5. Az SQL lekérdező nyelv

Az SQL a strukturált lekérdező nyelv (Structured Query Language) rövidítése, melyet az IBM dolgozott ki a DB2 relációs adatbáziskezelőjéhez. Ma már a relációs adatbáziskezelők szabványosított nyelve, bár több dialektusa, bővítése alakult ki.

5.1 Az SQL szerepe, tulajdonságai

Az SQL egy szabványosított lekérdező nyelv, melyet több relációs adatbáziskezelő ismer, különböző operációs rendszeri környezetben. Ennek óriási jelentősége van az adatbázis alkalmazások fejlesztőinek körében, mert így az alkalmazások a különböző operációs rendszerek és adatbáziskezelők között módosítás nélkül vagy csekély módosítással átvihetők.

Az SQL nem algoritmikus nyelv, nem tartalmaz algoritmus szerkezeteket (elágazás, ciklus stb.). Az SQL halmaz orientált nyelv, mely a relációkon dolgozik. A halmaz orientáltság azt jelenti, hogy nem kell definiálni a művelet végrehajtásának lépéseit, hanem a feladat nem eljárás szerű megfogalmazását kell megadni, melyek a reláció vagy relációk kiválasztott sorain hajtódnak végre. A művelet végrehajtásához optimális megoldás megtalálása a nyelvi processzor feladata, nem a programozóé. Például annak eldöntése, hogy egy adott visszakeresésben alkalmazhatók-e indexek, vannak-e indexek vagy építsen-e fel új indexet, a nyelvi processzor feladata. Az SQL nem rekurzív nyelv.

Az SQL nyelvnek két felhasználási lehetősége van:

- önálló SQL, vagy 4. generációs eszközbe építve
- beágyazott SQL

Az SQL nyelv önálló felhasználása esetén csak a nyelv utasításai állnak rendelkezésre. Ennek alkalmazására főként akkor kerülhet sor, ha nincs megfelelő alkalmazás az adott feladat elvégzésére, illetve az alkalmazások fejlesztői használják a negyedik generációs nyelvekbe építve. Ilyen eszközök a jelentés készítő, az űrlap készítő vagy menü készítő lehet.

A beágyazott SQL esetén egy harmadik generációs algoritmikus nyelvbe (C, PL/SQL, Pascal FORTRAN stb.) ágyazva alkalmazzuk az SQL nyelv elemeit. Ebben az esetben az algoritmikus feladatokat a harmadik generációs nyelvre, az adatbázissal kapcsolatos műveleteket pedig az SQL-re bízhatjuk. A beágyazott SQL alkalmazását könyvünkben nem tárgyaljuk.

Az SQL a következő elemekre osztható

- adatdefiníciós nyelv
- adatmanipulációs nyelv
- lekérdező nyelv
- adatvezérlő nyelv

A fejezet további részében ezekkel a részekkel és utasításaikkal ismerkedünk meg. Az utasítások ismertetésénél használt példákban az előzőekben tervezett iskolai adatbázist használjuk. A minta adatbázis tartalma a következő relációkat és adatokat tartalmazza. Az attribútumok neveiben szándékosan nem szerepelnek az ékezetes karakterek, mert ezeket általában nem fogadják el a rendszerek. A könnyebb érthetőség kedvéért a mintapéldában az órarend relációt használjuk az egyszerűbb SQL műveletek érdekében.

A parancsok ismertetésénél nagy betűvel írjuk az SQL parancsokon belüli fix szöveget, kisbetűvel pedig a felhasználó által megadható részeket. Szögletes zárójelbe tesszük a parancsok elhagyható részeit. A parancsok általános alakjába írt ... (három pont) az előző rész ismételhetőségére utal. A | (függőleges vonal) jelet az egymást kizáró paraméterek közé tesszük, ezek közül csak egy adható meg. A mintapéldákban az olvashatóság kedvéért általában több sorra bontottuk az SQL utasításokat, de azok egy sorban vagy másféle tördeléssel is leírhatók. A tárgyalás során nem csak a szabványos SQL parancsokat, hanem az ORACLE nyelvjárását ismertetjük, a teljesség igénye nélkül.

5.2 Az adatdefiniciós nyelv

Az adatdefiniciós nyelv segítségével hozhatjuk létre illetve szüntethetjük meg a relációkat, az indexeket illetve a nézet táblázatokat. A nézet táblázat az adatbázisban fizikailag nem létező relációs műveletek (szelekció, projekció, összekapcsolás, halmazműveletek) segítségével létrehozott táblázat, mely a relációkhoz hasonlóan kezelhető.

A relációk létrehozására a CREATE TABLE SQL utasítás szolgál, melynek általános alakja a következő:

```
CREATE TABLE reláció_név
(attribútum_név adattípus [(szélesség)] [CONSTRAINT megszorítás_név] [oszlop_megszorítás],
attribútum_név adattípus [(szélesség)] [CONSTRAINT megszorítás_név] [oszlop_megszorítás],
... ) [CONSTRAINT megszorítás_név] [tábla_megszorítás];
```

A szélesség megadása el is maradhat. A relációk és általában a nevek megadására a következő szabályok érvényesek:

- a névben csak az angol ABC betűi, a számjegyek és az _, #, $ karakterek szerepelhetnek

- a névnek betűvel kell kezdődnie
- a neveknek hatáskörükön belül egyedinek kell lennie (például nem lehet egy adatbázisban két azonos nevű reláció, egy relációban két azonos nevű attribútum, stb.)

A nevek hossza korlátozott, az Oracle-ben például 30. Az SQL az azonosítókban és a parancs szavakban általában nem tesz különbséget a kis és nagybetűk között.

