

**KENDE - KOTSIS - NAGY**

**Adatbázis-kezelés  
az ORACLE-rendszerben**

**GYAKORLAT**

**PANEM**

E felsőoktatási tankönyv a Művelődési és Köznevelési Minisztérium támogatásával, a Felsőoktatási Pályázatok irodája által lebonyolított felsőoktatási tankönyvtámogatási program keretében jelent meg.

Copyright © Kende Mária (III.Rész, IV.Rész)

Copyright © Kotsis Domokos (I.Rész)

Copyright © Nagy István (II.Rész, III.Rész)

A kiadásért felel a Panem Könyvkiadó Kft. ügyvezetője, 2002.

Panem Kft.  
1385 Budapest, Pf 809  
Hungary

ISBN 963 545 xxx x

Lektor: Juhász István  
Műszaki szerkesztő: Beck Zsuzsa

A Panem könyvek megrendelhetők az (1) 340-1515 hívószámú telefonon, illetve a 1385 Budapest, Pf. 809 levélcímen.

[panem@panem.hu](mailto:panem@panem.hu)

[www.panem.hu](http://www.panem.hu)

A könyv a Kiadó és a Szerzők gondos munkájának eredménye. Ennek ellenére előfordulhatnak benne hibák, melyeknek következményeiért sem a Szerzők, sem a Kiadó nem vállalnak felelősséget.

Minden jog fenntartva. Jelen könyvet, illetve annak részeit tilos reprodukálni, adatrögzítő rendszerben tárolni, bármilyen formában vagy eszközzel – elektronikus úton vagy más módon – közölni a kiadók engedélye nélkül.

# Tartalom vázlat

## I. Az információ és feldolgozása (Kotsis Domokos)

1. Az információ-feldolgozás kezdetei
2. A folyamat-szemléletű információ-feldolgozás
3. Az adatbázis-szemléletű információ-feldolgozás
4. Az információ-feldolgozás jelene és jövője

## II. A relációs adatmodell (Nagy István)

5. Relációalgebra
6. A relációs adatmodell szerkezete
7. A relációs adatbázis tervezése

## III. Adatbázis-kezelés az ORACLE környezetben (Kende Mária, Nagy István)

8. Az SQL\*Plus környezet és az SQL nyelv alapjai
9. Adatbázis-kezelés SQL nyelven
10. A PL/SQL nyelv

## IV. Adatbázis-kezelés Delphiben (Kende Mária)

11. A Delphi, mint fejlesztési és beágyazó környezet
12. Lokális adatbázisok használata
13. Oracle adatbázisok használata

Függelék, Tárgymutató, CD melléklet

# Tartalomjegyzék

Adatbázis-kezelés az ORACLE környezetben .....	10
Az SQL*Plus környezet és az SQL nyelv alapjai .....	12
Belépés az SQL*Plus környezetbe .....	12
Az SQL nyelv alapjai .....	14
Bevezetés az SQL használatába .....	14
Az Oracle alaptáblái .....	15
EMP tábla .....	18
DEPT tábla .....	18
SALGRADE tábla .....	19
A felhasználói adattáblák lekérdezése .....	19
Mezők adattípusai .....	20
Táblák elérése más felhasználó számára .....	21
Az SQL utasítások csoportjai (az SQL résznyelvei) .....	22
SELECT utasítás használatának alapjai .....	23
Az összes oszlop kiválasztása .....	23
Bizonyos oszlopok megjelenítése .....	24
Az oszlopfejlécek alapértelmezett megjelenítése .....	24
Másodlagos oszlopnevek .....	25
Aritmetikai operátorok .....	25
NULL értékek az aritmetikai kifejezésekben .....	27
Összefűzés operátor .....	29
Literálok .....	29
DISTINCT paraméter .....	30
Bevezetés a tábla kezelésébe .....	31
Tábla létrehozás .....	31
Tábla feltöltése .....	32
Tábla törlése .....	34
Az SQL*Plus környezet .....	35
SQL*Plus szerkesztési utasításai .....	35
SQL*Plus állomány-kezelési utasítások .....	39
Magyarázó szöveg használata a script programokban .....	39
Script programok futtatása .....	40
SQL*Plus környezet használata .....	41
Az adatlekérdező nyelv (a SELECT utasítás általános bemutatása) .....	43
WHERE utasításrész .....	43
Karakterláncok a WHERE feltételben .....	43
Összehasonlító operátorok .....	44
BETWEEN ... AND operátor .....	45
IN operátor .....	46
LIKE operátor .....	46
IS NULL operátor .....	47
LOGIKAI operátorok .....	47

AND operátor .....	48
OR operátor .....	48
NOT operátor.....	49
ORDER BY utasításrész.....	49
Függvényhasználat a SELECT utasításban .....	53
Egysoros függvények .....	53
Karakterkezelő függvények.....	53
Numerikus függvények.....	56
Dátumok használata függvények nélkül .....	57
Dátumkezelő függvények.....	58
Konverziós függvények.....	59
NVL függvény.....	62
Csoportfüggvények.....	62
További műveletvégzés NULL értékkel.....	64
A GROUP BY és a HAVING utasításrész .....	65
HAVING utasításrész.....	70
Halmazműveletek.....	71
Adatbázis-kezelés SQL nyelven.....	76
Táblák összekapcsolása .....	76
Egyszerű vagy egyen-összekapcsolás.....	77
Másodlagos táblanevek használata .....	80
Nem egyen-összekapcsolás .....	81
Belső összekapcsolás.....	81
Külső összekapcsolás .....	82
Tábla összekapcsolása önmagával.....	83
Allekérdezések .....	85
Egysoros allekérdezés .....	86
Többsoros allekérdezés .....	88
Allekérdezés a FROM utasításrészben .....	90
Korrelált allekérdezés.....	91
Adattáblák interaktív használata az SQL*Plus környezetben .....	93
Helyettesítő változók .....	93
& és && előtag.....	93
Felhasználói változók definiálása .....	96
DEFINE utasítás .....	97
ACCEPT utasítás.....	97
UNDEFINE utasítás .....	98
Az SQL*Plus környezet beállítása .....	99
SET utasítás .....	99
SQL*Plus formázási utasítások .....	100
COLUMN utasítás.....	101
Az adatkezelő nyelv (DML).....	104
A DML utasítások .....	104
INSERT utasítás .....	104
INSERT utasítás használata allekérdezővel.....	106
UPDATE utasítás .....	107

DELETE utasítás .....	109
Adatbázis-tranzakciók .....	111
Adatbázis objektumok definiálása (DDL) .....	115
Adatszótár.....	116
Adatszótár lekérdezése .....	117
Adattípusok a táblákban .....	117
A DDL utasítások .....	117
Tábla létrehozása (CREATE TABLE) .....	117
Tábla létrehozása allekérdezéssel .....	118
Tábla módosítás (ALTER) .....	119
ADD utasításrész (oszlop módosítás) .....	120
MODIFY utasításrész (oszlop módosítás) .....	120
DROP utasításrész (oszlop törlése) .....	121
SET UNUSED (törlésre jelölés) .....	122
DROP TABLE utasítás .....	123
RENAME utasítás .....	124
COMMENT utasítás .....	124
Megszorítások .....	125
NOT NULL megszorítás .....	126
UNIQUE megszorítás .....	127
PRIMARY KEY megszorítás .....	128
FOREIGN KEY megszorítás .....	128
CHECK megszorítás .....	130
Megszorítások lekérdezése .....	133
Megszorítások módosítása .....	134
Megszorítás hozzáadása .....	134
Megszorítás törlése .....	135
Megszorítás tiltása .....	136
Megszorítás engedélyezése .....	136
Tovagyűrűző megszorítások .....	137
Nézettáblák .....	137
Nézet létrehozása .....	138
DML műveletek elvégzése, tiltása nézetten keresztül .....	141
Nézet törlése .....	142
INLINE nézet .....	142
FELSŐ-N analízis .....	143
INDEX .....	144
Index létrehozása .....	145
Indexek törlése .....	146
Szinonima .....	147
Az adatvezérlő nyelv (DCL) .....	149
Jogosultságok, privilégium .....	149
Rendszer jogosultságok .....	149
Jelszó megváltoztatása .....	150
Felhasználói rendszerjogosultságok .....	150
Szerepkör .....	151

Objektumkezelési jogosultságok .....	152
Jogosultságok ellenőrzése.....	154
Objektumkezelési jogok visszavonása (REVOKE).....	155
PL/SQL nyelv.....	156
PL/SQL blokk szerkezete.....	158
Alapfogalmak .....	158
Deklarációs szegmens .....	158
Végrehajtható szegmens .....	158
Kivételkezelő szegmens .....	158
A blokkok szerkezete .....	158
Blokk típusok .....	159
Program típusok .....	160
Deklarációs szegmens .....	161
Egyszerű adattípusok.....	162
SQL*Plus változók használata PL/SQL blokkban .....	162
Változók deklarálása .....	162
Alapvető skaláris adattípusok.....	163
Skaláris változók deklarálása.....	164
Logikai változó deklarálása.....	164
%TYPE attribútum.....	165
%ROWTYPE .....	165
Értékadás változónak.....	166
Hozzárendelt változó és használata .....	167
Hivatkozás a hozzárendelt változóra PL/SQL blokkban.....	167
Hozzárendelt változó kiírása .....	167
PL/SQL változó kiírása a blokkon belüli utasítással.....	170
Végrehajtható szegmens.....	172
A végrehajtható szegmens szintaxisa .....	172
Azonosítók, literálok, megjegyzések.....	172
Belső függvények a PL/SQL-ben.....	173
SQL függvények.....	173
PL/SQL függvények.....	173
Operátorok.....	173
DQL utasítások a PL/SQL-ben.....	174
SELECT utasítás a PL/SQL-ben .....	175
DML utasítások a PL/SQL-ben .....	176
Beszúrás .....	176
Módosítás .....	177
Törlés.....	178
Vezérlési szerkezetek .....	180
IF utasítás .....	180
NULL érték a logikai feltételekben: .....	183
Ciklusutasítások.....	184
Egyszerű ciklus (LOOP).....	184
FOR ciklus.....	185
WHILE ciklus.....	188

Egymásba ágyazott vezérlési szerkezetek .....	189
Összetett adattípusok .....	191
PL/SQL rekordok .....	191
RECORD típusú változó használata .....	191
rekordértékkadás táblából .....	193
PL/SQL táblák .....	195
Hivatkozás gyűjtőtáblákra .....	197
Gyűjtőtábla-metódusok .....	197
Rekordtípusú gyűjtőtábla .....	198
SQL kurzor .....	200
Kurzorfüggvények .....	201
%FOUND, %NOTFOUND .....	201
%ROWCOUNT .....	201
%ISOPEN .....	202
Explicit kurzorok .....	203
A kurzor deklarálása .....	203
A kurzor megnyitása .....	204
Fetch utasítás .....	205
A kurzor bezárása .....	206
Kurzorfüggvények használata explicit kurzorok esetén .....	207
Kurzorok és rekordok .....	207
Kurzort használó FOR ciklusok .....	211
A kurzort használó FOR ciklus allekérdezéssel .....	212
Paraméterezett kurzorok .....	213
ROWID pszeudooszlop .....	214
FOR UPDATE utasításrész .....	216
CURRENT OF utasításrész .....	216
Allekérdezést tartalmazó kurzorok .....	219
Példák a kurzor használatára .....	221
1.feladat .....	222
2.feladat .....	223
3.feladat (Párosítási feladat) .....	225
Kivételkezelés szegmens .....	233
Kivételek fajtái .....	233
A kivétel működése .....	234
Definiált kivételek .....	234
Példák .....	236
Kivételek továbbadása .....	241
Kivétel továbbadása beágyazott blokkban .....	241
Alprogramok .....	242
Eljárások .....	242
Függvények .....	245
RETURN utasítás .....	248
Alprogramok deklarációja .....	248
Alprogramok elődeklarációja .....	249
Tárolt alprogramok .....	253

Aktuális és formális paraméterek .....	253
Az IN típusú paraméterek .....	253
Alapértelmezett paraméterhasználat .....	254
Az OUT típusú paraméterek .....	254
Az IN OUT típusú paraméterek .....	255
Rekurzív alprogram .....	256
Delphi, mint fejlesztési és beágyazó-környezet .....	261
Alkalmazásfejlesztés Delphi-ben .....	261
Adatbáziskezelő utasítások beágyazása Delphi környezetbe .....	265
Komponens-paletták .....	266
BDE komponens-paletta .....	266
Data Access komponens-paletta .....	267
Data Controls komponens-paletta .....	267
Adatbázis-elérés Delphiben .....	269
A Delphi BDE felülete, lokális adatbázisok elérése .....	269
BDE - SQL Links, adatbázis-elérés natív driverrel .....	271
Az SQL Links kapcsolat .....	271
A natív driverek .....	272
BDE – ODBC kapcsolat .....	274
ODBC driver beállítása Oracle adatbázishoz .....	275
ADO – OLE DB kapcsolódási felület .....	278
Az Oracle adatbázis külső elérése (ODAC) .....	280
Adatbázis-kezelés a gyakorlatban .....	281
Lokális adatbázisok kezelése .....	281
Példa (Tábla-megjelenítés, állapot-figyelés) .....	281
Példa (Rekordok megjelenítése, navigátor) .....	284
Példa (SELECT utasítás végrehajtása, eredménytábla megjelenítése) .....	285
Példa (Paraméteres lekérdezés.) .....	288
Példa (Tábla létrehozás, rács formázás, számított oszlop) .....	291
Adatfeldolgozás .....	298
Paradox táblalétrehozása és kezelése .....	302
Példák ADO lokális adatbáziskezelésre (Access) .....	308
Oracle adatbázisok használata .....	312
ODBC – Oracle .....	312
Oracle adatbázis-kezelés natív driverrel .....	315
Oracle adatbázis-kezelés ADO komponensekkel .....	319
Az ADO komponensek .....	319
A kapcsolat felépítése .....	319
AdatForrás meghatározása .....	319
Oracle adatbázis-kezelése ODAC komponensekkel .....	331
Az ODAC komponensek .....	331
A kapcsolat felépítése .....	331
AdatForrás meghatározása .....	331
Az adatforrás megjelenítése .....	333
Tárgymutató .....	354

# III. RÉSZ

## Adatbázis-kezelés az ORACLE környezetben

Annak érdekében, hogy az adatbázis-kezelés értékteremtő képességgé váljon, rendkívül fontos, hogy egyrészt az elsajátítandó ismereteket az alkalmazások szempontjából hiteles környezetben, másrészt viszont lehetőleg minél testhezállóbb, iskolai laboratóriumi, vagy otthoni környezetben gyakorljuk. E két igényt nehéz összehangolni.

Jelenleg a világon az egyik legelterjedtebb adatbázis-kezelő rendszer az ORACLE. E rendkívül nagyteljesítményű és hatékony rendszernek az ipari alkalmazók számára a megbízhatósága és a sok platformon való működőképessége vonzó, míg az e rendszerben fejlesztők számára az, hogy a Personal Oracle változat révén saját fejlesztői környezetükben is dolgozhatnak annak biztos tudatában, hogy a létrehozott alkalmazások az ipari környezetben is az elvárt és kipróbált módon fognak működni.

E fejlesztői eszköz egyben lehetőséget ad az adatbázis-kezeléssel ismerkedők számára is, hogy a fent említett két igénynek egyszerre megfelelően tudjanak tanulni, gyakorolni.

Ebben a részben az Olvasó megismerkedik a relációs adatbázis-kezelés ma már szabványosnak tekinthető nyelvi felületével, az SQL nyelvvel. A relációs adatbázis-kezelés algebrai alapjai és fogalmi rendszere az előző (II. részben) részben található. Ezzel összhangban ismertetjük az SQL szabványos relációs adatbázis-kezelő nyelvet, valamint példáinkban megoldjuk az algebrai rész példáit is. Egyúttal megismertetjük a tanulni vágyókat az ORACLE által kifejlesztett SQL\*Plus környezettel, amelynek révén egyrészt egyszerűen kipróbálható a tanult anyag, másrészt kialakíthatók egyszerűbb interaktív

felhasználói felületek. Ugyancsak e rész mutatja be a szintén ORACLE által létrehozott PL/SQL nyelvet, amely nem csupán az SQL kiterjesztésének tekinthető eljárásorientált magasszintű adatbázis-kezelő programozási nyelv, hanem a vele létrehozható tárolt eljárások, triggerek segítségével révén az adatbázis-kezelés egyik leghatékonyabb eszköze is.

E rész elsajátításához az előző részek elméleti ismeretein túl sok gyakorlás szükséges. Ehhez pedig elég, ha kitartással és egy jó számítógéppel rendelkezik a kedves Olvasó.

E gyakorlati részhez tartozó mintaprogramok, amelyeknek kiterjesztése `.sql`, betűvel és számmal ellátott program-állományokban találhatók meg a CD mellékleten (Például: `s057.sql`, vagy `t002.sql`).

## 8. FEJEZET

# Az SQL\*Plus környezet és az SQL nyelv alapjai

## Belépés az SQL\*Plus környezetbe

Elérkeztünk könyvünk azon fejezetéhez, amely bemutatja, hogyan lehet gyakorlatban működő adatbázis-lekérdezéseket, úgynevezett riportokat, jelentéseket elkészíteni az Oracle SQL\*Plus környezetében az SQL nyelv segítségével.

Az SQL (Structured Query Language) nyelv a relációs adatbázis-kezelő rendszerek szabványosított kezelőnyelve 1986 óta. Kifejlesztették a szabványt mind az ANSI, mind az ISO rendszerben. Jelenleg már az SQL99 szabvány is kidolgozás alatt van.

Az anyag tárgyalása során feltételezzük, hogy az olvasó a számítógép előtt ül, már ismeri a relációs adatbázis fogalmát és matematikai alapjait, továbbá a számítógépén telepítve van az Oracle Personal Edition Release valamilyen újabb (8, 8i, esetleg 9i) verziója, vagy az Oracle hálózati verzióján keresztül gépén telepítve van az Oracle Client verziója.

A Personal Oracle rendszer telepítésének menetét az A Függelék, az indítását a B Függelék tartalmazza, amely a CD-mellékleten található.

Az SQL\*Plus elindítása után megjelenő ablakba kell beírunk a megfelelő felhasználónevet, és az ehhez tartozó jelszót. (A Personal Oracle verziókban nem kell megadnunk a "Bejelentkezési jelszó" adatot, míg az Oracle Client verziójánál ezt a rendszergazda adja meg.)

Az indításhoz leggyakrabban használt felhasználói név – jelszó páros `system / manager`, de az alábbi párosokat is használhatjuk:

- `system      manager`
- `scott        tiger`
- `po8          po8`
- `sys          change_on_install`

Ez utóbbi a rendszeradminisztrátor kezdeti belépését teszi lehetővé.

Ezen párosok mindegyikére az SQL\*Plus program elindul. Megjelenik az SQL\*Plus szerkesztési felülete, amint azt a 8.1. ábrán is láthatjuk. Most már írhatunk SQL utasításokat, és azokat a rendszer végére is hajtja.



Az utasítás megértéséhez tudnunk kell, hogy a dátumokat az Oracle rendszer belső számformátumban tárolja évszázad, év, hónap, nap, óra, perc, másodperc alakban. Az alapértelmezett dátum a 'DD-MON-YY' karaktersorozat. Amennyiben ez az alapértelmezett formátum számunkra megfelelő, nincs tennivalónk. Ha másmilyen dátumformátumot szeretnénk kiírni, akkor a belső számformátumot a formátumkódok segítségével a megadott dátumformátumra kell konvertálni. Ezt a konverziót végzi el a `TO_CHAR` konverziós függvény. Értelemszerűen:

Formátumkód	Jelentése
DD	hónap napjainak sorszáma (max. 31)
mon	a hónap három kisbetűs rövidítése
month	a hónap teljes kisbetűs neve
YYYY	az év

A dátumformátumokra és a konverziós függvényekre még visszatérünk.

Az Oracle rendszerben beállítható nyelvek, dátum- és idő formátumok megtalálhatók például az Oracle on-line dokumentációban. (Oracle8i Server and SQL\*Plus/ Oracle8i National Language Support Guide/Language Support/Territory Support stb. valamint az Alter Session címszó alatt. Ez utóbbiról később még szó lesz.)

## Az SQL nyelv alapjai

### Bevezetés az SQL használatába

Célunk, hogy az olvasó minél előbb ismerkedjen meg az SQL nyelvvel, és főképp, hogy valóban használni is tudja, ezért sok példán keresztül mutatjuk be a nyelv használatát. A példák eleinte egyszerűek, majd egyre bonyolultabbak lesznek. Mindig igyekszünk kitérni a nyelv általános szabályaira, teljességre azonban nem törekszünk. A teljes Oracle SQL és SQL\*Plus nyelv leírása megtalálható az Oracle referenciakönyvekben, valamint az Oracle on-line dokumentációkban (lásd az irodalomjegyzéket).

A kezdeti tanulás megkönnyítése érdekében, felhasználjuk azokat az adattáblákat, amelyek minden Oracle-verzióban megtalálhatóak, és mindenki számára hozzáférhetők.

Az utasítások általános leírásánál használt szintaktikai jelek jelentése az alábbi:

- { } a kapcsos zárójel mindig egy kötelezően megadandó elemet jelent,
- [ ] a szögletes zárójel azt jelenti, hogy a benne szereplő szintaktikai elem megadása nem kötelező,
- | a függőleges vonal *vagy* kapcsolatot jelez,
- ... azt jelöli, hogy a ... előtti szintaktikai rész ismételtető.

## Az Oracle alaptáblái

A táblák a `scott` felhasználó sémájában (tulajdonában) vannak. Tehát a legközvetlenebb elérésük az, ha indításkor a felhasználónév a `scott` és a jelszó pedig a hozzá tartozó `tiger`. Ha először a `system/manager` módon léptünk be, akkor könnyedén átkapcsolhatunk a `scott` felhasználóra. Adjuk ki a következő utasítást.

```
SQL> CONNECT scott/tiger;
Kapcsolódva.
SQL>
```

Természetesen ebből vissza is térhetünk a `system` felhasználóhoz, illetve bármilyen más létező felhasználót kapcsolhatunk a rendszerhez.

```
SQL> CONNECT system/manager;
Kapcsolódva.
```

Az Oracle egyik alaptáblája az `emp` tábla (employee = alkalmazott). Nézzük meg, hogy mit, milyen adatokat, mezőket, rekordokat tartalmaz ez a tábla. Listázzuk ki az `emp` tábla tartalmát.

```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7369	SMITH	CLERK	7902	80-DEC-17	800	
7499	ALLEN	SALESMAN	7698	81-FEB-20	1600	300
...						

Mint láthatjuk, kicsit áttekinthetetlen formában jelent meg a lista. Az SQL\*Plus környezetében ennek a lekérdezésnek (riportnak) a kimenet megjelenítéséhez szükséges alapbeállítás sorhossz rövid. Az alapértelmezés szerinti beállítás `LineSize` (sorméret) 80, az `emp` tábla lekérdezéséhez nem elegendő. Az SQL\*Plus kimenete nem karakterekben, hanem pontokban számolja a szélességet.

### Mit tehetünk?

Az SQL\*Plus kimenet megjelenítési formáját számos paraméterrel befolyásolhatjuk. A paraméterek lekérdezhetők a `SHOW ALL` utasítással, valamint a kézikönyvekben is megtalálhatók.

```
SQL> SHOW ALL;
appinfo értéke ON és "SQL*Plus"-ra van beállítva
A(z) btitle OFF, és a következő SELECT utasítás első néhány karaktere
define "&" (hex 26)
...
linesize 80 --sor hosszának mérete
...
newpage 1
```

```

null "" -- NULL érték helyettesítő szövege
numformat ""
numwidth 10 -- szám ábrázolási szélessége
pagesize 14 -- oldalhossz
...
tab ON -- szöközőket használ a riportok és osz-
      lopok szövegének formázására
...
time OFF -- rendszeridő kiírása
A(z) ttitle OFF, és a következő SELECT utasítás első néhány karaktere
underline "--" (hex 2d) -- oszlopfejek aláhúzás karaktere
a USER: "SCOTT"
verify ON -- kiírja a változók régi és új értékét
wrap : a sorok tördelve lesznek

```

Most állítsuk be a sor hosszát az alapértelmezéstől eltérően.

A megváltoztatás, illetve a beállítás kétféleképpen is történhet:

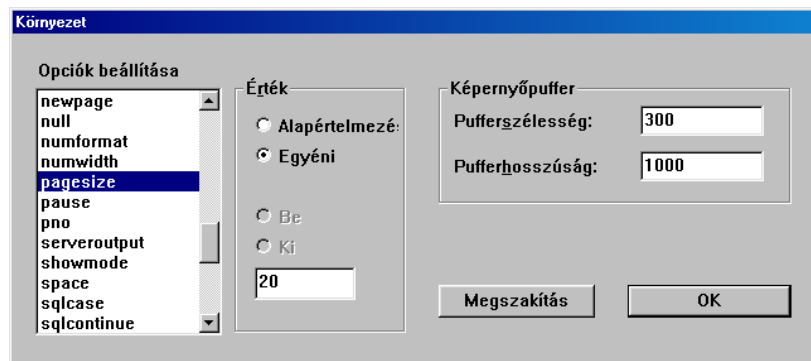
- Az SQL prompt után a `SET` paranccsal  
Állítsuk át a sorhosszt, tehát a kiíratási szélességet 300 – ra ( A maximum 999.)

```
SQL> SET linesize 300;
```

Nézzük meg sikerült-e?

```
SQL> SHOW linesize;
linesize 300
```

- Az SQL\*Plus felhasználói felületén az OPTION menüpont Környezet almenü segítségével, amint azt a 8.2 ábrán láthatjuk.  
Állítsuk be az egy oldalon megjelenő sorok számát az alapértelmezettől egyénire és 40-re.



8.2. ábra. Az SQL\*Plus felületének beállítása az SQL\*Plus Option menüjének segítségével

Kérdezzük le, hogy valóban átállította-e az oldalhosszt?

```
SQL> SHOW pagesize;
pagesize 40
```

Ha azt szeretnénk, hogy az alapbeállítástól eltérő beállításaink, az SQL\*Plus minden indításakor aktivizálódjanak, azaz ne legyen szükség minden alkalommal ezek újbóli beállítására, akkor a parancsokat a `login.sql` script programba (program állományba) szükséges mentenünk.

Ezt a következőképpen tehetjük meg:

```
SQL> SELECT * FROM emp;  
SQL> SAVE login  
Létrehozva: file login
```

Most szerkesszük a `login.sql` állományt az alábbi módon:

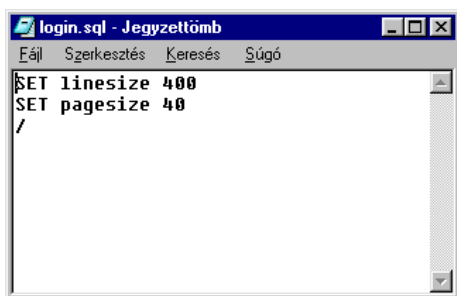
```
SQL> ed login
```

Megjelenik a jegyzettömb, mint alapbeállított szövegszerkesztő. Ezt láthatjuk a 8.3 ábrán.



8.3 ábra. Az alapbeállított szövegszerkesztő a `SELECT` utasítással

Töröljük az első sort (az előző lekérdezést). Írjuk be a környezeti beállításokhoz szükséges két parancsot, és mentjük el az állomány tartalmát. Ezt mutatja az alábbi 8.4. ábra.



8.4 ábra. Az alapbeállított szövegszerkesztő a környezeti beállításokkal

Ezután futtassuk le a script programot.

```
SQL> @ login
```

A beállítások a `login.sql` script program lefutása után életbe lépnek.

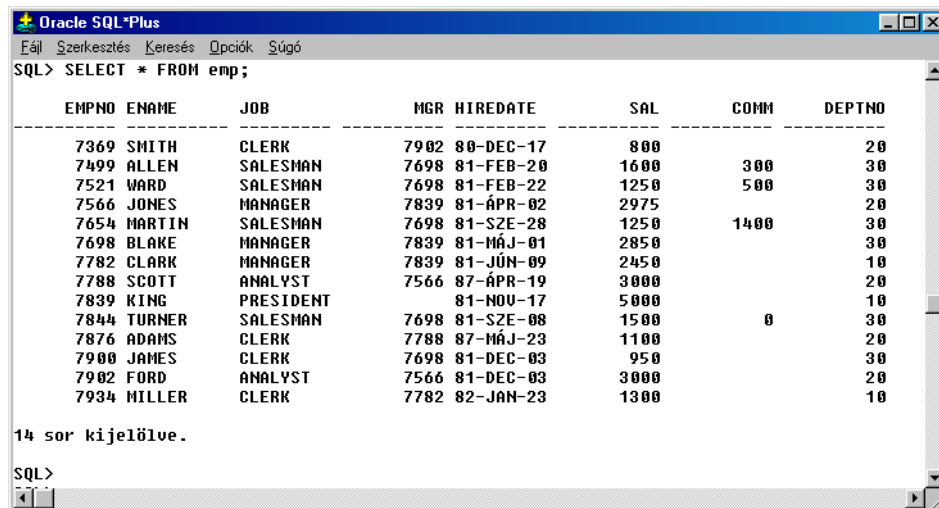
A `login.sql` script programot az SQL\*Plus az Oracle\Ora81\Bin könyvtárba mentette el. Innen az SQL\*Plus minden indításakor a `login` állomány automatikusan lefut.

Természetesen ez a beállítás bizonyos lekérdezéseknél megint nem ad elég áttekinthető kimeneti formátumot. Ilyen esetekben alkalmanként már átállíthatjuk a feladatnak megfelelően rövidebbre vagy hosszabbra az oldalhossz méretét. (Ezekre például a `DESC` tábla-lekérdezéseknél lesz szükségünk).

## EMP TÁBLA

Ez a tábla az alkalmazottak (employee) adatait tartalmazó tábla (lásd a 8.5. ábrát).

Listázzuk ki most az `emp` táblát a leggyakoribb, a `SELECT` lekérdezési utasítás segítségével.



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80-DEC-17	800		20
7499	ALLEN	SALESMAN	7698	81-FEB-20	1600	300	30
7521	WARD	SALESMAN	7698	81-FEB-22	1250	500	30
7566	JONES	MANAGER	7839	81-APR-02	2975		20
7654	MARTIN	SALESMAN	7698	81-SZE-28	1250	1400	30
7698	BLAKE	MANAGER	7839	81-MÁJ-01	2850		30
7782	CLARK	MANAGER	7839	81-JÜN-09	2450		10
7788	SCOTT	ANALYST	7566	87-APR-19	3000		20
7839	KING	PRESIDENT		81-NOV-17	5000		10
7844	TURNER	SALESMAN	7698	81-SZE-08	1500	0	30
7876	ADAMS	CLERK	7788	87-MÁJ-23	1100		20
7900	JAMES	CLERK	7698	81-DEC-03	950		30
7902	FORD	ANALYST	7566	81-DEC-03	3000		20
7934	MILLER	CLERK	7782	82-JAN-23	1300		10

14 sor kijelölve.

SQL>

8.5. ábra. Az `emp` tábla listázása

Az `emp` tábla oszlopainak jelentése:

<code>empno</code>	az alkalmazott azonosító száma,
<code>ename</code>	vezetéknév,
<code>job</code>	beosztás,
<code>mgr</code>	főnökének azonosító száma,
<code>hiredate</code>	belépés dátuma,
<code>sal</code>	fizetés,
<code>comm</code>	jutalék,
<code>deptno</code>	a részleg száma, ahol az alkalmazott dolgozik.

## Fontos

Az SQL utasítások elfogadott terminológiája szerint a táblaneveket kisbetűvel írjuk, míg a kulcsszavakat nagybetűvel.

## DEPT TÁBLA

Ez a tábla a vállalat részlegeire jellemző adatokat tartalmazza. A tábla neve `dept`, amely a `department` = részleg rövidítése.

```
SQL> SELECT * FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

A dept tábla oszlopainak jelentése:

deptno	az részleg száma,
dname	a részleg neve,
loc	a részleg telephelye

## SALGRADE TÁBLA

A tábla a fizetési adatok jellemzőit tartalmazza. Salgrade a salary grade rövidítése, fordítás szerint salary = fizetés, grade = kategória.

```
SQL> SELECT * FROM salgrade;
```

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

A salgrade tábla oszlopainak jelentése

grade	bérkategória,
losal	a kategória alsó szintje,
hisal	a kategória felső szintje.

## A felhasználói adattáblák lekérdezése

Az egyes felhasználók adatai lekérdezhetők a USER\_TABLES adatszótárbeli nézetből.

Az *adatszótár* táblák, nézetek együttese, amelynek tulajdonosa a sys és a system felhasználók, és installáláskor jön létre. Alapvető információkat ad az Oracle-rendszeréről és a felhasználóról.

A *nézet* adatbázis-objektum sok vonatkozásban hasonlít a táblára. Például ugyanúgy lehet lekérdezni, lehet rá hivatkozni lekérdezésekben. Lényeges különbség azonban, hogy nem tartozik hozzá fizikai tábla, hanem csupán egy fizikai táblára vonatkozó lekérdezés. Lényegében e táblára való hivatkozás nem más, mint a hozzá rendelt lekérdezés aktivizálása. Ebből következik például az a tulajdonsága, hogy "tartalma" automatikusan követi az általa hivatkozott fizikai tábla tartalmának változásait. Ebben az értelemben a nézet tekinthető a tábla logikai reprezentációjának is.

Mielőtt az utasítást kiadnánk, állítsuk át a linesize (sorhossz) beállítását 400-ra. A beállításokat minden újraindításkor meg kell tennünk (hacsak azokat el nem mentettük a

login.sql script programba). Ebben az esetben a gördítő sáv használható, és egy-egy sorban egy-egy táblára vonatkozó teljes információt megkapjuk.

```
SQL> SELECT * FROM user_tables;
```

TABLE_NAME	TABLESPACE_NAME					
INSTANCES	CACHE	TABLE_LO	SAMPLE_SIZE	LAST_ANAL	PAR	IOT_TYPE
BONUS	1	N	ENABLED	USERS	NO	
DEPT	1	N	ENABLED	USERS	NO	
EMP	1	N	ENABLED	USERS	NO	
SALGRADE	1	N	ENABLED	USERS	NO	

Ez a lista egy részlete, de a gördítősáv használatával tovább nézhetjük. A teljes eredményt figyeljük meg a saját képernyőnkön.

Ha csak a táblanevekre vagyunk kíváncsiak, akkor a következő utasítást adhatjuk ki:

```
SQL> SELECT * FROM user_catalog;
```

TABLE_NAME	TABLE_TYPE
BONUS	TABLE
DEPT	TABLE
EMP	TABLE
SALGRADE	TABLE

## Mezők adattípusai

Nézzük meg, hogy milyen adattípusok fordulnak elő az alaptáblákban. Ennek lekérdezése a DESC utasítással történik.

A DESC [RIBE] egy SQL\*Plus utasítás, és egy tábla oszlopdefinícióit jeleníti meg.

### Példa

Milyen adattípusok fordulnak elő az emp táblában?

```
SQL> DESC emp;
```

Név	Üres?	Típus
EMPNO	NOT NULL	NUMBER (4)
ENAME		VARCHAR2 (10)
JOB		VARCHAR2 (9)
MGR		NUMBER (4)
HIREDATE		DATE
SAL		NUMBER (7, 2)
COMM		NUMBER (7, 2)
DEPTNO		NUMBER (2)

Az következő adattípusok fordulnak elő:

NUMBER ( <i>n</i> )	egész szám, ahol <i>n</i> az értékes jegyek száma,
NUMBER ( <i>n</i> , <i>m</i> ) ,	valós szám, ahol <i>n</i> az összes karakterek száma, és <i>m</i> a tizedes jegyek száma,
NUMBER	szám, alapértelmezett mezőszélesség 10,
VARCHAR2 ( <i>n</i> ) ,	karakter sorozat, ahol <i>n</i> a maximális karakterek száma,
DATE	dátum és idő tárolására alkalmas típus (i.e. 4712. jan.01. és i.sz. 4712.dec.31 közé kell esnie).

Az `üres?` nevű oszlopban előforduló `NOT NULL` azt jelenti, hogy ebben az oszlopban minden sorban, a rekordoknak ebben a mezőjében mindig kell adatnak lenni. Ha az `üres?` oszlopban nincs semmi, azt jelenti, hogy ez az oszlopérték hiányozhat a rekordból.

## Táblák elérése más felhasználó számára

Ha visszatérünk, illetve átkapcsolunk egy másik felhasználóra, és ez a felhasználó kérdezi le az `emp` (`dept`, `stb.`) táblát, akkor a következő üzenetet kapjuk:

```
SQL> CONNECT system/manager;
Kapcsolódva.
```

A `scott` felhasználóról lekapcsolódik a rendszer, és átkapcsol a `system` felhasználóra.

### Példa

Kérdezze le a `system` felhasználó az `emp` táblát:

```
SQL> SELECT * FROM emp;
select * from emp
          *
Hiba a(z) 1. sorban:
ORA-00942: a tábla vagy a nézet nem létezik
```

Tehát a `system` felhasználó tulajdonában nincs `emp` (`dept`, vagy `salgrade`) nevű tábla.

Ha a `system` felhasználónak joga van látni a `scott` felhasználó tábláit, akkor a következő utasítással lekérdezheti a táblát.

```
SQL> SELECT * FROM scott.dept;

  DEPTNO DNAME                LOC
-----
10 ACCOUNTING                NEW YORK
20 RESEARCH                   DALLAS
30 SALES                      CHICAGO
40 OPERATIONS                 BOSTON
```

A `SELECT` lekérdezésben a táblára való hivatkozás úgy történik, hogy a tulajdonos (séma) neve után pontot teszünk, majd következik a táblanév. Ezt nevezzük minősített névnek. A tulajdonos a `scott`, a táblanév a `dept`. Tehát a táblanév hivatkozás helyes alakja abban az

esetben, ha nem a saját táblánkra hivatkozunk, hanem más táblájára, és jogosultságunk van más tulajdonos tábláit bizonyos kritériumoknak megfelelően használni: *tulajdonos.táblanév*. A felhasználók lekérdezhetők a következő paranccsal:

```
SQL> SELECT username FROM all_users;
USERNAME
-----
SYS
SYSTEM
OUTLN
DEMO
DBSNMP
SCOTT
PO8
...
```

A hozzáférési jogok is lekérdezhetők (lásd az Oracle referencia kézikönyvben).

## Az SQL utasítások csoportjai (az SQL résznyelvei)

Az SQL utasításokat az alkalmazásuk szempontjából a következőképpen csoportosíthatjuk.

- *Adatdefiniáló nyelv (DDL)*  
Azon SQL utasítások halmaza, amelyek az adatbázisban objektumokat definiálnak. Bizonyos feltételek mellett az objektumok (például táblák, nézetek stb.) szerkezetét megváltoztathatjuk, módosíthatjuk, megszüntethetjük. Ide tartozik a létrehozás (`CREATE`), a módosítás (`ALTER`), a megszüntetés (`DROP`), az átnevezés (`RENAME`) és a tartalom-törlés (`TRUNCATE`).
- *Adatlekérdező nyelv (DQL)*  
Az adatlekérdező nyelv (Data Query Language) tulajdonképpen az adatkezelő nyelv egy önálló csoportja, amelybe egyetlen utasítás tartozik, a `SELECT`. Segítségével a legkülönbözőbb szempontoknak, feltételeknek és csoportosításoknak megfelelő riportokat készíthetünk.
- *Adatkezelő nyelv (DML)*  
Az adatkezelő nyelv (Data Manipulation Language) utasításai közé azon SQL utasítások tartoznak, amelyek lehetővé teszik, hogy az adatbázis állapotát megváltoztathassuk, tartalmával, az adatokkal külön - külön dolgozzunk. Ide tartozik az új sorok, adatok bevitele (`INSERT`), adatok módosítása (`UPDATE`), sorok törlése (`DELETE`).  
Az adatkezelő nyelven belül önálló csoportot alkot, a DML utasítások által elvégzett változások kezelése, bonyolítása. Ezeknek az utasításoknak a segítségével logikai tranzakcióba csoportosíthatjuk az adاتمódosításokat. Ide tartozik a `COMMIT` utasítás, amely véglegesíti az adatbázisban az utolsó `COMMIT` óta végrehajtott változásokat. A `ROLLBACK` (visszagörgetés) utasítás az a tevékenység, amelyet akkor kell elvégeznünk, ha egy munkamenet adatbázis változásait (DML utasításokat) nem akarjuk véglegesíteni. A visszagörgetés visszaállít minden változtatást, az adatbázis szerkezetét a felhasználó által kezdeményezett módosítás, beszúrás vagy törlés előtti állapotba. A `SAVEPOINT` (mentési pont) utasítás lehetővé teszi, hogy tranzakció közben mentési pontot hozzunk létre, amely pontig visszagörgethetjük a tranzakciót.

- *Adatvezérlő nyelv (DCL)*

Az SQL utasítások önálló, adatbiztonsági csoportját alkotják (Data Control Language). Ezek az utasítások a felhasználók számára lehetővé teszik más felhasználó adatbázisához, tábláihoz, adataihoz való hozzáférést, felhasználást, változtatást. A GRANT utasítással kapcsolódási, hozzáférési, lekérdezési (SELECT), módosítási jogokat adhatunk felhasználóknak, a REVOKE utasítással pedig visszavonhatjuk e jogokat.

### Fontos

Az SQL utasításokat mindig pontosvesszővel kell lezárni.

## SELECT utasítás használatának alapjai

A SELECT a leggyakrabban használt SQL utasítás, a DQL résznyelv egyetlen utasítása. A lekérdezés szempontjainak megfelelő sorokat ad vissza az *eredménytáblában* a FROM utasításrészben megadott *paramétertábla(k)ból*, a WHERE utasításrészben megadott feltételek szerint. Először néhány egyszerű, feltételek nélküli SELECT utasítás mutatunk be.

Az SQL\*Plus környezetben az SQL utasításokat a prompt jel (SQL>) után írjuk be, a többsoros utasításokat a rendszer sorszámmal látja el.

A környezet ezeket az utasításokat az ún. SQL pufferben tárolja. A pufferben mindig a legutolsó utasítást találhatjuk meg, s azt a RUN utasítással, vagy a ”/” billentyű leütésével ismételten lefuttathatjuk.

Ha az utasításban hiba van, és ki szeretnénk javítani, akkor a helyes sorokat, részeket a Windows-ból jól ismert CTRL + C (másolás) és CTRL + V (beillesztés) billentyűkombinációkkal illeszthetjük a SQL prompt után, ezzel kicsit megkönnyítve munkánkat. Később más egyszerűbb módszerekre is kitérünk.

A SELECT utasítás legegyszerűbb alakja ( utasításrészek nélkül):

```
SELECT { *
      | {oszlop [[AS] másodlagos oszlopnév]
      | kifejezés [[AS] másodlagos oszlopnév]}
      [, {oszlop [[AS] másodlagos oszlopnév]
      | kifejezés [[AS] másodlagos oszlopnév]} ] ... }
FROM táblanév;
```

ahol	*	a táblázat összes oszlopát helyettesíti,
	<i>oszlop, kifejezés</i>	lehet egy oszlop neve, egy literál, egy matematikai formula, egy függvényhívás vagy függvények kombinációja,
	<i>másodlagos oszlopnév</i>	: ideiglenesen átnevezi az oszlopot vagy kifejezést.
	AS	kulcsszó választja el az oszlopnevet, kifejezést a másodlagos oszlopnévtől.

### Az összes oszlop kiválasztása

Az összes táblabeli oszlop az eredeti elhelyezkedési sorrendnek megfelelően megjeleníthető, ha a SELECT után csillagot (\*) írunk.

```
SQL> SELECT *
2     FROM emp;
```

Eredményét a 8.5. ábrán már láttuk.

## Bizonyos oszlopok megjelenítése

Listázzuk ki az emp tábla alkalmazottainak azonosító számát, nevét és belépési idejét.

```
SQL> SELECT empno, ename, hiredate
2     FROM emp;
```

EMPNO	ENAME	HIREDATE
7369	SMITH	80-DEC-17
7499	ALLEN	81-FEB-20
7521	WARD	81-FEB-22
7566	JONES	81-APR-02

...

14 sor kijelölve.

Listázzuk ki az emp tábla alkalmazottainak belépési dátumait és nevét.

```
SQL> SELECT hiredate, ename
2     FROM emp;
```

HIREDATE	ENAME
80-DEC-17	SMITH
81-FEB-20	ALLEN
81-FEB-22	WARD

...

14 sor kijelölve.

A listázandó oszlopok sorrendje tetszőleges.

## AZ OSZLOPFEJLÉCEK ALAPÉRTELMEZETT MEGJELENÍTÉSE

Alapértelmezett igazítás :

- *balra*: karakteres és dátum adattípusok esetén,
- *jobbra*: numerikus adatok esetén.

Nézzük meg:

```
SQL> SELECT empno, ename, hiredate
2     FROM emp;
```

EMPNO	ENAME	HIREDATE
7369	SMITH	80-DEC-17
7499	ALLEN	81-FEB-20
7521	WARD	81-FEB-22
7566	JONES	81-APR-02

...

14 sor kijelölve.

Az empno négyjegyű szám, de alapértelmezett számformátumban (10 jegy) jeleníti meg a fejléct és az oszloptartalmát is, mégpedig jobbra igazítva. Az ename oszlop 10 hosszú karaktersorozat balra igazítva, úgyszintén a dátum karaktersorozat, az alapértelmezett formátum szerint, és balra igazítva. A dátum kiírását jobban láthatjuk, ha magyar fejléccel látjuk el az oszlopnevet, mint azt a következő másodlagos oszlopnevek példáinál megfigyelhetjük. Az alapértelmezett kiírás nagybetűkkel történik. A karakteres és dátum adatokat a fejlécben a rendszer csonkolhatja, a numerikus adatok fejlécét azonban nem.

## Másodlagos oszlopnevek

A másodlagos oszlopnév olyan ideiglenes név, amelyet egy SQL utasításban egy oszlophoz rendelünk hozzá. Segítségével egyrészt átírhatjuk az oszlopok fejlécét, másrészt, hogy hivatkozhatunk rá egyes utasításrészekben. Másodlagos oszlopnevek megadása:

- Az oszlopnév és egy szóköz után írhatjuk a másodlagos nevet.
- Az oszlopnév után az AS kulcsszó után is megadhatjuk az új oszlopnevet.
- Ha a másodlagos név szóközt, vagy speciális jelet tartalmaz, vagy meg szeretnénk különböztetni a kis – és nagybetűket, akkor idézőjelek (") közé kell tenni.

### Példa

Listázzuk ki az emp táblát a következőképpen: Az első oszlop (empno) neve legyen kód, az ename legyen "név", a sal oszlop pedig fizetés.

```
SQL> SELECT empno kód,
2         ename "név",
3         sal AS fizetés
4         FROM emp;
```

KÓD	név	FIZETÉS
7369	SMITH	800
7499	ALLEN	1600
7521	WARD	1250
...		

14 sor kijelölve.

### Példa

Írassuk ki az alkalmazottak nevét és belépési dátumát és éves fizetését, amely egy kifejezés lesz.

```
SQL> SELECT ename AS név,
2         hiredate AS "Belépési dát",
3         sal * 12 AS "Éves fizetés"
4         FROM emp;
```

NÉV	Belépési	Éves fizetés
SMITH	80-DEC-17	9600
ALLEN	81-FEB-20	19200
WARD	81-FEB-22	15000
...		

14 sor kijelölve.

Láthatjuk, hogy a listázás a fejléct csonkolja, de az idézőjelbe tett karakterformátumnak megfelelően jeleníti meg. A kiírás azért csonkolja a fejléct, mert az, az alapértelmezett dátumformátum mezőszélességénél hosszabb.

## Aritmetikai operátorok

Előfordulhat, hogy nem közvetlenül a tábla oszlopaira vagyunk kíváncsiak, hanem az oszlopok tartalmának valamilyen matematikai művelettel való kiértékelésére.

A NUMBER és DATE típusú adatoknál ezt könnyedén megtehetjük.

Az SQL utasítások valamennyi részében előfordulhatnak az alábbi operátorok, kivéve a FROM utasításrészt.

+	összeadás	*	szorzás
-	kivonás	/	osztás

Az *aritmetikai kifejezések* tartalmazhatnak oszlopneveket, konstans értékeket és aritmetikai operátorokat.

### Fontos

Az aritmetikai operátorok kiértékelése a matematika szabályainak felel meg (\* / + -). Azonos precedenciájú operátorok esetén a kiértékelés balról jobbra történik. A kiértékelés sorrendjét a zárójelek használatával változtathatjuk meg.

### Példa

Mennyi lenne az alkalmazottak fizetése, ha 8%-kal emelnék a havi jövedelmüket.

```
SQL> SELECT empno, ename, sal*0.08 as "Fizetésemelés összege",
2         1.08*sal as "új havi jövedelem"
3         FROM emp;
```

EMPNO	ENAME	Fizetésemelés összege	új havi jövedelem
7369	SMITH	64	864
7499	ALLEN	128	1728
7521	WARD	100	1350
7566	JONES	238	3213

14 sor kijelölve.

Megfigyelhetjük, hogy bármelyik oszlop (például: sal), egy lekérdezésen belül, több aritmetikai kifejezésben is előfordulhat.

Figyeljünk fel arra, hogy az előző példában olyan oszlop (úgynevezett pseudooszlop) látható a kiíratásban, mely nem szerepelt a táblában, hiszen számított értékeket tartalmaz.

*Pseudooszlop:*

Olyan oszlop, amely a SELECT utasítás paramétertáblájában nem szerepel, de az eredménytáblájában már igen. Általában valamilyen kifejezés, vagy rendszeradat értékét tartalmazza. (Ilyen például: a NULL érték, a sorazonosítót megadó ROWID, egy sor sorszámát megadó ROWNUM, az aktuális rendszerdátumot és időt megadó sysdate, az aktuális felhasználó azonosítóját megadó UID, és az aktuális felhasználó nevét megadó USER stb.)

### Példa

Számoljuk ki az évi keresetet úgy, hogy Nézzük meg az előző feladat helyes mindenkinek még havonta adunk 222 dollárt. megoldását (zárójelhasználattal)! (Az alábbi megoldás hibás!)

### Példa

```
SQL> SELECT empno AS "azonosító",
2         ename AS "név",
3         sal+222*12
4         FROM emp;
```

```
SQL> SELECT empno AS "azonosító",
2         ename AS "név",
3         (sal+222)*12
4         FROM emp;
```

azonosító név	SAL+222*12	azonosító név	(SAL+222) *12
7369 SMITH	3464	7369 SMITH	12264
7499 ALLEN	4264	7499 ALLEN	21864
7521 WARD	3914	7521 WARD	17664
7566 JONES	5639	7566 JONES	38364

14 sor kijelölve.

14 sor kijelölve.

**Fontos**

Mint az előző példákban láthatjuk, a két eredmény nem ugyanaz. A helyes eredmény csak a kifejezés megfelelő zárójelezésével érhető el.

Megjegyezzük, hogy az aritmetikai kifejezésekben a valós számok leírásánál tizedespont szerepel.

**Példa**

A dátumokkal is végezhetünk aritmetikai műveleteket. Listázzuk ki, hogy az alkalmazottak hány éve vannak a vállalatnál. Ha a mai nap dátumából kivonjuk a belépés dátumát, megkapjuk, hogy hány napja dolgoznak itt. Ha ezt az értéket osztjuk 365-tel, akkor az évek számát kapjuk meg.

```
SQL> SELECT ename AS "NÉV",
2         (sysdate-hiredate)/365 AS "ÉVEK SZÁMA"
3         FROM emp;
```

NÉV	ÉVEK SZÁMA
SMITH	20.8284575
ALLEN	20.6503753
WARD	20.6448959
JONES	20.5380465
MARTIN	20.0476356
BLAKE	20.4585945

14 sor kijelölve.

A matematikai függvényeknél majd megmutatjuk, hogyan lehet levágni, kerekíteni a számokat, mert így nem sok értelme van az ÉVEK SZÁMA értéknek.

**NULL ÉRTÉKEK AZ ARITMETIKAI KIFEJEZÉSEKBEN**

Az úgynevezett NULL, vagy üres "érték" akkor használjuk egy sor valamely oszlopában, ha az ahhoz logikailag tartozó adatot nem ismerjük. A NULL érték minden adattípusú oszlopban szerepelhet, ha ezt valamilyen megszorítás nem tiltja meg. Ha az oszlop NUMBER típusú, a NULL akkor sem egyenlő a nullával.

Mi történik ilyenkor az aritmetikai kifejezésekben.

**Példa**

Listázzuk ki, mennyi lesz a havi bér és jutalék összege az egyes alkalmazottak esetében.

```
SQL> SELECT ename AS "név",
2         (sal + comm) AS
3         "összes jövedelem"
```

(folytatás a következő oldalon)

```

4      FROM emp;

név      összes jövedelem
-----
SMITH
ALLEN      1900
WARD      1750
JONES
MARTIN      2650
BLAKE
CLARK
SCOTT
KING
TURNER      1500
ADAMS
JAMES
FORD
MILLER

```

14 sor kijelölve.

(folytatás az előző oldalról)

Valami hiba lehet, mert minden dolgozónak van havi bére, legfeljebb jutaléka nincs. A listában csak azoknál látszik értékelhető összes jövedelem, akiknél a comm oszlopban is van érték. Azoknál a dolgozóknál, akiknek a comm oszlopértéke NULL nem jelent meg az összes jövedelem oszlopban sem érték.

### Mi a teendő?

Megoldás az NVL függvény használata, amely egy numerikus oszlop NULL értékét a nullával helyettesíti.

A comm oszlopban a számítás elvégzésének idejére helyettesítsük nulla számértékkel az oszlopértéket. (A függvény használatára az SQL függvényeknél még visszatérünk.)

```

SQL> SELECT ename AS "név",
2      ( sal +nvl(comm,0) ) AS "összes jövedelem"
3      FROM emp;

```

```

név      összes jövedelem
-----
SMITH      800
ALLEN      1900
WARD      1750
JONES      2975
MARTIN      2650
BLAKE      2850
CLARK      2450
SCOTT      3000
KING      5000
TURNER      1500
ADAMS      1100
JAMES      950
FORD      3000
MILLER      1300

```

14 sor kijelölve.

Megfigyelhetjük, hogy SMITH, BLAKE, CLARK, stb. alkalmazottnak az előző lista összes jövedelem oszlopában nem szerepelt érték, az NVL függvény használatával azonban megjelent a helyes érték.

## Összefűzés operátor

Karakterlánc típusú oszlopértékeket, vagy konstansokat lehet segítségével összefűzni. Két függőleges (||) vonallal jelöljük. Eredménye egy új, karakterlánc típusú oszlop.

### Példa

Fűzzük össze az `ename` és a `job` oszlopokat.

```
SQL> SELECT ename||job AS alkalmazott
      2      FROM emp;

ALKALMAZOTT
-----
SMITHCLERK
ALLENSALESMAN
WARDSALESMAN
...
14 sor kijelölve.
```

Az utasítás összefűzte az `emp` és `job` oszlopokat, de nem értelmezhető, mert közvetlenül egymás mellé írja ezeket az értékeket. Nézzük meg, mit tehetünk, hogy értelmezhető legyen.

## Literálok

Literálnak nevezzük a `SELECT` listájában szereplő karakterlánc, szám, vagy dátum típusú konstansokat. A dátum és a karakteres literálértékeket aposztrófok közé kell tennünk. Minden egyes literál minden megjelenített sorban egyszer kerül kiírásra.

### Példa

Fűzzük össze az `ename` és a `job` oszlopokat úgy, hogy `SMITH` egy `CLERK` (hivatalnok), `ALLEN` egy `SALESMAN` (kereskedő) stb. legyen.

```
SQL> SELECT ename||' egy '||job AS "Alkalmazott és foglalkozása"
      2      FROM emp;

Alkalmazott és foglalkozása
-----
SMITH egy CLERK
ALLEN egy SALESMAN
WARD egy SALESMAN
JONES egy MANAGER
MARTIN egy SALESMAN
BLAKE egy MANAGER
CLARK egy MANAGER
SCOTT egy ANALYST
KING egy PRESIDENT
TURNER egy SALESMAN
...
14 sor kijelölve.
```

Azért nem írja egymás alá az ' egy ' literált, mert az `ename` oszlop `VARCHAR2` típusú (`ename VARCHAR2(10)`). Azaz pontosan annyi karaktert ír, amennyi a név karaktereinek száma, maximum a megengedett (10), majd utána soronként közvetlenül hozzáfűzi a megadott literált. (Az `LPAD`, `RPAD` függvényekkel ez a probléma kiküszöbölhető).

### Példa

Jelöljük meg a havi fizetés pénznemét úgy, hogy minden egyes lekérdezett sorban jelenjen meg.

```
SQL> SELECT ename AS "név", sal||' dollár' as "Havi fizetés"
       2 FROM emp;
```

```
név          Havi fizetés
-----
SMITH        800 dollár
ALLEN        1600 dollár
WARD         1250 dollár
...
14 sor kijelölve.
```

Láthatjuk, hogy a numerikus adatokhoz is hozzáfűzhetünk literált.

## DISTINCT paraméter

A `DISTINCT` utasítás-paraméter használatával lehet a `SELECT` eredménytáblájának azonos sorait kiszűrni. Használata az eredménytábla rendezését is eredményezi. (Lásd az `ORDER BY` utasításrésznel.)

### Példa

Listázzuk ki az `emp` táblából a különböző részlegek azonosítóit.

```
SQL> SELECT DISTINCT deptno AS "részlegek száma"
       2 FROM emp;
```

```
részlegek száma
-----
10
20
30
```

Láthatjuk, hogy eddig az `emp` tábla mind a 14 sorát kilistázta. Most azonban, mivel csak a `deptno` oszlop ismétlődéseit szűrve listáztattuk ki az `emp` táblát, a kilistázott sorok száma éppen 3.

### Példa

Listázzuk ki a részlegeket és a foglalkozásokat.

```
SQL> SELECT deptno AS "Részlegek",
       2 job AS "Munkakörök"
       3 FROM emp;
```

```
Részlegek Munkakörök
```

`DISTINCT` paraméterrel

```
SQL> SELECT DISTINCT deptno AS
       2 "Részlegek",
       3 job AS "Munkakörök"
       4 FROM emp;
```

-----	-----	Részlegek	Munkakörö
-----	-----	-----	-----
20	CLERK		
30	SALESMAN	10	CLERK
30	SALESMAN	10	MANAGER
20	MANAGER	10	PRESIDENT
30	SALESMAN	20	ANALYST
30	MANAGER	20	CLERK
10	MANAGER	20	MANAGER
20	ANALYST	30	CLERK
10	PRESIDENT	30	MANAGER
30	SALESMAN	30	SALESMAN
20	CLERK		
30	CLERK		
20	ANALYST		
10	CLERK		

14 sor kijelölve.

9 sor kijelölve.

Csak azokat a sorokat listázza, ahol a részleg és a foglalkozás is különböző, az egyezőket nem.

Több 20-as részlegben dolgozó `CLERK` van, és több 30-as részlegben dolgozó `SALESMAN`, de mindegyikből csak egyet listáz ki. Tehát az eredménytábla csak az oszlopok különböző kombinációit tartalmazza. Figyeljük meg, hogy a lista rendezett.

## Bevezetés a tábla kezelésébe

A következőkben bemutatjuk a tábla-létrehozás alapjait, azért, hogy az olvasó az eddigieket saját tábláin is kipróbálhassa, és önálló feladatokat is el tudjon végezni. (A tábla kezelésével részletesen a 9. fejezet "Adatbázis-objektumok kezelése" alfejezet foglalkozik.)

A táblakezelés részletezése előtt néhány szót kell szólnunk az Oracle-rendszer egy nagyon fontos, a biztonságos használatot elősegítő szolgáltatásáról. Mivel az adatbázis-kezelés alapvető objektuma az adattábla (hiszen a tárolt adatok végül is ebben helyezkednek el), ezért minden olyan tevékenység, amely ezekre irányul, különös gondosságot igényel. Az Oracle-rendszer elterjedtségének egyik fő oka éppen az, hogy az adatbázis-kezelő műveletek jelentős része (de nem mindegyik!) egy olyan, úgynevezett tranzakció-kezelő alrendszer felügyelete alatt áll, amelynek segítségével a már elvégzett beavatkozások hatásai visszaállíthatóak, a műveletek "visszagörgethetőek". (Ezzel részletesen foglalkozik a 9. fejezet "Adatbázis-tranzakciók" alfejezete.)

## Tábla létrehozás

A tábla létrehozása a `CREATE TABLE` utasítással történik. Az utasítás DDL utasításcsoportba tartozik. Az utasítás alakja:

```
CREATE TABLE tábla
  (oszlop adattípus [DEFAULT kif.]
  [, oszlop adattípus [DEFAULT kif.] ... );
```

ahol *tábla* a tábla általunk megadott neve,

<i>oszlop</i>	az oszlop elnevezése,
<i>adattípus</i>	az oszlop adattípusa és hossza,
<i>DEFAULT kife.</i>	az alapértelmezett értéket megadó kifejezés.

Alapértelmezett érték adattípusa egyezzen meg az oszlop adattípusával. Értéke lehet literál (azaz konstans), kifejezés vagy SQL függvény.

### Példa

Hozzunk létre egy adattáblát. Legyen ez egy nagyon egyszerű tábla, mégpedig a II.rész osztályozó műveleteinek bevezető példájánál ismertetett tábla (t001.sql).

```
SQL> CREATE TABLE eladas
2   (szalon      VARCHAR2(8),
3   ev           NUMBER(4),
4   bevetel      NUMBER(7),
5   marka        VARCHAR2(10));
```

A tábla létrejött.

Kérdezzük le a tábla szerkezetét.

```
SQL> DESC eladas
```

Név	Üres?	Típus
-----	-----	-----
SZALON		VARCHAR2(8)
EV		NUMBER(4)
BEVETEL		NUMBER(7)
MARKA		VARCHAR2(10)

A tábla tehát fizikailag létrejött. Lekérdezhetjük másképp is, mégpedig hogy benne van-e a felhasználói katalógusban, vagy a felhasználói táblák között.

```
SQL> SELECT * FROM user_catalog;
```

TABLE_NAME	TABLE_TYPE
-----	-----
...	
ELADAS	TABLE
HELP	TABLE
PROBA	TABLE
...	

### Tábla feltöltése

Ezek után a táblát adatokkal fel kell tölteni. Hogy is történik ez? Az SQL és SQL\*Plus környezetben csak egyetlen módszer használható tábla feltöltésére, mégpedig az DML utasítás-csoportba tartozó `INSERT` utasítás. Az `INSERT` utasítás egy új sort szúr be a táblázatba, mégpedig mindig csak egyet.

Az utasítás alakja:

```
INSERT INTO tábla [(oszlop [, oszlop]...)]
VALUES (érték [, érték]...);
```

ahol *tábla* táblanév,  
*oszlop* oszlopnév,  
*érték* a megadni kívánt érték.

Láthatjuk, hogy az oszloplista megadása opcionális. Oszloplista megadása esetén, csak a megadott oszlopokhoz, de azok mindegyikéhez meg kell adnunk adatot. A listában nem szereplő oszlop értéke az alapértelmezett érték, vagy ha ez nincs, akkor `NULL` érték lesz.

Ha nem szerepel az oszloplista, akkor a definiált összes oszlopnak rendre értéket kell adnunk. Ha az oszlopok között van olyan, amelynek értéke `NULL`, akkor sem hagyhatjuk ki az értékadásból, hanem `NULL` értéket kell megadnunk.

### Példa

Töltsük fel a létrehozott eladás táblát. Célszerű script programot készíteni, és figyelmes másolással feltölteni a táblát (`t001.sql`). Figyeljünk arra, hogy a karakteres és a dátumértékeket aposztrófok közé tegyük.

```
SQL> INSERT INTO eladas
2 VALUES ('Pipacs',1999,20000,'Jaguar');
```

1 sor létrejött.

```
SQL> INSERT INTO eladas
2 VALUES ('Szekér',2001,28000,'Opel');
```

1 sor létrejött.

...

Listázzuk ki az eladas táblát!

```
SQL> SELECT * FROM eladas;
```

SZALON	EV	BEVETEL	MARKA
Pipacs	1999	20000	Jaguar
Pipacs	1999	700	Mercedes
Pipacs	2000	14000	Jaguar
Pipacs	2000	2000	Mercedes
Pipacs	2001	10300	Jaguar
Pipacs	2001	5000	Mercedes
Szekér	1999	15000	Porsche
Szekér	1999	8700	Opel
Szekér	2000	12500	Porsche
Szekér	2000	22300	Opel
Szekér	2001	10500	Porsche
Szekér	2001	28000	Opel

## Tábla törlése

Ha a táblára már nincs szükség, vagy azzal azonos néven újat szeretnénk készíteni, a táblát töröljük ki.

A tábla törlése a `DROP TABLE` utasítással történik, amely nem vonható vissza. A tábla törlését csak a tulajdonos teheti meg (vagy akinek tábla-törlési joga van).

### Példa

```
SQL> DROP TABLE eladas;
```

A tábla eldobva.

## Az SQL\*Plus környezet

Miután már képesek vagyunk a legegyszerűbb `SELECT` utasításokat, lekérdezéseket megírni, tegyünk egy kis kitérőt. Célunk, hogy megismerkedjünk az SQL\*Plus környezettel, és így a bonyolultabb lekérdezéseket már az SQL\*Plus felület segítségével hajthassuk végre.

Az SQL az alkalmazások és az Oracle rendszer közötti kommunikációra szolgáló adatbázis-kezelő nyelv. Az SQL\*Plus környezetben egy SQL utasítás kiadása azt jelenti, hogy az utasítás az ún. SQL\*Plus puffertérületére kerül. Ezen a memóriaterületen addig található meg, míg egy újabb SQL utasítás nem érkezik.

Az SQL\*Plus egy olyan, az Oracle által kifejlesztett felhasználói felület, amely felismeri az SQL utasításokat, és elküldi azokat az Oracle adatbázis-kezelőnek. Mit tesz lehetővé ez a környezet?

- Lehetővé teszi, az utasítások közvetlen begépelését, és végrehajtását.
- Az állományban tárolt SQL utasítások beolvasását és végrehajtását.
- A beépített sorszerkesztő használatával az SQL utasítások módosítását.
- Kezeli a környezeti beállításokat.
- Képes a lekérdezések eredményét egyszerű riportformátumban megjeleníteni.

SQL\*Plus környezetben végrehajthatunk PL/SQL nyelvű modulokat, deklarálhatunk változókat, indíthatunk parancsállományokat. A SQL\*Plus interaktív közreműködése mellett képes az `.sql` kiterjesztésű állományok tartalmát is feldolgozni. Ezeket a parancsállományokat script programoknak nevezzük. A script programok egy, vagy több SQL utasítást tartalmazhatnak. Az `.sql` kiterjesztésű script programba az utasítások beírhatóak tetszőleges szövegszerkesztő segítségével (például a Windows jegyzettömbjében), de az SQL\*Plus környezetben kiadott utasítás az SQL\*Plus pufferből állományba menthető, vagy hozzáfűzhető egy már létező állományhoz.

Az SQL\*Plus tehát olyan utasításvégrehajtó környezet, amelynek használatával SQL utasításokat küldhetünk az adatbázis kiszolgálónak, illetve szerkeszthetjük és menthetjük azokat. A végrehajtani kívánt utasításokat megadhatjuk az SQL promptnál (`SQL>`), vagy megírhatjuk script programban.

## SQL\*Plus szerkesztési utasításai

Az SQL\*Plus szerkesztési utasításai megkönnyítik munkánkat. Ezeket soronként gépellhetjük be. Egyszerre csak egyet írhatunk, mert ezeket nem tárolja az SQL\*Plus puffer. Egy SQL\*Plus utasítást a sor végére írt kötőjel (-) után folytathatunk a következő sorban.

### Fontos

Az SQL\*Plus pufferbe csak az utoljára kiadott SQL parancs kerül. A rendszer nem támogatja az SQL\*Plus (tehát környezeti) utasítások e pufferbe kerülését, ebben való szerkesztését, innen való végrehajtását.

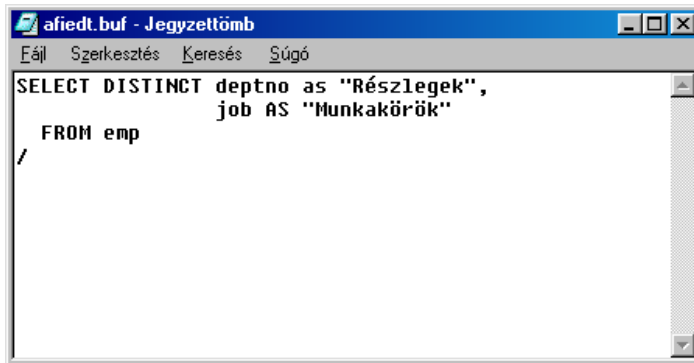
Az SQL\*Plus környezetben egy SQL utasítás beírásakor, ha az utasítás befejezése előtt leütjük az [Enter] billentyűt, az SQL\*Plus a következő sorban kiírja a sor számát.

```
SQL> SELECT *  
2
```

Az SQL\*Plus puffert pontosvesszővel (;), vagy törtvonallal (/) zárhatjuk le.  
Nézzük meg a puffer tartalmát. Ezt az EDIT (ED) SQL\*Plus utasítás kiadásával tehetjük meg.

```
SQL> EDIT
Kíírva: file afiedt.buf
```

Nézzük meg a 8.6.ábrát.



8.6.ábra. *afiedt.buf*

Megfigyelhetjük, hogy a pufferben pontosvessző helyett, amellyel az SQL utasításokat kell lezárni, a törtvonal helyettesítődik be.

Az SQL\*Plus puffer neve *afiedt.buf*. Megtalálható az Oracle program telepített (OraWin95, vagy Orahome1 stb.) könyvtárának BIN alkönyvtárában.

