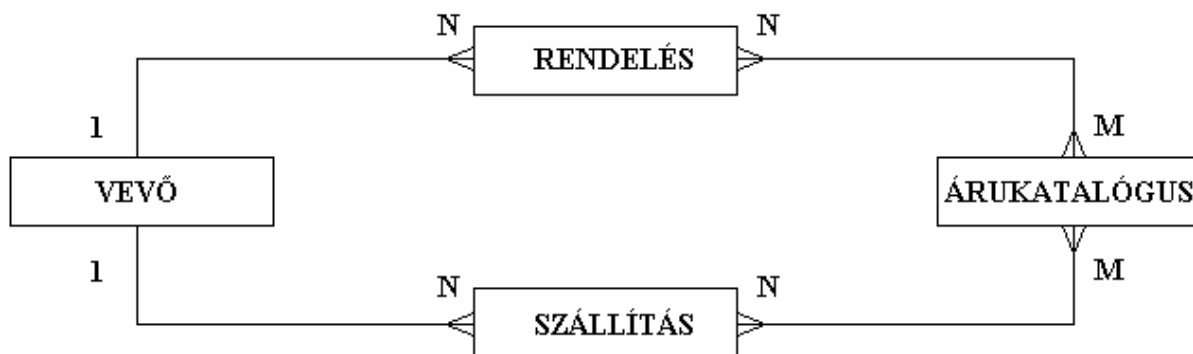
VEVŐ (Vevőkód, Vevőnév, Vevőcím, Vevő_bankszámla, Vevő_kedvezmény, Város, Távolsági_kedvezmény)

ÁRUKATALÓGUS (Negyedév, Árukód, Szín, Méret, Árunév, Áru-képe, Rendelési egység, Raktár – melyik raktárban tárolják + a raktár többi adata –, Kedvezmény_határ amely fölötti rendelés esetén kedvezmény jár, Mennyiségi_kedvezmény, Raktárkészlet, Egységár, Rendelési egység)

RENDELÉS (Rendelészám, Vevőkód, Rendelési_Dátum, vevő többi adata, Negyedév, Árukód, Szín, Méret, áru többi adata, Rendelt mennyiség, Leszállítva)

SZÁLLÍTÁS (**Szállításszám**¹, Vevőkód, Szállítási_Dátum, vevő többi adata, Negyedév, Árukód, Szín, Méret, áru többi adata, Szállított_mennyiség)

Második lépésként meghatározzuk az így definiált entitások közti kapcsolatokat. Ezt a 4.19. ábrán láthatjuk.



4.19. ábra: Csomagküldő szolgálat Entity – Relationship diagrammjának első változata.

A következő lépésben felbontjuk az ÁRUKATALÓGUS és a RENDELÉS illetve a SZÁLLÍTÁS közti több-több kapcsolatot a RENDELÉS-TÉTEL és a SZÁLLÍTÁS-TÉTEL bevezetésével. Ennek megfelelően módosítjuk a RENDELÉS és a SZÁLLÍTÁS relációkat és meghatározzuk az idegen kulcsokat (dőlt betűkkel jelölve).

RENDELÉS-TÉTEL (**Rendelésszám**, *Negyedév*, *Árukód*, *Szín*, *Méret*, áru egyéb jellemzői, Rendelt_mennyiség, Leszállítva)

SZÁLLÍTÁS-TÉTEL (**Szállításszám**, *Negyedév*, *Árukód*, *Szín*, *Méret*, áru egyéb jellemzői, Szállított_mennyiség)²

RENDELÉS (**Rendelésszám**, *Vevőkód*, vevő többi adata, Rendelési-Dátum)

SZÁLLÍTÁS (**Szállításszám**, *Vevőkód*, vevő többi adata, Szállítási-Dátum)

Az idegen kulcsok:

RENDELÉS-TÉTEL-ben: *Rendelésszám* hivatkozik RENDELÉS-re, *Negyedév*, *Árukód*, *Szín*, *Méret* ÁRUKATALÓGUS-ra

¹ Mivel feltételeink szerint egy vevőnek egy nap csak egyszer szállítunk, nem kellene "mesterségesen" bevezetni elsődleges kulcs céljából a *Szállításszám* attribútumot. Mégis megtesszük ezt. Egyrészt azért, mert a gyakorlatban könyvelési okokból úgyis szükség van rá. Másrészt azért, mert ha később mégis megengednénk a napi többszöri szállítást, akkor ezt a ((*Vevőkód*, *Szállítási_Dátum*) egyedi index (ld. ezen pont végén) duplikált indexre való módosításával igen egyszerűen meg tudjuk tenni, míg egy új elsődleges kulcs beillesztése SZÁLLÍTÁS-ba vagy a *Szállítási_Dátum*nak az egyértelművé tétele a szállítás pontos időpontjával (óra, perc) való kibővítésével elég bonyolult változtatásokat igényelne.

² Mivel egy szállításban lehetnek különböző katalógusok alapján megrendelt tételek, ezért a katalógusra jellemző *negyedév* értékét itt és nem SZÁLLÍTÁS-ban kell megadnunk. A megrendelés nyilván csak egy katalógus alapján történhet, így a *negyedévet* RENDELÉS-TÉTEL-ből kivihetnénk RENDELÉS-be. A rendelések és szállítások egységes kezelése érdekében mégis RENDELÉS-TÉTEL-ben adjuk meg.

SZÁLLÍTÁS-TÉTEL-ben: *Szállításszám* hivatkozik SZÁLLÍTÁS-ra, *Negyedév*, *Árukód*, *Szín*, *Méret* ÁRUKATALÓGUS-ra

RENDELÉS-ben és SZÁLLÍTÁS-ban: *Vevőkód* hivatkozik VEVŐ-re.

Ezek a relációk mind első normál formában vannak. Következő feladatunk ezeknek 3NF-be való átalakítása lesz.

VEVŐ nincsen 3NF-ben, mivel a *Távolsági-kedvezmény* csak a nem elsődleges kulcs *Várostól* függ. Ezért ezt felbontjuk két relációra:

VEVŐ (**Vevőkód**, Vevőnév, Vevőcím, Vevő_bankszámla, Vevő_kedvezmény, *Város*)

SZÁLLÍTÁSI-KEDVEZMÉNY (**Város**, Távolsági_kedvezmény)

VEVŐ-ben idegen kulcs *Város*, hivatkozik SZÁLLÍTÁSI-KEDVEZMÉNY-re.

RENDELÉS és SZÁLLÍTÁS sincs 3NF-ben. A „*vevő többi adata*” a *vevőkód*-tól függ, ami nem elsődleges kulcs. Így azokat egy új táblázatba kellene kivinnünk. Ez azonban már létezik, VEVŐ-ként definiáltuk. Ezért a vevő többi adatait, mint redundánsakat mindkét relációból egyszerűen elhagyhatjuk.

RENDELÉS-TÉTEL és SZÁLLÍTÁS-TÉTEL nincs 2NF-ben. Az áru egyéb jellemzői csak a *Negyedév*, *Árukód*, *Szín*, *Méret* oszlopoktól, az elsődleges kulcs egy részétől függnének. Ezeket kivisszük egy új táblázatba. Mivel az időbeli változásokat (*Negyedévtől* való függés) csak az árnál kell figyelembe vennünk, a *szín* és a *méret* pedig nem befolyásolja egy adott áru további jellemzőit ezt rögtön tovább bontjuk az alábbi módon:

ÁRUKATALÓGUS (**Negyedév**, **Árukód**, Egységár)

ÁRULEÍRÁS1 (**Árukód**, Árunév, Áru-képe, Rendelési egység, Raktárkód, raktár többi adata, Mennyiségi_kedvezmény, Kedvezmény_határ)

ÁRUVÁLASZTÉK (**Árukód**, **Szín**, **Méret**, Raktárkészlet)

Az ÁRULEÍRÁS1 relációt, mivel ebben a *raktár többi adata* csak a *Raktár-kódtól* függ, még tovább bontjuk:

ÁRULEÍRÁS (**Árukód**, Árunév, Áru-képe, Rendelési egység, *Raktárkód*, Mennyiségi_kedvezmény, Kedvezmény_határ)

RAKTÁR (**Raktárkód**, Raktár-cím, Raktár-vezető, Raktár_telefon)

Új idegen kulcsok:

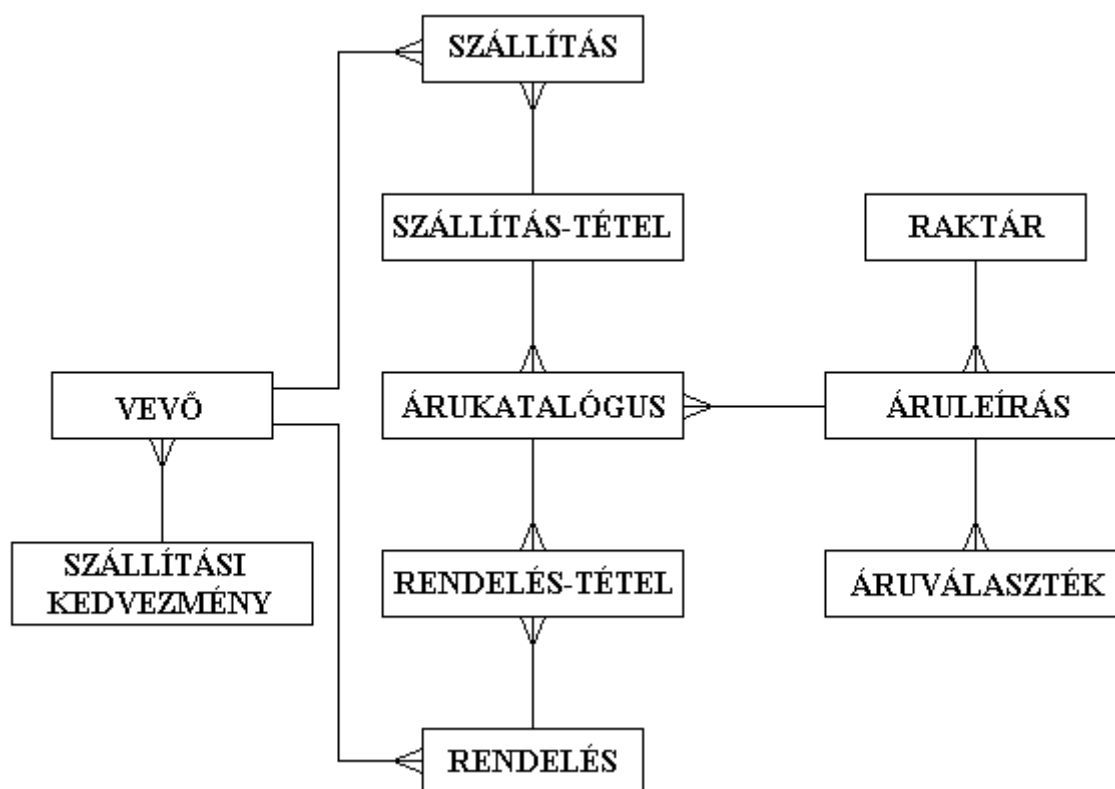
ÁRUKATALÓGUS-ban *Árukód* hivatkozik ÁRULEÍRÁS-ra.

ÁRULEÍRÁS-ban *Raktárkód* hivatkozik RAKTÁR-ra.

ÁRUVÁLASZTÉK-ban *Árukód* hivatkozik ÁRULEÍRÁS-ra.

A felbontások miatt RENDELÉS-TÉTEL és SZÁLLÍTÁS-TÉTEL-ben *Negyedév*, *Árukód* hivatkozik ÁRUKATALÓGUS-ra, *Árukód*, *Szín*, *Méret* hivatkozik ÁRUVÁLASZTÉK-ra.

Az így létrejött végleges adatmodellt a 4.20. ábrán láthatjuk.



4.20. ábra: Csomagküldő szolgálat harmadik normál formában lévő adatmodellje.

Az egyes relációkat, tulajdonságaikat, az elsődleges és az idegen kulcsokat az áttekinthetőség érdekében újra megadjuk.

VEVŐ

Vevőkód

Vevőnév

Vevőcím

Vevő_bankszámla

Vevő_kedvezmény

Város (SZÁLLÍTÁSI-KEDVEZMÉNY)

SZÁLLÍTÁSI-KEDVEZMÉNY

Város

Távolsági_Kedvezmény

SZÁLLÍTÁS

Szállításszám

Vevőkód (VEVŐ)

Szállítási_Dátum

SZÁLLÍTÁS-TÉTEL

Szállításszám (SZÁLLÍTÁS)

Negyedév

Árukód (ÁRUKATALÓGUS)

Szín

Méret (ÁRUVÁLASZTÉK)

Szállított_mennyiség

ÁRUKATALÓGUS

Negyedév

Árukód (ÁRULEÍRÁS)

Egységár

ÁRULEÍRÁS

Árukód

Árunév

Áru_képe

Rendelési_egység

Raktárkód (RAKTÁR)

Mennyiségi_kedvezmény

Kedvezmény_határ

RENDELÉS

Rendelésszám

Vevőkód (VEVŐ)

Rendelési_Dátum

RENDELÉS-TÉTEL

Rendelésszám (RENDELÉS)

Negyedév

Árukód (ÁRUKATALÓGUS)

Szín

Méret (ÁRUVÁLASZTÉK)

Rendelt_mennyiség

Leszállítva

ÁRUVÁLASZTÉK

Árukód (ÁRULEÍRÁS)

Szín

Méret

Raktárkészlet

RAKTÁR

Raktárkód

Raktár_cím

Raktár_vezető

Raktár_telefon

Az adatbázis hatékony működtetéséhez az alábbi indexeket hozzuk létre:

- Egyedi indexet építünk fel minden elsődleges kulcsra.
- Egyedi index SZÁLLÍTÁS-ban (Vevőkód, Szállítási_Dátum)-ra. (A napi egyszeri szállítás feltétele miatt).
- Duplikált indexet készítünk az alábbi oszlopokra:
 - minden idegen kulcs,
 - árukód,
 - vevőkód,
 - árunév,
 - szállítási-dátum,
 - rendelési-dátum,
 - vevőnév,
 - vevő_bankszámla.

Az oszlopok tulajdonságainak meghatározását és az adatbázis egyszerűbb használatához szükséges nézetek definiálását az olvasóra bizzuk (ld. a fejezet 19. feladatát).

4.8. Ellenőrző kérdések

1. Milyen szempontokat célszerű figyelembe venni az objektumok és attribútumaik elnevezésénél?
2. Milyen kulcsokat ismer, és mi ezeknek a szerepe?
3. Adjon példákat olyan idegen kulcs kapcsolatokra, amikor kaszkád módosítást vagy törlést célszerű definiálnunk!
4. Írja le, hogyan lehet a 4.3.c ábrán (4-10. oldal) látható relációkba új osztályt felvenni!
5. Fölcserélhető-e egy reláción a korlátozás és a vetület elvégzésének a sorrendje?
6. Foglalja össze, hogy az egyes relációs műveleteknek milyen SQL utasítások felelnek meg!
7. Állítsa elő azt a relációt, amelyik minden oktatóhoz megadja a hallgatóinak a nevét!
8. Állítsa elő azt a relációt, amelyik
 - minden hallgatóhoz minden hozzákapcsolódó adatot (név, évfolyam, milyen tárgyakat kinél vett fel, hogy vizsgázott, stb.) tartalmaz
 - csak azokra a hallgatókra tartalmazza ezeket az adatokat, akik legalább egy tantárgyat felvettek.
9. Mi a normalizálás célja?
10. Sorolja fel, milyen normál formákat ismer, és mik ezek jellemzői!
11. Adjon példát olyan relációra, amelyik
 - nincsen első normál formában.
 - első normál formában van, de nincsen második normál formában.
 - második normál formában van, de nincsen harmadik normál formában.
 - harmadik normál formában van, de nincsen Boyce-Codd normál formában.
 - Boyce-Codd normál formában van, de nincsen negyedik normál formában.
12. Hányadik normál formában van a következő adatokat tartalmazó egyedtípus? (Az elsődleges kulcsot vastag betűvel jelöljük.)

RENDELÉS

Rendelészám	Rendelés-dátum	Árukód	Rendelt-mennyiség	Egységár
R10	2006-10-16	A1	10	100
		A2	20	200
R20	2006-10-17	A1	30	100

13. Hányadik normál formában van a következő adatokat tartalmazó egyedtípus?

RENDELÉS

Rendelészám	Rendelés-dátum	Árukód	Rendelt-mennyiség	Egységár
R11	2006-10-16	A1	10	100
R12	2006-10-16	A2	20	200
R21	2006-10-17	A1	30	100

14. Hányadik normál formában van a következő adatokat tartalmazó egyedtípus?

RENDELÉS

<i>Rendelészám</i>	<i>Rendelés-dátum</i>	<i>Árukód</i>	<i>Rendelt-mennyiség</i>
R11	2006-10-16	A1	10
R12	2006-10-16	A2	20
R21	2006-10-17	A1	30

15. Mik lehetnek alternatív kulcsok a következő egyedtípusban?

GÉPJÁRMŰ

Rendszám, Alvázszám, Tulajdonos-neve, Motorszám, Teljesítmény, Szín,
CASCO-kötvény-száma, Kötelező-biztosítási-kötvény-száma,
Üzembe-helyezés-napja

16. Bizonyítsa be, hogy a 4.6.7.1. pontban megadott felbontás az eredeti reláció minden sorát egyszer és csak egyszer adja vissza!

17. Fel tudja-e a 4.6.7.1. pontban megadott TANÁCSADÁS relációt más módon is veszteség mentesen és teljesen reverzibilisen bontani BCNF relációkra?

18. Készítse el az alábbi feladat normalizált adatmodelljét:

A sportolók sportáganként különböző versenyeken vesznek részt. Egy személy több számban is indulhat. Elemezze azt a relációt, amelyik megadja, hogy ki, milyen sportágban milyen helyezést ért el! Nincs megosztott helyezés. Hogyan építi fel a reláció(ka)t, ha nyilván kell tartani azokat a sportolókat is, akik elindultak, de nem értek el helyezést?

19. Definiálja a 4.7.3. pontban megadott csomagküldő rendszer oszlopainak jellemzőit és készítse el az egyszerűbb használatához szükséges nézeteket, melyek tartalmazzák

- minden árunak minden tulajdonságát.
- a megrendelések minden fontos adatát egy megrendelőlevélben.
- a szállítások minden adatát egy szállítólevélben az igénybe vehető kedvezményekkel együtt.

20. Készítse el egy vállalat dolgozói nyilvántartásának Entity – Relationship diagrammját, definiálja az egyes relációk oszlopait és kulcsait!

Minden dolgozónál szükségünk van a következő adatokra (zárójelben az oszlop neve):

- személyi azonosító szám (SZEMSZ)
- név (NÉV)
- osztály kódja, ahol dolgozik (O-KÓD)
- osztály megnevezése (O-NÉV)
- téma kódja, amelyen dolgozik (T-KÓD)
- téma megnevezése (T-NÉV)
- munkaidejének hány százalékában dolgozik a témán (MEGOSZL)
- beosztása (BEOSZT)

- a beosztásához tartozó maximális és minimális fizetés (MAX-FIZ, MIN-FIZ)
- 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,
- a beosztás egyértelműen meghatározza a hozzátartozó maximális és minimális fizetést,
- egy dolgozó több osztályhoz is tartozhat és több témán is dolgozhat,
- egy dolgozó egy adott osztályon csak egy beosztásban dolgozhat, a különböző osztályokon azonban más lehet a beosztása,
- a témák, beosztások és osztályok között nem kell kapcsolatot teremtenünk,
- nem kell automatikusan ellenőriznünk, hogy a témák összességére fordított idő a dolgozó teljes munkaideje-e (100%).

5. AZ SQL NYELV

Ebben a fejezetben megismeri az olvasó az SQL nyelv felépítését és használatát. Mivel az SQL nyelv nagymértékben szabványosított, az itt leírtakat a gyakorlatban előforduló feladatok legnagyobb részében minden adatbázis-kezelő rendszerben egy az egyben alkalmazni lehet. Ezért a anyagot úgy állítottuk össze, hogy a tanulás mellett kézikönyvként is használható legyen. Emiatt bizonyos dolgokat több helyen is megismételünk, hiszen munka közben igen kellemetlen, ha a referenciaként használt könyvben állandóan kereszthivatkozásokra kell lapozni.

A gyakorlati felhasználó számára igyekeztünk minél több példán bemutatni az egyes elemek használatát. Ez magával vonta azt, hogy bizonyos esetekben a példában olyan funkciók is szerepelnek, melyeket csak később fogunk tárgyalni. Ilyenkor az első olvasásnál ezeket a részeket át lehet ugrni és csak a fejezet megfelelő részének megértése után térjen rá vissza az olvasó.

5.1. SQL szabvány

Az SQL (a Magyarországon nem használatos angolul *szíkvel*-nek mondott rövidítés) a Structured Query Language, szó szerint Strukturált Lekérdező Nyelv elnevezés rövidítése.

Valójában azonban sokkal több ennél. Egy teljes relációs adatbázis-kezelő nyelv, ami szabvánnyá lett. Napjainkban minden adatbázis-kezelő rendszer ezt illetve ennek valamely részhalmazát használja. A grafikus felületeken kommunikáló rendszerek, mint például a 6.1. alfejezetben ismertetett ACCESS is erre a nyelvre fordítják át a kijelölt feladatokat.

Az SQL nyelv kifejlesztése és szabványként való elfogadása egyike az informatika azon területeinek, ahol a számítástechnika gyakorlati kivitelezői és a számítástudomány elméleti szakemberei szinte a kezdetektől végig eredményesen működtek együtt. A relációs adatmodell elvi alapjait E. F. Codd rakta le 1970-ben megjelent cikkével. Erre alapozva dolgozták ki az IBM San Jose-i (Kalifornia) kutatólaboratóriumában az SQL nyelvet és készítették el az első, erre a nyelvre épülő adatbázis-kezelő rendszert. A módszert más szoftvergyártó cégek is átvették és hamarosan a legkülönbözőbb hardvereken és operációs rendszerek alatt nagy számban működtek SQL-en alapuló rendszerek. Az American National Standards Institute (ANSI) és az International Standards Organization (ISO) már 1978-ban ajánlásokat dolgozott ki a nyelv szabványosítására és ezeket több ízben (1986, 1989, 1992, 1999) pontosította. Ennek eredményeképpen alakult ki a nyelv mai formája. Az SQL ismeretében elvileg akármilyen relációs adatbázis-kezelő rendszerrel dolgozni tudunk, és az egyik rendszerben elkészített megoldásokat átvihetjük bármelyik másikra.

A valóságban természetesen nem ilyen szép a helyzet. A számítógéppel kapcsolatot biztosító operációs rendszer más-más környezetet szolgáltat, sőt, még ugyanazon operációs rendszerhez illeszkedő adatbázis-kezelő rendszerek is kissé „más dialektusban beszélhetnek”. Általában a szabványosított SQL-nek más-más részhalmazát engedik meg, de időnként speciális egyedi bővítési lehetőségeket is nyújtanak. Ennek ellenére biztosan állíthatjuk, hogy az SQL nyelv ismeretében bármely adatbázis-kezelő rendszerrel rövid időn belül el fogunk boldogulni.

Az SQL lényegében egy nem-procedurális nyelvnek felel meg. A felhasználó csak azt mondja meg, hogy *mit akar*. Azt, hogy ez miként, milyen lépések sorozatával valósuljon meg, a nyelv értelmezője, fordítóprogramja fogja meghatározni az adatbázis szerkezetének és az adatok eloszlásának ismeretében. Az optimális feldolgozás egyes lépéseinek meghatározásához az SQL értelmező az adatbázis adatszótárából, a katalógusból (ld. 7. fejezet) nyeri ki az információkat.

Az SQL nyelvet használhatjuk interaktívan, önálló nyelvként is, és beépíthetjük programokba is. A leggyakoribb befogadó nyelvek a C, C++, Java, PHP és a régóta üzemelő nagy feldolgozásoknál a szobatudósok által igazságtalanul már rég eltemetett COBOL.

A legtöbb SQL utasításnak az önálló és a programba beépíthető formája megegyezik. Kivétel ez alól a *SELECT* utasítás, mert ennek programba való kiadásakor biztosítanunk kell azokat a programváltozókat, ahová az adatokat beolvassuk. Ezen kívül van néhány olyan SQL utasítás is (pl. *DECLARE CURSOR*, *FETCH*), amit csak programból használhatunk.

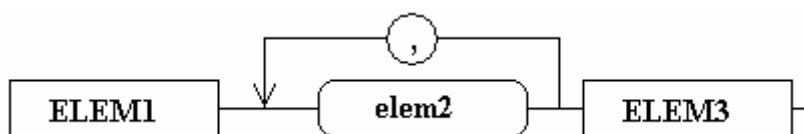
5.2. Szintaxis jelölés

Az SQL utasításoknak szigorú szintaktikai szabályai vannak. Amennyiben ezeket nem tartjuk be, akkor az utasításértelmező nem fogja tudni értelmezni őket. Ezért ismernünk kell, hogy az egyes utasításokat pontosan milyen formában adhatjuk ki. Az utasítások szintaktikáját legpontosabban metanyelven definiálhatnánk. Ez azonban az átlagos felhasználók többsége számára túl bonyolult lenne. Ezért a gyakorlatban régebben egy csupán alfanumerikus jeleket használó formátumot használtak, míg az utóbbi időkben az egyszerű grafikai szimbólumokat alkalmazó szintaxis diagramok terjedtek el. Ezek gyártó cégenként kissé különböznek, de felépítésüket tekintve lényegében azonosak. A továbbiakban az ORACLE által használatos szintaxis diagrammokat ismertetjük.

Az SQL utasításokat kis- és nagybetűkkel egyaránt írhatjuk. Különbség csak a szöveges literálként (állandóként) megadott paramétereknél van. Ezekben a kis- és nagybetűk különböznek. *CREATE TABLE HALLGATO ...*, *CREATE TABLE hallgato ...*, *create table hallgato*, ... egyaránt helyes és ugyanazt eredményezi, de *SELECT ... WHERE VNEV='Nemecsek' ...* és *SELECT ... WHERE VNEV='nemecsek' ...* különböző eredményt szolgáltatnak.

A csak a megadott formában írható parancs- és kulcsszavakat téglalapba zárva nagybetűkkel írva jelöljük. (A nagybetűk csak a szintaxis diagrammban kötelezőek. A konkrét utasításban kis- és nagybetűket egyaránt használhatunk.) Az általunk megválasztható formában megadható paramétereket, változókat legömbölyített alakzatban, kisbetűkkel írva jelöljük. A kizárólag az adott formában írható elválasztó jeleket (pl. vessző, pont, stb.) körbe zárva jelezzük.

A szintaxis diagrammot balról jobbra, illetve ha szükséges, akkor a sor végétől fentről lefelé a következő sor elejétől kezdve olvassuk. Amennyiben ettől az iránytól eltérhetünk, akkor ezt nyíllal jelezzük. A visszafelé mutató nyíl az elem – elvileg tetszőleges számú – megismételhetőségét jelenti.



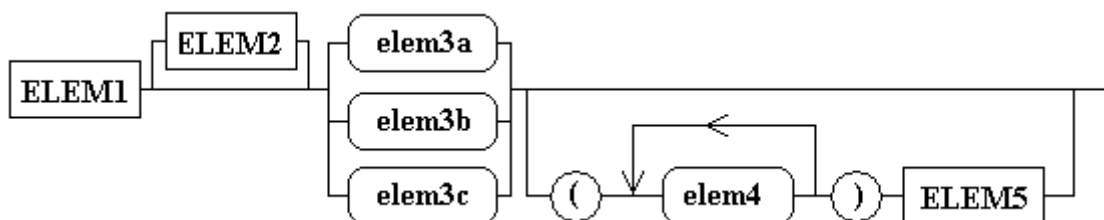
Az *ELEM1*-et és *ELEM3*-at pontosan ebben a formában kell írunk. Az *elem2*-t, mint azt a visszafelé mutató, nyíllal jelzett útvonal mutatja, többször és tetszőleges névvel, az adott paraméterre engedélyezett formában adhatjuk meg. Az egyes előfordulások közé a szintaxis diagramm által jelzett módon vesszőt (,) kell tennünk. (Az utolsó előfordulás után már nem kell a vessző!) Az *ELEM3* az *elem2* utolsó előfordulása után következik. A fenti szintaxis elem egy lehetséges megadási formája:

elem1 PARAM2,para2,Pr2 ELEM3

ahol *PARAM2*, *para2*, *Pr2* az *elem2* megengedett alakjai.

A főútvonalon, az utasítás elejétől a végéig vivő vízszintes vonalon levő elemeket kötelező megadni. Így az előző példában mindegyik elemet meg kell adnunk.

Ha több lehetőség között választhatunk, akkor az útvonal szétágazik. Ha az elem opcionális (elhagyható), akkor van egy egyenesen haladó, töretlen útvonal – a főútvonal. Ha egy és csakis egy lehetőséget kell megadnunk, akkor mindegyik útvonal valamelyik elemen keresztül vezet és nincs visszavivő út. Ha az elemek ismételhetőek, akkor van egy visszavivő út is és ebben jelezzük – ha van – az elválasztó jelet is.



Az *ELEM1*-et ebben a formában kell írunk. Az *ELEM2*-t elhagyhatjuk, de ha megadjuk, pontosan így kell írunk. Az *elem3a*, *elem3b*, *elem3c* paramétereiből egyet és csak egyet kell megadnunk tetszőleges névvel, az adott paraméterre engedélyezett formában. A következő, két elemből álló tagot elhagyhatjuk. Ha megadjuk, akkor az *elem4* paramétert zárójelek között akárhányszor megismételhetjük, ahogy azt a visszafelé mutató, nyíllal jelzett útvonal mutatja. A szóközön kívül külön elválasztó jel nem kell az ismétlődések közé. Utánuk az *ELEM5* tagot pontosan így ki kell írunk.

A fenti szintaxis elem lehetséges megadási formái például:

ELEM1 PARAM3b

ELEM1 ELEM2 pr3c (pr4 param4 PAR4) elem5

ahol *PARAM3b*, *pr3c* az *elem3b* illetve *elem3c* egy megengedett alakja, *pr4*, *param4*, *PAR4* pedig *elem4* lehetséges értékei.

Az utasítás parancsszavait, kulcsszavait és paramétereit legalább egy szóköz választja el.

5.3. A nyelv elemei

Formai szempontból az SQL nyelv a következő elemekből épül fel:

- SQL fenntartott szavak,
- azonosítók,
- állandók (literálok),
- operátorok,
- határoló jelek.

Tartalmi szempontból a következő alapelemeket különböztetjük meg:

- objektumok és azok komponensei,
- adattípusok,
- NULL érték,
- SQL kifejezések,
- SQL feltételek,
- SQL függvények,
- SQL utasítások.

Ezek közül az adattípusokat, az SQL függvényeket és utasításokat fontosságuk és terjedelmük miatt ezen fejezet külön alfejezeteiben, a többieket ebben az alfejezetben ismertetjük.

5.3.1. Fenntartott szavak

A fenntartott szavak az SQL utasítások meghatározására, pontosítására, olvashatóbbá tételére, valamint a különböző objektumuk tulajdonságainak leírására szolgálnak. Ezeket a szavakat csak az így meghatározott módon és csak az így kijelölt feladatra használhatjuk. A szintaxis jelölésben (ld. 5.2. alfejezet) ezeket téglalapba írt nagybetűkkel jelöljük. A fenntartott szavak listáját az 5.1. táblázat tartalmazza. A *dőlt betűkkel* szedett szavak az SQL ANSI szabvány szerint minden rendszerben foglaltak. A többiek rendszerspecifikusak, de használatuk a portabilitás érdekében nem ajánlott. Ugyancsak kerüljük a *SYS* kezdetű azonosítókat, mivel sok rendszer (pl. az ORACLE) ilyen kezdetű elemeket generál automatikus elnevezésekkor.

5.1. táblázat: **SQL fenntartott szavak**

ACCESS	ADD	ALL	ALTER	AND
ANY	AS	ASC	AUDIT	BETWEEN
BY	CHAR	CHECK	CLUSTER	COMMENT
COMPRESS	CONNECT	CREATE	CURRENT	DATE
DECIMAL	DEFAULT	DELETE	DESC	DISTINCT
DROP	ELSE	EXCLUSIVE	EXISTS	FILE
FLOAT	FOR	FROM	GRANT	GROUP
HAVING	IDENTIFIED	IMMEDIATE	IN	INCREMENT
INDEX	INITIAL	INSERT	INTEGER	INTERSECT
INTO	IS	LEVEL	LIKE	LOCK
LONG	MAXEXTENTS	MINUS	MLSLABEL	MODE
NOAUDIT	NOCOMPRESS	NOT	NOWAIT	NULL
NUMBER	OF	OFFLINE	ON	ONLINE
OPTION	OR	ORDER	PCTFREE	PRIOR
PRIVILEGES	PUBLIC	RAW	RENAME	RESOURCE
REVOKE	ROW	ROWID	ROWNUM	ROWS
SELECT	SESSION	SET	SHARE	SIZE
SMALLINT	START	SUCCESSFUL	SYNONYM	SYSDATE
TABLE	THEN	TO	TRIGGER	UID
UNION	UNIQUE	UPDATE	USER	VALIDATE
VALUES	VARCHAR	VARCHAR2	VIEW	WHENEVER
WHERE	WITH			

5.3.2. Azonosítók

Az SQL objektumoknak és alkotóelemeinek nevet kell adnunk, hogy hivatkozhatunk rájuk. Ezt a nevet hívjuk az adott adatbáziselem *azonosítójának*. A különböző elemek nevének képzésére szigorú szintaktikai szabályok érvényesek. Emellett a névnek az adott környezetben egyértelműnek is kell lenni.

Az SQL azonosítók az angol ABC betűiből, számokból és a név belsejében az _ (aláhúzás) jelből állhatnak. Maximális hosszuk rendszer és objektumfüggő. Az adatbázisok neve általában maximum 8, míg a táblázatok, nézetek, oszlopok neve 18 illetve egyes

rendszereknél akár 30 bájt hosszú is lehet. Néhány adatbázis-kezelő rendszerben az azonosítók bizonyos körülmények között tartalmazhatnak nemzeti karaktereket és a név belsejében szóközt is, de ezek használatát, mivel ez más környezetbe való áttérésnél problémát okozhat, lehetőleg kerüljük.

Hogy a különböző felhasználóknak ne kelljen azzal törődniük, hogy a többi felhasználó milyen azonosítókat használ, az SQL azonosítók több szintből tevődnek össze. Az egyes szintek között általában a . (pont) az elválasztó jel. Ezek közül csak a legelső szint nevét kötelező megadni.

A felhasználónak csak azzal kell törődnie, hogy az objektumainak az adatbázisban, az objektumok egyes komponenseinek pedig az objektumon belül egyértelmű legyen a neve. Így például ugyanaz a felhasználó nem hozhat létre saját magának egy *MINTA* nevű táblázatot és egy másik ugyancsak *MINTA* nevű táblázatot, de ilyen névvel más objektumot (pl. nézetet vagy sorszám generátort) sem. Ugyanakkor akárhány felhasználónak lehet *MINTA* nevű objektuma és abban egy darab, mindegyikben azonosan *OSZLOPI*-nek nevezett oszlopa is.

Az azonosító egyértelművé tételére minden objektumnév elé minősítőként odakerül az objektum tulajdonosának a neve. Ez az objektum létrehozójának, vagy annak az azonosítója, akinek a nevében azt az adatbázis felügyelő létrehozta. Osztott adatbázisoknál rendszertől függ, hogy elé vagy mögé jön még minősítőnek az adatbázis helyének a neve. Integrált adatbázisoknál, amelyek csak egy helyen vannak, ez természetesen felesleges. Osztott adatbázisoknál, ha külön nem adjuk meg, automatikusan az aktuálisan használt helyi adatbázis megfelelő nevű eleméről beszélünk. Ezek alapján egy adatbáziselem azonosítóját a következő módokon adhatjuk meg:

ADATBÁZIS-HELY.TULAJDONOS.OBJEKTUM-NÉV.ELEM-NÉV

TULAJDONOS.OBJEKTUM-NÉV.ELEM-NÉV@ ADATBÁZIS-HELY

ilyen *a teljes osztott* adatbázisban csak egy lehet

TULAJDONOS.OBJEKTUM-NÉV.ELEM-NÉV

ilyen egy helyi adatbázison belül csak egy lehet

OBJEKTUM-NÉV.ELEM-NÉV

ilyen *egy helyi* adatbázison belül *egy felhasználónak* csak egy lehet

ELEM-NÉV

ilyen egy felhasználónak egy objektumán belül csak egy lehet.

A fenti névhierarchiából elegendő annyi szintet megadnunk, amennyi az adott környezetben (általában SQL utasításban) az elemet egyértelművé teszi. Így például, ha egy integrált adatbázisban az *SQL00* felhasználó a saját *HALLGATO* táblázatából akarja kiválasztani a vezetéknévet tartalmazó *VNEV* oszlopot, akkor elegendő erre a

SELECT VNEV, ... FROM HALLGATO

módon hivatkozni, mert a *VNEV*-et a rendszer automatikusan kiegészíti az utasításból egyértelmű *SQL00.HALLGATO* minősített táblázat névvel. Ha viszont az *SQL30* táblázatából akarja ugyanezt az oszlopot, akkor a

SELECT VNEV, ... FROM SQL30.HALLGATO ...

utasításban a *VNEV* egyértelműen az *SQL30.HALLGATO* táblázatra vonatkozik. A

SELECT SQL30.HALLGATO.VNEV, ... FROM SQL30.HALLGATO,
HALLGATO ...

utasításban viszont már egyértelművé kell tenni, hogy a mindkét táblázatban szereplő VNEV oszlop közül melyiknek az értékét kell venni.

5.3.3. Állandók (literálok)

A állandókat SQL kifejezésekben használjuk. Megadásukkal egy állandó értéket határozunk meg. Tartalmuk szerint megkülönböztetünk

- Numerikus
- Alfánnumerikus (karakter)
- Dátum/időpont típusú állandókat.

A *numerikus állandók* legalább 1 és legfeljebb a rendszer számbábrázolásának megfelelő számú számjegyet (általában maximum 18 jegy) és esetleg egy előjelet valamint egy tizedes pontot vagy vesszőt tartalmaznak. Az, hogy a rendszer tizedes ponttal vagy tizedes vesszővel dolgozzon, általában beállítható. Sok rendszer megengedi a lebegőpontos állandók használatát is (Pl. -1.234 E 12 formában, ami $-1,234 \cdot 10^{12}$ -t jelent).

Az *alfánnumerikus állandók* tetszőleges alfánnumerikus karaktereket (betűk, számjegyek, különleges karakterek) tartalmazhatnak. Az állandó kezdetét és végét ' (aposztróf) jelzi. Egyes rendszerekben lehet az aposztróf helyett idézőjelet (") használni. Az aposztrófok közötti jelsorozat az állandó értéke. A literálok maximális hossza rendszertől függ, általában 256 és 4000 bájt között van. Egyes adatbázis-kezelő rendszerek különleges ABC-k (japán, kínai, stb.) írásjeleit is megengedik.

A *dátum/időpont* típusú állandókkal dátum/időpont típusú (*DATE/TIME*) adatelemeknek (ld. 5.4.3. pont) adhatunk értéket, hasonlíthatjuk össze velük. Értéküket általában az alfánnumerikus állandókhoz hasonlóan aposztrófok között adjuk meg és formátumuk meg kell feleljen a rendszerben vagy az utasításban egyedileg meghatározott dátum formátumnak. Magyarországon a leggyakrabban használt dátum formátumok:

'évvévvévv-hóhó-napnap'

'évvévvévv/hóhó/napnap'

'évvévvévv-hóhó-napnap.óraóra.percperc.mpmp'

'óraóra.percperc.mpmp'

(Az évek kivételével az egyes elemeket egy vagy két számjeggyel is írhatjuk, a nap 24 órás, pont helyett kettőspont vagy pontosvessző is lehet az elválasztó jel).

A állandókra példákat az 5.2. táblázatban mutatunk be.

5.2. táblázat: **Példák állandókra**

Numerikus	12	-2,3567	+1234567	-1,23 E 44
Alfánnumerikus	'ABC'	'abc'	'12'	'A1@b.hu'
Dátum	'2006-12-31'	'1999-1-1'	'2008/02/29'	'15.45.58'

5.3.4. Operátorok és feltételek

Az SQL operátoroknak egy vagy két operandusa (argumentuma) lehet. Az operátorok speciális jelek (pl. + vagy *) vagy speciális SQL kulcsszavak (pl. *IS NULL* vagy *LIKE*) lehetnek.

Az operandus lehet

- oszlop
- állandó (literál)
- SQL függvény
- kifejezés
- operátor az operandusaival

Egyes speciális operátoroknál (IN, ANY, ALL) a második operandus egy több elemből álló lista is lehet.

Az operátorokkal összekapcsolt operandusok által meghatározott kifejezés egy egyértelmű értéket határoz meg. Ennek kiértékeléséhez figyelembe kell venni az operandusok precedencia sorrendjét. Ez (a legmagasabbal kezdve) a következő:

- előjel (+, -)
- szorzás és osztás (*, /)
- összeadás és kivonás (+, -)
- összehasonlító operátorok (pl. <, =)
- NOT logikai operátor
- AND logikai operátor
- OR logikai operátor

Azonos precedencia szinten belül a kiértékelés balról jobbra történik. A precedencia szabályt zárójelekkel felülbírálhatjuk.

Az SQL nyelvben használatos legfontosabb operátor típusok

- aritmetikai operátorok,
- összehasonlító operátorok,
- logikai operátorok,
- halmaz operátorok.

Az operátorokat – típusuktól függően – a *SELECT* utasítás kiválasztási listájában, vagy a *WHERE* vagy *HAVING* utasítás-kiegészítésében, szubszelektek összekapcsolásában, (ld. 5.6.3.2. pont) az *INSERT* utasítás *VALUE* (ld. 5.6.3.5. pont), a *DELETE* utasítás *WHERE* (ld. 5.6.3.3. pont) és az *UPDATE* utasítás *SET* és *WHERE* (ld. 5.6.3.4. pont) utasítás-kiegészítésében használhatjuk.

Az operátorok használatát példákon mutatjuk be. Ezeket az első nekifutáskor – ha tartalmuk még nem érthető - az olvasó kihagyhatja és az SQL utasítások megértése után tanulmányozza majd át újra.

5.3.4.1. Aritmetikai operátorok

Az *aritmetikai operátorokat* aritmetikai és dátum kifejezésekben, valamint előjelként használjuk ugyanúgy, mint a matematikai kifejezésekben. Az operátorok precedencia szabályai is ugyanazok.

Példa:

```
select * from merleg where egyenleg <= regi_egyenleg - 10000
```

5.3.4.2. Összehasonlító operátorok

Az összehasonlító operátorokat feltételekben használjuk kifejezések összehasonlítására. A rendszer az operátor két oldalán levő elemeket összehasonlítja az operátor által meghatározott módon. Ennek az eredménye vagy „IGAZ” (TRUE) vagy „HAMIS” (FALSE). A matematikában használatos operátorok (=, <, <=, >, >=, <> nem egyenlő jelölése) mellett használhatjuk még az 5.3. táblázatban látható speciális SQL összehasonlító operátorokat is.

5.3. táblázat: Speciális SQL összehasonlító operátorok

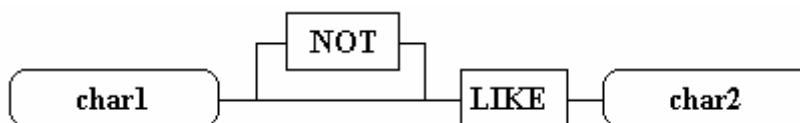
Operandus	Feladata	Példa
IN	Bármelyikkel egyenlő vizsgálata.	Select * from tantargy where tid IN ('SQL', 'DB', 'ORCL') (Kiválasztja TANTARGY-ból azokat a sorokat, melyeknél a TID tantárgykód a felsorolt értékek bármelyikével egyenlő.)
ANY vagy SOME	Bármelyikre igaz vizsgálata.	select * from hallgato where hid = ANY (select distinct hid from mithallgat where osztalyzat = 5) (Kiválasztja HALLGATO-ból azon hallgatók adatait, akiknek van jeles osztályzatuk. A belső beágyazott szelekt kiválasztja MITHALLGAT-ból azon hallgatók azonosítóit, akiknek van jeles osztályzatuk, a külső szelekt pedig azok összes adatát, akiknek azonosítója bármelyik jeles azonosítójával megegyezik)
ALL	Mindegyikre igaz vizsgálata.	select * from sql00.mithallgat where gyakpont > ALL (select gyakpont from sql30.mithallgat) (Kiválasztja SQL00.MITHALLGAT-ból azokat a sorokat, ahol a GYAKPONT értéke nagyobb minden SQL30.MITHALLGAT GYAKPONT értéknél.
BETWEEN x AND y	Két érték között (>= x és <=y) vizsgálata.	select * from mithallgat where gyakpont BETWEEN 51 and 60 (Kiválasztja azokat a sorokat, melyekben GYAKPONT a megadott két érték között van, a határértékeket is beleértve)
EXISTS	Létezés vizsgálata.	insert into sql30.hallgato select * from sql00.hallgato h00 where EXSISTS (select * from sql80.hallgato h80 where h00.hid = h80.hid) (Csak azokat a sorokat írja be SQL00.HALLGATO-ból SQL30.HALLGATO-ba, melyek HID-je szerepel SQL80.HALLGATO-ban is.

Operandus	Feladata	Példa
IS NULL	NULL érték vizsgálata.	select hid, tid from mithallgat where osztalyzat IS NULL (kiválasztja azokat a hallgatókat és azokat a tantárgyakat, amikből még nem vizsgáztak le, azaz <i>OSZTÁLYZAT NULL érték</i>)
LIKE	Lásd külön felsorolva	

Megjegyzések:

- Csak azonos típusú operandusokat lehet összehasonlítani. Az összehasonlítási sorrend
 - numerikus operandusoknál a számok előjeles értéke
 - dátum típusú operandusoknál az időpont
 - Jelsorozatoknál a belső kód szerinti sorrend. A szokásos ASCII kódnál karaktereknél a növekvő sorrend: szóköz, speciális karakterek, számjegy, nagy betűk, kis betűk.
 - Ügyeljünk számok karakteres ábrázolásánál! '2' > '12', de '02' < '12' és '2' < '12'
- A *NOT* megfordítja az utána következő operátor igazságtáblázatát.
- Az *ANY* és a *SOME* egyenértékűek. Előttük az =, <>, <, >, <=, >= operátorok valamelyikének kell állni. A feltétel IGAZ, ha az összehasonlítás az ANY-lista legalább egy elemére igaz, HAMIS, ha egyre sem.
- Az *ALL* előtt az =, <>, <, >, <=, >= operátorok valamelyikének kell állni. A feltétel IGAZ, ha az összehasonlítás az ALL-lista mindegyik elemére igaz, HAMIS, ha legalább egyre nem igaz.
- A *BETWEEN*-nél a határértékek is beletartoznak a feltételnek eleget tevő értékek körébe. (>= ÉS <= -nek felel meg.)
- Az összehasonlító operátorral meghatározott (egyszerű) feltételeket logikai operátorokkal (*AND*, *OR*) összetett feltételekké kapcsolhatjuk össze. Ügyeljünk a műveletek sorrendjére és ha szükséges, akkor ezt zárójelekkel tegyük érthetőbbé, illetve egyértelművé.
- A *NULL* értékre vonatkozó speciális szabályok:
 - A *NULL* értékre csak az *IS NULL* vagy az *IS NOT NULL* operátorokkal kérdezhetünk rá.
 - Ha az =, <, >, <=, >=, <> operátorok bármelyik operandusa *NULL érték*, akkor a feltétel értéke se nem „IGAZ”, se nem „HAMIS”, hanem „MEGHATÁROZATLAN” (ld. 5.3.4.3. pont 5.7. táblázat).
 - A *NULL* értéket semmilyen más értékkel nem hasonlíthatjuk kiértékelhető módon össze. Ha pl. *EVF* értéke egy sorban a *NULL érték*, akkor ez a sor az *EVF = 4*, *EVF > 4*, *EVF < 4*, *EVF <> 4* feltételek egyikénél sem jelenik meg. Kivétel, hogy növekvő rendezettségű sorrendben (*ORDER BY ... ASC*) való megjelenítésnél a *NULL* értékek (beállítástól függően) a rendezett halmaz elejére/végére, csökkenő sorrendnél pedig (*ORDER BY ... DESC*) a végére/elejére kerülnek.
- Speciális operátor a *LIKE* operátor, ami egy jelsorozatot hasonlít össze egy adott mintával. Igen gyakran használjuk azokban az esetekben, amikor a keresendő jelsorozat értéket nem ismerjük egészen pontosan.

Formátuma:



Magyarázat:

A művelet a *char-1* karakter értéket szolgáltató oszlop, kifejezés vagy függvény értékét karakterenként összehasonlítja az ugyancsak karakter értéket adó *char-2*-ben levő mintával, mely *joker*, vagy más néven *helyettesítő*, vagy *maszk karakter(ek)e*t is tartalmazhat. A szabványos joker karaktereket az 5.4. táblázatban foglaltunk össze. A joker jelek megváltoztathatók, egyes rendszerekben a % helyett a * (csillag) az alapértelmezés.

5.4. táblázat: **Joker karakterek és funkciójuk**

Joker karakter	Funkciója
% (százalék jel)	Akárhány karaktert helyettesíthet. Pl.: 'ABcd', 'AB', 'ABX12' mindegyikére igaz a LIKE 'AB%' feltétel.
_ (aláhúzás)	Pontosan egy karaktert helyettesít. Pl.: 'x0123', 'x2123' mindegyikére igaz a LIKE 'x_123' feltétel, de 'x11123'-ra hamis. (Erre a LIKE 'x%123' igaz.

Megjegyzések:

- A *LIKE* feltétel *igaz* ha a *char-1* tartalma megegyezik a *char-2* mintával, azaz a nem joker karakterek sorrendben karakterről karakterre azonosak mind a kettőben. A *NOT LIKE* feltétel *igaz* ha a *char-1* tartalma bármiben eltér a *char-2* minta joker(ek) nélküli karaktereitől.
- Az összehasonlításnál a kis- és nagybetűk különböznek. Ha ez problémát okozhat, használjuk az *UPPER* vagy a *LOWER* függvényt. (ld. 5.5.2.1. pont és a 2. példa)
- A *joker jel(ek)*-et a minta bármely részén elhelyezhetjük.
- *LONG* típusú oszlopra nem használhatjuk a *LIKE* feltételt.
- Ha a *LIKE* feltétel *char-2* mintájában az első karakter % vagy _ (joker karakter) akkor az SQL nem tud indexet használni. Az ilyen feltételeket rossz hatékonyságuk miatt lehetőleg kerüljük.

Példák:

1. Ki akarjuk íratni MITHALLGAT-ból azoknak a hallgatóknak az adatait, akik az 'ORCL', 'DB2', 'SQL', 'MSQL' tantárgyak bármelyikét felvették.

Utasítás:

```
select * from mithallgat  
where tid in ('ORCL', 'DB2', 'SQL', 'MSQL');
```

2. Meg akarjuk tudni, kik azok a hallgatók, akik asszonynevükön szerepelnek a HALLGATO táblában. Mivel nem tudjuk, hogy a VNEV vezetéknev és a KNEV

keresztnev oszlopokban milyen betűkkel adták meg a neveket, az *UPPER* függvényt is használjuk.

Utasítás:

```
select vnev,knev from hallgato
where upper(vnev) like '%NÉ' or upper(knev) like '%NÉ';
```

Eredmény: Megkapjuk az összes olyan hallgatót, akinek vagy a vezetéknév- vagy a keresztnév „né”, „NÉ”, vagy a logikailag nem valószínű „Né”, vagy „nÉ”-re végződik. (Természetesen ezek nem biztosan férjezett nők. Akinek például *Spiné György* a neve, az valószínűleg nem az.)

- (Elégé elítélendő módon) nem készítettünk idegen kulcsokat a *MITHALLGAT* táblában a *HID* hallgató azonosítóra. Meg akarjuk tudni, hogy van-e a táblázatban nem létező hallgató.

Utasítás:

```
select hid, tid from mithallgat where not exists
(select hid from hallgato where mithallgat.hid = hallgato.hid);
```

Eredmény: Kiírja mindazon *HID,TID* párokat, melyet olyan azonosítóval rendelkező hallgató vett fel, aki nem szerepel a *HALLGATO* táblázatban.

5.3.4.3. Logikai operátorok

A *logikai operátorok* (*AND*, *OR*, *NOT*) egy vagy két feltételből egy új feltételt állítanak elő. Használatukra a Boole-algebra szabályai érvényesek. A *NULL érték* miatt azonban az összetett feltételek igazságtáblázatai kiegészülnek az 5.5., 5.6. és 5.7. táblázatban látható módon. A ? az jelzi, hogy a feltétel (például *NULL* értékű operandus miatt) nem értékelhető ki, az eredmény meghatározhatatlan.

5.5. táblázat: A *NOT* operandus igazságtáblázata

Feltétel	IGAZ	HAMIS	?
NOT feltétel	hamis	igaz	?

5.6. táblázat: Az *AND* operandus igazságtáblázata

	Feltétel-2			
Feltétel-1	AND	IGAZ	HAMIS	?
	IGAZ	igaz	hamis	?
	HAMIS	hamis	hamis	hamis
	?	?	hamis	?

5.7. táblázat: Az **OR** operandus igazságtáblázata

	Feltétel-2			
Feltétel-1	OR	IGAZ	HAMIS	?
	IGAZ	igaz	igaz	igaz
	HAMIS	igaz	hamis	?
	?	igaz	?	?

Példa:

1. Utasítás részlet:

... where oszlop1_ertek in (1, 4, 8, 88)
or (oszlop1_ertek = 0 and oszlop2_ertek between 100 and 9999)

Eredmény: Az *in*, *or*, *=*, *and*, *between* operátorok (logikai) feltételeket hoznak létre, melyeknek végeredménye az operandusok értékétől függ. Igaz ha oszlop1_ertek az 1, 4, 8, 88 értékek bármelyikével egyenlő, vagy az oszlop1_ertek = 0 és ugyanakkor 100 <= oszlop2_ertek <= 9999.

5.3.4.4. Halmaz operátorok

A halmazoperátorok két szubszelekt eredményéből egy új eredményt állítanak elő. Ez a szubszelektok részeredményeinek a soraiból áll össze. A kiválasztott sorokat a halmazoperátor határozza meg. Az SQL halmaz operátorokat és funkcióikat az 5.8. táblázatban foglaltuk össze.

5.8. táblázat: SQL halmaz operátorok

Operátor	Funkciója
UNION	Mindkét szubszelekt minden sorát tartalmazza, de az azonos sorokat csak egyszer.
UNION ALL	Mindkét szubszelekt minden sorát tartalmazza.
INTERSECT	Csak azokat a sorokat tartalmazza, amelyek mindkét szubszelektben előfordulnak, de az azonos sorokat csak egyszer.
MINUS	Csak azokat a sorokat tartalmazza, amelyek az első szubszelektben előfordulnak, de a másodikban nem. Az azonos sorok csak egyszer szerepelnek.

Megjegyzések:

- Kettőnél több szubszelekt eredményét a halmaz operátorok ismételt alkalmazásával kapcsolhatjuk össze. Az operátorok egyenrangúak, a kiértékelés balról jobbra történik.
- Ha nincs utána rendezés, akkor a *UNION ALL* gyorsabb, mint a *UNION*. Ezért ha logikailag tudjuk, hogy nem lehetnek az eredményben azonos sorok, akkor mindig a *UNION ALL*-t használjuk.
- A *szubszelekt*-ek kiválasztási listája azonos számú és azonos típusú elemeket kell tartalmazzon azonos sorrendben rendezve. (Az egyes elemek nevei különbözhetnek az egyes szubszelektekben.)

Példák:

1. Írassuk ki saját és *SQL30 TANTARGY* táblázatából a tantárgyak neveit (*TMEGNEV*) és oktatóit (*OKTATO*). Tüntessük fel azt is, hogy az adat melyik táblázatból származik (saját táblázatunknál a '*SAJÁT*' szöveget). Az eredményt rendezzük tantárgyak, majd oktatók szerint, ezen belül előbb a *saját* soraink jöjjenek.

Utasítás:

```
select tmegnev, oktato, 'SAJÁT' as TULAJ from tantargy
UNION ALL
select tmegnev, oktato, 'SQL.30' as TULAJ from sql30.tantargy
order by 1,2,3;
```

2. Írassuk ki saját és *SQL30 TANTARGY* táblázatából azon tantárgyak neveit (*TMEGNEV*) és oktatóit, amelyek mindkét táblázatban szerepelnek. Az eredményt rendezzük tantárgyak, majd oktatók szerint.

Utasítás:

```
select tmegnev, oktato from tantargy
INTERSECT
select tmegnev, oktato from sql30.tantargy
order by 1,2;
```

5.3.4.5. Feltételek

A logikai operátorokkal előállított feltételek kiértékelése a matematikában szokásos módon történik. Különbséget jelent, ha valamelyik operandusban *NULL érték* szerepel. A *NULL értéket* tartalmazó kifejezések kiértékelését az 5.3.7. pontban tárgyaljuk, az így kiegészített igazság táblázatokat pedig az 5.5., 5.6., 5.7. táblázatokban foglaltuk össze. A speciális SQL operátorokkal képzett feltételek kiértékelését az ott megadott példákon nyomon követhetjük.

5.3.5. Határoló jelek

Az SQL fenntartott szavakat és azonosítókat egyaránt írhatjuk kis- és nagybetűkkel is. Minden egyes önálló elemet egybe kell írunk. A nyelv különböző elemeit viszont szóközzel választjuk el. A szóközők száma tetszőleges lehet. Literálon belül a kis- és nagybetűs ábrázolásmód és a szóközők száma is lényeges. Az azonosítók különböző szintjeit egy ponttal (.) választjuk el egymástól. Ha ugyanazon elemtípus ismétlődik, akkor ezeket a szintaktikában meghatározott jel, általában vessző (;), esetenként szóköz választja szét. Az utasítások végét pontosvessző (;), egyes programnyelvekben az *END-EXEC* kifejezés jelzi. Programban az utasítás kezdetét is jelezni kell. Erre szolgál az *EXEC SQL* jelsorozat.

5.3.6. Objektumok

Egy adatbázis fizikailag is és logikailag is különböző típusú hierarchikus elrendezésű objektumokból épül fel. Sajnos a terminológia csak a hierarchia alsóbb szintjein egységes. A felső szinteken például az IBM *Database* fogalma nagyjából az ORACLE *Schema (séma)* fogalmával azonos és a logikai és fizikai objektumok összekapcsolási módja is különbözik. Az átlagfelhasználó azonban szerencsére ebből nem sokat vesz észre. A legfontosabb objektumok hierarchiában a következők:

- Adatbázis (Database)
 - logikai felosztás:
 - séma (Schema)
 - táblázat (Table),
 - nézet (View),
 - tárolt nézet (Materialized View),
 - sorszám generátor (Sequence),
 - szinonima (Synonym),
 - index (Index),
 - trigger (Trigger)
 - fizikai felosztás:
 - adat fájlok
 - tablespace
 - adat blokk, lap (Page)
 - adatbiztonsági fájlok,
 - katalógus fájlok.

Ezeknek az objektumoknak a többségét részletesen ismertettük a relációs modellnél (ld. 4.3. alfejezet). Ezért itt csak néhány kiegészítést teszünk, ami ezen objektumoknak az SQL-hez való kapcsolódását teszi érthetőbbé.

5.3.6.1. Adatbázis (Database)

Az ORACLE és sok más rendszer terminológiájában ez az adatbázis-kezelő rendszer által egységesen kezelendő adatok összessége. A rendszert az adatbázis szerver gép működteti.

5.3.6.2. Séma (Schema)

A séma egy felhasználó objektumainak az összessége. A séma neve megegyezik a séma tulajdonosának az azonosítójával. Az objektumok azonosítójának első komponense mindig a séma-név. Ha nem adjuk meg explicit, hogy melyik sémában dolgozunk, akkor a rendszer feltételezi, hogy a saját sémánkban vagyunk és ennek megfelelően cselekszik. A saját sémánkban a saját objektumainknak mi vagyunk a tulajdonosa és ezekre minden jogosultságunk megvan. Mások objektumaival csak akkor dolgozhatunk, ha erre az objektum tulajdonosától engedélyt kaptunk (ld. 5.6.2.4. pont).

Az IBM terminológiában a sémának lényegében a *Database* felel meg, de ott az több felhasználó objektumait is tartalmazhatja és a neve sem kötött. Ebben minden jogosultsággal csak az adott adatbázis adatbázis-felügyelője rendelkezik. Az objektumok azonosítói a tulajdonos nevével egészülnek ki. Mindenki automatikusan a saját objektumaival dolgozik és csak ezekre van meg minden jogosultsága.

Sémát általában csak az adatbázis-felügyelő hozhat létre. A saját sémánk objektumait a *CREATE* ... utasításokkal hozhatjuk létre (ld. 5.14. táblázat.: pont), a *DROP* utasításokkal szüntethetjük meg (ld. 5.6.2.2. pont) és az *ALTER*-rel módosíthatjuk (ld. 5.6.2.3. pont).

TÁBLÁZAT (TABLE)

Az adatok tárolására szolgál. Megfelel egy fizikailag is létező relációnak (ld. 4.3. alfejezet). A logikailag összetartozó adatok alkotják a táblázat *sorait*. Ezek felelnek meg a reláció egyedeinek, előfordulásainak. Az egyedek tulajdonságait reprezentálják a táblázat *oszlopai*. A sorok egyértelmű azonosítására szolgál a táblázat *elsődleges kulcsa* (*Primary*

Key). Más táblázatokkal való egy – több (1:N) kapcsolatot az *idegen kulcsok* (*Foreign Key*) biztosítanak. A táblázatnak minden oszlopát egyenként definiálnunk kell (ld. 5.4. alfejezet).

Amikor a táblázatot létrehozuk, akkor az alapértelmezéssel vagy explicit megadjuk, hogy az melyik *Tablespace*-ben, melyik adatfájlban jöjjön létre, hol tároljuk fizikailag az adatait. Ugyancsak megadhatjuk azt is, hogy betöltéskor mennyi helyet hagyjunk szabadon a későbbi bővítések beszúrására, mi szerint sűrűsödjenek a rekordok.

NÉZET (VIEW)

A nézet egy vagy több táblázatból meghatározott szempontok alapján kiválasztott adatok reprezentálására szolgál (ld. 4.3. alfejezet). Megfelel egy virtuális relációnak, melynek adatait a táblázat(ok)ban ténylegesen tárolt aktuális adatokból a nézet definíciójának megfelelően választja ki a rendszer. Lekérdezésekben a nézeteket ugyanúgy használhatjuk, mint a táblázatokat. Adatok változtatásánál bizonyos korlátozások lehetnek. Vannak olyan nézetek, melyeken keresztül nem lehet adatokat változtatni. Ilyenek például azok, amelyek csoportképző függvényt vagy joint tartalmaznak. Nem írhatunk be olyan nézeten keresztül, amelyik nem tartalmazza bármely bázistáblázatuknak egy *NOT NULL*-ként definiált oszlopát.

Azokat a táblázatokat, amelyekből a nézet adatait kiválasztjuk nevezzük a nézet *alap* vagy *bázis táblázat(ai)nak*. A bázis táblázat adatainak megváltoztatása magával vonja a nézeten keresztül látott adatok változását is. Hasonlóan, ha nézeten keresztül változtatunk (ha lehet), akkor ez a bázis táblázat(ok) megváltoztatását jelenti.

TÁROLT NÉZET (MATERIALIZED VIEW)

A nézet adatait a rendszer minden egyes lekérdezésnél újra kiszámolja. Ez, különösen aggregátumok kiszámolásánál igen hosszadalmas lehet. Ha az adatok csak ritkán változnak, akkor az újraszámolás általában fölösleges is. Ilyenkor definiálhatunk egy tárolt nézetet, amelyik a nézetben definiált adatokat fizikailag is tartalmazza. Így lekérdezéskor ezeket nem kell újra kiszámolni. Amennyiben az alaptáblázatban bármi változás történik, a tárolt nézet adatai is újraszámítódnak.

A tárolt nézetek helyet foglalnak el, de ritkán változó adatoknál meggyorsítják a lekérdezést. Ezért gyakran alkalmazzák adattárházakban (ld. 9. fejezet).

SORSZÁM GENERÁTOR (SEQUENCE)

A sorszámgenerátor minden egyes meghíváskor egy, az addigiaktól különböző számot szolgáltat. Ezek lehetnek véletlen számok, vagy egy monoton számsorozat egymást követő elemei. Gyakran használják a rekordokba mesterségesen beillesztett elsődleges kulcs generálására. Különösen hasznos akkor, ha új adatok bevitele több, egymástól független helyről is történhet, mert így biztosítható, hogy duplikált kulcs hiba nem fog előfordulni.

TRIGGER

A trigger valójában nem objektum, hanem egy tárolt SQL utasítás-sorozat, amelyik egy táblázaton vagy nézeten végrehajtandó utasítás(ok)hoz kapcsolódik. Minden alkalommal mielőtt (*BEFORE* trigger) vagy miután (*AFTER* trigger) a triggert kiváltó utasítás végrehajtódik a táblázaton/nézeten, a triggerben definiált tárolt utasítás-sorozat is automatikusan végrehajtódik. Ezzel biztosíthatjuk az adatok módosítás utáni konzisztenciáját. A trigger működését bármelyik adatmódosító utasítás (*DELETE*, *INSERT*, *UPDATE*) kiválthatja

Triggereket használhatunk többek között

- ugyanazon vagy különböző táblázatok/nézetek adatai közti összefüggések automatikus biztosításához,
- auditáláshoz,
- a felhasználó figyelmeztetésére,
- táblázatnak/nézetnek vagy részeinek automatikus duplikálására,

Tipikus példa a trigger használatára, hogy ha egy hallgatót törölünk a *HALLGATO* táblázatból, akkor a triggerrel az adatait (vagy azok egy részét) automatikusan beírjuk az erre a célra létrehozott *VOLT_HALLGATO* táblázatba.

5.3.7. NULL érték

Ha egy oszlopnak nem adunk explicit vagy alapértelmezéssel értéket, akkor annak tartalma *NULL*, más néven a *NULL érték* lesz. Ez nem tévesztendő össze sem a *0* számmal, sem a szóközzel. Lényeges különbség van aközött, hogy valaki nulla fizetésért önkéntesként dolgozik (ez a *0 érték*), vagy nem tudom, mennyi a jövedelme (ez a *NULL érték*).

A *NULL érték* minden adattípusnál megengedett. Nem használható azonban olyan oszlopokban, amelyeket

- *NOT NULL*-ként definiáltunk,
- *elsődleges kulcs*-ként, vagy annak részeként definiáltunk (ld. 5.14. táblázat: pont).

Minden aritmetikai művelet és függvény, amelyiknek bármelyik komponense a *NULL érték* ugyancsak a *NULL értéket* adja eredményül. Így például ha az *OSZLOP1* oszlop aktuális értéke a *NULL érték*, akkor a *10+OSZLOP1*, *10*OSZLOP1* kifejezések vagy a *LOWER(OSZLOP1)* függvények értéke is a *NULL érték* lesz. Az argumentumra *NULL értékeket* tartalmazó sorokat a *COUNT(*)* kivételével egyik *csoport függvény* sem veszi figyelembe. Ezért előfordulhat, hogy ha *NUMERIKUS_OSZLOP* *NULL értéket* is tartalmazhat, akkor *AVG(NUMERIKUS_OSZLOP)* és *SUM(NUMERIKUS_OSZLOP)/COUNT(*)* nem egyenlő, mivel *COUNT(*)*-ba a *NULL értékű* sorok is beszámítódnak, a *SUM*-ba viszont nem. A *NULL értékkel* kapcsolatos műveleti szabályokat az operátoroknál, a logikai operátoroknál (ld. 5.3.4.3. pont) ismertettük.

Azért, hogy a *NULL értékkel* is tudjunk közvetlenül dolgozni, lehetőség van ennek meghatározott értékre való átkonvertálására. Ilyenkor, ha az eredmény *NULL érték* lenne, akkor helyette a megadott érték jelenik meg az eredményben. Ennek formája rendszerfüggő. Egyik lehetőség az *NVL(kifejezés-1, kifejezés-2)* (más rendszerben a *VALUE* vagy *COALESCE*) függvény alkalmazása, amelynek értéke *kifejezés-1*, ha az nem *NULL érték* és *kifejezés-2* különben (ld. 5.3.8. pontban a 4. példa).

Példák:

1. Meg akarjuk tudni a *HALLGATO* táblázatunkból, hogy kiknek nem ismerjük a születési időpontját (*SZDATUM*).

Utasítás:

```
select vnev,knev,hid from hallgato
where szdatum is null order by vnev,knev;
```

Eredmény: ABC sorrendben kilistázza azoknak a nevét és azonosítóját, akiknek nem ismerjük a születési időpontját.

2. Meg akarjuk tudni, kik azok a hallgatók, akik 1980 előtt születtek.

Utasítás:

```
select vnev,knev,hid,szdatum from hallgato
where szdatum < '1980-01-01' order by vnev,knev;
```

Eredmény: Megkapjuk az összes olyan hallgatót, aki 1980. január 1. előtt született.

Ez nem biztos, hogy elegendő számunkra, hiszen azok között is lehetnek még 1980 előtt születettek, akiknek születési idejét nem ismerjük. Hogy ezekről is legyen valami információnk, kiegészítjük az utasítást:

Utasítás:

```
select szdatum,vnev,knev,hid from hallgato where szdatum < '1980-01-01'
or szdatum is null order by szdatum, vnev,knev;
```

Eredmény: Először születési sorrendben kilistázza azoknak a nevét és azonosítóját, akik 1980. január 1. előtt születettek, majd ABC sorrendben azoknak a nevét és azonosítóját, akiknek nem ismerjük a születési időpontját.

5.3.8. SQL kifejezések

Egy SQL kifejezés egy vagy több oszlop, állandó, SQL kifejezés, SQL függvény és operátor kombinációja, ami eredményül egy értéket szolgáltat. A kifejezések bármilyen adattípust reprezentálhatnak. Általában bármely adattípus helyett használhatók SQL utasításokban, függvényekben és feltételekben.

A legegyszerűbb kifejezés az egy oszlopból, egy állandóból, egy SQL függvényből, vagy egy speciális SQL értékből álló egy tagú kifejezés. (ld. 1. példa). Összetett kifejezéseket képezhetünk egyszerű vagy összetett kifejezésekből aritmetikai operátorokkal (ld. 2. példa) vagy, amikor a függvény argumentuma egy függvény vagy kifejezés (ld. 3. példa). Egy kifejezésben általában csak azonos adattípusok szerepelhetnek. Ha különböző adattípusokat tartalmaz, azokat azonos típusra kell konvertálnunk. (Néha a rendszer ezt automatikusan is elvégzi, de ebben ne bízunk!)

A kifejezés adattípusa rendszerint megegyezik a kifejezés összetevőinek adattípusával. Néhány speciális kivételtől eltekintve (pl. *CONCAT*, *NULL érték* átalakítása), ha egy kifejezés bármely tagja a *NULL érték* akkor a kifejezés értéke is a *NULL érték* lesz.

Példák:

1. Egy elemből álló kifejezések:

vnev	hallgato.vnev	sql30.hallgato.vnev
'NINCS ADAT'	12345	'12345'
upper(vnev)	count(*)	'2004-03-30'
sysdate vagy current date		NULL

2. Összetett kifejezések:

gyakpont + 10	mithallgat.gyakpont - 10	sql30.mithallgat.gyakpont * 10
(30 * osztalyzat + gyakpont)/100		current date + 10
(avg(osztalyzat) + avg(sql30.osztalyzat))/2		SYSDATE + 10

3. Összetett függvény kifejezések:

```
upper(initcap(vnev))  
concat(concat(concat(concat('ir_szam',' '), 'varos'),' '), 'lakcim')  
round(avg(osztalyzat),1)
```

4. Műveletek *NULL* értékek esetén.

A *FIZETES* és a *JUTALOM* oszlopok egyaránt tartalmazhatnak *NULL* értéket is. Ekkor az $AVG(FIZETES + JUTALOM)$ és az $AVG(FIZETES) + AVG(JUTALOM)$ értéke eltérhet, mert az elsőbe nem számítanak be azok a sorok, amelyekben *akár FIZETES, akár JUTALOM NULL érték* volt. Ha elfogadjuk, hogy ebben az esetben a *NULL* értéket 0-nak tekintjük, akkor $AVG(NVL(FIZETES,0) + NVL(JUTALOM,0))$ a helyes.

5.4. Adattípusok

Az adatbázis táblázataiban minden oszlopnak és állandónak (literál) megfelel egy adattípus. Ezt a táblázat létrehozásakor a *CREATE TABLE* utasításban (ld. 5.14. táblázat: pont) kell definiálnunk. Minden adattípusnak meghatározott tulajdonságai vannak és csak az adattípusnak megfelelő műveleteket lehet elvégezni velük. Így például karakter típusú adatokkal nem lehet aritmetikai műveleteket végezni, két dátum típusú adat különbségét képezhetjük, de összegét nem. Numerikus vagy dátum típusú adatnak nem adhatjuk az 'egy' értéket, míg a '2004-03-30' érték mind karakter mind dátum adattípusba bevihető. (Utóbbiba csak akkor, ha ez megfelel az aktuálisan érvényes dátum formátumnak.)

A leggyakoribb adattípusok a következők:

- karakter,
- numerikus,
- dátum/időpont,
- sorazonosító (ROWID),
- hosszú adatok.

A fentieken kívül léteznek más adattípusok és magunk is definiálhatunk különféle adattípusokat. Mivel azonban ezeket csak igen ritkán és speciális célokra használják, velük a továbbiakban nem foglalkozunk.

Az adatbázisokkal elsősorban elméletben, vagy csak az oktatásával foglalkozó szakemberek kedvéért azonban megemlítjük még a logikai (Boolean) adattípust is. Ez egy logikai változónak felel meg, amelyik csak az IGAZ (TRUE), vagy a HAMIS (FALSE) értékeket veheti fel. A gyakorlatban működő nagy adatbázisokban azonban ezeket szinte sehol sem használják.¹

Az adatbázis-kezelő rendszerek általában nem engedik meg, hogy egy műveletben különböző adattípusok szerepeljenek. Ezért általában egy kifejezés minden tagjának, egy feltétel operandusainak, az összetartozó input és output adatelemeknek a típusa azonos kell legyen. Ha ez nem áll fenn, akkor egy konvertáló függvény segítségével azonos adattípusra kell átalakítani őket. Bizonyos esetekben az adatbázis-kezelő rendszer utasításértelmezője automatikusan átkonvertál a különböző adattípusok között. Így például különböző karakter típusok vagy eltérő numerikus ábrázolásmódok között soha nem szükséges konvertálni. Ennek ellenére javasoljuk, hogy ha pl. egy utasítás vagy függvény eredeti inputja nem a kívánt formában van, akkor azt előbb alakítsuk át a megfelelő adattípusra és formátumra.

¹ A szerző 30 éves adatbázisokkal kapcsolatos tevékenysége során csak egyszer került olyan helyzetbe, hogy a tiszta elvi koncepciót kedvelő elméleti szakértők kívánságára egy személyi nyilvántartásban a csupán két értéket (férfi, nő) felvehető *NEM* adattípust logikainak definiálja. De végül ez sem valósult meg, mert a hölgy munkatársak és felhasználók kifogásolták, hogy a nőkre úgy kérdezzük rá: *HAMIS?*

5.4.1. Karakter adattípus

Karakter típusú adatok befogadására szolgáló adattípus. Az így definiált oszlopban bármilyen karakter tárolható. Azt, hogy egy adott bájtnak milyen karakter felel meg, az adatbázis generálásakor meghatározott kódkészlet határozza meg. Ez utána már nem változtatható meg. A legtöbb adatbázis-kezelő rendszer alapértelmezésben az ASCII kóddal dolgozik, az IBM UDB viszont az EBCDIC-vel. A karakter típusú adatok definiálási formáit az 5.9. táblázatban foglaltuk össze.

5.9. táblázat: **Karakter adattípusok definiálása.**

Adattípus	Definíció
CHAR(<i>n</i>)	Pontosan <i>n</i> bájtból álló karaktersorozat. $1 \leq n \leq 255$.
CHAR	Pontosan egy bájt. Bármilyen karaktert tartalmazhat.
VARCHAR(<i>n</i>) VARCHAR2(<i>n</i>)	Legfeljebb <i>n</i> bájtból álló karaktersorozat. $1 \leq n$ értékét mindig meg kell adnunk. Maximális értéke rendszertől függ, az ORACLE-ben 2000, IBM UDB-ben 32706.

Megjegyzések:

- Ha adatbevitelkor a tárolandó karaktersorozat hosszabb az oszlop definiált hosszánál, (karakterek-száma $> n$), akkor hibajelzést kapunk és az érték nem kerül be az oszlopba.
- *CHAR*(*n*) adattípusnál, ha a karakterek száma kisebb az oszlop hosszánál, akkor a jelsorozat a végén szóközökkel kiegészítve, a rögzített hosszúságban kerül be az oszlopba. A különböző összehasonlítások is az így kiegészített tartalom alapján történnek meg.
- *VARCHAR* (ORACLE-ben *VARCHAR2*) adattípusnál, ha a karakterek száma kisebb az oszlop maximális hosszánál, a jelsorozat tényleges hossza lesz az oszlop tárolt hossza. Az ilyen formában tárolt adatok rendszerint kevesebb helyet foglalnak el, mintha ugyanazokat a karaktereket rögzített hosszúságúnak definiált *CHAR*(*n*)-ként tárolnánk. Figyelembe kell vennünk azonban, hogy az oszlop aktuális hosszát is tárolnunk kell (ez 2 bájt) és minden hozzáférésnél mindig dinamikusan meg kell határoznunk, ami időt jelent. Ezért a gyakorlatban csak akkor érdemes változó hosszúsággal dolgoznunk, ha az átlagos helymegtakarítás rekordonként legalább 10-15 bájt. A *VARCHAR* típusú adatok tényleges hosszát a *LENGTH* függvénnyel (ld. 5.5.2.1. pont) kérdezhetjük le.
- Karakter típusú és numerikus illetve dátum típusú adatok között a különböző konvertáló függvényekkel (ld. 5.5.2.4. pont) hajthatunk végre konverziót.

Példa:

1. *FIX_CHAR* definíciója CHAR(10), *VALTOZO_CHAR*-é VARCHAR(10) illetve VARCHAR2(10).

A beviendő érték	<i>FIX_CHAR</i> értéke	<i>VALTOZO_CHAR</i> értéke
'A1234'	'A1234 '	'A1234'
'x <> -12'	'x <> -12 '	'x <> -12'
'12345678910'	HIBA! Az adat túl hosszú, nem kerül be az oszlopba.	HIBA! Az adat túl hosszú, nem kerül be az oszlopba.