Az attribútumokra megadható adattípusok köre adatbáziskezelőnként változhat, a következők a legtöbb relációs adatbáziskezelőben használhatók:

- CHAR [(hossz)]
  megadott maximális hosszúságú karakterlánc, csak a karakterláncnak megfelelő hosszúságú területet foglalja el, szinonimája a VARCHAR. Maximum 255 karakterig.
- NUMBER [(szélesség, tizedes)]
  valós szám megadása, a szélesség mellett a tizedespont utáni jegyek száma is megadható, hasonlóan használható a FLOAT
- INTEGER
  egész típusú érték megadása, hozzá hasonló, de számábrázolási tartományában eltérő típus még a SMALLINT, szinonimája a DECIMAL
- DATE
  dátum megadása
- RAW
  a karakterhez hasonló típus, de az adatok értelmezésére nincs semmilyen feltételezés, így segítségükkel tetszőleges bináris adatokat tárolhatunk, például ábrákat is. Maximális hossza 255.
- LONG
  Maximum 64 KByte hosszú szöveg. Relációnként csak egy ilyen lehet, és csak korlátozott helyeken használható
- LONGRAW
  Mint a RAW, de 64 KByte hosszú adat.

Az attribútum típusát megadva, a reláció azon oszlopába más típusú adat nem vihető be, erről az adatbáziskezelő gondoskodik illetve szükség esetén figyelmeztet. Az opcionális oszlop megszorítás a következőket tartalmazhatja.

- NOT NULL az attribútum definíciójában arra utal, hogy az adat megadása kötelező, azaz nem vihető be olyan sor a relációban, ahol az így definiált adat nincs kitöltve.
- PRIMARY KEY ez az oszlop a tábla elsődleges kulcsa.
- UNIQUE ez az oszlop a tábla kulcsa.
- CHECK(feltétel) csak feltételt kielégítő értékek kerülhetnek be az oszlopba.
- [FOREIGN KEY] REFERENCES tábla [(oszlop)], ez az oszlop külső kulcs.

A tábla megszorításban több oszlopra vonatkozó korlátozásokat adhatunk meg.

- PRIMARY KEY(oszlop1[, oszlop2, ...]) ezek az oszlopok együtt alkotják az elsődleges kulcsot.
- UNIQUE(oszlop1[, osylop2, ...]) ezek az oszlopok együtt kulcsot alkotnak.
- CHECK(feltétel) csak feltételt kielégítő sorok kerülhetnek be a táblába.
- FOREIGN KEY (oszlop1[, oszlop2, ...]) REFERENCES tábla(oszlop1[, oszlop2, ...]), az oszlopok külső kulcsot alkotnak a megadott tábla oszlopaihoz.

A megszorításokhoz nevet is rendelhetünk, mely hasznos lehet a megszorítás módosítása, törlése esetén (ALTER TABLE)

A minta adatbázisban található relációk létrehozása a következő utasításokkal történhet:

```
CREATE TABLE Diakok
(Diak_azonosito NUMERIC (4) PRIMARY KEY,
Nev CHAR (30) NOT NULL,
Cim CHAR (40) NOT NULL,
Telefon CHAR (15)
Osztaly CHAR (3) NOT NULL);
```

```
CREATE TABLE Tanarok
(Tanar_azonosito NUMERIC (4) PRIMARY KEY,
Nev CHAR (30) NOT NULL,
Cim CHAR (40) NOT NULL,
Telefon CHAR (15));

CREATE TABLE Orarend
(Tanar_azonosito NUMERIC (4) NOT NULL REFERENCES Tanarok(Tanar_azonosito),
Tantargy CHAR (20) NOT NULL,
Idopont NUMERIC (2), NOT NULL,
Osztaly CHAR (3) NOT NULL),
Terem NUMERIC (3) NOT NULL)
PRIMARY KEY(Tanar_azonosito, Idopont)
UNIQUE(Osztaly, Idopont);

CREATE TABLE Osztalyzatok
(Diak_azonosito NUMERIC (4) NOT NULL REFERENCES Diakok(Diak_azonosito),
Tantargy CHAR (20) NOT NULL,
Datum DATE NOT NULL,
Osztalyzat NUMERIC (1) VALID(BETWEEN 1 AND 5));

CREATE TABLE Hianyzasok
(Diak_azonosito NUMERIC (4) NOT NULL REFERENCES Diakok(Diak_azonosito),
Datumtol DATE NOT NULL,
Datumig DATE,
Igazolt CHAR (1))
PRIMARY KEY (Diak_azonosito, Datumtol);
```

A CREATE TABLE utasítással létrehozott táblázatok definícióját csak korlátozottan módosíthatjuk, újabb attribútumot adhatunk a relációhoz vagy egy attribútum szélességét megnövelhetjük. Egy attribútum szélességének csökkentésére illetve törlésére nincs közvetlen mód. Ez csak úgy érhető el, hogy a módosításoknak megfelelő üres relációt hozunk létre, amibe a szükséges adatokat átmásoljuk, majd az eredeti relációt töröljük. A reláció újabb attribútummal való bővítésére az alábbi parancs szolgál:

```
ALTER TABLE reláció_név
ADD attribútum_név adattipus [(szélesség)];
```

Az új attribútum a reláció utolsó oszlopa lesz. Ebben az esetben a NOT NULL módosító nem adható meg.