Az SQL\*Plus szerkesztési parancsok a következők:

A[PPEND] <i>szöveg</i>	Szöveget fűz az aktuális sor végéhez.
C[HANGE]/ <i>régi</i> / <i>új</i>	A <i>régi</i> szöveget az <i>újra</i> cseréli az aktuális sorban.
C[HANGE]/ <i>szöveg</i> /	Törli a <i>szöveget</i> az aktuális sorból.
CL[EAR] BUFF[ER]	Üríti az SQL puffert.
DEL	Törli az aktuális sort.
DEL <i>n</i>	Törli az <i>n</i> -edik sort sort.
DEL <i>n m</i>	Törli <i>n</i> -től <i>m</i> -ig a sorokat.
I[NPUT]	Sorokat szűr be.
I[NPUT] <i>szöveg</i>	Beszúr egy " <i>szöveget</i> " tartalmazó sort.
L[IST]	Listázza az SQL pufferben lévő sorokat.
L[IST] <i>n</i>	Megjeleníti a <i>n</i> -edik sort.
L[IST] <i>n m</i>	Listázza az <i>n</i> -től <i>m</i> -edik sorszámu sort.
R[UN]	Megjeleníti és futtatja az SQL pufferben lévő sorokat.
<i>n</i>	Megadja annak a sornak a számát, amit aktuálissá akarunk

tenni.

`n text` Az *n.* sort a *textre* cseréli.

`0 text` Beszúr egy sort az első sor elé.

A szögletes ( [ ] ) zárójelbe tett karakterek opcionálisak, azaz a parancsok rövidített alakja is érvényes. Próbáljunk ki néhányat. A többit az olvasóra bízunk.

### Példa

Az alábbi programban cseréljük ki a második utasítássorban a dollár szöveget forintra, majd ellenőrizzük, és futtassuk le.

Ez tehát, mint láttuk már jól működő `SELECT` utasítás.

```
SQL> SELECT ename AS "név",
2         sal||' dollár' AS "Havi fizetés"
3         FROM emp;
```

Második sor listázása, aktuálissá tétele.

```
SQL> l2
2*         sal||' dollár' as "Havi fizetés"
```

Cseréljük a szövegeket.

```
SQL> c/' dollár'/' forint'
2*         sal||' forint' as "Havi fizetés"
```

A teljes `SELECT` listázása.

```
SQL> l
1 SELECT ename AS "név",
2         sal||' forint' as "Havi fizetés"
3*        FROM emp
```

Futtassuk le.

```
SQL> r
1 SELECT ename AS "név",
2         sal||' forint' as "Havi fizetés"
3*        FROM emp
```

név	Havi fizetés
SMITH	800 forint
ALLEN	1600 forint
WARD	1250 forint
JONES	2975 forint
MARTIN	1250 forint
...	

Töröljük ki a második sort.

```
SQL> DEL2
SQL> l
```

```

1      SELECT ename AS "név",
2*      FROM emp

```

Ahhoz, hogy ez lefusson, az első sor végéről törölni kell a vesszőt (,).

```

SQL> l1
1*      SELECT ename AS "név",
SQL> C/,/
1*      SELECT ename AS "név"

```

Listázzuk ki.

```

SQL> l
1      SELECT ename AS "név"
2*      FROM emp

```

Fűzzünk az első sorhoz szöveget A[PPEND], azaz egy új oszlopot is listázzunk ki. Legyen ez a fizetés.

```

SQL> l1
1*      SELECT ename AS "név"
SQL> A,sal AS "Fizetés"
1      SELECT ename AS "név",sal AS "Fizetés"

```

Listázzuk a puffer tartalmát.

```

SQL> l
1      SELECT ename AS "név", sal AS "Fizetés"
2      FROM emp
3*

```

Futtassuk le.

```

SQL> r
1      SELECT ename AS "név",sal AS "Fizetés"
2      FROM emp
3*

```

```

név          Fizetés
-----
SMITH          800
ALLEN         1600
WARD          1250
...
14 sor kijelölve.

```

Mint láthatjuk a parancsok szerkesztése sokkal könnyebb, mintha soronként másoljuk, javítjuk azokat.

## SQL\*Plus állomány-kezelési utasítások

Munkánk további könnyítésére az alábbi SQL\*Plus állomány-kezelő utasításait használhatjuk. A megírt SQL utasításokat állományba menthetjük, onnan betölthetjük, szerkeszthetjük, közvetlenül nyomtatóra is küldhetjük.

SAV[E] <i>állomáynév</i> [.ext] [REP[LACE]APP[END]	Az Append opció használatával hozzáfűzhetjük a puffer tartalmát egy létező állományhoz, a Replace opcióval pedig felülírhatunk egy létező állományt. Az állomány alapértelmezett kiterjesztése .sql
GET <i>állomáynév</i> [.ext]	Beírja a pufferbe egy korábban mentett állomány tartalmát. Az állomány alapértelmezett kiterjesztése .sql
STA[RT] <i>állomáynév</i> [.ext] @ <i>állomáynév</i>	Futtat egy korábban mentett script programot. Futtat egy korábban mentett script programot. Hatása ugyanaz, mint a START utasításé.
ED[IT]	Elindítja a szerkesztőt, és a puffer tartalmát beírja az afiedt.buf állományba
ED[IT] [ <i>állomáynév</i> [.ext]]	Elindítja a szerkesztőt egy korábban már mentett állomány tartalmának szerkesztéséhez.
SPO[OL] <i>állomáynév</i> [.ext]   OFF   OUT]	Állományba menti a lekérdezések eredményét. Az OFF bezárja a gyűjtő állományt. Az OUT bezárja a gyűjtő állományt, és elküldi az eredményeket a rendszernyomtatóra.
EXIT	Kilépés az SQL*Plus környezetből.

## Magyarázó szöveg használata a script programokban

A futtatható állományokba magyarázó szöveget (megjegyzéseket) helyezhetünk el. Megjegyzéseket a következő formában írhatunk:

- `-- megjegyzés` A két kötőjelet követő sor rész tartalma megjegyzésnek számít.
- `REM[ARK]` Egy egysoros megjegyzés kezdetét jelöli. Csak script programban használható, és nem szerepelhet SQL utasításban belül.
- `/* megjegyzés */` Több soros megjegyzés, SQL utasításban is használható.

### Példa

Írjunk egy SELECT utasítást, amely listázza a dept táblát másodlagos oszlopnevekkel.

```
SQL> SELECT deptno AS " Részleg száma",
2         dname AS " Részleg neve",
3         loc AS " Telephely"
4     FROM dept;
```

Részleg száma	Részleg neve	Telephely
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

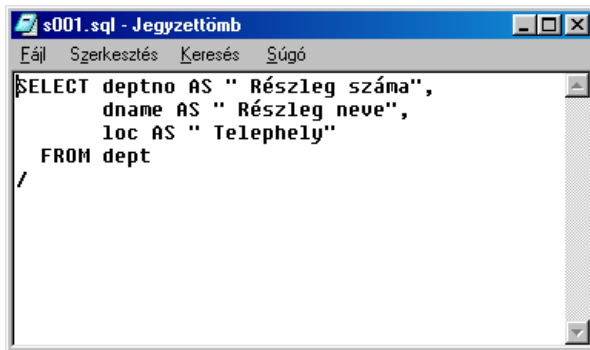
Mentsük el a C:\A könyvtárba s001.sql néven ezt a megírt SELECT utasítást.

```
SQL> SAVE C:\A\s001.sql
Létrehozva: file C:\A\s001.sql
```

Ha a menteni kívánt állománynak nem adunk meg elérési utat, oda menti, ahol az afiedt.buf állomány is van, tehát a bin könyvtárba. (Ne írjuk tele a bin könyvtárt felhasználói állományokkal!). Hívjuk be ezt az állományt a puffer szerkesztőjébe:

```
SQL> ED C:\A\s001.sql
```

Ez az ED[IT] SQL\*Plus utasítással tehető meg. Ennek az SQL\*Plus utasításnak a hatására megjelenik a jegyzettömb, a betöltött szerkesztendő állománnyal együtt. Ezt láthatjuk a 8.7.ábrán. Itt szerkeszthetjük, javíthatjuk, módosíthatjuk a tartalmat, majd a File menüpont, Mentés, vagy Mentés másképp almenüjével elmenthetjük.



8.7. ábra. Script program szerkesztése

## Script programok futtatása

A script programok indítása történhet:

- @ állománynév
- Start állománynév
- @@ állománynév

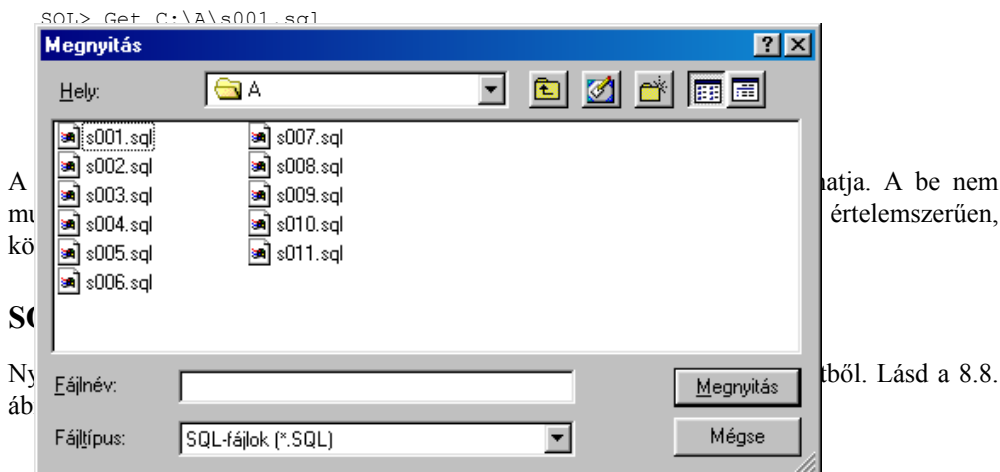
A @ elindít egy SQL\*Plus futtatható állományt. Az @@ működése hasonló, de az elindítandó állományt abban a könyvtárban keresi, amelyikben a hívó állomány van. (A @ a hívó parancsállomány indításakor az aktuális könyvtárban keres.)

Futtassuk is le a script programot.

```
SQL> @ C:\A\s001.sql
```

Részleg száma	Részleg neve	Telephely
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Töltsük be az állományt a pufferbe. Használjuk a `GET` SQL\*Plus utasítást



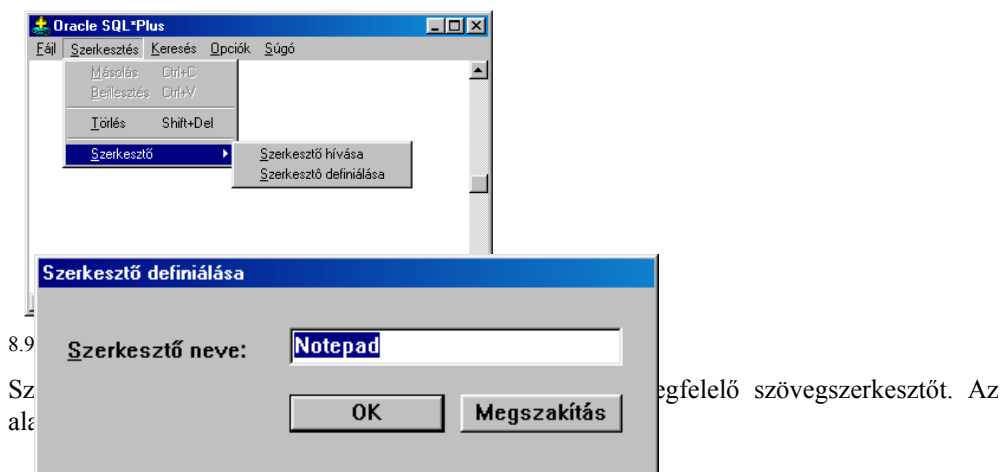
8.8.ábra. Állomány megnyitás SQL\*Plus környezetből

Közvetlenül az SQL\*Plus felületre, pufferbe tölti be az állományt.

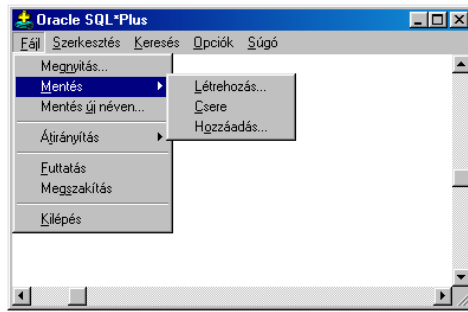
#### Megjegyzés:

Lesznek olyan szintaktikailag jó programok, amelyek nem futnak le. Miért? Azért, mert a puffer nem támogatja az SQL\*Plus utasításokat. A 8.8. ábrán látható programokat elláttuk már megjegyzésekkel, így azok sem futnak le, mert nincs pontosvessző a végén, és `REM[ARK]` megjegyzés van a legelején.

(Ezek a programok – mivel script programok – közvetlenül csak SQL\*Plus utasítással futtathatók le. Ezek a következők: `STA[RT]`, `@`, `@@`). Ha az állományt szerkeszteni szeretnénk, akkor a Szerkesztés menüt gördítsük le, amint azt a 8.9. ábrán láthatjuk. Kattintsunk a szerkesztő hívása menüpontra, és a jegyzetomb a betöltött állománnyal megjelenik.



8.1  
Sz  
áb  
az  
alr



athatjuk is az állományt, mint azt a 8.11. ábrán láthatjuk. Ha rákattintunk a Futtatás menüpontra, az SQL\*Plus elvégzi a megadott SQL utasítást. A Fájll menü, és a Mentés menü tartalmazza a számunkra megfelelő menüpontokat.

8.11. ábra. SQL\*Plus Fájll menüje.

Gyakorlatilag azokat a funkciókat tudjuk a menük segítségével végrehajtani, amelyeket az SQL prompt jel után kiadott SQL\*Plus utasításokkal. Az Átírányítás menüpont segítségével az állomány átírányítható .LST állományba is, melyet pedig már kinyomtathatunk.

## Az adatlekérdező nyelv (a SELECT utasítás általános bemutatása)

Miután megismerkedtünk az egyszerű SELECT utasítással, ennek használatával az SQL\*Plus környezetben, valamint a script programok szerkesztésével és futtatásával, lépünk egyet tovább. Nézzük meg, milyen utasításrészekkel bővíthető a SELECT utasítás. Mivel így már hosszabb utasításokat kapunk, ezért nem célszerű ezeket az SQL\*Plus környezetben közvetlenül begépelni, hanem inkább valamelyik – az olvasó által megkedvelt – script program írási lehetőségekkel elkészíteni, illetve módosítani.

A SELECT utasítás általánosabb alakját, az egyes utasításrészek bemutatása után ismertetjük. A teljes SELECT utasítás az Oracle8i referencia kézikönyvben megtalálható.

### WHERE utasításrész

Már említettük a SELECT utasítás bevezetésénél, hogy lehetőség van adott feltételeknek megfelelő sorok kiválasztására a paramétertáblából az eredménytáblába. Ekkor SELECT utasítás a feltételben szereplő logikai kifejezést teljesítő soroknak az utasításban megadott oszlopait listázza ki.

A WHERE utasításrész a FROM utasításrészt követi. Alakja:

```
WHERE feltétel (ek)
```

ahol

WHERE	a lekérdezést a feltételnek megfelelő sorokra korlátozza,
<i>feltétel</i>	oszlopnevekből, kifejezésekből, konstansokból és összehasonlító operátorokból álló logikai kifejezés, melynek értéke Igaz (True) vagy Hamis (False) lehet.

### Karakterláncok a WHERE feltételben

#### Példa

Tekintsünk egy egyszerű példát. Listázzuk ki az emp táblából a kereskedőket (salesman). Ezzel a listázandó sorok száma lényegesen csökken. Ha megnézzük az emp táblát, látjuk, hogy mindössze négy kereskedő található benne. A feladat megoldása a CD mellékleten, az s002.sql script programban található.

Ha önállóan akarjuk a feladatot megoldani, akkor készítsük el a programot a jegyzet-tömbben, vagy az SQL\*Plus felületen, és mentjük el egy script programba, egy számunkra megfelelő könyvtárba. Hibás futás esetén ezt könnyen tudjuk módosítani, javítani a puffer segítségével. A kijavított utasítássorozatot futtassuk le újra, s ha jó, akkor mentjük el.

A SELECT utasításunk bővül a WHERE utasításrésszel és azzal a feltétellel, hogy a munkakör (job) salesman legyen.

```

SQL> SELECT ename AS " név",          név      munkakör      részleg
2          job AS " munkakör",      -----
3          deptno AS " részleg"      ALLEN      SALESMAN      30
4 FROM emp                          WARD      SALESMAN      30
5 WHERE job = 'SALESMAN'            MARTIN     SALESMAN      30
                                     TURNER      SALESMAN      30

```

### Fontos

Amikor karaktersorozatot hasonlítunk össze a feltételben, egyrészt aposztrófok közé kell tenni az összehasonlítandó karakterláncot, másrészt teljes egyezés szükséges. Az összehasonlító algoritmus *megkülönbözteti a kis és a nagybetűket*. Automatikus betűkonvertálás az SQL utasítások feldolgozásában nem szerepel, de karakterkonvertáló függvények megtalálhatók az SQL nyelvben (UPPER, LOWER, stb).

A dátum is karaktersorozat, a dátumérték formátuma kötött. Az alapértelmezett dátumformátum: DD-MON-YY.

### Példa

Listáztassuk ki annak az alkalmazottnak az azonosító számát, a nevét, a főnökének azonosító számát és a havi fizetését, akinek belépési dátuma 1981. április 2-án van.

A megoldást az s003.sql script program tartalmazza.

```

SQL> SELECT empno AS "azonosító", ename AS " név",
2          mgr AS "főnök azonosítója", sal AS "fizetés"
3 FROM emp
4 WHERE hiredate = '81-ÁPR-02'

```

azonosító	név	főnök azonosítója	fizetés
7566	JONES	7839	2975

## Összehasonlító operátorok

Az összehasonlító operátorokat két kifejezés értékének összehasonlítására használjuk.

A WHERE utasításrész feltételében két kifejezést nemcsak az egyenlőség jellel (=) lehet összehasonlítani, mint ahogyan azt az előbbi példákön láthattuk, hanem egyéb összehasonlító operátorok használata is megengedett. Az operátorok mindegyike működik oszlopokra, kifejezésekre és literálokra is.

A WHERE utasításrészben a következő alakban használhatók:

WHERE *kifejezés operátor összehasonlító-érték*

Az összehasonlító operátorok az alábbiak

=	egyenlő	<	kisebb, mint
>	nagyobb, mint	<=	kisebb, vagy egyenlő
>=	nagyobb, vagy egyenlő	<>	nem egyenlő

### Példa

Listázzuk ki azoknak az alkalmazottaknak a nevét, a fizetését és a jutalékát, akiknek jutaléka kisebb a havi fizetés felénél. A megoldást az s004.sql script program tartalmazza.

SQL>	SELECT	ename AS név,	NÉV	FIZETÉS	JUTALÉK
2		sal AS fizetés,	-----	-----	-----
3		comm AS jutalék	ALLEN	1600	300
4	FROM	emp	WARD	1250	500
5	WHERE	comm < sal/2	TURNER	1500	0

Megfigyelhetjük, hogy a `WHERE` utasításrész nem tartalmaz explicit értéket (`comm < sal/2`). A két összehasonlítandó érték a relációs adattábla két oszlopának a `sal` és `comm` oszlopok soronkénti megfelelő értéke.

## BETWEEN ... AND operátor

Az összehasonlítás két határ által megadott intervallumba eső értéket vizsgál. Kiértékeli a tartományba eső értékeket, és ha ezek megfelelnek a feltételnek, az őket tartalmazó sorokat listázza ki a `SELECT` utasítás. A `BETWEEN ... AND` operátor értéktartományt vizsgál. Az alsó és felső határ beletartozik a vizsgált tartományba. Mindig az *alsó* értékhatárt kell először megadni!

### Példa

Keressük meg azokat az alkalmazottakat, akik 1981. április és szeptember között léptek be a vállalathoz. Listázzuk ki a nevüket (`ename`), a fizetésüket (`sal`) és a belépési dátumukat (`hiredate`). A megoldást az `s005.sql` script program tartalmazza. (Ha a fejléc hosszabb, mint a karakteres típus mezőszélessége, akkor azt csonkolja.)

SQL>	SELECT	ename AS név,	NÉV	FIZETÉS	Belépési
2		sal AS fizetés,	-----	-----	-----
3		hiredate AS "Belépési dátum"	JONES	2975	81-ÁPR-02
4	FROM	emp	MARTIN	1250	81-SZE-28
5	WHERE	hiredate	BLAKE	2850	81-MÁJ-01
6		BETWEEN '81-ÁPR-01'	CLARK	2450	81-JÚN-09
7		AND '81-SZE-30'	TURNER	1500	81-SZE-08

### Példa

Listázzuk ki azokat az alkalmazottakat, akiknek fizetése 1000 és 2000 dollár között van. A megoldást az `s006.sql` script program tartalmazza.

SQL>	SELECT	ename AS név,	NÉV	FIZETÉS
2		sal AS fizetés	-----	-----
3	FROM	emp	ALLEN	1600
4	WHERE	sal BETWEEN 1000 AND 2000	WARD	1250
			MARTIN	1250
			TURNER	1500
			ADAMS	1100
			MILLER	1300

6 sor kijelölve.

## IN operátor

Az `IN` operátorral azokat a rekordokat választhatjuk ki, amelyeknek a feltételben megadott oszlopértékei szerepelnek az `IN` operátor utáni zárójelbe tett felsorolásban. Az `IN` operátort valamennyi adattípussal használhatjuk. A karakter és a dátumértéket aposztrófok közé kell tenni.

### Példa

Keressük azoknak az alkalmazottaknak a nevét és a fizetését, akiknek az azonosító számuk (`empno`) 7521 és 7844.

A megoldást az `s007.sql` script program tartalmazza.

SQL>	SELECT	ename	AS	név,	NÉV	FIZETÉS
2		sal	AS	fizetés	-----	-----
3	FROM	emp			TURNER	1500
4	WHERE	empno	IN	(7521,7844);	WARD	1250

## LIKE operátor

A `LIKE` operátor mintaillesztést végez. Azt hasonlítja össze, hogy az adott karaktermintának melyik oszlopérték felel meg. A karakterminta karakteres literálokat és számokat is tartalmazhat. Értelmezhetünk ún. joker (helyettesítő) karaktereket is.

A joker karakterek kétfélék lehetnek:

- a `%` jel nulla számú, vagy több karakter helyettesítésére szolgál
- az `_` (alulvonás) egy karaktert helyettesít.

Joker karaktereket akkor használunk, ha nem ismerjük a pontos karakterláncot, azaz a keresendő értéket. A mintaegyeztetési (joker) karaktereket kombinálhatjuk is.

### Példa

Keressük azoknak az alkalmazottaknak a neveit, akiknek neve `M` betűvel kezdődik. Mivel nem ismerjük a pontos mintát, ezért helyettesítő karaktert használunk.

A megoldást az `s008.sql` script program tartalmazza.

SQL>	SELECT	ename	AS	név	NÉV
2	FROM	emp			-----
3	WHERE	ename	LIKE	'M%'	MARTIN
					MILLER

A helyettesítő karakterek kombinálhatók is, ahogy azt a következő példán megfigyelhetjük.

### Példa

Keressük azokat az alkalmazottakat, akik nevének harmadik betűje `N`, a többit nem tudjuk.

A megoldást az `s009.sql` script program tartalmazza.

SQL>	SELECT	ename	AS	név	NÉV
2	FROM	emp			-----
3	WHERE	ename	LIKE	'__N%'	JONES
					KING

A `LIKE` operátort egyes `BETWEEN` utasítások rövidítéseként is használhatjuk. A dátum is karaktersorozat, tehát a `'87%'` tartalmaz minden 87-es dátumot, mivel a `%` a további karaktereket helyettesíti.

### Példa

Keressük azon alkalmazottakat, akik 1987. jan.1 és dec. 31 között léptek be a vállalathoz.

A megoldást az `s010.sql` script program tartalmazza.

```
SQL> SELECT ename AS név,          NÉV          Belépési
      2      hiredate AS "Belépési dátum"  -----
      3      FROM emp                SCOTT      87-ÁPR-19
      4      WHERE hiredate LIKE '87%'    ADAMS      87-MÁJ-23
```

## IS NULL operátor

Az `IS NULL` operátorral megvizsgálhatjuk, hogy egy érték `NULL` érték-e. A `NULL` érték még szám esetén sem jelenti a 0 számot. A `NULL` értéket nem kereshetjük egyenlőség (`=`) operátorral, hiszen egyetlen értékkel sem egyenlő. A `NULL` értékeket egy úgynevezett pszeudooszlop tartalmazza.

### Példa

Keressük azokat a dolgozókat, akik nem kapnak jutalékot.

A megoldást az `s011.sql` script program tartalmazza.

```
SQL> SELECT ename AS név,          NÉV          Részleg száma      JUTALÉK
      2      deptno AS "Részleg száma",  -----
      3      comm AS jutalék            SMITH              20
      4      FROM emp                  JONES              20
      5      WHERE comm IS NULL        BLAKE              30
                                     CLARK              10
                                     ...
                                     10 sor kijelölve.
```

## LOGIKAI operátorok

A logikai operátorok logikai műveleteket végeznek, és ennek alapján egyetlen eredményt hoznak létre, illetve ellenkezőjére fordítják egyetlen feltétel eredményét.

A logikai operátorok az alábbiak:

- `AND` igaz értéket ad, ha mindkét feltétel igaz.
- `OR` igaz értéket ad, ha legalább az egyik feltétel igaz.
- `NOT` igaz értéket ad, ha az operátort követő feltétel nem igaz (hamis).

A logikai operátorok eredménye az igazságtábla alapján mindig meghatározható.

Az igazságtáblák a következők:

1. táblázat. Igazságtáblák

<i>AND</i>	i	h	NULL
i	i	h	NULL
h	h	h	h
NULL	NULL	h	NULL

<i>OR</i>	i	h	NULL
i	i	i	i
h	i	h	NULL
NULL	i	NULL	NULL

i: Igaz (TRUE)  
h: Hamis (FALSE)

i: Igaz (TRUE)  
h: Hamis (FALSE)

## Fontos

A logikai operátorok kiértékelési sorrendje: NOT, AND, OR. A logikai operátorok kiértékelését megelőzi bármely eddig bemutatott összehasonlító operátor (BETWEEN...AND, LIKE, IN stb.). Először az összehasonlító operátorok kiértékelése történik, majd a NOT operátort előbb, és utoljára az OR operátort. Természetesen ez a kiértékelési sorrend zárójelek használatával megváltoztatható.

## AND OPERÁTOR

Az **AND** operátor által összehasonlított minden feltételnek igaznak kell lennie, hogy az utasítás kiválassza a rekordot. Az **AND** operátor csak akkor választ ki sorokat, ha a kiértékelésben feltüntetett minden feltétele igaz.

## Példa

Listázzuk ki azokat, akiknek munkaköre hivatalnok (clerk), valamint akiknek havi fizetése több mint 1100 dollár. Figyeljünk arra, hogy a kis és a nagybetűk minden karakteres keresésben különbözőeknek számítanak. A karakterláncokat aposztrófok közé kell tenni!

A megoldást az `s013.sql` script program tartalmazza.

SQL>	SELECT	ename	név,	NÉV	MUNKAKÖR	FIZETÉS
2		job	AS munkakör,	-----	-----	-----
3		sal	AS fizetés	ADAMS	CLERK	1100
4	FROM	emp		MILLER	CLERK	1300
5	WHERE	job	= 'CLERK'			
6	AND	sal	>= 1100;			

**OR OPERÁTOR**

Az OR operátor a logikai "vagy" kapcsolatot jelenti két feltétel között. Értéke igaz lesz, ha vagy az egyik, vagy a másik, vagy mindkét kiértékelt feltétel igaz.

## Példa

Listázzuk ki azokat az alkalmazottakat, akiknek a fizetése kisebb, mint 1500, vagy a jutaléka nem NULL értékű.

A megoldást az `s012.sql` script program tartalmazza.

SQL>	SELECT	ename AS "Név",	Név	Fizetés	Jutalék
2	sal AS "Fizetés",	-----	-----	-----	-----
3	comm AS "Jutalék"	SMITH	800		
4	FROM emp	ALLEN	1600	300	
5	-- OR kapcsolat	WARD	1250	500	
6	WHERE sal < 1200	MARTIN	1250	1400	
7	OR (comm IS NOT NULL);	TURNER	1500	0	
		ADAMS	1100		
		JAMES	950		

Mindazokat a sorokat kilistáztattuk, amelyek a két feltételt az igazságtáblának megfelelően külön-külön, vagy egyszerre kielégítik, azaz SMITH, ADAMS, JAMES, valamint ALLEN, WARD, MARTIN, TURNER. Ez igaz, mert TURNER jutalékának értéke 0, azaz nem NULL.

## NOT OPERÁTOR

A NOT operátor a feltételek tagadását jelenti. A NOT operátort a BETWEEN, LIKE, IN és NULL operátorokkal együtt is használhatjuk.

### Példa

Listázzuk ki azon alkalmazottakat, akik *nem* 1981.jan.1 és dec.31 között léptek be a vállalathoz, és a munkakörük *nem* analyst (elemző). A megoldást az s014.sql fájl tartalmazza.

```
SQL> SELECT ename AS név, hiredate AS "Belépés", job AS munkakör
2 FROM emp
3 WHERE hiredate NOT LIKE '81%'
4 AND job NOT IN ('ANALYST');
```

NÉV	Belépés	MUNKAKÖR
SMITH	80-DEC-17	CLERK
ADAMS	87-MÁJ-23	CLERK
MILLER	82-JAN-23	CLERK

### Példa

Listázzuk ki azokat az alkalmazottakat, akik nevében *nincs* A betű, és nem 1981.január 1 és december 31. között léptek be a céghez.

A megoldást az s015.sql script program tartalmazza.

```
SQL> SELECT ename AS név,
2 hiredate AS "Belépés",
3 job AS munkakör
4 FROM emp
5 WHERE ename NOT LIKE '%A%'
6 AND hiredate NOT BETWEEN
7 '81-JAN-01' AND '81-DEC-31';
```

NÉV	Belépési	MUNKAKÖR
SMITH	80-DEC-17	CLERK
SCOTT	87-ÁPR-19	ANALYST
MILLER	82-JAN-23	CLERK

## ORDER BY utasításrész

Az utasításrész hatására az Oracle rendezi a már megfelelő szempontok (WHERE utasításrész) szerint leválogatott rekordokat. Az ORDER BY utasításrész a SELECT utasítás legutolsó része.

A rendezés történhet oszlopnév, kifejezés (pszeudooszlop), vagy másodlagos oszlopnév szerint. A sorok rendezése egy vagy több szempont szerint is történhet. Több szempont szerinti rendezés esetében a sorok rendezése először az elsőként megadott paraméter alapján, majd ezen belül a másodikként megadott paraméter alapján, stb. történik. A rendezés a beállított paraméter szerint lehet növekvő (ascent) vagy csökkenő (descent). A rendezés iránya minden oszlop, illetve kifejezés után beállítható.

Az eddig ismertetett típusú oszlopok mindegyike szerint lehet rendezni.

Az utasításrész egyszerű alakja:

```
ORDER BY {oszlopnév | kifejezés | másodlagos oszlopnév} [ASC | DESC]
[ {, oszlopnév | , kifejezés | , másodlagos oszlopnév} [ASC | DESC] ] ...
```

ahol *kifejezés* függvények halmaza  
 ASC növekvő sorrend,  
 DESC csökkenő sorrendet jelent.

Alapértelmezett rendezési szempont a *növekvő rendezés*. Ebben az esetben a NULL érték a legutolsó kilistázott érték. Csökkenő rendezés esetén a NULL érték az első a listában.

Ha nem használunk ORDER BY utasításrészt, a rendezési sorrend definiálatlan, és elképzelhető, hogy az Oracle rendszer ugyanannak a lekérdezésnek az eredményét minden alkalommal más sorrendben jeleníti meg.

Olyan oszlop szerint is rendezhetünk, amely nem szerepel a SELECT kulcsszó utáni listában, sőt pszeudooszlop szerint is végezhető rendezés.

A SELECT utasításnak az újonnan bevezetett utasításrészekkel kibővített alakja:

```
SELECT { *
      | {oszlop [[AS] másodlagos oszlopnév]
        | kifejezés [[AS] másodlagos oszlopnév]}
      [, {oszlop [[AS] másodlagos oszlopnév]
        | kifejezés [[AS] másodlagos oszlopnév]} ] ... }
FROM táblanév;
[WHERE feltételek]
[ORDER BY {oszlopnév | kifejezés | másodlagos oszlopnév} [ASC | DESC]
[, {oszlopnév | kifejezés | másodlagos oszlopnév} [ASC | DESC] ... ]
```

Ismételten felhívjuk a figyelmet arra, hogy az ORDER BY utasításrész a SELECT utasítás *utolsó* része kell, hogy legyen!

### Példa

Listázzuk ki az emp tábla sorait (ename, deptno, hiredate oszlopait) a belépési dátum szerint rendezve.

Megoldás az s016.sql script programban.

Rendezzük ugyanezt fordítva

Megoldás az s017.sql script programban.

```
SQL> SELECT ename as név,
2         deptno AS részleg,
3         hiredate AS "belépés"
4       FROM emp
5       -- alapértelmezett rendezés
6       ORDER BY hiredate;
```

NÉV	RÉSZLEG	belépés
SMITH	20	80-DEC-17
ALLEN	30	81-FEB-20
WARD	30	81-FEB-22
JONES	20	81-ÁPR-02
BLAKE	30	81-MÁJ-01
CLARK	10	81-JÚN-09
TURNER	30	81-SZE-08
MARTIN	30	81-SZE-28
KING	10	81-NOV-17

```
SQL> SELECT ename as név,
2         deptno AS részleg,
3         hiredate AS "belépés"
4       FROM emp
5       -- fordított rendezés DESC
6       ORDER BY hiredate DESC;
```

NÉV	RÉSZLEG	belépés
ADAMS	20	87-MÁJ-23
SCOTT	20	87-ÁPR-19
MILLER	10	82-JAN-23
JAMES	30	81-DEC-03
FORD	20	81-DEC-03
KING	10	81-NOV-17
MARTIN	30	81-SZE-28
TURNER	30	81-SZE-08
CLARK	10	81-JÚN-09

JAMES	30	81-DEC-03	BLAKE	30	81-MÁJ-01
FORD	20	81-DEC-03	JONES	20	81-ÁPR-02
MILLER	10	82-JAN-23	WARD	30	81-FEB-22
SCOTT	20	87-ÁPR-19	ALLEN	30	81-FEB-20
ADAMS	20	87-MÁJ-23	SMITH	20	80-DEC-17

**Példa**

Rendezzük az emp tábla azon sorait az éves fizetés szerint, amelyekben a job lehet SALESMAN és CLERK. A megoldást az s020.sql script program tartalmazza.

SQL> SELECT	ename AS "név",	név	részleg	fizetés
2	deptno AS "részleg",	-----	-----	-----
3	12*sal AS "fizetés"	SMITH	20	9600
4	FROM emp	JAMES	30	11400
5	WHERE job	ADAMS	20	13200
6	IN ('SALESMAN','CLERK')	WARD	30	15000
7	ORDER BY "fizetés";	MARTIN	30	15000
		MILLER	10	15600
		TURNER	30	18000
		ALLEN	30	19200

8 sor kijelölve.

**Példa**

Rendezzük az emp tábla sorait két szempont szerint: elsőnek a részleg, majd a fizetés szerint.

Megoldás az s018.sql script programban.

```
SQL>SELECT ename AS név,
2      deptno AS részleg,
3      job AS munkakör
4  FROM emp
5  ORDER BY részleg,
6      munkakör DESC;
```

NÉV	RÉSZLEG	MUNKAKÖR
-----	-----	-----
KING	10	PRESIDENT
CLARK	10	MANAGER
MILLER	10	CLERK
JONES	20	MANAGER
SMITH	20	CLERK
ADAMS	20	CLERK
SCOTT	20	ANALYST
FORD	20	ANALYST
ALLEN	30	SALESMAN
MARTIN	30	SALESMAN
WARD	30	SALESMAN
TURNER	30	SALESMAN
BLAKE	30	MANAGER
JAMES	30	CLERK

**Példa**

Rendezzük tovább, harmadik szempont a szerint. harmadik szempont a névsor szerinti rendezés legyen.

Megoldás az s019.sql script programban.

```
SQL> SELECT ename AS név,
2      deptno AS részleg,
3      job AS munkakör
4  FROM emp
5  ORDER BY részleg,
6      munkakör DESC, név;
```

NÉV	RÉSZLEG	MUNKAKÖR
-----	-----	-----
KING	10	PRESIDENT
CLARK	10	MANAGER
MILLER	10	CLERK
JONES	20	MANAGER
ADAMS	20	CLERK
SMITH	20	CLERK
FORD	20	ANALYST
SCOTT	20	ANALYST
ALLEN	30	SALESMAN
MARTIN	30	SALESMAN
TURNER	30	SALESMAN
WARD	30	SALESMAN
BLAKE	30	MANAGER
JAMES	30	CLERK

Nézzük meg a `DISTINCT` paraméter hatását. Minden név különböző, ne soroljuk fel a név oszlopot a szelekciós listában. Mi történik akkor? (Az `s021.sql` script program tartalmazza a megoldást.)

#### Példa

```
SQL> SELECT DISTINCT
2      deptno AS részleg,
3      sal as fizetés
4      FROM emp;
```

RÉSZLEG	FIZETÉS
10	1300
10	2450
10	5000
20	800
20	1100
20	2975
20	3000
30	950
30	1250
30	1500
30	1600
30	2850

12 sor kijelölve.

Megfigyelhetjük, hogy az azonosakat nem listázza kétszer. Ezek a következők: a 20-as részleg 3000-es, és a 30-as részleg 1250-es sora. Ezért a listának 12 sora van.

Listázzuk ki az `emp` tábla részleg, fizetés és név oszlopait rendezve úgy, hogy az első szempont a részleg, a második a fizetés alapértelmezett sorrendje legyen. Vizsgáljuk meg a `DISTINCT` paraméter hatását. (A megoldás az `s022.sql` script programban található.)

#### Példa

```
SQL> SELECT
2      deptno AS részleg,
3      sal AS fizetés,
4      ename AS név
5      FROM emp
6      ORDER BY részleg, fizetés;
```

RÉSZLEG	FIZETÉS	NÉV
10	1300	MILLER
10	2450	CLARK
10	5000	KING
20	800	SMITH
20	1100	ADAMS
20	2975	JONES
20	3000	SCOTT
20	3000	FORD
30	950	JAMES
30	1250	WARD
30	1250	MARTIN
30	1500	TURNER
30	1600	ALLEN
30	2850	BLAKE

#### Példa

```
SQL> SELECT DISTINCT
2      deptno AS részleg,
3      sal AS fizetés,
4      ename AS név
5      FROM emp;
```

RÉSZLEG	FIZETÉS	NÉV
10	1300	MILLER
10	2450	CLARK
10	5000	KING
20	800	SMITH
20	1100	ADAMS
20	2975	JONES
20	3000	FORD
20	3000	SCOTT
30	950	JAMES
30	1250	MARTIN
30	1250	WARD
30	1500	TURNER
30	1600	ALLEN
30	2850	BLAKE

Az eredmény, mint láthatjuk, a teljes egyezés a két különböző megoldás esetében.

## Függvényhasználat a SELECT utasításban

A függvény egy előre meghatározott művelet, amely egy SQL utasításban nevének és aktuális paramétereinek megadásával hívható meg, és mindig értéket ad vissza. A függvények a következőkre használhatók:

- adatokkal végzett számításokra
- egyedi adatelemek módosítására
- sorok csoportján végzett műveletekre
- megjelenítendő dátumok és számok formázására
- oszlopok adattípusának konvertálására

Az Oracle SQL-ben számos függvény található, amelyek két nagy csoportba oszthatók, ezek az egysoros, és a többsoros, azaz csoportfüggvények. Az egysoros függvények egyszerre egy soron végeznek műveletet, és soronként egy eredményt adnak vissza. Lehetnek:

- karakterkezelő
- számkezelő (numerikus)
- dátumkezelő
- konverziós
- általános függvények

A csoportfüggvények a sorok csoportjain végeznek műveletet, és csoportonként egy eredményt adnak vissza (például átlag). Megjegyezzük, hogy a fent felsorolt függvényeken kívül még továbbiak is léteznek.

### Egysoros függvények

Az egysoros függvényeket adatelemek kezelésére használjuk. A függvényeknek egy vagy több bemenő paraméterük lehet, és minden sorhoz egy értéket adnak vissza.

Használhatók a SELECT, a WHERE és az ORDER BY utasításrészben. A függvények egymásba is ágyazhatók.

A függvényhívás alakja:

*függvénynév* ({*oszlop* | *kifejezés*} [, *paraméter*]...)

ahol *függvénynév* a függvény neve,  
*oszlop* bármelyik adattábla oszlop,  
*kifejezés* bármilyen számított kifejezés vagy karakterlánc,  
*paraméterek* a függvény által használandó paraméterek.

### KARAKTERKEZELŐ FÜGGVÉNYEK

A karakterkezelő függvények bemenő paramétere karakter, karaktersorozat (sztring), kimenő értéke karakter, vagy numerikus érték. A karakterkezelő függvények az alábbiak:

LOWER({ <i>oszlop</i>   <i>kifejezés</i> })	A karakterlánc minden elemét kisbetűvé alakítja át.
UPPER({ <i>oszlop</i>   <i>kifejezés</i> })	A karakterlánc minden elemét nagybetűvé alakítja át.
INITCAP({ <i>oszlop</i>   <i>kifejezés</i> })	Minden szó első betűjét nagybetűvé, a többi kisbetűvé alakítja át.

CONCAT({oszlop1   kifejezés1}, {oszlop2   kifejezés2})	A két megadott karakterláncot összefűzi. Azonos a (  ) összefűzés operátorral.
SUBSTR({oszlop kifejezés}, m[, n])	A karakterlánc <i>m</i> -edik pozíciójától kezdődően <i>n</i> karaktert ad vissza.
LENGTH({oszlop   kifejezés})	A karakterlánc hosszát adja vissza.
INSTR({oszlop   kifejezés}, n)	A megnevezett karakter karakterpozíciójának sorszámát adja vissza.
LPAD({oszlop   kifejezés}, n, 'kitöltő')	Jobbra igazítja a karakterláncot úgy, hogy balról kitölti a megadott <i>kitöltő</i> karakterrel, összesen <i>n</i> karakter hosszúra.
RPAD({oszlop   kifejezés}, n, 'kitöltő')	Balra igazítja a karakterláncot úgy, hogy jobbról kiegészíti a <i>kitöltő</i> karakterrel, hogy a hossza éppen <i>n</i> legyen.

#### Példa (az INITCAP, UPPER függvény használatára)

Jelenítsük meg azoknak az alkalmazottaknak a nevét (csak nagy kezdőbetűvel), akiknek foglalkozása *analyst* (lásd *s023.sql*).

```
SQL> SELECT INITCAP(ename),
2         job AS foglalkozás
3       FROM emp
4      WHERE job = UPPER('analyst');
```

```
INITCAP(EN FOG LALKOZ
-----
Scott      ANALYST
Ford       ANALYST
```

#### Példa

Listázzuk ki az *emp* táblából azokat a sorokat, ahol a névben van 'T' betű. Listázzuk ki a név hosszát is. Keressük meg a 'T' betű hányadik a névben. Fűzzük össze a nevet és foglalkozást, de közéjük helyezzük el az ' egy ' szócskát. Megjegyezzük, hogy a *CONCAT* függvény mindig csak két paramétert fűz össze (lásd *s024.sql*).

```
SQL> SELECT CONCAT(ename, ' egy ') || INITCAP(job) AS
2         "Név és foglalkozás",
3         LENGTH(ename) AS "név hossza",
4         INSTR(ename, UPPER('t')) AS "T betű sorszáma"
5       FROM emp
6      WHERE INSTR(ename, UPPER('t')) <> 0;
```

```
Név és foglalkozás      név hossza T betű sorszáma
-----
SMITH egy  Clerk        5              4
MARTIN egy  Salesman    6              4
SCOTT egy  Analyst      5              4
TURNER egy  Salesman    6              1
```

**Példa** (az RPAD, LPAD, SUBSTR függvény használatára)

Listázzuk ki az emp táblából a céghez decemberben belépők azonosítószámait jobbra igazítva, kitöltve a ”” karakterrel, nevüket, valamint fizetésüket, ez utóbbit balról kitöltve a ”-” kötőjellel és a belépési dátumukat.

Az RPAD és LPAD függvények adott hosszra kitöltik a kiírandó karakterláncot, a SUBSTR kivágja a megadott karakterpozíciótól (4) kezdve, adott számú (3) karaktert. (Lásd s025.sql)

```
SQL> SELECT RPAD(empno,10,' ') AS azonosító,
2         ename AS név,
3         LPAD(sal,10,'-') AS fizetés,
4         hiredate AS belépés
5 FROM emp
6 WHERE SUBSTR(hiredate,4,3) = 'DEC';
```

AZONOSÍTÓ	NÉV	FIZETÉS	BELÉPÉS
7369*****	SMITH	-----800	80-DEC-17
7900*****	JAMES	-----950	81-DEC-03
7902*****	FORD	-----3000	81-DEC-03

Ha az utolsó karakter pozíciója nem ismert, akkor a SUBSTR második paramétere -1 lehet.

**Példa**

Listázzuk ki azokat az alkalmazottakat, akik nevének utolsó betűje ”R”.

```
SQL> SELECT ename AS név
2 FROM emp
3 WHERE SUBSTR(ename,-1,1)='R';
```

```
NÉV
-----
TURNER
MILLER
```

Ezek után már könnyű szépen összefűzni a nevet és a foglalkozást az emp tábla listázásakor. (lásd s026.sql).

```
SQL> SELECT RPAD(INITCAP(ename),10,' ') || LOWER(job)
2 AS "név és foglalkozás"
3 FROM emp;
```

```
név és foglalkozás
-----
Smith      clerk
Allen      salesman
Ward       salesman
Jones      manager
Martin     salesman
Blake      manager
Clark      manager
Scott      analyst
King       president
Turner     salesman
Adams      clerk
```

```
James      clerk
Ford       analyst
Miller     clerk
```

## NUMERIKUS FÜGGVÉNYEK

Egy numerikus függvény paramétere szám, és számot is ad vissza. A numerikus függvények az olvasó előtt nyilván ismertek. Ilyenek például: `sin()`, `cos()`, `tan()`, `exp()`, `ln()`, `sqrt()`, stb. Az alábbiakban néhányat ismertetünk ezek közül.

<code>ROUND({oszlop   kifejezés}, n)</code>	A kerekítendő értéket <i>n</i> tizedesjegyre kerekíti. Ha <i>n</i> értékét nem adjuk meg, akkor egész számra kerekít.
<code>TRUNC({oszlop   kifejezés}, n)</code>	A paraméterbeli értéket csonkolja <i>n</i> tizedesjegyre. Ha <i>n</i> értékét nem adjuk meg, akkor egész számra csonkol.
<code>MOD(m,n)</code>	Megadja az <i>m</i> <i>n</i> -nel való egészosztásának maradékát.
<code>POWER(alap, kitevő)</code>	Az <i>alapot</i> a <i>kitevőre</i> emeli.

### Példa

Nézzünk először egy numerikus példát.

Kerekítsük a 238.1249 számot 2 tizedesre. Csonkoljuk ugyanezt a számot. Döntsük el, hogy a 7 páros szám-e? Számítsuk ki a  $2^{10}$  értéket.

```
SQL> SELECT ROUND(238.1259,2) AS "Kerekítés",
2          TRUNC(238.1259,2) AS "Csonkolás",
3          MOD(7,2) AS "Maradékos osztás",
4          POWER(2,10) AS "Hatványozás"
5          FROM dual;

Kerekítés  Csonkolás  Maradékos osztás  Hatványozás
-----
238.13     238.12         1                1024
```

A bemutatott programhoz ismét a `sys` felhasználó tulajdonában levő, de mindenki számára elérhető `DUAL` segéd táblát használtuk fel. A `DUAL` segéd táblának egyetlen `DUMMY` (néma) nevű oszlopa van, amely egyetlen "X" karakterrel jelölt, szimbólikus értékkel rendelkező sort tartalmaz. E segéd tábla tehát lehetővé teszi, hogy az általunk kijelölt műveletek eredményét a művelet típusának megfelelő mezőben táblázatszerűen kiírathassuk.

Nézzük meg ezt a gyakorlatban.

<pre>SQL&gt; SELECT * 2      FROM DUAL;  D - X  SQL&gt; SELECT sysdate 2      FROM DUAL;  SYSDATE ----- 01-JÚL-21</pre>	<pre>SQL&gt; SELECT Initcap('varga') 2      FROM DUAL;  INITC ----- Varga  SQL&gt; SELECT SUBSTR('Tankönyv',4,5) 2      FROM DUAL;  SUBST ----- könyv</pre>
---	---

A `DUAL` segédtablát sok Oracle eszköz bemutatására fel tudjuk használni. A `DUAL` tábla akkor hasznos, ha valamilyen felhasználói táblában nem szereplő értéket, például egy konstans, egy kifejezést vagy függvényértéket csak egy alkalommal akarunk lekérdezni, például abból a célból, hogy egy `SELECT` utasításban helyes szintaktikával használjuk. A `DUAL` táblával történő kiíratás ugyanis visszaadja az adott konstans, kifejezés vagy függvény értékének a `SELECT` utasítás által elfogadott formátumát. Tipikus példa a dátum lekérdezése az aktuálisan érvényes dátumformátum megállapításához.

### DÁTUMOK HASZNÁLATA FÜGGVÉNYEK NÉLKÜL

A dátumot az Oracle belső numerikus formában tárolja: mégpedig évszázad, év, nap, óra, perc, másodperc alakban. Alapértelmezett megjelenítési és beviteli formája az Oracle magyar nyelvű verzióiban:

```
SQL> SELECT  sysdate
      2      FROM DUAL;
SYSDATE
-----
01-JÚL-21
```

Ez megfelel az `YY-MON-DD` alapértelmezett magyar nyelvű dátumformátumnak. Figyeljünk arra, hogy nem olyan régen léptük át a 2000. évet.

A dátumokkal aritmetikai műveleteket is végezhetünk. Ügyeljünk arra, hogy az adatbázisok többsége még az előző évezredben készült. Ha a dátumokkal aritmetikai műveleteket akarunk végezni, akkor `YYYY-MON-DD` formában (az év mindig négyjegyű legyen) végezzük el a számításokat.

Aritmetikai műveletek dátumokkal.

Művelet	Eredmény	Megjegyzés
dátum + szám	dátum	Bizonyos napokat ad hozzá a dátumhoz.
dátum – szám	dátum	Bizonyos napot levon a dátumból.
dátum – dátum	napok száma	Kivonja egyik dátumot a másiktól.
dátum + szám/24	dátum	Bizonyos számú órát ad hozzá a dátumhoz.

### Példa

Kíváncsiak vagyunk arra, hogy hány hónapja léptek be a vállalathoz a hivatalnokok (`clerk`).

```
SQL> SELECT  ename AS név,
      2      ROUND((sysdate-hiredate)/365*12)
      3      AS "Hónapok száma"
      4  FROM emp
      5  WHERE job= UPPER('clerk');
```

```
NÉV          Hónapok száma
-----
SMITH                247
ADAMS                170
JAMES                236
MILLER              234
```

## DÁTUMKEZELŐ FÜGGVÉNYEK

A dátumkezelő függvények `DATE` típusú értéket adnak vissza, az egy `MONTH_BETWEEN` függvény kivételével, amely numerikus értéket szolgáltat. A dátumkezelő függvények az Oracle dátumokon végeznek műveleteket.

Az alábbiakban a teljesség igénye nélkül ismertetünk néhány dátumkezelő függvényt.

<code>MONTHS_BETWEEN(dátum<sub>1</sub>, dátum<sub>2</sub>)</code>	Két <i>dátum</i> között eltelt hónapok száma.
<code>ADD_MONTHS(dátum, n)</code>	Hozzáad <i>n</i> darab naptári hónapot a dátumhoz. Az <i>n</i> értéke pozitív és negatív is lehet.
<code>NEXT_DAY(dátum, 'nap')</code>	Meghatározza a megadott dátum utáni első ' <i>nap</i> ' nevű nap dátumát.
<code>LAST_DAY(dátum)</code>	Kiszámolja a megadott dátum hónapjában az utolsó nap dátumát.
<code>ROUND(dátum, 'formátum')</code>	Dátum kerekítése, az igényelt formátumban.
<code>TRUNC(dátum, 'formátum')</code>	Dátum csonkolása, az igényelt formátumban.

### Példa

Számoljuk ki, hogy hány hónapja léptek be a vállalathoz a hivatalnokok (`clerk`).

```
SQL> SELECT ename AS név,
2          TRUNC(MONTHS_BETWEEN(sysdate,hiredate))
3          AS "Hónapok száma"
4  FROM emp
5  WHERE LOWER(job) ='clerk';
```

NÉV	Hónapok száma
SMITH	247
ADAMS	169
JAMES	235
MILLER	233

Az eredmény az előző számítással majdnem azonos. Az eltérés a `ROUND` illetve a `TRUNC` függvények használatából adódik.

### Példa

A mai nap

A hónap utolsó napja.

```
SQL> SELECT sysdate
2  FROM DUAL;
```

```
SQL> SELECT LAST_DAY(sysdate) AS utolsó
2  FROM DUAL;
```

SYSDATE
01-DEC-18

UTOLSÓ
01-DEC-31

Hányadika lesz a következő pénteken?

```
SQL> SELECT NEXT_DAY(sysdate,'péntek')
2          AS következő
3  FROM DUAL;
```

KÖVETKEZŐ

01-DEC-21

Milyen dátum lesz 15 hónap múlva?

```
SQL> SELECT ADD_MONTHS(sysdate,15)
2      FROM DUAL
```

ADD\_MONTH

03-MÁR-18

## KONVERZIÓS FÜGGVÉNYEK

Az Oracle adattípusokon kívül az Oracle8 adatbázistáblák új oszlopai ANSI, DB2, SQL/DS adattípusokkal is definiálhatók. Az Oracle ezeket az adattípusokat Oracle8 adattípusokká konvertálja.

Az Oracle képes az adatokat automatikusan a kívánt típusra konvertálni. Az adattípus-konverziót elvégezheti explicit módon a felhasználó, és implicit módon az Oracle.

### Implicit adattípus-konverziók :

Eredeti típus	Konvertált típus	
VARCHAR2 vagy CHAR	NUMBER	} Értékadásnál
VARCHAR2 vagy CHAR	DATE	
NUMBER	VARCHAR2	
DATE	VARCHAR2	
VARCHAR2 vagy CHAR	NUMBER	} Kifejezések kiértékelésekor
VARCHAR2 vagy CHAR	DATE	

### Explicit adattípus-konverzió (konvertáló függvények)

TO_CHAR( <i>érték</i> )	Az <i>érték</i> -ben megadott számot alakítja át karakterlánccá.
TO_NUMBER( <i>karakterlánc</i> )	A számot tartalmazó <i>karakterlánc</i> adatot számmá alakítja.
TO_DATE( <i>dátumérték</i> , <i>formátum</i> )	A NUMBER, CHAR vagy VARCHAR2 típusú <i>dátumérték</i> adatot átalakítja adott formátumú dátummá.

A konvertáló függvények teljes szintaktikai alakja:

```
TO_CHAR({szám | dátum}[,formátummaszk] [,nemzeti_nyelvi_param])
TO_NUMBER(karakter[,formátummaszk] [,nemzeti_nyelvi_param])
TO_DATE(karakter[,formátummaszk] [,nemzeti_nyelvi_param])
```

ahol

*formátummaszk* a kiírandó dátumformátumára jellemző  
*nemzeti\_nyelvi\_param* például 'nls\_date\_language=english'

A formátummaszk jellemzői:

- aposztrófok között kell állnia,
- bármely érvényes dátumformátum-elemet tartalmazhat,

- tartalmazhatja az úgynevezett *fm* elemet, amely eltávolítja a kitöltő szóközöket és bevezető nullákat.

#### A dátum formátummaszk elemei (nem teljes a felsorolás):

YYYY	teljes évszám
YEAR	teljes évszám betűkkel
MM	a hónap neve két számjeggyel
MONTH	a hónap teljes neve
MON	a hónap nevének három nagybetűs rövidítése
mon	a hónap nevének három kisbetűs rövidítése
WW	a hét sorszáma az évben
W	a hét sorszáma a hónapban
DDD	a nap sorszáma az évben
DD	a nap sorszáma a hónapban
D	a nap sorszáma a héten
DY	a hét napjának hárombetűs rövidítése
DAY	a nap teljes neve

#### Példa

Írassuk ki a mai dátumot, úgy hogy a dátum kiírásában az év négy számjeggyel, a hónap, és a nap neve nagybetűkkel legyen kiírva.

```
SQL> SELECT TO_CHAR(sysdate, 'YYYY MONTH DAY')
2      FROM DUAL;
```

```
TO_CHAR(SYSDATE, 'YYYYMONT
-----
2001 DECEMBER    KEDD
```

#### Példa

Írassuk ki a hivatalnok (CLERK) foglalkozású alkalmazottak nevét és belépési idejét a következő formátumnak megfelelően: teljes év, szünet, hónap kisbetűs teljes neve, szünet, a napok száma a vezető nullák nélkül (lásd `s028.sql`).

```
SQL> SELECT ename AS név,
2      TO_CHAR(hiredate, 'YYYY month fmDD')
3      AS "Belépési dátum"
4      FROM emp
5      WHERE job = 'CLERK';
```

NÉV	Belépési dátum
-----	-----
SMITH	1980 december 17
ADAMS	1987 május 23
JAMES	1981 december 3
MILLER	1982 január 23

-- vezető nullákat az *fm*-el letiltottuk

#### Példa

Listázzuk ki azokat az alkalmazottakat, akik 1981. június 01. után kerültek a vállalathoz.

```
SQL> SELECT ename AS név,
2      TO_CHAR(hiredate, 'YYYY mon fmDD') AS "Belépés"
3      FROM emp
```

```

4      WHERE hiredate > TO_DATE('1981.jún.1','YYYY.mon.DD');

NÉV          Belépés
-----
MARTIN       1981 sze 28
CLARK        1981 jún 9
SCOTT        1987 ápr 19
KING         1981 nov 17
...
9 sor kijelölve.

```

**Példa**

Számoljuk ki hány éve dolgoznak az alkalmazottak a cégnél. A cégnél töltött ledolgozott évek száma szerint jutalomban fognak részesülni (tehát szorozni kell majd az évek számával).

Próbálkozzunk a DUAL tábla segítségével.

```

SQL> SELECT TO_CHAR(sysdate,
2      'YYYY') AS ÉV
3      FROM dual;

ÉV
----
2001

```

Az eredmény karaktersorozat, hiszen balra igazít. Ha a fejléc hosszabb a kataktersorozat hosszánál, akkor csonkol. Ezzel nem tudunk még számolni.

Karaktersorozat átalakítása számmá:

```

SQL> SELECT TO_NUMBER
2      (TO_CHAR(sysdate, 'YYYY'))
3      AS "évszám"
4      FROM dual;

évszám
-----
2001

```

Ez már szám, hiszen az eredményt jobbra igazította.

Akkor tehát hány éve dolgoznak az alkalmazottak a cégnél?

```

SQL> SELECT ename AS név,
2      TO_NUMBER(TO_CHAR(sysdate, 'YYYY'))
3      - TO_NUMBER(TO_CHAR(hiredate, 'YYYY'))      -- kivonás
4      AS "évek száma"
5      FROM emp;

NÉV          évek száma
-----
SMITH                21
ALLEN                20
WARD                20
...
14 sor kijelölve.

```

## NVL FÜGGVÉNY

Az NVL függvény a NULL értéket tényleges értéké alakítja át. A függvény visszatérési értéke a baloldali paraméterének aktuális értéke (*operandus*), ha az nem NULL, egyébként a visszaadott érték a *helyettesítő* paraméter értéke lesz.

Az NVL függvény alakja:

*NVL(operandus, helyettesítő)*

Az *operandus* egy oszlopnév, a *helyettesítő* lehet literál, oszlopnév vagy kifejezés. A *helyettesítő* adattípusának meg kell egyeznie az *operandus* adattípusával. Az NVL függvény dátum, karakteres és numerikus adatoknál használható.

Az NVL függvény azért szükséges, mert ha a kifejezésben szerepel a NULL értéket tartalmazó oszlop, akkor a kifejezést – mint már láttuk is (havi összes jövedelem = havi bér + jutalék) – nem lehet kiértékelni (illetve az értéke NULL lesz). Az NVL függvény használatával az eredmény már értékelhető lesz. A megoldást az `s029.sql` script program tartalmazza. E példa „NULL értékek az aritmetikai kifejezésekben” címszó alatt is megtalálható.

### Példa

- NVL (fizetés, 0)
- NVL (belépés, sysdate)
- NVL (*munkakör*, 'még nincs')

## Csoportfüggvények

A függvények másik nagy csoportja a csoportfüggvény. A csoportfüggvények a táblázat több sorának azonos oszlopán végeznek műveleteket, és egyetlen értéket adnak vissza. A csoport lehet az egész tábla, illetve a tábla sorainak valamely részhalmaza. Ezt a részhalmazt a csoportosítással (GROUP BY) lehet előállítani. Először nézzük meg, a teljesség igénye nélkül, az SQL csoportfüggvényeit, s ezeknek szintaktikai leírását.

AVG ([DISTINCT   ALL] <i>oszlopnév</i> )	Az <i>oszlopnév</i> nevű oszlop adatainak átlagértéke (a NULL értékeket figyelmen kívül hagyja).
COUNT ( { *   [DISTINCT   ALL] <i>kif</i> } )	Megszámlálja a sorokat, ahol a <i>kif</i> nevű oszlopkifejezés értéke nem NULL. A * paraméterrel az összes sort megszámolja ( a többször szereplő és a NULL értéket tartalmazó sorokat is).
MAX ([DISTINCT   ALL] <i>kif</i> )	A <i>kif</i> nevű numerikus oszlopkifejezés maximális értéke (a NULL értéket figyelmen kívül hagyja).
MIN ([DISTINCT   ALL] <i>kif</i> )	A <i>kif</i> nevű numerikus oszlopkifejezés minimális értéke (a NULL értéket figyelmen kívül hagyja).
STDDEV ([DISTINCT   ALL] <i>kif</i> )	A <i>kif</i> nevű numerikus oszlopkifejezés szórásnégyzetének értéke (a NULL értéket figyelmen kívül hagyja).
SUM ([DISTINCT   ALL] <i>oszlopnév</i> )	Az <i>oszlopnév</i> nevű oszlop adatainak összege (a NULL értékeket figyelmen kívül hagyja).
VARIANCE ([DISTINCT   ALL] <i>kif</i> )	A <i>kif</i> nevű numerikus oszlopkifejezés szórásának értéke (a NULL értéket figyelmen kívül hagyja).

**Megjegyzés**

- Az ALL alapértelmezett paraméter (ezért van aláhúzva), tehát nem szükséges kiírnunk.
- Az átlag (AVG), az összeg (SUM), a szórásnégyzet (STDDEV), a szórás (VARIANCE) függvények csak numerikus adatokkal használhatók, a többi függvénynek bármilyen adattípusú lehet a paramétere.
- Az előzőkben szereplő "oszlopkifejezés", vagy "oszlopnév" megjelölés a később bevezetésre kerülő GROUP BY csoportképző utasítás rész alkalmazása esetén természetesen nem a teljes oszlopot, hanem annak a csoportképzésben kijelölt részhalmazait jelenti csupán.

**Példa**

Számoljuk ki az emp alaptáblából az alkalmazottak átlagfizetését, összes bértömegét, a maximális és a minimális havi bért. (Lásd s030.sql.)

```
SQL> SELECT TRUNC(AVG(sal),1) AS "Átlag fizetés",
2          SUM(sal) AS "Bértömeg",
3          MAX(sal) AS "Legnagyobb fizetés",
4          MIN(sal) AS "Legkisebb fizetés"
5          FROM emp;
```

Átlag fizetés	Bértömeg	Legnagyobb fizetés	Legkisebb fizetés
2073.2	29025	5000	800

**Példa**

Keressük meg ki lépett be legkorábban, és ki a legkésőbb a céghez.

Megjegyezzük, hogy a MAX, illetve MIN paramétere ekkor nem numerikus adat, hanem dátum. (Lásd s030.sql.)

```
SQL> SELECT MIN(hiredate) AS "Legrégebbi",      Legrégebb LegÚjabb
2          MAX(hiredate) AS "LegÚjabb"          -----
3          FROM emp;                             80-DEC-17 87-MÁJ-23
```

**Példa**

Hány sorból áll az emp tábla, és a comm oszlop? (Lásd s030.sql.)

SQL> SELECT COUNT(*) FROM emp;	SQL> SELECT COUNT(comm) FROM emp;
COUNT(*)	COUNT(COMM)
-----	-----
14	4

**Példa**

Az emp tábla comm oszlopában van NULL érték. Számoljuk ki a kiadott jutalék átlagát.

(Lásd s030.sql.)

```
SQL> SELECT AVG(comm)          átlag jutalék
2          AS "átlag jutalék"   -----
3          FROM emp;           550
```

A számolás helyes. Itt COUNT(comm) értékkel oszt az átlagszámítás során.

**Példa**

Nézzük meg a pótlékok átlagát az összes alkalmazottra vetítve, két tizedes jegyre kerekítve. (Lásd s030.sql.) Az átlagszámítás során itt viszont 14-el oszt az NVL függvény miatt.

```
SQL> SELECT ROUND(AVG(NVL(comm,0)),2)          Vetített átlag
      2          AS "Vetített átlag"          -----
      3      FROM emp;                      157.14
```

**További műveletvégzés NULL értékkel**

Miután megismertük az NVL és a csoportfüggvényeket, térjünk ki a NULL értékkel végezhető műveletekre. Az SQL a NULL értékű kifejezéseket háromféleképpen dolgozza fel:

1. Aritmetikai műveletek esetén nem végzi el a kijelölt műveleteket a tábla NULL értéket tartalmazó soraiban.
2. Ha a NULL értéket NVL függvénnyel helyettesítjük, akkor az NVL függvény helyettesítő értékével már elvégzi a kijelölt műveleteket.
3. Csoportfüggvények figyelmen kívül hagyják az oszlopok azon értékeit, amelyek NULL értékűek.

**Példa**

Tekintsünk az előző három esetre egy-egy példát.

1. Havi jövedelem számítása =  
fizetés + jutalék

```
SQL> SELECT ename AS év,
      2          (sal + comm) AS
      3          jövedelem
      4      FROM emp;
```

ÉV	JÖVEDELEM
-----	-----
SMITH	
ALLEN	1900
WARD	1750
JONES	
MARTIN	2650
BLAKE	
CLARK	
SCOTT	
KING	
TURNER	1500
ADAMS	
JAMES	
FORD	
MILLER	

2. Havi jövedelem számítása =  
fizetés + jutalék

```
SQL> SELECT ename AS év,
      2          (sal + NVL(comm,0)) AS
      3          jövedelem
      4      FROM emp;
```

ÉV	JÖVEDELEM
-----	-----
SMITH	800
ALLEN	1900
WARD	1750
JONES	2975
MARTIN	2650
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

3. Jutalék átlagszámítása (a NULL értékek figyelmen kívül hagyásával).

```
SQL> SELECT AVG(comm) AS "átlag jutalék"          átlag jutalék
      2      FROM emp;                          -----
                                           550
```

## A GROUP BY és a HAVING utasításrész

A csoportfüggvényekkel néhány, a teljes táblára vonatkozó számítást tud az SQL elvégezni. Az adatbázis-kezelésekor viszont általában nem az egész táblára vonatkozó csoportműveletekre kíváncsi a felhasználó, hanem a paramétertáblák bizonyos kiválasztott, azonos tulajdonságú oszlopok csoportjain elvégzett műveletek eredményére. A sorok valamilyen szempont szerinti csoportosítását adja meg a GROUP BY utasításrész. A feldolgozott csoportok közül azok kerülnek be a SELECT eredménytáblájába, amelyek a HAVING utasításrészben megadott feltételeknek eleget tesznek. (Az opcionális HAVING paraméter hiányában persze minden csoportot kiválaszt.) Az utasításrészek sorrendje tehát kötött. A hozzájuk tartozó tevékenységek sorrendje a következő:

- résztábla képzés,
- csoportok létrehozása,
- a csoportfüggvény kiértékelése,
- a feltételnek megfelelő csoport-reprezentáns sorok kiválasztása,
- a listázandó csoport-reprezentáns sorok rendezése

A GROUP BY és a HAVING utasításrésszel bővített SELECT utasítás alakja:

```
SELECT kifejezés [, kifejezés] ...
FROM tábla
[WHERE feltétel]
[GROUP BY kifejezés [, kifejezés] ...]
[HAVING csoportfeltétel]
[ORDER BY kifejezés]
```

A SELECT után megadott kifejezések a következők lehetnek:

- konstans,
- paraméter nélküli függvény,
- oszlop, másodlagos oszlopnév,
- valamilyen csoportfüggvény, amely azonos a GROUP BY utasításrészben megadott kifejezések valamelyikével.

### Megjegyzés

- A GROUP BY és a HAVING utasításrészben *nem szerepelhet* másodlagos oszlopnév.
- A GROUP BY utasításrészben szereplő oszlopokat nem kötelező szerepeltetni a SELECT listában, de a paramétertáblában szerepelniük kell.
- A HAVING utasításrész határozza meg azt, hogy a GROUP BY csoportjai közül milyen feltételnek eleget tevő csoportok eredményeit szeretnénk megjelentetni az eredménylistában.
- Az ORDER BY tartalmazhat csoportfüggvényeket és oszlopokat a GROUP BY utasításrészből, illetve ezek kombinációit.
- A GROUP BY utasításrész használatakor, az Oracle rendszer implicit módon növekvő sorrendbe teszi az eredményhalmazt, a paraméterében felsorolt kifejezések sorrendjében. (Emlékeztetünk arra, hogy az alapértelmezett növekvő rendezés megváltoztatásához a DESC rendezési opciót használhatjuk.)

