SQL (Structured Query Language)
strukturált lekérdező nyelv rövid ismertetése

Tartalomjegyzék:

1. ÁLTALÁNOS JELLEMZŐK....1

2. SELECT LEKÉRDEZŐ UTASÍTÁS....3

2.1. Egyszerű lekérdezések....3
2.2. Az eredménysorok rendezése....5
2.3. Sorok kiválasztása, lekérdezés keresési feltétellel....5
2.4. Származtatott adatok, SQL sor-függvények....9
2.5. Csoportok képzése, SQL csoport-függvények....12
2.6. Egymásba ágyazott SELECT....15
2.7. Táblák összekapcsolása (Join)....16
2.8. Hierarchia lekérdezése ORACLE-ben....19

3. DML (ADATMANIPULÁCIÓS, ADATKEZELŐ NYELV)....20

3.1. Adatok lekérdezése - lásd. 2. fejezet....21
3.2. Adatok felvitele....21
3.3. Adatok módosítása....22
3.4. Adatok törlése....23

4. DDL (ADATDEFINÍCIÓS NYELV)....24

4.1. Alaptábla létrehozása, módosítása, törlése....26
4.2. Nézettábla létrehozása, törlése....30
4.3. Indexek létrehozása, törlése....32
4.4. Szekvenciák létrehozása, törlése ORACLE-ben....33

5. DCL (ADATVEZÉRLŐ NYELV)....34

5.1. Tranzakciókezelés....34
5.2. Felhasználók kezelése, felhasználói jogosultságok szabályozása....36

1. Általános jellemzők

Az **SQL** nyelv a relációs adatbázis-kezelők (**RDBMS:** Relational Batabase Management System) (pl. ORACLE, MS SQL Server, INGRES, INFORMIX, DB2, SYBASE, NOVELL XQL, PROGRESS, stb.) szabványos adatbázis nyelve.

Jellemzői:

- Nem algoritmikus, deklaratív nyelv:

  Parancsnyelv jellegű, azaz lényegében azt fogalmazhatjuk meg vele, hogy mit akarunk csinálni, a hogyant azaz a feladat megoldási algoritmusát nem kell a felhasználónak megadni. Nincsenek benne az algoritmikus nyelvekben megszokott utasítások (ciklusok, feltételes elágazások, stb.)

- Halmazorientált:

  Táblákat mint a sorok (rekordok) halmazát tekintjük. Az utasításban megfogalmazott feltételnek eleget tevő összes sor részt vesz a műveletben.

- Teljes adat - nyelv függetlenség: fizikai, és logikai szinten

  Fizikai szinten: az adatok fizikai tárolási, hozzáférési, ábrázolási módjában bekövetkező változáskor nincs szükség a programok módosítására.

  Logikai szinten: a programokat nem befolyásolja az adatok logikai szerkezetének megváltozása: pl. sorok, oszlopok sorrendje, új oszlopok, indexek, tablespace-ek, stb.

- Szabványos:

  Illeszkedik az SQL szabványhoz: SQL89 (SQL1), SQL92 (SQL2), SQL99 (SQL3).

  Egy szabványnak kötelező, ajánlott, szabadon választott részei vannak.

  A szabványban van egy SQL utasításcsoport, amelyet minden SQL alapú szoftver implementációnak meg kell valósítani, de mindegyik implementáció plusz lehetőséget is nyújt a standard SQL-hez képest, azaz felülről kompatibilis a szabvánnyal.

Megjegyzés: a legújabb SQL szabvány az objektumorientált alapelveket is magába foglalja. Objektumrelációs adatbázis (**ORDB** Object Relational Database) és az ezt kezelő **ORDBMS** (Object Relational Database Management System) rendszer a relációs (adatstruktúra) és az objektumorientált (adatstruktúra + adatok viselkedése) technológia egyesítése. Ez a relációs adatmodell objektumorientált kiterjesztésén alapszik. Például a beépített (Built-in) oszloptípusokon és operátorokon túl lehetséges a felhasználói (User-Defined) típus- és operátor definiálás, amelyekkel új típusokat (pl. objektumot attribútumokkal, metódusokkal) és műveleteket lehet definiálni, majd használni lehet azokat az SQL utasításokban a beépített típusokhoz és operátorokhoz hasonlóan.

Ezen anyag az ORACLE SQL nyelvének csak a relációs adatmodellt és elsősorban az SQL szabványt realizáló részével foglalkozik!

Fontosabb használati módjai:

- Önállóan fejlesztő eszközökben: pl.: SQL*Plus, Oracle*Developer (Form, Report készítő), stb.
- Beágyazva procedúrális programozási nyelvekbe. pl.: C/C++, ADA, COBOL, stb. befogadó gazdanyelvekbe, melyeket először előfordítóval (precompiler) kell lefordítani.
- PL/SQL (az ORACLE saját procedúrális nyelve, az SQL nyelv procedúrális kiterjesztése) és JAVA nyelvekben. Ezen nyelvekben írt programegységek tárolhatók és futtathatók a ORACLE adatbázis szerveren.

Az SQL nyelv utasításainak főbb csoportjai:

DDL adatdefiníciós nyelv (**D**ata **D**efinition **L**anguage),
adatbázis és a szerkezeti elemek kialakítása:
CREATE, ALTER, DROP, RENAME

DML adatmanipulációs nyelv (**D**ata **M**anipulation **L**anguage),
adatok karbantartása (bevitele, módosítása, törlése), lekérdezése:
INSERT, UPDATE, DELETE, SELECT
Megj: a SELECT utasítást korábban szokás volt önálló
DQL (**D**ata **Q**uery **L**anguage) lekérdező nyelvnek is nevezni.

DCL adatvezérlő nyelv (**D**ata **C**ontrol **L**anguage),
tranzakció kezelése, lefoglalások:
COMMIT, ROLLBACK, SAVEPOINT, LOCK, UNLOCK
adatvédelem, felhasználói hozzáférés szabályozása:
GRANT, REVOKE

Egyéb nem szabványosított utasítások: Tárolási előírások, működési körülmények beállítása, adatbázis mentése, visszatöltése, helyreállítása, exportja, importja, stb.

A nyelv elemei: karakterkészlet (jelkészlet), kulcsszavak, elválasztó jelek, azonosítók, konstansok (literálok), függvények, műveleti jelek (operátorok), kifejezések, stb.

Lexikális konvenciók: az utasítások írhatók kis- és nagybetűvel, egy vagy több sorba, az utasítás végét ; (pontosvessző) jelzi.

Megjegyzések (comment) elhelyezése:

/* több soron áthúzódható megjegyzés */
-- sor végéig tartó megjegyzés

*Az SQL utasítások szintaxisának megadásánál nem törekszünk a teljességre, néhány esetben nem is adunk meg minden opciót, előírást. Ezek megtalálhatók az SQL referencia kézikönyvekben. A Backus-Naur Form (**BNF**) **szintaxisban** egy szintaktikus egység vesszővel elválasztott ismételhetőségét rövidség miatt egyszerűen **, …** módon jelöljük, nem írva ki előtte ismételten az ismételhető egységet.*

2. SELECT lekérdező utasítás

Egy vagy több tábla vagy nézettábla tartalmát kérdezhetjük le, de használhatjuk beépítve más SQL utasításban is (alselect). Legáltalánosabb formája a következő:

SELECT ...	oszlopok kiválasztása (projekció)	**(6)**
FROM ...	táblanév(-ek) (Descartes szorzat)	**(1)**
[WHERE ...]	sorok kiválasztása (szelekció)	**(2)**
[CONNECT BY ... [START WITH ...]]	hierarchia kezelés	**(3)**
[GROUP BY ...]	csoportosítás	**(4)**
[HAVING ...]	csoportok közötti válogatás	**(5)**
[{UNION | UNION ALL | INTERSECT | MINUS} alselect]	halmazműveletek	**(7)**
[ORDER BY ...]	eredménysorok rendezése	**(8)**
[FOR UPDATE OF …]	kiválasztott sorok zárolása az updatelés idejére	**(9)**

Az utasításrészek megadási sorrendje kötött. A kiértékelést a ()-ben megadott sorrendben célszerű követni. Az utasítás általános alakja elég bonyolult, így csak fokozatosan adjuk meg a részletesebb szintaktikáját.

2.1. Egyszerű lekérdezések

SELECT [**ALL** | **DISTINCT**] { [táblanév.]* | o_kifejezés [[**AS**] o_aliasnév] } ,...
FROM táblanév [t_aliasnév] ,...;

ALL	- alapértelmezés: az összes sort visszaadja.
DISTINCT	- csak az egymástól különböző sorokat adja vissza
o_aliasnév	- a lekérdezés eredményében nem az oszlop v. kifejezés valódi neve, hanem az o_aliasnév jelenik meg.
t_aliasnév	- a táblanév rövidítésére használható az utasításban.

Az o_aliasnév és a t_aliasnév csak az adott utasításban érvényes.

o_kifejezés - legegyszerűbb esetben ez egy oszlop neve, de állhat itt oszlopnevek, konstansok és függvényhívások összekapcsolva aritmetikai operátorokkal (*, /, +, -) vagy konkatenáló karakteres operátorral (||), zárójelezés is megengedett.
konstans: adattípusa lehet: karakteres, numerikus, dátum:
Pl.: 'Karaktersor', 1234.56, 25E-3, '21-MAJ-74' vagy '74-MÁJ-21'
Karakteres típus megkülönbözteti a kis- és nagybetűt.
Dátum konstans függ az alapértelmezett dátum formátumtól és nyelvtől:
Pl. 'DD-MON-YY' vagy 'YY-MON-DD' vagy 'RR-MON-DD'
Ez a dokumentum többnyire az angol nyelvet és dátumformátumot használja! Az aktuális beállítások megtekinthetők:

```
SELECT * FROM v$nls_parameters;
```

függvények: később ismertetjük.
oszlopnév: [táblanév.]oszlopnév
Minősített név kell, ha egy oszlopnév az utasítás több táblájában is előfordul. Az oszlopnév helyén álló * a tábla összes oszlopát jelenti.

Minden alkalmazott minden adata:

```
SELECT * FROM alkalmazott;
```

Minden alkalmazott néhány adata:

```
SELECT a_nev, beosztas
      FROM alkalmazott;
SELECT a_nev, 'Szöveg ' || beosztas
      FROM alkalmazott;
SELECT a_nev, (fizetes+premium)*12+1000 AS jövedelem
      FROM alkalmazott;
```

A különböző beosztások lekérdezése:

```
SELECT DISTINCT beosztas
      FROM alkalmazott;
```

Az alkalmazottak két táblából származtatható adatai:
(Táblák összekapcsolása később lesz részletezve!
Itt ugyanaz az utasítás van többféleképpen leírva.)

```
SELECT a_nev, cim, telephely.t_kod, t_nev
        FROM alkalmazott, telephely
        WHERE alkalmazott.t_kod= telephely.t_kod;
SELECT a_nev, cim, T.t_kod, t_nev
        FROM alkalmazott A, telephely T
        WHERE A.t_kod=T.t_kod;
SELECT a_nev, T.*
        FROM alkalmazott A, telephely T
        WHERE A.t_kod=T.t_kod;
```

2.2. Az eredménysorok rendezése

SELECT ...
FROM ...
…
ORDER BY {o_kifejezés| o_aliasnév | o_sorszám} [**ASC** | **DESC**] ,...;

A rendezés alapértelmezésben növekvő, DESC hatására csökkenő lesz a sorrend. A rendezés értelmezve van minden alaptípusnál. A NULL érték igen nagy értéket képvisel.

```
SELECT * FROM alkalmazott
        ORDER BY t_kod, beosztas DESC;

SELECT * FROM alkalmazott
        ORDER BY premium;

SELECT t_kod, a_nev, fizetes+NVL(premium, 0) jovedelem
                FROM alkalmazott
                ORDER BY 1 DESC, jovedelem;
```

2.3. Sorok kiválasztása, lekérdezés keresési feltétellel

SELECT ...
FROM ...
WHERE feltétel
…;

Csak a feltételnek eleget tevő (feltétel logikai értéke igaz) sorok vesznek részt a további műveletekben. Mindegyik alaptípus szerepelhet a feltételt alkotó összehasonlító műveletekben.