Az attribútum értéke a már meglevő sorokban NULL (definiálatlan) lesz. Az SQL nyelv megkülönbözteti a nulla numerikus értéket és a NULL, még nem definiált értéket. Például a diákok adatai közé a születési évet felvehetjük a következő paranccsal:

```
ALTER TABLE Diakok ADD szul_ev INT (4);
```

Egy reláció attribútumának szélességét meg lehet növelni az ALTER TABLE paranccsal.

```
ALTER TABLE reláció_név
MODIFY attribútum_név adattipus (új_szélesség) [NOT NULL];
```

Abban az esetben, ha az attribútum csak NULL értékeket tartalmaz, akkor lehetőség van az adattípus módosítására és a szélesség csökkentésére is. Például a név attribútum szélességének megnövelése a következő paranccsal történhet.

```
ALTER TABLE Diakok MODIFY nev CHAR (40) NOT NULL;
```

Teljes relációk törlésére is lehetőséget biztosít az SQL nyelv a

```
DROP TABLE reláció_név;
```

utasítással. Ezután a relációban tárolt valamennyi adat és a reláció definíciója is törlődik. A diákok személyi adatait tartalmazó reláció törlése a következő paranccsal lehetséges:

```
DROP TABLE Diakok;
```

A nézettáblázat az adatbázisban létező reláción vagy relációkon végrehajtott művelet eredményét tartalmazó olyan új táblázat, amely mögött a valóságban nem áll megfelelő táblázat. Nézettáblát a

```
CREATE VIEW nézettábla_név [alias_név, alias_név ...
AS lekérdezés;
```

paranccsal hozhatunk létre.

A lekérdező utasítás formáit a lekérdező nyelv tárgyalása során részletezzük. A 3/b osztály névsorát tartalmazó nézettáblázatot hoz létre a következő parancs.

CREATE VIEW 3b AS SELECT * FROM Diakok WHERE osztaly = '3/b';

Akár több táblázatból is vehetünk oszlopokat a nézettáblába. A nézettábla segítségével a három relációra felbontott órarend relációt összevonhatjuk egy táblázatba a kényelmesebb kezelés érdekében.

CREATE VIEW orarend FROM tanr-to_id, tantargy-osztaly, ora
AS SELECT tanar_azonosito, tantargy, osztaly, idopont, terem
FROM Tanar-to_id A, Tantargy-osztaly B, Ora C
WHERE C.to_id = B.to_id AND C.to_id = A.to_id;

Az itt szereplő lekérdező (SELECT) utasítás két összekapcsolás műveletet is tartalmaz, melyeket csak később magyarázunk meg.

A nézettáblák megszüntetése a relációkhoz hasonlóan a

DROP VIEW nézettábla_név;

paranccsal lehetséges.

A relációkhoz indexeket is rendelhetünk, melyek helyes megválasztása esetén a lekérdezések felgyorsíthatók. Az indexek létrehozására a következő utasítás szolgál:

CREATE [UNIQUE] INDEX index_név ON reláció (attribútum, attribútum, ...);

Az index létrehozásánál a UNIQUE módosító megadásával a reláció valamennyi sorában különbözőnek kell lennie az index kulcsnak. Általában a reláció kulcsok esetén használható csak (index kulcs = reláció kulcs).

Hozzunk létre egy indexet a Diákok reláció Diák_azonosító attribútuma alapján:

CREATE UNIQUE INDEX Diak ON Diakok (Diak_azonosito);

A parancsban megadott UNIQUE következtében a reláció minden sorában különböző Diák_azonosítónak kell szerepelnie, illetve már létező érték bevitele hibát eredményez. Ez tulajdonképpen a célunk is, mert ez a reláció kulcs. Az index létrehozása után ugyanaz a Diák_azonosító nem vihető be mégegyszer, ilyen esetben hibaüzenetet kapunk.

Az indexek megszüntetése a

DROP INDEX index_név ON [reláció]

parancs segítségével történhet.

Ezen a két parancson kívül nincs nincs szükség további az indexek kezelésére szolgáló parancsra. A relációkhoz kapcsolódó indexek használatáról az adatbáziskezelő optimalizáló algoritmusok alapján dönt. Az indexeket létrehozásuktól a megszüntetésükig az adatbáziskezelő automatikusan frissíti, a módosításoknak megfelelően. Figyelem, az indexek számának növelésével a relációkon végrehajtott módosítások, törlések végrehajtási ideje növekszik.

5.3 Az adatmanipuláiciós nyelv

Az SQL adatmanipulációs része biztosítja a relációk feltöltését, az attribútumok módosítását és a sorok törlését. A relációk feltöltésére az INSERT SQL parancs szolgál, melynek általános alakja a következő:

INSERT INTO reláció [(attribútum_név, attribútum_név, ...)]
VALUES (érték, érték, ...);

Egy utasítás segítségével egy sor adható meg az adott relációhoz. Az attribútum nevek megadása csak akkor kötelező, ha nem minden attribútumhoz rendelünk értéket, vagy az attribútumok értékét nem a definiálás sorrendjében adjuk meg. A NOT NULL megjelöléssel definiált attribútumok megadása kötelező az INSERT parancsnál, ellenkező esetben hibaüzenetet kapunk. Az attribútumok és a VALUES után álló értékek sorrendjének és típusának meg kell felelnie egymásnak. A definiálatlan értékekre a NULL érték is beállítható, mely numerikus értékekre sem azonos a nullával.

Adjunk egy új sort a Diákok relációhoz

INSERT INTO Diakok (Diak_azonosito, Nev, Cim, Osztaly)
VALUES (435, 'Nagy Istvan', 'Budapest O utca 3.', '3.b');

A telefon attribútum értékét nem adtuk meg, így értéke NULL. A reláció létrehozásánál NOT NULL jelzéssel ellátott attribútumokat kötelező minden újabb sor megadásánál kitölteni. Ellenkező esetben hibaüzenetet kapunk. De bevihetjük az előző adatokat az alábbi formában is:

```
INSERT INTO Diakok
VALUES (435, 'Nagy Istvan', 'Budapest Ó utca 3.', NULL, '3.b');
```

Az INSERT utasítás lehetőséget biztosít arra is, hogy a relációt egy másik relációból átvett értékekkel töltsük fel. Ekkor az értékek megadása helyén egy lekérdező utasítás állhat. A lekérdezés eredményei kerülnek be a megadott relációba, egyszerre akár több sor is. Itt is igaz, hogy a lekérdezés eredmény relációjának attribútumai sorrendjének és típusának meg kell felelnie a feltöltendő reláció definíciójának.

```
INSERT INTO reláció_név [(attribútum_név, attribútum_név, ...)]
lekérdező_utasítás;
```

A lekérdező utasítások formájával a következő alpontban foglalkozunk, de már itt is bemutatunk egy példát. Töltsük át a diákok adatai közül a 3/a osztályba járókét egy külön relációba:

```
INSERT INTO 3a
SELECT * FROM diakok WHERE osztaly = '3/a';
```

A 3a relációt természetesen először létre kell hozni a diákok relációval egyező szerkezettel, hogy a fenti parancs működjön. A diákok reláció tartalma nem változik.