Nagyon gyakori hiba kezdő SQL programozóknál, hogy a SELECT utasítás után felsorolt oszlopok nem a csoportosítandó oszlopra, vagy csoportfüggvényre vonatkoznak. Ekkor hibáüzenetek kapunk (amint a következő példában az látható is).

**Példa**

Listázzuk ki foglalkozás szerinti csoportosításban a dolgozókat.

```
SQL> SELECT job, ename
      2      FROM emp
      3      GROUP BY job;

SELECT job, ename
      *
```

Hiba a(z) 1. sorban:  
ORA-00979: nem GROUP BY kifejezés

A megoldás tehát helytelen!

Olyan oszlopot ugyanis nem lehet feltüntetni a `SELECT` listában, amire a `GROUP BY`, tehát a csoportosítás nem vonatkozik.

Nézzük az előbbi feladat helyes megoldását hagyományos SQL utasításrészek felhasználásával. (Figyeljünk fel arra, hogy pusztán az adatok csoportosításához nem volt szükség a `GROUP BY` utasításra.)

```
SQL> SELECT job, ename
      2      from emp
      3      ORDER BY job;
```

JOB	ENAME
ANALYST	FORD
ANALYST	SCOTT
CLERK	ADAMS
CLERK	JAMES
CLERK	MILLER
CLERK	SMITH
MANAGER	BLAKE
MANAGER	CLARK
MANAGER	JONES
PRESIDENT	KING
SALESMAN	ALLEN
SALESMAN	MARTIN
SALESMAN	TURNER
SALESMAN	WARD

14 sor kijelölve.

Egy másik megoldás rendezéssel és a kimeneti lista szűrésével (SQL\*Plus utasításokkal jobban áttekinthető listát kaptunk):

```
SQL> BREAK ON job
```

```
SQL> SELECT job, ename
      2      from emp
      3      ORDER BY job;
```

JOB	ENAME
ANALYST	FORD
	SCOTT
CLERK	ADAMS
	JAMES
	MILLER
	SMITH
MANAGER	BLAKE
	CLARK
	JONES
PRESIDENT	KING
SALESMAN	ALLEN
	MARTIN
	TURNER
	WARD

14 sor kijelölve.

```
SQL> CLEAR BREAKS
```

A `BREAK` egy SQL\*Plus utasítás. A `BREAK ON` hatására, amikor az SQL\*Plus érzékel egy adott változást az utasításban megadott kifejezés (`job`) értékében, akkor végrehajtja a kijelölt akciót, azaz megjeleníti a kimeneten a különböző `job` oszlopértékeket. A `CLEAR BREAKS` eltávolítja a meglévő beállításokat. A megjelenítést formázó SQL\*Plus utasításokról a 9. fejezetben lesz még szó.

**Példa**

Mennyi az egyes foglalkozási csoportban dolgozók létszáma?

```
SQL> SELECT job AS foglalkozás, COUNT(ename) AS létszám
      2 FROM emp
      3 GROUP BY job;
```

FOGLALKOZ	LÉTSZÁM
ANALYST	2
CLERK	4
MANAGER	3
PRESIDENT	1
SALESMAN	4

**Példa**

Listázzuk ki az emp tábla foglalkozás (job) oszlopának értékeit.

Az egyik megoldás szerint a csoportosítás szempontja a job, és ez egyúttal a SELECT lista része is.

```
SQL> SELECT job
      2 FROM emp
      3 GROUP BY job;
```

JOB
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN

A másik megoldásban a DISTINCT paramétert használjuk fel az ismétlődő foglalkozási nevek kiszűrésére.

```
SQL> SELECT DISTINCT job
      2 FROM emp;
```

JOB
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN

Figyeljük meg a listázás eredménye mindkét esetben azonos és mindkét esetben rendezett is, holott rendezést nem írtunk elő. Ennek az az oka, hogy egy úgynevezett implicit (növekvő) rendezést mind a GROUP BY, mind a DISTINCT végez. Ha azonban például fordított sorrendű (tehát csökkenő) rendezésben szeretnénk listázni, akkor mindkét esetben alkalmazni kell az utasítás végén az ORDER BY job utasításrészt.

```
SQL> SELECT job
      2 FROM emp
      3 GROUP BY job
      4 ORDER BY job DESC;
```

JOB
SALESMAN
PRESIDENT
MANAGER
CLERK
ANALYST

```
SQL> SELECT DISTINCT job
      2 FROM emp
      3 ORDER BY job DESC;
```

JOB
SALESMAN
PRESIDENT
MANAGER
CLERK
ANALYST

**Példa**

Listázzuk ki foglalkozási csoportok szerint a bértömeget.

Megengedett a másodlagos oszlopnév használata is a SELECT listában.

```
SQL> SELECT job, SUM(sal)
2     FROM emp
3     GROUP BY job;
```

JOB	SUM(SAL)
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

```
SQL> SELECT job, SUM(sal) AS "Bértömeg"
2     FROM emp
3     GROUP BY job;
```

JOB	Bértömeg
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

### Példa

Listázzuk ki foglalkozásonként és azon belül pedig részlegenként csoportosítva a bérösszegeket.

Nézzük meg ezt a listát fordított csoportosításban.

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY job, deptno;
```

JOB	DEPTNO	SUM(SAL)
ANALYST	20	6000
CLERK	10	1300
CLERK	20	1900
CLERK	30	950
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850
PRESIDENT	10	5000
SALESMAN	30	5600

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY deptno, job;
```

JOB	DEPTNO	SUM(SAL)
CLERK	10	1300
MANAGER	10	2450
PRESIDENT	10	5000
ANALYST	20	6000
CLERK	20	1900
MANAGER	20	2975
CLERK	30	950
MANAGER	30	2850
SALESMAN	30	5600

Rendezzük ezt eredménytáblát job, majd azon belül deptno szerint a rendező utasításrész felhasználásával

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY deptno, job
4     ORDER BY job, deptno;
```

JOB	DEPTNO	SUM(SAL)
ANALYST	20	6000
CLERK	10	1300
CLERK	20	1900
CLERK	30	950
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850
PRESIDENT	10	5000
SALESMAN	30	5600

Megfigyelhetjük, hogy ez a rendezett eredmény egyezik az első csoportosítási szempont (job, deptno) eredménnyel.

### Példa

A GROUP BY utasításrészben feltüntetett oszlopoknak nem kötelező szerepelniük a SELECT listában. Az alábbiakban erre láthatunk példát.

```
SQL> SELECT job, deptno, SUM(sal)
      2 FROM emp
      3 GROUP BY job, deptno;
```

JOB	DEPTNO	SUM(SAL)
ANALYST	20	6000
CLERK	10	1300
CLERK	20	1900
CLERK	30	950
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850
PRESIDENT	10	5000
SALESMAN	30	5600

```
SQL> SELECT deptno, SUM(sal)
      2 FROM emp
      3 GROUP BY job, deptno;
```

DEPTNO	SUM(SAL)
20	6000
10	1300
20	1900
30	950
10	2450
20	2975
30	2850
10	5000
30	5600

```
SQL> SELECT job, deptno
      2 FROM emp
      3 GROUP BY job, deptno;
```

JOB	DEPTNO
ANALYST	20
CLERK	10
CLERK	20
CLERK	30
MANAGER	10
MANAGER	20
MANAGER	30
PRESIDENT	10
SALESMAN	30

```
SQL> SELECT SUM(sal)
      2 FROM emp
      3 GROUP BY job, deptno;
```

SUM(SAL)
6000
1300
1900
950
2450
2975
2850
5000
5600

### Példa

Rendezzük az eredménylistát fizetés szerint. A rendezés szempontja a csoportosítás eredménye, amelyik ezúttal egy függvény.

Végezzük el az eredménylista rendezését a foglalkozás, ezen belül pedig fizetés szerint.

```
SQL> SELECT job, deptno, SUM(sal)
      2 FROM emp
      3 GROUP BY job, deptno
      4 ORDER BY SUM(sal);
```

JOB	DEPTNO	SUM(SAL)
CLERK	30	950
CLERK	10	1300
CLERK	20	1900

```
SQL> SELECT job, deptno, SUM(sal)
      2 FROM emp
      3 GROUP BY job, deptno
      4 ORDER BY job, SUM(sal);
```

JOB	DEPTNO	SUM(SAL)
ANALYST	20	6000
CLERK	30	950
CLERK	10	1300

MANAGER	10	2450	CLERK	20	1900
MANAGER	30	2850	MANAGER	10	2450
MANAGER	20	2975	MANAGER	30	2850
PRESIDENT	10	5000	MANAGER	20	2975
SALESMAN	30	5600	PRESIDENT	10	5000
ANALYST	20	6000	SALESMAN	30	5600

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY job, deptno
4     ORDER BY job DESC,
5             SUM(sal) DESC;
```

JOB	DEPTNO	SUM(SAL)
-----	-----	-----
SALESMAN	30	5600
PRESIDENT	10	5000
MANAGER	20	2975
MANAGER	30	2850
MANAGER	10	2450
CLERK	20	1900
CLERK	10	1300
CLERK	30	950
ANALYST	20	6000

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY job, deptno
4     ORDER BY job DESC,deptno,
5             SUM(sal) DESC;
```

JOB	DEPTNO	SUM(SAL)
-----	-----	-----
SALESMAN	30	5600
PRESIDENT	10	5000
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850
CLERK	10	1300
CLERK	20	1900
CLERK	30	950
ANALYST	20	6000

## HAVING utasításrész

A HAVING utasításrész a GROUP BY csoportosító utasításrésszel együtt használható. Használatának eredményeképpen a csoportokból a megadott feltételeknek megfelelő csoportokat, illetve azok reprezentáns sorait helyezi az eredménytáblába.

A HAVING utasításrésszel kiegészített SELECT utasítás tevékenységeinek sorrendje:

- a megfelelő sorok kiválasztása (WHERE),
- a sorok csoportosítása (GROUP BY),
- a kijelölt csoportműveletek elvégzése (például AVG, MAX, MIN, SUM )
- a HAVING mellett szereplő feltételnek megfelelő kiválogatás elvégzése.

### Példa

Csak azokat a csoportokat listázzuk ki, ahol a csoport bérösszege nagyobb 2000-nél.

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY job,deptno
4     HAVING SUM(sal) > 2000;
```

JOB	DEPTNO	SUM(SAL)
-----	-----	-----
ANALYST	20	6000
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850
PRESIDENT	10	5000
SALESMAN	30	5600

Listázzuk ki azokat a foglalkozási csoportokat, ezen belül részlegenként csoportosítva, ahol az összes bér nagyobb 2000-nél, és a foglakozás `manager`.

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     GROUP BY job,deptno
4     HAVING SUM(sal) > 2000 AND
5           job = UPPER('manager');
```

JOB	DEPTNO	SUM(SAL)
MANAGER	10	2450
MANAGER	20	2975
MANAGER	30	2850

Adjunk meg egy újabb feltételt, mégpedig csak azokat a sorokat csoportosítsa, ahol a részleg nem egyenlő 20-al. Végül rendezzük az eredménytábla sorait részleg szerint csökkenő sorrendben.

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     WHERE deptno <> 20
4     GROUP BY job,deptno
5     HAVING SUM(sal) > 2000 AND
6           job = UPPER('manager')
```

JOB	DEPTNO	SUM(SAL)
MANAGER	10	2450
MANAGER	30	2850

```
SQL> SELECT job, deptno, SUM(sal)
2     FROM emp
3     WHERE deptno <> 20
4     GROUP BY job,deptno
5     HAVING SUM(sal) > 2000 AND
6           job = UPPER('manager')
7     ORDER BY deptno DESC
```

JOB	DEPTNO	SUM(SAL)
MANAGER	30	2850
MANAGER	10	2450

A `WHERE` feltétellel egy előzetes részhalmazát alkothatjuk a paramétertáblák sorainak, amelyeket ezután lehet csoportosítani, és a csoportosítási feltételnek megfelelően szűrni, végül az utolsó `SELECT` részművelet mindig a rendezés.

## Halmazműveletek

Azokat a lekérdezéseket, amelyek két lekérdezés eredményét kombinálják, összetett lekérdezésnek nevezzük. Itt olyan összetett lekérdezéseket mutatunk be, amelyek két `SELECT` eredménytábláját, mint halmaz között végeznek különböző halmazműveletet. E halmazműveletek a következők lehetnek:

UNION	<i>Egyesítés, monoúnió.</i> Az összekapcsolt lekérdezések minden sorát ki-választja, de az eredményhalmazban az azonosak csak egyszer fordul-nak elő. (Monohalmazt ad vissza.)
UNION ALL	<i>Multiúnió.</i> Minden egyes lekérdezés sorát visszaadja. Ezért sortöbb-szörözéseket is tartalmazhat. (Multihalmazt ad vissza.)
INTERSECT	<i>Metszet.</i> A két összekapcsolt lekérdezés különböző sorait adja vissza. (Monohalmazt ad vissza.)

MINUS

*Kivonás.* Az első lekérdezés azon sorait adja vissza, amelyeket a második lekérdezés nem tartalmaz. (Monohalmazt ad vissza.)

A halmaz műveletek azonos precedenciájúak. Az SQL az azonos precedenciájú műveleteket balról jobbra hajtja végre. Ha ezen változtatni szeretnénk, akkor zárójeleznünk kell.

A két `SELECT` listában az oszlopoknak sorrendben, darabszámban és adattípusban meg kell egyezniük (a nevüknek nem).

Ha a lekérdezés eredménye karakter típusú, az összetett utasítás által visszaadott érték típusa a következő lehet:

- Ha mindkét `SELECT` adatai `CHAR` típusúak, akkor az eredmény is `CHAR` típusú lesz.
- Ha az egyik lekérdezés eredménye `VARCHAR2` típusú, akkor a halmaz művelettel összekapcsolt lekérdezés eredménye is `VARCHAR2` típusú lesz.

### Példa

Készítsünk két egyoszlopos táblázatot, amely egyjegyű számokat tartalmaz. A táblázat létrehozása az `shalmaz1.sql` fájlban található. Az `X` és `Y` tábla listázásának eredménye az alábbi:

```
CREATE TABLE X
(szamok NUMBER(2));
```

```
SQL> SELECT * FROM X;
```

```

      SZAMOK
-----
          1
          2
          3
          4
          5
          6
          5
          6
```

```
CREATE TABLE Y
(szamok NUMBER(2));
```

```
SQL> SELECT * FROM Y;
```

```

      SZAMOK
-----
          1
          5
          5
          6
          7
```

### Példa

Két lekérdezés egyesítése.

```
SQL> SELECT * FROM X
2      UNION
3      SELECT * FROM Y;
```

```

      SZAMOK
-----
          1
          2
          3
          4
          5
          6
          7
```

Két lekérdezés multiúniója.

```
SQL> SELECT * FROM X
2      UNION ALL
3      SELECT * FROM Y;
```

```

      SZAMOK
-----
          1
          2
          3
          4
          5
          6
          5
          6
          1
          5
          5
          6
          7
```

Képezzük két halmaz különbségét.

```
SQL> SELECT * FROM X
2     MINUS
3     SELECT * FROM Y;

      SZAMOK
-----
          2
          3
          4
```

Csak azokat a sorokat jeleníti meg, amelyek az első lekérdezés visszaadott, de a második nem.

Képezzük két halmaz metszetét.

```
SQL> SELECT * FROM X
2     INTERSECT
3     SELECT * FROM Y;

      SZAMOK
-----
          1
          5
          6
```

Azokat a sorokat kapjuk meg két lekérdezés metszete által, amelyek mindkét lekérdezésben szerepelnek.

### Példa

Második példaszorunk legyen már valamivel gyakorlatiasabb. Legyen két rendelés listánk, amelyeknek felépítése azonos, de a felvételük különböző időben történt. A listák létrehozása a `rendel_lista.sql` script programban található.

A táblák a következő rendelések listáját tartalmazzák:

```
SQL> SELECT * FROM rendel_lista1;
```

NEV	RSZAM	RIDO
Gyertya	1111	01-JAN-10
Pumpa	1122	01-FEB-15
Pumpa	1122	00-JAN-24
Kipufogo	2202	01-MÁR-12

```
SQL> SELECT * FROM rendel_lista2;
```

NEV	RSZAM	RIDO
Fotengely	3333	01-ÁPR-22
Kipufogo	2202	00-AUG-12
Kipufogo	2202	01-DEC-20

A két rendelés egyesítése:

```
SQL> SELECT * FROM rendel_lista1
2     UNION
3     SELECT * FROM rendel_lista2;
```

NEV	RSZAM	RIDO
Fotengely	3333	01-ÁPR-22
Gyertya	1111	01-JAN-10
Pumpa	1122	00-JAN-24
Pumpa	1122	01-FEB-15
Kipufogo	2202	00-AUG-12
Kipufogo	2202	01-MÁR-12
Kipufogo	2202	01-DEC-20

Egyesítsük a két tábla `nev` és `rszam` oszlopait, a dátumot most ne vegyük figyelembe.

#### Monohalmaz

```
SQL> SELECT nev, rszam
2     FROM rendel_lista1
3     UNION
4     SELECT nev, rszam
5     FROM rendel_lista2;
```

NÉV	RSZÁM
-----	-----
Főtengely	3333
Gyertya	1111
Pumpa	1122
Kipufogó	2202

#### Multihalmaz

```
SQL> SELECT nev, rszam
2     FROM rendel_lista1
3     UNION All
4     SELECT nev, rszam
5     FROM rendel_lista2;
```

NÉV	RSZÁM
-----	-----
Gyertya	1111
Pumpa	1122
Pumpa	1122
Kipufogó	2202
Főtengely	3333
Kipufogó	2202
Kipufogó	2202

Készítsük el a rendeléstáblák különbségét.

```
SQL> SELECT nev, rszam
2     FROM rendel_lista1
3     MINUS
4     SELECT nev, rszam
5     FROM rendel_lista2;
```

NEV	RSZAM
-----	-----
Gyertya	1111
Pumpa	1122

```
SQL> SELECT *
2     FROM rendel_lista1
3     MINUS
4     SELECT *
5     FROM rendel_lista2;
```

NEV	RSZAM	RIDO
-----	-----	-----
Gyertya	1111	01-JAN-10
Pumpa	1122	00-JAN-24
Pumpa	1122	01-FEB-15
Kipufogó	2202	01-MÁR-12

Készítsük el a két rendelés metszetét. Figyeljük meg a dátum hatását.

```
SQL> SELECT *
2     FROM rendel_lista1
3     INTERSECT
4     SELECT *
5     FROM rendel_lista2;
```

nincsenek kijelölve sorok

```
SQL> SELECT nev, rszam
2     FROM rendel_lista1
3     INTERSECT
4     SELECT nev, rszam
5     FROM rendel_lista2;
```

NEV	RSZAM
-----	-----
Kipufogó	2202

### Példa

Készítsük el az `emp` tábla egyesítését önmagával.

```
SQL> SELECT empno, ename, job, sal, comm, deptno
2     FROM emp
3     UNION
4     SELECT empno, ename, job, sal, comm, deptno
5     FROM emp;
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7369	SMITH	CLERK	800		20
7499	ALLEN	SALESMAN	1600	300	30
7521	WARD	SALESMAN	1250	500	30
7566	JONES	MANAGER	2975		20
7654	MARTIN	SALESMAN	1250	1400	30
7698	BLAKE	MANAGER	2850		30
7782	CLARK	MANAGER	2450		10
7788	SCOTT	ANALYST	3000		20
7839	KING	PRESIDENT	5000		10
7844	TURNER	SALESMAN	1500	0	30
7876	ADAMS	CLERK	1100		20
7900	JAMES	CLERK	950		30
7902	FORD	ANALYST	3000		20
7934	MILLER	CLERK	1300		10

Készítsük el az emp tábla egyesítését önmagával (Multiunió).

```
SQL> SELECT empno, ename, job, sal, comm, deptno
2      FROM emp
3      UNION ALL
4      SELECT empno, ename, job, sal, comm, deptno
5      FROM emp;
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7369	SMITH	CLERK	800		20
7499	ALLEN	SALESMAN	1600	300	30
7521	WARD	SALESMAN	1250	500	30
7566	JONES	MANAGER	2975		20
7654	MARTIN	SALESMAN	1250	1400	30

28 sor kijelölve.

### Feladat:

Az olvasó próbáljon előállítani egy hatalmas adathalmazt az emp tábla önmagával való egyesítésével, és mérje meg, melyik halmazművelet gyorsabb a UNION vagy a UNION ALL?

A megoldás az s\_Union1.sql, s\_Union3.sql, s\_UnionMulti1.sql, s\_UnionMulti2.sql, és s\_UnionMulti3.sql fájlokban látható. Érdekes eredményre jut.

## 9. FEJEZET

# Adatbázis-kezelés SQL nyelven

Egy adatbázis nem csak egy táblából állhat. A minta adatbázisunk is három táblából épül fel. Ezek az `emp`, a `dept` és a `salgrade` táblák a `scott` felhasználó tulajdonában vannak. Az adatbázis-kezelés gyakorlatában ritkán fordul elő, hogy egy eredménytábla adatai egyetlen bemenő táblából (paramétertáblából) álljanak elő. Az esetek többségében nem egy, hanem több paramétertáblánk van. A reláció algebrából tudjuk, hogy a táblákat többféle módon lehet összekapcsolni, a korábbi SQL tanulmányainkból pedig azt, hogy miként oldhatunk meg egy-egy lekérdezést SQL utasításokkal. Most azt mutatjuk meg, hogyan lehet több táblás lekérdezést megvalósítani az SQL-ben.

Figyeljük meg a `dept` táblát. Három oszlopa van: a `deptno` (azonos az `emp` tábla `deptno` oszlopával nemcsak neve és típusa, de jelentése szerint is), a `dname` (a részleg neve) és a `loc` (a telephely megnevezése).

Ezekután különböző kérdéseket tehetünk fel. Hol dolgozik például `KING`? Mennyi a `New York`-iak átlagkeresete? Hol keresik a legtöbbet, vagy éppen a legkevesebbet? Melyik városban a legnagyobb a munkaerő vándorlás? Mennyi az egyes városban dolgozók átlagéletkora? És így tovább...

## Táblák összekapcsolása

Ahhoz, hogy ezekre, vagy az ilyen jellegű kérdésekre választ tudjunk adni, a táblákat össze kell kapcsolni. A táblák összekapcsolását a kulcsok teszik lehetővé. Egy táblában az oszlopok egy halmazát akkor nevezzük kulcsnak, ha segítségével a tábla minden különböző sora egyértelműen azonosítható. (A kulcsokkal a II. rész 6. fejezetében foglalkoztunk részletesen.) Egy táblát egy másik táblával olyan módon tudunk logikailag összekapcsolni, hogy szerepeltetjük benne a másik tábla, úgynevezett elsődleges kulcsának oszlopait (melyeket itt idegen kulcsnak nevezünk).

Például a `dept` mintatáblában a `deptno` oszlop elsődleges kulcs, míg az `emp` tábla ugyanilyen nevű és típusú `deptno` oszlopa idegen kulcs. Az `emp` tábla elsődleges kulcsa az `empno`.

A II. részben ismertettük a táblák tipikus összekapcsolásait. Ezek a Descartes-szorzat, a természetes összekapcsolás (Natural Join), és az általános összekapcsolás (Théta összekapcsolás). Ezek (a természetes összekapcsolás kivételével) SQL utasítások segítségével megvalósíthatók. A következőkben azt mutatjuk meg, hogyan.

Két vagy több tábla összekapcsolásának lényege, hogy

- a `FROM` utasításrészben hivatkozzunk az összekapcsolandó táblákra,
- a `WHERE` utasításrészben megadjuk az összekapcsolási feltételt,
- a `SELECT` listában fel kell tüntetni, hogy a hivatkozott oszlop mely tábla oszlopa.

Az összekapcsolás formája (szintaktikája):

```
SELECT  tábla1.oszlop1 [,táblai.oszlopi] ...
        FROM tábla1 [, táblai] ...
        WHERE logikai kifejezés;
```

ahol

`táblai.oszlopi` az *i*-edik tábla listázandó oszlopa  
`logikai kifejezés` egy olyan logikai kifejezés, mely az összekapcsolandó táblák megfelelő oszlopait tartalmazza

Ha két vagy több táblában azonos nevű oszlopok vannak, akkor a `SELECT` kulcsszó után (a *szelekciós listában*) úgynevezett *minősített oszlopneveket* kell használni, vagyis az oszlop neve elé – ponttal elválasztva – be kell írni a tábla nevét is. Egy *minősített oszlopnév* csak az aktuális `SELECT` utasításban érvényes. Figyeljünk fel arra, hogy *n* tábla összekapcsolásához (*n* – 1) feltétel szükséges.

Az összekapcsolásnak két típusa van

- egyen-összekapcsolás
- nem egyen-összekapcsolás

Összekapcsolási módszerek lehetnek:

- belső összekapcsolás
- külső összekapcsolás
- tábla összekapcsolása önmagával

Amikor két vagy több tábla összekapcsolásánál *kimarad a feltétel* (lásd az alábbi példát), vagy *érvénytelen a megadott feltétel*, akkor Descartes-szorzat jön létre. A Descartes-szorzat a paraméter-táblák sorait megadott sorrendben, de az összes lehetséges módon párosítva (összefűzve) tartalmazza, így igen nagy méretű eredménytábla keletkezik.

## Egyszerű vagy egyen-összekapcsolás

### Példa

Listázzuk ki az alkalmazottak nevét valamint részlegük nevét.

```
SQL > SELECT emp.ename AS név, dept.dname AS "Részleg neve"
      2      FROM emp, dept;

NÉV      Részleg neve
-----
SMITH     ACCOUNTING
```

```

ALLEN      ACCOUNTING
WARD       ACCOUNTING
JONES      ACCOUNTING
MARTIN     ACCOUNTING
...
JAMES      OPERATIONS
FORD       OPERATIONS
MILLER     OPERATIONS

```

56 sor kijelölve.

Sem SMITH, sem ALLEN stb. nem ACCOUNTING. Ez azt jelenti, hogy az emp és dept tábla ename és dname oszlopán – mivel nincs összekapcsolási feltétel megadva – elvégezte a Descartes-szorzatot (56 sor). Így nem derül ki, hogy melyik dolgozó melyik részlegben (dname) dolgozik.

Tehát a 14 darab emp-táblabeli sorból és 4 darab dept-táblabeli sorból 56 sort állított elő. Ez egy kicsit sok, ráadásul nem is ad megoldást.

Próbáljuk meg kiszűrni az azonosakat.

A jó megoldás tehát a következő. (s040.sql)

```

SQL> SELECT DISTINCT
      2 emp.ename AS név,
      3 dept.dname AS "Részleg neve"
      4 FROM emp, dept;

```

NÉV	Részleg neve
ADAMS	ACCOUNTING
ADAMS	OPERATIONS
ADAMS	RESEARCH
ADAMS	SALES
ALLEN	ACCOUNTING
ALLEN	OPERATIONS
...	
TURNER	OPERATIONS
TURNER	RESEARCH
TURNER	SALES
WARD	ACCOUNTING
WARD	OPERATIONS
WARD	RESEARCH
WARD	SALES

56 sor kijelölve.

```

SQL> SELECT
      2 emp.ename AS név,
      3 dept.dname AS "Részleg neve"
      4 FROM emp, dept
      5 WHERE emp.deptno = dept.deptno;

```

NÉV	Részleg neve
SMITH	RESEARCH
ALLEN	SALES
WARD	SALES
JONES	RESEARCH
MARTIN	SALES
BLAKE	SALES
CLARK	ACCOUNTING
SCOTT	RESEARCH
KING	ACCOUNTING
TURNER	SALES
ADAMS	RESEARCH
JAMES	SALES
FORD	RESEARCH
MILLER	ACCOUNTING

14 sor kijelölve.

A DISTINCT paraméter nem talált azonosat. Megoldás *nem* jó. Mi is történt? *Kifejejtettük a feltételt*, ami összekapcsolja a két táblát. Mindkét táblának van deptno azonos típusú oszlopa, s ezek egyezősége adja a feltételt.

A megoldás jó. A két tábla azonos oszlopait általános összekapcsolással összekapcsoltuk.

Ez az "egyen-összekapcsolás". Az összekapcsolás az egyenlőség (=) operátorral történik, azaz összekapcsoló két oszlop értékei egyenlők. Ez egyben belső összekapcsolás is. Ha nem az egyenlőség az összehasonlítás operátora (hanem például a <= vagy a < operátorok), akkor "nem egyen-összekapcsolás"-ról beszélünk.

A `SELECT` utasításban egy vagy több tábla összekapcsolásakor több kiválasztási szempontot is megadhatunk az `AND` operátorral a `WHERE` utasításrészben.

**Példa**

Listázzuk ki azoknak az alkalmazottaknak az azonosító számát, a nevét, a részlegének nevét és telephelyét, akiknek a nevében "R" betű van. (s041.sql)

```
SQL> SELECT  emp.empno AS azonosító,
2           emp.ename AS név,
3           dept.dname AS "Részleg neve",
4           dept.loc AS telephely
5   FROM emp, dept
6  WHERE emp.deptno = dept.deptno
7         AND emp.ename LIKE ('%R%');
```

AZONOSÍTÓ	NÉV	Részleg neve	TELEPHELY
7521	WARD	SALES	CHICAGO
7654	MARTIN	SALES	CHICAGO
7782	CLARK	ACCOUNTING	NEW YORK
7844	TURNER	SALES	CHICAGO
7902	FORD	RESEARCH	DALLAS
7934	MILLER	ACCOUNTING	NEW YORK

**Másodlagos táblanevek használata**

A valódi adatbázis táblanevek általában elég hosszúak, mert utalnak a tábla konkrét tartalmára. Azért, hogy ezeket a hosszú neveket ne kelljen a `SELECT` utasításban ennyiszor leírni, másodlagos táblaneveket célszerű használni. A táblaneveket a `FROM` utasításrészénél kell megnevezni. Másodlagos táblanévvel a minősített oszlopnév használat lesz jóval egyszerűbb. Javasolható, hogy a tábla másodlagos neve legyen közérthető és rövid. (Legfeljebb 30 karakter hosszú lehet.) Ha másodlagos nevet használunk a paramétertábla neveknél, akkor az egész `SELECT` utasításban is azt kell használnunk. A másodlagos név csak az aktuális `SELECT` utasításban érvényes.

**Példa**

Listázzuk ki telephelyenként (városnév) az alkalmazottak átlagfizetését.

Nézzük meg ugyanezt másodlagos táblanevekkel. Legyen

az *emp* tábla másodlagos neve: *e*,  
 a *dept* tábla másodlagos neve: *d*.

```
SQL> SELECT dept.loc, ROUND(avg(sal))
2   FROM emp, dept
3  WHERE emp.deptno = dept.deptno
4  GROUP BY dept.loc;
```

```
SQL> SELECT d.loc, ROUND(avg(sal))
2   FROM emp e, dept d
3  WHERE e.deptno = d.deptno
4  GROUP BY d.loc;
```

LOC	ROUND (AVG (SAL) )
CHICAGO	1567
DALLAS	2175
NEW YORK	2917

LOC	ROUND (AVG (SAL) )
CHICAGO	1567
DALLAS	2175
NEW YORK	2917

Mint látjuk, a két `SELECT` utasítás működése azonos.

## Nem egyen-összekapcsolás

Figyeljük meg a harmadik táblánkat, a `salgrade` (fizetési besorolás) táblát.

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Az `emp` tábla `sal` (fizetés) oszlopával egyik oszlop sem egyezik meg. Viszont a fizetés alsó és felső határát adja a `losal` és `hisal`, amelyek között nincs átfedés.

### Példa

Listázzuk k, hogy az egyes alkalmazottak melyik bérkategóriába esnek. Legyen az `emp` paramétertábla másodlagos neve `e`, a `salgrade` tábla neve pedig `s`.

```
SQL> SELECT e.ename, e.sal, s.grade
2      FROM emp e, salgrade s
3      WHERE e.sal BETWEEN s.losal AND s.hisal;
```

ENAME	SAL	GRADE
JAMES	950	1
SMITH	800	1
ADAMS	1100	1
MARTIN	1250	2
WARD	1250	2
MILLER	1300	2
ALLEN	1600	3
TURNER	1500	3
BLAKE	2850	4
CLARK	2450	4
JONES	2975	4
FORD	3000	4
SCOTT	3000	4
KING	5000	5

## Belső összekapcsolás

Eddigiekben csak olyan táblaösszekapcsolást használtunk, amelyben a különböző táblák összekapcsolásban résztvevő oszlopainak a nem NULL értékei által jött létre kapcsolat. Ez esetenként adatvesztést eredményezett. Tekintsük a következő példát.

### Példa

Listázza ki a mintatábláinkhoz tartozó vállalat részlegeinek nevét és a részlegen dolgozók létszámát.

SQL> SELECT d.dname AS részlegnév,	RÉSZLEGNÉV	LÉTSZÁM	Sz
2 count(e.ename) AS létszám	-----	-----	
3 FROM emp e, dept d	ACCOUNTING	3	
4 WHERE e.deptno = d.deptno	RESEARCH	5	
5 GROUP BY d.dname;	SALES	6	

Az utasítás sikeresen ”lefutott”, az eredmény helyes, valóban ezeken az részlegeken ennyi ember dolgozik. De hol a 40-es számú, *OPERATIONS* részleg? (Hiszen ez is szerepel a *dept* táblában!) Az ugyan igaz, hogy ezen a részlegen jelenleg nincs egyetlen alkalmazott sem, ám, ha a nyilvántartásban nem szerepel, akkor nem is lesz (ugyanis a munkafeltevő nem is fogja tudni, hogy a vállalatnak van ilyen részlege). Az ilyen jellegű problémák megoldásában segít az úgynevezett külső összekapcsolás.

## Külső összekapcsolás

A külső összekapcsolás két tábla olyan összekapcsolása, amikor az összekapcsoló oszlop valamely értékének nincs megfelelője a másik táblában. Ekkor az eredménytáblában a hiányzó oszlopérték *NULL* érték lesz. Külső összekapcsolásra a külső összekapcsolási operátort használjuk, amely a plusz (+) jel.

### Példa

Tekintsük a fenti példát, de most a megoldásban használjunk külső összekapcsolást.

SQL> SELECT d.dname AS részlegnév,	RÉSZLEGNÉV	LÉTSZÁM
2 count(e.ename) AS létszám	-----	-----
3 FROM emp e, dept d	ACCOUNTING	3
4 WHERE e.deptno(+) = d.deptno	OPERATIONS	0
5 GROUP BY d.dname;	RESEARCH	5
	SALES	6

Lám, előkerült az *OPERATIONS* részleg! (Az *OPERATIONS* rekordban a *LÉTSZÁM* oszlopbeli érték azért 0, mivel az *emp* táblában ehhez a részleghez nem tartozik dolgozó, tehát a számuk nulla.)

A külső összekapcsolási operátor, a plussz jel a feltétel kifejezésének csak egyik oldalán állhat, mégpedig azon az oldalon, ahol hiányzik az oszlopérték. Az utasítás alakja:

```
SELECT tábla1.oszlop1 [, táblai.oszlopi] ...
FROM tábla1 [,táblai] ...
WHERE logikai kifejezés;
```

ahol *tábla<sub>1</sub>*, *tábla<sub>2</sub>* az összekapcsolandó táblák nevei,  
*tábla<sub>1</sub>.oszlop* a szelekciós lista minősített oszlopnevei,  
*logikai kifejezés* egy olyan logikai kifejezés, mely az összekapcsolandó táblák megfelelő oszlopaikat tartalmazza a relációs kifejezés egyik oldalán álló külső összekapcsolás operátorának jelével (+).

### Példa

Listázzuk ki az alkalmazottak nevét, részlegét, és a részleg nevét is. Legyen az *emp* tábla másodlagos neve *e*, a *dept* tábláé pedig *d*.

```
SQL> SELECT e.ename, d.deptno, d.dname
2     FROM emp e, dept d
3     WHERE e.deptno(+) = d.deptno;
ENAME          DEPTNO DNAME
```

```
-----
CLARK           10 ACCOUNTING
KING            10 ACCOUNTING
MILLER          10 ACCOUNTING
SMITH           20 RESEARCH
ADAMS           20 RESEARCH
FORD            20 RESEARCH
SCOTT           20 RESEARCH
JONES           20 RESEARCH
ALLEN           30 SALES
BLAKE           30 SALES
MARTIN          30 SALES
JAMES           30 SALES
TURNER          30 SALES
WARD            30 SALES
                40 OPERATIONS
```

A 40-es deptno számnak megfelelő részlegben senki sem dolgozik, így az ename oszlopba NULL érték került.

15 sor kijelölve.

## Tábla összekapcsolása önmagával

Az SQL módot ad arra, hogy egy táblát önmagával kapcsoljuk össze. Ez lehetőséget ad arra, hogy ugyanazt a táblát két különböző objektum reprezentációjának tekintsük, és ilyen módon különböző kiértékeléseket végezhesünk rajta. A táblát tehát úgy fogjuk fel, mintha két tábla lenne. A műveletvégzés belső reprezentációjában ennek megfelelően két memóriaterület tartozik a két "különböző" táblához, külön belső változók, stb. Ahhoz, hogy a SELECT utasításban ezt le tudjuk írni, nem elég, ha a FROM után kétszer írjuk fel a tábla nevét, másodlagos táblaneveket is kell használnunk.

### Példa

Listázzuk ki minden egyes alkalmazott főnökének a nevét. Az eredeti emp táblában csak a főnök azonosító száma van meg (mgr), de a főnök is alkalmazottja a cégnek, tehát azonosító szám szerint megtalálhatjuk a nevét. Az emp táblának két másodlagos neve legyen a dolgozo, és a főnök.

```
SQL> SELECT dolgozo.ename || ' főnöke ' || főnök.ename AS "dolgozó    főnök"
2     FROM emp főnök, emp dolgozo
3     WHERE főnök.empno = dolgozo.mgr;
```

```
dolgozó      főnök
-----
SMITH főnöke FORD
ALLEN főnöke BLAKE
WARD főnöke BLAKE
JONES főnöke KING
MARTIN főnöke BLAKE
BLAKE főnöke KING
```

Azért listáz 14 sor helyett 13-at, mert KING elnöknek nincs főnöke. Itt a NULL értéket figyelmen kívül hagyja.

CLARK főnöke KING  
SCOTT főnöke JONES  
TURNER főnöke BLAKE  
ADAMS főnöke SCOTT  
JAMES főnöke BLAKE  
FORD főnöke JONES  
MILLER főnöke CLARK

13 sor kijelölve.

Az összekapcsolás feltétele, hogy a főnök azonosító száma egyezzen meg a dolgozó főnökének számával.

## Allekérdezések

Az SQL nyelv lehetőséget ad arra, hogy bizonyos SQL utasításokon belül is használjunk `SELECT` utasításokat, amelyeket ilyen esetekben allekérdezésnek (subselect) nevezünk.

Mikor van erre szükség? Például egy `SELECT` utasításban akkor van ilyen megoldásra szükség, ha a feltételben (a `WHERE` utasításrészben) valamely adattábla/adattáblák kiértékeléséből származó adatra van szükségünk.

Az allekérdezés tehát egy olyan, kifejezésekben használható `SELECT` utasítás, amely értéke(ke)t ad át az őt használó külső utasításnak. Bizonyos esetekben az allekérdezés is átvehet adatot a külső (az allekérdezést használó) utasítás paramétertáblájának éppen feldolgozás alatt álló sorából.

Például, ha arra vagyunk kíváncsiak, hogy kinek a fizetése kisebb vagy nagyobb a 7566-os azonosítójú dolgozóénál. Egy egyszerű `SELECT...FROM...WHERE` utasításban, amikor valamely (a 7566-ostól eltérő azonosítójú) dolgozó adatait dolgozzuk fel, nem tudjuk, hogy mennyi a 7566-os azonosítójú dolgozó havi fizetése. (Ehhez valamilyen belső változóra volna szükségünk, amit viszont az SQL nem biztosít.) A feladat mégis megoldható SQL-ben, mégpedig az allekérdezés segítségével. Az alkalmazott módszer a következő:

- először: meg kell határozni a 7566-os azonosítójú dolgozó havi fizetését,
- másodszor: ki az, aki ennél többet keres.

A megoldás során két lekérdezést kell összekapcsolni. Összekapcsolás a `WHERE` feltételben történik. Az először végrehajtott `SELECT` utasítás az allekérdezés, az úgynevezett *belső* `SELECT`, ennek eredménye lesz a *külső* `SELECT` utasítás `WHERE` feltételében az összehasonlító operátor jobboldali értéke.

### Példa

Kinek nagyobb a fizetése, mint a 7566-os azonosítószámú alkalmazotté?

```
SQL> SELECT empno AS azonosító,
2      ename AS név,
3      sal AS fizetés
4      FROM emp
5      WHERE sal >
6          (SELECT sal
7            FROM emp
8            WHERE empno =7566);
```

} külső SELECT

} belső SELECT  
(mindig zárójelbe kell tenni.)

AZONOSÍTÓ	NÉV	FIZETÉS
7788	SCOTT	3000
7839	KING	5000
7902	FORD	3000

Az allekérdezéseknek akkor van nagy jelentőségük, ha a külső utasítás eredménytáblájában valamelyik sor kiválasztása egy teljes tábla kiértékelésétől függ.

Nézzük meg az allekérdezés felépítését és szabályait. Az allekérdezés általános alakja:

```
(SELECT belső szelekciós lista
  FROM tábla
  [WHERE kifejezés operátor kifejezés]
  [GROUP BY kifejezés]
  [HAVING csoportfeltétel])
```

Néhány szabály az allekérdezésekkel kapcsolatban:

- Először a belső SELECT utasítás hajtódik végre, ennek eredménye lesz a külső utasítás lekérdezésének feltétele.
- Allekérdezés a WHERE, a HAVING és a FROM utasításrészekben szerepelhet, ez utóbbi esetben INLINE nézetnek nevezzük.
- Az allekérdezést mindig zárójelbe kell tenni.
- Az allekérdezés mindig az összehasonlító operátor jobb oldalán legyen.
- Az allekérdezés nem tartalmazhat ORDER BY utasításrészt. (Az ORDER BY utasításrész egyszer szerepelhet, és ez mindig a külső SELECT utolsó utasításrésze.)

Az allekérdezések típusai:

- Egysoros
- Többsoros

Az allekérdezésekben szereplő operátorok is lehetnek egysorosak, illetve többsorosak.

Egysoros operátorok: >, =, >=, <, <=, <>.

Többsoros operátor: IN, ANY, ALL

## Egysoros allekérdezés

Az egysoros allekérdezés jellemzője, hogy a belső SELECT egyszer fut le, ezután kerül sor a külső utasítás végrehajtására. A belső SELECT *egyetlen* sort ad vissza végrehajtása után.

### Példa

Listázzuk ki a JAMES munkakörével egyező munkakörű alkalmazottakat. (s044.sql)

```
SQL> SELECT ename AS név, job AS munkakör
2   FROM emp
3   WHERE job =
4         (SELECT job
5           FROM emp
6           WHERE ename = UPPER('james'));
```

NÉV	MUNKAKÖR
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK

} külső (fő) SELECT.

} belső SELECT .  
Egyetlen értéket ad  
vissza: James  
munkakörét, ami clerk

**Példa**

Listázzuk ki azokat az alkalmazottakat, akiknek munkaköre JAMES munkakörével egyezik meg, és fizetése nagyobb vagy egyenlő ADAMS-éval.

```
SQL> SELECT ename AS név,                külső
      2      job AS munkakör,
      3      sal AS fizetés
      4  FROM emp
      5  WHERE job =
      6          (SELECT job                belső
      7              FROM emp
      8              WHERE ename = UPPER('james'))
      9      AND sal >=
     10          (SELECT sal                belső
     11              FROM emp
     12              WHERE ename = 'ADAMS');
```

NÉV	MUNKAKÖR	FIZETÉS
ADAMS	CLERK	1100
MILLER	CLERK	1300

A SELECT utasítás három un. lekérdezési blokkból (allekérdezésből) áll, két belső és egy külső blokkból. Mindkét belső blokk egy-egy sort (értéket) ad vissza

**Példa**

Listázzuk ki azokat az alkalmazottakat, akiknek fizetése a vállalatbeli minimális havi fizetésnek legalább a kétszerese.

```
SQL> SELECT ename AS név,                Külső
      2      job AS munkakör,
      3      sal AS fizetés
      4  FROM emp
      5  WHERE sal/2 >=
      6          (SELECT MIN(sal)
      7              FROM emp);
```

} Belső SELECT, melynek eredménye 800.

NÉV	MUNKAKÖR	FIZETÉS
ALLEN	SALESMAN	1600
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

```
SQL> SELECT MIN(sal)
      2  FROM emp;

      MIN(SAL)
      -----
              800
```

7 sor kijelölve.

Nézzünk egy példát a GROUP BY és a HAVING utasításrészek alkalmazására. Itt az allekérdezés a HAVING utasításrészben szereplő csoportfeltétel összehasonlító operátorának jobb oldalán áll.

**Példa**

Listázzuk ki azokat a csoportokat, amelyeknek átlagfizetése nagyobb a vállalat összes dolgozójának átlagfizetésénél.

Először határozzuk meg a vállalati átlagfizetést (ez lesz a belső SELECT):

```
SQL> SELECT  ROUND(AVG(sal))
2      FROM emp;

ROUND(AVG(SAL))
-----
2073
```

Ezután a feladat megoldása:

```
SQL> SELECT deptno, ROUND(AVG(sal))
2      FROM emp
3      GROUP BY deptno
4      HAVING AVG(sal) >=
5                  (SELECT AVG(sal)
6                  FROM emp);

DEPTNO  ROUND(AVG(SAL))
-----
10      2917
20      2175
```

**Többsoros allekérdezés****Példa**

Listázzuk ki azokat a munkaköröket, ahol az átlagfizetés minimuma meghaladja a 20-as részleg átlagbérét.

```
SQL> SELECT deptno, ROUND(AVG(sal))
2      FROM emp
3      GROUP BY deptno
4      HAVING AVG(sal) >=
5                  (SELECT AVG(sal)
6                  FROM emp
7                  GROUP BY deptno);
```

```
(SELECT AVG(sal)
*)
```

Hiba a(z) 5. sorban:  
ORA-01427: egy-soros kérdés egynél több  
sorral tér vissza

Az allekérdezés nem egysoros. HIBA!  
Megoldás: többsoros allekérdezés.

A helyes megoldáshoz más összehasonlító operátorok használatát kell igénybe venni. Ezek a következők:

- |        |  |
|--------|--|
| IN     | A lista bármely elemével egyenlő.  |
| ANY    | Egy listát ad vissza. Ez a lista lehet számok, stringek vagy az allekérdezés listája. Ha a listában van legalább egy olyan elem, amely kielégíti a feltételt (>, <, <=, >=, =, !=), akkor a WHERE utasításrész feltétele teljesül. |
| ALL    | Az ALL is egy listát ad vissza. A WHERE feltétel operátorai a (>, <, <=, >=, =, !=). A feltétel akkor teljesül, ha a visszaadott lista mindegyik elemének megfelel a vizsgált érték.   |
| EXISTS | Az EXISTS operátorral ellátott feltétel teljesül, ha az allekérdezés legalább egy sort ad vissza. Az allekérdezésben szereplő SELECT kulcsszó után oszlop, literál vagy csillag (*) is állhat.                                     |

**Példa**

Keressük azokat az alkalmazottakat, akiknek havi jövedelme kisebb, mint részlegének átlag-jövedelme.

Hogy megértsük az ANY és ALL működését, határozzuk meg először a részlegenkénti átlagfizetést.

```
SQL> SELECT ROUND(AVG(sal)), deptno
2      FROM emp
3      GROUP BY deptno;
```

ROUND(AVG(SAL))	DEPTNO
2917	10
2175	20
1567	30

**Megoldás az ANY operátorral**

```
SQL> SELECT ename, sal
2      FROM emp
3      WHERE sal <= ANY
4            (SELECT ROUND(AVG(sal))
5             FROM emp
6             GROUP BY deptno);
```

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
MARTIN	1250
BLAKE	2850
CLARK	2450
TURNER	1500
ADAMS	1100
JAMES	950
MILLER	1300

10 sor kijelölve.

Listázza, ha *valamelyiknél* kisebb.

**Megoldás az ALL operátorral**

```
SQL> SELECT ename, sal
2      FROM emp
3      WHERE sal <= ALL
4            (SELECT ROUND(AVG(sal))
5             FROM emp
6             GROUP BY deptno);
```

ENAME	SAL
SMITH	800
WARD	1250
MARTIN	1250
TURNER	1500
ADAMS	1100
JAMES	950
MILLER	1300

7 sor kijelölve.

Listázza, ha *mindegyiknél* kisebb.

**Példa (Az EXISTS operátor használata)**

Listázzuk ki azon részleg dolgozóinak nevét, munkakörét, fizetését, jutalékát és részlegének nevét, amelyiken van legalább egy olyan dolgozó, akinek van jutaléka. (s048.sql)

```
SQL> SELECT e.ename, e.job, e.sal, e.comm, d.dname
2      FROM emp e, dept d
3      WHERE EXISTS (SELECT *
4                    FROM emp ee
5                    WHERE ee.comm is not NULL
6                          AND e.deptno = ee.deptno)
7      AND e.deptno = d.deptno
8      ORDER BY d.deptno;
```

ENAME	JOB	SAL	COMM	DNAME
-----	-----	-----	-----	-----
ALLEN	SALESMAN	1600	300	SALES
WARD	SALESMAN	1250	500	SALES
MARTIN	SALESMAN	1250	1400	SALES
BLAKE	MANAGER	2850		SALES
TURNER	SALESMAN	1500	0	SALES
JAMES	CLERK	950		SALES

6 sor kijelölve.

Tehát van olyan dolgozó – mégpedig a `SALES` részlegen – akinek van jutaléka. Ezért a feladatkiírás értelmében kilistáztuk részlegének minden dolgozóját (természetesen azokat is, akiknek nincs jutaléka).

#### Példa (A `NOT EXISTS` operátor használata)

Listázzuk ki azon részleg dolgozóinak nevét, munkakörét, fizetését, jutalékát és részlegének nevét, amelyiken nincs olyan dolgozó, akinek van jutaléka. (s049.sql)

```
SQL> SELECT e.ename, e.job, e.sal, e.comm, d.dname
2     FROM emp e, dept d
3     WHERE NOT EXISTS (SELECT *
4                       FROM emp ee
5                       WHERE ee.comm is not NULL
6                             AND e.deptno = ee.deptno)
7     AND e.deptno = d.deptno
8     ORDER BY d.deptno;
```

ENAME	JOB	SAL	COMM	DNAME
-----	-----	-----	-----	-----
CLARK	MANAGER	2450		ACCOUNTING
KING	PRESIDENT	5000		ACCOUNTING
MILLER	CLERK	1300		ACCOUNTING
SMITH	CLERK	800		RESEARCH
ADAMS	CLERK	1100		RESEARCH
FORD	ANALYST	3000		RESEARCH
SCOTT	ANALYST	3000		RESEARCH
JONES	MANAGER	2975		RESEARCH

8 sor kijelölve.

## Allekérdezés a `FROM` utasításrészben

A `SELECT` utasítás `FROM` utasításrészében is használhatunk allekérdezéseket. (Az ilyen allekérdezéseket `INLINE` nézeteknek nevezzük.)

#### Példa

Listázzuk ki telephelyenként (városnév) az átlagfizetést.(s042.sql)

```
SQL> SELECT dept.loc, dept.deptno , ROUND(al.átlag,1)
2     FROM dept, (SELECT avg(sal) as átlag, deptno
3                 FROM emp
```

```

4          GROUP BY deptno) al
5 WHERE al.deptno = dept.deptno;

```

Az allekérdezés maga egy úgynevezett *nézettábla*, ezért el kell nevezni. Legyen a neve: al

```

LOC          DEPTNO  ROUND (Al.ÁTLAG,1)
-----
NEW YORK      10      2916.7
DALLAS        20      2175
CHICAGO       30      1566.7

```

(Ez a feladat megegyezik az egyen-összekapcsolásnál – másodlagos táblanevek használatánál – bemutatott példával.)

Megfigyelhetjük, ha a `FROM` utasításrészben allekérdezés van, akkor azt mindig el kell látni másodlagos táblanévvel. Ennek a nézettáblának (allekérdezésnek) oszloplistájára a külső `SELECT` utasításban a nézettábla minősítő nevével együtt hivatkozhatunk (al.átlag).

### Példa

Listázzuk ki azokat az alkalmazottakat, akiknek fizetése nagyobb részlegük átlagánál, és írjuk ki azt is, mennyivel tér el a fizetésük az átlagtól.(s046.sql)

```

SQL> SELECT e.ename AS név, e.sal AS fizetés,
2         al.átlag, e.sal - al.átlag AS eltérés
3 FROM emp e,
4         (SELECT ROUND(AVG(sal)) AS átlag,
5          deptno AS részleg
6          FROM emp
7          GROUP BY deptno) al
8 WHERE e.deptno= al.részleg
9        AND e.sal > al.átlag;

```

NÉV	FIZETÉS	ÁTLAG	ELTÉRÉS
KING	5000	2917	2083
FORD	3000	2175	825
SCOTT	3000	2175	825
JONES	2975	2175	800
ALLEN	1600	1567	33
BLAKE	2850	1567	1283

## Korrelált allekérdezés

*Korrelált* allekérdezésről akkor beszélünk, ha a belső `SELECT` kiértékelése függ a külső `SELECT` éppen feldolgozás alatt álló sorától, és fordítva, a külső `SELECT` aktuális sorának feldolgozását befolyásolja a belső `SELECT` kiértékelésének eredménye.

### Példa

Listázzuk ki rendezve azokat a foglalkozási csoportokat, amelyekbe egynél többen tartoznak, és azokat a dolgozókat, akik e kiválasztott csoportokba tartoznak.

```

SQL> SELECT e1.job AS foglalkozás,
2         e1.empno AS azonosító, e1.ename AS név

```

```
3      FROM emp e1
4      WHERE e1.job IN
5          (SELECT e2.job
6             FROM emp e2
7             WHERE e1.empno <> e2.empno)
-- A feltétel bal oldali tagja a külső paramétertábla oszlopértéke
-- Önmaga sorát kizárja a külső vizsgálatból
8      ORDER BY e1.job;
```

FOGLALKOZ	AZONOSÍTÓ	NÉV
-----	-----	-----
ANALYST	7788	SCOTT
ANALYST	7902	FORD
CLERK	7369	SMITH
CLERK	7876	ADAMS
CLERK	7934	MILLER
CLERK	7900	JAMES
MANAGER	7566	JONES
MANAGER	7782	CLARK
MANAGER	7698	BLAKE
SALESMAN	7499	ALLEN
SALESMAN	7654	MARTIN
SALESMAN	7844	TURNER
SALESMAN	7521	WARD

13 sor kijelölve.

## Adattáblák interaktív használata az SQL\*Plus környezetben

Az SQL\*Plus környezet lehetőséget ad arra, hogy az SQL utasításokat a felhasználó interaktívan használja.

Az SQL\*Plus környezettel már foglalkoztunk, bemutattuk a script programok használatát. Az alábbiakban ismertetjük az interaktivitást (a felhasználóval való párbeszédet) lehetővé tevő SQL\*Plus eszközt, a helyettesítő változók használatát. Segítségükkel ugyanaz a script program (SQL\*Plus program) a felhasználó választól függően más és más eredménytáblákat hoz létre. További SQL\*Plus utasítások segítségével a létrehozott eredménytáblákat fejlécekkel, dátummal stb. lehet megjeleníteni, azokat formázni lehet.

### Helyettesítő változók

A `SELECT` utasításokat a felhasználó igényének megfelelően kell megírunk. Ez azt jelenti, hogy általában nem tudunk minden adatot (értéket, oszlopnevet, stb.) előre megadni, hiszen éppen ezek függhetnek az aktuális felhasználótól. Ilyenkor helyettesítő változókat használunk.

#### & és && előtag

Használata: *&változónév*

Az SQL\*Plus környezetben a változónevek első karaktere az `&` előtag. Ha egy változó még nem volt definiálva, akkor ennek hatására a program futása felfüggesztődik, kérdést tesz fel a SQL\*Plus, a válasznak beírt karaktersorozat vagy szám bekerül a változóba, és a program ezután már a megadott értékkel fut tovább.

Egy `&` előtag esetén, amennyiben a változó nem definiált, akkor az SQL\*Plus *nem* definiálja a változót, és *nem őrzi* meg annak értékét, csak behelyettesíti az összes olyan helyre, ahol *&változónév* áll.

A dupla `&&` olyan előtag, amellyel ellátott változót, ha még nem definiált, az SQL\*Plus *definiálja és megőrzi* az értékét. Így ha a változót többször szeretnénk felhasználni a programunkban célszerű `&&` előtagot használni.

#### Példa

Listázzunk ki két, a felhasználó által megadott táblát.

Az `s059.sql` program:

```
SELECT * FROM &tabla;
SELECT * FROM &tabla;
```

Az `s059.sql` program futtatása:

```
SQL> @ s059
Adja meg a(z) tabla értékét: dept
régi    2:    from &tabla
```

```
új 2: from dept
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Adja meg a(z) tabla értékét: salgrade

```
régi 2: from &tabla
```

```
új 2: from salgrade
```

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

### Példa

Listázzuk ki a felhasználó által megadott részleg átlagfizetését. (s050.sql)

Első futtatás:

```
SQL> SELECT deptno AS részl,
2         ROUND(AVG(sal)) AS
3         "átlag fizetés"
4         FROM emp
5         GROUP BY deptno
6         HAVING deptno =&reszleg;
```

Adja meg a(z) részleg értékét: 10

```
régi 5: HAVING deptno =&reszleg
```

```
új 5: HAVING deptno =10
```

RÉSZL	átlag fizetés
10	2917

Második futtatás:

Adja meg a(z) részleg értékét: 30

```
régi 5: HAVING deptno =&reszleg
```

```
új 5: HAVING deptno =30
```

RÉSZL	átlag fizetés
30	1567

Harmadik futtatás:

Adja meg a(z) részleg értékét: 20

```
régi 5: HAVING deptno =&reszleg
```

```
új 5: HAVING deptno =20
```

RÉSZL	átlag fizetés
20	2175

Minden futtatáskor az SQL\*Plus megkéri a felhasználót, hogy adja meg a változó értékét. Kiírja ellenőrzésképp a régi értéket és az új értéket is. Ez szintén az SQL\*Plus környezeti alapértelmezett beállítása. Ki is lehet kapcsolni ezt az ellenőrzést (SET verify {ON|OFF}).

### Példa

Mekkora a megadott alkalmazott fizetése?

```
SQL> SELECT ename AS név,
2         empno AS azonosító,
3         sal AS fizetés
4         FROM emp
5         WHERE ename = UPPER('&vnev')
Adja meg a(z) vnev értékét: scott
régi 5: WHERE ename = UPPER('&vnev')
```

Kapcsoljuk ki az ellenőrzést.

```
SQL> SET verify OFF -- kikapcsol
```

Futtassuk le a pufferből még egyszer.

```
SQL> run
```

Adja meg a(z) vnev értékét: king

```

új 5: WHERE ename = UPPER('scott')
NÉV          AZONOSÍTÓ    FIZETÉS
-----
KING          7839        5000
SCOTT         7788        3000

```

**Példa**

Adjuk meg, hogy ki lépett be egy megadott dátum után a vállalathoz. A dátumot a következőképp szeretnénk bekérni: 1988.07.12 (s050.sql.)

Ebben az esetben a bekért változó *dátum*.

```

SQL> SELECT ename AS név,
2         hiredate as dátum
3       FROM emp
4       WHERE hiredate > TO_DATE('&dátum','YYYY.MM.DD');
Adja meg a(z) dátum értékét: 1981.11.01
régi 4: WHERE hiredate > TO_DATE('&dátum','YYYY.MM.DD')
új 4:  WHERE hiredate > TO_DATE('1981.11.01','YYYY.MM.DD')

```

```

NÉV          DÁTUM
-----
SCOTT        87-DEC-09
KING         81-NOV-17
ADAMS        87-MÁJ-23
JAMES        81-DEC-03
FORD         81-DEC-03
MILLER       82-JAN-23

```

**Példa**

Helyettesítő változó: *oszlopnév* és *feltétel*. (Az s050.sql tartalmazza a megoldást.)

```

SQL> SELECT ename AS név,
2         empno AS azonosító,
3         &oszlopnév
4       FROM emp
5       WHERE &feltétel
6       ORDER BY &rendezőoszlop;
Adja meg a(z) oszlopnév értékét: sal
Adja meg a(z) feltétel értékét: sal>=2500
Adja meg a(z) rendezőoszlop értékét: ename

```

```

NÉV          AZONOSÍTÓ    SAL
-----
BLAKE         7698        2850
FORD          7902        3000
JONES         7566        2975
KING          7839        5000
SCOTT         7788        3000

```

**Példa**

Helyettesítő változók: *oszlopnév*, *feltétel*, *táblanév*

```

SQL> SELECT deptno AS részleg,
2         &oszlop1, &oszlop2
3       FROM &tabla
4       WHERE &feltétel;
Futtassuk le újra a pufferből
SQL> run

```

Adja meg a(z) oszlop1 értékét: ename  
 Adja meg a(z) oszlop2 értékét: sal  
 Adja meg a(z) tabla értékét: emp  
 Adja meg a(z) feltétel értékét: sal  
 <1500

RÉSZLEG	ENAME	SAL
20	SMITH	800
30	WARD	1250
30	MARTIN	1250
20	ADAMS	1100
30	JAMES	950
10	MILLER	1300

Adja meg a(z) oszlop1 értékét: dname  
 Adja meg a(z) oszlop2 értékét: loc  
 Adja meg a(z) tabla értékét: dept  
 Adja meg a(z) feltétel értékét: dname  
 LIKE'%R%'

RÉSZLEG	DNAME	LOC
20	RESEARCH	DALLAS
40	OPERATIONS	BOSTON

A dept táblából azokat a részlegeket, részlegneveket és telephelyeket listáztuk ki, ahol a részleg nevében "R" betű előfordul.

Azoknak a név és fizetés oszlopait listáztuk ki, akiknek fizetése kisebb, mint 1500.

### Példa

(Két && előtag helyettesítő változóra.)

```
SQL> SELECT ename AS név, sal AS fizetés, &oszlopnev
2      FROM emp
3      ORDER BY &oszlopnev DESC;
Adja meg a(z) oszlopnev értékét: deptno
régi  1: SELECT ename AS név, sal AS fizetés, &oszlopnev
új    1: SELECT ename AS név, sal AS fizetés, deptno
régi  3: ORDER BY &oszlopnev DESC
új    3: ORDER BY deptno DESC
14 sor kijelölve.
```

NÉV	FIZETÉS	DEPTNO
ALLEN	1600	30
WARD	1250	30
...		
KING	5000	10
MILLER	1300	10

14 sor kijelölve.

Futtassuk le a megírt SELECT utasítást. Egyetlen kérdés tesz fel (első sor). A 3-dik sorban már nem tesz fel kérdést. Ezután akárhányszor lefuttathatjuk, nem tesz fel kérdést.

Próbáljuk többször lefuttatni. Mindig ugyanazt az eredménytáblát adja, mivel nem kér be új változót. Az && előtag definiálja is a helyettesítő változót, amelynek értéke mindaddig megmarad, míg ki nem töröljük, illetve be nem zárjuk az SQL\*Plus környezetet.

### Felhasználói változók definiálása

Egy SELECT utasítás végrehajtása előtt felhasználói változókat definiálhatunk a DEFINE és az ACCEPT SQL\*Plus utasításokkal.

## DEFINE UTASÍTÁS

Az utasítás alakja

DEFINE <i>változó</i> = <i>érték</i>	Létrehoz egy CHAR típusú felhasználói <i>változót</i> , és egy értéket rendel hozzá.
DEFINE <i>változó</i>	Megjeleníti a <i>változót</i> , az értékét és az adattípusát.
DEFINE	Megjeleníti az összes felhasználói <i>változót</i> , értékeivel és adattípusával együtt

```
SQL> DEFINE
DEFINE SQLPLUS RELEASE = "801070000" (CHAR)
...
DEFINE RC = "1" (CHAR)
DEFINE OSZLOPNÉV = "deptno" (CHAR)
```

Ha csak adott változóra vagyunk kíváncsiak:

```
SQL> DEFINE oszlopnev
DEFINE OSZLOPNEV = "deptno" (CHAR)
```

Definiáljunk egy új változót:

```
SQL> DEFINE datum = 01-JÚL-10
```

Nézzük meg, hogy a felhasználói változók listája tartalmazza-e:

```
SQL> DEFINE
...
DEFINE OSZLOPNEV = "deptno" (CHAR)
DEFINE DATUM = "01-JÚL-10" (CHAR)
```

## ACCEPT UTASÍTÁS

Az ACCEPT utasítás definiál egy változót, a felhasználóhoz üzenetet küld, és a felhasználó választát eltárolja az általa definiált változóban (a felhasználói választ el is tudja rejteni, így alkalmas jelszó bevitelére).

Az utasítás alakja:

```
ACCEPT változó [adattípus] [FORMAT formátum] [PROMPT üzenet] [HIDE]
```

ahol

<i>változó</i>	A változó neve. Ha még nem létezik ez a változó, létrehozza.
<i>adattípus</i>	NUMBER, CHAR, DATE (maximum 240 bájt).
FOR[ <i>MAT</i> ] <i>formátum</i>	Meghatározza a formátummodellt. Dátum formátumról már volt szó, számformátumokról majd lesz szó. (A15- 15 karakteres mezőszélesség, 99,999- 5 számjegyes kiírás, ezres határolóval)
PROMPT <i>üzenet</i>	A felhasználó adatmegadása előtt megjeleníti az üzenetet.
HIDE	Elrejt a felhasználó által megadott adatot (jelszó).

**Fontos**

Ismételten felhívjuk a figyelmet, hogy az SQL\*Plus utasításokat a *pufferből nem lehet lefuttatni*, mert az SQL nem támogatja azt. Ezért az SQL\*Plus utasításokat tartalmazó programok esetén vagy a megírt script programot futtatjuk le, vagy pedig külön futtatjuk az ACCEPT utasítást az SQL prompt (SQL>) után, és a pufferből csupán a SELECT utasítást. Ha az Olvasó figyel erre, sok problémától megkíméli önmagát.

### Példa

Tekintsük az alábbi programot:

```
ACCEPT reszlegnev PROMPT 'Kérem a részleg nevét: '
SELECT *
  FROM dept
 WHERE dname = UPPER('&reszlegnev');
```

A fenti program futtatása:

```
@ c:\a\s051.sql
Kérem a részleg nevét: research
régi   2:   WHERE dname = UPPER('&reszlegnev')
új     2:   WHERE dname = UPPER('research')

  DEPTNO DNAME          LOC
-----
      20 RESEARCH      DALLAS
```

Nézzük meg, hogy definiált-e a részlegnév?

```
SQL> DEFINE
...
DEFINE OSZLOPNEV      = "deptno" (CHAR)
DEFINE DATUM          = "01-JÚL-10" (CHAR)
DEFINE RESZLEGNEV     = "research" (CHAR)
```

Ha újra futtatjuk a programot, akkor az ACCEPT *felülírja* az előző, már definiált változó értéket. Az alábbi futtatásból ez kiderül. A reszlegnev megváltozott.

```
SQL> @c:\a\s051.sql
Kérem a részleg nevét: sales

  DEPTNO DNAME          LOC
-----
      30 SALES          CHICAGO

SQL> DEFINE
...
DEFINE _RC             = "1" (CHAR)
DEFINE OSZLOPNEV      = "deptno" (CHAR)
DEFINE DATUM          = "01-JÚL-10" (CHAR)
DEFINE RESZLEGNEV     = "sales" (CHAR)
```

### UNDEFINE UTASÍTÁS

Az ACCEPT, a DEFINE vagy az && előtaggal definiált SQL\*Plus felhasználói változók mindaddig megőrzik értéküket, míg egy UNDEFINE utasítással nem töröljük őket, vagy ki nem lépünk az SQL\*Plus környezetből.

### Töröljük a datum változót

```
SQL> UNDEFINE datum          -- törlés
SQL> DEFINE                   -- lekérdezés
...
DEFINE OSZLOPNEV              = "deptno" (CHAR)
DEFINE RESZLEGNEV             = "research" (CHAR)
SQL>
```

A datum változó nincs már a felhasználói listában.

## Az SQL\*Plus környezet beállítása

Az SQL\*Plusban igen sok paraméter áll rendelkezésre az eredménytáblák, listák megjelenítési formájának beállítására. Mint már említettük, a paramétereket és azok alapértelmezéseit lekérdezhethetjük a `SHOW ALL` utasítással.

```
SQL> SHOW ALL
appinfo értéke ON és "SQL*Plus"-ra van beállítva
arraysize 15
autocommit OFF
...
```

A alapértelmezett az első, illetve a konkrét érték.

Amennyiben a `SHOW` utasítás után magát a beállítandó paramétert adjuk meg, megtudjuk a paraméter aktuális beállítását.