5.4.2. Numerikus adattípus

Numerikus adatok befogadására szolgáló adattípus. Rendszertől függ, hogy miként kell definiálni a különböző számbábrázolásban tárolandó számokat, és hogy milyen számformátumok megengedettek. A leggyakoribb definiálási formákat az 5.10. táblázatban foglaltuk össze. (Nem mindegyik rendszerben érvényes mindegyik forma.)

5.10. táblázat: Numerikus adattípusok definiálása

Adattípus	Ábrázolt szám
NUMBER(p,s) vagy DECIMAL(p,s) vagy DEC(p,s)	Összesen $1 \leq p \leq 18$ (egyes rendszereknél 38) jegyből álló előjeles szám, melyből a s a tizedesjeltől jobbra álló jegyek száma.
NUMBER(p) vagy DECIMAL(p) DEC(p)	Összesen $1 \leq p \leq 18$ (egyes rendszereknél 38) jegyből álló előjeles tömörített formában tárolt egész szám.
NUMBER	Egy jegyű tömörített formában tárolt előjeles egész szám.
REAL vagy FLOAT	4 bájtban ábrázolt előjeles lebegőpontos szám.
FLOAT DOUBLE PRECISION	8 bájtban ábrázolt előjeles lebegőpontos szám.
INTEGER vagy INT	4 bájtban tárolt előjeles egész szám.
SMALLINT	2 bájtban tárolt előjeles egész szám.

Megjegyzések:

- Ha a beviendő jelsorozat nem ad a definiált formátumnak megfelelő érvényes számot, akkor hibajelzést kapunk.
- Ha a tárolandó adat értékes jegyei nem férnek be a definiált oszlopba (pl. *NUMBER/DECIMAL*-nál az *értékes jegyek száma* $> p - s$, *SMALLINT* abszolút értéke > 32767), akkor hibajelzést kapunk és az érték nem kerül be az oszlopba.
- Ha a tizedes jel utáni jegyek nem férnek be az oszlopba, akkor az érték kerekítve kerül be az oszlopba.
- Interaktív SQL-ben a numerikus adatoknak a képernyőn való megjelenítési formáját szabályozhatjuk. Ennek módja azonban rendszerfüggő.
- Különbözőképpen definiált numerikus adatokat összehasonlításokban, kifejezésekben a rendszer automatikusan egységes alakra alakít át. Karakter típusú és numerikus adatokat a megfelelő konvertáló függvényekkel (ld. 5.5.2.4. pont) konvertálhatjuk egymásba.

Példa:

1. SZAM definíciója NUMBER(4,2) illetve DEC(4,2)

A beviendő érték	A tárolt érték
1.23	01.23
31.2	31.20
1.234	01.23
1.235	01.24
9	09.00
0.001	00.00
-12.34	-12.34
1,2,3	HIBA! Két tizedes jel
123	HIBA! Túl nagy érték!

5.4.3. Dátum/időpont adattípus (DATE/TIME)

Dátumot és időt tartalmazó adatok befogadására szolgáló adattípus, mely a megfelelő dátum/idő értéket egy bináris számként tárolja, de a felhasználó számára az általa definiált formában jeleníti meg. Bár ilyen adatokat karakter és numerikus formában is tárolhatunk, célszerű ezeket mégis mindig dátum/időpont típusú változóban tárolni, mert ezáltal

- automatikusan ellenőrizni tudjuk, hogy érvényes dátumot és/vagy időpontot adtunk meg. (Pl. a *2003-02-29* dátumra hibajelzést kapunk, míg a *2004-02-29* dátumot a rendszer elfogadja.)
- Könnyen tudjuk a dátumokat különböző formában megjeleníteni. (Pl. a magyar *2007-03-30* vagy *2007.03.30* forma helyett a *30-MARCH-2007* angolt vagy *3/30/2007* amerikai.)
- Könnyen tudunk velük műveleteket végezni. (Pl. hány nap van az adott hónap utolsó napjáig, a 2007.02.27 utáni 3. nap 2007.03.02, de a 2008.02.27 utáni 2008.03.01., hány nap telt el két dátum között, stb.)

A dátum/időpont adattípusok definiálása is rendszerfüggő. Van, ahol a *DATE*-ként definiált adattípus a naptári dátum mellett a napi időt is tartalmazza másodperc pontosságra, más rendszerekben a *DATE* adattípus csak a dátumot, a *TIME* csak a napon belüli időpontot tartalmazza és a kettőt együtt mikroszekundum pontossággal a *TIMESTAMP* adattípus tartalmazza (IBM UDB/DB2).

Az alapértelmezett dátumformátum rendszer függő, amit természetesen igényeink szerint módosíthatunk. Ennek módja adatbázis-kezelő rendszerenként eltérő. Az ORACLE-ben például az *NLS_DATE_FORMAT* inicializálási paraméter határozza meg. Ha nem adunk meg konvertáló függvényt, akkor minden dátum/idő típusú érték az alapértelmezett formában jelenik meg a képernyőn és állandóként is így kell megadnunk. Karakter és dátum/időpont típusú adatokat a megfelelő konvertáló függvényekkel (ld. 5.5.2.4. pont) konvertálhatjuk egymásba. Az SQL nyelvben leggyakrabban használt konvertáló és dátum függvényeket az 5.5.2.3. és az 5.5.2.4. pontokban ismertetjük.

A dátum aritmetika szabályai szerint az alábbi műveleteket végezhetjük el dátum/időpont típusú változókkal:

- Hozzáadhatunk vagy kivonhatunk napokat egy adott dátumból. Az előjeles egészszám a dátumhoz hozzáadandó/kivonandó napok számát adja meg. (ld. 1. példa).

- Hozzáadhatunk vagy kivonhatunk órákat, percek egy adott dátumból. Az órákat (*óra/24*), a percek (*perc/1440*) formában kell megadnunk.
- Két dátum típusú adat különbsége a kettő között eltelt napok számát adja meg. (ld. 2. példa).
- Ügyeljünk arra, hogy dátum/időpont értékek NEM lehetnek szorzók, osztók, osztandók és két ilyen típusú értéket NEM adhatunk össze.

Példák:

1. A *DATUM* oszlop *DATE* típusként van definiálva.

Utasítás:

```
select datum+1 as holnap, datum-7 egy_hete from ...
```

Eredmény: a *holnap*ként kiválasztott oszlop értéke a *datum* utáni nap lesz, *egy_hete* pedig az egy héttel előtti nap.

2. Ha az *ESEDEKES* oszlop aktuális értéke '2007-05-31', akkor *ESEDEKES* - '2007-04-30' értéke 31 lesz.

5.4.4. Sorazonosító (ROWID)

Minden sornak van egy belső címe a táblázatban. Ezt tartalmazza hexadecimális formában a minden rekordnak részét képező *ROWID* pszeudooszlop. Ez a teljes rekord kiolvasásakor (*SELECT * FROM ...*) nem jelenik meg, csak ha külön rákérdezzünk. Ha ismerjük a *ROWID* értékét, akkor a rekordhoz a leggyorsabban ennek alapján férhetünk hozzá. Ennek ellenére, ezt ne használjuk elsődleges kulesként, mert ha az adatokat újraszervezzük, ez a cím megváltozhat.

A *ROWID* értékét az adatbázis-kezelő rendszer határozza meg, ezért adat beírásakor nem adhatjuk meg az értékét és nem is módosíthatjuk.

5.4.5. Hosszú adatok

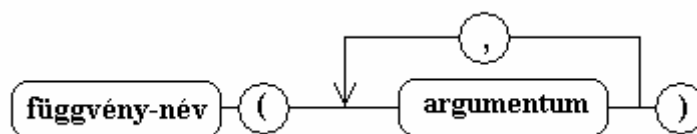
Az igen hosszú adatok befogadására különböző rendszerspecifikus lehetőségek vannak. Az egyik a *LONG* adattípus deklarálása (pl. ORACLE). Ez maximálisan 2 Gb-ot hosszúságú karaktersorozatok befogadására szolgál. A tárolt hossz változó, a tényleges adattartalomtól függ. Aktuális értékét a *LENGTH* függvénnyel (ld. 5.5.2.1. pont) kérdezhetjük le. A hosszú adattípusok használata hasonló a karakter adattípushoz, de bizonyos korlátozások vannak. Ezek közül a legfontosabbak:

- Egy táblázat csak egy *LONG* oszlopot tartalmazhat.
- *LONG* oszlopra nem építhető index.
- Csak NULL vagy NOT NULL típusú integritási feltétel adható meg rá.
- *LONG* oszlop nem szerepelhet *WHERE*, *GROUP BY*, *ORDER BY* klauzulában, SQL függvényben, kifejezésben, feltételben.

Sok rendszerben lehetőség van sok adatot tartalmazó objektumoknak, például MS-Word dokumentumoknak, képeknek a táblázathoz való csatolására. Ezt például az IBM UDB/DB2 nagy objektumok (*LOB*) tárolására szolgáló segédtáblázatok definiálásával oldja meg. ACCESS-ben történő realizálását a 6.1.2.1. pontban ismertetjük.

5.5. A legfontosabb SQL függvények

Igen kényelmessé teszik az SQL használatát a rendszerbe beépített SQL függvények. Az SQL függvények általános formája:



A függvény mindig egy eredményt szolgáltat, melynek értéke függ a függvény fajtájától és az argumentum(ok) aktuális értékétől. Ha egy függvényben *argumentum* értéke a *NULL érték*, akkor – néhány kivételtől eltekintve (pl. *CONCAT*, *NVL*) – az eredménye is a *NULL érték* lesz.

A függvények argumentuma(i) lehetnek:

- oszlop értékek,
- állandók (literálok),
- SQL függvények,
- SQL kifejezések.

Az SQL függvényeket a *SELECT* utasítás kiválasztási listájában, vagy a *WHERE* vagy *HAVING* klauzulájában, az *INSERT* utasítás *VALUE*, a *DELETE* utasítás *WHERE* és az *UPDATE* utasítás *SET* és *WHERE* klauzulájában használjuk (ld. 5.6. pont). Ha a függvény a *SELECT* utasítás kiválasztási listájában szerepel, akkor célszerű lehet a fejlécben az adott oszlopnak nevet adni az *AS fejléc_név* kiegészítéssel.

Attól függően, hogy a függvény az adatok egy csoportjára jellemző eredményt ad, vagy egy adott oszlop (illetve ami ezzel egyenértékű, egy argumentum) aktuális értékéből állítja elő a függvényértéket beszélünk

- *csoport* függvényekről,
- *oszlop* vagy más néven *skalár* függvényekről.

5.5.1. Csoport függvények

A csoport függvények aggregátum képzésre szolgálnak. Nem az egyes sorokra, hanem a kiválasztott sorok összességére, vagy azok meghatározott csoportjaira szolgáltatnak egy, az összes sorra, vagy a csoportok minden sorára jellemző értéket. Az SQL nyelvben leggyakrabban használt csoport függvényeket az 5.11. táblázat tartalmazza.

5.11. táblázat: Legfontosabb csoport függvények

Függvény	Funkció
AVG (numerikus argumentum)	A kiválasztott sorokból az argumentum értékek átlaga.
COUNT(*) vagy COUNT(argumentum)	A kiválasztott sorok száma. <i>COUNT(argumentum)</i> csak azon sorok száma, melyekben az argumentum értéke nem <i>NULL érték</i> .
MAX(argumentum)	A kiválasztott sorokból az argumentum legnagyobb értékének meghatározása.
MIN (argumentum)	A kiválasztott sorokból az argumentum legkisebb értékének meghatározása.
SUM (numerikus argumentum)	A kiválasztott sorokban az argumentum összegének kiszámítása.

Megjegyzések:

- Az átlag és az összegfüggvény argumentuma csak numerikusan definiált oszlop, numerikus állandó, numerikus kifejezés vagy numerikus eredményt szolgáltató SQL függvény lehet.
- A maximum és a minimum meghatározásánál az SQL rendszer rendezési sorrendje (általában ASCII vagy EBCDIC) érvényes.
- Az argumentumra *NULL* értékeket tartalmazó sorokat a *COUNT(*)* kivételével egyik függvény sem veszi figyelembe. Ezért vigyáznunk kell, mert ha az argumentum *NULL* érték is lehet, akkor *AVG(argumentum)* és *SUM(argumentum)/COUNT(*)* eltérhetnek egymástól. A *NULL* értéket tartalmazó sorokat az *NVL* függvény segítségével vehetjük számításba.
- Az *AVG*, *SUM* és a *COUNT* függvényekben az argumentum előtt megadhatjuk a *DISTINCT* kulcsszót is. Ekkor az aggregátum képzésben az SQL csak a különböző argumentum értékeket veszi figyelembe. Ilyenkor a *COUNT* függvényben is az oszlop nevét kell megadnunk. (ld. 2. példa).
- Ha a kiválasztott csoportban a *WHERE* vagy *HAVING* feltételnek egy rekord sem tesz eleget, akkor a függvény értéke a *NULL* érték lesz, kivéve a *COUNT* függvényt, amely a 0 numerikus értéket veszi fel.

Példák:

1. Számoljuk ki minden tantárgyra (*TID*) a tantárgy osztályzatainak átlagát, és a tantárgyból elért legnagyobb pontszámot.

Utasítás:

```
select tid,avg(osztalyzat) as atlag_osztalyzat,max(gyakpont) as  
max_gyakpont from mithallgat group by tid;
```

Eredmény: Azok osztályzata, akik még nem vizsgáztak (*NULL* érték) nem számítódik be az átlagba.

2. Adjuk meg az összes levizsgázott hallgató számát. Ha valaki több tárgyból vizsgázott, akkor is csak egyszer vegyük figyelembe.

Utasítás:

```
select count(distinct hid) as vizsgazok_szama from mithallgat  
where osztalyzat is not null;
```

Magyarázat: Mivel nem egyes csoportokra, hanem az egész táblázatra kell a rekordok számát meghatároznunk, a *GROUP BY* klauzulát nem adjuk meg. A *DISTINCT* megadásával biztosítjuk, hogy ha egy hallgatónak több vizsgája is van, akkor is csak egyszer szerepeljen az eredményben.

5.5.2. Oszlop függvények

Az oszlop függvények elnevezése és formája korántsem olyan egységes, mint a csoport függvényeké. Így például az aktuális dátumot és időt szolgáltató függvény neve az ORACLE-ben *SYSDATE*, az ACCESS-ben *DATE()* illetve *TIME()*, míg az UDB/DB2-ben *CURRENT DATE*, *CURRENT TIME*, *CURRENT TIMESTAMP*. Ezért a továbbiakban a

függvényeknek általában csupán a funkcióira utalunk. Csak azoknak a függvényeknek adjuk meg a leggyakoribb formáját, melyek (szinte) minden rendszerben azonosak.

Az oszlop függvények az argumentum(ok) értékét alakítják át más formába. A függvénytől függ, hogy az argumentum milyen adattípus lehet, és hogy az eredmény milyen adattípus lesz.

Amennyiben az argumentumon nem megengedett műveletet akarunk végrehajtani, az utasítás már az értelmezéskor vagy a végrehajtáskor hibajelzéssel leáll.

Az argumentum típusától függően vannak

- karakter függvények,
- numerikus függvények,
- dátum/időpont függvények,
- konvertáló függvények,
- speciális függvények.

5.5.2.1. Karakter függvények

Az SQL karakter függvényeknek karakter értéket szolgáltató argumentuma(i) van(nak), melyeket kiegészíthet(nek) a funkciót részletező numerikus argumentum(ok). A függvények eredményként egy karakter típusú értéket adnak vissza. (Kivétel a *LENGTH* függvény, amelyik egy numerikus értéket szolgáltat). Az 5.12. táblázat az SQL nyelvben leggyakrabban használt karakter függvényeket és fő formájukat tartalmazza.

5.12. táblázat: SQL karakter függvények

Függvény	Funkció
CONCAT(<i>char-1</i> , <i>char-2</i>)	Konkatenálja a két argumentumot (ld. 1. példa).
INITCAP(<i>char</i>)	Átalakítás nagy kezdőbetűkre.
LENGTH(<i>char</i>)	Visszaadja az argumentum aktuális hosszát.
LOWER(<i>char</i>) UPPER(<i>char</i>)	Az argumentumban minden alfanumerikus karaktert kis- illetve nagybetűsre alakít át (ld. 2. példa).
LTRIM(<i>char</i>) RTRIM(<i>char</i>)	Levágja a kezdő ill. a befejező szóközöket.
REPLACE(<i>char</i> , <i>c1</i> , <i>c2</i>) TRANSLATE(<i>char</i> , <i>c1</i> , <i>c2</i>)	A <i>char</i> argumentumban minden <i>c1</i> karaktert kicserél <i>c2</i> -re.
SUBSTR(<i>char</i> , <i>n</i> , <i>m</i>)	Visszaadja az argumentumnak az <i>n</i> -edik bájtól kezdődő <i>m</i> bájt hosszúságú részét. Ha <i>m</i> -et nem adjuk meg, akkor az <i>n</i> -edik bájtól az argumentum végéig lévő karaktereket adja vissza.

5.5.2.2. Numerikus függvények

Az SQL numerikus függvényeknek numerikus értéket szolgáltató argumentuma(i) van(nak), és eredményként egy numerikus értéket adnak vissza. Amennyiben az operandus nem numerikus típusú és erre lehetőség van, akkor azt az SQL értelmező átkonvertálja numerikus formába.

A leggyakrabban használt numerikus függvények:

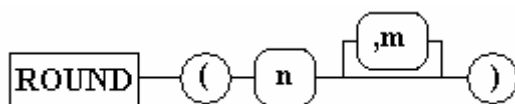
- ABS - Abszolút érték kiszámolása.
- FLOOR - A legnagyobb, de az argumentumnál még nem nagyobb egész meghatározása.
- MOD - Egész osztás maradékának kiszámolása.
- POWER – Hatványozás.

- ROUND – Kerekítés.
- SIGN - Előjel meghatározása.
- SQRT - Négyzetgyökvonás.
- TRUNC – Levágás.

A fentiekén kívül használhatjuk a szokásos jelölési móddal a szögfüggvényeket (argument radiánban), a hiperbolikus szögfüggvényeket, a logaritmus és az exponenciális függvényt is. (A felsorolás nem teljes.)

A numerikus függvények közül fontossága miatt bemutatjuk a ROUND függvény használatát.

Szintaktikája:



A ROUND SQL függvény a numerikus eredményt adó n argumentumot m tizedes jegyre kerekítve adja vissza.

Megjegyzések:

- Az n argumentum lehet NUMBER adattípusú oszlop, numerikus állandó (literál), numerikus SQL függvény, vagy numerikus kifejezés.
- n értéke tetszőleges szám, m értéke viszont csak egész szám lehet.
- Ha $m = 0$, vagy elhagyjuk, akkor n -t egészre kerekíti.
- Ha $m < 0$, akkor 10 -nek a $(-m)$ -ik hatványára kerekít.
- Az 5 értéket fölfelé kerekíti.

5.5.2.3. Dátum/időpont függvények

Az SQL dátum/időpont függvények dátum/időpont típusú érték(ek)ből állítanak elő meghatározott algoritmus alapján általában egy dátum/időpont vagy más típusú értéket. Formájuk erősen rendszerfüggő, de szinte mindegyik rendszerben megtalálható a *YEAR(dátum)*, *MONTH(dátum)*, *DAY(dátum)* függvény, ami a *dátum* típusú változóból visszaadja annak az év, hónap, nap részét. (ld. 4. példa). Ezekkel a kifejezésekkel argumentum nélkül számolhatunk is. Például a *DATUM + 2 MONTH* kifejezés a *DÁTUM* 2 hónappal megnövelt értékét adja. (*DATUM*='2006-12-31' esetén '2007-02-28'-at).

5.5.2.4. Konvertáló függvények

Az SQL konvertáló függvények egy adott adattípusú értékből állítanak elő egy másik adattípusnak megfelelő értéket. Az ORACLE-ben ezek a *TO_DATE*, *TO_NUMBER*, *TO_CHAR* függvények, amelyek dátum vagy. numerikus típusú adatot konvertálnak át karakterre, illetve karakter típusú adatot konvertálnak át dátumra illetve számra. A DB2/UDB-ben ezt a feladatot a *DATE* illetve a *DIGITS* függvény végzi el.

A különböző adattípusok tartalmát általában csak úgy tudjuk összehasonlítani, hogy azonos típusra konvertáljuk őket. Ugyanakkor azonos típusú, de különböző formátumú adatokat (pl. fix és változó hosszúságú karakterek vagy decimális, lebegő pontos és egész számok) korlátozás nélkül is összehasonlíthatunk.

5.5.2.5. Egyéb függvények

Az SQL nyelvben leggyakrabban használt egyéb függvények:

- *NVL (argumentum-1,argumentum-2)*: ha *argumentum-1* aktuális értéke *NULL* érték, akkor a függvény az *argumentum-2* értéket szolgáltatja (ld. 3. példa). (UDB/DB2-ben a függvény neve *VALUE*, ACCESS-ben nem függvényként, hanem formátumként lehet beállítani, ld. 6.1.2.1. pont)
- *USER*: értéke az aktuális felhasználó azonosítója lesz.

A leggyakoribb numerikus, dátum, konvertáló és egyéb függvényeknek az Oracle-ben illetve az IBM UDB/DB2-ben használt elnevezését az 5.13. táblázatban foglaltuk össze. Az SQL függvényeknek az ACCESS-ben használható formáját a 6.1.2.3. pontban ismertetjük részletesen.

5.13. táblázat: A leggyakoribb numerikus, dátum, konvertáló és egyéb függvények, és az Oracle-ben, illetve az IBM UDB/DB2-ben használt elnevezésük.

Függvény név	Funkció
<i>Dátum és numerikus függvények</i>	
ABS	Abszolút érték kiszámolása.
FLOOR	A legnagyobb, de az argumentumnál még nem nagyobb egész.
MOD	Egészosztás maradéka.
POWER	Hatványozás.
SIGN	Előjel meghatározása.
SQRT	Négyzetgyökvonás.
ADD_MONTHS	Hónap hozzáadása.
LAST_DAY	Hónap utolsó napjának megadása.
ROUND	Dátum vagy szám kerekítése.
SYSDATE vagy CURRENT DATE, CURRENT TIMESTAMP	A pillanatnyi dátum és pontos idő meghatározása.
TRUNC	Dátum vagy szám lerövidítése.
YEAR, MONTH, DAY	Dátum megfelelő részének a kivágása.
<i>Konvertáló függvények</i>	
TO_CHAR vagy VARCHAR	Dátum vagy numerikus adattípus átkonvertálása karakterre.
TO_DATE vagy DATE	Karakter adattípus konvertálása dátumra.
TO_NUMBER vagy DECIMAL	Karakter adattípus konvertálása numerikusra.
<i>Egyéb függvények:</i>	
NVL vagy VALUE, COALESCE	NULL érték átalakítása.
USER	Aktuális felhasználó meghatározása.

Példák:

1. *IR_SZAM*, *VAROS* és *LAKCIM* változó hosszúságú karakter típusú változók aktuális értékei: *1098*, *Budapest*, *Fővám tér 8*. A három oszlopból egy oszlopot akarunk összeállítani *TELJES_CIM* néven.

Utasítás:

```
select concat(concat(concat(concat(ir_szam,' '),varos),' '),lakcim)
as TELJES_CIM from ...
```

Eredmény:

```
TELJES_CIM
-----
1098 Budapest Fővám tér 8
```

Megjegyzés: Az egyes mezők értékeit szóközzel akarjuk elválasztani. Ezért közéjük egy szóközt is konkatenálni kell.

2. A tantárgynevet tartalmazó *TMEGNEV* oszlop aktuális értéke: *Oracle-1*, de nem tudjuk, hogy ez így, *ORACLE-1* vagy *oracle-1* formában van tárolva. Egyértelművé tehetjük az összehasonlítást az *UPPER* (vagy a *LOWER*) függvénnyel.

Utasítás:

```
select tid,upper(tmegnev) as tantargy_nev,oktato from tantargy
where upper(tmegnev) = 'ORACLE-1'
```

Eredmény: A *WHERE* feltétel függetlenül attól, hogy melyik formában tároltuk a tantárgy megnevezését *IGAZ* lesz. Ha nem használnánk az *UPPER* függvényt, akkor csak abban az esetben lenne igaz, ha az adatbázisban is pontosan így, csupa nagybetűvel tároltuk volna.

3. Ha az *OKTATO* még nem ismert, akkor a lekérdezésben a helyén a '*MÉG NINCS KIJELÖLVE*' szöveget akarjuk az eredményben megjeleníteni.

Utasítás:

```
select tid,tmegnev,nvl(oktato,'MÉG NINCS KIJELÖLVE') from tantargy;
```

Eredmény: Az összes olyan sorban, ahol *OKTATO NULL érték*, a megadott szöveg fog megjelenni, a többiben *OKTATO* aktuális értéke lesz.

4. Listát készítünk az 1988-ban született hallgatókról.

Utasítás:

```
select vnev, knev from hallgato where year(szdatum) = 1988 order by vnev, knev;
```

Eredmény: Kiválasztja az 1988-ban születettek nevét és ABC sorrendben kiírja.

5. *OSZLOP_1* aktuális értéke *1234.56789*.

Utasítás:

```
select round(oszlop_1,3) as oszlop_1
, round(oszlop_1, -2) as oszlop_1_szazasok
from ...
```

Eredmény: *OSZLOP_1* értéke *1234.568*, *OSZLOP_1_SZAZASOK*-é pedig *1200* lesz.

5.6. SQL utasítások

5.6.1. SQL utasítások összefoglalása

A relációs adatbázisokban minden műveletet SQL utasítások segítségével végzünk el. Ebben meghatározzuk az elvégzendő feladatot. Hogy ez miként hajtódik végre, azt általában az SQL optimalizáló határozza meg. Az SQL utasításokat kiadhatjuk interaktívan vagy magas szintű nyelven (általában COBOL, C++, Java) megírt programból az alkalmazási programba beépített utasításként. A legtöbb SQL utasítás interaktív és beépíthető formája megegyezik. Az interaktív módban megadandó illetve eredményként megjelenő adatoknak a programban megadott speciális változók, úgynevezett *host* változók felelnek meg.

Az SQL utasítások formailag SQL fenntartott szavakból, az utasítás által érintett objektumokból illetve részeikből, valamint elválasztó, határoló jelekből állnak. Az utasítás első szava vagy első szavai meghatározzák, milyen utasítás kerül végrehajtásra (pl. *CREATE TABLE* vagy *UPDATE*). Az utasítás végét általában a pontosvessző (;) jelzi.

Az utasításokat szigorú szintaktikai szabályok szerint kell megadnunk. Ezért minden egyes utasításhoz mellékeljük a használatához szükséges szintaktikai szabályok ábráját is. A szintaxis diagrammok értelmezését az 5.2. pontban tárgyaltuk. Amennyiben egy utasítás nem felel meg a szintaktikai szabályoknak (pl. hibásan írt kulcsszó, kötelezően megadandó elem hiánya, hibás elválasztó jel), akkor az utasítás nem kerül végrehajtásra és hibajelzést kapunk. A végrehajtott utasítás eredményéről az adatbázis-kezelő rendszer visszajelzést ad, melyben jelzi, mi történt. Interaktív üzemmódban ez valamilyen üzenet (pl. *Table created*, *No rows updated*, vagy *A tábla létrejött, Egy sor sem módosult*). Programban ez egy kommunikációs terület (*SQLCA* = *SQL Communication Area*) feltöltése az utasítások által kiváltott események pontos leírásával. Ezek közül legfontosabbak az *SQLCODE* illetve az *SQLSTATE* változók, amelyek értéke 0, ha az utasítás hibátlanul hajtott végre. Ettől eltérő érték az utasítás hibájára, vagy a normálistól valamilyen módon eltérő eredményre utal (például a módosítandó rekord nem létezik).

Általában minden olyan objektummal dolgozhatunk, amelyiknek mi vagyunk a tulajdonosa. Ezekben minden SQL utasítást végrehajthatunk. Más objektumaival csak akkor dolgozhatunk és csak olyan műveleteket végezhetünk el, amelyekre az objektum tulajdonosától engedélyt kaptunk (*GRANT*), vagy speciális rendszer privilégiumunk (pl. adatbázis-felügyelő) van

Az SQL utasításokat a következő csoportokba oszthatjuk:

- adatleíró (Data Definition Language, DDL) utasítások,
- adatkezelő (Data Management Language, DML) utasítások,
- tranzakció vezérlő utasítások,
- interaktív SQL ülést vezérlő utasítások,
- rendszervezérlő utasítások,
- programba beágyazott speciális utasítások.

Mivel az SQL utasítások alapvető funkciói és formái minden adatbázis-kezelő rendszerben azonosak, ezért a következőkben ismertetendő utasítások leírását olyan formában adjuk meg, hogy azt az olvasó a gyakorlati munkájához referencia könyvként is használhassa. Természetesen az utasítások speciális lehetőségeit az adott rendszer dokumentációjában találhatja meg.

5.6.2. Adatleíró utasítások

Az adatleíró (Data Definition Language, DDL) utasítások legfontosabb feladatai:

- Objektumok létrehozása (*CREATE* utasítások), módosítása (*ALTER* utasítások), megszüntetése (*DROP* utasítások).
- Jogosultságok megadása, visszavonása (*GRANT*, *REVOKE*).
- Az adatbázis optimális működéséhez szükséges statisztikák összegyűjtése.

A *CREATE*, *ALTER* és *DROP* típusú utasítások a végrehajtásuk ideje alatt kizárólagos módon lezárják a szóban forgó objektumot. Így ahhoz addig, amíg a művelet véget nem ér, más semmilyen módon nem férhet hozzá. A többi műveletet egymással és más műveletekkel párhuzamosan is végezhetjük ugyanazon az objektumon a párhuzamos feldolgozás általános szabályai szerint (ld. 8. fejezet).

A legfontosabb adatleíró utasításokat és funkciójukat az 5.14. táblázatban foglaltuk össze. Ezek közül a leggyakrabban használatos utasításokat az alábbiakban ismertetjük.

Léteznek ezeken kívül olyan adatleíró utasítások, amelyek egy felhasználó objektumait tartalmazó logikai/fizikai egységeket (*DATABASE/SCHEMA*) illetve az egyes objektumok fizikai tárolási módját (*TABLESPACE*) kezelik. Ezekkel elsősorban az adatbázis-felügyelő dolgozik. Használata erősen rendszerfüggő, ismertetésük meghaladja ennek a könyvnek a kereteit.

5.14. táblázat: Legfontosabb adatleíró utasítások és funkciójuk.

Utasítás	Funkció
ALTER SEQUENCE	Szekvencia (számgenerátor) paramétereinek megváltoztatása.
ALTER TABLE	Táblázat felépítésének módosítása.
ALTER TRIGGER	Trigger aktiválása, felfüggesztése.
CREATE INDEX	Index létrehozása.
CREATE SEQUENCE	Szekvencia (számgenerátor) létrehozása.
CREATE SYNONYM	Szinonima létrehozása.
CREATE TABLE	Táblázat létrehozása.
CREATE TRIGGER	Trigger létrehozása.
CREATE VIEW	Nézet létrehozása.
DROP INDEX	Index megszüntetése.
DROP SEQUENCE	Szekvencia (számgenerátor) megszüntetése.
DROP SYNONYM	Szinonima megszüntetése.
DROP TABLE	Táblázat megszüntetése.
DROP TRIGGER	Trigger megszüntetése.
DROP VIEW	Nézet megszüntetése.
GRANT	Jogosultságok adása objektumokra.
REVOKE	Jogosultságok visszavonása.

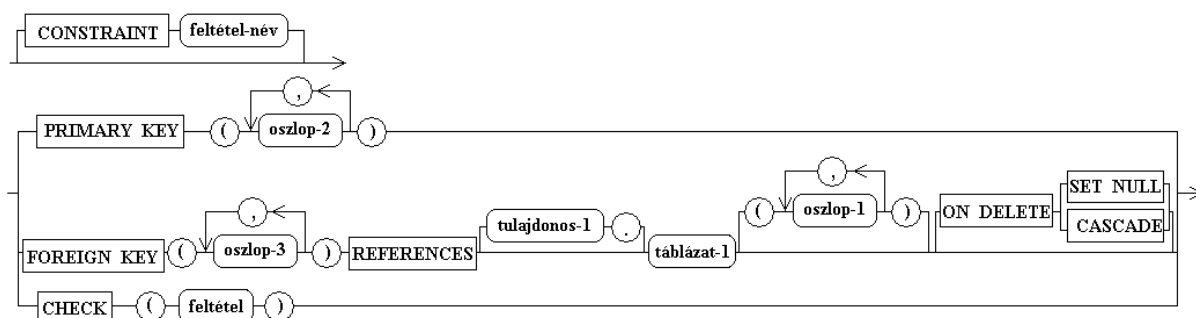
5.6.2.1. Objektumok létrehozása (*CREATE ...*)

Az objektumokat létrehozó SQL utasításokból a táblázatokat, nézeteket és az indexeket létrehozó utasításokat tárgyaljuk részletesebben.

CREATE TABLE

Funkciója: Táblázat létrehozása.

táblázat feltétel:



Jogosultság:

Saját táblázatot akkor hozhatunk létre, ha van CREATE TABLE jogosultságunk. Ez a jogosultságunk általában megvan. Ezen kívül felhasználható tárolási hellyel kell rendelkezünk abban a tablespaceben, amelyikben a táblázatot létre kívánjuk hozni.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
CREATE TABLE	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.
tulajdonos	A táblázat tulajdonosának a neve. Ha nem adjuk meg, akkor a sajátunk.
táblázat	A létrehozott táblázat neve.
oszlop	A táblázat oszlopainak neve (és sorrendje). Az n -ediknek felsorolt oszlop lesz a táblázat n -edik oszlopa. Kötelező megadni
adattípus	Megszabja milyen típusú adatokat tartalmaz az oszlop. A legfontosabb adattípusok (ld. 5.4. alfejezet): • <i>CHAR</i> (fix hosszúságú karakter) • <i>VARCHAR</i> vagy <i>VARCHAR2</i> (változó hosszúságú karakter) • <i>NUMBER</i> vagy <i>DECIMAL</i>(numerikus) • <i>DATE</i>, <i>TIME</i> vagy <i>TIMESTAMP</i> (dátum/időpont) • <i>LONG</i> (hosszú karakter) Az adattípust kötelező megadni az oszlop definiálásánál.
DEFAULT	Nem kötelező kulcsszó. Jelzi, hogy az oszlopnak van alapértelmezett értéke. Ha beíráskor nem adunk explicit értéket ennek az oszlopnak, akkor annak tartalma az alapértelmezett érték lesz.
kifejezés	Ez lesz az alapértelmezett érték. Típusa meg kell feleljen az oszlop adattípusának és be kell férjen az oszlop által meghatározott adatterületre.
CONSTRAINT	Nem kötelező kulcsszó. Jelzi, hogy az oszlopra vagy a táblázatra <i>feltétel</i> -t definiáltunk. Csak olyan érték, vihető be a táblázatba, amelyik a korlátozásban meghatározott feltételnek megfelel.

Kulcsszó vagy paraméter	Jelentése
feltételnév	A megadott feltétel elnevezése. A rendszer automatikusan kiegészíti a táblázat tulajdonosának a nevével. Ez jelenik meg a hibaüzenetekben, ha a feltételnek nem megfelelő adatot akarunk bevinni az oszlopba, és ezt a nevet kell megadnunk, ha a feltételt módosítani vagy törölni akarjuk az <i>ALTER TABLE</i> utasítással. Ha nem adjuk meg, akkor a rendszer automatikusan hozzárendel egy nevet, ami egyértelműen azonosítja a feltételt az adatbázisban.
NULL	Nem kötelező kulcsszó. Jelzi, hogy az oszlop tartalmazhat <i>NULL</i> értéket.
NOT NULL	Nem kötelező kulcsszó összetétel. Jelzi, hogy az oszlop <i>nem</i> tartalmazhat <i>NULL</i> értéket.
PRIMARY KEY	Nem kötelező kulcsszó összetétel. Jelzi, hogy az oszlop (kombináció) elsődleges kulcs lesz.
oszlop-2	Csak táblázat feltételnél adható meg. Erre az oszlop(ok)ra épül az egyedi index illetve az elsődleges kulcs.
FOREIGN KEY , REFERENCES	Nem kötelező kulcsszó kombinációk. Jelzik, hogy a táblázatban <i>idegen kulcs</i> -ot definiálunk. A <i>FOREIGN KEY</i> -t csak táblázat feltételnél adhatjuk meg, ott viszont – ha létezik ez a feltétel – kötelező.
oszlop-3	Csak táblázat feltételnél adható meg. Ez az oszlop (kombináció) lesz az idegen kulcs.
tulajdonos-1	Az idegen kulccsal hivatkozott táblázat tulajdonosa. Ha nem adjuk meg, akkor a saját táblázatunk.
táblázat-1	Erre a táblázatra hivatkozunk az idegen kulccsal.
oszlop-1	<i>táblázat-1</i> -nek erre az oszlopára/oszlopaira hivatkozunk az idegen kulccsal. Ez ott elsődleges kulcs, vagy egyedi index kell legyen rajta.
ON DELETE CASCADE	Nem kötelező kulcsszó kombináció. Definiálja, hogy ha az idegen kulccsal hivatkozott rekordot törlik a hivatkozott táblázatból, akkor az összes rá hivatkozó rekord is törlődik a hivatkozó táblázatból. Ha nem adjuk meg sem ezt, sem az <i>ON DELETE SET NULL</i> klauzulát, akkor a hivatkozott táblázatból olyan rekordot, amelyikre hivatkozás van, nem lehet törölni.
ON DELETE SET NULL	Nem kötelező kulcsszó kombináció. Definiálja, hogy ha az idegen kulccsal hivatkozott rekordot törlik a hivatkozott táblázatból, akkor az összes rá hivatkozó rekordban az idegen kulcs értéke a <i>NULL</i> érték lesz. Ha nem adjuk meg sem ezt, sem az <i>ON DELETE CASCADE</i> klauzulát, akkor a hivatkozott táblázatból olyan rekordot, amelyikre hivatkozás van nem lehet törölni.

Kulcsszó vagy paraméter	Jelentése
CHECK	Nem kötelező kulcsszó. Jelzi, hogy az oszlopba illetve a táblázatba beírható adatokra ellenőrzési feltételt adtunk meg.
feltétel	A beírandó adatoknak eleget kell tenniük ennek a feltételnek. Amennyiben ez nem teljesül, akkor a rekordot nem tudjuk beírni.

Megjegyzések:

- A táblázat neve nem lehet azonos a tulajdonosa egy másik táblázatának, nézetének, szekvenciájának, szinonimájának nevével sem.
- Az oszlopok nevének egy táblázaton belül egyértelműnek kell lenniük. Különböző táblázatoknak azokat az oszlopait, amik ugyanazokat az adatokat tartalmazzák célszerű azonos névvel definiálni.
- A *táblázat-feltételeket* az oszlop definíciók után kell megadnunk.
- A *feltétel-nevek* a sémán belül egyértelműek kell legyenek. Célszerű a neveket beszédessé tennünk, mert így az üzeneteket könnyebben értelmezhetjük.
- Miután a *CREATE TABLE* utasítással létrehoztunk egy üres táblázatot, azt az *INSERT* utasítással tölthetjük fel adatokkal, a *DELETE* utasítással törölhetünk, a *SELECT* utasítással választhatunk ki belőle és az *UPDATE* utasítással módosíthatunk benne adatokat.
- Meglevő táblázatot a *DROP TABLE* utasítással szüntethetünk meg és az *ALTER TABLE* utasítással módosíthatjuk a felépítését.
- Elsődleges kulcs, vagy annak összetevője nem tartalmazhat *NULL* értéket és nem lehet *LONG* típusú oszlop.
- Ahhoz, hogy egy táblázatra idegen kulccsal hivatkozhassunk, a táblázatnak már léteznie kell, a hivatkozott oszlopokon elsődleges kulcs vagy egyedi index kell legyen és *REFERENCES* jogosultságunk kell legyen rá.
- Összetett (több oszlopból álló) elsődleges és idegen kulcsot csak *táblázat feltétel*-ként definiálhatunk (ld. 3. példa).
- Ha nem adunk meg *tablespace*-t, akkor a táblázat az alapértelmezett *tablespace*-ben jön létre.
- Ha sem *NULL*, sem *NOT NULL* feltételt nem adunk meg, akkor az oszlop tartalmazhat *NULL* értéket.

Példák:

1. Létrehozzuk a hallgatók adatait tartalmazó *HALLGATO* táblázatunkat. A *HID* hallgatói azonosítót definiáljuk elsődleges kulcsnak és megköveteljük, hogy ezen kívül legalább a vezetéknévét (*VNEV*) megadják a hallgatónak és ellenőrizzük, hogy az évfolyam (*EVF*) értéke csak 1 és 5 között lehet. A keresztnévet (*KNEV*) és a születési dátumot (*SZDATUM*) nem kötelező megadni, A *TULAJ* oszlopot, mely a rekord beírójának azonosítóját tartalmazza majd később, módosításként adjuk hozzá a táblázathoz az *ALTER TABLE* utasítással. (ld. 5.6.2.3. pont 1. példa)

Utasítások:

```
create table hallgato(vnev char(20) constraint con_hallg_vnev not null
,knev char(20)
,hid char(5) constraint con_hallg_hid not null constraint pk_hallg_hid primary key
```

```
,szdatum date
,evf number(1) constraint con_hallg_evf check(evf between 1 and 5));
```

Eredmény: A táblázat a saját sémánkban létrejön. Mások – ha csak nincs speciális adatbázis felügyelői privilégiumuk – csak akkor dolgozhatnak vele, ha erre jogosultságot adunk nekik. Ha a feltételeknek a *CONSTRAINTS feltételnév* megadásával nem adtunk volna „beszédes” nevet, akkor a rendszer rendelt volna hozzá. Ez a 2. és 3. feladatban létrehozott táblázatokra is érvényes. A vezetéknév és keresztnév definiálhattuk volna változó hosszúságúra is (ld. 5.6.2.3. pont 1. példa).

2. Létrehozzuk a tantárgyak adatait tartalmazó *TANTARGY* táblázatunkat. A *TID* tantárgy azonosítót definiáljuk elsődleges kulcsnak és megköveteljük, hogy ezen kívül legalább a tantárgy megnevezését (*TMEGNEV*) megadják. Ellenőrizzük, hogy az óraszám (*ORASZAM*) nem negatív és a vizsgakötelezettség (*VIZSGA*) értéke csak 'I' vagy 'N' lehet. Definiáljuk, hogy ha nem adunk *VIZSGA*-nak értéket, akkor az automatikusan 'I' legyen. Az óraszámot és az oktató nevét (*OKTATO*) nem kötelező megadni.

Utasítás:

```
create table tantargy(
tid char(4) constraint con_tant_tid not null constraint pk_tant_tid primary key
,tmegnev char(12) constraint con_tant_tmegnev not null
,oktato char(20)
,oraszam number(1) constraint con_tant_oraszam check(oraszam >=0)
,vizsga char(1) default 'I' constraint con_tant_vizsga check (vizsga in('I','N')) );
```

3. Létrehozzuk a hallgatókat és a tantárgyakat összekapcsoló *MITHALLGAT* táblázatunkat. Az elsődleges kulcs (*HID,TID*) összetett, ezért csak *táblázat-feltétel*-ként tudjuk definiálni. A *HID* és *TID* idegen kulcsokat *oszlop-feltétel*-ként definiáljuk. A gyakorlatokon szerzett pontok számának (*GYAKPONT*) alapértelmezése 0, Ellenőrizzük az osztályzatot, $1 \leq \text{OSZTALYZAT} \leq 5$.

Utasítás:

```
create table mithallgat(
hid char(5) constraint con_mithlg_hid not null
constraint fk_mithlg_hallg references hallgato(hid)
,tid char(4) constraint con_mithlg_tid not null
constraint fk_mithlg_tant references tantargy(tid)
,gyakpont number(3) default 0
,osztalyzat number(1) constraint con_mithlg_osztalyzat
check(osztalyzat between 1 and 5)
,constraint pk_mithlg_hid_tid primary key(hid,tid)
);
```

CREATE VIEW

Funkciója: Nézet létrehozása.

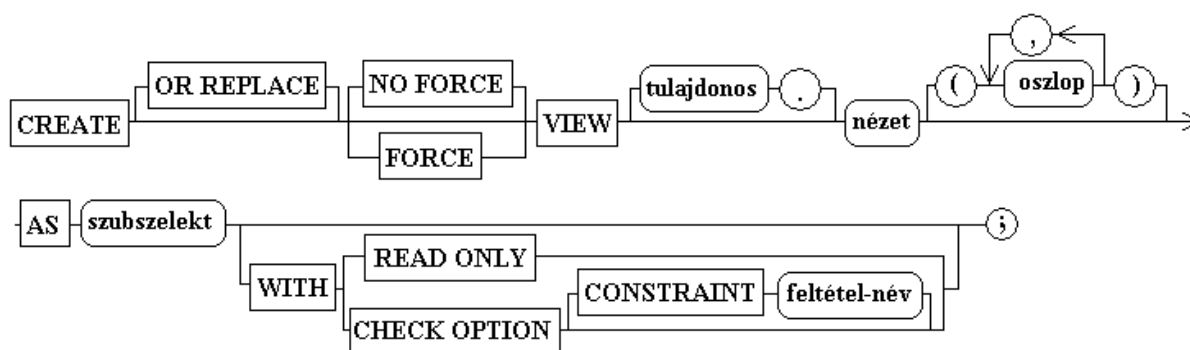
Magyarázat:

A nézetben fizikailag nincsenek adatok. Csak logikailag tartalmazza a bázistáblázat(ok) azon adatait, amelyeket a nézet definiálásakor jelöltünk ki. A nézetet definiáló szubszelekt sorai adják a nézet sorait, oszlopai a nézet oszlopait. A nézet logikai és a bázis táblázat(ok) fizikai adatai mindig megegyeznek. Ezért, ha egy táblázat adatait módosítjuk, ez automatikusan az összes ráépülő nézet adatait is módosítja. Hasonlóképpen, ha egy nézetben módosítjuk a logikai adatokat, ez egyben a megfelelő bázis táblázatok adatainak fizikai módosítását is jelenti.

Nézeteket a következő célokból hozunk létre:

- Adatbiztonság érdekében. A teljes táblázat helyett annak csak a nézet által meghatározott soraihoz és oszlopaihoz engedélyezzük a hozzáférést.
- Utasítások egyszerűbbé tétele érdekében. Egy vagy több táblázatból bonyolult módon kiválasztott adatokat nézetként definiálhatunk. A továbbiakban a hosszas kiválasztás újra leírása helyett a nézetnévvel hivatkozhatunk ugyanazokra az adatokra.
- Táblázatok oszlopainak más néven való megjelenítésére, az eredeti adatok mellett vagy helyett azokból származtatott új adatok állandó megjelenítésére.

A CREATE VIEW egyszerűsített szintaxisa:



Jogosultság:

Nézetet általában akkor hozhatunk létre, ha rendelkezünk a SELECT, INSERT, DELETE vagy UPDATE privilégiumok valamelyikével az összes olyan táblázatra, amelyekre a nézet hivatkozik.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
CREATE, VIEW, AS	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.
OR REPLACE	Elhagyható kulcsszavak. Ha megadjuk és létezik már azonos nevű nézet, akkor az törlődik. Ha nem adjuk meg és már létezik azonos nevű nézet, akkor az utasítás nem hajtódik végre.

Kulcsszó vagy paraméter	Jelentése
FORCE	Akkor is létrehozza a nézetet, ha van olyan bázis táblázata, amelyik nem létezik, vagy használatára nincs jogosultságunk. A nézet létrehozásakor figyelmeztetést kapunk. A nézetet ilyenkor nem tudjuk használni.
NO FORCE	A nézet csak akkor jön létre, ha az összes bázis táblázat létezik és a használatukra meg van a jogosultságunk. Ez az alapértelmezés.
tulajdonos	A nézet tulajdonosának a neve. Ha nem adjuk meg, akkor a sajátunk.
nézet	A létrehozott nézet neve.
oszlop	a nézet oszlopainak neve (és sorrendje). Az n -edik oszlop tartalmazza a szubszelekt kiválasztási listája n -edik elemének adatait. Ha nem adjuk meg, akkor a nézet oszlopoi megegyeznek a szubszelekt oszlopaival.
szubszelekt	Meghatározza, hogy a bázis táblázat(ok)ból hogyan választjuk ki a nézet oszlopait és sorait
WITH READ ONLY	Nem kötelező kulcsszavak. Jelzik, hogy a nézetet csak olvasni lehet. A nézeten keresztül nem lehet sem beírni, sem törölni, sem módosítani adatokat.
WITH CHECK OPTION	Nem kötelező kulcsszavak. Jelzik, hogy a nézetbe csak olyan értékeket írhatunk be, illetve csak olyan értékre módosíthatunk, amelyek megfelelnek a nézet definíciójának. (ld. 5.6.3.4. pont 3. példa) Ha nem így definiáltuk, akkor bevihetünk, illetve módosíthatunk a bázis táblában a nézet feltételeit nem teljesítő értékekre is. Ezeket természetesen nem látjuk a nézetben.
CONSTRAINT	Nem kötelező kulcsszó. Jelzi, hogy a nézet sorainak kiválasztására szolgáló feltételnek nevet adtunk.
feltételnév	A nézet sorainak kiválasztására szolgáló feltétel neve. Ha nem adjuk meg, akkor a rendszer automatikusan hozzárendel egy nevet, ami egyértelműen azonosítja a feltételt az adatbázisban.

Megjegyzések:

- Nézetet építhetünk táblázat(ok)ra és/vagy más nézetekre is. Azokat a táblázatokat, amelyekből a nézet adatai származnak nevezzük a nézet *bázis táblázatainak*.
- A nézet neve nem lehet azonos ugyanannak a tulajdonosnak egy másik nézete, táblázata, szekvenciája vagy szinonimája nevével. Ha a létrehozandó nézettel azonos nevű táblázat, szekvencia vagy nézet van már, akkor hiba jelzést kapunk. Kivétel, ha megadtuk az *OR REPLACE* opciót, amikor is az azonos nevű nézet automatikusan felülíródik az újjal.
- Nézetet közvetlenül nem tudunk módosítani. Meglevő nézet definícióját úgy tudjuk megváltoztatni, hogy a *CREATE VIEW* utasításban megadjuk az *OR REPLACE* opciót. Ez törli a korábbi definíciót és az új leírás alapján hozza létre a nézetet. A korábban kiadott jogosultságok továbbra is érvényben maradnak.
- Nézetre, mivel azok fizikailag nem tartalmaznak adatokat, nem építhetünk indexet.

- Nézeteket a táblázatokhoz hasonlóan használhatunk az SQL utasításokban. Kivételek ezek alól a csak olvasható nézetek, amelyek nem szerepelhetnek az *INSERT*, *DELETE*, *UPDATE* utasítások egyikében sem.
- Egy nézet csak olvasható, ha a definíciójában megadtuk a *WITH READ ONLY* kiegészítést, vagy ha a nézet adatait kiválasztó szubszelekt *JOIN*-t, számított értéket vagy *GROUP* klauzulát tartalmaz.
- Nézetten keresztül csak akkor vihetünk be adatokat, ha a bázis táblázat minden *NOT NULL*-ként definiált oszlopának értéket adunk.
- Az oszlop neveket akkor célszerű megadni, ha a nézet oszlopait a bázis táblázat(ok) oszlopaitól eltérő módon akarjuk elnevezni.
- Meg kell adnunk a nézet oszlopainak neveit, ha a szubszelekt listájában nem csak oszlopnevek, vagy oszlopnévként használható *AS lista-elem-helyettesítő nevek* szerepelnek.
- Ha a nézet definícióban oszlopnevet adunk, akkor minden oszlopot meg kell neveznünk.
- A saját nézeteinkre minden jogosultságunk meg van. Mások, ha nincsen általános adatbázis-felügyelői jogosultságuk, csak akkor dolgozhatnak velük, ha erre a *GRANT* utasítással jogosultságot adunk nekik. A *GRANT* utasítással kiadott jogosultságokat a *REVOKE* utasítással vonhatjuk vissza. (ld. 5.6.2.4. pont)

Példák:

1. A személyiségi jogok védelme miatt létrehozuk a *MITHALLGAT1* táblázatot, amelyik a *MITHALLGAT* adatait tartalmazza a hallgatók azonosítására szolgáló *HID* oszlop nélkül. Ezt a nézetet már mindenki olvashatja.

Utasítások:

```
create view mithallgat1 as select tid,gyakpont,osztalyzat from mithallgat;
grant select on mithallgat1 to public;
```

Eredmény: Mindenki megtekintheti minden egyes tantárgy eredményeit, de azokat nem tudja a hallgatókhoz hozzákapcsolni.

2. Létrehozuk a mintaadatbázisunk minden adatát tartalmazó *OKTATAS* nézetet. Ebből közvetlen lekérdezéssel megkaphatunk minden olyan információt, amit e nélkül a bázis-táblázatokból csak *JOIN* műveletek segítségével tudnánk előállítani. (A nézet a join miatt automatikusan csak olvasható, így nem kell megadnunk a *WITH CHECK OPTION*-t).

Utasítás:

```
create or replace view oktatás
(hid,vnev,knev,szdatum,evf,tulaj,tid,tmegnev,oktato,orszam,vizsga,
gyakpont,osztalyzat)
as select
h.hid,h.vnev,h.knev,h.szdatum,h.evf,h.tulaj,t.tid,t.tmegnev,t.oktato,
t.orszam,t.vizsga,m.gyakpont,m.osztalyzat
from hallgato h,tantargy t,mithallgat m
where h.hid=m.hid and t.tid = m.tid;
```

Eredmény: Bármely információt megkaphatunk az OKTATAS nézetből. Így például azt, hogy Kiss Endre, milyen tantárgyakból hogyan vizsgázott, a joinokkal összekapcsolt HALLGATO, TANTARGY és MITHALLGAT táblázatok helyett egyszerűen a

```
select vnev,knev,tmegnev,osztalyzat from oktatás
where vnev='Kiss' and knev='Endre';
```

utasítás szolgáltatja.

Az áttekinthetőség érdekében a nézet definiálásában az egyes táblázatok neve helyett, ahol lehetett a rövidebb *h,t,m* minősítő neveket használtuk.

Hatékonysági szempontból azonban célszerűbb azokat az információkat, amelyek ugyanabból az egy táblázatból elérhetőek a táblázat közvetlen lekérdezésével előállítani.

CREATE INDEX

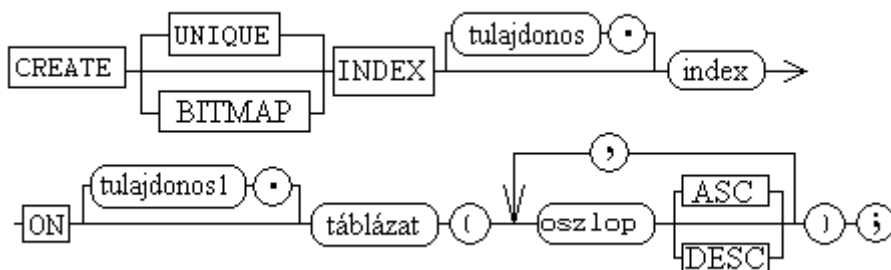
Funkciója: Index létrehozása.

Magyarázat:

Indexet hoz létre a megadott táblázat egy vagy több oszlopán. Az indexek segítségével meggyorsíthatjuk az indexelt oszlopok értéke alapján

- egyes rekordok elérését,
- rekordok szekvenciális feldolgozását.

A CREATE INDEX egyszerűsített szintaxisa:



Jogosultság:

Indexet akkor hozhatunk létre egy táblázaton, ha az a sajátunk vagy van rá index privilégiumunk.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
CREATE, INDEX, ON	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.
UNIQUE	A definiált index egyedi. A táblázatban nem lehet két rekord, amelyekben az index értéke megegyezik.
BITMAP	Bit-térkép index jön létre.
tulajdonos	Az index tulajdonosának a neve. Ha nem adjuk meg, akkor a sajátunk.
index	A létrehozott index neve.

tulajdonos1	Az indexet tartalmazó táblázat tulajdonosának a neve. Ha nem adjuk meg, akkor a saját táblázatunk
táblázat	Annak a táblázatnak a neve, amelyikre az indexet felépítjük.
oszlop	Erre az oszlop(ok)ra épül az index
ASC	Az index az oszlop értékekre növekvő sorrendben szolgáltatja a rekordokat. Ez az alapértelmezés.
DESC	Az index csökkenő sorrendben szolgáltatja a rekordokat.

Megjegyzések:

- Ha nem adtunk meg *UNIQUE*-ot, akkor az index duplikált lesz.
- Egy táblázatra akárhány indexet építhetünk. A gyakorlatban csak azokra az oszlopokra vagy oszlopkombinációkra érdemes indexet készíteni, amelyek alapján gyakran el kell érni rekordokat. A felesleges indexek helyet foglalnak el és lelassítják az adatok változtatását.
- *Dátum/TIMESTAMP* típusú értékekre célszerű csökkenő indexet építeni. Így a legújabb adatokat látjuk legelőször.
- Célszerű az indexek elnevezésére egységes logikát alkalmazni. Például minden indexnév „I” vagy „IND”-del kezdődik, utána következik rövidítve a táblázat neve és az oszlop(ok) neve, amely(ek)re épül. Az egyes elemeket aláhúzás választja el. Eszerint a *HALLGATO* táblázat *VNEV* (vezetéknév) oszlopára épülő index neve például célszerűen *I_HALLG_VNEV* vagy *IND_HLG_VNEV*.
- Összetett indexnél az indexoszlopok megadásának sorrendje lényeges. A keresés mindig az első oszlop értékeivel kezdődik. Ezért a névszerinti keresésre szolgáló indexben az indexoszlopok sorrendje *VNEV,KNEV*. Ez az index egyben alkalmas a vezetéknév alapján történő keresésre is, arra fölösleges még egy indexet felépítenünk. Ha viszont névnapok ünneplésére gyakran keressük a megadott keresztnévű hallgatókat is, akkor célszerű egy külön indexet felépíteni a *KNEV* (keresztnév) oszlopra is (lásd a következő megjegyzést).
- Ha nem adjuk meg az index teljes értékét, akkor a keresést az index csak akkor gyorsítja meg, ha a keresendő érték pontosan ismert része az index legelején van. Így hiába építettünk fel a *HALLGATO* táblázatra egy a vezetéknév- (*VNEV*) és keresztnév- (*KNEV*) együttesen tartalmazó összetett indexet ebben az oszlop sorrendben (*I_HALLG_VNEV_KNEV*), ez semmit nem gyorsít sem a *WHERE KNEV='Hugó'* sem a *WHERE VNEV LIKE '%agy'* típusú feltételek kiértékelésénél.

Példa:

1. Készítsünk indexet a *HALLGATO* táblázat név alapján való hozzáféréseinek a felgyorsítására.

Utasítás:

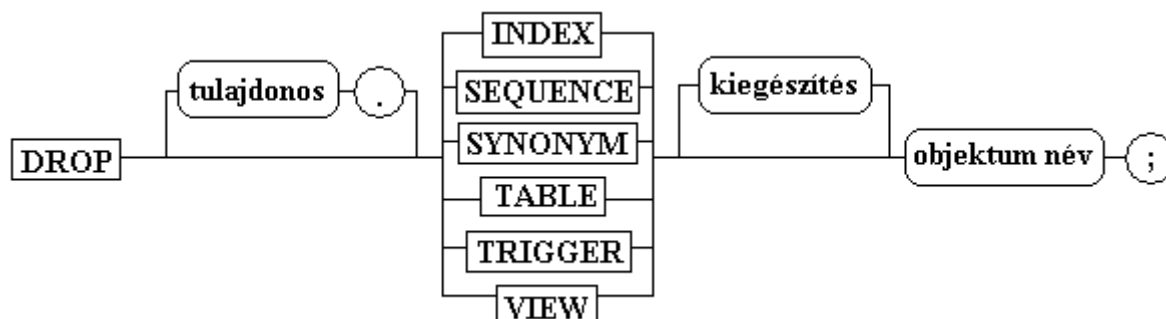
```
create index i_hallg_vnev_knev on hallgato(vnev, knev);
```

Eredmény Az index létrejön. Ha eddig nem volt index a vezetéknévre, akkor a hallgatók név vagy vezetéknév alapján való megtalálása a táblázatból nagy táblázat esetén nagyságrendekkel gyorsabb lesz.

5.6.2.2. Objektumok megszüntetése (DROP ...)

Funkciójuk: A *DROP ...* típusú utasításokkal meglévő objektumokat szüntethetünk meg.

Általános szintaxisuk:



Jogosultság:

Objektumot akkor szüntethetünk meg, ha az a saját objektumunk, vagy általános rendszerjogosultságunk van az adott típusú objektum megszüntetésére (Pl. adatbázis-felügyelő).

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
DROP	Kötelező kulcsszó. Megadja, hogy egy objektumot megszüntető utasítás kerül végrehajtásra.
tulajdonos	Az objektum tulajdonosának a neve. Ha nem adjuk meg, akkor az a saját objektumunk.
INDEX SEQUENCE SYNONYM TABLE TRIGGER VIEW	Kötelező kulcsszavak, melyekből egyet és csak egyet kell megadni. Meghatározza a megszüntetendő objektum típusát.
objektum név	A törlendő objektum neve.
kiegészítés	Kiegészíti a törlési utasítást.

Megjegyzések:

- A kiegészítések közül a legfontosabb a *CASCADE CONSTRAINT*. Ha megadjuk, akkor megszünteti a *táblázat*-ra hivatkozó idegen kulcsokat. Ha ezt nem adjuk meg, akkor olyan táblázatot, amelyikre idegen kulccsal hivatkozunk, nem tudunk megszüntetni.
- Mind a *DROP TABLE táblázat vagy nézet*, mind a *DELETE FROM táblázat vagy nézet* utasítás törli a *táblázat vagy nézet* összes sorát. A kettő között az a különbség, hogy *DELETE* esetén az üres táblázat illetve nézet megmarad, így abba újra vihetünk be adatokat. A *DROP TABLE* illetve a *DROP VIEW* utasítás viszont nemcsak az adatokat, hanem a teljes táblázatot/nézetet is törli. Így abba csak akkor vihetünk be újra adatokat, ha ismét létrehozzuk. Ha sok adatot tartalmazó táblázatot akarunk üressé tenni, akkor a *DROP TABLE* sokkal gyorsabb.

- A *DROP TABLE* utasítás automatikusan megszünteti a táblázatra épülő indexeket és triggereket is.

Példa:

1. Meg akarjuk szüntetni a *HALLGATO* táblázatot. Mivel a *MITHALLGAT* táblázat hivatkozik rá, ezt csak úgy tehetjük meg, hogy megadjuk a *CASCADE CONSTRAINTS* klauzulát is.

Utasítás:

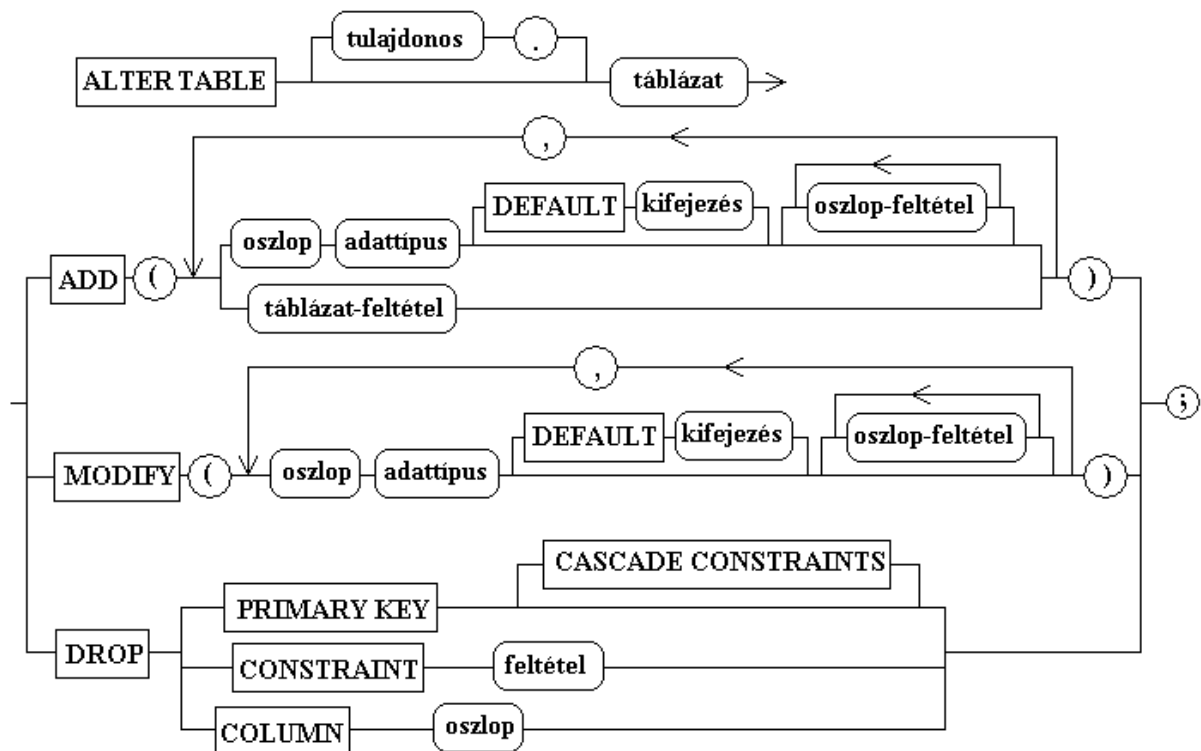
drop table hallgato cascade constraints;

Eredmény: A *HALLGATO* táblázat többé nem létezik és megszűnik a *MITHALLGAT*-ból ráhivatkozó *fk_mithallg_hallg* idegen kulcs is.

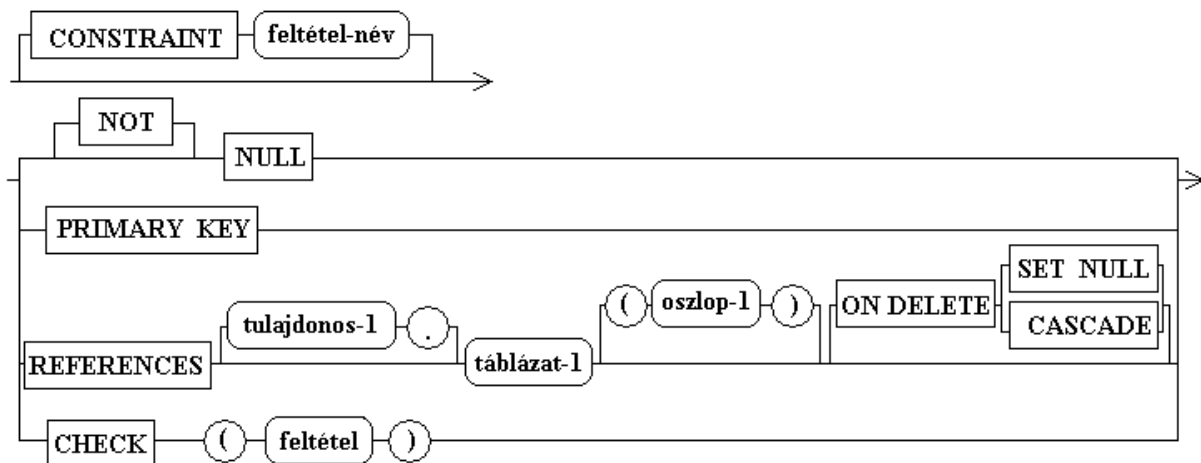
5.6.2.3. Objektumok módosítása (*ALTER ...*)

Funkciójuk: Az *ALTER ...* típusú utasításokkal meglévő objektum (táblázat, index, szekvencia, trigger, tablespace) jellemzőit változtathatjuk meg. Egyes rendszerekben ezen kívül módosíthatjuk vele a jelszavunkat és az interaktív SQL ülés paramétereit is. Legfontosabb közülük az *ALTER TABLE* utasítás. Itt csak ezt tárgyaljuk.

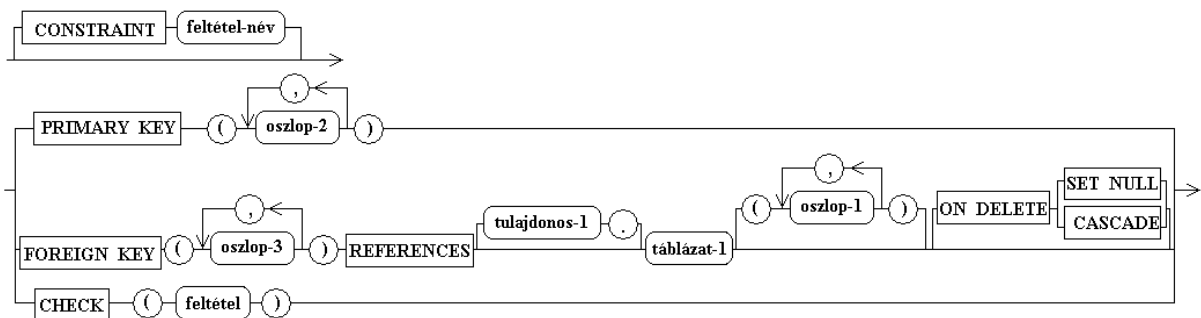
ALTER TABLE egyszerűsített szintaxisa:



ahol oszlop feltétel:



táblázat feltétel:



Jogosultság:

- A saját táblázatunkat minden korlátozás nélkül módosíthatjuk.
- Más táblázatát csak akkor módosíthatjuk, ha van rá ALTER jogosultságunk vagy rendelkezünk általános adatbázis-felügyelői jogosultsággal.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
ALTER TABLE	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.
tulajdonos	A módosítandó táblázat tulajdonosa. Ha nem adjuk meg, akkor a saját táblázatunk
táblázat	A módosítandó táblázat neve.
ADD	Nem kötelező kulcsszó. Jelzi, hogy új oszlopot vagy táblázat-feltételt adunk hozzá a táblázathoz.
MODIFY	Nem kötelező kulcsszó. Jelzi, hogy meglévő oszlop definícióját módosítjuk a táblázatban.
DROP	Nem kötelező kulcsszó. Jelzi, hogy meglévő oszlopot, oszlop-feltételt, vagy táblázat-feltételt szüntetünk meg.
oszlop	a hozzáadandó, módosítandó illetve megszüntetendő oszlop neve.

Kulcsszó vagy paraméter	Jelentése
adattípus	Megszabja milyen típusú adatokat tartalmaz az új illetve a módosított tulajdonságú oszlop. A legfontosabb adattípusok (ld. 5.4. alfejezet): • <i>CHAR</i> (fix hosszúságú karakter) • <i>VARCHAR</i> vagy <i>VARCHAR2</i> (változó hosszúságú karakter) • <i>NUMBER</i> vagy <i>DECIMAL</i>(numerikus) • <i>DATE</i>, <i>TIME</i> vagy <i>TIMESTAMP</i> (dátum/időpont) • <i>LONG</i> (hosszú karakter) Az adattípust kötelező megadni új oszlop definiálásánál. Meglévő oszlopnál, ha nem adjuk meg, akkor megmarad az eredeti adattípus.
DEFAULT	Nem kötelező kulcsszó. Jelzi, hogy az új oszlopnak van alapértelmezett értéke, illetve meglévő oszlopnak megváltozik az alapértelmezett értéke. Ha beírásakor nem adunk explicit értéket ennek az oszlopnak, akkor annak tartalma az alapértelmezett érték lesz. A már bent lévő adatok megtartják a korábbi értéküket.
kifejezés	Ez lesz az alapértelmezett érték. Típusa meg kell feleljen az oszlop adattípusának és be kell férjen az oszlop által meghatározott adatterületre.
CONSTRAINT	Nem kötelező kulcsszó. Jelzi, hogy az oszlopra vagy a táblázatra új feltételt definiálunk, vagy meglévőt módosítunk, törölünk. Az oszlopba vagy a táblázatba nem vihető be akármilyen érték, csak olyan, amelyik a korlátozásban meghatározott feltételnek megfelel.
feltételnev	Az új, illetve a törlendő vagy módosítandó feltétel elnevezése. A rendszer automatikusan kiegészíti a táblázatot tartalmazó séma nevével. Ez jelenik meg a hibaüzenetekben, ha a feltételnek nem megfelelő adatot akarunk bevinni az oszlopba, és ezt a nevet kell megadnunk, ha a feltételt módosítani vagy törölni akarjuk az <i>ALTER TABLE</i> utasítással. Ha újonnan definiált feltételnél nem adjuk meg, akkor a rendszer automatikusan hozzárendel egy nevet, amivel egyértelműen azonosítja a feltételt az adatbázisban.
NULL	Nem kötelező kulcsszó. Jelzi, hogy az oszlop tartalmazhat <i>NULL</i> értéket.
NOT NULL	Nem kötelező kulcsszó összetétel. Jelzi, hogy az oszlop nem tartalmazhat <i>NULL</i> értéket.
PRIMARY KEY	Nem kötelező kulcsszó összetétel. Jelzi, hogy az oszlop (kombináció) elsőleges kulcs.
FOREIGN KEY, REFERENCES	Nem kötelező kulcsszó kombinációk. Jelzik, hogy a táblázatban idegen kulcsot definiálunk. A <i>FOREIGN KEY</i>-t csak táblázat feltételnél adhatjuk meg, de ott viszont – ha létezik ez a feltétel – akkor kötelező.
tulajdonos-1	Az idegen kulccsal hivatkozott táblázat tulajdonosa. Ha nem adjuk meg, akkor a saját táblázatunk.
táblázat-1	Erre a táblázatra hivatkozunk az idegen kulccsal.

Kulcsszó vagy paraméter	Jelentése
oszlop-1	táblázat-1-nek erre az oszlopára/oszlopaira hivatkozunk az idegen kulccsal. Ez ott elsődleges kulcs, vagy egyedi index kell legyen rajta.
ON DELETE CASCADE	Nem kötelező kulcsszó kombináció. Definiálja, hogy ha az idegen kulccsal hivatkozott rekordot törlik a hivatkozott táblázatból, akkor az összes rá hivatkozó rekord is törlődik a hivatkozó táblázatból. Ha nem adjuk meg sem ezt, sem az ON DELETE SET NULL klauzulát, akkor a hivatkozott táblázatból olyan rekordot, amelyikre hivatkozás van nem lehet törölni.
ON DELETE SET NULL	Nem kötelező kulcsszó kombináció. Definiálja, hogy ha az idegen kulccsal hivatkozott rekordot törlik a hivatkozott táblázatból, akkor az összes rá hivatkozó rekordban az idegen kulcs értéke a NULL érték lesz. Ha nem adjuk meg sem ezt, sem az ON DELETE CASCADE klauzulát, akkor a hivatkozott táblázatból olyan rekordot, amelyikre hivatkozás van nem lehet törölni.
CHECK	Nem kötelező kulcsszó. Jelzi, hogy az oszlopba illetve a táblázatba beírható adatokra ellenőrzési feltételt adtunk meg.
feltétel	A beírandó adatoknak eleget kell tenniük ennek a feltételnek. Amennyiben ez nem teljesül, akkor a rekordot nem tudjuk beírni.

Megjegyzések:

- Az ALTER TABLE utasítással módosíthatjuk a táblázat következő jellemzőit (a felsorolás nem teljes):
 - A táblázat oszlopainak definícióit.
 - A táblázatra érvényes általános integritási feltételeket (elsődleges kulcs, idegen kulcsok). Ahhoz, hogy egy táblázatra idegen kulccsal hivatkozhatunk, a táblázatnak már léteznie kell, a hivatkozott oszlopokon elsődleges kulcs vagy egyedi index kell legyen és *REFERENCES* jogosultságunk kell legyen rá.
 - Az egyes oszlopok adataira fennálló korlátozásokat.
 - Adattárolási jellemzőket.
 - Megszüntethetünk meglévő, vagy hozzáadhatunk új oszlopokat.
 - Megszüntethetünk meglévő, vagy hozzáadhatunk új oszlop-feltételeket.
 - Megszüntethetünk meglévő, vagy hozzáadhatunk új táblázat-feltételeket.
 - Ideiglenesen felfüggeszthetjük, vagy visszaállíthatjuk feltételek érvényességét vagy triggerek működését.
- Az ADD által hozzáadott új oszlop a táblázat utolsó oszlopa lesz.
- Az új oszlop értéke a táblázat már meglévő soraiban a *NULL érték* illetve az oszlopra *DEFAULT*-ként megadott érték lesz.
- Meglévő oszlop módosítására a következő kiegészítések érvényesek:
 - A feltételek közül csak a *NULL/NOT NULL* feltételt módosíthatjuk. Természetesen *NOT NULL*-ra csak akkor módosíthatunk, ha a táblázatban benn lévő adatok között nincsen *NULL érték*. Ha más feltételt akarunk módosítani, akkor azt előbb meg kell szüntetnünk, majd újként létre kell hoznunk (lásd 3. példa).
 - Az adattípus megváltoztatása vagy hosszának a csökkentése általában csak akkor lehetséges, ha az oszlop értéke a táblázat minden sorában a *NULL érték*.

Kivétel ez alól a *CHAR* és *VARCHAR/VARCHAR2* közti váltás, mely feltöltött adatokkal is megengedett, ha a mező hosszát nem csökkentjük.

- Karakter típusú oszlop hosszát és numerikus típusú oszlop pontosságát minden korlátozás nélkül növelhetjük.
- Oszlop megszüntetéskor
 - az oszlopra érvényes korlátozások is törölődnek. Ha idegen kulccsal hivatkozunk a törlendő oszlopra, vagy más oszlop is szerepel a feltételben, akkor meg kell adnunk a *CASCADE CONSTRAINTS* kiegészítést is.
 - az oszlopra felépített és az oszlopot tartalmazó indexek is megszűnnek.
 - olyan oszlopot, amelyik elsődleges kulcs, vagy része az elsődleges kulcsnak nem szüntethetünk meg. Ezt csak úgy tehetjük meg, hogy előbb *DROP PRIMARY KEY*-vel megszüntetjük az elsődleges kulcsot, majd ezután megszüntethetjük az oszlopot.
- Egy *DROP* klauzulában csak egy elemet szüntethetünk meg, de egy *ALTER TABLE* utasításban akárhány *DROP* klauzulát megadhatunk.
- Olyan elsődleges kulcs, vagy egyedi index, amelyikre idegen kulccsal hivatkoznak csak a *CASCADE CONSTRAINTS* kiegészítés megadásával szüntethető meg.
- Egyes egyszerűbb adatbázis-kezelő rendszerek nem ismerik az *ALTER* típusú utasításokat. Ezekben, ha egy objektumot módosítani kell, akkor azt egy *DROP* utasítással megszüntetjük, majd újra létrehozuk a kívánt formában. Természetesen táblázat esetében előtte az adatokat kimentjük, majd az új táblázatba visszatöltjük.

Példák:

1. Módosítjuk a hallgatók adatait tartalmazó *HALLGATO* táblázatunkat. A *VNEV*, *KNEV* vezeté- és keresztnévet hosszabbra, *VARCHAR (30)* (ill. *VARCHAR2(30)*)-ra definiáljuk. Ezenkívül létrehozuk a *TULAJ CHAR(8)* új oszlopot is, amelyik alapértelmezésként a sort beíró SQL azonosítóját, a *USER SQL* függvényt tartalmazza.