A relációkban szereplő mezők tartalmát az UPDATE utasítással módosíthatjuk.

```
UPDATE reláció_név
SET attribútum_név = érték, attribútum_név = érték, ...
[WHERE feltétel];
```

Az UPDATE utasítás segítségével egyidőben a relációk több sorát is módosíthatjuk. A SET után adhatjuk meg a módosítandó attribútumot és értékeit. A WHERE után egy feltétel adható meg, az utasítás csak a reláció azon sorain dolgozik, melyekre a feltétel értéke igaz. A WHERE rész el is maradhat, ekkor a reláció összes sorára vonatkozik az UPDATE parancs. A feltételek alakjára részletesen a lekérdező nyelv ismertetésénél térünk ki, de a szokásos összehasonlító operátorok itt is használhatók. Például az Osztályzatok relációban az összes osztályzatot egyesre (vagy ötösre?) állíthatjuk egy UPDATE paranccsal:

```
UPDATE Osztalyzatok SET Osztalyzat = 1;
```

Az egyenlőség jobb oldalán a reláció attribútumaiból álló kifejezés is állhat. Ilyenkor az aktuális sor tartalma alapján értékelődik ki a kifejezés. Az értékek módosítása esetén a feltételben egy lekérdező utasítás is szerepelhet, melynek segítségével egy másik relációból vett értékeket használhatunk.

```
UPDATE reláció
SET attribútum = (lekérdező utasítás)
[WHERE feltétel];
```

A relációk sorait törölhetjük a DELETE parancs segítségével.

```
DELETE FROM reláció_név
[WHERE feltétel];
```

A feltételben az UPDATE parancshoz hasonlóan egy zárójelek közé tett lekérdező utasítás is megadható. Töröljük ki a Diákok közül a 1234 azonosítójút.

```
DELETE FROM Diakok WHERE Diak_azonosito = 1234;
```

A kérdés csak az, hogy megtehetjük-e ezt az adatbázis konzisztenciájának elvesztése nélkül? Ha az Orarend vagy a Hianyzasok relációban szerepel a törölt diák azonosítója, akkor ezek inkonzistenssé teszik az adatbázisunkat. Helyesebb az alábbi három törlő parancsot kiadni, ha semmi szin alatt sem szeretnénk elveszteni az adatbázis konzisztenciáját.

```
DELETE FROM Hianyzasok WHERE Diak_azonosito = 1234;
DELETE FROM Osztalyzatok WHERE Diak_azonosito = 1234;
DELETE FROM Diakok WHERE Diak_azonosito = 1234;
```

A WHERE alparancs elmaradása esetén a reláció összes sora törlődik. Tételezzük fel, hogy az iskolából kicsapott diákok nevét egy kicsapottak relációban őrizzük a kics oszlopban. Ennek alapján a diákok relációból a következő paranccsal törölhetjük őket:

```
DELETE FROM Diakok WHERE nev IN (SELECT kics FROM kicsapottak);
```

5.4 A lekérdező nyelv

A lekérdező nyelv egyetlen utasításból áll, mely számos alparancsot tartalmazhat, és a lekérdező utasítások többszörös mélységben egymásba ágyazhatók. A SELECT utasítás általános alakjának megadása helyett részletesen áttekintjük az egyes

tipikus lekérdezési utasításokat, az egyszerűektől a komplikáltakig. Figyelem, a szelekció művelete és a SELECT utasítás csak nevében hasonló, egymásnak nem felelnek meg.

Először tekintsük át az egy relációra vonatkozó lekérdezéseket. A projekció műveletét a következő utasítással valósíthatjuk meg:

SELECT [DISTINCT]
attribútum_név, attribútum_név, ... FROM reláció_név;

A megadott reláció felsorolt attribútumai jelennek meg az utasítás hatására soronként. A DISTINCT módosító megadása esetén csak az egymástól különböző sorok láthatók.Például a Diakok reláció Diak_azonosito és Nev attribútumainak lekérdezése a

SELECT Diak_azonosito, nev FROM Diakok;

paranccsal történhet.

A különböző tanár-tantárgy párosítások lekérdezése az Orarend relációból a következő paranccsal történhet:

SELECT DISTINCT Tanar_azonosito, Tantagy FROM Orarend;

A szelekció művelet megvalósítása esetén a SELECT utasítást egy feltétellel egészítjük ki:

SELECT attribútum_név, attribútum_név, ... FROM reláció_név
WHERE feltétel;

Ha az attribútum nevek helyett csak "*"-ot adunk meg, akkor az eredményben a reláció valamennyi attribútuma szerepelni fog:

SELECT * FROM Diakok WHERE osztaly = '3/b';

Megadott attribútumok esetén a projekció és a szelekció művelete összevonható egy utasítással:

SELECT idopont, tantargy FROM orarend WHERE osztaly = '3/b';

A keresési feltételben szerepelhetnek összehasonlító operátorok, melyek numerikus, karakteres és dátum típusú adatok esetén is használhatóak.