```
SQL> SHOW verify
verify ON
```

A `verify` (ellenőrző) paraméter jelenleg bekapcsolt.

## SET utasítás

A rendszerváltozók beállítása a `SET` utasítással történik.

Az utasítás alakja.

`SET rendszerváltozó érték`

ahol

*rendszerváltozó* szabályozza az SQL\*Plus környezet egy - egy jellemzőjét,  
*érték* az adott rendszerváltozó lehetséges értéke.

A teljesség igénye nélkül ismertetünk néhány rendszerváltozót.

<code>ARRAY[SIZE]{20   <i>n</i>}</code>	Megadja az SQL*Plus által egyszerre betölthető sorok számát. Értéke 1 és 5000 között lehet.
<code>COLSEP{_<i> szöveg</i>}</code>	Beállítja, hogy az oszlopok között milyen karakter jelenjen meg.
<code>DEF[INE]{&amp;  <i>szimbólum</i>}</code>	Megadja a helyettesítő változó előtagját.

FEED[BACK] {6   <i>n</i>   OFF   ON}	Lista után kiírja "... sor kijelölve", de csak akkor, ha a kiírt sorok száma legalább <i>n</i> (alapértelmezés 6).
HEA[DING] {ON   OFF}	ON engedélyezi az oszlopfejlécek (oszlopnév, aláhúzás) kiírását. OFF ezt letiltja.
LIN[ESIZE] {80   <i>n</i> }	A kiírandó karakterek számát adja meg egy sorban. <i>n</i> maximális értéke 999 lehet.
NEW[PAGE] {1   <i>n</i> }	Megadja azon üres sorok számát, amelyet egyik lap befejezése és a másik lap kezdete között kihagy a rendszer. A 0 megadása lapdobást jelent.
NUM[FORMAT] {formátum}	A formátumnak megfelelően jelennek meg a számok. (formátumokat lásd a COLUMN utasításnál.)
NUM[WIDTH] {10   <i>n</i> }	A számok alapértelmezett mezőszélességét állítja be.
PAGES[IZE] {14   <i>n</i> }	Megadja egy lapra listázandó sorok számát.
SERVEROUT[PUT] {ON   OFF}	Lehetővé teszi, hogy megjelenítsük a PL/SQL eljárások változóinak kiírását a DBMS_OUTPUT csomag felhasználásával.
TERM[OUT] {ON   OFF}	Bekapcsolt állapotban a képernyőre írja a lekérdezések eredménylistáját.
VER[IFY] {ON   OFF}	Az SQL*Plus kiírja a régi és új változók értékét, mielőtt az adatbekérő SQL*Plus utasítás végrehajtódna.

A beállításokat az SQL\*Plus indításakor mindig meg kell ismételni. Ez elkerülhető, ha inicializáljuk az SQL\*Plus környezetet a III. rész elején leírtak alapján egy `login.sql` nevű script programmal (oly módon, hogy ebbe írjuk a környezeti beállításokat).

### Feladat

Próbáljunk egy olyan inicializáló `login.sql` script programot létrehozni, amely beállítja a sorhosszt és az oldalanként listázandó sorok számát (mentés a `SAVE` utasítással!).

## SQL\*Plus formázási utasítások

Az eredménylisták megjelenési formáját módosíthatjuk a formázási utasításokkal.

Ezek az utasítások a következők:

COLUMN [ <i>oszlopnév paraméter</i> ]	Oszlopformátumot szabályozza a paraméternek megfelelően.
TTI[TLE] [ <i>szöveg</i>   OFF   ON]	Minden oldal tetején megjelenő fejléc megadása.
BTI[TLE] [ <i>szöveg</i>   OFF   ON]	Minden oldal alján megjelentő lábléc megadása.
BRE[AK] [ON <i>kifejezés</i> ]	Kiszűri az ismétlődő értékek megjelenítését, és sortöréssel tagolja az adatsorokat. (ON hatására a megadott kifejezés értékének megváltoztatásakor hajtja végre a tevékenységet.)

### Fontos

Ezeket a beállításokat a munkamenet végén kapcsoljuk ki!

## COLUMN UTASÍTÁS

A `COLUMN` utasítás az oszlopok és az oszlopfejlécek megjelenési formáját szabályozza a paramétereknek megfelelően. A paraméterek megadhatók egymás után vesszővel elválasztva, vagy külön `COLUMN` utasításokkal is.

Alábbiakban bemutatunk néhány paramétert, amelyek segítségével szép listák készíthetők.

<code>CLEAR</code>	A megadott oszlopformátumok törlése.
<code>FORMAT <i>formátum</i></code>	Megadja az oszlop adatok megjelenítési formáját.
<code>HEADING <i>szöveg</i></code>	Egy oszlop fejlécének beállítására szolgál. Alapértelmezés az oszlopnév. Ha ezen változtatni akarunk, meg kell adni a <i>szöveget</i> . Ha a szövegben üres vagy speciális karakter van, aposztrófok közé kell tenni.
<code>JUSTIFY {LEFT CENTER RIGHT}</code>	Az <i>oszlopfejléc</i> igazítása. Alapértelmezés: számoknál jobbra igazítás, minden más típusnál balra igazítás.
<code>NOPRINT</code> és <code>PRINT</code>	Letiltja (elrejt), vagy engedélyezi az oszlop megjelenítését.

Ezek után már csak a formátummodell ismertetése van hátra, s azután formázhatjuk is a megjelenő tábláinkat!

A `COLUMN` formátummodellek (azonos a `SET NUMFORMAT` beállításnál alkalmazható formátumokkal) szabályozzák a kiírandó adatok megjelenítési formáját:

Formátum	Leírás
<code>A15</code>	15 karakter szélességben jelenik meg.
<code>99999990</code>	A nullák vagy kilenceselek száma határozza meg a megjeleníthető számjegyek számát.
<code>99999</code>	5 értékes számjegy, vezető nullák nem jelennek meg.
<code>09999</code>	5 értékes számjegy, a vezető nullákkal együtt.
<code>S9999</code>	Az S karakter helyén az előjel jelenik meg.
<code>\$9999</code>	Minden szám elé \$ jel kerül.
<code>L999</code>	Az L karakter helyén a helyi pénznem jelenik meg.
<code>.</code>	A tizedes pont jele.
<code>,</code>	Ezres elválasztó jel.

### Példa

Listázzuk ki azokat az alkalmazottakat, akiknek fizetése kisebb, mint 3000 \$. Az eredménylistának adjunk kétsoros középre igazított fejléct, és ez legyen Alkalmazottak bérkimutatása. Legyen középre igazított lábléce Béremelésre javasoltak szöveggel. A job oszlop fejléce legyen kétsoros, és minden kategóriát csak egyszer tüntessen fel. A fizetés oszlopban jelenjen meg a fizetés két tizedesjegy pontossággal, a pénznem és szerepeljen benne ezres elválasztójel. A script program tartalma (`s052.sql`):

```
SET PAGESIZE 22
SET LINESIZE 50
SET FEEDBACK OFF
TTITLE 'Alkalmazottak | bérkimutatása' -- A függőleges vonal törés
BTITLE 'Béremelésre javasoltak'
```

```

BREAK ON job                                -- szűri az ismétlődéseket
COLUMN job HEADING 'Foglalkozási|kategória' FORMAT A15
COLUMN ename HEADING 'Alkalazott neve' FORMAT A15
COLUMN sal HEADING 'Havi fizetés' FORMAT $99,999.99

SELECT job, ename, sal
FROM emp
WHERE sal < 3000
ORDER BY job, ename;

SET FEEDBACK ON

```

Futtassuk le a script programot:

```
SQL> @ c:\a\s052
```

A FUTÁS EREDMÉNYE:

Cs. Júl 26		lap	1
Alkalmazottak bérkimutatása			
Foglalkozási kategória	Alkalmazott neve	Havi fizetés	
-----	-----	-----	
CLERK	ADAMS	\$1,100.00	
	JAMES	\$950.00	
	MILLER	\$1,300.00	
	SMITH	\$800.00	
MANAGER	BLAKE	\$2,850.00	
	CLARK	\$2,450.00	
	JONES	\$2,975.00	
SALESMAN	ALLEN	\$1,600.00	
	MARTIN	\$1,250.00	
	TURNER	\$1,500.00	
	WARD	\$1,250.00	
Béremelésre javasoltak			

Futtatás után állítsuk vissza a környezeti beállításokat, mert különben minden listánk ilyen formátummal jelenik meg.

### Példa

Állítsuk át a fizetés értéket a helyi pénznemre:

```
SQL> COLUMN sal FORMAT L99999
```

Listázzuk ki az emp tábla sal oszlopát:

```
SQL> select sal from emp;
```

```

Szo Júl 28                                lap    1
                                Alkalmazottak
                                bérkimutatása

```

```

      Havi fizetés
      -----
              Ft800
              Ft1600
              Béremelésre javasoltak
14 sor kijelölve.

```

Láthatjuk, hogy a formázási beállítások megmaradtak, kivéve a `sal` oszlopét, amit átállítottunk.

Ha fejléc, lábléc nélkül, és az átnevezett oszlopokat eredeti nevükön szeretnénk listázni, akkor minden formátum beállítást ki kell kapcsolni vagy visszaállítani.

```

SQL> SET linesize 400
SQL> TTITLE OFF           -- fejléc kikapcsolása
SQL> BTITLE OFF           -- lábléc kikapcsolása
SQL> COLUMN sal FORMAT 99999 -- sal oszlop számformátumának
                           -- visszaállítása
SQL> column job OFF       -- a job oszlopfejléc kikapcsolása
SQL> COLUMN ename OFF     -- ename oszlopfejléc kikapcsolása
SQL> CLEAR BREAKS        -- az összes BREAK beállítás törlése

```

Listázzuk most ki ismét az `emp` táblát.

```

SQL> SELECT ename, job, sal
       2 FROM emp;

```

ENAME	JOB	SAL
SMITH	CLERK	800
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
...		

14 sor kijelölve.

## Az adatkezelő nyelv (DML)

Az adatkezelő nyelv, a DML (Data Manipulation Language) az SQL nyelv fontos része. A DML utasítások segítségével

- új sort szűrhatunk a táblába,
- létező sort módosíthatunk,
- létező sort törölhetünk a táblából.

Az adattáblákon végzett fenti tevékenységek bármelyike megváltoztatja a tábla tartalmát. E változtatásokat tranzakciónak nevezzük. Az Oracle e tranzakciókat "jegyzőkönyvezi", és a későbbiek folyamán – ha szükséges – ezek "visszagörgethetők", azaz visszaállítható az adattáblák egy korábbi állapota. (Erről részletesen jelen fejezet "Adatbázis-tranzakciók" című alfejezetében lesz szó.)

Tranzakció tehát logikailag az egy munkafázisba tartozó DML utasítások sorozata. Nézzünk egy példát a tranzakcióra! Tekintsünk egy banki átutalást (innen ered a tranzakció neve is). A folyamat a következő: pénzt veszünk fel az egyik számláról, ezt a pénzt átutaljuk, s közben feljegyezzük egy naplóba, hogy milyen összeget, honnan, hová utalunk. A számla-egyenlegeket fenn kell tartani, és egyensúlyban kell lenni. Ha bármelyik művelet nem sikerül, mindhárom utasítást vissza kell vonni az egyenleg fenntartása miatt. Ekkor a tranzakciót nem véglegesítjük.

## A DML utasítások

### INSERT utasítás

Az `INSERT` utasítás egy új sort szűr be a táblába. (Több sorhoz több utasításra van szükség.)

Az utasítás alakja:

```
INSERT INTO tábla [(oszlop [, oszlop] ... )]
VALUES (érték [, érték] ...);
```

Az oszloplista megadása nem kötelező abban az esetben, ha a tábla minden oszlopának értéket adunk (az értékadás a tábla oszlopainak definiálási sorrendjében történik). Ha azonban felsoroljuk az oszlopokat, akkor az értékadás az oszlopok felsorolási sorrendjében történik.

Adatmegadáskor a karakteres és a dátum adatokat aposztrófok közé kell tenni. Ha a karakter típusú adat `NULL` értékű, akkor azt két aposztrófok egymásutánírásával (' ') jelöljük.

### Példa

Írjunk egy új sort a `dept` táblához. Először azonban nézzük meg az alapértelmezett oszlop-sorrendet.

```
SQL> DESC dept
Név                Üres?    Típus
-----
DEPTNO              NOT NULL NUMBER(2)
DNAME                VARCHA2 (14)
LOC                  VARCHA2 (13)
```

Szúrjunk hozzá egy új sort. Nem szerepel oszlopnév, tehát minden oszlopértéket meg kell adni.

```
SQL> INSERT INTO dept
      2      VALUES (50, 'FEJLESZTŐ', 'BUDAPEST');

1 sor létrejött.
```

Mivel a `loc` és a `dname` oszlop lehet `NULL` érték, ezért elfogadja a következőt:

```
SQL> INSERT INTO dept
      2      VALUES (50, 'FEJLESZTŐ', '');

1 sor létrejött.
```

vagy az alábbi is:

```
SQL> INSERT INTO dept
      2      VALUES (50, '', 'BUDAPEST');

1 sor létrejött.
```

Szúrjunk hozzá ismét egy sort, de most két megadott oszlopot töltünk fel.

```
SQL> INSERT INTO dept (deptno,dname)
      2      VALUES (60,'PIACKUTATÓ');

1 sor létrejött.
```

Tekintsünk egy másik esetet. Győrből létrejött az új 70-es részleg, de még nincs neve. Ne soroljuk fel az oszloplistában. Így a táblázatnak csak két oszlopát töltjük fel. A középső (`dname`) kimarad. `NULL` érték megadása nélkül létrehozta az új sort.

```
SQL> INSERT INTO dept (deptno, loc)
      2      VALUES (70, 'GYŐR');

1 sor létrejött.
```

Szúrunk hozzá még egy sort. Létrejött egy piackutató részleg, csak nem tudjuk a számát, ezért `NULL` értéket szeretnénk beszúrni.

```
SQL> INSERT INTO dept
      2      VALUES (NULL,'PIACKUTATÓ', 'SZEDED');
INSERT INTO dept
      *
Hiba a(z) 1. sorban:
ORA-01400: NULL ide nem szűrhető be:
("SCOTT"." " DEPT"."DEPTNO")
```

`Null` érték nem szűrhető be olyan oszlopba, amely `NOT NULL` értékű, amely azt jelenti, hogy az oszlop minden sorában értéke kell, hogy legyen. A `dept` tábla `DESC` utasítással való lekérdezésekor láttuk, hogy a `deptno` nem lehet üres, mert `NOT NULL`.

A karakter típusú oszlop `NULL` érték megadását közvetlenül is megtehetjük. Szegeden létrejött egy új 80-as részleg, de még nincs neve. Adjunk a névnek `NULL` értéket.

```
SQL> INSERT INTO dept
      2      VALUES (80, NULL, 'SZEGED');
```

1 sor létrejött.

### Példa

Érték beszúrása *helyettesítő változóval* is lehetséges. Szűrjük be a következő sort, de egyik értéket sem tudjuk előre megadni.

```
SQL> INSERT INTO dept (deptno, dname, loc)
      2      VALUES (&szam, '&reszleg_nev', '&hely');
Adja meg a(z) szam értékét: 90
Adja meg a(z) reszleg_nev értékét: RAKTÁR
Adja meg a(z) hely értékét: TATA
régi 2: VALUES (&szam, '&reszleg_nev', '&hely')
új 2: VALUES (90, 'RAKTÁR', 'TATA')
```

1 sor létrejött.

Listázzuk ki a megnövelt dept táblát.

```
SQL> SELECT * FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	FEJLESZTŐ	BUDAPEST
60	PIACKUTATÓ	
70		GYŐR
80		SZEGED
90	RAKTÁR	TATA

9 sor kijelölve.

Láthatjuk hogy a táblához új sorokat tett, mégpedig úgy, ahogyan mi akartuk.

### INSERT UTASÍTÁS HASZNÁLATA ALLEKÉRDEZÉSSSEL

Az INSERT utasításban megengedett az allekérdezés használata, ha az értékeket azonos felépítésű táblából (vagy nézetből) vesszük. Ilyen esetben elmarad a VALUES utasításrész.

Az utasítás alakja:

```
INSERT INTO tábla [(oszlop [, oszlop] ...)]
      allekérdezés;
```

### Példa

Van egy teljesen hasonló táblánk csak magyar nyelven mint a dept. Ez a reszleg tábla. Nincs 40-es részlegszámú sora. *Szűrjük be ezt a dept táblából.*

```
SQL> INSERT INTO reszleg (reszlegszam, reszlegnev, telephely)
      2      SELECT deptno, dname, loc
```

```

3          FROM dept
4          WHERE deptno = 40;

```

1 sor létrejött.

Ellenőrizzük.

```
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
20	KUTATÓ	GYŐR
30	KERESKEDŐ	HATVAN
40	OPERATIONS	BOSTON

Valóban beszúrta a 40-es sort.

## UPDATE utasítás

A sorok egyes oszlopértékeinek *módosítására* szolgáló DML utasítás az UPDATE.

Az utasítás általános alakja:

```

UPDATE tábla
    SET oszlop = érték [, oszlop = érték] ...
    [WHERE feltétel];

```

### Fontos

Ha a WHERE feltételt nem adjuk meg, a módosítás a teljes oszlopra vonatkozik.

### Példa

Tegyük fel, megtudtuk, hogy a dept tábla 60-as részlegének telephelye *Hatvan* lesz. *Módosítsuk* a sort a megfelelő értékkel.

```

SQL> UPDATE dept
2     SET loc = 'HATVAN'
3     WHERE deptno = 60;

```

1 sor módosítva.

### Példa

Egyszerre *több oszlop módosítása*. A 70-es részleg adatai frissültek, a részlegnév logisztika lett és Szolnokon nyílt meg, nem pedig Győrben.

```

SQL> UPDATE dept
2     SET dname = 'LOGISZTIKA', loc= 'SZOLNOK'
3     WHERE deptno = 70;

```

1 sor módosítva.

### Példa

*Módosítás allekérdezéssel.*

A 80-as részleg legyen ugyan olyan, mint az 50-es.

```
SQL> UPDATE dept
2     SET (dname,loc)=
3         (SELECT dname, loc
4            FROM dept
5            WHERE deptno = 50)
6     WHERE deptno = 80;
```

1 sor módosítva.

Nézzük meg az allekérdezéssel módosított dept táblát.

```
SQL> SELECT * FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	FEJLESZTO	BUDAPEST
60	PIACKUTATO	HATVAN
70	LOGISZTIKA	SZOLNOK
80	FEJLESZTO	BUDAPEST
90	RAKTAR	TATA

9 sor kijelölve.

### Példa

Módosítsuk a 70-es részleg nevét sales-re.

```
SQL> UPDATE dept
2     SET dname = 'SALES';
```

9 sor módosítva.

```
SQL> SELECT * FROM dept;
```

DEPTNO	DNAME	LOC
10	SALES	NEW YORK
20	SALES	DALLAS
30	SALES	CHICAGO
...		

...

9 sor kijelölve.

Minden dname SALES lett.  
Elfelejtettük megadni a  
módosítandó feltételt!

Ez most baj! *Mi a teendő?*

A visszagörgetés, a ROLLBACK. A tranzakciót nem véglegesítjük.

```
SQL> ROLLBACK;
A visszaállítás befejeződött.
```

```
SQL> SELECT * FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	FEJLESZTO	BUDAPEST
60	PIACKUTATO	HATVAN
70	LOGISZTIKA	SZOLNOK
80	FEJLESZTO	BUDAPEST
90	RAKTAR	TATA

10 ACCOUNTING	NEW YORK
20 RESEARCH	DALLAS
30 SALES	CHICAGO
40 OPERATIONS	BOSTON

Mivel nem véglegesítettük a tranzakciót, visszagörgethettük, és így az eredeti táblát kaptuk vissza.

## DELETE utasítás

A DELETE utasítással egy táblából sorokat törölhetünk.

Az utasítás alakja:

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

A törlési utasítás a WHERE feltételben megadott sorokra vonatkozik. Ha a WHERE feltétel hiányzik a táblából, az egész tábla törlődik.

### Példa

Töröljük ki azt a sort a dept táblából, amelyben a deptno éppen 60, valamint azt a sort, amelyben a dname (részleg neve) FEJLESZTŐ.

```
SQL> DELETE FROM dept
2   WHERE deptno = 60;

1 sor törölve.
SQL> DELETE dept
2   WHERE dname = 'FEJLESZTŐ';

2 sor törölve.
```

### Példa

Töröljük azt a sort, amelyben a deptno értéke 20. (A deptno a dept táblában PRIMARY KEY).

```
SQL> DELETE FROM dept
2   WHERE deptno =20;
DELETE from dept
*
Hiba a(z) 1. sorban:
ORA-02292: integritás megszorítás (SCOTT.EMP_FOREIGN_KEY)
megsértés - gyermek rekord található meg
```

Ez nem törölhető, mert – mint tudjuk – az emp tábla deptno oszlopa külső kulcs (Foreign key), azaz az emp tábla deptno oszlopértékei függnek, integritási megszorításban vannak a dept tábla deptno értékeivel.

### Példa

Törölhetünk egyszerre több sort is. Töröljük ki az emp táblából mindazokat, akik 1981.11.01 után léptek be.

```
SQL> DELETE
  2   FROM emp
  3   WHERE hiredate > TO_DATE('1981.09.01','YYYY.MM.DD');
```

8 sor törölve.

```
SQL> SELECT ename, job, hiredate
  2   FROM emp;
```

ENAME	JOB	HIREDATE
SMITH	CLERK	80-DEC-17
ALLEN	SALESMAN	81-FEB-20
WARD	SALESMAN	81-FEB-22
JONES	MANAGER	81-ÁPR-02
BLAKE	MANAGER	81-MÁJ-01
CLARK	MANAGER	81-JÚN-09

6 sor kijelölve.

### Példa

Törölés *másik táblából* származó adatok alapján. Az emp1 tábla ugyanazokat az adatokat tartalmazza, mint az emp.

Töröljük az emp1 táblából azokat a kereskedőket, akiknek telephelye CHICAGO.

```
SQL> DELETE from emp1
  2   WHERE deptno=
  3           (SELECT deptno
  4             FROM dept
  5             WHERE loc ='CHICAGO');
```

6 sor törölve.

Ellenőrizzük:

```
SQL> SELECT ename, job, deptno
  2   FROM EMP1
  3   ORDER BY deptno;
```

ENAME	JOB	DEPTNO
CLARK	MANAGER	10
KING	PRESIDENT	10
MILLER	CLERK	10
SMITH	CLERK	20
ADAMS	CLERK	20

...

8 sor kijelölve.

## Adatbázis-tranzakciók

A tranzakció az SQL utasításoknak egy olyan sorozata, amelyet egységként kezel a rendszer. Egy tranzakció a DML utasítások olyan sorozata, amelyek egyetlen adatmódosítás összetartozó lépéseit végzi el. Egy tranzakciót a `COMMIT` utasítással zárhatunk le. Egy teljes tranzakciót vagy annak egy részét érvényteleníthetjük a `ROLLBACK` utasítással. Így biztosítható az adatbázis konzisztenciája (ellentmondásmentessége, épsége) olyan esetekben, amikor valamilyen hiba megszakítja egy tranzakcióban lévő utasítások feldolgozását, vagy kiderül, hogy az adatbázis módosítása hibás volt.

Az Oracle a tranzakciókat zárolási mechanizmus segítségével az úgynevezett többverziós konzisztencia modell alapján kezeli. Minden tranzakció létrehozza a saját adatbázis másolatát, tehát létezik olyan időszak, amikor az adatbázisnak egymást átfedő verziói is léteznek.

A zár olyan eszköz, amely megakadályozza – több felhasználó esetén – a különböző tranzakciók közötti káros kölcsönhatást. Az Oracle a zárat automatikusan kezeli. Létezik SQL utasítás is a táblák zárolására, ez a `LOCK TABLE` utasítás. A zárolással a többi felhasználó hozzáférést korlátozhatjuk. A cél az, hogy az adatintegritás ne sérüljön. A zárolási módoktól függetlenül a zár érvényben marad mindaddig, amíg a felhasználó nem véglegesíti, vagy vissza nem görgeti a tranzakciót.

Alaphelyzetben a rendszer a tranzakción belül az utasításszintű konzisztenciát biztosítja. Ez azt jelenti, hogy egy lekérdezés alatt az adatok állapota nem változik meg. Ha azt szeretnénk, hogy az egész tranzakció alatt a lekérdezések számára az adatok ne változzanak, akkor tranzakciószintű olvasási konzisztencia szükséges. Ilyen esetben a csak olvasható tranzakciót szükséges használni, amelyet a `SET TRANSACTION` utasítással lehet beállítani. Ekkor csak lekérdezéseket és bizonyos vezérlő utasításokat hajthatunk végre. Más felhasználó módosíthatja ugyan a tranzakció által lekérdezett adatokat, de ezek a tranzakció számára nem látszanak. A tranzakció első utasítása ekkor a `SET TRANSACTION` parancs, amit úgy érhetünk el, hogy előtte kiadunk egy `COMMIT`, vagy `ROLLBACK` utasítást, aminek hatására a rendszer által az adatok számára lefoglalt erőforrások felszabadulnak.

A tranzakció befejezése után új tranzakció kezdődik.

Beszélhetünk tranzakcióbeli SQL utasításokról, ezek a tranzakcióban résztvevő DML utasítások, amelyek az adatbázis adott tábláinak megváltozását eredményezik. Ilyenek az `INSERT`, az `UPDATE` és a `DELETE`.

A tranzakciót kezelő utasítások a `ROLLBACK` (visszagörgetés) és `SAVEPOINT` (mentési pont) a tranzakciót lezáró utasítás pedig a `COMMIT` utasítás.

A DDL és DCL utasítások végrehajtása automatikusan véglegesít, tehát ezek az utasítások implicit módon zárják le a tranzakciót.

Ha a tranzakción belül kiadott SQL utasítások sorozatának végrehajtása a kívánt eredményt hozta, akkor a `COMMIT` utasítással véglegesítjük az adatbázison történt változtatásokat. Ha a módosítások nem a kívánt eredményt szolgáltatták, akkor az utasításokat nem hagyjuk jóvá, visszagörgetjük a `ROLLBACK` utasítással. Ha a DML utasítások elvégzésének sorozatában biztosan tudjuk, hogy az addig elvégzett műveletek hibátlanok, akkor mentési pontot illeszthetünk be. Ebben az esetben lehetséges a visszagörgetés a mentési pontig. Így több részletben ellenőrizhetjük az adatbázisok módosításának helyességét, hogy az minden esetben, konzisztens maradjon.

Vizsgáljuk meg a tranzakciót kezelő SQL utasításokat.

A `SAVEPOINT` utasítás lehetővé teszi, hogy a tranzakció közben mentési pontot állítsunk be. A mentési pont arra szolgál, hogy a visszagörgetést szabályozni tudjuk. A mentési pont megadásával a logikailag összetartozó SQL utasítások egy-egy csoportját elnevezhetjük. A `ROLLBACK` utasításban, ha szükséges, a megadott névre hivatkozva érvényteleníthetjük az addig elvégzett utasításokat. A `SAVEPOINT` utasítással egymás utáni, vagy egymásba ágyazott utasításcsoportokat is képezhetünk, és ezzel irányíthatjuk a hibák kezelését.

A mentési pont megadásának alakja:

```
SAVEPOINT mentési_pont_neve
```

A `ROLLBACK` utasítás visszavonja a függőben lévő módosításokat, amit az utolsó véglegesítés óta elvégeztünk. Így az adatbázis a módosítások előtti állapotba kerül vissza. A parancs kiadása feloldja a táblákban elhelyezett zárat is. A visszagörgetés nemcsak az utolsó mentési pontig, hanem az időben korábban kiadott mentési pontig történhet.

Az utasítás alakja:

```
ROLLBACK [TO [SAVEPOINT] mentési_pont_neve ]
```

Több mentési pont is beilleszthető. A visszagörgetés a megadott mentési pontig érvényes, és az időben később kiadott mentési pontokat, zárat feloldja. Ez azt jelenti, hogy ha kiadtunk egy `COMMIT` utasítást, majd egy `A1` mentési pontot, időben utána egy `A2` mentési pontot, akkor a következő visszagörgetések lehetségesek:

```
ROLLBACK TO A2      visszagörgetés az A2 mentési pontig.
ROLLBACK TO A1      visszagörgetés az A1 mentési pontig, és az A2 mentési pont
                    és a felhasználó által kiadott zárat törlése,
ROLLBACK             visszagörgetés a COMMIT utasításig, minden mentési pont törlése,
                    a táblákban elhelyezett zárat feloldása.
```

A `COMMIT` utasítás véglegesíti az adatmódosításokat, a végrehajtott DML utasításokat, lezárja a tranzakciót. Mindaddig, míg ki nem adjuk a `COMMIT` utasítást, a DML utasítások hatása átmeneti. A DML utasítások eredményének helyességét a `SELECT` utasítással ellenőrizhetjük.

Az SQL\*Plus rendszerben létezik egy `AUTO COMMIT` rendszerváltozó, amit ki (`OFF`) és be (`ON`) lehet kapcsolni. Bekapcsolt állapotában minden egyes DML utasítást lefutása után véglegesít! Ha az `AUTO COMMIT` rendszerváltozó ki van kapcsolva, akkor a felhasználó `COMMIT` utasítással érvényesítheti a végrehajtott DML műveletek sorozatát.

### Példa

Hozzunk létre a `dept` táblához teljesen hasonló táblát, hogy azon bemutassuk a tranzakciókat. Megtalálható a `reszleg.sql` script programban.

```
SQL> @ c:\a\reszleg
A tábla létrejött.
1 sor létrejött.
...
```

Létrejött-e a tábla, és feltöltöttük – e adatokkal. Kérdezzük le.

```
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
20	KUTATÓ	GYŐR

30 KERESKEDŐ	HATVAN
40 GYÁR	SZEGED

Érvénytelenítsük.

```
SQL> ROLLBACK;
A visszaállítás befejeződött.
SQL> SELECT * FROM reszleg;
```

nincsenek kijelölve sorok

A `reszleg` tábla tartalmát törölte, magát a táblát nem, mert a `CREATE` implicit módon véglegesíti a tranzakciót. Tehát a tábla megmaradt, de nincsenek sorai.

```
SQL> desc reszleg;
```

Név	Üres?	Típus
RESZLEGSZAM	NOT NULL	NUMBER(2)
RESZLEGNEV		VARCHAR2(14)
TELEPHELY		VARCHAR2(13)

Töltsük fel adatokkal újra a táblát, és utána véglegesítsük. Futtassuk le a `reszleg.sql` script programot.

```
SQL> @ c:\a\reszleg
1 sor létrejött.
...
SQL> COMMIT;
A jóváhagyás befejeződött.
```

Módosítsuk a `reszleg` táblát. Szúrjunk be két sort.

```
SQL> INSERT INTO reszleg VALUES (50,'PIACKUTATÓ','');
1 sor létrejött.
SQL> INSERT INTO reszleg VALUES (60,'RAKTÁR','TATA');
1 sor létrejött.
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
20	KUTATÓ	GYŐR
30	KERESKEDŐ	HATVAN
40	GYÁR	SZEGED
50	PIACKUTATÓ	
60	RAKTÁR	TATA

6 sor kijelölve.

Tegyük ide mentési pontot, A1-t

```
SQL> SAVEPOINT A1;
A mentési pont létrejött.
```

Mivel az 50-es részlegnek nemadtunk telephelyet, módosítsuk a sort. Legyen a telephely Eger.

```
SQL> UPDATE reszleg
      2     SET telephely = 'EGER'
      3     WHERE részlegszám = 50;
```

1 sor módosítva.

Töröljük azt a sort, amelyik telephelye Győrött van

```
SQL> DELETE FROM reszleg
      2     WHERE telephely = 'GYŐR';
1 sor törölve.
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
30	KERESKEDŐ	HATVAN
40	GYÁR	SZEGED
50	PIACKUTATÓ	EGER
60	RAKTÁR	TATA

Tegyük ide is egy mentési pontot, az A2-t.

```
SQL> SAVEPOINT A2;
A mentési pont létrejött.
```

Szűrjünk be egy 70-es részlegszámú sort.

```
SQL> INSERT INTO reszleg VALUES (70,'LOGISZTIKA', 'PÉCS');
```

1 sor létrejött.

```
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
30	KERESKEDŐ	HATVAN
40	GYÁR	SZEGED
50	PIACKUTATÓ	EGER
60	RAKTÁR	TATA
70	LOGISZTIKA	PÉCS

6 sor kijelölve.

Görgezzük vissza, azaz érvénytelenítsük az utolsó módosítást, a 70-es sor beszúrását.

```
SQL> ROLLBACK TO A2;
```

A visszaállítás befejeződött.

```
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
30	KERESKEDŐ	HATVAN
40	GYÁR	SZEGED
50	PIACKUTATÓ	EGER
60	RAKTÁR	TATA

Görgessük vissza A1-ig.

```
SQL> ROLLBACK TO A1;
```

A visszaállítás befejeződött.

```
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
20	KUTATÓ	GYŐR
30	KERESKEDŐ	HATVAN
40	GYÁR	SZEGED
50	PIACKUTATÓ	
60	RAKTÁR	TATA

6 sor kijelölve.

Görgessük vissza egészen a COMMIT utasításig, vagyis a tranzakció kezdetéig.

```
SQL> ROLLBACK;
```

A visszaállítás befejeződött.

```
SQL> SELECT * FROM reszleg;
```

RESZLEGSZAM	RESZLEGNEV	TELEPHELY
10	KÖNYVELŐ	BUDAPEST
20	KUTATÓ	GYŐR
30	KERESKEDŐ	HATVAN
40	GYÁR	SZEGED

## Adatbázis objektumok definiálása (DDL)

Az Oracle adatbázis több objektumot tartalmaz. Minden elemet, adatszerkezetet meg kell határozni az adatbázis tervezése során, hogy a felhasználó megfelelő módon hivatkozhaszon rá. Mivel könyvünk célja nem elsősorban az adatbázis megtervezése, hanem annak felhasz-

nálása, ezért csak röviden ismertetjük ezeket, hogy a felhasználó kezelni tudja az adatbázis objektumokat.

Egy objektum neve teljesítse a következőket:

- legfeljebb 32 karakter hosszú lehet, kivéve az adatbázis nevét, amely legfeljebb 8 karakteres lehet,
- a névben megengedett karakterek: az angol ábécé betűi, a számjegyek, a "\$", a "#" és az "\_" karakterek,
- egy felhasználó saját objektumainak neve különböző kell, hogy legyen,
- a kis és nagybetűket a rendszer nem tekinti különbözőnek.
- névként nem használhatóak az SQL, PL/SQL által foglalt kulcsszavak.

Javasoljuk az objektumok úgynevezett beszédes elnevezését. Ez azt jelenti, hogy az objektum neve feltétlenül utaljon a tartalmára.

Jelen alfejezetben az alábbi adatbázis-objektumokat ismertetjük:

Tábla	Adatokat tárol. Sorokból és oszlopokból áll.
Nézet	Egy vagy több tábla lekérdezéséből létrehozott táblaszerű objektum, amelyet eltárol a rendszer.
Index	Táblához kapcsolódó segédadat, amely felgyorsítja egyes lekérdezések végrehajtását.
Szinonima	Objektumok helyettesítő neve.

## Adatszótár

Az Oracle adatbázisában kétféle tábla van:

- felhasználói
- adatszótárbeli

A felhasználói táblát a felhasználó hozza létre. Az Oracle által létrehozott és kezelt táblák gyűjteménye az adatszótár.

Az adatszótár tábláinak tulajdonosa a sys felhasználó. (A sys tulajdonos tipikusan adatbázis adminisztrátor). Az alaptáblákat a felhasználók ritkán használják, mert a felhasználó számára nehezen értelmezhető. A felhasználók általában az *adatszótár nézeteit* használják, mert ebben tárolja az adatbázis adminisztrátor a felhasználókra vonatkozó adatokat.

Az adatszótár tárolja a felhasználók nevét, jogosultságait, adatbázis-objektumaik nevét, a táblákra vonatkozó megszorításokat, és egyéb vizsgálati adatokat.

Az adatszótár nézetei négy kategóriába sorolhatók. A csoportokat az előtagok különböztetik meg egymástól.

Előtag	Leírás
USER_	A felhasználó tulajdonában lévő objektumok adatait tartalmazza.
ALL_	A felhasználó számára elérhető összes objektum adatait tartalmazza.
DBA_	Csak a DBA (database administrator) szerepkörű felhasználók férhetnek hozzá.

## Adatszótár lekérdezése

A felhasználó leggyakrabban használt nézetei:

- user\_tables
- user\_objects
- user\_catalog

Ezek szokásos módon (a `SELECT` és a `DESC` utasításokkal) kérdezhetők le. A `user_tables` szinonimája a `tabs`, a `user_catalog` szinonimája a `cat`, ez tehát lekérdezhető így is:

```
SQL> SELECT * FROM cat;
TABLE_NAME                                TABLE_TYPE
-----
...
DEPT                                       TABLE
DUMMY                                    TABLE
EMP                                       TABLE
EMP1                                     TABLE
RESZLEG                                  TABLE
SALGRADE                                 TABLE
```

## Adattípusok a táblákban

Adattípusok	Leírás
<code>VARCHAR2(méret)</code>	Változó hosszúságú karakteres adat, maximális hossza: <i>méret</i> .
<code>CHAR(méret)</code>	Kötött hosszúságú karakteres adat, hossza: <i>méret</i> .
<code>NUMBER(m,t)</code>	Változó hosszúságú numerikus adat ( <i>m</i> = mezőszélesség, mely az esetleges előjelet is tartalmazza, <i>t</i> = ebből a tizedesek száma, az <i>m</i> és a <i>t</i> megadása egyaránt elmaradhat).
<code>DATE</code>	Dátum és időérték.
<code>LONG</code>	Változó hosszúságú karakteres adat (max. 2 GByte).
<code>CLOB</code>	Változó hosszúságú egybájtos karakterekből álló adat (max. 4 GByte).
<code>RAW(méret)</code>	Kötött hosszúságú bináris adat, hossza: <i>méret</i> (max. 255 Byte).
<code>LONG RAW</code>	Változó hosszúságú bináris adat (max. 2 GByte).
<code>BLOB</code>	Bináris adat (max. 4 GByte).
<code>BFILE</code>	Mutató egy bináris állományra.

## A DDL utasítások

### Tábla létrehozása (CREATE TABLE)

A `CREATE TABLE` egy DDL (Data Definition Language) utasítás. A tábla létrehozása azonnal végrehajtásra kerül, és bekerül az adatszótárba.

A tábla létrehozásához megfelelő jogosultság szükséges, és megfelelő memóiahelynek kell rendelkezésre állni. Jogokat az adatbázis-adminisztrátor adhat az adatvezérlő (DCL) nyelv utasításaival. (Erről röviden még szólunk).

A tábla létrehozás rövid alakja:

```
CREATE TABLE [felhasználó.]tábla
    (oszlop adattípus [DEFAULT kif] [oszlop_megszorítás]
    [, oszlop adattípus [DEFAULT kif] [oszlop_megszorítás] ...);
```

: felhasználó	a tábla tulajdonosának neve,
tábla	a létrehozandó tábla neve,
oszlop	a tábla oszlopa,
adattípus	az oszlop adattípusa,
DEFAULT kif	alapértelmezett kifejezés. Ha az INSERT utasításban nem adunk értéket, a kif lesz az oszlop értéke. Értéke literál, kifejezés, vagy SQL függvény (például sysdate) lehet. Az alapértelmezett kif adattípusának meg kell egyeznie az oszlop adattípusával.
oszlop_megszorítás	megszorítások (még tárgyaljuk)

### Példa

Hozzunk létre egy táblát. Legyen ez a `reszleg` tábla.

```
SQL> CREATE TABLE reszleg (
2   reszlegszam          NUMBER(2) NOT NULL,
3   reszlegnev           VARCHAR2(14),
4   telephely            VARCHAR2(13));
```

A tábla létrejött.

Reszlegszam oszlop adattípusa 2 számjegyű szám, és megszorítása a NOT NULL.

```
SQL> DESC reszleg;
Név                Üres?      Típus
-----
RESZLEGSZAM        NOT NULL   NUMBER(2)
RESZLEGNEV                     VARCHAR2(14)
TELEPHELY                     VARCHAR2(13)
```

### TÁBLA LÉTREHOZÁSA ALLEKÉRDEZÉssel

Az SQL módot ad arra, hogy allekérdezéssel is létrehozhatunk táblát. Ez abban az esetben szükséges, ha az eredeti tábla meghatározott részhalmazát, mint önálló táblát szeretnénk előállítani.

Általános alak:

```
CREATE TABLE tábla
    [(oszlop [DEFAULT kif] [megszorítás] [, oszlop [DEFAULT kif] [megszorítás]] ...)]
AS
    allekérdezés;
```

A rendszer létrehoz egy új táblát, és ebbe helyezi el az allekérdezés által visszaadott sorokat.

A megadott oszlopok számának meg kell egyeznie az allekérdezés `SELECT` utasítása által visszaadott oszlopok számával. Ha nem adunk meg oszlopokat, akkor a létrejövő új tábla oszlopai megegyeznek az allekérdezés oszlopainak nevével.

### Példa

Hozzunk létre egy alkalmazott nevű táblát az `emp` tábla hivatalnokaiból (`clerk`).

```
SQL> CREATE TABLE alkalmazott
2  ( azonosito, nev, evés_fizetes)
3  AS
4  SELECT empno, ename, sal* 12
5  FROM EMP
6  WHERE job = 'CLERK';
A tábla létrejött.
```

```
SQL> SELECT * FROM alkalmazott;
```

AZONOSITO	NEV	EVES_FIZETES
7369	SMITH	9600
7876	ADAMS	13200
7900	JAMES	11400
7934	MILLER	15600

Ez valóban a hivatalnokok résztáblája, mégpedig az éves fizetéssel. A tábla felépítése az alábbi:

```
SQL> desc alkalmazott
Név          Üres?      Típus
-----
AZONOSITO    NOT NULL  NUMBER(4)
NEV          VARCHAR2(10)
EVES_FIZETES NUMBER
```

### Tábla módosítás (ALTER)

Az `ALTER TABLE` utasítással a már létező tábla szerkezetét változtathatjuk meg. Felvehetünk új oszlopot, módosíthatunk egy már meglévő oszlopot és megváltoztathatjuk az alapértelmezett értékét.

Az utasítás alakja:

- új oszlop felvétele
 

```
ALTER TABLE táblanév
      ADD      (oszlop adattípus [DEFAULT kif] [megszorítás]
               [, oszlop adattípus [DEFAULT kif] [megszorítás]] ...)
```
- oszlop módosítása
 

```
ALTER TABLE táblanév
```

```
MODIFY (oszlop adattípus [DEFAULT kif] [megszorítás]
        [, oszlop adattípus [DEFAULT kif] [megszorítás]]...);
```

- oszlop törlése (8.1.5 verzió után)

```
ALTER TABLE táblanév
```

```
DROP COLUMN oszlop
```

ahol

táblanév	a tábla neve,
oszlop	az új, módosítandó vagy törlendő oszlop neve,
adattípus	az új oszlop, vagy már létező oszlop módosítandó adattípusa,
DEFAULT kif	az új alapértelmezett kifejezés.

### ADD UTASÍTÁSRÉSZ (OSZLOP MÓDOSÍTÁS)

Az ADD utasításrészrel a táblába új oszlopot vehetünk fel. Az új oszlop a tábla szerkezetében az utolsó helyre kerül.

#### Példa

Vegyünk fel egy új oszlopot a most létrehozott `alkalmazott` táblába. Az új oszlop neve legyen `beosztás`.

```
SQL> ALTER TABLE alkalmazott
2 ADD (beosztas VARCHAR2(12));
```

A tábla módosítva.

```
SQL> SELECT * FROM alkalmazott;
```

AZONOSITO	NEV	EVES_FIZETES	BEOSZTAS
7369	SMITH	9600	
7876	ADAMS	13200	
7900	JAMES	11400	
7934	MILLER	15600	

Az új oszlop a tábla utolsó oszlopa lesz. Ha az új oszlop beszúrásakor voltak már értékei a soroknak, az új oszlop minden értéke `NULL` lesz.

### MODIFY UTASÍTÁSRÉSZ (OSZLOP MÓDOSÍTÁS)

Megváltoztathatjuk az oszlopok adattípusát, méretét és alapértelmezett értékét.

A módosítás hatása a tábla csak azon oszlopaira vonatkozik, amelyeket a módosítás után szűrtünk be.

A módosítási lehetőségek:

- növelhetjük a számértéket tartalmazó oszlopok szélességét és pontosságát,
- csökkenthetjük az oszlopok szélességét, ha csak `NULL` érték szerepel bennük, illetve a táblázatnak még nincsenek kitöltött sorai,
- megváltoztathatjuk az oszlopok adattípusát, ha csak `NULL` értéket tartalmaznak,
- módosíthatjuk a `CHAR` típusú oszlopot `VARCHAR2` típusúra, illetve a `VARCHAR2` típust `CHAR` típusra, ha az oszlop csak `NULL` értéket tartalmaz, vagy ha nem változtatjuk meg a méretét.

**Példa**

Módosítsuk az alkalmazott tábla név oszlopának méretét. Legyen maximum 15 karakter hosszú.

```
SQL> ALTER TABLE alkalmazott
      2 MODIFY ( nev VARCHAR2(15));
```

A tábla módosítva.

Módosítsuk a nevét a 7876 azonosító számú alkalmazottnak.

```
SQL> UPDATE alkalmazott
      2 SET nev = 'MAGYAR ALADÁR'
      3 WHERE azonosito = 7876;
```

1 sor módosítva.

```
SQL> SELECT * FROM alkalmazott;
```

AZONOSITO	NEV	EVES_FIZETES	BEOSZTAS
7369	SMITH	9600	
7876	MAGYAR ALADÁR	13200	
7900	JAMES	11400	
7934	MILLER	15600	

A módosítás megtörtént, mert az új név 14 karakter hosszú, ami az eredeti maximum 10 karakter hosszú név oszlopba nem fért volna el.

**DROP UTASÍTÁSRÉSZ (OSZLOP TÖRLÉSE)**

Egy táblából a `DROP COLUMN` utasításrész használatával törölhetünk oszlopokat. A törlendő oszlopok tartalmazhatnak adatokat. Egyszerre egy oszlopot törölhetünk. Törlés után *nem lehet* helyreállítani, nem lehet visszagörgetni!

Töröljük az alkalmazott táblából a beosztás oszlopot.

```
SQL> ALTER TABLE alkalmazott
      2 DROP COLUMN beosztas;
```

A tábla módosítva.

```
SQL> DESC alkalmazott;
```

Név	Üres?	Típus
AZONOSITO	NOT NULL	NUMBER(4)
NEV		VARCHAR2(10)
EVES_FIZETES		NUMBER

Valóban kitörölte.

**Megjegyzés**

Ez az utasítás az Oracle-rendszerben csak a 8i verziótól kezdve működik!

### SET UNUSED (TÖRLÉSRE JELÖLÉS)

A `SET UNUSED` utasításrészrel jelölhetjük meg azokat az oszlopokat, amelyeket később törölni akarunk. Tulajdonképpen az oszlopok logikai törlése, használaton kívül helyezése mindaddig, amíg a fizikai törlés meg nem történik. Az `UNUSED` utasításrészrel megjelölt oszlopok nem érhetők el többé a felhasználó számára, de az oszlop által elfoglalt memóriaterület nem szabadult fel. A törlésre megjelölt oszlop neve újra felhasználható a fizikai törlése előtt is. A `SET UNUSED` utasításrész gyorsabban hajtódik végre, mint a `DROP` utasításrész. A munka végeztével pedig, az eddig törlésre kijelölt oszlopok memóriaterületét egyszerre lehet felszabadítani a `DROP UNUSED COLUMNS` utasításrészrel.

Az utasítás alakja:

```
ALTER TABLE tábla
SET UNUSED [column] (oszlop [, oszlop]...);
```

A hozzá tartozó *fizikai törlés* utasítása:

```
ALTER TABLE tábla
DROP UNUSED COLUMN;
```

### Példa

Jelöljük meg törlésre a `reszlegnev` oszlopot a `reszleg` táblából.

Lehetséges a `SET UNUSED` és a `SET UNUSED COLUMN` utasításrészekkel.

```
SQL> ALTER TABLE reszleg
2 SET UNUSED (reszlegnev);
```

A tábla módosítva.

```
SQL> DESC reszleg;
Név          Üres?      Típus
-----
RESZLEGSZAM  NOT NULL  NUMBER(2)
TELEPHELY                                VARCHAR2(13)
```

```
SQL> ALTER TABLE reszleg
2 SET UNUSED COLUMN reszlegnev;
```

A tábla módosítva.

```
SQL> DESC reszleg;
Név          Üres?      Típus
-----
RESZLEGSZAM  NOT NULL  NUMBER(2)
TELEPHELY                                VARCHAR2(13)
```

Szúrjunk be egy ugyanolyan nevű új oszlopot.

```
SQL> ALTER TABLE reszleg
2 ADD reszlegnev char(5);
```

A tábla módosítva.

```
SQL> DESC reszleg;
Név          Üres?      Típus
-----
RESZLEGSZAM  NOT NULL  NUMBER(2)
TELEPHELY                                VARCHAR2(13)
RESZLEGNEV                                CHAR(5)
```

Láthatjuk, hogy ez már egy másik oszlop, mert eddig a `reszlegnev` a középső oszlop volt.

A `SET UNUSED` utasításrészrel egyszerre több oszlop is megjelölhető törlésre.

### Példa

Jelöljük ki törlésre a `reszleg` táblából a `reszlegnev` és `telephely` oszlopokat.

```
SQL> ALTER TABLE reszleg
      2 SET UNUSED (reszlegnev, telephely);
```

A tábla módosítva.

```
SQL> DESC reszleg;
Név          Üres?      Típus
-----
RESZLEGSZAM   NOT NULL   NUMBER(2)
```

### Példa

Már három oszlopot megjelöltünk törlésre, *töröljük* is ki a memóriából.

```
SQL> ALTER TABLE reszleg
      2 DROP UNUSED COLUMNS;
```

A tábla módosítva.

A NOT NULL megszorítással ellátott oszlopot is meg lehet jelölni törlésre.

```
SQL> ALTER TABLE reszleg
      2 SET UNUSED COLUMN reszlegszam;
```

A tábla módosítva.

```
SQL> DESC reszleg;
Név          Üres?      Típus
-----
TELEPHELY    VARCHAR2(13)
```

## DROP TABLE utasítás

A felhasználói táblákat a felhasználó, a tulajdonos törölheti, vagy aki DROP ANY TABLE jogosultsággal rendelkezik. A törlés a tábla definícióját, minden adatát és a hozzá tartozó indexeket is törli a memóriából. A táblák törlése maga után vonja, hogy a táblára épülő egyéb objektumok (például nézet) érvénytelenné válnak.

Az utasítás alakja:

```
DROP TABLE tábla;
```

### Fontos

A DROP TABLE utasítás  *visszavonhatatlan!* (Minden DDL utasítás implicit jóváhagyást hajt végre.)

### Példa

Töröljük a reszleg táblát.

```
SQL> DROP TABLE reszleg;
A tábla eldobva.

SQL> DESC reszleg;
ERROR:
ORA-04043: a(z) reszleg objektum nem
létezik
```

A reszleg tábla objektum már nem létezik, tehát törlődött.



```
SQL> SELECT * FROM user_tab_comments;
```

TABLE_NAME	TABLE_TYPE
COMMENTS	
ALKALMAZOTT	TABLE
DEPT	TABLE
DOLGOZO	TABLE
A hivatalnokokra vonatkozó résztábla	
EMP	TABLE

## Megszorítások

Egy adatbázis általában több táblából áll, ahol a táblák kapcsolatban állhatnak egymással. A függőségek leírásánál (II. rész) már megismertük a kulcsok fogalmát.

Az integritási megszorítások olyan szabályok, amelyek a táblák oszlopaira vonatkoznak. A szabályok olyan alapvetők, hogy maga az Oracle rendszer automatikusan ellenőrzi teljesülésüket. A leggyakrabban használt megszorítások biztosítják azt, hogy ha az előírt feltételek nem teljesülnek, akkor a DML utasítások ne hajtsódjanak végre (az adatok bevitele, törlése, módosítása ne történjen meg). Ezek a DML műveletek csak az integritási megszorítások teljesülése esetén sikeresek. Ilyen integritási megszorítás például a `NOT NULL` megszorítás is. Létezik például olyan megszorítás, amely az oszlop értéktartományára vonatkozik. Könyvünk azokkal az általános megszorításokkal foglalkozik, melyek a biztonságos adatbázis-kezeléshez szükségesek.

Az integritási megszorítások típusai:

<code>NOT NULL</code>	Biztosítja, hogy az oszlop minden sorában (minden oszlopérték) valódi érték legyen. (Nem lehet <code>NULL</code> -érték.)
<code>UNIQUE</code>	Egyedi érték. A megszorítás biztosítja, hogy az oszlop minden értéke különböző legyen. (Lehet <code>NULL</code> -érték.)
<code>PRIMARY KEY</code>	Elsődleges kulcs. Biztosítja, hogy a tábla elsődleges kulcs oszlopai egyértelműen azonosítani tudják a sort. Az elsődleges kulcsok értékei (egy vagy több oszlop) mind különböző értéket vesznek fel. (Nem lehet <code>NULL</code> -érték.)
<code>FOREIGN KEY</code>	Külső kulcs, idegen kulcs vagy hivatkozási integritási megszorítás. Segítségével kijelölhetünk egy, vagy több oszlopot idegen kulcsként, s ez egy kapcsolatot hoz létre a hivatkozott tábla egyik oszlopával, vagy oszlopainak kombinációjával. Ezzel biztosíthatjuk, hogy az oszlopba csak a hivatkozott elsődleges kulcsnak megfelelő adatok kerüljenek. Az idegen kulcs értékeinek meg kell egyezniük a hivatkozott oszlop(ok) értékeinek egyikével. Azt a táblát, melyben valamely oszlophoz az idegen kulcs tulajdonságot rendeljük, <i>gyermektáblának</i> , míg a

	másik táblát <i>szülőtáblának</i> nevezzük (persze csak erre az oszlopra vonatkozóan).
CHECK	Az oszlop értékére vonatkozó feltétel megadása. A feltételnek új adat bevitele esetén teljesülni kell.

A megszorításokat célszerű elnevezni. A nevek legyenek egyértelműek, mert akkor könnyű rájuk hivatkozni. Ha a felhasználó nem ad nevet a megszorításoknak, akkor az Oracle egy `SYS_Cn` alakú nevet generál. A név azonosítását, egyediségét az *n* (egész) sorszám biztosítja.

Az integritási megszorítások létrehozhatók:

- a tábla létrehozásával együtt, és
- utólag a tábla létrehozása után.

A megszorításokat szintén az adatszótár nézeteiben tekinthetjük meg.

A megszorítás lehet:

- táblaszintű, és
- oszlopszintű.

Kényszer táblaszintű megadásának alakja:

```
CREATE TABLE [felhasználó.] tábla
    (oszlop adattípus [DEFAULT kifejezés] [oszlop_megszorítás]
    [, oszlop adattípus [DEFAULT kifejezés] [oszlop_megszorítás] ]
    [tábla_megszorítás] [, tábla_megszorítás] ...);
```

ahol *oszlop\_megszorítás* az oszlopra vonatkozó integritási megszorítás,  
*tábla\_megszorítás* a táblára vonatkozó integritási megszorítás.

A megszorításokat két szinten definiálhatjuk a `CONSTRAINT` kulcsszó segítségével, ezek:  
*oszlop\_megszorítás* egyetlen oszlopra vonatkozik, és az adott oszlop definíciójának részeként adjuk meg,

*oszlop* [`CONSTRAINT megszorítás_név`] *megszorítás\_típus*

*tábla\_megszorítás* egy vagy több oszlopra vonatkozik, és a tábla oszlopainak definíciójától függetlenül adjuk meg,

*oszlop* [, *oszlop*]... [`CONSTRAINT megszorítás_név`] *megszorítás\_típus*

Tábla szintű megszorítás a `NOT NULL` kivételével bármelyik típusú megszorítás lehet.

## NOT NULL megszorítás

A `NOT NULL` megszorítást *csak oszlopszinten* lehet megadni. Ez a megszorítás biztosítja, hogy az oszlop minden sorában valódi érték legyen, egyik oszlopérték sem lehet kitöltetlen, azaz `NULL`-érték.

### Példa

Hozzuk létre a `reszleg` táblát, és adjunk a `reszlegszam`-nak `NOT NULL` megszorítást.

```
SQL> CREATE TABLE reszleg (
2     reszlegszam    NUMBER(2)    CONSTRAINT reszleg_szam NOT NULL,
3     reszlegnev     VARCHAR2(14),
4     telephely      VARCHAR2(13));
```

A tábla létrejött.

```
SQL> DESC reszleg;
Név                Üres?      Típus
-----
RESZLEGSZAM        NOT NULL   NUMBER(2)
RESZLEGNEV          NULL      VARCHAR2(14)
TELEPHELY           NULL      VARCHAR2(13)
```

A NOT NULL megszorítás neve: `reszleg_szam`

Próbáljuk ki. Szűrjünk be egy sort, melynél a `reszlegszam`-nak Null értéket adunk.

```
SQL> INSERT INTO reszleg VALUES (NULL, 'KERESKEDŐ', 'EGER')
INSERT INTO reszleg VALUES (NULL, 'KERESKEDŐ', 'EGER')
*
Hiba a(z) 1. sorban:
ORA-01400: NULL ide nem szűrhető be:
("SCOTT"."RESZLEG"."RESZLEGSZAM")
```

## UNIQUE megszorítás

A **UNIQUE** megszorítás azt biztosítja, hogy az adott oszlop értékei mind különbözzenek egymástól. Ha a megszorítás oszlopokra vonatkozik, akkor az oszlopok értékeinek kombinációja csak egyszer fordulhat elő. A **UNIQUE egyedi kulcs**, ha egy oszlopra vonatkozik, és **egyedi összetett kulcs**, ha több oszlopra vonatkozik.

Tartalmazhatnak-e az egyedi kulcsok **NULL** értéket? A válasz igen. Miért? Mert a **NULL** értéket nem tekinti a rendszer semmi más értékkel sem azonosnak. Ez tehát azt jelenti, hogy egy oszlopban (oszlopokban) szereplő **NULL** érték mindig kielégíti az **UNIQUE** megszorítást.

Az **UNIQUE** megszorítás tábla és oszlopszinten is megadható.

### Példa

Hozzuk létre a részlegnévre az **UNIQUE** megszorítást.

```
SQL> CREATE TABLE reszleg (
2     reszlegszam          NUMBER(2) CONSTRAINT reszleg_szam NOT NULL,
3     reszlegnev           VARCHAR2(14),
4     telephely            VARCHAR2(13),
5     CONSTRAINT reszleg_nev UNIQUE(reszlegnev));
```

Próbáljuk feltölteni adatokkal a táblát.

```
SQL> INSERT INTO reszleg VALUES (10, 'KÖNYVELŐ', 'BUDAPEST');
1 sor létrejött.
```

Próbáljunk meg ugyanolyan `reszlegnev` értéket (`KÖNYVELŐ`) megadni egy újabb sor beszúrásakor.

```
SQL> INSERT INTO reszleg VALUES (20, 'KÖNYVELŐ', 'EGER')
INSERT INTO reszleg VALUES (20, 'KÖNYVELŐ', 'EGER')
*
```

```
Hiba a(z) 1. sorban:
ORA-00001: a(z) (SCOTT.RESZLEG_NEV)
eggyediségre vonatkozó megszorítás nem teljesül
```

UNIQUE megszorítás neve: *reszleg\_nev*.

Az Oracle az UNIQUE megszorítás betartatásához egy egyedi indexet hoz létre ahhoz az oszlophoz, amelyhez a felhasználó egyedi kulcsot rendelt.

## PRIMARY KEY megszorítás

Elsődleges kulcsot hoz létre a táblához. Minden táblának csak egy elsődleges kulcsa lehet. Az elsődleges kulcs egy vagy több oszlopból állhat, és az oszlop(ok) értékei egyértelműen azonosítják a tábla sorait. A megszorítás biztosítja, hogy az oszlop, vagy oszlopok minden sorában különböző értékek (mégpedig valódi, azaz kitöltött és nem NULL értékek) legyenek.

A PRIMARY KEY megadható tábla vagy oszlopszinten.

### Példa

Adjunk a *reszleg* tábla *reszlegszam* oszlopának PRIMARY KEY megszorítást.

```
SQL> CREATE TABLE reszleg (
2     reszlegszam    NUMBER(2)    CONSTRAINT reszleg_szam NOT NULL,
3     reszlegnev     VARCHAR2(14),
4     telephely      VARCHAR2(13),
5     CONSTRAINT elsodleges_reszleg_szam PRIMARY KEY (reszlegszam));
```

A tábla létrejött.

Az elsődleges kulcs neve: *elsodleges\_reszleg\_szam*

## FOREIGN KEY megszorítás

Az idegen kulcs, a hivatkozási integritási megszorítás biztosítja, hogy az adatbázisunk ellentmondásmentes legyen. Az idegen kulcs értékei meg kell, hogy egyezzenek a hivatkozott oszlop(ok) értékeinek egyikével. Az idegen kulcs tábla vagy oszlopszinten hozható létre. Összetett idegen kulcs csak táblaszinten hozható létre. Az idegen kulcsot a gyermektáblában hozzuk létre, és hivatkozott oszlopot tartalmazó tábla lesz a szülőtábla.

Az idegen kulcs általános alakja a tábla létrehozásakor:

```
oszlop adattípus [CONSTRAINT megszorítás_név] REFERENCES tábla (oszlop)
[ON DELETE CASCADE]
```

vagy

```
CONSTRAINT megszorítás_név FOREIGN KEY (oszlop [, oszlop] ...)
REFERENCES tábla (oszlop [, oszlop]...) [ON DELETE CASCADE]
```

ahol

```
FOREIGN KEY      táblaszintű definiáláskor azonosítja a
REFERENCES       gyermektábla oszlopát,
                  azonosítja a szülőtáblát és annak oszlopát,
```

ON DELETE CASCADE azt jelzi, hogy a szülőtábla egy sorának törlésekor a rendszer a gyermektábla függő sorait is törölje.  
(Erre példát a CHECK megszorítás végén láthatunk.)

Ha a törlendő oszlopokra vonatkozó megszorításokban a hivatkozott összes oszlopot töröljük, akkor nem szükséges használni a – későbbiekben ismertetendő – CASCADE CONSTRAINTS utasításrészt (tovább gyűrűző megszorítást).

### Példa

Tekintsük példaként az alkalmazott és a reszleg táblát. A reszleg tábla elsődleges kulcsa a reszlegszám oszlopra vonatkozik. Ugyanez a reszlegszám érték szerepel az alkalmazott táblában r\_azon néven. Az r\_azon tudjuk, hogy csak olyan értéket vehet fel, amelyik érték a reszleg tábla reszlegszám oszlopában van. Hogy ezt biztosítsuk, az r\_azon oszlophoz FOREIGN KEY idegen kulcsot rendelünk.

Hozzuk létre az alkalmazott táblát úgy, hogy az r\_azon oszlop idegen kulcs legyen. Az elsődleges kulcs a reszleg tábla reszlegszám oszlopa.

```
SQL> SELECT * FROM alkalmazott;
```

AZONOSITO	NEV	EVES_FIZETES	R_AZON
7369	SMITH	9600	
7876	ADAMS	13200	
7900	JAMES	11400	
7934	MILLER	15600	

reszlegszám és az  
r\_azon legyen azonos.

### A reszleg tábla

Név	Üres?	Típus
RESZLEGSZAM	NOT NULL	NUMBER(2)
RESZLEGNEV		VARCHAR2(14)
TELEPHELY		VARCHAR2(13)

Az alkalmazott táblát létrehozó CREATE parancs az idegen kulccsal együtt (alkalm1.sql):

```
SQL > CREATE TABLE alkalmazott (
2   azonosito      NUMBER(4) NOT NULL,
3   nev            VARCHAR2(10),
4   eves_fizetes   NUMBER(7,2),
5   r_azon         NUMBER(2) NOT NULL,
6   CONSTRAINT alkalmazott_reszleg_kulcs FOREIGN KEY (r_azon)
7   REFERENCES reszleg (reszlegszám) );
```

A tábla létrejött.

Töltsük fel az alkalmazott táblát. Az r\_azon értéke legyen például 8. Tudjuk, hogy a megengedett elsődleges kulcs 10, 20, 30 vagy 40 lehet, hiszen korábban ezeket az értékeket vittük fel a reszleg tábla reszlegszám oszlopába.

```
SQL> INSERT INTO alkalmazott
2   VALUES (4567, 'KELEMEN', 22000, 8);
```

```

INSERT INTO alkalmazott
*
Hiba a(z) 1. sorban:
ORA-02291: integritás megszorítás
(SCOTT.ALKALMAZOTT_RESZLEG_KULCS)
megsértés - a szülő kulcs nem található meg

```

Emlékeztetünk arra, hogy a **FOREIGN KEY** megszorítás neve: `alkalmazott_reszleg_kulcs`.

## CHECK megszorítás

A **CHECK** megszorítás olyan feltételt definiál egy oszlop számára, amelyet az oszlop minden sorának ki kell elégítenie. A feltételekben a **SELECT** utasítás **WHERE** utasításrészében használt szerkezeteket alkalmazhatjuk.

- Nem hivatkozhatunk pszeudooszlopra (`ROWNUM`, `ROWID`, stb).
- Nem hivatkozhatunk a `sysdate`, `UID`, `USER` stb. függvényekre.

Egy oszlop definíciójában több **CHECK** megszorítás is megengedett, számuk nincs korlátozva.

A **CHECK** megszorítást megadhatjuk oszlop és táblaszinten.

**CHECK** megszorítás alakja:

```
CONSTRAINT megszorítás_név CHECK (feltétel)
```

### Példa

Adjuk azt a **CHECK** megszorítást az `alkalmazott` tábla azonosító oszlopa számára, hogy az értéke 1000 és 9000 között legyen.

```

SQL> CREATE TABLE alkalmazott (
2     azonosito          NUMBER(4) NOT NULL,
3     nev                VARCHAR2(10),
4     eves_fizetes       NUMBER(7,2),
5     r_azon             NUMBER(2) NOT NULL,
6     CONSTRAINT alkalmazott_reszleg_kulcs FOREIGN KEY (r_azon)
7         REFERENCES reszleg (reszlegszam),
8     CONSTRAINT alkalmazott_check_kulcs CHECK
9         (azonosito BETWEEN 1000 AND 9000));

```

A tábla létrejött.

Vigyünk be adatot (tegyük fel, hogy véletlenül háromjegyű azonosító számot írtunk be):

```

SQL> INSERT INTO alkalmazott
2     VALUES (444, 'KELEMEN', 22000,30);
INSERT INTO alkalmazott
*
Hiba a(z) 1. sorban:
ORA-02290: ellenőrző megszorítás (SCOTT.ALKALMAZOTT_CHECK_KULCS)
megsértése

```

Emlékeztetünk arra, hogy a **CHECK** megszorítás neve: `alkalmazott_check_kulcs`.

**Példa**

A CHECK megszorítás hivatkozhat olyan kifejezésekre is, amelyben a megszorítással rendelkező sor másik oszlopokértékei szerepelnek. Példánk egy hallgatói nyilvántartás, ahol az eredmény a hallgató jegyeinek átlaga. Ezt a vizsgálati feltételt – mint megszorítást – adjuk meg az eredmény oszlopra. (Saját sorának oszlopaira hivatkozás.)

Hozzuk létre az alábbi `diak` táblát.

```
SQL> CREATE TABLE diak
2  (hallg_azon  NUMBER(6),
3  nev          VARCHAR2(15),
4  szul_ev      NUMBER(4) NOT NULL,
5  kemia        NUMBER(1),
6  fizika       NUMBER(1),
7  eredmény     NUMBER(4,2),
8  CONSTRAINT diak_azon_kulcs PRIMARY KEY(hallg_azon),
9  CONSTRAINT ered_check_kulcs CHECK (eredmény = (kemia+fizika)/2));
```

A tábla létrejött.

```
SQL> desc diak;
```

Név	Üres?	Típus
HALLG_AZON	NOT NULL	NUMBER(6)
NEV		VARCHAR2(15)
SZUL_EV	NOT NULL	NUMBER(4)
KEMIA		NUMBER(1)
FIZIKA		NUMBER(1)
EREDMENY		NUMBER(4,2)

Töltsük fel egy sorral.

```
SQL> INSERT INTO diak
2  VALUES (1234, 'KELEMEN', 1970, 4, 3, NULL);
```

```
SQL> SELECT * FROM diak;
```

HALLG_AZON	NEV	SZUL_EV	KEMIA	FIZIKA	EREDMENY
1234	KELEMEN	1970	4	3	

Módosítsuk a tábla sorát úgy, hogy megadjuk az eredményt. Legyen ez 6.1 ami nem egyenlő a két mező átlagával (vagyis a megszorításban megadott feltétellel).

```
SQL> UPDATE diak
2  SET eredmény = 6.1
3  WHERE hallg_azon = 1234;
UPDATE diak
*
```

Hiba a(z) 1. sorban:

ORA-02290: ellenőrző megszorítás (SYSTEM.ERED\_CHECK\_KULCS) megsértése

Most adjuk meg a helyes eredményt.

```
SQL> UPDATE diak
2     SET eredmény = 3.5
3     WHERE hallg_azon = 1234;
```

1 sor módosítva.

```
SQL> SELECT * FROM diak;
```

HALLG_AZON	NEV	SZUL_EV	KEMIA	FIZIKA	EREDMENY
1234	KELEMEN	1970	4	3	3.5

Hozzunk létre vele összefüggésben egy másik táblát, ennek neve legyen adat. Ebben van egy idegen kulcs a hallgató azonosító oszlopra, CASCADE opcióval. Ez azt jelenti, hogy ha a szülőtábla egy sorát kitöröljük, a gyermektábla vele összefüggésben lévő sora is törlődik.

```
CREATE TABLE adat
(hallg_azon      NUMBER(6),
 anyja_neve     VARCHAR2(15),
 lakhely        VARCHAR2(20),
 irányít       NUMBER(4),
 CONSTRAINT idegen_azon FOREIGN KEY (hallg_azon)
 REFERENCES diak (hallg_azon) ON DELETE CASCADE)
```

A két tábla adatai:

```
SQL> SELECT * FROM diak;
```

HALLG_AZON	NEV	SZUL_EV	KEMIA	FIZIKA	EREDMENY
1234	KELEMEN	1970	4	3	3.5
2345	MOLNÁR	1979	5	3	4

```
SQL> SELECT * FROM adat;
```

HALLG_AZON	ANYJA_NEVE	LAKHELY	IRANYIT
1234	KIS ROZÁL	BUDAPEST	1111
2345	HALÁPOVICS KATA	DEBRECEN	3024

Töröljük a szülőtábla egyik sorát:

```
SQL> DELETE FROM diak
2     WHERE hallg_azon = 1234;
```

1 sor törölve.

Nézzük meg, kitörölte-e a sort mindkét táblából.