Utasítás:

```
alter table hallgato
modify (vnev varchar2(30),knev varchar2(30))
add (tulaj char(8) default user);
```

Eredmény: A táblázat definíciója módosul. A már benn levő sorokban az új *TULAJ* mező értéke a *saját azonosítónk* lesz, a többi adat változatlan marad.

2. Módosítjuk a tantárgyak adatait tartalmazó *TANTARGY* táblázatunkat. A *VIZSGA* alapértelmezése *N* legyen.

Utasítás:

```
alter table tantargy modify (vizsga default 'N') ;
```

Eredmény: A táblázat definíciója módosul. A már bent levő sorokban a *VIZSGA* értéke változatlan marad. Az ezután beírt sorokba – ha nem adtunk neki értéket – automatikusan '*N*' kerül be.

- Tovább módosítjuk a hallgatók adatait tartalmazó *HALLGATO* táblázatunkat. Az évfolyam ellenőrzésére szolgáló *CON_HALLG_EVF* feltételt (*evf between 1 and 5*) akarjuk megváltoztatni. Ezt csak úgy tehetjük meg, hogy megszüntetjük a feltételt, majd az új definícióval létrehozuk.

Utasítás:

```
alter table hallgato drop constraint con_hallg_evf
add constraint con_hallg_evf check (evf between 0 and 6);
```

Eredmény: Mivel a már bent levő rekordok a korábbi feltétel miatt teljesítik az új feltételt is, a táblázat definíciója módosul.

5.6.2.4. Jogosultságok kezelése (*GRANT, REVOKE*)

Ezekkel az utasításokkal jogosultságokat adhatunk meg objektumok használatára, illetve megadott jogosultságokat vonhatunk vissza.

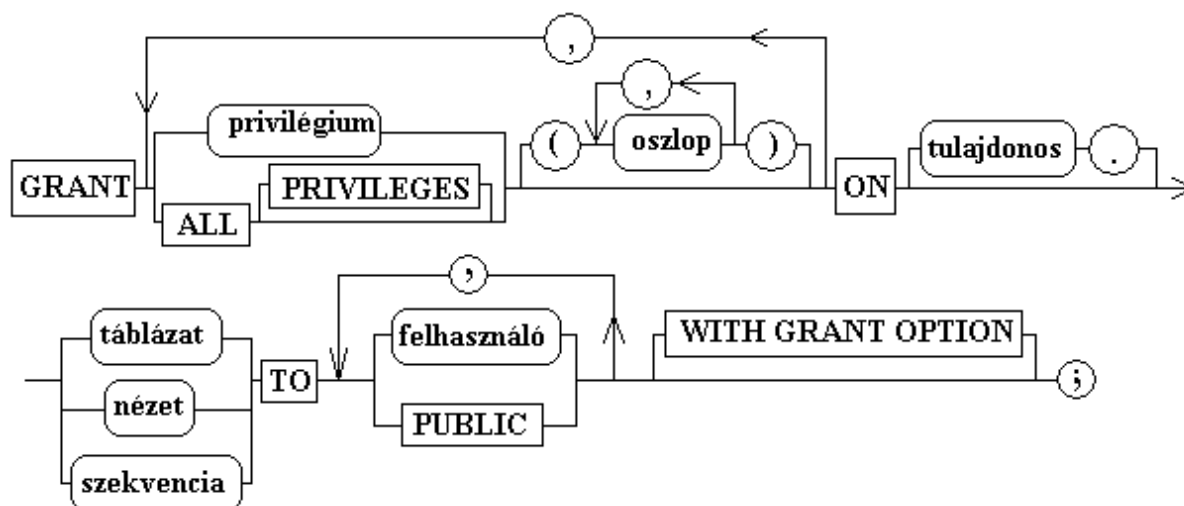
GRANT

Funkciója: Objektumok különféle módon való használatára adhatunk jogosultságot egyes felhasználóknak illetve felhasználói csoportoknak.

Jogosultság:

Objektumon való műveletre akkor adhatunk jogosultságot, ha az a saját objektumunk. Más objektumára csak akkor adhatunk, ha azon az adott műveletre *WITH GRANT OPTION*-nal kaptunk továbbadható jogosultságot. Ez a korlátozás az adatbázis-felügyelőre is érvényes.

Egyszerűsített szintaxisa:



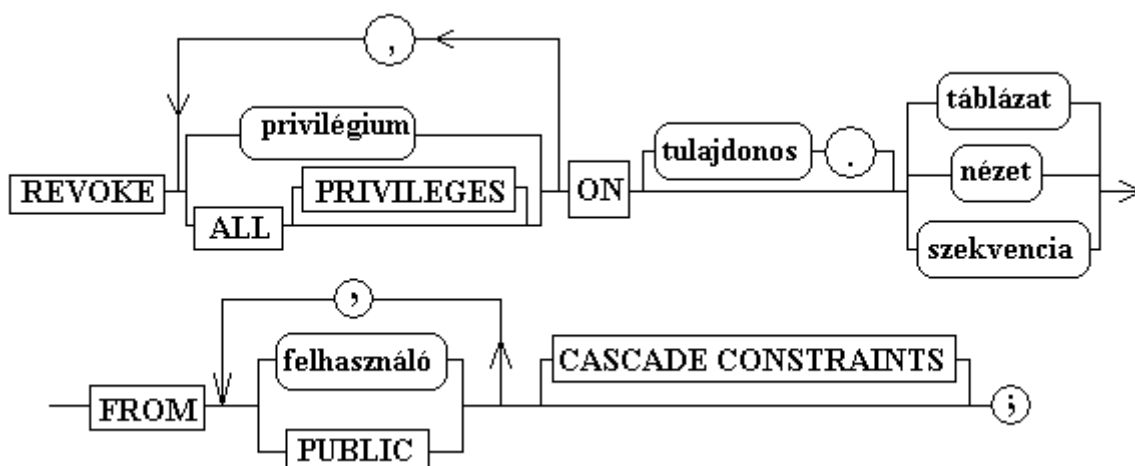
REVOKE

Funkciója: Objektumokra kiadott jogosultságokat vonhatunk vissza egyes felhasználóktól illetve felhasználói csoportoktól. A visszavont jogosultságú műveletet az, akitől a jogosultságot megvonták a továbbiakban nem hajthatja végre az objektumon.

Jogosultság:

Egy adott objektumra kiadott jogosultságot az vonhatja vissza, aki azt a *GRANT* utasítással engedélyezte.

Egyszerűsített szintaxisa:



Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése														
GRANT, ON, TO illetve REVOKE, ON, FROM	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.														
privilegium	Az itt felsorolt privilégiumokat, jogosultságokat adjuk meg. A következő jogosultságok bármilyen kombinációja megengedett, de nem minden objektumra engedélyezhető mindegyik. <table><tr><td>ALTER</td><td>objektum definíciójának módosítása</td></tr><tr><td>DELETE</td><td>törlés táblázatból, nézetből</td></tr><tr><td>INDEX</td><td>index létrehozása táblázaton</td></tr><tr><td>INSERT</td><td>adatbevitel táblázatba, nézetbe</td></tr><tr><td>REFERENCES</td><td>idegen kulccsal való hivatkozás táblázatra</td></tr><tr><td>SELECT</td><td>olvasás táblázatból, nézetből, szekvenciából</td></tr><tr><td>UPDATE</td><td>módosítás táblázatban, nézetben</td></tr></table>	ALTER	objektum definíciójának módosítása	DELETE	törlés táblázatból, nézetből	INDEX	index létrehozása táblázaton	INSERT	adatbevitel táblázatba, nézetbe	REFERENCES	idegen kulccsal való hivatkozás táblázatra	SELECT	olvasás táblázatból, nézetből, szekvenciából	UPDATE	módosítás táblázatban, nézetben
ALTER	objektum definíciójának módosítása														
DELETE	törlés táblázatból, nézetből														
INDEX	index létrehozása táblázaton														
INSERT	adatbevitel táblázatba, nézetbe														
REFERENCES	idegen kulccsal való hivatkozás táblázatra														
SELECT	olvasás táblázatból, nézetből, szekvenciából														
UPDATE	módosítás táblázatban, nézetben														
ALL PRIVILEGES	Nem kötelező kulcsszó kombináció. Az objektumra engedélyezhető összes privilégium megadását jelenti.														
oszlop	Csak az itt felsorolt oszlop(ok)ra érvényes a jogosultság. Kizárólag az INSERT, UPDATE és REFERENCES jogosultsággal használható.														
tulajdonos	Az objektum tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk.														
táblázat, nézet, szekvencia	Annak az objektumnak a neve, amelyikre a jogosultságot adjuk.														

Kulcsszó vagy paraméter	Jelentése
felhasználó	Annak a felhasználónak (vagy csoportnak) az azonosítója, aki a jogosultságot kapja, vagy akitől visszavonjuk. Legalább egy <i>felhasználót</i> vagy a <i>PUBLIC</i> felhasználói csoportot meg kell adnunk.
PUBLIC	Nem kötelező kulcsszó. Az adatbázis minden felhasználója megkapja a jogosultságot.
WITH GRANT OPTION	Nem kötelező kulcsszó kombináció. A felhasználó az így kapott jogosultságokat a <i>GRANT</i> utasítással továbbadhatja másoknak.
CASCADE CONSTRAINTS	Nem kötelező kulcsszó kombináció. Törli az összes idegen kulcsot, amit a visszavont <i>REFERENCES</i> jogosultsággal adtak ki. Kötelező megadni, ha <i>REFERENCES</i> vagy <i>ALL PRIVILEGES</i> jogosultságot vonunk vissza, és a táblázatra definiáltak idegen kulcsot a visszavont jogosultság alapján.

Megjegyzések:

- A jogosultság az utasítás kiadásával azonnal hatályba lép illetve érvénytelenné válik.
- *PUBLIC* speciális felhasználói csoport:
 - Ha a *PUBLIC* felhasználói csoportnak adjuk a jogosultságot, akkor az adatbázis minden felhasználója megkapja ezt a jogosultságot.
 - Ha a *PUBLIC* felhasználói csoporttól vonjuk vissza a jogosultságot, akkor az adatbázis minden felhasználójától visszavonjuk. Ennek ellenére azok a felhasználók, akik ezzel a jogosultsággal más úton is rendelkeztek (például azonosítójukra közvetlenül kiadott *GRANT* alapján) továbbra is elvégezhetik a műveletet.
- Ha egy jogosultságot több módon is megkaptunk (például az azonosítónkra és *PUBLIC* révén is), akkor az egyik visszavonása esetén a másik módon szerzett jogosultság érvényben marad (lásd 2. példa).
- A saját objektumainkra minden jogosultságunk megvan. Ezért ezekre nem adhatunk ki és nem is vonhatunk vissza jogosultságot.
- Nézetre csak akkor adhatunk jogosultságot, ha a nézet összes bázis táblázatán rendelkezünk a kiadandó jogosultsággal.
- Azt, hogy milyen objektumokra milyen jogosultságaink vannak, illetve milyen jogosultságokat kiknek adtunk ki, a megfelelő katalógus relációkból tudhatjuk meg.
- Ha valakinek *INDEX* vagy *REFERENCES* jogosultságot adtunk egy táblázatunkra és az illető idegen kulcsot, vagy egyedi indexet épít rá, és nem ellenőrizzük, hogy milyen adatokat visz be ebbe, illetve az erre hivatkozó táblázatokba, akkor problémánk lehet a táblázat adatainak módosításakor. (Pl. már bevitt egy olyan rekordot a hivatkozó táblájába, ami miatt nem törölhetjük a hivatkozott adatot a saját táblánkból, mert létezik rá hivatkozás).
- A *GRANT* utasítással rendszer privilégiumokat (pl. objektumok létrehozása) is adhatunk egyes felhasználóknak illetve felhasználói csoportoknak. Ezt az utasítást általában csak az adatbázis-felügyelő adhatja ki. Az így adott jogosultságokat *REVOKE*-kal vissza is vonhatja.
- Minden jogosultság és felhasználó legfőbb egyszer szerepelhet az utasításban.

Példák:

1. Az *SQL00* felhasználónak minden, a többieknek csak olvasási jogosultságot adunk a *HALLGATO* táblázatunkra.

Utasítások:

```
grant all privileges on hallgato to sql00;  
grant select on hallgato to public;
```

Eredmény: *SQL00* mindent csinálhat a *HALLGATO* táblázatunkon, a többiek pedig olvashatják az adatait.

2. Az *SQL00* felhasználótól visszavonunk minden jogosultságot.

Utasítás:

```
revoke all privileges on hallgato from sql00;
```

Eredmény: *PUBLIC*-től az olvasásra a jogosultságot nem vontuk vissza.

Ezért *SQL00* továbbra is olvashatja a *HALLGATO* táblázatunkat a *PUBLIC* jogosultság révén, de más műveletet nem végezhet rajta.

5.6.3. Adatkezelő utasítások

Az adatkezelő (Data Manipulation Language, DML) utasítások segítségével már létező, meghatározott típusú adatbázis objektumokból

- választhatunk ki sorokat (*SELECT*),
- írhatunk be új sorokat (*INSERT*),
- módosíthatjuk meglévő sorok oszlopait (*UPDATE*),
- törölhetünk meglévő sorokat (*DELETE*).

5.6.3.1. Az adatkezelő utasítások közös jellemzői

Egy adatkezelő utasítás mindig egy vagy több, az utasításban egyértelműen meghatározott adatbázis objektumra (általában táblázatra vagy nézetre) vonatkozó műveletet jelent és azoknak az utasítás paramétereitől és a tényleges adatoktól függően 0, 1, vagy több sorát érinti. Annak a *SELECT* utasításnak az eredménye, amelyik kiválasztja a *MITHALLGAT* táblázatból azokat a hallgatókat, akik az *SQL* kódú tantárgyat felvették, a táblázat adataitól függően 0,1 vagy akárhány sor lehet.

Az utasításokat csak az hajthatja végre, akinek az abban szereplő objektum(ok)on az adott utasítás elvégzésére jogosultsága van. Az objektum tulajdonosa minden műveletet elvégezhet a saját objektumain. A többi felhasználó (a legegyszerűbb személyi számítógépes rendszerek, illetve az adatbázis-felügyelő kivételével) csak olyan műveleteket végezhet más objektumán, amire a tulajdonostól a *GRANT* utasítással (ld. 5.6.2.4. pont) engedélyt kapott. Így például lehetséges, hogy az *SQL30* azonosítóval rendelkező felhasználó az *SQL00* felhasználó *HALLGATO* táblázatának adatait olvashatja, új adatokat vihet be, módosíthatja bármely hallgatónál, hogy melyik évfolyamra jár, de más adatot nem változtathat meg és nem is törölhet hallgatókat onnan.

Az adatkezelő utasításokat kiadhatjuk párbeszédesen és programnyelvből is a fogadó nyelv utasításai közé beépítve. A kétfajta utasítás formátuma a legtöbb esetben azonos.

Néhány speciális utasítás illetve utasításváltozat azonban csak programból használható. Ezeket az 5.7. pontban ismertetjük.

A szintaktikailag helyes utasítás(oka)t a rendszer először átalakítja relációs műveletek sorozatává. Utána ellenőrzi, hogy a kérelmezőnek van-e jogosultsága ezek mindegyikének elvégzésére. Amennyiben igen, úgy az adatok mennyiségéről és eloszlásáról az adatbázis katalógusából rendelkezésre álló információk alapján megállapítja a (szerinte) optimális műveleti sorrendet és adatelérési utakat, és generálja az operációs rendszer számára az ehhez szükséges utasításokat. Nagy adatbázisoknál az elérési út optimalizálása igen fontos, ezen múlik a rendszer hatékonysága. Erről részletesen beszélünk a 2.7. alfejezetben és néhány gyakorlati problémát és megoldásukat ismertetjük a 8.5. alfejezetben.

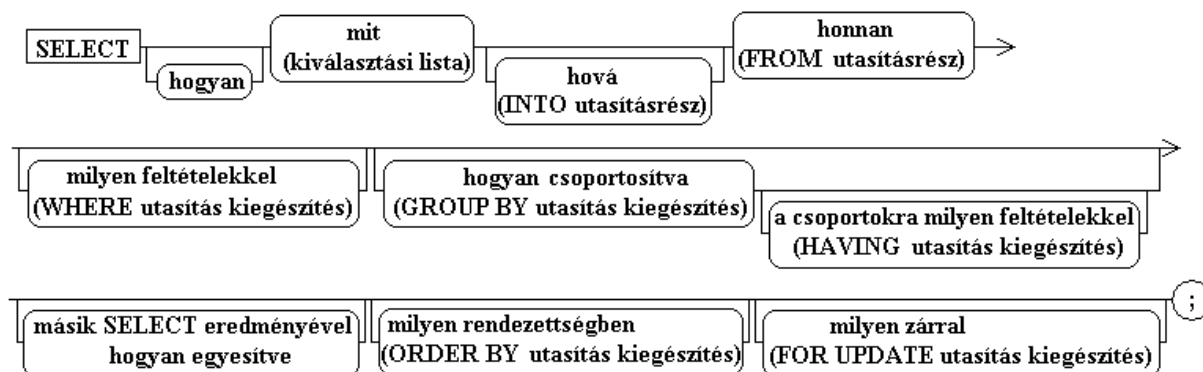
Az utasítás elvégzéséről (vagy eredménytelenségéről) illetve a végrehajtás folyamán esetleg észlelt hibáról a rendszer értesítést küld a felhasználónak. Párbeszédés üzemmódban ez egy szöveges magyarázattal kiegészített hibakód. Programból való alkalmazáskor egy előre definiált programterületre helyezi ezeket az információkat.

Ha többen dolgoznak egyidejűleg ugyanazokkal az adatokkal, akkor a rendszer automatikusan gondoskodik arról, hogy az adatok konzisztensek maradjanak. Az ezt biztosító zármechanizmust a 8.3. alfejezetben ismertetjük részletesebben.

5.6.3.2. Sorok kiválasztása (SELECT)

Funkciója: Adatok kiválasztása táblázatból, nézetből.

Általános szintaxisa:



Magyarázat:

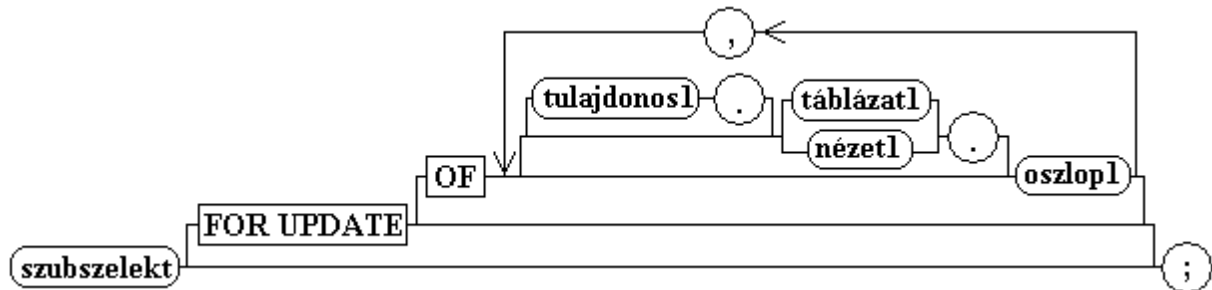
A SELECT utasítás a legösszetettebb és a leggyakrabban használt SQL utasítás. Ezért először egy általános áttekintést adunk az utasításban rejlő lehetőségekről, majd ezután ismertetjük a legfontosabb klauzulákat.

A SELECT utasításban megadhatjuk, hogy

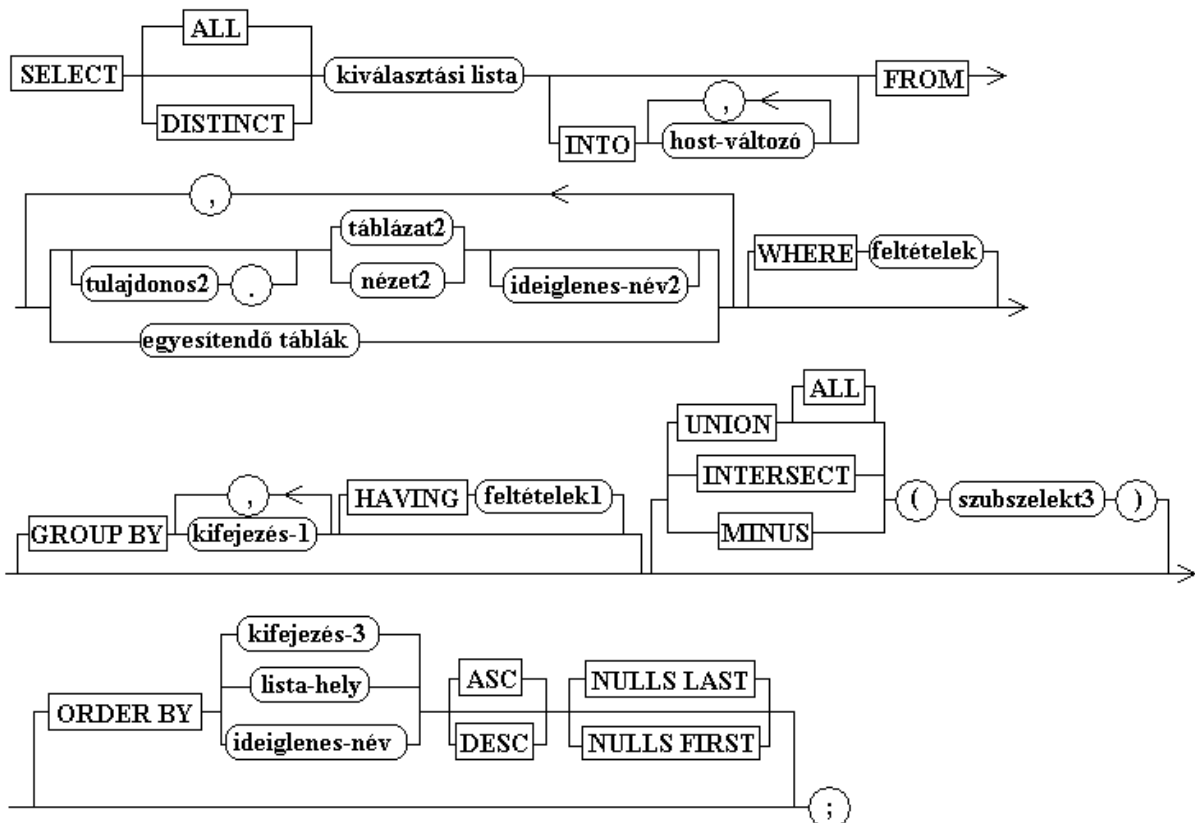
- *hogyan* (teljesen azonos rekordokat csak egyszer, vagy ahányszor előfordulnak),
- *mit* (milyen oszlopokat, kifejezéseket, ez a *kiválasztási lista*),
- *honnan* (milyen táblázatokból, nézetekből, ez a *FROM utasításrész*),
- *hová* (milyen programváltozókba, ez a csak programból használható *INTO utasításrész*),
- milyen feltételek teljesülése esetén (ez a WHERE utasítás kiegészítés),
- milyen csoportosításban, (ez a GROUP BY utasítás kiegészítés),
- az egyes csoportokra milyen feltételek teljesülése esetén, (ez a HAVING utasítás kiegészítés),

- *más szelekt eredményével miként egyesítve* (mindegyik *SELECT* rekordjait, csak a közös rekordokat vagy csupán azokat, amelyeket csak az első *SELECT* szolgáltat),
- *milyen rendezettségben*, (ez az *ORDER BY* utasítás kiegészítés) válasszuk ki a kívánt adatokat.

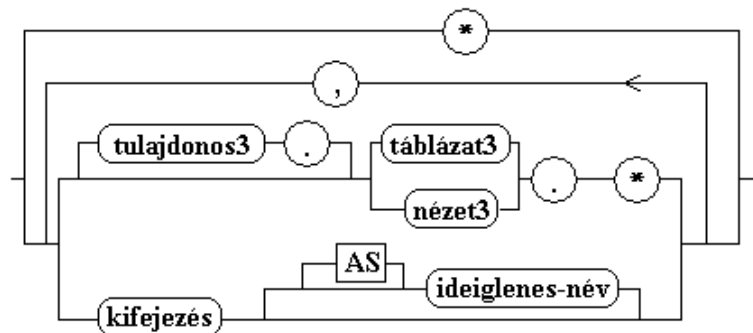
Egyszerűsített szintaxisa:



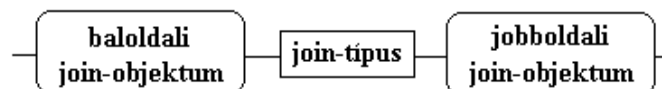
ahol szubszelekt egyszerűsített szintaxisa:



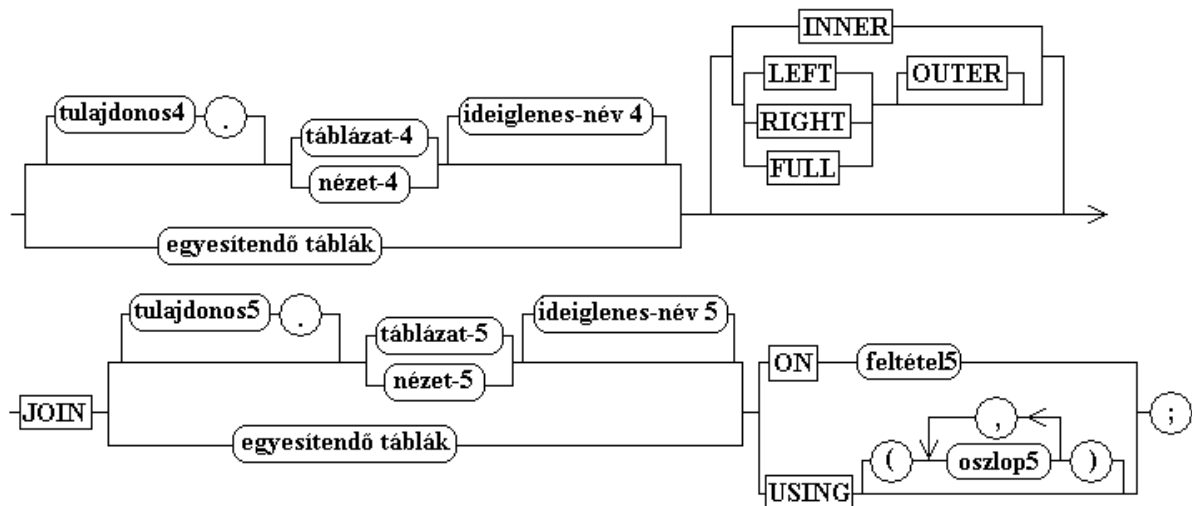
a kiválasztási lista egyszerűsített kifejtése:



az egyesítendő táblák általános szintaxisa:



az egyesítendő táblák részletes, de egyszerűsített szintaxisa:



Jogosultság:

- Táblázatból akkor választhatunk ki sorokat, ha az a saját táblázatunk, vagy van rá *SELECT* privilégiumunk.
- Nézeteken keresztül akkor választhatunk ki a bázis táblázatából sorokat, ha a nézet tulajdonosának van a bázis táblázatra *SELECT* privilégiuma. Ezen kívül nekünk is kell rendelkezünk *SELECT* privilégiummal a nézetre, ha az nem a sajátunk.
- Ha adatbázis-felügyelői privilégiumunk van, akkor bármelyik táblából, nézetből kiválaszthatunk sorokat.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
SELECT	Kötelező kulcsszó. Megadja, milyen utasítás kerül végrehajtásra.
DISTINCT	Elhagyható kulcsszó. Jelzi, hogy ha a feltételek alapján teljesen azonos sorokat kapnánk eredményül, akkor azokból az utasítás csak egy rekordot szolgáltatson vissza eredményül (lásd 2. példa.)

Kulcsszó vagy paraméter	Jelentése														
ALL	Elhagyható kulcsszó. Jelzi, hogy az utasítás a feltételeknek megfelelő összes rekordot adja vissza eredményül. Az azonos rekordokat többször megkapjuk az eredmény-listában. Ez az alapértelmezés.														
kiválasztási lista	Az itt felsorolt elemek mindegyikét kiválasztja a feltételeknek megfelelő rekordokból. Az egyes elemek a következők: <table> <tr> <th>Jelölés</th><th>Jelentése</th></tr> <tr> <td>*</td><td> Az összes oszlop megjelenik az eredményben. Ha ez a kiválasztási-lista egyetlen eleme, akkor a FROM klauzulában felsorolt összes táblázat/nézet minden oszlopa része lesz az eredménynek. Ha egy táblázat vagy nézet neve után áll, akkor csak ennek az objektumnak az összes oszlopa kerül be az eredménybe. (Ezekhez természetesen még további elemeket is felvehetünk a kiválasztási-listába.) </td></tr> <tr> <td>tulajdonos3</td><td>A <i>táblázat3</i> vagy <i>nézet3</i> tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázatnév vagy nézetnév közé pontot (.) kell tennünk.</td></tr> <tr> <td>táblázat3 nézet3</td><td> Ebből a táblázatból vagy nézetből választjuk ki az összes oszlopot. Ha egy objektumnak csak egyes oszlopait akarjuk kijelölni, akkor azokat <i>kifejezés</i>-ként, egyenként kell felsorolnunk. </td></tr> <tr> <td>kifejezés</td><td>Általában egy oszlop vagy egy állandó (literál). Pontos leírását az SQL kifejezéseknél találjuk. (ld. 5.3.8. pont) Az eredményben a kifejezések a felsorolás sorrendjében jelennek meg. Interaktív üzemmódban a fejlécben alapértelmezésként az oszlop neve jelenik meg. Ha a kifejezés nem oszlop és nem adtunk neki ideiglenes nevet, akkor a rendszer generálja a felíratot.</td></tr> <tr> <td>AS</td><td>Az olvasást megkönnyítő, elhagyható kulcsszó.</td></tr> <tr> <td>ideiglenes- név</td><td>Ebben a szelekt utasításban ez lesz az adott kifejezés neve. Ez jelenik meg az eredmény fejlécében és használhatjuk ezt a nevet is az ORDER BY klauzulában. Célszerű megadnunk, ha <i>kifejezés</i> nem egy táblázat/nézet oszlop (lásd 3. példa).</td></tr> </table>	Jelölés	Jelentése	*	Az összes oszlop megjelenik az eredményben. Ha ez a kiválasztási-lista egyetlen eleme, akkor a FROM klauzulában felsorolt összes táblázat/nézet minden oszlopa része lesz az eredménynek. Ha egy táblázat vagy nézet neve után áll, akkor csak ennek az objektumnak az összes oszlopa kerül be az eredménybe. (Ezekhez természetesen még további elemeket is felvehetünk a kiválasztási-listába.)	tulajdonos3	A <i>táblázat3</i> vagy <i>nézet3</i> tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázatnév vagy nézetnév közé pontot (.) kell tennünk.	táblázat3 nézet3	Ebből a táblázatból vagy nézetből választjuk ki az összes oszlopot. Ha egy objektumnak csak egyes oszlopait akarjuk kijelölni, akkor azokat <i>kifejezés</i>-ként, egyenként kell felsorolnunk.	kifejezés	Általában egy oszlop vagy egy állandó (literál). Pontos leírását az SQL kifejezéseknél találjuk. (ld. 5.3.8. pont) Az eredményben a kifejezések a felsorolás sorrendjében jelennek meg. Interaktív üzemmódban a fejlécben alapértelmezésként az oszlop neve jelenik meg. Ha a kifejezés nem oszlop és nem adtunk neki ideiglenes nevet, akkor a rendszer generálja a felíratot.	AS	Az olvasást megkönnyítő, elhagyható kulcsszó.	ideiglenes- név	Ebben a szelekt utasításban ez lesz az adott kifejezés neve. Ez jelenik meg az eredmény fejlécében és használhatjuk ezt a nevet is az ORDER BY klauzulában. Célszerű megadnunk, ha <i>kifejezés</i> nem egy táblázat/nézet oszlop (lásd 3. példa).
Jelölés	Jelentése														
*	Az összes oszlop megjelenik az eredményben. Ha ez a kiválasztási-lista egyetlen eleme, akkor a FROM klauzulában felsorolt összes táblázat/nézet minden oszlopa része lesz az eredménynek. Ha egy táblázat vagy nézet neve után áll, akkor csak ennek az objektumnak az összes oszlopa kerül be az eredménybe. (Ezekhez természetesen még további elemeket is felvehetünk a kiválasztási-listába.)														
tulajdonos3	A <i>táblázat3</i> vagy <i>nézet3</i> tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázatnév vagy nézetnév közé pontot (.) kell tennünk.														
táblázat3 nézet3	Ebből a táblázatból vagy nézetből választjuk ki az összes oszlopot. Ha egy objektumnak csak egyes oszlopait akarjuk kijelölni, akkor azokat <i>kifejezés</i>-ként, egyenként kell felsorolnunk.														
kifejezés	Általában egy oszlop vagy egy állandó (literál). Pontos leírását az SQL kifejezéseknél találjuk. (ld. 5.3.8. pont) Az eredményben a kifejezések a felsorolás sorrendjében jelennek meg. Interaktív üzemmódban a fejlécben alapértelmezésként az oszlop neve jelenik meg. Ha a kifejezés nem oszlop és nem adtunk neki ideiglenes nevet, akkor a rendszer generálja a felíratot.														
AS	Az olvasást megkönnyítő, elhagyható kulcsszó.														
ideiglenes- név	Ebben a szelekt utasításban ez lesz az adott kifejezés neve. Ez jelenik meg az eredmény fejlécében és használhatjuk ezt a nevet is az ORDER BY klauzulában. Célszerű megadnunk, ha <i>kifejezés</i> nem egy táblázat/nézet oszlop (lásd 3. példa).														
INTO	Csak programból használható, ott viszont kötelező. Jelzi, hogy a kiválasztás eredménye <i>host változó(k)ba</i> (ld. 5.7.3. pont) kerül.														
host-változó	A programnak ezen változóiba kerül a kiválasztott sor. A kiválasztási lista első elemének tartalma lesz az első host változó értéke, ... , <i>n</i> -edik eleme az <i>n</i> -edik host változó értéke.														
FROM	Kötelező kulcsszó. Jelzi a FROM utasításrész kezdetét. Ebben adjuk meg, honnan kell az eredménysorokat kiválasztani.														

Kulcsszó vagy paraméter	Jelentése
tulajdonos2	A <i>táblázat2</i> vagy <i>nézet2</i> tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázatnév vagy nézetnév közé pontot (.) kell tennünk.
táblázat2 nézet2	Ebből/ezekből kell a feltételeknek megfelelő sorokat kiválasztani (lásd 1. és 2. példa).
ideiglenes-név2	Ebben a szelekt utasításban ez lesz az adott táblázat/nézet neve. Ezt használjuk az oszlopok, kifejezések minősítéséhez. Kötelező megadnunk olyan objektumokra, amelyek többször szerepelnek a FROM klauzulában (lásd 3. példa).
egyesítendő táblák	Az itt felsorolt táblázatokat, nézeteket, egyesített objektumokat kell a megadott módon egyesíteni.
JOIN	Elhagyható kulcsszó. Ez jelzi, hogy a két oldalán lévő objektumok egyesítéséből kell majd a feltételeknek megfelelő sorokat kiválasztani.
tulajdonos4	Az egyesítésben balról résztvevő <i>táblázat-4</i> vagy <i>nézet-4</i> tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázatnév vagy nézetnév közé pontot (.) kell tennünk.
táblázat-4 nézet-4	A bal oldali egyesítendő táblázat/nézet neve.
tulajdonos5	Az egyesítésben jobbról résztvevő <i>táblázat-5</i> vagy <i>nézet-5</i> tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázatnév vagy nézetnév közé pontot (.) kell tennünk.
táblázat-5 nézet-5	A jobb oldali egyesítendő táblázat/nézet neve.
INNER LEFT RIGHT FULL	Elhagyható kulcsszavak. Csak egy adható meg közülük. Jelzi, milyen típusú egyesítést kell végrehajtani. Ha nem adjuk meg, akkor alapértelmezésként belső egyesítés (<i>inner join</i>) kerül végrehajtásra.
OUTER	Megértést megkönnyítő, elhagyható kulcsszó. Csak <i>LEFT</i> , <i>RIGHT</i> , <i>FULL</i> -al együtt adható meg.
ON	Elhagyható kulcsszó. Megadása azt jelzi, hogy a két oldalán lévő objektumok egyesítését <i>feltétel3</i> alapján kell elvégezni (lásd 6b. példa).
feltétel5	Ennek alapján hajtódik végre az egyesítés. A két join-objektumnak csak a feltétel3-nak megfelelő sorai kerülnek be az egyesített (ideiglenes) táblázatba (lásd 6b. példa).
USING	Elhagyható kulcsszó. Megadása azt jelzi, hogy a két oldalán lévő objektumok egyesítését az oszlop3 oszlop(ok) alapján kell elvégezni (lásd 6c. példa).
oszlop5	Csak ez(ek) az oszlop(ok) szerepel(nek) mindkét objektumból a <i>join-feltételben</i> (lásd 6c. példa).
WHERE	Elhagyható kulcsszó. Mutatja a WHERE utasítás kiegészítés kezdetét. Jelzi, hogy csak az utána következő feltétel(ek)nek megfelelő rekordok kerülnek kiválasztásra.
feltételek	Leírásukat az SQL feltételeknél találjuk (ld. 5.3.4.5. pont).

Kulcsszó vagy paraméter	Jelentése	
GROUP BY	Nem kötelező kulcsszavak. Mutatja a GROUP BY utasítás kiegészítés kezdetét. Ha megadjuk, akkor a kiválasztási-lista csoport függvénye(i) a kiválasztott sorokból a klauzulában megadott kifejezés(ek) (általában oszlopok) minden egyes értékéhez egy, a függvény argumentumának a csoportra jellemző értékét rendeli(k) hozzá (lásd 4. és 5. példa).	
kifejezés-1	Az utasítás által képzett csoportokra az jellemző, hogy egy csoporton belül az összes kiválasztott sorban ennek a kifejezésnek az értéke ugyanaz. Egy értékhez csak egy csoport tartozik.	
HAVING	Mutatja a HAVING utasítás kiegészítés kezdetét. Az egyes csoportokat csak akkor jeleníti meg, ha az itt megadott feltétel az egész csoportra teljesül.	
feltételek1	Az itt megadott feltételnek kell az egész csoportra teljesülni (lásd 5. példa).	
UNION	Elhagyható kulcsszó. Megadása jelzi, hogy a baloldalán lévő szubszelekt által kiválasztott sorokat az <i>UNION</i> művelet szabályai szerint (ld. 5.3.4.4. pont) válogatjuk össze a <i>szubszelekt3</i> által kiválasztott adatokkal.	
UNION ALL	Elhagyható kulcsszavak. Megadásuk jelzi, hogy a baloldalán lévő szubszelekt által kiválasztott sorokat az <i>UNION ALL</i> művelet szabályai szerint (ld. 5.3.4.4. pont) válogatjuk össze a szubszelekt3 által kiválasztott adatokkal (lásd 7 példa).	
INTERSECT	Elhagyható kulcsszó. Megadása jelzi, hogy a baloldalán lévő szubszelekt által kiválasztott sorokat az <i>INTERSECT</i> művelet szabályai szerint (ld. 5.3.4.4. pont) válogatjuk össze a szubszelekt3 által kiválasztott adatokkal (lásd 8. példa).	
MINUS	Elhagyható kulcsszó. Megadása jelzi, hogy a baloldalán lévő szubszelekt által kiválasztott sorokat a <i>MINUS</i> művelet szabályai szerint (ld. 5.3.4.4. pont) válogatjuk össze a <i>szubszelekt3</i> által kiválasztott adatokkal.	
szubszelekt3	Az általa kiválasztott rekordokat egyesíti a megadott művelet szabályai szerint a halmazoperátor bal oldalán lévő szubszelekt rekordjaival (lásd 7. és 8. példa).	
ORDER BY	Elhagyható kulcsszó. Mutatja az ORDER BY utasítás kiegészítés kezdetét. Megadása jelzi, hogy a kiválasztott sorokat rendezve jelenítjük meg. Rendezési szempontként a következőket adhatjuk meg:	
	jelölés	jelentése
	kifejezés-3	Az eredménysorok rendezése ennek a kifejezésnek az értéke szerint történik. A gyakorlatban ez rendszerint a kiválasztási lista egy oszlopa.
	lista-hely	Értéke: <i>n</i> (egész szám) A rendezés a kiválasztási lista <i>n</i> -edik elemének értéke szerint történik (lásd 4. példa).
	ideiglenes-név	A rendezés a kiválasztási listában ezzel az ideiglenes névvel definiált kifejezésnek az értéke szerint történik (lásd 3. példa).

Kulcsszó vagy paraméter	Jelentése
ASC	Elhagyható kulcsszó. Megadása jelzi, hogy az eredmény rendezése növekvő sorrendben történik. Ez az alapértelmezés.
DESC	Elhagyható kulcsszó. Megadása jelzi, hogy az eredmény rendezése csökkenő sorrendben történik.
NULLS FIRST	Elhagyható kulcsszavak. Megadásuk jelzi, hogy a <i>NULL</i> értékek a rendezett lista elejére kerülnek. Csökkenő sorrendnél ez az alapértelmezés.
NULLS LAST	Elhagyható kulcsszavak. Megadásuk jelzi, hogy a <i>NULL</i> értékek a rendezett lista végére kerülnek. Növekvő sorrendnél ez az alapértelmezés.
FOR UPDATE	Nem kötelező kulcsszavak. Mutatja a <i>FOR UPDATE</i> utasítás kiegészítés kezdetét. Megadásuk jelzi, hogy a <i>SELECT</i> utasítás által kiválasztott sorokat más nem módosíthatja és nem zárhatja el a tranzakciónk befejeztéig. Általában kurzor definiálásánál használják.
OF	Elhagyható kulcsszó. Megadása jelzi, hogy csak az utána megadott táblázatok/nézetek sorainak zárolását nem engedjük meg másoknak.
tulajdonos1	A táblázat1 vagy nézet1 tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk. A tulajdonos és a táblázat vagy nézet neve közé pontot (.) kell tennünk.
táblázat1 nézet1	Csak az ezekből az objektumokból kiválasztott sorok zárolását nem engedjük meg másoknak.
oszlop1	Formai okokból a <i>FOR UPDATE</i> klauzulában megadott objektumok mindegyikéhez az objektum egy oszlopát meg kell adnunk.

Megjegyzések:

- Az egyes utasításrészek és kiegészítések kezdetét a rá jellemző kulcsszó (szavak) jelzi(k). Végét a következő utasításrész vagy kiegészítés kulcsszava(i) illetve az utasítás/szubszelekt/beágyazott szelekt vége jelenti.
- Ha sem *ALL*-t, sem *DISTINCT*-et nem adunk meg, akkor a *SELECT* utasítás az összes azonos sort visszaadja eredményül. A *SELECT* kiválasztási lista ... és a *SELECT ALL* kiválasztási lista ... utasítások hatása azonos.
- A kiválasztási listában a kifejezést csak akkor minősíthetjük a tulajdonos nevével, ha a kifejezést tartalmazó elemek táblázatát/nézetét a *FROM* klauzulában a tulajdonos nevével együtt adtuk meg.
- Ha a *FROM* klauzulában két táblázat/nézet áll egymás után és nem szerepel köztük a *JOIN* művelet, akkor ennek az eredménye a két objektum keresztszorzata lesz (ld. 4.5.5. pont). (A gyakorlatban ez szinte soha nem jön létre, mert a keresztszorzat sorait a *WHERE* feltételben korlátozzuk és így biztosítjuk a *JOIN* létrejöttét. Egyes rendszerekben csak az utóbbi módon, a *WHERE* feltételben tudjuk a *JOIN* műveletét megvalósítani.)
- Egy szubszelektben – elvileg – akárhány objektumot egyesíthetünk, de csak egy külső egyesítést definiálhatunk. A join végrehajtása általában balról jobbra történik. Ügyeljünk azonban arra, hogy az egyesítés jelentősen lelassíthatja a *SELECT* műveletet.

- Ha több táblázatból választunk ki elemeket, akkor kötelező a *táblázat/nézet* névvel minősíteni azokat az oszlopneveket, amelyek a *FROM* utasítás rész több objektumában is előfordulnak. Kötelező még a *tulajdonosnév*vel való minősítés is, ha a *FROM* klauzulában azonos nevű táblázatok/nézetek szerepelnek (lásd 3. és 11. példa).
- Kötelező az *ideiglenes táblázat* név megadása akkor, ha ugyanaz a táblázat többször szerepel a *FROM* klauzulában (lásd 3. példa).
- A közös oszlopokra történő egyesítést (*join*, *outer join*) egyes rendszerekben megadhatjuk a *FROM* klauzulában az egyesítendő táblák és az egyesítés típusának megadásával. Az oszlopok alapján végrehajtandó egyesítést megfogalmazhatjuk a *WHERE* klauzulában is a *join oszlopokra* megadott egyenlőség formájában is (lásd 6. példa).
- Ügyeljünk arra, hogy a természetes (alapértelmezett) *join* nem adja meg azokat a rekordokat, amelyekben bármelyik *join* oszlop értéke *NULL* érték.
- Ha az egyesítésnél a táblázatok/nézetek sorrendje kétséges lehet, akkor zárójelekkel tegyük egyértelművé a végrehajtás sorrendjét.
- Ha az egyesítéshez *JOIN ... USING* kiegészítést adunk meg, akkor az egyesítésre szolgáló oszlopok neve mindkét objektumban azonos kell legyen. Csak a *USING* után felsorolt azonos nevű oszlopok vesznek páronként részt az egyesítésben. Ezeket az oszlopneveket *nem szabad* minősíteni.
- Ha nem adunk meg *WHERE* klauzulát, akkor a *FROM* utasítás rész által meghatározott (ideiglenes) objektum minden sora kiválasztásra kerül.
- Ha megadjuk a *WHERE* klauzulát, akkor a *FROM* utasítás rész által meghatározott (ideiglenes) objektumnak csak azok a sorai kerülnek kiválasztásra, amelyek a *WHERE*-ben meghatározott feltételeknek megfelelnek.
- A *WHERE* utasítás kiegészítés feltételeiben a *FROM* utasítás kiegészítés elemeinek bármelyik oszlopa szerepelhet, függetlenül attól, hogy az benne van-e a kiválasztási listában.
- Ha megadtuk a *GROUP BY* klauzulát, akkor a kiválasztási lista csak az alábbi típusú elemekből állhat:
 - állandó (literál),
 - csoport függvény,
 - olyan kifejezés, amelyik a *GROUP BY* klauzulában szerepel,
 - olyan kifejezés, amelyet a fent felsoroltakból képezhetünk, és a csoport minden kiválasztott elemére ugyanazt az értéket adja.
- Ha a *GROUP BY* használatkor egy csoporton belül egy sor sem kerül kiválasztásra, vagy az összes kiválasztott sorban a kiválasztási listában levő függvény argumentuma *NULL* érték, akkor a függvény értéke az adott csoportra a *NULL* érték lesz.
- A *GROUP BY* klauzulában a csoportosításra szolgáló kifejezés a utasítás kiegészítés elemeinek bármelyik oszlopát tartalmazhatja, függetlenül attól, hogy az szerepel-e a kiválasztási listában.
- Ha nem adunk meg *ORDER BY* klauzulát, akkor az eredmény várható sorrendjéről nem tudunk mit mondani. Előfordulhat, hogy ugyanaz a lekérdezés megismételve más sorrendben szolgáltatja a rekordokat.
- Ha több rendezési szempontot adunk meg, akkor a rendszer először az elsőként megadott szempont szerint rendezi az eredményt, majd ezen belül a másodiknak megadott szerint és így tovább (lásd 1. és 7. példa).

- *ORDER BY lista_hely*-et akkor célszerű megadni, ha a rendező kifejezés neve nagyon hosszú, vagy ha halmaz operátor(ok)kal összekapcsolt *szubszelekt*-ek végeredményét akarjuk rendezni (lásd 7. példa).
- Nem adhatunk meg *ORDER BY*-t a *WHERE* utasítás kiegészítés beágyazott szelektjében.
- Ha *SELECT DISTINCT ...* -et adunk meg, akkor a rendezési oszlop(ok)nak szerepelni kell a *kiválasztási listában* is.
- Rendezéskor az összehasonlítási sorrend:
 - numerikus kifejezéseknél a számok előjeles értéke,
 - dátum/időpont típusú kifejezéseknél a dátum majd az időpont,
 - Jelsorozatoknál az adatbázis-kezelő rendszer kódtáblázata (ASCII vagy EBCDIC kód) szerinti sorrend. ASCII karaktereknél a növekvő sorrend: szóköz, speciális karakterek, számjegy, nagy betűk, kis betűk, EBCDIC-nél: szóköz, speciális karakterek, kis betűk, nagy betűk, számjegy.
- A *NULL* értékek alapértelmezett rendezésnél általában a rendezett lista végére kerülnek.
- A *NULLS FIRST*, *NULLS LAST* kulcsszavakat nem mindegyik rendszer ismeri. Ilyenkor az alapértelmezéstől függ, hogy a *NULL* értékek a rendezett halmaz elejére vagy a végére kerülnek.
- Ügyeljünk arra, hogy ha numerikus vagy dátum/időpont típusú kifejezéseket az áttekinthetőbb megjelenítés érdekében karakter formába alakítunk át, akkor az összehasonlítás is karakter formátum szerint történik.
- A *SELECT ... INTO* formát csak programban használhatjuk és ott is csak akkor, ha az utasítás legfeljebb egy sort választ ki.
- Egy *SELECT* utasításban – elvileg – akárhány szubszelekt eredményét összevonhatjuk a *UNION*, *UNION ALL*, *INTERSECT*, *MINUS* halmaz operátorok segítségével. A halmaz műveletek végrehajtása balról jobbra történik.
- Ha halmaz operátor segítségével több szubszelekt eredményét vonjuk össze, akkor csak az utolsó szubszelekt után adhatjuk meg az *ORDER BY* klauzulát.
- A halmaz operátorokkal összekapcsolt szubszelekt kiválasztási listái azonos számú és azonos típusú elemeket kell tartalmazzanak azonos sorrendben rendezve. A lista nem tartalmazhat *LONG* adattípusú elemet (ld. 5.4.5. pont). (Az egyes elemek nevei, hosszai különbözhetnek az egyes szubszelektben.)
- A *FOR UPDATE* utasítás kiegészítés megadásával biztosítjuk, hogy a tranzakciónk befejeztéig mások ne módosíthassák és ne zárhassák el az általunk kiválasztott rekordokat. Ennek használata azonban adatbázis-kezelő rendszerenként eltérő módon korlátozott.
- Ha csak meghatározott táblázat(ok)ból/nézet(ek)ből kiválasztott sorok zárolását akarjuk megtiltani, akkor az *OF* utáni felsorolás után meg kell adnunk a táblázat/nézet egy oszlopát is. Ez az adott objektum bármelyik oszlopa lehet.
- Ha a *WHERE* klauzulában *kifejezés* = (*beágyazott szelekt*) (vagy = helyett <, <=, >, >=, <> szerepel), akkor a beágyazott szelekt értelemszerűen csak egyetlen egy oszlopot választhat ki egyetlen egy értékkel. *IN*, *ANY*, *ALL*, *SOME* esetén akárhány értéke lehet a kiválasztott oszlopnak.

Korrelált és nem korrelált beágyazott szelekt

Ha a feltételek beágyazott szelektet tartalmaznak, akkor a *SELECT* utasítás hatékonysága nagy mértékben függhet

- a beágyazott szelekt felsorolásának sorrendjétől,

- hányszor kell az utasítást a *SELECT*-en belül végrehajtani.

Ha a *WHERE* és/vagy *HAVING* utasítás kiegészítés feltételei nem tartalmazznak beágyazott szelektet, akkor az SQL optimalizáló a feltételek felsorolásának sorrendjétől függetlenül határozza meg az optimális elérési utat a katalógus táblázatokban levő statisztika alapján. Beágyazott szelektek esetén azonban ezek kiértékelése a beágyazott szelektek felsorolásának sorrendjében történik. Ezért, ha ezeket *AND* (*ÉS*) operátorral kapcsoljuk össze, akkor célszerű először azokat felsorolni, amelyek nagy valószínűséggel *FALSE* (*HAMIS*) eredményt adnak. Ebben az esetben ugyanis az ezután álló beágyazott szelektek végrehajtása elmarad. Hasonló módon, *OR* (*VAGY*) operátorral összekapcsolt beágyazott szelekteknel az(oka)t tegyük előbbre a felsorolásban, amely(ek) nagy valószínűséggel fognak *TRUE* (*IGAZ*) eredményt adni.

Ha a *WHERE* és/vagy *HAVING* utasítás kiegészítés feltételei tartalmazznak beágyazott szelektet, akkor annak formájától függ, hogy az egyetlen egyszer kerül csak végrehajtásra (*nem korrelált beágyazott szelekt*), vagy pedig az egyéb feltételeknek megfelelő rekordok minden egyes sorára (*korrelált beágyazott szelekt*). Nyilvánvaló, hogy az utóbbi végrehajtási ideje sokkal hosszabb lesz. A kétféle beágyazott szelektet láthatjuk a 9. és 10. példákban. Amennyiben módunk van rá, akkor igyekezzünk a feladatot nem korrelált beágyazott szelekt formájában megfogalmazni.

A beágyazott szelekt helyett gyakran megadhatjuk az SQL utasítást táblázatok egyesítéseként is, és fordítva, join helyett gyakran használhatunk beágyazott szelektet (lásd 11. példa). Hogy mikor melyik hatékonyabb, arra nincs általános szabály.

Példák:

1. Meghatározott oszlopok és sorok rendezett kiválasztása egy táblázatból.

Utasítás:

```
select vnev,knev,szdatum from sql00.hallgato
where szdatum >='1980-01-01' and evf=1
order by vnev,knev;
```

Eredmény: Az *SQL00.HALLGATO* táblázatból kiválasztja ABC sorrendben azon elsőévesek vezetéknévét, keresztnévét és születési dátumát, akik 1980-ban vagy később születtek.

2. Példa a *SELECT* és a *SELECT DISTINCT* közti különbségre:

Utasítás-1:

```
select distinct tmegnev from oktatás order by tmegnev;
```

Eredmény:

```
TMEGNEV
-----
Adatbázisok
ORACLE
```

Utasítás-2:

```
select tmegnev from oktatás order by tmegnev;
```


Eredmény:

TMEGNEV

Adatbázisok

Adatbázisok

Adatbázisok

Adatbázisok

Adatbázisok

ORACLE

ORACLE

ORACLE

3. Példa az ideiglenes táblázatnevek használatára.

Az *ALKALMAZOTT-INF* táblázatból (ld. 4-30. oldalon a 4.11. ábra) ABC sorrendben ki akarjuk választani a dolgozók (*VEZ-NEV*) és a főnökük (*FONOK*) nevét. Ehhez meg kell határoznunk ugyanebből a táblázatból a főnök azonosító kódjával (*FONOK*) azonos alkalmazotti azonosítóhoz (*SZSZAM*) tartozó nevet (*VEZ-NEV* és *SZSZAM* egy másik sorban van). Az utasításban *kötelező* a táblázatok *ideiglenes táblázatnév*-vel való azonosítása, mivel a két táblázat azonos. Kötelező a kiválasztott oszlopok minősítése is, mivel mindkettőből a vezetéknév (*VEZ-NEV*) választjuk ki, de az egyik az alkalmazott neve (*A1.VEZ-NEV*), a másik az ebben levő *A1.FONOK* azonosító értékkel egyenlő azonosítójú *A2.SZSZAM*-hoz tartozó *A2.VEZ-NEV* név, ami a főnök nevét adja. Célszerű a kiválasztott oszlopoknak ideiglenes nevet (*ALK_NEV*, *FONOK_NEV*) adni.

Utasítás:

```
select a1.vez_nev as alk_nev,a2.vez_nev as fonok_nev
from alkalmazott a1, alkalmazott a2
where a1.fonok=a2.szszam;
```

4. Példa a GROUP BY utasítás kiegészítés használatára.

Határozzuk meg a *TANTARGY* táblázatból minden tantárgyra (*TID*) a tantárgy osztályzatainak átlagát, és a tantárgyból elért legnagyobb pontszámot. Az eredményt csökkenő átlag, azon belül *TID* szerint rendezzük.

Utasítás:

```
select tid,avg(osztalyzat) as atlag_osztalyzat
,max(gyakpont) as max_gyakpont from mithallgat
group by tid order by 2 desc, tid;
```

5. Példa a GROUP BY és a HAVING utasítás kiegészítés használatára.

Írassuk ki a maximális és a minimális gyakorlati pontszámokat (*GYAKPONT*) azokra a tantárgyakra, melyeknek azonosítója „DB”-vel kezdődik és az átlag osztályzat 3 fölött van. (Az első feltétel minden sorra vonatkozik, így az a WHERE klauzulába kerül.)

Utasítás:

```
select tid,max(gyakpont) as max_gyakpont,min(gyakpont) as
min_gyakpont from mithallgat where tid like 'DB%'
group by tid having avg(osztalyzat) > 3;
```

6. Példa a különböző egyesítések (join) használatára.

A minta táblázatainkból (ld. 4.2. alfejezet) oktatónkénti listát készítünk, hogy melyik oktató milyen osztályzatokat adott az egyes hallgatóknak. A hallgatók személyiségi jogainak védelmében csak az oktató nevét adjuk meg, a hallgatókat csupán az azonosítójukkal (*HID*) szerepeltetjük a listában. Az ehhez szükséges join műveletet többféleképpen fogalmazhatjuk meg. Néhányat bemutatunk:

Utasítás-a

```
select oktato, hid, osztalyzat
from tantargy, mithallgat where tantargy.tid = mithallgat.tid
order by oktato,hid;
```

Utasítás-b:

```
select oktato, hid, osztalyzat
from tantargy join mithallgat on tantargy.tid = mithallgat.tid
order by oktato,hid;
```

Utasítás-c:

```
select oktato, hid, osztalyzat
from tantargy join mithallgat using (tid)
order by oktato,hid;
```

Eredmény: A három utasítás egyenértékű. Egyik sem adja meg azokat az oktatókat, akiknek egy hallgatója sincsen. Ha lehetnek ilyen esetek és ezeket is meg akarjuk kapni, akkor az egyesítést balról, alapértelmezéssel (*left outer join*) kell végrehajtanunk.

Utasítás_d:

```
select oktato, hid, osztalyzat
from tantargy left outer join mithallgat using.(tid)
order by oktato,hid;
```

Tovább bonyolíthatná a helyzetet, ha számolnunk kellene például azzal is, hogy lehetnek diákok, akik nem létező tárgyból kaptak osztályzatot. (Ilyen nem lehet a MITHALLGAT definíciója miatt.)

Gondot okozhat még, ha egy hallgató több tantárgyból is ugyanattól az oktatótól kapta az osztályzatát. Ebben az esetben a kiválasztási listát ki kell bővítenünk pl. a *tmegnev* oszloppal.

7. Példa az *UNIO* alkalmazására.

Írassuk ki saját és *SQL30 TANTARGY* táblázatából a tantárgyak neveit (*TMEGNEV*) és oktatóit (*OKTATO*). Tüntessük fel azt is, hogy az adat melyik táblázatból származik. Az eredményt rendezzük tantárgyak, majd oktatók szerint. Ezen belül előbb a saját adatunk jelenjen meg.

Utasítás:

```
select tmegnev, oktato,'SAJÁT' as TULAJ from tantargy UNION ALL
select tmegnev, oktato,'SQL.30' as TULAJ from sql30.tantargy
order by 1,2,3;
```

Eredmény:

TMEGNEV	OKTATO	TULAJ
-----	-----	-----
DB2	Quittner	SAJÁT
DB2	Quittner	SQL30
ORACLE	Quittner	SAJÁT
ORACLE	Quittner	SQL30
ORACLE	Valaki	SQL30
ORACLE	XYZ	SAJÁT

8. Példa az *INTERSECT* alkalmazására.

Írassuk ki saját és *SQL30 TANTARGY* táblázatából azon tantárgyak neveit (*TMEGNEV*) és oktatóit, amelyek azonos párosításban mindkét táblázatban szerepelnek. Az eredményt rendezzük tantárgyak, majd oktatók szerint.

Utasítás:

```
select tmegnev, oktato from tantargy INTERSECT
select tmegnev, oktato from sql30.tantargy
order by 1,2;
```

Eredmény:

TMEGNEV	OKTATO
-----	-----
DB2	Quittner
ORACLE	Quittner

9. Példa nem korrelált beágyazott szelektre.

Válasszuk ki azokat a hallgatókat, akik az *12345* azonosítójú hallgatóval azonos évfolyamra járnak.

Utasítás:

```
select * from hallgato
where evf = (select evf from hallgato where hid ='12345');
```

Eredmény:

A beágyazott szelektet a rendszer csak egyszer számolja ki. Az így kapott, most már állandóként kezelhető *EVF* értéke alapján keresi ki a feltételnek megfelelő hallgatókat. Ha nincsen ilyen azonosítójú hallgató (a beágyazott szelekt üres), akkor senkit sem választ ki.

10. Példa korrelált beágyazott szelektre:

Tegyük fel, hogy (elégge elítélendő módon) nem készítettünk idegen kulcsokat a *MITHALLGAT* táblázatban a *HID* hallgató azonosítóra. Meg akarjuk tudni, hogy van(nak)-e a táblázatban nem létező hallgató(k) és hogy az(ok) milyen tárgya(ka)t vett(ek) fel.

Utasítás:

```
select hid, tid from mithallgat
where not exists
(select hid from hallgato where mithallgat.hid = hallgato.hid);
```

Eredmény:

Kiírja mindazon *HID*, *TID* párokat, melyet olyan azonosítóval rendelkező hallgatók vettek fel, akik nem szerepelnek a *HALLGATO* táblázatban. Itt a *MITHALLGAT* minden sorára végre kell hajtani az *EXISTS* beágyazott szelektjét, mert csak így állapítható meg, hogy létezik-e a másik táblázatban hozzátartozó pár.

11. Példa beágyazott szelekt és join egymásba konvertálására.

Az *SQL30 TANTARGY* táblázatából kiválogatjuk azokat a sorokat, amelyek *TID* azonosítója az *SQL00 TANTARGY* táblázatában is szerepel.

Utasítás-1:

```
select * from sql30.tantargy t30
where exists (select tid from sql00.tantargy t00 where t00.tid=t30.tid);
```

A beágyazott szelektet join-nal helyettesíthetjük

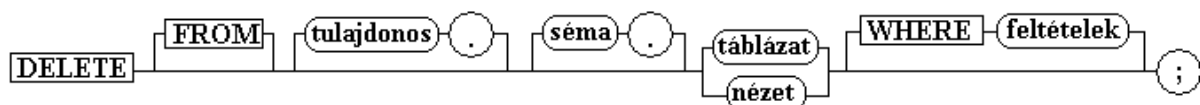
Utasítás-2:

```
select sql30.tantargy.* from sql30.tantargy join sql00.tantargy using (tid);
```

5.6.3.3. Sorok törlése (*DELETE*)

Funkciója: Sor(ok) törlése táblázatból, nézetből.

Egyszerűsített szintaxisa:



Jogosultság:

- Táblázatból akkor törölhetünk sorokat, ha az a saját táblázatunk, vagy van rá *DELETE* jogosultságunk.
- Nézeteken keresztül akkor törölhetünk a bázis táblázatból sorokat, ha a nézet tulajdonosának van a bázis táblázatra *DELETE* jogosultsága. Ezen kívül nekünk is kell rendelkezünk *DELETE* jogosultsággal a nézetre, ha az nem a saját sémánkban van.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
DELETE, FROM	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.
tulajdonos	A táblázat vagy nézet tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk.
táblázat nézet	Innen kell a kiválasztott sorokat törölni. Nézet esetén a bázis táblázat sorai törölődnek.
WHERE	Elhagyható kulcsszó. Jelzi, hogy csak az utána következő feltételnek megfelelő rekordokra érvényes az utasítás.
feltételek	Leírásukat az SQL feltételeknél találjuk. (ld. 5.3.4.5. pont)

Megjegyzések:

- Ha nem adjuk meg a *WHERE* klauzulát, akkor minden sort törölünk. Az üres táblázat megmarad, így abba újra vihetünk be adatokat. (A *DROP TABLE* illetve a *DROP VIEW* utasítás nem csak az adatokat, hanem a teljes táblázatot/nézetet is törli. Így abba csak akkor vihetünk be újra adatokat, ha ismét létrehozuk.)
- Ha megadjuk a *WHERE* klauzulát, akkor csak azokat a sorokat töröljük, amelyek a kiválasztási feltétel(ek)nek megfelelnek.
- Azokat a sorokat, amelyekre idegen kulccsal hivatkozunk csak akkor lehet törölni, ha ezt külön engedélyeztük (*ON DELETE CASCADE* vagy *SET NULL*), vagy az aktuális adatokban nincsen rájuk érvényes hivatkozás, ld. 5.14. táblázat: pont).
- Amíg nem adtunk *COMMIT*-ot vagy *ROLLBACK*-et (ld. 5.6.4. pont) és nem dolgozunk *AUTO COMMIT ON*-nal, az összes törlést visszavonhatjuk.

Példák:

1. Törölni akarjuk a *MITHALLGAT* táblázat minden sorát.

Utasítás:

```
delete from mithallgat;
```

Eredmény: Törli a táblázat minden sorát.

2. Törölni akarjuk a *MITHALLGAT* táblázatból az *12345* azonosítójú hallgató minden adatát.

Utasítás:

```
delete from mithallgat where hid='12345';
```

Eredmény: Törli a táblázatból az *12345* azonosítójú hallgató minden sorát.

3. Törölni akarjuk *HALLGATO*-ból azokat a hallgatókat, akik egy tantárgyat sem vettek fel.

Utasítás:

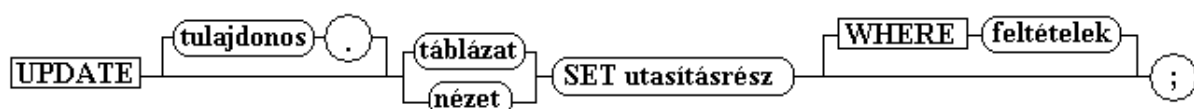
```
delete from hallgato where not exists  
(select hid from mithallgat where mithallgat.hid=hallgato.hid);
```

Eredmény: Egyenként végigmegy *HALLGATO* minden során. Megnézi, hogy az adott *hid*-vel egyező létezik-e *MITHALLGAT*-ban és ha nem, akkor törli *HALLGATO*-ból az adott sort.

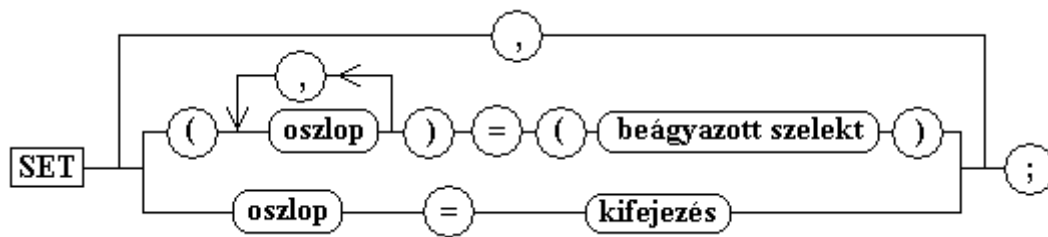
5.6.3.4. Sorok módosítása (*UPDATE*)

Funkciója: Meglévő sor(ok) módosítása táblázatban, nézetben.

Egyszerűsített szintaxisa:



ahol a set utasítás kiegészítés:



Jogosultság:

- Táblázatban akkor módosíthatunk sorokat, ha az a saját táblázatunk, vagy van rá *UPDATE* privilégiumunk.
- Nézetén keresztül akkor módosíthatunk a bázis táblázatban sorokat, ha a nézet tulajdonosának van a bázis táblázatra *UPDATE* privilégiuma. Ezen kívül nekünk is kell rendelkezünk *UPDATE* privilégiummal a nézetre, ha az nem a sajátunk.

Kulcsszavak és paraméterek

Kulcsszó vagy paraméter	Jelentése
UPDATE	Kötelező kulcsszó. Megadja, milyen utasítás kerül végrehajtásra.
tulajdonos	A táblázat vagy nézet tulajdonosának a neve. Ha nem adjuk meg, akkor az a saját objektumunk.
táblázat nézet	Ebben kell a kiválasztott sorokat módosítani. Nézet esetén a bázis táblázat sorai módosulnak.
oszlop	Annak az oszlopnak a neve, amelyiknek az értékét módosítjuk. Azoknak az oszlopoknak az értéke, amelyeket nem soroltunk fel a SET klauzulában változatlan marad.
beagyazott szelekt	Minden oszlopra pontosan egy értéket szolgáltat. A kiválasztási-lista n -edik elemének értéke lesz az n -edik oszlop módosított értéke.
kifejezés	Általában egy oszlop vagy egy állandó (literál). Pontos leírását az SQL kifejezéseknél találjuk (ld. 5.3.8. pont).
WHERE	Elhagyható kulcsszó. Jelzi, hogy csak az utána következő feltételnek megfelelő rekordokra érvényes az utasítás.
feltételek	Leírásukat az SQL feltételeknél találjuk (ld. 5.3.4.5. pont)

Megjegyzések:

- Egy *UPDATE*-tel több oszlop és több sor is módosítható.
- Ha nem adjuk meg a *WHERE* klauzulát, akkor minden sort módosítunk.
- Ha megadjuk a *WHERE* klauzulát, akkor csak azokat a sorokat módosítjuk, amelyek a kiválasztási *feltétel(ek)*nek megfelelnek.
- Nem tudjuk módosítani az oszlopot és hibajelzést kapunk, ha a módosított érték nem felel meg az oszlop adattípusának, (például túl nagy szám vagy túl hosszú

jelsorozat), vagy ellentmond valamilyen integritási feltételnek (pl. egyedi index, idegen kulcs nem megengedett módosítása).

- Nem lehet módosítani csak olvasható nézetben.
- Ha a módosítható nézetet *CREATE VIEW ... WITH CHECK OPTION*-nal definiáltunk, akkor csak olyan értékre módosíthatunk, amelyik megfelel a nézet definíciójának (lásd 3. példa). Ha nem így definiáltuk, akkor módosíthatunk a bázis táblában a nézet feltételeit nem teljesítő értékekre is. Ezeket természetesen nem látjuk a nézetben.
- Amíg nem adtunk explicit *COMMIT*-ot vagy *ROLLBACK*-et, (ld. 5.6.4. pont) addig az összes módosítást visszavonhatjuk.

Példák:

1. Javítani akarjuk a tanulmányi statisztikát. Az összes adatbázis tantárgyat felvett hallgatónak (tid = 'AB'), akinek 50 alatt van a *GYAKPONT* pontszáma 50-re növeljük a pontszámát.

Utasítás:

```
update mithallgat set gyakpont = 50
where tid = 'AB' and gyakpont < 50;
```

Eredmény: Megváltoztatja 50-re *GYAKPONT* értékeit azokban a rekordokban, ahol az 50-nél kevesebb volt és a tantárgy azonosítója *AB* volt. Azoknál a hallgatóknál, akiknél *GYAKPONT* értéke *NULL érték* volt az nem változott! Ha ezeket is módosítani akarjuk, át kell írni a feltételt:

```
where tid = 'AB' and (gyakpont < 50 or gyakpont is null);
```

2. Minden hallgatónak, aki nem jelesre vizsgázott eggyel feljavítjuk az eredményét, ha gyakorlati pontjainak száma legalább húszszorosa az elért vizsgaeredményének és oktatója '*Quittner*' volt. (Pl. a közepes (3) helyett jót (4) kap, ha gyakorlati pontjainak száma elérte a 60-at (=3*20).

Utasítás:

```
update mithallgat set osztalyzat = osztalyzat + 1
where osztalyzat < 5 and gyakpont > = osztalyzat * 20
and tid in (select tid from tantargy where upper(oktato) like '%QUITTNER%');
```

Mivel nem tudjuk pontosan, hogyan van az oktató neve írva, –lehet csak nagy kezdőbetűvel vagy csupa nagybetűvel, állhat előtte például „Dr”, utána lehet még a keresztnéve is –, ezért használjuk a *LIKE* és az *UPPER* függvényeket.

3. Az *EVFOLYAM_4* definíciója:

```
create view evfolyam_4 as select * from hallgato where evf = 4
with check option constraint con_evf_4;
```

Utasítás:

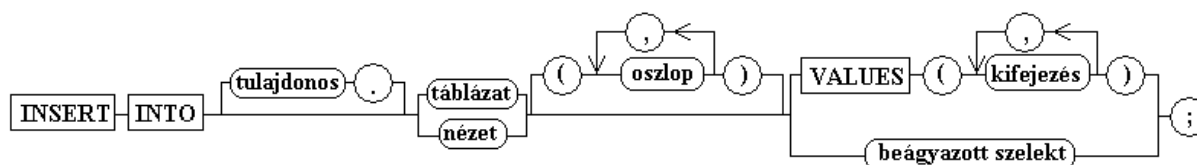
```
update evfolyam_4 set evf = 5;
```

Eredmény: Hibás! A nézetet *WITH CHECK OPTION*-nal definiáltuk és $evf = 5$ ellentmond a kiválasztási feltételnek. Hibaüzenetet kapunk!

5.6.3.5. Sorok bevitele (*INSERT*)

Funkciója: Sor(ok) beírása táblázatba, nézetbe.

Egyszerűsített szintaxisa:



Jogosultság:

- Csak a saját táblázatunkba/nézetünkbe írhatunk be sorokat, vagy akkor, ha van rá *INSERT* privilégiumunk.
- Nézeteken keresztül akkor írhatunk be a bázis táblázatába sorokat, ha a nézet tulajdonosának van a bázis táblázatra *INSERT* privilégiuma. Ezen kívül nekünk is kell rendelkezünk *INSERT* privilégiummal a nézetre, ha az nem a sajátunk.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
INSERT INTO	Kötelező kulcsszavak. Megadják, milyen utasítás kerül végrehajtásra.
tulajdonos	A táblázat vagy nézet tulajdonosának a neve. Ha nem adjuk meg, akkor a saját objektumunk.
táblázat nézet	Ebbe kell a sorokat beírni. Nézet esetén a bázis táblázat sorai módosulnak.
oszlop	Annak az oszlopnak a neve, amelyikbe az új sorban az új értéket beírjuk. Az oszlopokat a felsorolásban vesszővel választjuk szét. Azoknak az oszlopoknak az értéke, amelyeket nem soroltunk fel az alapértelmezés lesz.
VALUES	Választható kulcsszó. Jelzi, hogy ezután következnek egyenként az oszlopokba beírandó értékek.
kifejezés	Általában egy oszlop vagy egy állandó (literál). Pontos leírását az SQL kifejezéseknél találjuk (ld. 5.3.8. pont). Minden egyes <i>kifejezés</i> megfelel a <i>táblázat/nézet</i> egy oszlopának. Az <i>n</i> -edik <i>kifejezés</i> értéke lesz az <i>n</i> -edik oszlop értéke.
beágyazott szelekt	Minden oszlopa megfelel a táblázat/nézet egy oszlopának. A kiválasztási lista <i>n</i> -edik elemének értéke lesz az <i>n</i> -edik oszlop értéke.

Megjegyzések:

- Az utasítás végrehajtásakor a *VALUES* mindig egy sort ír be. A *beágyazott szelekt* annyi sort ír be, ahányat eredményül ad.
- Az *oszlopok* száma és sorrendje meg kell egyezzen a *kifejezések* illetve a *beágyazott szelekt* kiválasztási listájában levő elemek számával és sorrendjével. A beírt sorban az első oszlop értéke lesz az első kifejezés vagy a kiválasztási lista első elemének értéke, a második oszlopé a második kifejezés illetve listaelem, és így tovább.
- Ha nem adunk meg oszlopot, akkor a táblázat vagy nézet minden oszlopának értéket kell adnunk. Az értékadás sorrendje a táblázat vagy nézet oszlopainak sorrendjével kell megegyezzen. Ha a táblázatot vagy a nézetet nem módosítottuk, ez megegyezik a *CREATE TABLE* illetve a *CREATE VIEW* utasításban az oszlopok definiálási sorrendjének. Vagy egy oszlopot sem adunk meg, vagy minden olyan oszlopot meg kell adnunk, amelyekbe értéket viszünk be.
- Ha a felsorolásban valamelyik oszlopot nem adjuk meg, akkor annak értéke az alapértelmezett (default) illetve *NULL érték* lesz az újonnan bevitt sorban.
- Mindenképpen értéket kell adnunk azoknak az oszlopoknak, amelyeket *NOT NULL*-ként definiáltunk.
- Az utasítás nem hajtódik végre, azaz nem tudjuk bevinni a sort és hibajelzést kapunk, ha a beviendő érték nem felel meg az oszlop adattípusának, (például túl nagy szám vagy túl hosszú jelsorozat), vagy ellentmond valamilyen integritási feltételnek (pl. egyedi index, idegen kulcs, nem megengedett érték). A hibaüzenetben általában megjelenik az oszlop és az integritási feltétel neve.
- Ha a *beágyazott szelekt* olyan sort választ ki, amelyet bármilyen okból nem lehet bevinni, akkor az utasítás nem hajtódik végre. A bevitel leáll, az addig beírt sorok is törlődnek és hibajelzést kapunk. Tehát minden vagy semmi. Vagy az összes kiválasztott sor beíródik vagy egy sem.
- Ha a nézetet *WITH CHECK OPTION*-nal definiáltuk, akkor a nézeten keresztül csak olyan értékek vihetők be, amelyek megfelelnek a nézet definiálási feltételeinek (lásd 4. példa). Ha nem így definiáltuk, akkor bevitethetünk a bázis táblába a nézet feltételeit nem teljesítő adatokat is. Ezeket természetesen nem látjuk a nézetben.
- Nagy adatmennyiség kötegelt (batch) üzemmódban való beviteléhez a fejlettebb adatbázis-kezelő rendszerek speciális segítséget nyújtanak. Ilyen például az ORACLE-ben az SQL*loader, vagy az UDB/DB2-ben a LOAD szolgáltató program. Ezek használata erősen rendszerfüggő, ismertetésük meghaladja ennek a könyvnek a kereteit.

Példák:

1. Adatokat akarunk egyenként beírni a *HALLGATO* táblázat minden oszlopába.

Utasítás:

```
insert into hallgato values('Kiss', 'Elek','10000','1982-03-31', 1, user);
```

Eredmény: Ha még nem volt *10000* azonosítójú hallgató, akkor egy új sor kerül be a táblázatba ezekkel az adatokkal. Az *SZDATUM* születési dátumot az alapértelmezett formában kell megadnunk. A *user* fenntartott SQL szó, értéke az aktuális felhasználó bejelentkezési azonosítója.

Ha már volt *10000* azonosítójú hallgató, akkor hibaüzenetet kapunk és a *HALLGATO* táblázat változatlan marad.