Egyszerű feltételek tipikus formái:

o_kifejezés relációs_operátor {o_kifejezés | (egy_értéket_adó_alselect)}
relációs_operátorok (théta operátorok): =, !=, <>, <, >, <=, >=

o_kifejezés [NOT] BETWEEN kif1 AND kif2
kif1 és kif2 közé esés (zárt intervallum)
(kif1 <= o_kifejezés **<=** kif2)

o_kifejezés [NOT] LIKE 'karakterminta'
illeszkedik-e a megadott karaktermintára. Dzsóker karakterek:
% tetszőleges hosszú karaktersorra illeszkedés az adott pozíciótól,
_ tetszőleges karakterre illeszkedés az adott pozícióban

o_kifejezés IS [NOT] NULL

NULL értékkel való egyezés vizsgálata
(o_kifejezés=NULL helytelen, mindig NULL lesz az eredmény)

NULL érték: Amíg egy mezőnek nincs definiált értéke (mert ismeretlen vagy nincs is jelentése az adott helyzetben), addig **NULL** értékként kezeli az adatbázis-kezelő, ami általában nem tekinthető sem nullának, sem más értéknek.
A NULL értékkel nem lehet számolni (definiálatlan, azaz NULL lesz a kifejezés eredménye vagy a függvény értéke) és a legtöbb csoportfüggvény ignorálja (kihagyja) a NULL értéket. Az **NVL** függvényt kell használnunk ahhoz, hogy átmenetileg konvertáljuk egy NULL értékű kifejezés értékét egy kezelhető értékké:

NVL(o_kifejezés, helyettesítő_érték)

A függvény eredménye azonos az első argumentum által meghatározott értékkel, ha az nem NULL, ellenkező esetben a második argumentum értékét adja vissza.

```
SELECT a_nev, fizetes+premium jövedelem
      FROM alkalmazott;
SELECT a_nev, fizetes+NVL(premium,0) AS "jövedelem értéke"
      FROM alkalmazott;
```

Azon alkalmazottak adatai (név, fizetés, prémium, fizetés+prémium), amelyekre teljesül a WHERE feltétel:

```
SELECT a_nev, fizetes, premium, fizetes+NVL(premium, 0) jövedelem
      FROM  alkalmazott
      WHERE beosztas = 'SZERELO';          -- beosztásuk szerelő
v.    WHERE belepes > '01-JAN-80';         -- adott dátum után léptek be
v.    WHERE fizetes > (SELECT AVG(fizetes)
      FROM alkalmazott);                   -- fizetésük nagyobb az átlagfizetéstől
v.    WHERE fizetes = (SELECT fizetes
      FROM alkalmazott WHERE a_kod='1234');        -- jó, 1 értéket ad vissza
v.    WHERE fizetes = (SELECT fizetes
      FROM alkalmazott WHERE beosztas='ELADO')     -- hiba, több értéket ad vissza
v.    WHERE fizetes BETWEEN 20000 AND 30000;       -- 20000 ≤ fizetés ≤ 30000
v.    WHERE a_nev LIKE 'H%';               -- nevük H betűvel kezdődik
v.    WHERE a_nev LIKE '_O%';              -- nevük 2. betűje O
v.    WHERE premium IS NULL;               -- prémium értéke definiálatlan
v.    WHERE premium = NULL;                -- helytelen, nem ad vissza egyetlen sort sem
```

"Halmazos" feltételek tipikus formái:

"Halmaz" lehet:

(kifejezés lista)
(alselect) -- több oszlopot és sort is visszaadhat

o_kifejezés [NOT] IN (halmaz)

a megadott halmazban szerepel-e?

o_kifejezés relációs_operátor {ANY|SOME} (halmaz)

teljesül-e a reláció a halmaz valamely (legalább egy) elemére?
(megj: =ANY azonos az IN relációval, ANY és SOME azonos)

o_kifejezés relációs_operátor ALL (halmaz)

teljesül-e a reláció a halmaz minden egyes (összes) elemére?
(megj: <>ALL azonos a NOT IN relációval)

(o_kifejezés_lista) előbbi_műveletek_egyike (halmaz)

⇑ relációs operátor itt csak =, <> lehet

[NOT] EXISTS (alselect)

az alselect visszaad-e legalább egy sort?

Azon alkalmazottak adatai, amelyekre teljesül a WHERE feltétel:

```
SELECT  *  FROM  alkalmazott
        WHERE beosztas IN ('UGYINTEZO', 'SZERELO', 'ELADO');
v.      WHERE fizetes IN (SELECT DISTINCT fizetes
                FROM alkalmazott WHERE t_kod='40');
v.      WHERE fizetes =ANY (SELECT DISTINCT fizetes
                FROM alkalmazott WHERE t_kod='40');
v.      WHERE fizetes > ALL (SELECT DISTINCT fizetes
                FROM alkalmazott WHERE t_kod='40');
v.      WHERE (fizetes, NVL(premium, 0)) = ANY
                ((13000,4000),(20000, 0));
v.      WHERE (fizetes, NVL(premium, 0)) IN
                (SELECT fizetes, NVL(premium, 0)
              FROM alkalmazott WHERE t_kod='40');
```

Azon telephelyek adatai, ahol nincs még alkalmazott:

```
SELECT * FROM telephely T
        WHERE NOT EXISTS (SELECT * FROM alkalmazott A
                                WHERE T.t_kod=A.t_kod);
```

Összetett keresési feltételek:

Az eddigi feltételeket logikai operátorokkal kapcsolhatjuk össze.
Logikai operátorok: **NOT, AND, OR**

Logikai értékek: True, False, NULL (az SQL-ben csak a NULL van expliciten használva)

Igazságtáblázatok:

AND	True	False	Null
True	T	F	N
False	F	F	F
Null	N	F	N

OR	True	False	Null
True	T	T	T
False	T	F	N
Null	T	N	N

NOT	
True	F
False	T
Null	N

T: True
F: False
N: Null

```
SELECT a_nev, beosztas, t_kod
        FROM alkalmazott
        WHERE fizetes NOT BETWEEN 20000 AND 30000
                AND beosztas NOT IN('TELEPHELYVEZETO', 'ELADO');
SELECT a_kod, a_nev, premium
        FROM alkalmazott
        WHERE premium IS NULL  OR  t_kod='20';
```

A műveletek precedenciája (kiértékelés sorrendje):

Megadjuk az operátor csoportokat a precedenciájuk csökkenő sorrendjében, ettől zárójelezéssel lehet eltérni. Azonos precedenciájú műveletek esetén a balról-jobbra szabály érvényes.

1. Aritmetikai operátorok (*, /, +, -)
   (közöttük a sorrend és a kötési irány a szokásos)
2. Karakteres operátor (||)
3. Összehasonlító operátorok (=, !=, <>, <, >, <=, >=,
   [NOT] IN, ANY, ALL, [NOT] BETWEEN, [NOT] EXISTS
   [NOT] LIKE, IS [NOT] NULL)
   (precedenciájuk azonos)
4. Logikai operátorok (NOT, AND, OR)
   (közöttük a sorrend és a kötési irány szokásos)

```
SELECT a_nev, t_kod, fizetes, 100+fizetes*12
        FROM alkalmazott
        WHERE t_kod=40  OR  t_kod=30 AND fizetes+100>20000;

SELECT a_nev, t_kod, fizetes, 100+fizetes*12
        FROM alkalmazott
        WHERE (t_kod=40  OR  t_kod=30) AND fizetes+100>20000;
```

Halmazműveletek selectek között:

Két kompatíbilis lekérdezés (eredménytábla oszlopainak száma egyezik és típus szerint is rendre kompatibilisek) halmazműveletekkel kapcsolható össze:

UNION - Egyesítés eredménye: legalább az egyik táblában előforduló sorok, sor duplikáció nincs.

UNION ALL - Egyesítés eredménye: a táblákban előforduló sorok, sor duplikációt megenged.

INTERSECT - Metszet eredménye: mindkét táblában előforduló közös sorok.

MINUS - Kivonás eredménye: az első táblából elhagyjuk a második táblában előforduló sorokat.

Vegyük az unióját (vagy a metszetét vagy a különbségét) a telephely táblából származó t_kod értékek halmazának és az alkalmazott táblából származó t_kod értékek halmazának:

```
SELECT t_kod FROM telephely
  UNION        -- vagy UNION ALL, vagy MINUS, vagy INTERSECT
  SELECT t_kod FROM alkalmazott
ORDER BY t_kod;
```

Azon dolgozók adatai, akik prémiuma megegyezik BOGNAR nevűek prémiumával vagy TOTH nevűek fizetésével:

Két részhalmaz egyesítésével:

```
SELECT a_nev, beosztas, fizetes, NVL(premium, 0)
    FROM alkalmazott
    WHERE NVL(premium, 0) IN
    (SELECT NVL(premium, 0) FROM alkalmazott
                        WHERE a_nev = 'BOGNAR'
    UNION
    SELECT fizetes FROM alkalmazott
                        WHERE a_nev = 'TOTH');
```

Nem egyesítve a két részhalmazt:

```
SELECT a_nev, beosztas, fizetes, NVL(premium, 0)
    FROM alkalmazott
    WHERE NVL(premium, 0) IN
    (SELECT NVL(premium, 0) FROM alkalmazott
                        WHERE a_nev = 'BOGNAR')
    OR  NVL(premium, 0) IN
    (SELECT fizetes FROM alkalmazott
                        WHERE a_nev = 'TOTH');
```

2.4. Származtatott adatok, SQL sor-függvények

Származtatott adatok azok, amelyeket a tábla adatbázisban tárolt adataiból lehet kiszámítani kifejezések segítségével. Ezeket oszlopkifejezésnek is szokás nevezni és az SQL utasításokban több helyen megengedett a használatuk.
Az **oszlopkifejezés** oszlopnevek, konstansok és függvényhívások összekapcsolva aritmetikai operátorokkal (*, /, +, -) vagy konkatenáló karakteres operátorral (||), zárójelezés is megengedett.

A függvények részletes leírása megtalálható a referencia kézikönyvekben. A legtöbb függvény a NULL értékekkel nem tud számolni, NULL lesz a függvény értéke. A NULL érték kezelésére az **NVL** függvény alkalmazható, amelyet már korábban ismertettünk. A függvényhívások egymásba ágyazhatók.

Típusai:
Sor-függvények (Row Functions): soronként egy eredményt adnak vissza.
Csoport-függvények (Aggregate Functions)**:** a sorok egy csoportjára egy értéket adnak vissza.
(ezeket egy külön fejezetben ismertetjük).

A **sor függvények** többsége a legtöbb programozási nyelvben megtalálható. Ezek minden sorra külön kiértékelődnek, függetlenül a többi sortól és soronként egy eredményt adnak vissza.
Egy egyszerű sor függvény kipróbálásához a **DUAL** rendszertáblát használhatjuk, amelynek egy sora és egy oszlopa van:

```
SELECT  FV_NÉV(arg1, …),…  FROM DUAL
```

Aritmetikai függvények: ABS, POWER, SQRT, SIGN, ROUND, TRUNC, MOD, FLOOR, CEIL, SIN, COS, TAN, stb.

POWER (3, 2)	→	9
ROUND (45.923, 2)	→	45.92
ROUND (45.923, 0)	→	46

ROUND (45.923, -1) → 50
TRUNC (45.923, 2) → 45.92
TRUNC (45.923) → 45
TRUNC (45.923, -1) → 40
MOD (1600, 300) → 100

Karakteres függvények: LOWER, UPPER, INITCAP, LENGTH, SUBSTR, LPAD, RPAD, LTRIM, RTRIM, CHR, ASCII, stb.

Pl.:
```
SELECT * FROM alkalmazott
WHERE LOWER(beosztas)='elado';
```

INITCAP ('alma') → Alma
UPPER ('alma') → ALMA
LOWER('Alma') → alma
LENGTH ('alma') → 4
SUBSTR ('alma', 1, 3) → alm
INSTR ('xyalmaxyalmaxyxy', 'xy', 3, 2) → 13
LPAD ('alma', 6, '*') → **alma
RPAD ('alma', 6, '*') → alma**
LTRIM ('xyxXalmax', 'xy') → Xalmax
RTRIM ('xalmaXxyx', 'xy') → xalmaX

Dátum függvények: SYSDATE, ADD_MONTHS, MONTHS_BETWEEN, LAST_DAY, TRUNC, ROUND, stb.

A teljes **dátum** és **idő** tárolásra kerül az adatbázisban speciális belső formában:
évszázad, év, hónap, nap, óra, perc, másodperc

A standard dátum megadási formátum és nyelv beállítható. Az adatszótárból lekérdezhetők az aktuális beállítások: **SELECT * FROM v$nls_parameters;**
'DD-MON-YY' pl.: '03-MAR-94' angol nyelven
'YY-MON-DD' pl.: '94-MÁR-03' magyar nyelven
Adatfelvitelnél az ilyen formában megadott dátumok tárolásához az évszázadot az aktuális évszázadnak, az időpontot pedig éjfélnek veszi.

YY helyett RR is alkalmazható: 'DD-MON-RR', 'RR-MON-DD', ebben az esetben az évszázad képzése a következő:

		A 2 számjeggyel megadott év:	
		00-49	**50-99**
Az aktuális év utolsó 2 számjegye:	**00-49**	Aktuális évszázad	Előző évszázad
	50-99	Következő évszázad	Aktuális évszázad

Műveletek: dátumok kivonása -- eredmény napokban,
dátumhoz nap hozzáadása, kivonása -- eredmény dátum,
dátumok összehasonlítása.

```
SELECT SYSDATE, SYSDATE+7, SYSDATE-7
FROM DUAL;

SELECT belepes, SYSDATE-belepes
FROM alkalmazott
```

```
WHERE belepes > '01-JAN-83';
```

```
SYSDATE                                  →    aktuális dátum és idő
ADD_MONTHS ('21-JUN-74', 2)              →    21-AUG-74
MONTHS_BETWEEN ('01-SEP-95', '11-JAN-94') →19.6774194
LEAST ( '01-JUL-94', '13-JUL-94' )       →    01-JUL-94 a legkisebb
NEXT_DAY(SYSDATE, 'Friday')              →a dátumhoz legközelebbi péntek dátuma
LAST_DAY(SYSDATE)                        →a dátum hónapjának utolsó napja
```

Konverziós függvények - explicit adatkonverzió:
Implicit adatkonverzió: a rendszer ha tudja, akkor szükség esetén automatikusan átalakítja az egyik típusú adatot egy másik típusra:

```
SELECT a_nev, belepes, t_kod, fizetes+'10'
FROM alkalmazott
WHERE belepes>'80-jan-01' OR t_kod>40;
```

TO_CHAR(dátum_kif, 'formátum'), TO_CHAR(szám_kif, ' formátum '),
TO_DATE('dátum_kar', ' formátum '), TO_NUMBER('szám_kar', ' formátum ')

Szám formátumban használható elemek: 9 (szám) 0 (zérus)
. (tizedespont) , (ezredes elválasztók) megjelenítésére, kezelésére, stb.

```
SELECT a_nev, TO_CHAR(fizetes, '099,999.99')
        FROM alkalmazott;

SELECT TO_CHAR(TO_NUMBER('123,456', '999,999'), '999999.00') FROM DUAL;
```

Dátum formátumban használható elemek:
Century, **Y**ear, **M**onth, **D**ay, **H**our, **MI**nute, **S**econd, **W**eek, **Q**uarter, stb.