```
SQL> SELECT * FROM diak;
```

HALLG_AZON	NEV	SZUL_EV	KEMIA	FIZIKA	EREDMENY
2345	MOLNÁR	1979	5	3	4

```
SQL> SELECT * FROM adat;
```

HALLG_AZON	ANYJA_NEVE	LAKHELY	IRANYIT
2345	HALÁPOVICS KATA	DEBRECEN	3024

A megszorítás tovább gyűrűzött, és a gyermektáblából (adat) is kitörölte a sort.

## Megszorítások lekérdezése

A megszorításokat az adatszótár nézeteiből kérdezhetjük le. A megszorítások nevét és típusát a USER\_CONSTRAINTS adatszótár nézetből, míg az oszlopokat a USER\_CON\_COLUMNS adatszótár nézetből kérdezhetjük le.

A lekérdezések általános alakja:

```
SELECT constraint_name, constraint_type
FROM user_constraints
WHERE table_name = 'TÁBLANÉV';
```

illetve

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'TÁBLANÉV';
```

Kérdezzük le e megszorításokat:

```
SQL> SELECT constraint_name, constraint_type
2 FROM user_constraints
3 WHERE table_name = 'RESZLEG';
```

CONSTRAINT_NAME	C
RESZLEG_SZAM	C
ELSODLEGES_RESZLEG_SZAM	P

ahol a *constraint\_type* a "c" oszlop, a kulcs típusa:

P	PRIMARY KEY
U	UNIQUE
C	CHECK
R	REFERENCES, <i>hivatkozási megszorítás (foreign key)</i>

```
SQL> SELECT constraint_name, constraint_type
2 FROM user_constraints
3 WHERE table_name = 'ALKALMAZOTT';
```

CONSTRAINT_NAME	C
SYS_C001107	C
SYS_C001108	C

```

ALKALMAZOTT_CHECK_KULCS      C
ALKALMAZOTT_RESZLEG_KULCS    R

```

Kérdezzük le az egész nézetet, ám előtte állítsuk a `linesize` sorhosszt minimum 400-ra, és figyeljük meg a lekérdezés eredményét, a gördítősáv használatával.

```
SQL> SELECT * FROM user_constraints;
```

Kérdezzük le az oszlopokra adott megszorításokat, táblánként.

```

SQL> SELECT constraint_name, column_name
2     FROM user_cons_columns
3     WHERE table_name = 'RESZLEG';

```

illetve

```

SQL> SELECT constraint_name, column_name
2     FROM user_cons_columns
3     WHERE table_name = 'ALKALMAZOTT';

```

E megszorítások listái:

CONSTRAINT_NAME -----	CONSTRAINT_NAME -----
COLUMN_NAME -----	COLUMN_NAME -----
ELSODLEGES_RESZLEG_SZAM RESZLEGSZAM	ALKALMAZOTT_CHECK_KULCS AZONOSÍTÓ
RESZLEG_SZAM RESZLEGSZAM	ALKALMAZOTT_RESZLEG_KULCS R_AZON
	SYS_C001107 AZONOSITO
	SYS_C001108 R_AZON

## Megszorítások módosítása

Integritási megszorításokat megadhatunk, törölhetünk, engedélyezhetünk és letilthatunk, de nem módosíthatjuk a megszorítások szerkezetét.

### MEGSZORÍTÁS HOZZÁADÁSA

Ha egy tábla elkészült, utólag is adhatunk megszorításokat a tábla oszlopaira.

Általános alak:

```

ALTER TABLE tábla
ADD [CONSTRAINT megszorítás] típus (oszlop);

```

ahol *tábla* a tábla neve,  
*megszorítás* a megszorítás neve,

*típus*                      a megszorítás típusa,  
*oszlop*                    annak az oszlopnak a neve, amire a megszorítás vonatkozik.

### Példa

Adjunk utólag UNIQUE megszorítást a *reszleg* tábla *telephely* oszlopának.

```
SQL> ALTER TABLE reszleg
2     ADD CONSTRAINT
3         utolag_egyedi UNIQUE (telephely);
```

A tábla módosítva.

A NOT NULL megszorítást a MODIFY utasításrészsel adhatjuk meg. A NOT NULL megszorítással szabályozott oszlopot csak akkor vehetjük fel, ha a táblában nincsenek sorok, mert egy oszlop felvételekor nem adhatjuk meg az oszlop adatait a tábla már létező soraiban.

```
SQL> ALTER TABLE dolgozo
2     MODIFY név CONSTRAINT nem_null NOT NULL;
```

A tábla módosítva.

```
SQL> desc dolgozo;
Név                                Üres?      Típus
-----
AZONOSITO                         NOT NULL  NUMBER(4)
NEV                               NOT NULL  VARCHAR2(15)
EVES_FIZETES                      NUMBER
```

### MEGSZORÍTÁS TÖRLÉSE

A megszorítások törölhetők a táblákból. Mindenképpen tudnunk kell a megszorítás nevét ahhoz, hogy törölni tudjuk.

Az utasítás alakja:

```
ALTER TABLE tábla
    DROP { {PRIMARY KEY | UNIQUE (oszlop [, oszlop]...)} [CASCADE] |
    CONSTRAINT megszorítás }
```

ahol az *oszlop*                      annak az oszlopnak a neve, amelynek megszorítását törölni akarjuk,  
    CASCADE opció            azt jelenti, hogy az összes törlendő megszorítástól függő minden megszorítást töröl.

### Példa

Töröljük az utólag létrehozott UNIQUE kulcsot a *reszleg* táblából. Ha nem tudjuk, kérdezzük le az adatszótárbeli nézetekből.

```
SQL> ALTER TABLE reszleg
2     DROP CONSTRAINT utolag_egyedi
```

A tábla módosítva.

## MEGSZORÍTÁS TILTÁSA

Az integritási megszorítást, kulcsokat ideiglenesen letilthatjuk a `DISABLE` utasításrésszel. Ideiglenesen letilthatjuk a `CASCADE` opcióval kijelölt, másik táblára ható (tovagyűrűző) integritási megszorítást.

A megszorítás tiltásának általános alakja:

```
ALTER TABLE
    DISABLE CONSTRAINT megszorítás_név [CASCADE]
```

### Példa

A `diak`, `adat` tábla között van tovaggyűrűző megszorítás. Mint láttuk a `diak` tábla sorát kitorölve, törölődött az `adat` tábla ugyanolyan `hallg_azon` számú sora.

Tiltsuk le e tovaggyűrűzést.

```
SQL> ALTER TABLE adat
    2     DISABLE CONSTRAINT idegen_azon CASCADE;
```

A tábla módosítva.

```
SQL> DELETE FROM diak
    2     WHERE hallg_azon = 1234;
```

1 sor törölve.

Nézzük meg most az `adat` táblát.

```
SQL> SELECT * FROM adat;
```

HALLG_AZON	ANYJA_NEVE	LAKHELY	IRANYIT
2345	HALÁPOVICS KATA	DEBRECEN	3024
1234	KIS ROZÁL	BUDAPEST	1111

Mivel a `CASCADE` hatását ideiglenesen felfüggesztettük, az `adat` gyermektábla függése megszűnt a `diak` táblától (tehát az `adat` tábla már nem gyermektábla a `HALLG_AZON` oszlopra vonatkozóan), és így az 1234 azonosítójú sor nem törölődött.

## MEGSZORÍTÁS ENGEDÉLYEZÉSE

A tábla definíciójában megadott, de letiltott megszorításokat engedélyezhetjük az `ENABLE` utasításrésszel. Egyedi vagy elsődleges kulcs bekapcsolásakor egyedi index kerül a táblához.

A megszorítás engedélyezésének általános alakja:

```
ALTER TABLE tábla
    ENABLE CONSTRAINT megszorítás_név;
```

### Példa

Állítsuk vissza a letiltott `FOREIGN KEY` megszorítást az `adat` tábla `hallg_azon` oszlopára.

```
SQL> ALTER TABLE adat
    2     DISABLE CONSTRAINT idegen_azon ;
```

A tábla módosítva.

```
SQL> ALTER TABLE adat
2     ENABLE CONSTRAINT idegen_azon ;
```

A tábla módosítva.

## TOVAGYÚRÚZÓ MEGSZORÍTÁSOK

A `CASCADE CONSTRAINTS` utasításrész hatására a rendszer törli az összes olyan integritási megszorítást, amely a törlésre kerülő oszlopok elsődleges kulcsára hivatkoznak.

Ha olyan oszlopot szeretnénk törölni, amin `PRIMARY KEY` megszorítás van, akkor a `CASCADE CONSTRAINT` utasításrész megadásával az oszlop mégis törölhető.

Az utasítás általános alakja (csak a 8i verziótól érvényes):

```
ALTER TABLE tábla
DROP (oszlop) CASCADE CONSTRAINT;
```

```
SQL> ALTER TABLE reszleg
2     DROP (reszlegszam) CASCADE CONSTRAINTS;
```

A tábla módosítva.

```
SQL> desc reszleg;
Név                Üres?      Típus
-----
RESZLEGNEV          VARCHAR2 (14)
TELEPHELY           VARCHAR2 (13)
```

A `reszlegszam` oszlopot töröltük.

## Nézettáblák

A következő tárgyalásra kerülő objektum a nézettábla. Az SQL lehetőséget ad arra, hogy egy, vagy több, valamely adatbázis részét alkotó, úgynevezett fizikai adattáblából egy logikai táblát, úgynevezett nézettáblát (vagy más néven "view"-t, virtuális táblát) hozzunk létre. A nézettábla egy olyan objektum, amely maga fizikailag nem tárol adatokat, csak egy kiválogató-lekérdező utasítást (egy `SELECT` utasítást), amelynek segítségével fizikai, vagy más, már létező nézettáblákra hivatkozik. A nézettábla által hivatkozott táblákat *alaptábláknak* nevezzük.

Gyakorlati jelentőségét az adja, hogy egyrészt a gyakran használt lekérdezéseket segítségével eltárolhatjuk (fizikai tárolása az adatszótárban történik), másrészt összetett lekérdezéseket egyszerűbbekre tudunk visszavezetni (tehermentesítve így a gondolkodást). Fontos tulajdonsága, hogy mivel ő maga nem tárol fizikailag adatot, így az adattáblák adatainak módosításakor a "tartalma" automatikusan frissítődik.

Használata hasonló az adattáblákhoz. Mivel egy `SELECT` utasítás `FROM` részében ugyanúgy lehet rá hivatkozni, mint egy fizikailag létező adattáblára, másrészt egy `INSERT`, vagy egy `UPDATE` utasítással majdnem (!) ugyanúgy lehet adatot bevinni, vagy módosítani nézettáblában, mint adattáblában. Természetesen az adatbevitel fizikailag az (esetleg többszörös nézettábla-hivatkozáson elért) adattáblá(k)ban történik, ám a "majdnem" szócska jelzi, hogy azért ennek korlátai vannak.

Nyilván nem lehet nézetablán keresztül olyan adattáblába sort beszúrní, amelynek valamely `NOT NULL` megszorítással ellátott oszlopát a nézetábla nem tartalmazza (hiszen akkor az az oszlop nem kapna értéket az új rekordban, ami a `NOT NULL` megszorítás által tiltott).

Továbbá az is nyilvánvaló, hogy nem lehet a nézetábla pszeúdóoszlopainak értéket adni (tehát olyan nézetáblabeli oszlopoknak, amelyekhez nem tartozik fizikai oszlop – azaz valamely adattábla oszlopa –, hanem csak valamilyen számított érték, oszlopkifejezés). Az a kérdés persze, hogy egy nézetábla mely oszlopa pszeúdóoszlop és melyik nem, esetenként alapos vizsgálatot igényel, hiszen a láncszerű nézetáblahivatkozások miatt a lánc mentén esetleg minden nézetáblát ellenőrizni kell.

Végül nem lehet egy nézetábla oszlopainak értéket adni, ha a nézetábla-létrehozó utasításban ezt letiltottuk. Ennek révén a fizikai adattáblák megvédhetők egy felhasználói beavatkozástól.

Lényeges eltérés viszont a nézetáblák és az adattáblák között, hogy egyrészt a nézetábla struktúrája egyszerűen bővíthető (pszeúdóoszloppal), másrészt "struktúrájának bővítését" korlátozzák az általa hivatkozott adattáblák struktúrái (nem lehet valódi adatoszloppal bővíteni).

Nézetáblát készíthetünk, ha nincs jogunk az egész adattábla használatához, csak egy részhalmazához, vagy éppen fordítva, az adattábla egy részét el szeretnénk rejteni más felhasználó elől.

Mint említettük az Oracle a nézetáblákat fizikailag eltárolja az adatszótárban. Ezek tulajdonságai lekérdezhetők a `USER_VIEWS`, `USER_CATALOG`, `ALL_VIEWS`, stb. rendszer-nézetáblákból.

Foglaljuk össze, miért használunk nézetáblát:

- Segítségükkel a gyakori lekérdezések eltárolhatók.
- Használatukkal a felhasználó egyszerű (származtatott) lekérdezésekből építhet fel bonyolult lekérdezéseket.
- Nézetekkel korlátozhatjuk az adathoz való hozzáférést, mert egy nézetben csak a benne hivatkozott adattáblák oszlopait látjuk.
- Nézetek segítségével adatfüggetlenség biztosítható a felhasználók és az alkalmazói programok számára, mivel a nézetáblákkal azok csak az adattáblák engedélyezett oszlopai láthatók, és esetleg azokat is csak olvasható módon (megfelelő beállítás esetén).

Egy nézetáblát *egyszerűnek* nevezünk, ha csak egy táblából származik, nem tartalmaz pszeúdóoszlopot (azaz egysoros, vagy többsoros függvényt, oszlopkifejezést), és nem tartalmaz csoportképzést. Egyébként a nézetáblát *összetettnek* nevezzük. Az adatmódosító (a DML utasítások) csak egyszerű nézetáblákon hajthatók végre (persze ezeken sem mindig).

## Nézet létrehozása

Nézetet allekérdezés segítségével hozunk létre. Ez a beágyazott lekérdezés nem tartalmazhat `ORDER BY` utasításrészt.

Az utasítás alakja:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW nézet_név
    [(másodlagos név [, másodlagos név]...)]
AS allekérdezés
    [ WITH READ ONLY | WITH CHECK OPTION [CONSTRAINT megszorítás] ];
```

ahol

OR REPLACE	Újra létrehozza a nézetet, ha ilyen néven már létezett.
FORCE	A rendszer nem ellenőrzi, hogy az alaptábla létezik-e, a nézetet viszont létrehozza. A felhasználó így, alaptábla nélkül a nézetet nem tudja használni.
NOFORCE	A nézetet csak akkor hozza létre, ha az alaptáblák mindegyike létezik (alapértelmezés).
<i>nézet_név</i>	A felhasználó által megadott név.
<i>másodlagos név</i>	Másodlagos neveket adhatunk meg a nézetet definiáló allekérdezés által visszaadott oszlopoknak, kifejezéseknek. Ha nem adunk meg másodlagos neveket, akkor a nézet oszlopai öröklik az allekérdezés oszlopneveit. A lekérdezésben megadott másodlagos név is használható a nézet oszlopnevének megváltoztatására.
AS <i>allekérdezés</i>	Azonosítja azokat az oszlopokat, amelyek a nézetben szerepelni fognak. Az allekérdezés egyszerű vagy összetett, több egymásba ágyazott lekérdezési blokkból állhat. Az allekérdezés WHERE feltétel utasításrésze választja ki a nézetben szereplő sorokat.
WITH READ ONLY	A nézetben csak lekérdezéseket hajthatunk végre, adatmanipulációt (DML utasításokat) nem.
WITH CHECK OPTION	használatával lehetőségünk van a hivatkozási integritás ellenőrzésére is. Az utasításrész nem engedi, hogy nézetben keresztül végrehajtott INSERT, UPDATE utasítások olyan sorokat hozzanak létre, amelyeket a nézet nem választhat ki. A <i>megszorítás</i> használható megszorítások nevének megadására, és az adatérvényesség ellenőrzésére is.

## Példa

Hozzunk létre egy nézetet az emp tábla manager-eiből.

```
SQL> CREATE VIEW manager
2 AS
3 SELECT empno AS azonosito,
4        ename AS nev,
5        sal AS fizetes
6 FROM emp
7 WHERE job = 'MANAGER';
```

A nézet létrejött.

```
SQL> CREATE OR REPLACE VIEW manager
2 (azon_szam, nev, eves_fizetes)
3 AS
4 SELECT empno, ename, sal*12
5 FROM emp
6 WHERE job = 'MANAGER';
```

A nézet létrejött.

Kérdezzük le

```
SQL> SELECT * FROM manager;
```

AZONOSITO	NEV	FIZETES
7566	JONES	2975

Kérdezzük le

```
SQL> SELECT * FROM manager;
```

AZON_SZAM	NEV	EVES_FIZETES
7566	JONES	35700
7698	BLAKE	34200

7698 BLAKE	2850	7782 CLARK	29400
7782 CLARK	2450		

Az `OR REPLACE` paraméterrel megváltoztathatjuk a nézet definícióját anélkül, hogy törölnénk és újra felépítenénk azt. Ebben az esetben a nézetre vonatkozó jogosultságok is megmaradnak.

Nézetek lekérdezése ugyanúgy történik, mint a táblák lekérdezése.

### Példa

Listázzuk ki azokat a `manager`-eket, akiknek éves fizetése kisebb, mint 35000.

```
SQL> SELECT * FROM manager
2 WHERE éves_fizetes < 35000;
```

AZON_SZAM	NEV	EVES_FIZETES
7698	BLAKE	34200
7782	CLARK	29400

### Kérdezzük le az adatszótárból

```
SELECT * FROM user_views;
```

...

VIEW_TYPE	OWNER	VIEW_TYPE
MANAGER	72	SELECT empno , ename , sal*12 FROM emp WHERE job = 'MANAGER'

...

Az adatszótárban a `SELECT` utasítást az adatszótár nézetében egy `long` típusú oszlopban tárolja.

### Példa (Összetett nézet létrehozására)

Hozunk létre egy olyan nézetet, amely az alkalmazottak telephelyenkénti csoportjainak maximális és átlagfizetését tartalmazza. Ehhez a lekérdezéshez két tábla, csoportfüggvény szükséges. Az `emp` tábla másodlagos neve `e`, a `dept` tábla másodlagos neve `d` (`s060.sql`).

```
SQL> CREATE VIEW osszetett_nezet (telephely, max_fizetes, atlag_fizetes)
2 AS
3 SELECT d.loc, MAX(e.sal), ROUND(AVG(e.sal))
4 FROM dept d, emp e
5 WHERE d.deptno = e.deptno
6 GROUP BY d.loc;
```

A nézet létrejött.

```
SQL> SELECT * FROM osszetett_nezet ;
```

TELEPHELY	MAX_FIZETES	ATLAG_FIZETES
CHICAGO	2850	1567
DALLAS	3000	2175
NEW YORK	5000	2917

Szűkítsük a nézet csoportjainak körét:

```
SQL> CREATE OR REPLACE VIEW osszetett_nezet
2      (telephely, max_fizetes, atlag_fizetes)
3      AS
4      SELECT d.loc, MAX(e.sal), ROUND(AVG(e.sal))
5      FROM dept d, emp e
6      WHERE d.deptno = e.deptno
7      GROUP BY d.loc
8      HAVING AVG(e.sal) > 2000;
```

A nézet létrejött.

```
SQL> SELECT * FROM osszetett_nezet;
```

TELEPHELY	MAX_FIZETES	ATLAG_FIZETES
DALLAS	3000	2175
NEW YORK	5000	2917

### Példa

Listázzuk ki fizetés szerint rendezve azokat a dolgozókat, akiknek fizetése a dallasi és New-York-i telephely átlagfizetését meghaladja. (A lekérdezésben nézettábla legyen.)

Elnevezések: emp: e, az osszetett\_nezet: nez.

```
SQL> SELECT DISTINCT e.ename, e.sal
2      FROM emp e, osszetett_nezet nez
3      WHERE e.sal > ALL nez.atlag_fizetes
4      ORDER BY e.sal;
```

ENAME	SAL
CLARK	2450
BLAKE	2850
JONES	2975
FORD	3000
SCOTT	3000
KING	5000

### DML műveletek elvégzése, tiltása nézeten keresztül

Az egyszerű nézeteken keresztül bizonyos szabályok betartásával DML műveleteket végezhetünk. Módosíthatunk, törölhetünk, beszúrhatunk sorokat, ha a nézet nem tartalmazza a következő utasításrészeket:

- csoportfüggvényeket,
- GROUP BY utasításrészt,
- DISTINCT paramétert,
- ROWNUM pszeudooszlopot,
- általában nem tartalmazhat oszlopkifejezéseket (mint például 12\* sal).

A módosítás nézetén keresztül nem megengedett (ezeken kívül) akkor sem, ha az alaptábla tartalmaz olyan alapértelmezett érték nélküli, NOT NULL megszorítással ellátott oszlopot, amelyet a nézet nem.

Ha a nézetet WITH READ ONLY paraméterrel hoztuk létre, akkor ezzel megtiltottuk DML műveletek elvégzését.

### Példa

```
SQL> INSERT INTO manager
      2 VALUES (4567, 'NÉZET IDA', 3450);
INSERT INTO manager
      *
```

Hiba a(z) 1. sorban:  
ORA-01400: NULL ide nem szúrható be: ("SCOTT"."EMP"."DEPTNO")

A deptno oszlopnak NOT NULL megszorítása van, a nézetben pedig nem szerepel a deptno, ezért ez a művelet nem hajtható végre

### Nézet törlése

Egy nézet törlése nem vezet adatvesztéshez, mert a nézet nem tartalmaz adatokat. A nézetet DROP VIEW utasítással törölheti a tulajdonos, illetve, aki DROP ANY VIEW jogokkal rendelkezik.

Általános alakja:

```
DROP VIEW nézet;
```

### Példa

Töröljük a manager nézetet.

```
SQL> DROP VIEW manager;
```

A nézet eldobva.

Ezután a manager nézet már nem található az adatszótárban.

### INLINE nézet

INLINE nézetnek egy másodlagos névvel rendelkező allekérdezést nevezünk. Leggyakrabban a SELECT utasítás FROM utasításrészében van, de előfordulhat a szelekciós lista oszlopnév mezőiben is. Az INLINE nézet egy adatforrást definiál a SELECT utasítás számára. Az INLINE nézet nem egy önálló objektum.

### Példa

Listázzuk ki azokat az alkalmazottakat, akiknek fizetése kisebb, mint a részlegük átlagfizetése (s061.sql).

Elnevezések: emp tábla: e, az allekérdezés másodlagos neve: at.

```
SQL> SELECT e.ename, e.sal, at.deptno, at.átlag_fizetés
      2 FROM emp e,
      3      (SELECT deptno, ROUND(AVG(sal)) AS átlag_fizetés
```

```

4          FROM emp
5          GROUP BY deptno) at
6  WHERE e.deptno = at.deptno
7          AND e.sal < at.átlag_fizetés;

```

ENAME	SAL	DEPTNO	ÁTLAG_FIZETÉS
CLARK	2450	10	2917
MILLER	1300	10	2917
SMITH	800	20	2175
ADAMS	1100	20	2175
MARTIN	1250	30	1567
JAMES	950	30	1567
TURNER	1500	30	1567
WARD	1250	30	1567

### Példa

Írassuk ki a 20-as részleg dolgozóinak nevét, munkakörét, fizetését és részlegük átlagfizetését, ám ezúttal az inline nézet a szelekciós lista oszlopnév mezőiben legyen. (s061\_2.sql)

```

SQL> SELECT ename as név, job as foglalk, sal as fizetés,
2         (SELECT AVG(sal)
3         FROM emp
4         WHERE deptno = 20) AS átlag_fizetés
5 FROM emp
6 WHERE deptno =20

```

NÉV	FOGLALK	FIZETÉS	ÁTLAG_FIZETÉS
SMITH	CLERK	800	2175
JONES	MANAGER	2975	2175
SCOTT	ANALYST	3000	2175
ADAMS	CLERK	1100	2175
FORD	ANALYST	3000	2175

### FELSŐ-N ANALÍZIS

A **FELSŐ-N** lekérdezést akkor alkalmazunk, amikor valamilyen feltétel alapján úgynevezett top-listát akarunk készíteni, vagyis az alaptábla valamely oszlopában szereplő  $n$  legnagyobb vagy legkisebb értéket keressük, és ennek megfelelő sorokat szeretnénk kilistázni.

A **FELSŐ-N** analízis tehát egy adott oszlop értékei közül az  $n$  legnagyobb, illetve legkisebb értéket adja vissza, ahol  $n$  egy természetes szám.

Az utasítás egy **INLINE** nézetből áll, amely előállítja az adatok rendezett listáját. A legnagyobb érték kikereséséhez a **DESC** paraméter használata szükséges.

A külső lekérdezés korlátozza a megjelenő sorok számát éppen  $N$ -re. Ebben a lekérdezésben szerepel a **ROWNUM** pszeudooszlop, amely 1-től kezdve számozza a visszaadott sorokat, és a **WHERE** utasításrész, amely korlátozza a visszaadott sorok számát.

Az utasítás általános alakja:

```
SELECT [oszloplista], ROWNUM
FROM (SELECT [oszloplista]
      FROM tábla
      ORDER BY TOP_N_oszlop)
WHERE ROWNUM <= N;
```

ahol

<i>oszloplista</i>	a listázandó oszlopok
<i>TOP_N_oszlop</i>	az az oszlop, amely szerinti toplistát készítjük
ROWNUM	pszeudooszlop, tartalmazza azt a sorszámot, ahányadiknak a sort a rendszer a táblából kiválasztotta.

### Példa

Listázzuk ki a négy, legrégebben a cégnél lévő alkalmazottat. (`s062.sql`)

```
SQL> SELECT ROWNUM AS sorszám, al.név, al.belépés
2      FROM (SELECT ename AS név, hiredate AS belépés
3              FROM emp
4              ORDER BY hiredate ) al
5      WHERE ROWNUM <= 4
```

SORSZÁM	NÉV	BELÉPÉS
1	SMITH	80-DEC-17
2	ALLEN	81-FEB-20
3	WARD	81-FEB-22
4	JONES	81-ÁPR-02

### Példa

Listázzuk ki a 3 legjobban kereső dolgozót. (`s062.sql`).

```
SQL> SELECT ROWNUM AS sorszám, al.név, al.fizetés
2      FROM (SELECT ename AS név, sal AS fizetés
3              FROM emp
4              ORDER BY sal DESC) al
5      WHERE ROWNUM <= 3
```

SORSZÁM	NÉV	FIZETÉS
1	KING	5000
2	SCOTT	3000
3	FORD	3000

### Megjegyzés

A FELSŐ-N analízis csak a 8i verziótól kezdve működik.

## INDEX

Az Oracle rendszer indexe szintén egy felhasználói objektum, amely egy mutató használatával felgyorsítja a sorok visszakeresését.

Az indexet létrehozhatjuk explicit módon, vagy hagyhatjuk, hogy a rendszer automatikusan hozza létre azokat.

Mikor van létrehozására szükség?

Van például egy több ezer soros táblánk, amelynek egyik oszlopa dátum (belépési, születési, elkészítési, stb.). Olyan lekérdezéseket szeretnénk végrehajtani rendszeresen, amelyeknek feltétele nagyon sok esetben bizonyos dátum vagy rövid időszak. Ebben az esetben, ha nincs index a dátum oszlophoz, minden lekérdezés esetén végigvizsgálja a rendszer a sorok kereséséhez az egész táblát. Ez bizony igen lassan megy. Ilyen esetekben célszerű a dátum oszlophoz indexet rendelni, amely alapján sokkal gyorsabb kiszűrni a feltételben szereplő időtartománynak, vagy bizonyos évnél megfelelő sorokat.

Az index közvetlen és gyors hozzáférést biztosít a tábla soraihoz. Használatával csökken a lemezműveletek száma, mert közvetlenül kijelöli a keresett adat helyét. Az Oracle rendszer automatikusan használja és karbantartja az indexeket. Az index létrehozása után már nem kell a felhasználónak vele foglalkoznia.

Az indexek bármikor létrehozhatók és törölhetők, nincs hatása az alaptáblára.

Tábla eldobásakor (törlésekor) a hozzá tartozó indexek is törlődnek.

## Index létrehozása

Index létrehozása kétféleképpen történhet:

- automatikusan,
- felhasználó által.

Automatikusan a rendszer létrehoz indexet, ha a tábla valamelyik oszlopára `PRIMARY KEY`, vagy `UNIQUE` megszorítást állítottunk be. Az index neve ilyenkor a megszorítás nevével egyezik meg.

*Felhasználói index:*

Ez nem egyedi index. Létrehozhatunk például egy `FOREIGN KEY` oszlopindexet egy lekérdezésben megadott összekapcsoláshoz, s ezzel gyorsíthatjuk az adatok lekérdezését.

Indexet egy vagy több oszlopra is létrehozhatunk.

Az utasítás általános alakja:

```
CREATE INDEX index
ON tábla (oszlop [, oszlop]...);
```

ahol *index* az index neve,

*tábla* a tábla neve,

*oszlop* a tábla indexelni kívánt oszlopainak neve.

Mikor célszerű indexet használni?

- Ha az oszlopot gyakran használjuk `WHERE` feltételben vagy összekapcsolásban.
- Ha az oszlop értékei széles tartományba esnek.
- Ha az oszlopban sok `NULL` érték szerepel.
- Ha a tábla igen nagy, és a lekérdezések többsége csak a sorok legfeljebb 2-4 %-át adja vissza.

Sok index létrehozása nem feltétlenül gyorsítja a keresést. A rendszernek az indexelt táblákon végrehajtott minden egyes DML utasítás után frissítenie kell az indexeket.

Nagyon megfontolandó a felhasználói index létrehozása!

**Példa**

Hozzunk létre indexet az emp tábla ename oszlopához.

```
SQL> CREATE INDEX nev_ind
2      ON emp(ename);
```

Az index létrejött.

Az indexek lekérdezhetők az adatszótárbeli nézetekből, ezek: USER\_INDEX, vagy USER\_IND\_COLUMNS.

Kérdezzük le az indexeket az emp táblából!

```
SQL> SELECT ic.index_name, ic.column_name,
2          ic.column_position col_pos, ix.uniqueness
3      FROM user_indexes ix, user_ind_columns ic
4      WHERE ic.index_name = ix.index_name
5            AND ic.table_name = 'EMP'
```

INDEX_NAME	COLUMN_NAME	COL_POS	UNIQUENES
EMP_PRIMARY_KEY	EMPNO	1	UNIQUE
NEV_IND	ENAME	1	NONUNIQUE

Kérdezzük le az összes indexet

```
SQL> SELECT *
      FROM user_indexes;
```

A lista egy részlete:

INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME
...			
NEV_IND	NORMAL	SCOTT	EMP
DEPT_PRIMARY_KEY	NORMAL	SCOTT	DEPT

**Indexek törlése**

Az index törölhető az adatszótárból.

Az utasítás alakja:

```
DROP INDEX index_név
```

**Példa**

Töröljük a nev\_ind indexet, amelyet az emp tábla ename oszlopához hoztunk létre.

```
SQL> DROP INDEX nev_ind;
```

Az index eldobva.

Már nem találjuk meg az adatszótárban.

## Szinonima

A szinoníma helyettesítő név. Mikor lehet erre szükség?

- Más felhasználó tábláira minősítő név.táblanévvel hivatkozunk. Ha szinonimát adunk egy másik felhasználó táblájának, akkor minősítő név nélkül hivatkozhatunk a szinonimával a táblára.
- Jelentős szerepe van még a hosszú nevek helyettesítésénél is, lerövidíthetjük vele ezekre való hivatkozásokat (Gyakran lesz hosszú neve a nézeteknek).

A szinonima *létrehozásának* általános alakja:

```
CREATE [PUBLIC] SYNONYM szinonima_név
FOR objektum;
```

ahol PUBLIC minden felhasználó számára elérhető szinonimát hoz létre,  
 szinonima\_név a létrehozandó szinonima, helyettesítő név,  
 objektum az az objektum (például tábla, nézet stb.), amelyhez a szinonimát létrehozzuk.

A saját magunk által létrehozott szinonima neve nem egyezhet meg egyetlen saját tulajdonú objektumunk nevével sem.

### Példa

A system felhasználónévvel léptünk be, és ott dolgozunk. A scott felhasználó tábláit szeretnénk felhasználni. Ezeket mindig a scott.táblanév minősített hivatkozással kérdezhetjük le. Rendeljünk szinonimát a dept táblához.

```
SQL> SELECT * FROM scott.dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
SQL> SELECT * FROM reszlegsins;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
SQL> CREATE SYNONYM reszlegsins
2 FOR scott.dept;
```

A szinoníma létrejött.

A szinonimák is lekérdezhetők az adatszótár nézeteiből.

Ezek : USER\_SYNONYMS, ALL\_SYNONYMS

Kérdezzük le:

```
SQL> SELECT * FROM user_synonyms;
```

SYNONYM_NAME	TABLE_OWNER	TABLE_NAME
CATALOG	SYS	CATALOG
COL	SYS	COL
RESZLEGSIN	SCOTT	DEPT

PRODUCT_USER_PROFILE	SYSTEM	SQLPLUS_PRODUCT_PROFILE
PUBLICSYN	SYS	PUBLICSYN
SYSCATALOG	SYS	SYSCATALOG
SYSFILES	SYS	SYSFILES
TAB	SYS	TAB
TABQUOTAS	SYS	TABQUOTAS

9 sor kijelölve.

A szinonima ugyanúgy törölhető, mint a többi objektum. A törlési utasításának alakja:

```
DROP SYNONYM szinonima_név;
```

### **Példa**

Töröljük a `reszlegsin` szinonimát.

```
SQL> DROP SYNONYM reszlegsin;
```

A szinonima eldobva.

Az adatszótár nézetéből kitörlődött.

# Az adatvezérlő nyelv (DCL)

## Jogosultságok, privilégium

A mai világban a több felhasználós rendszereknél – mint amilyen az Oracle adatbáziskezelő rendszer is – egyrészt védeni kell az adatbázist, másrészt a felhasználó számára biztosítani kell a munkájához szükséges adat és objektumhozzáférést. Az Oracle több szinten is biztosítja ezt úgy, hogy a felhasználónak jogosultságokat ad.

A jogosultság lehetővé teszi a felhasználónak, hogy bizonyos tevékenységeket (SQL utasításokat) elvégezzen. Csak olyan tevékenységeket tud a felhasználó végezni az Oracle rendszerben, amelyhez jogosultságot kapott.

Jogosultságot az adatbázis adminisztrátor, a DBA (Data Base Administrator) ad. A DBA tulajdonképpen egy kiemelt – a többi fölé rendelt - felhasználó, aki egyéb teendői mellett jogosultságokat adhat az adatbázis objektumok használatára.

A jogosultság lehet:

- rendszer szintű védelem (egyes rendszerkomponensek elérése),
- adatbázis védelem (adatbázishoz való hozzáférés, adatvédelem),
- objektumkezelési (objektumok tartalmának kezelése),
- felhasználói jogok (további jogok átadása más felhasználónak vagy szerepkörnek).

Szerepkör a jogosultságok egy olyan csoportja, amelyet az egyik felhasználó egy másik felhasználónak adhat.

Könyvünk csak a felhasználók számára adott, illetve továbbadható jogokkal foglalkozik.

## Rendszer jogosultságok

Több, mint 80féle különböző rendszerjogosultság adható a felhasználónak, illetve a szerepkörnek. A rendszerjogosultságot az adatbázis adminisztrátor (DBA) állítja be.

Az adatbázis adminisztrátor alapvető jogosultságai:

CREATE USER	felhasználó létrehozása,
DROP USER	felhasználó törlése,
DROP ANY TABLE	bármelyik tábla törlése,
BACKUP ANY TABLE	bármelyik tábla archiválása (biztonsági mentése).

Az adatbázis adminisztrátor a CREATE USER utasítással hozhat létre új felhasználót.

Az utasítás alakja:

```
CREATE USER felhasználó
IDENTIFIED BY jelszó;
```

### Fontos

A felhasználónak ilyenkor még semmilyen jogosultsága sincs!

### Példa

Lépjünk be a `system` felhasználó jelszával, és hozzunk létre egy `hallgato` nevű felhasználót, akinek a jelszava is legyen `hallgato`.

```
SQL> CREATE USER hallgato
      2 IDENTIFIED BY hallgato;
```

A felhasználó létrejött.

## Jelszó megváltoztatása

Az adatbázis adminisztrátor hozza létre a felhasználói bejegyzéseket, és megadja a kezdeti belépési jelszót. Ez a jelszó ismert, ezért a felhasználónak joga van ezt megváltoztatni. A jelszó megváltoztatása az `ALTER USER` utasítással történik.

Az utasítás alakja:

```
ALTER USER felhasználó
      IDENTIFIED új jelszó;
```

## Példa

Módosítsuk a `hallgato` felhasználói jelszót `kiss1`-re.

```
SQL> ALTER USER hallgato
      2 IDENTIFIED BY kiss1;
```

A felhasználó módosítva.

Próbáljunk belépni az új jelszóval.

```
SQL> connect hallgato/kiss1
ERROR:
ORA-01045: a(z) HALLGATO felhasználónak nincs CREATE SESSION jogosultsága;
a belépés visszautasítva
```

Figyelmeztetés: A továbbiakban már nincs hozzákapcsolódva az ORACLE-hez.

Így tehát a `Kiss1` jelszavú hallgató nem férhet hozzá az Oracle adatbázishoz.

Meg kell adni bizonyos rendszerjogosultságokat.

## Felhasználói rendszerjogosultságok

Az adatbázis adminisztrátornak meg kell adnia bizonyos rendszerjogosultságokat a felhasználó számára, hogy az használni tudja a rendszert.

Az adatbázis adminisztrátor a `GRANT` utasítással adhat jogokat a felhasználó számára. Az adható jogok (a teljesség igénye nélkül):

<code>CREATE SESSION</code>	kapcsolódás az adatbázishoz,
<code>CREATE TABLE</code>	tábla létrehozása a felhasználó saját tulajdonában,
<code>CREATE VIEW</code>	nézet létrehozása a felhasználó saját tulajdonában,
<code>CREATE PROCEDURE</code>	tárolt eljárás, függvény vagy csomag létrehozása a felhasználó saját tulajdonában.

A jogosultságok megadási utasításának általános alakja:

```
GRANT jogosultság [, jogosultság] ...
      TO felhasználó [, felhasználó] ...
```

ahol *jogosultság felhasználó* a megadható jogosultságok egyike, az a felhasználó, akinek a DBA a jogosultságokat adja.

### Példa

Adjunk jogot a `hallgato` felhasználónak (jelszava `kiss1`) az adatbázishoz való hozzáféréshez, tábla és nézet létrehozásához.

```
SQL> GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW
      2      TO hallgato;
```

Az engedélyezés sikeresen befejeződött.

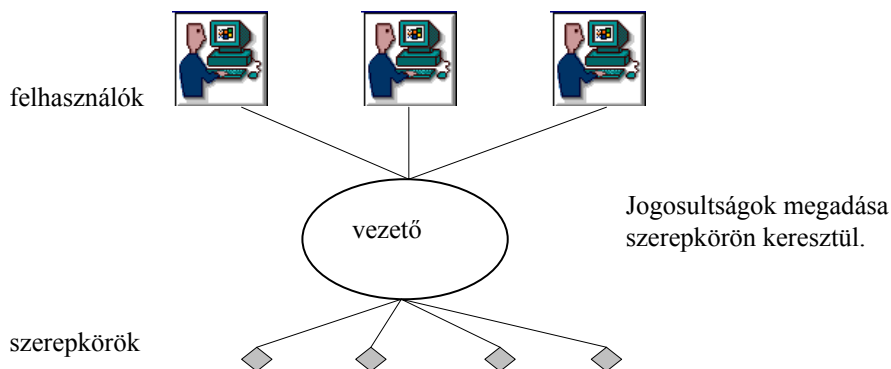
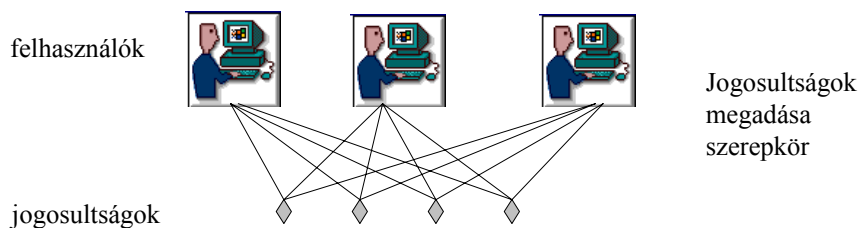
Próbáljon `Kiss1` jelszavú `hallgato` most belépni a saját jelszavával.

```
SQL> connect hallgato/kiss1;
Kapcsolódva.
```

Sikerült. Ezen a területen önállóan dolgozhatunk.

### Szerepkör

A szerepkör a kapcsolódó jogosultságok elnevezett gyűjteménye, amely lehetővé teszi, hogy egyetlen hozzárendeléssel a benne szereplő összes jogot megadjuk a felhasználónak. Ezzel a módszerrel könnyebben kezelhetjük a jogosultságokat, egyszerre több felhasználónak adhatunk, illetve vonhatunk is vissza jogosultságokat.



Szerepkör létrehozása és hozzárendelése (DBA adhatja):

```
CREATE ROLE szerepkör;
```

Jogosultság adása szerepkörnek:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW  
TO szerepkör;
```

Felhasználó hozzárendelése szerepkörhöz:

```
GRANT szerepkör  
TO felhasználó1 [, felhasználó2] ...
```

### Példa

Legyen egy *tanar* nevű szerepkör, adjunk jogokat a szerepkörnek, és rendeljünk hozzá felhasználókat.

```
SQL> CREATE ROLE tanar;
```

A szerepkör létrejött.

```
SQL> GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW  
2 TO tanar;
```

Az engedélyezés sikeresen befejeződött.

Hozzunk létre még három diák felhasználót.

```
SQL> CREATE USER diak1  
2 IDENTIFIED BY diak1;
```

A felhasználó létrejött.

```
SQL> CREATE USER diak2  
2 IDENTIFIED BY diak2;
```

A felhasználó létrejött.

```
SQL> CREATE USER diak3  
2 IDENTIFIED BY diak3
```

A felhasználó létrejött.

Adjon a szerepkör (*tanar*) egyszerre több felhasználónak jogot.

```
SQL> GRANT tanar  
2 TO diak1, diak2, diak3;
```

Az engedélyezés sikeresen befejeződött.

Lépünk be mint *diak2* felhasználó.

```
SQL> connect diak2/diak2  
Kapcsolódva.
```

A jogosultságokat az SQL-hez a felhasználók mindegyike egyszerre megkapta.

## Objektumkezelési jogosultságok

Az objektumkezelési jogosultság egy adott műveletnek valamelyik objektumon való végrehajtási joga. Minden objektumhoz a jogosultságok meghatározott csoportja tartozik.

Az objektum tulajdonosa a saját objektumához minden jogosultsággal rendelkezik.

Az objektum tulajdonosa engedélyezheti más felhasználó számára a saját objektumaira vonatkozó jogosultságokat.

Az objektumkezelési jogosultságok az alábbiak lehetnek:

```
ALTER
DELETE
EXECUTE      futtatási jog
INDEX
REFERENCES
SELECT
UPDATE
```

Az objektumkezelési jogosultság megadásának alakja:

```
GRANT {objektumkezelési_jog | ALL} [(oszlopok)]
    ON objektum
    TO {felhasználó | szerepkör | PUBLIC}
    [WITH GRANT OPTION];
```

ahol

<i>objektumkezelési_jog</i>	megadni kívánt jogosultság
ALL	összes objektumkezelési jogosultság
<i>oszlopok</i>	azoknak a tábláknak vagy nézeteknek az oszlopai, amelyre a jogosultságok megadjuk
TO	az a felhasználó vagy szerepkör, akinek megadjuk a jogosultságot
PUBLIC	minden felhasználó számára megadjuk a jogosultságot
WITH GRANT OPTION	engedélyezzük a felhasználónak, hogy továbbadja a jogosultságokat.

### Példa

A scott felhasználó adjon jogot a hallgato-nak a reszleg tábla lekérdezéséhez.

```
SQL> GRANT SELECT ON reszleg TO hallgato;
Az engedélyezés sikeresen befejeződött.
```

Adja meg a módosítás jogát is, majd ellenőrizzük lekérdezéssel.

```
SQL> GRANT UPDATE
2   ON reszleg
3   TO hallgato;
```

Az engedélyezés sikeresen befejeződött.

```
SQL> CONNECT hallgato/kiss1
Kapcsolódva.
```

```
SQL> SELECT * FROM scott.reszleg;
```

```
SQL> UPDATE scott.reszleg
2   SET telephely = 'DEBRECEN'
3   WHERE reszlegnev = 'GYÁR';
```

1 sor módosítva.

```
SQL> SELECT * FROM scott.reszleg;
```

RESZLEGNEV	TELEPHELY	RESZLEGNEV	TELEPHELY
-----	-----	-----	-----
KÖNYVELŐ	BUDAPEST	KÖNYVELŐ	BUDAPEST
KUTATÓ	GYŐR	KUTATÓ	GYŐR
KERESKEDŐ	HATVAN	KERESKEDŐ	HATVAN
GYÁR	SZEGED	GYÁR	DEBRECEN

## Jogosultságok ellenőrzése

A jogosultságokat is nyilvántartja a rendszer az adatszótár nézeteiben. Ebből mindig megtudhatjuk, milyen jogosultságunk van.

A jogosultságra vonatkozó nézetek:

ROLE_SYS_PRIVS	Szerepköröknek adott rendszerjogosultságok.
ROLE_TAB_PRIVS	Szerepköröknek adott táblajogosultságok.
USER_ROLE_PRIVS	A felhasználó számára elérhető jogosultságok.
USER_TAB_PRIVS_MADE	A felhasználó objektumára adott objektumkezelési jogosultságok.
USER_TAB_PRIVS_RECD	A felhasználónak adott objektumkezelési jogosultságok.
USER_COL_PRIVS_MADE	A felhasználó objektumainak oszlopaira adott objektumkezelési jogosultságok.
USER_COL_PRIVS_RECD	A felhasználó egyes oszlopokra kapott objektumkezelési jogosultságai.

## Példa

Kérdezzük le, milyen jogokat adott a `scott` felhasználó.

```
SQL> CONNECT scott/tiger
Kapcsolódva.
```

```
SQL> SELECT * FROM user_tab_privs_made;
```

GRANTEE	TABLE_NAME	GRANTOR	PRIVILEGE	GRA
-----	-----	-----	-----	----
HALLGATO	RESZLEG	SCOTT	SELECT	NO
HALLGATO	RESZLEG	SCOTT	UPDATE	NO

## Példa

A `system` felhasználó adott szerepkör jogosultságot a `tanar`-nak. Kérdezzük le.

```
SQL> SELECT * FROM role_sys_privs;
```

ROLE	PRIVILEGE	ADM
-----	-----	----
AQ_ADMINISTRATOR_ROLE	DEQUEUE ANY QUEUE	YES
...		
TANAR	CREATE SESSION	NO
TANAR	CREATE TABLE	NO
TANAR	CREATE VIEW	NO

190 sor kijelölve.

## Objektumkezelési jogok visszavonása (REVOKE)

Más felhasználónak adott jogosultságokat a `REVOKE` utasítással vonhatjuk vissza. A `WITH GRANT OPTION` utasításrész használatával a továbbadott jogokat (ha voltak) is visszavonhatjuk.

A `REVOKE` utasítás általános alakja:

```
REVOKE {jogosultság [,jogosultság] ... | ALL }
      ON objektum
      FROM {felhasználó [,felhasználó] ... | szerepkör | PUBLIC}
      [CASCADE CONSTRAINTS];
```

ahol

<i>jogosultság</i>	visszavonni kívánt jogosultság
ALL	összes objektumkezelési jogosultság
<i>oszlopok</i>	azoknak a tábláknak vagy nézeteknek az oszlopai, amelyekre a jogosultságok visszavonjuk
TO	az a felhasználó vagy szerepkör, akitől visszavonjuk a jogosultságot
PUBLIC	minden felhasználótól visszavonjuk a jogosultságot
CASCADE CONSTRAINTS	hatására az objektumra a <code>REFERENCES</code> jogosultság segítségével megadott hivatkozási megszorítások is törlődnek.

### Példa

Vonjuk vissza a `scott` felhasználó által a hallgatónak adott jogosultságokat (`UPDATE`, `SELECT`).

```
SQL> CONNECT scott / tiger
Kapcsolódva.
```

```
SQL> REVOKE UPDATE
      2   ON reszleg
      3   FROM hallgato;
```

```
SQL> REVOKE SELECT
      2   ON reszleg
      3   FROM hallgato;
```

A visszavonás sikeresen befejeződött. A visszavonás sikeresen befejeződött.

## 10. FEJEZET

# PL/SQL nyelv

A PL/SQL nyelvet az Oracle a 6.0 verzió után fejlesztette ki. Ez már egy valódi adatbázis-kezelő nyelv, az SQL procedurális kiterjesztése (PL - Procedural Language).

A PL/SQL nyelvben megtalálhatók a magasszintű programnyelvekre jellemző elemek, így tartalmazznak adattípusokat, változókat, konstansokat. Jellemző a blokkszerkezet és az alprogram használat. Megtalálhatók a nyelv elemei között az értékadó utasítások, a feltételes utasítások, ciklusok, szokásos kivételkezelések és a struktúrált adatok kezelése. Az adattáblák sorainak feldolgozása úgynevezett kurzorokkal történik. További elemei a tárolt alprogramok (eljárások és függvények), a csomagok és a triggerek.

Tárolt alprogramok alatt az adatszótár nézeteiben (az adatbázis-szerveren) tárolt olyan algoritmus-végrehajtó objektumokat értjük, amelyek minden alkalmazás számára elérhetők. A csomagok a PL/SQL elemek összetartozó csoportja, változók, konstansok és kurzorok, előre megírt alprogramok gyűjteménye.

A triggerek pedig események, szabályok szintén előre megírt programja, amely megadott táblához van hozzárendelve, és amelyet az Oracle automatikusan lefuttat, ha az adott táblát megváltoztató (INSERT, UPDATE vagy DELETE) utasítás bekövetkezik.

A legtöbb SQL utasítás minden további nélkül használható a PL/SQL-ben. Kivételt képeznek a DDL és a DCL utasítások (például a CREATE TABLE, ALTER TABLE, DROP TABLE). Ezen utasítások használatát a dinamikus SQL csomag alkalmazása oldja fel.

A PL/SQL nyelv már lehetővé teszi, hogy a 80-as évek modern szoftverfejlesztési eszközeit használjuk. Melyek is ezek?

- Az adatok egységbe zárása (encapsulation),
- a kivételkezelés (exception handling),
- az információ elrejtése (information hiding), és
- általában az objektum orientált megközelítés (object orientation).

Az Oracle hatékony fejlesztői környezettel az Oracle Developerrel is rendelkezik, ennek eszközei az Oracle Forms, az Oracle Reports, és az Oracle Graphics.

Az Oracle Developer alkalmazásokat tartalmazó része megosztott könyvtárakat használ, amely helyileg és távolról is elérhető. A Developer alkalmazások részeként deklarált eljárások és függvények nem azonosak az adatbázisban tárolt eljárásokkal és függvényekkel, amelyekkel mi is foglalkozunk, habár felépítésük azonos. Az Oracle Developer nem témája könyvünknek.

A PL/SQL az SQL adattípusait is használja. A megosztott adattípusok integrálják PL/SQL nyelvet az Oracle rendszer adatszótárával. A kifejlesztett PL/SQL nyelv összeköti

az adatbázistechnológia kényelmes elérését, és a procedurális programozási lehetőségek iránti igényt.

A PL/SQL saját "motorral" rendelkezik, amely független az Oracle rendszer "motorjától". Ezzel a teljesítmény megnövekszik. Miért? Ez a motor kiszűri az SQL utasításokat, amelyet elküld az Oracle rendszer SQL utasítás végrehajtójának. A fennmaradó helyi adatokat a PL/SQL motorban található procedurális utasításvégrehajtó dolgozza fel. A teljesítmény mértéke az aktuális környezettől függően eltérő lehet. A kiszűrt SQL utasításokat egy blokkba csoportosíthatjuk, és a teljes blokkot egyetlen hívással küldhetjük a kiszolgálónak. A PL/SQL nyelv utasításai nem az alkalmazásban (a munkaállomáson), hanem az adatbázisban, a kiszolgáló rendszerben (az adatbázis-szerveren) futnak.

# PL/SQL blokk szerkezete.

## Alapfogalmak

Az SQL\*Plus környezetben a PL/SQL nyelvű programozás alapfogalma a blokk, ahol egy blokk a hagyományos programozásnyelvi terminológiában leginkább a program fogalmának felel meg. Egy PL/SQL blokk sokszor csak egy egyszerű utasítássorozat, amely esetleg egy `SELECT` utasítást vesz körül valamilyen interaktív környezettel annak érdekében, hogy a felhasználó azt az aktuális igényeinek megfelelően tudja paraméterezni. Más esetben azonban egy összetett vezérlési szerkezet, amely egy bonyolult tábla-algoritmust valósít meg. A különböző igényekhez különböző eszközöket biztosít a PL/SQL. A következőkben bemutatjuk a PL/SQL blokkok felépítését és azok típusait.

Egy blokk három funkcionálisan jól elkülöníthető részből, szegmensből állhat: a deklarációs, a végrehajtható és a kivételkezelő részből. A deklarációs rész és a kivételkezelő rész használata opcionális, míg a végrehajtható rész használata kötelező egy blokkban. Az SQL\*Plus környezetben egy blokkot törtvonal (`/`) zár, közvetlenül az `end;` utáni sor elején.

### Deklarációs szegmens

A deklarációs szegmenst a `DECLARE` kulcsszó vezeti be. Ez tartalmazza a blokkban felhasznált összes változót, kurzort, és a felhasználó által definiált kivételeket.

### Végrehajtható szegmens

A `BEGIN` és az `END` kulcsszavak között találhatók az összes olyan SQL utasítástok, amelyek az adatbázis adatainak lekérdezésére, módosítására szolgálnak, illetve a blokk adatainak kezelésére szolgáló PL/SQL utasítások, alprogramhívások, vezérlési szerkezetek.

### Kivételkezelő szegmens

Az `EXCEPTION` kulcsszó vezeti be. A végrehajtandó részben felmerülő hibák, és abnormális állapotok esetén végrehajtandó műveleteket határozza meg. Helye az `END` utasítás előtt van.

### A blokkok szerkezete

A blokk felépítésének általános alakja

```
[<< blokk_név >> ]
[DECLARE
    deklarációk]
BEGIN
    végrehajtható utasítások
[EXCEPTION
    kivételkezelő]
END [blokk_név]
```

A végrehajtandó szegmensbe újabb blokkok ágyazhatók be.

**Szabályok:**

- Az SQL és a PL/SQL utasításokat pontosvesszővel (;) zárjuk le.
- A névtelen blokkot az SQL\*Plus környezetben a törtvonal (/) futtatja le.
- Az SQL\*Plus pufferbe való adatbevitelt egy ponttal (.) zárhatjuk le.  
A PL/SQL blokkot a puffer egyetlen folyamatos utasításnak tekinti. A benne lévő pontosvessző nem zárja le, és nem futtatja a puffert.

**Megjegyzés**

Az utasításokat egy sorba is fűzhetjük, de ugyanúgy, ahogy az SQL `SELECT` utasításánál sem tettük, itt sem javasolt, mert a program érthetlenné, nehezen szerkeszthetővé és nehezen követhetővé válik.

PL/SQL blokkot körülfoghat:

- SQL parancsállomány,
- PL/SQL eljárás (`PROCEDURE`) vagy függvény (`FUNCTION`),
- gazdanyelvi program,
- trigger.

## Blokk típusok

A PL/SQL minden egysége egy vagy több blokkból áll, amelyek különállóak, vagy egymásba ágyazhatók is lehetnek. A blokkok tetszőleges számú beágyazott blokkot tartalmazhatnak. A blokkok lehetnek névtelen blokkok, névvel ellátott blokkok és alprogramok. Alprogramok alatt a PL/SQL nyelvben eljárásokat (`PROCEDURE`) és függvényeket (`FUNCTION`) értünk.

**Blokk**

Egy blokknak lehet neve, de lehet név nélküli is. A blokk a deklarálása helyén hajtódik végre, a benne szereplő utasítások futás közben kerülnek átadásra a PL/SQL motornak. Egy blokk átadható egy gazdanyelv előfordító programjának, a Server Managernek, vagy elhelyezhető egy SQL\*Plus kódba. Ilyen blokkokból állnak az Oracle Developer triggerei is. A blokkot legegyszerűbb interaktív módon használni. A név nélküli blokkot névtelen blokknak nevezzük. Névvel ellátott blokkban a `DECLARE` kulcsszó előtt `<<` és `>>` szimbólumok között egy *címke* áll, ez a blokk neve. A névvel ellátott blokkot `end` *címke*; utasítással zárjuk le.

**Alprogramok**

Az alprogramok névvel ellátott, paraméterezhető PL/SQL blokkok, amelyek lehetnek eljárások és függvények. Nevükkel és paraméterükkel bármikor aktivizálhatók. Általában eljárást használunk műveletek sorozatának elvégzéséhez, függvényt pedig egy-egy érték kiszámításához. A függvény az eljárással ellentétben visszaad egy értéket. Alprogramok bármely PL/SQL blokkban vagy csomagban létrehozhatóak. Az alprogramok részei a fej, és a törzs, ahol a törzs részei a már említett deklarációs, végrehajtható és kivételkezelő rész.

Az alprogramok előnyei:

- modularitás,
- struktúrálhatóság,
- újrafelhasználhatóság,

- absztrakció.

A témáról részletesen az "Alprogramok" című alfejezetben lesz szó.

## Program típusok

Megemlítjük az Oracle-ben előforduló program típusokat:

- Névtelen vagy nevesített blokk Névvvel nem rendelkező, vagy névvel ellátott PL/SQL blokk. Beágyazható alkalmazásba, vagy elindítható interaktív módon. Használható minden PL/SQL környezetben.
- Tárolt eljárás vagy függvény Névvvel rendelkező, az Oracle rendszerben tárolt PL/SQL blokk, amely paramétereket vehet át, és többször elindítható. Használható az Oracle rendszerekben.
  - Alkalmazásban lévő eljárás vagy függvény Névvvel rendelkező, valamely Oracle Developer alkalmazásban vagy megosztott könyvtárban tárolt PL/SQL blokk. Többször hívható. Használható Oracle Developer komponensekben (például Forms).
- Csomag Névvvel rendelkező PL/SQL modul, amely összetartozó eljárásokból, függvényekből és változókból, kurzorokból áll. Használható Oracle rendszerekben és az Oracle Developer komponensekben (pl. Forms).
- Adatbázis trigger Az adatbázis valamelyik táblájához társított PL/SQL blokk, amely automatikusan elindul, ha egy DML utasítás bekövetkezik. Használható Oracle rendszerekben.
- Alkalmazás trigger Egy alkalmazás valamelyik eseményéhez társított automatikusan elinduló PL/SQL blokk. Használható: Oracle Developer komponensekben (pl. Forms).

## Deklarációs szegmens

A változók hivatkozási nevek a PL/SQL-ben is, mint minden programnyelvben. A változókat deklarálni kell. A deklarált változót minden SQL és PL/SQL blokkban kifejezésekben, hivatkozásokban felhasználhatjuk. A változók felhasználása igen széleskörű.

Mire szolgálnak a változók?

- adatok átmeneti tárolására,
- tárolt értékek kezelésére,
- adatbázishoz való hozzáférés nélkül számítások elvégzésére,
- a változók több utasításban, többször is felhasználhatók,
- egyszerű karbantartásra, %TYPE, és %ROWTYPE attribútumok használatával az adatbázis oszlopainak definíciója alapján is deklarálhatunk változókat, rekordokat. Ez azért jó, mert ha az adatbázis oszlopainak definíciója megváltozik, a változó adattípusa is követi ezt futási időben. Ezért adatoktól független kódot írhatunk, ezzel csökkennek a karbantartás költségei.

A PL/SQL karakterkészlete a következő karakterekből állhat:

- az angol abc kis és nagybetűi A..Z, a..z
- számjegyek 0..9
- tabulátorjel, üres karakter, enter
- szimbólumok: ( ) + - \* / < > = ! ~ ; : . ' @ % , " # \$ ^ & \_  
| { } ? [ ]

A PL/SQL nem érzékeny a kis és nagybetűkre, kivéve a sztringeket, a literálokat. A változónevek maximum 30 karakter hosszúak lehetnek, és mindenképpen betűvel kell kezdődniük. A sztring és literál karaktersorozatot aposztrófok közé kell tenni.

- A változókat a blokkok deklarációs részében kell deklarálni. A deklarációk lefoglalják a helyet a változó értékek számára a deklarált adattípusnak megfelelően. Megadhatjuk a változó kezdeti értékét és a NOT NULL kényszert is.

Például, legyen a kezd nevű változó egy szám, kezdőértéke pedig 1.

```
kezd      NUMBER := 1;
```

- A blokk végrehajtható szegmensében a változók értékeit módosíthatjuk. A változó a továbbiakban az új értéket fogja tárolni. Például:

```
x := x + 100;
```

- Alprogramoknak a paraméterein keresztül adhatunk át értéket. Három paraméterátadási mód létezik:

IN (alapértelmezett)	érték
OUT	eredmény
IN OUT	érték-eredmény

Hivatkozási változókat az SQL adatkezelési utasításokban bemeneti és kimeneti változókhoz egyaránt használjuk.

## Egyszerű adattípusok

A PL/SQL változók alapvető típusai a következők:

- **Skaláris** Egyetlen értéket tartalmaz. Lényegében ezek az adattípusok megegyeznek a táblák oszloptípusaival, de lehetnek logikai változók is.
- **Összetett** Rekordok, táblák és változó tömbök.
- **Hivatkozási** Mutatók.
- **LOB** Lokátorok. Igen nagy objektumok esetén.

Nem PL/SQL változók:

- **Hozzárendelt és környezeti változók**, amelyeket az előfordítókbán deklarálhatunk. Ezek a Forms alkalmazások képernyőmezői, és az SQL\*Plus változók.

### SQL\*Plus változók használata PL/SQL blokkban

A PL/SQL nem rendelkezik saját be és kimeneti lehetőségekkel, ezért ahhoz, hogy a felhasználó által megadott értékek bekerüljenek változókba, ezáltal a blokkba, és onnan a visszakapott értékek feldolgozhatók legyenek a futtató környezetben, az SQL\*Plus-ra kell támaszkodnunk.

SQL\*Plus környezetben helyettesítő (más néven globális, hozzárendelt, vagy gazdakörnyezeti) változókat használunk. A helyettesítő változók segítségével futásidejű értékeket (számokat, karaktereket) adhatunk át egy PL/SQL blokknak.

#### Fontos

A PL/SQL blokkban is ”&” jellel hivatkozunk a helyettesítő változóra, ugyanúgy, ahogy az SQL-ben hivatkozunk az SQL\*Plus helyettesítő változókra.

Szöveges értéket a program a *PL/SQL blokk végrehajtása előtt behelyettesíti* a blokkba, ezért a hozzárendelt változókba nem lehet ciklus segítségével *különböző értéket* behelyettesíteni. A hozzárendelt változó mindig csak egy értékkel helyettesíthető be.

SQL\*Plus környezeti (hozzárendelt) változói segítségével adhatunk át futás közben értéket a PL/SQL blokkból az SQL\*Plus környezetnek. Ha a PL/SQL blokkban hivatkozunk a hozzárendelt változóra, akkor a változó elé kettőspontot (:) kell tenni.

### Változók deklarálása

A változókat a blokk, az alprogramok deklarációs szegmensében kell deklarálni, amelyet a **DECLARE** kulcsszó vezet be.

A deklarációs utasítás általános alakja:

*változónév* [CONSTANT] *adattípus* [NOT NULL] [:= | DEFAULT *kif*];

ahol

<i>változónév</i>	a változó neve,
CONSTANT	a változó értéke állandó, kezdőértéket kötelező megadni,
<i>adattípus</i>	a változó adattípusa,
NOT NULL	kényszeríti a változót, hogy mindenképpen legyen értéke. Kezdőértékek megadása kötelező.

*kif* bármilyen PL/SQL kifejezés lehet, literál, másik változó vagy operátorokat tartalmazó kifejezés.

#### JavaSlat

- A változók és konstansok elnevezésénél egységes irányelveket célszerű követni. Például változóneveket *v\_* előtaggal, konstansokat *c\_* előtaggal és gazdakörnyezeti, hozzárendelt változót *g\_* előtaggal szoktak kezdeni.
- Változókhoz kezdeti érték a *:=* operátorral vagy *DEFAULT* kulcsszóval adható.
- Egy sorban csak egy változót deklaráljunk.
- A változók neve ne egyezzen meg a blokkban használt táblák oszlopneveivel.

#### Példa

```
DECLARE
v_belep      DATE;
v_dat        DATE := Sysdate + 7;
g_alap       NUMBER (4) DEFAULT 2000;
g_premium    NUMBER(4) := 1500;
c_honap      CONSTANT NUMBER(2) := 12;
```

### Alapvető skaláris adattípusok

A skaláris adattípusok négy kategóriába sorolhatók: szám, karakter, dátum és logikai típus. A karakteres és szám adattípusnak altípusai is vannak.

Alapvető skaláris típusok:

<code>VARCHAR2(max_hossz)</code>	Változó hosszúságú karaktersorozat. Maximális hossz 32 767 bájt.
<code>NUMBER[(pontosság, tizedesjegyek)]</code>	Pozitív és negatív egész és tört szám.
<code>DATE</code>	Dátum és idő alaptípusa.
<code>CHAR[(fix_hossz)]</code>	Rögzített hosszúságú karaktersorozat.
<code>LONG</code>	Változó hosszúságú karakter adattípus. Maximális hossz kb. 2 Mb.
<code>LONG RAW</code>	Bináris adat, bájtos karakterlánc. Maximális hossz kb 2 Mb.
<code>BOOLEAN</code>	Logikai érték. Három értéket vehet fel: <i>TRUE</i> , <i>FALSE</i> , <i>NULL</i> .
<code>BINARY_INTEGER</code>	Pozitív és negatív 10 jegyű egész szám (±2147483647)
<code>PLS_INTEGER</code>	Előjeles egész számok alaptípusa. Tárolásához kevesebb hely szükséges. Legfeljebb 10 jegyű egész lehet.

Altípusok lehetnek a következők:

- *egész szám:* `DEC`, `DECIMAL`, `DOUBLE PRECISION`, `INTEGER`, `INT`, `NUMERIC`, `REAL`, `SMALLINT`;
- *valós szám:* `FLOAT` (bináris\_pontosság),
- *karakter típus:* `CHAR`, `CHAR(n)`, `VARCHAR`, `STRING`, `LONG`, `RAW`, `LONG RAW`, `ROWID`, stb.

A ROWID egy pszeudooszlop, amelyet az Oracle automatikusan hozzárendel minden táblához. Egyértelműen meghatározza a tábla minden sorát, mint egy egyszerű kulcs. A ROWID adattípusa a hexadecimális sztringként tárolt rowid típus.

Egyéb adattípusok is vannak a PL/SQL-ben, közülük például a multimédiás adatbázisokhoz jól használható a LOB:

LOB nagy objektumok adattípusa; BFILE, BLOB, CLOB, NCLOB. Strukturálatlan objektumoknál használják.

## Skaláris változók deklarálása

### Példa

```
v_job      VARCHAR2(10);
v_szam     BINARY_INTEGER:= 0;
v_fizetes  NUMBER(9,2) :=0;
v_datum    DATE:= sysdate + 7;
c_ado      CONSTANT NUMBER(4,1) := 12.5;
v_feltetel BOOLEAN NOT NULL := TRUE;
```

Mielőtt a PL/SQL egy SELECT utasítás által lekérdezett oszlop értékét változóhoz rendeli – szükség esetén – a forrásoszlop adattípusát a változó adattípusává konvertálja. Ha a PL/SQL nem tudja meghatározni az implicit konverziót, fordítási hibaüzenetet ad. Ebben az esetben 10.1 táblázat által mutatott adattípus konverziós függvényeket lehet használni.

10.1 táblázat. Implicit adattípus konverzió

	BIN_ -INT	CHAR	DATE	LONG	NUM- BER	PLS_ -INT	RAW	RO- WID	VAR- CHAR2
BIN_ -INT		X		X	X	X			X
CHAR	X		X	X	X	X	X	X	X
DATE		X		X					X
LONG		X					X		X
NUM- BER	X	X		X		X			X
PLS_ -INT	X	X		X	X				X
RAW		X		X					X
ROWID		X							X
VAR- CHAR2	X	X	X	X	X	X	X	X	

## Logikai változó deklarálása

A PL/SQL-ben az SQL és a procedurális változók összehasonlíthatók. Ezek az egyszerű vagy összetett összehasonlítások a logikai kifejezések. A logikai kifejezésekben változókat, logikai operátorokat és logikai műveleteket használhatunk. A logikai kifejezés eredménye

TRUE (igaz), FALSE (hamis) és NULL (hiányzó, nem alkalmazható, vagy ismeretlen) értéket vehet fel.

### Példa

```
DECLARE
  kezd      NUMBER := 1;
  veg       NUMBER := 10;
  felt      BOOLEAN := (kezd < veg);
```

Ebben az esetben a `felt` logikai változó igaz értéket vesz fel, értéke tehát TRUE.

## %TYPE attribútum

Ha deklarálunk új PL/SQL változót (például, egy tábla oszlopértékeinek feldolgozásához), akkor a változó típusa teljesen egyezzen meg az oszlop adattípusával. Ez legegyszerűbben úgy valósítható meg, hogy a korábban már deklarált oszlop vagy változó adattípusát nem adjuk meg újra explicit módon, hanem a %TYPE attribútumot használjuk. Ennek előnye, hogy az adatbázis módosításait a változó automatikusan követi.