Összehasonlító operátorok

Operátor	Értelmezés
=	egyenlő
!= <> ^=	nem egyenlő
>	nagyobb
>=	nagyobb egyenlő
"<"	kisebb
"<="	kisebb egyenlő

Az összehasonlító operátorok segítségével attribútumokat és konstansokat hasonlíthatunk össze. A szöveg és dátum konstansokat idézőjelek között kell megadni. Az alapértelmezés szerinti dátum formátum nap-hónap-év. A hónap a hónap nevének három betűs angol rövidítése, az év pedig évezred és évszázad nélkül értendő. Az SQL tartalmaz olyan összehasonlító operátorokat is, melyek nem egy adott értékkel, hanem az értékek egy halmazával történő összehasonlítást eredményeznek.

Összehasonlító operátorok halmazokra

Operátor	Értelmezés
BETWEEN x AND y	adott értékek közé esik
IN (a, b, c, ...)	az értékek között található
LIKE minta	hasonlít a mintára

Az IN esetén egy halmazt adhatunk az elemek felsorolásával. A LIKE operátort karakteres mezők összehasonlítására alkalmazhatjuk. Két speciális karakter adható meg a mintában, a % jel tetszőleges hosszúságú karakter sorozatot helyettesít, az _ aláhúzás karakter pedig egy tetszőleges karaktert. Például:

Szöveges minta megadása

Operátor	Értelmezés
LIKE 'a%'	minden 'a' betűvel kezdődő
LIKE 'x_'	minden 'x'-el kezdődő kétbetűs
LIKE '%a%'	minden 'a' betűt tartalmazó
LIKE '_a%x'	második betű 'a' és 'x'-re végződő

Az attribútumokra használható még egy speciális összehasonlító operátor, az IS NULL, melyek segítségével eldönthetjük, hogy a mező ki van-e töltve. Több keresési feltételt is összekapcsolhatunk a logikai operátorokkal, illetve segítségükkel a halmaz műveleteket valósíthatjuk meg.

Logikai operátorok

Operátor	Értelmezés
NOT	Logikai tagadás
AND	Logikai és
OR	Logikai vagy

Az összehasonlító operátorok precedenciája (végrehajtási sorrendje) csökkenő sorrendben:

1. =, !=, <>, ^=, >, >=, <, <=
2. NOT
3. AND
4. OR

Nézzünk néhány példát a lekérdezésekre. A 3.a osztályba járó diákok adatai:
SELECT * FROM Diakok WHERE Osztaly = '3/a';

A matematikát tanító tanárok azonosítói (DISTINCT itt is használható!):
SELECT DISTINCT Tanar_azonosito FROM Orarend WHERE Tantargy = 'matematika';

A 'C' betűvel kezdődő nevű diákok:
SELECT Nev, Osztaly FROM Diakok WHERE nev = 'C%';

A 3/a-ba járó diákok, akiknek nincs otthon telefonjuk: SELECT Nev FORM Diakok WHERE Osztaly = '3.a' AND Telefon IS NULL;

Termek, ahol matematika vagy informatika órát tartanak:
SELECT Terem FROM Orarend WHERE Tantargy = 'matematika' OR tantargy = 'informatika';
vagy ugyanez az IN felhasználásával:
SELECT Terem FROM Orarend WHERE Tantargy IN ('matematika', 'informatika');

Matematikából hármas és ötös közötti osztályzatot szerzett diákok:
SELECT Diak_azonosito FROM Osztalyzatok WHERE Osztalyzat BETWEEN 3 AND 5 AND tantargy = 'matematika';

Telefonnal rendelkező diákok:
SELECT Diak_azonosito FROM Diakok WHERE NOT IS NULL Telefon;

Az eddigi lekérdezések eredményei a sorok tárolt sorrendjében kerültek kiírásra. Az SQL lehetőséget biztosít a lekérdezés eredménysorainak rendezésére az ORDER BY alparancs segítségével.

SELECT attribútum, attribútum, ... FROM reláció
[WHERE feltétel]
ORDER BY attribútum [ASC|DESC], attribútum
[ASC | DESC], ...;

Az ORDER BY után megadott attribútumok alapján ASC esetén (ez az alapértelmezés) növekvő, vagy DESC esetén csökkenő sorrendbe rendezi az eredmény sorait. Ha több attribútumot adunk meg a rendezési feltételben, akkor a megadás sorrendje alapján történik a rendezés, azaz először az elsőnek megadott attribútum alapján rendezi sorba a sorokat, ha ez az attribútum azonos két sorra, akkor a másodikként megadott attribútum alapján, és így tovább. Karakteres attribútumoknál a rendezés a karakterek kódjai alapján történik (ASCII vagy EBCD), azaz például nevek esetén az angol ABC szerint. Például a 3/a osztály órarendje időrendi sorrendben az alábbi lekérdezéssel kapható meg:

```
SELECT Idopont, Tantargy, Terem FROM Orarend WHERE Osztaly = '3/a'
ORDER BY Idopont;
```

Egy terem elfoglaltsága időrendben:

```
SELECT Idopont, Tantargy, Osztály FROM Orarend WHERE Terem = 104
ORDER BY Idopont;
```

Osztályonkénti névsor az összes diákra:

```
SELECT Nev, Osztály FROM Diakok ORDERED BY Osztaly, Nev;
```

A lekérdezés eredményét csoportosíthatjuk és a csoportok között is további szelekciót alkalmazhatunk a GROUP BY és HAVING alparancsokkal.

```
SELECT attribútumok FROM reláció
[WHERE feltétel]
GROUP BY attribútum
[HAVING csoport_feltétel];
```

A GROUP BY alparancs után megadott attribum azonos értékei alapján csoportosítja az SQL a lekérdezés eredményeit és a csoport feltételnek megfelelő sorok kerülnek az eredménybe. A csoportok képzésekor az eredmények között az azonos csoportba tartozó sorokból levezetett további eredmények megjelenítésére is lehetőséget biztosít az SQL. Ez a következő függvényekkel valósítható meg:

Függvények halmazokra és eredményeik
az alábbi adatok esetén: 1, 2, 3, 4, 1, 4, 4, NULL, 5

Függvény	**Értelmezés**	**Eredmény**	
		ALL	**DISTINCT**
AVG (attribútum)	átlag	3	3
COUNT (attribútum)	nem NULL elemek száma	8	5
COUNT (*)	sorok száma NULL is	9	6
MAX (attribútum)	maximális elem	5	5
MIN (attribútum)	minimális elem	1	1
SUM (attribútum)	összeg	24	15
STDDEV(attribútum)	szórás		

A függvények argumentuma előtt megadható a DISTINCT vagy ALL módosító. DISTINCT esetén csak a különböző értékek, ALL esetén minden érték részt vesz a számításban. Az alapértelmezés az ALL. A definiálatlan, NULL értékek nem szerepelnek a számításban.

Az ötnél nagyobb létszámú osztályok:

```
SELECT Osztály, COUNT (*) FROM Diakok GROUP BY Osztaly
HAVING COUNT (*) > 5
```

Diákok tanulmányi átlaga matematikából:

```
SELECT Diak_azonosito, AVG (Osztalyzat)
FROM Osztalyzatok
WHERE tantargy = 'matematika'
GROUP BY Diak_azonosito;
```

A függvények alkalmazhatók a GROUP BY nélkül is. Például a tanárok száma:

```
SELECT COUNT (*) FROM Tanarok;
```

Matematika osztályzatok száma és átlaga:
SELECT COUNT (*)', AVG (Osztalyzat) FROM Osztalyzatok
WHERE Tantargy = 'matematika';

A lekérdezésekben az attribútumok és a csoportokra vonatkozó függvények mellett az attribútumok aritmetikai kifejezéseit is használhatjuk. A kifejezésekben az alapműveletek használhatók, illetve zárójelek a műveletek végrehajtási sorrendjének módosítására. A karakteres adatok konkatenálására a || operátor használható.

Osztályzatok kiírása fordítva, vagyis az egyes a legjobb:
SELECT Diak_azonosito, 6 - Osztályzat FROM Osztalyzatok;

A kifejezésekben az alapműveletek mellett számos függvényt is használhatunk a karakteres, numerikus és dátum tipusú adatokra.

Karakteres függvények

Függvény	Magyarázat	Példa
ASC (szöveg)	A szöveg első karakterének ASCII kódja	ASC ('abc') = 65
CHR (egész)	A számnak megfelelő kódú karakter	CHR (65) = 'a'
INITCAP (szöveg)	A szavak kezdőbetűit nagybetűvé	INITCAP ('ló pál') = 'Ló Pál'
INSTR (szöveg1, szöveg2, kezdet, hányadik)	A szöveg1-ben a szöveg2 hányadik előfordulása a kezdettől. Hányadik és kezdet elmaradhat	INSTR ('abcd', 'cd') = 3
LENGTH (szöveg)	A szöveg hosszát adja	LENGTH ('abc') = 3
LOWER (szöveg)	Kisbetűssé alakítja a szöveget	LOWER ('ABC') = 'abc'
LPAD (szöveg, hossz, karakterek)	A szöveget kiegészíti balról a megadott karakterekkel az adott hosszig, Karaktereket nem kötelező megadni, ekkor szóköz.	LPAD ('x', 3) = ' x' LPAD ('x', 5, '12') = '1212x'
LTRIM (szöveg, karakterek)	A szöveg elejéről levágja a karakterekkel egyező részt. Karaktereket nem kötelező megadni, ekkor szóköz.	LTRIM (' x') = 'x LTRIM 'KUKURIKU', 'UK' = 'RIKU'
RPAD (szöveg, hossz, karakter)	A szöveget kiegészíti jobbról a megadott karakterekkel az adott hosszig, Karaktereket nem kötelező megadni, ekkor szóköz	RPAD ('x', 3) = 'x ' RPAD ('x', 5, '12') = 'x1212'
RTIM (szöveg, karakter)	A szöveg végéről levágja a karakterekkel egyező részt. Karaktereket nem kötelező megadni, ekkor szóköz.	LTRIM ('x ') = 'x' LTRIM ('KUKURIKU', 'UKI' = 'KUKUR'
SUBSTR (szöveg, kezdet, hossz)	A szöveg része a kezdet pozíciótól adott hosszban. Hossz nélkül a szöveg végéig	SUBSTR ('abcd', 2, 1) = 'b' SUBSTR ('abcd', 3) = 'cd'
TRANSLATE (szöveg, mit, mire)	A szövegben előforduló mit karaktereket kicseréli a mire karaktereire	TRANSLATE ('abc', 'ab', 'AB') = 'ABc')
UPPER (szöveg)	Nagybetűssé alakítja a szöveget	UPPER ('abc') = 'ABC'

Nem minden függvény található meg minden adatbáziskezelőben, előfordulhat, hogy más névvel találja meg.

Numerikus függvények

Függvény	Magyarázat	Példa
ABS (érték)	Abszolút érték)	ABS (-1) = 1

CEIL (érték))	Az értéknél nagyobb vagy egyenlő legkisebb egész)	CEIL (6.12) = 7
FLOOR (érték))	Az értéknél kisebb vagy egyenlő legnagyobb egész)	FLOOR (3.95) = 3
MOD (érték, osztó))	Osztási maradék)	MOD (8, 3) = 2
POWER (érték, kitevő))	Hatványozás)	POWER (3, 4) = 81
ROUND (érték, pontosság))	Kerekítés a megadott jegyig. Negatív pontosság is megadható.)	ROUND (123.456, 1) = 123,5 ROUND (163.456,-2) = 200
SIGN (érték))	Előjel függvény)	SIGN (-3) = -1
SQRT (érték))	Négyzetgyök vonás)	SQRT (144) = 12
TRUNC (érték, pontosság))	Csonkítás a megadott jegyig. (Negatív pontosság is megadható.)	TRUNC (123.456, 1) = 123.4 TRUNC (163.456,-2) = 100

Nem minden függvény található meg minden adatbáziskezelőben, előfordulhat, hogy más névvel találja meg.