- Adatokat akarunk egyenként beírni a hallgató táblázatba. Csak a *hid* , *vnev* , *knev* oszlopoknak adunk értéket. Miután felsoroljuk az oszlopokat, nem kötelező ezeket a definiálási sorrendben megadnunk. Csak arra kell ügyelnünk, hogy a *VALUES* értékei is ugyanebben a sorrendben legyenek.

Utasítás:

```
insert into hallgato(hid,vnev,knev) values ('10001','Nagy','Ferenc');
```

Eredmény: Ha még nem volt *10001* azonosítójú hallgató, akkor egy új sor kerül be a táblázatba ezekkel az adatokkal. Az explicit meg nem adott oszlopok értéke az alapértelmezés lesz.

- Át akarjuk venni *SQL30.HALLGATO* adatait. A saját adatainkat szabványosítva, nagy kezdő betűkkel, majd csupa kis betűvel akarjuk tárolni. Ehhez használjuk az *initcap* és a *lower* SQL függvényeket (ld. 5.5.2. pont).

Utasítás:(lehet, hogy hibás)

```
insert into hallgato
select initcap(lower(vnev)),initcap(lower(knev)),hid,szdatum,evf,tulaj
from sql30.hallgato;
```

Eredmény: Attól függően, hogy volt-e a két táblázatban azonos *hid* vagy minden sort átveszünk vagy egyet sem. Ennek kiküszöbölésére egy beágyazott szelekttel kiszűrjük az egyező *hid*-ket és csak az eltérőket írjuk be.

Utasítás (javított):

```
insert into hallgato
select initcap(lower(vnev)),initcap(lower(knev)),hid,szdatum,evf,tulaj
from sql30.hallgato h30
where not exists (select hid from hallgato h where h30.hid=h.hid);
```

Eredmény: Átvesszük *SQL30* azon hallgatóit, akik nem voltak a mi táblázatunkban. (Az áttekinthetőség érdekében a táblázatoknál használtuk a *h* és *h30* ideiglenes neveket. Ezen a szelekten belül így jelöljük a *saját* és az *SQL30.HALLGATO* táblázatot).

- Az *EVFOLYAM_4* definíciója:

```
create view evfolyam_4 as select * from hallgato
where evf=4 with check option constraint evf_4;
```

Utasítás:

```
insert into evfolyam_4(hid,vnev,knev,evf)
values('12345','Gipsz','Jakab',5);
```

Hibás! A nézetet *CHECK OPTION*-nal definiáltuk és *evf=5* ellentmond a kiválasztási feltételnek.

5.6.4. Tranzakcióvezérlő utasítások (COMMIT, ROLLBACK)

A tranzakcióvezérlő utasítások az adatkezelő utasítások hatását véglegesítik, vagy hatástalanítják.

Amikor egy adatbázisban változtatásokat hajtunk végre, azokat a rendszer mindaddig ideiglenesnek tekinti, amíg azokat egy ténylegesen vagy burkoltan kiadott tranzakcióvezérlő utasítással nem véglegesítjük, vagy vissza nem állítjuk a kiinduló állapotot. Ennek az a magyarázata, hogy nincs értelme egy olyan utasítást, vagy utasítássorozatot végrehajtani, amelyik több sort érintene, de ezek között van olyan, amelyekre valamelyik utasítás nem hajtható végre, vagy valamelyik lépés után a rendszer meghibásodik.

Tipikusan ilyen eset állhat elő, amikor egy *DELETE* utasítással egyszerre több sort törölnénk. A tényleges törlést egyesével, soronként hajtja végre a rendszer. Ha ekkor előfordul, hogy a törlendő sorok között van egy olyan, amelyik nem törölhető, mert például idegen kulccsal hivatkozunk rá, akkor egy sor sem törölhető. Vissza kell állítani az eddig törölt sorokat is. De előfordulhat az is, hogy sok rekord egyedi módosítása után jövünk rá, hogy mindegyik módosítás hibás volt és egyszerűen vissza akarjuk állítani az eredeti állapotot. Gyakori eset az is, amikor egy feldolgozás csak akkor lesz eredményes, ha a több utasításból álló tranzakció mindegyik lépése hibátlanul végrehajtott. Például egy banki átutalás csak akkor tekinthető végrehajtottnak, ha az átutaló számlájáról levették az összeget (*UPDATE*), a kedvezményezett számláján jóváírták (újabb *UPDATE*) és a pénzmozgásért járó jutalékot rögzítették a banki nyilvántartásban (*INSERT*).

Ahhoz, hogy az eredeti állapotot vissza lehessen állítani, a rendszer minden adatbázisbéli változtatásról naplót vezet. Ebben minden változtatáshoz feljegyzi, hogy mikor, mi történt és hogy mit mire vagy hogyan módosítottak. A kiinduló állapot visszaállítása ennek a naplónak az adatai alapján történik.

A tranzakcióvezérlő utasításokat a tranzakciót végrehajtó felhasználó adhatja ki. Hatásuk csak arra a tranzakcióra terjed ki, amelyikben kiadták.

A két leggyakrabban használt tranzakcióvezérlő utasítás a *COMMIT* és a *ROLLBACK*.

A *COMMIT* utasítás funkciója: Véglegesíti az adatbázisban az utolsó véglegesítési pont óta kiadott változtatásokat.

Egyszerűsített szintaxisa:

COMMIT ;

A *ROLLBACK* utasítás funkciója: Visszaállítja az adatbázist az utolsó véglegesítési pontban érvényes állapotba. Az azután végrehajtott változtatásokat érvényteleníti.

Egyszerűsített szintaxisa:

ROLLBACK ;

Véglegesítési pontnak számít:

- az interaktív SQL session (ülés) vagy a program kezdete és vége,
- *COMMIT* vagy *ROLLBACK* utasítás,
- az adatbázishoz való csatlakozás (*CONNECT*),
- *AUTOCOMMIT ON* beállítása esetén minden SQL utasítás vége,

- az adatleíró utasítások (*CREATE ...*, *DROP ...*) végrehajtása általában automatikusan egy-egy *COMMIT* utasítás végrehajtását is jelentik az utasítás elkezdése előtt és az utasítás befejezése után.

Megjegyzések:

- A *COMMIT* és a *ROLLBACK* csak a saját sessionünkben vagy az ugyanabban a programban lefutott utolsó tranzakció(k)ban végrehajtott változtatásokat véglegesíti illetve érvényteleníti.
- Lehetőségünk van minden egyes SQL utasítás után a változtatások azonnali, automatikus véglegesítésére. Ehhez *AUTO COMMIT ON* üzemmódra kell beállítani az adatbázis-kezelő rendszert.
- Egy SQL session (ülés) vagy program két véglegesítési pont közti részét *logikai munkaegységnek* vagy *tranzakciónak* nevezzük. Ha egy munkaegység abnormálisan fejeződik be, akkor az automatikus *ROLLBACK*-et jelent.
- Amíg nem adtunk ki *COMMIT*-ot vagy *ROLLBACK*-et, addig mások nem láthatják a logikai munkaegységben végrehajtott változtatásokat. Helyette vagy az utolsó véglegesítési pontkor érvényes adatokat látják (pl. ORACLE), vagy meghatározott idejű várakozás után a feldolgozásuk hibajelzéssel befejeződik (pl. UDB/DB2).
- Amíg nem adtunk ki *COMMIT*-ot vagy *ROLLBACK*-et, addig mások nem változtathatják a logikai munkaegységben megváltoztatott rekordjainkat. Ilyenkor a feldolgozásuk vagy hibajelzéssel abnormálisan befejeződik, vagy vár, amíg mi nem hozunk létre egy véglegesítési pontot. Ez párhuzamos munkánál problémát okozhat (ld. 8.2. alfejezet).
- Ha nem automatikus *COMMIT*-tal dolgozunk (*AUTO COMMIT OFF*), akkor is adjunk ki rendszeres időközönként *COMMIT*-ot. Ezzel kevésbé fogjuk gátolni mások munkáját, akik ugyanazokkal az adatokkal dolgoznának. További érv, hogy ha a mi feldolgozásunkban valami baj történne és abnormálisan leállna, akkor kevesebb munkánk veszik kárba.

A tranzakcióvezérlő utasítások közé tartozik a *SAVEPOINT* utasítás is. Ezzel visszaállítási pontot helyezhetünk el az adatbázisban.

Egyszerűsített szintaxisa:

SAVEPOINT **viSSzaállítási-pont** **;**

A *ROLLBACK TO VISSZAÁLLÍTÁSI_PONT* utasítással visszaállíthatjuk az adatbázisnak az ebben az időpontban érvényes állapotát (ld. 2. példa).

Példák:

1. Annak, aki nem vizsgázott le, automatikusan elégtelent akarunk adni. (A *MITHALLGAT* táblázat *OSZTALYZAT* oszlopát módosítjuk).

Utasítások:

commit;

update mithallgat set osztalyzat = 1 where osztalyzat is null;

Rájöttünk, hogy ez nem helyes és szeretnénk visszaállítani az eredeti állapotot.

Újabb utasítás:

rollback;

Eredmény: Akinek nem volt osztályzata , annak továbbra sem lesz.

2. Visszaállítási pontokat helyezünk el az adatbázisban.

Utasítások:

```
commit;  
insert into ...  
savepoint vissza_1 ;  
select ...  
update ...  
savepoint vissza_2 ;  
delete from ...  
rollback to savepoint vissza_2;
```

Eredmény: Az *INSERT* és az *UPDATE* eredménye megmarad, de a *DELETE*-tel törölt sorok visszaíródnak. (A *ROLLBACK TO SAVEPOINT vissza_1* hatására az *INSERT* hatása még érvényben maradt volna, míg *ROLLBACK* esetén egy változtatás sem maradna meg.)

5.7. SQL utasítások beépítése programokba

Az SQL nyelv használható önálló, interaktívan működő adatbázis-kezelő rendszerekben, de beépíthető magas szintű programozási nyelvekbe is, Napjainkban a leggyakoribb befogadó nyelv (host language) a COBOL, C, C++ és a Java. De például az ORACLE kifejlesztett egy speciális nyelvet kifejezetten adatbázis alkalmazások számára (PL/SQL), és az adatbázis kezelésére az ORACLE-t használó programok zöme ezen a nyelven íródott.

A programba a program feladatának elvégzéséhez szükséges utasítások mellé az SQL használatához beépítünk:

- *SQL utasításokat.* Ezekkel határozzuk meg, hogy mi történjen az adatbázisban (ld. 5.6. alfejezet).
- *host változókat.* Ezek a programban olyan speciális változók, amelyekbe az SQL az adatbázisból való beolvasásnál beviszi az adatbázisban lévő értéket, kiíraskor pedig kiírja belőlük a változó aktuális értékét az adatbázisba. A host változókat és használatukat az. 5.7.3. pontban tárgyaljuk.
- *SQL kommunikációs területet.* Ebben az adatbázis-kezelő rendszer az SQL utasítások végrehajtásának eredményességét illetve az esetleges hiba okát jelzi a programnak, hogy az ennek kiértékelése után döntsön a feldolgozás további menetéről (ld. 5.7.2. pont).
- utasításokat, melyekkel az SQL utasítások végrehajtásakor bekövetkező előre látott és az előre nem látható rendkívüli *eseményeket kezeljük.* Ezeket is az 5.7.2. pontban ismertetjük.

5.7.1. SQL utasítások csoportosítása

Az interaktívan használható SQL utasítások programból is kiadhatók ugyanabban a formában. Az adatkezelő utasításoknál azonban a WHERE feltételek kibővíthetők a WHERE

CURRENT OF CURSOR feltétellel, a SELECT utasításnak pedig speciális formáját kell használnunk, Ezeket az 5.7.4. pontban, a kurzor használatánál tárgyaljuk.

Vannak olyan SQL utasítások, amelyek csak programból adhatók ki. Ezek közül a legfontosabbak:

- BEGIN DECLARE SECTION és END DECLARE SECTION,
- CLOSE és OPEN,
- DECLARE,
- EXECUTE és EXECUTE IMMEDIATE,
- FETCH,
- INCLUDE,
- PREPARE,
- WHENEVER.

A programba beépíthető SQL utasításokat többféle módon is csoportosíthatjuk. Az egyik lehetséges csoportosítás az alábbi:

- statikus SQL utasítások,
- dinamikus SQL utasítások,
- nem végrehajtható utasítások.

A *statikus SQL utasításoknál* már a program megírásakor tudjuk, hogy milyen adatbázis objektumokkal milyen műveletet akarunk elvégezni (pl. új sorokat írunk be a *HALLGATO* táblázatba). Legfőljebb egyes paraméterek aktuális értékét kapcsolja a végrehajtáskor a rendszer a programbeli host változókhoz (pl. az egyes oszlopok értéke az újonnan beírt sorban). Az adatbázis adatainak eléréséhez szükséges (optimalizált) utat(ka)t a programmal együtt hozzuk létre és önálló *elérési csomagként* illetve *tervként* (package, plan) hozzákapcsoljuk a programhoz. Általában ekkor határozzuk meg a programban használt objektumok tulajdonosát is. Ez az elérési út – függetlenül az adatbázisban történt változásoktól – egészen addig érvényben marad, amíg a programnak az adatbázissal kommunikáló részének szerkezetét nem módosítjuk, vagy valamilyen ok miatt (pl. megszűnt az eléréshez felhasznált index) a generált elérési út nem használható többé.

A *dinamikus SQL utasításoknál* a program létrehozásakor még nem ismerjük az utasítást, vagy olyan fontos paramétere (pl. a táblázat neve) hiányzik, hogy nem lehet az elérési utat meghatározni. Az utasítás pontos formája csak futáskor derül ki, ezért az adatok elérési módját a rendszer minden alkalommal dinamikusan, a program futásakor állapítja meg. Ez időbe kerül, ezért az ilyen utasításokat tartalmazó programok végrehajtási ideje általában hosszabb, minthogyha ezeket az utasításokat statikus formában adtuk volna meg. A fő ok azonban, amiért a dinamikus SQL utasítások használatát üzemszerűen működő adatbázisokban nem engedélyezik az, hogy ilyenkor az üzemeltetés nem tudja ellenőrizni a program elindításakor, hogy mi történhet az adatbázisban. Ezért – elméletileg bármilyen szép és érdekes lehet – gyakorlati mellőzöttsége miatt a dinamikus SQL utasításokkal a könyvünkben tovább nem foglalkozunk.

A *nem végrehajtható SQL utasítások* a program fordításához, vagy nem az adatbázishoz közvetlenül kapcsolódó műveletek elvégzéséhez (pl. mi a teendő, ha egy törlendő vagy módosítandó rekordot sem találtunk) adnak útmutatást.

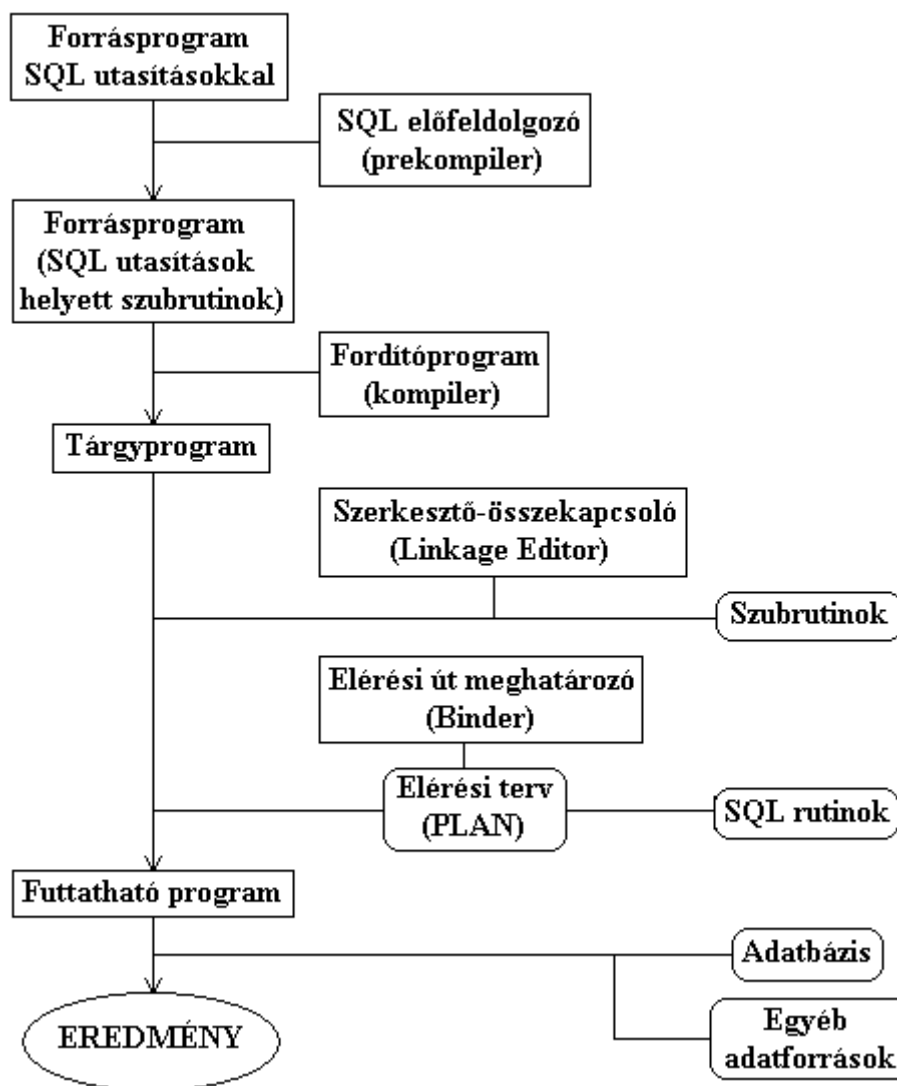
Az SQL utasítások egy másik csoportosítási módja az, hogy *igényelnek-e kurzort vagy sem*. Ennek az a magyarázata, hogy az SQL adatkezelő műveletek egyszerre több rekordra is vonatkozhatnak. Egy *SELECT* például akárhány sort kiválaszthat. Ezzel ellentétben a hagyományos programnyelvek egyszerre csak egy rekordot tudnak feldolgozni. Ezt az ellentmondást hidalja át a *kurzor* (ld. 5.7.4. pont), ami megmutatja, hogy az SQL által rendelkezésre bocsátott rekordok közül most éppen melyikkel foglalkozzon a befogadó program. Másképpen kell azokat az utasításokat felépíteni, amelyekben szükség lehet kurzorra.

5.7.2. SQL programok elkészítése

Az SQL utasításokat is tartalmazó magas szintű nyelven megírt program előkészítése és feldolgozása a következő lépésekből áll:

- az SQL utasítások felismerése és az utasításnak a megfelelő szubrutin behívásával való helyettesítése (prekompilálás, előfordítás),
- a forrásprogram fordítása és szerkesztése,
- az optimalizált elérési utak meghatározása és a programhoz való hozzácsatolása (BIND),
- futtatás.

A folyamatot az 5.1. ábrán láthatjuk.



5.1. ábra: SQL program előkészítésének és feldolgozásának főbb lépései.

Ahhoz, hogy az SQL előfeldolgozó (prekompiler) felismerje az SQL utasításokat, a forrásprogramban jelezni kell minden egyes SQL utasítás elejét és végét. Mindent, ami ezek között van a prekompiler SQL utasításnak tekint. Szintaktikusan ellenőrzi, értelmezi, majd átalakítja a befogadó nyelv utasításaivá. Ami viszont ezeken kívül van, azt változtatlanul hagyja. Ezt, az SQL már átalakított utasításaival együtt a forrásnyelvi fordítóprogram fogja feldolgozni.

Ha a precompiler az értelmezés folyamán szintaktikai hibát talál, azt jelzi. Ilyenkor általában nem generál forráskódot a nyelvi fordító számára sem.

Az SQL utasítások megjelölésének szintaktikája függ a befogadó nyelvtől. Az utasítás kezdetét általában az *EXEC SQL*, a végét pedig az *END-EXEC* vagy a pontosvessző (;) jelzi.

Amennyiben a programba már előre megírt SQL utasításokat akarunk beilleszteni, arra az *INCLUDE SQL* utasítást használhatjuk. Ezt a következő módon építhetjük be a programba:

```
EXEC SQL
INCLUDE ... a bemásolandó SQL utasításokat tartalmazó fájl vagy fájlmember
neve
END-EXEC
```

Így másoljuk be a programba az *SQL kommunikációs területet (SQLCA)* is. Az adatbázis-kezelő rendszer ebben értesíti a programot minden egyes SQL utasítás elvégzése után a végrehajtás körülményeiről. Ez a terület tartalmaz egy rövid szöveget, ami, ha hiba történt, azt azonosítja, egy mezőt, amelyikbe bekerül az utasítás által beírt, törölt vagy módosított rekordok száma, figyelmeztetést esetleges hibára (pl. nem tudjuk beolvasni egy oszlop értékét, mert túl sok karakterből áll, vagy túl nagy szám). A legfontosabb mezői azonban az *SQLCODE* illetve az *SQLSTATE*. Ezek minden SQL művelet után egyértelműen jelzik a művelet eredményességét vagy hibás voltát. Mindkettő értéke akkor és csak akkor 0, ha a művelet hibátlanul hajtott végre. Negatív *SQLCODE* hibás végrehajtást jelent és értéke utal a hiba jellegére. Például *SQLCODE* = -404 azt jelzi, hogy az adott karakterek, -406 azt, hogy az adott szám értékes jegyei nem férnek el az oszlopban, ezért az *INSERT* vagy az *UPDATE* művelet (adat bevitel/módosítás) nem hajtott végre. Pozitív *SQLCODE* valamilyen különleges, de nem okvetlenül hibát jelentő eseményt jelent. Például *SQLCODE* = +100 értelmezése *NOT FOUND*, nincs ilyen rekord. (Ez kurzorral való munkánál azt jelenti, hogy a feldolgozandó rekordoknak a végére értünk). Az egyes *SQLCODE* értékek jelentése rendszerenként különbözhet, míg az azonos *SQLSTATE* értékek minden rendszerben ugyanarra a hibára utalnak. Így az *SQLSTATE* ellenőrzése adatbázis-kezelő rendszerektől független programozási módot kínál, de történeti okokból mégis inkább az *SQLCODE* vizsgálata a gyakoribb.

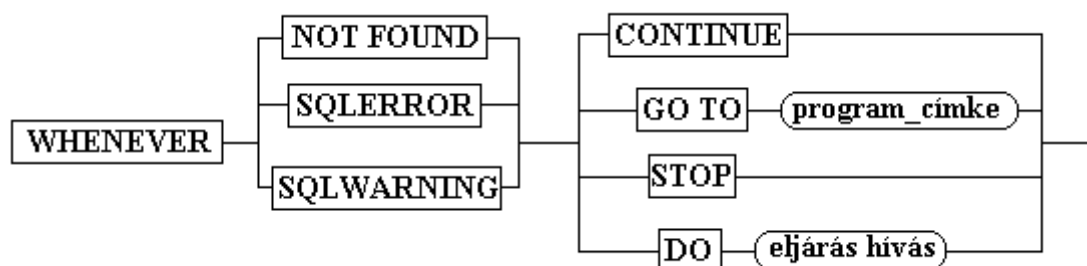
Természetesen attól, hogy egy adatbázis művelet hibás volt, attól a programunk minden további nélkül vagy egy hibaüzenet kiadásával folytatódhat. Attól, hogy ezer adat bevitelkor egyet nem tudtunk bevinni, nem kell leállítani a továbbiak feldolgozását. Ugyanakkor előfordulhat az is, hogy ha egy adatot nem tudtunk hibátlanul beolvasni, akkor a további feldolgozásnak már semmi értelme sincs és be kell fejezni az egész programot esetleg a program kezdetekor fennálló állapot visszaállításával (*ROLLBACK*, ld. 5.6.4. pont). Ezért elvileg minden SQL utasítás után meg kellene vizsgálnunk az *SQLCODE/SQLSTATE* változót és a feldolgozás további menetét annak értékétől kellene függővé tennünk. Ha ez az érték 0, 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 írását. Egyszerűbb helyette a különleges események kezelésére szolgáló *WHenever* utasítás használata.

WHenever

Funkciója: Programelágazás generálása az SQL utasítás hibátlan vagy hibás végrehajtásától függően

Egyszerűsített szintaxisa:



Jogosultság:

A program készítője bármikor kiadhatja.

Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
WHENEVER	Kötelező kulcsszó. A programszövegben innentől érvényesül az utasítás hatása.
NOT FOUND	Választható kulcsszó. Akkor igaz a feltétel, ha nincs a feltételnek megfelelő rekord, vagy a kurzorral való feldolgozás végére értünk. (<i>SQLCODE</i> = +100, <i>SQLSTATE</i> = '02000')
SQLERROR	Választható kulcsszó. Akkor igaz a feltétel, ha az SQL művelet hibával fejeződött be (<i>SQLCODE</i> < 0).
SQLWARNING	Választható kulcsszó. Akkor igaz a feltétel, ha olyan esemény következett be, ami esetleg hibát okozhat. (<i>SQLCODE</i> > 0 de < 100).
CONTINUE	Választható kulcsszó. Ha igaz a feltétel, akkor a program a következő utasítással folytatódik. Ha egyáltalában nem adtunk meg <i>WHENEVER</i> -t, akkor ez az alapértelmezés.
GO TO	Választható kulcsszó. Programelágazás generálódik.
program_címke	A feltétel teljesülése esetén az itt megadott címkén folytatódik a program.
STOP	Választható kulcsszó. A program befejeződik. Ez egyben implicit <i>ROLLBACK</i> -et is jelent.
DO	Választható kulcsszó. Ha igaz a feltétel, akkor a program az utána specifikált eljárás vagy szubrutin hívásával folytatódik.
eljárás hívás	Formája függ a befogadó programnyelvtől. Meghatározza a végrehajtandó eljárást/szubrutint és annak paramétereit.

Megjegyzések:

- Ha egy eseményre nem adtunk meg *WHENEVER*-t, akkor alapértelmezésként erre *CONTINUE* érvényes. Ha egyáltalában nem adtunk meg *WHENEVER*-t, akkor alapértelmezésként minden SQL eseményre a *CONTINUE* érvényes. Ebben az esetben a program semmilyen SQL hibát nem kezel le.
- Ha speciális SQL hibákat egyedi módon kívánunk kezelni (pl. duplikált kulcsnál a hiba miatt nem fejezzük be a programot, hanem kiírjuk a hibás rekordot és tovább

folytatjuk a feldolgozást), akkor ezeket kivesszük az általános `SQLERROR` ágból. A speciálisan kezelendő SQL utasítás előtt kiadjuk a *WHENEVER SQLERROR CONTINUE* utasítást. Az SQL utasítás végrehajtása után egyedi hibakezelést végzünk az

if sqlcode = 'speciálisan kezelendő hiba kódja' then *hibakezelési eljárás*
else *szokásos hibakezelési eljárás*

utasítás segítségével¹. Végül a *WHENEVER SQLERROR* eredeti formában való kiadásával a program további részére újra visszaállítjuk a szokványos hibakezelést.

- Ne felejtjük el, hogy a *WHENEVER* érvényességi hatályát nem az utasítások végrehajtási sorrendje, hanem a program szövegében lévő helye határozza meg (ld. 1. példa).
- Ha a hibakezelő részben (ami *SQLERROR* esetén kerül végrehajtásra) is van SQL utasítás, akkor meg kell elé adni a *WHENEVER SQLERROR CONTINUE* utasítást, mert különben, ha a hibakezelő részben SQL hiba lépne fel, akkor a program végtelen ciklusba kerülne.

Példa:

1. Az 5.2. ábrán bemutatjuk a *WHENEVER* használatát.

```
Program kezdete ...
EXEC SQL.
    WHENEVER NOT FOUND GO TO VÉGE
END-EXEC.
EXEC SQL.
    WHENEVER SQLERROR GO TO HIBA
END-EXEC.
    ... program (és SQL) utasítások
EXEC SQL.
    SQL utasítás
    itt, ha nem talál rekordot (NOT FOUND) a VÉGE,
    SQL hiba esetén a HIBA címre ugrik
END-EXEC.
    ... program utasítások
EXEC SQL.
    WHENEVER NOT FOUND GO TO MINCS
END-EXEC.
    ... program (és SQL) utasítások
HOSSZU-MEZO-KIVETEL.
EXEC SQL.
    WHENEVER SQLERROR CONTINUE
END-EXEC.
EXEC SQL.
    INSERT INTO HALLGATO VALUES ...
END-EXEC.
```

¹ Példáinkban részben az olvashatóság, részben a nagy adatbázisokkal kapcsolatos munkáknál való elterjedtsége miatt befogadó nyelvként (host language) a COBOL-t használjuk.

```

IF SQLCODE = - 404 GO TO HOSSZU-MEZO-HIBA.
    itt, ha túl sok karaktert akarunk bevinni valamelyik mezőbe,
    akkor a HOSSZU-MEZO-HIBA címkén, más hiba esetén a
    következő utasítás alapján a HIBA címkén folytatódik a program.
IF SQLCODE < 0 GO TO HIBA.
VISSZA-NORMAL-HIBAKEZELESRE.
EXEC SQL.
    WHENEVER SQLERROR GO TO HIBA
END-EXEC.
HOSSZU-MEZO-KIVETEL-VEGE.
EXIT
    ... program (és SQL) utasítások
EXEC SQL.
    Itt, ha nem talál rekordot (NOT FOUND) a NINCS,
    SQL hiba esetén továbbra is a HIBA címre ugrik
END-EXEC.
...
HIBA.
    ... Itt történik az SQL hibák lekezelése
NINCS.
    ... NOT FOUND feltétel egyik lekezelési módja
    (pl. sorozat feldolgozás közben).
    ...
VÉGE.
    ... NOT FOUND feltétel másik lekezelési módja
    (pl. adatok beolvasásának a végén)...
HOSSZU-MEZO-HIBA.
    ... Az oszlopba be nem férő, túl hosszú karaktersorozatok lekezelése.
GO TO HOSSZU-MEZO-KIVETEL-VEGE.

```

5.2. ábra: **WHENEVER** használata.

NOT FOUND esetén, attól függően, hogy az a leírt program melyik részén álló utasítás végrehajtáskor állt elő, a *VÉGE*, vagy a *NINCS* címkére ugrik a program. *SQLERROR* (hiba) bekövetkezésekor mindig a *HIBA* címkére ágazik el, kivéve a külön megjelölt *INSERT* utasításnál, ahol az oszlopba be nem férő karakter típusú adat beírási kísérlete után a *HOSSZU-MEZO-HIBA* címen folytatódik a feldolgozás. *SQLWARNING* (figyelmeztetés) esetén, mivel erre nem adtunk direktívát, mindig az alapértelmezéssel (*CONTINUE*), a következő utasítással folytatódik a program.

5.7.3. Host változók

Igen rugalmatlan lenne a programunk, ha a statikus SQL utasításokat mindig csak ugyanazokkal az adatokkal tudnánk végrehajtani. A *host változók* segítségével elérhetjük, hogy a statikusan megadott utasítás a program futásakor 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 deklarálnunk a forrásnyelv szintaktikája szerint. Típusa meg kell feleljen annak az oszlopnak az adattípusának, amelybe a host változó aktuális értékét kiírjuk, vagy ahonnan beolvassuk. A legtöbb implementációban a deklarálást a *BEGIN DECLARE SECTION* és az *END DECLARE SECTION* SQL utasítások közé kell tennünk.

Lehetőség van az adatbázis táblázatok oszlopainak megfelelő host változók deklarálásának automatikus generálására is.

A host változókat a programban ugyanúgy használhatjuk, mint a többi változót, az SQL utasításokban pedig ahol erre lehetőség van, úgy, mint bármely oszlopot. Az egyedüli eltérés, hogy az SQL utasításon belül a nevük elé egy kettőspontot (:) kell írunk. Ez jelzi az SQL precompilernek, hogy ennek a névnek nincsen az adatbázisban megfelelője, a befogadó (host) programban van deklarálva. Az áttekinthetőség érdekében célszerű az összetartozó változókra és oszlopokra programban és az adatbázisban azonos neveket használni, vagy ezek csak egy, az oszlopnév elé tett prefixben különbözzenek egymástól.

Mint azt említettük, az *INSERT*, *DELETE* és az *UPDATE* utasítások párbeszédés és programbeli formátuma megegyezik. Így például, ha a programban módosítani szeretnénk annak a hallgatónak a keresztnévét, akinek az azonosítóját a *HLG-HID* programváltozó tartalmazza a *HLG-KNEV* aktuális értékére, akkor azt az 5.3. ábrán látható módon tehetjük meg:

```
... értékadás HLG-HID és HLG-KNEV host változóknak,  
SQL hibakezelés megadása  
EXEC SQL.  
    UPDATE HALLGATO  
    SET KNEV = :HLG-KNEV  
    WHERE HID = : HLG-HID  
END-EXEC.
```

5.3. ábra: Host változó használata.

A host változót a hozzá kapcsolódó oszlop definíciójával kompatibilisen kell deklarálni. Numerikus adatokat az SQL általában átkonvertál, de túlsordulásnál természetesen hibát jelez. Karakteres megfeleltetésnél azonos hosszra kiegyenlít illetve levág. Kivétel ez alól az adatbázisba való beírás. Itt a host változó teljes tartalma be kell férjen az oszlopba.

Az egyes host változókat kiegészíthetjük egy indikátorral is. Az indikátort a programban 2 bájt hosszúságú, előjeles, numerikus változóként kell deklarálnunk. Ilyenkor az SQL utasítások bizonyos hibák, figyelmeztetések felléptekor nem fognak 0-tól eltérő *SQLCODE/SQLSTATE* eredményt adni. A hibára, figyelmeztetésre az utasítás végrehajtása után az indikátor 0-tól különböző értéke utal. Különösen hasznos ez, ha az eredmény *NULL* értéket szolgáltat(hat), mert ezt a hagyományos programnyelvekben nem tudjuk lekezelni (ld. 1. példa).

Példa:

1. Meg akarjuk kapni programban azoknak a 4. éves hallgatóknak a számát és osztályzatuk átlagát, akik az '*Adatbázis*' tantárgyból vizsgáztak. Mivel az eredmény biztosan csak egy sort szolgáltat, használhatjuk a *SELECT ... INTO* utasítást.

Utasítás-1:

```
exec sql.  
    select count(*), avg(osztalyzat) into :letszam, :atlagosztalyzat from oktatás  
    where evf = 4 and tmegnev = 'Adatbázis'  
end-exec.
```

Eredmény: $SQLCODE=SQLSTATE=0$ és a LETSZAM illetve az ATLAGOSZTALYZAT változók tartalmazni fogják a keresett értékeket, ha volt legalább egy a feltételeknek megfelelő hallgató és az le is vizsgázott. Különben a NULL érték programbeli kezelhetetlensége miatt SQL hibát vagy figyelmeztetést kapunk.

Indikátor használatával mindig $SQLCODE=SQLSTATE=0$ eredményt kapunk. Ha valamelyik eredmény NULL érték lenne, akkor azt az indikátorában a -1 érték mutatja. A programban kell gondoskodnunk arról, hogy ilyen esetben másképpen dolgozzuk fel a kapott értékeket. Az IND-ATLAGOSZTALYZAT indikátor mezőt a befogadó programban kell deklarálnunk. A LÉTSZÁM-hoz nem szükséges indikátort hozzárendelnünk, mert ha egy rekord sem felel meg a feltételeknek, a COUNT(*) függvény 0-t ad eredményül.

Utasítás-2:

```
exec sql.
    select count(*), avg(osztalyzat) into :letszam
    ,:atlagosztalyzat:ind-atlagosztalyzat from oktatás
    where evf = 4 and tmegnev = 'Adatbázis'
end-exec.
if letszam = 0 then                . . . nem volt egy hallgató sem feldolgozása.
if ind-atlagosztalyzat = -1 then    . . . nem számolható átlag feldolgozása.
```

5.7.4. Kurzor használata

Ha biztosak vagyunk benne, hogy a kiválasztás eredménye legföljebb egy rekord lehet, akkor programban használhatjuk az 5.6.3.2. pontban tárgyalt *SELECT* utasítást a *SELECT ... INTO* formában. A kiválasztási lista oszlopainak tartalma egyenként az *INTO* klauzulában megadott host változókba kerül.

A hagyományos programnyelvek egyszerre csak egy rekorddal tudnak dolgozni. Ezért, ha a *SELECT* utasítás több sort is szolgáltathat, szükségünk van egy *kurzorra*. Ez meghatározza, hogy a kiválasztott sorok közül éppen melyik kerüljön feldolgozásra a programban.

Ahhoz, hogy a kurzorral dolgozhassunk, azt

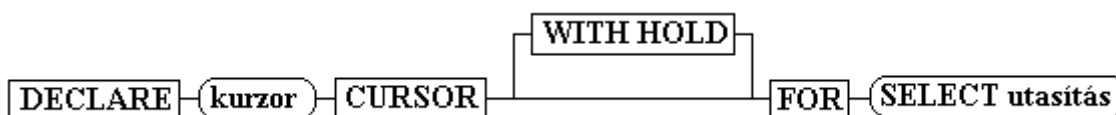
- először definiálni kell (*DECLARE*),
- a már definiált kurzort meg kell nyitni (*OPEN*),
- rá kell állni a feldolgozandó sorra és azt be kell olvasni a programba (*FETCH*),
- az utolsó sor feldolgozásának befejeztével be kell zárni a kurzort (*CLOSE*).

Ezeket a csak programban használható utasításokat az alábbiakban ismertetjük.

DECLARE

Funkciója: Kurzor deklarálása a befogadó programban.

Egyszerűsített szintaxisa:



Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
DECLARE, CURSOR, FOR	Kötelező kulcsszavak. Meghatározzák, milyen utasítás kerül végrehajtásra.
kurzor	A deklarált kurzor neve.
WITH HOLD	Elhagyható kulcsszavak. Ha megadjuk, akkor <i>COMMIT/ROLLBACK</i> után megmarad a kurzor pozíciója.
<i>SELECT</i> utasítás	Tetszőleges <i>SELECT</i> utasítás (ld. 5.6.3.2. pont), mely nem tartalmazhat <i>INTO</i> klauzulát.

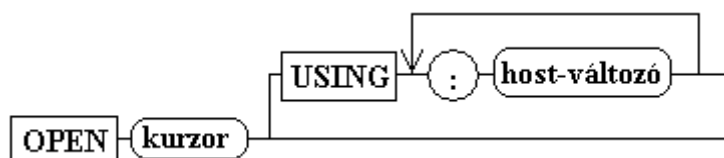
Megjegyzések:

- Minden olyan *SELECT* utasításra, amelyik több sort *olvashat be*, definiálnunk kell egy kurzort a program eljárási részében még a kurzor megnyitása (*OPEN*) előtt.
- Egy programban akárhány kurzort definiálhatunk és tarthatunk nyitva, de a kurzor neve a programon belül egyedi kell legyen.
- A kurzor deklarálása nem jelenti az adatok tényleges kiválasztását és beolvasását. Ez az *OPEN* hatására, a kurzor megnyitásakor történik meg.
- A *COMMIT/ROLLBACK* utasítások (ld. 5.6.4. pont) automatikusan bezárják a kurzort, kivéve, ha azt a *WITH HOLD* opcióval deklaráltuk. Ezért, ha ezeket nem a kurzor feldolgozásának a végén, hanem közben adjuk ki és nem adtuk meg *WITH HOLD*-ot, akkor a következő *FETCH*-nél a feldolgozás az első sortól fog újra folytatódni.

OPEN

Funkciója: Megnyitja a deklarált kurzort.

Egyszerűsített szintaxisa:



Kulcsszavak és paraméterek:

kulcsszó vagy paraméter	jelentése
OPEN	Kötelező kulcsszó. Meghatározza, milyen utasítás kerül végrehajtásra.
kurzor	A megnyitandó kurzor egyértelmű neve a programban.
USING	Elhagyható kulcsszó. Megadásával jelezzük, hogy a programból paramétereket adunk át a <i>SELECT</i> utasításnak.
host változó	Ennek a megnyitáskor aktuális értékével hajtódik végre az utasítás.

Megjegyzések:

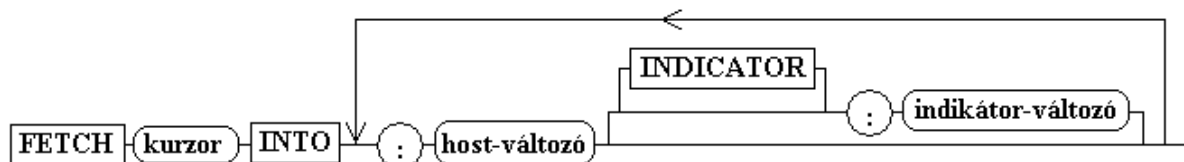
- A kurzort, mielőtt megnyitnánk, deklarálni kell.

- Az *OPEN* hatására a kiválasztott sor(ok) bekerülnek egy ideiglenes táblázatba. Ezt a táblázatot az SQL automatikusan kezeli. Sem a nevét, sem a méretét, sem másféle paraméterét nem kell sem a programban, sem a futtatási környezetben definiálnunk Tartalma, akármikor is dolgozzuk fel az egyes sorait, mindig a kurzor megnyitásakor fennálló állapotot jelzi.
- A megnyitás után a kurzor az ideiglenes táblázat első sora elé mutat. Az első *FETCH* utasítás pozicionálja az első sorra, a következők pedig mindig egy sorral tovább léptetik.
- A program kezdetekor minden kurzor lezárt állapotban van. Egy lezárt kurzort bármikor és akárhányszor megnyithatunk.

FETCH

Funkciója: A következő sorra pozicionálja a kurzort és beírja ennek a sornak az értékeit a host változókba,

Egyszerűsített szintaxisa:



Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
FETCH, INTO	Kötelező kulcsszavak. Meghatározzák, milyen utasítás kerül végrehajtásra.
kurzor	Ezt a kurzort pozicionálja az utasítás.
host változó	A kurzor aktuális pozíciójának megfelelő érték(ek) kerül(nek) be a host változó(k)ba.
INDICATOR	Olvashatóságot segítő, elhagyható kulcsszó. Jelzi, hogy az adott host változóhoz indikátort kapcsoltunk.
indikátor-változó	Ez kapcsolódik indikátorként az előtte megadott host változóhoz.

Megjegyzések:

- A *FETCH* utasítás mindig a kurzorral beolvasott ideiglenes táblázat *következő* sorára pozicionál. Az így kijelölt sorból a kiválasztási listával kiválasztott oszlopoknak az értékeit beviszi az *INTO* klauzulában megadott host változókba. A kiválasztási lista *n*-edik elemének értéke lesz az *n*-edik host változó értéke. Ha ez *NULL* érték is lehet, akkor jelöljük ki indikátor-változót is. Ez -1 lesz *NULL* érték beolvasásakor, amit a programban lekérdezhetünk.
- A kurzor által kijelölt sort módosíthatjuk, illetve törölhetjük az *UPDATE* illetve a *DELETE* utasítás *WHERE* klauzulájában megadott *WHERE CURRENT OF kurzornév* feltétellel. A *kurzornév* nevű kurzor pozíciója módosítás után változatlan marad. Törlés után a közvetlenül a törölt sor után következő sor *elé*

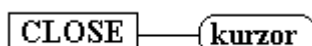
mutat és egy *FETCH*-cel tudjuk a soron következő rekordra pozicionálni (ld. 1. példa).

- Az ideiglenes táblázat végét a *FETCH* utasításra kapott *SQLCODE=100*, *SQLSTATE='02000'* illetve *WHENEVER-nél a NOT FOUND* feltétel bekövetkezte jelzi. Ezután a kurzort lezárjuk (*CLOSE*), hogy a kurzor által beolvasott rekordok zárait (ld. 8.3. alfejezet) felszabadítsuk.

CLOSE

Funkciója: Lezárja a megnyitott kurzort.

Egyszerűsített szintaxisa:



Kulcsszavak és paraméterek:

Kulcsszó vagy paraméter	Jelentése
CLOSE	Kötelező kulcsszó. Meghatározza, milyen utasítás kerül végrehajtásra.
kurzor	Ez a kurzor kerül bezárásra.

Megjegyzések:

- Csak megnyitott kurzort zárhatunk be. A bezáráskor megszüntetjük a kurzor által a beolvasott rekordokra automatikusan ráhelyezett zárat is.
- Ha lezárt kurzorra *FETCH* utasítást adunk ki, hibajelzést kapunk.

Példa:

1. A *FELDOLGOZAS* eljárásban feldolgozzuk az összes *MITHALLGAT* rekordot, ahol a hallgató egy vizsgaköteles tárgyból (*VIZSGA = 'I'* a *TANTARGY* táblázatban) levizsgázott. Ezzel egyidejűleg töröljük *MITHALLGAT*-ból azokat a rekordokat, amelyek azokra a hallgató-tantárgy (*HID-TID*) párosításokra vonatkoznak, melyekből kötelező a vizsga, de a hallgató nem vizsgázott le. Ezt, azaz *OSZTALYZAT NULL* értékét a hozzákapcsolódó indikátor -1 értéke jelzi.

Az ehhez szükséges programrészek (COBOL nyelven) az 5.4. ábrán láthatóak:

```

DATA DIVISION.
* HOST VÁLTOZÓK DEKLARÁLÁSA
  EXEC SQL
    BEGIN DECLARE SECTION
  END-EXEC.
01MTH-REKORD.
  05 MTH-HID                PIC X(5).
  05 MTH-TID                PIC X(4).
  05 MTH-GYAKPONT           PIC 999 COMP-3.
  05 MTH-OSZTALYZAT         PIC 9 COMP-3,
  
```



```

01 TNT-REKORD.
    05 TNT-TID                      PIC X(4).
    05 TNT-TMEGNEV                  PIC X(12).
    05 TNT-OKTATO                   PIC X(15).
    05 TNT-ORASZAM                  PIC 9 COMP-3.
    05 TNT-VIZSGA                   PIC X.
...
* INDIKÁTOR VÁLTOZÓK
    77 IND-MTH-OSZTALYZAT          PIC S9(4)      COMP-4.
    77 IND-TNT-OKTATO              PIC S9(4)      COMP-4.
...
    EXEC SQL
        END DECLARE SECTION
    END-EXEC.
...
PROCEDURE DIVISION.
* SQL HIBAKEZELÉS
    EXEC SQL
        WHENEVER SQLERROR GO TO HIBA
    END-EXEC.
    EXEC SQL
        WHENEVER SQLWARNING CONTINUE
    END-EXEC.
* KURZOR DEKLARÁLÁSA.
    EXEC SQL
        DECLARE BEOLVAS CURSOR FOR
        SELECT HID, TID, GYAKPONT, OSZTALYZAT,
        TMEGNEV,OKTATO
        FROM MITHALLGAT JOIN TANTARGY USING (TID)
        WHERE VIZSGA='I' ORDER BY HID,TID
    END-EXEC.
...
* KURZOR MEGNYITÁSA
    EXEC SQL
        OPEN BEOLVAS
    END-EXEC.
* KIVÁLASZTOTT SOROK BEOLVASÁSA..
    PERFORM FELDOLGOZAS THRU FELDOLGOZAS-VEGE
    UNTIL SQLCODE = 100.
* KURZOR LEZÁRÁSA.
    EXEC SQL
        CLOSE BEOLVAS
    END-EXEC.
...
FELDOLGOZAS.

```

```

EXEC SQL
  FETCH BEOLVAS INTO :MTH-HID,:
  MTH-TID,:MTH-GYAKPONT,
  MTH-OSZTALYZAT:IND-MTH-OSZTALYZAT,
  MTH-TMEGNEV,:MTH-OKTATO:IND-TNT-OKTATO
END-EXEC.
* HA NINCS OSZTALYZAT, TÖRLÉS, KÜLÖNBEN A REKORD
  FELDOLGOZÁSA.
  IF IND-MTH-OSZTALYZAT = -1 THEN
    EXEC SQL
      DELETE FROM MITHALLGAT
      WHERE CURRENT OF BEOLVAS
    END-EXEC
  ELSE
    FELDOLGAZAS-VEGE. EXIT.
  ...
* SQL HIBÁK KEZELÉSE.
  HIBA.

```

normál feldolgozás

SQL hibakezelés.

5.4. ábra: **Kurzor helyzete alapján való feldolgozás.**

5.8. Ellenőrző kérdések

1. Miért nem célszerű egy táblázat jellemzőit a *CREATE TABLE* utasítás szövege alapján ellenőrizni?
2. Miért nincsen *ALTER VIEW* utasítás? Mi helyettesíti ezt?
3. Az igen sok adatot tartalmazó *HALLGATO* táblázatra a felsoroltak közül melyik indexeket célszerű létrehozni:

```

I_HALLG_NEV (VNEV+KNEV együttesen),
I_HALLG_VNEV (VNEV), I_HALLG_NEV (VNEV,KNEV),
I_HALLG_KNEV (KNEV), I_HALLG_NEV (KNEV,VNEV)

```

(zárójelben az index oszlopai), ha a táblázatban gyakran keresünk

- a) csak név szerint,
 - b) csak vezetéknév szerint,
 - c) csak vezetéknév és keresztnév szerint együttesen,
 - d) vezetéknév és keresztnév szerint együttesen vagy vezetéknév szerint,
 - e) csak keresztnév szerint,
 - f) vezetéknév és keresztnév szerint külön-külön és együttesen is.
4. Ha egy táblázaton több oszlopot vagy oszlop kombinációt is kijelölhetünk elsődleges kulcsnak, akkor mit kell tennünk az adatok integritásának biztosítására az elsődleges kulcsnak nem kijelölt oszlopokkal?
 5. Igaz-e, hogy a kiválasztási listában mindig szerepelni kell annak az oszlopnak, amelyikre
 - a) az eredményt rendezni akarjuk,
 - b) a *WHERE* klauzulában feltételt adunk,
 - c) a *GROUP BY* klauzulában csoportokat képezünk.

6. Miért célszerű a nézetet ellenőrzéssel (*WITH CHECK OPTION*) létrehozunk? Csak olvasható nézetnél miért nincs erre szükségünk? Származik-e abból hátrányunk, ha a csak olvasható nézetet is ellenőrzéssel definiáljuk?
7. Idegen kulccsal összekötött táblázatokban a hivatkozó vagy a hivatkozott táblázatba kell-e előbb bevinnünk az adatokat? Mikor és hogyan térhetünk el ettől, de miért nem célszerű ez?
8. Meg akarjuk állapítani a legidősebb hallgató(k) összes adatát. Miért hibás az alábbi *SELECT* utasítás:

```
select vnev,knev,hid,max(szdatum), evf from hallgato;
```

Hogyan kellene átírni ezt az utasítást?

9. Igaz-e, hogy ha a *SELECT* utasítás eredményét több oszlop szerint rendezzük, akkor az mindegyik oszlopra vagy csak növekedő, vagy csak csökkenő lehet?
10. *DATE* típusú adatokat milyen rendezettségben célszerű megjeleníteni? Igaz-e ez, ha az adatokat karakterként definiáltuk, de érvényes dátum formában vesszük be?
11. Mi a különbség a *WHERE* utasítás kiegészítés nélkül kiadott *DELETE FROM TABLE ...* és a *DROP TABLE ...* utasítások között? Mikor melyiket célszerűbb használni?

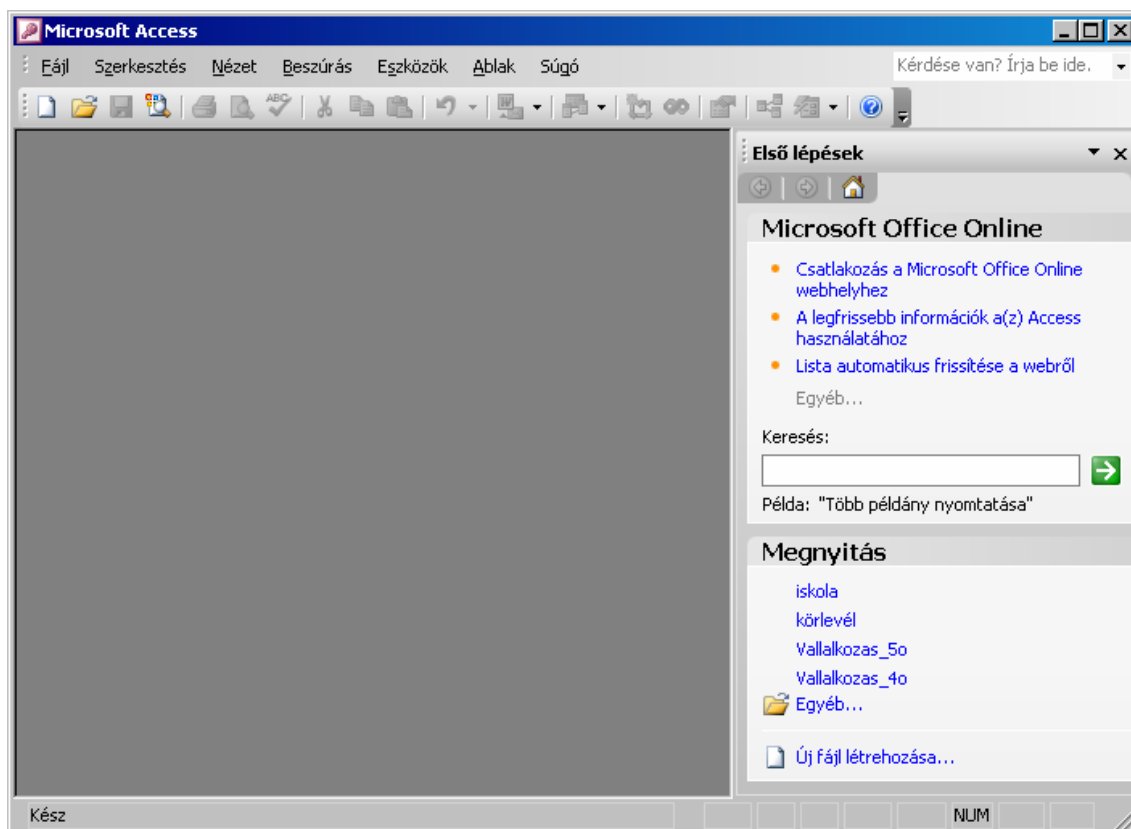
6. ADATBÁZIS-KEZELŐ RENDSZEREK A GYAKORLATBAN

Ebben a fejezetben két konkrét rendszerben, a Microsoft ACCESS-ben és a PHP MySQL-ben bemutatjuk, miként tudjuk megvalósítani a 4.2. alfejezetben megtervezett adatmodellre épülő adatbázist. Ehhez ismertetjük a két rendszer legfontosabb jellemzőit, majd létrehozunk segítségükkel az adatbázist.

6.1. MS ACCESS

A Microsoft Office programcsomag része az ACCESS adatbázis-kezelő. Könyvünkben a jelenlegi legfrissebb verziót, az ACCESS 2003-at mutatjuk be, de az elmondottak zöme érvényes a korábbi verziókra is és nagy valószínűséggel érvényes lesz a következő verziókra is. A program számos lehetőséget kínál kezdő és gyakorlottabb felhasználók számára adatbázisok felépítésére és az azokban lévő adatok kezelésére is. Emellett sok olyan felhasználóbarát lehetőséggel is rendelkezik, ami nem illik bele a relációs modell logikájába. Ezek közé tartoznak a különféle automatizmusok és varázslók, amik megkönnyítik a munkát, de nem gondolkodnak helyettünk. Könyvünk, már csak terjedelmi okok miatt sem törekedhet teljességre. Nem mutatjuk meg a program minden lehetőségét, inkább törekszünk egy példán keresztül azokat a módszereket bemutatni, amelyek segítségével biztonsággal létre tudunk hozni egy új, és módosítani tudunk egy korábban jól megtervezett adatbázist. A mintapéldánk a 4.2. alfejezetben ismertetett példa lesz néhány programspecifikus módosítással.

6.1.1. Szerkezet, felület, kezelés



6.1. ábra: Az ACCESS kezdőablaka

Az ACCESS ablakának felépítése az Office programcsomag többi programjához hasonló, de számos specifikus elemet tartalmaz. A menüsor és az eszköztárak is dinamikusan változnak aszerint, hogy éppen mivel dolgozunk. Amíg egyetlen adatbázisunk sincs megnyitva, addig a menüsor csak a szokásos menüket tartalmazza: *Fájl, Szerkesztés, Nézet, Beszúrás, Eszközök, Ablak, Súgó*. A megjelenő eszköztár az *Adatbázis*, de csak az új adatbázis létrehozása, egy meglévő adatbázis megnyitása, a keresés és a súgó ikonok élnek. Az ablak jobb oldalán a *munkaablak* található, melynek segítségével az adott szituációban leggyakrabban használt feladatok végezhetőek el. A munkánkhoz nem feltétlenül van rá szükségünk. A *Nézet* menü segítségével bármikor ki- és bekapcsolhatjuk. Az adatbázis-kezelő ablaka a 6.1. ábrán látható.

Új üres adatbázis létrehozásakor azonnal meg kell mondanunk, hogy milyen néven és hova mentjük el az adatbázist. Innen kezdve minden változást ugyanebbe az állományba tudunk menteni és a későbbiekben újra megnyitni. Ha már létező adatbázist nyitunk meg, akkor azzal minden, az adott adatbázisban készített objektumot láthatunk.

Az adatbázis-kezelő objektumai külön ablakokban nyílnak meg a program ablakán belül. Maga az adatbázis is egy ablakban jelenik meg, melynek neve megegyezik az adatbázis nevével. Az adatbázis ablakában szintén van eszközsor. Ezen kívül az ablak bal oldalán található két lista. Az első lista az objektumtípusok listája, a második pedig az általunk létrehozott csoportok listája. A csoportokba tetszőlegesen összeválogatott objektumok hivatkozásait gyűjthetjük össze. Az első listában szereplő objektumtípusok:

- Táblák,
- Lekérdezések,
- Űrlapok,
- Jelentések,
- Lapok,
- Makrók,
- Modulok.

Egy objektumcsoportot kiválasztva az ablak jobb oldalán listában találjuk az új objektumok létrehozására szolgáló ikonokat és a meglévő objektumainkat. Bármelyiket választva egy új ablakot tudunk megnyitni. Az ablakok között a szokásos módokon, például a tálcán lévő gombokkal tudunk váltani. Az ablakok külön-külön átméretezhetőek, bezárhatóak. Ha azonban az adatbázis ablakát zárjuk be, akkor azzal minden objektum-ablakot és magát az adatbázist is bezárjuk. A *menü* és az *eszköztárak* mindig az aktív ablaknak megfelelően változnak. A konkrét menüpontokat az egyes feladatoknál fogjuk tárgyalni.

Az objektumok megnyitására különböző nézetekben van lehetőség. Minden objektumtípusnál szerepel a *tervező nézet*, melyben az adott objektum szerkezetét tudjuk megnézni és módosítani. Ezen kívül a táblákat és lekérdezéseket *adatlap nézetben*, az űrlapokat *űrlap nézetben* és a jelentéseket *nyomtatási kép nézetben* is megtekinthetjük. Ezekre az adott pontokban visszatérünk.

6.1.2. Adatbázis tervezés

Tekintsük át az adatbázis létrehozásának lépéseit. Bármelyik adatbázis-kezelő programot is választjuk munkánkhoz, először nem árt papíron, vagy legalább gondolatban megtervezni az adatbázist, amelyet létre kívánunk hozni. Ehhez a könyv 3. és 4. fejezete nyújt segítséget. Az alábbi pontokban azt tekintjük át, hogy a megtervezett adatbázist hogyan tudjuk létrehozni az ACCESS adatbázis-kezelőben. Sorra vesszük a táblák, feltételek, kapcsolatok és indexek létrehozásának módjait.

6.1.2.1. Táblák

Táblák létrehozására két alapvető módon van lehetőségünk. *Varázsló* segítségével és *tervező nézetben*. A varázslóval elsősorban típusfeladatoknak megfelelő objektumokat állíthatunk elő. Elvileg lehetőség van a táblázat létrehozására az adatok beírásával is, de ez ellentmondana annak a logikának, hogy kettéválasztjuk a tervezést az adatkezeléstől. Mivel a varázslóval létrehozott táblázatot is tervező nézetben tudjuk módosítani, a varázsló lépéseit röviden tekintjük át és aztán a tervező nézetet részletezzük.

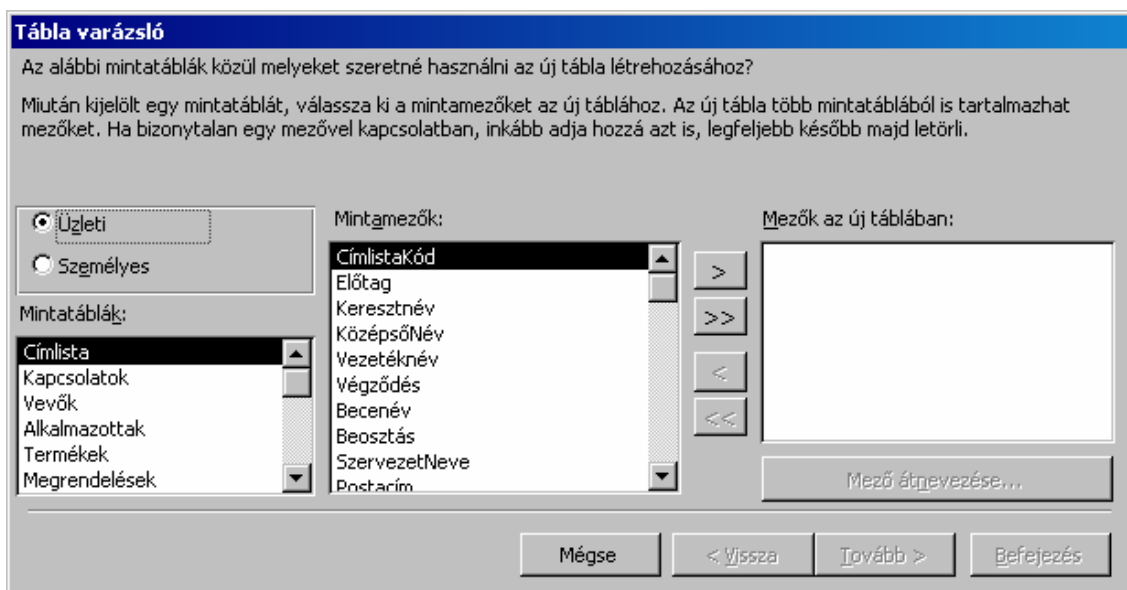
Ez a művelet a CREATE TABLE SQL utasításnak (ld. 5.14. táblázat.: pont) felel meg.

Példa

Három táblát fogunk létrehozni: HALLGATÓ, TANTÁRGY, MITHALLGAT. Az ACCESS magyar verziójában használhatunk az objektumok elnevezésében ékezetes karaktereket is, így maradhatunk a 4.2. alfejezet példájában használt elnevezéseknél.

Az első lépésünk legyen a HALLGATÓ tábla létrehozása varázsló segítségével. A varázsló megnyitásának lehetséges módjai például a *Beszűrés* menü *Tábla* menüpontjára megjelenő *Új tábla* ablakban a *Tábla varázsló* választása, vagy az adatbázis ablakában a *Táblák* objektumcsoportot választva a jobb oldali listában a *Tábla létrehozása varázsló segítségével* pont választása.

Bármelyiket választjuk, a 6.2. ábrán látható *Tábla varázsló* ablakhoz jutunk. Itt választhatunk, hogy az üzleti, vagy a személyes táblázatok között keressük meg a számunkra megfelelőt. A *HALLGATÓK* táblát az üzleti lista vége felé fogjuk megtalálni.



6.2. ábra: Tábla varázsló

Megjelölve a megfelelő táblát a lehetséges attribútumok hosszú sorából kiválaszthatjuk a számunkra szükségeseket. A kiválasztott mezőket a > nyíl segítségével tudjuk a jobb oldali listára tenni és a < nyíl segítségével tudjuk visszavenni onnan. Ha valamelyik mező neve nem pontosan az, amire szükségünk van, akkor lehetőségünk van az átnevezésre is a *Mező átnevezése* gombbal. A példánkban lévő mezők közül csak a *vezetéknév*, a *keresztnév* és a *hallgatói azonosító* szerepel a listában. Ezt a hármat helyezzük a kiválasztott listára. A *Mező átnevezése* gomb segítségével nevezzük át őket *vnev*, *knev* és *hid* névre.

A *Tovább* gomb megnyomása után módosíthatunk a tábla nevén is. Ezt írjuk át *HALLGATÓ*-ra. Ebben a pontban dönthetjük el azt is, hogy a program önállóan válassza-e ki az elsődleges kulcsot, vagy mi adjuk meg. Ha a programra bízunk, akkor azt a mezőt fogja választani, aminek a nevében eredetileg az azonosító szó szerepelt. Ennek a mezőnek az adattípusa *Számláló* lesz. Mivel számunkra szöveg adattípus kell, válasszuk azt a lehetőséget, hogy mi határozzuk meg az elsődleges kulcsot. (Az adattípusokra kicsit részletesebben még kitérünk.)

Ebben az esetben a megjelenő ablakban egy listából választhatjuk ki a megadott mezőkből az elsődleges kulcsot, és választókapcsoló segítségével megadhatjuk, hogy *Számláló*, *Szám* vagy *Szöveg* adattípusú legyen. Válasszuk a harmadik lehetőséget (*Számok és betűk, amelyeket én írok be új rekordok hozzáadásakor*).

A *Tovább* gomb megnyomása után minden adat rendelkezésre áll az adatbázis-kezelőnek, hogy a táblát létrehozza, már csak azt kell eldöntenünk, hogy a létrehozás után mivel folytassuk a munkát. Mivel a tábla még nem teljesen olyan, amilyenre terveztük, válasszuk az *A táblaterv módosításával* választókapcsolót a *Befejezés* gomb megnyomása előtt. Ha minden mezőt megtaláltunk volna a listában, akkor is érdemes a táblatervet megnézni, mert a varázsló elrejtí előlünk az adattípusokat és egyéb beállításokat, amelyeket tervező nézetben ellenőrizhetünk és módosíthatunk.

Ha már létezett volna az adatbázisban másik táblánk, a varázsló az utolsó lépés előtt kiegészült volna még egy ablakkal, amelyben megadhattuk volna a létező táblákhoz a kapcsolatokat. A kapcsolatok beállítására a 6.1.2.6. pontban térünk vissza.

Mezőnév	Adattípus	Leírás
hid	Szöveg	
vnév	Szöveg	
knév	Szöveg	

Mezőtulajdonságok

Általános	Megjelenítés
Mezőméret	4
Formátum	
Beviteli maszk	
Cím	
Alapértelmezett érték	
Érvényességi szabály	
Érvényesítési szöveg	
Kötelező	Igen
Nulla hosszúság engedélyezése	Nem
Indexelt	Igen (nem lehet azonos)
Unicode-tömörítés	Nem
IME-mód	Nem beállított
IME-mondatmód	Nincs konverzió
Intelligens címkék	

A mezőnév legfeljebb 64 karakter hosszú lehet a szóközöket is beleértve. Az F1 billentyű lenyomására megjelenik a súgó a mezőnevekről.

6.3. ábra: A HALLGATÓ tábla tervező nézetben

A *Befejezés* gomb megnyomása után létrejön a tábla az általunk megadott néven és azonnal meg is nyílik egy új ablakban tervező nézetben. Nézzük meg, hogy a kiválasztott mezőknek milyen adattípusa lett (6.3. ábra) és hogy a hiányzó mezőket hogyan tudjuk létrehozni. Rögtön észrevehetjük például, hogy a *hid* mező adattípusa 4 hosszúságú szöveg

lett, a példánkban viszont 5 karakteren tároljuk a hallgatói azonosítót. Ezt az alábbiakban módosítani fogjuk.

Az ablak felső részében egy-egy sor felel meg egy-egy mezőnek. Minden mezőnek van neve, adattípusa és lehet leírása amelyben bővebben le lehet írni, hogy mit tárolunk az adott mezőben. A sorok elején a szürke lapocskákkal lehet kijelölni, hogy melyik mező az aktuális (ezeket jelölőlapkának hívjuk). Az elsődleges kulcsot a jelölőlapkán látható kulcs szimbólum jelzi (🔑).

A lehetséges adattípusok a következők¹:

- Szöveg: az SQL *varchar*, (az Oracle *varchar2*) típusának felel meg. Maximum 255 karaktert (számot, betűt, írásjelet) tud tárolni.
- Feljegyzés: az SQL *long* típusának felel meg. 255 karakternél hosszabb szöveg. Legfeljebb 65 536 karaktert tud tárolni.
- Szám: numerikus adatok tárolására alkalmas, pontossága a megadott mezőmérettől függ. A mezőméret megadását lejjebb tárgyaljuk. Formájától függ, hogy az SQL *number*, *integer*, vagy más numerikus adattípusának felel-e meg.
- Dátum/Idő: az SQL *date* adattípusa. Dátumot és időpontot jelentő adatok tárolására alkalmas. A formátum megadását lejjebb tárgyaljuk.
- Pénznem: numerikus adatok fixpontos tárolására alkalmas. Pontossága 15 számjegy és 4 tizedesjegy. A számítások során nem kerekít.
- Számláló: automatikusan egyedi számot ír be egy-egy rekord hozzáadásakor.²
- Igen/Nem: az SQL *boolean* adattípusának felel meg. Egy biten tárol igaz vagy hamis értéket.
- OLE objektum: Microsoft Word vagy Microsoft Excel dokumentumok, képek, hangok és más programban készült bináris adatok tárolására alkalmas. Az OLE objektumok csatolása vagy beágyazása OLE objektum típusú mezőkben lehetséges a Microsoft ACCESS táblákban. Az OLE objektumok vezérlőelemek segítségével jeleníthetők meg az űrlapokon és a jelentésekben.
- Hiperhivatkozás: hiperhivatkozás tárolására alkalmas, maximum 64 000 karakter.
- Keresés varázsló: segítségével olyan mező hozható létre, amely lehetővé teszi, hogy a beírandó értéket egy listából válasszuk ki. Ezt a listát mi is létrehozhatjuk a varázsló segítségével, vagy ha már van másik táblánk az adatbázisban, akkor annak mezőit is megadhatjuk listának. Ha a lista több mezőből áll, akkor megadhatjuk, hogy melyik mező értéke kerüljön tárolásra a táblában. Ha csak néhány érték közül választhatunk, akkor ez nagyon megkönnyíti az adatbevitelt.

Az ablak alsó részében mindig az aktuális mező részletesebb tulajdonságait láthatjuk. Hogy pontosan melyek ezek a tulajdonságok, az attól függ, hogy az adott mező milyen adattípusú. Nézzük meg a leggyakrabban használt tulajdonságokat:

- *Mezőméret*: Szöveg adattípusnál a mezőbe írható karakterek maximális száma. 1 és 255 közötti érték. Szám adattípusnál ezzel adhatjuk meg a tárolni kívánt számok pontosságát a 6.1. táblázat alapján. A többi adattípus mérete meghatározott, így nem állítható. A gazdaságos adattárolás érdekében érdemes mindig a lehető legkisebb mezőméretet választani, ami még megfelel az adatok tárolására.

¹ Az SQL adattípusokat lásd az 5.4. alfejezetben.

² Ez megfelel annak, hogy a mező értékét egy az adott táblázatra vonatkozó önálló sorszámgenerátor típusú (Sequence) SQL objektumból vesszük. (ld. 5.3.6. pont)

6.1. táblázat: A Szám adattípus mezőmérete

Mezőméret	Leírás	Decimális pontosság	Tárolási méret
Bájt	0 és 255 közötti egész számok	nincs	1 bájt
Decimális	10 ³⁸ -nál kisebb abszolút értékű számok pozitív és negatív irányban is	28	12 bájt
Egész	-32 768 és +32 767 közötti egész számok	nincs	2 bájt
Hosszú egész	-2 147 483 648 és +2 147 483 647 közötti egész számok	nincs	4 bájt
Egyszeres	lebegőpontos ábrázolás 4 bájton	7	4 bájt
Dupla	lebegőpontos ábrázolás 8 bájton	15	8 bájt

- *Tizedeshelyek:* Szám és Pénznem adattípusnál megadható a megjeleníteni kívánt tizedeshelyek száma.
- *Új érték:* csak a Számláló adattípusnál adható meg: lehet növekvő, vagy véletlenszerű.
- *Formátum:* Szöveg és Feljegyzés adattípusnál speciális karakterek segítségével megadhatjuk, hogy a beírt adat milyen formátumban jelenjen meg és ; után, hogy NULL érték, vagy nulla hosszúságú karakterlánc esetén mi jelenjen meg a cellában. A rekord tényleges tartalmát nem változtatja meg, csak a megjelenítés módját. Néhány használható szimbólum: @: szövegkarakter, <: minden karakter kisbetűs, >: minden karakter nagybetűs (a < és > jelek közül csak egyet lehet alkalmazni és az az egész karaktersorra érvényes lesz). Példák a 6.2. táblázatban láthatók. Dátum/Idő és Szám adattípusnál egy listából választhatjuk ki a dátum és a számformátumot. Például az Általános dátum formátum a dátumot és az időt is mutatja másodperc pontossággal és a hónapokat kétjegyű számmal jelöli, a Százalék formátum a számot százalékos formában két tizedesjegy pontossággal jeleníti meg. Igen/Nem adattípusnál a formátumnál adhatjuk meg, hogy mi legyen a két érték, ami közül választani lehet: igaz/hamis, igen/nem, be/ki. OLE objektumnál nincs ilyen tulajdonság.

6.2. táblázat: Példák formátum megadására

Beállítás	Adat	Megjelenítés
@@-@@	1234	12-34
	3	- 3
	ASGH	AS-GH
>	monaco	MONACO
	Monaco	MONACO
	Monaco	monaco
	monaCO	monaco
@;"Nincs"	Monaco	Monaco
	NULL érték	Nincs
>(@@)-@@@@;"Hiányzik"	Monaco	(MO)-NACO
	Paris	(P)-ARIS
	California	CALI(FO)-RNIA
	NULL érték	Hiányzik

- *Beviteli maszk*: megadhatjuk, hogy milyen formában lehessen bevinni az adatokat. A beírt adatok ebben a formában kerülnek tárolásra. Választhatunk a beépített maszkok közül, vagy létrehozhatunk saját maszkot is. A sor szélén megjelenő  ikonra kattintva megjelenik a *beviteli maszk varázsló*, aminek segítségével beállíthatjuk a kívánt maszkot, vagy ha már gyakorlottak vagyunk, be is gépelhetjük magunktól. *Feljegyzésnél*, *Számlálónál*, *Igen/Nem-nél* és *OLE objektumnál* nincs. Példák a 6.3. táblázatban láthatók.

6.3. táblázat: Példák maszk megadására

Karakter	Jelentés	Példa	Adat
0	számjegy, amit kötelező megadni	0000	1144
9	számjegy, amit nem kötelező megadni	(90)9000-000	(30)5473-578 (1) 444-444
#	számjegy, szóköz, vagy előjel	#999	-20 2000
L	betű, amit kötelező megadni	L00	A11 Z02
?	betű, amit nem kötelező megadni	??99	AS99 D78 12
&	bármilyen karakter, amit kötelező megadni	&00	A11 456
.,:;- / ()	határolójelek, amik ebben a formában megjelennek	90:00	12:00 9:30
\	az utána következő karakter változatlanul megjelenik	L\L00	gL11 AL12

- *Cím*: adatlap nézetben, űrlapon, jelentésben az itt megadott néven szerepel a mező. Ha nem adjuk meg, akkor a mező neve egyben a címe is¹.
- *Alapértelmezett érték*: ha semmit nem írunk be, akkor ez az érték kerül tárolásra a mezőben. Mindig csak az újonnan bevitt rekordokra vonatkozik, visszamenőleg nem. *Számlálónál* és *OLE objektumnál* nem adható meg. Megfelel a CREATE TABLE utasítás DEFAULT paraméterének.
- *Érvényességi szabály*²: a sor szélén megjelenő  ikon segítségével megnyitható kifejezés-szerkesztő ablakban adhatunk meg függvények és operátorok segítségével szabályt, feltételt arra, hogy milyen adatokat vihetünk be az adott mezőbe. Részletesebben a következő pontban tárgyaljuk. *Számlálónál* és *OLE objektumnál* nem adható meg.
- *Érvényesítési szöveg*: megadhatjuk, hogy ha az érvényességi szabálynak nem megfelelő adatot ír be a felhasználó, akkor milyen hibaüzenetet kapjon. *Számlálónál* és *OLE objektumnál* nem adható meg.
- *Kötelező és Nulla hosszúság engedélyezése*: az ACCESS megkülönbözteti, hogy egy mező értéke nulla hosszúságú karakterlánc, vagy NULL érték (ld. 5.3.7.

¹ A *Cím* elnevezés megtévesztő lehet, Access-ben nem a memóriacímet jelenti, hanem a megjelenítendő oszlopnevet. A könyv további részében a *Cím* helyett a *Név* megnevezést fogjuk használni.

² Ez megfelel az SQL-ben definiált Constraint CHECK paraméterének (ld. 5.14. táblázat: pont).

pont). Ha egy mező értékét kötelező megadni, de a nulla hosszúság engedélyezett, akkor lehetőség van például a SPACE megnyomásával nulla hosszúságú karakter bevitelére. A nulla hosszúság engedélyezése *Szám*, *Dátum/Idő*, *Pénznem* és *Igen/Nem*-nél nem adható meg.

- *Indexelt*: a 6.1.2.5. pontban tárgyaljuk részletesebben.

Ha bármit változtatunk a tábla definícióján, akár egy-egy mező tulajdonságait, akár a mezők sorrendjét, akár új mezőket adunk hozzá, vagy törölünk ki, el kell mentenünk a táblát. Ha a mentést elmulasztanánk, a tervezésből kilépéskor rákérdez a program. Menteni a szokásos módokon tudunk: a *Fájl* menü *Mentés* parancsával, az eszköztár  ikonjával, vagy a *Ctrl+s* billentyűkombinációval.

Példa

1. Módosítsuk a HALLGATÓK tábla meglévő három mezőjét a következőknek megfelelően:

Mezőnév	Leírás	Mezőméret	Kötelező
vnév	Hallgató vezetékeve	25	igen
knév	Hallgató keresztnéve	25	nem
hid	Hallgató azonosító	5	igen

Ehhez a *tábla tervező nézetében* módosítanunk kell ezeknek a mezőknek a paramétereit. Ez valójában az SQL ALTER TABLE utasítás (ld. 5.6.2.3. pont) végrehajtását jelenti.

Ha szeretnénk a mezők sorrendjét megváltoztatni (az ACCESS automatikusan az elsődleges kulcsot tette előre), akkor egér segítségével kijelölés után vonszolással át tudjuk helyezni a mezőt.

Objektum, így tábla törlésére is az adatbázis ablakból van módunk. Kijelöljük a törölendő objektumot az adatbázisablakban és kiadjuk a törlés parancsot: a *Szerkesztés* menü *Törlés* parancsával, az eszköztár  ikonjával, vagy a *Del* billentyűvel. Ez az SQL DROP TABLE utasításának felel meg (ld. 5.6.2.2. pont).