A deklaráció alakja:

```
változónév      {tábla.oszlop | r_változónév} %TYPE
ahol  tábla.oszlop  annak a tábla oszlopának a neve, amely oszlopa szerinti adattípust
                    akarjuk a változónak adni,
      r_változónév  annak a korábban deklarált változónak a neve, amelynek
                    adattípusát akarjuk a változónak adni.
```

A PL/SQL egy blokk fordításakor határozza meg a változó adattípusát és méretét, ezért lesz kompatibilis azzal az oszloppal, amelynek adataival a változót feltöltjük.

### Példa

```
v_vnev      emp.ename%TYPE;      --hivatkozás az emp tábla ename oszlopára
v_premium   NUMBER(7);
v_min_prem  v_premium%TYPE := 3000;  --hivatkozás deklarált változóra
```

A %TYPE attribútummal deklarált változóra nem vonatkozik az oszlop NOT NULL kényszere, ezért ha a változót így deklaráljuk, NULL kezdőértéket rendelhetünk hozzá.

## %ROWTYPE

Összetett adattípusok

Az összetett adattípusok (gyűjtők) a következők:

```
RECORD      a tábla egy sora,
NESTED TABLE beágyazott tábla,
VARRAY      változó tömb.
```

A RECORD egy sor. A RECORD adattípus a sorban egymáshoz kapcsolódó, de különböző adattípusú adatokat, egyetlen logikai egységként kezeli.

Ha egy rekord típusú változó tartalmazza egy rekord teljes értékhalmozát, akkor az adattípust megadhatjuk a %ROWTYPE attribútummal.

Összetett adatokról később még lesz szó.

**Példa**

```
DECLARE
  Type dolgozo_rekord      emp%ROWTYPE;
```

A `dolgozo_rekord` egy rekord típus a PL/SQL blokkban, amely az `emp` tábla egy sorát fogja feldolgozni.

**Értékadás változónak**

A PL/SQL változó értékadásához, módosításához a blokkban, a program törzsben hozzárendelési utasítást kell írni.

A hozzárendelési utasítás alakja:

*változó* := kifejezés;

**Példa**

Legyen egy változónk dátum típusú, egy másik pedig karakter típusú. Deklaráljuk ezeket, és a blokkban kapjanak értéket, változzon meg a `NUMBER` típusú `jutalek` változó értéke a blokkban.

```
DECLARE
  v_nev          VARCHAR2(10)
  v_belep        DATE
  v_jutalek       NUMBER(7) := 20000;
BEGIN
  v_nev := 'KELEMEN';           -- hozzárendelés
  v_belep := '01-08-05';        -- hozzárendelés
  v_jutalek := v_jutalek + 5000; -- érték megváltoztatása
  ...
END;
```

Az értékadás másik módja az adattábla megfelelő attributumértékének hozzárendelése a változóhoz. Ebben az esetben a `SELECT` vagy a `FETCH` utasítást használjuk.

(A `FETCH` utasítás beágyazott SQL parancs, amely a lekérdezés eredményének egy vagy több sorát tölti be változóba. Részletesen lásd később!).

**Példa**

Az `emp` tábla alkalmazottai átlagfizetésének 10 %-a legyen a fizetésemelés.(s071.sql)

```
SQL> DECLARE
2   v_fiz_emel    emp.sal%TYPE;
3   BEGIN
4     SELECT ROUND(AVG(sal) * 0.10)
5     INTO v_fiz_emel      -- a változóba kerül a lekérdezett érték
6     FROM emp;
7   END;
```

A PL/SQL eljárás sikeresen befejeződött.

Szeretnénk megtudni, hogy a `v_fiz_emel` változónak mennyi lett az értéke. A képernyőn való megjelenítés egyik (és egyben a legegyszerűbb) módja a hozzárendelt változó használata.

## Hozzárendelt változó és használata

A hozzárendelt változók lehetnek numerikus vagy karakteres típusúak, és az SQL\*Plus környezetben lehet őket megadni. Segítségükkel lehet

- a felhasználó által futási időben megadott értékeket PL/SQL programnak átadni, és
- egy PL/SQL programból (például megjelenítés céljára) értékeket átvenni.

A PL/SQL program ezeket ugyanúgy használhatja, mint bármely más változót. A gazdakörnyezetben vagy a hívó környezetben deklarált változókra PL/SQL utasításokban is hivatkozhatunk.

Az SQL\*Plus környezetben a `VARIABLE` utasítás használatával deklarálhatunk hozzárendelt változókat. Ezek képernyőn is megjeleníthetők.

Az utasítás általános alakja:

```
VAR[iable] [ változónév [ NUMBER | CHAR | CHAR (n) | VARCHAR2 (n) |
                                NCHAR | NCHAR (n) | NVARCHAR2 (n) |
                                CLOB | NCLOB | REF_CURSOR ] ];
```

Az alábbiakban azt mutatjuk meg, hogy miként adhatjuk át ennek a hozzárendelt változónak egy, a PL/SQL blokkban deklarált és felhasznált változó értékét.

### Hivatkozás a hozzárendelt változóra PL/SQL blokkban.

Hivatkozni kettősponttal (`:`) lehet PL/SQL blokkban a hozzárendelt, környezeti változóra.

```
...
Begin
...
:hozzaarendelt_valtozo := PL/SQL változó vagy kifejezés;
...
END;
```

### Hozzárendelt változó kiírása

Az SQL\*Plus változót a `PRINT` utasítással jeleníthetjük meg a képernyőn.

A `PRINT` utasítással a `VARIABLE` utasítással létrehozott változó értékét írathatjuk ki.

#### Példa

Jelenítsük meg az előző feladatban kiszámolt fizetésemelés összegét. (`s070.sql`)

```
SQL> VARIABLE g_fiz_ki    NUMBER           -- SQL*Plus változó deklarálása
SQL> DECLARE
2     v_fiz_emel    emp.sal%TYPE ;         -- PL/SQL változó
3 BEGIN
4     SELECT ROUND (AVG (sal) * 0.10)
```

```

5      INTO v_fiz_emel      -- A változó értéket kap a lekérdezésből
6      FROM emp;
7      :g_fiz_ki := v_fiz_emel; -- Az SQL*PLUS változónak átadja a PL/SQL
8                                  -- változó az értékét
9  END;

```

A PL/SQL eljárás sikeresen befejeződött.

```
SQL> PRINT g_fiz_ki
```

```

      G_FIZ_KI
-----
          207

```

### Példa

Kérjünk be egy karaktersorozatot a felhasználótól, fűzzünk hozzá egy literált, kérjünk be egy számot, és adjunk hozzá egy konstans. (s071.sql)

```

-- Környezeti változó deklarálása
VARIABLE uzen VARCHAR2(30);          -- karakteres változó
VARIABLE szam number;                -- numerikus változó

-- SQL*Plus változó definiálása
ACCEPT uzen PROMPT 'Add meg az üzenetedet: '; -- bekérés
ACCEPT szam PROMPT 'Add meg a számot: ';

-- Névtelen blokk
DECLARE                                -- deklaráció
    v_uzenet      varchar2(30);
    v_szam        number;
    konst CONSTANT NUMBER := 888;      -- konstans

BEGIN                                -- végrehajtható rész
    v_uzenet := CONCAT('&uzen',' MŰKÖDIK'); -- változó értéket kap
-- Az értékét a hozzárendelt változónak átadja
    :uzen      := v_uzenet ;
    v_szam     := &szam;                -- változó értéket kap
    :szam      := v_szam + konst;      -- átad egy kifejezést
-- a környezeti változónak

END;
/

```

### Ismétlés

Egy SQL\*Plus változó (hozzárendelt változó) lekérdezése a DEFINE, megszüntetése az UNDEFINE utasítással történik (természetesen csak akkor, ha ACCEPT utasítással, vagy && előtagú változóval hoztuk létre).

Futtassuk le a script programot.

```

SQL> @ c:\a\s071
Add meg az üzenetedet: A számítógép
Add meg a számot: 23451
régi 5:      v_uzenet := CONCAT('&uzen',' MŰKÖDIK');
új 5:      v_uzenet := CONCAT('A számítógép',' MŰKÖDIK');

```

```

régi 7:      v_szam  := &szam;
új 7:      v_szam  := 23451;
A PL/SQL eljárás sikeresen befejeződött.

```

Írassuk ki a hozzárendelt változókat.

```

SQL> PRINT uzen

UZEN
-----
A számítógép  MŰKÖDIK

SQL> PRINT szam

      SZAM
-----
      23451

```

Kérdezzük le, definiált – e a változó.

```

SQL> define
...
DEFINE UZEN          = "A számítógép" (CHAR)
DEFINE SZAM          = "23451" (CHAR)

```

### Példa

Írjuk ki a felhasználó által megadott azonosítójú alkalmazott éves fizetését. (s072.sql)

```

-- s072.sql
-----
VARIABLE      g_fiz NUMBER;

ACCEPT g_azon PROMPT 'Kérem a keresendő alkalmazott azonosító számát'

-- névtelen blokk
DECLARE
    v_fiz emp.sal%type;      -- változó
BEGIN
    SELECT      sal
    INTO        v_fiz        -- a változóba teszi a lekérdezett értéket
    FROM        emp
    WHERE       empno = &g_azon;

    v_fiz      := 12* v_fiz;  -- kiszámítjuk az éves fizetést
    :g_fiz     := v_fiz;     -- átadjuk a hozzárendelt változónak
END;
/

```

Futtassuk le a állományt

```

SQL> @ c:\a\s072
Kérem a keresendő alkalmazott azonosító számát 7788
régi 7:      WHERE      empno = &g_azon;
új 7:      WHERE      empno = 7788;

```



A PL/SQL eljárás sikeresen befejeződött.

### Példa

Kíváncsiak vagyunk arra, hogy mennyi lesz a felhasználó által megadott alkalmazott éves fizetése, ha a havi fizetését 12 %-kal megemeljük. (s074.sql)

```
ACCEPT g_azon PROMPT 'Kérem az azonosító számot '

DECLARE
    v_fiz    emp.sal%type;      -- változó
BEGIN
    SELECT    sal AS fiz
    INTO      v_fiz             -- a változóba teszi a lekérdezett értéket
    FROM      emp
    WHERE     empno = &g_azon;

    v_fiz := 12* (v_fiz*1.12);  -- kiszámítjuk az éves megemelt fizetést

    DBMS_OUTPUT.PUT_LINE('A kívánt dolgozó éves fizetése: '    -- kiíratás
                          ||v_fiz||' dollár');

END;
/
```

Futtassuk le.

```
SQL> @ c:\a\s074
Kérem az azonosító számot 7788
A kívánt dolgozó éves fizetése: 40320 dollár
```

A PL/SQL eljárás sikeresen befejeződött.

## Végrehajtható szegmens

A blokkok végrehajtható utasításai a `BEGIN` és az `END;` között helyezkednek el.

### A végrehajtható szegmens szintaxisa

A PL/SQL nyelv az SQL nyelv kiterjesztése, így mindazon szabályok, amelyek az SQL-nyelvre vonatkoznak, értelemszerűen érvényesek a PL/SQL nyelvre is.

- A lexikai egységeket (például azonosító, literál) egy vagy több szóköz, illetve esetleg más össze nem téveszthető határolójel választja el egymástól. Szóközt lexikai egységen belül nem használhatunk (kivéve aposztrófok között).
- Az utasításokat itt is több sorba írhatjuk, de kulcsszavakat nem bonthatunk meg.

Röviden összefoglaljuk a használható szimbólumokat, ezek határoló jelek, és fontos jelentésük van a PL/SQL-ben.

	Egyszerű szimbólumok		Összetett szimbólumok
+	összeadás	<>	összehasonlító
-	kivonás	!=	összehasonlító (nem egyenlő)
*	szorzás		összefűzés
/	osztás	--	egysoros megjegyzés
=	összehasonlítás (egyenlő)	/*	megjegyzés kezdete
@	távoli hivatkozás	*/	megjegyzés vége
;	utasítás lezáró jele	:=	hozzárendelés

### Azonosítók, literálok, megjegyzések

- **Azonosító**  
Az azonosítók a PL/SQL programegységek (konstansok, változók, kivételek, kurzorok, kurzorváltozók, alprogramok, csomagok) nevei.  
Jellemzői: maximum 30 karakter hosszú lehet, betűvel kell kezdődnie, ne legyen azonos olyan táblanév oszlopnevével, amely táblát a blokkban felhasználunk, ne legyen fenntartott név (a jó strukturáltság érdekében a fenntartott szavakat nagybetűvel írjuk).
- **Literálok:**  
A karakteres és dátum literálok mindig aposztrófok között legyenek.
- **Megjegyzések**  
Az egysoros megjegyzést két kötőjel (--) vezeti be.  
A többsoros megjegyzést /\* és \*/ közé tesszük.

## Belső függvények a PL/SQL-ben

### SQL függvények

A végrehajtható részben használható SQL függvénytípusok:

- Egysoros numerikus
  - Egysoros karakterkezelő
  - Adattípus konvertáló
  - Dátumkezelő függvények
  - GREATEST, LEAST
  - egyéb
- } Ugyanúgy, mint az SQL-ben.

Nem használható függvények:

- DECODE

A DECODE helyett használható az alábbi lekérdezés:

```
SELECT DECODE (oszlopnév)
      INTO változó FROM dual;
```

- Csoportfüggvények: AVG, MIN, MAX, COUNT, SUM, STDDEV és VARIANCE ,  
amelyek a blokkokban, csak az SQL utasításokban fordulhatnak elő.

### PL/SQL függvények

- hibakezelő,
- numerikus,
- karakterkezelő,
- konverzió,
- dátumkezelő,
- egyéb.

Az adattípus konverzió: TO\_CHAR(érték, formátummaszk)  
TO\_DATE(érték, formátummaszk),  
TO\_NUMBER(érték, formátummaszk).

## Operátorok

Az alábbi operátorok ugyanúgy használhatók, mint az SQL-ben:

- logikai,
- aritmetikai,
- összefűzési,
- zárójel,
- hatványozás (\*\*).

A műveletek végrehajtási sorrendje a műveletek precedenciájától függ. Erre mindig ügyeljünk. Nem kötelező, de célszerű zárójeleket használni a logikai kifejezésekben. A különböző műveletek és operátorok egymáshoz viszonyított precedencia értéke:

legmagasabb:	** , NOT	hatványozás, negálás	↓
	*, /	szorzás, osztás	
	+, -,	összeadás, kivonás, összefűzés	
	=, !=, <, >, <=>, IS NULL, LIKE, BETWEEN, IN	hasonlítás	
	AND	logikai ÉS	
legalacsonyabb	OR	logikai VAGY	

A NULL érték használata esetén, figyeljünk a következő szabályokra:

- NULL értéket tartalmazó kifejezések értéke és összehasonlításának eredménye mindig NULL.
- NULL érték logikai negáltja szintén NULL.
- Feltételes vezérlési szerkezetek utasításai nem kerülnek végrehajtásra, ha a logikai feltétel kiértékelésének eredménye NULL.

### Megjegyzés

A kód áttekinthetősége érdekében tördeljük a hosszú sorokat. Üres sorok elhelyezésével, tabulátor használatával, magyarázó szöveg beillesztésével tegyük olvashatóbbá a programot. A változónevek, hozzárendelt változók, globális változók, felhasználói változók, konstansok neveit mindenki alakítsa saját magának egységes kezdőbetűkkel. Az azonos szinten elhelyezkedő programszerkezeteket írjuk egymás alá.

### Példa (hatványozásra)

Számoljuk ki a 2-es szám 5-ödik hatványát. (s069.sql)

```
-- s069.sql
-----
-- Hatványozás a PL/SQL-ben
-----
SET SERVEROUTPUT ON

DECLARE
  aa  NUMBER:= 2;
  bb  NUMBER:= 5;
  cc  NUMBER;
begin
  cc:=aa**bb;
  DBMS_OUTPUT.PUT_LINE('A hatványozás eredmény: '||cc);
end;
/
```

Futtassuk le.

```
SQL> @ c:\a\s069
A hatványozás eredmény: 32

A PL/SQL eljárás sikeresen befejeződött.
```

## DQL utasítások a PL/SQL-ben

Amikor a felhasználó az adatbázisból ki akarja olvasni az adatokat, vagy meg akarja változtatni bizonyos adatok értékeit, akkor SQL utasításokat használ.

A PL/SQL nyelv:

- adatlekérdezéshez egyetlen *DQL* utasításként a `SELECT` utasítást használja,
- a teljes adatkezelési nyelvet (*DML*), valamint a tranzakció vezérlő parancsokat *támogatja*.
- *nem támogatja* az adatdefiniáló (*DDL*) nyelv utasításait, és az adatvezérlő (*DCL*) nyelv utasításait (ezeket a parancsokat az SQL\*Plus-ban külön kell kiadnunk),
- blokkjai nem tranzakciós egységek, így a `COMMIT`, `ROLLBACK`, `SAVEPOINT` utasításokat blokkon belül kiadhatjuk.

## SELECT utasítás a PL/SQL-ben

A PL/SQL –ben az adatbázisból `SELECT` utasítással kérdezhetjük le a tábla sorait.

Általános alak:

```
SELECT szelekt_lista
      INTO {változónév [,változónév]...|rekordnév}
      FROM tábla
      [WHERE feltétel];
```

ahol:

<i>szelekt_lista</i>	legalább egy oszlopot tartalmazó lista, amely SQL kifejezéseket, továbbá sor vagy csoportfüggvényeket tartalmazhat,
<i>változónév</i>	skaláris változó, ebben tároljuk a lekérdezett értéket,
<i>rekordnév</i>	a PL/SQL rekord (ebben tároljuk <i>egy lekérdezett sor</i> értékét),
<i>tábla</i>	táblanév,
<i>feltétel</i>	oszlopnevekből, kifejezésekből, konstansokból és összehasonlító operátorokból állhat, és szerepelhet benne PL/SQL változó és konstans is,
INTO	utasítás rész, amelynek <i>használata kötelező</i> . Ebben az utasításrészben adjuk meg azokat a változókat, amelyben a <code>SELECT</code> utasítás által visszaadott értéket tároljuk. Minden lekérdezett értékhez meg kell adni a lekérdezés sorrendjében egy-egy változót.

### Fontos

Az SQL `SELECT` utasításokat a PL/SQL-ben úgy kell megírunk, hogy *pontosan csak egy sort* adjanak vissza, és e sor oszlop értékei az `INTO` után megadott változókba kerüljenek!

### Példa

Írassuk ki a `dept` táblából a kereskedők részlegének azonosító számát és helyét. (`s076.sql`)

```
SET SERVEROUTPUT ON

DECLARE
  v_reszleg NUMBER(2);
  v_hely    VARCHAR2(15);
BEGIN
  SELECT deptno, loc                                -- szelekt lista
```

```
-- a szelekt listának megfelelő változókba helyezi a táblából lekérdezett
-- adatokat
      INTO v_reszleg, v_hely
      FROM dept
      WHERE dname = 'SALES';
DBMS_OUTPUT.PUT_LINE( v_reszleg || ' ' || v_hely);  --kiíratás
END;
```

Futattási eredmény:

```
SQL> @ c:\a\s076
Az input csonkolva 1 karakterre
30  CHICAGO
```

A PL/SQL eljárás sikeresen befejeződött.

### Példa (csoportfüggvényre)

-- Számítsuk ki a 30-as részlegen dolgozók összes bérét

```
SET SERVEROUTPUT ON
```

```
DECLARE
    v_ossz_ber      emp.sal%TYPE;
    v_reszleg       NUMBER(2)  NOT NULL :=30;
BEGIN
    SELECT SUM(sal)
      INTO v_ossz_ber
    FROM emp
    WHERE deptno = v_reszleg;
    DBMS_OUTPUT.PUT_LINE('Az 30-as részleg dolgozóinak havi bére
                        összesen : ' || v_ossz_ber);
END;
```

```
SQL> @ c:\a\s077
Az input csonkolva 1 karakterre
Az 30-as részleg dolgozóinak havi bére összesen :  9400
```

A PL/SQL eljárás sikeresen befejeződött.

## DML utasítások a PL/SQL-ben

Az adatkezelést, az adatbázistáblák tartalmának módosítását a DML utasításokkal végezhetjük el. A DML utasítások használatát a PL/SQL támogatja. A DML utasítások a következők: INSERT, UPDATE, DELETE.

### Beszúrás

Az INSERT utasítással új sort szúrhatunk be.

#### Példa

Van egy dolgozo táblánk, amely a következőket tartalmazza.

```
SQL> SELECT * FROM dolgozo;
```

AZONOSÍTÓ	NÉV	ÉVES_FIZETÉS
7369	SMITH	9600
7876	MAGYAR ALADÁR	13200
7900	JAMES	11400
7934	MILLER	15600

Szűrjünk be egy új sort. A felhasználó adja meg az aktuális adatokat. (s078.sql)

```
ACCEPT v_azon PROMPT ' Azonosító '
ACCEPT v_nev PROMPT ' Név: '
ACCEPT v_fiz PROMPT ' Fizetés: '

BEGIN
  INSERT INTO dolgozo
  VALUES (&v_azon, '&v_nev', &v_fiz);
END;
/
```

Futtassuk le:

```
SQL> @c:\a\s078
Azonosító 8765
Név: KELEMEN EDE
Fizetés: 12500
régi 3: VALUES (&v_azon, '&v_nev', &v_fiz);
új 3: VALUES (8765, 'KELEMEN EDE ', 12500);
```

A PL/SQL eljárás sikeresen befejeződött.

Ellenőrizzük:

```
SQL> SELECT * FROM dolgozo;
```

AZONOSÍTÓ	NÉV	ÉVES_FIZETÉS
7369	SMITH	9600
7876	MAGYAR ALADÁR	13200
7900	JAMES	11400
7934	MILLER	15600
8765	KELEMEN EDE	12500

## Módosítás

Az UPDATE utasítással módosíthatjuk a tábla adatait.

### Példa

Növeljük az empl tábla összes hivatalnok munkakörű (clerk) alkalmazott fizetését 1000-rel.

```
-- s079.sql
```

```

-----
DECLARE
  v_fizetesemeles emp1.sal%type := 1000;
-- névteklen blokk
BEGIN
  UPDATE emp1
  SET sal = sal + v_fizetesemeles
  WHERE job = 'CLERK';
END;
/

```

Futtassuk le, és ellenőrizzük.

## Törlés

DELETE utasítással törölhetünk adatokat a táblából.

### Példa

Töröljük ki az emp1 táblából azokat, akiknek fizetése kisebb 3000-nél.

```

SQL> DECLARE
2   v_fiz emp1.sal%TYPE := 3000;
3 BEGIN
4   DELETE FROM emp1
5   WHERE sal < v_fiz;
6 END;
/

```

A PL/SQL eljárás sikeresen befejeződött.

```
SQL> SELECT * FROM emp1;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	87-DEC-09	3000		20
7839	KING	PRESIDENT		81-NOV-17	5000		10
7902	FORD	ANALYST	7566	81-DEC-03	3000		20

Valóban kitörölte azokat.

### Példa

Emeljük meg a felhasználó által megadott nevű alkalmazott fizetését, a részlegének átlagfizetésének 10%-kával. Írassuk ki a megemelt összeget. Módosítsuk ezzel az emp adattáblát.

(s075.sql)

```

-- s075.sql
-----
SET SERVEROUTPUT ON

-- változó deklarálás
VARIABLE g_nev          VARCHAR2;
VARIABLE g_reszlegsza NUMBER;

```

```

-- Adatbekérés
ACCEPT g_nev PROMPT 'Kérem a nevet: '
ACCEPT g_reszlegszam PROMPT 'Melyik részlegben dolgozik '

-- Névtelen blokk

DECLARE
    v_fiz_emel    emp.sal%TYPE;
    v_uj_fiz      emp.sal%TYPE;
    v_deptno      emp.deptno%TYPE;
    v_nev         emp.ename%TYPE;
BEGIN
    -- értékadás
    v_deptno := &g_reszlegszam;
    v_nev    := '&g_nev';

    -- A csoportonkénti átlagfizetés 10%-ának meghatározása, és egy érték át-
    -- adása ebből a változónak
    SELECT ROUND(AVG(sal)*0.1)
        INTO v_fiz_emel
        FROM emp
        GROUP BY deptno
        HAVING deptno = v_deptno ;

    -- A keresett dolgozo fizetésének lekérdezése
    SELECT sal
        INTO v_uj_fiz
        FROM emp
        WHERE ename = v_nev;

    -- Az új fizetés meghatározása
    v_uj_fiz := v_uj_fiz + v_fiz_emel;

    -- A tábla fizetés attribútumának módosítása a megadott alkalmazottnál
    UPDATE emp
        SET sal = v_uj_fiz
        WHERE ename = v_nev ;

    -- Képernyőn megjelenítés
    DBMS_OUTPUT.PUT_LINE('Emelés mértéke  ' || v_fiz_emel || ' Forint');

    DBMS_OUTPUT.PUT_LINE('A megemelt fizetés  ' || v_uj_fiz || ' Forint');
END;
/

```

Futtassuk le.

```

SQL> @ c:\a\s075
Kérem a nevet: FORD
Melyik részlegben dolgozik 20
Emelés mértéke 218 Forint
A megemelt fizetés 3218 Forint

A PL/SQL eljárás sikeresen befejeződött.

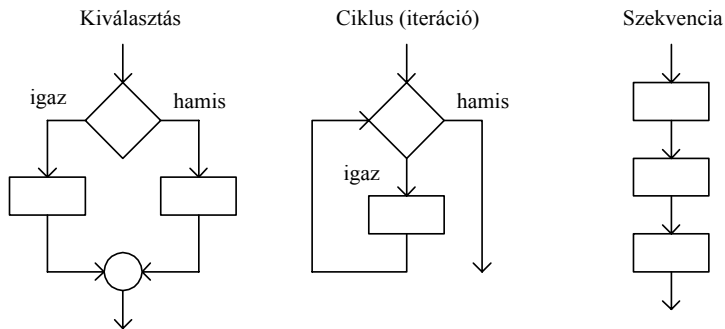
```

Kérdezzük le.

```
SQL> select ename,sal,deptno
      2  FROM emp1;
```

ENAME	SAL	DEPTNO
-----	-----	-----
FORD	3218	20
...		

## Vezérlési szerkezetek



A vezérlési szerkezetek a számítógép bármelyik programnyelvében a következő alapszerkezeteken alapulnak: szekvencia, kiválasztás, ciklus. A PL/SQL nyelvben is ezek az alapszerkezetek. Összetett szerkezetként ezek különböző kombinációi is előfordulhatnak.

A szekvencia (soros) szerkezet az utasítások sorozatát hajtja végre az utasítások sorrendjének megfelelően.

A kiválasztás egy feltételtől függő elágazás. A feltétel igaz vagy hamis voltától függően, az egyik vagy másik ág utasítássorozatát hajtja végre. Feltétel bármilyen változó vagy kifejezés lehet, amelynek eredménye logikai érték; igaz vagy hamis.

A ciklus a ciklusmagban lévő utasítások sorozatát addig hajtja végre, amíg a ciklusfeltétel teljesül.

### IF utasítás

Gyakran nem egy feltételtől függnek a tevékenységek, hanem a körülményeknek megfelelően a feltételek sorozatától. A PL/SQL nyelv IF vezérlő szerkezete erre módot ad.

A feltétel teljesülésétől függően egy vagy több ágon folytatódhat a program. Gyakorlatilag az IF utasításnak három alakja lehetséges:

- IF ... THEN...END IF;  
Ha a feltétel teljesül akkor elvégzi az utasításokat, különben kilép az IF szerkezetből és folytatja a programot.
- IF ... THEN...ELSE...END IF;

A feltétel teljesülése esetén a `THEN` ágon folytatja az utasítások végrehajtását, különben az `ELSE` ágon.

- `IF ... THEN...ELSIF... ELSIF... ELSE...END IF;`  
Ha az első feltétel teljesül, akkor a `THEN` ágon folytatódik a program. Ha az első feltétel nem teljesül, értéke hamis vagy `NULL`, akkor az `ELSIF` ágon folytatódik az utasítás. Az `ELSIF` további feltételeket vezet be. Ha minden feltétel hamis, vagy `NULL` akkor hajtódik végre az `ELSE` ág.

Az utasítás általános alakja:

```
IF feltétel THEN
    utasítások
[ELSIF feltétel THEN
    utasítások]
[ELSIF feltétel THEN
    utasítások]
[ELSE
    utasítások]
END IF;
```

ahol:

<i>feltétel</i>	logikai változó vagy kifejezés, <code>TRUE</code> , <code>FALSE</code> , <code>NULL</code> érték lehet,
<i>utasítások</i>	egy vagy több PL/SQL vagy SQL utasítás. Ezek is tartalmazhatnak további feltételes utasításokat,
<code>THEN</code>	igaz ág,
<code>ELSIF</code>	ha az első feltétel hamis, akkor folytatódik ezen az ágon a program, ahol további feltételek adhatók,
<code>ELSE</code>	ha egyetlen feltétel sem teljesül, akkor folytatódik ezen az ágon az utasítás,
<code>END IF</code>	az <code>IF</code> szerkezet lezárása.

### Fontos

- `ELSIF` egy szó (nem `ELSEIF` !),
- `END IF` két szó,
- Több `ELSIF` szerepelhet egy `IF` utasításban. Számuk nem korlátozott.
- A jobb olvashatóság érdekében strukturáljuk az `IF` utasítást.

### Példa

Írjunk PL/SQL programot, amely a felhasználó által megadott egész szám értékétől függően a különböző kiírásokat hajt végre (`s081.sql`). Kövessük figyelmesen végig a példát.

```
ACCEPT a PROMPT ' Add meg a szám értékét '

DECLARE
    b    VARCHAR2(30);

BEGIN
    IF &a > 999 THEN
        b := 'a szám négyjegyű vagy nagyobb';
    ELSIF &a > 500
        b:= ' félezernél több';
```

```

ELSIF (&a > 99) AND (&a < 499) THEN
    b:= 'háromjegyű, de félezer-nél kisebb' ;
ELSIF &a > 9 THEN
    b:= 'a szám kétjegyű' ;
ELSE
    b:= 'a szám egyjegyű';
END IF;
DBMS_OUTPUT.PUT_LINE(b);
END;
/

```

Futtassuk le.

```

SQL> @ c:\a\s081
Add meg a szám értékét 2134
Az input csonkolva 1 karakterre
a szám négyjegyű vagy nagyobb

A PL/SQL eljárás sikeresen befejeződött.

```

Futtassuk le újból.

```

SQLSQL> @ c:\a\s081
Add meg a szám értékét 4> @ c:\a\s081
Add meg a szám értékét 273
háromjegyű, de félezer-nél kisebb

A PL/SQL eljárás sikeresen befejeződött.

```

Futtassuk le újból.

```

a szám egyjegyű

A PL/SQL eljárás sikeresen befejeződött.

```

### Példa

Jutalmazzuk a dolgozókat. A jutalom munkakörönként változó. A CLERK-ek 1000 dollárt kapnak, a MANAGER-ek 800 dollárt, az ANALYST-ok 700 dollárt a többi 300 dollárt kap, viszont a PRESIDENT-nek ebből nem jut. Módosítsuk a fizetéseket ezzel az összeggel. (s082.sql)

```

SET SERVEROUTPUT ON

ACCEPT g_munkakor PROMPT ' Kérem a dolgozó munkakörét: '
ACCEPT g_azon PROMPT ' Dolgozó azonosítója: '

DECLARE
    v_premium          empl.sal%TYPE;
    v_munkakor          empl.job%TYPE;
    v_azon              empl.empno%TYPE;

BEGIN
    v_munkakor := UPPER ('&g_munkakor');

```

```

v_azon := &g_azon;

IF v_munkakor = 'CLERK' THEN
    v_premium := 1000;
ELSIF v_munkakor = 'MANAGER' THEN
    v_premium := 800;
ELSIF v_munkakor = 'ANALYST' THEN
    v_premium := 700;
ELSIF v_munkakor = 'PRESIDENT' THEN
    v_premium := 0;
ELSE
    v_premium := 300;
END IF;

DBMS_OUTPUT.PUT_LINE(v_munkakor || ' dolgozóinak prémiuma ' || v_premium ||
' dollár');
--módosítás
UPDATE emp1
SET sal = sal+v_premium
WHERE empno = v_azon;
END;
/

```

Futtassuk le!

```

SQL> @ c:\a\s082
Kérem a dolgozó munkakörét: analyst
Dolgozó azonosítója: 7788
régi 7:      v_munkakor := UPPER ('&g_munkakor');
új 7:      v_munkakor := UPPER ('analyst');
régi 8:      v_azon := &g_azon;
új 8:      v_azon := 7788;
ANALYST dolgozóinak prémiuma 700 dollár

```

A PL/SQL eljárás sikeresen befejeződött.

Ellenőrizhetjük lekérdezéssel.

Végrehajtás előtt:

```

...
7788 SCOTT      ANALYST      7566 87-DEC-09      3000      20

```

Végrehajtás után

```

...
7788 SCOTT      ANALYST      7566 87-DEC-09      3700      20
...

```

### NULL ÉRTÉK A LOGIKAI FELTÉTELEKBEN:

Numerikus, karakteres vagy dátumkifejezések összehasonlításával logikai kifejezéseket írhatunk. A NULL értéket általában az IS NULL operátorral kezeljük.

- Az IS NULL feltétel értéke akkor TRUE, ha a vizsgált változó NULL.
- Bármely NULL értéket tartalmazó kifejezés eredménye NULL lesz, kivéve az összefűzés művelet eredményét, amelyben a NULL értékek üres karakterláncként jelennek meg.

**Példa**

Ha `v_alap` `NULL` értékű

```
v_alap := NULL;
```

akkor a következő logikai kifejezések értéke is `NULL`.

```
v_alap > 100
v_alap * 1.8
```

Egy összefűzés eredményeként egy karakterlánc akkor sem `NULL`, ha a benne szereplő egyik sztring `NULL`:

```
v_string := NULL;
'PL'||v_string||'SQL' értéke    nem NULL
```

**Ciklusutasítások**

A ciklus a ciklusmagban szereplő utasítást vagy utasítások sorozatát többször végrehajtja.

A ciklusutasítások háromfélék lehetnek a PL/SQL nyelvben. Ezek:

- egyszerű vagy végtelen ciklus a `LOOP`,
- a `FOR` ciklus,
- a `WHILE` ciklus.

**EGYSZERŰ CIKLUS (LOOP)**

Az egyszerű, vagy végtelen ciklus az utasításokat feltétel nélkül hajtja végre. Amikor az utasítások sorozata után az `END LOOP` utasításhoz ér, a vezérlés mindig visszakérül a ciklus elejét jelző `LOOP` utasításra. Ha nem szükséges a felhasználónak, hogy az utasításokat végtelen sokszor hajtsa végre a ciklus (!), akkor felhasználhatja az `EXIT`, vagy az `EXIT WHEN` utasítást a ciklusból való kilépésre.

Az egyszerű ciklus ciklusmagjában szereplő utasítás/utasítások legalább egyszer végrehajtódnak. Ha a ciklusmag nem tartalmaz `EXIT` vagy `EXIT WHEN` utasítást (amely a ciklus kilépési feltételét adja), akkor a ciklus végtelen ciklus lesz.

Megjegyezzük, hogy az `EXIT` utasítást a ciklusmagon belül önállóan is használhatjuk, sőt az `IF` utasítás belsejében is.

Egy `LOOP` ciklus több `EXIT` utasítást is tartalmazhat.

Az utasítás általános alakja:

<pre>LOOP     ciklusmag END LOOP;</pre>	<pre>LOOP     ciklusmag     [EXIT;]     [EXIT WHEN feltétel;] END LOOP;</pre>
---	---

**Példa**

Hozzunk létre egy `rendel` táblát. Legyen három oszlopa a `rendelszam`, `sorszam`, `kod`. Szűrjünk be a táblába a megadott rendelésszám alá öt új sorszámot. (`s083.sql`)

A tábla létrehozása.

```
CREATE TABLE rendel
(rendelszam    NUMBER(4),
 sorszam       NUMBER(2),
 kod           VARCHAR2(6))
/
```

A névtelen blokk:

```
DECLARE
  v_rszam       rendel.rendelszam%TYPE;
  v_sorszam     rendel.sorszam%TYPE :=1;      -- kezdő értékadás
BEGIN
  LOOP
    INSERT INTO rendel (rendelszam, sorszam)
    VALUES (&v_rszam, v_sorszam);
    v_sorszam := v_sorszam +1;
    EXIT WHEN v_sorszam > 5;                  --kilépési feltétel
  END LOOP;
END;
/
Adja meg a(z) v_rszam értékét: 307
régi   7:      VALUES (&v_rszam, v_sorszam);
új     7:      VALUES (307, v_sorszam);
```

A PL/SQL eljárás sikeresen befejeződött.

Kérdezzük le a rendel tábla sorait:

```
SQL> SELECT * FROM rendel;
```

RENDELSZAM	SORSZAM	KOD
307	1	
307	2	
307	3	
307	4	
307	5	

### Fontos

Az EXIT utasításnak mindig a LOOP cikluson belül kell lennie.

### FOR CIKLUS

Meghatározott számú ismétlésre a FOR ciklust használjuk. A FOR ciklus a FOR...LOOP utasítással kezdődik, és az END LOOP utasítással fejeződik be. Ez visszaadja a vezérlést a LOOP kulcsszó előtti vezérlő utasításra, amely meghatározza, hogy a ciklust hányszor kell végrehajtani. A FOR ciklus ciklusváltozója rendre felveszi a megadott intervallum minden értékét növekvőleg vagy csökkenőleg, és az adott érték mellett végrehajtódik a ciklus magja.

Az utasítás általános alakja:

```
FOR ciklusváltozó IN [REVERSE] alsó_határ ..felső_határ
  LOOP
    utasítás;
    [utasítás;]
    ...
  END LOOP;
```

ahol:

<i>ciklusváltozó</i>	Implicit módon deklarált egész változó. Kezdetben felveszi az alsó vagy felső (REVERSE) határ értékét, majd értéke automatikusan nő, vagy csökken (REVERSE) egyesével a ciklusmag minden egyes végrehajtása után, amíg el nem éri a felső vagy alsó határt.
REVERSE	Kulcsszó hatására a ciklusváltozó a felső határtól indul, és egyesével csökken az alsó határig.
<i>alsó_határ</i>	A ciklusváltozó kezdőértéke, alsó határa.
<i>felső_határ</i>	A ciklusváltozó értékének felső határát adja meg.

A ciklusváltozót nem kell deklarálni, ezt a rendszer implicit módon egész változóként automatikusan elvégzi.

Ciklusváltozó jellemzői:

- A ciklusváltozó csak a cikluson belül értelmezett, a ciklus lefutása után a ciklusváltozó definiálatlan.
- A ciklusváltozó aktuális értéke felhasználható a ciklusmag utasításaiban.
- A ciklusváltozónak a felhasználó nem adhat értéket. Értékadó utasítás bal oldalán nem szerepelhet.
- Az alsó és felső határ szám, változó, vagy kifejezés is lehet.
- Az `index` név nem lehet a ciklusváltozó neve, mert az kulcsszó.

### Példa

Hozunk létre egy `proba` nevű táblát, amelynek egy számoszlopa van. Töltsük fel az oszlopot 1-től 5-ig természetes számmal.

a.) Legyen ez az alsó és felső határ konstans szám.

```
SQL> CREATE TABLE proba
2      (szam NUMBER(3) );
```

A tábla létrejött.

Töltsük fel a táblát

```
SQL> BEGIN
2      FOR ind IN 1..5
3      LOOP
4          INSERT INTO proba (szam)
5              VALUES (ind);
6      END LOOP;
7  END;
8  /
```

A PL/SQL eljárás sikeresen befejeződött.

Listázzuk ki a létrehozott `proba` táblát

```
SQL> SELECT * FROM proba;
      SZAM
-----
1
```

2  
3  
4  
5

b.) Legyen az alsó és felső határ változó.

```
DECLARE
    also_hatar      NUMBER(2) := 4;
    felso_hatar     NUMBER(2) := 10;
BEGIN
    FOR ind IN also_hatar .. felso_hatar
        LOOP
            INSERT INTO proba (szam)
                VALUES (ind*2);
        END LOOP;
END;
/
```

Listázzuk ki a feltöltött táblát.

```
SQL> SELECT * FROM proba;
```

```
          SZAM
-----
...
          8
         10
         12
         14
         16
         18
        20
```

12 sor kijelölve.

Legyen az alsó és felső határ kifejezés:

```
ACCEPT g_also PROMPT ' Alsó határ : '

DECLARE v_also  NUMBER (2);
BEGIN
    v_also := &g_also;
    FOR ind IN v_also.. &v_also+10
        LOOP
            INSERT INTO proba (szam)
                VALUES (ind*3);
        END LOOP;
END;
/
Alsó határ :6
régi   3:   v_also := &g_also;
új     3:   v_also := 6;
```

A PL/SQL eljárás sikeresen befejeződött.

Az input csonkolva 2 karakterre  
Listázzuk ki ismét a feltöltött táblát!

```
SQL> SELECT * FROM proba;
```

```

      SZAM
-----
        18
        21
        24
        27
        30
        33
        36
        39
        42
        45
        48

```

11 sor kijelölve.

### WHILE CIKLUS

A **WHILE** egy úgynevezett előtesztelő ciklus. A ciklusmag addig hajtódik végre, amíg a feltétel igaz. Amikor a feltétel hamissá válik, a ciklusmag nem hajtódik többször végre, az utasítás az **END LOOP** utáni utasításon folytatódik.

Az utasítás általános alakja:

```

WHILE feltétel
LOOP
    utasítás;
    [utasítás;]...
END LOOP;
```

A ciklus magja addig ismétlődik, amíg a feltétel igaz. Ha a feltétel hamissá válik, vagy belépéskor a feltétel **FALSE**, a ciklus magja nem hajtódik végre.

- Ha a ciklusmagban nem változtatjuk meg a feltétel értékét, akkor végtelen ciklus keletkezik.
- Ha a feltétel **NULL**, a ciklus egyszer sem hajtódik végre.

### Példa

Szűrjünk be egy új rendelési szám alá a felhasználó által megadott darabszámú új tételt.

(s085.sql)

```

ACCEPT g_rendelszam PROMPT 'RENDELÉSI SZÁM: '
ACCEPT g_darab PROMPT ' Hány tétel lesz : '

DECLARE
    sorszam          rendel.sorszam%TYPE := 1;
BEGIN
    WHILE sorszam <= &g_darab
    LOOP
        INSERT INTO rendel (rendelszam, sorszam)
```

```

VALUES (&g_rendelszam, sorszam);
sorszam := sorszam + 1;
END LOOP;
END;
/
SQL> @ c:\a\s085
RENDELÉSI SZÁM: 444
Hány tétel lesz : 6
Az input csonkolva 1 karakterre
régi 4: WHILE sorszam <= &g_darab
új 4: WHILE sorszam <= 6
régi 7: VALUES (&g_rendelszam, sorszam);
új 7: VALUES (444, sorszam);

A PL/SQL eljárás sikeresen befejeződött.

```

Liastázzuk ki!

```

SQL> SELECT * FROM rendel;

RENDELSZAM      SORSZAM KOD
-----
307             1
...
307             5
444             1
444             2
444             3
444             4
444             5
444             6

11 sor kijelölve.

```

### EGYMÁSBA ÁGYAZOTT VEZÉRLÉSI SZERKEZETEK

Vezérlési szerkezeteket több mélységben egymásba ágyazhatunk. A beágyazott ciklus befejezése nem jelenti a külső ciklus befejezését, csak akkor, ha kivétel történt. A ciklusok címkézésével azonban rögtön a külső ciklusból is kiléphetünk az `EXIT` utasítással. A címkéknek a szokásos elnevezési szabályok szerint kell nevet adnunk, és rájuk az alábbi szabályok vonatkoznak:

- a címkét az utasítás elé kell tenni, ugyanabba vagy külön sorba,
- a ciklusokat a `LOOP` kulcsszó elé írt névvel címkézhetjük,
- a címkéket ”lúdlábak” közé kell tenni: `<<címke>>`,
- címkével ellátott ciklusok nevét a könnyebb áttekinthetőség kedvéért az `END LOOP` után megismételhetjük.

### Példa részlet

```

...
BEGIN
    <<külső_ciklus>>
    LOOP

```

```
        v_szam := v_szam + 1;
EXIT WHEN v_szam > 10;
    <<belső_ciklus>>
    LOOP
        ...
        EXIT külső_ciklus WHEN befejez :='IGEN';
        -- mindkét ciklus befejezése
        EXIT belső_ciklus WHEN belső_változo :='IGEN';
        -- csak a belső ciklus befejezése
        END LOOP belső_ciklus;
        ...
    END LOOP külső_ciklus;
END;
```

## Összetett adattípusok

A PL/SQL összetett adattípusai közül a `RECORD` típussal és a `TABLE` típussal foglalkozunk.

A `RECORD` típus segítségével olyan változókat tudunk létrehozni, amelyek révén egy adattábla egy sora tárolható a benne szereplő összes oszlopértékkel. Ekkor a változóból az egyes oszlopértékek *változónév.oszlopnév* módon, tehát minősített névvel érhetők el. A gyakorlatban különösen akkor használjuk, amikor egy táblát ciklikusan soronként dolgozunk fel.

A `TABLE` típust egy adattáblából történő kigyűjtés eredményének átmeneti tárolására használjuk, ezért *gyűjtőtáblának*, vagy *PL/SQL táblának* is nevezzük. Szintén tárolhatja egy adattábla több oszlopát (akár egész sorát is), és ekkor az egyes oszlopértékek szintén minősített névvel érhetők el.

Az összetett adattípusok használata könnyen áttekinthetővé teszi programunkat, és használatuk a programozási munkát is kényelmesebbé teszi.

### PL/SQL rekordok

Az adattáblák egyes sorait neveztük rekordnak. Egy rekord egymással logikai kapcsolatban álló – úgynevezett mezőkben (vagy oszlopokban) tárolt – adatértékek egy csoportja, ahol a mezők mindegyikének saját neve és adattípusa van (gondoljunk például az `emp` tábla soraira). Egy sorból vagy annak egy részhalmazából definiálhatunk egy úgynevezett *rekordtípust*. Segítségével egy adattáblából egy *rekordtípusú változóba* "kiemelt" sort, vagy esetleg annak egy részét egy egységként tudjuk kezelni. A rekordtípusú változókat a továbbiakban *PL/SQL rekordoknak*, vagy csak egyszerűen *rekordoknak* fogjuk nevezni.

### RECORD TÍPUSÚ VÁLTOZÓ HASZNÁLATA

Egy rekordtípusú változó (egy rekord) típusát megadhatjuk egy felhasználói típusleírással és e leíráshoz tartozó típusnévvel, valamint valamely adat- vagy nézettábla szerkezetére való közvetlen hivatkozással a `%ROWTYPE` attribútum segítségével.

A felhasználói típusleírás általános alakja:

```
TYPE rekordtípus_név IS RECORD (meződefiníció [,meződefiníció] ...);
```

ahol a *meződefiníció* tartalma:

```
mezőnév {mezőtípus | változó%TYPE | tábla.oszlop%TYPE }  
[NOT NULL] [{ := | DEFAULT} kif]
```

ahol:

<i>típus_név</i>	a létrehozott rekord típus neve (ezt használjuk a rekordtípusú változó deklarálásához),
<i>mezőnév</i>	a rekord egy mezőjének (oszlopának) a neve,
<i>mezőtípus</i>	a mező adattípusa (ez a <code>TABLE</code> kivételével bármely PL/SQL adattípus lehet, de a <code>%TYPE</code> használatával hivatkozni lehet egy tábla valamely oszlopának típusára is),
<i>kif</i>	kezdőérték (közvetve típusmegadásra is használható),
DEFAULT	alapértelmezett érték,

NOT NULL kényszer (ha ezt a megszorítást alkalmazzuk, akkor kötelező a kezdőérték megadása).

Így egy *rekordtípusú változó* deklarálása:

*változónév* {*rekordtípus\_név* | *tábla*%ROWTYPE;}

ahol a *tábla*%ROWTYPE típus révén a változó felveszi a megadott tábla rekordszerkezetét.

A rekordtípusú változók egyes mezőire – a táblák oszlopneveihez hasonlóan – *változónév.mezőnév* alakban kell hivatkozni, bármelyik oldalán is szerepeljenek egy értékadásnak.

### Példa

Deklaráljunk egy rekordot egy új alkalmazott adatainak tárolásához, felhasználói típusleírással.

```
DECLARE
TYPE dolgozo_rekord_típus IS RECORD
(azonosító emp.empno%TYPE NOT NULL :=6543,
név emp.ename%TYPE,
munkakör emp.job%TYPE,
részleg emp.deptno%TYPE
fizetés emp.sal%TYPE);
```

Változó deklarálása (memóiahely lefoglalása) egy ilyen rekord számára:

```
egy_dolgozo dolgozo_rekord_típus
```

Az új alkalmazott *reszl* mezőjére ponttal hivatkozhatunk a következőképp:

```
egy_dolgozo.reszl
```

Legyen az új alkalmazott neve *Katona*. Ekkor az erre vonatkozó értékadás

```
egy_dolgozo.név := 'Katona';
```

### Példa

Deklaráljunk egy rekordot egy alkalmazott számára a %ROWTYPE attribútum segítségével.

```
DEKLARE
alkalmazott_rekord emp%ROWTYPE
```

Ekkor az *alkalmazott\_rekord* rekord szerkezete megegyezik a hivatkozott *emp* tábla szerkezetével:

Név	Üres?	Típus
EMPNO	NOT NULL	NUMBER (4)
ENAME		VARCHAR2 (10)
JOB		VARCHAR2 (9)
MGR		NUMBER (4)
HIREDATE		DATE
SAL		NUMBER (7,2)
COMM		NUMBER (7,2)
DEPTNO	NOT NULL	NUMBER (2)

Hivatkozás a jutalék oszlopra:

```
alkalmazott_rekord.comm
```

## REKORDÉRTÉKADÁS TÁBLÁBÓL

A rekordtípusú változók (a rekordok) használatának előnye, hogy közvetlenül, több oszlopérték egyetlen értékadással lekérdezhető egy adat- vagy nézettáblából. Attól függően, hogy egy rekordot felhasználói típusmegadással, vagy a %ROWTYPE attribútum segítségével deklaráltunk, különbözőképpen kell végeznünk az értékadást. A táblából való értékadás a SELECT és a FETCH utasítás segítségével végezhető. Ezek közül a SELECT használatát már bemutattuk a "SELECT utasítás a PL/SQL-ben" alfejezetben, a FETCH utasításra pedig majd az "Explicit kurzorok" alfejezetben fogunk visszatérni.

Ha felhasználói típusmegadást használtunk a változó deklarálásakor, akkor ügyelni kell arra, hogy az alkalmazott utasítás oszlopnév listájában az oszlopok sorrendje egyezzen meg a rekordváltozó definíciójában szereplő oszlopok számával és sorrendjével (a neveknek nem kell megegyezni!).

Ha azonban a %ROWTYPE attribútum segítségével deklaráltuk a rekordváltozót, akkor az alkalmazott utasításban a (%ROWTYPE előtt) megadott tábla teljes sorszerkezetére kell hivatkozni, mégpedig kötelezően a \* paraméterrel.

### Fontos

Ügyeljünk arra, hogy a %ROWTYPE attribútum segítségével deklarált rekordváltozónak való értékadás esetén nem használható a \* helyett a teljes táblaszerkezet megadása az értékadó utasításban (például a SELECT kulcsszó után az összes oszlop felsorolása), mivel a PL/SQL ezt nem ismeri fel a %ROWTYPE attribútum használatával kijelölt teljes táblaszerkezettel azonos megadásnak (még ha a kettő láthatóan meg is egyezik). Ebben az esetben arra is figyelni kell, hogy a rekordváltozó egyes mezőinek (oszlopértékeinek) kiírásánál a hivatkozott adat- vagy nézettábla oszlopneveit kell használni. A %ROWTYPE attribútum segítségével definiált rekordváltozók használatának előnye, hogy a hivatkozott tábla szerkezetének változása esetén sem kell az utasítást megváltoztatni, mivel a rekordváltozó szerkezete automatikusan követi azt.

Az óvatosság azonban a másik irányban is fontos. Ha egy rekordváltozó felhasználói definíciójában egy tábla minden oszlopát (a táblabeli sorrendben) akár a táblaoszlopokra való hivatkozással is defináltunk (%TYPE módon), az értékadás során *nem használhatjuk* a \* paramétert az előbbiekkal azonos okokból. Ekkor tehát kötelező az értékadó utasításban a rekordváltozó definíciójában szereplő oszlopnevek mindegyikét a definícióban szereplő sorrendben kiírni. Ennek a fajta rekordhasználatnak az az előnye az előzővel szemben, hogy itt mind a rekordváltozó mezőnevei, mind az értékadó utasítás oszlopnévlistájában szereplő nevek eltérhetnek a hivatkozott tábla belső oszlopneveitől, így az adott alkalmazás jobban érthetővé, áttekinthetőbbé tehető.

### Példa

Deklaráljunk egy rekordot az emp tábla egy alkalmazottjára, és írassuk ki a program futása közben azt az alkalmazottat, aki a maximális jutalékot kapta.

A programot az s087.sql script program tartalmazza.

```

SET SERVEROUTPUT ON
-- Rekord deklarációja
DECLARE
    alkalmazott_rekord      emp%ROWTYPE;

-- a maximális jutalékú dolgozó lekérdezése, rekordváltozóba helyezése
BEGIN
    SELECT *
      INTO alkalmazott_rekord
    FROM emp
    WHERE comm =
          (SELECT MAX(comm)
           FROM emp);

-- képernyőre írassuk ki a rekord néhány mezőjét a DBMS_OUTPUT csomag
-- segítségével
    DBMS_OUTPUT.PUT_LINE('A dolgozó: '
                          ||alkalmazott_rekord.ename||', '
                          ||alkalmazott_rekord.job||', '
                          ||alkalmazott_rekord.sal||' dollár, '
                          ||alkalmazott_rekord.sysdate||', '
                          ||alkalmazott_rekord.comm||' dollár';

END;
/

```

Futtassuk le

```

SQL> @ c:\a\s087
Az input csonkolva 1 karakterre
A dolgozó: MARTIN, SALESMAN, 1250 dollár, 81-SZE-28, 1400 dollár

A PL/SQL eljárás sikeresen befejeződött.

```

Valóban MARTIN kereskedő kapta a legtöbb jutalékot.

### Példa

A dolgozó táblából töröljük a legkevesebbet kereső dolgozót, mert kilépett. A kilépett dolgozót tegyük bele a kilépők (kilepok) nevű, azonos felépítésű táblába. Írassuk ki ellenőrzésképpen a kilépő dolgozó nevét, majd véglegesítsük a DML utasításokat. Futtassuk le a programot. Láthatjuk a táblát az s088.sql script program megjegyzésében.

```

SQL> @ c:\a\s088
DOC> AZONOSITO NÉV                EVES_FIZETES
DOC>-----
DOC>      7369 SMITH                9600
DOC>      7876 MAGYAR ALADÁR        13200
DOC>      7900 JAMES                11400
DOC>      7934 MILLER              15600
DOC>      8765 KELEMEN EDE          12500 */
Az input csonkolva 1 karakterre
A kilépő dolgozó neve: SMITH

A PL/SQL eljárás sikeresen befejeződött.

```

A program az s088.sql script programban van, amelynek tartalma a következő:

```

DECLARE
    dolgozo_rekord    dolgozo%ROWTYPE;
BEGIN
    SELECT *
    INTO  dolgozo_rekord
    FROM dolgozo
    WHERE eves_fizetes =
            (SELECT MIN(eves_fizetes)
             FROM dolgozo);

    -- törlés a dolgozo táblából
    DELETE FROM dolgozo
    WHERE azonosito = dolgozo_rekord.azonosito;

    DBMS_OUTPUT.PUT_LINE('A kilépő dolgozó neve: '||dolgozo_rekord.nev);

    -- beszúrás a kilepok táblába
    INSERT INTO kilepok
    VALUES (dolgozo_rekord.azonosito, dolgozo_rekord.nev,
            dolgozo_rekord.eves_fizetes);

    -- véglegesítés
    COMMIT;                                END;
/

```

Ellenőrizzük a kilepok táblát.

```

SQL> SELECT * FROM kilepok;

  AZONOSITO NÉV                EVES_FIZETES
-----
7369 SMITH                      9600

```

## PL/SQL táblák

A `TABLE` típusú változókat PL/SQL, vagy gyűjtőtábláknak hívjuk. Adat-, vagy nézettáblák lekérdezett sorainak a memóriában listászerű tárolására alkalmasak. A használat szempontjából úgy tekinthetjük őket, mintha memóriaváltozók volnának. (Mivel e táblák mérete korlátatlan, nem igazán memóriaváltozók, valójában egy file-kezelési mechanizmus tartozik hozzájuk a háttérben. Ennek tényét nem célszerű figyelmen kívül hagyni, ugyanis a feladataink végrehajtási idejét e táblák nem megfelelő használata jelentősen megnövelheti.)

Az adatbázisban a gyűjtőtáblák úgy látszanak, mint egy egyoszlopos adattábla. Az Oracle a gyűjtőtáblákban nem rendezi a sorokat, azonban, ha lekérdezzük a gyűjtőtábla sorait egy PL/SQL változóba, akkor a sorok egytől kezdődő indexszel rendelkeznek. Ez adja az egyedi sorok tömörszerű használatát.

A gyűjtőtábla tulajdonképpen egy egydimenziós tömb. Két fontos dologban különbözik a tömbtől:

- Egy tömb mérete rögzített, a gyűjtőtábla mérete dinamikusan növekedhet.

- A tömb "tömör" (a bejegyzések rendre egymás után kell, hogy következzenek), egyes elemeket nem lehet törölni belőlük. Kezdetben a gyűjtőtábla is tömör, de lehet ritkítani (az adatok akkor már nem közvetlenül egymás után következnek).

321	29	15	74	345	710	37	58	104
$x_{(1)}$	$x_{(2)}$	$x_{(3)}$	$x_{(4)}$	$x_{(5)}$	$x_{(6)}$	$x_{(7)}$	$x_{(8)}$	$x_{(9)}$

Rögzített felső határ

321		15	74	345		37		104
$x_{(1)}$	$x_{(2)}$	$x_{(3)}$	$x_{(4)}$	$x_{(5)}$	$x_{(6)}$	$x_{(7)}$	$x_{(8)}$	$x_{(9)}$

Rögzítetlen, dinamikus

A gyűjtőtáblában az elsődleges kulcsként használható index oszloppal érhetjük el az egyes sorokat. Ha olyan indexre hivatkozunk, amelyhez nem tartozik adat, akkor NO-DATA-FOUND kivétel keletkezik.

Egy gyűjtőtábla az alábbi két komponenset tartalmazza:

- Egy BINARY\_INTEGER adattípusú elsődleges kulcsot a gyűjtőtábla indexeléséhez,
- egy skaláris vagy rekord adattípusú oszlopot a gyűjtőtábla elemeinek tárolásához.

A típusdefiníció általános alakja:

```
TYPE típus_név IS TABLE OF
    {oszloptípus | változó%TYPE | tábla.oszlop%TYPE | tábla%ROWTYPE}
    [NOT NULL]
    INDEX BY BINARY_INTEGER;
```

Az azonosító, változó deklarálása

*azonosító*      *típus\_név*;

ahol:

<i>típus_név</i>	a TABLE típus neve,
IS TABLE OF	kulcsszó,
<i>oszloptípus</i>	standard PL/SQL adattípus, amely lehet rekord is,
<i>változó</i> %TYPE	hivatkozás korábban deklarált változóra,
<i>tábla.oszlop</i> %TYPE	hivatkozás adatbázisbeli tábla oszlopának típusára,
<i>tábla</i> %ROWTYPE	hivatkozás a tábla egy sorára,
NOT NULL	kényszer megadás,
INDEX BY BINARY_INTEGER	utasításrész megadása kötelező azért, hogy a felhasználó tudja, hogy az index egész típusú.

A TABLE deklaráció új táblatípust definiál, amelyhez gyűjtőtábla (típusú változó) hozható létre. A gyűjtőtáblának egyetlen adatoszlópa van, és egy egész-szám típusú elsődleges kulcsa, valamint tetszőleges számú sora lehet, azaz a sorok száma dinamikusan növekedhet. A gyűjtőtábla tetszőleges indexű sorára hivatkozhatunk, ha a tábla neve után zárójelben megadjuk a sor indexét. Az elsődleges kulcs értéke negatív is lehet.

A gyűjtőtáblák szerkezete

<u>elsődleges kulcs</u>	<u>oszlop</u>
-------------------------	---------------

...		...
1		Magyar Aladár
2		JAMES
3		Miller
...		...

Ha olyan indexre hivatkozunk, amelyikhez nem tartozik adat, akkor a NO\_DATA\_FOUND kivételt váltjuk ki.

A tábla elsődleges kulcsként funkcionáló indexének kezdőértéket kell kapnia. A tábla adatoszlópa az INSERT, az UPDATE, a FETCH, vagy a SELECT utasítással, illetve alprogram segítségével kaphat értéket. Ügyeljünk arra, hogy a *gyűjtőtábla adatoszlópa azonos adattípusú legyen a neki értéket adó adattábla oszloppal, vagy kifejezéssel.*

## HIVATKOZÁS GYŰJTŐTÁBLÁKRA

Általános alak:

*gyűjtőtáblanév(elsődleges\_kulcs\_érték)*

Például

```
nevek_tábla(2) := 'JAMES'
```

### Példa

```
DECLARE
  TYPE nev_tábla_típus IS TABLE OF dolgozo.nev%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE eves_fizetes_tábla_típus IS TABLE OF dolgozo.eves_fizetes%TYPE
    INDEX BY BINARY_INTEGER;
  nev_tábla          nev_tábla_típus;
  eves_fizetes_tábla eves_fizetes_tábla_típus;
BEGIN
  ...
  nev_tábla(1) := 'Kiss';
  eves_fizetes_tábla(3) := 15600;
  IF nev_tábla.EXISTS(1) THEN
    INSERT INTO ...
  ...
END;
```

## GYŰJTŐTÁBLA-METÓDUSOK

A gyűjtőtáblák metódusai megkönnyítik a gyűjtőtáblák használatát. A metódusok beépített eljárások vagy függvények, amelyeket táblákon használhatunk, és a gyűjtőtáblára hivatkozó minősített névvel hívhatjuk meg.

Általános alak

*táblanév.metodusnév[(paraméterek)]*

Metódus	Leírás
EXISTS ( <i>n</i> )	True értéket ad vissza, ha létezik az <i>n</i> -ik elem.
COUNT	A gyűjtőtáblában aktuálisan szereplő elemek számát adja vissza.
FIRST, LAST	A gyűjtőtábla első, illetve utolsó indexértékét adja vissza. Üres tábla esetén null értéket ad vissza.
PRIOR ( <i>n</i> )	A táblában az <i>n</i> -ik indexet megelőző index értékét adja vissza.
NEXT ( <i>n</i> )	A táblában az <i>n</i> -ik indexet követő index értékét adja vissza.
EXTEND ( <i>n</i> , <i>i</i> )	Megnöveli a tábla méretét: EXTEND egy NULL elemet fűz a táblához, EXTEND ( <i>n</i> ) <i>n</i> db NULL elemet fűz a táblához, EXTEND ( <i>n</i> , <i>i</i> ) <i>n</i> másolatot készít az <i>i</i> -ik elemről, és hozzáfüzi azokat a táblához.
TRIM ( <i>n</i> )	Elemeket távolít el a tábla végéről. TRIM eltávolít egy elemet a tábla végéről, TRIM ( <i>n</i> ) <i>n</i> elemet távolít el a tábla végéről.
DELETE ( <i>n</i> )	Elemeket távolít el a táblából. DELETE ( <i>n</i> ) A tábla <i>n</i> -ik elemét eltávolítja. DELETE ( <i>m</i> , <i>n</i> ) Az <i>m</i> .. <i>n</i> indextartományba eső elemeket távolítja el. DELETE Az összes elemet eltávolítja.

*Megjegyzés*

Az EXTEND és a TRIM függvény csak az Oracle 8-as verziójától használhatóak.

**REKORDTÍPUSÚ GYŰJTŐTÁBLA**

Egy adattábla oszlopértékei a gyűjtőtáblára vonatkozó minősített névvel érhetők el az alábbi hivatkozással:

*tábla(index).mező*

**Példa**

Növeljük meg a JAMES nevű alkalmazott fizetését 20%-kal.

```

DECLARE
    TYPE dolg_tabela_tipus IS TABLE OF emp1%ROWTYPE
        INDEX BY BINARY_INTEGER;
    dolg_tabela    dolg_tabela_tipus;
    sor_szam       NUMBER(2) := 1;
BEGIN
    -- az 1. sorszámu adat legyen JAMES
    dolg_tabela(sor_szam).ename := 'JAMES';
    UPDATE emp1
        SET sal = 1.2 * sal

```

```

        WHERE ename = dolg_tabla(sor_szam).ename;
COMMIT;
END;
/

```

Ellenőrizzük.

A program futása előtt:

```

SQL> SELECT ename, job, sal
       2     FROM emp1;

```

```

ENAME          JOB          SAL
-----
...
JAMES          CLERK          950
...

```

A program futtatása után:

```

...
JAMES          CLERK          1140
...

```

### Példa

Készítsünk egy névtelen PL/SQL blokkot, amely egy gyűjtőtáblát készít a dept tábla részleg neveiből, majd írassuk ki a neveket.

A program listája:

```

-- s090-1.sql
-- PL/SQL TABLE típus használata 1. --

SET SERVEROUTPUT ON

DECLARE
    TYPE saját_típus IS TABLE OF dept.dname%TYPE
        INDEX BY BINARY_INTEGER;
    reszlegnev    saját_típus;
    ind          BINARY_INTEGER := 10;

BEGIN
    -- tábla feltöltése
    LOOP
        SELECT dname
           INTO reszlegnev(ind)
          FROM dept
         WHERE deptno = ind;

        -- közbenső kiiratás
        DBMS_OUTPUT.PUT_LINE('Eddigi elemek száma: ' || reszlegnev.COUNT ||
                               ' utolsó index: ' || reszlegnev.LAST);

        -- kilépés
        ind := ind + 10;
        EXIT when ind = 50;
    END LOOP;

```

```
-- tábla kiíratása
ind := reszlegnev.FIRST;
DBMS_OUTPUT.PUT_LINE('-----');
LOOP
    EXIT WHEN NOT reszlegnev.EXISTS(ind);
    DBMS_OUTPUT.PUT_LINE(reszlegnev(ind));
    ind := reszlegnev.NEXT(ind);
END LOOP;
END;
/
```

A program futtatása:

```
SQL> @ c:\a\s090.sql
Eddigi elemek száma: 1   utolsó index: 10
Eddigi elemek száma: 2   utolsó index: 20
Eddigi elemek száma: 3   utolsó index: 30
Eddigi elemek száma: 4   utolsó index: 40
-----
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

A PL/SQL eljárás sikeresen befejeződött.

## SQL kurzor

Az eddigi eszközeinkkel csupán egyetlen sor adatait tudtuk egy `SELECT` utasításon kívül feldolgozni (illetve egy rekordkigyűjtést valamely `SELECT`-beli csoportfüggvénnyel). Ez azonban egy komolyabb feldolgozáshoz nem elég. A PL/SQL blokkban általában több sort szeretnénk feldolgozni.

Az PL/SQL lehetővé teszi adattáblák soronkénti feldolgozását az úgynevezett kurzor segítségével. Ennek során az Oracle egy munkaterületet hoz létre, egy memóriaterületet foglal le, ahol a feldolgozandó, (a kurzorhoz tartozó `SELECT` utasítással) a lekérdezett adatokat, (gyakorlatilag e `SELECT` által visszaadott) az eredménytáblát átmenetileg tárolja. Ehhez a lefoglalt munkaterülethez azonosítót rendelhetünk (ez a kurzor neve), és a későbbiek során e névvel hivatkozhatunk rá. A kurzor tehát egy szimbólikus mutató, amely ennek az átmeneti tárolóhelynek (eredménytáblának) a kezdetére mutat. Ha ezt a mutatót (egy ciklus segítségével) végigmozgatjuk az átmeneti eredménytábla minden egyes során, akkor e sorokat egyenként feldolgozhatjuk. Nagyon fontos, hogy egy kurzorterület megnyitása után a kurzorterületen lévő adatok nem frissülnek (egészen a következő megnyitásig), és azok csak feldolgozásra használhatók, azon keresztül az adattábla adatai nem változtathatók meg.

Az Oracle-ben kétféle kurzor létezik:

- az *implicit*, és
- az *explicit* kurzor.

A PL/SQL nyelv minden PL/SQL blokkban kiadott DML (`INSERT`, `DELETE`, `UPDATE`) és explicit kurzorral nem rendelkező DQL (`SELECT`) utasításhoz létrehoz egy úgynevezett

*implicit kurzort*. Az ekkor keletkező átmeneti eredménytáblának nincs neve, azt az Oracle rendszer automatikusan kezeli. A létrehozott implicit kurzorra `SQL` előnévű függvénnyel hivatkozhatunk.

A programozó saját maga is deklarálhat, elnevezhet kurzort (elnevez egy lekérdezést). Ekkor *explicit kurzorról* beszélünk. Az explicit kurzort vezérelhetjük az `OPEN`, `FETCH` és `CLOSE` utasításokkal.

## Kurzorfüggvények

A kurzorfüggvények segítségével ellenőrizhetjük, hogy mi történt a kurzor legutolsó használata során. A PL/SQL átmenetileg lefoglal egy területet az utasítás végrehajtásához szükséges információk, és egyéb jellemzők tárolására.

A PL/SQL az alábbi kurzorfüggvényeket definiálja:

- `%FOUND`,
- `%NOTFOUND`,
- `%ROWCOUNT`,
- `%ISOPEN`.

Ezeket a függvényeket általában a blokk kivételkezelő részében használjuk fel. Információt adnak a DML (`INSERT`, `UPDATE`, `DELETE`) és DQL (`SELECT`) utasításokról, mivel a PL/SQL nem jelez hibát, ha egyetlenegy sort sem érintettek a kiadott utasítások.

### **%FOUND, %NOTFOUND**

A `%FOUND` és a `%NOTFOUND` kurzorfüggvények jelzik az utoljára végrehajtott SQL (DML és DQL) utasítások sikerességét.