Az angol szavak kezdőbetűjét használjuk a dátumelemek számokkal való leírásának megformázásához. Egyes dátumelemek (**Y**ear, **M**onth, **D**ay) teljes kiírását vagy annak 3 betűs rövidítését (MON, DY) is kérhetjük, valamint a kis- vagy nagybetűs írásmód is jelölhető.

```
SELECT a_nev, TO_CHAR(belepes, 'CC   YYYY.MM.DD   HH24:MI:SS')
        FROM alkalmazott;

SELECT                                          -- TO_DATE nélkül hibajelzés
        ROUND(TO_DATE('20-NOV-93', 'DD-MON-YY'), 'YYYY'),       -- 01-JAN-94
          TRUNC(TO_DATE('20-NOV-93', 'DD-MON-YY'), 'YYYY'),     -- 01-JAN-93
        ROUND(TO_DATE('20-NOV-93', 'DD-MON-YY'), 'MM'),         -- 01-DEC-93
          TRUNC(TO_DATE('20-NOV-93', 'DD-MON-YY'), 'MM')              -- 01-NOV-93
        FROM DUAL;

SELECT SYSDATE, TO_CHAR(SYSDATE,
        ' "DATUM=" YYYY.MM.DD "   NAP="DAY "   NEGYEDEV="Q "HET="WW' )
        FROM DUAL;

SELECT SYSDATE, TO_CHAR(SYSDATE,
        'Year  Month  Day    Mon  Dy')
        FROM DUAL;
```

Egy adott dátum (pl. a születésnapunk) napját írassuk ki betűvel:

```
SELECT TO_CHAR(TO_DATE('1982-01-31', 'YYYY-MM-DD'), 'YYYY-MM  Day')
      FROM DUAL;
```

Egyéb függvények: NVL, GREATEST, LEAST, DECODE, stb.

```
SELECT a_nev, fizetes, premium, GREATEST(fizetes, NVL(premium,0))
      FROM alkalmazott;
SELECT a_nev,
      DECODE(beosztas, 'IGAZGATO', 'Főnök', 'ELADO', 'Árus', 'Egyéb')  Foglalkozás
      FROM alkalmazott;
```

2.5. Csoportok képzése, SQL csoport-függvények

A kiválasztott sorok csoportosítására, csoportokon műveletek végzésére és a csoportok közötti válogatásra alkalmazható.

SELECT ...
FROM ...
...
GROUP BY o_kifejezés ,...
[HAVING csoportkiválasztási_feltétel]
…;

A **GROUP BY** után megadott oszlopkifejezések (legtöbbször ez egy vagy több oszlopnév) határozza meg azt, hogy a sorok csoportosítása mely oszlop(ok) értéke szerint történjen. A **HAVING** részben adható meg a csoportok közüli válogatás feltétele.

A létrejövő csoportokra különféle beépített, úgynevezett csoportfüggvények használhatók:
AVG( ), MIN( ), MAX( ), COUNT( ), SUM( ), stb.

Alakja: **FV_NÉV** ([DISTINCT | ALL] o_kifejezés)
COUNT ({ * | [DISTINCT | ALL] o_kifejezés})

A csoportfüggvények a számolásnál ignorálják (kihagyják) a NULL értéket, kivéve a COUNT(*).
Ha nincs GROUP BY egy SELECT-ben, akkor egyetlen csoportot képez a teljes eredmény:

```
SELECT AVG(fizetes), AVG(DISTINCT fizetes), MIN(fizetes), MAX(fizetes), COUNT(fizetes),
SUM(fizetes)
      FROM alkalmazott;

SELECT COUNT(*), COUNT(premium), COUNT(beosztas), COUNT(DISTINCT beosztas)
      FROM alkalmazott;

SELECT AVG(premium), AVG(NVL(premium,0))
      FROM alkalmazott;

SELECT * FROM alkalmazott
  WHERE fizetes > (SELECT AVG(fizetes) FROM alkalmazott);  -- akik fizetése nagyobb az átlagfizetéstől
```

Egy csoportot képeznek azon sorok, amelyekben a GROUP BY után álló csoportképző oszlopok (oszlopkifejezések) azonos értékűek. Egy-egy csoportból csak egy sor, a csoportazonosítóhoz tartozó "gyűjtősor" lesz visszaadva. Azon sorok is részt vesznek a csoportosításban, ahol csoportképző oszlop értéke NULL.

A SELECT és HAVING részben csak a következők szerepelhetnek:

- GROUP BY -ban szereplő oszlop (v. oszlopkifejezés ugyanolyan formában),
- csoportfüggvények, argumentumuk tetszőleges oszlopkifejezés is lehet,
- paraméter nélküli függvények (pl. SYSDATE) és konstansok.

Határozzuk meg az alkalmazottak telephelyenkénti csoportjaira (t_kod szerint képzett csoportokra) az átlagfizetést, a létszámot, a fizetések összegét, a minimális és a maximális fizetést:

```
SELECT  t_kod, AVG(fizetes)  átlag_fiz, COUNT(*) létszám, SUM(fizetes),
        MIN(fizetes), MAX(fizetes)
        FROM alkalmazott
        GROUP BY t_kod;
/* Itt t_kod mellett más oszlop lekérdezése hibás lenne */
```

Rendezzük az eredménysorokat:

```
SELECT …
        FROM alkalmazott
        GROUP BY t_kod
      ORDER BY t_kod DESC;
```

Csak azon csoportokat írassuk ki, ahol az átlagfizetés 22000-nél nagyobb:

```
SELECT …
        FROM alkalmazott
        GROUP BY t_kod
        HAVING AVG(fizetes)>22000;
```

Csak azon csoportokat írassuk ki, ahol az átlagfizetés nagyobb mint az összes dolgozó átlagfizetése:

```
SELECT …
        FROM alkalmazott
        GROUP BY t_kod
        HAVING AVG(fizetes)>(SELECT AVG(fizetes) FROM alkalmazott);
```

Csak azon csoportokat írassuk ki, ahol legalább hárman dolgoznak:

```
SELECT …
        FROM alkalmazott
        GROUP BY t_kod
      HAVING COUNT(*) >= 3;
```

Csak a 20000-nál nagyobb fizetésű alkalmazottak vegyenek részt a műveletben:

```
SELECT …
        FROM alkalmazott
        WHERE fizetes>20000
        GROUP BY t_kod;
```

Határozzuk meg az alkalmazottak telephelyenként, azon belül beosztásonként képzett csoportjaira átlagfizetést és a létszámot:

```
SELECT t_kod, beosztás, AVG(fizetes), COUNT(*)
        FROM alkalmazott
        GROUP BY t_kod, beosztas;
```

ORACLE8i verziótól kezdve lehetőség van az eddigi normál csoportosításon túl olyan **szupercsoportosítások** megadására is, amelyek a normál csoportosítás "gyűjtősorai" mellett még különböző szintű és fajta összegfokozatoknak megfelelő szupergyűjtő-sorokat is visszaadnak. Szintaktika:

GROUP BY {o_kif, … | **CUBE**(o_kif,...) | **ROLLUP**(o_kif,...) }
[HAVING csoportkiválasztási_feltétel]

ROLLUP(o_kif1, ... , o_kifN) elvégzi a csoportosításokat az **első N, N-1, N-2, … 0 darab o_kif szerint**. Ezek közül az első N darab o_kif szerinti adja az eddigi normál gyűjtősorokat, a többi pedig a szupergyűjtő-sorokat. Összesen **N+1** darab különböző csoportosítás van végrehajtva.

```
SELECT t_kod, beosztas, AVG(fizetes), COUNT(*)
        FROM alkalmazott
        GROUP BY ROLLUP(t_kod, beosztas);
```

CUBE(o_kif1, ... , o_kifN) elvégzi a csoportosításokat az **o_kif -ek összes lehetséges kombinációja szerint.** Ezek közül az első N darab o_kif szerinti adja az eddigi normál gyűjtősorokat, a többi pedig a szupergyűjtő-sorokat. Összesen $\mathbf{2^N}$ darab különböző csoportosítás van végrehajtva.

```
SELECT t_kod, beosztas, AVG(fizetes), COUNT(*)
        FROM alkalmazott
        GROUP BY CUBE(t_kod, beosztas);
```

Mindkét esetben a szupergyűjtő-sorokban az "összesen oszlopérték" egy speciális NULL értékkel van reprezentálva, melyet egy speciális függvénnyel tudunk kezelni:
GROUPING(o_kif) = 1 ha o_kif **"**összesen oszlopértéket" reprezentáló speciális NULL érték,
= 0 egyéb érték és a normál NULL esetén.

```
SELECT  premium, GROUPING(premium), AVG(premium), COUNT(*)
                FROM alkalmazott
                GROUP BY ROLLUP(premium);
/*Kiegészíthető: */  HAVING GROUPING(premium)=1;      -- csak a szupersorokat adja vissza
/*vagy:        */     HAVING premium IS NULL;          -- szupersorok és a premium IS NULL sorok

SELECT DECODE(GROUPING(t_kod), 1, 'Össz t_kód', t_kod) AS  t_kód,
        DECODE(GROUPING(beosztas), 1, 'Össz beosztás', beosztas) AS  beosztás,
        COUNT(*) "Alk. szám", AVG(fizetes) * 12 "Átlag fiz"
                FROM alkalmazott
                GROUP BY CUBE (t_kod, beosztas);
/*Kiegészíthető: */  HAVING GROUPING(t_kod)=1 OR GROUPING(beosztas)=1;
```

T_KÓD	BEOSZTÁS	Alk. szám	Átlag fiz
10	IGAZGATO	1	648000
10	Össz beosztás	1	648000
20	ELADO	1	156000
20	TELEPHELYVEZETO	1	426000
20	VIZSGABIZTOS	1	240000
20	Össz beosztás	3	274000
30	ELADO	1	159000
30	SZERELO	1	216000
30	TELEPHELYVEZETO	1	348000
30	Össz beosztás	3	241000
40	ELADO	2	141000
40	TELEPHELYVEZETO	1	450000
40	VIZSGABIZTOS	1	252000
40	Össz beosztás	4	246000
50	ELADO	1	156000
50	SZERELO	1	264000

```
50          TELEPHELYVEZETO          1     390000
50          Össz beosztás            3     270000
60          SZERELO                  1     252000
60          TELEPHELYVEZETO          1     300000
60          Össz beosztás            2     276000
Össz t_kód ELADO                     5     150600
Össz t_kód IGAZGATO                  1     648000
Össz t_kód SZERELO                   3     244000
Össz t_kód TELEPHELYVEZETO           5     382800
Össz t_kód VIZSGABIZTOS              2     246000
Össz t_kód Össz beosztás            16   283687.5
```

```
SELECT DECODE(GROUPING(t_kod), 1, 'Össz t_kód', t_kod) AS  t_kód,
        DECODE(GROUPING(beosztas), 1, 'Össz beosztás', beosztas) AS  beosztás,
        COUNT(*) "Alk. szám", AVG(fizetes) * 12 "Átlag fiz"
                FROM alkalmazott
                GROUP BY ROLLUP (t_kod, beosztas)
/*Kiegészíthető: */   HAVING GROUPING(t_kod)=1 OR GROUPING(beosztas)=1;
```

2.6. Egymásba ágyazott SELECT

Mint már korábban láttuk az SQL nyelvben megengedett, hogy egy SELECT utasításban (vagy más SQL utasításban) SELECT-ek (alselect) is előforduljanak (mélység maximálva van).

Típusai: Egyszerű és Korrelált (vagy kapcsolt) típus

Egyszerű típus: a belső SELECT önmagában is kiértékelhető és belülről kifelé haladva lesznek feldolgozva. A kiértékelés menete:

- a belső SELECT kiértékelődik és egy, vagy több sort vagy oszlopértéket átad a külső SELECT-nek,
- a külső SELECT ezen értékek alapján összeállítja az eredményt.

```
Egy értéket (egy sor, egy oszlop) ad át a külsőnek:
(azon alkalmazottak, akiknek fizetése nagyobb az összes alkalmazott átlagfizetéstől)
SELECT *  FROM  alkalmazott
            WHERE fizetes > (SELECT AVG(fizetes)  FROM alkalmazott);
```

```
Több sort ad át a külsőnek:
(azon telephelyek adatai, amelyeken nincs még alkalmazott)
SELECT *     FROM telephely
        WHERE t_kod NOT IN (SELECT DISTINCT t_kod
                                    FROM alkalmazott);
```

Ha **több sorból** egy vagy több értéket adunk át a külsőnek, akkor csak a **"halmazos" operátorokat** használhatjuk: [NOT] IN, relációs_operátor ANY, relációs_operátor ALL, [NOT] EXISTS

```
Több sorból több értéket ad át a külsőnek:
(azon alkalmazottak adatai, akiknek fizetése és prémiuma nem egyezik meg az ELADO beosztásúak
fizetésével és prémiumával)

SELECT  a_nev, beosztas, fizetes, NVL(premium,0)
        FROM alkalmazott
       WHERE (fizetes, NVL(premium, 0)) NOT  IN
                    (SELECT DISTINCT fizetes, NVL(premium, 0)  FROM alkalmazott
                    WHERE beosztas='ELADO');
```

Korrelált (kapcsolt) típus: a belső SELECT olyan, a külső SELECT-re történő hivatkozást tartalmaz, amely miatt a belső önmagában nem kiértékelhető. A kiértékelés menete:

- a külső SELECT átad egy sort a belsőnek,
- a belső SELECT így már kiértékelhető, visszaadja az eredménysort vagy sorokat,
- a külső SELECT elvégzi a további értékelést.

Ez ismétlődik a külső SELECT minden sorára, míg összeáll az eredmény.