6.1.2.2. Feltételek

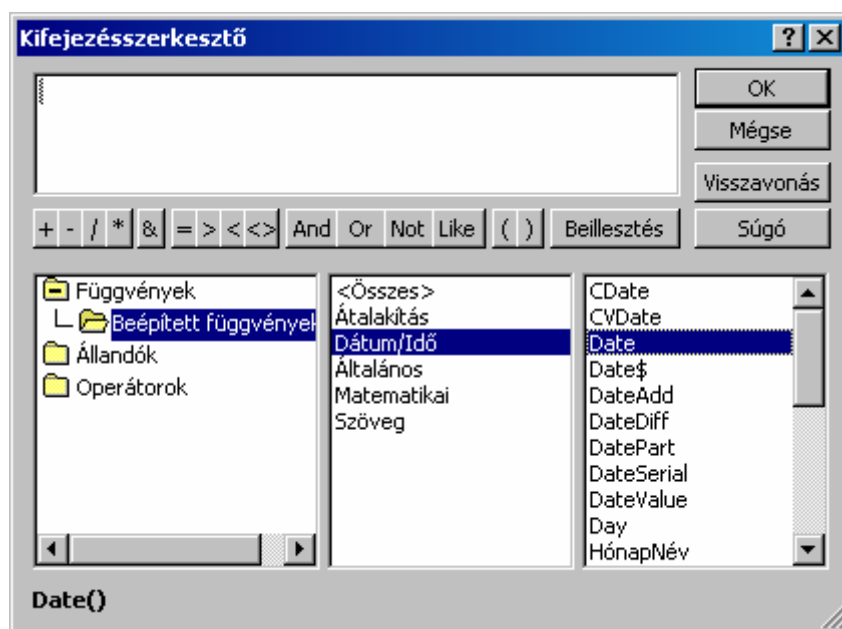
Azért emeltük ki a feltételek megadását a tulajdonságok közül, mert a *kifejezés-szerkesztő* ablak segítségével nem csak a tábla definiálásakor találkozhatunk és nem csak oszlopfeltételeket határozhatunk meg vele.¹ A 6.4. ábrán a *Kifejezés-szerkesztő* ablakát látjuk. Az ablak felső részében van lehetőségünk a kifejezés megszerkesztésére. Ehhez használhatjuk a billentyűzetet, különleges karaktereket beírhatunk a beviteli doboz alatt található gombokkal és beépített kifejezéseket beilleszthetünk az ablak alsó felében található listák segítségével.

A beépített kifejezések csoportosítva találhatók meg. Az elsődleges csoportosítás látható a bal oldali listában. Itt mindig megjelennek a *függvények*, az *állandók* és az *operátorok*. Ezen kívül más csoportok is megjelenhetnek, például az adatbázis táblái. A függvények még ebben a listában tovább bonthatók, ha az előtte megjelenő  ikonra kattintunk. Ha nincsenek importált, vagy a felhasználó által létrehozott függvények, akkor csak a beépített függvények találhatók itt.

¹ SQL-ben is több szituációban van szükségünk feltételek megfogalmazására (ld. 5.3.4. pont). Tábla definiálásakor a CONSTRAINT kulcsszó után tudunk az oszlopokra, vagy az egész táblára vonatkozó feltételt megadni (ld. 5.14. táblázat.: pont), lekérdezésekben a WHERE és a HAVING kulcsszavak után (ld. 5.6.3.2. pont).

A baloldali csoportok közül választva a középső listában újabb csoportosítás jelenik meg, a jobboldaliban pedig a kiválasztott csoportba tartozó elemek. A csoportok közül az első mindig az <Összes>. A beépített függvényeknél a függvénykategóriákat látjuk (pl.: *Matematikai*, *Szöveg*), az állandóknál csak <Összes> van, az operátoroknál a csoportok *Aritmetikai*, *Összehasonlítás* és *Logikai*. Ha szerepel a baloldali listában tábla, akkor a középső listában annak mezői láthatók, a jobb oldalon pedig az <Érték>.

A listákból dupla kattintással lehet egy-egy kifejezést a szerkesztő területre tenni a kurzor aktuális pozíciójába. A legtöbb esetben a jobboldali listából választunk. Ha a középső listából választunk, akkor a jobboldali lista első eleme kerül a szerkesztőbe. Függvények választásánál előfordul, hogy a kifejezés-szerkesztőbe a függvény úgy kerül be, hogy az argumentumai helyén a <<szöveg>> vagy a <<szám>> kiírás jelenik meg. Ha ezekre egyszer rákattintunk, akkor kijelöltté válnak és helyükre lehet írni, vagy beilleszteni a megfelelő argumentumot. Ha több kifejezést szúrunk be egymás után anélkül, hogy operátort tennénk közéjük, az ACCESS automatikusan a <<Kif>> szöveget jeleníti meg a kifejezések között, ezzel jelezve, hogy oda operátort vár.



6.4. ábra: Kifejezés-szerkesztő

Példák

1. Hozzuk létre a hiányzó oszlopokat a következő tulajdonságokkal:

Mező-név	Adattípus	Leírás	Mező-méret	Feltétel (Érvényességi szabály)	Érvényesítési szöveg	Alapértelmezett érték	Kötelező
szdátum	Dátum/Idő	Születési dátum					nem
évf	Szám	Évfolyam	Bájt	1 <= evf <= 5	Az évfolyam 1 és 5 közötti szám lehet	1	nem
tulaj	Szöveg	Az adat beírójának kódja	8			A saját azonosítónk	nem

A feltétel megadásán kívül minden egyértelmű, a feltétel megadását nézzük részletesebben. A táblázatban megadott módon nem lehet a feltételt beírni, csak két külön feltételként, amelyeket az és logikai operátorral fűzünk össze. A feltételt be is gépelhetjük, vagy a  ikonra kattintva meg is szerkeszthetjük. Ilyen egyszerűbb szabályok beírása általában egyszerűbb gépeléssel. Tehát a szabály:

$$\geq 1 \text{ AND } \leq 5.^1$$

A módosítások után mentjük a táblát!

Az adott példában nincs rá szükségünk, de lehetőség van nem csak oszlopfeltételt, hanem táblafeltételt is megadni, pl. hogy egyik mező értéke nem lehet nagyobb, mint egy másik mezőből képzett érték. Ehhez nem egy-egy mező tulajdonságait kell változtatnunk, hanem az egész tábla tulajdonságát. A *Nézet* menüben a *Tulajdonságok* menüpontra, vagy az *Adatbázis eszköztár*on a  ikonra kattintva megadhatjuk a táblázat tulajdonságait. Ezek között szintén van *érvényességi szabály* és *érvényesítési szöveg*. A többi tulajdonság közül a *rendezést* és a *szűrőt* a 6.1.3.4. pontban fogjuk tárgyalni.

2. Tétélezzük fel, hogy nincsenek csodagyerekek és különleges esetek, így minden elsőnek be kell töltenie idén minimum a 18., minden másodévesnek minimum a 19. életévét, és így tovább.

Ehhez meg kell tudnunk állapítani a hallgatók korát és szabályként be kell állítani, hogy az évfolyam nem lehet nagyobb az életkornál 17-tel kisebb számmal. Az életkor az aktuális és a születési dátum évének különbsége. Állítsuk be ezt az érvényességi szabályt:

$$\text{YEAR}(\text{DATE}()) - \text{YEAR}([\text{sz dátum}]) - 17 \geq [\text{évf}]$$

Láthatjuk, hogy a mezőnevek szögletes zárójelek között jelennek meg. Ezt a szabályt már érdemesebb a *kifejezés-szerkesztővel* írni be. A YEAR() és a DATE() függvény is a *dátumfüggvények* között található. DATE() az aktuális dátumot, YEAR() a *dátum típusú* argumentum év részét szolgáltatja. Állítsunk be érvényesítési szöveget is, például: „Ilyen fiatalon nem járhat erre az évfolyamra”! A módosítások után mentjük a táblázatot!

3. Hozzuk létre a TANTÁRGY és a MITHALLGAT táblákat is! Most az EGYETEM adatbázis ablakában ne a *Tábla Varázslót* hívjuk elő, hanem válasszuk a *Tábla létrehozása Tervező nézetben* pontot!

A már ismert ablak jön elő, de teljesen üresen. Ekkor minden mezőt nekünk kell begépelni.² A táblának az első mentéskor tudunk nevet adni, minden további mentésnél a már megadott névre ment. Hogy melyik mező legyen az *elsődleges kulcs* (*Primary Key*, ld. 4.4.2. és 5.14. táblázat.: pont), azt úgy tudjuk megadni, hogy kijelöljük a megfelelő mező(ke)t és kiadjuk a *Szerkesztés* menü *Elsődleges kulcs* parancsát, vagy az eszköztáron a  ikont megnyomjuk. Az ikon újra megnyomásával tudjuk az elsődleges kulcsot megszüntetni. Több mező kijelöléséhez használhatjuk a *Ctrl* gombot. Ha más mezőn állva nyomjuk meg az elsődleges kulcs ikont, akkor az előző elsődleges kulcs megszűnik.

¹ A logikai operátoroknál az ACCESS magyar változatában is az angol elnevezéseket kell használnunk.

² A TANTÁRGY tábla vizsga mezőjének adattípusát választhatnánk egykarakteres szövegnek, mint a 4.2. pont példájában, de megadhatjuk az Igen/Nem adattípust is és akkor nem kell külön érvényességi szabályt beállítanunk arra, hogy milyen értékeket adhatunk meg.

6.1.2.3. Operátorok és függvények

Az előző pontban, a feltételek megadásánál már találkoztunk néhány operátorral és függvénnyel. Ebben a pontban szeretnénk összefoglalni az ACCESS eszközeit folyamatosan párhuzamot vonva az SQL nyelv elemeivel (ld. 5.3.4. pont és 5.5. alfejezet).

Operátorok

Az ACCESS operátorai között megtaláljuk a 5.3.4. pontban tárgyalt *aritmetikai*, *összehasonlító* és *logikai* operátorokat is. A precedencia szabályok is ugyanúgy érvényesek. A legfontosabb operátorok, melyeket megtalálunk a *kifejezés-szerkesztőben*:

- Aritmetikai operátorok
 - összeadás, kivonás (+, −)
 - szorzás, osztás (*, /)
 - egészosztás (\): az osztás eredményének egészrészét adja eredményül
 - hatványozás (^)
 - maradékos osztás (Mod): két szám osztásakor keletkező maradékot adja eredményül
- Összehasonlító operátorok:
 - <, <=, <>, >, >=
 - Between
- Logikai operátorok:
 - And
 - Not
 - Or

Az operátorok között felsorolva nem találjuk, de nagyon hasznosak még az & és a Like operátorok.

Az & kétoperandusú operátor szövegösszefűzésre szolgál. Két szöveg kifejezést fűz össze egy szöveggé szóköz nélkül.¹ Például a "Fekete" & "Péter" művelet eredménye "FeketePéter". Ha a név összefűzését helyesen szeretnénk megoldani, akkor szükségünk van egy szóközre is. Tehát: "Fekete" & " " & "Péter". Ennek eredménye "Fekete Péter".

Az Like ugyanúgy működik, mint az SQL Like operátora (ld. 5.3.4.2. pont) azzal a különbséggel, hogy nem tesz különbséget kis- és nagybetűs karakterek között, és a helyettesítő karakterek (más néven joker karakterek) eltérnek, ahogy azt a 6.4. táblázat mutatja.

6.4. táblázat: **Helyettesítő karakterek és funkciójuk**

Helyettesítő karakter	Funkciója
* (csillag)	Akárhány karaktert helyettesíthet
? (kérdőjel)	Pontosan egy karaktert helyettesít
# (kettőskereszt)	Pontosan egy számjegyet helyettesít
[a-d]	A megadott karaktertartományba eső egy db. karaktert helyettesíti (a határok is benne vannak)

¹ Kifejezésben a szövegeket mindig idézőjelek közé kell írunk. A dátumot is javasolt idézőjelek, vagy kettőskereszt (#) között megadni, mert bizonyos dátumformátumokat (pl. 1999.10.11) csak így fogad el a rendszer. (Bizonyos esetekben, ha lefelejtjük a jeleket, akkor az ACCESS magától kiteszi azokat.)

Függvények

Az ACCESS függvényei között is megtaláljuk azokat a kategóriákat, amelyeket az SQL nyelvben, de lehet hogy eltérő néven (ld. 5.5. alfejezet). Az 6.5. táblázat áttekintést ad a leggyakrabban használt függvényekről. Néhány függvény nevét lefordították magyarra, de az SQL kifejezésekben ezek a függvények is az eredeti (angol) nevükön jelennek meg. A táblázatban a magyar függvényneveket – ahol voltak – zárójelben dőlt betűkkel jelöltük.

6.5. táblázat: Függvényelnevezések összehasonlítása SQL-ben és az ACCESS-ben

SQL függvények	Funkció	A függvény megjelenése ACCESS-ben
<i>Csoport függvények</i>		
AVG (numerikus argumentum)	Átlag	Avg(kif)
COUNT(*) vagy COUNT(argumentum)	Darabszám	Count(kif)
MAX(argumentum)	Maximum	Max(kif)
MIN (argumentum)	Minimum	Min(kif)
SUM (numerikus argumentum)	Összeg	Sum(kif)
<i>Karakter függvények</i>		
CONCAT(char-1,char-2)	Szövegösszefűzés	& operátor
INITCAP(char)	A szókezdő karaktereket nagybetűssé alakítja, de a többi karakteren nem változtat.	StrConv(karakterlánc;átalakítás) A átalakítás paramétert számmal lehet megadni. A 3 felel meg az SQL függvénynek
LENGTH(char)	Karakterek száma	Len(szöveg)
LOWER(char) UPPER(char)	Kisbetűssé és nagybetűssé alakít	LCase(szöveg) UCase(szöveg)
LTRIM(char) RTRIM(char)	Levágja a kezdő ill. a befejező szóközöket	LTrim(szöveg) Rtrim(szöveg)
REPLACE(char,c1,c2) TRANSLATE (char,c1,c2)	Karakterek kicserélése	Replace(szöveg;keresett;csere) (Csere)
SUBSTR(char,n,m)	Az argumentum egy részletét adja eredményül	Mid(szöveg;kezdet;hossz)
<i>Dátum és numerikus függvények</i>		
ADD_MONTHS	Hónap hozzáadása	DateAdd(időtartam;szám;dátum) (nem csak hónapot tud hozzáadni, a hónap hozzáadáshoz az időtartamnál „m”-et kell megadni.)
LAST_DAY	Hónap utolsó napja	ennek megfelelő függvény nincs
ROUND	Kerekítés	Round(szám;tizedeshelyek) (Kerek) (10 pozitív hatványaira nem tud kerekíteni, dátumok esetén számként kezeli a dátumokat)

SQL függvények	Funkció	A függvény megjelenése ACCESS-ben
SYSDATE vagy CURRENT DATE, CURRENT TIMESTAMP	Aktuális dátum ill. idő	Date(), Time()
TRUNC	Dátum vagy szám lerövidítése	ennek megfelelő függvény nincs
YEAR, MONTH, DAY	Dátum megfelelő részének kivágása	Year, Month, Day
<i>Konvertáló függvények</i>		
TO_CHAR vagy VARCHAR	Karakterre konvertálás	CStr(kif)
TO_DATE vagy DATE	Dátumra konvertálás	CDate(kif)
TO_NUMBER vagy DECIMAL	Numerikus adatra konvertálás	CByte, CDb1, CDec, CInt stb.

6.1.2.4. Táblák importálása

Az ACCESS lehetőséget nyújt arra, hogy más programmal készített, más adatbázisban lévő táblázatokat átvegyünk az aktuális adatbázisunkba. Ehhez az adatbázis ablakának kell aktívnak lennie. A *Fájl* menüben találjuk a *Külső adatok átvétele* pontban az *Importálást*. A megjelenő *Importálás* ablak a szokásos megnyitás ablakhoz hasonló. Itt megkereshetjük azt az állományt, amelyikben az a táblázat található, amelyiket importálni szeretnénk. A *Fájltípus* listában megadhatjuk, hogy milyen fájlok között keresünk. Itt láthatjuk, hogy nem csak adatbázisokból vehetünk át táblázatot, hanem pl. táblázatkezelővel készített táblázatból, vagy egyszerű szöveges állományból is.

Attól függően, hogy milyen típusú állományt választottunk, a következő lépésekben további információkat kell megadnunk, hogy pontosítsuk a táblázat átvételének módját.

Ha ACCESS adatbázisból importálunk, akkor nem csak táblázatokat, hanem egyéb objektumokat is átvehetünk: lekérdezéseket, űrlapokat, jelentéseket, stb. A *Beállítások* gomb megnyomására lenyíló területen megadhatjuk, hogy a táblának csak a definícióját, vagy a benne lévő adatokat is átvesszük-e, hogy a lekérdezéseket lekérdezésként, vagy táblaként importáljuk és hogy az adott objektumon kívül mást is átveszünk-e (pl. a tábla kapcsolatait).

Ha olyan objektumot importálunk, amely valamilyen módon kapcsolódik más objektumokhoz, akkor ügyelnünk kell arra, hogy vagy a kapcsolódó objektumokat is átvegyük, vagy a mi objektumaink között is legyen megfelelő objektum. Például ha importálunk egy lekérdezést egy adatbázisból, ami az ottani HALLGATO táblából kérdez le adatokat, akkor az a mi adatbázisunkban csak akkor fog működni, ha nekünk is van egy HALLGATO táblánk és a lekérdezésben szereplő mezők is azonos névvel mind léteznek. A példánál maradva: nekünk csak HALLGATÓ táblázatunk van, ezért a lekérdezés nem működik (az egyezésnél a kisbetű, nagybetű nem számít, de az ékezet igen). Az importálás azért megtörténik, tervező nézetben az objektum módosítható, csak adatlap nézetben kapunk hibaüzenetet.

Ha táblázatkezelővel készített fájlból, pl. Excel munkafüzetből importálunk, akkor meg kell adnunk az állomány azon részét, ami az adatokat tartalmazza (pl. a megfelelő munkalapot), azt hogy az oszlopfejlécek képezhetők-e az első adatsorból, valamint azt, hogy csak az adatokat akarjuk átvenni egy meglévő táblánkba, vagy egy új táblázatot akarunk létrehozni. Az utóbbi esetben mezőnként megadhatunk bizonyos jellemzőket (a többit később,

tervező nézetben módosíthatjuk), azt hogy melyik legyen az elsődleges kulcs és hogy mi legyen a tábla neve.

Ha csak az adatokat importáljuk egy meglévő táblába, akkor ügyelnünk kell a megfelelő tábladefinícióra. Ha az átvételre kerülő adatok nem felelnek meg a táblázat meződefinícióinak, akkor az importálás részleges lesz. Hibaüzenetben kapunk értesítést arról, hogy milyen adataink vesznek el az importálás során (ha egy oszlop adattípusa nem megfelelő, akkor a mező értékei törlésre kerülnek – ha megengedett a NULL érték, ha ismétlődő elsődleges kulcs lenne, akkor a rekord kerül törlésre). Ilyenkor eldönthetjük, hogy a hiányos adatokkal folytatjuk az importálást, vagy felfüggesztjük. Érdemes a táblát megfelelő módon módosítani és aztán újra megpróbálni az importálást.

Importáláskor gyakorlatilag duplikálás történik, hiszen az eredeti állományban is megmarad az objektum és az aktuális állományban is létrejön egy másolat erről az objektumról. A két objektum ezentúl önállóan használható, azaz egymástól függetlenül lehet őket változtatni. Ez integritási veszélyeket rejthet magában, úgyhogy csak akkor éljünk vele, ha egy tábla adatait adott állapotban véglegesen szeretnénk átvenni. Más esetekben egy tábla átvételére a csatolás a megfelelő módszer. Csatolni csak táblákat lehet, lekérdezéseket nem.

Csatoláshoz is az adatbázis ablakának kell aktívnak lennie. A *Fájl* menüben találjuk a *Külső adatok átvétele* pontban a *Csatolást*. A megjelenő *Csatolás* ablak a szokásos megnyitás ablakhoz hasonló. Itt megkereshetjük azt az állományt, amelyikben az a táblázat található, amelyiket csatolni szeretnénk. A *Fájl típus* listában megadhatjuk, hogy milyen fájlok között keresünk. Itt láthatjuk, hogy nem csak adatbázisokból csatolhatunk táblázatot, hanem pl. táblázatkezelővel készített táblázatból, vagy egyszerű szöveges állományból is.

Ha csatolunk egy táblát és a benne lévő adatokat módosítjuk, akkor azok az eredeti helyükön is módosulnak és ez fordítva is igaz. Ha az eredeti helyen módosulnak az adatok, akkor azok a csatolásnál is megváltoznak. A tábla definícióját csak az eredeti helyen lehet módosítani és a módosítások rögtön megjelennek az új helyen is. A csatolt tábla adatai közül a mezők leírása és címe¹ az állományokban függetlenül is módosítható.

6.1.2.5. Indexek

Az ACCESS-ben is van lehetőség indexek definiálására. Bármely mezőre, mezőkombinációra építhetünk indexet. A mező általános tulajdonságainál az „*indexelt*” sorban 3 különböző érték közül választhatunk:

- nem: ebben az esetben a mező nem indexelt,
- igen (lehet azonos): ebben az esetben duplikált indexünk van, azaz az index megengedi az értékek ismétlődését,
- igen (nem lehet azonos): egyedi index, azaz az indexértékek nem ismétlődhetnek. Az elsődleges kulcs helyből ilyen indexszel rendelkezik és nem is módosítható.² (Az indexekről ld. 2.4.1. és a 4.4.4.2. pont, indexek létrehozása SQL-ben ld. 5.14. táblázat.: pont)

¹ Az ACCESS címnek nevezi a mezők megjelenített nevét. Ez megegyezhet a mezők nevével, de különbözhet is attól.

² Ez megfelel a CREATE UNIQUE INDEX, az előző a CREATE INDEX SQL utasításnak (ld. 5.6.2.1. pont)

Van lehetőség összetett indexek definiálására és a tábla összes indexének áttekintésére is a *Nézet* menü *Indexek* menüpontjával. A 6.5. ábrán látható ablak felső részében láthatjuk az indexeink listáját (az index nevét, melyik mezőre vonatkozik, növekvő, vagy csökkenő sorrendű-e¹), alsó részében pedig az éppen kiválasztott index tulajdonságait. A tulajdonságok:

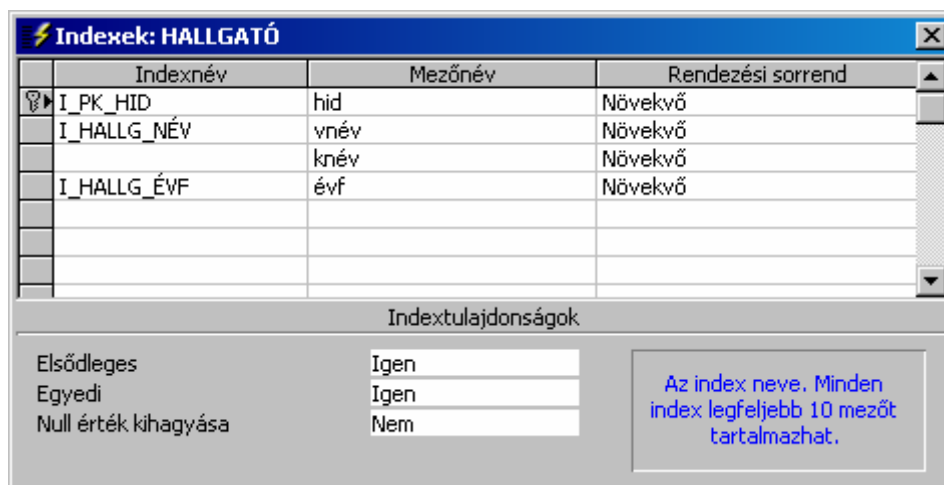
- Elsődleges: ha az értéke igen, az adott index az elsődleges kulcs.
- Egyedi: ha igen, akkor nem lehetnek azonos indexértékek.
- NULL érték kihagyása: ha igen, akkor az indexelt mezőben NULL értéket tartalmazó rekordok nem tartoznak bele az indexbe.

Több oszlopra csak így tudunk indexet definiálni (ld. 6.5. ábra). Ügyeljünk arra, hogy ilyenkor az indexben az oszlopok sorrendje megegyezik azzal, ahogy az ablakban felsoroltuk őket.

Példa

1. Állítsuk be a HALLGATÓ tábla indexeit!

A *Nézet* menü *Indexek* menüpontjában látjuk, hogy már van két indexünk. Az egyik az elsődleges kulcs. Ennek neve automatikusan PrimaryKey. Lehetőség van átnevezésre. Írjuk át a nevét I_PK_HID-re. Minden egyéb tulajdonsága maradhat változatlan. A másik indexünk a VNEV mezőre a VNEV nevű index (alapértelmezés szerint az index neve megegyezik a mező nevével, de átnevezhető). Módosítsuk a VNEV indexünket úgy, hogy összetett index legyen, azaz a VNEV és KNEV mezőkre közösen épüljön ebben a sorrendben. A neve legyen: I_HALLG_NÉV. Ezt úgy tehetjük meg, hogy az indexnévhez két mezőnév tartozik, ahogy az a 6.5. ábrán látható. Készítsünk indexet az évfolyamra is I_HALLG_ÉVF néven!



6.5. ábra: A HALLGATÓ Tábla Indexek ablaka kiegészítve

6.1.2.6. Kapcsolatok

Ha létrehoztuk tábláinkat, akkor megadhatjuk a köztük lévő kapcsolatokat is. Ez megfelel az idegen kulcsok (ld. 4.4.3.1. pont) definiálásának. A kapcsolatok ablakot az *Eszközök* menü *Kapcsolatok* pontjával, vagy az eszköztár  ikonjával lehet megnyitni. Az ablak első megnyitásakor megjelenik a *Tábla megjelenítése* ablak, ahol megadhatjuk, hogy

¹ Növekvő sorbarendezésnél a karakterek sorrendje a következő: szóköz, írásjelek, számjegyek, betűk a magyar ábécé szerinti sorban függetlenül attól, hogy kis vagy nagybetű. Csökkenő sornál a sorrend megfordul. Ha NULL érték is benne van az indexben, akkor növekvő sorrend esetén a lista elején, csökkenőnél a lista végén található.

melyik tábláinkat szeretnénk a *Kapcsolatok* ablakban látni. A *Kapcsolatok* ablak az adatbázis tábláinak és a köztük lévő kapcsolatok megjelenítésére, valamint kapcsolatok létrehozására és törlésére szolgál. Itt tehát nem tudunk létrehozni, vagy törölni táblákat, csak megjeleníteni, vagy elrejtteni tudjuk őket.

Példa

1. Állítsuk be a HALLGATÓ és a MITHALLGAT, valamint a TANTÁRGY és a MITHALLGAT táblák közötti kapcsolatokat a HID és a TID mezők segítségével, azaz definiáljuk MITHALLGAT-ban a HID és a TID idegen kulcsokat.

A *Tábla megjelenítése* ablakban válasszuk ki mind a három táblánkat (végighúzzhatjuk rajtuk az egeret) és nyomjuk meg a *Hozzáadás* gombot. A *Bezárás* gombbal becsukhatjuk az ablakot. Ha később további táblát szeretnénk a *Kapcsolatok* ablakban megjeleníteni, akkor az eszköztáron a  ikonnal újra meg tudjuk nyitni az ablakot. Ha véletlenül felesleges táblát is hozzáadtunk (például egy táblát véletlenül kétszer adtunk hozzá), akkor a kapcsolatok ablakban kattintással kijelölhetjük és elrejthetjük a *Del* gombbal, vagy a jobb egérgombbal előhívott gyorsmenüben, vagy a *Kapcsolatok* menüben a *Tábla elrejtése* paranccsal.

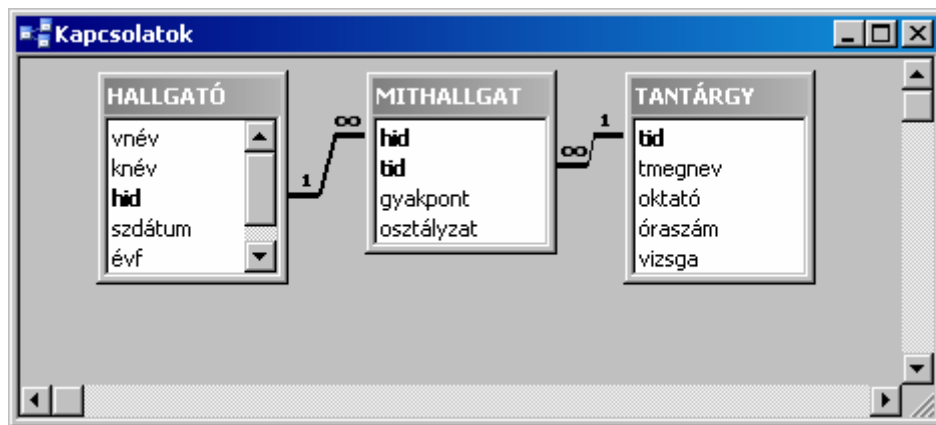
A kapcsolatokat legegyszerűbben az egér segítségével tudjuk létrehozni. Kapcsoljuk össze a HALLGATÓ táblát a MITHALLGAT táblával a HID mező segítségével, azaz a MITHALLGAT tábla HID mezője idegen kulcsként hivatkozzon a HALLGATÓ tábla HID mezőjére. Ehhez az egyik tábla HID mezőjére álljunk az egérrel és a bal gombot nyomva tartva húzzuk azt a másik tábla HID mezőjére. Ekkor megjelenik a *Kapcsolatok szerkesztése* ablak, melyben látjuk a két kapcsoló mezőt és a lehetőséget, hogy a hivatkozási integritás megőrzésre kerüljön. *Ezt minden esetben kapcsoljuk be, hiszen ez jelenti az idegen kulcs létrehozását!* Itt állíthatjuk be azt is, hogy az adatok módosításánál vagy törlésénél megengedjük-e a tovaggyűrűző (kaskád) változtatást (ld. 4.4.3.2. pont).

Mivel egy elsődleges kulcsot kapcsoltunk egy nem elsődleges kulcshoz a kapcsolat típusa *Egy-a-többhöz*, 1:N. Ha két elsődleges kulcsot kapcsoltunk volna össze, akkor a típus *Egy-az-egyhez*, 1:1 lenne¹. Olyan mezőket, amelyek közül egyiknek sincs egyedi indexe, nem tudunk összekapcsolni.

Ha mindent beállítottunk, akkor a *Létrehozás* gombbal a kapcsolat létrejön. Az 1:N kapcsolatot az ACCESS az 1 és a ∞ jelekkel jelzi. *Kapcsolatot törölni* a kapcsolat kijelölése után a *Del* gombbal, vagy a gyorsmenü *Kapcsolat törlése* pontjával lehet. A kapcsolatok létrehozáskor, módosításkor, törléskor azonnal mentésre kerülnek.

A *Kapcsolatok* ablak elrendezését külön menteni kell. Ha létrehoztunk egy kapcsolatot és aztán a kapcsolatban résztvevő egyik táblát elrejtjük a *Kapcsolatok* ablakban, akkor a kapcsolat sem látszik, de attól a kapcsolat még nem szűnt meg. Az eszköztáron a  ikon segítségével az összes kapcsolt táblát megjeleníthetjük a kapcsolatokkal együtt. A *Kapcsolatok* ablakban a táblákat jelképező téglalapok átméretezhetők és áthelyezhetők, így a leginkább átlátható formába rendezhetjük őket. Az elrendezést menteni kell. A *Kapcsolatok* ablak az EGYETEM adatbázis kapcsolataival a 6.6. ábrán látható.

¹ 1:1 kapcsolatot nagyon ritkán definiálunk két tábla között. Gyakorlatilag létre tudjuk hozni a kapcsolatot a táblák között, amíg nincs bennük adat, vagy teljesen azonos elsődleges kulcsú adatokkal vannak feltöltve, de aztán nagyon nehezen tudunk velük dolgozni.



6.6. ábra: Az EGYETEM adatbázis kapcsolatai

6.1.3. Adatkezelés

A 6.1.2. pontban azt néztük meg, hogy hogyan lehet egy adatbázist létrehozni. Ebben a pontban azt tekintjük át, hogy hogyan tudjuk adatokkal feltölteni az adatbázist, a feltöltött adatokat hogyan tudjuk megnézni, módosítani, törölni. Ezekhez a műveletekhez a tábláinkat *adatlap nézetben* kell megnyitnunk. Az adatlap nézetben a tábla oszlopainak nevét, majd alatta az egyes sorait látjuk. Alapértelmezés szerint, ha egy táblát megnyitunk, az automatikusan adatlap nézetben nyílik meg. Ebben a táblázat egyes adataival tudunk dolgozni. A táblázat felépítését a tervező nézetben változtathatjuk meg. A tervező és az adatlap nézet között az eszköztáron lévő és gombok segítségével, vagy a *Nézet menü* megfelelő pontjaival tudunk váltogatni. Adatok bevitelére, módosítására *Úrlapon* is lehetőségünk van. Erre a 6.1.5. pontban fogunk visszatérni.

6.1.3.1. Adatbevitel

Adatlap nézetben táblázatos formában van lehetőségünk az adatok bevitelére. A táblázat fejlécét a tábla mezőinek nevei alkotják. A cellák között a *Tab*, az *Enter* és a kurzormozgató billentyűkkel tudunk mozogni, vagy az egérrel kattinthatunk másik cellába. Az új sor helyét a sor elején a jelölő lapkán lévő * jelzi. Ha ez a sor egyben az aktuális sor, akkor a * helyett látszik.

Ha egy cellába olyan értéket írunk, amit az adott cellába nem írhatnánk, mert más az adattípus, vagy megsértünk egy érvényességi szabályt, akkor hibaüzenet jelenik meg és addig nem mehetünk új cellába, amíg nem javítjuk ki az értéket megfelelő értékre. Ha van olyan érvényességi szabály, ami nem egyetlen mezőre, hanem a táblára vonatkozik és azt sértjük meg, vagy egy egyedi indexű mezőbe írunk ismétlődő értéket, akkor ezt nem a mező, hanem a rekord beírásának befejezésekor észleli az ACCESS. Ilyenkor nem engedi meg a rekord beírását az adatbázisba, azaz ha új rekordot akarunk kezdeni, akkor hibaüzenet jön és nem tudunk továbbmenni.

Ha állítottunk be alapértelmezést, akkor az megjelenik a megfelelő mezőben, de átírható. A rekord bevitelét új rekordra menetellel, vagy a *Shift + Enter* billentyűkombináció megnyomásával tudjuk befejezni. A bevitt adatok rögtön bekerülnek az adatbázisba, nem kell külön menteni őket.

Ez a művelet az INSERT SQL utasításnak felel meg (ld. 5.6.3.5. pont).

Ha a táblánkhöz kapcsolódik idegen kulccsal egy másik tábla, akkor a bevitt rekordok sorának elején, közvetlenül a jelölőlapka után megjelenik egy + jel. Erre a jelre kattintva a táblázaton belül megnyílik egy kisebb táblázat, amiben a kapcsolódó táblázat kapcsoló

rekordjait adhatjuk meg, illetve, ha már vannak kapcsolódó rekordok, akkor azokat láthatjuk és a következő pontokban leírt módon módosíthatjuk, vagy törölhetjük őket.

A 6.1.2.4. pontban már utaltunk rá, hogy másik táblázatokból importálással hozhatunk át adatokat az adatbázisunkba. Microsoft alkalmazásokból a vágólap segítségével is vihetünk be adatokat a tábláinkba, ha azok megfelelően tagoltak.

6.1.3.2. Adatmódosítás

Adatlap nézetben a módosítani kívánt rekord sorára állunk. A sor bal szélén, a jelölő lapkán egy  jelzi, hogy melyik az aktuális sor. A cellák között az előző pontban leírtak szerint tudunk mozogni. Ha egy cellára állunk és elkezdünk gépelni, akkor a cella tartalmát felülírjuk. Ilyenkor az *Esc* gombbal még vissza tudjuk vonni a változtatásunkat. Ha nem akarjuk a cella teljes tartalmát felülírni, csak néhány karaktert akarunk módosítani, akkor *szerkesztő módba* kell mennünk. Vagy az egérrel kattintsunk duplán a cellán, vagy nyomjuk meg az *F2 billentyűt*. Ekkor a cellán belül tudunk a kurzormozgató billentyűkkel mozogni és a kurzor helyén írni, vagy törölni. Ha elkezdünk módosítani egy rekord bármelyik mezőjén, a  helyett  jelenik meg a jelölőlapkán mindaddig, amíg a teljes rekord szerkesztését be nem fejeztük. Ha befejeztük egy rekord módosítását (más rekordra kattintással, vagy *Shift + Enterrel*), akkor a módosítás azonnal bekerül az adatbázisba. Nem kell külön menteni.

Természetesen a módosítást csak akkor tudjuk tárolni, ha az új érték is megfelel az adott mezőre vonatkozó integritási feltételeknek.

A rekord módosítása az UPDATE SQL utasításnak felel meg (ld. 5.6.3.4. pont).

6.1.3.3. Törlés

Adatlap nézetben a sorok elején található jelölőlapkák segítségével kijelölhetjük a tábla sorait. Egérhúzással több egymás utáni sort is kijelölhetünk. Nem egymás utáni sorok kijelölésére nincs mód. A kijelölt sorokat a szokásos módokon törölhetjük, pl. a *Delete* gombbal vagy a *Tábla adatlap eszköztár*  gombjával.

Sorok törlésekor, ha a sorok törölhetők, akkor egy üzenetet kapunk, hogy a sorok visszavonhatatlanul törölődnek a táblából. Itt még a *Nem* gomb megnyomásával visszavonhatjuk a törlés előtti állapotra. Ez megfelel az SQL-ben kiadott DELETE utasításnak (ld. 5.6.3.3. pont), majd a *Nem* az utána kiadott ROLLBACK, az *Igen* a COMMIT utasításnak (ld. 5.6.4. pont). Ha olyan sort akarunk törölni, amire hivatkozik egy másik tábla rekordja, például olyan hallgatót akarunk törölni, aki benne van a MITHALLGAT táblában és a kapcsolat beállításánál nem engedték meg a kaszkád törlést, (ld. 4.4.3.2. pont) akkor a sort nem tudjuk kitörölni. Erről is üzenetben értesülünk.

6.1.3.4. Adatok megjelenítése, formázás, rendezés, szűrés

Adatlap nézetben lehetőségünk van az adatok megjelenítésének különböző módosításaira. A módosításokat menteni kell. Kijelöléssel és áthúzással megváltoztathatjuk az oszlopok sorrendjét. Ezzel a tábla definícióját is módosítjuk. A mezőneveket elválasztó vonalakra kattintva az egérrel változtathatjuk az oszlopok szélességét, dupla kattintással a leghosszabb tartalomhoz igazíthatjuk azt. A *Formázás* eszköztár és a *Formátum* menü segítségével formázhatjuk a táblázat adatait és a táblázat megjelenését. A formázások mindig az egész táblázatra vonatkoznak, nem lehet rekordonként, vagy mezőnként formázni. A *Formátum* menü segítségével elrejtethetünk és felfedhetünk oszlopokat.

A tábla adatait *adatlap nézetben* alapértelmezésben az elsődleges kulcs alapján növekvő sorrendben rendezve látjuk. Ezt a rendezési sorrendet egyszerűen megváltoztathatjuk. Egy másik mező szerinti rendezettséghez kattintással ráállunk az adott

mezőnévre vagy a mező bármelyik cellájára. A Tábla adatlap eszköztáron a   ikonok segítségével növekvő, vagy csökkenő sorrendbe rendezhetünk, vagy a *Rekordok* menü *Rendezés* pontja alatt választhatjuk a Rendezés-növekvő vagy a Rendezés-csökkenő parancsot. Több egymás melletti mezőt is kijelölhetünk az egér húzásával. Így lehet több oszlop szerint rendezni. Több oszlop szerinti rendezés esetén a kijelölt oszlopok közül a bal szélső alapján rendez először és az ott található egyező értékek esetén veszi csak figyelembe a következő oszlopokat balról jobbra haladva. A rendezésben a karakterek sorrendje megegyezik az indexeknél tárgyaltakkal (ld. 6-14. oldal).

Nem egymás melletti mezőket nem tudunk kijelölni és a rendezésben szereplő mezők sorrendjét sem tudjuk megváltoztatni. Ha mégis ilyen rendezésre lenne szükség, akkor a *Rekordok* menü *Szűrő* pontján belül az *Írányított szűrés/rendezés* parancsot választjuk. Mivel ennek segítségével összetettebb szűrést is megadhatunk, pontosabb leírását lejjebb adjuk meg.

A rendezés az SQL SELECT utasításának (ld. 5.6.3.2. pont) ORDER BY klauzulájának megadását jelenti.

Lehetőség van az adatok különböző szempontok szerinti szűrésére, ami a SELECT utasítás WHERE klauzulájának felel meg. Ha egy szűrést mentünk a táblához, akkor azt új megnyitáskor a  segítségével bármikor ki-be kapcsolhatjuk. Az ablak alján mindig látható, hogy összesen hány rekord látható a táblában. Ha a szűrő be van kapcsolva, akkor a *(szűrt)* kiírás látható a rekordok száma mellett.

Szűrésre több lehetőségünk van. A legegyszerűbb a *szűrés kijelöléssel*. Ezt akkor alkalmazhatjuk, ha egy mező alapján pontos egyezés alapján szeretnénk szűrni. Például szűrjük a *Aladár* keresztnévű hallgatókat (*knév="Aladár"*), vagy azokat a tárgyakat, melyeket Quittner tanít (*oktató="Quittner"*). A szűréshez a mező egy olyan rekordjára kell állni, amelyikben a megfelelő érték található és kiadni a *Szűrés kijelöléssel* parancsot. Ezt kiadhatjuk a *Tábla adatlap eszköztár*  ikonjával vagy a *Rekordok* menü, *Szűrő* pontján belül, vagy a gyorsmenüből. Több mező alapján úgy tudunk szűrni, hogy a szűréseket egymás után adjuk ki. Ekkor a feltételek között ÉS kapcsolat lesz. Például, ha szűrjük az elsőéves *Aladár* keresztnévű hallgatókat, akkor először szűrünk mondjuk az *Aladár* névre és aztán az évfolyamra (*évf=1*). Ekkor csak az elsős Aladárok jelennek meg.¹

További lehetőség a *szűrés űrlappal*. Szintén elindítható a *Tábla adatlap eszköztárról* a  ikonnal, vagy a *Rekordok* menü *Szűrés* pontja alól. Itt minden mezőnél egy listából választhatjuk ki, hogy melyik rekordok kerüljenek bele a szűrésbe. Akkor érdemes alkalmazni, ha több mező alapján szűrünk, de mindenhol csak egyenlőség feltételünk van.

Ha a szűrés feltételében más relációt szeretnénk használni, vagy nem ÉS, hanem VAGY kapcsolat van a feltételeink között, akkor a *Rekordok* menü *Szűrő* pontjának *Írányított szűrés/rendezés* parancsát kell választanunk. Ekkor a 6.7. ábrán látható ablak jelenik meg kitöltetlenül. Ebben az ablakban megadhatjuk, hogy melyik mezőre milyen feltételeink vannak és milyen rendezést szeretnénk megadni. A feltételek beírásához használhatjuk a már megismert *kifejezés-szerkesztőt* is a gyorsmenüben a *Szerkesztés* pontot választva.

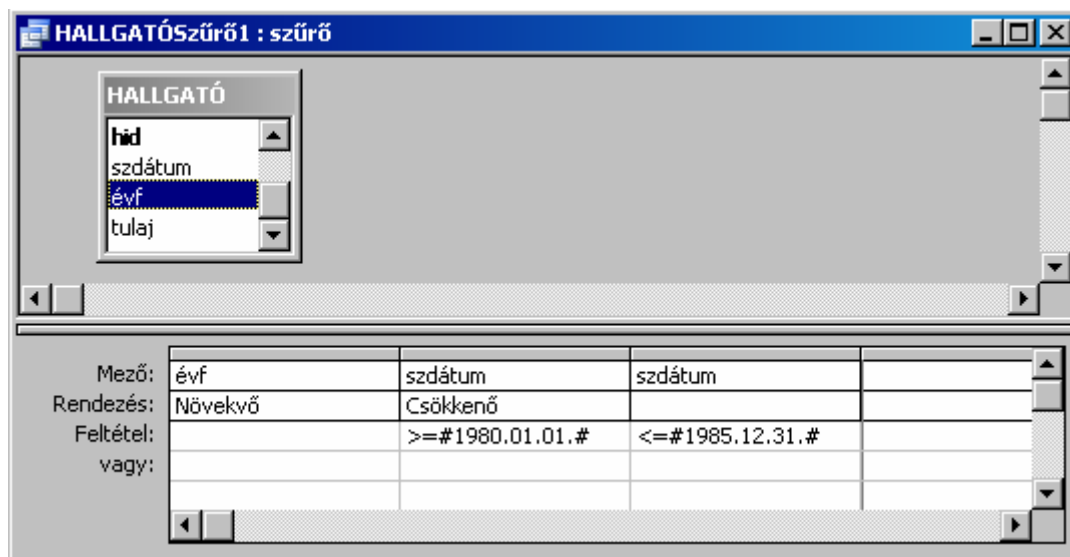
Példa

1. Állítsunk be a HALLGATÓ táblára évfolyam szerinti növekvő rendezést és azon belül születési dátum szerinti csökkenő rendezést. Szűrjük az adatokat azokra a hallgatókra, akik 1980. január 1. és 1985. december 31. között születtek.

¹ A szűrés kijelöléssel lehetőség párja a szűrés kizárással. Ez az eszköztáron nem jelenik meg. Működése hasonló, azzal a különbséggel, hogy a feltétel nem az egyenlőség, hanem a nem egyenlőség. Szűrhetjük például a MITHALLGAT táblából az összes rekordot, ahol az osztályzat nem 1. Ügyeljünk a NULL értékekre! Azokat a rekordokat, amelyekben *nincsen* osztályzat, ezzel a szűrő feltétellel nem kapjuk meg.

A megoldás a 6.7. ábrán látható. Mivel a születési dátumra vonatkozó két feltételünk között ÉS kapcsolat van, azokat egymás mellett kell feltüntetnünk¹. A VAGY kapcsolatot a feltételek egymás alá írásával tudjuk megadni. Természetesen alkalmazhatjuk egy kifejezésen belül is az AND és az OR logikai operátorokat is.

Mentsük a tábla módosítását. Zárjuk be a táblát és nyissuk meg újra. A rendezés megmarad, de újra az összes rekord látható lesz. A szűrő azért megmaradt, most már csak a ▼ ikont kell megnyomni, hogy csak a szűrt adatokat lássuk.



6.7. ábra: A HALLGATÓ tábla szűrője²

Ilyen összetett szűrés beállítása helyett nyugodtan létrehozhatunk önálló lekérdezést. A lekérdezéseket a következő pontban tárgyaljuk.

Ha a táblát tervező nézetben nyitjuk meg, akkor a Tábla tulajdonságai ablakban láthatjuk a *szűrő* és a *rendezés* szöveges leírását. Ha a leírás nem fér ki, akkor a gyorsmenüben válasszuk a Nagyítás pontot. Természetesen a rendezés és a szűrés itt is módosítható.

6.1.4. Lekérdezések

A lekérdezések abban különböznek a szűrésektől, hogy nem a táblához tartoznak, hanem külön objektumként kerülnek az adatbázisba. Lekérdezést készíthetünk táblákból és lekérdezésekből is. Ebben a pontban ahol táblát írunk, ott mindenütt lekérdezést is értünk alatta. Ahol ettől eltérünk, azt külön jelezzük. Egy táblából akárhány lekérdezést készíthetünk és egy lekérdezésben is szerepelhet több tábla és lekérdezés. Az ACCESS lekérdezésnek nevez különböző típusú utasításokat is. Az általunk lekérdezésnek nevezett utasítások a ACCESS-ben a választó lekérdezés nevet kapták. A lekérdezések mentés nélkül is futtathatók, de többszöri felhasználásra el is menthetők. Az elmentett választó lekérdezések valójában SQL nézetek (View, ld. 5.3.6.2. pont) és létrehozásuk megfelel a CREATE VIEW SQL utasításnak (ld. 5.14. táblázat.: pont). Az egyszeri alkalomra lefuttatott lekérdezések és az előző pontban ismertetett szűrők a SELECT utasítás (ld. 5.6.3.2. pont) különböző formáit jelentik.

¹ A két feltétel összekapcsolása helyett használhatnánk a BETWEEN operátort is.

² Dátum literál megadásakor a dátumot # jelek közé kell írunk. Ha elmulasztjuk, az ACCESS kiteszi helyettünk.

6.1.4.1. Választó lekérdezések létrehozása grafikusán

Ahogy táblákat, úgy lekérdezéseket is létre tudunk hozni varázsló segítségével és tervező nézetben is. Tervező nézetben grafikus formában tudjuk megtervezni lekérdezéseinket, de lehetőségünk van a lekérdezések SQL parancsát is megnézni, ott módosítani.

Ebben a fejezetben a választó lekérdezések létrehozását nézzük át tervező nézetben, a további pontokban pedig áttekintjük az egyéb lehetőségeket. A varázsló kipróbálását az olvasóra bizzuk. A lekérdezések SQL kódját és azokat a lekérdezéseket, amelyeket csak ennek segítségével lehet létrehozni külön pontban tekintjük át.

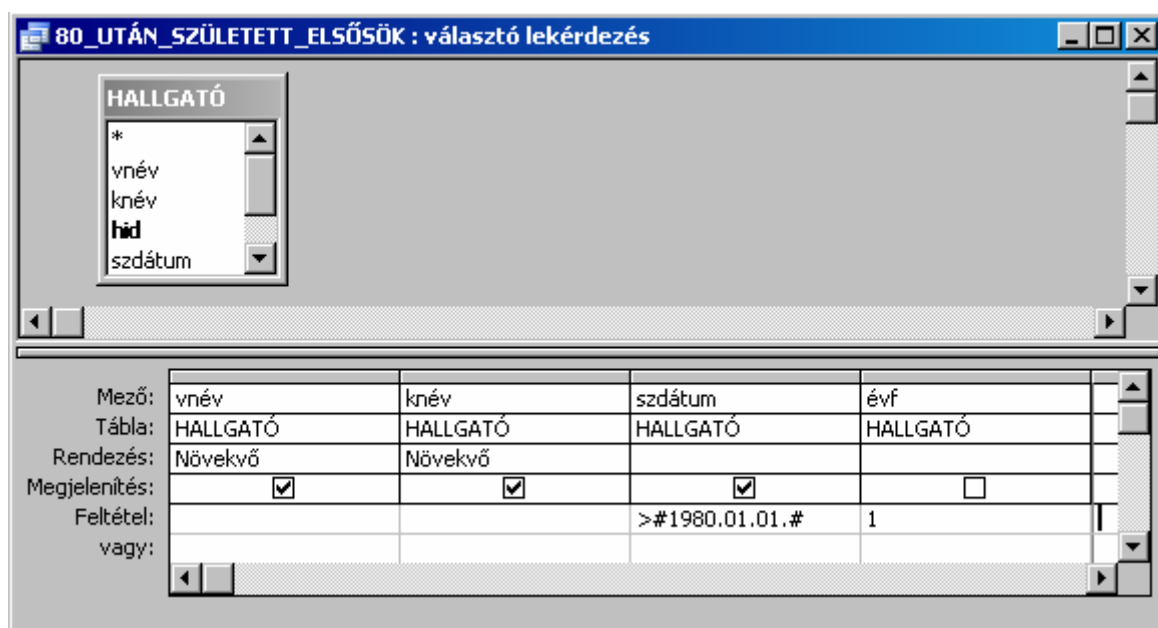
Nyissuk meg a *lekérdezés tervező* ablakot! Erre több lehetőségünk is van, például az adatbázis ablakban a *Lekérdezések* csoportban válasszuk a *Lekérdezés létrehozása tervező nézetben* pontot! A megjelenő ablakkal együtt megjelenik a kapcsolatnál megismert *Tábla megjelenítése* ablak. Itt megadhatjuk, hogy melyik táblá(k)ból és lekérdezésekből készítjük a lekérdezést.

A menüsor kiegészül a *Lekérdezés* menüponttal. Ebben a menüpontban tudunk különböző beállításokat megadni a lekérdezésünkkel kapcsolatban. A lekérdezés tervező ablaka hasonlít az irányított szűrés ablakához, de több sor jelenik meg. A *Mező* alatt megjelenik a *Tábla*, ahol azt is meg kell adnunk, hogy az adott mezőt melyik táblából kérdezzük le. A *Megjelenítés* sorban megadhatjuk, hogy a kiválasztott mezőre csak rendezés és feltételek vonatkoznak, vagy meg is akarjuk őket jeleníteni a lekérdezés eredményében. A *Nézet* menüben további sorokat kapcsolhatunk be. Ezekre a konkrét példánál térünk ki. Beállíthatjuk a lekérdezésben szereplő mezők formátumát is a mező nevével a gyorsmenüből a *Tulajdonságok* pontot, vagy az eszköztáron a  ikont választva. A formátum megadása ugyanúgy működik, mint a táblák definiálásánál.

Egyszerű választó lekérdezés egy tábla adataiból

Példa

1. Kérdezzük le az 1980. január 1. után született elsős hallgatók nevét és születési dátumát név szerint rendezve!



Mező:	vnév	knév	szdátum	évf
Tábla:	HALLGATÓ	HALLGATÓ	HALLGATÓ	HALLGATÓ
Rendezés:	Növekvő	Növekvő		
Megjelenítés:	☒	☒	☒	☐
Feltétel:			>#1980.01.01.#	1
vagy:				

6.8. ábra: Egyszerű választó lekérdezés

A lekérdezésben szereplő mezőket megadhatjuk például begépeléssel, vagy az ablak felső részében a tábla megfelelő mezőjére dupla kattintással. Ez utóbbi esetben a Tábla sor automatikusan kitöltésre kerül. A példa megoldása a 6.8. ábrán látható.¹ A lekérdezést a *Lekérdezés* menü *Futtatás* parancsával, vagy a *Lekérdezéstervezés eszköztár* ! ikonjával futtathatjuk. A lekérdezés eredményét *adatlap nézet*re váltással is megnézhetjük. A lekérdezésnek mentéskor adhatunk nevet. A lekérdezésünk neve legyen: 80_UTÁN_SZÜLETETT_ELSŐSÖK. Mentett lekérdezést megnyithatunk *adatlap*, vagy *tervező nézet*ben. Adatlap nézetben a táblák adatlap nézeténél tárgyalt műveletek végezhetők el. A menüsor is az ott megszokott lesz.

Lekérdezés több tábla adataiból

Példa

1. Készítsünk lekérdezést OKTATÁS néven, amely minden adatot tartalmaz.

Ehhez a *Tábla megjelenítése* ablakban adjuk meg mind a három táblát. Az ablak felső részén láthatjuk, hogy a beállított kapcsolataink itt is megjelennek. Ha így készítünk lekérdezést, akkor a megadott táblák természetes (belső) joinjából (ld. 4.5.6.1. pont) tudunk lekérdezni. Ha nem szeretnénk egyesével megadni a lekérdezendő mezőket, akkor választhatjuk a mezőlisták tetején megtalálható *-ot, ami az összes mezőt jelenti. Ha mind a három táblából a *-ot választjuk, akkor a *hid* és a *tid* oszlopok duplán fognak szerepelni. Erre nincs szükség, úgyhogy válasszuk a HALLGATÓK.*, a TANTÁRGY.* és a MITHALLGAT táblából a *gyakpont* és az *osztályzat* mezőket! A megoldás a 6.9. ábrán látható.

	HALLGATÓ.*	TANTÁRGY.*	gyakpont	osztályzat
Mező:			gyakpont	osztályzat
Tábla:	HALLGATÓ	TANTÁRGY	MITHALLGAT	MITHALLGAT
Rendezés:				
Megjelenítés:	☒	☒	☒	☒
Feltétel:				
vagy:				

6.9. ábra: Lekérdezés több táblából

¹ Az évfolyamot nem jelenítjük meg, mivel annak értéke minden sorban 1, és ez a lekérdezés elnevezéséből is egyértelműen kiderül.

Megjelenített adatok számának korlátozása

Lekérdezésnél megadható, hogy egy táblából valamelyik szempontból csak a legnagyobb vagy legkisebb valahány adatot jelenítsük meg. Ehhez az adott mező szerint rendeznünk kell, és meg kell adnunk, hogy hány értéket szeretnénk megjeleníteni. Ezt a számot az ACCESS csúcsértéknek nevezi.

Példa

1. Kérdezzük le a 3 legfiatalabb hallgató nevét és évfolyamát!

A HALLGATÓ táblából szükségünk van a *vnév*, *knév*, *évf* és *szdátum* mezőkre, de ez utóbbit nem fogjuk megjeleníteni. Születési dátum szerint csökkenő rendezést kell beállítanunk, hogy a fiatalok legyenek a lista elején. A csúcsértéket a *Lekérdezéstervezés* eszköztáron a Σ melletti kombinált listában lehet megadni. Ha nem adunk meg itt értéket, akkor az *Összes* felirat látszik. Ha olyan értéket akarunk megadni, ami nincs a listában (a 3 is ilyen), akkor azt be kell gépelnünk és utána *Entert* kell nyomnunk. A megoldás a 6.10. ábrán látható. Mentsük a lekérdezést FIATAL_HALLGATÓK néven!

Mező:	vnév	knév	évf	szdátum
Tábla:	HALLGATÓ	HALLGATÓ	HALLGATÓ	HALLGATÓ
Rendezés:				Csökkenő
Megjelenítés:	☒	☒	☒	☐
Feltétel:				
vagy:				

6.10. ábra: Megjelenített adatok számának korlátozása a lekérdezésben

Kifejezés megadása

A lekérdezés mezőjében különböző kifejezéseket is megadhatunk. Ilyenkor a lekérdezésben az oszlop neve a kifejezés képlete lesz. Ha más nevet szeretnénk megjeleníteni, akkor a *Mező* sorában először a megjelenítendő nevet kell megadnunk¹ és kettőspont (:) után a kifejezést.

¹ Ez megfelel a SELECT utasítás kiválasztási listájában lévő AS kulcsszónak és az utána megadott ideiglenes névnek SQL-ben (ld. 5.6.3.2. pont).

Kérdezzük le a hallgatók vezetéknévét és keresztnévét összevonva a *név* oszlopban, a tanult tárgyak nevét a *tantárgy* oszlopban és a kapott osztályzatot a *jegy* oszlopban! Mentsük a lekérdezést VIZSGAEREDMÉNY néven! A megoldás a 6.11. ábrán látható¹.

VIZSGAEREDMÉNY : választó lekérdezés

HALLGATÓ

- * vnév
- knév
- hid
- szdátum

MITHALLGAT

- * hid
- tid
- gyakpont
- osztályzat

TANTÁRGY

- * tid
- tmegnev
- oktató
- óraszám

Mező: Név: [vnév] & " " & [knév] Tantárgy: tmegnev Jegy: osztályzat

Tábla: TANTÁRGY MITHALLGAT

Rendezés:

Megjelenítés: ☒ ☒ ☒

Feltétel: vagy:

6.11. ábra: Kifejezés megadása lekérdezésben

Összesítés a lekérdezésben

Lekérdezésben csoportosíthatjuk is a rekordokat és ezekre a csoportokra statisztikai mutatókat számíthatunk ki. Ezzel egy GROUP BY klauzulával kiegészített SQL SELECT utasítást adunk meg (ld. 5.6.3.2. pont). Ehhez meg kell jelenítenünk a lekérdezés tervezésekor az *Összesítés* sort. Ezt megtehetjük a *Nézet* menü *Összesítés* menüpontjával, vagy a *Lekérdezéstervezés eszköztár* Σ ikonjával. Az összesítés sorban többek között az alábbi értékek között választhatunk:

- Group By: azoknál a mezőknél állítjuk be, amelyek szerint csoportosítani akarunk,
- Sum: a mező egy csoportba tartozó értékeit összegzi,
- Avg: a mező egy csoportba tartozó értékeit átlagolja,
- Min: a mező egy csoportba tartozó értékeinek minimumát adja,
- Max: a mező egy csoportba tartozó értékeinek maximumát adja,
- Count: a mező egy csoportba tartozó értékeinek darabszámát adja,²
- First: a mező egy csoportba tartozó értékei közül az elsőt adja,
- Last: a mező egy csoportba tartozó értékei közül az utolsót adja,³
- Expression: ha az összesített érték valamilyen kifejezésben szerepel, akkor a teljes kifejezést a mezőbe kell írni és a listából az Expression-t (kifejezés) kell választani,

¹ A vezetéknévnek és a keresztnévnek az & (konkatenáló) operátorral való összefűzésénél vigyáztunk arra, hogy ne kerüljön a két név egybeírásra. Ezért volt szükség arra, hogy egy szóközt is hozzáfűzzünk a vezetéknévhez a keresztnév előtt.

² Ezek a függvények megegyeznek az SQL csoport függvényekkel (ld. 5.5.1. pont). Alkalmazásuk korlátaira és értékük meghatározására az ott elmondottak érvényesek.

³ A sorrend a fizikai (vagy a definiált indexnek megfelelő) elhelyezkedésben első vagy utolsó. Független az adatlap nézetben beállított sorrendtől.

- Where: a mezőre megadott feltétel Where feltétel lesz és nem Having feltétel, azaz a feltétel az egyes adatokra és nem az összesített adatokra vonatkozik. Ezeket a mezőket nem tudjuk megjeleníteni a lekérdezésben, mert a Where feltétellel párhuzamosan nem tudjuk megadni, hogy az adott mezőt csoportosítóként, vagy számított mezőként szeretnénk-e megjeleníteni. Ha meg akarjuk jeleníteni, akkor külön még egyszer hozzá kell adnunk a lekérdezéshez.

Példa

1. Kérdezzük le a MITHALLGAT táblából tantárgyanként az átlagosztályzatot és a megszerzett maximális pontot! Javítsuk ki a lekérdezést, hogy csak azokra a rekordokra vonatkozzon, ahol az osztályzat nem 1-es! Mentsük a lekérdezést TANTARGYANKÉNTI_EREDMÉNYEK néven!

A megoldás a 6.12. ábrán látható.

Mező:	tid	átlagosztályzat: osztályzat	maxpont: gyakpont	osztályzat
Tábla:	MITHALLGAT	MITHALLGAT	MITHALLGAT	MITHALLGAT
Összesítés:	Group By	Avg	Max	Where
Rendezés:				
Megjelenítés:	☒	☒	☒	☐
Feltétel:				>1
vagy:				

6.12. ábra: Összesítés megadása lekérdezésben

Paraméteres lekérdezés

Paraméteres lekérdezésre akkor van szükségünk, ha ugyanazt a lekérdezést többször futtatjuk, de egy meghatározott részét mindig változtatjuk a lekérdezésnek. Ez a meghatározott rész lehet a feltétel, vagy a mezőkifejezés is. Ha a lekérdezést paraméterekkel fogalmazzuk meg, akkor futtatáskor a program a felhasználótól kéri, hogy az adott futásnál milyen értéket vegyen fel a paraméter. Egy lekérdezésben több paramétert is alkalmazhatunk. A paramétereknek nevet kell adni, ami különbözik a lekérdezésben szereplő táblák mezőinek nevétől. Futáskor a program erre a névre fog rákérdezni, tehát érdemes beszélő nevet adni a paramétereknek. A lekérdezés az összes paraméterérték megadása után hajtódik végre.

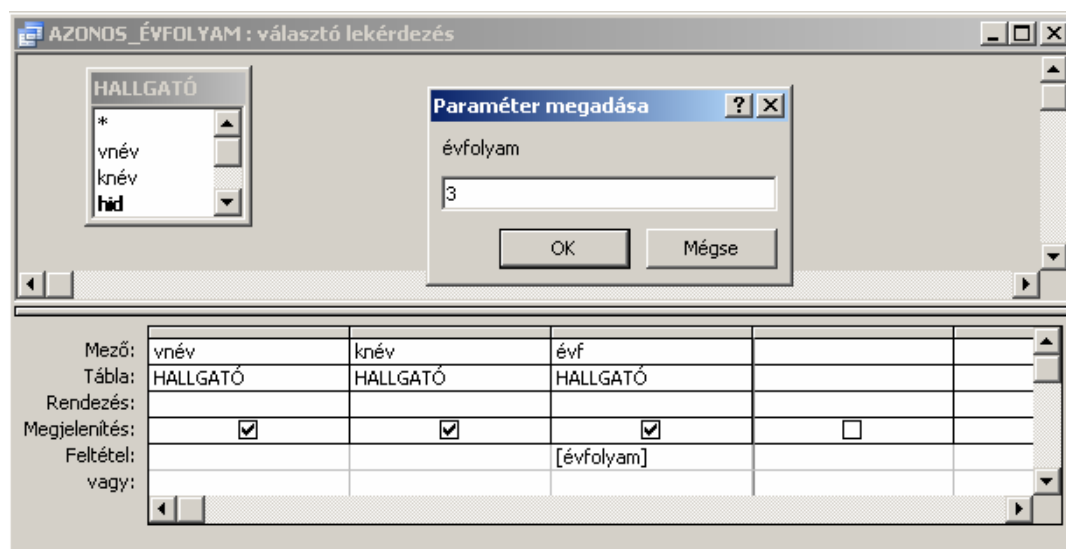
Paraméterek megadásának két módja van. Az egyik, hogy a *Lekérdezés* menü *Paraméterek* pontjában létrehozuk a paramétert. Itt megadhatjuk a nevét és az adattípusát is. A másik, hogy csak felhasználjuk, ilyenkor csak nevet adunk a paraméternek. A használatban az a különbség, hogy ha rossz adattípusú adatot adunk meg, akkor a létrehozott paraméter esetén rögtön hibaüzenetet kapunk és tudunk javítani, míg az adattípus nélküli paraméternél elindul a lekérdezés és nem hoz eredményt, vagy hibaüzenettel megáll. Ilyenkor újra kell indítani a lekérdezés futtatását.

Ugyanazt a paramétert egy lekérdezésen belül többször is felhasználhatjuk. A lekérdezésben a paramétert szögletes zárójelek ([]) között kell megadni.

Példák:

1. Kérdezzük le a HALLGATÓ táblából az adott évfolyamra járó hallgatók nevét és évfolyamát! Mentsük a lekérdezést AZONOS_ÉVFOLYAM néven!

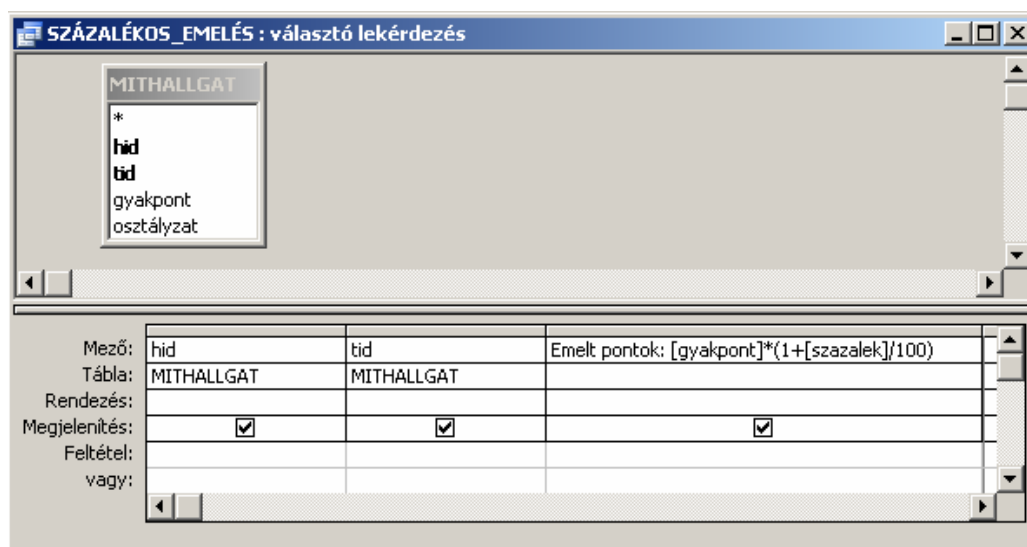
A megoldás a 6.13. ábrán látható. A lekérdezés lefuttatásakor az ACCESS rákérdez az ÉVFOLYAM paraméter aktuális értékére. Miután ezt megadtuk, végrehajtja a lekérdezést.



6.13. ábra: Paraméteres lekérdezés a paramétert kérő ablakkal

2. Kérdezzük le a MITHALLGAT táblából a hallgatók és a tantárgyak kódját és azt, hogy hány gyakorlati pontjuk lenne, ha valahány százalékkal megemelnénk a pontokat *Emelt pontok* néven! Azt, hogy hány százalékkal emelünk, paraméterrel lehessen megadni! Ez egyben arra is példa, hogy miként képezhetünk egy lekérdezésben új mezőt más mező(k) értékeiből. Mentsük a lekérdezést SZÁZALÉKOS_EMELES néven!

A megoldás a 6.14. ábrán látható.



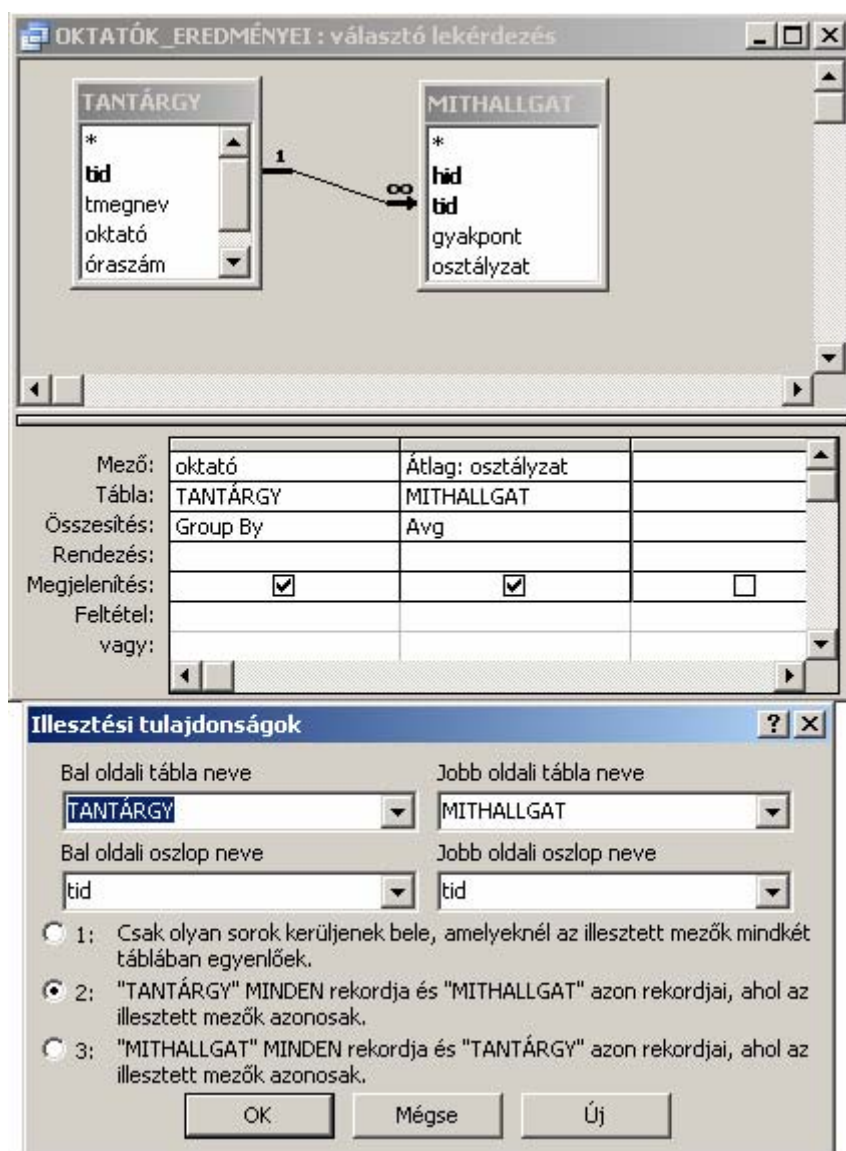
6.14. ábra: Paraméter a mezőben

Lekérdezés több tábla adataiból külső joinnal

Az összekapcsolt táblákból készített lekérdezéseknél automatikusan belső join jön létre a táblák között. Egy-egy lekérdezés erejéig ezeket a kapcsolatokat módosíthatjuk. (A külső joinról ld. 4.5.6.2. pont.) Miután a lekérdezéshez hozzáadtuk a megfelelő táblákat, a kapcsolat vonalára jobb egérgombbal kattintva a gyorsmenüből kiválaszthatjuk az *Illesztési tulajdonságok* pontot. Itt be tudjuk állítani, hogy melyik táblából jelenjenek meg azok az adatok is, amelyekhez nem kapcsolódik adat a másik táblából, vagyis jobb, vagy baloldali külső joint hozunk-e létre. Ha a beállítást változtatjuk, akkor a vonal nyíllá alakul.

Példák

1. Készítsünk lekérdezést, amely megadja, hogy az egyes oktatók hallgatói milyen átlagosztályzatot értek el. Ha van olyan oktató, akinek nincs még egyetlen hallgatója sem, az is jelenjen meg a lekérdezésben. A két lekérdezett mező felirata *oktató* és *átlag* legyen! Mentsük a lekérdezést OKTATÓK_EREDMÉNYEI néven.

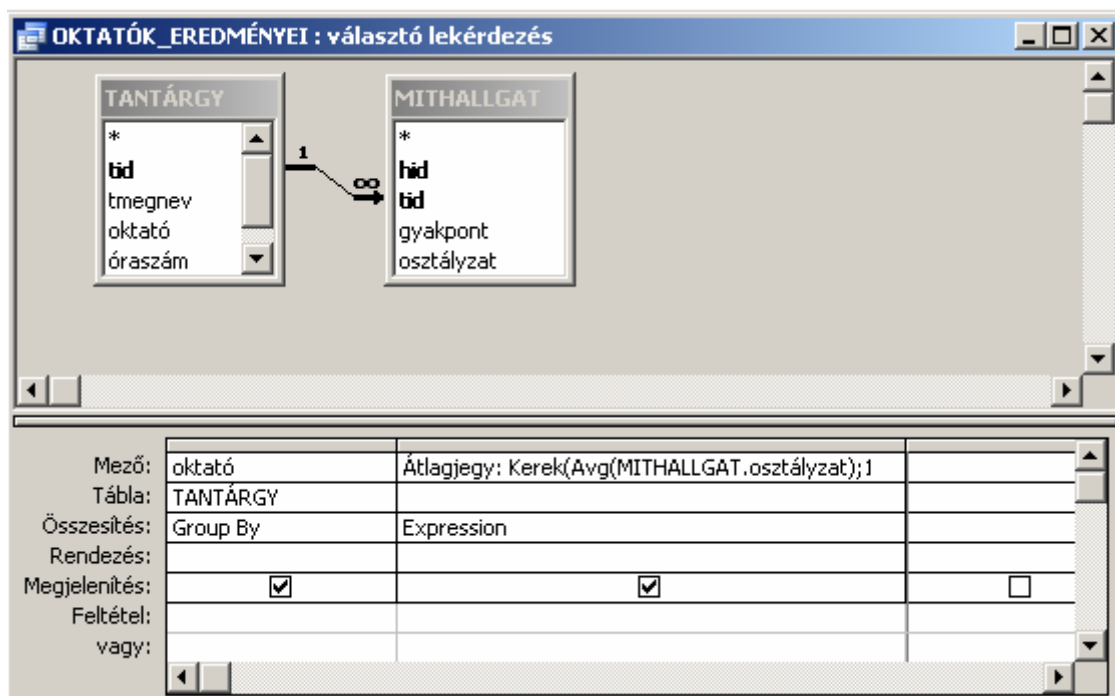


6.15. ábra: Külső join és az *Illesztési tulajdonságok* ablak

A megoldás a 6.15. ábrán látható. Ha futtatjuk a lekérdezést és olyanok az adataink, akkor tapasztalhatjuk, hogy az Avg függvény nem kerekít, hanem akár 14 tizedesjegyet is megjelenít.

2. Módosítsuk az előző lekérdezést úgy, hogy az átlagosztályzat egy tizedesre legyen kerekítve! A lekérdezést mentjük rá az előző OKTATÓK_EREDMÉNYEI lekérdezésre!

A megoldás a 6.16. ábrán látható. A kerekítéshez a *kerek* függvényt használtuk fel (ld. 6.1.2.3. pont). Mivel nem egyszerű összesítés szerepel a lekérdezésben, hanem egy összetett kifejezés, a *lekérdezés tervező* ablakában az átlag oszlopnál a teljes kifejezést a *Mező* sorba kell írni és az *Összesítés* sorban az *Expression* kifejezést kell választani.



6.16. ábra: Külső join és kerekítés

6.1.4.2. Egyéb lekérdezések létrehozása grafikusán

Ha megnyitjuk a lekérdezés-tervezőt, alapértelmezés szerint mindig választó lekérdezés létrehozását ajánlja fel. Ezen a *Lekérdezés* menüpontban, vagy az eszköztár *Futtatás* előtti ikonjával változtathatunk. Az ikon mindig az éppen aktuális lekérdezés-típusnak megfelelő. Amikor rákattintunk, egy legördülő listából lehet választani az alábbi lekérdezés-típusok közül:

- keresztáblás lekérdezés,
- táblakészítő lekérdezés,
- frissítő lekérdezés,
- hozzáfűző lekérdezés,
- törlő lekérdezés.

Az alábbiakban részletesen ismertetjük ezeket a lekérdezéseket. A lekérdezés elnevezése mellett feltüntettük az utasításnak megfelelő ikont is.