E függvényekre az alábbi alakban lehet hivatkozni implicit kurzor esetén:

`SQL%FOUND`,  
`SQL%NOTFOUND`.

A `%FOUND` egy logikai értéket ad vissza. Értéke igaz, ha a legutóbbi SQL utasítás legalább egy sort megvizsgált, vagy feldolgozott. Ha nincs több feldolgozandó sor, akkor a `%NOTFOUND` lesz igaz. Az utasítások feltételrészében (például az `IF`, vagy `EXIT WHEN` után) használható. Értéke `TRUE`, `FALSE` vagy `NULL`.

A `%NOTFOUND` a `%FOUND` logikai ellentettje. Azonban, ha a `%FOUND NULL` értékű, akkor a `%NOTFOUND` is `NULL` értékű.

### **%ROWCOUNT**

Az SQL (DML és DQL) utasítással megvizsgált, feldolgozott sorok számát adja vissza. Használhatjuk meghatározott számú sor feldolgozásához, de leggyakrabban a kivételkezelő részben használjuk. (Például, ha a `SELECT` utasításnak éppen egyetlen sort kell visszaadnia.)

A `%ROWCOUNT` értéke 0, ha nem sikerült sort lekérdezni,  
 értéke *n*, ha a visszaadott sorok száma *n*.

Implicit kurzor esetén a hivatkozott alakja:

`SQL%ROWCOUNT`

**%ISOPEN**

A kurzor megnyitását ellenőrzi. Ha a kurzor nincs megnyitva, akkor értéke `FALSE`. Implicit kurzor esetén értéke mindig `FALSE`, mert a PL/SQL egy utasítás végrehajtása után mindig automatikusan bezárja az implicit kurzort.

Implicit kurzor esetén a hivatkozott alakja:

`SQL%ISOPEN`

**Példa**

Töröljük ki az `emp1` (azonos az `emp` táblával) kereskedőit (`SALESMAN`), és írassuk ki a kitörölt kereskedők számát.

A megoldás az `s080.sql` script programban található.

```
VARIABLE sortorol VARCHAR2(30)          -- hozzárendelt SQL*Plus változó

DECLARE
    v_foglalkozás    emp1.job%TYPE := 'SALESMAN';  -- PL/SQL változó
BEGIN
    DELETE FROM emp1
        WHERE job = v_foglalkozás;
    -- adjuk át az értékeket a hozzárendelt változónak
    :sortorol := (SQL%ROWCOUNT||'   sort kitöröltünk');
END;
/
```

A kurzor végigmegy a feltételnek megfelelő sorokon, és mindig az éppen aktuális sorra mutat.

7499 ALLEN	SALESMAN	←	aktuális sor az első esetben
7521 WARD	SALESMAN		
7566 JONES	MANAGER		
7654 MARTIN	SALESMAN		
7839 KING	PRESIDENT		
7844 TURNER	SALESMAN	←	aktuális sor a negyedik esetben

Futtassuk le a fenti script programot, és írassuk ki az SQL\*Plus változót.

```
SQL> @ c:\a\s080
Az input csonkolva 1 karakterre

A PL/SQL eljárás sikeresen befejeződött.

SQL> PRINT sortorol

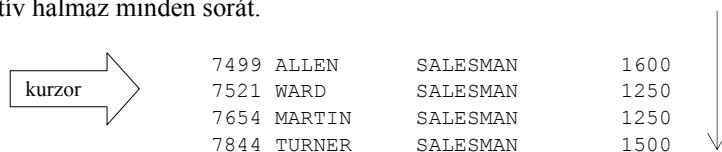
SORTOROL
-----
4      sort kitöröltünk
```

## Explicit kurzorok

Az implicit kurzorokról már beszéltünk. Ezeket az Oracle rendszer minden egysoros lekérdezéshez és DML utasításhoz hozzárendeli. A kurzor egy nem-megosztott SQL terület, vagyis egy olyan terület, amelyen az adatbázis táblák visszaadott sorait statikusan tárolja az Oracle rendszer. Ha a lekérdezés több sort ad vissza, és ezt a felhasználó tovább szeretné feldolgozni, lehetősége van arra, hogy névvel ellátott memóriaterületet, kurzort deklaráljon. Ezt a kurzordeklarációt a PL/SQL blokk, az alprogram vagy csomag deklarációs részében kell megtenni. Igen gyakran használunk explicit kurzorokat, amelyeket aztán ciklusokban tudunk vezérelni, beleértve a kurzort használó `FOR` ciklust is. (Az implicit kurzor nem vezérelhető).

Explicit kurzor használatával külön-külön dolgozhatjuk fel a többsoros `SELECT` utasítás által visszaadott sorok halmazát. Ezeket a visszaadott sorokat *eredmény* vagy *aktív halmaznak* nevezzük. A kurzor az aktív halmaz aktuális sorára mutat.

Aktív halmaz lehet, például az `emp` tábla kereskedőinek halmaza. A kurzor végigjárja az aktív halmaz minden sorát.



7499	ALLEN	SALESMAN	1600
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7844	TURNER	SALESMAN	1500

Az explicit kurzor deklarálása és vezérlése a `DECLARE`, az `OPEN`, a `FETCH` és a `CLOSE` utasításokkal történik.

- A kurzor deklarációjában elnevezzük a kurzort, és meghatározzuk a benne végrehajtandó lekérdezés szerkezetét.
- `OPEN` utasítás: végrehajtja a kurzorhoz társított lekérdezést, meghatározza az aktív halmazt, és az első sorra helyezi a kurzort (mutatót), további felhasználásra megnyitja a kurzor területét.
- `FETCH` utasítás: behívja a kurzorhoz tartozó `SELECT` utasítás eltárolt eredménytáblájának aktuális sorát, és továbblépteti a kurzort a következő sorra.
- `CLOSE` utasítás: bezárja a kurzort.

A fenti vezérlőutasítások használatának logikai szerkezetét mutatja be a 10.1. ábra.

Az explicit kurzor jellemzői

- A lekérdezés által visszaadott első sor után a következő sorokat is képes egyesével feldolgozni.
- Nyomon követi, hogy éppen melyik sor feldolgozása van folyamatban.
- Lehetőséget nyújt arra, hogy a programozó vezérelje a kurzort a PL/SQL blokkban.

### A kurzor deklarálása

A PL/SQL-ben nem lehet hivatkozni olyan változóra, amely nincs deklarálva. Ezért a kurzort a deklarációs részben deklarálni kell.

A deklarációs utasítás alakja:

```
CURSOR kurzornév IS
  SELECT utasítás;
```

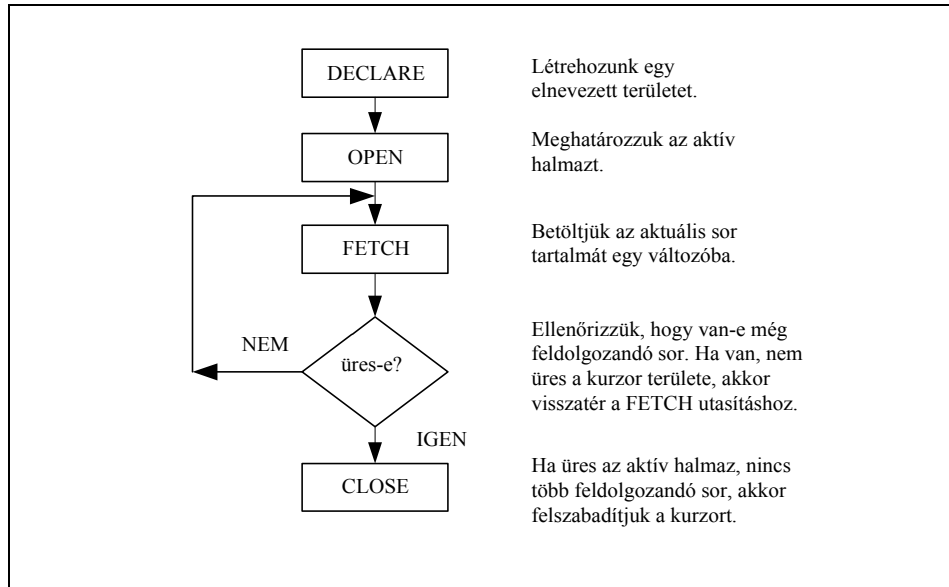
ahol *kurzornév*

az általunk adott hivatkozási név, amely nem egy deklarált PL/SQL változó. Bár értéket a kurzornévhez nem rendelhetünk (csak felhasználhatjuk valamely kifejezésben), azonban elnevezésben, hivatkozásokban a kurzor és a változó ugyanazokat a szabályokat követi. A kurzorokat célszerű az adatbázistáblák után elnevezni.

`SELECT utasítás`

`SELECT utasítás (INTO nélkül).`

Ha fontos a sorok feldolgozási sorrendje, akkor használjuk allekérdezésben az `ORDER BY` utasításrészt.



10.1. ábra A kurzorvezérlő utasítások használatának logikai szerkezete

### Példa

Olvassuk be egyesével az alkalmazottakat az `emp` táblából. Deklaráljunk egy kurzort.

```

DECLARE
CURSOR alkalmazott_kurzor IS
  SELECT ename, job, sal
  FROM emp;

```

Olvassuk be a kereskedő alkalmazottakat az `emp` táblából. Deklaráljunk egy kurzort.

```

DECLARE
CURSOR kereskedo_kurzor IS
  SELECT ename, job, sal
  FROM emp
  WHERE job = 'SALESMAN';

```

### A kurzor megnyitása

A kurzor megnyitásakor történik a kurzorhoz tartozó `SELECT` utasítás végrehajtása, amelynek eredménytáblája az aktív halmaz.

Ha a lekérdezés nem választ ki sorokat (tehát az aktív halmaz üres), akkor a kurzor megnyitó utasítás nem vált ki hibaüzenetet, kivételkezelést (ellentétben az SQL\*Plus környezetben kiadott `SELECT` utasítással, ahol az üres eredménytáblát egy hibaüzenet

követi). A kurzor rendszerfüggvényeinek használatával azonban ellenőrizhetjük, hogy a lekérdezés valóban adott-e vissza sorokat és mennyit.

A kurzor megnyitó utasítás alakja:

`OPEN kurzornév;`

Az `OPEN` egy végrehajtható utasítás, amely a 10.1. ábrán látható módon a `DECLARE CURSOR` és a `FETCH` utasítás között helyezkedik el, és

- dinamikus lefoglal egy területet,
- elemzi a `SELECT` utasítást,
- végrehajtja a kurzorhoz rendelt lekérdezést (a `SELECT` utasítást),
- a visszaadott sorokat egy átmeneti területen tárolja,
- ha a lekérdezésekben voltak paraméterek, akkor az `OPEN` utáni paraméterlistában azok aktuális értékeit adhatjuk meg,
- a mutatót az aktív halmaz első sorára helyezi.

### Fontos

- Mint már korábban jeleztük, a kurzor megnyitása után a kurzorterületen lévő aktív halmaz (a kurzorhoz tartozó `SELECT` utasítás eredménytáblája) nem frissítődik az eredeti adattábla módosításakor. Ennek figyelembevétele igen fontos minden olyan esetben, amikor egy táblasor feldolgozásának eredményeként e sor eredeti tartalma megváltozik, és e módosított tartalomtól függ a további feldolgozás.
- Ha az explicit kurzorban szerepel a `FOR UPDATE` utasításrész, akkor az `OPEN` utasítás zárolja (más felhasználó részére el nem érhetővé teszi) az adattábla feldolgozás alatt álló sorát (record lock). (A `FOR UPDATE` utasításrészt még bővebben tárgyaljuk.)
- Egy `FOR` ciklusban a kurzor megnyitása (`OPEN`), a letöltés (`FETCH`) és a lezárás (`CLOSE`) implicit módon megy végbe (lásd a "Kurzort használó `FOR` ciklusok" pontban!).

### Fetch utasítás

A `FETCH` utasítás egyenként beolvassa az aktív halmazban (a `SELECT` által lekérdezett) szereplő sorokat. A beolvasott értékek elhelyezése megfelelő számú és sorrendű PL/SQL változóba vagy rekordba történik. A sor beolvasása után a kurzorterület mutatója tovább lép.

A `FETCH` utasítás alakja:

`FETCH kurzornév INTO [rekordnév | változó1 [, változó2] ...];`

ahol *kurzornév* a deklarációban szereplő kurzor neve,  
*rekordnév* a beolvasott adatokat tároló rekord neve (deklarálhatjuk a `%ROWTYPE` attribútummal),  
*változó* kimeneti PL/SQL változó(k) az eredmény tárolásához.

### Megjegyzés

- Az `INTO` listájában ugyanannyi változót kell felsorolni, mint a szelekciós listában.
- Rekord használata esetén, a kurzorban is, és a `INTO` listában is rekordra hivatkozhatunk.
- Mindig ellenőrizzük, hogy van-e az aktív halmaznak sora, mert az üres halmaz nem jelez kivételt.

**Példa**

```

FETCH alkalmazott_kurzor
  INTO v_ename, v_job, v_sal

```

**A kurzor bezárása**

A `CLOSE` utasítással lezárjuk a kurzort, az aktív halmaz területe definiálatlanná válik, felszabadul a lefoglalt munkaterület.

Bezárt kurzorból nem lehet beolvasni, ha mégis megpróbáljuk `INVALID_CURSOR` kivétel keletkezik. A nyitott kurzorok maximális száma az alapértelmezés szerint 50. Ezt az `OPEN_CURSORS` paraméter határozza meg.

A kurzor lezárása után a kurzor definíciója nem vész el. Egy újabb `OPEN` kurzor utasítás segítségével mindaddig megnyitható, amíg a kurzor explicit deklarációja érvényes.

A `CLOSE` utasítás alakja:

```
CLOSE kurzornév;
```

**Példa**

A hivatalnokoknak nagyon alacsony a fizetésük, így egyszeri 1000 dolláros jutalomban részesülnek. Mennyi lesz ebben a hónapban a havi jövedelmük? Módosítsuk az `empl` tábla sorait erre az értékre. (`s091.sql`).

```

SET SERVEROUTPUT ON
DECLARE
  v_azonosito emp.empno%TYPE;          -- PL/SQL változó deklaráció
  v_nev       emp.ename%TYPE;
  v_fiz       emp.sal%TYPE;

  CURSOR dolg_kurzor IS                -- kurzor deklaráció
    SELECT empno, ename, sal
    FROM emp
    WHERE job  = 'CLERK';

BEGIN

  OPEN dolg_kurzor;                    -- kurzor megnyitás
  LOOP
    FETCH dolg_kurzor                  -- kurzor beolvasás
      INTO v_azonosito, v_nev, v_fiz;
    EXIT WHEN dolg_kurzor%NOTFOUND;    -- ellenőrzés
    v_fiz := v_fiz +1000;              -- feldolgozás

    UPDATE empl                        -- módosítás
      SET sal = v_fiz
      WHERE job = 'CLERK';
    DBMS_OUTPUT.PUT_LINE(v_azonosito||' '||
                          v_nev||' '||v_fiz||' dollár ');

  END LOOP;
  CLOSE dolg_kurzor;                  -- kurzor bezárás
COMMIT;                              -- véglegesítés
END;
/

```

**Futtatás**

```
SQL> @ C:\a\s091
7369 SMITH 1800 dollár
7876 ADAMS 2100 dollár
7900 JAMES 1950 dollár
7934 MILLER 2300 dollár
```

A PL/SQL eljárás sikeresen befejeződött.

**Kurzorfüggvények használata explicit kurzorok esetén**

Explicit kurzorok esetén a kurzorfüggvényeket a korábban mondottak értelmében használhatjuk. Ha az explicit kurzorral több sort szeretnénk feldolgozni, általában ciklust használunk. A ciklusmag minden egyes végrehajtása visszaad egy sort. Sikertelenség esetén a `%NOTFOUND` függvény értéke `TRUE` lesz. Mielőtt a kurzorra hivatkoznánk egy-egy sor beolvasása után, ellenőrizzük, hogy történt-e valóban beolvasás. Figyeljünk arra, hogy kilépési feltétel hiányában végtelen ciklust kapunk.

**Példa**

```
IF NOT dolgozo_kurzor%ISOPEN
THEN
  OPEN dolgozo_kurzor
  ...
END IF;
LOOP
  FETCH dolgozo_kurzor ...
  ...
```

Ha megadott számú sort szeretnénk beolvasni, használjuk a `%ROWCOUNT` attribútumot.

```
...
LOOP
  FETCH dolgozo_kurzor ...
  INTO ...
  EXIT WHEN dolgozo_kurzor%ROWCOUNT > szám
END LOOP;
...
```

Ha nem tudjuk az aktív kurzorterület méretét (a visszaadott sorok számát), akkor azt kell figyelniünk, hogy sikerült-e újabb sort behozni. A kilépési feltétel ilyenkor:

```
...
EXIT WHEN dolgozo_kurzor%NOTFOUND OR
        dolgozo_kurzor%NOTFOUND IS NULL;
...
```

**Kurzorok és rekordok**

Kurzorból beolvasott sort legkényelmesebben PL/SQL rekordban tárolhatjuk.

Rekordot definiálhatunk explicit kurzor kiválasztott oszlopainak listája alapján. Ekkor a beolvasott sor értékei közvetlenül a rekordba kerülnek.

**Példa**

```

DECLARE
    CURSOR dolgozo_kurzor IS
        SELECT empno, ename, job
        FROM emp;
    dolgozo_rekord      dolgozo_kurzor%ROWTYPE;

BEGIN
    OPEN dolgozo_kurzor;
    LOOP
        FETCH dolgozo_kurzor
        INTO dolgozo_rekord
    ...
    END LOOP;
END;
/

```

**Példa**

Határozzuk meg a felhasználó által megadott részleg dolgozóinak jutalékát, ha a jutalék a fizetés 30 %-a lesz. Írassuk ki a képernyőre (s092.sql).

```

SET SERVEROUTPUT ON
ACCEPT reszlegszam PROMPT 'MELYIK RÉSZLEGET JUTALMAZZUK? '
DECLARE
    v_jutalek      emp.comm%TYPE;
    v_reszlegszam   emp.deptno%TYPE;
    CURSOR dolg_kurzor IS                -- kurzor deklaráció
        SELECT empno, ename, sal, comm
        FROM emp
        WHERE deptno = &reszlegszam;
    dolgozo_rekord  dolg_kurzor%ROWTYPE;  -- rekord deklaráció

BEGIN
    OPEN dolg_kurzor;                    -- megnyitás
    LOOP
        FETCH dolg_kurzor                -- beolvasás
        INTO dolgozo_rekord ;            -- rekord változóba írás
        EXIT WHEN dolg_kurzor%NOTFOUND;  -- kilépés
        v_jutalek := dolgozo_rekord.sal*0.30;  -- feldolgozás

        DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.ename||' '|| v_jutalek);
    END LOOP;
    CLOSE dolg_kurzor;                    -- bezárás
END;
/

```

**Futtassuk le**

```

SQL> @ c:\a\s092
MELYIK RÉSZLEGET JUTALMAZZUK? 20
régi   9:                WHERE deptno = &reszlegszam;
új     9:                WHERE deptno = 20;
régi  25:                WHERE deptno = &reszlegszam;
új    25:                WHERE deptno = 20;

```

```
SMITH  240
JONES  892.5
SCOTT  900
ADAMS  330
FORD   900
A PL/SQL eljárás sikeresen befejeződött.
```

### Példa

Készítsünk egy névtelen PL/SQL blokkot, amely egy gyűjtőtáblát készít az emp tábla dolgozóinak neveiből, és a részlegük telephelyének dept táblabeli neveiből, majd írassuk ki a neveket. A feladatot oldjuk meg kurzorhasználattal.

A program listája:

```
-- s090-2.sql
-- PL/SQL TABLE típus használata 2. --
-- Tábla-létrehozás kurzorral --

SET SERVEROUTPUT ON

DECLARE
  egysor  varchar2(55);
  TYPE tabla_tipus IS TABLE OF egysor%TYPE
    INDEX BY BINARY_INTEGER;
  CURSOR kurzor IS
    SELECT emp.ename || ' ' || dept.loc || '-ból'
      FROM emp, dept
     WHERE emp.deptno=dept.deptno
     ORDER BY emp.ename;
  tablazat  tabla_tipus;
  ind       BINARY_INTEGER;

BEGIN
  -- tábla feltöltése
  ind := 1;
  OPEN kurzor;
  LOOP
    FETCH kurzor
      INTO tablazat(ind);
    EXIT WHEN kurzor%NOTFOUND;
    ind := ind + 1;
  END LOOP;
  CLOSE kurzor;

  -- tábla kiíratása
  ind := tablazat.FIRST;
  WHILE ind IS NOT NULL LOOP
    -- EXIT WHEN NOT EXISTS(ind);
    DBMS_OUTPUT.PUT_LINE('indexszám: ' || ind || ' ' ||
      'tartalma: ' || tablazat(ind));
    ind := tablazat.NEXT(ind);
  END LOOP;
END;
/
```

```

SQL> @s090-2
indexszám: 1      tartalma: ADAMS DALLAS-ból
indexszám: 2      tartalma: ALLEN CHICAGO-ból
indexszám: 3      tartalma: BLAKE CHICAGO-ból
indexszám: 4      tartalma: CLARK NEW YORK-ból
indexszám: 5      tartalma: FORD DALLAS-ból
indexszám: 6      tartalma: JAMES CHICAGO-ból
indexszám: 7      tartalma: JONES DALLAS-ból
indexszám: 8      tartalma: KING NEW YORK-ból
indexszám: 9      tartalma: MARTIN CHICAGO-ból
indexszám: 10     tartalma: MILLER NEW YORK-ból
indexszám: 11     tartalma: SCOTT DALLAS-ból
indexszám: 12     tartalma: SMITH DALLAS-ból
indexszám: 13     tartalma: TURNER CHICAGO-ból
indexszám: 14     tartalma: WARD CHICAGO-ból

```

A PL/SQL eljárás sikeresen befejeződött.

### Példa

Térjünk vissza a korábban vizsgált adattáblákhoz. Emeljük fel a hivatalnokok havi fizetését 20%-kal. Pusztán DML utasításokkal – explicit kurzor nélkül – ez a feladat nem megoldható. A megoldás az s089-2.sql script programban található, amelynek tartalma a következő.

```

SET SERVEROUTPUT ON
DECLARE
  CURSOR dolgozo_kurzor IS          -- kurzor deklaráció
    SELECT ename,sal,job
    FROM emp1
    WHERE job ='CLERK';

  -- tábladeklaráció. Figyeljünk arra, hogy a kurzor után deklaráljuk a
  -- táblát, és teljes adattípus egyezés szükséges
  TYPE dolg_tabla_típus IS TABLE OF
    dolgozo_kurzor%ROWTYPE          -- kurzorral azonos típus
    INDEX BY BINARY_INTEGER;

  nevtabla          dolg_tabla_típus;      -- táblatípusú
  dolgozo_rekord    dolgozo_kurzor%ROWTYPE; -- rekord típusú
  sorszam           NUMBER(2):=1;
  fizetes           emp1.sal%TYPE;

BEGIN
  OPEN dolgozo_kurzor;
  LOOP
    FETCH dolgozo_kurzor
      INTO nevtabla(sorszam);      -- névtáblába teszi a megnyitott kurzort
    EXIT WHEN dolgozo_kurzor%NOTFOUND; -- kilépés
    sorszam := sorszam +1;          -- névtábla indexének növelése
  END LOOP;
  CLOSE dolgozo_kurzor;            -- kurzor bezárása
  DBMS_OUTPUT.PUT_LINE('név      új fizetés    régi fizetés');
  -- fizetés értékének megváltoztatása
  FOR ind IN 1..sorszam-1

```

```

LOOP
    fizetes := nevtabla(ind).sal*1.2;
    DBMS_OUTPUT.PUT_LINE(nevtabla(ind).ename||'    '||
                          fizetés||'    '|| nevtabla(ind).sal);
    -- módosítás
    UPDATE emp1
        SET sal = fizetés
        WHERE sal = nevtabla(ind).sal;
    END LOOP;                                -- ciklus vége
COMMIT;
END;
/

```

### Futtassuk le

```

SQL> @ c:\a\s089-2
név    új fizetés régi fizetés
SMITH   960      800
ADAMS   1320     1100
JAMES   1140      950
MILLER  1560     1300

```

A PL/SQL eljárás sikeresen befejeződött.

### Ellenőrizzük az emp1 tábla tartalmát :

```

SQL> SELECT ename, sal
2      FROM emp1
3      WHERE job = 'CLERK';

```

ENAME	SAL
SMITH	960
ADAMS	1320
JAMES	1140
MILLER	1560

## Kurzort használó FOR ciklusok

Ha egy explicit kurzor sorait FOR ciklusban dolgozzuk fel, akkor ez gyorsabban történik, mint az egyéb ciklusutasítások esetén, mivel a kurzor megnyitása, sorainak betöltése, egyesével való léptetése, és az összes sor feldolgozása utáni bezárása *implicit* módon (tehát felhasználói OPEN, FETCH, és CLOSE utasítás kiadása nélkül) történik.

Az utasítás általános alakja:

```

FOR rekordnév IN kurzornév LOOP
    utasítások
    utasítások
    ...
END LOOP;

```

### Megjegyzés

A rekordot nem kell külön deklarálni, mert deklarálása implicit módon történik. Érvényességi köre kizárólag a ciklusra terjed ki.

**Példa**

Készítsük el a korábbi (s092.sql) példa megoldását FOR ciklussal. Tehát határozzuk meg a felhasználó által megadott részleg dolgozóinak jutalékát, ha a jutalék a fizetés 30 %-a lesz. Írassuk ki a képernyőre. (s093.sql)

```

SET SERVEROUTPUT ON
ACCEPT reszlegszam PROMPT 'MELYIK RÉSZLEGET JUTALMAZZUK? '
DECLARE
    v_jutalek          emp.comm%TYPE;
    CURSOR dolg_kurzor IS                                -- kurzor deklaráció
        SELECT empno, ename, sal
        FROM emp
        WHERE deptno =:&reszlegszam;

BEGIN
    FOR dolgozo_rekord IN dolg_kurzor                    -- implicit megnyitás,
    LOOP                                                  -- rekordba helyezés
        v_jutalek := dolgozo_rekord.sal*0.30;            -- feldolgozás
        DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.ename||'   '||
                               v_jutalek);
    END LOOP;                                            -- implicit bezárás
END;
/

SQL> @ c:\a\s093
MELYIK RÉSZLEGET JUTALMAZZUK? 20
régi    7:                WHERE deptno =:&reszlegszam;
új      7:                WHERE deptno =20;
SMITH   240
JONES   892.5
SCOTT   900
ADAMS   330
FORD    900

```

A PL/SQL eljárás sikeresen befejeződött.

**A KURZORT HASZNÁLÓ FOR CIKLUS ALLEKÉRDEZÉSEL**

A kurzort nem kell deklarálni.

**Példa**

Írassuk ki azokat az alkalmazottakat, akik 1981.szeptember.01 előtt léptek be a céghez.

A megoldást tartalmazó s094.sql script program tartalma:

```

--s094.sql
-- Kurzort használó FOR ciklus allekérdezéssel
-- A kurzort nem kell deklarálni

SET SERVEROUTPUT ON
BEGIN
    FOR dolgozo_rekord IN
        (SELECT ename, job, hiredate
         FROM emp
         WHERE hiredate < '81-SZE-01')

```

```

LOOP
    DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.ename||' '||
                          dolgozo_rekord.hiredate||' '||
                          dolgozo_rekord.job);
END LOOP;                                -- implicit bezárás
END;
/

SQL> @ c:\a\s094
SMITH  80-DEC-17   CLERK
ALLEN  81-FEB-20   SALESMAN
WARD   81-FEB-22   SALESMAN
JONES  81-ÁPR-02   MANAGER
BLAKE  81-MÁJ-01   MANAGER
CLARK  81-JÚN-09   MANAGER

A PL/SQL eljárás sikeresen befejeződött.

```

## Paraméterezett kurzorok

Paraméterek segítségével a kurzornak megnyitáskor adhatunk át olyan értéket, amelyeket futtatása során kívánunk használni. A blokkban az explicit kurzort így egymás után többször is megnyithatjuk és bezárhatjuk, minden egyes alkalommal eltérő aktív halmazt kapunk vissza. A kurzor definíciójában megadott minden formális paraméterhez meg kell adni a megfelelő aktuális paramétert az `OPEN` utasításban.

A paraméterek adattípusai megegyeznek a skaláris változók adattípusaival, de a paraméterek méretét nem adjuk meg. A paraméterek nevére a kurzor lekérdezési kifejezésében hivatkozunk.

Általános alak:

```

CURSOR kurzornév
    [(paraméter_név adattípus [, paraméter_név adattípus]...)]
IS
    select_utasítás;

```

ahol a paraméter szintaxisa:

```
paraméter_név [IN] adattípus [{:= | DEFAULT} kif]
```

A deklarációban megadott sorrend szerint adjuk át a paraméterek értékeit. Az értékek átadása történhet:

- PL/SQL-ből,
- környezeti változóból,
- literálokból.

Ha a feldolgozás kurzorral történik, azaz mindig a megfelelő kurzorral beolvasott sort szeretnénk módosítani, vagy törölni, akkor rögzíteni kell a kurzor által beolvasott aktuális sort, és módosításkor erre a rögzített sorra kell hivatkozni.

Az aktuális kurzor sorának rögzítése, illetve zárolása a `FOR UPDATE` utasításrészsel történik. Az aktuális utasítássorra hivatkozás a `ROWID` pszeudooszlop, vagy a `CURRENT OF` utasításrészsel történhet.

Ha a `FOR` ciklus saját kurzoráról van szó, a paramétereket a kurzor neve után zárójelben adjuk meg.

**Példa**

Jutalmazzuk ismét a felhasználó által megadott osztályt. Használjunk paraméterezett kurzort. Az s095.sql script program tartalma:

```
-- SQL*Plus változó
ACCEPT reszlegyszam PROMPT 'MELYIK RÉSZLEGET JUTALMAZZUK? '
DECLARE
  v_jutalek          emp.comm%TYPE;          -- felhasználói változó
  v_reszlegyszam     emp.deptno%TYPE;        -- felhasználói változó
  CURSOR dolg_kurzor (p_reszlegyszam IN NUMBER) -- paraméterezett kurzor
  IS
    SELECT empno, ename, sal
    FROM emp
    WHERE deptno =p_reszlegyszam;

BEGIN
  v_reszlegyszam := &reszlegyszam;          -- a felhasználói változó értéket kap
-- paraméterezett kurzor aktivizálás
  FOR dolgozo_rekord IN dolg_kurzor (v_reszlegyszam) -- aktuális paraméter
  LOOP
    v_jutalek := dolgozo_rekord.sal*0.30;      -- feldolgozás
    DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.ename||' '|| -- kiiratás
                          v_jutalek);
  END LOOP;
END;
/

SQL> @ c:\a\s095
MELYIK RÉSZLEGET JUTALMAZZUK? 10
régi 11:      v_reszlegyszam := &reszlegyszam;
új 11:      v_reszlegyszam := 10;
CLARK  735
KING  1500
MILLER 390
```

**ROWID pszeudooszlop**

Ha a feldolgozás kurzorral történik, azaz mindig a megfelelő kurzor által beolvasott sort szeretnénk módosítani, vagy törölni, (DML utasítások), akkor *rögzíteni* kell a kurzor által beolvasott aktuális sor pozícióját, és a módosításkor erre a rögzített sorra kell hivatkozni.

Az aktuális kurzor sorának rögzítése, illetve zárolása a **FOR UPDATE** utasításrésszel történik. Az aktuális utasítássorra hivatkozás a **ROWID pszeudooszlop**, vagy a **CURRENT OF** utasításrésszel történhet.

A pszeudooszlop egy olyan "oszlop", amely bármikor lekérdezhető, a lekérdezésnél egy konkrét értéket szolgáltat, de fizikailag egy tábla sem tartalmazza.

Az Oracle pszeudooszlopaihoz tartozik:

ROWID	Az Oracle által minden táblához automatikusan generált sorazonosító. Ha az UPDATE ... WHERE és SELECT ... FOR UPDATE utasításokban a ROWID-t használjuk, akkor ez biztosítja számunkra, hogy a megfelelő sorok módosulnak.
ROWNUM	Megadja a sor sorszámát.

<code>SYSDATE</code>	Az aktuális rendszeridőt és a dátumot adja meg.
<code>UID</code>	Megadja az aktuális felhasználó azonosítóját.
<code>USER</code>	Az aktuális felhasználó nevét adja meg.

`ROWID` *adattípus* egy sort egy Oracle adatbázisban egyedien azonosító érték tárolására képes típus. A `ROWID` pszeudooszlop is ilyen egyedien azonosító értéket szolgáltat. (Ekvivalens a `VARCHAR2(18)`-al.)

A `ROWID` pszeudooszlop tartalmazza a sor logikai címét. A logikai cím adatbázisszinten egyedi.

A `ROWID` egy `SELECT` utasításban lekérdezhető, használható a `WHERE` utasításrészben, de semmiképpen nem változtatható meg. A `WHERE` utasításrészben megadva, a sor gyors elérését biztosítja. A sor logikai címe megváltozhat, ha a sort tartalmazó táblát exportáljuk, majd importáljuk.

### Példa

Kérdezzük le az `emp` tábla valamelyik alkalmazottjának `ROWID` számát. (s096.sql)

```
SET SERVEROUTPUT ON;
ACCEPT sor_azon PROMPT 'Kinek a sorazonosítójára kíváncsi, a nevét kérem '
DECLARE
    sorazonosito    VARCHAR2(20);          -- A ROWID értéket tárolja
    adat            emp.ename%TYPE;
    jelolt_sor      VARCHAR2(40);

BEGIN
    adat := '&sor_azon';
    SELECT ROWID
    INTO sorazonosito
    FROM emp
    WHERE ename = adat;

    DBMS_OUTPUT.PUT_LINE('sorkijelölő adat:      ' || adat);

    SELECT ename
    INTO jelolt_sor
    FROM emp
    WHERE ROWID = sorazonosito;

    DBMS_OUTPUT.PUT_LINE('sorazonosító:      ' || sorazonosito);
    DBMS_OUTPUT.PUT_LINE('jelolt_sor:      ' || jelolt_sor);
END;
/

SQL> @ c:\a\s096
Kinek a sorazonosítójára kíváncsi, a nevét kérem SCOTT
régi 7:  adat := '&sor_azon';
új 7:  adat := 'SCOTT';
sorkijelölő adat:      SCOTT
sorazonosító:      AAAFrHAADAAADuAAH
jelolt_sor:      SCOTT
```

A PL/SQL eljárás sikeresen befejeződött.

Néhány további futtatás, az érdekesség kedvéért:

```
sorkijelölő adat:      FORD
sorazonosító:         AAAFrHAADAAAADuAAM
jelolt_sor:           FORD

sorkijelölő adat:      SMITH
sorazonosító:         AAAFrHAADAAAADuAAA
jelolt_sor:           SMITH
```

## FOR UPDATE utasításrész

Az UPDATE PL/SQL változata az SQL változattal azonos módon működik, annyi különbséggel, hogy itt a WHERE utasításrészben megadható a CURRENT OF opció, amely csak a megadott kurzor utoljára lehívott sorát módosítja. A CURRENT OF opció használatához a kurzorhoz rendelt lekérdezésben meg kell adni a FOR UPDATE utasításrészt.

A FOR UPDATE lezárja a sorazonosítót ott, ahol módosítani (UPDATE), törölni (DELETE) szeretnénk. Az Oracle rendszer a tranzakció befejezésekor feloldja a zárat.

Általános alak:

```
SELECT ...
FROM ...
...
FOR UPDATE [OF oszlophivatkozás] [NOWAIT];
```

ahol

*oszlophivatkozás* oszlop abban a táblában, amelyben a lekérdezést végrehajtjuk (oszlopok listája is használható).

NOWAIT Oracle hibát okoz, ha a sorokat egy másik felhasználó zárta.

A FOR UPDATE a sorok módosítását a tranzakció ideje alatt zároló utasításrész a SELECT utasítás legutolsó utasításrésze az ORDER BY után.

A FOR UPDATE utasításrészt a CURRENT OF használata előtt a kurzor deklarálásánál használni kell.

## CURRENT OF utasításrész

Ha az explicit kurzor aktuális sorára hivatkozunk, a WHERE CURRENT OF utasításrészt kell használnunk. Ezáltal módosítani és törölni is tudjuk az aktuális sort, anélkül, hogy explicit módon hivatkoznánk a ROWID pseudooszlopra.

Általános alak

```
WHERE CURRENT OF kurzor;
```

ahol:

*kurzor* a deklarált kurzor neve.

Ügyeljünk arra, hogy ciklusban ne adjuk ki a COMMIT utasítást, mivel a FOR UPDATE zárolja a sorokat. Ha megnyitjuk a kurzort, akkor a feldolgozás alatt álló sor zárva van, és ez a zár csak a tranzakció bejezésével (módosítás, törlés) kerül feloldásra. Ha közben a COMMIT utasítást kiadjuk, nem lehet a FETCH utasítással új sort behívni, mert a PL/SQL hibaüzenetet

ad. Ha mi szeretnénk vezérelni COMMIT utasítással, akkor ne használjuk a FOR UPDATE és CURRENT utasításrészt, hanem csak a ROWID pseudooszlopot.

### Példa

Módosítsuk az emp1 tábla felhasználó által megadott foglalkozású dolgozóinak jutalékát a fizetésük 30 %-ával. (s097.sql)

```
ACCEPT g_foglalkozas PROMPT 'Kérem a jutalomban részesülő részleg nevét: '
```

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    v_azon      emp.empno%TYPE;
    v_nev       emp.ename%TYPE;
    v_fiz       emp.sal%TYPE;
    v_jutalom   emp.comm%TYPE;
```

```
-- Paraméterezett kurzor deklaráció
```

```
CURSOR dolg_kurzor(p_foglalkozas CHAR) IS
```

```
    SELECT empno, ename, sal, comm
```

```
    FROM emp1
```

```
    WHERE job = p_foglalkozas
```

```
    FOR UPDATE OF comm NOWAIT;
```

```
-- kurzor rögzítése
```

```
BEGIN
```

```
    v_jutalom :=0;
```

```
    OPEN dolg_kurzor('&g_foglalkozas');
```

```
-- kurzor megnyitása
```

```
    LOOP
```

```
        FETCH dolg_kurzor
```

```
-- sorok beolvasása
```

```
        INTO v_azon, v_nev, v_fiz, v_jutalom;
```

```
-- változókba
```

```
        EXIT WHEN dolg_kurzor%NOTFOUND;
```

```
-- kilépési feltétel
```

```
        v_jutalom := NVL(v_jutalom,0) + v_fiz*0.3;
```

```
-- feldolgozás
```

```
-- Módosítás (DML utasítás), ezért kell a kurzort rögzíteni
```

```
    UPDATE emp1
```

```
        SET comm = v_jutalom
```

```
        WHERE CURRENT of dolg_kurzor; --csak az aktuális sort módosítja
```

```
-- Kiíratás a képernyőre
```

```
    DBMS_OUTPUT.PUT_LINE(v_azon||' '||v_nev || ' '||
```

```
        v_jutalom||' dollár ');
```

```
END LOOP;
```

```
    CLOSE dolg_kurzor;
```

```
-- A kurzor lezárása
```

```
COMMIT;
```

```
END;
```

```
/
```

```
SQL> @ c:\a\s097
```

```
Kérem a jutalomban részesülő részleg nevét: CLERK
```

```
régi 15: OPEN dolg_kurzor('&g_foglalkozas');
```

```
új 15: OPEN dolg_kurzor('CLERK');
```

```
7369 SMITH 240 dollár
```

```
7876 ADAMS 330 dollár
```

```
7900 JAMES 285 dollár
7934 MILLER 390 dollár
```

A PL/SQL eljárás sikeresen befejeződött.

### Ellenőrzés lekérdezéssel

```
SQL> SELECT empno, ename, comm
2      FROM emp1
3      WHERE job ='CLERK';
```

EMPNO	ENAME	COMM
7369	SMITH	240
7876	ADAMS	330
7900	JAMES	285
7934	MILLER	390

### Példa

(A ROWID használatának bemutatása.)

Módosítsuk az éppen lekérdezett dolgozó fizetését 10%-kal. (s098.sql)

```
SET SERVEROUTPUT ON
DECLARE                                     -- deklarációk
    CURSOR dolg_kurzor IS
        SELECT ename, job, ROWID          -- szelekciós listában a ROWID
        FROM emp1;
    -- változó deklarációk
    saját_név          emp.ename%TYPE;
    saját_foglalkozás   emp.job%TYPE;
    saját_ROWID         ROWID;

-- végrehajtható utasítások
BEGIN
    -- a fejléc kiírása
    DBMS_OUTPUT.PUT_LINE(' saját azonosító      név      ');
    OPEN dolg_kurzor;                      -- kurzor megnyitása
    LOOP                                  -- ciklus indítása
        FETCH dolg_kurzor                 -- egy sor beolvasása
        INTO saját_név, saját_foglalkozás, saját_ROWID;
        EXIT WHEN dolg_kurzor%NOTFOUND;   -- kilépési feltétel
        -- nyomkövetés
        DBMS_OUTPUT.PUT_LINE(saját_ROWID || '      ' || saját_név);

    UPDATE emp1                          --DML utasítás
        SET sal = sal* 1.1
        WHERE ROWID = saját_ROWID;       -- sor azonosítása
    END LOOP;                            -- ciklus vége
    CLOSE dolg_kurzor;                   -- kurzor bezárása
END;                                     -- blokk vége
/
```

Futtatás után:

saját azonosító	név
AAAFtgAADAAAACPAAB	SMITH
AAAFtgAADAAAACPAAB	ALLEN
AAAFtgAADAAAACPAAC	WARD
AAAFtgAADAAAACPAAD	JONES
AAAFtgAADAAAACPAAE	MARTIN
AAAFtgAADAAAACPAAF	BLAKE
AAAFtgAADAAAACPAAG	CLARK
AAAFtgAADAAAACPAAH	SCOTT
AAAFtgAADAAAACPAAI	KING
AAAFtgAADAAAACPAAJ	TURNER
AAAFtgAADAAAACPAAK	ADAMS
AAAFtgAADAAAACPAAL	JAMES
AAAFtgAADAAAACPAAM	FORD
AAAFtgAADAAAACPAAN	MILLER

A PL/SQL eljárás sikeresen befejeződött.

Ellenőrizzük

```
SQL> SELECT ename, sal
      2      FROM emp1;
```

ENAME	SAL
-----	-----
SMITH	880
ALLEN	1760
WARD	1375
...	

14 sor kijelölve.

A módosítás soronként megtörtént.

## Allekérdezést tartalmazó kurzorok

Allekérdezés fogalmát az SQL `SELECT` utasítás tárgyalásakor bemutattuk, és olyan lekérdezést értettünk rajta, amely egy másik (külső) SQL adatkezelési utasításban szerepel. Kiértékelésekor az allekérdezés egy értéket vagy értékhalmozatot ad át a külső utasításnak.

Gyakran használunk allekérdezést a `WHERE` utasításrészben. A `FROM` utasításrészben alkalmazva ideiglenes adatforrást készíthetünk az adott lekérdezéshez (ezt inline nézetnek neveztük).

### Példa

Írassuk ki azokat, akiknek fizetése kisebb, mint az összes dolgozó átlagfizetése. Az allekérdezés a `WHERE` utasításrész feltételében van. (`s100.sql`)

```
SET SERVEROUTPUT ON
DECLARE
    nev          emp.ename%TYPE;
```

```

fizetes    emp.sal%TYPE;

CURSOR dolg_kurzor IS
  SELECT ename, sal
    FROM emp
   WHERE sal < (SELECT AVG(sal)
                FROM emp)
   ORDER BY ename;
-- allekérdezés
--rendezés

BEGIN
  DBMS_OUTPUT.PUT_LINE('név          fizetés');
  DBMS_OUTPUT.PUT_LINE('-----');
  OPEN dolg_kurzor ;
  LOOP
    FETCH dolg_kurzor
      INTO nev, fizetes;
    EXIT WHEN dolg_kurzor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(nev||'      '||fizetes);
  END LOOP;
END;
/

SQL> @ c:\a\s100
Az input csonkolva 1 karakterre
név          fizetés
-----
ADAMS         1100
ALLEN         1600
JAMES         950
MARTIN        1250
MILLER        1300
SMITH         800
TURNER        1500
WARD          1250

```

A PL/SQL eljárás sikeresen befejeződött.

### Példa

Az egyes dolgozók fizetése mennyivel tér el csoportjuk átlagfizetésétől? Allekérdezés a `kurzor` FROM utasításrészében. (s099.sql)

```

SET SERVEROUTPUT ON
DECLARE
  nev          emp.ename%TYPE;
  fizetes      emp.sal%TYPE;
  v_atlag      NUMBER;
  elteres      NUMBER;

  CURSOR dolg_kurzor IS
    SELECT e.ename, e.sal, al.atlag
      FROM emp e,
           (SELECT ROUND(AVG(sal)) AS atlag, deptno
            FROM emp
            GROUP BY deptno) al

```

```

        WHERE e.deptno = al.deptno;
BEGIN
    DBMS_OUTPUT.PUT_LINE('név          eltérés');
    DBMS_OUTPUT.PUT_LINE('-----');
    OPEN dolg_kurzor ;
    LOOP
        FETCH dolg_kurzor
            INTO nev, fizetes, v_atlag;
        EXIT WHEN dolg_kurzor%NOTFOUND;
        elteres := v_atlag-fizetes;          -- különbség képzése
        DBMS_OUTPUT.PUT_LINE(nev||'      '||elteres);
    END LOOP;
END;
/

```

```
SQL> @ c:\a\s099
```

Az input csonkolva 1 karakterre

```
név          eltérés
```

```
-----
```

```

CLARK      467
KING      -2083
MILLER     1617
SMITH     1375
ADAMS     1075
FORD      -825
SCOTT     -825
JONES     -800
ALLEN     -33
BLAKE     -1283
MARTIN     317
JAMES     617
TURNER     67
WARD      317

```

A PL/SQL eljárás sikeresen befejeződött.

## Példák a kurzor használatára

A következő három feladat a dolgozo táblára vonatkozik, létrehozása az s110-1.sql script program lefuttatásával valósítható meg.

```
SQL> SELECT * FROM dolgozo;
```

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ
-----	-----	-----	-----	-----	-----
001	Kocsis	Endre	98-AUG-02	KF	48000
002	Nagy	Ida	99-AUG-06	RTV	36000
007	Kiss	Ferenc	98-SZE-07	HT	56000
112	Varga	Ede	98-AUG-12	RTV	72000
324	Balogh	Zsolt	99-FEB-03	RTV	65000
722	Szeles	Emese	98-MÁJ-13	HT	68000
854	Kiss	Paula	98-AUG-12	HT	42000
652	Sas	Elek	97-MÁR-06	RTV	70000
712	Vas	Erika	98-MÁJ-07	FF	66000

003    Havas            Tibor            97-MÁJ-03 FF            80000

10 sor kijelölve.

### 1.FELADAT

- A dolgozó táblát bővítjük egy új, változtatható méretű karakteres oszloppal, amelynek neve legyen `csillag`, a deklarált hossza pedig 20.
- Készítsünk egy PL/SQL blokkot, amely annyi csillagot (\*) helyez a `csillag` oszlopba, ahányszor 10000-el osztható a dolgozó havi fizetése (matematikai kerekítést alkalmazunk).
  - 1.1. Kérjünk be egy részleg nevet SQL\*Plus helyettesítő változóval.
  - 1.2. Deklaráljunk egy változót, amely a csillagokat fogja tartalmazni.
  - 1.3. 10000 Ft-onként fűzzünk egy újabb csillagot a karakterlánchoz.
  - 1.4. Módosítsuk a bekért részleg dolgozóinak, azaz a dolgozó tábla megfelelő sorainak `csillag` oszlopát a csillagokat tartalmazó karakterlánccal.
  - 1.5. Hagyjuk jóvá a változásokat.

### Megoldás

- 1.részfeladat

```
ALTER TABLE dolgozo
  ADD csillag VARCHAR2(20);
```

- 2. részfeladat (s111.sql)

```
SET SERVEROUTPUT ON
SET VERIFY OFF
```

```
ACCEPT g_reszleg PROMPT 'Adja meg a részleg nevét: '
```

```
DECLARE
  v_csillag          dolgozo.csillag%TYPE;
  v_csillagszam      number(4);
  v_reszlegnev       dolgozo.reszl%TYPE;

  CURSOR dolgozo_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
    SELECT vnev,fiz,csillag
      FROM dolgozo
     WHERE reszl = p_reszleg
    FOR UPDATE OF csillag NOWAIT;
```

```
csillag_rekord      dolgozo_kurzor%ROWTYPE;
```

```
BEGIN
  v_reszlegnev := UPPER('&g_reszleg');
  OPEN dolgozo_kurzor (v_reszlegnev);
  LOOP
    v_csillag := '';
    FETCH dolgozo_kurzor
      INTO csillag_rekord;
```

```

EXIT WHEN dolgozo_kurzor%NOTFOUND;
v_csillagszam := TRUNC(csillag_rekord.fiz/10000);
v_csillag := '';
FOR n IN 1..v_csillagszam
    LOOP
        v_csillag := CONCAT(' ', v_csillag);
    END LOOP;
DBMS_OUTPUT.PUT_LINE(csillag_rekord.vnev||csillag_rekord.fiz||
                    v_csillag);

UPDATE dolgozo
    SET csillag = v_csillag
    WHERE CURRENT OF dolgozo_kurzor;
END LOOP;
CLOSE dolgozo_kurzor;
COMMIT;
END;
/

```

Futtassuk le.

```

SQL> @c:\a\s111.sql
Adja meg a(z) g_reszleg értékét: rtv
Nagy    36000    ***
Varga   72000    *****
Balogh  65000    *****
Sas     70000    *****

```

A PL/SQL eljárás sikeresen befejeződött.

```
SQL> select * from dolgozo;
```

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ	CSILLAG
001	Kocsis	Endre	98-AUG-02	KF	48000	
002	Nagy	Ida	99-AUG-06	RTV	36000	***
007	Kiss	Ferenc	98-SZE-07	HT	56000	
112	Varga	Ede	98-AUG-12	RTV	72000	*****
324	Balogh	Zsolt	99-FEB-03	RTV	65000	*****
722	Szeles	Emese	98-MÁJ-13	HT	68000	
854	Kiss	Paula	98-AUG-12	HT	42000	
652	Sas	Elek	97-MÁR-06	RTV	70000	*****
712	Vas	Erika	98-MÁJ-07	FF	66000	
003	Havas	Tibor	97-MÁJ-03	FF	80000	

10 sor kijelölve.

## 2.FELADAT

- A dolgozo táblát bővítsük egy új, változtatható méretű karakteres oszloppal, amelynek neve legyen csillag, a deklarált hossza pedig 20.
- Készítsünk egy PL/SQL blokkot, amely a felhasználó által bekért részleg minden dolgozójának a csillag nevű oszlopába annyi csillagot helyez, ahány részlegbeli kollégájának a fizetése  $\pm 10000$  Ft-tal eltér a sajátjához képest (természetesen saját magát ne számítsa bele). A módosított dolgozo tábla adott részlegbeli dolgozókból álló résztábláját írassuk ki a képernyőre.

*Megoldás*

## ▪ 1.részfeladat

```
ALTER TABLE dolgozo
  ADD csillag VARCHAR2(20);
```

## ▪ 2. részfeladat (s112.sql)

```
SET SERVEROUTPUT ON
SET VERIFY ON
```

```
-- Az esetleges korábbi próbafuttatások eredményének törlése
UPDATE dolgozo
  SET csillag=NULL;
```

```
ACCEPT g_reszleg PROMPT 'Adja meg a részleg nevét: '
```

```
DECLARE
  v_csillag          dolgozo.csillag%Type;
  v_csillagszam      number(4);
  v_reszlegnev       dolgozo.reszl%Type;
  CURSOR dolgozo_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
    SELECT vnev,fiz,csillag
      FROM dolgozo
     WHERE reszl = p_reszleg
   FOR UPDATE OF csillag NOWAIT;
  csillag_rekord     dolgozo_kurzor%ROWTYPE;
BEGIN
  v_reszlegnev := UPPER('&g_reszleg');
  OPEN dolgozo_kurzor (v_reszlegnev);
  LOOP
    v_csillag := '';
    FETCH dolgozo_kurzor
      INTO csillag_rekord;
    EXIT WHEN dolgozo_kurzor%NOTFOUND;
    SELECT COUNT (*)
      INTO v_csillagszam
    FROM dolgozo
   WHERE (vnev <> csillag_rekord.vnev) AND -- nem azonos önmagával
        (reszl = v_reszlegnev) AND
        -- a két fizetési határ között kell lennie
        ((fiz <= csillag_rekord.fiz + 10000) AND
         (fiz >= csillag_rekord.fiz - 10000));
    -- Részkiíratás
    DBMS_OUTPUT.PUT_LINE('-- Részkiíratás: '|| csillag_rekord.vnev||
                          '   '||csillag_rekord.fiz||
                          '   csillagszam: '||v_csillagszam);

    v_csillag := '';
    FOR n IN 1.. v_csillagszam
      LOOP
        v_csillag:=CONCAT('*', v_csillag);
      END LOOP;

    DBMS_OUTPUT.PUT_LINE(csillag_rekord.vnev||'   '||csillag_rekord.fiz||
```

```

        '    '||v_csillag);

    UPDATE dolgozo
        SET csillag = v_csillag
        WHERE CURRENT OF dolgozo_kurzor;
    END LOOP;
    CLOSE dolgozo_kurzor;
    COMMIT;
END;
/

```

SQL> @ c:\a\s112

10 sor módosítva.

```

Adja meg a részleg nevét rtv
régi 12:  v_reszlegnev := UPPER('&g_reszleg');
új 12:  v_reszlegnev := UPPER('rtv');
-- Részkiíratás: Nagy 36000 csillagszam: 0
Nagy 36000
-- Részkiíratás: Varga 72000 csillagszam: 2
Varga 72000 **
-- Részkiíratás: Balogh 65000 csillagszam: 2
Balogh 65000 **
-- Részkiíratás: Sas 70000 csillagszam: 2
Sas 70000 **

```

A PL/SQL eljárás sikeresen befejeződött.

Listázzuk ki a dolgozo tábla tartalmát !

SQL> SELECT \* FROM dolgozo;

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ CSILLAG
001	Kocsis	Endre	98-AUG-02	KF	48000
002	Nagy	Ida	99-AUG-06	RTV	36000
007	Kiss	Ferenc	98-SZE-07	HT	56000
112	Varga	Ede	98-AUG-12	RTV	72000 **
324	Balogh	Zsolt	99-FEB-03	RTV	65000 **
722	Szeles	Emese	98-MÁJ-13	HT	68000
854	Kiss	Paula	98-AUG-12	HT	42000
652	Sas	Elek	97-MÁR-06	RTV	70000 **
712	Vas	Erika	98-MÁJ-07	FF	66000
003	Havas	Tibor	97-MÁJ-03	FF	80000

10 sor kijelölve.

### 3.FELADAT (PÁROSÍTÁSI FELADAT)

- A dolgozo táblát bővítjük egy új, változtatható méretű karakteres oszloppal, amelynek neve legyen `partner`, deklarált hossza 20, inicializált értéke pedig `'---'`.
- Készítsünk egy PL/SQL blokkot, amely a felhasználó által bekért részleg minden dolgozójának a `partner` nevű oszlopába beírja egy olyan azonos részlegbeli kollégájának a nevét, akinek fizetése  $\pm 10000$  Ft a sajátjához képest és partnerként

még nincs senkihez sem párosítva (természetesen saját magát ne számítsa bele). Ezzel egyidejűleg a megtalált kolléga `partner` nevű oszlopába beírja a vizsgált dolgozó nevét. A módosított `dolgozo` tábla adott részlegbeli dolgozókból álló résztábláját írassuk ki a képernyőre.

### Megjegyzés

Az bemutatásra kerülő megoldásokban szeretnénk a kurzor használatának logikáját bemutatni. Ennek érdekében két olyan megoldást is bemutatunk, amelyek ugyan hibásak, ám mégis igen tanulságosak, mivel a bennük szereplő hibák egyáltalán nem nyilvánvalóak, mint ahogy a jó megoldáshoz vezető út is sok mellékesnek tűnő megfontolást igényel.

## 1. részfeladat

```
ALTER TABLE dolgozo
ADD partner VARCHAR2(10) DEFAULT '---';
```

## 2. részfeladat

### 1. Megoldás (helytelen működés)

```
-----
-- s113-1.sql -- nem működik jól
-----

SET SERVEROUTPUT ON
SET VERIFY ON

ACCEPT g_reszleg PROMPT 'Adja meg a részleg nevét: '

DECLARE
    v_reszlegnev      dolgozo.reszl%TYPE;
    CURSOR dolgozo_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
        SELECT vnev, fiz, partner
        FROM dolgozo
        WHERE (reszl = p_reszleg) and
              (partner = '---')
    FOR UPDATE OF partner NOWAIT;
    dolgozo_rekord    dolgozo_kurzor%ROWTYPE;
    CURSOR partner_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
        SELECT vnev, fiz, partner
        FROM dolgozo
        WHERE (reszl = p_reszleg) and
              (partner = '---')
    FOR UPDATE OF partner NOWAIT;
    partner_rekord    dolgozo_kurzor%ROWTYPE;

BEGIN
    v_reszlegnev := UPPER('&g_reszleg');
    OPEN dolgozo_kurzor(v_reszlegnev);
    LOOP
        FETCH dolgozo_kurzor
        INTO dolgozo_rekord;
        EXIT WHEN dolgozo_kurzor %NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||' ciklusa: ');
        OPEN partner_kurzor(v_reszlegnev);
```

```

LOOP
  FETCH partner_kurzor
    INTO partner_rekord;
  EXIT WHEN partner_kurzor%NOTFOUND;
  IF (partner_rekord.vnev <> dolgozo_rekord.vnev) and
    (dolgozo_rekord.fiz <= partner_rekord.fiz + 10000) and
    (dolgozo_rekord.fiz >= partner_rekord.fiz - 10000)
  THEN
    DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||
      ' - '||partner_rekord.vnev);

    UPDATE dolgozo
      SET partner = partner_rekord.vnev
      WHERE CURRENT OF dolgozo_kurzor;
    UPDATE dolgozo
      SET partner = dolgozo_rekord.vnev
      WHERE CURRENT OF partner_kurzor;
    EXIT;
  END IF;
END LOOP;
CLOSE partner_kurzor;
DBMS_OUTPUT.PUT_LINE('----- '||dolgozo_rekord.vnev||
  ' ciklusának vége');

END LOOP;
CLOSE dolgozo_kurzor;
COMMIT
END;
/

```

```

SQL> @ C:\A\s113-1
Adja meg a részleg nevét: rtv
Nagy ciklusa:
----- Nagy ciklusának vége
Varga ciklusa:
--- - ---
Varga - Balogh
----- Varga ciklusának vége
Balogh ciklusa:
--- - ---
Balogh - Sas
----- Balogh ciklusának vége
Sas ciklusa:
----- Sas ciklusának vége

```

A PL/SQL eljárás sikeresen befejeződött.

## 2. Megoldás (helytelen működés)

```

-----
-- s113-2.sql -- nem működik jól
-----

```

```

SET SERVEROUTPUT ON
SET VERIFY ON

```

```

ACCEPT g_reszleg PROMPT 'Adja meg a részleg nevét: '

```

```

DECLARE

```

```

v_reszlegnev      dolgozo.reszl%TYPE;
CURSOR dolgozo_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
  SELECT vnev, fiz, partner, ROWID sorazonosito
    FROM dolgozo
   WHERE (reszl = p_reszleg) and
         (partner = '---')
  FOR UPDATE OF partner NOWAIT;
dolgozo_rekord    dolgozo_kurzor%ROWTYPE;
CURSOR partner_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
  SELECT vnev, fiz, partner, ROWID sorazonosito
    FROM dolgozo
   WHERE (reszl = p_reszleg) and
         (partner = '---')
  FOR UPDATE OF partner NOWAIT;
partner_rekord    dolgozo_kurzor%ROWTYPE;

BEGIN
  v_reszlegnev := UPPER('&g_reszleg');
  OPEN dolgozo_kurzor(v_reszlegnev);
  LOOP
    FETCH dolgozo_kurzor
      INTO dolgozo_rekord;
    EXIT WHEN dolgozo_kurzor %NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||' ciklusa: ');
    OPEN partner_kurzor(v_reszlegnev);
    LOOP
      FETCH partner_kurzor
        INTO partner_rekord;
      EXIT WHEN partner_kurzor%NOTFOUND;
      IF (partner_rekord.vnev <> dolgozo_rekord.vnev) and
         (dolgozo_rekord.fiz <= partner_rekord.fiz + 10000) and
         (dolgozo_rekord.fiz >= partner_rekord.fiz - 10000)
      THEN
        DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||
                              ' - '||partner_rekord.vnev);

        UPDATE dolgozo
          SET partner = partner_rekord.vnev
         WHERE ROWID = dolgozo_rekord.sorazonosito;
        UPDATE dolgozo
          SET partner = dolgozo_rekord.vnev
         WHERE ROWID = partner_rekord.sorazonosito;
        EXIT;
      END IF;
    END LOOP;
    CLOSE partner_kurzor;
    DBMS_OUTPUT.PUT_LINE('----- '||dolgozo_rekord.vnev||
                          ' ciklusának vége');

  END LOOP;
  CLOSE dolgozo_kurzor;
  COMMIT;
END;
/

=====
-- A rossz működésekhez tartozó futási eredmények --

```

```
=====
```

```
SQL> UPDATE dolgozo
      SET partner = '---';
11 sor módosítva.
```

```
SQL> SELECT * FROM dolgozo;
```

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ	PARTNER
001	Kocsis	Endre	98-Aug-02	KF	48000	---
002	Nagy	Ida	99-Aug-06	RTV	36000	---
007	Kiss	Ferenc	98-Sze-07	HT	56000	---
112	Varga	Ede	98-Aug-12	RTV	72000	---
324	Balogh	Zsolt	99-Feb-03	RTV	65000	---
722	Szeles	Emese	98-Máj-13	HT	68000	---
854	Kiss	Paula	98-Aug-12	HT	42000	---
652	Sas	Elek	97-Már-06	RTV	70000	---
712	Vas	Erika	98-Máj-07	FF	66000	---
003	Havas	Tibor	97-Máj-03	FF	80000	---
003	Havas	Tibor	97-Máj-03	FF	80000	---

```
11 sor kijelölve.
```

```
SQL> SELECT * FROM dolgozo;
```

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ	PARTNER
001	Kocsis	Endre	98-Aug-02	KF	48000	---
002	Nagy	Ida	99-Aug-06	RTV	36000	---
007	Kiss	Ferenc	98-Sze-07	HT	56000	---
112	Varga	Ede	98-Aug-12	RTV	72000	Balogh
324	Balogh	Zsolt	99-Feb-03	RTV	65000	Sas
722	Szeles	Emese	98-Máj-13	HT	68000	---
854	Kiss	Paula	98-Aug-12	HT	42000	---
652	Sas	Elek	97-Már-06	RTV	70000	Balogh
712	Vas	Erika	98-Máj-07	FF	66000	---
003	Havas	Tibor	97-Máj-03	FF	80000	---
003	Havas	Tibor	97-Máj-03	FF	80000	---

```
11 sor kijelölve.
```

### Megjegyzés

Feltehető a kérdés, valóban helytelenek-e a fenti megoldások, és ha igen, akkor mi az oka a helytelen működésnek.

A megoldások (a két fenti program azonos megoldást ad!) valóban helytelenek. Bár a kiválasztott személyekre a részlegre és fizetésre vonatkozó feltételek teljesülnek, nyilván nem állhatnak partneri kapcsolatban a fenti módon, hiszen ha például Vargának partnere Balogh, akkor Baloghnak Varga kell, hogy legyen a partnere, és nem pedig Sas.

Mi az oka e hibának? Az alapvető ok az (és ez egyben a kurzorhasználat csapdája), hogy mivel a kurzorterület nem frissül (vagyis az adattáblán bekövetkező változások nem jelennek meg a már megnyitott aktív halmazon, a kurzorterületen), így a kurzorváltó által behozott (a kurzorterületről kapott) adatok nem mindig az aktuális állapotot tükrözik.

Nevezhetnénk ezt a jelenséget ”kurzorvakságnak”. Mindez nem okoz problémát mindaddig, míg az aktuálisan kijelölt rekord feldolgozásához szükséges adatok a feldolgozás folyamán nem változnak. Amellett azonban egy rekord feldolgozása kihat az éppen vizsgált tábla más rekordjára is, és egyben a feldolgozáshoz szintén szükség van más rekordok tartalmára, a ”kurzorvakság” miatt a feldolgozás az adattábla korábbi állapotára vonatkozik csupán, és így hibás lesz.

### 3. Megoldás (helyes működés)

```
-----
-- s113-3.sql -- JÓL MŰKÖDIK
-----

SET SERVEROUTPUT ON
SET VERIFY ON

-- ALTER TABLE dolgozo
--   ADD partner VARCHAR2(10) DEFAULT '---';

-- Az esetleges korábbi próbafuttatások eredményének törlése
UPDATE dolgozo
  SET partner = '---';

ACCEPT g_reszleg PROMPT 'Adja meg a részleg nevét: '

DECLARE
  v_reszlegnev      dolgozo.reszl%TYPE;
  v_dolg_partner    dolgozo.partner%TYPE;
  v_part_partner    dolgozo.partner%TYPE;
  CURSOR dolgozo_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
    SELECT vnev, fiz, ROWID sorazonosito
      FROM dolgozo
     WHERE (reszl = p_reszleg) and -- A kurzorterületre (az aktiv
        (partner = '---')         -- halmazba) kerülés feltétele
    FOR UPDATE OF fiz NOWAIT;
  dolgozo_rekord    dolgozo_kurzor%ROWTYPE;
  CURSOR partner_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
    SELECT vnev, fiz, ROWID sorazonosito
      FROM dolgozo
     WHERE (reszl = p_reszleg) and -- A kurzorterületre (az aktiv
        (partner = '---')         -- halmazba) kerülés feltétele
    FOR UPDATE OF fiz NOWAIT;
  partner_rekord    dolgozo_kurzor%ROWTYPE;

BEGIN
  v_reszlegnev := UPPER('&g_reszleg');
  OPEN dolgozo_kurzor(v_reszlegnev);
  LOOP
    FETCH dolgozo_kurzor
      INTO dolgozo_rekord;
    EXIT WHEN dolgozo_kurzor %NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||' ciklusa: ');
    SELECT partner
      INTO v_dolg_partner      -- Mivel a kurzorterület nem frissül,
```

```

FROM dolgozo          -- ezért az eredeti adattáblát vizsgáljuk.
-- WHERE CURRENT OF dolgozo_kurzor;          -- A SELECT utasításban
WHERE ROWID = dolgozo_rekord.sorazonosito; -- NEM lehet CURRENT OF!
IF v_dolg_partner = '---' -- Az eredeti adattáblában sem szerepel még
THEN                      -- partner, vagy csak nem frissült az adat?
OPEN partner_kurzor(v_reszlegnev);
LOOP
  FETCH partner_kurzor
  INTO partner_rekord;
EXIT WHEN partner_kurzor%NOTFOUND;
SELECT partner
  INTO v_part_partner -- Mivel a kurzorterület nem frissül,
  FROM dolgozo        -- ezért az eredeti adattáblát vizsgáljuk.
  WHERE ROWID = partner_rekord.sorazonosito;
IF (partner_rekord.sorazonosito <> dolgozo_rekord.sorazonosito)
  AND ((dolgozo_rekord.fiz - partner_rekord.fiz)
      BETWEEN -10000 and +10000)
  AND (v_part_partner = '---')
THEN
  DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||' - '||
    partner_rekord.vnev);

  UPDATE dolgozo
    SET partner = partner_rekord.vnev
  WHERE CURRENT OF dolgozo_kurzor;
  UPDATE dolgozo
    SET partner = dolgozo_rekord.vnev
  WHERE CURRENT OF partner_kurzor;
  EXIT;
END IF;
END LOOP;
CLOSE partner_kurzor;
END IF;
DBMS_OUTPUT.PUT_LINE('----- '||dolgozo_rekord.vnev||
  ' ciklusának vége');

END LOOP;
CLOSE dolgozo_kurzor;
COMMIT;
END;
/

=====
-- A jó működéshez tartozó futási eredmények --
=====

SQL> @ C:\A\s113-3

10 sor módosítva.

Adja meg a részleg nevét: rtv
régi 21: v_reszlegnev := UPPER('&g_reszleg');
új 21: v_reszlegnev := UPPER('rtv');
Nagy ciklusa:
----- Nagy ciklusának vége
Varga ciklusa:
Varga - Balogh

```

```

----- Varga ciklusának vége
Balogh ciklusa:
----- Balogh ciklusának vége
Sas ciklusa:
----- Sas ciklusának vége

```

A PL/SQL eljárás sikeresen befejeződött.