Amennyiben a külső és a belső SELECT is ugyanazt a táblát használja, a belső SELECT csak aliasnévvel tud a külső SELECT táblájára hivatkozni.

```
Azon dolgozók adatai, akik fizetése nagyobb a telephelyükön dolgozók átlagfizetésétől:
SELECT a_nev, fizetes, t_kod
        FROM alkalmazott X
        WHERE fizetes > (SELECT AVG(fizetes)
                                FROM alkalmazott
                                WHERE t_kod = X.t_kod);

Azon telephelyek adatai, ahol nincs alkalmazott:
SELECT * FROM telephely
        WHERE NOT EXISTS (SELECT * FROM alkalmazott
                                WHERE telephely.t_kod=t_kod);
```

2.7. Táblák összekapcsolása (Join)

A relációs modellben a táblák közötti kapcsolat megvalósítása külső kulccsal (kapcsolóoszloppal) történik. Ezzel adható meg, hogy az egyik tábla (szülő tábla) egy adott sorához egy másik tábla (gyerek tábla) mely sorai tartoznak. Külső kulcs (Foreign Key): az egyik táblában lévő kapcsoló-oszlop(ok), amely hivatkozik egy másik tábla elsődleges (Primary Key) kulcsára (vagy egy egyedi Unique kulcsára). Egy vagy több oszlop alkothatja. A hivatkozott tábla általában egy másik tábla, de lehet ugyanaz. Lásd még a CREATE TABLE utasítást. A külső kulcs értéke csak létező kulcs értékre hivatkozhat, vagy NULL értékű.

A SELECT utasításban a táblákat össze kell kapcsolni, ha több táblából származó adatokat akarunk visszanyerni.

```
SELECT ...
   FROM táblanév1 [aliasnév1], táblanév2 [aliasnév2] ,...
   WHERE kapcsolóoszlop1 összehasonlító_operátor kapcsolóoszlop2
            [AND további_kapcsoló_feltétel]
            [{AND | OR} egyéb_feltétel]
   …;
```

A join műveletben szereplő táblák száma nincs korlátozva. A táblák valódi és nézettáblák egyaránt lehetnek. Az oszlopneveket minősíteni kell ha több táblában van azonos nevű oszlop. Az itt jelzett összekapcsolást belső (inner join) összekapcsolásnak is szokás nevezni és az SQL szabvány más típusú szintaktikát is megenged. A join típusai a kapcsolófeltétel alapján lehet:

- egyenlőségen alapuló - equijoin
- nem egyenlőségen alapuló - nem equijoin

Két tábla összekapcsolása **egyenlőségen** alapuló összekapcsolással:

```
Az alkalmazottak és telephelyük adatai:
SELECT a_nev, telephely.t_kod, t_nev, cim
```

```
        FROM alkalmazott, telephely
        WHERE alkalmazott.t_kod = telephely.t_kod;
```

Az előző alias nevekkel:

```
SELECT a_nev,  T.*
        FROM alkalmazott  A, telephely  T
        WHERE A.t_kod = T.t_kod;
```

Az alkalmazottak és közvetlen főnökük adatai:

```
SELECT  X.t_kod, X.a_nev , Y.a_nev, Y.t_kod
        FROM alkalmazott  X, alkalmazott  Y
        WHERE X.fonok = Y.a_kod;
```

Az alkalmazottak és közvetlen főnökük valamint a közvetlen főnökük telephelyének adatai:

```
SELECT  X.t_kod, X.a_nev , Y.a_nev, Y.t_kod, t_nev, cim
        FROM alkalmazott  X, alkalmazott  Y, telephely T
        WHERE  X.fonok = Y.a_kod  AND  Y.t_kod=T.t_kod;
```

Az összekapcsolt táblák soraira egyéb szűrőfeltételek is alkalmazhatók, a sorok csoportosíthatók, rendezhetők:

```
SELECT t_nev, beosztas, COUNT(*) "Alk. szám", AVG(fizetes)*12 "Átlag jöv"
        FROM alkalmazott A, telephely T
        WHERE A.t_kod = T.t_kod  AND fizetes>20000
        GROUP BY t_nev, beosztas
        HAVING  t_nev LIKE('%AUTO%')
      ORDER BY t_nev DESC, beosztas DESC;
```

Egy tábla önmagával való összekapcsolása **nem egyenlőségen** alapuló joinnal:
(Elengedhetetlen a tábla-aliasnév használata.)

Azon alkalmazottak akiknek a fizetése több mint KOVACS fizetése:

```
SELECT X.a_nev, X.fizetes, Y.a_nev, Y.fizetes
        FROM alkalmazott  X, alkalmazott  Y
        WHERE X.fizetes > Y.fizetes AND Y.a_kod = '1234';
```

Általában egymásba ágyazott SELECT-el is megoldhatók egyes joinos feladatok:

```
SELECT a_nev, fizetes
        FROM alkalmazott
        WHERE fizetes > (SELECT fizetes FROM alkalmazott WHERE a_kod = '1234');
```

Azon alkalmazottak adatai, akik fizetése több mint a velük azonos beosztásúak átlagfizetése:

```
SELECT a_kod, a_nev, beosztas, fizetes  FROM alkalmazott  X
        WHERE fizetes > (SELECT AVG(fizetes) FROM alkalmazott  Y
                                WHERE X.beosztas=Y.beosztas);
```

Írassuk ki az azonos beosztásúak átlagfizetését is:
(Megj. GROUP BY -ban X.a_kod már elvégzi a szükséges csoportosítást, azon belül X.a_nev, X.beosztas, X.fizetes újabb csoportokat nem hoz létre, de fel kell venni, hogy a SELECT után beírható legyen.)

```
SELECT X.a_kod, X.a_nev, X.beosztas, X.fizetes, AVG(Y.fizetes)
        FROM alkalmazott  X, alkalmazott  Y
        WHERE  X.beosztas=Y.beosztas
        GROUP BY X.a_kod, X.a_nev, X.beosztas, X.fizetes
        HAVING X.fizetes > AVG(Y.fizetes);
```

ORACLE8i-től a **táblanevek helyett szelekciós utasítás** is állhat, mely nem tartalmaz a szelekciós utasításon kívüli hivatkozásokat. Ez újabb típusú feladatok megoldására és más megoldások kialakítására ad lehetőséget.

FROM {táblanév | (alselect)} [t_aliasnév] , …

Az előző feladat megoldása:

```
SELECT a_kod, a_nev, X.beosztas, fizetes, cs_atlag
        FROM alkalmazott  X,   (SELECT beosztas, AVG(fizetes) cs_atlag FROM alkalmazott
                                GROUP BY beosztas) CS
        WHERE fizetes > cs_atlag AND X.beosztas=CS.beosztas;
```

Írassuk ki, hogy egy-egy telephelyen dolgozók létszáma és fizetése hány %-át teszi ki az összes alkalmazott létszámának és fizetésének:

```
SELECT A.t_kod "Telephely",  (A.csop_szam/B.ossz_szam)*100   "% alkalmazott",
                             (A.csop_fiz/B.ossz_fiz)*100   "% fizetés"
    FROM   (SELECT t_kod, COUNT(*) csop_szam, SUM(fizetes) csop_fiz  FROM alkalmazott
                        GROUP BY t_kod) A,
           (SELECT COUNT(*) ossz_szam, SUM(fizetes) ossz_fiz  FROM alkalmazott) B ;
```

Külső join, olyan join, amelynél ha az egyik táblából származó kapcsolóoszlopbeli értékhez nincsen kapcsolható sor a másik táblában, ilyenkor NULL értékeket tartalmazó sorral egészíti ki az egyik táblából származó sorokat. Ezeket a sorokat a normál belső join nem adná vissza.
Azon tábla kapcsolóoszlopát jelöljük meg **(+)** jellel, a külső join műveleti jelével, amelyből NULL értékű sorral való kiegészítést kérünk. Mindkettőt megjelölni nem lehet.

Listázzuk ki a telephelyek adatait és az ott dolgozókat névsor szerint, azon a telephelyek is legyenek kiírva ahol még nem dolgozik senki:

```
SELECT  T.t_kod, t_nev, cim, a_nev
        FROM telephely  T, alkalmazott  A
        WHERE T.t_kod = A.t_kod (+)
      ORDER BY T.t_kod, a_nev;
```

Listázzuk ki a telephelyek adatait és hogy egy-egy telephelyen hányan dolgoznak:
(Megj. GROUP BY -ban T.t_kod már elvégzi a szükséges csoportosítást, azon belül t_nev, cím újabb csoportokat nem hoz létre, de fel kell venni, hogy a SELECT után beírható legyen.)

```
SELECT  T.t_kod, t_nev, cim, COUNT(a_nev)
        FROM telephely  T, alkalmazott  A
        WHERE T.t_kod = A.t_kod (+)
      GROUP BY T.t_kod, t_nev, cim;
```

Írassuk ki az alkalmazottak és a közvetlen főnökük nevét. Akinek nincs főnöke ott a '----' karaktersor jelenjen meg:

```
SELECT  X.a_nev, NVL(Y.a_nev, '------')
        FROM alkalmazott  X, alkalmazott  Y
        WHERE X.fonok = Y.a_kod (+);
```

Néhány további ORACLE8i kiegészítés:
ORACLE8i-től **oszlopkifejezésekben állhat skalár értékű alselect** (egy sorból egy oszlopértéket ad vissza, ha nincs visszaadott sor akkor értéke NULL, ha több sort adna vissza akkor hibajelzést ad), az alselect korrelált is lehet. Ez a skalár értékes alselect a legtöbb SQL utasításban használható, ahol oszlopkifejezés áll a szintaxisban:

Írassuk ki az alkalmazott nevét, fizetését, valamint az alkalmazottal azonos beosztásúak átlagfizetéséből számolt értéket:

```
SELECT a_nev, fizetes, ( SELECT AVG(fizetes) FROM alkalmazott
                    WHERE X.beosztas=beosztas)+1000 AS "Csop fiz + 1000"
FROM alkalmazott X;
```

ORACLE8i-től **oszlopkifejezésekben állhat CASE kifejezés:**

CASE WHEN feltétel **THEN** return_kif
[**WHEN** feltétel **THEN** return_kif] … -- az első igaz WHEN return_kif -t adja,
[ELSE return_kif**] END** -- egyébként ezt a return_kif -t vagy NULL-t ad vissza

```
SELECT a_nev, premium, premium+1000,
        1000 + CASE WHEN premium > 20000 THEN premium
        WHEN premium IS NULL THEN 0 ELSE 20000 END "CASE prémium"
FROM alkalmazott;

SELECT a_nev, fizetes, premium,
        CASE WHEN fizetes > 20000 THEN 'Magas fiz'
        WHEN premium IS NULL THEN 'Null prémium' ELSE 'Egyéb eset' END "CASE szöveg"
FROM alkalmazott;
```

2.8. Hierarchia lekérdezése ORACLE-ben

Az SQL nyelv ORACLE-ben használt bővítése lehetővé teszi, hogy egy táblában kódolt hierarchikus adatokat a hierarchiának megfelelő sorrendben lekérdezzük.

SELECT [LEVEL] ...
FROM táblanév
…
CONNECT BY {PRIOR o_kifejezés = o_kifejezés | o_kifejezés = **PRIOR** o_kifejezés}
[**START WITH** o_kifejezés = o_kifejezés]
…;

CONNECT BY - specifikálja a kapcsolatot a szülő és a gyerek sorok között, amely alapján a tábla sorait hierarchiába lehet kapcsolni.

PRIOR: a **PRIOR** operátoros kifejezés reprezentálja a **szülőt**, először ez lesz kiértékelve a kiválasztott sorra, majd a másik oldali kifejezés a tábla összes sorára. Azon sorok lesznek **a gyerek sorok**, amelyekre a feltétel igaz. (A fabejárási irányt határozza meg ezáltal.)

START WITH : melyik csomópont(ok)on induljon.

LEVEL: pszeudó oszlop, amely lehetővé teszi annak a szintnek a lekérdezését, amelyen az egyes adatok a hierarchiában szerepelnek (pszeudó oszlop: nincs a táblában tárolva, de úgy viselkedik mint egy normál oszlop)

Alkalmazottak fa-szerkezetű lekérdezése a gyökér felől (főnök ⇒beosztott irány) indulva
pl. SZABO összes beosztottja:

```
SELECT LEVEL, a_nev, beosztas, a_kod, fonok
        FROM alkalmazott
        CONNECT BY PRIOR a_kod = fonok START WITH a_nev = 'SZABO';
```

Ugyanez szépítve:

```
COLUMN nev FORMAT a30;
SELECT LEVEL, LPAD(' ', 2*LEVEL, ' ') || a_nev nev, beosztas, a_kod, fonok
  …;
```

```
     LEVEL NEV                            BEOSZTAS        A_KO FONO
---------- ------------------------------ --------------- ---- ----
         1   SZABO                        IGAZGATO        1239
         2     NEMETH                     TELEPHELYVEZETO 1238 1239
         3       MOLNAR                   SZERELO         1235 1238
         3       CSIKOS                   ELADO           1236 1238
         2     HORVAT                     TELEPHELYVEZETO 1248 1239
         3       HALASZ                   SZERELO         1247 1248
         2     NAGY                       TELEPHELYVEZETO 1250 1239
         3       KOVACS                   ELADO           1234 1250
         3       TOTH                     VIZSGABIZTOS    1237 1250
         2     PAPP                       TELEPHELYVEZETO 1251 1239
         3       KIRALY                   VIZSGABIZTOS    1244 1251
         3       HERCEG                   ELADO           1245 1251
         3       KISS                     ELADO           1249 1251
         2     KELEMEN                    TELEPHELYVEZETO 1252 1239
         3       BALOGH                   SZERELO         1240 1252
         3       BOGNAR                   ELADO           1246 1252
```

Több csomópontból indulva:

```
SELECT LEVEL, LPAD(' ', 2*LEVEL) || a_nev nev, beosztas, a_kod, fonok
        FROM alkalmazott
        CONNECT BY PRIOR a_kod = fonok START WITH beosztas = 'TELEPHELYVEZETO';
```

Levelek felől indulva (beosztott ⇒ főnök irány) pl. KISS összes főnöke:

```
SELECT LEVEL, LPAD(' ', 2*LEVEL) || a_nev nev, beosztas, a_kod, fonok
        FROM alkalmazott
        CONNECT BY a_kod = PRIOR fonok START WITH a_nev = 'KISS';
```

```
     LEVEL NEV                            BEOSZTAS        A_KO FONO
---------- ------------------------------ --------------- ---- ----
         1   KISS                         ELADO           1249 1251
         2     PAPP                       TELEPHELYVEZETO 1251 1239
         3       SZABO                    IGAZGATO        1239
```

Szintenkénti átlagfizetés:

```
SELECT LEVEL, AVG(fizetes) FROM alkalmazott
        CONNECT BY PRIOR a_kod = fonok START WITH a_nev = 'SZABO'
        GROUP BY LEVEL
        ORDER BY LEVEL;
```

3. DML (Adatmanipulációs, adatkezelő nyelv)

Adatok lekérdezése: SELECT, melyet korábban ismertettünk. Megj: a SELECT utasítást korábban szokás volt önálló **DQL** (**D**ata **Q**uery **L**anguage) lekérdező nyelvnek is nevezni.