Dátum függvények

Függvény	Magyarázat	Példa
ADD_MONTH (dátum, n)	A dátumhoz n hónapot ad	ADD_MONTH ('10-MAY-93',2) = '10-JUL-93'
LAST_DAY (dátum)	A dátumban szereplő hónap utolsó apja	LAST_DAY ('1-JAN-93') = '31-JAN-93')
MONTH_BETWEEN (dátum1, dátum2)	A két dátum közötti idő hónapokban	MONTH_BETWEEN (
NEXT_DAY (dátum, nap)	A dátum utáni első nap nevű napra eső dátum	NEXT_DAY ('10-MAY-93','TUESDAY') = '11-MAY-93')
ROUND (dátum, formátum)	Dátum kerekítése a megadott formátum szerint	
TO_CHAR (dátum, formátum)	Dátum megadott karakteres formátumba konvertálása	TO_CHAR (4-DEC-58, 'YY.MM.DD') = '58.12.04'
TO_DATE (szöveg, formátum)	A szöveg dátummá alakítása a formátum szerint	TO_DATE ('58.12.04', 'YY.MM.DD') = 4-DEC-58
TRUNC (dátum, formátum)	Dátum csonkítása a megadott formátum szerint	

Nem minden függvény található meg minden adatbáziskezelőben, előfordulhat, hogy más névvel találja meg.

A SELECT utasítás a relációk közötti szorzás művelet végrehajtására is alkalmas. Ehhez több reláció nevét kell megadni a FROM után. Például az összes lehetséges diák és tanár azonosító pár visszaírása:
SELECT diak_azonosito, tanat_azonosito FROM Diakok, Tanarok;

Ennél komplikáltabb szorzás műveletet is megfogalmazhatunk. Például a 3/a osztályban rendezendő körmérkőzéses sakk bajnokság összes lehetséges párosításának előállítása:
SELECT A.nev, B.nev
FROM Diakok A, Diakok B WHERE A.osztaly = '3/a' AND B.osztaly = '3/a' AND
A.diak_azonosito <> B.diak_azonosito;

A különböző diák azonosítókra vonatkozó feltétel megadására azért van szükség, hogy az önmagukkal alkotott párok ne kerüljenek be a lekérdezés eredményébe.

A SELECT SQL utasítás segítségével eddig is sokfajta kérdésre kaptuk meg közvetlenül a választ. Az összekapcsolás segítségével még komplexebb kérdéseket oldhatunk meg egy lépésben.

SELECT [reláció.]attribútum, [reláció.]attribútum, ...
FROM relációk

```
WHERE kapcsoló_attribútum operátor
kapcsoló_attribútum;
```

Az összekapcsolás esetén mindig legalább két relációval dolgozunk. Ha a két relációban azonos az attribútumok neve, akkor minősített névvel, reláció.attribútum segítségével hivatkozhatunk a kívánt elemre.

A diákok neve mellé írjuk ki az osztály, tantárgy és osztályzat értéket:

```
SELECT Nev, Osztaly, Tantargy, Osztalyzat FROM Diakok, Osztalyzatok
WHERE Diakok.Diak_azonosito = Osztalyzatok.Diak_azonosito;
```

Az előző egy egyen összekapcsolás volt, de például a 12 azonosítójú diák osztályzatainál jobbat szerzők listája is előállítható az egy reláción végrehajtott összekapcsolással:

```
SELECT DISTINCT A.Diak_azonosito, A.Osztalyzat
FROM Osztalyzatok A, Osztalyzatok
WHERE B.Osztalyzat < A.Osztalyzat AND B.Diak_azonosito = 12;
```

Ebben a lekérdezésben az 'A' és 'B' ugyanannak a relációnak az alias nevei, melyek segítségével két sort vizsgálhatunk ugyanabból a relációból. Az előző lekérdezést módosítsuk úgy, hogy a diákok neve is megjelenjen:

```
SELECT Nev, A.Osztalyzat FROM Osztalyzatok A, Osztalyzatok B, Diakok
WHERE B.Osztalyzat < A.Osztalyzat AND B.Diak_azonosito = 12; AND
Diakok.Diak_azonosito = A.Diak_azonosito;
```

Az SQL szabvány egy másik megoldást is biztosít a belső összekapcsolásra az INNER JOIN kulcsszavakkal:

```
SELECT [reláció.]attribútum, [reláció.]attribútum, ...
FROM első_tábla INNER JOIN második tábla
ON első_tábla.kulcs_mező = második_tábla.kulcs_mező;
```

A korábbi példa, a diákok neve mellé írjuk ki az osztály, tantárgy és osztályzat értéket, így is megfogalmazható:

```
SELECT Nev, Osztaly, Tantargy, Osztalyzat FROM Diakok INNER JOIN Osztalyzatok
ON Diakok.Diak_azonosito = Osztalyzatok.Diak_azonosito;
```

A LEFT JOIN illetve RIGHT JOIN kulcsszavakkal az úgynevezett külső összekapcsolás is megvalósítható. Ebben az esetben az egyik tábla minden sora megjelenik az eredményben, akkor is, ha nincs az összekapcsolás feltételének megfelelő sor a másik táblában. A LEFT JOIN esetén az első tábla, míg a RIGHT JOIN esetén a második tábla minden sora bekerül az eredmény táblába. A külső összekapcsolás nem fogalmazható meg a WHERE feltételben.

```
SELECT [reláció.]attribútum, [reláció.]attribútum, ...
FROM első_tábla LEFT JOIN második tábla
ON első_tábla.kulcs_mező = második_tábla.kulcs_mező;
```