Kereszt táblás lekérdezés

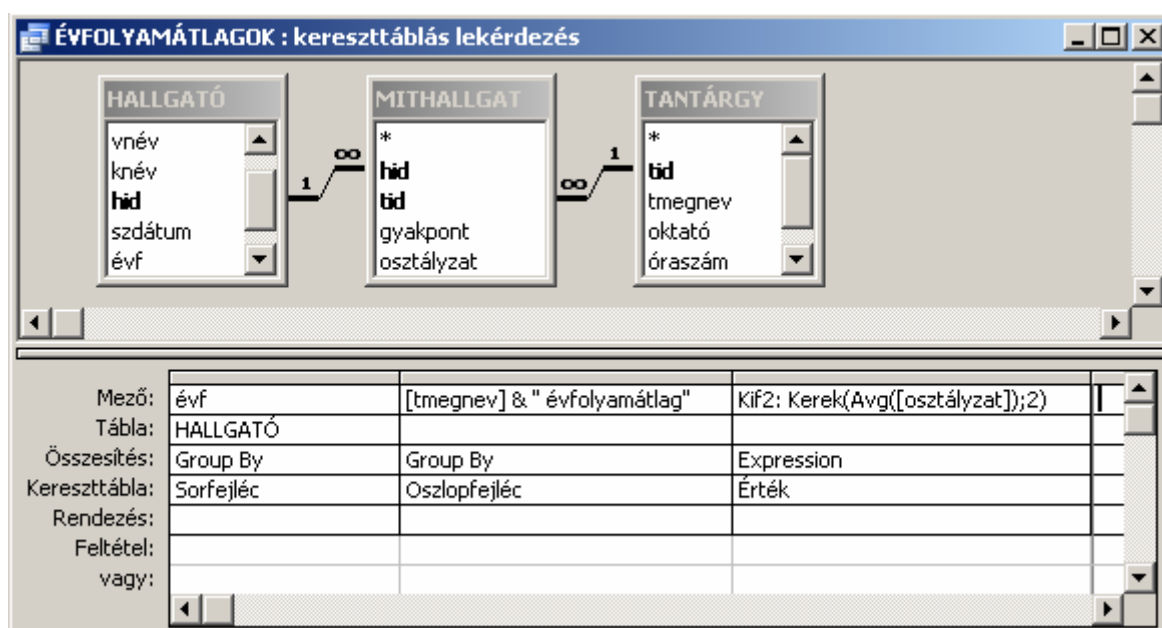
A kereszt táblás lekérdezés egy kimutatáshoz hasonló táblázatot hoz létre. Meg kell adnunk legalább egy sor- és egy oszlopfejléceket és egy értéket. A sor és oszlopfejlécek alapján fog összeadni és az értékmezőnél beállított függvényt fogja kiszámolni. A kereszt táblás lekérdezés mindig összesítő lekérdezés. Tehát a lekérdezés tervezőablakában megjelenik az összesítő lekérdezésnél megismert Összesítés sor a már megismert választási lehetőségekkel. Ezen kívül megjelenik egy Kereszt tábla sor is. Itt a következő lehetőségek közül választhatunk minden mezőnél:

- Sorfejléc: a mező értékei alkotják a táblázat sorainak elnevezéseit.
- Oszlopfejléc: a mező értékei alkotják a táblázat oszlopainak elnevezéseit.
- Érték: a mező értékeiből számítja ki a program a megadott csoport függvényét.
- (nem látható): ez a mező csak csoportosításban, rendezésben, vagy feltételben szerepel, de a táblázatban nem jelenik meg.

Adatlap nézetben nem látszik, hogy az oszlopfejléc és a számított értékek melyik mezőből származnak. Érdekes lehet a tervezésnél erre gondolni és úgy megfogalmazni az oszlopfejléc kifejezést, hogy szövegösszefűzéssel hozzáfűzzük, hogy milyen értékek találhatók a táblázatban.

Példa

1. Készítsünk lekérdezést, ami évfolyamonként és tantárgyanként megadja az osztályzatok átlagát. Mentsük a lekérdezést ÉVFOLYAMÁTLAGOK néven! A megoldás a 6.17. ábrán látható.



6.17. ábra: Kereszt táblás lekérdezés

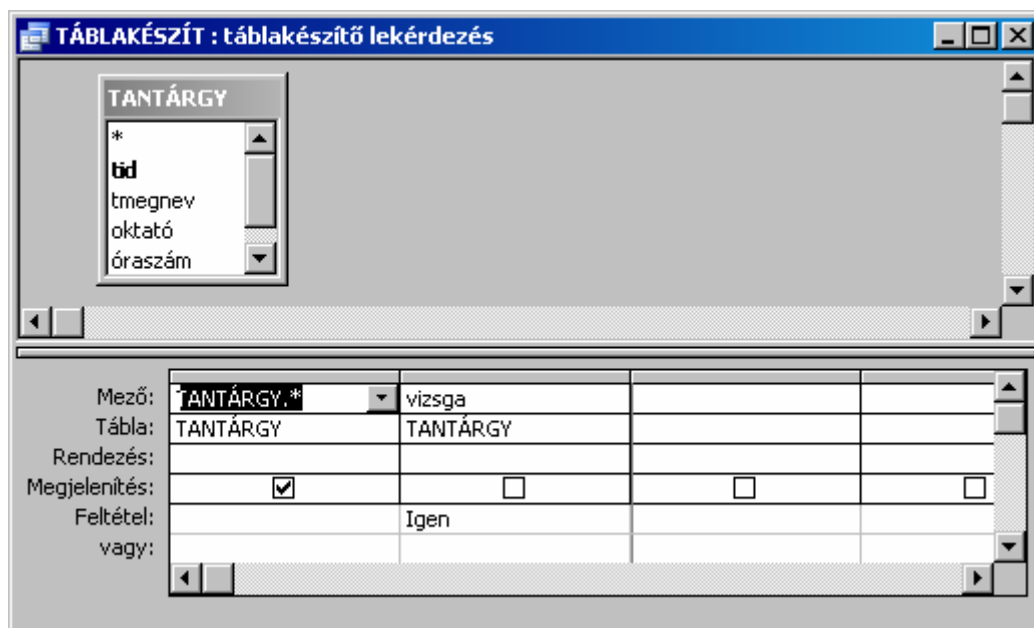
Táblakészítő lekérdezés

Ezzel a lekérdezéssel meglévő táblákból és nézetekből egy új, adatokkal feltöltött táblát hozunk létre. A lekérdezésnél meg kell adnunk, hogy mi legyen a neve az új táblának és hogy az a jelenlegi adatbázisba kerüljön, vagy egy másikba. Ezután a lekérdezést egy egyszerű választólekérdezésként kell megadnunk. Az új tábla a lekérdezésben megadott

mezőkkel és adatokkal fog létrejönni. A futtatás és az adatlap nézet itt különbözik. Az adatlap nézet csak megmutatja, hogy hogy néz ki a lekérdezés, a futtatás viszont létre is hozza az új táblát. Újra futtatás névegyezés miatt nem lehetséges, csak ha a korábban létrehozott táblát töröljük. Az új táblában az adattípusok megmaradnak, de nem lesznek indexek és elsődleges kulcs sem. Ezeket utólag, a tábla tervező nézetében létrehozhatjuk.

Példa

- Definiáljunk egy új táblát VIZSGÁK néven, ami a tantárgy táblából tartalmaz minden adatot azokról a tantárgyakról, amiből van vizsga! A megoldás a 6.18. ábrán látható.¹



6.18. ábra: Táblakészítő lekérdezés

Frissítő lekérdezés

A frissítő, vagy módosító lekérdezés segítségével a tábla adataiban akárhány sor módosítása hajtható végre egyszerre. A tervező rácsból hiányzik a rendezés és a megjelenítés sor, mert ez a lekérdezés a tábla tartalmát módosítja, nem megjeleníti. Ehelyett szerepel a módosítás sor, ahol megadhatjuk, hogy az adott mező értékei mire módosuljanak. Ha nem adunk meg feltételt, akkor a tábla minden rekordjában módosul a mező értéke, ha adunk feltételt, akkor csak a feltételnek megfelelő rekordokban. Adatlap nézetben megnézhetjük, hogy melyik rekordok fognak módosulni. Futtatáskor kiírja a program, hogy a módosítás hány sort érint, és nyugtázás után végrehajtja a módosítást. Ha egy rekordban a mező értéke korábban is ugyanaz volt, mint amire most módosítjuk az akkor is módosításnak számít. A módosítás sorba beírhatunk konkrét értéket, vagy kifejezést is.

Ha a módosítás során egy, vagy több rekordban az adott mező értéke olyan értékre módosulna, ami nem felel meg a tábla definíciójának, akkor figyelmeztető üzenetet kapunk,

¹ Igen óvatosan bánjunk az ilyen redundáns táblák létrehozásával! Csak akkor használjuk, ha az adatbázisnak egy adott pillanatban meglévő szerkezetét és állapotát akarjuk az új táblában rögzíteni!

Ha ugyanis egy tantárgynál a vizsgakötelezettség megváltozik, akkor nem csak a TANTÁRGY, hanem a VIZSGÁK táblát is módosítanunk kell.

Sokkal helyesebb, ha tábla helyett tárolt lekérdezőként, nézetként definiáljuk VIZSGÁK-at. Ugyanúgy használhatjuk, mintha tábla lenne, de így mindig szinkronban lesz a TANTÁRGY és a VIZSGÁK tartalma.

amiből kiderül, hogy hány rekordot nem lehet módosítani és milyen okból. Eldönthetjük, hogy mi történjen: nem futtatjuk a lekérdezést és módosítunk a lekérdezésen, vagy figyelmen kívül hagyjuk a figyelmeztetést és a szabályt nem sértő rekordokra végrehajtjuk a módosítást.

Ha egy módosító lekérdezést többször lefuttatunk, akkor minden futtatásnál a módosult adatok tovább módosulnak. Ha végrehajtottunk egy frissítést, akkor azt már nem tudjuk visszavonni.

Ez a lekérdezés típus megfelel az SQL UPDATE utasításnak (ld. 5.6.3.4. pont).

Példa

1. Készítsünk frissítő lekérdezést, ami a HALLGATÓ tábla rekordjaiban az évfolyam értékét megnöveli 1-gyel! Ha vannak olyan hallgatóink, akik ötödévesek, akkor a frissítést nem tudjuk végrehajtani, mert megsértjük azt az érvényességi szabályt, hogy az évfolyam 1 és 5 között kell legyen. Módosítsuk a lekérdezést úgy, hogy azokra a rekordokra vonatkozzon, ahol az évfolyam nem 5.¹ Ha többször futtatnánk a lekérdezést, akkor legkésőbb a 4. futtatás után eljutnánk oda, hogy minden hallgató ötödéves lenne. Ezt ne tegyük meg. Csak egyszer futtassuk a lekérdezést! A megoldás a 6.19. ábrán látható.

6.19. ábra: Frissítő lekérdezés

Hozzáfüző lekérdezés +!

Ezzel a lekérdezéssel egy meglévő táblába tudjuk bevinni egy lekérdezés eredményét. Hasonlóan működik, mint a táblakészítő lekérdezés, csak itt előre kell létrehozni a táblát, amibe a választó lekérdezés adatai kerülnek. Vigyázni kell arra, hogy a lekérdezett mezők száma, típusa és sorrendje megfeleljen a tábla mezőinek.

Az utasítás megfelel a beágyazott szelekttel megadott INSERT SQL utasításnak (ld. 5.6.3.5. pont).

¹ Ezzel a formailag helyes megoldással azonban egy komoly logikai hibát vihetünk be az adatbázisba. Ha korábban voltak ötödévesek, akkor mostantól évfolyam szerint nem fogunk tudni különbséget tenni köztük és azok között, akik a módosítás eredményeként most lettek ötödévesek.

Törlő lekérdezés

Ezzel a lekérdezéssel akárhány rekordot tudunk törölni egy táblából. A tervezőrács a frissítő lekérdezés rácsához hasonló, csak a Módosítás sor helyett Törlés szerepel. A Törlés sornak csak akkor van jelentősége, ha a törlés feltételét beágyazott szelekttel másik táblából adnánk meg. Egyszerű törlés esetén nem kell vele foglalkoznunk, automatikusan a *Where* kulcsszó jelenik meg. Ha nem adunk meg feltételt, akkor a tábla minden rekordja törlődik. Ha a táblára idegen kulccsal hivatkozik másik tábla és olyan rekordot akarunk törölni, amire van hivatkozó rekord, akkor erről figyelmeztetést kapunk és ha a kapcsolat beállításánál nem engedjük meg a kaszkád törlést (ld. 4.4.3.2 és 6.1.2.6. pontok), akkor ugyanúgy, mint a frissítésnél eldönthetjük, hogy a törölhető sorok törlődjenek, vagy visszatérünk a tervezéshez.

Az utasítás megfelel a DELETE SQL utasításnak (ld. 5.6.3.3. pont).

6.1.4.3. ACCESS SQL

Az ACCESS adatbázis-kezelő háttérében is SQL motor működik. Mint azt eddig is láttuk, minden utasításunk gyakorlatilag egy SQL utasítás. Az előző pontokban tárgyalt lekérdezéseknél mindig lehetőségünk van az utasítás SQL kódjának megnézésére is a *Nézet* menü *SQL nézet* menüpontjával, vagy a nézetváltó ikonon az SQL választásával. Az utasítások szintaktikája többnyire megegyezik a 5. fejezetben bemutatottakkal. Bizonyos utasításokat az ACCESS automatikusan másként fogalmaz meg, de ha beírjuk, elfogadja amúgy is. Például a feltételeknél a szükségesnél több zárójelet használ, de ha mi írjuk, nem baj, ha csak a szükséges minimális számú zárójelet használjuk. Fontos szintaktikai különbség, hogy a táblák és a mezők nevét, ha nem egy szóból állnak, akkor []-be kell írni (ha egy szóból állnak, akkor sem okoz hibát, ha kiteszük a zárójelet)¹.

Van néhány olyan feladat, amit csak a grafikus lekérdezés-szerkesztő felület segítségével nem lehet megoldani. Ezekhez mindenképpen az SQL kódba kell beleírunk. Megtehetjük azt is, hogy a teljes lekérdezést SQL-ben írjuk meg, de választhatjuk azt a lehetőséget is, hogy amit tudunk, megszerkesztünk a grafikus felületen és csak utána módosítunk a kódon. Nézzünk néhány példát, amiben szükségünk van az SQL kódra.

Azonos sorok kihagyása

A lekérdezés nem biztos, hogy tartalmazza az eredeti tábla kulcsát, így előfordulhat, hogy ismétlődő sorok kerülnek a lekérdezett táblázatba. Ha nem szeretnénk, hogy az azonos sorok ismétlődjenek, akkor a kész lekérdezés SQL nézetében a SELECT kulcsszó után be kell írunk a DISTINCT kulcsszót.

Példa

1. Kérdezzük le, hogy *Quittner Pál* óráit hányadéves hallgatók hallgatják!

Ha a szokásos módon elkészítjük a lekérdezést, az oktato mezőhöz meghatározzuk a feltételt és megjelenítjük az évfolyam mezőt (a lekérdezéshez mindhárom táblázatot hozzá kell adnunk, mert a MITHALLGAT táblából ugyan nem kérünk adatot, de az teremti meg a kapcsolatot a másik két tábla között), akkor minden évfolyam annyiszor fog szerepelni, ahány olyan évfolyamos hallgató felel meg a feltételnek. Nekünk azonban elég minden évfolyamot egyszer megjeleníteni. Menjünk át SQL nézetbe és írjuk be a szükséges DISTINCT kulcsszót.

¹ Ezért gyakorlati szempontból javasoljuk, hogy minden objektum- és mezőnév csak egy szó legyen, vagy ha mégis több szóból akarjuk összerakni, akkor ezeket aláhúzás jellel (_) kapcsoljuk össze.

Mentsük a lekérdezést QUITTNER_ÉVFOLYAM néven! (A lekérdezés a módosítással együtt is választólekérdezés.)

Megoldás:

```
SELECT DISTINCT HALLGATÓ.evf
FROM HALLGATÓ INNER JOIN (TANTÁRGY INNER JOIN MITHALLGAT
ON TANTÁRGY.tid=MITHALLGAT.tid)
ON HALLGATÓ.hid = MITHALLGAT.hid
WHERE (((TANTÁRGY.aktato) = "Quittner Pál"))
ORDER BY HALLGATÓ.evf;
```

Ha korábban eltároltuk az OKTATÁS lekérdezést (ld. 6.1.4.1. pont), akkor egyszerűbben is megfogalmazhatjuk a feladatot:

```
SELECT DISTINCT evf FROM OKTATÁS
WHERE aktato = "Quittner Pál"
ORDER BY evf;
```

Egyesítő lekérdezés

Az egyesítő lekérdezéssel lehetőségünk van két (vagy több) tábla és/vagy tárolt lekérdezés (nézet) uniójának vagy metszetének (ld. 4.5.7. és 4.5.8. pont) előállítására. Ennek feltétele, hogy a komponensek unió kompatibilisek legyenek, egyező sorrendben lévő, azonos számú és típusú oszlopból álljanak. A lekérdezéseket megszerkeszthetjük külön és az SQL kódokat egymás alá másolva összefűzhetjük őket az UNION vagy az INTERSECT kulcsszóval. Az unió (UNION) mindkét objektum sorait szolgáltatja, a metszet (INTERSECT) csak azokat, amelyek mindkettőben benne vannak.

Példa

1. Készítsünk lekérdezést, ami kiírja az adatbázisban szereplő összes nevet (a hallgatók nevét összefűzve) és külön oszlopba, hogy „hallgató”, vagy „oktató”. Mentsük a lekérdezést NEVEK néven!

Megoldás:

```
SELECT [vnev]&" "&[knev] As NÉV, "Hallgató" As Beosztás
FROM HALLGATÓ
UNION
SELECT aktato, "Oktató"
FROM TANTÁRGY;
```

Beágyazott szelekt

Előfordulhat, hogy egy lekérdezés feltételét egy másik lekérdezéssel tudjuk megfogalmazni. Ehhez a feltételbe a lekérdezést SQL kóddal kell beírunk.

Példa

1. Kérdezzük le azon hallgatók adatait, akik egy évfolyamra járnak egy bizonyos kódú hallgatóval. A kódot paraméterként lehessen megadni! Mentsük a lekérdezést ÉVFOLYAMTÁRSÁK néven!

A megoldás a 6.20. ábrán látható. Ez a lekérdezés is paraméteres választó lekérdezés, a *kod* paraméter aktuális értékét kell megadnunk.

ÉVFOLYAMTÁRSAK : választó lekérdezés

HALLGATÓ

- * (selected)
- vnév
- knév
- hid
- szdátum

Mező:	HALLGATÓ.*	évf	
Tábla:	HALLGATÓ	HALLGATÓ	
Rendezés:			
Megjelenítés:	☒	☐	☐
Feltétel:	(select évf from HALLGATÓ where hid=[kod])		
vagy:			

6.20. ábra: Szelekt a feltételben

Egyéb SQL utasítások

Lekérdezésként indítva, ha áttérünk SQL nézetre majdnem minden korábban tanult SQL utasítást megírhatunk és futtathatunk. Használhatjuk a

- CREATE TABLE,
- DROP TABLE,
- ALTER TABLE,
- INSERT,
- UPDATE,
- DELETE

utasításokat. Az utasítások szintén elmenthetők. Természetesen bizonyos utasítások nem futtathatók többször. Például nem hozhatunk létre két ugyanolyan nevű táblát. Az első három felsorolt utasítás adatdefiniáló lekérdezés néven szerepel (📄).

6.1.5. Űrlap

Az adatkezelésnél már említettük, hogy adatok kezelésére (bevitelére, módosítására, törlésére) lehetőség van űrlapokon keresztül is (ld. 6.1.3. pont). Az ACCESS változatos lehetőségeket ajánl fel az űrlapokkal kapcsolatban. Gyorsvarázsló segítségével létrehozhatunk automatikus űrlapokat a tábláinkhoz, lekérdezéseinkhez, vagy varázsló segítségével összetettebb űrlapokat készíthetünk. Tervező nézetben formázhatjuk, átalakíthatjuk űrlapjainkat. Lehetőségünk van olyan felhasználóbarát űrlapok létrehozására, amelyek megkönnyítik az adatbevitelt, minimálisra csökkentik a hibázás lehetőségét listák alkalmazásával. Ezek elkészítése azonban már komoly munkát igényel.

Űrlap készíthető tábláinkból és a legtöbb lekérdezésünkből is. Kivétel ez alól az adatdefiniáló, a frissítő, a törlő, hozzáfűző és táblakészítő lekérdezés. Ha az űrlap olyan mezőket tartalmaz, amelyek egyértelműen tartoznak egy tábla rekordjaihoz, akkor tudunk

rajtuk módosítani, ha nem, akkor csak megnézni tudjuk őket átláthatóbb formában, mint az adott objektum adatlap nézetében.

Tekintsük át az űrlapok létrehozásának lehetőségeit a legegyszerűbbtől egy összetettebb feladat megoldásáig.

6.1.5.1. Űrlap készítése gyorsvarázslóval

A gyorsvarázslóval készített űrlap (az ACCESS ezt autoűrlapnak nevezi) a tábla, vagy lekérdezés minden mezőjét beviteli mezőben jeleníti meg egy oszlopos űrlapon. Az oszlopos űrlap azt jelenti, hogy egyszerre csak egy rekord adatai láthatók, a rekord mezői pedig egymás alatt oszloposan helyezkednek el.

Ha a tábla idegen kulccsal hivatkozik egy másik táblára, akkor összetett űrlap jön létre, azaz nem csak a hivatkozó tábla adatait tudjuk megtekinteni, hanem a másik, a hivatkozott táblában lévő, hozzá kapcsolódó adatokat is. A kapcsolódó adatok egy adatlap űrlapon jelennek meg.

Az adatlap űrlap úgy néz ki, mint a tábla adatlap nézete. Az űrlap alján mindig látható, hogy hány rekordunk van összesen, és azok közül melyiknél tartunk. Léptető nyilak segítségével léphetünk a rekordok között. A 6.21. ábrán a HALLGATÓ táblából létrehozott autoűrlap látható.

vnév	knév	hid	szdátum	évf	tulaj
Dévaj	Dávid	00005	1984.05.07.	2	SQL00

tid	gyakpont	osztályzat
AB00	90	5
ELFS	20	2
*	0	

Rekord: 1 összesen 2

6.21. ábra: Gyorsvarázslóval létrehozott űrlap

Ilyen űrlap létrehozásának legegyszerűbb módja, ha objektumaink közül kiválasztjuk azt, amelyikből az űrlapot készíteni szeretnénk és az eszköztáron a  ikont választjuk, vagy a *Beszúrás* menüből az *AutoŰrlapot* választjuk.

Az űrlapot a szokásos módokon menthetjük. Nevét a mentéskor kell neki adnunk.¹

Az űrlap gyorsvarázsló használatának másik módja, ha az objektumtípusok közül az űrlapot választjuk. Az adatbázisablak eszköztárán a  űi ikonnal új objektumot tudunk létrehozni. A megjelenő ablakban megadhatjuk, hogy melyik táblából, vagy lekérdezésből készüljön az űrlap. A felsorolt listában több *AutoŰrlap* lehetőséget találunk. Ezek:

- Oszlopos: ha itt választjuk az oszlopos elrendezést, akkor a kapcsolt tábla adatai nem jelennek meg.
- Táblázatos: a rekordok egyszerre jelennek meg a képernyőn (ha túl sok van, akkor görgetősáv jelenik meg). Minden adat beviteli mezőben van. Hasonlít az adatlaphoz, de nem ugyanaz.
- Adatlap: megegyezik a tábla, vagy lekérdezés adatlap nézetével.
- Kimutatás és Kimutatásdiagram: ezek az adatok kimutatásszerű megjelenítésére szolgáló nézetek valójában és nem igazán illeszkednek az űrlapok logikájába ezért e könyv kereteiben nem tárgyaljuk őket.

Példa

1. Hozzunk létre gyorsvarázslóval űrlapot a HALLGATÓ táblából! A megoldás a 6.21. ábrán látható.

6.1.5.2. Űrlap létrehozása varázslóval

Ha az új objektum listájából a varázslót választjuk, akkor nem kell az első lapon megadnunk a táblát, ugyanis varázslóval több táblából, lekérdezésből is készíthetünk egy űrlapot. Ha az Űrlapok objektumlistájából az *Űrlap létrehozása varázsló segítségével* pontot választjuk, akkor helyből erre a második oldalra jutunk. Itt tudjuk megadni, hogy melyik táblák, lekérdezések melyik mezőit szeretnénk az űrlapra tenni. Csak olyan táblák adatait kombinálhatjuk, melyek kapcsolatban állnak egymással. Ha olyan adatokat szeretnénk az űrlapra tenni, amelyek nem kapcsolódnak, akkor figyelmeztetést kapunk és nem tudunk továbblépni.

Ha csak egy táblából választottunk, akkor a következő lépésben azt kell megadnunk, hogy milyen felépítésű legyen az űrlap. Itt a megismert lehetőségeken kívül a sorkizárt szerepel még, ami elrendezésében olyan, mint a táblázatos, de egyszerre csak egy rekord látszik.

A következő lépésben az űrlap stílusát választhatjuk ki különböző beépített stílusok közül. Ha ezt is megadtuk, akkor már csak az űrlap nevét kell jóváhagynunk és eldönthetjük, hogy a munkát tervezés, vagy űrlap nézetben folytatjuk.

Ha kapcsolódó táblákból választottunk, akkor a következő lépésben eldönthetjük, hogy egyszerű űrlapunk legyen, azaz minden adat egyetlen űrlapon jelenjen meg, vagy a két tábla adatai különüljenek el. Ez utóbbi esetben még mindig két lehetőségünk van. A segédűrlap, amikor a csatolt tábla űrlapja beleágyazódik a főűrlapba és a csatolt űrlap, amikor a csatolt tábla űrlapja külön űrlapként jelenik meg. A főtábla űrlapján egy gomb segítségével tudjuk ki-be kapcsolni a csatolt űrlapot.

A következő ablakban megadhatjuk a szerkezetet, utána a stílust és végül a nevet. Mivel akár segédűrlapot, akár csatolt űrlapot választottunk gyakorlatilag két űrlapunk van, természetesen két nevet kell adnunk.

¹ Ha csak egy táblázatba/lekérdezésbe viszünk be adatokat, akkor az űrlapot és a táblázatot/lekérdezést is célszerű azonos névvel jelölni. Ha egy táblázathoz/lekérdezéshez több űrlap tartozik, vagy egy űrlappal több táblázatba/lekérdezésbe viszünk be adatokat, akkor az űrlap nevével utaljunk arra, hogy az milyen objektum(ok)hoz kapcsolódik.

Példa

1. Készítsünk űrlapot tantárgyanként a hallgatók osztályzatáról! A hallgatóknak jelenítsük meg a teljes neve mellett az évfolyamát is. A főűrlapot mentjük TANTÁRGYANKÉNTI_OSZTÁLYZATOK néven, a segédűrlapot pedig HALLGATÓ_LISTA néven!

A megoldás során a következő lépéseket kell végrehajtanunk:

- Az *űrlap varázsló*ban a HALLGATÓ táblából kiválasztjuk a vezetéket és keresztnévet valamint az évfolyamot, a MITHALLGAT táblából az osztályzatot és a TANTÁRGY táblából a tárgyak megnevezését.
- A második lépésben az adatok elrendezésénél a TANTÁRGY táblát emeljük ki. (Nézzük meg, hogy a másik két tábla választásával hogy nézne ki az űrlap felépítése.) Válasszuk a mintaablak alatt az *Űrlap segédűrlappal* beállítást (ld. 6.22. ábra).

6.22. ábra: Űrlap varázsló

- A harmadik lépésben a segédűrlap szerkezetét adhatjuk meg, ez lehet táblázatos, vagy adatlap (a kimutatásokat ne válasszuk).
- A nekünk tetsző stílus kiválasztása után adjunk az űrlapoknak megfelelő nevet és nézzük meg a végeredményt. Egy lehetséges megoldás a 6.23. ábrán látható.

TÁRGYANKÉNTI_OSZTÁLYZATOK

tmegnev:

HALLGATÓ_LISTA

vnév	knév	évf	osztályzat
Álvó	Áladár	2	5
Dévaj	Dávid	2	2
Kutas	Lajos	2	2

Rekord: összesen 10

Rekord: összesen 5

6.23. ábra: Varázslóval készített űrlap a tantárgyankénti osztályzatokról

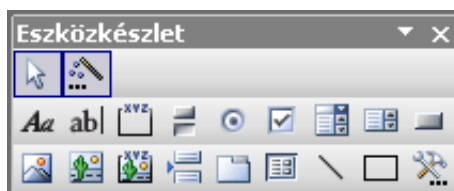
6.1.5.3. Űrlap tervének módosítása

Tervező nézetben az űrlap űrlap nézetének módosítását tudjuk végrehajtani. Teljesen nulláról is összeállíthatunk egy űrlapot tervező nézetben, de általában érdemes varázslóval elkészíteni a vázat és csak a módosításokat készíteni el tervező nézetben. A könyv terjedelme nem teszi lehetővé minden lehetőség bemutatását, úgyhogy egy konkrét példán keresztül nézünk meg néhány hasznos opciót.

Az űrlap varázslóval történő létrehozása után menjünk tervező nézetbe. Minden űrlap három részből áll: űrlapfej, törzs és űrlapláb. A varázslóval készült űrlapnak se feje, se lába nincs, de a tervező nézetben látszik, hogy lehetne. A *Nézet* menüben ki és be tudjuk kapcsolni, hogy legyen-e űrlapfej és űrlapláb.

Ugyanígy a *Nézet* menüben kapcsolhatjuk be, hogy legyen-e oldalfaj és oldalláb. Ennek nyomtatásban van jelentősége. Ha nyomtatásban az űrlap nem fér el egy oldalon, akkor az oldalfaj és oldalláb tartalma minden oldalon megjelenik, míg az űrlapfej és űrlapláb csak egyszer jelenik meg az egész űrlap elején és végén. Az űrlap minden részének mérete egérvonszóval módosítható.

Tervező nézetben megjelenik az *Eszközkészlet* eszköztár (6.24. ábra), ami szintén a *Nézet* menüben kapcsolható ki-be. Ezen az eszköztáron találjuk meg az űrlapon elhelyezhető vezérlőket. Az űrlapfejbe szeretnénk írni az űrlap címét. Ehhez a *Aa Felirat* eszközre van szükségünk. Kiválasztva az eszközt rajzoljunk egy téglalapot az űrlapfej részbe, majd írjuk bele a „Hallgatók adatai” szöveget. A téglalap a sarkainál és oldalainál fogva átméretezhető és a bal felső sarkánál fogva áthelyezhető.



6.24. ábra: Űrlap eszközkészlete

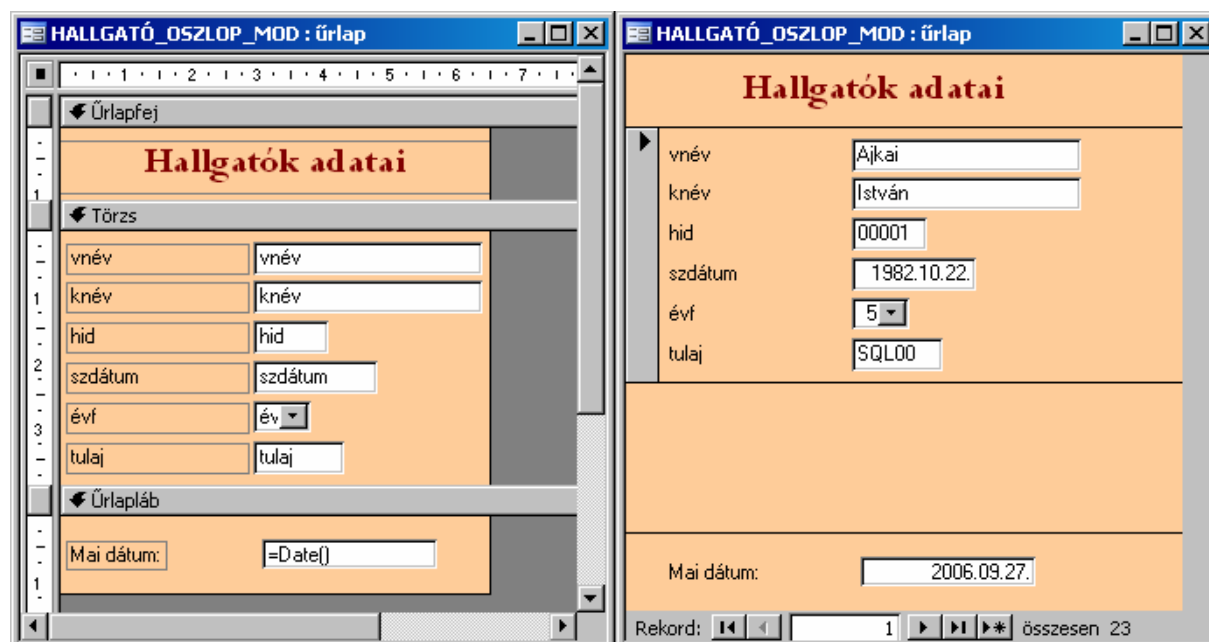
Az űrlap bármelyik része kijelölhető. Külön az űrlapon elhelyezett vezérlőelemek és külön az űrlap részei is a megfelelő szürke sávra kattintva. Az *eszköztár*, a *gyorsmenü*, vagy a *Nézet* menü *Tulajdonságok* pontjával az éppen kijelölt elem tulajdonságait nézhetjük és változtathatjuk meg.

Az űrlap bal felső sarkánál lévő jelölőlapka segítségével jelölhetjük ki az egész űrlapot és akkor a tulajdonságok ablak az egész űrlap tulajdonságait fogja tartalmazni. A tulajdonságok között vannak formai tulajdonságok: szín, betűtípus, méretbeállítások. Ezenkívül lehetnek adattulajdonságok, ahol az tekinthetjük meg, hogy az adott elem hogyan kapcsolódik az adatbázishoz. Láthatjuk például, hogy az űrlap megőrzi a kiinduló tábla szűrő és rendezés tulajdonságait. Az esemény tulajdonságoknál az űrlapelemhez kapcsolódó eseményekhez tudunk eljárásokat csatolni. Ezekre ebben a könyvben nem térünk ki. Mindig található egy Összes fül, ahol csoportosítás nélkül megtaláljuk együtt az összes tulajdonságot.

Példák

1. Hozzunk létre varázsló segítségével egy oszlopos űrlapot szabványos stílussal a HALLGATÓ tábla minden mezőjéből. Mentsük el HALLGATÓ_OSZLOP_MOD néven és nyissuk meg tervező nézetben. Módosítsuk az űrlapot úgy, hogy legyen címe, az alján látható legyen mindig az aktuális dátum és az évfolyamot listából lehessen kiválasztani. Az űrlap megjelenésén is módosítsunk (háttérszín, feliratok színe).

A megoldás tervező és űrlap nézete a 6.25. ábrán látható.



6.25. ábra: Hallgatók adatai űrlap tervező és űrlap nézetben

Állítsunk be az űrlapnak nekünk tetsző háttérszínt! Az űrlapfejben lévő címet formázzuk, emeljük ki!

Az űrlapládba helyezzünk el az eszközkészletről egy beviteli mezőt (**abl**)! A beviteli mező mindig két összekapcsolt elem, ugyanis a beviteli mezőhöz mindig tartozik egy felirat is. A felirat legyen az, hogy „*Mai dátum:*”, a beviteli mezőbe pedig állítsuk be kifejezésként a mai dátumot adó függvényt: `=date()`!

A következő lépés, hogy az évfolyamot ne beírni lehessen, hanem listából kiválasztani. Ehhez válasszuk ki az évfolyam beviteli mezőjét és a *gyorsmenüben*, vagy a *Formátum* menüben válasszuk a *Típus megváltoztatása* pontban megjelenő listából a *Kombi panelt*! Ezzel már legördülő listánk van, de a lista nem tartalmaz elemeket. Ehhez még a *Tulajdonságok* ablak *Adat* lapján be kell állítanunk, hogy milyen elemeket tartalmazzon a lista.

A sorforrás típusának állítsuk be, hogy *Lista* a sorforrás tulajdonsághoz pedig írjuk be az évfolyamokat pontosvesszővel elválasztva. Most már ha űrlap nézetbe megyünk, akkor láthatjuk, hogy valóban a listában csak az általunk megadott évfolyamok közül lehet választani.

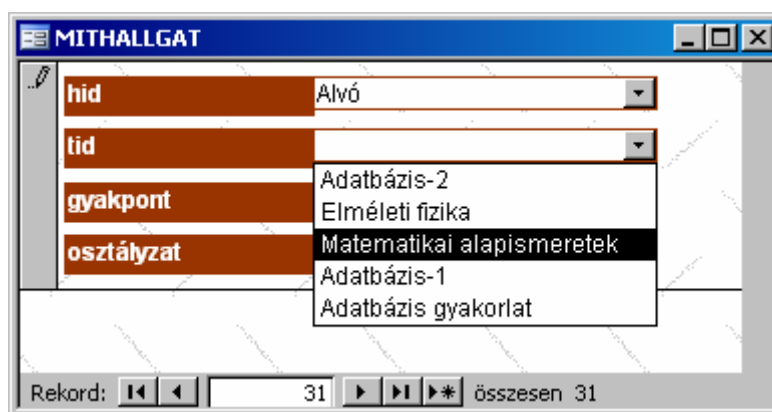
Vigyázat, ha az adatok nézegetése közben a listából másik évfolyamot választunk, a változtatás azonnal bekerül az adatbázisba visszakerdezés és megerősítés nélkül. Mentsük az űrlapot!

2. Hozzunk létre oszlopos űrlapot a MITHALLGAT tábla adataiból! Módosítsuk az űrlapot úgy, hogy HID és TID helyett listából lehessen kiválasztani a hallgatók és a tantárgyak nevét!

A megoldáshoz a HID beviteli mezőjét alakítsuk át listapanellé, majd módosítsuk a tulajdonságait a következőképpen:

- Sorforrás típusa: Tábla/lekérdezés
- Sorforrás: `SELECT HALLGATÓ.vnev, HALLGATÓ.knev, HALLGATÓ.hid FROM HALLGATÓ`; Az SQL utasítást a  gomb megnyomásával a szokásos grafikus módban szerkeszthetjük.
- Kötött oszlop: 3. Ez azt jelenti, hogy a lista 3. oszlopa fog bekerülni a táblába.
- Oszlopszám: 2. Ez azt jelenti, hogy a sorforrásnak megfogalmazott lekérdezés első két oszlopa jelenik meg a listában.

Ennek alapján könnyen megszerkeszthetjük a tid helyére is a tantárgyak listáját. Egy lehetséges megoldás a 6.26. ábrán látható.



6.26. ábra: MITHALLGAT űrlap legördülő listával

6.1.6. Jelentés

Az adatbázisban szereplő adatok rendezett megjelenítése, összesítésekkel való kiegészítése valósítható meg jelentések segítségével. Ugyanúgy, mint az űrlapoknál lehetőségünk van gyorsvarázsló, vagy varázsló használatára, vagy készíthetünk, módosíthatunk jelentéseket tervező nézetben.

Az adatbázis ablakban a Jelentéseket választva az ablak eszköztárán az  ikonra kattintva a következő lehetőségek közül választhatunk:

- Tervező nézet: jelentések létrehozásánál nem javasoljuk a tervező nézet használatát, csak meglévő jelentések módosításánál.
- Jelentés varázsló: több lépésben adhatjuk meg az igényeinknek megfelelő jelentés adatait.
- AutoJelentés: Oszlopos: gyorsvarázsló, melynek segítségével a megadott táblázatból, vagy lekérdezésből azonnal létrejön egy oszlopos jelentés. Ennek

elrendezése olyan, mint az oszlopos űrlapé, azaz az egyes tulajdonságok egymás alatt jelennek meg mellettük a tulajdonság értékével. Az egyes rekordok oszlopai egymás alatt jelennek meg. Összegzések nem kerülnek a jelentésbe. Az oldalak alján megjelenik az aktuális dátum és az oldalszám.

- AutoJelentés: Táblázatos: gyorsvarázsló, melynek segítségével a megadott táblázatból, vagy lekérdezésből azonnal létrejön egy táblázatos jelentés. Ennek elrendezése olyan, mint a táblázatos űrlapé, azaz az egyes rekordok a táblázat sorai, az egyes tulajdonságok pedig az oszlopai. Összegzések nem kerülnek a jelentésbe. Az oldalak alján megjelenik az aktuális dátum és az oldalszám.
- Diagram varázsló: a megadott táblából az értékeket tartalmazó mezőkből számolt csoport függvények különböző csoportosításban ábrázolhatók.
- Címke varázsló: segítségével a táblázat adatai címke elrendezésben jeleníthetők meg és nyomtathatók ki, így lehetővé téve, hogy a gyárilag készült etikett lapokra az adatokat megfelelően tudjuk kinyomtatni.

Jelentés készítésekor mindig választhatunk, hogy a kész jelentést *tervező*, vagy *nyomtatási kép* nézetben szeretnénk-e látni. A szokásos módon a *Nézet* menüből, vagy az eszköztár megfelelő ikonjával válthatunk a nézetek között.

Nyomtatási kép nézetben az eszköztár ikonjaival megadhatjuk, hogy hány lapot szeretnénk látni egyszerre, az ablak alján pedig láthatjuk, hogy hányadik oldalon vagyunk és a nyilak segítségével tudunk lépkedni a lapok között.

Az eszköztáron a *Beállítás* gombot megnyomva az *Oldalbeállítás* ablakot hozhatjuk elő, ahol margókat, az oldal elhelyezést és hasábokat állíthatunk be.

Tervező nézetben láthatjuk, hogy a jelentés is több részre tagolódik. Ezek:

- jelentésfej/jelentésláb: az egész jelentés elején és végén jelenik meg egyszer, pl. a jelentés címe, záró adatok.
- oldalfej/oldalláb: a jelentés minden oldalának tetején és alján megjelenik, pl. oldalszám.
- csoportosítási szintenként csoportfej/csoportláb: minden csoport elején és végén megjelenik, pl. a csoport közös adatai és a csoportra kiszámolt összesítések.
- törzs: a csoportokban szereplő, rekordonként ismétlődő elemek.

A jelentésfej/láb és az oldalfej/láb a *Nézet* menüben közvetlenül ki/be kapcsolható. A csoportosítási szintek a *Nézet* menü *Rendezés és csoportosítás* pontjában, vagy az eszköztár  ikonjával állíthatók be. Az egyes elemek átméretezése az űrlapoknál tanult módon működik.

Kész jelentésünket rich text formátumban exportálhatjuk a MICROSOFT WORD programba és ott a szövegszerkesztő program eszközeivel tovább formázhatjuk. Az exportálást megtehetjük a *Fájl* menü *Exportálás* parancsával, vagy az eszköztár  ikonjával.

Példák

1. Készítsünk gyorsvarázslóval táblázatos jelentést a HALLGATÓ tábla adataiból!

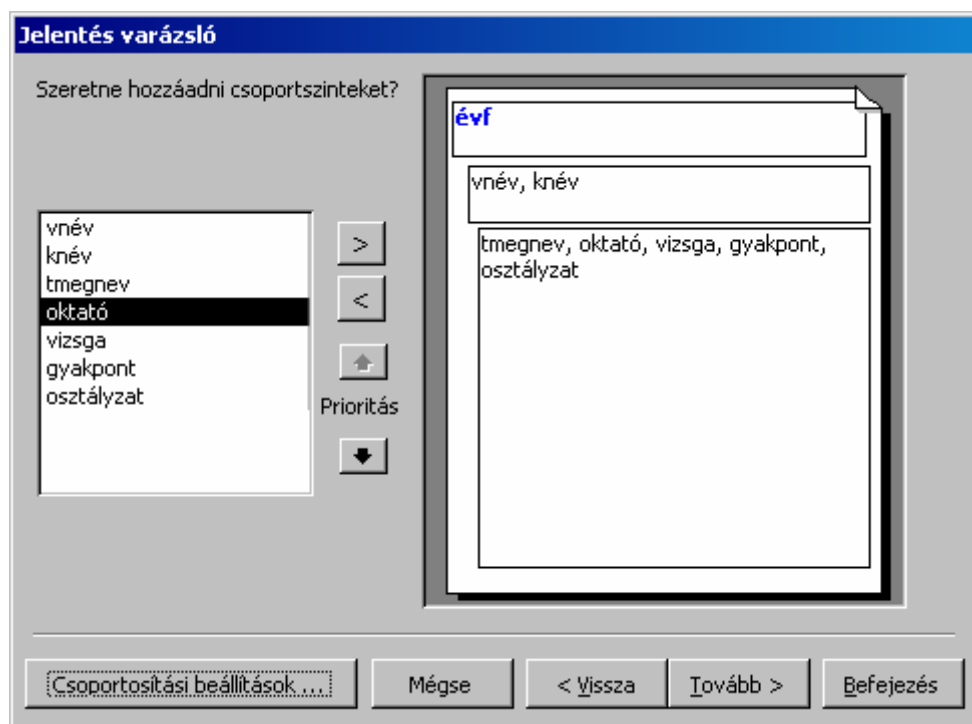
Az *Új jelentés* ablakban válasszuk az *AutoJelentés: Táblázatos* pontot és válasszuk ki a listából a HALLGATÓ táblát. A jelentés minden további kérdés nélkül megjelenik *Nyomtatási kép* nézetben. Mentsük a jelentést a szokásos módok valamelyikén HALLGATÓ_TÁBLÁZAT néven.

2. Készítsünk varázslóval jelentést hallgatónként az egyes tárgyakból elért eredményekről. A hallgatókról jelenítsük meg a nevüket és évfolyamukat. A tantárgyakról a nevüket, az

oktatójukat, hogy volt-e belőlük vizsga és a gyakorlati pontot és osztályzatot is. Átlagoljuk hallgatónként, évfolyamonként és összesen is az eredményeket.

A varázslónál megadhatjuk a HALLGATÓ, TANTÁRGY és MITHALLGAT táblákat külön is, de megadhatjuk a 6.1.4.1. pontban készített OKTATÁS nézetet is. Válogassuk ki a megfelelő mezőket (vnev, knév, évf, tmegnev, oktató, vizsga, gyakpont, osztályzat).

A második lépésben kell megadnunk, hogy melyik tábla alapján legyen a fő csoportosítás. Ez esetünkben a HALLGATÓ. A *Tovább* gomb megnyomása után további csoportszinteket adhatunk meg. Itt tudjuk kiemelni a hallgatók adatai közül az évfolyamot csoportosító szintként (ld. 6.27. ábra). Csoportok képzésekor a *Csoportosítási beállítások*¹ gomb segítségével megadható intervallum a csoportképzéshez (pl. kétévfolyamonként összesítsen). Ezt most nem módosítjuk.



6.27. ábra: Jelentés varázsló, csoportosítási szintek beállítása

A következő ablakban adhatjuk meg a rendezést és az *Összesítési beállításokat*². A csoportosító mezők alapján eleve megvalósul rendezés, csak azon belül adhatjuk meg most a rendezést. Ez legyen a tantárgymegnevezés alapján. Meg lehet adni több rendezési szintet és szintenként rendezési sorrendet. Mivel hallgatónként minden tantárgy csak egyszer szerepelhet az adatbázisunk definíciója szerint, felesleges több rendezési szintet megadni.

Az *Összesítési beállítások* gomb megnyomására a 6.28. ábrán látható ablakban tudjuk megadni, hogy melyik mezőkre (csak a számokat és a logikai értékeket tartalmazó mezők jelennek meg), milyen függvényeket szeretnénk kiszámoltatni. Adjuk meg a gyakorlati pontokra és az osztályzatra is az átlagot.

¹ A *Csoportosítási beállítások*nál dátum esetén megadhatjuk, hogy milyen egységenként csoportosítsunk (évenként, negyedévenként, havonta, stb.), szöveg esetén pedig a kezdő karakterek számát.

² Összesítési beállításokat csak akkor adhatunk meg, ha van csoportosítás.

Összesítési beállítások

Milyen összesítési értékeket szeretne számítani?

Mező	Össz	Átl	Min	Max
vizsga	☐	☐	☐	☐
gyakpont	☐	☒	☐	☐
osztályzat	☐	☒	☐	☐

OK
Mégse

Megjelenítés

☒ Törzs és Összesítés
☐ Csak Összesítés

☐ Az összegek százalékának számítása

6.28. ábra: Jelentés varázsló, összesítési beállítások

Ha ezekkel megvagyunk, akkor már csak formai beállításokat kell megadnunk: az elrendezést, a lapok tájolását (álló, vagy fekvő)¹, stílusát és a jelentés címét. Tájolásnak érdemes jelen feladatnál a fekvőt választani, vagy *Tömör* stílussal állót, a többi szabadon választható. A jelentés címe legyen: HALLGATÓI_EREDMÉNYEK. Tekintsük meg a jelentést nyomtatási képként. Megoldásunk egy részlete látható a 6.29. ábrán.

HALLGATÓI_EREDMÉNYEK

vnév	Nagy
knév	Zoltán

tmegnev	oktató	vizsga	gyakpont	osztályzat
Adatbázis-1	Quittner Pál	☒	20	1
Elméleti fizika	Albert Zweistein	☐	32	4

Összegezés 'hid' = 00019 (2 törzsrekord)

Avg	26	2,5

Összegezés 'évf' = 2 (12 törzsrekord)

Avg	53,333333	3,166667

évf 3

vnév	Boldog
knév	Boldizsár

tmegnev	oktató	vizsga	gyakpont	osztályzat
Adatbázis-1	Quittner Pál	☒	15	1

Összegezés 'hid' = 00003 (1 törzsrekord)

Avg	15	1

Oldal: 2

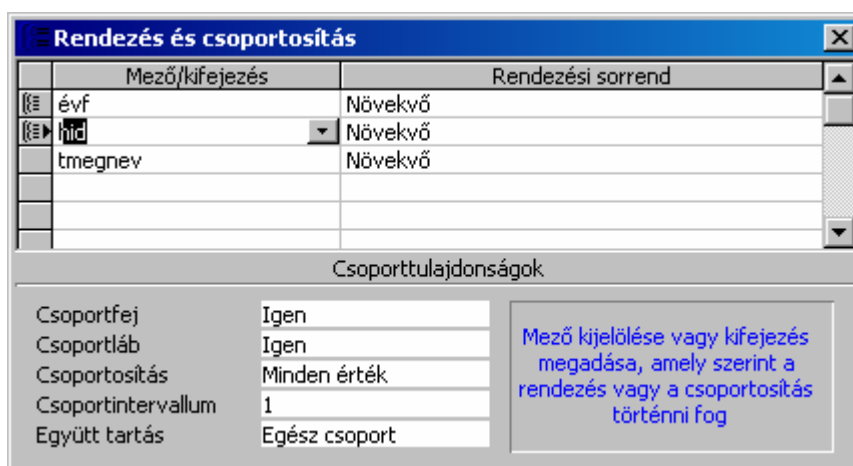
6.29. ábra: HALLGATÓI_EREDMÉNYEK jelentés egy részlete

¹ Ha sok mezőt akarunk egymás mellett megjeleníteni, akkor érdemes a fekvő, míg ha keveset, akkor az álló tájolást választani.

3. Módosítsuk az előző példában készített jelentést úgy, hogy a jelentés címe vízszintesen középre legyen igazítva, egy hallgató adatain belül ne legyen sortörés, minden évfolyam új oldalon kezdődjön és az összegzéseknél a felirat *Avg* helyett *Átlag* legyen. Az átlag értékek 1 tizedesre legyenek kerekítve. Mielőtt a módosítást elvégeznénk, készítsünk egy másolatot a jelentésről *HALLGATÓI_EREDMÉNYEK_RÉGI* néven. A módosított lekérdezést mentjük rá a *HALLGATÓI_EREDMÉNYEK* jelentésre és ha meggyőződünk róla, hogy a módosított jelentés megfelel a terveinknek akkor a *HALLGATÓI_EREDMÉNYEK_RÉGI* jelentést töröljük.

Nyissuk meg a jelentésünket tervező nézetben. Az űrlapoknál megtanult módon tudjuk a jelentés egyes részeinek tulajdonságait megváltoztatni. A következő lépéseket hajtsuk végre:

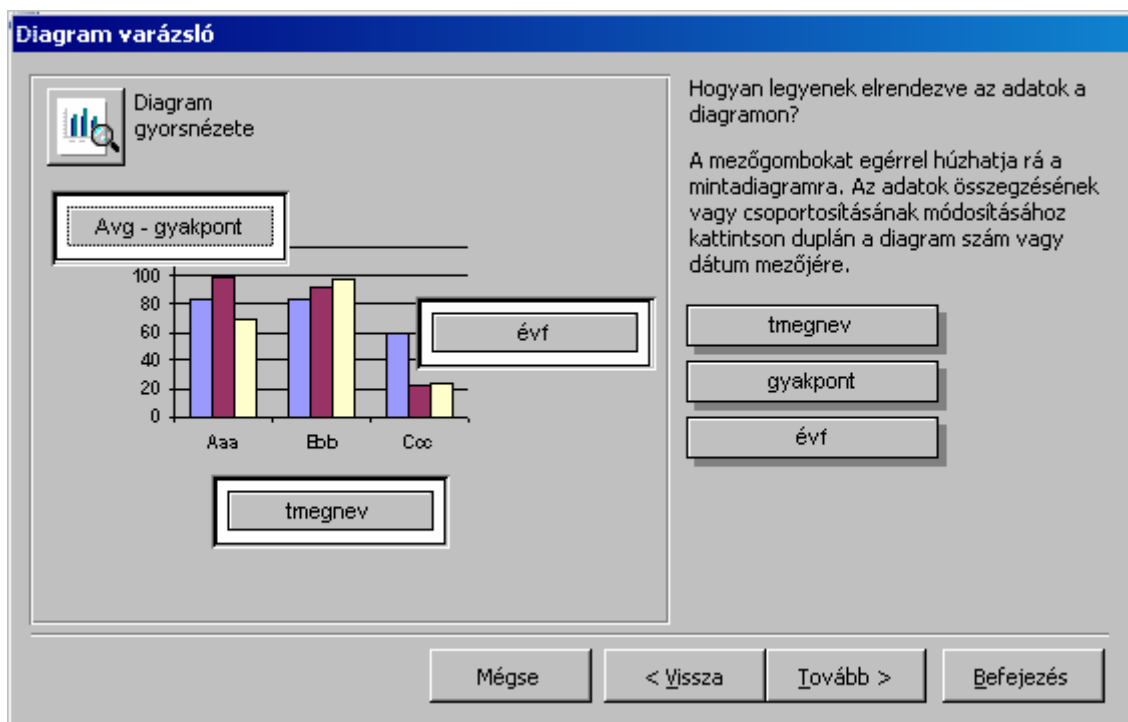
- A jelentésfejben lévő címkét szélesítsük ki a jelentés teljes szélességére és állítsunk be rá középre igazítást.
- Az eszköztáron válasszuk a  *Rendezés és csoportosítás* ikont. A megjelenő ablakban a *hid*-hez megadhatjuk, hogy az *Együtt tartás* az egész csoportra vonatkozzon (ld. 6.30. ábra).



6.30. ábra: **Rendezés és csoportosítás ablak**

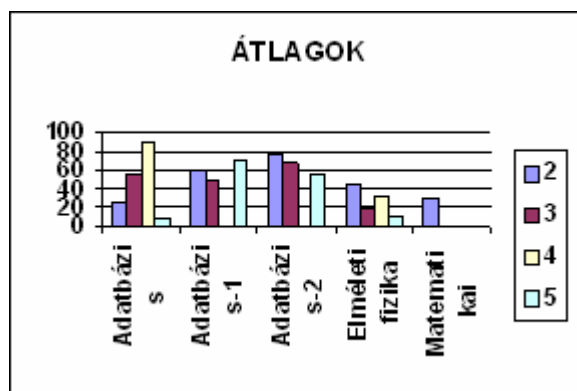
- Az oldaltörés megadásához be kell kapcsolni az *Eszközkészletet (Nézet menü)*. Itt válasszuk a  ikont és kattintsunk az *évf lábléc* alá. Ezzel a lábléc tartalma alatt mindig oldaltörés fog következni.
 - A *hid* és az *évf* láblécben is írjuk át az *Avg* címke tartalmát *Átlag*-ra.
 - Az átlagértéket tartalmazó mezőben módosítsuk a képletet, hogy az átlag 1 tizedesre kerekítve jelenjen meg (a *Round* függvény segítségével).
4. Készítsünk oszlopdiagramot évfolyamonként a tantárgyak átlagpontszámáról! Mentsük a diagramot *ÁTLAGOK* néven.

Az *Új jelentés* ablakban válasszuk a *Diagram varázslót* és adjuk meg az *Oktatás* nézetet kiindulásnak. A következő ablakban a diagram mezőinek adjuk meg az évfolyamot, a tantárgyak megnevezését és a gyakorlati pontszámot. Ezután adjuk meg, hogy oszlopdiagramot akarunk készíteni. A következő ablakban kell megadnunk, hogy melyik mező milyen módon szerepeljen a diagramon. A tengelyen szerepeljenek a tárgyak, a különböző oszlopok jelentsék az évfolyamokat és az ábrázolt mennyiség a gyakpontok összege helyett az átlaguk legyen (ld. 6.31. ábra). (A téglalapra duplán kattintva lehet más függvényt választani.)



6.31. ábra: Diagram varázsló elrendezés ablaka

Ezek után már csak nevet kell adnunk a diagramnak. Ez legyen ÁTLAGOK. Az elkészült diagramon tervező nézetben természetesen lehet szépíteni. Változtatás nélkül az általunk készített diagram látható a 6.32. ábrán.



6.32. ábra: Évfolyamonkénti tantárgyátlagok diagramja

6.1.7. Biztonság, adatvédelem

Ebben a pontban két fontos dologra kell kitérnünk. Az egyik az adatok illetéktelen hozzáférők elleni védelme, a másik az adatbázis biztonsága abban az értelemben, hogy a használók ne ronthassák el azt. Az adatvédelemmel kapcsolatos menüpontokat az ACCESS-ben az *Eszközök* menü *Adatvédelem* pontjában találjuk.

Az illetéktelen felhasználók elleni védelem legegyszerűbb lehetősége az, hogy jelszóval látjuk el az adatbázist. Ebben az esetben az Access az adatbázis megnyitásakor kéri a jelszót és annak ismeretének hiányában nem nyitja meg az adatbázist. Már nyitott adatbázisban azonban bármit megtehet a felhasználó. Mód van felhasználói szintű védelem beállítására is oly módon, hogy különböző felhasználói csoportokat hozunk létre és a

csoportokhoz rendeljük a megfelelő jogosultságokat. Az Access a jogosultságokra vonatkozó információkat egy *munkacsoport információs fájl*ban tárolja.

A felhasználói szintű védelem segítségével meghatározhatjuk, hogy az egyes felhasználói csoportok számára mely objektumok (táblák, lekérdezések, űrlapok, jelentések) láthatók és hogy milyen műveleteket végezhetnek rajtuk. Ezekon kívül a teljes adatbázisra vonatkozó jogosultságokat is beállíthatunk. Ezek az adatbázis többszörözésének joga, jelszó megadásának joga és az indítási beállítások módosításának joga. Ha például egy felhasználó jelszót adhat meg az adatbázis egy részének eléréséhez, azzal korlátozza az adott rész elérését mások számára. Annak a felhasználói csoportnak, akinek a fenti jogosultságokat meg akarjuk adni, *Rendszergazda* engedélyt kell adnunk. A *Rendszergazda* engedély a fent felsorolt mindhárom jogosultságot együtt jelenti. Azok nem kezelhetők külön egymástól.

Az adatbázis-kezelő programok fontos jellemzője, hogy milyen biztonsági megoldásokat kínálnak arra, hogy az adatbázis integritása ne sérüljön. Az Access adatbázis-kezelőben lehetőségünk van többek között idegen kulcsok beállítására, feltételek megfogalmazására, külső adatok importálása helyett csatolására és a többszörözött adatbázisok szinkronizálására.

6.1.8. Több felhasználós kezelés

Az Access adatbázis-kezelő elsősorban egyfelhasználós működésre van optimalizálva, de a program újabb verziói lehetőséget nyújtanak többfelhasználós kezelésre is. A lehetőségek:

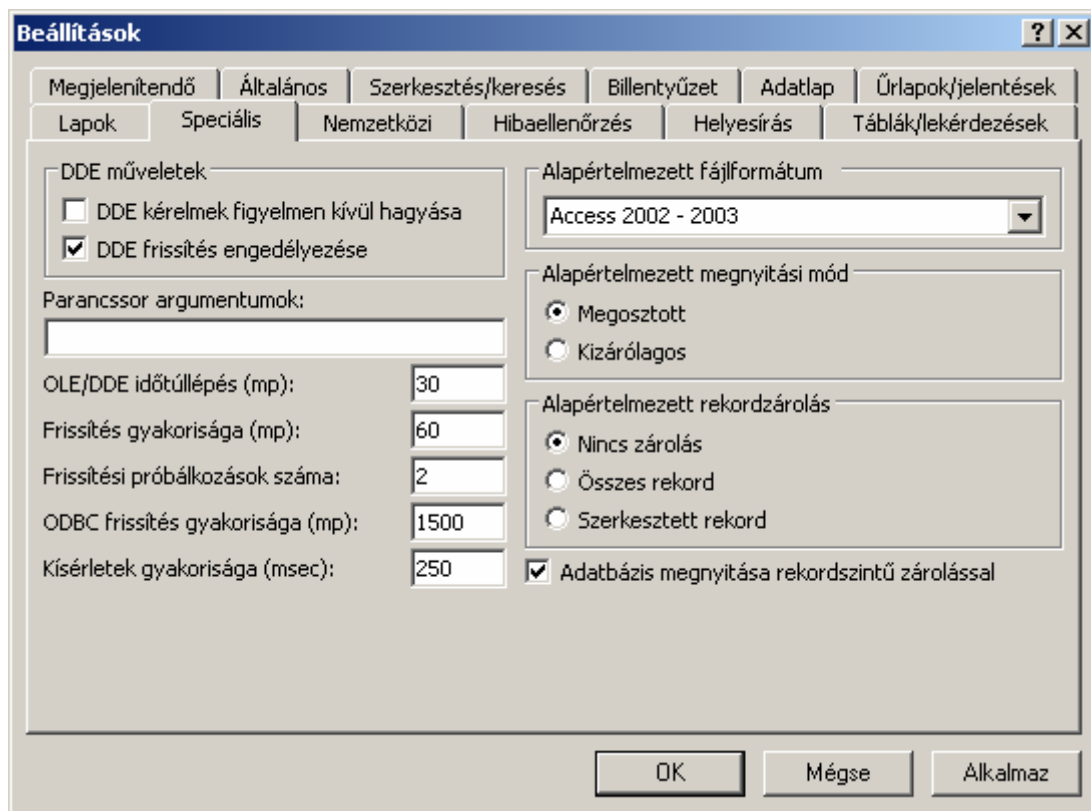
- az adatbázis a hálózaton van, és minden felhasználó ugyanazzal az adatbázissal dolgozik párhuzamosan
- az adatbázisból többszörözéssel több példány készül, ezeket a példányokat használja minden felhasználó a saját gépén, természetesen ebben az esetben időről időre szükség van az adatbázisok szinkronizálására
- a kettő közötti köztes megoldás az úgynevezett előtér-háttér (front-end, back-end) konfiguráció, azaz a táblákat tartalmazó adatbázis a hálózaton van és az egyes felhasználók gépein a táblákra mutató hivatkozásokat tartalmazó adatbázis-objektumok vannak.

Természetesen ha az első megoldást használjuk, vagy a harmadik megoldásban a hálózaton lévő táblákot módosítunk, akkor szükség van a 8.3. alfejezetben tárgyalt zárrakra. Access-ben is különböző szintű zárrak használatára van lehetőség. A legtágabb, ha az egész adatbázist kizárólagos módban nyitjuk meg. Ez gyakorlatilag azt jelenti, hogy az egész adatbázist elzárjuk mások elől. A felhasználói jogosultságok beállításánál meg lehet adni, hogy az adott felhasználói csoport megnyithassa-e az adatbázist kizárólagos módban.

Ha nem az egész adatbázist zárjuk el, akkor választhatunk rekordszintű, vagy lapszintű zárolás között. A zárolási beállításokat az *Eszközök* menü *Beállítások* pontjára megnyíló ablak *Speciális* fülén adhatjuk meg (ld. 6.33. ábra). Ha ezt beállítjuk, akkor a következő megnyitástól lesz érvényes. Ha rekordszintű zárolást állítunk be, akkor még mindig vannak további lehetőségeink a zárolási stratégia meghatározására:

- a *Nincs zárolás* kapcsolóval megakadályozhatjuk a rekordok szerkesztés közbeni zárolását: ez az alapértelmezés, általában csak akkor használják, ha egy felhasználó van csak. „Optimista” zárolásnak is nevezik. Ha két felhasználó akarja ugyanazt a rekordot módosítani, akkor amelyik másodszor akarja menteni, az üzenetet kap, amely után eldöntheti, hogy mi történjen az ő módosításával.
- az *Összes rekord* kapcsolóval szerkesztéskor az adott űrlap, vagy adatlap összes rekordját zárolhatjuk: ez idő alatt a rekordok megtekinthetők, de nem módosíthatók, nem törölhetők és új rekordokat sem vehetünk fel.

- a *Szerkesztett rekord* kapcsolóval csak az éppen szerkesztett rekordot zárolhatjuk. Ezt „pessimista” zárolásnak is nevezik.



6.33. ábra: Beállítások ablak Speciális fül

Űrlap és adatlap nézetben minden zárolt rekord jelölőlapkáján megjelenik a zárolt rekordot jelző szimbólum: . Ha optimista zárolást használunk, akkor is látszik a jelölőlapkán, ha egy másik felhasználó éppen szerkeszt egy rekordot. Ennek a jele: .

Beállíthatjuk azt is, hogy az Access milyen időközönként kísérelje meg elmenteni a más felhasználók által zárolt, megváltoztatott rekordokat, illetve milyen időközönként frissítse a rekordokat.

6.2. Internet-alapú adatbázis-kezelési technikák: PHP-MySQL

Az 5.7. alfejezetben tárgyaltuk az SQL utasítások programba ágyazását. Ebben az alfejezetben egy olyan példát fogunk bemutatni, amelyhez az eszközök ingyenesen telepíthetők. A befogadó nyelv a PHP lesz, melyben az SQL egy speciális változatát a MySQL nyelvet használjuk majd. A fejezetben szereplő példák az 5. fejezetben ismertetett egyetemi adatbázist használják.

6.2.1. Néhány szó a PHP nyelvről

A könyv keretei nem teszik lehetővé a PHP nyelv részletes ismertetését. Csak azokra a jellemzőkre és tudnivalókra térünk ki, amelyek a MySQL használatához feltétlenül szükségesek. A további érdeklődést kielégíthetik az irodalomjegyzékben található művek.

A PHP egy olyan programnyelv, melynek segítségével lehetséges webes adatbázis alapú alkalmazások működtetése. Jellemzően HTML oldalakon használják, de a PHP kód

csak a szerveren jelenik meg, a felhasználó már csak a kód által generált HTML oldalt látja. A könyv példáinál feltételezzük, hogy az olvasó ismeri a HTML nyelv alapjait.

A HTML oldalba ágyazott PHP utasítások kezdetét és végét meg kell jelölni. Ezt több módon megtehetjük. A hagyományos nyitó és záróelemek:

```
<?php
utasítások . . .
?>
```

Az utasításokat soronként kell írunk és pontosvesszővel (;) kell lezárunk.

A PHP kódba is tudunk HTML kódot ágyazni a print() függvény segítségével. A többi függvénytől eltérően a print függvény után a zárójelek elhagyhatók.

Az alábbiakban példát látunk a PHP kódba ágyazott HTML kódra:

```
<php
print "<h1>Ez a főcím</h1>";
?>
```

Gyakran lehet szükségünk a kódban szövegösszefűzésre, pl. változótartalmak és HTML kódok együttes használatára. PHP-ban a szövegösszefűzés a pont (.) karakterrel történik.

A programban használatos változók neve egy dollárjelből (\$) és az utána álló névből áll (például: \$a, \$elso_szam). A PHP nem deklaratív nyelv, ezért a változókat nem szükséges előre létrehozni és meghatározni adattípusukat, azok létrejönnek első használatkor és a bennük lévő adathoz igazodik a típusuk. Műveletvégzéskor a művelet és a műveletben szereplő értékek jellege szerint alapértelmezett típuskonverzió jön létre.

Ahhoz, hogy az adatbázisból származó adatokat fel tudjuk dolgozni, ismernünk kell a különböző vezérlési szerkezetek szintaktikáját is. Az elágazás szintaxisa:

```
if (feltétel) {
    // ha a feltétel igaz, akkor ezek az utasítások futnak le
} else {
    // ha a feltétel hamis, akkor ezek az utasítások futnak le
}
```

Ismétlésből több féléte ismer a programnyelv, ezek a While ciklus, a Do... While ciklus és a For ciklus. A szintaxisok az alábbiakban láthatók:

```
while (feltétel) {
    // ismétlődő utasítások
}

do {
    // ismétlődő utasítások
} while (feltétel);

for (ciklusváltozó=kezdeti érték; feltétel; módosítás){
    // ismétlődő utasítások
}
```

A HTML oldalakba ágyazott űrlapok segítségével kommunikálhatunk a felhasználóval. Az űrlap által küldött adatokat a \$_REQUEST globális tömb segítségével használhatjuk fel. Például egy űrlapot tartalmazó oldalon szerepel az alábbi vezérlőelem:

```
<input type="text" name="vezeteknev">
```

A feldolgozást végző PHP kódban a \$_REQUEST['vezeteknev'] kóddal tudjuk felhasználni a bekért adatot. Fontos megemlítenünk még egy függvényt, a mysql_real_escape_string()-et, mely a felhasználó által bekért karaktersorozatot úgy alakítja át, hogy perjeleket szűr a karakterláncba, hogy levédje az aposztrófokat és idézőjeleket¹. Ezt az átalakítást mindig el kell végeznünk.

6.2.2. Adatbázis-kezelés a PHP nyelvben

Mielőtt elkezdenénk dolgozni egy adatbázissal, csatlakoznunk kell a kiszolgálóhoz. Az erre szolgáló függvény a mysql_connect([számítógépnév],[felhasználónév],[jelszó]). Sikeres kapcsolatfelvétel esetén a függvény egy kapcsolatazonosítót ad eredményül, amit egy változóban eltárolunk. Ha hiba van, akkor a die() függvénnyel kiléphetünk a programból és a hiba leírását a mysql_error() függvénnyel kaphatjuk meg.

```
$kapcsolat = mysql_connect("localhost","felhasznalo","jelszo");  
if (!$kapcsolat){  
    die("mysql_error()");  
}
```

A kapcsolat bontására a mysql_close(kapcsolatazonosító) függvény szolgál.

Ha csatlakoztunk az adatbázis kiszolgálóhoz, akkor még ki kell választanunk az adatbázist a mysql_select_db(adatbázisnév, [kapcsolatazonosító]) függvény segítségével. Ha az adatbázis létezik és használhatjuk, akkor a függvény visszatérési értéke igaz.

```
$adatbazis = "pelda";  
mysql_select_db($adatbazis) or die (mysql_error());
```

Adatkezelő utasítások kiadására szolgáló utasítás PHP-ben a mysql_query(utasitas,kapcsolat). A gyakorlatban az utasítást is egy változóban szoktuk tárolni és paraméternek ezt a változót adjuk meg.

```
$utasitas = "insert into MITHALLGAT values('11111','ORA',80,4)";  
mysql_query($utasitas,$kapcsolat) or die (mysql_error());
```

Ha az utasítást nem statikusan szeretnénk megadni, hanem a programban használt változókat szeretnénk használni (host változók, ld. 5.7.3. pont), akkor ezt minden további nélkül megtehetjük a PHP-ben ismert változóhivatkozási módon, azaz a \$ jel használatával.

```
$azonosito="11112";  
$utasitas = "insert into MITHALLGAT values('$azonosito','ORA',80,4)";  
mysql_query($utasitas,$kapcsolat) or die (mysql_error());
```

¹ A felhasználó által beírt karakterek között lehetnek olyanok, amiket a fordító program nem tud értelmezni, vagy nem szöveggént, hanem az utasítás részeként próbálja értelmezni azokat és ezt lehet megelőzni a levédéssel.

6.2.3. A MySQL nyelv sajátosságai

A MySQL nyelv alapjaiban megegyezik az 5. fejezetben ismertetett SQL nyelvvel néhány elemében azonban eltér attól. Ezeket az eltéréseket tekintjük át ebben a pontban.

Az SQL nyelvénél felsorolt operátorok használhatók MySQL-ben is (ld. 5.3.4. pont). Némelyik operátor más formában is használható. Az alábbi lista nem teljes:

- NOT, vagy !
- AND, vagy &&
- OR, vagy ||
- <>, vagy !=
- a <=> operátor nem ad NULL értéket, ha az egyik argumentum NULL érték, hanem ha mindkét érték NULL érték akkor IGAZ, ha csak az egyik, akkor HAMIS (ld. 5.3.4.3. pont).

A LIKE operátor ugyanúgy működik, a helyettesítő karakterek (%,_) és jelentésük az 5.4. táblázatban láthatók.

A tárgyalt adattípusok a következő neveken találhatók meg:

- Karakter adattípusok: CHAR(n), VARCHAR(n), LONGTEXT
- Numerikus típusok: DEC vagy DECIMAL, FLOAT vagy REAL, DOUBLE PRECISION, INTEGER vagy INT, SMALLINT
- Dátum, időpont: DATE, DATETIME, TIMESTAMP

A MySQL függvényei az SQL függvényekkel összehasonlításban a 6.6. táblázatban láthatók.

6.6. táblázat: Függvények összehasonlítása SQL-ben és MySQL-ben

SQL függvények	Funkció	A függvény megjelenése MySQL-ben
<i>Csoport függvények</i>		
AVG (numerikus argumentum)	Átlag	AVG(kif)
COUNT(*) vagy COUNT(argumentum)	Darabszám	COUNT(*) vagy COUNT(kif)
MAX(argumentum)	Maximum	MAX(kif)
MIN (argumentum)	Minimum	MIN(kif)
SUM (numerikus argumentum)	Összeg	SUM(kif)
<i>Karakter függvények</i>		
CONCAT(char-1,char-2)	Szövegösszefűzés	CONCAT(char-1, char-2, ...)
INITCAP(char)	A szókezdő karaktereket nagybetűssé alakítja, de a többi karakteren nem változtat.	nincs
LENGTH(char)	Karakterek száma	LENGTH(char), CHAR_LENGTH(char)
LOWER(char) UPPER(char)	Kisbetűssé és nagybetűssé alakít	LCASE(char), LOWER(char) UCASE(char), UPPER(char)

SQL függvények	Funkció	A függvény megjelenése MySQL-ben
LTRIM(char) RTRIM(char)	Levágja a kezdő ill. a befejező szóközöket	LTRIM(char) RTRIM(char)
REPLACE(char,c1,c2) TRANSLATE(char,c1,c2)	Karakterek kicserélése	REPLACE(char1;c1;c2)
SUBSTR(char,n,m)	Az argumentum egy részletét adja eredményül	MID(char;m;n), SUBSTRING(char;m;n)
<i>Dátum és numerikus függvények</i>		
ADD_MONTHS	Hónap hozzáadása	DATE_ADD(date, INTERVAL expr unit)
LAST_DAY	Hónap utolsó napja	LAST_DAY
ROUND	Kerekítés	ROUND
SYSDATE vagy CURRENT DATE, CURRENT TIMESTAMP	Aktuális dátum ill. idő	SYSDATE(), CURDATE(), CURTIME()
TRUNC	Dátum vagy szám lerövidítése	TRUNCATE
YEAR, MONTH, DAY	Dátum megfelelő részének kivágása	YEAR, MONTH, DAY
<i>Konvertáló függvények</i>		
TO_CHAR vagy VARCHAR	Karakterre konvertálás	DATE_FORMAT(date), FORMAT(numerikus argumentum)
TO_DATE vagy DATE	Dátumra konvertálás	STR_TO_DATE(char,format)
TO_NUMBER vagy DECIMAL	Numerikus adatra konvertálás	nincs
<i>Egyéb függvények</i>		
NVL vagy VALUE, COALESCE	NULL érték átalakítása	IFNULL
USER	Aktuális felhasználó meghatározása	USER

A PHP nyelv sajátossága, hogy olyan függvényeket is használ, aminek segítségével könnyű különböző adatbázis-kezelőket, köztük a MySQL-t elérni. Ezeket a 6.7. táblázatban soroltuk fel.

6.7. táblázat: A PHP adatbázis-kezelő függvényei

Függvény	Funkció
mysql_connect(számítógépnév, felhasználónév, jelszó)	kapcsolódás a kiszolgálóhoz
mysql_error()	hiba esetén a hiba leírását adja
die()	hiba esetén a futtatás megszakítása
mysql_real_escape_string(szöveg)	a karakterláncot levédi (ld. 6.2.1. pont)
mysql_close(kapcsolatazonosító)	kapcsolat bontása
mysql_select_db(adatbázisnév, kapcsolatazonosító)	adatbázis kiválasztása

Függvény	Funkció
mysql_query(utasítás, kapcsolat)	adatkezelő utasítások kiadása
mysql_num_rows(eredményazonosító)	a lekérdezés eredményének sorainak a száma
mysql_fetch_row(eredményazonosító)	a lekérdezés eredménytáblájának következő rekordjának mezőit adja egy tömbben (ld. 6.2.4. pont),
stripslashes()	a karakterláncból eltávolítja a védő karaktereket (vissza perjel – slash: \)

Az adatleíró utasítások megegyeznek az 5.6.2. pontban tárgyaltakkal. Objektumok létrehozására CREATE, törlésére a DROP, módosítására az ALTER utasítások használhatók.

A MySQL adatbázis fő objektuma is a táblázat. Definiálhatunk feltételeket és indexeket is. Fontos eltérés, hogy idegen kulcsok definiálására nincs lehetőség. A dokumentációban szerepel az idegen kulcs megadása, és nem jelez hibát idegen kulcs definiálásakor, de a kulcsot nem hozza létre. Az adatok integritására a felhasználónak, illetve a programozónak kell figyelnie. Egyedi kulcs esetén az AUTO_INCREMENT¹ kulcsszóval egy automatikus számlálóból veszi az oszlop értékét új rekord felvételekor.

Index definiálásakor a UNIQUE kulcsszóval egyedi indexet hozhatunk létre. Az indexoszlop megadásakor szöveges oszlopoknál megadhatjuk, hogy az első hány karakter szerepeljen az indexben, ezáltal csökkenthetjük az indexek tárolási helyét.

Létrehozhatunk és megszüntethetünk nézeteket és triggereket is az 5. fejezetben leírtaknak megfelelően. Az 5. pontban leírtaktól eltérően a MySQL nyelvben ALTER VIEW utasítás is van, melynek szintaxisa megegyezik a CREATE VIEW szintaxisával és eredményeképpen a meglévő nézet helyett azonos néven egy másik nézet jön létre.

Jogosultságok kezelésére ugyanúgy a GRANT és a REVOKE utasítások használhatók.

Az 5.6.3. pontban leírt adatkezelő utasítások, a SELECT, INSERT, DELETE és UPDATE is az ott leírt szabályokkal működnek. Programba ágyazva a SELECT utasítás eredményének feldolgozásához kurzorra van szükség. Ezt a következő pontban részletezzük.

6.2.4. Kurzor használata PHP MySQL-ben

A kurzor használatára általános példát láthattunk az 5.7.4. pontban. Nézzük most meg, hogy hogyan kezelhetjük egy lekérdezés eredményrekordjait PHP-ben. A lekérdezést az sql_query() függvénnyel kell kiadnunk. A függvény visszatérési értéke egy eredményazonosító lesz. Ezt felhasználva különböző függvények segítségével adatokat nyerhetünk a lekérdezés eredményéről, illetve lekérhetjük az egyes rekordokat. Például az eredménysorok számát a mysql_num_rows() függvénnyel kaphatjuk meg.