Adatok karbantartása: INSERT, UPDATE, DELETE

Az utóbbiak elkészített táblák adatokkal való feltöltésére, létező adatok módosítására és törlésére alkalmasak:

INSERT - Új sor(ok) felvitele
UPDATE - Meglévő sor(ok) módosítása
DELETE - Meglévő sor(ok) törlése

Megjegyzések:

- A karbantartó műveletek által kezelt adatoknak ki kell elégíteni az integritási megszorításokat.
- Egy-egy karbantartó utasítással csak egy táblát kezelhetünk. Az utasításokban szereplő alselectek, más táblá(k)ra is hivatkozhatnak és korrelációban lehetnek a karbantartó utasítással kezelt táblával.
- A rendszer a táblákhoz kapcsolódó index-objektumokat automatikusan karbantartja.
- A korszerűbb RDBMS-ekben DB-ben tárolt triggerek (programok) készíthetők, amelyek a karbantartó műveletek végrehajtása előtt vagy után automatikusan aktivizálódnak és módosíthatják azok hatását.

3.1. Adatok lekérdezése - lásd. 2. fejezet

3.2. Adatok felvitele

INSERT INTO táblanév [aliasnév] [(oszlopnév , ...)]
{**VALUES** (érték , ...) | alselect};

- Oszlopnév, érték pároknak típusban és értéktartományban illeszkedni kell.
- Ha az oszlopnév felsorolás elmarad, akkor az összes oszlopra vonatkozik az értékadás abban a sorrendben, ahogy a CREATE TABLE utasításban szerepelnek az oszlopok.
- NULL értékek bevitele is lehetséges, ahol ez megengedett.
- Speciális értékek megadása kifejezéssel is történhet: SYSDATE+7, DEFAULT, , USER, stb. Skalár értéket adó alselect itt nem lehet. Dátumkonstansokat az alapértelmezett dátumformátum szerint vagy TO_DATE konverziót alkalmazva adhatunk meg. Ha az időpont elmarad az alapértelmezett idő éjfél.
- Az alselect más táblából nyert adatok felvitelét teszi lehetővé.
- **ORACLE8i-**től lehetséges adatfelvitel több táblába egy alselectből feltétel nélkül vagy feltétellel.

```
INSERT INTO telephely
            VALUES (80, 'IRODAK', 'DEBRECEN');
```

T_KOD: 70 hibás lenne.
ORA-00001: unique constraint (ORACLE21.SYS_C0025153) violated
T_KOD: NULL hibás lenne.
ORA-01400: cannot insert NULL into ("ORACLE21"."TELEPHELY"."T_KOD")

```
INSERT INTO alkalmazott
   VALUES('1999', 'PEPITA', 'IGAZGATO', '01-MAY-77', 54000, 75000, '10', NULL);
```

A_KOD: 1239, NULL hibás lenne.
T_KOD: 99, NULL hibás lenne.
FONOK: 1998 hibás lenne.
ORA-02291: integritás megszorítás (LNAGY15.SYS_C009313) megsértés, szülő kulcs nem található meg

NULL értékek felvitele hiányos oszloplistával vagy a NULL kulcsszóval:

```
INSERT  INTO  alkalmazott (a_kod, a_nev, beosztas, t_kod, fizetes)
            VALUES(1999, 'PEPITA', 'IGAZGATO', '10', 20000);

INSERT INTO telephelyx                    -- telephelyx táblának már létezni kell
            SELECT * FROM telephely
            WHERE cím < > 'BUDAPEST';
```

Évszázad és időpont explicit megadása:

```
INSERT INTO …
VALUES ( … TO_DATE('1982-01-31 08:00', 'YYYY-MM-DD HH24:MI'), …)

SELECT a_nev, TO_CHAR(belepes, 'YYYY-MM-DD HH24:MI:SS') FROM alkalmazott;
```

3.3. Adatok módosítása

UPDATE táblanév [aliasnév]
SET { oszlopnév = o_kifejezés | oszlopnév = (alselect) |
(oszlopnév ,…) = (alselect) } ,...
[**WHERE** feltétel];

- SET: adja meg, hogy mely oszlopok régi értékét kell módosítani és melyek az új értékek, amelyeket egy sort visszaadó alselect is szolgáltathat. NULL, DEFAULT is használható.
- WHERE feltétel: az UPDATE - ben résztvevő sorokat válogatja ki, enélkül a tábla összes sora részt vesz a műveletben.

A vidéki dolgozóknak adjunk 10%-os fizetésemelést:

```
UPDATE alkalmazott
        SET fizetes = 1.1*fizetes
        WHERE t_kod IN ( SELECT t_kod FROM telephely
                        WHERE cim < > 'BUDAPEST');
```

Helyezzük át KISS-t az 50-es osztályra és adjunk 1000 Ft. fizetésemelést.

```
UPDATE alkalmazott
        SET fizetes = fizetes + 1000, t_kod='50'
        WHERE a_nev = 'KISS';
```

Integritási megszorítási hibák:

Helyezzük át KISS-t az 99-es osztályra és adjunk 1000 Ft. fizetésemelést.

```
UPDATE alkalmazott
        SET fizetes = fizetes + 1000, t_kod='99'
        WHERE a_nev = 'KISS';
ORA-02291: integrity constraint (ORACLE21.SYS_C0025156) violated - parent key not found

UPDATE telephely SET t_kod=20 WHERE t_kod=40;
ORA-00001: unique constraint (ORACLE25.SYS_C0026262) violated
```

Primary Key nem updatelhető, ha már van rá hívatkozás, egyébként igen.

```
UPDATE telephely SET t_kod=99 WHERE t_kod=40;
ORA-02292: integrity constraint (ORACLE25.SYS_C0026265) violated - child record found

UPDATE telephely SET t_kod=71 WHERE t_kod=70;  -- jó
```

Példa alselectre:

A szerelők fizetése ill. prémiuma legyen 10%-al több, mint az eladók átlagfizetése ill. prémiuma és adjunk egy új beosztásnevet a szerelőknek:

```
UPDATE alkalmazott
    SET   (fizetes, premium) = (SELECT 1.1*AVG ( fizetes ), 1.1*AVG ( premium)
                                FROM alkalmazott
                                  WHERE beosztas = 'ELADO' ),
          beosztas='SZERELŐ'
    WHERE beosztas = 'SZERELO';
```

ORACLE8i -től kezdve skalár értéket adó alselect lehet a kifejezésekben és a korreláció is megengedett:

A szerelők eddigi fizetését növeljük meg az eladók átlagfizetésének 10%-val:

```
UPDATE alkalmazott
    SET   fizetes = fizetes + ( SELECT 0.1*AVG ( fizetes ) FROM alkalmazott
                  WHERE beosztas = 'ELADO' )
    WHERE beosztas = 'SZERELO';
```

Az alkalmazottak eddigi fizetését növeljük meg az azonos beosztásúak átlagfizetésének 10%-val:

```
UPDATE alkalmazott X
    SET   fizetes = fizetes + ( SELECT 0.1*AVG ( fizetes ) FROM alkalmazott
                  WHERE X.beosztas = beosztas );
```

3.4. Adatok törlése

DELETE [FROM] táblanév [aliasnév]
[**WHERE** feltétel];

- WHERE feltétel: a DELETE - ben résztvevő sorokat válogatja ki, enélkül a tábla összes sora részt vesz a műveletben.

Egy sor törlése:

```
DELETE FROM telephely
    WHERE t_kod = 70;
```

Tábla teljes tartalmának törlése:

```
DELETE  FROM  alkalmazott;
```

ORACLE-ben az előzőtől hatékonyabb megoldás is van, mert tablespace helyfelszabadítást is végez:

```
TRUNCATE TABLE  alkalmazott;
```

Tábla részlet törlése:

```
DELETE FROM alkalmazott
    WHERE t_kod IN ( SELECT t_kod
                     FROM telephely WHERE cim = 'DEBRECEN');
```

Jó! Mert DEBRECEN-ben dolgozók egyike sem főnök, nincs rájuk hivatkozás az alkalmazott táblában!

Hivatkozási megszorítási hiba: ha olyan sort törlünk egy táblában, amelynek elsődleges kulcsára hivatkozás van egy másik vagy ugyanazon táblában (lásd még a CREATE TABLE utasítás külső kulcs megszorításának a törlésre vonatkozó ON DELETE .. szabályozását). Vagyis alaphelyzetben szülő táblában olyan sor nem törölhető, melyhez vannak sorok a gyerek táblában!

```
DELETE FROM telephely
WHERE t_kod= '10';
ORA-02292: integrity constraint (ORACLE21.SYS_C0025156) violated - child record found

  DELETE FROM alkalmazott
        WHERE t_kod IN ( SELECT t_kod
                              FROM telephely WHERE cim = 'BUDAPEST');
ORA-02292: integrity constraint (ORACLE35.SYS_C0026822) violated - child record found
```

4. DDL (Adatdefiníciós nyelv)

Az adatbázis és az adatbázis objektumok (pl. szerkezeti elemek)

- létrehozására - CREATE
- törlésére - DROP
- módosítására - ALTER szolgál.

Az alapvető adatbázis objektumokat az adatbázis tervezéskor definiáljuk.

SQL szabványos adatbázis objektum fajták:

- **Tábla - TABLE**: a felhasználói adatok tárolására szolgál, sorokból és oszlopokból áll.
- **Nézettábla -VIEW**: más táblákból, nézettáblákból levezetett virtuális tábla.
- **Index - INDEX**: lekérdezéseket gyorsítja, karbantartásuk időigényes.
- **Szinonima - SYNONYM**: objektumok alternatív neve.

További ORACLE RDB objektumok:

- **Szekvencia - SEQUENCE**: egyedi, elsődleges kulcs értéket generáló objektum.
- **Materializált nézettábla (pillanatfelvétel)**: **MATERIALIZED VIEW | SNAPSHOT:** más táblákból, nézettáblákból származtatott adatok tárolására szolgáló tábla. Frissítése állítható.
- **Felhasználók és jogosultságok kezelése:** USER, PROFILE, SCHEMA, ROLE, stb.

- **Programegységek:** függvények (FUNCTION), eljárások (PROCEDURE), csomagok (PACKAGE), triggerek (TRIGGER), melyek PL/SQL vagy JAVA procedúrális nyelven készíthetők.
- **Adatbázis és a logikai szerkezet kialakítása:** DATABASE, TABLESPACE, DATABASE LINK, stb.
- **Működési egységek paramétereinek kezelése**: SYSTEM (instance), SESSION (felhasználó adatbázishoz való kapcsolódása).

További ORACLE ORDB objektumok ORACLE8i verziótól:

- Felhasználó által definiált típusok és műveletek: TYPE, OPERATOR, stb.

A legtöbb objektumoknak csak a definíciója tárolódik az DD-ben, külön adattárhelyet nem igényelnek. Egyes objektumok (TABLE, INDEX, MATERIALIZED VIEW | SNAPSHOT) adatok tárolására szolgálnak, így létrehozásánál tárolási előírások is megadhatók, melyeket mi nem részletezünk mivel ezek nem részei az SQL szabványnak, továbbá ismertetni kellene hozzá az ORACLE tárolási mechanizmusát.

Az **objektumok azonosítói** max. 30 hosszúak, betűvel kell, hogy kezdődjenek és A-Z, 0-9, _, #, $ karaktereket tartalmazhatnak. Kis- és nagybetű között nem tesz különbséget a rendszer.

Az **objektum tulajdonosa** az a felhasználó, aki létrehozta. A létrehozáshoz megfelelő **jogosultság** (privilégium) kell. Alaphelyzetben minden felhasználó a saját adatbázis-sémáját definiálja és használja, hacsak explicit módon mást nem ad meg. A tulajdonos nevét használjuk az objektum nevének előtagjaként, ha más tulajdonában lévő objektumot akarunk definiálni vagy használni és van rá jogosultságunk. Egy tulajdonos objektumneveinek egyedinek kell lenni és nem lehet objektumnévként használni az SQL utasításokban használt kulcsszavakat.