Az előző példát alakítsuk át úgy, hogy azoknak a diákoknak a neve is megjelenjen a listában akiknek még nincs osztályzatuk:

```
SELECT Nev, Osztaly, Tantargy, Osztalyzat FROM Diakok LEFT JOIN Osztalyzatok
ON Diakok.Diak_azonosito = Osztalyzatok.Diak_azonosito;
```

Az összekapcsolás művelete mellett a lekérdezések további egymásba ágyazása is lehetséges. A WHERE alparancsban az összehasonlítás egyik oldalán szerepelhet egy újabb, zárójelbe tett SELECT utasítás. A mélyebben elhelyezkedő SELECT utasítás több sort is visszaadhat a relációból. Ezeket az eseteket az ANY, ALL és EXISTS segítségével is kezelhetjük. Az ANY és ALL esetén az összehasonlítás egyik oldalára egy listát írhatunk, mely származhat egy beágyazott lekérdezésből is. A legrosszabb osztályzat matematikából:

```
SELECT DISTINCT Osztalyzat FROM Osztalyzatok
WHERE Tantargy = 'matematika' AND Osztalyzat <= ALL
(SELECT Osztalyzat FROM Osztalyzatok
WHERE Tantargy = 'matematika');
```

Ugyanezt a kérdést egyszerűbben is megfogalmazhatjuk:

```
SELECT MIN (Osztalyzat) FROM Osztalyzatok WHERE Tantargy = 'matematika';
```

Az ANY és ALL használható a WHERE részben fix listákkal is, például x > ALL (12,21,8), x nagyobb, mint a lista legnagyobb eleme vagy y < ANY (5,4,7), y kisebb, mint a lista egyik eleme. Ezeket a feltételeket általában egyszerűbben is le tudjuk írni, az előző példák esetén x > 21 illetve y < 7. Az EXISTS esetén mindig egy újabb beágyazott lekérdezés következik. Ha egy beágyazott lekérdezés talál a feltételt kielégítő sort, akkor igaz értéket kapunk, különben hamisat. Például kinek nincs még osztályzata matematikából a 3/b-ben:

```
SELECT Nev FROM Diakok D WHERE NOT EXISTS
(SELECT * FROM Osztalyzatok
WHERE D.Diak_azonosito = Diak_azonosito AND
tantargy = 'matematika');
```

Beágyazott lekérdezések használatára még egy példa, kíváncsiak vagyunk Kiss János osztálytársainak a nevére:
SELECT Nev FROM Diakok WHERE Osztaly =
(SELECT Osztaly FROM Diakok WHERE Nev = 'Kiss János');

Az egyenlőség vizsgálat esetén a belső lekérdezés csak egy értéket adhat vissza. A relációkon értelmezett halmazműveletek bizonyos esetekben az OR, AND, NOT és IN operátorok segítségével is elvégezhetők. Az SQL ezek mellett az INTERSECT, UNION és MINUS műveleteket is biztosítja. Ezek segítségével két lekérdezést kapcsolhatunk össze a következő formában:

Halmaz muveletek megvalósítása az SQL-ben

Unió	Metszet	Különbség
SELECT .... UNION SELECT ...	SELECT ... INTERSECT SELECT ...	SELECT ... MINUS SELECT ...

A MINUS kulcsszó helyett az EXCEPT használandó néhány adatbáziskezelőben

Válasszuk ki azokat a termeket, ahol a 3/b-nek vagy a 3/a-nak vannak órái:
SELECT Tanterem FROM Orarend WHERE Osztaly = '3/b'
UNION
SELECT Tanterem FROM Orarend WHERE Osztaly = '3/a';

Ugyanez az IN operátorral rövidebb:
SELECT Tanterem FROM Orarend WHERE Osztaly IN ('3/a', '3/b');

Azon osztályok, melyeknek a 101 és 102 teremben is van órája:
SELECT Osztaly FROM Orarend WHERE Tanterem = 101
INTERSECT
SELECT Osztaly FROM Orarend WHERE Tanterem = 102;

Azon osztályok, melyeknek a 101-es teremben van, de a 102-es teremben nincs órája: SELECT Osztaly FROM Orarend WHERE Tanterem = 101
MINUS
SELECT Osztaly FROM Orarend WHERE Tanterem = 102;

Az utóbbi két lekérdezés is megoldható a halmazműveletek nélkül, de csak egymásba ágyazott lekérdezésekkel.

5.5 A vezérlő nyelv

A SQL vezérlő nyelv több funkciót lát el, ezek közül most csak a tranzakciók kezeléséhez szükséges parancsokat ismertetjük. A többi parancsról az adatbázis adminisztrátor parancsainál beszélünk. A logikailag egybe tartozó SQL utasításokat tranzakcióknak nevezzük. Az adatbázis ellentmondás mentes tartalmának megőrzéséhez a tranzakcióknak maradéktalanul végre kell hajtódniuk. Azonban egy tranzakció megszakadása esetén is gondoskodni kell az adatbázis konzisztenciájának megőrzéséről. Erre a COMMIT és ROLLBACK parancs pár szolgál. Az ORACLE nem az eredeti relációkon dolgozik. A sikeresen végrehajtott tranzakciók után a COMMIT parancs kiadásával véglegesíthetjük a tranzakció során végrehajtott változtatásokat a relációkban. A ROLLBACK parancs segítségével pedig visszaléphetünk az előző véglegesített állapothoz. Bizonyos parancsok automatikusan COMMIT-ot eredményeznek (CREATE TABLE, QUIT stb.). De az AUTOCOMMIT rendszerváltozó beállításától függően minden parancs kiválthat egy COMMIT-ot is.
SET AUTOCOMMIT ON

<<Előző fejezet Tartalom Következő fejezet>>