Példa

1. Kérdezzük le az elsős hallgatók adatait ABC sorrendben és írjuk ki, hogy hány elsős hallgató van.

```
...
$utasitas = "select * from HALLGATO where evf = 1 order by vnev";
$eredmeny = mysql_query($utasitas);
$sorok_szama = mysql_num_rows($eredmeny);
```

¹ Ez megfelel annak, amikor a mező értékét egy sorszámgenerátor típusú (Sequence) objektumból vesszük (ld. 5.3.6. pont)


```
print "<p>Jelenleg $sorok_szama darab elsős hallgató van az adatbázisban</p>"
...
```

Az eredményazonosítóval megadott eredmény sorait a *mysql_fetch_row()* függvénnyel érhetjük el. Ez a függvény mindig az eredménytábla következő rekordjának mezőit adja eredményül egy tömbben.

Példa

1. Írassuk ki az elsős hallgatók vezetéki és keresztnévét egy táblázatban.

```
...
$utasitas = "select * from HALLGATO where evf = 1 order by vnev";
$eredmeny = mysql_query($utasitas);
print "<table border = \"1\">";1
while ($egy_sor = mysql_fetch_row($eredmeny)) {
    print "<tr>";
    foreach ($egy_sor as $mezo){
        print "<td>".stripslashes($mezo)."</td>";
    }
    print "</tr>"
}
print "</table>"
...
```

6.3. Ellenőrző kérdések

1. Access adatbázis-kezelőben hozzunk létre egy új adatbázist és mentjük el EGYETEM néven.
2. Hozzuk létre a 4.2. alfejezetben definiált táblázatokat tervező nézetben. A mezők minden tulajdonsága feleljen meg az ott megadottaknak. Módosítsuk mindhárom tábla indexeit is úgy, hogy az indexeknek beszélő nevük legyen és minden indexet hozzunk létre, amire szükség lehet.
3. Állítsuk be a 4.2. alfejezetben megfogalmazott feltételeket a táblák mezőire érvényesítési szabályként. Mindegyikhez adjunk meg érvényesítési szöveget is!
4. Állítsunk be beviteli maszkot a mindhárom táblában az azonosítókra úgy, hogy a hallgatói azonosító kötelezően 5 számjegyből álljon, a tantárgyi azonosító pedig kötelezően betűvel kezdődjön és 4 karakterből álljon.
5. Állítsuk be a három tábla közötti kapcsolatokat a kaszkád módosítást és törlést megtiltva!
6. Vigyünk be néhány adatot mindhárom táblába!
7. Rendezzük a TANTÁRGY tábla adatait az oktató neve, azon belül a tantárgy megnevezése szerint ABC sorrendbe!
8. Állítsunk be szűrőt a MITHALLGAT táblára úgy, hogy csak azok a rekordok látszódnak, ahol a gyakorlati pontszám magasabb 50-nél!
9. Hozzuk létre az alábbi lekérdezéseket:

¹ A kódban szereplő \ karaktereket az utánuk következő karakter levédésére szolgálnak, azaz a jelenlegi példánál azt jelentik, hogy idézőjel még nem az utasítás végét jelenti, hanem azt is ki kell írnia az utasításnak.

- a) Kérdezzük le B betűvel kezdődő nevű hallgatók adatait a HALLGATÓ táblából!
 - b) Kérdezzük le a hallgatók vezeték és keresztnévét, évfolyamát és osztályzatait név és azon belül osztályzat szerinti rendezésben!
 - c) Kérdezzük le oktatónként, hogy hány hallgatót tanítanak!
 - d) Kérdezzük le a paraméteresen megadott tantárgy hallgatói névsorát!
 - e) Kérdezzük le az egyes hallgatók átlagosztályzatát. Ha van olyan hallgató, akinek még nincsen egyetlen eredménye sem, az is jelenjen meg a lekérdezésben! Az átlagok legyenek 2 tizedesre kerekítve!
 - f) Kérdezzük le kereszttáblában születési évenként és tantárgyanként csoportosítva a legnagyobb pontszámokat!
 - g) Kérdezzük le paraméteresen megadott hallgató átlagánál jobb eredményű hallgatók nevét és átlagosztályzatát!
10. Hozzunk létre gyorsvarázslóval űrlapot a TANTÁRGY táblából!
11. Módosítsuk az előző pontban létrehozott űrlapot a következőkkel:
- a) Az űrlapfejbe írjuk be, hogy TANTÁRGY!
 - b) Az űrlaplábba illesszük be az aktuális dátumot!
 - c) Az óraszámot listából lehessen kiválasztani, a választható értékek 1, 2, 3, 4, 5, 6 legyenek!
12. Készítsünk varázslóval jelentést, ami tartalmazza a hallgatók nevét, születési dátumát, évfolyamát, a tárgyakat amiket tanulnak és az eredményeiket: pontszámot és osztályzatot is. Az adatok születési évenként, azon belül hallgatónként legyenek csoportosítva. A gyakorlati pontszámokból és az osztályzatokból számítsunk átlagot!
13. Módosítsuk az előző pontban létrehozott jelentést a következőkkel:
- a) Egy hallgató adatain belül ne legyen oldaltörés!
 - b) Az átlagok mindenhol 2 tizedesre legyenek kerekítve!
 - c) Az összesítéseknél a címkén az Avg feliratot cseréljük ki Átlag-ra!
14. Mik a Php utasítások nyitó és záró elemei?
15. Hogyan nyitható meg egy adatbázis PHP-ben?
16. Milyen feladatokat látnak el a programba beépíthető MySQL függvények?
17. Melyek a legfontosabb különbségek az 5. fejezetben tárgyalt SQL és az ebben a fejezetben tárgyalt MySQL nyelv között?
18. Hogyan működik a kurzor PHP-MySQL-ben?

7. KATALÓGUS, ADATSZÓTÁR

7.1. Katalógus táblázatok

Az adatbázisnak az adatok mellett tartalmaznia kell az adatok definícióit, kapcsolatait, tárolási és használati módjuknak a leírását. Az adatbázisnak ezt az önálló részét katalógusnak, vagy adatszótárnak (Data Dictionary, Data Directory) nevezik. A relációs modellre épülő adatbázisokban a katalógus maga is több táblázat és nézet összessége, melyeket összefoglalóan *katalógus relációknak* nevezünk. Ezekkel ugyanúgy dolgozhatunk, mint az adatokat tartalmazó táblázatokkal és az ezekből előállított nézetekkel.

A katalógus relációk tartalmazzák a rendszer minden objektumának, azok részeinek, egymás közti kapcsolatainak és elérhetőségüknek a leírását. Amennyiben valamely objektum felépítésében, tulajdonságaiban bármilyen változás következik be, a rendszer automatikusan módosítja a megváltozott objektum jellemzőit nyilvántartó katalógus relációkat. Ha például egy új táblázatot hozunk létre, akkor annak jellemzői (neve, létrehozója, létrehozásának ideje, fizikai tárolási helye, stb.) automatikusan bekerülnek a táblázatok általános jellemzőit nyilvántartó katalógus relációba, az új táblázat oszlopainak jellemzői pedig az adatbázis összes oszlopát nyilvántartó relációba.

A katalógus az adatbázisnak nélkülözhetetlen és leggyakrabban használt része. Amikor az adatbázisban valamilyen műveletet akarunk végrehajtani, akkor az SQL utasítás-értelmező a katalógusban nézi meg, hogy az utasításban szereplő objektumok és azok elemei léteznek-e és hogy az utasítás kiadója jogosult-e ezeken a kívánt műveletek elvégzésére. Amennyiben az eredmény pozitív, akkor az SQL optimalizáló a katalógusban lévő statisztikai adatok, az adatok tényleges eloszlása és fizikai tárolási jellemzői alapján meghatározza a kért adatokhoz vezető optimális elérési utat. Interaktív feldolgozásnál ezek a műveletek minden egyes utasítás végrehajtása előtt lefutnak. Programba beépített statikus SQL utasításoknál (ld. 5.7.1. pont) ez az elemzés csak egyszer, a program előkészítésekor történik meg és az elérési terven (*PLAN*, ld. 5.7.2. pont) keresztül hozzákapszólódik a programhoz. Futáskor már az így előre meghatározott elérési stratégia hajtódik végre.

A katalógus hatékony használatát a gyakran keresett katalógus adatokra felépített indexek segítik.

Az, hogy a katalógusban milyen relációkban, milyen formában, milyen információk vannak, az erősen rendszerfüggő. Ugyancsak rendszerfüggő, hogy az egyes felhasználók a teljes katalógusból milyen információkhoz férhetnek hozzá. Az IBM UDB/DB2 rendszerében az adatbázis (DATABASE) felhasználói lényegében a katalógus minden táblázatát láthatják. Ezeknek a tulajdonosa a *SYSIBM* és nevük mindig *SYS*-szel kezdődik. Az utánuk következő névrész utal arra, hogy miről tartalmaz információt az adott katalógus reláció. Az ORACLE-ben a felhasználók csak a katalógus táblázatokra épült nézetek megtekintésére jogosultak. Ezekre minden felhasználó ugyanazokkal a nevekkkel, un. *nyilvános (PUBLIC) szinonimákkal* hivatkozhat, de mindenki csak a saját tulajdonában (sémájában) lévő objektumokról, vagy az általa használható objektumokról kaphat felvilágosítást. Az előbbiek nevei mind *USER_*, az utóbbiak mind *ALL_*-lal kezdődnek. Léteznek még a teljes rendszerinformációt szolgáltató *DBA_* prefixszel kezdődő katalógus relációk is, melyeket csupán az adatbázis felügyelő használhat. Az ACCESS-ben a katalógus lényegében el van rejtve a felhasználók elől.

Nagy rendszerek katalógusa több száz relációból is állhat. A 7.1. táblázatban felsoroltuk a két legnagyobb adatbázis-kezelő rendszer, az ORACLE és az IBM/UDB/DB2 legfontosabb katalógus relációit.

7.1. táblázat: A legfontosabb katalógus relációk ORACLE-ben és UDB/DB2-ben. Az ORACLE szinonimanév előtt *USER_* vagy *ALL_* prefix áll, az IBM táblázatnév előtt pedig *SYSIBM.SYS*

Oracle	IBM UDB/DB2	Tartalom
CATALOG	TABLES,SYNONYMS TRIGGERS	Táblázatok, nézetek, szekvenciák
CONSTRAINTS	CHECKS, RELS, FOREIGNKEYS	A táblázatokra és azok részeire definiált korlátozások, feltételek
IND_COLUMNS	KEYS	Indexek oszlopai
INDEXES	INDEXES	Indexek
OBJECTS	TABLES,SYNONYMS TRIGGERS	Objektumok
	PLAN	Programokhoz tartozó elérési tervek
SYNONYMS	SYNONYMS	Szinonimák
TAB_COL_STATISTICS	COLSTATS	Oszlop statisztikák
TAB_COLS	COLUMNS	Táblázatok, nézetek oszlopai
TABLES	TABLES	Táblázatok (nézetek)
TAB_PRIVS, TAB_PRIVS_MADE	DBAUTH, TABAUTH	Objektumokra vonatkozó jogosultságok
TRIGGERS	TRIGGERS	Triggerek
USERS	USERAUTH	Felhasználók
VIEWS	VIEWS, VIEWDEPS	Nézetek

A katalógus relációkat általában csak lekérdezni lehet. Nem lehet

- katalógus-relációt megszüntetni,
- felépítését módosítani,
- explicit elzárni mások elől,
- közvetlen utasítással beírni, törölni, módosítani sorokat. (Ez alól kivételek a statisztikai adatokat tartalmazó oszlopok ld. 7.3. alfejezet).

A katalógus lekérdezésével információkat nyerhetünk

- az adatbázis objektumairól, illetve azok részeiről,
- az egyes objektumokhoz való hozzáférési jogokról,
- a különböző objektumok közti összefüggésekről,
- az adatok fizikai elhelyezéséről,
- amennyiben statisztikát készítettünk, akkor a különböző objektumok elemeinek számáról és értékek szerinti eloszlásáról.

Az alábbiakban vázlatosan ismertetjük a legfontosabb katalógus relációk főbb jellemzőit. Megjegyezzük, hogy nem mindegyik rendszerben létezik az összes itt felsorolt katalógus táblázat/nézet, vagy tartalmuk az itt ismertetettől némileg eltérhet.

Az egyszerűbb nyelvhasználat érdekében sokszor az egyes katalógus relációkat is csak katalógusnak nevezzük. Így például az indexek jellemzőit tartalmazó katalógus reláció helyett röviden csak index katalógust mondunk.

7.1.1. Katalógus relációk az objektumokról és összetevőikről

Minden katalógusban megtaláljuk a legfontosabb objektum típusok önálló táblázatokban összefoglalt jellemzőit. A felhasználó által elérhető katalógus reláció vagy közvetlenül ez a táblázat vagy egy erre épülő nézet. Így önálló katalógus reláció van a

- táblázatokról,
- nézetekről,
- táblázatok/nézetek minden egyes oszlopáról,
- indexekről és azok komponenseiről,
- szinonimákról,
- szekvenciákról,
- triggerekről,
- az adatbázis fizikai és logikai egységeiről,
- a lefordított programokhoz hozzárendelt adatelérési módokról.

Ezek a relációk tartalmazzák a bennük foglalt objektumok minden meghatározó jellemzőjét.

A fentieken kívül egyes rendszerekben léteznek olyan katalógus relációk is, amelyek általános áttekintés céljából összefoglalják az adatbázis legfontosabb objektumtípusairól azok nevét, típusát, tulajdonosát, létrehozásának és utolsó módosításának az idejét.

7.1.1.1. Táblázat katalógus

Ez a katalógus az adatbázis minden egyes táblázatáról (esetleg nézetéről is, beleértve a katalógus táblázatokat/nézeteket is) tartalmaz egy sort. A táblázat katalógus legfontosabb oszlopai:

- a táblázat (nézet) neve,
- a táblázat (nézet) tulajdonosa (sémája),
- típusa (táblázat vagy nézet),
- létrehozásának és utolsó módosításának ideje (ez az ORACLE-ben nem az egyedi objektum típusok katalógusában van, hanem az általános objektum katalógusban)
- oszlopok száma,
- hány táblázattól függ,
- hány táblázat függ tőle,
- csak táblázatoknál
 - a sorok száma,
 - hány lapot foglal el,
 - fizikai tárolási helye,
 - átlagos rekordhossz,
 - az utolsó statisztika elkészítésének ideje.

7.1.1.2. Oszlop katalógus

Ez a katalógus az adatbázis minden egyes táblázatának és nézetének minden egyes oszlopáról tartalmaz egy sort. A katalógus reláció legfontosabb oszlopai:

- Az oszlopot tartalmazó táblázat
 - neve,
 - tulajdonosa,
- az oszlop
 - neve,
 - sorszáma a táblázatban/nézetben,
 - a benne tárolható adattípus,
 - a benne tárolható adat (maximális) hossza,
 - tartalmazhat-e *NULL* értéket,
 - alapértelmezett érték,
 - átlagos oszlophossz,

- különböző értékek száma,
- leggyakoribb értékek,
- az utolsó statisztika elkészítésének ideje.

7.1.1.3. *Index és indexkomponens katalógus*

Az *index katalógus* az adatbázis minden egyes indexéről tartalmaz egy sort. A katalógus reláció legfontosabb oszlopai:

- Az index
 - neve,
 - tulajdonosa (sémája),
 - oszlopainak száma,
 - típusa,
- az indexet tartalmazó táblázat
 - neve,
 - tulajdonosa (sémája),
- létrehozásának és utolsó módosításának ideje (ez az ORACLE-ben nem az egyedi objektum típusok katalógusában van, hanem az általános objektum katalógusban),
- egyedi vagy duplikált,
- sűrűsödő-e és ha igen, a sűrűsödés mértéke,
- B+ fa szintjeinek és az index-lapoknak a száma,
- tárolóhely foglalása,
- a különböző indexértékek száma,
- az utolsó statisztika elkészítésének ideje.

Az *indexkomponens katalógus* az adatbázis minden egyes indexének minden egyes összetevőjéről (oszlopáról) tartalmaz egy sort. A katalógus reláció legfontosabb oszlopai:

- Az index
 - neve,
 - tulajdonosa (sémája),
- az index oszlopának
 - neve,
 - sorrendje az indexen belül,
 - növekvő vagy csökkenő az oszlopon belül a rendezettség.

7.1.1.4. *Objektumokkal kapcsolatos egyéb katalógusok*

További katalógusok tartalmazhatják

- a szinonima szótárt, mely lehetővé teszi, hogy az azt definiáló felhasználó az általa szinonimaként meghatározott nevekkal hivatkozzon a rendszer különböző objektumaira,
- az adatbázis feloszlását fizikai és logikai részekre,
- fizikai adathordozókhoz való hozzárendelést és helyfoglalást,
- a lefordított és tárolt felhasználói programoknak az adatbázis elérés módját meghatározó részeinek leírását,
- az egyes oszlopokra fennálló különleges feltételek leírását (pl. milyen értékek tárolhatók bennük).

7.1.2. Katalógus a hozzáférési jogokról

Több felhasználó által hozzáférhető adatbázisoknál elengedhetetlen, hogy pontosan meg legyen határozva, ki mire jogosult. Ki hozhat létre vagy módosíthat táblázatokat, ki olvashat, törölhet vagy módosíthat adatokat egy már meglévő táblázatban, ki építhet indexeket vagy hivatkozhat idegen kulccsal egy táblázatra, ki futtathat az adatbázissal dolgozó programokat, stb.

Az általános alapelv az, hogy aki létrehozta az objektumot vagy akinek a nevében azt az adatbázis felügyelő létrehozta (az objektum tulajdonosa, akinek a sémájában van), az bármit csinálhat vele, és ezeket a jogokat a *GRANT* és a *REVOKE* utasítással tovább is adhatja másoknak, illetve vissza is vonhatja tőlük (ld. 5.6.2.4. pont). Éppen ezért a hozzáférési jogokat leíró katalógusok mindig tartalmazzák az adott objektum létrehozójának és a használatára jogosultaknak az azonosítóját. Ebből állapítja meg működés közben az adatbázis-kezelő rendszer, hogy az utasítás kiadója végrehajthatja-e a kért műveletet.

A jogosultságokat ki lehet adni egyes felhasználókra vagy a felhasználók meghatározott csoportjaira is. A legtöbb rendszerben van egy különleges felhasználói csoport, amelyiknek a neve *PUBLIC*. Ha ennek adunk engedélyt, akkor az adott objektumon így engedélyezett műveletet az adatbázis minden felhasználója elvégezheti.

7.1.2.1. Általános rendszer jogosultságok katalógusa

Ezek a katalógus relációk minden felhasználóra tartalmazzák sok más egyéb mellett

- az adatbázishoz való hozzákapsolódásra jogosultak adatait,
- milyen objektumokat hol hozhatnak létre,
- a maximálisan felhasználható tároló helyet,
- van-e adatbázis felügyelői jogosultságuk.

7.1.2.2. Katalógus az objektumokra való jogosultságokról

Azokat a jogosultságokat sorolja fel, amelyekkel az egyes felhasználók rendelkeznek, hogy az adott objektummal dolgozhassanak. Az, hogy egy katalógus reláció csak egy objektum típusra vonatkozó jogosultságokat tartalmaz-e, vagy külön katalógusban vannak az adott és a kapott jogosultságok, az rendszer függő. Táblázatoknál/nézeteknél a legfontosabb jogosultságokat a 7.2. táblázatban foglaltuk össze. Ha egy felhasználó ugyanabban a táblázatban többféle jogosultságot is kapott, akkor azok mindegyike önálló sorként szerepel a katalógusban.

7.2. táblázat: Legfontosabb jogosultságok táblázatokon/nézeteken elvégezhető műveletekre.

Jogosultság	Elvégezhető művelet
ALL	minden művelet
ALTER	táblázat definíciójának módosítása
DELETE	Sor(ok) törlése táblázatból, nézetből
INDEX	index létrehozása táblázaton
INSERT	adatbevitel táblázatba, nézetbe
REFERENCES	idegen kulccsal való hivatkozás táblázatra
SELECT	olvasás táblázatból, nézetből
UPDATE	adatok módosítása táblázatban, nézetben

Az objektum jogosultságokat leíró katalógusok legfontosabb oszlopai a következők:

- a jogosultságot adó azonosítója,
- a jogosultságot kapó azonosítója,
- az objektum típusa,
- az objektum neve és tulajdonosa,
- mire vonatkozik a jogosultság,
- tovább adható-e a jogosultság,
- a jogosultság megadásának ideje.

Bizonyos műveleteknél (pl. *UPDATE*) a jogosultság tovább is finomítható és az egész táblázat/nézet helyett annak egyes oszlopaire korlátozható. Akkor ez így szerepel a katalógusban.

7.1.2.3. Katalógus a programok használati jogáról

Ha az adatbázis-kezelő utasításokat nem párbeszédés üzemmódban adták ki, hanem statikusan beépítik az alkalmazási programba, akkor a futtatáshoz lefordított program önálló modulként (*csomag, package*) tartalmazza az adatbázis-elérési utasításokat is (ld. 5.7. alfejezet). Amennyiben ezt a programot bárki futtathatná, ezeket az elérési utasításokat lemásolhatná, hozzákapcsolhatná a saját programjához, akkor az adatbázishoz való hozzáférést nem lehetne ellenőrizni. Éppen ezért külön katalógus táblázat(ok) tartalmazzák ezeknek az adatbázis-elérési moduloknak a jellemzőit és azt, hogy kik jogosultak ezekkel dolgozni, ezeket futtatni. Miután ilyenkor az ellenőrzés egyszer, a program elkészítésekor történik, ennek időpontja fontos adata ezeknek a katalógusoknak. Ezt az adatbázis-kezelő rendszer a program futása előtt összehasonlítja az elérni kívánt objektum utolsó változtatási időpontjával. Amennyiben ez későbbi, akkor újra elvégzi az ellenőrzést. Ha a változások nem akadályozzák, akkor a program – most már az új elkészítési/ellenőrzési dátummal – lefut, különben hibajelzéssel leáll.

7.1.3. Katalógus a függőségekről

Ezek a katalógus táblázatok az idegen kulcs kapcsolatokat, az adatbázis nézeteknek, programoknak, elérési utaknak és terveknek a táblázatoktól való függőségét tartalmazzák. Ezek ismerete fontos, mert például egy táblázat törlésekor (*DROP*) automatikusan megszűnik az összes ráépülő nézet is. Így azok, akik eddig mit sem sejtettek a táblázatról, mert csupán egy ráépülő nézettel dolgoztak, nem tudnak tovább dolgozni. Ugyancsak működésképtelenek lesznek azok a programok is, amelyek ezzel a törölt táblázattal vagy nézettel kívánnának dolgozni. Ezeket a programokat a katalógus alapján jobb megkeresni és a hibát még a futás előtt kijavítani.

Ha egy nézet több táblázatra épül, vagy egy program több táblázattal dolgozik, akkor a függőséget mindegyikre egy külön sor jelzi a katalógusban.

A katalógus relációk legfontosabb oszlopai:

- az alapobjektum neve, típusa és létrehozója,
- a függő objektum neve, típusa (nézet, program, adatbázis-elérés) és létrehozója,
- a függőség létrehozásának ideje,
- idegen kulcsnál a kulcs kapcsolat jellemzői (korlátozott, kaszkád, *NULL értékre állítás*).

Gyakorlati tanács, hogy ha egy objektumot meg akarunk szüntetni, akkor előbb nézzük meg a katalógusban, hogy ez milyen további hatásokkal jár. Általában olyan táblázatot nem is tudunk megszüntetni, amelyikre egy másik táblázat idegen kulccsal hivatkozik.

7.2. Katalógus osztott adatbázisokban

Az osztott adatbázisok katalógusának a centralizált rendszer katalógusa által nyújtott szolgáltatásokon kívül információt kell adni arról is, hogy

- az egyes rekordokat fizikailag hol, melyik helyen tároljuk,
- ha több helyről kell egy logikai rekordot összeszedni, akkor annak összetevői hol találhatóak,
- milyen adatokat tárol többszörösen a rendszer, a duplikátumok hol találhatóak és hogyan kell konzisztenciájukat biztosítani.

Ezek a megfelelően definiált új, az osztott rendszerek specialitásait és igényeit is figyelembevevő katalógus táblázatokból megkaphatók. Sokkal nagyobb problémát jelent a gyakorlatban az, hogy hol tároljuk magát a katalógust. Alapjában véve a következő lehetőségek közül választhatunk:

1. Központi, egy helyen tárolt, centralizált katalógus.
2. A teljes katalógust többszörözzük. Minden önálló adatbázis helynek saját, teljes példánya van.
3. Particionált katalógus. Minden önálló adatbázis hely csak az ottani adatbázisról vezet egy helyi katalógust. A teljes katalógus az egyes helyeken lévő katalógusok uniója.
4. Minden hely vezeti a saját helyi katalógusát és ezen kívül egy helyen egy központi nyilvántartás is van (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. megoldásnál ennek előnyeit részben elveszítjük. A központi katalógus miatt az egyes helyeken történő feldolgozások ki vannak szolgáltatva a központnak. A központi katalógus-szolgáltatás bármilyen hibája minden adatbázis helyi működését befolyásolhatja, esetleg működésképtelenné is teheti.

A 2. módszer gyorsan módosuló rendszereknél szinte kivihetetlen, hacsak nem engedjük meg, hogy a katalógusok adatai bizonyos ideig inkonzisztensek legyenek. Az összes katalógus példány egyidejű, azonnali aktualizálása olyan bonyodalmas eljárásokhoz vezethet, hogy a tényleges munka megbénulhat.

A 3. módszer csak akkor előnyös, ha a műveletek csaknem kizárólag a helyi adatbázisra korlátozódnak. Ha máshol tárolt adatokra van szükségünk, akkor ezek megtalálásához átlagosan a többi adatbázis helyi katalógusainak a felét át kell vizsgálnunk, kedvezőtlen esetben (márpedig ezek valószínűsége Murphy törvénye alapján jóval nagyobb), akár minden helyi katalógust át kell néznünk.

A 4. módszer kiküszöböli ezt a hátrányt, de ismét behozza, - bár kisebb mértékben – a központi helytől való függőséget.

A gyakorlatban az 5. módszer kecsegtet a legjobb eredménnyel, ha biztosítjuk az alábbi szabályok betartását:

- Minden felhasználónak minden objektum névre (ld. 5.3.2. pont), amellyel dolgozhat létrehozunk egy, az egész rendszerben érvényes *saját szinonimát*, mely az alábbiakból áll:
 - az objektum létrehozójának az azonosítója,
 - létrehozásának helye (ahol a *CREATE* utasítás lefutott),
 - minden minősítés nélküli neve (Pl. *HALLGATO* vagy *TANTARGY*),
 - első tárolási helye.

Ez a név egyértelmű a rendszerben és sohasem változik meg, akárhol lesz később tárolva az objektum.

A felhasználók általában a minősítés nélküli, vagy a létrehozó azonosítójával minősített névvel hivatkoznak egy objektumra. Ekkor a rendszer automatikusan a helyi adatbázisban keresi. Más helyen lévő objektumra az előbb definiált szinonimával hivatkoznak. Hogy így is megtalálja az adatbázis-kezelő rendszer a keresett objektumot, minden helyi katalógusban tároljuk

- az egész rendszerre érvényes szinonimák adatait,
- az összes *ott tárolt* objektum jellemzőit,
- az összes *ott létrehozott* objektum *pillanatnyi* tárolási helyét.

Bármely objektumra a minősítés nélküli névvel illetve azt minősítve a létrehozója azonosítójával és a hellyel hivatkozva (vagy ha ez nincs, akkor alapértelmezésként az adott hellyel és felhasználóval kiegészítve), a kérdéses hely helyi katalógusából minden információt megkaphatunk, ha az objektum ott van tárolva.

Ugyanezt legfőljebb két nem helyi lekérdezéssel megkaphatjuk a nem helyben tárolt objektumokról is, ha a szinonimával kérdezzük rá. Ha az objektum a helyi adatbázisban van, akkor a helyi katalógusból olvassuk ki az információkat. Ha nincs ott, akkor a helyi szinonima táblázatból megállapítjuk, hol hozták létre. Annak helyi katalógusából – ha még ott van az objektum – rögtön megkapunk mindent, vagy megtudjuk, hol van pillanatnyilag. Így ehhez az adatbázishoz kapcsolódva a második távoli lekérdezéssel jutunk hozzá a keresett információhoz.

A katalógus aktualizálása itt abból áll, hogy

- új objektum létrehozásakor
 - az objektum jellemzői bekerülnek a helyi katalógusba,
 - a helyi katalógusok szinonima táblázatába bekerül egy új bejegyzés,
- objektum módosulásakor
 - ha az objektum tárolási helye nem változik, akkor aktualizálódik a helyi katalógus,
 - ha a helye változik, akkor az adatok mellett az objektum jellemzői átkerülnek az új hely katalógusába és a létrehozás helyén módosul a katalógusban a pillanatnyi tárolás helye.

Ha például *SQL00*-nak a Budapesten létrehozott *HALLGATO* táblázatára *SQL30* a *HALLGATO00* szinonimát hozta létre, akkor ezzel *SQL30* bárhol dolgozhat a *HALLGATO00* névvel. Ha a táblázatot időközben áthelyezik Budapestről a debreceni adatbázisba, akkor

- a létrehozás helyén, a budapesti katalógusban a pillanatnyi tárolás helye Budapestről Debrecenre módosul,
- *SQL00.HALLGATO* katalógus bejegyzései törlődnek a budapesti katalógusból,
- *SQL00.HALLGATO* jellemzői bekerülnek a debreceni katalógusba.

7.3. Statisztikák

Az adatbázis hatékony működtetéséhez szükséges optimális adatelérési utak meghatározásához az SQL optimalizáló a katalógus relációkból szerzi meg az információkat. Ehhez nem elég az adatok szerkezete, szükséges a tárolt adatok számának és eloszlásának az ismerete is. Nem mindegy, hogy egy csupán 100 és egy 10 millió rekordot tartalmazó táblázat join-jánál az első sorain megyünk egyenként végig, hogy a másodikból index alapján meghatározzuk az összetartozó sorokat, vagy a második 10 millió rekordján. Ugyancsak nem mindegy, hogy ha duplikált indexnél egy értékhez 100 különböző rekord tartozik, akkor azokat 100 fizikai olvasással 100 különböző lapról olvassuk be, vagy az index szerint sűrűsödő rekordokból egy, legfőljebb két olvasással egy vagy két lapról. Ezért az objektum katalógusok a bennük fizikailag tárolt adatok számáról és eloszlásáról is tartalmaznak információkat. (Egyes rendszereknél külön katalógus relációk is vannak ezek tárolására).

Elvileg ezeknek a statisztikáknak állandóan naprakésznek kellene lenni. Ez azt jelentené, hogy minden adatváltoztatásnál ezeket módosítani kellene. Ez akkora terhet róna a rendszerre, hogy gyakorlatilag csak a saját adminisztrációjával foglalkozna. Éppen ezért a katalógus statisztikák aktualizálása csak külön kérésre, vagy egy erre szolgáló SQL utasítás (pl. *ANALYZE*, *ORACLE*) vagy egy speciális szolgáltató program (*RUNSTATS*, *UDB/DB2*) végrehajtásakor történik meg. Ebben szabályozhatjuk azt is, hogy milyen objektumokról, milyen mélységű statisztikát kérünk. Mivel az optimális elérési út általában nem különösebben érzékeny arra, hogy egy táblázat 10 vagy 13 millió adatot tartalmaz, vagy hogy egy indexben 1000 vagy 1500 különböző érték van, ezeket a statisztikákat elegendő csak időnként, pl. minden éjjel egyszer, amikor kisebb on-line terhelés van lefuttatni. Bár a katalógust SQL utasításokkal nem lehet módosítani, ezeket a statisztikai adatokat az adatbázis felügyelő manuálisan egyenként is beviheti, tetszés szerint megváltoztathatja.

A katalógusban tárolt legfontosabb statisztikai adatok az alábbiak:

- az utolsó statisztika időpontja,
- táblázatban
 - a sorok száma,
 - a használt lapok száma,
- indexnél
 - index-lapok száma,
 - B+ fa szintjeinek száma,
 - sűrűsödés mértéke,
 - különböző index-értékek száma,
- oszlopoknál a leggyakoribb oszlopértékek és ezek gyakorisága.

7.4. Ellenőrző kérdések

1. Mikor lehetünk biztosak abban, hogy a *CREATE* utasításban leírtak megfelelnek az adatbázis aktuális állapotának?
2. Hogyan tudjuk megállapítani, hogy melyek azok az objektumok, amelyeket létrehozásuk óta még egyszer sem változtattunk meg?
3. Meg tudjuk-e állapítani, hogy egy táblázat hány változtatáson ment keresztül?
4. Hogyan tudjuk egy lekérdezéssel megkapni egy index és összes indexoszlopa jellemzőit?
5. Megkapjuk-e minden jogosultságunkat, ha a kapott jogosultságok katalógusában a jogosultságot kapó (*GRANTEE*) oszlopára úgy kérdezzük rá, hogy

where grantee = user vagy where grantee ='saját azonosítónk'

6. Hogyan kell kiegészíteni az előző lekérdezést, hogy a helyes választ megkapjuk?
7. Milyen katalógus relációkban történnek változások, ha egy táblázatot megszüntetünk?
8. Milyen katalógus relációkban történnek változások, ha egy táblázat minden sorát töröljük?

8. GYAKORLATI PROBLÉMÁK ÉS MEGOLDÁSUK

Egy adatbázisban sokan dolgozhatnak egyszerre, egymással párhuzamosan. A hatékony munkához a felhasználóknak az alkalmazásaikat úgy kell elkészíteniük, hogy azok

- a lehető leggyorsabb módon érjék el és dolgozzák fel az adatokat, optimalizálják a saját alkalmazásukat. Ezt hatékony programok tervezésével (melynek témája kívül esik könyvünk keretein) és az adatszerkezet és az adatelérési módok megfelelő megválasztásával (ld. 2.5. alfejezet) érhetjük el.
- minél kevésbé zavarják, hátráltassák mások munkáját az adatbázisban (és természetesen, viszonzásul, a többiek se akadályozzák az ő munkájukat). Ez időnként csak saját munkájuk hatékonyságának rovására valósítható meg. Mivel nem mindenkitől várható el ilyen erős közösségi elkötelezettség, a jó adatbázis-kezelő rendszerek automatikusan rákényszerítik a felhasználókat a közös munka alapszabályainak betartására.

Amíg mindenki csak olvasni akarná az adatbázis adatait, addig nincs probléma. Ha többen nézik ugyanazt, abból még nem lesz semmi gond. Amint azonban bármelyik felhasználó változtatni kíván a többiek által is használt adatokon, problémák merülnek fel. Mit láthatnak a többiek a változtatásaiból? Mi legyen, ha ők is módosítani akarnak ugyanazokban a rekordokban? Az eredeti, vagy a módosított adatokkal dolgozzanak? Egyidejű változtatási kérelmeknél melyiknek legyen elsőbbsége? Ha a változtatás egy táblázat több sorát is érinti, akkor egy másik, ezzel egyidejűleg dolgozó felhasználó a sorok egy részénél még a régi, más részükénél már az éppen megváltoztatott adatokat kapná meg. Ezáltal nagy valószínűséggel téves következtetéseket vonna le. Például az egyik utasítás minden dolgozó fizetését növeli 10000 forinttal (`UPDATE ALKALMAZOTTAK SET FIZETES = FIZETES + 10000`), míg egy másik ugyanakkor fizetési statisztikát készítene osztályonként ugyanezen adatokról (`SELECT OSZTALY,AVG(FIZETES) FROM ALKALMAZOTTAK GROUP BY OSZTALY`), akkor attól függően, hogy a csoportátlag képzés pillanatában a csoport hány tagjánál történt már meg a fizetésemelés, más és más eredményt kapnánk. Ez nem engedhető meg. Ezért a jó adatbázis-kezelő rendszerek automatikus védelmet nyújtanak az egyidejű hozzáférésekből származó inkonzisztenciák ellen.

A 8.2. alfejezetben bemutatjuk azokat a problémákat, amelyek több-felhasználós adatbázisokban a párhuzamos hozzáférés miatt felléphetnek. A 8.3. alfejezetben ismertetjük ezek megoldását a *zárak* használatával. A 8.4. alfejezetben fontossága miatt külön tárgyaljuk a párhuzamos feldolgozás talán legveszélyesebb hibaforrását, a *patthelyzetet* és a patthelyzetek feloldását. A 8.5. alfejezetben néhány saját gyakorlatunkban előfordult példán mutatjuk be, miként lehet utólag elemezve akár triviális butaságokat is elkövetni, de bonyolult hibákat is kijavítani.

8.1. Tranzakciók

A párhuzamos munkák legfontosabb eleme a *tranzakció*. Sok rendszerben ezt az adatbázis használatával kapcsolatban logikai munkaegységnek (Logical Unit Of Work, LOW) is nevezik. Ez olyan egymást követő adatbázis-műveletek sorozata, melyek vagy teljes egészükben mind megvalósulnak, vagy egy elemük sem hajtódik végre, illetve nem véglegesítődik. Ilyen utasítássorozatokra mutattunk be példákat az 5.6.4. pontban. Kívülről, a *feldolgozás egésze szempontjából* nézve a tranzakció egy *atomi művelet*, nem bontható

további részekre.¹ A tranzakció az adatbázist az egyik konzisztens állapotából átviszi egy másik konzisztens állapotába.

Egy tranzakció minden utasításának az eredményét véglegesíthetjük az adatbázisban a COMMIT, hatástalaníthatjuk, azaz visszaállíthatjuk az adatbázist a tranzakció kezdetén fennálló állapotba a ROLLBACK utasítással (ld. 5.6.4. pont).²

A párhuzamos feldolgozás koordinálásának a lényege az, hogy a tranzakciók egymástól függetlenül fussanak, ne dolgozhassanak ugyanazokkal a rekordokkal, ha az nem egyértelmű állapothoz vezetne. Ha valamilyen okból mégis konfliktus alakulna ki közöttük, miként azt a következő alfejezetben tárgyaljuk, akkor az úgy oldódjon meg, hogy az adatbázis utána is konzisztens, egyértelműen definiálható állapotban maradjon.

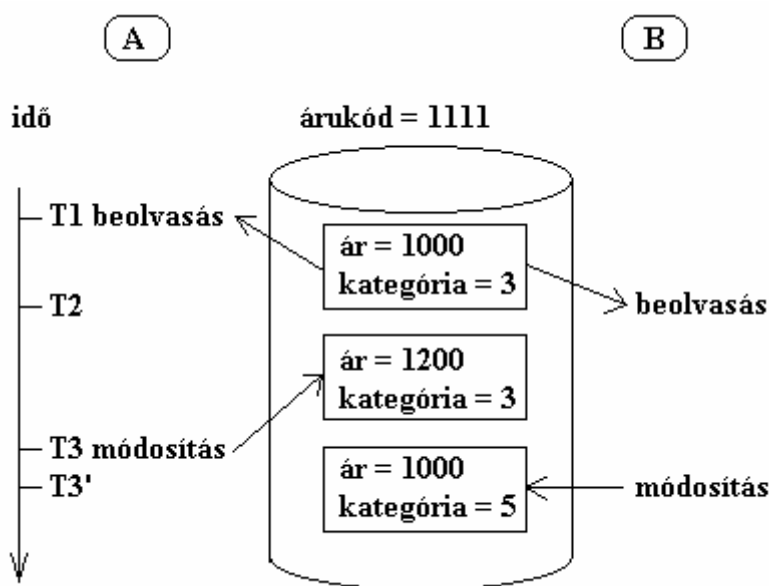
8.2. Problémák párhuzamos feldolgozásnál

A párhuzamos feldolgozásból eleve adódnak a következő hibalehetőségek:

- elveszett módosítás,
- nem véglegesített adatok feldolgozása,
- munka inkonzisztens adatokkal,
- patthelyzet.

Az első három helyzetet ebben, a patthelyzetet a 8.4. alfejezetben ismertetjük.

8.2.1. Elveszett módosítás



8.1. ábra: **Elveszett módosítás.**

Tekintsük a 8.1. ábrán látható helyzetet. Az A felhasználó beolvassa a T1, B pedig a T2 időpontban az 1111 kódú, 3-as kategóriába sorolt, 1000 Ft értékű áru-rekordot.³ Ezután A módosítja T3-kor az árát 1200 Ft-ra, és visszaírja az adatbázisba a megváltoztatott rekordot az új értékekkel (1111;3;1200). Eközben T3'-kor B is módosítja a beolvasott rekordját. Átsorolja

¹ Ez az elnevezés természetesen ugyanúgy elnagyolt, mint a tovább nem bontható atomok fogalma. De mindkettő elfogadott a mindennapi, illetve az informatikai szóhasználatban.

² A visszaállítás pontosabban csak az adatbázisnak az adott tranzakció által érintett részére vonatkozik.

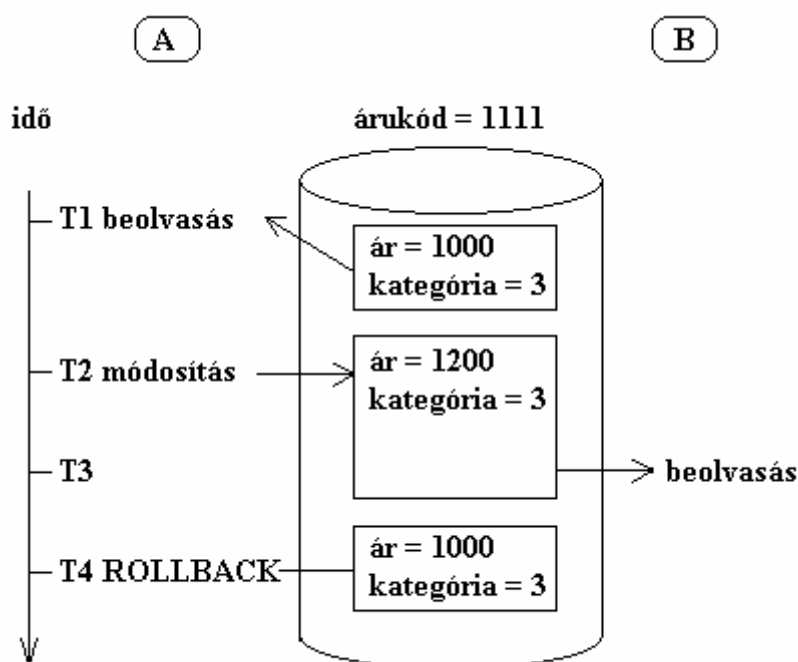
³ Az adatok mozgatása (beolvasása, kiírása) az adatbázisban *logikailag* mindig rekord (sor) szinten történik. A beolvasott rekordokban a további munka már történhet mező (oszlop) szinten.

az 5-ös kategóriába, majd visszaírja a helyére a nála érvényes új értékekkel (1111;5;1000). Attól függően, hogy $T3$ vagy $T3'$ következett be előbb, vagy A , vagy B módosítása elveszik. (Az ábrán A módosítása).

Ez nyilván megengedhetetlen. Az ilyen hibák ellen védekezni kell.

8.2.2. Nem véglegesített adatok feldolgozása

Az adatbázis-kezelő rendszer lehetővé teszi egy tranzakción belül a módosított adatok eredeti állapotának a visszaállítását. Ezért előfordulhatna, hogy két párhuzamosan futó tranzakció közül a második az első által már módosított, de még nem véglegesített rekordokkal dolgozik. Eközben az első valamilyen okból a módosítást egy ROLLBACK-kel visszavonja, érvényteleníti. Ezáltal a második tranzakció hibás, érvénytelen adatokkal dolgozott.



8.2. ábra: Nem véglegesített adatok feldolgozása.

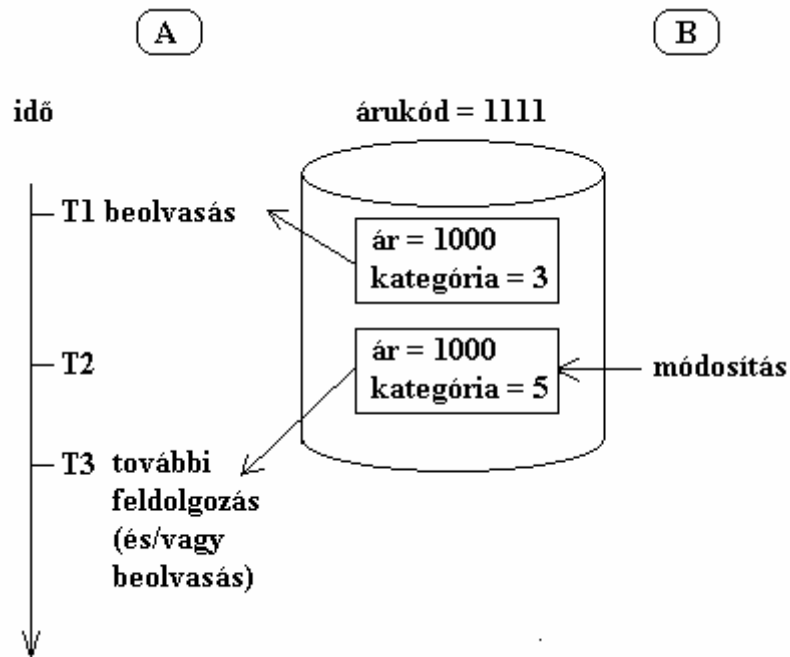
Ezt láthatjuk a 8.2. ábrán. A B tranzakció azzal a feltételezéssel dolgozik a $T3$ -kor történt beolvasás alapján, hogy az 1111 kódú áru ára 1200 Ft, holott az a $T4$ -kor végrehajtott ROLLBACK következményeképpen maradt az eredeti 1000 Ft. Hogy ez milyen hibát okoz, azt nem lehet megjósolni.

Ezért az ilyen hibák előfordulását is meg kell akadályozni.

8.2.3. Munka inkonzisztens adatokkal

A probléma lényegét a 8.3. ábrán láthatjuk. Az A feldolgozás $T1$ -kor beolvassa az előző példákban szereplő áru adatait. Különböző számításokat végez vele. Eközben $T2$ -kor B módosítja ezt a rekordot, átteszi az árut az 5-ös kategóriába. $T3$ -kor A tovább dolgozik az áru általa már beolvasott rekordján. Számára ez még mindig a 3-as kategóriában van, holott azt B már megváltoztatta. Jelen esetben az sem segítene rajta, ha újra beolvasná a rekordot. Nem tudná eldönteni, hogy korábbi eredményei a 3-as kategória feltételezésével érvényesek-e, vagy át kell-e teljesen vagy részben számolnia mindent az 5-ös kategóriával. Csak akkor

folytathatná biztonsággal a feldolgozást, ha az újraolvasásnál meggyőződne arról, hogy senki nem változtatta meg az *1111 árukódú* rekord adatait a két beolvasás között.



8.3. ábra: Munka inkonzisztens adatokkal.

8.3. Zárak

Nagy adatbázisokban egymással párhuzamosan sok felhasználó dolgozik egyidejűleg ugyanazokkal az adatokkal is. A párhuzamos feldolgozásból adódó hibák kiküszöbölésére az adatbázis-kezelő rendszerek *zárat* használnak. Az adatbázis konzisztenciájának érdekében minden tranzakció az adatbázisnak általa éppen használt részére egy zárat helyez, amit mindaddig fenntart, amíg az ott lévő adatok feldolgozását be nem fejezte. Más tranzakciók addig, amíg a zár érvényben van az adatbázisnak a zárral védett részéhez nem, vagy csak korlátozottan férhetnek hozzá. A zárolási mechanizmust egy videotéka működéséhez hasonlíthatjuk. Minden beiratkozott személy kölcsönözhet videoszalagot és DVD-t. De ha valaki már kikölcsönzött egyet, akkor azt más addig nem viheti el, amíg azt vissza nem hozzák. Ugyanakkor, a kölcsönvevővel együtt mások is megtekinthetik még ugyanazt az anyagot.

Az adatbázisban használt zárrakra jellemző

- a zár mérete,
- a zár erőssége,
- a zár időtartama,

8.3.1. A zárok mérete

Az adatbázis-kezelő rendszer elzárhatja

- a teljes adatbázist (Database Lock). Ezt nemigen alkalmazzák, mert ez a párhuzamos feldolgozást lehetetlenné teszi.
- a feldolgozandó adatokat tartalmazó adathordozó logikailag összetartozó részét (Tablespace Lock). Ezt általában a szolgáltató programok (ld. 1.3.4. pont)

használják, amikor nagy adatmennyiségek újraszervezésének, betöltésének zavartalanságát, megszakítás mentes végrehajtását kívánják biztosítani.

- a feldolgozandó adatokat tartalmazó táblázatot (Table Lock).
- a feldolgozandó adatokat tartalmazó lapot (Page Lock).
- a feldolgozandó adatokat tartalmazó sort (rekordot) (Row Lock).

Minél nagyobb az elzárt terület, annál kevesebb zárra van szükségünk, annál kisebb rendszeradminisztrációt jelent a zárok nyilvántartása, de annál jobban csökken a párhuzamos feldolgozás lehetősége. Ezért a legtöbb rendszer észszerű kompromisszumként alapértelmezésben a feldolgozandó adatokat tartalmazó lapokat zárja el. Ez azonban módosítható. A zár aktuális méretét az adatbázis-felügyelő vagy a felhasználó esetenként maga is meghatározhatja. Ezt a zárat az adatbázis-kezelő rendszer automatikusan ráteszi az adatbázisnak a használatba vett (vagy használatba venni kívánt) részére, illetve ha valamilyen okból nem tudja feltenni, akkor előjegyzésbe veszi a zárolási szándékot. Kivételt ez alól csak a táblázat szintű elzárás képez. A

- LOCK TABLE táblázat-név IN SHARE MODE illetve a
- LOCK TABLE táblázat-név IN EXCLUSIVE MODE

SQL utasítás kiadásával a felhasználó maga dönti el, mikor teszi fel ezt a zárat az utasításban megadott táblára.

Az adatleíró utasítások (ld. 5.6.2. pont) végrehajtásuk idejére automatikusan elzárják az utasításban érintett katalógus táblázatokat.

8.3.2. A zárok erőssége

A *zár erőssége* határozza meg, hogy más tranzakciók milyen módon használhatják a zárolt elemeket. Ennek alapján van

- *Osztott zár.* (Share vagy S zár). Bárki más olvashatja az így zárolt elemeket, de senki nem változtathatja meg őket.
- *Kizárólagos zár.* (Exclusive vagy X zár). Más tranzakciók nem férhetnek hozzá az így zárolt elemekhez.
- A zármechanizmus finomítása, a párhuzamos feldolgozás elősegítése és a patthelyzet (ld. 8.4. alfejezet) kialakulásának megakadályozása érdekében egyes adatbázis-kezelő rendszerek további zárokat is alkalmaznak, melyek egy későbbi feldolgozási szándékra utalnak. Ezek közül az IBM UDB/DB2-ben rendszeresített *módosítási zárat* (Update vagy U zár) ismertetjük még.

A további példákban az egyszerűség kedvéért rekord szintű zárokról beszélünk, de az elmondottak ugyanúgy érvényesek a magasabb szintű zárookra is.

Osztott zár (S) esetén bárki olvashatja az így zárolt rekordot, rátehet egy újabb S zárat. Senki más nem zárhatja el egy X zárral. Az eredeti S zár feltevője is csak akkor változtathatja meg a zár erősségét X-re, ha másnak nincs a rekordon S zára. A normál SELECT utasítás (ld. 5.6.3.2. pont) által elhelyezett S zár a rekord beolvasása után azonnal megszűnik, kivéve, ha az utasításban megismételhető olvasást (Repeatable Read, RR) kötöttünk ki.

A módosítási zár (U) azt jelzi, hogy a rekordot valamikor később esetleg meg akarjuk változtatni. Erre példa egy módosítható kurzorral (SELECT ... FOR UPDATE) való beolvasás (ld. 5.7.4. pont). Amíg a rekordot ténylegesen nem változtatjuk meg, addig bárki olvashatja, rátehet egy S zárat. De U vagy X zárat nem rakhat rá. A módosítás pillanatában az U zár átvált X zárba (feltéve, ha időközben más nem tett ugyanerre a rekordra egy S zárat). Az U zárok előnyeit a 8.4. alfejezetben a 8.7. ábrán láthatjuk.

Kizárólagos (X) zárat nem tehetünk más módon már zárolt rekordra. Ugyanígy, ha X zár van már egy rekordon, akkor erre más semmiféle zárat nem rakhat, nem végezhet a rekordban semmiféle műveletet. (A zár feltevője természetesen minden olyan műveletet

végezhet rajta, melyre jogosultsága van.) Kivétel ez alól a nem véglegesített adatok beolvasását is engedélyező olvasás (Uncommitted Read, UR). Ez végrehajtható kizárólagos módon elzárt rekordokra is. Természetesen ilyenkor a beolvasó felelőssége, hogy az esetleg hibás adatokból milyen következtetéseket von le.

Az S és az X zárat mindegyik többfelhasználós adatbázis-kezelő rendszer használja. Az U zárat nem mindegyik ismeri.

Akkor, amikor egy tranzakció egy rekordot be akar olvasni, vagy meg kíván változtatni, akkor egy S, U vagy X zárat helyez a rekordra. Ezeket a záratokat az adatbázis-kezelő rendszer zármenedzser komponense kezeli automatikusan, és ez tartja nyilván a zárolt rekordokat is. Ha egy másik tranzakció is dolgozni szeretne ugyanezzel a rekorddal, akkor az is rá akarja tenni a megfelelő erősségű zárat. Ha egy rekordon nincsen semmiféle zár, akkor értelemszerűen bármilyen zár feltehető rá. (Feltéve természetesen, hogy a tranzakció végrehajtója jogosult a rekordban a kívánt művelet elvégzésére). A 8.1. táblázat mutatja, hogy milyen zárok kompatibilisek egymással, melyek tehetők fel ugyanarra a rekordra egyidejűleg.

8.1. táblázat: Zár kompatibilitás

Első tranzakció zárja	További tranzakció(k) zárja(i)		
	Osztott (S)	Módosítási (U)	Kizárólagos (X)
Osztott (S)	+	+	-
Módosítási (U)	+	-	-
Kizárólagos (X)	-	-	-

A + jel azt jelenti, hogy a zárok kompatibilisek. Az új zár(ak) is felrakható(k) ugyanarra a rekordra az előző(k) mellé. A - jel a zárok inkompatibilitását jelzi. Ilyen zárok nem lehetnek egyidejűleg ugyanazon a rekordon. Természetesen bármely tranzakció megváltoztathatja a saját maga által feltett zár erősségét, feltéve hogy az új zár kompatibilis a rekordon már rajta lévő, más tranzakciók által föltett zárokkal.

Amikor egy tranzakció egy olyan erősségű új zárat kíván elhelyezni, amelyik nem kompatibilis a rekordon már rajta lévő(k)vel, akkor ez a tranzakció

- vár addig, amíg az inkompatibilis zár meg nem szűnik,
- vár a rendszer paramétereitől meghatározott ideig (általában 30 – 120 másodpercig),
 - ha ez alatt az idő alatt az inkompatibilitást okozó zár megszűnik, akkor ráteszi a saját zárát,
 - ha az inkompatibilitás fennmarad, akkor hibajelzést ad (és ennek eredményeként általában ROLLBACK-kel befejeződik),
- azonnal hibajelzést ad (és ennek eredményeként általában ROLLBACK-kel befejeződik).

8.3.3. A zárok időtartama

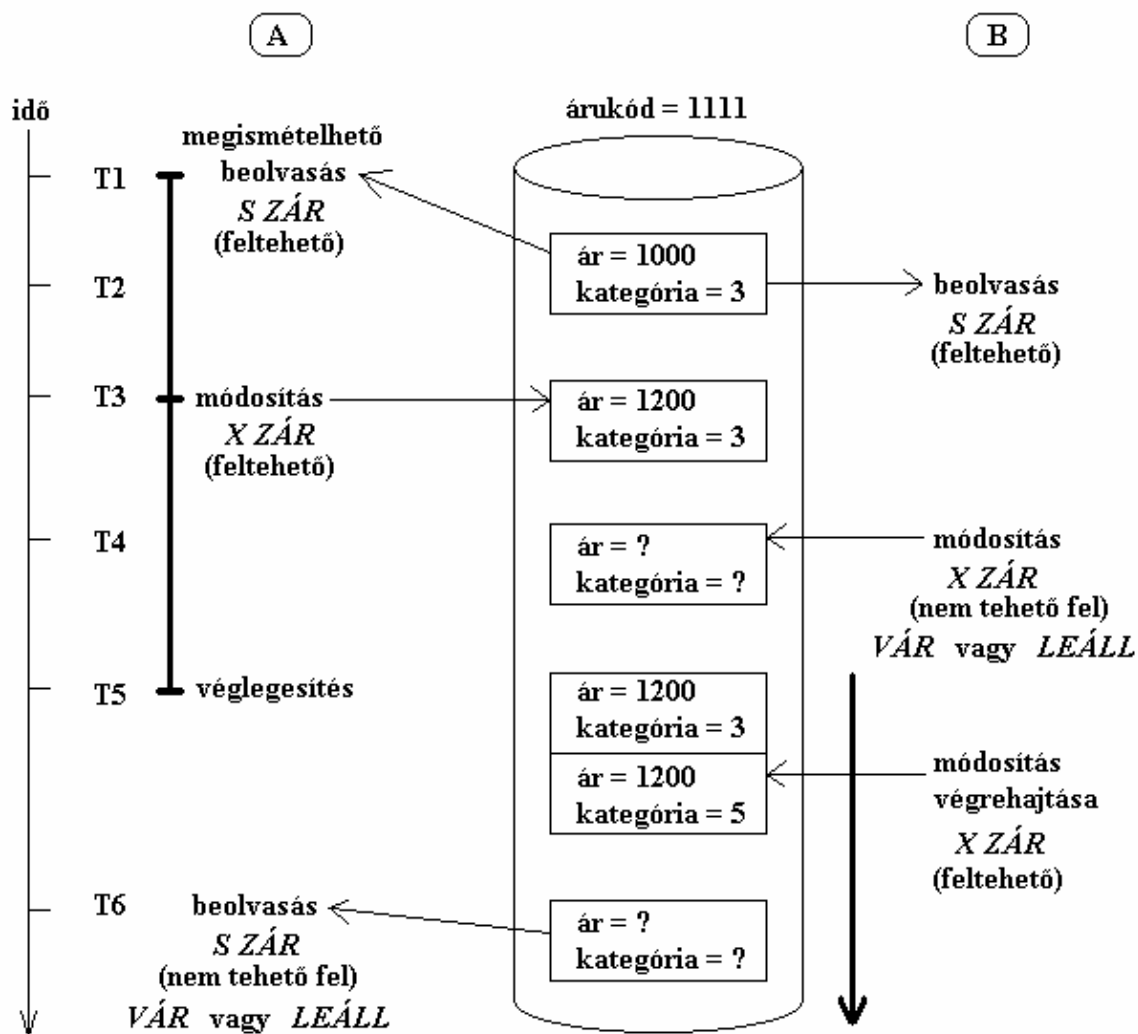
A zár általában az utasítás kiadásakor lép életbe, és a COMMIT/ROLLBACK végrehajtásakor szűnik meg.

A patthelyzetet teljesen kiküszöbölhetjük, ha az adatbázisnak azokat a részeit, melyeket a tranzakciók folyamán kizárólagosan is akarunk használni, rögtön a feldolgozás kezdetekor X zárral elzárjuk. Lap és sor szintű zároknál ezt szinte soha sem tehetjük meg. Az elzárandó lapokat, rekordokat nem ismerjük előre. Több utasításból álló tranzakciók esetén ezeket általában csak az egyes utasítások végrehajtásakor, a WHERE feltételben szereplő aktuális értékekből tudjuk meg. Statikus SQL utasításokkal (ld. 5.7.1. pont) dolgozó programokban azonban már a program elindításakor tudjuk, hogy az milyen táblázatokkal és

TABLESPACE-ekkel fog dolgozni. Így ezeket már a program indulásakor is elzárhatjuk a feldolgozásban igényelt legerősebb zárral, és elzárva tarthatjuk egészen a feldolgozás végéig. Ilyenkor patthelyzet nem tud kialakulni, de a párhuzamos feldolgozás esélyeit nagymértékben leszűkítettük.

8.3.4. A zár mechanizmus működése

A zárok működésének megértése céljából nézzük meg a 8.4. ábrán látható példát.



8.4. ábra: A zár mechanizmus működése.

A tranzakciók lényegében ugyanazok, mint amiket a 8.2. alfejezetben ismertettünk. A zárok alkalmazása következtében azonban az ott fellépő problémák itt nem jelentkeznek. A T1 időpontban az A tranzakció megismételhetően olvassa be az 1111 árukódú rekordot. Ezért egy tartós S zárat rak rá. Ez nem zavarja a B tranzakciót a rekord olvasásában T2-kor egy pillanatnyi S zárral, hiszen az S zárok kompatibilisek egymással. T3-kor, a rekord módosításakor A egy X zárat tesz az eddigi S helyett a rekordra. Emiatt B nem tudja azt T4-kor módosítani. Ezt csak a zár felszabadítása, a T5-kor végrehajtott véglegesítés után teheti

meg¹. Láthatjuk, hogy normális esetben mindkét módosítás végrehajtott. Ha viszont a második nem történt meg, akkor erről a hibajelzés révén tudomást szerzünk és ennek megfelelően cselekedhetünk. A módosítás következtében *T5*-kor most *B* rak egy *X* zárat a rekordra. Ez megakadályozza *T6*-kor *A*-t abban, hogy egy még nem véglegesített rekordot beolvasson.²

Ha *T2*-kor *B* módosítani akarná a rekordot, akkor *X* zárat kellene felraknia rá. Ezt azonban az *A* által már feltett *S* zár miatt nem teheti. Így, ha *A* újra be akarná olvasni a rekordot, akkor azt változatlan értékekkel kapná meg.

A záruk ilyen módon történő alkalmazásával a párhuzamos feldolgozásból adódó problémákat kiküszöböltük. Ugyanakkor a párhuzamos feldolgozás lehetősége – bár némileg korlátozott mértékben, de fennmaradt. Minél hamarabb oldjuk fel a zárat, minél gyorsabban véglegesítjük COMMIT/ROLLBACK-kel az adatbázis állapotát, annál kevesebbet kell a párhuzamosan futó tranzakcióknak várni, annál hatékonyabb lesz a működés.

Mivel minden véglegesítés adminisztrációs terhet jelent a rendszer számára, a gyakorlatban kompromisszumként rendszerint 0,5 – 5 percenként, illetve nagy tömegű adat sikeres feldolgozásakor 200 – 1000 rekordonként célszerű egy-egy COMMIT-ot közbeiktatni.

8.4. Patthelyzet

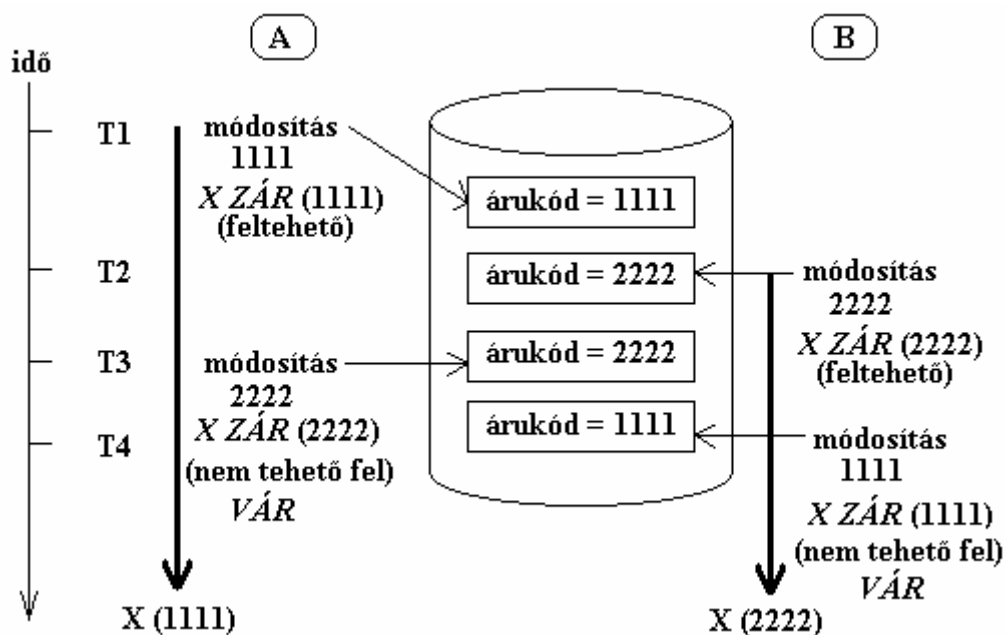
A párhuzamos feldolgozásban a legnagyobb veszélyt a *patthelyzetek* jelentik. Ezek akkor *alakulhatnak ki*, ha egyidejűleg futó tranzakciók *kizárólagosan* is lefoglalhatnak erőforrásokat, általában adatbázis lapokat vagy rekordokat. Ilyenkor két vagy több tranzakció vár olyan erőforrásokra, amelyeket egyikük *kizárólagos* módon zárolt, de nem tud felszabadítani, mert ő maga is vár olyan erőforrásra, melyet egy másik tart zárva. Két tranzakció által kialakított tipikus patthelyzetet láthatunk a 8.5. ábrán.

Az *A* tranzakció *T1*-kor módosítja az *1111* rekordot, *X* zárat tesz fel rá. Ugyanezt csinálja *B* *T2*-kor a *2222* rekorddal. *T3*-kor *A*-nak szüksége lenne a *2222* rekordra, de nem férhet hozzá a *B* által feltett *X* zár miatt. (Nem véglegesített adatok beolvasását a feldolgozás logikája nem engedi meg.) Ezért *A* vár egészen addig, amíg *B* nem véglegesíti a változtatásokat *2222*-ben. Ez általában időben bekövetkezik. *A* némi várakozás után folytathatja munkáját. Balszerencsés esetben azonban *B*-nek *T4*-kor pont az *1111* kulcsú rekordra lenne szüksége. De ehhez nem férhet hozzá *A* *X* zárja miatt. Ezt *A* nem tudja felszabadítani, mivel vár a *2222* rekordra, amit viszont *B* nem tud szabaddá tenni. A „róka fogta csuka, csuka fogta róka” tipikus esete.

Ha a várakozási idő nincsen korlátozva, akkor időtlen időig várakozna mindkét tranzakció. Ha limitálva van, akkor előbb-utóbb valamelyik tranzakció hibajelzéssel leállhat, az általa elzárt rekord felszabadulhat, és a másik tranzakció továbbléphet. Ez azonban – még ha be is következik, percekig eltarthat és oly mértékben lelassítaná az adatbázis működését, hogy az gyakorlatilag használhatatlan lenne.

¹ Ha nincs a zár felszabadulására várakozási idő, vagy az közben letelt volna, akkor *B* hibajelzést (SQLCODE = -911, SQLSTATE = 40001) kapott volna, aminek alapján eldöntheti, hogy ROLLBACK-kel befejezze-e a tranzakciót (ez az általános), a rekord kihagyásával tovább folytassa-e a feldolgozást, vagy később újra kísérelje meg a rekord elérését, hátha az időközben már felszabadult.

² Egyes rendszerek, így például az ORACLE is megengedi a nem véglegesített rekordok olvasását. Ekkor *A* a rekord eredeti, nem módosított állapotát látja. Problémát okozhat, ha ebből *A* téves következtetéseket von le. Ezért a legtöbb rendszer ezt csak akkor engedi meg, ha *A* kifejezetten jelzi, tudomása van arról, hogy esetleg már nem érvényes adatokat olvas be, azaz kifejezetten UNCOMMITTED READ (UR) paraméterrel adta meg a SELECT utasítását.



8.5. ábra: Patthelyzet

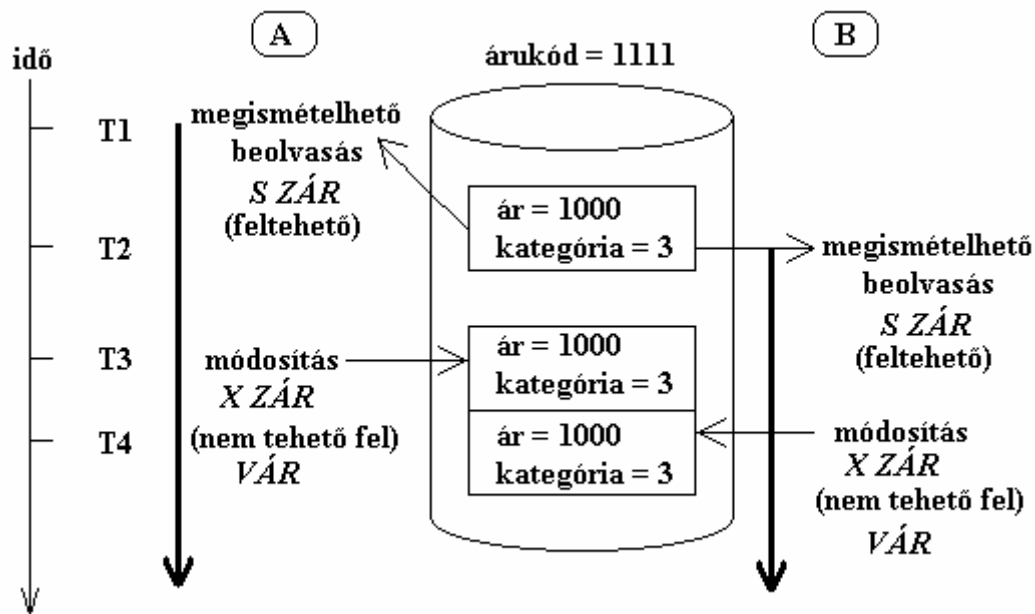
Elvileg meg lehetne akadályozni a patthelyzet kialakulását, ha a tranzakciók rögtön induláskor feltennék az általuk menet közben igényelni kívánt összes zárat. Ezt azonban, mint azt már a 8.3.3. pontban megindokoltuk, a gyakorlatban nem lehet az adatbázisokban megvalósítani. Ezért minden adatbázis-kezelő rendszer, mint szükséges rosszat tudomásul veszi, hogy *patthelyzetek kialakulhatnak*. Amit ez ellen tehetnek, csupán annyi, hogy igyekeznek ennek valószínűségét, és a bekövetkezésük miatti várakozási időt minimálisra csökkenteni. Minél kisebb elemeket zárolunk, annál kisebb a valószínűsége annak, hogy két tranzakciónak egyidejűleg arra legyen szüksége. Ekkor viszont több zárra van szükségünk, aminek kezelése és nyilvántartása megnöveli a rendszer adminisztrációs munkáját.

Az idővesztés minimalizálása érdekében az adatbázis-kezelő rendszer záratokat kezelő komponense 5-10 másodpercenként végigvizsgálja a lefoglalt és a lefoglalni kívánt erőforrások listáját. Ebből meg tudja állapítani, hogy létrejött-e patthelyzet. Ha igen, akkor az egyik tranzakciót hibaüzenettel leállítja, annak zárai feloldódnak. A másik tranzakció tehát az emiatti pár másodperces várakozás után akadálytalanul dolgozhat tovább. A leállított tranzakciót természetesen újra kell indítani. De erről legalább igen hamar értesülünk.

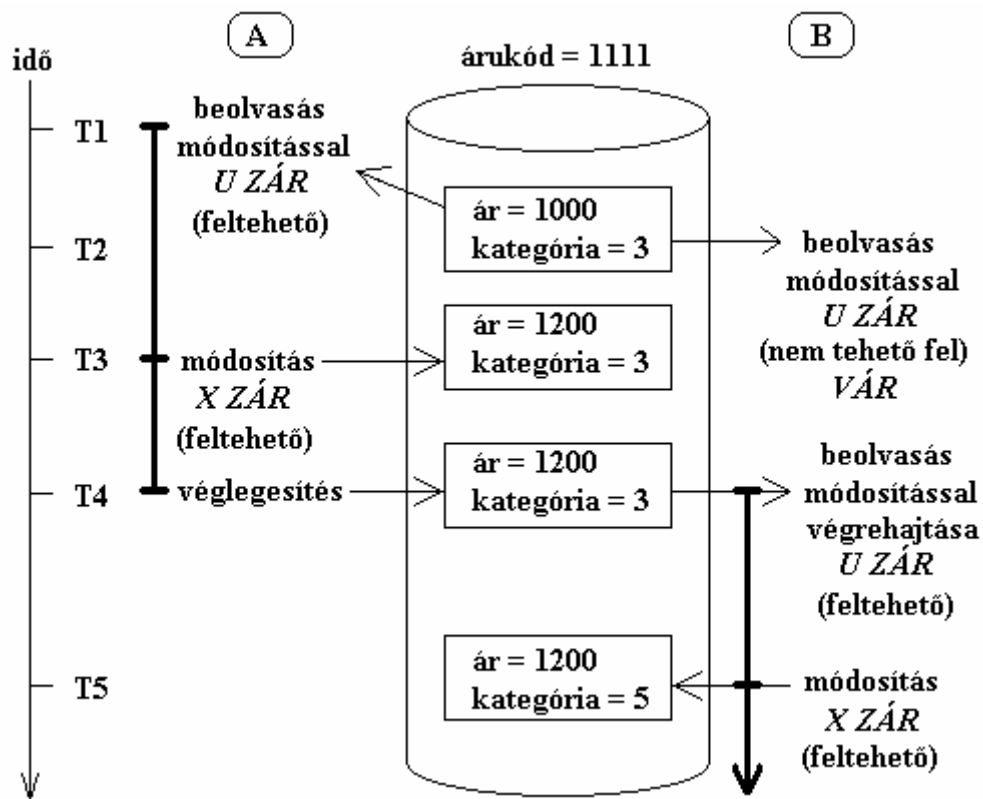
Végül a 8.6. és 8.7. ábrán bemutatjuk, miként alakulhat ki patthelyzet ugyanazon rekord megismételhető módon való beolvasásakor is, és hogy hogyan segít ezen az U záruk bevezetése.

Az *A* tranzakció *T1*-kor, a *B* *T2*-kor olvassa be az *1111* árukódú rekordot megismételhető (RR) olvasással. Miután nem tudják biztosan, hogy meg fogják-e menetközben változtatni, egyikük sem tesz rá X zárat, hiszen ekkor *B* biztosan nem tudna párhuzamosan dolgozni. *A* *T3*-kor, *B* pedig *T4*-kor szeretné ténylegesen megváltoztatni ezt a rekordot. Ezért a meglévő S zárat át akarják alakítani az erősebb X zárrá. De ezt egyikük sem teheti meg a másik S zára miatt. Létrejött a patthelyzet!

Ha *A* *T1*-kor gondol arra, hogy alakulhat úgy a feldolgozás, hogy módosítani akarja a rekordot, akkor S helyett U zárat tétet fel rá (ld. 8.7. ábra). Ha *B* csak olvasni kívánná az *1111* rekordot, akkor ezt minden korlátozás nélkül megteheti. Az S és az U záruk kompatibilisek egymással. Ha viszont *B* is számít esetleges módosításra, akkor neki is U zárat kellene feltetnie. Ez a már létező U zár miatt nem lehetséges. Várakozni kényszerül, míg a *T4* időpontban az *A* tranzakció véget nem ér. Ami viszont bekövetkezik, mert patthelyzet nem alakulhatott ki.



8.6. ábra: Patthelyzet ugyanazon rekord miatt.



8.7. ábra: Patthelyzet megakadályozása U zár alkalmazásával

Ez a példa is mutatja, valamit valamiért. A hatékony párhuzamos feldolgozást csak észszerű kompromisszumokkal lehet megvalósítani. Mint azt már többször hangsúlyoztuk, „Nincs potya kaja”, de legalább elfogadható áron lakjunk jól.

8.5. Problémák és megoldásuk

Ebben az alfejezetben a szerzők közül az idősebb (QP) több évtizedes gyakorlati tapasztalatából ismertet néhány olyan hibát, melyet elsősorban főnökei, munkatársai, beosztottjai, de őszintén bevallva időnként ő maga követett el, majd szerencsére ki is javított. Utólag visszatekintve, e tévedéseknek a többsége annyira nyilvánvaló volt, hogy soha sem lett volna szabad ezeket elkövetni. Gondos tervezéssel, a feladat alapos végiggondolásával mindegyikük elkerülhető lett volna. De mindenki, aki csak kicsit is komolyabban dolgozott olyan informatikai rendszerekkel, melyeket „profi” fejlesztők készítettek a számára, vagy megfordítva, neki kellett mások által használandó szoftvert előállítani, tudja, hogy hiba nélküli informatikai rendszer nem létezik. Legfőleg még nem vették észre az összes hibáját.

A jó adatbázis-felügyelőt az különbözteti meg a kezdőtől, hogy a jó már sokkal több hibát követett el, mint a kezdő. A kiváló pedig még többet. Hiába mondják, hogy ugyanabba a verembe, stackbe (bocsánat a számítástechnikai szakkifejezésért, magyarul folyóba) nem léphetsz kétszer bele, és hogy a bölcs a más kárán tanul, még a legbölcsebb adatbázis-felügyelő és felhasználó is csak azokból a hibákból okul igazán, amelyeket saját maga is elkövetett. Ha valaki napokat töltött el egy ravasz hiba megtalálásával és kijavításával, akkor úgy véli, ugyanazt a hibát még egyszer nem fogja már elkövetni. De ha nem módosít pályát, vagy nem vonul ideiglenes vagy örök nyugdíjba, akkor bizony nagy valószínűséggel újra beleesik ugyanabba a verembe. De ekkor már sokkal hamarabb, esetleg már néhány óra alatt rájön a hiba okára. Kijavítja, és többet már nem fogja ugyanezt elkövetni. Vagy ha mégis, akkor már tíz perc alatt megoldja a problémát, és ezután ... Ha elég hosszú ideig dolgozik a szakmában és elég sok helyen megfordul, akkor ennek a majdnem végtelen ciklusnak a végére ezen probléma kezelésének országosan elismert szakértője lesz.

Miután hosszú életem során elég sok hibát elkövettem már (sajnos nem csak adatbázisokkal kapcsolatban), ezek tanulságait levonva néhány „tipikus eset” ismertetésével szeretnék az olvasók számára segítséget nyújtani, hogy hasonló hibákba ne essenek bele, vagy megoldásukra ötleteket kapjanak.

Egy igazán profi adatbázis-felügyelő teljesítményére az jellemző, hogy hasznos munkaidejének mintegy 70%-át azoknak a problémáknak a megoldása teszi ki, amelyek abból adódtak, hogy a felhasználók és a menedzserek *nem fogadták meg* a tanácsait, javaslatait. A megmaradt 30%-ban pedig azokat a problémákat próbálja kiküszöbölni, amelyek azért alakultak ki, mert *megfogadták* a tanácsait. Amíg ezt az arányt tartja, addig munkája eredményes.

8.5.1. Be kellett a rácsos kaput zárni

A példákban ismertetendő rendszerben 16 darab több processzoros óriási IBM mainframe volt. Ezekre 60 darab középgep csatlakozott, melyeken összesen 2000 PC lógott adatlekérdezési és beviteli célokból. Az adatállományt az IBM gépeken néhány szekvenciális fájl kivételével UDB/DB2-ben, a középgepeken INFORMIX-ban tárolták. A rendszer az üzemszerű működéskor mintegy 250 rekord típust és 2 milliárd rekordot tartalmazott. Ebben a biztonsági mentéseket és a hatékonyság érdekében elvégzett rendszeres átszervezéseket nem számítva naponta tíz millió tranzakciót kellett végrehajtani. Az informatikai fejlesztésen és a karbantartáson éveken keresztül 150-250 ember dolgozott folyamatosan.