Más tulajdonában lévő **objektumra való hivatkozás** (ha van hozzá jogunk):

[felhasználó_név.]objektum_név

```
SELECT * FROM kiss.telephely;
```

Osztott adatbázisok esetén másik adatbázisban, más tulajdonában lévő objektumra hivatkozás (ha van hozzá jogunk):

[felhasználó_név.]objektum_név[@db_link]

```
SELECT  *  FROM  nagy.telephely@oktatas.budapest.com;
```

felhasználó_név: a tulajdonos neve ("schema" az ORACLE kézikönyvekben), ha elmarad, akkor a saját tulajdonban lévő (amilyen néven bejelentkeztünk) objektumra hivatkozunk.
db_link: távoli adatbázisra való hivatkozást adja, ha elmarad, akkor a lokális adatbázisban lévő (amelyik adatbázisba először bejelentkeztünk) objektumra hivatkozunk. (A db_link a CREATE DATABASE LINK utasítással definiálandó.)

Szinonimával az objektumoknak, objektumhivatkozásoknak adhatunk alternatív nevet:

CREATE [PUBLIC] SYNONYM szinonima_név **FOR** objektum_hivatkozás;

Szinonima megszüntetése:
DROP SYNONYM szinonima_név;

```
CREATE  SYNONYM  teleph FOR nagy.telephely@oktatas.budapest.com;
```

Ezután egyszerűbben adhatjuk ki a távoli adatbázis táblájára vonatkozó lekérdezést:

```
SELECT  *  FROM  teleph;
```

Megtehetjük például azt, hogy szinonimát használva írjuk az SQL utasításokat, és szükség esetén csak a szinonima definícióját változtatjuk. Osztott adatbázisok esetén megfelelő szinonimák kialakításával a felhasználók és a fejlesztők úgy érzékelhetik, mintha azok egyetlen adatbázisban lennének (nem kell tudni, hogy melyik objektum hol található).

Az objektumok definíciói, a felhasználókra és a rendszerre vonatkozó adatok az **adatszótárban (DD: Data Dictionary)** vannak tárolva táblák formájában. Ezen táblák az adatbázis létrehozásakor jönnek létre és a rendszer tartja karban. A DD-beli táblákban tárolt információk könnyebb lekérdezésére nézettáblákat is kialakítottak.

Pl. SELECT * FROM DICTIONARY; /* összes elérhető DD tábla, nézettábla és rövid leírása*/
SELECT * FROM USER_TABLES; /* a felhasználó táblái */
SELECT * FROM TAB; /* a felhasználó táblái, kevés oszloppal */
SELECT * FROM USER_OBJECTS; /* a felhasználó objektumai */

```
DESCRIBE USER_OBJECTS
DESCRIBE USER_CONSTRAINTS

SELECT  object_name, object_type  FROM USER_OBJECTS
WHERE object_type='TABLE';

SELECT constraint_name, constraint_type, r_constraint_name
FROM USER_CONSTRAINTS;
```

A közel 100 db. DD tábla és nézettábla elnevezései:

USER_... azon objektumok adatai, amelyeknek a felhasználó a tulajdonosa (pl. USER_TABLES),

ALL_... azon objektumok adatai, amelyekhez a felhasználónak hozzáférési jogosultsága van, más tulajdonában is lehetnek ezek (pl. ALL_TABLES),

DBA_... az adatbázisban lévő összes objektum adatai (SELECT_ANY_TABLE jog kell, pl. DBA_TABLES),

V$… a szerver (instance) paraméterei, teljesítmény adatai, GV$… több szerverre (pl. V$NLS_PARAMETERS a nyelvi beállításokat tartalmazza).

4.1. Alaptábla létrehozása, módosítása, törlése

Alaptábla létrehozása:

CREATE TABLE táblanév
(oszlopnév típus [(méret)] [DEFAULT kifejezés] [oszlopmegszorítások] , …
[táblamegszorítás],…)
[tárolási és egyéb előírások]
[**AS** alselect];

típus [(méret)]: beépített (Built-in) típusok és értéktartományok:

CHAR[(m)] - fix hosszú karaktersor, max. 2000 byte, default 1 byte, blank-padded

VARCHAR2(m) - változó hosszú karaktersor, max. 4000 byte, kötelező a méret, nem blank-padded (szóközzel nem egészíti ki a megadott méretre).

CHAR esetén `'A '='A'` igaz;
VARCHAR2 esetén `'A '='A'` nem igaz, de `'A '>'A'` igaz.

NUMBER[(p [,s])] - egész és valós számok, értéktartomány: 1E-130 .. 1E127 és maximum 38 decimális jegy precizitással (pontossággal) tud tárolni.

NUMBER - lebegőpontos valós számok, 38 decimális jegy precizitással.

NUMBER(p), NUMBER(p,s) - fixpontos egész és valós számok.

p: precizitás adja az összes számjegyek számát, max. 38 lehet.
s: skálatényező [-84, 127], default érték: 0. Ha s>=0, akkor a tizedesponttól jobbra eső jegyek számát adja, azaz ennyi tizedesjegyre kerekíti a tároláshoz megadott adatokat. Ha s<0, akkor a tizedesponttól balra az s-dik jegyre lesz kerekítve a kezelt érték.

DATE - dátum és idő

RAW - bináris adatok, max. 4000 byte
LONG ill. LONG RAW - hosszú karakteres ill. bináris adatok, max. 2Gbyte
CLOB ill. BLOB - (large object) karakteres ill. bináris adatok max. 4Gbyte text, image, video adatokhoz. Stb.

Egyes más adattípusok az SQL szabvány vagy más adatbázis-kezelőkkel való kompatibilitás miatt vannak. ORACLE8i-től már felhasználó által definiált típusok és operátorok is lehetnek.

ORDBMS: Object Relational Batabase Management System: a beépített (Built-in) típusokon és operátorokon túl lehetséges felhasználói (User-Defined) típus- és operátor definiálás (lásd: CREATE TYPE …; CREATE OPERATOR …; utasításokat) amelyekkel új típusokat (pl. objektumot attribútumokkal, metódusokkal) és műveleteket lehet definiálni.

DEFAULT érték: az oszlop default értéke. Ha pl. az INSERT-nél nincs megadva, akkor automatikusan ezt veszi, de a DEFAULT kulcsszóval is hivatkozhatunk rá az INSERT, UPDATE utasításokban.

```
belepes  DATE DEFAULT SYSDATE,
```

Példaként lásd még a mintatáblákat:

```
CREATE TABLE telephely
  (t_kod    VARCHAR2(2) PRIMARY KEY,
   t_nev    VARCHAR2(15),
   cim      VARCHAR2(15));

CREATE TABLE alkalmazott
  (a_kod    VARCHAR2(4) PRIMARY KEY,
   a_nev    VARCHAR2(15),
   beosztas VARCHAR2(15),
   belepes  DATE,
   fizetes  NUMBER(6) NOT NULL,
   premium  NUMBER(6),
   t_kod    VARCHAR2(2) NOT NULL REFERENCES telephely(t_kod),
   fonok    VARCHAR2(4) REFERENCES alkalmazott (a_kod);)
```

Tábla készítése egy másik, már létező tábla alapján: ha nem adunk meg oszlopdefiníciókat, azokat az alselect szolgáltatja.

```
CREATE TABLE telephely40
      AS  SELECT t_nev, cim FROM telephely
        WHERE t_kod = '40';
```

Itt a kiválasztott sorok is átkerülnek az új táblába. Az adatok átkerülését letilthatjuk pl. a

```
WHERE  1=2;
```

szelekciós feltétel használatával.

Tábla létrejöttének ellenőrzése:

```
DESCRIBE telephely40
```

Csak az oszlop név, típus, méret és a NOT NULL megszorítás kerül át az új táblába.
Előírhatnánk az új táblára új oszlopneveket, az alselect oszlopaihoz illeszkedő típust, default értéket, megszorításokat is.

Integritási megszorítások (szabályok) (CONSTRAINT):

- Szabályok, melyek biztosítják az adatbázis konzisztenciáját (használható legyen az adatbázis, logikai ellentmondásokat ne tartalmazzon)
- Betartásukat az RDBMS automatikusan biztosítja. Nem enged meg olyan adatbázis műveletek, amelyek révén az adatok a szabályokat megszegnék.

Oszlop megszorítás: értékeinek meg kell felelni az oszlop adattípusának és az értéktartományba kell esni. Oszlop kitöltési kötelezőség előírható (NOT NULL).

Tábla elsődleges kulcsa (PRIMARY KEY): a tábla sorait egyedileg azonosítja, megkülönbözteti egymástól. Egy vagy több oszlop alkothatja. Megszorítás: Értékének egyedinek kell lenni (duplikáció nem megengedett). Nem lehet NULL értékű, összetett kulcs esetén egyetlen része sem.

Külső kulcs (FOREIGN KEY): táblák közötti logikai kapcsolat megvalósítása: az egyik táblában lévő kapcsoló-oszlop(ok), amelyek hivatkoznak egy másik tábla elsődleges (PRIMARY KEY) kulcsára (vagy egy egyedi UNIQUE kulcsára). Egy vagy több oszlop alkothatja. A hivatkozott tábla általában egy másik tábla, de lehet ugyanaz. Hivatkozási (REFERENCE) vagy külső kulcs megszorítás: adattípusuk azonos, a külső kulcs értéke csak létező kulcs értékre hivatkozhat, vagy NULL értékű.

Egyéb megszorítások az üzleti elvárásoknak megfelelően. Ha a működési szabályt le tudjuk írni DDL utasításokkal, akkor a betartását az RDBMS automatikusan biztosítja, egyébként az alkalmazói szoftver készítésénél kell annak betartásáról gondoskodni.

A megszorítások szintaktikai elhelyezésük szerint lehetnek oszlop vagy tábla szintűek. Az egy oszlopot érintő megszorításokat megadhatjuk közvetlenül az oszlop definíciója után (vesszőt nem kell tenni közéjük), míg a több oszlopot érintő megszorításokat (pl. összetett kulcsok) csak tábla szintű megszorításként írhatjuk elő (vesszőt kell tenni közéjük).

Megszorítások (CONSTRAINT) általános alakja:

[**CONSTRAINT** megszorítás_név] megszorítás [megszorítás_állapot]

```
fizetes  NUMBER(6) CONSTRAINT fiz1_alk CHECK(fizetes > 0) DISABLE
                CONSTRAINT fiz2_alk NOT NULL,
a_kod VARCHAR2(4) CONSTRAINT pk_alk PRIMARY KEY,
```

A megszorítások nevét a felhasználó megadhatja, egyébként a rendszer generál egy azonosító nevet **SYS_Cn** formátumban. Hibajelzésekben jelenik meg ez a név, valamint használhatjuk azokat az ALTER TABLE utasításban is. A megszorítás állapotban a megszorítási szabályok működtetésével kapcsolatos dolgokat lehet definiálni, például be- vagy kikapcsolt (ENABLE, DISABLE) állapot. Alaphelyzetben a megszorítások be vannak kapcsolva.

Oszlop szintű megszorítások szintaktikája:

[**NOT] NULL**	az oszlopérték megadása kötelező-e, alapértelmezés NULL
UNIQUE	egyedi kulcs, az oszlop értékének egyedinek kell lennie, NULL lehet
PRIMARY KEY	az oszlop a tábla elsődleges kulcsa, (=> UNIQUE és NOT NULL)
REFERENCES tábla [(oszlop)] [ON DELETE {CASCADE | SET NULL}]	Referencia megszorítás + Törlési megszorítás. Külső kulcsot (kapcsoló oszlopot) definiálunk, amely csak olyan értéket vehet fel, amely szerepel a hivatkozott tábla megfelelő oszlopában, vagy NULL értékű lehet. Ha az oszlop elmarad a hivatkozott tábla PRIMARY KEY oszlopát veszi. Adattípusuk azonos.

ON DELETE lehetővé teszi a szülőtábla (hivatkozott tábla) sorainak törlését, akkor is van rájuk hivatkozás a gyerek táblában:
CASCADE esetén törli a gyereksorokat is, SET NULL esetén a kapcsolódó gyereksorok külső kulcs értékét NULL -ra állítja.
Egyes SQL implementációkban hasonló módon ON UPDATE megszorítást is előírhatunk. ORACLE ezt nem engedi. Alaphelyzet: a szülőtábla kulcs értéke nem módosítható, ha van rá hivatkozás a gyerek táblában.