Az on-line tranzakciók indítása, új adatok bevitele PC-kről történt. Az adatok zömét a középgepeken ellenőrizték, és onnan küldték tovább a megfelelő mainframere. Bizonyos ellenőrzéseket azonban nem lehetett azonnal elvégezni, az ehhez szükséges adatok a nagygepeken voltak. Ezeket a rendszer összegyűjtötte. Éjszaka, kötegelt üzemmódban hajtották őket végre. Ennek eredményét a középgepeken keresztül visszaküldték a PC-kre és az ellenőrzés befejeztét nyugtázták. A tranzakciókban résztvevő rekordok státuszmezőjében a

mainframen jelezték ezt: feldolgozásra váró - feldolgozott - visszaküldött - visszaérkezett. A rendben visszaérkezett tranzakciók egy idő után törlődtek a nagygépen. A kötegelt feldolgozás végét egy erre szolgáló speciális bit jelezte. Ha szabadot mutatott, a középgepek nekiugrottak a mainframenek. Átvitték a DB2-s táblázatok adatait és módosították a státusváltókat.

A rendszertesztnél semmi probléma nem merült fel. Amikor azonban 4-5 középgep egyidejűleg rohamozta meg a DB2-t, beállt a katasztrófa. Pontosabban a patthelyzet. A DB2-ben ugyanis a zárméret automatikusan a lap. Mivel egy tranzakció több táblázatot, és egy táblázaton belül több rekordot is érintett, a módosított rekordokat tartalmazó lapokat a rendszer automatikusan elzárta a többi felhasználó elől. Mivel a státusmódosítások még egy tranzakció rekordjainál sem következtek be egyidejűleg, és ugyanarra a lapra különböző tranzakciók lapjai is kerülhettek, az elzárt lapok pillanatokon belül összekeveredtek. Létrejött a patthelyzet.

Az első gondolat az volt, hogy a középgepek egymás után vegyék át az adatokat. Ekkor viszont kifutottunk volna az időből. Így maradt a másik, utólag visszagondolva pofonegyszerű megoldás. Megváltoztattuk a zárméretet! Nem a lapokat, hanem csak a megváltozott rekordokat zároltuk. Ezekben a táblázatokban a feldolgozásban éppen érintett rekordok száma nem túl nagy, legföljebb 15-20 ezer lehet. Ekkora mennyiségnél a lapzárak helyett az egyedi sorzárak alkalmazásából adódó rendszeradminisztrációs többletmunka elviselhető. Ilyen módon a patthelyzetek és az ebből következő kényszerű programleállások száma napi százról a már kezelhető 2-3 -ra csökkent.

A következő példában az előbbi korrekció ellentettjét mutatjuk be. Nem szabad mindig kicsire választanunk a zárolási egységeket azért, hogy a párhuzamos feldolgozás lehetőségét megnöveljük. Egy törzsadat-változtatást végző programban a programozó nem véglegesítette a változásokat (ami egyben a zárok feloldását is maga után vonja) feldolgozás közben, hanem csak a program végén. Ez általában nem okozott gondot, mivel egy feldolgozásban legföljebb néhány tízezer rekord módosult. Egyszer azonban – még a feltöltés állapotában – a változások száma (új adatok bevitele) a több milliót is elérte. Ilyen sok zár kezelésére a rendszernek nem volt elegendő munkaterülete. Ebbe nemcsak ez a program halt bele. Az egész adatbázis-kezelő rendszer leállt!

A helyes megoldás ez lett volna. Kiegészítjük a programot, hogy időnként (pl. minden 500 rekord feldolgozása után) adjon ki egy COMMIT-ot, szabadítsa fel a zárokat. Ez azonban a program átdolgozását jelentette volna. Mivel a problémát sürgősen meg kellett oldani, egy ideiglenes, de gyors megoldást választottunk a közlekedésből jól ismert példa alapján. Ha túl sok baleset van egy úton, akkor a legbiztosabb az egész utat elzárni mindennemű közlekedés elől. Lapok helyett induláskor az egész táblázatot elzártuk. Így millió lapzár helyett csak egy táblázat zár maradt. Igaz, hogy ezen idő alatt senki más sem férhet a táblázathoz. De ilyen nagy volumenű módosítás esetén ne is tegye – volt a magyarázat. És különben is, ez csak egy ideiglenes megoldás, mihelyst időnk lesz, átdolgozzuk a programot.

Természetesen az „ideiglenes” megoldás véglegessé vált. Mivel a törzsadat karbantartó program csak éjszaka futott, senki nem vette észre, hogy az egyik legfontosabb táblázat hosszabb ideig el volt zárva minden más feldolgozás elől. Történt azonban, hogy valamilyen ok miatt a programot csak hajnalban indították el, és még akkor is futott, amikor az on-line feldolgozás elkezdődött volna. De senki sem tudott dolgozni, mert mindenkinek szüksége lett volna erre az elzárt táblázatra. Az üzemből viszonylag hamar kiderítették a hiba okát. Jogosan úgy döntöttek, hogy az on-line feldolgozásnak adnak elsőbbséget. Ezért a törzsadat-karbantartó programot mesterséges külső beavatkozással leállították. Igen ám, de ez az adatbázis konzisztens állapotának megtartása miatt egy ROLLBACK-et vont maga után. Ennek a programnak – miután nem volt benne COMMIT – az elindításától kezdve végrehajtott összes módosítását érvényteleníteni kellett. Ez pedig olyan régen történt, hogy a

helyreállítás alapjául szolgáló, a változtatásokat nyilvántartó fájl már archiválva lett, nem volt közvetlenül elérhető. Miután az idő sürgetett, a helyreállítást is leállították, aminek eredményeként az egész adatbázis-kezelő rendszert újra kellett indítani. Azt viszont nem lehetett addig, amíg az adatbázist a változtatások előtti állapotba vissza nem állították. Ezek a munkák végül majdnem egy napot vettek igénybe. Addig senki nem dolgozhatott az adatbázissal. Ezután az epizód után a karbantartó programot átírtuk. Lapzárakkal és rendszeres COMMIT-tal dolgozik, immár mindenki teljes megelégedésére.

Konklúzió: Nem mindegy, hogy mit és hogy mekkora lakattal zárunk le.

8.5.2. Utolsó pár, előre fuss!

Viszonylag sok időt takarítottunk meg a következő egyszerű trükkel is.

A napi tranzakciókat tartalmazó különféle táblázatokban az elsődleges kulcs az egyedtípus azonosítója mellett egy folyamatosan növekvő sorszám volt. Ennek értékét új adat bevitelkor a program határozta meg a

```
SELECT MAX(SORSZÁM) FROM ...  
WHERE AZONOSÍTÓ = :inputból jövő azonosító
```

SQL utasítás alapján. Az új sorszám a kiválasztott maximális sorszám +1 lesz.

Ehhez végig kellett olvasni az adott azonosítóhoz tartozó összes index bejegyzést, esetenként akár több százat is.

Az *AZONOSÍTÓ* + *SORSZÁM* összetett indexben a *SORSZÁM* szerinti rendezettséget csökkenőre módosítva az első indexbejegyzés megadta azonnal az utolsó sorszám értékét. Még az adott azonosító értékhez tartozó indexeket sem kellett végigolvasni.

Sok kicsi sokra megy. Mivel a műveletet nagyon sokszor kellett végrehajtani, a futási idő érzékelhetően lecsökkent.

Konklúzió: Nem mindegy, melyik kályhától indulunk el, ha már teli a szoba.

8.5.3. Hülye-biztos programokkal csak hülyék dolgoznak

A felhasználók és a rendszerszervezők biztosítani szeretnék magukat minden lehetséges meglepetés ellen. Mindent, mindenhol le akarnak ellenőrizni, hogy még ha szemét is menne be, akkor se jöjjön szemét ki. Az ilyen programokat hülye-biztosnak tekinthetjük. De egy idő után annyira nehézkessé válhatnak, hogy csak a hülyék fogják ezeket használni. Papíron, tesztkörnyezetben az ilyen „hülye-biztosítás” nem okoz gondot. Legföljebb meghosszabbítja a program tervezéséhez, kódolásához és az adatbázis kialakításához szükséges időt. De amikor egy régebbi változathoz esetleg megmaradt hiba kiszűrése miatt naponta végig kell vizsgálni tíz milliós táblázatokat, akkor ez a biztonság már igen sokba kerülhet.

Az egyik 100 millió rekordot tartalmazó törzsadat táblázatba a kezdeti hibás tervezés miatt belekerülhettek olyan adatok, melyeknek keletkezési dátuma vagy a bevitelért felelős fiók ismeretlen volt. Ezért a szervező minden feldolgozási ciklus előtt beépített egy ellenőrzést, vajon vannak-e ilyen adatok. Ez a teljes táblázat soros végigolvasását, ennél az adatmennyiségnél már napi 10-15 percet jelentett. Persze lehetne indexet is építeni ezekre a mezőkre. Ekkor az ellenőrzési idő másodpercekre csökkenne. Csak éppen az indexek tárolására még 4-5 Gigabyte-ra lenne szükség. Ez pedig még egy ekkora rendszernél is megfontolandó.

Az adatok elemzése azt mutatta, hogy a 100 millió rekordból csupán egynek volt ismeretlen a keletkezési dátuma és ezernél is kevesebb nem tartozott egyik fiókhoz sem. De ezt mindig leellenőrizték. A helyes megoldás: egyszer kijavítani a hibás adatokat. Az

ellenőrző feltételt beépíteni az adatbázisba és ez által a hibás adatok bevitele lehetetlen lesz. A napi ellenőrzést nem kell többé végrehajtani.

Egy idő után végül meg is valósítottuk ezt.

Konklúzió: Ne akarjuk a programot minden áron hülye-biztossá tenni. Inkább eleve óvjuk meg a hülyéktől.

8.5.4. Nincs optimális optimalizáló

A nagy információs rendszerek napjainkban a relációs modellre épülnek. Ezekben létrehozhatunk a hatékony feldolgozás érdekében különböző objektumokat (tablespace-k, indexek), meghatározhatunk paramétereket (sűrűsödés, üres helyek fenntartása későbbi bővítésekre, zárolási egységek). De abba, hogy egy adott feladat megoldásához a rendszer milyen utat válasszon, közvetlen beavatkozási lehetőségünk az utóbbi néhány év kivételével általában nem volt.

A feldolgozási programoknál a gyakorlatban legtöbbször a fordítás utáni BIND lépésnél (ld. 5.7.2. pont) határozza meg az SQL optimalizáló a – szerinte optimális – elérési utat. Az optimalizáláshoz szükséges információk a katalógusban (ld. 7. fejezet) vannak. Ezeket azonban csak időnként, a statisztikákat elkészítő szolgáltató programok futtatásakor aktualizálják, mert különben minden erőforrás rámenne a statisztikák készítésére. Ezért fontos, hogy a statisztikai adatok eloszlása lényegében azonos legyen az elérési út meghatározásakor (a BIND időpontja) és a program futásakor.

Hogy az milyen bajokat okozhat, ha nem így van, ismét egy életből vett példán mutatjuk be. Itt csak effektív csalással tudtuk az SQL optimalizálót a helyes útra terelni.

A napközbeni on-line feldolgozás különböző típusú tranzakciókat olvasott be, tárolt, majd törölt. Az ideiglenesen tárolt rekordok száma menetközben több százezer is lehetett. Ezek többségének típuskódja '00' volt. Az ilyen kódú tranzakciók az on-line feldolgozás végén törölődtek az adatbázisból. Így amikor a statisztika készült, az azt mutatta, hogy ilyen tranzakció egyáltalában nincsen. A programok gyakran kerestek rekordokat a tranzakcióban szereplő személy azonosítója és a típuskód = '00' feltétel alapján. Bár az első feltétel szerint a kiválasztás majdnem egyértelmű (egy sorszám jelzi még, az illetőnek aznap ez hányadik tranzakciója volt), az optimalizáló mégis a típuskódra épített indexek alapján ment végig a rekordokon, hiszen itt a statisztika azt ígérte, hogy rögtön megállapíthatja, nincs találat. A valóság azonban az volt, hogy ahogy telt az idő, úgy lett egyre több '00' típusú rekord a táblázatokban. Több tízezer ezen indexértékhez tartozó rekordot kellett végigolvasni az adott tulajdonos néhány rekordjának megtalálásához.

Új statisztika és BIND itt csak akkor segített volna, ha azt napközben, adatokkal feltöltött táblázatokon végeznénk el. Ez azonban zavarta volna a feldolgozás menetét. Itt már csak a csalás segített. Rendes statisztika helyett mesterségesen meghatározott értékeket vittünk be a katalógus megfelelő helyeire. Így az optimalizáló azt látta, érdekesebb neki a tranzakció tulajdonosára épülő index néhány rekordját végigolvasni és megvizsgálni, vajon '00'-e ott a típuskód. Ezt a hamisítást egyszer hajtottuk végre. A programokat újra 'BIND-eltük'. Azóta ezeknek a táblázatoknak a statisztikáját nem aktualizáltuk. A feldolgozás ideje csaknem egy nagyságrenddel lecsökkent.

Egyes rendszerekben (pl. ORACLE, UDB/DB2 újabb verziói) ma már lehetőség van arra, hogy mi adjuk meg közvetlenül az utasításban, milyen úton kívánjuk elérni az adatokat. Ekkor nem kényszerülnénk csalásra. Ez persze veszélyeket is rejt. Nem mindig leszünk okosabbak az SQL optimalizálónál!

Első konklúzió: Csak annak a statisztikának higgyél, amelyiket magad hamisítottál meg.

Második konklúzió: Egy stradivari hangjait csak egy nagy művész tudja igazán kihozni.

8.6. Ellenőrző kérdések

1. Milyen problémák léphetnek fel párhuzamos feldolgozásnál?
2. Milyen szempontok szerint csoportosíthatjuk a záratokat?
3. Milyen záratok kompatibilisek egymással?
4. Mi a különbség az ismételhető olvasás (Repeatable Read) és a nem véglegesített adatok olvasása (Uncommitted Read) között?
5. Mi a patthelyzet kialakulásának az előfeltétele?
6. Miért nem tudjuk adatbázisokban teljesen megszüntetni a patthelyzetet?

9. ADATTÁRHÁZAK

A könyv eddigi fejezeteiben végig feltételeztük, hogy az adatbázisokat felhasználó szervezetek egy egységes, integrált adatbázist hoztak létre és azt működtetik. A valóság azonban még ma sem ilyen szép! A legtöbb helyen egyes részlegek, alkalmazási területek a saját igényeiknek megfelelő önálló adatbázissal (is) dolgoznak, melyek csak többé-kevésbé illeszkednek be az integrált rendszerbe. Némi túlzással azt is mondhatnánk, hogy a vállalatok, szervezetek megfulladnak az adatok tengerében, de éhen halnak az információk hiányától. Magyarországon még mindig nem sikerült létrehozni az egységes államigazgatási információs rendszert, amelynek az alrészei (belügyi, egészségügyi, adó, stb.) teljes mértékben átjárhatóak és kompatibilisek egymással. Ennek részben technikai okai is vannak, de jelentős mértékben hozzájárul az a tévhit is, ami a személyi adatok eltúlzott védelméből adódik.

A hagyományos adatbázisok elsősorban a szervezet napi feladatait támogatják. A szervezet működéséhez szükséges *tranzakciók* regisztrálását, folyamatos feldolgozását végzik el. A vezetői döntésekhez azonban másféle információk is szükségesek. A pillanatnyilag érvényes adatok mellett nagy súly helyeződik a különböző szempontok szerint összevont adatok aggregátumaira és ezek időbeli változásaira. Bár a hagyományos adatbázisokkal ilyen feladatokat is meg lehet oldani, célszerűbbnek látszott a működéshez és az elemzéshez szükséges adatokat szétválasztani. Az elemzéshez szükséges adatok tárolására, egyszerű és hatékony elérésére hozták létre az *adattárházakat* (*Data Warehouse*) és az ezekben rejlő információk felderítésére szolgáló speciális módszereket, elsősorban az *adatbányászatot* (*Data Mining*). A 9.1 táblázatban összefoglaljuk, melyik módszer milyen feladatok elvégzésére a legalkalmasabb.

9.1. táblázat: A hagyományos adatbázisok és az adattárházak összehasonlítása.

Módszer	Fő funkciók	Jelleg
Hagyományos adatbázis	Lekérdezések, tranzakciók, módosítások	Dinamikus, elsősorban az aktuális állapotot tükrözi
Adattárház	Lekérdezések igen nagy statikus adatmennyiségből	Időbeli változásokat mutatja, Adatok elemzésére szolgál
Adatbányászat	Az adattárház adatai közt fennálló (rejtett) összefüggésekre derít fényt	Ad hoc és nem pontosan definiált feladatok megoldása

9.1. Az adattárház jellemzői.

Az adattárház a relációs adatbázisoknak egy speciális típusa, amelyet nem tranzakció feldolgozásra, hanem lekérdezések és elemzések végrehajtására terveztek. Különböző tranzakciós adatokból származtatott adatokat tárol egységes formában időbeli változásuk és felhasználási területük szerint csoportosítva. Bevezetésükkel a tranzakció és az elemző feldolgozás szétválik. Az első továbbra is az adatbázisokban történik, míg a második (egyre inkább) az adattárházakban valósul meg.

Az adattárházhoz hozzátartozik a kiszolgáló környezete is. Ez az adattárházba bekerülő adatok kiválasztását, átvitelét, egységes formátumra való átkonvertálását, betöltését, on-line történő automatikus elemzését és az eredményeknek a felhasználó számára áttekinthető megjelenítését is biztosítja.

Az adattárház főbb jellemzői:

- Témakörök, felhasználók (pl. beszerzés, eladás, árukészlet,) szerint orientált. Ha ezek a specializált igények szerint szerkezetileg függetlenek egymástól és önállóan lekérdezhetőek, akkor az adattárház így elkülönülő részeit *adatpiacoknak (Data Mart)* nevezzük.
- Integrált. Egységes a különböző helyekről származó adatok elnevezése, tárolási formája.
- Időfüggő. Az események időbeli változása nyomon követhető.
- Statikus. Az egyszer beírt adatok soha sem módosulnak. A változást csak új adatok beírása, vagy elavultak végleges törlése jelenti. Ezekkel a műveletekkel nem folyamatosan, hanem csak időnként aktualizálják az adattárház adatait.

9.2. A hagyományos adatbázisok és az adattárházak összehasonlítása.

A tranzakció feldolgozásra orientált hagyományos adatbázis komplex adatszerkezetét normalizált, általában 3NF relációkban (ld. 4.6.4. pont) ábrázolják. Ezekre a feldolgozás hatékonyabbá tétele érdekében néhány indexet építenek és rendszeresen képezik különböző relációk joinját, ami nagy táblázatoknál meglehetősen időigényes. A redundanciát kerülik és csak ritkán tárolnak az alapadatokból kiszámolható származtatott adatokat. Az egyes rekordokat rendszeresen módosítják és az utolsó állapot hatékony elérésére optimalizálnak. A feladatok jelentős részét előre be lehet programozni.

Ezzel szemben az adattárházak a relációs modell lehetőségeit kibővítik a 9.4.2. pontban ismertetett *csillag szerkezettel* és a *dimenziókkal*. Lényegesen több indexszel dolgoznak, de jóval kevesebb joint használnak. A gyakran szükséges joinok helyett inkább redundánsan tárolnak adatokat 1NF és 2NF relációkban. Mivel a már beírt adatok tartalmát *sohasem módosítják*, az alacsonyabb normalizációs fokból adódható anomáliák (ld. 4.6.3. pont) nem okoznak problémát. A gyors lekérdezés érdekében összevont adatokat, aggregátumokat is rendszeresen tárolnak (Pl. időszakok, vagy hely szerint összegzett adatokat). Az adatokat hosszú időszakra visszamenőleg is tárolják. Az adattárházakban nagyságrendekkel több adatot tárolnak és többel is dolgoznak egyidejűleg, mint a tranzakciós adatbázisokban.

A fenti eltéréseket a 9.2 táblázatban foglaltuk össze.

9.2. táblázat: Tranzakció orientált adatbázisok és adattárházak fő eltérései.

	Tranzakció feldolgozásra orientált adatbázis	Adattárház
Feladatok	Elsősorban adott feladatokra optimalizált	Ad hoc jellegű lekérdezések
Adatok változtatása	Bármikor lehetséges	Bent lévő adatokat nem lehet változtatni. Új adatok bevitele, elavultak törlése csak meghatározott intervallumokban lehetséges
Adatmodell	Normalizált (3NF)	Részben normalizált, és csillag. Lekérdezésre optimalizált
Tipikus műveletek	Néhány rekord kiválasztása, módosítása, beírása, törlése	Sok millió rekord beolvasása
Adatok időbeli változásának tárolása	Aktuális, vagy csak rövid időre visszamenőleges	Hosszú időre visszamenőleg tárolt

9.3. Adattárház architektúra

Minden adattárház a következő műveletekkel és komponensekből épül fel.

- adatforrások,
- adatkiválasztás,
- adatelőkészítés és bevitel,
- eredeti és származtatott adatok és definíciójuk tárolása,
- on-line analitikus elemző,
- felhasználói analizáló eszközök,
- adatbányászati eszközök,
- eredmény megjelenítők.

Az alábbiakban röviden ismertetjük ezek feladatait.

Adatforrások. Az adattárházba különböző forrásokból, tranzakciós adatbázisokból, különféle adatfájlokból, esetenként közvetlen inputból kerülnek be az adatok. Ugyancsak innen kerülnek be az adatok értelmezésére szolgáló adatleírások, az úgynevezett *metaadatok* is. (Ezek lényegében a 7. fejezetben tárgyalt katalógusnak felelnek meg.). Ahhoz, hogy ezekből összeálljon az adattárház adattartalma, az alábbi lépésekre van szükségünk:

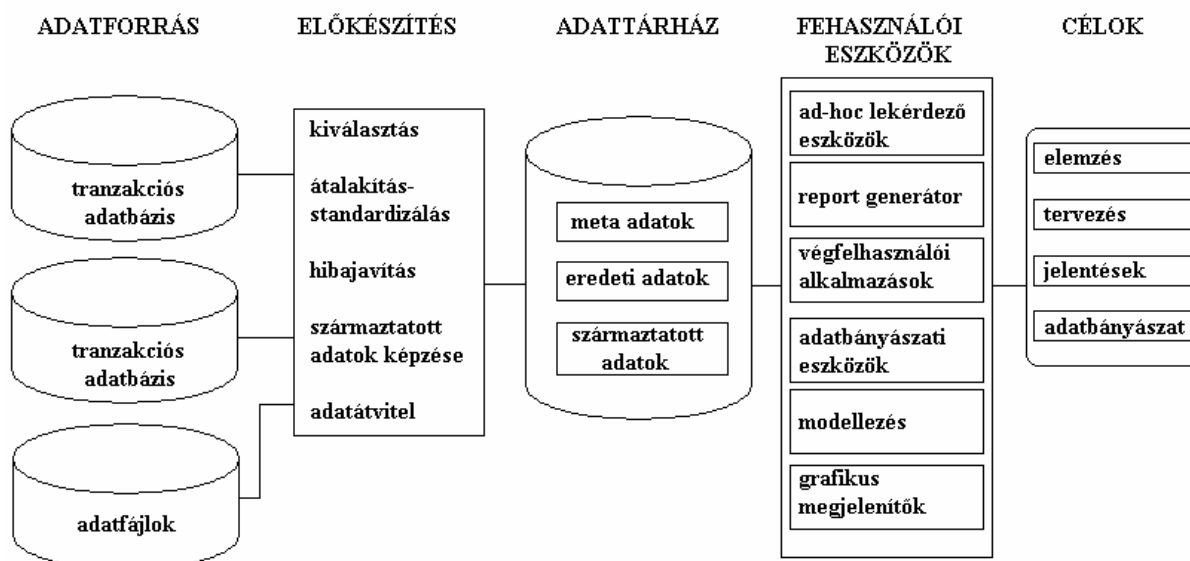
Az adatok kiválasztása. Az esetenként több tucat adatforrás adataiból kijelöljük azokat, amelyeket át kell vinnünk az adattárházba. Ez a kiválasztás az adatok tartalma és az elemzési feladatok alapján történik. Tartalmazza azokat, amelyeket az adattárházban eredeti formájában is tárolunk és azokat is, amelyekből számítással, aggregátumok készítésével, más adatokkal való összevonással (pl. joinnal) olyan *származtatott adatokat* képezünk, melyeket szintén tárolni fogunk az adattárházban.

Az adatok egységesítése, ellenőrzése, javítása. A különböző helyekről származó adatok elnevezése, formája, értelmezése eltérő lehet. Ezért ezek az adattárházba való bevitel előtt egy egységesítési, tisztítási és ellenőrzési folyamaton mennek keresztül. Ez lehet egy egyszerű, automatikusan elvégezhető mértékegység transzformáció (pl. Fahrenheit fokok átszámítása Celsiusra, azonos áruk mennyisége kg-ra), nevek és hozzájuk kapcsolódó címek (Dr., ifj., gróf, stb.) azonos formába való átírása, különböző azonosító kulcsok (pl. katalógus szám és alkatrész szám) egységesítése. De lehet egy hibajavítással kombinált ellenőrzés is. (Pl. ugyanahhoz a személyhez két eltérő lakcím tartozik). Ugyancsak idetartozik az azonos adatok egységes értelmezése is. Tipikus példa erre, amikor nem használják a *NULL* értéket. (Pl. ha nem ismerik valaminek a kezdési vagy befejezési időpontját egy „értelmezhetetlen” dátumot, ismeretlen számérték helyett csupa 0-t vagy 9-et írnak be). Az így konszolidált adatokból állítják elő az összegző adatokat, lekérdezésekre optimalizált 3NF-nél alacsonyabban normalizált táblázatokat is, melyeket a tranzakciós rendszer kiválasztott adataival együtt tárolnak.

A periodikusan, általában naponta vagy hetente történő tényleges *adatátvitel* két módon valósulhat meg. Az egyszerűbb módszer az, amikor az adatforrásokból mindig minden adatot átvisznek az adattárházba. Mivel azonban igen nagy adatmennyiségek feldolgozásáról van szó, a gyakorlatban az adattárház első feltöltése után a legtöbbször csak inkrementális kiválasztással és átvittel dolgoznak. Ez azt jelenti, hogy az adatforrásokból csak azokat az adatokat veszik tekintetbe, melyek az adattárházba való utolsó átvitel után keletkeztek.

Az adattárházak eredményes használatához hozzátartoznak még a benne lévő információk kinyerését szolgáló eszközök, így a különféle on-line analitikus elemző szoftverek, adatbányászati eszközök és az eredmények grafikus megjelenítésére szolgáló interfészek. Ezek tárgyalása azonban meghaladja ezen könyv kereteit.

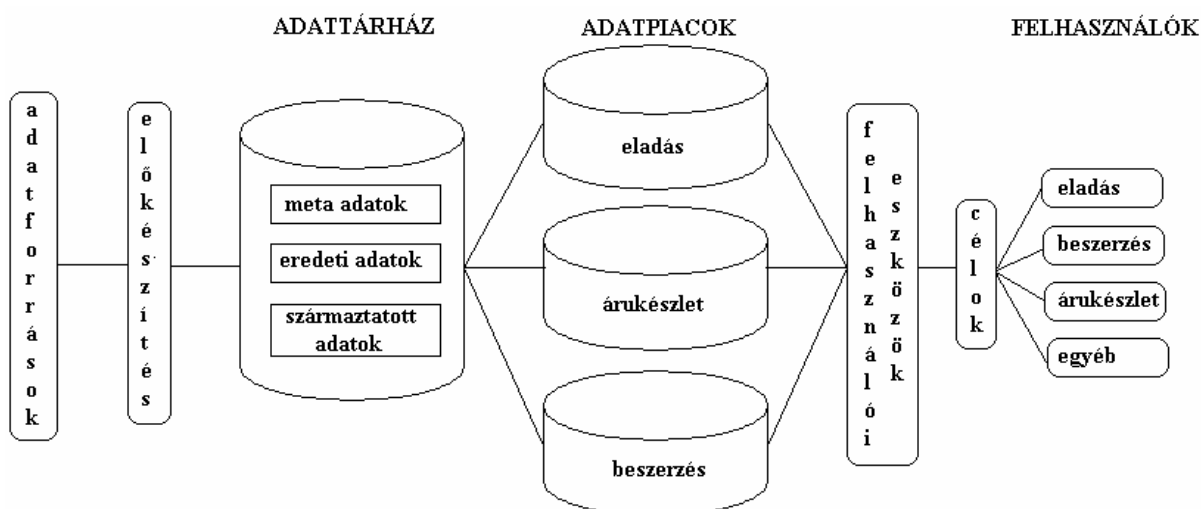
Az adattárház felépítési és működési architektúráját a 9.1 ábrán láthatjuk.



9.1. ábra: Az adattárház felépítési és működési architektúrája.

Az egyszerűbb és gyorsabb használhatóság érdekében a rengeteg adatot tartalmazó adattárházakból gyakran redundánsan duplikálnak olyan részeket, melyek kizárólag egyes alkalmazási területek információs igényeit elégítik ki. Az így kapott „mini adattárházakat” nevezzük *adatpiacnak*. Az adatpiacok már az adattárházból kapják az adatokat, de olyan szerkezetben, összetételben, hogy az optimálisan megfeleljen az adott részterület (pl. eladás, beszerzés) feldolgozási igényeinek. Tartalmazzák a saját szerkezetük leírásának metaadatait is.

Az adatpiacokkal kiegészített adattárház felépítését a 9.2. ábrán mutatjuk be. Az áttekinthetőség kedvéért a 9.1 ábrán már részletezett komponenseket nem tüntettük fel.



9.2. ábra: Adatpiacokkal kiegészített adattárház

9.4. Adatszerkezet

Az adattárház szerkezetét tekintve egy relációs adatbázis az alábbi speciális kiegészítésekkel:

- a táblázatokba csak írni lehet,
- új adattípus került bevezetésre, a dimenzió,
- új relációk közti kapcsolat került bevezetésre, a csillag szerkezet.

9.4.1. Csak beírható táblázatok

A tranzakció orientált adatbázisok táblázatain bármilyen adatkezelő műveletet elvégezhetünk. Így ezek sorait módosíthatjuk, törölhetjük is. Egy ilyen művelet után a rekord eredeti tartalma elveszik a táblázatból. A megmaradó rekordok mindig az éppen aktuális értékeket tartalmazzák. A korábbi, a változtatás előtti értékek legfőljebb korlátozottan, rövid időre visszamenőleg állnak rendelkezésre, vagy állíthatók vissza a tárolt tranzakciókból és a biztonsági mentésekből.

A *csak beírható táblázatokban (insert only table)* a meglevők megtekintésén kívül csak új sorokat írhatunk be. Sorok tartalmának módosítása vagy teljes törlése tilos. (Ez utóbbi alól az elavult adatokat tartalmazó sorok végleges törlése az adatbázisból az egyetlen kivétel.) Az adattárházak adatait kizárólag csak beírható táblázatokban tároljuk. Ezáltal azok rekordjainak minden értéke hosszú távon, az adattárházban tárolandó adatok teljes időintervallumán belül a rendelkezésünkre áll. Hogy az így előállt adathalmazt értelmezni tudjuk, a beírt rekordok mindegyikét kiegészítjük egy *időbélyeggel (timestamp)*, ami megadja, mikor került be a rekord a táblázatba és egy *művelet jelző oszloppal*, ami megmutatja, hogy a beírásakor egy új rekord jött-e létre (C, Create), egy már korábban is meglevő rekord módosított adatait tartalmazza-e (U, Update), vagy már nem is tartozik a táblázatba, mert töröltük onnan (D, Delete). A rekord aktuális értékét a legfrissebb időbélyeghez tartozó sor szolgáltatja. Ha ebben a művelet értéke D, akkor a rekord tartalmilag már megszűnt, nem része a táblázatnak.

A kétféle ábrázolásmód közti különbséget egy példán mutatjuk be. Leegyszerűsített adatbázisunk illetve adattárházunk takarékbetétkönyvek nyilvántartására szolgál és a következő adatokat tartalmazza:

- Tranzakciós adatbázisoknál
 - az érvényben lévő betétkönyv azonosítója, aktuális egyenlege és az utolsó módosítás időpontja,
 - normál tranzakciónál (betét/kivét), hogy az melyik betétkönyvre vonatkozik, mekkora az összege (+ betét, - kivét) és mikor történt,
 - speciális tranzakciónál (új betétkönyv megnyitása, meglevő megszüntetése) tartalmazza a speciális művelet jelzését is.

Itt tehát egy aktív betétkönyvhöz mindig pontosan 1, egy megszüntetethez 0 rekord tartozik.

BETÉTKÖNYVEK			TRANZAKCIÓK			
A	150	07 - 01 - 17	A	100	07 - 01 - 03	új
B	200	07 - 01 - 03	B	200	07 - 01 - 03	új
C	270	07 - 01 - 17	C	300	07 - 01 - 03	új
D	390	07 - 01 - 17	A	+50	07 - 01 - 17	
			C	-30	07 - 01 - 17	
			D	400	07 - 01 - 17	új
			D	-10	07 - 01 - 17	

2007 - 01 - 17 ÁLLAPOT

9.3. ábra: Tranzakciós adatbázis táblázatai és adatai 2007. január 17-én

BETÉTKÖNYVEK			TRANZAKCIÓK			
A	245	07 - 01 - 17	A	100	07 - 01 - 03	új
B	300	07 - 01 - 03	B	200	07 - 01 - 03	új
D	390	07 - 01 - 17	C	300	07 - 01 - 03	új
E	50	07 - 02 - 02	A	+50	07 - 01 - 17	
			C	-30	07 - 01 - 17	
			D	400	07 - 01 - 17	új
			D	-10	07 - 01 - 17	
			C	-270	07 - 02 - 02	megszüntetés
			A	110	07 - 02 - 02	
			E	50	07 - 02 - 02	új
			A	-15	07 - 02 - 02	
			B	100	07 - 02 - 28	

2007 - 02 - 28 ÁLLAPOT

9.4. ábra: Tranzakciós adatbázis táblázatai és adatai 2007. február 28-án

- Adattárházaknál minden betétkönyvre szerepel a rekordban
 - a betétkönyv azonosítója,
 - a módosítás utáni egyenlege,
 - a módosítás ideje,
 - hogyan keletkezett az új rekord (C, U vagy D).

Itt egy betétkönyvhöz, akár élő, akár megszüntetett, legalább egy de több rekord is tartozhat.

BETÉTKÖNYVEK				PÉNZFORGALOM (NAPI)								
A	100	07 - 01 - 03	C	07-01-03	3	0	3	3	0	600	0	+600
B	200	07 - 01 - 03	C	07-01-17	1	0	4	2	2	450	40	+410
C	300	07 - 01 - 03	C									
A	150	07 - 01 - 17	U									
C	270	07 - 01 - 17	U									
D	400	07 - 01 - 17	C									
D	390	07 - 01 - 17	U									

PÉNZFORGALOM (HAVI/ÉVI)							
üres							

PÉNZFORGALOM (HAVI/ÉVI)

üres

BETÉTKÖNYVEK (HAVI/ÉVI)

üres

2007 - 01 - 17 ÁLLAPOT

9.5. ábra: Adattárház táblázatai és adatai 2007. január 17-én

Az adattárházban ezen kívül tároljuk még a tranzakciókat és mivel a pénzforgalomról naponta, havonta és évente, a betétkönyvekről havonta és évente összesítőket kell készítenünk az alábbi aggregátumokat is:

- Pénzforgalmi összesítő:
 - időpont és időtartam jelzése,
 - az adott intervallumban az új és megszüntetett betétkönyvek száma,
 - az adott intervallumban az összes művelet, a befizetések, a kivételek száma, összege, egyenlege.
- Takarékbetétkönyv összesítő:
 - a betétkönyv azonosítója,

- időpont és időtartam jelzése,
- a betétkönyv megnyitásának és megszüntetésének időpontja,
- egyenlege az időszak elején és végén,
- átlagos egyenlege az adott időtartam alatt,
- az adott intervallumban a betétkönyvbe történő összes művelet, a befizetések, a kivétek száma, összege, egyenlege.

Az adattárház várható lekérdezéseit figyelembe véve képeztük ezeket az aggregátumokat és a bennük levő redundanciákat.

BETÉTKÖNYVEK				PÉNZFORGALOM (NAPI)								
A	100	07 - 01 - 03	C	07-01-03	3	0	3	3	0	600	0	+600
B	200	07 - 01 - 03	C	07-01-17	1	0	4	2	2	450	40	+410
C	300	07 - 01 - 03	C	07-02-02	1	1	4	2	2	160	285	-125
A	150	07 - 01 - 17	U	07-02-28	0	0	1	1	0	300	0	+300
C	270	07 - 01 - 17	U									
D	400	07 - 01 - 17	C									
D	390	07 - 01 - 17	U									
C	0	07 - 02 - 02	D									
A	260	07 - 02 - 02	U									
E	50	07 - 02 - 02	C									
A	245	07 - 02 - 02	U									
B	300	07 - 02 - 28	U									

PÉNZFORGALOM (HAVI)												
07-01	4	0	7	5	2	1050	40	+1010				
07-02	1	1	5	3	2	460	285	+175				

BETÉTKÖNYVEK (HAVI)												
A	07-01	07-01-03	null	0	150	112,9	2	2	0	150	0	150
B	07-01	07-01-03	null	0	200	180,6	1	1	0	200	0	200
C	07-01	07-01-03	null	0	270	257,4	2	1	1	300	30	270
D	07-01	07-01-17	null	0	390	176,1	2	1	1	400	10	390
A	07-02	07-01-03	null	150	245	241,6	2	1	1	110	15	95
B	07-02	07-01-03	null	200	300	203,6	1	1	0	100	0	100
C	07-02	07-01-03	07-02-02	270	0	19,3	1	0	1	0	270	-270
D	07-02	07-01-17	null	390	390	390	0	0	0	0	0	0
E	07-02	07-02-02	null	0	50	48,2	1	1	0	50	0	50

PÉNZFORGALOM (ÉVI)												
BETÉTKÖNYVEK (ÉVI)												
üres				2007 - 02 - 28 ÁLLAPOT								

9.6. ábra: Adattárház táblázatait és adatait 2007. február 28-án

A táblázatok tartalmát két különböző időpontban a tranzakciós adatbázisban a 9.3, 9.4., az adattárházban a 9.5 és 9.6 ábrákon mutatjuk be. Azokról a napokról, amikor nem történt semmi, nem készítettünk bejegyzést. Az azonos betétkönyvben ugyanazon a napon történt pénzmozgások sorrendjét a tranzakció órájának, percének ideje alapján meg tudjuk határozni, de ezeket az áttekinthetőség érdekében nem tüntettük fel. A betétkönyv megnyitása automatikusan egy betét elhelyezését, megszüntetése egy kivétet (esetleg 0 összeggel) is jelent.

A tranzakciós adatbázisokban a párhuzamos feldolgozásnál záarakkal (ld. 8.3. alfejezet) kell biztosítanunk az adatok konzisztenciáját. Ez külön adminisztrációt ró az adatbázis-kezelő rendszerre, megnehezíti a párhuzamos munkát és patthelyzetet (ld. 8.4. alfejezet) is létrehozhat. Az adattárházakban záarakra még igen hosszú idő alatt lefutó lekérdezések esetében sincsen szükség. A lekérdezés eredménye mindig a lekérdezés kiadásakor fennálló állapotot tükrözi. Ha a lekérdezés végrehajtása alatt kerültek be új rekordok a táblázatba, ezek a bennük lévő időbélyeg miatt nem fognak az eredménybe bekerülni. Még akkor sem, ha a fizikai kiválasztás pillanatában már bent vannak az adattárházban.

9.4.2. Csillag elrendezés

Az adattárházak és adatpiacok adatmodellje a *csillag elrendezésre (Star schema)* épül. Ennek a középpontjában áll a minden elemi adatot és azok kereshetőösszegzései, származtatott adatait tartalmazó *ténytáblázat (Fact table)*. Ennek elsődleges kulcsa az összes különböző lekérdezési szempont konkaténációja, adatai pedig ezen szempontok minden kombinációjához tartozó elemi és összegzett érték. A tény-táblázat normalizált. (ld. 4.6.4. pont).

A központi ténytáblázathoz kapcsolódnak idegen kulccsal a *dimenziók (dimension)*, melyek a különböző lekérdezési szempontokat írják le. Ezek nem normalizált táblázatok. Legtöbbször egy (vagy több) hierarchiát valósítanak meg. Elemei a vizsgálni kívánt szempont elemei és ezeknek a hierarchia különböző szintjein elhelyezkedő értékei.

Grafikusan ábrázolva a ténytáblázatot és az azt körbevevő dimenziókat egy csillagszerű alakzatot kapunk (ld. 9.7. ábra). Innen származik az elrendezés elnevezése.

A fenti, nehezen átlátható definíciót egy egyszerű példával tesszük érthetővé.

Adattárházunk egy nemzetközi hipermarket lánc eladásait tartalmazza. Ezeket három szempont, idő, eladási hely és termék szerint akarjuk elemezni.

A tény-táblázat elemei az eladások, melyek az eladás dátumát, helyét, valamint az eladott termék mennyiségét, eladási értékét és a rávetített eladási költséget tartalmazzák. A legegyszerűbb elemi érték, amit be akarunk vonni az elemzésbe egy adott terméknek egy adott áruházban a napi forgalma. Ezért az adattárház feltöltésekor (ld. 9.3. alfejezet) az áruházak produkciós adatbázisából nem az egyes eladások adatait, hanem ezeknek termékek szerinti napi összegzését vesszük be az adattárházba. A napi áruházi termék-eladások mellett vizsgálni akarjuk a forgalmat heti, havi, negyedéves és éves bontásban, áruajták és árucsoportok (pl. élelmiszerek, ruházati cikkek, híradástechnikai eszközök, stb.) szerint, valamint az áruházak területi eloszlása (város/megye, ország) alapján és bármely szempont szerint annak minden előfordulására összegezve is. (Tejes időszak, minden termék és minden ország).

Mindegyik vizsgálati szempont egy dimenzió lesz. Az egyes dimenziók elemei és jellemzőik:

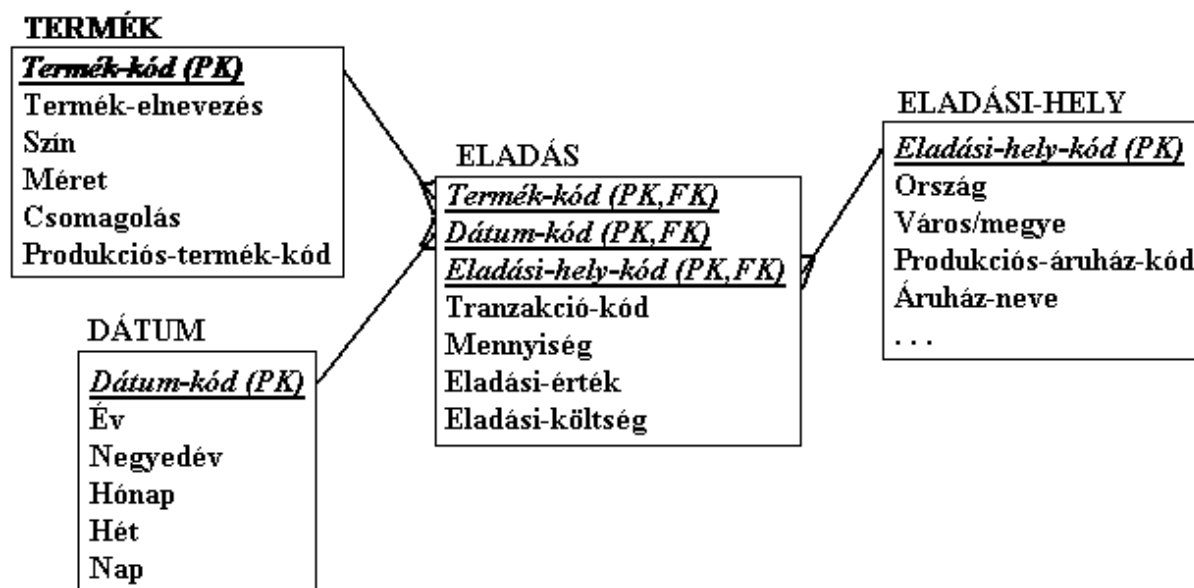
- Dátum dimenzió: teljes tárolt idő, évek, negyedévek, hónapok, hetek, napok.
- Termék dimenzió: minden termék, termékcsoportok, egyedi termékek.
- Eladási hely dimenzió: minden áruház, országok, városok/megyék, áruház.

Az adattárház logikai felépítését a 9.7. ábrán, a ténytáblázat néhány önkényesen kiválasztott adatát a 9.8 ábrán, az egyes dimenziókat pedig a 9.9., 9.10. és a 9.11. ábrán láthatjuk.

PK, Primary Key, elsődleges kulcs (része), *FK*, Foreign Key, idegen kulcs.

A tény-táblázatnak eleme például a budapesti Csillag áruházban (Eladási-hely-kód: 0000001) 2005.01.04-én (Dátum-kód: D050010004) eladott 48-as méretű, kék Prince melegítők (Termék-kód: RSM0010) darabszáma, eladási összértéke és eladási költségeinek összértéke (ld. a 9.8. ábra első sorát). Ugyancsak elemei az ebből a termékből ezen a napon

az egész világon (Eladási-hely-kód: XXXXXXXX), valamint a teljes elemzési időszakban (Dátum-kód: XXXXXXXXXXXX) Budapesten (Eladási-hely-kód: 0000100) eladott darabszámok és értékek (3. és 9. sor).



9.7. ábra: Csillag elrendezés logikai felépítése.

ELADÁS						
Termék-kód	Dátum-kód	Eladási-hely-kód	Tranzakció-kód	Mennyiség	Eladási-érték	Eladási-költség
RSM0010	D050010001	0000001	123456	21	2100	1800
RSM0010	D050010001	0000002	123458	14	1400	1000
RSM0010	D050010001	XXXXXXX		1200	129000	105550
RSM0002	D050010002	0000001	123578	5	1000	600
RSM0010	W050000100	0000100		77	7700	6800
RSM0002	M050010000	0000100		97	20000	12000
R123400	M050010000	0000001		88	4400	2200
RSM0002	Q051000000	0000002		103	21000	13200
RSM0010	XXXXXXXXXX	0000100		100	10000	8800
RSM0010	XXXXXXXXXX	XXXXXXXX		4567	456700	401500
RSM0002	XXXXXXXXXX	XXXXXXXX		4321	48000	32600
R123400	XXXXXXXXXX	XXXXXXXX		1000	50000	25000

9.8. ábra: Az ELADÁS tény táblázat néhány önkényesen kiválasztott adata.

Az egyes dimenziók azonosítására elsődleges kulcsként mesterségesen képzett kódot használunk. Az entitás esetleg létező természetes azonosítója, produkciós adatbázis-béli elsődleges kulcsa a dimenzió egy attribútuma. Ennek az az oka, hogy a természetes azonosítók bár ritkán, de időnként mégis megváltozhatnak. Ezt a változást azonban a csak beírható tény táblázattól, melyek a dimenziók elsődleges kulcsaira idegen kulccsal hivatkoznak, távol akarjuk tartani.

DÁTUM

<i>Dátum-kód</i>	<i>Év</i>	<i>Negyedév</i>	<i>Hónap</i>	<i>Hét</i>	<i>Nap</i>
D050010001	2005		01		01
D050010002	2005		01		02
W050000100	2005			01	
M050010000	2005		01		
Q051000000	2005	1			
XXXXXXXXXX	minden				

9.9. ábra: A **DÁTUM** dimenzió néhány önkényesen kiválasztott adata.

ELADÁSI-HELY

<i>Eladási-hely-kód</i>	<i>Ország</i>	<i>Város/megye</i>	<i>Produktions- áruház-kód</i>	<i>Áruház-név</i>
0000001	Magyar	Budapest	ABC001	Csillag
0000002	Magyar	Budapest	X1320D	Otthon
0000100	Magyar	Budapest		
0000200	Magyar	Pécs		
0010000	Magyar			
0000090	Német	Berlin	89MMN0	Stern
0001700	Német	München		
XXXXXXX	minden			

9.10. ábra: Az **ELADÁSI-HELY** dimenzió néhány önkényesen kiválasztott adata.

TERMÉK

<i>Termék-kód</i>	<i>Termék-elnevezés</i>	<i>Szín</i>	<i>Méret</i>	<i>Csomagolás</i>	<i>Produktions-kód</i>
RSM0010	Prince melegítő	kék	48	nincs	XM1234
RSM0002	Adidas melegítő	piros	56	doboz	XM2132
RST0010	Puma trikó	fehér	44	doboz	XT0012
RST0011	Puma trikó	kék	44	doboz	XT0013
RSM0003	melegítő		46		
RSMK004	melegítő		48		
RST0001	trikó		44		
RS00013	sport ruha				
R123400	ruha				
E201000	zöldség				
EK02000	kenyér				
EKB0001	saját kenyér	barna	1	fólia	LZT768012
E010000	élelmiszer				
XXXXXXX	minden				

9.11. ábra: A **TERMÉK** dimenzió néhány önkényesen kiválasztott adata

Az adattárházból az információk zömét a tény táblázat és a dimenziók egyesítésével SQL lekérdezésekkel (join, ld. 5.6.3.2. pont) vagy a kifejezetten erre a célra kifejlesztett szoftver eszközök segítségével kapjuk meg.

Az adatpiacok és a nem nagyon komplex adattárházak felépítését jól írja le a csillag elrendezésű adatmodell. Bonyolult adatszerkezetek is modellezhetők több tény táblázattal és az ezekhez tartozó egyedi és/vagy közös dimenziókkal. (Az idő/dátum mindig közös dimenzió). A dimenziókon belüli hierarchiát is beépíthetjük a modellbe. Az így kibővített ábrázolási módot hívjuk hópehely (*Snowflake*) modellnek.

9.5. Adatbányászat

Az adattárházak felhasználásának egyik leggyorsabban fejlődő területe az *adatbányászat (Data Mining)*. Az adatbányászat célja az, hogy miként lehet nagy adatbázisokban

- rejtett tudást,
- új összefüggéseket,
- eddig nem ismert szabályokat,
- nem várt mintákat felfedezni.

Ha pontosan tudjuk, hogy mit keresünk, akkor használhatjuk az SQL nyelvet is az adattárházban való keresésre. Tipikus példák erre:

- Ki, mikor, milyen terméket vásárolt?
- Mekkora az átlagos forgalom egy régióban júliusban?

Ha bizonytalanok vagyunk, mit is keresünk, próbálkozások sokaságával akkor is megkaphatjuk a választ sorozatosan módosított SQL lekérdezésekkel. Ehelyett azonban célszerűbb adatbányászati módszereket alkalmazni, mert ezek segítségével az eredményt automatikusan megkapjuk.

Igen nagy szerepe van az adatbányászatnak a marketingben. Közhely, hogy a reklámköltségek legalább 80 %-a kidobott pénz. De sajnos nem tudjuk, melyik az a 20 %, amit érdemes kiadnunk erre a célra. Egy luxus cikket forgalmazó cég nyilvánvalóan nem fogja elküldeni a drága katalógusát a lakótelepi panel lakásokba. Inkább a jómódú villanegyedek lakóira fog koncentrálni. De egy diszkont áruház lánc közepes és jobb minőségű termékeinél már nem ilyen egyértelmű, hogy hová célszerű a szóróanyagokat koncentrálni, honnan jönnek, vagy honnan jöhetnek a legtöbbet vásárló vevők. Adatbányászati módszerekkel erre választ tudunk kapni.

További tipikus adatbányászati kérdések:

- Melyek a legfontosabb ügyfél típusok?
- Melyek a legfontosabb ügyfélvásárlási szokások?
- Miért maradnak vagy lépnek ki a dolgozók a vállalattól?
- Mikor és milyen áron érdemes egy új terméket a piacra dobni?

Jelentős szakirodalma van és számos adatbányászati szoftver eszköz is támogatja azokat a módszereket, amelyek segítségével az ilyen és ezekhez hasonló, nehezen algoritmizálható kérdésekre válaszolni tudunk. De ne felejtsük el, hogy a gyakorlati szakemberek szerint az adatbányászatban a munka 80 százalékát az adatok előállítására fordítjuk és csak 20 százalékát teszi ki a tényleges elemzés.

9.6. Ellenőrző kérdések

1. Mik a legfőbb különbségek a tranzakció orientált adatbázisok és az adattárházak között?
2. Mit tartalmaznak a metaadatok?
3. Miért tárolnak az adattárházban összevont adatokat?
4. Miért nem okoz problémát az adattárházakban és az adatpiacokon a nagyfokú redundancia és a harmadik normál forma hiánya?
5. Az idő miért szerepel mindig az adattárházak és adatpiacok dimenziói között?

10. IRODALOMJEGYZÉK

Könyvünket általános összefoglaló tankönyvnek szántuk. Ezért szándékosan nem adtunk meg konkrét irodalmi utalásokat egyes témakörökre, részterületekre. Ugyanakkor tisztában vagyunk vele, hogy egy könyv sem tartalmazhat mindent és olyan jól, ahogyan azt a szerzők meg szeretnék volna írni, és amit az egyes olvasók tudni kívánnak a különböző témákról. Ezért az alábbiakban felsorolunk néhány olyan könyvet, melyek hasznos segítséget nyújthatnak az adatbázisokkal való munka még részletesebb, vagy más szempontok szerint történő megismerésére.

A relációs modellben, 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” reláció.

Access 2003 súgó és útmutató

<http://office.microsoft.com/hu-hu/access/FX100646921038.aspx>

Adrians, P. – Zantige, D.: Adatbányászat, Budapest, 2002

Czenky Márta: Adatmodellezés SQL és Access alkalmazás SQL Server és ADO, ComputerBooks, Budapest, 2005

Date, J. C.: An Introduction to Database Systems, Addison Wesley, 2004

Denne, N.: DB2, Theorie und Praxis, DGD, Wiesbaden, 2001

Dr. Szelezsán: Adatbázisok LSI, Budapest, 2000)

Halassy B.: Az adatbázis-tervezés alapjai és titkai, IDG, Budapest, 1994

Halassy B.: Ember, Információ, rendszer, IDG, Budapest, 1996

Halassy, B.: Adatmodellezés, Nemzeti Tankönyvkiadó, Budapest, 2002

Hoffer, J.A – Prescott, M. B. – McFadden, F. R.: Modern Database Management, Prentice Hall, New Jersey, 2002

Inmon, W.H, - Welch, J.D. – Glassey, K.L.: Managing the Data Warehouse, John Wiley and Sons, 1997

Kovács, Kovácsné, Ozsváth: Adatkezelés az MS Access 7.0 használatával , ComputerBooks, Budapest, 1997

Marakas, G.M.: Modern Data Warehousing and Mining, Prentice Hall, New Jersey, 2003

Meloni, J. C.: Tanuljuk meg a MySQL használatát 24 óra alatt, Kiskapu, Budapest, 2003

MySQL Documnetation <http://dev.mysql.com/doc/>

Pétery Kristóf: Microsoft Office Access 2003 – Lekérdezéstől testre szabásig (magyar változat), Mercator Stúdió, Szentendre, 2005

Quittner P. – Kotsis D.: Számítástechnika rendszerszervezőknek, Akadémiai Kiadó, Budapest, 1989

Quittner P.: Adatbázis-kezelés a gyakorlatban, Akadémiai Kiadó, Budapest, 1993

Quittner P.: ORACLE, www.uni-corvinus.hu/~qp/Oracle/ORACLE_osszefoglalo.html

Quittner P: ORACLE gyakorlat,

www.uni-corvinus.hu/~qp/Oracle/Gyakorlat/Oracle_gyakorlat.ppt

Quittner, P.: A Practical Approach to Database Systems, Akadémiai Kiadó, Budapest, 1994

Ullman, J. - Widom, J.: A First Course in Database systems, Prentice Hall, New Jersey, 2002

Ullman, Widom: Adatbázisrendszerek, Panem, Budapest, 1998)

Watson, R.T.: Data Management, Databases and Organizations, John Wiley and Sons, 1999

Zandstra, M.: Tanuljuk meg a PHP5 használatát 24 óra alatt, Kiskapu, Budapest, 2005

11. TÁRGYMUTATÓ

\$

\$_REQUEST, 6-49

A,Á

ACCESS, 6-1
adat, 1-1
adat többszörözés, 1-25
adatátvitel, 9-3
adattányaszt, 9-1, 9-11
adattábazis, 1-18
adattábazis-felügyelő, 1-17
adattábazis-kezelő rendszer, 1-18
adatbevitel, 2-10, 2-15, 4-11, 6-17
adatelérés, 2-7
adat-felügyelő, 1-17
adatfüggetlenség, 1-7
adatkezelő nyelv, 1-14
adatkezelő utasítás, 5-50, 6-17
adatlap nézet, 6-17
adatlap űrlap, 6-35
adattíró nyelv, 1-13
adattíró utasítás, 5-30
adatmodell, 3-1
adatt módosítás, 6-18
adattpiac, 9-2
adatt szerkezet, 9-4
adatt szervezés, 2-1
 index, 2-11, 2-23
 közvetlen elérés, 2-11
 lineáris, 2-8
 soros, 2-8, 2-22
 szekvenciális, 2-8, 2-22
adatt szótár, 1-15, 7-1
adatt tárház, 9-1
adatt tárház architektúra, 9-3
adatt árolás, 2-1
adattípus, 4-2, 5-18, 6-5, 6-50
adatt ömörítés, 2-27
adatt védelem, 6-45
add_month(), 5-27, 6-12, 6-51
alapértelmezett érték, 4-2, 5-32, 6-7
alkalmazási program, 1-3
all, 5-8
all jogosultság, 7-5
állandó, 5-6, 6-8
általános rendszer jogosultságok katalógusa, 7-5
alter, 5-42, 6-52
alter jogosultság, 7-5
alter sequence, 5-30
alter table, 5-30, 5-42, 6-8, 6-34
alter trigger, 5-30
alter view, 6-52
alternatív kulcs, 4-7
alternative key, 4-7
and, 6-20
anomália, 4-32, 9-2
any, 5-8
architektúra, 1-6
argumentum, 5-7

aritmetikai operátor, 5-7, 6-9, 6-11
asc, 5-57
átlag, 5-23, 6-12, 6-50
átnevezés, 4-14
attribútum, 3-1, 3-4
auto_increment, 6-52
AutoJelentés, 6-40
AutoŰrlap, 6-35
avg(), 5-23, 6-12, 6-24, 6-50
azonosító, 3-5, 4-3, 5-4

B

B+ fá, 2-13
bájt, 6-6
beágyazott szelekt, 5-59, 5-68, 6-31, 6-33
beírási anomália, 4-32
belső join, 4-17, 5-55, 6-22
belső szint, 1-12
between, 5-8, 6-20
bevitel, 5-68
beviteli maszk, 6-7
bitmap index, 2-16
bit-térképes index, 2-16
biztonság, 6-45
boolean, 6-5
Boyce-Codd normál forma, 4-40
bővítés, 4-23

C

candidate key, 4-7
cbyte(), 6-13
cdate(), 6-13
cdbl(), 6-13
cdec(), 6-13
char, 5-19, 6-50
char_length(), 6-50
ciklus, 6-48
cim, 6-7
címké varázsló, 6-41
cint(), 6-13
close, 5-81, 5-84
coalesce(), 5-27, 6-51
commit, 5-71, 6-18
concat(), 5-25, 6-12, 6-50
constraint, 5-32, 6-7, 6-8
count(), 5-23, 6-12, 6-24, 6-50
create, 4-24, 5-30, 6-52, 9-5
create index, 5-30, 5-39
create sequence, 5-30
create synonym, 5-30
create table, 5-30, 6-3, 6-34
create trigger, 5-30
create view, 5-30, 5-36, 6-20, 6-52
curdate(), 6-51
current date(), 5-27, 6-13, 6-51
current timestamp(), 5-27, 6-13, 6-51
curtime(), 6-51

Cs

csak beírható, 9-5
csatolás, 6-14, 6-46
csatolt űrlap, 6-36
csere(), 6-12
csillag elrendezés, 9-8
csillag szerkezet, 9-4
csoport függvény, 5-23, 6-12, 6-50
csoportfej, 6-41
csoportképzés, 4-23
csoportláb, 6-41
csoportosítás, 6-24
csoportosítási beállítás, 6-42
cstr(), 6-13
csúcsérték, 6-23

D

darab, 5-23, 6-12, 6-50
Data Control Language, 1-14, 5-47, 5-71
Data Definition Language, 1-13, 5-30
Data Dictionary, 7-1
Data Manipulation Language, 1-14, 5-50
Data Mart, 9-2
Data Mining, 9-1, 9-11
Data Warehouse, 9-1
date, 6-5, 6-50
date(), 5-27, 6-10, 6-13
date/time, 5-21
date_add(), 6-51
date_format(), 6-51
dateadd(), 6-12
datetime, 6-50
dátum, 6-5
dátum függvény, 6-12, 6-51
dátum/időpont, 5-21
dátum/időpont függvény, 5-26
day(), 5-27, 6-13, 6-51
DCL, 1-14, 5-47, 5-71
DDL, 1-13, 5-30
dec, 6-50
decimal, 6-50
decimal(), 5-27, 6-13, 6-51
decimális, 6-6
declare, 5-81
default, 4-2, 5-32, 6-7
dekomponálás, 4-33, 4-38
dekompozíció, 4-33, 4-38
delete, 5-64, 6-18, 6-32, 6-34, 6-52, 9-5
delete jogosultság, 7-5
desc, 5-57
Descartes szorzat. *Lásd* keresztszorzat
determináns, 4-27
diagram varázsló, 6-41
die(), 6-49, 6-51
difference, 4-22
dimenzió, 9-4, 9-8
dinamikus utasítás, 5-74
distinct, 5-24, 5-53, 6-32
DML, 1-14, 5-50
domain, 4-2
double precision, 6-50
drop, 5-41, 6-52
drop index, 5-30
drop sequence, 5-30
drop synonym, 5-30

drop table, 5-30, 6-8, 6-34
drop trigger, 5-30
drop view, 5-30
dupla, 6-6
duplikált index, 6-14

E,É

egész, 6-6
egy felhasználós rendszer, 1-20
egyéb függvény, 5-27, 6-51
egyed, 5-14
egyedi index, 6-14, 6-52
egyedtipus, 3-1, 3-4
egy-egy kapcsolat, 6-16
egyesítés, 4-17
egyesítő lekérdezés, 6-33
egyszeres, 6-6
egy-több kapcsolat, 4-9, 6-16
együtt tartás, 6-44
elágazás, 6-48
elemi tényadat, 1-2
elérési mód, 1-12
elhelyezési kulcs, 2-7, 4-12
első normál forma, 4-28
elsődleges kulcs, 2-7, 3-5, 4-3, 4-7, 6-4, 6-10, 9-8
elvesztett módosítás, 8-2
end-exec, 5-76
entitás, 3-1, 3-4
eredményazonosító, 6-52
érvényesítési szöveg, 6-7
érvényességi szabály, 6-7
és, 6-20
exec sql, 5-76
expression, 6-24
exsists, 5-8
extend, 4-23

F

Fact table, 9-8
felhasználó, 1-4
felhasználói interfész, 1-15
felhasználói szintű védelem, 6-45
felirat, 6-38
feljegyzés, 6-5
feltétel, 5-7, 5-32, 6-7, 6-8, 6-46
fenntartott szavak, 5-4
fetch, 5-81, 5-83
first(), 6-24
fizikai szint, 1-12, 2-1
float, 6-50
for update, 5-57
Foreign Key, 4-8, 9-8
format(), 6-51
formátum, 6-6, 6-18
formázás, 6-18
frissítő lekérdezés, 6-30
from, 5-54
front-end, back-end, 6-46
full join, 5-55
funkcionális függés, 4-26
funkcionális teljes függés, 4-28
futtatás, 6-22
függőség, 4-26
függőségi diagram, 4-27
függőségi kapcsolat, 4-8

függőségi katalógus, 7-6
függvény, 5-23, 6-8, 6-12

G

globális tömb, 6-49
grafikus interfész, 1-15, 6-1
grant, 5-30, 5-47, 6-52
group by, 4-23, 5-56, 6-24

H

halmaz operátor, 5-12
hálós adatmodell, 3-2
hardver, 1-2
harmadik normál forma, 4-35
hashing, 2-19, 2-23
határoló jel, 5-13
having, 5-56
helyettesítő karakter. *Lásd* joker karakter
hierarchikus adatmodell, 3-2
hiperhivatkozás, 6-5
hivatkozási feltétel, 4-8
hivatkozási integritás, 6-16
hivatkozó reláció, 4-8
hivatkozott reláció, 4-8
hópehely modell, 9-10
horizontális adatmegosztás, 1-25
host változó, 5-73, 5-79, 6-49
hosszú adat, 5-22
hosszú egész, 6-6
hozzáférési idő, 2-5
hozzáférési jogok katalógusa, 7-5
hozzáfűző lekérdezés, 6-31
HTML, 6-47

I, Í

idegen kulcs, 4-8, 6-15, 6-46, 9-8
idő, 6-5
idő függvény, 6-12, 6-51
időbélyeg, 9-5
ifnull(), 6-51
igen/nem, 6-5
illesztési tulajdonságok, 6-27
importálás, 6-13, 6-46
in, 5-8
index, 2-11, 2-23, 6-8, 6-14, 6-52
index jogosultság, 7-5
index katalógus, 7-4
index létrehozása, 5-39
indexkomponenes katalógus, 7-4
indikátor, 5-80
információ, 1-1
információs rendszer, 1-1
initcap(), 5-25, 6-12, 6-50
inkonzisztens adat, 8-3
inner join, 4-17, 5-55
insert, 5-68, 6-17, 6-31, 6-34, 6-52
insert jogosultság, 7-5
insert only table, 9-5
int, 6-50
integer, 6-5, 6-50
integritás, 4-6
intersect, 4-22, 5-12, 5-56, 6-33
irányított szűrés, 6-19

is null, 5-8
ismétlés, 6-48

J

jelentés, 6-40
jelentés gyorsvarázsló, 6-40
jelentés tervezése, 6-41
jelentés varázsló, 6-40
jelentésfej, 6-41
jelentésláb, 6-41
jelölőlapka, 6-5, 6-39
jelszó, 6-45
jogosultság, 5-47
jogosultság kezelése, 5-47
join, 4-17, 4-20, 5-55, 6-22, 6-27
join-index, 2-18
joker karakter, 5-10, 6-11, 6-50

K

kapcsolat, 3-1, 3-5, 4-8, 6-15
kapcsolat törlése, 6-16
kapcsolatazonosító, 6-49
kapcsolatok szerkesztése, 6-16
karakter, 5-19
karakter függvény, 5-25, 6-12, 6-50
kardinalitás, 4-2
kaszád, 4-12, 6-16, 6-18, 6-32
kaszád törlés, 4-12
katalógus, 1-15, 7-1
kerek(), 6-12, 6-51
kerekítés, 6-12, 6-51
keres, 5-25
keresés varázsló, 6-5
keresési kulcs, 2-7, 4-13
keresztsszorzat, 4-16
keresztstábla, 6-29
keresztstáblás lekérdezés, 6-29
kifejezés, 5-17, 6-23, 6-30
kifejezés-szerkesztő, 6-8, 6-19, 6-23
kijelölés, 4-24
kiválasztás, 5-51
kizárólagos mód, 6-46
kizárólagos zár, 8-5
kliens-szerver architektúra, 1-21
kódtáblázat, 2-27
kombinált adatmegosztás, 1-26
koncepcionális szint, 1-12
konvertáló függvény, 5-26, 6-13, 6-51
korlátozás, 4-15
korlátozott törlés, 4-11
közvetlen elérés, 2-11
kulcs, 2-7, 4-6
 alternatív kulcs, 4-7
 elhelyezési kulcs, 4-12
 elsődleges kulcs, 3-5, 4-3, 4-7, 6-4, 9-8
 idegen kulcs, 4-8, 6-15, 6-46, 9-8
 keresési kulcs, 4-13
 sűrűsödési kulcs, 4-12
kurzor, 5-74, 5-81, 6-52
különbség, 4-22
külső adatok átvétele, 6-13
külső join, 4-20, 5-55, 6-27
külső szint, 1-12

L

lapszintű zárolás, 6-46
last(), 6-24
last_day(), 5-27, 6-12, 6-51
lcase(), 6-12, 6-50
left join, 5-55
lehetséges kulcs, 4-7
lekérdezés, 4-6, 6-20
 beágyazott szelekt, 6-33
 egyesítő lekérdezés, 6-33
 frissítő lekérdezés, 6-30
 hozzáfűző lekérdezés, 6-31
 keresztáblás lekérdezés, 6-29
 összesítő lekérdezés, 6-24
 paraméteres lekérdezés, 6-25
 ábrakészítő lekérdezés, 6-29
 törlő lekérdezés, 6-32
 választó lekérdezés, 6-21
lekérdezés tervező, 6-21
lemez felépítése, 2-3
len(), 6-12
length(), 5-25, 6-12, 6-50
levédés, 6-49
like, 5-8, 6-11, 6-50
lineáris adatszervezés, 2-8
literál. *Lásd* állandó
logikai munkaegetység, 5-72
logikai operátor, 5-11, 6-9, 6-11
logikai szint, 1-12
long, 6-5
longtext, 6-50
lower(), 5-25, 6-12, 6-50
ltrim(), 5-25, 6-12
LTrim(), 6-51

M

magasabb normál formák, 4-40
második normál forma, 4-31
maszk karakter. *Lásd* joker karakter
materializált nézet, 4-6
materialized view, 4-6, 5-15
max(), 5-23, 6-12, 6-24, 6-50
maximum, 5-23, 6-12, 6-50
megjelenítés, 6-21
mentés, 6-8
metaadat, 1-15, 9-3
metszet, 4-22, 6-33
mező, 1-2, 2-7, 6-21
mező átnevezése, 6-3
mezőméret, 6-5
Microsoft Word, 6-41
mid(), 6-12, 6-51
min(), 5-23, 6-12, 6-24, 6-50
minimum, 5-23, 6-12, 6-50
minus, 5-12, 5-56
módosítás, 2-10, 4-12, 5-65, 6-18
módosítási anomália, 4-32
módosítási zár, 8-5
month(), 5-27, 6-13, 6-51
munkacsoport információs fájl, 6-46
mysql_close(), 6-49, 6-51
mysql_connect(), 6-49, 6-51
mysql_error(), 6-49, 6-51
mysql_fetch_row(), 6-52, 6-53
mysql_num_rows(), 6-52

mysql_query(), 6-49, 6-52
mysql_real_escape_string(), 6-49, 6-51
mysql_select_db(), 6-49, 6-51

N

natural join, 4-17, 5-55
negyedik normál forma, 4-42
nem véglegesített adat, 8-3
nem végrehajtható utasítás, 5-74
nézet, 4-6, 4-39, 5-15, 6-20, 6-52
nézet létrehozása, 5-36
normalizáció, 4-25
normalizálás, 4-25, 4-28, 4-31, 4-35, 4-40, 4-42, 9-2
NULL érték, 4-2, 5-16, 6-7
NULL értékre állítás, 4-11
number, 6-5
numerikus adattípus, 5-20
numerikus függvény, 5-25
nvl(), 5-27, 6-51

Ny

nyomtatási kép, 6-41

O,Ó

objektum, 5-13, 6-2
objektum létrehozása, 5-30
objektum megszüntetése, 5-41
objektum módosítása, 5-42
objektum törlése, 6-8
objektumokra való jogosultságok katalógusa, 7-5
oldalbeállítás, 6-41
oldalfej, 6-38, 6-41
oldalláb, 6-38, 6-41
oldaltörés, 6-44
OLE objektum, 6-5
open, 5-81, 5-82
operációs rendszer, 1-2
operandus, 5-7
operátor, 5-7, 5-8, 5-11, 5-12, 6-8, 6-11, 6-50
optimális elérési út, 2-25
optimista zárolás, 6-46
or, 6-20
order by, 5-56, 6-19
oszlop, 5-14
oszlop függvény, 5-24
oszlop katalógus, 7-3
oszlopdiaqram, 6-44
oszlopféjléc, 6-29
oszlopos űrlap, 6-35
osztott adatbázis, 1-22, 7-7
osztott zár, 8-5
outer join, 4-20, 5-55

Ö,Ő

összeg, 5-23, 6-12, 6-50
összehasonlító operátor, 5-8, 6-9, 6-11
összekapcsolás, 4-17
összesítés, 6-24
összesítési beállítás, 6-42
összesítő lekérdezés, 6-24
összetett index, 2-25, 6-15
összetett űrlap, 6-35

P

paraméter, 6-25
paraméteres lekérdezés, 6-25
párhuzamos feldolgozás, 8-2
páthelyzet, 8-8
pénznem, 6-5
pesszimista zárolás, 6-47
PHP, 6-47
PHP-MySQL, 6-47
pillanatfelvétel, 4-6
pointer, 2-9
precedencia, 6-11
prekompiler, 5-75
Primary Key. *Lásd* elsődleges kulcs
print(), 6-48
programba ágyazás, 5-73, 6-47
programba beépíthető rendszer, 1-3
programok használati joga katalógus, 7-6
programozási nyelvhez illesztés, 1-14
project, 4-15
projekció, 4-15
pszeudooszlop, 5-22
public, 7-5

R

real, 6-50
references jogosultság, 7-5
rekord, 1-2, 2-7
rekordazonosító, 2-4
rekordszintű zárolás, 6-46
reláció, 4-1
reláció foka, 4-1
reláció fokszáma, 4-1
relációs adatmodell, 3-3
relációs művelet, 4-13
rename, 4-14
rendezés, 2-28, 6-18
rendezés és csoportosítás, 6-44
rendszergazda engedély, 6-46
replace(), 5-25, 6-12, 6-51
restrict, 4-15
restrikció, 4-15
revoke, 5-30, 5-47, 6-52
right join, 5-55
rollback, 5-71, 6-18
round(), 5-27, 6-12, 6-51
rowid, 2-4, 5-22
rtrim(), 5-25, 6-12
RTrim(), 6-51

S

savepoint, 5-72
schema, 5-13, 5-14
segédürlap, 6-36
select, 5-51, 6-20, 6-52
select jogosultság, 7-5
séma, 5-13, 5-14
sequence, 5-15
smallint, 6-50
snapshot, 4-6
Snowflake, 9-10
some, 5-8
sor, 5-14

sorazonosító, 5-22
sorfejléc, 6-29
soros adatszerzés, 2-8, 2-22
sorszám generátor, 5-15
SQL, 6-32
SQL előfeldolgozó, 5-75
SQL kifejezés, 5-17
SQL kommunikációs terület, 5-73
SQL optimalizáló, 2-25, 7-1, 7-8
SQL utasítás, 5-29
sql_query(), 6-52
sqlcode, 5-76
sqlstate, 5-76
Star schema, 9-8
statikus utasítás, 5-74
statisztika, 7-8
str_to_date(), 6-51
strconv(), 6-12
stripslashes(), 6-52, 6-53
substr(), 5-25, 6-12, 6-51
sum(), 5-23, 6-12, 6-24, 6-50
summarize, 4-23
sűrűsödés, 2-21
sűrűsödési kulcs, 4-12
sysdate(), 5-27, 6-13, 6-51

Sz

szám, 6-5
számláló, 6-5
számosság. *Lásd* kardinalitás
származtatott adat, 1-2
szekvenciális adatszerzés, 2-8, 2-22
szelektlista, 5-54
szervezési mód, 1-12
szinkronizálás, 6-46
szinonima, 2-20
szinonima szótár, 7-4
szinonima táblázat, 7-7
szintaxis, 5-2
szoftver, 1-2
szolgáltató program, 1-16
szöveg, 6-5
szöveges információ-visszakereső rendszer, 1-6
szöveghossz, 5-25
szövegösszefűzés, 5-25, 6-11, 6-12, 6-50
szűrés, 6-18
szűrés kijelöléssel, 6-19
szűrés kizárással, 6-19
szűrés úrlappal, 6-19

T

tábla, 6-21
tábla megjelenítése, 6-15
tábla módosítása, 5-42
tábla varázsló, 6-3
táblafeltétel, 6-10
táblakészítő lekérdezés, 6-29
táblázat, 4-5, 5-14, 6-3
táblázat katalógus, 7-3
táblázat létrehozása, 5-30, 6-3
táblázat módosítása, 6-8
táblázat törlése, 6-8
table, 4-5, 5-14
tárolási mód, 1-12
tárolt nézet, 5-15

tény adatbázis, 1-6
ténytáblázat, 9-8
természetes join, 4-17, 5-55, 6-22
time(), 6-13
times, 4-16
timestamp, 6-50, 9-5
tizedeshelyek, 6-6
to_char(), 5-27, 6-13, 6-51
to_date(), 5-27, 6-13, 6-51
to_number(), 5-27, 6-13, 6-51
tovagyűrűző törlés. *Lásd* kaszkád
több felhasználó, 6-46
többértékű funkcionális függőség, 4-28
törlés, 2-10, 4-11, 4-12, 5-64, 6-8, 6-18
törlési anomália, 4-32
törlő lekérdezés, 6-32
translate(), 5-25, 6-12, 6-51
tranzakció, 5-72, 8-1
tranzakciós adatbázis, 9-2
tranzakcióvezérlő utasítás, 5-71
trigger, 5-15, 6-52
trunc(), 5-27, 6-13, 6-51
truncate(), 6-51
tulajdonság, 3-1, 3-4, 5-14
tulajdonságok, 6-39

U,Ú

ucase(), 6-12, 6-50
új adatbázis, 6-2
unió, 4-21, 6-33
union, 4-21, 5-12, 5-56, 6-33
union all, 5-56
unique, 6-52
update, 5-65, 6-18, 6-31, 6-34, 6-52, 9-5
update jogosultság, 7-5
upper(), 5-25, 6-12, 6-50
user(), 5-27, 6-51

Ü,Ű

űrlap, 6-34

felirat, 6-38
űrlap gyorsvarázsló, 6-35
űrlap tervezése, 6-38
űrlap varázsló, 6-36
űrlapfej, 6-38
űrlapláb, 6-38

V

vagy, 6-20
választó lekérdezés, 6-20, 6-21
változó, 6-48
változó hosszúságú mező, 2-27
változtatási anomália, 4-32
value(), 5-27, 6-51
values, 5-68
varchar, 5-19, 6-5, 6-50
varchar(), 5-27, 6-13, 6-51
varchar2, 5-19
vertikális adatmegosztás, 1-26
vetület, 4-15
vezérlésellenőrző nyelv, 1-14, 5-47, 5-71
view, 4-6, 4-39, 5-15, 6-20, 6-52

W

whenever, 5-76
where, 5-55, 5-64, 5-66, 6-19, 6-25

Y

year(), 5-27, 6-10, 6-13, 6-51

Z

zár, 6-46, 8-4
zár erőssége, 8-5
zár időtartama, 8-6
zár mérete, 8-4