CHECK (feltétel) az oszlop az itt megadott feltételnek eleget tevő értékeket vehet fel.

```
fizetes  NUMBER(6) CHECK(fizetes > 0)  NOT NULL,
beosztas VARCHAR2(15) CHECK(beosztas=UPPER(beosztas)),
```

Tábla szintű megszorítások szintaktikája: Több oszlopot érintő megszorításokat (pl. összetett elsődleges, egyedi és külső kulcsot) csak tábla szintű megszorításként lehet definiálni. Ezen megszorítás szintaktikája csak kis mértékben tér el az eddigiektől, ezért csak néhány példát adunk:

```
/* pl. egy CHECK feltételben több oszlop szerepel */
CONSTRAINT c_fiz_prem_alk CHECK (fizetes+NVL(premium,0) < 200000),

/* pl. egy szamla táblának összetett az elsődleges kulcsa, szamla_sorszam évente 1-ről indul */
PRIMARY KEY(ev, szamla_sorszam),

/* pl. egy szamla_tetel tábla összetett külső kulccsal kapcsolódik a szamla táblához, és ha egy számla
törölve lesz a tételek is törlődjenek */
FOREIGN KEY (tev, tszamla_sorszam)
        REFERENCES szamla(ev, szamla_sorszam) ON DELETE CASCADE,
```

Alaptábla definíciójának módosítása:

```
ALTER TABLE táblanév
    ADD …              -- pl. új oszlop, új megszorítások megadása
    MODIFY …           -- meglévő definíciós elemek megváltoztatása,
                          megszorítások állapotának változtatása,
    DROP …             -- definíciós elemek eltávolítása
    …;                 -- egyéb változtatások
```

A megvalósításokban sok szabály erősen korlátozza a tábla definíciók módosítását.
Általános alapszabály: az ALTER TABLE utasítás végrehajtása csak akkor lesz sikeres, ha a tábla módosított definíciójának eleget tesznek a táblában már meglévő adatok és a hivatkozási megszorítások sem sérülnek (pl. oszlopok eltávolítása).
Bizonyos esetekben az ORACLE megengedi az ettől való eltérések szabályozását: például egy megszorítás állapotának ENABLE NOVALIDATE módon való bekapcsolásával a táblában meglévő adatok nem lesznek ellenőrizve ezen megszorítással, csak azon adatok, amelyek ezután lesznek DML utasításokkal megadva.
Néhány példa:

Tábla bővítése új oszloppal: a táblában már meglévő sorok az új oszlopban NULL értéket kapnak, ebből következik, hogy nem adható meg NOT NULL specifikáció, ha léteznek sorok:

```
ALTER TABLE telephely
    ADD (telefon VARCHAR2 (10));
```

Oszlop adattípusa, mérete csak "odaillő" másikra változtatható:

```
ALTER TABLE telephely
```

MODIFY (cim CHAR(20) DEFAULT 'BUDAPEST' NOT NULL);

Nem törölhetünk olyan oszlopot, amelyre más megszorítások hivatkoznak, de megadható (CASCADE CONSTRAINTS opcióval), hogy az oszlopra hivatkozó megszorítások is törlődjenek automatikusan.

```
ALTER TABLE telephely DROP (t_kod); -- hibajelzés
ALTER TABLE telephely DROP (t_kod) CASCADE CONSTRAINTS; -- jó
```

Kerülő úton megoldható minden táblamódosítás: egy új táblát kell definiálni, fel kell tölteni beágyazott alselecttel, a régi táblát törölni kell, majd az új táblát át kell nevezni a régi nevére a következő utasítással:

RENAME régi_objektumnév **TO** új_objektumnév;

Alaptábla törlése (adatokkal, indexekkel együtt):

DROP TABLE táblanév [CASCADE CONSTRAINTS];

CASCADE CONSTRAINTS opcióval adható meg, hogy a táblára hivatkozó megszorítások is törlődjenek automatikusan.

```
DROP TABLE telephely;  -- hibajelzés
DROP TABLE telephely CASCADE CONSTRAINTS; -- jó
```

4.2. Nézettábla létrehozása, törlése

A nézettábla más táblákból vagy nézettáblákból levezetett (szelekciós utasítással származtatott) virtuális tábla. Olyan mint egy ablak, melyen keresztül az alapjául szolgáló adatok egy részére lehet rálátni. A nézettáblában adatok nem tárolódnak, csak a definíciója kerül be az adatszótárba. A nézettáblát ugyanúgy használhatjuk SQL parancsokban mint az alaptáblákat.

Létrehozása, vagy a meglévő definíció felülírása:

CREATE [OR REPLACE] VIEW nézettábla_név [(oszlopnév,...)]
AS szelekciós_utasítás
[WITH CHECK OPTION] [CONSTRAINT megszorítás_név]
[WITH READ ONLY];

Törlése:

DROP VIEW nézettábla_név;

Használatának előnyei:

- Az adatvédelem, adathozzáférés szabályozásának egyik módja lehet azáltal, hogy meghatározott felhasználók csak a nézettáblán keresztül férhetnek hozzá az alaptáblákban lévő adatok egy részhalmazához, az alaptáblákhoz nem kapnak hozzáférési jogot.
- A nézettáblát használhatjuk az adatbázis komplexitásának elrejtésére: egy vagy több táblára, nézettáblára támaszkodó bonyolult lekérdezés eredményének könnyebb kezelésére. Az összetett adatlekérdezés gyakori használata helyett elegendő a nézettáblát használni.

Amikor nézettáblán keresztül kérdezzük le az adatokat, a rendszer mindig az alaptáblákból veszi az adatokat. Ha az alaptáblák tartalma módosul, akkor módosul a nézettábla is. Egyszerű

nézettáblán (VIEW egy táblára és kifejezéseket, függvényeket, DISTINCT, GROUP BY részt, stb. nem tartalmaz) keresztül az alaptábla is karbantartható az INSERT, UPDATE, DELETE műveletekkel. A WITH CHECK OPTION -el megadhatjuk, hogy csak a nézettáblát definiáló szelekciós utasítás WHERE feltételének is megfelelő adatok kerülhessenek be az alaptáblába a nézettáblán keresztül (amilyen adatokat lekérdezhetünk a nézettáblán keresztül, csak azokat tarthassuk karban). A DML műveleteket le is tilthatjuk a WITH READ ONLY záradékkal.

```
CREATE OR REPLACE VIEW videk
            AS SELECT  *  FROM telephely
                  WHERE cim <> 'BUDAPEST';
            WITH CHECK OPTION;

DESCRIBE videk;
SELECT * FROM videk;

SELECT view_name, text  FROM USER_VIEWS;

INSERT INTO videk  VALUES ('33', 'JAPAN AUTO', 'BUDAPEST'); -- hibajelzés
     ORA-01402: view WITH CHECK OPTION where-clause violation
INSERT INTO videk  VALUES ('33', 'JAPAN AUTO', 'EGER'); -- jó

Karbantartásra nem alkalmas, összetett nézettáblák:
CREATE OR REPLACE VIEW   alk_tel
            AS  SELECT a_nev, beosztas, t_nev, cim  FROM alkalmazott A, telephely T
            WHERE A.t_kod = T.t_kod;

CREATE OR REPLACE VIEW csoport
AS SELECT  t_kod, AVG(fizetes)  atlag_fiz, COUNT(*) letszam
            FROM alkalmazott
            GROUP BY t_kod;

CREATE OR REPLACE VIEW csoport (t_kod, atlag_fiz, letszam)
AS SELECT  t_kod, AVG(fizetes), COUNT(*)
            FROM alkalmazott
            GROUP BY t_kod;
```

Materializált nézettábla, pillanatfelvétel ORACLE8i-től:

Készítése hasonló a nézettáblához, szintén más táblákból, nézettáblákból levezetett tábla, csak ez nem virtuális tábla lesz, hanem egy ténylegesen létrejövő valódi tábla adatokkal, amely csak olvasható. Szokás még a replikátum, másolat elnevezés is. REFRESH záradékban megadható, hogy mikor és hogyan történjen az automatikus frissítésük (meghatározott időpontokban periódikusan, vagy azonnal az alaptábla változásakor, stb.), ha a záradékot elhagyjuk a frissítés elmarad. További használatuk az alaptáblákhoz hasonló.

CREATE {SNAPSHOT | MATERIALIZED VIEW } nézettábla_név [(oszlopnév,...)]
[tárolási és egyéb előírások]
[**REFRESH** … frissítési paraméterek]
AS szelekciós_utasítás;

Használatának előnyei:

- Osztott adatbázisok esetén egy távoli szerver gépen tárolt (master) adatokról másolatot készítünk a lokális szerver gépeken, azzal a céllal, hogy a hálózati forgalom csökkenjen, a rendszer gyorsabb legyen és a hálózati hiba miatti leállást elkerüljük. Például egy országos áruházláncban a termékek árának karbantartása csak egy központi adatbázisban történik, erről kapnak az egyes áruházak automatikusan (pl. éjszakánként) másolatot, amelyet gyakran olvasnak.
- Döntéstámogató, adattárházas technológiát használó rendszerekben sok, változatlan alapadat (lezárt időszakok) alapján kell bonyolult kigyűjtéseket, összesítő számításokat végezni. Ezen számítások eredményét tárolhatjuk materializált nézettáblában frissítést nem kérve, majd többször felhasználhatjuk a további elemzésekben.

4.3. Indexek létrehozása, törlése

Az indexek (indexobjektumok) az adott táblára vonatkozó lekérdezések felgyorsítását szolgálják, ugyanakkor lassítják a táblával kapcsolatos karbantartó műveleteket. Létrehozásuk és megszüntetésük nincs hatással az SQL utasítások és az alkalmazások kódjára. Egy táblához több index definiálható. Ha már egyszer létre van hozva egy index, az RDBMS automatikusan karbantartja és ha az optimalizálója (szabályalapú vagy költségalapú) úgy dönt, akkor használja is adott esetben.

A tábla PRIMARY KEY és UNIQUE megszorítású oszlopaihoz **automatikusan** létrejön index objektum.

Létrehozás manuálisan:

CREATE [UNIQUE] INDEX indexnév
ON táblanév (oszlopnév [**ASC** | **DESC**] ,...) -- indexkulcsot alkotó oszlopok
[tárolási és egyéb előírások];

```
CREATE INDEX alk_ind1 ON alkalmazott (a_nev);
```

ALTER INDEX … utasítással csak a tárolási jellemzők változtathatók.

Törlése:
DROP INDEX indexnév;

UNIQUE: az indexkulcs csak egyedi értékeket vehet fel, duplikáció nem megengedett.
ASC, DESC: az oszlopbeli értékek növekvő v. csökkenő sorrendjében lehet az indexképzés.

Az indexkulcsok az indexobjektumban rendezetten helyezkednek el, így egy adott kulcsérték vagy egy mintára illeszkedő (LIKE operátor) kulcsok keresése gyorsabb lesz. Az index alkalmazása esetén a rendszer az indexobjektumban keresi a kívánt kulcsértéket vagy értékeket, majd az azok mellé írt mutató (fizikai cím) alapján megtalálja az adattábla sorát illetve sorait.

Az ORACLE az indexobjektum készítésnél B* fa (Balance - kiegyensúlyozott fa) struktúrát használ. Az index lapokba (ágakba) szervezett rendezett oszlopértékekből (indexkulcs) és mutatókból áll.
ORACLE-ben minden tábla ROWID nevű pszeudó oszlopa a sor fizikai címét tartalmazza, mely mutatóként fel van használva az index készítésénél és le is kérdezhető:

```
SELECT a_nev, ROWID FROM alkalmazott;
```
DD-ben: USER_INDEXES, USER_IND_COLUMNS

Néhány irányelv: azon oszlopok legyenek az index kulcsai, amelyek értékei ismeretében gyakran akarunk a tábla soraihoz hozzáférni. (Lásd még a rendszer optimalizációs módszereit.)

- Az oszlopot gyakran használjuk a WHERE feltételben, vagy kapcsoló feltételben.
- Az oszlop nagy számú NULL értéket tartalmaz.
- Az oszlop értéke szerint gyakran rendezzük a lekérdezés eredményét.
- A tábla nagy és a legtöbb lekérdezés kevés (pl. 2-4%) sort ad vissza.
- Stb.

Az ORACLE-beli az INDEX definíció az itt ismertetettől több lehetőséget is megenged és az adatelérés teljesítményének növelésére van még más lehetőség is: pl. **CLUSTER.** A klaszter több tábla fizikailag együtt tárolt csoportja. A klaszter használatával elérhetjük, hogy egymáshoz fizikailag közel tárolódjanak azok az adatok, amelyekre gyakran hivatkozunk együtt, továbbá azt is, hogy az egyes adatcsoportok közös elemeit csak egyszer tároljuk. A CREATE CLUSTER, DROP CLUSTER utasításokkal tudunk klasztereket definiálni és törölni. A **HASH** záradék segítségével hash klaszter is definiálható, amely stabil méretű táblák esetén célszerű, ahol a kulcs érték könnyen leképezhető tárolási címmé.

4.4. Szekvenciák létrehozása, törlése ORACLE-ben

A szekvencia egyedi értékekből álló sorozatot készítő objektum. Jól használható például egy oszlop (UNIQUE, PRIMARY KEY) egyedi értékeinek automatikus előállítására. Leegyszerűsíthető vele az új egyedi érték megadása többfelhasználós rendszerben is. Egyes legenerált sorszámok felhasználatlanul kimaradhatnak a következő esetekben: tranzakció visszagörgetésénél, rendszer összeomlásakor.

A szekvencia definiálását csak egy példán keresztül mutatjuk meg:

```
CREATE SEQUENCE alk_seq
    START WITH 2000 INCREMENT BY 1
    MAXVALUE 9999 NOCYCLE;
```

Törlése:

```
DROP SEQUENCE alk_seq;
```

Módosítása:

```
ALTER SEQUENCE alk_seq …;
```

Használata: NEXTVAL, CURRVAL pszeudó oszlopokkal.
A szekvencia soronkövetkező egyedi értékének előállítása:

```
alk_seq.NEXTVAL
```

A szekvencia aktuális értékének lekérdezése:

```
alk_seq.CURRVAL
```

Pl.
```
INSERT INTO alkalmazott
        VALUES(alk_seq.NEXTVAL, 'PEPITA', 'IGAZGATO',
                NULL, NULL, NULL, '10', NULL);

SELECT alk_seq.CURRVAL FROM DUAL;
```

DD-ben: USER_SEQUENCES tábla;

5. DCL (Adatvezérlő nyelv)

5.1. Tranzakciókezelés

Tranzakció: az adatfeldolgozás logikai egysége. Egy feladat szempontjából **összetartozó DML utasítások** sorozata, amelyek mindegyikének sikeresen kell végrehajtódni ahhoz, hogy ellentmondásmentes (konzisztens) legyen az adatbázis. A tranzakció végén kiadható parancsok:

COMMIT;
Véglegesítés: rögzíti a tranzakció folyamán végrehajtott adatmódosításokat az adatbázisban.

ROLLBACK;
Visszagörgetés: visszaállítja a tranzakció megkezdésekor (a legutolsó COMMIT-nál) rögzített állapotot az adatbázisban, így a tranzakcióhoz tartozó utasítások egyikének sem érvényesül a hatása.

Mindkettő végrehajtható explicit és implicit módon.

Implicit tranzakciókezelés: DDL (pl. CREATE), DCL (pl. GRANT) utasítások után automatikusan végrehajtódik a COMMIT. Az eszközökből való normális kilépés után automatikusan, beállítástól függően, COMMIT vagy ROLLBACK történik. Abnormális programbefejezés esetén automatikus ROLLBACK lesz kiadva a legközelebbi indításkor.

SAVEPOINT mentési_pont;
Lehetővé teszi, hogy a tranzakció közben olyan pontot képezzünk, amely pontig részlegesen visszagörgethetjük a tranzakciót. Ekkor a

ROLLBACK TO mentési_pont;
paranccsal a megadott pontig lehet a visszagörgetést elvégezni.

1. COMMIT;
2. INSERT …;
3. SAVEPOINT A;
4. UPDATE …;
5. SAVEPOINT B;
6. INSERT …;
7. DELETE …;
8. ROLLBACK; -- 1-ig 8. ROLLBACK TO A; -- 3-ig 8. ROLLBACK TO B; -- 5-ig

Pl.
1. Listázzuk ki az alkalmazott tábla tartalmát,
2. Adjuk ki a COMMIT parancsot, ez a tranzakció kezdete
3. Töröljük a szerelő beosztású dolgozókat,
4. Adjunk mindenkinek fizetésemelést,
5. Listázzuk ki az alkalmazott tábla tartalmát (a változások látszanak),
6. Adjuk ki a ROLLBACK parancsot, ez a tranzakció vége
7. Listázzuk ki az alkalmazott tábla tartalmát (egyezik 1-el).

Többfelhasználós rendszert is figyelembe véve a következőket lehet mondani:

Adatok állapota a **COMMIT és ROLLBACK előtt**:

- Az adatok előző állapota visszaállítható, mivel a módosítások csak az "adatbázis-pufferben" (változás naplóban) léteznek.
- Az aktuális felhasználó SELECT-el láthatja a végrehajtott módosításokat.
- A többi felhasználó nem látja a módosításokat, csak a tranzakció előtti állapotot.
- A módosítás alatt álló sorok automatikusan lefoglalódnak (zárolódnak), azaz a többi felhasználó nem módosíthatja azokat, csak lekérdezheti.

Adatok állapota a **COMMIT után**:

- Az adatmódosítások a pufferből véglegesen adatbázisba kerülnek. Az adatok előző állapota nem állítható vissza.
- Minden felhasználó látja a módosítás eredményét.
- A módosított sorokra alkalmazott lefoglalások feloldódnak.

Adatok állapota a **ROLLBACK után**:

- Az adatmódosítások semmissé válnak. Visszaállítódik az adatok előző állapota.
- Minden felhasználó a tranzakció előtti állapotot látja.
- A módosított sorokra alkalmazott lefoglalások feloldódnak.

Megjegyzések:

- **Implicit utasításszintű visszagörgetés:** Ha egy tranzakció DML utasításának végrehajtása során hiba van, akkor a rendszer csak a hibás utasítást görgeti vissza, a többi DML módosítása megmarad. Ehhez a rendszer a DML utasítás előtt implicit mentési pontot használ. A felhasználónak ekkor is explicit módon (COMMIT vagy ROLLBACK) kell befejeznie a tranzakciót.
- **Kétfázisú COMMIT:** Osztott adatbázisok esetén osztott tranzakciók lehetnek: egy tranzakció több (helyi és távoli) adatbázisban lévő táblát használ. A rendszerek úgynevezett kétfázisú COMMIT mechanizmusa garantálja azt, hogy az osztott módosítást végző tranzakciók biztonságosan végrehajtódjanak: csak minden szerveren sikeresen lezajlott érvényesítés után, egy második fázisban történik meg a végleges COMMIT, vagy az egész tranzakció vissza lesz görgetve minden szerveren. Így biztosítva van az adatok konzisztenciája osztott adatbázisok esetén is.

Konkurens adathozzáférés, lefoglalások vezérlése:

Minden komoly adatbázis-kezelőnek biztosítani kell: Sok felhasználó egyidejű (konkurens) hozzáférését az adatbázishoz. Megtartani az adatbázis konzisztenciáját. Csak olyan konkurens munka engedélyezett, amely az adatbázis konzisztenciáját nem veszélyezteti.

A rendszernek automatikusan biztosítani kell, hogy az adatbázis konzisztencia fennmaradjon akkor is, ha több felhasználó egyidőben ugyanazzal az objektummal, vagy annak ugyanazon adatrészével szeretne dolgozni. (Például valaki módosít egy sort, és a tranzakció befejezése előtt egy másik felhasználó törölni vagy módosítani akarja azt.)

<u>Lefoglalások (lock):</u>

A mai rendszerek fejlett sor szintű lefoglalási mechanizmust alkalmaznak. Az alapértelmezés szerinti implicit lefoglalások a következőket jelentik:

Karbantartó DML utasítások esetén olyan a lefoglalás, amely biztosítja, hogy ugyanazt a sort egyszerre nem módosíthatja két felhasználó. Az érintett sorok lefoglalódnak, egy másik felhasználó lekérdezheti, de nem módosíthatja, a tábla többi sorát viszont módosíthatja.

Ha az egyik felhasználó tranzakciója lefoglal sorokat, megtörténhet hogy egy másik felhasználó tranzakciója nem folytathatja a munkáját mindaddig, amíg az első tranzakciója be nem fejeződik (várnia kell a lefoglalások feloldásáig). Megtörténhet, hogy az első tranzakció is elakad és épp egy, a másik által lefoglalt sor(ok) miatt. Ezt nevezzük holtpontnak (deadlock).

USER1 tranzakciója	USER2 tranzakciója
UPDATE telephely …	UPDATE alkalmazott …
UPDATE alkalmazott …	UPDATE telephely …
Vár USER2-re	Vár USER1-re

Egy holtpontnak több mint két szereplője is lehet. Mindegyik vár egy olyan eseményre, amelyet egy másik holtpontra jutott résztvevőnek kellene végrehajtani. A rendszer észreveszi automatikusan a holtpontok létrejöttét és intézkedik: például utasításszintű rollback-et hajt végre az egyik résztvevőnél, szükség esetén újabb résztvevőnél, amíg mindenki normálisan nem folytathatja a munkáját.

Speciális helyzetekben kiadhatók explicit módon is tábla és sor szintű lefoglalások. Az explicit lefoglalások (LOCK TABLE …) szintjeit, módjait, fajtáit nem ismertetjük.

SELECT utasítás nem okoz lefoglalást, és nem kell várni az olvasóknak a foglaltság feloldására, továbbá mindig **olvasási konzisztens** adatokat lát az olvasó. **Olvasási konzisztencia:** a rendszer a lekérdezés kezdetekor érvényes konzisztens adatokat használja, akkor is ha több tábla szerepel a SELECT-ben, vagy sokáig tart a lekérdezés és időközben egy másik felhasználó tranzakciója módosította és kommitálta ugyanezen adatokat.

Emellett még előfordulhat, hogy a felhasználó egymást követő két lekérdezésében más-más adatokat lát. Szükség lehet az olvasási konzisztencia biztosítására több egymást követő lekérdezésnél.

SET TRANSACTION READ ONLY;
biztosítja, hogy a SELECT utasítások egy sorozata az adatbázist ugyanabban az állapotban lássák.

Pl.
```
COMMIT;
SET TRANSACTION READ ONLY;
SELECT ...;
SELECT ...;
COMMIT;          /* zárja az olvasási tranzakciót */
```

5.2. Felhasználók kezelése, felhasználói jogosultságok szabályozása

Rendszergazda (**DBA**: Database Administrator): felügyeli az egész rendszer működését, munkáját speciális programok (és SQL scriptek) segítik.
Feladatai: Felhasználók felvétele, jogosultságaik kiosztása, gazdálkodás az erőforrásokkal, rendszerhibák kiküszöbölése, mentések, visszatöltések, a rendszer működésének figyelése,

hangolása, stb. A rendszer installálása után a DBA ezen teendők elvégzéséhez szükséges jogokat megkapja.

DBA veszi fel az új felhasználókat a rendszerbe:

CREATE USER felhasználó_név
IDENTIFIED BY jelszó
[tárolási tartomány, erőforrás korlátozások, profilok];

Módosítás, törlés az **ALTER USER** …; **DROP USER** …; parancsokkal végezhető el.

A felhasználó is használhatja ezt az utasítást de csak a saját jelszavának módosítására:
ALTER USER felhasználó_név
IDENTIFIED BY jelszó;

DBA írja elő, hogy a felhasználó mit csinálhat a rendszerben, azaz milyen jogai vannak. ORACLE-ben minden felhasználóhoz kapcsolódik automatikusan egy séma (SCHEMA), amely a felhasználó által definiált (CREATE …) adatbázis objektumokat foglalja magába. A séma neve azonos a felhasználói névvel.

Elemi jogok szintjei:

- **Rendszer szintű jogok (privilégiumok):** a rendszer használatával kapcsolatos jogok, azaz milyen tevékenységeket (utasításokat) hajthat végre. DBA adhatja ezeket.
  Néhány példa:
  CREATE USER, CREATE SESSION, CREATE [ANY] TABLE, CREATE [ANY] VIEW,
  CREATE [ANY] SEQUENCE, ALTER [ANY] TABLE, DROP [ANY] TABLE, stb.

- **Objektum szintű jogok (hozzáférési jogok):** az objektumok használatával kapcsolatos jogok, azaz konkrét objektumokkal mit csinálhat. Minden felhasználó a saját objektumait korlátozás nélkül használhatja. Más felhasználók objektumaihoz, csak konkrét jogok birtokában férhet hozzá. Hozzáférési jogokat az objektum létrehozója (tulajdonosa) vagy a DBA adhat másoknak. A konkrétan megadható objektum jogok az objektum típusától függenek.
  Például tábla esetén:
  SELECT, INSERT, DELETE, UPDATE, REFERENCES, stb.

Szerepkör (ROLE): Objektum, melybe jogokat lehet elhelyezni. Olyan mint egy jogosítvány: jogok egy halmaza névvel ellátva, melyet kiadhatunk a felhasználóknak.
A DBA hozza létre ezeket a szerepköröket, majd jogokat rendel hozzájuk.

CREATE ROLE szerepkör_név …;
Pl. CREATE ROLE konyvelo;

Vannak célszerűen kialakított, **előredefiniált szerepkörök.** Példaként három ilyen szerepkört adunk meg, amellyel az adatbázist használók három alapvető típusának (egyszerű felhasználó, fejlesztő, adatbázis adminisztrátor) jogait, tevékenységi körét előírhatjuk.

CONNECT - az egyszerű felhasználók számára kialakítva. Lekérdezheti a táblákat és használhatja az adatkarbantartó utasításokat minden olyan táblára, amelyhez a tábla tulajdonosa jogot adott. Nem definiálhat, nem törölhet objektumokat csak nézettáblát.

RESOURCE - a fejlesztők számára kialakítva. Rendelkezik a CONNECT jogokkal, ezenkívül definiálhat, törölhet objektumokat, valamint az általa létrehozott objektumokra vonatkozóan jogokat adhat tovább más felhasználóknak.

DBA - az adatbázis adminisztrátor (DBA) számára kialakítva. Rendelkezik a RESOURCE jogokkal, ezenkívül bármely felhasználó adataiba betekinthet, felvehet új felhasználókat, jogokat adhat és visszavonhat, PUBLIC-nak minősíthet adatokat, teljes adatbázis export/import, stb.

Rendszerjogok adása:

GRANT {rendszer_jog | szerepkör | **ALL [PRIVILEGES]**} , ...
TO {felhasználó_név | szerepkör | **PUBLIC**}, …
[IDENTIFIED BY jelszó]
[WITH ADMIN OPTION];

WITH ADMIN OPTION esetén a felhasználó a kapott jogokat kiadhatja (visszavonhatja) más felhasználóknak vagy szerepköröknek, PUBLIC az összes felhasználót, ALL PRIVILEGES az összes rendszerjogot jelenti. IDENTIFIED BY záradék csak egy felhasználónál alkalmazható és ha a felhasználó még nem létezne, akkor létrehozza.

```
GRANT CREATE TABLE, CREATE VIEW TO konyvelo;
GRANT  konyvelo TO kiss;
GRANT CONNECT, RESOURCE TO nagy;
```

Rendszerjogok visszavonása:

REVOKE {rendszer_jog | szerepkör | **ALL [PRIVILEGES]**},…
FROM {felhasználó_név | szerepkör | **PUBLIC**}, …;

Ha egy felhasználónak minden privilégiumát megszüntetjük az objektumai még megmaradnak az adatbázisban előtte célszerű ezeket DROP-al törölni.

Objektumjogok adása:

GRANT {objektum_jog [(oszlop, …)] | **ALL [PRIVILEGES]**},…
ON objektum
TO {felhasználó_név | szerepkör | **PUBLIC**}
[**WITH GRANT OPTION**];

WITH GRANT OPTION esetén a felhasználó a kapott jogokat továbbadhatja más felhasználóknak, PUBLIC az összes felhasználót, ALL PRIVILEGES az adott objektumhoz tartozó összes objektumjogot jelenti. Az objektum létrehozója automatikusan minden jogot megkap az objektumához továbbadható módon.

Táblánál az adathozzáférés oszlopok szintjéig finomítható!

```
GRANT UPDATE(t_nev, cim), SELECT
  ON telephely  TO kiss;
```

Objektumjogok visszavonása:

REVOKE {objektum_jog [(oszlop, …)] | **ALL [PRIVILEGES]**},…
ON objektum
FROM {felhasználó_név | szerepkör | **PUBLIC**},…;

A másoknak, a WITH GRANT OPTION lehetőséggel továbbadott jogosultságok is visszavonódnak.