```
SQL> SELECT * FROM dolgozo;
```

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ	PARTNER
001	Kocsis	Endre	98-Aug-02	KF	48000	---
002	Nagy	Ida	99-Aug-06	RTV	36000	---
007	Kiss	Ferenc	98-Sze-07	HT	56000	---
112	Varga	Ede	98-Aug-12	RTV	72000	Balogh
324	Balogh	Zsolt	99-Feb-03	RTV	65000	Varga
722	Szeles	Emese	98-Máj-13	HT	68000	---
854	Kiss	Paula	98-Aug-12	HT	42000	---
652	Sas	Elek	97-Már-06	RTV	70000	---
712	Vas	Erika	98-Máj-07	FF	66000	---
003	Havas	Tibor	97-Máj-03	FF	80000	---

10 sor kijelölve.

## Kivételkezelés szegmens

A PL/SQL-ben a kivétel egy azonosítóval ellátott esemény, amelynek hatására a blokk végrehajtása félbeszakad.

A PL/SQL blokk harmadik része a kivételkezelés, s ezt az `EXCEPTION` kulcsszó vezeti be. Az `EXCEPTION` szakasz a PL/SQL blokk azon része, ahová a vezérlés kivétel esetén adódik. A kivételkezelő rész utasításai a helytelen működésből adódó hibákat lekezelik, és így a program futása nem szakad meg futási idejű (run-time) rendszer-hibaüzenettel, hanem a fejlesztő által megtervezett módon folytatódik.

### Kivételek fajtái

A bekövetkezett kivétel lehet felhasználó-definiált és a PL/SQL által automatikusan kiváltott rendszerkivétel.

- *Felhasználó-definiált kivétel*

A felhasználó-definiált kivételeket a `RAISE` utasítással kezelhetjük. A `RAISE` utasítás kiváltja a megnevezett kivételt. A kivétel a felhasználó által deklarált kivételek vagy rendszerkivételek valamelyike lehet.

A `RAISE` parancs önmagában csak egy blokk kivételkezelő részében állhat, amely az aktuális kivételek továbbadására alkalmas.

Általános alak:

```
RAISE [kivétel];
```

Előre nem definiált kivétel deklarálása:

```
DECLARE
    kivételnév      EXCEPTION
```

Társíthatjuk a `STANDARD` kivételkezelő csomaghoz, és rendeljünk hozzá szabványos hibakódot a következő utasítással:

```
PRAGMA EXCEPTION_INIT (kivételnév, hibakód);
```

ahol: *kivételnév*      egy korábban deklarált kivétel neve,  
           *hibakód*        egy szabványos Oracle server hiba kódszáma.

- *PL/SQL által automatikusan kiváltott rendszerkivétel*, amely lehet előre definiált, vagy előre nem definiált.

A kivételkezelő általános alakja:

```
EXCEPTION
    WHEN {kivétel1 [OR kivétel2] ... }
    THEN
        utasítás1;
        [utasítás2;]
        ...
    [WHEN {kivétel3 [OR kivétel4] ... }
```

```

      THEN
        utasítás3;
        [utasítás4; ]
        ...]
[WHEN OTHERS
  THEN
    utasítás5;
    [utasítás6; ]
    ...]

```

ahol:

<i>kivétel</i>	az előre definiált kivétel szabványos neve, illetve a deklarálás szakaszban deklarált felhasználó által definiált kivétel neve.
<i>utasítás<sub>i</sub></i>	egy PL/SQL vagy SQL utasítás.
OTHERS	nem kötelező utasításrész, amely lehetővé teszi a fel nem sorolt kivételek elfogását. Minden kezeletlen kivételt elfog.

A `WHEN OTHERS` ágból csak egy szerepelhet, és ha szerepel, akkor a `WHEN` ágak közül az utolsónak kell lennie.

A kivételkezelőben ugyanúgy használhatunk változókat, mint a blokk végrehajtható részében, azaz lokális változókra közvetlenül a nevükkel, más blokkok változóira pedig a másik blokk nevével történő minősített hivatkozást használhatjuk.

## A kivétel működése

Ha egy kivétel bekövetkezik, akkor a program normális működése abbamarad, a vezérlés az aktuális blokk kivételkezelő részére ugrik. A rendszer megvizsgálja a kivételkezelő `WHEN` utasításrészét, ellenőrzi, hogy a bekövetkezett kivétel neve szerepel-e ott. Ha megtalálja a megfelelő nevet, akkor az abban az ágba leírt utasítások hajtódnak végre.

Ha a `WHEN` ágak egyikében sem szerepel a kivétel neve, de van `WHEN OTHERS` ág, akkor az ott előírt utasítások hajtódnak végre. Az előírt tevékenység végrehajtása után a blokkot követő első végrehajtható utasításon folytatódik a program.

Ha az aktuális blokkban nincs a kivétel kezelésére előírt tevékenység, akkor félbeszakad a blokk működése, és a kivétel továbbadódik a beágyazó blokkba, vagy a környezetnek. A bekövetkezett kivétel ilyen módon, addig adódik tovább, míg a kivételt le nem kezeljük, vagy el nem jutunk a legkülső blokkba. Ha itt sem kezeljük le a kivételt, az alprogram futása (futási idejű hibával) félbeszakad.

## Definiált kivételek

Az előre definiált kivételeket a `STANDARD` csomagban találhatjuk meg.

Az előre definiált kivételek a következők:

KIVÉTEL NEVE	HIBA KÓD	LEÍRÁS
ACCES_INT0_NULL	ORA-06530	Értékhozzárendelési kísérlet nem inicializált objektum attribútumához.
COLLECTION_IS_NULL	ORA-06531	Kísérlet EXISTS metódustól eltérő gyűjtőmetódus alkalmazására nem inicializált beágyazott táblára, vagy VARRAY típusú adatra.
CURSOR_ALREADY_OPEN	ORA-06511	Kísérlet egy már megnyitott kurzor megnyitására OPEN utasítással
DUP_VAL_ON_INDEX	ORA-00001	Kísérlet INSERT vagy UPDATE utasítással egyedi indexben már meglévő érték beszúrására.
INVALID_CURSOR	ORA-01001	Nem megengedett kurzorművelet történt. Nem deklarált kurzort akarunk megnyitni, nem nyitott kurzort akarunk lezárni, nem nyitott kurzorba akarunk FETCH utasítással olvasni.
INVALID_NUMBER	ORA-01722	Karakterlánc sikertelen konvertálása számmá.
LOGIN_DENIED	ORA-01017	Bejelentkezési kísérlet az Oracle rendszerbe sikertelen, érvénytelen a felhasználói azonosító vagy jelszó.
NO_DATA_FOUND	ORA-01403	A SELECT utasítás nem ad vissza sort. (Nem azonos azzal az esettel, amikor a FETCH utasítás nem tudott sort beolvasni.)
NOT_LOGGED_ON	ORA-01012	Adatbázis műveleteket adtunk ki anélkül, hogy kapcsolódtunk volna az Oracle rendszerhez.
PROGRAM_ERROR	ORA-06501	A PL/SQL valamilyen problémába ütközött a program végrehajtása közben. (Belső probléma).
ROWTYPE_MISMATCH	ORA-06504	A hozzárendelésben szereplő környezeti kurzorváltozó és a PL/SQL kurzorváltozó visszaadott adattípusai nem kompatibilisek.
STORAGE_ERROR	ORA-06500	Nem elegendő a memória a PL/SQL számára, vagy valamilyen memóriahibát észlelt.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	A gyűjtemény elemeinek számánál nagyobb indexszámmal hivatkoztunk egy beágyazott táblára, vagy VARRAY elemre.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Az érvényes tartományon kívül eső indexszámmal (például -1) hivatkoztunk egy beágyazott táblára vagy VARRAY elemre.
TIMEOUT_ON_RESOURCE	ORA-00051	Időtúllépés történt, miközben az Oracle egy erőforrásra várakozott.

TOO_MANY_ROWS	ORA-01422	Egy SELECT utasítás, amelynek egyetlen sort kellett volna visszaadnia, több sort eredményezett.
VALUE_ERROR	ORA-06502	Ha egy oszlop vagy PL/SQL változó értéke sérült (például csonkolás miatt). Ez a fajta hiba általában típusok közötti konverzió, mező értékének másik mezőbe való másolása, vagy egy változó pontosságának nem megfelelő számítás elvégzése közben lép fel.
ZERO_DIVIDE	ORA-01476	Nullával való osztás.

## Példák

Készítsünk egy PL/SQL blokkot, amely a felhasználótól bekér egy fizetésértéket. Ha van ilyen fizetéssel rendelkező alkalmazott, akkor kiírja az aktuális jutalmat, amely fizetésének 40%-a, amennyiben nincs ilyen fizetésű dolgozó, akkor hibaüzenetet küld. Ha több megadott fizetésű dolgozó van, akkor szintén hibaüzenetet küld. Bármilyen más hiba esetét is vizsgáljuk le. (s101.sql)

```

SET SERVEROUTPUT ON
SET VERIFY OFF

ACCEPT p_fizetes PROMPT 'Kérem a dolgozó fizetését: '
DECLARE
    v_nev      emp.ename%TYPE;
    v_fiz      emp.sal%TYPE := &p_fizetes;

BEGIN
    SELECT  ename
      INTO  v_nev
    FROM    emp
    WHERE   sal = v_fiz;
    v_fiz := v_fiz*0.40;
    DBMS_OUTPUT.PUT_LINE(v_nev||' jutalma: '|| v_fiz);

EXCEPTION
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Senkinek sincs'||' '|| v_fiz||' dolláros
fizetése');
WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('TÖBB '|| v_fiz ||' dollár fizetés van');
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Egyéb hiba');
END;
/

```

### Futtatás

```

SQL> @ c:\a\s101
Kérem a dolgozó fizetését: 2345

```

Senkinek sincs 2345 dolláros fizetése.

A PL/SQL eljárás sikeresen befejeződött.

SQL> @ c:\a\s102

Kérem a dolgozó fizetését: 1600

ALLEN jutalma: 640

A PL/SQL eljárás sikeresen befejeződött.

SQL> @ c:\a\s102

Kérem a dolgozó fizetését: 3000

TÖBB 3000 dollár fizetés van.

A PL/SQL eljárás sikeresen befejeződött.

**Szebb megoldás, ha létrehozunk egy hibauzenet táblát, és az üzeneteket ebbe írjuk, majd ezt kérdezzük le egy SELECT utasítással. (s102.sql)**

```
SET SERVEROUTPUT ON
```

```
SET VERIFY OFF
```

```
-- CREATE TABLE hibauzenet
-- (hiba      VARCHAR2(50));
```

```
-- Az esetleges korábbi próbafuttatások eredményének törlése
DELETE FROM hibauzenet;
```

```
ACCEPT p_fizetes PROMPT 'Kérem a dolgozó fizetését: '
```

```
DECLARE
```

```
  v_nev      emp.ename%TYPE;
```

```
  v_fiz      emp.sal%TYPE := &p_fizetes;
```

```
BEGIN
```

```
  SELECT ename
```

```
    INTO v_nev
```

```
    FROM emp
```

```
    WHERE sal = v_fiz;
```

```
  v_fiz := v_fiz*0.40;
```

```
  DBMS_OUTPUT.PUT_LINE(v_nev||' jutalma: '|| v_fiz);
```

```
  INSERT INTO hibauzenet
```

```
    VALUES(' >> Nincs hiba!');
```

```
EXCEPTION
```

```
WHEN NO_DATA_FOUND THEN
```

```
  INSERT INTO hibauzenet
```

```
    VALUES('Nincs olyan alkalmazott akinek bére ='|| v_fiz);
```

```
WHEN TOO_MANY_ROWS THEN
```

```
  INSERT INTO hibauzenet
```

```
    VALUES('TÖBB ilyen alkalmazott van');
```

```
WHEN OTHERS THEN
```

```
  INSERT INTO hibauzenet
```

```

VALUES ('EGYÉB hiba');
END;
/

-- Az itt következő utasítás nem része a PL/SQL bloknak
SELECT * FROM hibauzenet;
```

### Futtatás

```

SQL> @ c:\a\s102
Kérem a dolgozó fizetését: 2345

A PL/SQL eljárás sikeresen befejeződött.

HIBA
-----
Nincs olyan alkalmazott akinek bére =2345

SQL> @ c:\a\s102
1 sor törölve.

Kérem a dolgozó fizetését: 1600
ALLEN jutalma: 640

A PL/SQL eljárás sikeresen befejeződött.

HIBA
-----
>> Nincs hiba!
```

### Példa

Készítsünk saját kivételt egy egyszerű SQL módosító utasításhoz, amely a felhasználó által megadott azonosító számú alkalmazott jutalékát változtatja meg. (s104.sql)

```

ACCEPT p_azon PROMPT'Kérem a dolgozó azonosító számát '
DECLARE
  v_azon emp.empno%TYPE;
  nincs_alkalmazott_hiba EXCEPTION;          -- saját kivétel definiálása
BEGIN
  v_azon := &p_azon;
  UPDATE empl
    SET comm = 777
    WHERE empno = v_azon;
  IF SQL%NOTFOUND                                -- implicit kurzor attribútum
    THEN RAISE nincs_alkalmazott_hiba;          -- saját kivétel kezelése
  END IF;
EXCEPTION
  WHEN nincs_alkalmazott_hiba THEN
    DBMS_OUTPUT.PUT_LINE('Ellenőrizzük! Mert ilyen alkalmazott nincs!');
END;
/
```

**Futtatás jó azonosító számmal:**

```
SQL> @ c:\a\s104
Kérem a dolgozó azonosító számát  7902

A PL/SQL eljárás sikeresen befejeződött.
```

```
SQL> SELECT ename, comm
2      FROM emp1
3      WHERE empno = 7902;
```

```
ENAME          COMM
-----
FORD              777
```

**Futtatás helytelen azonosító számmal:**

```
SQL> @ c:\a\s104.sql
Kérem a dolgozó azonosító számát  4567
Ellenőrizzük! Mert ilyen alkalmazott nincs!

A PL/SQL eljárás sikeresen befejeződött.
```

A kivételkezelés a fentiek szerint tehát jól működik.

Nézzük, hogyan fut le a fenti feladat megoldása hibakezelés nélkül a helytelenül megadott azonosító szám esetén.

```
SQL> UPDATE emp1
2      SET comm =777
3      WHERE empno= &empno
Adja meg a(z) empno értékét: 2345

0 sor módosítva.
```

**Példa**

Írjunk PL/SQL blokkot, amely megszámlolja, azoknak az alkalmazottaknak a számát, akiknek a fizetése a felhasználó által megadott fizetésnek +100, illetve – 100 dollár tartományba esnek.

A kivétel ekkor az az esemény, amelyik a felhasználó számára tulajdonképpen helytelen működés. Legyen helytelen az az eset, amikor nincs olyan alkalmazott, akinek a bére a megadott tartományban van, illetve ha egynél több ilyen alkalmazott van. A jó megoldás, ha éppen egy ilyen alkalmazott van. (s103.sql)

```
ACCEPT p_fizetes PROMPT 'Kérem a fizetés megadását: '

DECLARE
  v_fizetes          emp.sal%TYPE;
  db_szam             NUMBER(4);
  -- kivételek deklarálása
  nincs_alkalmazott  EXCEPTION;
  tobb_alkalmazott    EXCEPTION;
```

```

BEGIN
  v_fizetes := &p_fizetes;
  SELECT COUNT(ename)
    INTO db_szam
  FROM emp
 WHERE sal Between (v_fizetes-100) AND (v_fizetes +100);
  IF db_szam = 0
  THEN
    RAISE nincs_alkalmazott;          --felhasználói kivétel bekövetkezett
  ELSIF db_szam = 1
  THEN
    DBMS_OUTPUT.PUT_LINE('Egy darab ilyen alkalmazott van.');
```

```

  ELSIF db_szam > 1
  THEN
    DBMS_OUTPUT.PUT_LINE('darabszám :'|| db_szam);
    RAISE tobb_alkalmazott;          -- felhasználói kivétel
  END IF;

-- Kivételkezelés
EXCEPTION
  WHEN nincs_alkalmazott
  THEN
    DBMS_OUTPUT.PUT_LINE('Nincs ilyen alkalmazott. ');
  WHEN tobb_alkalmazott
  THEN
    DBMS_OUTPUT.PUT_LINE('Több ilyen alkalmazott van.');
```

```

  WHEN OTHERS
  THEN
    DBMS_OUTPUT.PUT_LINE('Egyéb hiba');
END;
/
```

#### Futtatások:

```

SQL> @ c:\a\s103
Kérem a fizetés megadását: 800
Egy darab ilyen alkalmazott van

A PL/SQL eljárás sikeresen befejeződött.

SQL> @ c:\a\s103
Kérem a fizetés megadását: 2200
Nincs ilyen alkalmazott.

A PL/SQL eljárás sikeresen befejeződött.

SQL> @ c:\a\s103
Kérem a fizetés megadását: 2980
darabszám :3
Több ilyen alkalmazott van

A PL/SQL eljárás sikeresen befejeződött.
```

A feladatot a program tehát jól megoldotta.

## Kivételek továbbadása

A kivételt nem kell elfognunk a PL/SQL blokkban, továbbadhatjuk a hívó környezet felé, és ekkor a hívó környezet kezeli a hibát. Minden környezet egyéni módon teszi ezt.

### Kivétel továbbadása beágyazott blokkban

Akkor beszélünk beágyazott blokkban történő kivétel továbbadásáról, amikor egy beágyazott blokk kezeli a kivételt, valamint a blokk a szokásos módon fejeződik be, és a program futása a beágyazó (külső) blokkban folytatódik, a belső blokk `END` utasítása után. Ha nincs a belső blokkban a kivétel elfogva, akkor a kivétel továbbkerül a külső blokkba mindaddig, amíg nem talál megfelelő kivételkezelőt. Előfordulhat, hogy a külső blokkok egyike sem képes a kivétel elfogására, ekkor kezeletlen kivétel adódik át a környezetbe. Ha a kivétel továbbadódik a külső blokkba, a külső blokk utasításai nem hajtódnak már végre.

Az ilyen kivételkezelés előnye, hogy a jellegzetesebb kivételkezelést a saját blokkban elvégezhetjük, az általánosabb hibák kezelését pedig a külső blokkban végezhetjük el.

# Alprogramok

Az alprogram olyan névvel ellátott PL/SQL blokk, amelynek paramétereit adhatunk. Két típusa van; az eljárás (`PROCEDURE`), és a függvény (`FUNCTION`). Általában az eljárás egy műveletsort végez el, a függvény egy értéket számol ki.

## Eljárások

Az eljárás egy olyan alprogram, amely végrehajtja a kijelölt műveleteket, és a paramétereit segítségével kommunikál az őt hívó környezettel.

Általános alakja:

```
PROCEDURE név [(paraméter [, paraméter],...)]
IS
    lokális deklaráció
BEGIN
    végrehajtható utasítások
[EXCEPTION
    kivételkezelés]
END [név];
```

A *paraméter* szintaxisa

```
paraméter_név [IN | OUT | IN OUT] adattípus [{:=|DEFAULT}];
```

Paraméterekre a `NOT NULL` kényszer nem adható meg. Az adattípusra kényszer, megkötés sem adható meg.

```
PROCEDURE ... (azon_szam    NUMBER(4))    IS
-- helytelen, mert megkötést tartalmaz, mégpedig a számjegyek számára (4).

PROCEDURE ... (azon_szam    NUMBER)       IS          -- ez helyes
```

Az eljárásnak két része van, a specifikációs rész (a fej), és az eljárás törzse.

Az eljárás *specifikációs része* a `PROCEDURE` kulcsszóval kezdődik, és az eljárás nevével vagy a paraméterlistával fejeződik be. A paraméter-deklaráció opcionális. Ha nincs paraméterlista, az eljárást zárójelek nélkül írjuk.

Az eljárás *törzse* az `IS` kulcsszóval kezdődik, és az `END [eljárás_név]` kulcsszóval fejeződik be. Az *eljárás\_név* az `END` kulcsszó után opcionális.

Az eljárás törzse három részből áll

- deklarációs szegmensből,
- végrehajtható szegmensből, és
- kivételkezelő szegmensből.

Deklarációs szegmens: tartalmazza a lokális deklarációkat, amelyek az `IS` és a `BEGIN` kulcsszavak között helyezkednek el. A `DECLARE` kulcsszót, amely a névtelen blokk deklarációs részét bevezeti, itt nem használjuk.

Végrehajtható szegmens: tartalmazza a `BEGIN` és az `EXCEPTION`, vagy (az `EXCEPTION` hiánya esetén) az `END` kulcsszavak között elhelyezkedő utasításokat. A

végrehajtható szegmens akár üres is lehet, amelyet a NULL üres utasítással lehet megadni.

Kivételkezelő szegmens: Az EXCEPTION és END közötti utasítások (opcionális).

### Példa

Emeljük a felhasználó által megadott azonosító számú dolgozó fizetését, a felhasználó által megadott összeggel. Alkalmazzunk felhasználói kivételt, előre definiált kivételt a programban. A megoldást az s105.sql script programban találhatjuk.

A program két eljárást tartalmaz, az egyik a nyomtat paraméterezett eljárás, amely mindig az aktuális hibaüzenetet küldi ki, amelyet a megfelelő kivételkezelő ágakon kap meg.

A másik a feldolgozó procedure, a fiz\_emeles paraméterezett eljárás. A fiz\_emeles eljárás paramétere a dolgozó azonosító száma, valamint a fizetésemelés összege. Ezeket a felhasználó adja meg. Az azonosító szám segítségével választjuk ki az empl táblából az éppen aktuális dolgozót, akinek fizetésemelést adunk. Ha helytelen azonosító számot adtunk meg, ennek megfelelő sor nincs a táblában, NO\_DATA\_FOUND kivétel történik.

Ha az azonosító szám olyan dolgozót talál, akinek fizetése éppen NULL értékű, felhasználói kivétel keletkezik.

Ha megtalálja a lekérdezés a felhasználó által megadott azonosítójú dolgozót, akinek van is fizetése, akkor a növelés értékével megemeli a fizetését, módosítja a táblát.

Az eljárás hívása az aktuális paraméterekkel történik:

```
fiz_emeles (v_azon, fiz_emel)
```

ahol v\_azon és fiz\_emel a névtelen blokk változói.

```
DECLARE
    v_azon      empl.empno%TYPE;
    fiz_emel    empl.sal%TYPE;
    szov        VARCHAR2(30);

PROCEDURE nyomtat( szoveg  VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(szoveg);
END;

PROCEDURE fiz_emeles (azonosito_szam  char, noveles  INTEGER) IS
    aktualis_fizetes INTEGER;      -- deklarációs szegmens
    fizetes_hiba  EXCEPTION;      -- felhasználói kivétel deklarálása
BEGIN                                -- végrehajtható szegmens
    SELECT sal
    INTO aktualis_fizetes
    FROM empl
    Where empno = azonosito_szam;
    IF aktualis_fizetes IS NULL      -- ha a lekérdezett érték NULL
    THEN
        RAISE fizetes_hiba ;        -- felhasználói kivétel
    ELSE
        UPDATE empl
        SET sal = sal + noveles
        WHERE empno = azonosito_szam;
    END IF;
```

```

EXCEPTION                                -- kivételkezelő szegmens
  WHEN NO_DATA_FOUND THEN                -- ha nincs ilyen sor
    szov := 'H I B A , nincs adat';
    nyomtat (szov);                      -- eljáráshívás
  WHEN fizetés_hiba THEN                  -- ha felhasználói kivétel történt
    szov := 'Fizetés nulla';
    nyomtat (szov);                      -- eljáráshívás
END fiz_emeles;

BEGIN                                    -- főprogram
  v_azon := '&p_azon';
  fiz_emel := &p_fizetes;
  fiz_emeles(v_azon, fiz_emel);          -- eljáráshívás
END;
/

```

Futtassuk le olyan azonosítóval, amilyen azonosítójú sor nem létezik az `empl` táblában. Ha jó a program, rá kell futnia a `NO_DATA_FOUND` rendszerkivételt kezelő ágra, és ki kell írnia a megfelelő üzenetet.

```

SQL> @ c:\a\s105
Kérem az dolgozó számát                5678
Kérem a dolgozó fizetésemelését        500
H I B A , nincs adat

A PL/SQL eljárás sikeresen befejeződött.

```

Futtassuk le, ha a fizetés oszlopérték véletlenül `NULL` értékű. Legyen ez a `FORD` nevű alkalmazottnál. A program jó működése esetén ráfut a felhasználó által definiált kivételre, és ennek megfelelő hibaüzenetet írja ki.

Futtatás előtt:

```

...
7900 JAMES                950                30
7902 FORD                  20
7934 MILLER              1300                10

```

Futtatás után:

```

SQL> @ c:\a\s105
Kérem az dolgozó számát                7902
Kérem a dolgozó fizetésemelését        400
Fizetés nulla

A PL/SQL eljárás sikeresen befejeződött.

```

Futtassuk le akkor is, ha minden bemenő adat megfelelő. Emeljük a 7900 azonosítójú `JAMES` alkalmazott fizetését 400 dollárral.

```

SQL> @ c:\a\s105
Kérem az dolgozó számát                7900
Kérem a dolgozó fizetésemelését        400

A PL/SQL eljárás sikeresen befejeződött.

```

Ellenőrizzük `SELECT` utasítással:

```
SQL> SELECT empno, ename, sal
      2      FROM emp1;

EMPNO ENAME          SAL
-----
...
      900 JAMES          1350
...
```

## Függvények

A függvények olyan alprogramok, amelyeknek egy visszatérési értékük van. A függvényeknek és eljárásoknak a felépítése azonos, kivéve, hogy a függvényben szerepel a `RETURN` záradék.

A függvények általános alakja:

```
FUNCTION név [(paraméter [,paraméter]...)] RETURN adattípus
IS
    lokális deklaráció
BEGIN
    végrehajtható utasítások
[EXCEPTION
    kivételkezelés]
END [név];
```

ahol a *paraméter* szintaxisa következő:

```
paraméter_név [IN | OUT | IN OUT] adattípus [{ := | DEFAULT}]
```

A `NOT NULL` kényszer a függvény paramétereire és a visszatérési értékére nem megengedett.

Az eljáráshoz hasonlóan a függvény is két részből áll; a specifikációs részből, és a függvény törzséből.

Specifikációs rész: A `FUNCTION` kulcsszóval kezdődik, és a `RETURN` záradékkal fejeződik be, amely a visszaadott adattípust definiálja. A paraméter deklarációja opcionális.

Függvény törzs: Az `IS` kulcsszóval kezdődik, és az `EXCEPTION` illetve `END` kulcsszóval fejeződik be, amelyet az opcionális függvéynév követ.

A függvény törzse is három részből áll:

- deklarációs szegmensből,
- végrehajtható szegmensből, és
- kivételkezelő szegmensből.

A deklarációs szegmens itt is tartalmazza a lokális deklarációkat, de nem használjuk a `DECLARE` kulcsszót.

A végrehajtható szegmens a végrehajtható utasításokat tartalmazza. Egy vagy több `RETURN` utasításrész kell, hogy legyen a végrehajtható részben.

A kivételkezelő szegmens tartalmazza a kivételek bekövetkeztekor szükséges utasításokat az `EXCEPTION` és az `END` kulcsszavak között.

### Példa

Új dolgozókat szeretne a cég felvenni. Hivatalnokok jelentkezését várják. Akkor tudják felvenni őket, ha a jelentkező által kért fizetés a hivatalnokok maximális és minimális bére között van. Ha a kért fizetés a két értéken kívül van, akkor bérfeszültség léphet fel. Ezt írja ki a program. Ha a jelentkező feltételei megfelelnek, vegyük fel az `empl` táblába. (`s106.sql`)

A névtelen blokkban a már ismert `nyomtat` paraméterezett eljárás, és a megfelelő fizetést kiszámító `fiz_ok` függvény található meg.

A `fiz_ok` függvénynek két paramétere van: a fizetés (`fizet`) és a részleg (`reszleg`). Ezek a felhasználótól kapnak értéket. Ha a `fizet` értéke a részleg `min_fiz` és `max_fiz` értékei közé esik (tehát a visszatérő záradék `RETURN (fizet >= min_fiz) AND (fizet <= max_fiz);` teljesül), akkor a függvény `TRUE` logikai értékkel tér vissza. Ha a `fiz_ok` értéke `TRUE`, akkor a `THEN` ágon végrehajtja a sor beszúrását, vagyis az új dolgozó felvételét a céghez. Ha `FALSE`, akkor hívja a `nyomtat` eljárást, amelynek csak egy bemenő (`IN`) paramétere van, ez pedig a névtelen blokkban deklarált `szov` változó, amelynek kezdőértéket is adtunk. (`s106.sql`)

```
--Környezeti változók
ACCEPT p_azon PROMPT 'Kérem az új dolgozó számát          '
ACCEPT p_nev  PROMPT 'Kérem az új dolgozó nevét           '
ACCEPT p_belep_dat PROMPT 'Kérem az új dolgozó belépési idejét '
ACCEPT p_rszam PROMPT 'Kérem az új dolgozó részlegének számát '
ACCEPT p_reszl PROMPT 'Kérem az új dolgozó részlegét       '
ACCEPT p_fiz PROMPT 'Kérem az új dolgozó fizetését        '

SET SERVEROUTPUT ON
SET VERIFY OFF

-- Névtelen blokk
DECLARE
    v_azon          emp.empno%TYPE;
    v_nev           emp.ename%TYPE;
    v_belep_dat     emp.hiredate%TYPE;
    v_reszleg_szam  emp.deptno%TYPE;
    uj_reszl        emp.job%TYPE;
    uj_fiz          emp.sal%TYPE;
    szov            VARCHAR2(20) := 'BÉRPROBLÉMA' ;

PROCEDURE nyomtat(szoveg IN VARCHAR2) IS -- eljárás
BEGIN
    DBMS_OUTPUT.PUT_LINE(szoveg);
END;

FUNCTION fiz_ok(fizet INTEGER, reszleg CHAR) RETURN BOOLEAN IS -- függvény
    min_fiz  INTEGER;
    max_fiz  INTEGER;
BEGIN
    SELECT MIN(sal), MAX(sal)
    INTO min_fiz, max_fiz
    FROM empl
```

```

        WHERE job = reszleg;
        RETURN (fizet >= min_fiz) AND (fizet <= max_fiz); -- RETURN utasításrész
    END fiz_ok;

BEGIN
    v_azon := '&p_azon';
    v_nev := '&p_nev';
    v_belep_dat := '&p_belep_dat';
    uj_reszl := '&p_reszl';
    uj_fiz := &p_fiz;
    v_reszleg_szam := &p_rszam;

    IF fiz_ok(uj_fiz,uj_reszl)
    THEN
        INSERT INTO empl (empno,ename,job,hiredate,sal,deptno)
        VALUES ( v_azon,v_nev,uj_reszl,v_belep_dat,uj_fiz, v_reszleg_szam);
    ELSE
        nyomtat (szov);
    END IF;

    COMMIT;
END;
/

```

Futtassuk le Kalapos úrral, aki jelentkezett felvételre, és 2222 dollárt kér.

```

SQL> @ c:\a\s106
Kérem az új dolgozó számát          8774
Kérem az új dolgozó nevét          KALAPOS
Kérem az új dolgozó belépési idejét 01-AUG-15
Kérem az új dolgozó részlegének számát 30
Kérem az új dolgozó részlegét      CLERK
Kérem az új dolgozó fizetését      2222
BÉRPROBLÉMA

```

A PL/SQL eljárás sikeresen befejeződött.

Mint láthatjuk, a jelentkező többet kér, mint bármelyik, a vállalatnál már dolgozó hivatalnok, így bérfeszültség keletkezhet.

Futtassuk le más jelentkezővel is.

```

SQL> @ c:\a\s106
Kérem az új dolgozó számát          9876
Kérem az új dolgozó nevét          FOGARASI
Kérem az új dolgozó belépési idejét 01-AUG-08
Kérem az új dolgozó részlegének számát 30
Kérem az új dolgozó részlegét      CLERK
Kérem az új dolgozó fizetését      1250

```

A PL/SQL eljárás sikeresen befejeződött.

Tehát Fogarasi úr nem kért túl sokat, így őt felvették.

Ellenőrizzük SELECT utasítással:

```
SQL> SELECT * FROM empl;
```

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
7902	FORD	ANALYST	81-DEC-03		20
7934	MILLER	CLERK	82-JAN-23	1300	10
9876	FOGARASI	CLERK	01-AUG-08	1250	30

## RETURN utasítás

A `RETURN` utasítás hatására az alprogram befejezi működését, és visszatér a hívó környezetbe.

A `RETURN` utasítást ne cseréljük össze a `RETURN` záradékkal, amely a függvény eredményének adattípusát definiálja.

Az alprogram több `RETURN` utasítást tartalmazhat, végrehajtva ezek közül egyet, az alprogram azonnal befejeződik.

Az eljárásban (`PROCEDURE`) is lehet `RETURN` utasítás, de az nem tartalmazhat kifejezést. Az utasítás egyszerűen visszatér.

A függvényben (`FUNCTION`) a `RETURN` utasításnak kifejezést kell tartalmaznia, amely a `RETURN` utasítás végrehajtásakor értékelődik ki. A kiértékelt eredményt a függvény azonosítója kapja meg, amelyet a `RETURN` záradék adattípusa definiál.

## Alprogramok deklarációja

Alprogramot minden PL/SQL blokkban vagy csomagban deklarálhatunk. Az alprogramok deklarációja a deklarációs rész végén legyen, minden más deklaráció után.

Mivel a PL/SQL nyelvben minden azonosítót, változót, eljárást használata előtt deklarálnunk kell, ezért az alprogramokat is deklarálni kell meghívásuk előtt. Ha ezt elmulasztjuk, akkor hibaüzenetet kapunk.

### Példa

```
PROCEDURE fiz_emeles(..) IS
BEGIN
  ...
  if atlag THEN                                -- hiba , deklarálatlan változó
  ...
END;

FUNCTION atlag RETURN NUMBER IS
BEGIN
  ...
END;
```

Ekkor hibaüzenetet kapunk, mivel az *atlag* nevű függvényt előbb hívtuk meg, mint ahogy deklaráltuk volna a programban. A probléma kiküszöbölhető, ha az *atlag* függvényt a *fiz\_emeles* eljárás előtt deklaráljuk. Vannak esetek, amikor ez mégsem elég. Képzeljük el, azt az esetet, amikor két alprogram kölcsönösen hivatkozik egymásra (ezt

körbehivatkozásnak is nevezik). Ekkor nyilván semmilyen sorrendezésük nem oldja meg a deklarálatlan hivatkozás problémáját.

## Alprogramok elődeklarációja

Olyan esetben, amikor nem tudjuk előre, hogy az alprogramjaink milyen sorrendben fognak egymásra hivatkozni, vagy körbehivatkozás áll fenn, a deklarálatlan hivatkozás problémája az úgynevezett elődeklarációval végezhető el. Ez a fentiekén túl még használatos

- ABC rendbe írt alprogramok esetén,
- rekurzív alprogramok esetén,
- csoportos alprogramoknál a csomagokban.

Az elődeklaráció esetén az elődeklarált alprogram fejrészét kell csupán a programszövegbe beírni a reá történő hivatkozás előtt.

### Példa

```
FUNCTION atlag RETURN NUMBER ;           --előrehivatkozás
PROCEDURE fiz_emeles(..) IS
BEGIN
...
  IF atlag THEN ...
...
END;
...
FUNCTION atlag RETURN NUMBER IS
BEGIN
...
END;
```

### Példa

Emeljük meg a felhasználó által megadott dolgozó fizetését, ha a fizetése annak a részlegnek az átlagfizetése alatt van, amelybe önmaga tartozik. Ha fizetése NULL, akkor felhasználói kivétel keletkezik. Ha nincs olyan dolgozó, akkor NO\_DATA\_FOUND kivétel keletkezik. Ha a dolgozó fizetése nagyobb a részlegének átlagfizetésénél, akkor a következő üzenet jelenjen meg: Sajnos fizetését nem emeljük.

A program tartalmazzon egy átlagot kiszámító függvényt, egy nyomtat eljárást, egy fizetésemelést elvégző eljárást. (s107.sql)

```
-- Környezeti változók bekérése
ACCEPT p_azon PROMPT 'Kérem a dolgozó azonosító számát '
ACCEPT p_fiz PROMPT 'Kérem a dolgozó fizetésemelését '
ACCEPT p_reszl PROMPT 'Kérem a dolgozó részlegét '

SET SERVEROUTPUT ON
SET VERIFY OFF

--Névtelen blokk
DECLARE
  v_azon          emp.empno%TYPE;
  v_reszleg       emp.job%TYPE;
  fiz_emel        emp.sal%TYPE;
  szov            VARCHAR2(30);
```

```

PROCEDURE nyomtat(szoveg  VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(szoveg);
END;

FUNCTION atlag RETURN NUMBER IS
    atl  NUMBER;
BEGIN
    SELECT ROUND(AVG(sal))
        INTO atl
        FROM emp
        GROUP BY job
        HAVING job =v_reszleg;
    RETURN atl ;
END;

PROCEDURE fiz_emeles(azon_szam  CHAR,
                    foglalkozas  CHAR,
                    noveles      INTEGER) IS

    aktualis_fiz  INTEGER;
    fiz_hiba      EXCEPTION;

BEGIN
    SELECT sal
        INTO aktualis_fiz
        FROM emp1
        WHERE empno||job = azon_szam||foglalkozas;
    IF aktualis_fiz IS NULL
    THEN
        RAISE fiz_hiba;
    ELSIF aktualis_fiz < atlag THEN
        szov:= 'Az átlag '|| TO_CHAR(atlag);
        nyomtat(szov);
        UPDATE emp1
            SET sal= sal+ noveles
            WHERE empno = azon_szam;
    ELSE
        szov := 'A fizetését sajnos nem emeljük';
        nyomtat(szov);
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        szov := 'H I B A , nincs adat';
        nyomtat( szov);
    WHEN fiz_hiba THEN
        szov := 'Fizetés nulla';
        nyomtat( szov);
END fiz_emeles;

BEGIN
    v_azon  := '&p_azon';
    fiz_emel :=  &p_fiz;
    v_reszleg := UPPER('&p_reszl');
-- főprogram

```

```

-- nyomkövetéshez, atlag függvényhívás
-- DBMS_OUTPUT.PUT_LINE('átlag '|| atlag);

    fiz_emeles(v_azon,v_reszleg, fiz_emel);      -- eljáráshívás
END;
/

```

### Futtatási eredmények.

Az emp1 tábla futtatás előtt:

```
SQL> SELECT ename, job, sal
       2      FROM emp1;
```

ENAME	JOB	SAL
SMITH	CLERK	800
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
JONES	MANAGER	2975
MARTIN	SALESMAN	1250
BLAKE	MANAGER	2450
CLARK	MANAGER	2783
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
TURNER	SALESMAN	1500
ADAMS	CLERK	1100
JAMES	CLERK	1350
FORD	ANALYST	
MILLER	CLERK	1300

14 sor kijelölve.

#### 1.) Első futtatás:

FORD fizetése, mint fent látjuk NULL értékű, próbáljuk ki a programot.

```
SQL> @ c:\a\s107
Kérem a dolgozó azonosító számát 7902
Kérem a dolgozó fizetésemelését 470
Kérem a dolgozó részlegét analyst
Fizetés nulla

```

A PL/SQL eljárás sikeresen befejeződött.

#### 2.) Második futtatás:

Nem talált ilyen azonosítójú dolgozót.

```
SQL> @ c:\a\s107
Kérem a dolgozó azonosító számát 7654
Kérem a dolgozó fizetésemelését 345
Kérem a dolgozó részlegét clerk
átlag 1038
H I B A , nincs adat

```

A PL/SQL eljárás sikeresen befejeződött.

#### 3.) Harmadik futtatás:

Emeljük meg a fizetését a 7844 azonosítójú dolgozó (SALESMAN) fizetését.

```
SQL> @ c:\a\s107
Kérem a dolgozó azonosító számát 7844
Kérem a dolgozó fizetésemelését 444
Kérem a dolgozó részlegét SALESMAN
A fizetését sajnos nem emeljük
```

A PL/SQL eljárás sikeresen befejeződött.

A program ezen az ágon is jól működött. A kereskedők átlagfizetése (a nyomkövető utasítás aktivizálása esetén láthatóan) 1400. Mivel pedig TURNER fizetése 1500, ezért az ő fizetését nem emelhetjük.

#### 4.) Negyedik futtatás

Ekkor minden bemenő adat megfelel (CLARK, aki MANAGER, azonosítója 7782):

```
SQL> @ c:\a\s108
Kérem a dolgozó azonosító számát 7782
Kérem a dolgozó fizetésemelését 333
Kérem a dolgozó részlegét manager
Az átlag 2758
```

A PL/SQL eljárás sikeresen befejeződött.

Ellenőrizzük a táblát:

```
SQL> SELECT empno, ename, job, sal
2 FROM empl;
```

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	800
7499	ALLEN	SALESMAN	1600
7521	WARD	SALESMAN	1250
7566	JONES	MANAGER	2975
7654	MARTIN	SALESMAN	1250
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2783
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7844	TURNER	SALESMAN	1500
7876	ADAMS	CLERK	1100
7900	JAMES	CLERK	1350
7902	FORD	ANALYST	
7934	MILLER	CLERK	1300

Látható, hogy 7782 azonosítójú CLARK fizetését valóban megemeltük.

## Tárolt alprogramok

Az Oracle képes eltárolni az adatbázis-területen a felhasználó által deklarált alprogramokat. Ez az újra-felhasználhatóság hatékonyabbá teszi a programozást, és mivel az adatbázis-terület a szerveren van, ezért a programfutás is gyorsabb.

Alprogram létrehozása és tárolása az Oracle adatbázisban a `CREATE PROCEDURE` és `CREATE FUNCTION` utasításokkal történik. Mindkét utasítás interaktívan végrehajtható az SQL\*Plus környezetben.

Például a `dolgozo_kilep` eljárást a következőképp kell létrehozni:

```
CREATE PROCEDURE dolgozo_kilep (ki_azon NUMBER) AS
BEGIN
    DELETE FROM emp
    WHERE empno = ki_azon;
END;
```

Ha tárolt alprogramot készítünk, akkor az `IS` kulcsszó helyett az `AS` kulcsszót használjuk.

## Aktuális és formális paraméterek

Az alprogramokkal az információcsere a paramétereken keresztül történik. Az alprogram deklarációban szereplő paramétereket *formális paramétereknek*, míg az alprogram hívásakor a paraméterlistában szereplő változókat, kifejezéseket *aktuális paramétereknek* nevezzük.

```
fiz_emeles (azon_szam, emeles);
fiz_emeles (6789, alapfizetés+1000);
```

A jó programozó különböző neveket használ a formális és aktuális paraméterlistában. Adott alprogram esetén a formális és aktuális paraméterlista elemeinek *típusa, sorrendje* rendre meg kell egyezzen.

## Az IN típusú paraméterek

Az `IN` típusú (érték) paraméter csupán átadja értékét a hívó alprogramnak. Mivel az alprogramban tehát *bemenő paraméterként* – egyfajta konstansként – viselkedik, ezért értékét megváltoztatni nem lehet.

Például fordítási hibát eredményez a következő programrészlet:

```
PROCEDURE tartoz_szamla (szamlaszam IN INTEGER, menny IN REAL) IS
    minimum_emeles      CONSTANT REAL;
    szolgáltatás        CONSTANT REAL;
BEGIN
    ...
    IF menny < minimum_emeles
    THEN
        menny := menny + szolgáltatás;           -- szintaktikai hiba
```

```

    END IF;
    ...
END;
```

A programrészlet hibás, mivel a `menny` változó az eljárásban új értéket kapott, pedig mint bemenő paraméter az eljárásban nem vehet fel új értéket.

Az `IN` típusú paraméterek helyére az alprogram hívásakor konstanst, értékkel rendelkező változót, vagy kifejezést egyaránt lehet írni.

## Alapértelmezett paraméterhasználat

Az `IN` típusú paramétereket használó alprogramok meghívhatók hiányos paraméterlistával, ha az `IN` típusú paramétereket alapértelmezett értékkel láttuk el. A hiányos paraméterlista használatának alapelve az, hogy mindig csak a paraméterlista végéről lehet elhagyni egy vagy több paramétert az egyértelmű azonosíthatóság miatt. Az elhagyott paraméterek helyett ekkor az alapértelmezett (default) paraméter-értékek helyettesítődnek be az aktuális paraméterlistában.

### Példa

```

PROCEDURE uj_reszleg_letrehozas
    (uj_reszleg_nev      CHAR      DEFAULT 'MARKETING ',
     uj_hely              CHAR      DEFAULT 'BUDAPEST') IS
BEGIN
    INSERT INTO DEPT
        VALUES (50, uj_reszleg_nev, uj_hely);
    ...
END;
```

A fenti eljárás meghívható a következő alakok bármelyikével

```

uj_reszleg_letrehozas;
uj_reszleg_letrehozas('LOGISZTIKA');
uj_reszleg_letrehozas('LOGISZTIKA', 'SZEGED');
```

de nem hívható meg az alábbi alakok egyikével sem

```

uj_reszleg_letrehozas(, 'SZEGED');
uj_reszleg_letrehozas('SZEGED');
```

## Az OUT típusú paraméterek

Az alprogram `OUT` (eredmény) paramétere visszatérési értéket ad a hívó programnak. Az alprogramban az `OUT` paraméter úgy viselkedik, mint egy kimenő változó. Ezért értékét nem adhatja át az alprogramon belül egy másik változónak, vagy önmagának.

Például fordítási hibát eredményez a következő:

```

PROCEDURE kalk_jutalek (azon_szam IN INTEGER, jutalek OUT REAL) IS
    belep_dat          DATE;
BEGIN
```

```

SELECT sal*0.1, hiredate
  INTO jutalek, belep_dat
  FROM emp
  WHERE empno = azon_szam;
IF MONTHS_BETWEEN (sysdate, belep_dat) > 60
  THEN
    jutalek := jutalek + 5000;          -- szintaktikai hiba
  END IF;
...
END;
```

Hibás, mert a `jutalek` egy értékadó utasítás mindkét oldalán szerepel. Pedig mivel `OUT` típusú változó, így csak értéket kaphat, eredmény lehet, azaz csak bal oldalán szerepelhet egy értékadó utasításnak.

Az `OUT` típusú paraméterek helyére az alprogram hívásakor csak változót szabad írni, tehát például az alábbi eljáráshívás hibás:

```
kalk_jutalek(111, fiz+előleg);          -- szintaktikai hiba.
```

Az `OUT` aktuális paraméternek a hívás előtt ugyan lehet értéke, de ezt az értéket elveszti az alprogram hívásakor. Mint minden PL/SQL változót, az `OUT` típusút is lehet `NULL` értékkel inicializálni.

## Az IN OUT típusú paraméterek

Az `IN OUT` típusú (érték-eredmény) paraméter híváskor átadja inicializált értékét az alprogramnak, és a megváltoztatott értékkel tér vissza. Az alprogramban az `IN OUT` paraméter úgy működik, mint egy inicializált változó. Ezért érték rendelhető hozzá, és értékét átadhatja egy másik változónak. Ez azt jelenti, hogy az `IN OUT` paramétert úgy használhatjuk, mint egy egyszerű változót. Értéke megváltoztatható többféle módon.

Például:

```

PROCEDURE kalk_jutalek(azon_szam IN INTEGER, jutalek IN OUT REAL) IS
  belep_dat          DATE;
  jutalek_hiba       EXCEPTION;
BEGIN
  SELECT sal*0.1, hiredate
    INTO jutalek, belep_dat
    FROM emp
    WHERE empno = azon_szam;
  IF jutalek IS NULL
  THEN
    RAISE jutalek_hiba;
  END IF;
  ...
  IF MONTHS_BETWEEN (sysdate, belep_dat) > 60
  THEN
    jutalek := jutalek + 500;
  END IF;
  ...
```

```

EXCEPTION
  WHEN jutalek_hiba THEN
  ...
END kalk_jutalek;

```

A `jutalek` most IN OUT paraméter, tehát az alprogramba lépésekor megtartja értékét, és az alprogramon belül ezt az értéket meg is lehet változtatni.

A 10.1. táblázat összefoglalva mutatja a háromféle paraméter jellemzőit.

10.1. táblázat Paraméter módok

IN	OUT	IN OUT
az IN kulcsszót nem kötelező kiírni, mivel ez az alapértelmezés	az OUT kulcsszót kötelező kiírni.	az IN OUT kulcsszó párt kötelező kiírni.
átadja az értékét az alprogramnak	értékkel tér vissza a hívóhoz.	átadja a korábbi értékét az alprogramnak, és az új értékkel tér vissza a hívóhoz.
úgy viselkedik, mint egy konstans	úgy viselkedik, mint egy inicializálatlan változó	úgy viselkedik, mint egy inicializált változó
az alprogramban nem kaphat értéket	az alprogramban értéket kell kapnia, de ott nem használható fel az értéke.	az alprogramban felhasználható a korábbi értéke, és új értéket is kaphat.
aktuális paraméterként konstans, inicializált változó, vagy kifejezés lehet.	aktuális paraméterként csak változó lehet	aktuális paraméterként csak változó lehet
aktuális paraméterként a hivatkozott formális paraméter értéke.	aktuális paraméterként az alprogramban hozzárendelt értéket veszi fel.	aktuális paraméterként mindkét irányban értéket továbbít.

## Rekurzív alprogram

Készítsük el a faktoriális ( $n!$ ) kiszámító programot.

$1! =$

$2! = 1 * 2 = 2 * 1!$

$3! = 1 * 2 * 3 = 6 = 3 * 2!$

Az algoritmus formulája tehát:  $n * (n-1)!$  (`s108.sql`)

```

ACCEPT felh_szam PROMPT ' Hány faktoriális számoljak? '

DECLARE
  szam      NUMBER(10);

```

```

FUNCTION faktor (n    POSITIVE) RETURN INTEGER IS
BEGIN
    IF n = 1
    THEN
        RETURN 1;
    ELSE
        RETURN n * faktor(n - 1);          -- rekurzív hívás
    END IF;
END faktor;

BEGIN
    szam := &felh_szam;
    DBMS_OUTPUT.PUT_LINE( faktor(szam));
END;
/

```

### Futtatás

```

SQL> @ c:\a\s108
Hány faktoriálist számoljak?  5
120

A PL/SQL eljárás sikeresen befejeződött.

SQL> @ c:\a\s108
Hány faktoriálist számoljak?  20
2432902008176640000

A PL/SQL eljárás sikeresen befejeződött.

```

Sok probléma megoldható rekurzióval és iterációval (azaz ciklushasználattal) is. Az iteratív alprogramot könnyebb általában elkészíteni, mint a rekurzív programot. Ezzel szemben a rekurzív program rendszerint egyszerűbb, rövidebb és ezért könnyebb nyomkövetni. Hasonlítsunk össze egy iteratív és rekurzív alprogramot.

### Példa

Írjuk ki a felhasználó által megadott számig a Fibonacci számokat. (s109.sql) A Fibonacci számok a következők: 0, 1, 1, 2, 3, 5, 8, 13, 21,...

```

ACCEPT felh_szam PROMPT 'Hány tagot számoljak?  '
ACCEPT felh_válasz PROMPT 'Iterációval:i, vagy rekurziv megoldással: r '

DECLARE
    szam          NUMBER(10);
    válasz        VARCHAR2(1);
    szam_hiba      EXCEPTION;

FUNCTION fibo(n POSITIVE) RETURN INTEGER IS          -- rekurzív megoldás
BEGIN
    IF n = 1
    THEN
        RETURN 1;
    ELSIF n = 2 THEN

```

```

        RETURN 1;
    ELSE
        RETURN fibo(n-1) + fibo(n-2);           -- rekurzív hívás
    END IF;
END fibo;

FUNCTION fibo_iter(n POSITIVE) RETURN INTEGER IS --iteratív megoldás
    elozo_1    INTEGER := 0;
    elozo_2    INTEGER := 1;
    uj_tag     INTEGER;

BEGIN
    IF n = 1
    THEN
        RETURN 1;
    ELSIF n = 2 THEN
        RETURN 1;
    ELSE
        DBMS_OUTPUT.PUT_LINE('A tagok: '||elozo_1); -- 0 érték kiírása
        FOR ind IN 2..n LOOP
            uj_tag := elozo_1 + elozo_2;
            elozo_1 := elozo_2;
            elozo_2 := uj_tag;
            DBMS_OUTPUT.PUT_LINE('A tagok: '||
                                uj_tag);           -- minden tag kiírható
        END LOOP;
        RETURN uj_tag;                             -- visszatérési érték
    END IF;
END fibo_iter;

BEGIN                                           -- főprogram
    szam := &felh_szam;
    válasz := UPPER('&felh_válasz');

    IF szam < 0
    THEN
        RAISE szam_hiba;                       -- kivétel, ha nem pozitív
    END IF;
    IF válasz = 'I'
    THEN
        DBMS_OUTPUT.PUT_LINE('Végeredmény '|| fibo_iter(szam));
    ELSIF válasz = 'R' THEN
        DBMS_OUTPUT.PUT_LINE('A rekurzív eredmény: '|| fibo(szam));
    ELSE
        DBMS_OUTPUT.PUT_LINE('Rossz választ adtál');
    END IF;

EXCEPTION
    WHEN szam_hiba THEN
        DBMS_OUTPUT.PUT_LINE('Rossz számot adtál');
END;
/

SQL> @ c:\a\s109
Hány tagot számoljak? 10

```

```
    Iterációval:i, vagy rekurzív megoldással: r  i
A tagok:  0
A tagok:  1
A tagok:  2
A tagok:  3
A tagok:  5
A tagok:  8
A tagok: 13
A tagok: 21
A tagok: 34
A tagok: 55
Végeredmény  55
```

A PL/SQL eljárás sikeresen befejeződött.

```
SQL> @ c:\a\s109
Hány tagot számoljak? 10
    Iterációval:i  vagy rekurzív megoldással: r  r
A rekurzív eredmény:  55
```

A PL/SQL eljárás sikeresen befejeződött.

### Futtatás hibás szám esetén, a kivételkezelés tesztelése

```
SQL> @ c:\a\s109
Hány tagot számoljak? g
    Iterációval:i, vagy rekurzív megoldással: r  i
Rossz számot adtál
```

A PL/SQL eljárás sikeresen befejeződött.

# IV. RÉSZ

## AZ ORACLE ADATBÁZISOK HASZNÁLATA DELPHIBEN

Modern korunkban, a XXI. század elején az adatbáziskezelés megjelent az élet minden területén. Egy adatbázis-kezelő rendszer használhatóságához azonban az is szükséges, hogy program felülete szép és könnyen kezelhető legyen. A fejlesztői környezettől tehát elvárjuk a hatékony adatbázis-kezelési képességeket és a felhasználóval való kifinomult kapcsolattartás képességét is.

Kétségtelen, hogy a nagy adatbázis-rendszereket készítő cégek – így az ORACLE is – rendelkeznek, valamilyen felhasználói felület készítésére alkalmas eszközzel (az ORACLE esetén ez az Oracle Developer). Ezek az eszközök általában igen kellemes és gyors fejlesztést tesznek lehetővé. Annak oka, hogy mi most mégsem ezt ismertetjük részben az, hogy jelen sorozat egy későbbi kötete fogja ezt az eszközt részletesen bemutatni. Másik oka azonban az, hogy a Borland cég Delphi rendszere a hazai számítástechnikával foglalkozó szakemberek és diákok között egyaránt jól ismert. A Borland cég igen nagy erőfeszítéseket tett annak érdekében, hogy a Delphi bármely alkalmazás, és különösen adatbázis-alkalmazás számára egy megfelelő, kényelmes és sokoldalú fejlesztő eszközként jelenjen meg.

Bár az alábbiak alapján a kezdők is létre tudják hozni saját Delphi felületüket, a megértést lényegesen gyorsítja korábbi Delphi ismeret és gyakorlat.

## 11. FEJEZET

# Delphi, mint fejlesztési és beágyazó-környezet

Ebben a fejezetben bemutatjuk a Delphi fejlesztő rendszer alapvető eszközkészletét, felépítésének logikáját, valamint azokat a szabványokat, amelyek révén más – elsősorban adatbázis-kezelő – alkalmazásokhoz kapcsolódni képes. Mivel a Delphi összefoglalja az objektumközpontú és vizuális programozás előnyeit, így rendkívül sokoldalú fejlesztési környezetet biztosít.

## Alkalmazásfejlesztés Delphi-ben

A számítógép az adatbázisokat mindig fájlyszerűen tárolja. A tárolás módjára többféle lehetőség adódik:

- az egész adatbázis egyetlen állományban tárolható,
- felbonthatók táblákra, indexekre vagy más összetevőkre, és ezek a részek külön állományokban, de általában egy könyvtárban találhatók.

A Delphi mindkét módszert támogatja az adatbázis formátumától függően.

- A Paradox és dBase táblák az adatbázisokat egy-egy könyvtárban, táblánként külön állományban tárolják.
- Az Access, az Interbase, és a legtöbb SQL adatkiszolgáló az egész adatbázist egyetlen állományban tárolja.

A Delphi adatbázis-alkalmazások nem közvetlenül kommunikálnak az adatforrásokkal, az adatbázisfájlokat nem közvetlenül kezelik. Az adatbázisok kezeléséhez szükséges egy adatbázismotor. Ezt a szerepet tölti be a Borland Database Engine (BDE), vagy a Microsoft ActiveX Data Objects (ADO).

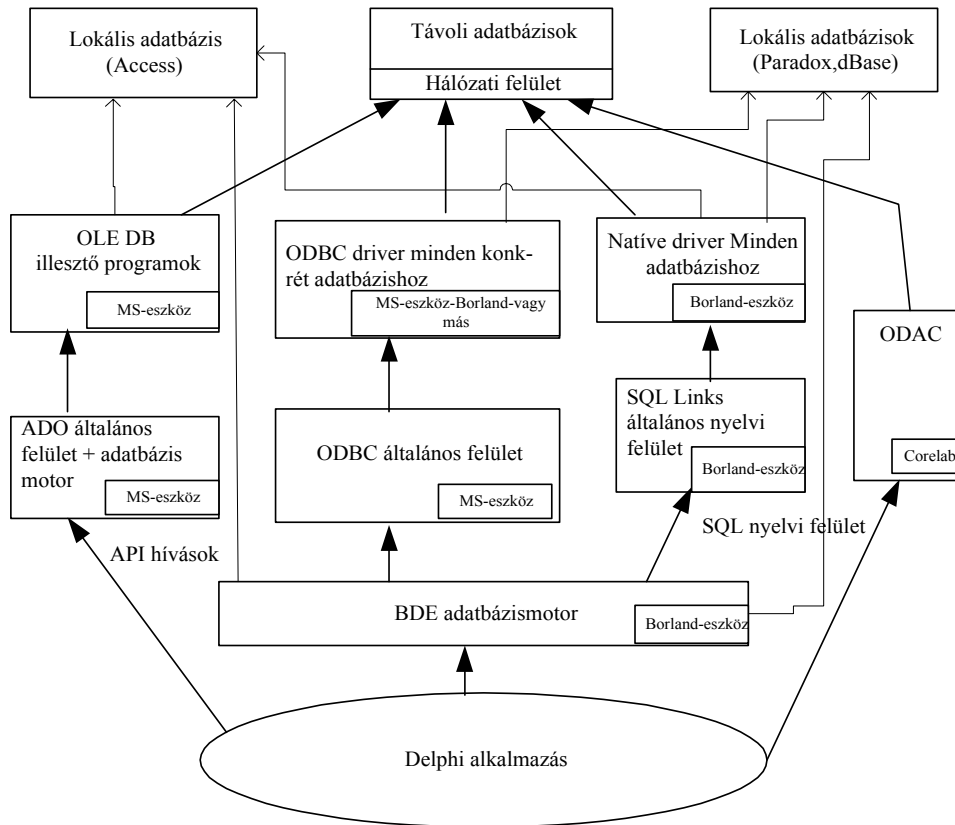
A BDE az adatbázis-állományokra alias (ál-, illetve másodlagos) nevekkkel hivatkozik. Alias neveket a Windows operációs rendszer vezérlőpultján található BDE adminisztrátor vagy a Delphi Database Explorer segítségével hozhatunk létre. Természetesen Delphi programból is létrehozhatók a `Session` komponens `AddStandardAlias`, `AddAlias` eljárásai segítségével.

A BDE API körülbelül 200 eljárást és függvényt tartalmaz, amelyeket a programozó általában nem közvetlenül ér el, hanem a Delphi komponensein keresztül. A BDE olyan

rutinokat tartalmaz, amely az adattáblák frissítésére, bejegyzések lezárására, alapvető I/O műveletek elvégzésére alkalmas.

A BDE számos adatforrást közvetlenül el tud érni. Ilyenek a Paradox, dBase, FoxPro és Access táblák. A BDE jól együttműködik a Borland SQL Links-ével, és ezen keresztül távoli SQL kiszolgálókhoz is kapcsolódhat. Az SQL Links tulajdonképpen illesztőprogramok összessége. Ez a lehetőség azonban csak a Delphi Enterprise változatában áll a felhasználó rendelkezésére. Távoli adatbázis kiszolgálók például az Oracle, a Sybase, vagy az Interbase.

A kapcsolódásokat a 11.1. ábrán láthatjuk.



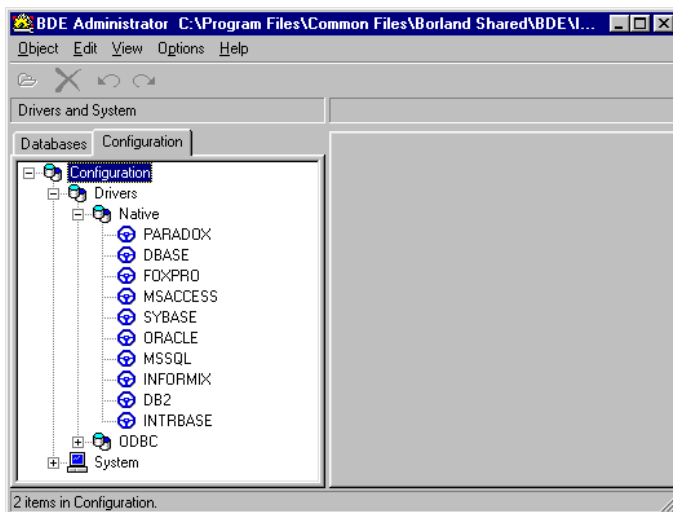
11.1 ábra Kapcsolódási felületek

Ha más (nem relációs) adatbázist vagy adatformátumot kell elérni, akkor a BDE kapcsolatba tud lépni az ODBC (Open DataBase Connectivity) meghajtókkal is. A BDE telepítése után párhuzamosan több alkalmazás is osztozhat rajta. Ha a program írásakor a BDE lehetőségeit kihasználjuk elérhetjük a lokális és távoli adatbázis-szerveren levő, illetve az ODBC meghajtók által támogatott állományokat is. A BDE olyan magasszintű

tulajdonságokkal is rendelkezik (gyorsítótárazás, heterogén táblaegyesítés), amelyekkel az ADO még nem. Ha azonban az ADO vagy a régebbi verzióknál kizárólagos ODBC között lehet választani, akkor az ADO használata javasolt.

A távoli adatbázis-kiszolgálókkal való kapcsolat létesítéséhez a Borland SQL Links meghajtók használhatók. Ezek megtalálhatók a Delphi Enterprise változatában, és az adott platform natív nyelvének megfelelően működnek. (Interbase, Oracle stb.). Ezt jól láthatjuk a Start/Beállítások/Vezérlőpult/BDE adminisztrátor programjának indításakor (11.2 ábra).

Ha a távoli adatbázis-kiszolgálókkal a kapcsolatot BDE natív driverén keresztül akarjuk megvalósítani, akkor az adatbázis adminisztrátor konfigurációs paneljét kell helyesen beállítanunk.



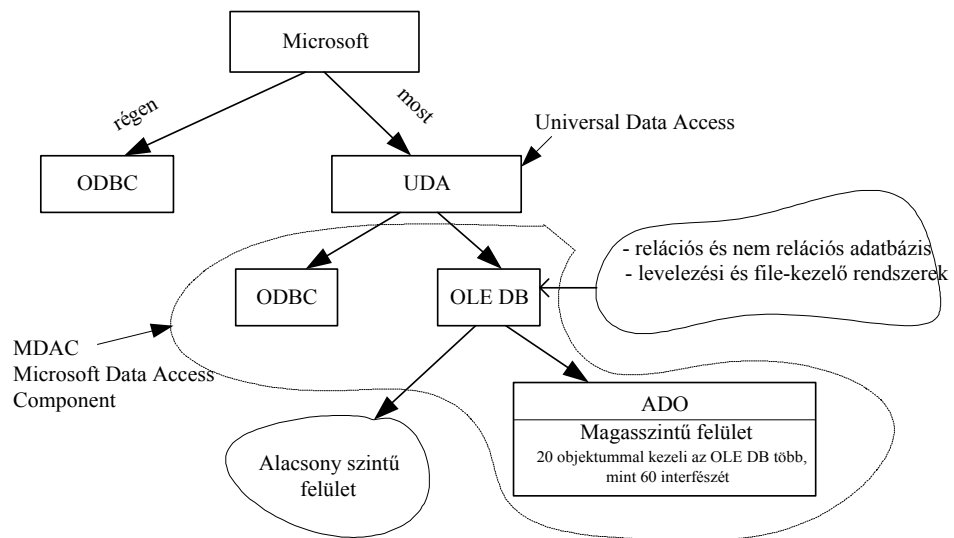
11.2. ábra. Adatbázis adminisztrátor

Ha egyéb, olyan kiszolgálót szeretnénk használni, amelyekhez a Borland még nem fejlesztett ki natív felületet, de az adatbázisnak van ODBC meghajtója, akkor a Start/Beállítások/Vezérlőpult/ODBC DataSource (32 bit) meghajtóprogramok (driver-ek) kiválasztásával az ODBC adminisztrátor segítségével hozhatjuk létre a kapcsolatot.

A Borland Enterprise változatában találjuk az ADO nevű felületet, amelynek technológiája kapcsolódik a Microsoft által kifejlesztett UDA (Universal Data Access) technológiához. Az UDA-t a Microsoft Data Access Component (MDAC) komponensgyűjtemény felhasználásával tudják megvalósítani (11.4 ábra). Ennek részei:

- ADO a Microsoft magasszintű felülete,
- OLE DB rendszerszintű felület (az OLE DB kapcsolódási felületek gyűjteménye, amelyek relációs és nem relációs adatforrások elérését biztosítják),
- a régebbi verziókkal való kapcsolattartáshoz szükséges az ODBC.

A komponens-paletta ADO lapján található azok a komponensek, amelyek a BDE megkerülésével az OLE DB-n keresztül érik el az adatforrásokat. Természetesen a felhasználóknak rendelkezniük kell ADO, illetve OLE DB futásidejű csomagokkal, melyek a Windows 2000 operációs rendszer részei.

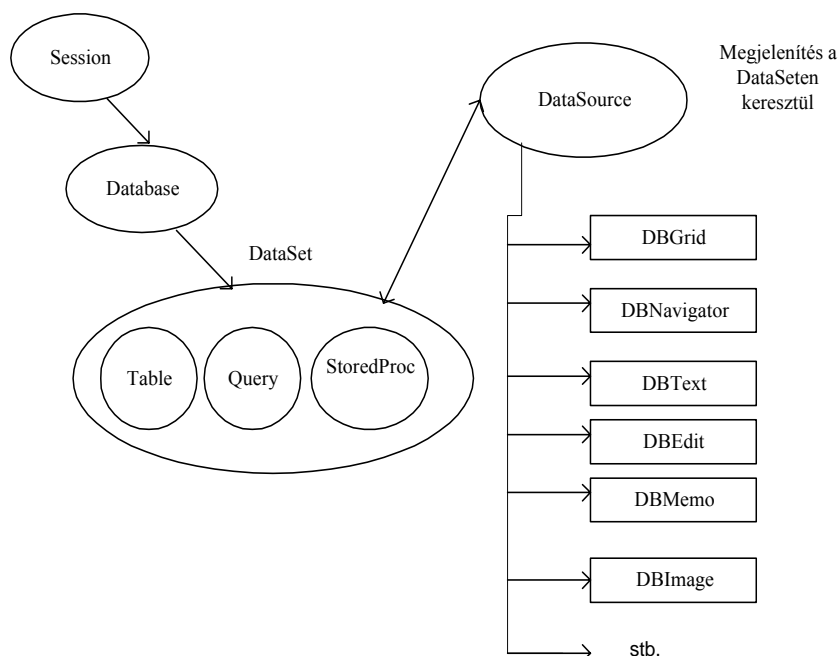


11.3. ábra A Microsoft kapcsolódási felületeinek kialakítása

## Adatbáziskezelő utasítások beágyazása Delphi környezetbe

A Delphi igen sok adatbázis-kezeléshez kapcsolódó komponenssel rendelkezik. Ezeket a Data Controls, Data Access valamint a BDE komponenspalettán találhatjuk meg. Ezek a komponensek a BDE központú adatbázisokhoz kapcsolódnak. BDE kapcsolattal a lokális adatbázisokat érjük el a legegyszerűbben.

A Delphi 6.0 verziójában a BDE komponens-paletta elemei - a Data Access komponens-paletta rovására – kibővültek. A BDE komponens-palettán találhatók meg az adatbázissal, ennek felügyeletével, az adatforrással (vagy más néven adatkészlettel, adathalmazzal) kapcsolatos komponensek. Itt találhatók a Table, a Query és a StoredProc komponensek is, melyeknek közös az őjük a TDataSet komponens. A komponensek kapcsolatát, valamint a Delphi adatbázis-kezelésben elfoglalt szerepét láthatjuk a 11.4. ábrán. Egy adatbázis komponens több adathalmazzal is kapcsolatban állhat, és ezt a DataSet tulajdonságnál lehet beállítani. Minden adatforrásnak, adathalmaznak ismernünk kell az állapotát (State), hogy az adatforrásban keresni, módosítani, az adatforrást megjeleníteni tudjuk. Az adatbázis, az adattábla és a lekérdezés megjelenítési komponenseinek kapcsolatát láthatjuk a 11.4. ábrán.



11.4. ábra Az adatbáziskezelő komponensek kapcsolata



### Megjegyzés

A Query komponensen keresztül a szabványosított SQL utasításoknak csak egy részét (Borland által támogatott Local SQL) használhatjuk.



A StoredProcedure komponens, azaz a tárolt eljárások végrehajtásának objektuma. Segítségével az adatbázis-szerveren tárolt eljárások futtathatók le. A tárolt eljárások is paraméterezhetők a Params tulajdonságon keresztül.



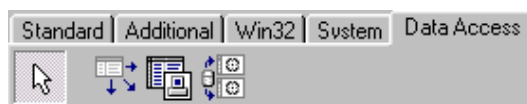
A BatchMove komponens segítségével az adatbázis-műveleteket végezhetünk a táblákon vagy résztáblákon.

A BDE palettán található még az adatbázisok késleltetett frissítésére szolgáló komponens (UpdatesQL), valamint a CliensDataSet komponens is.

## Data Access komponens-paletta

A komponenseknek egy része nem látható, nem vizuális komponens, azonban számos eszköz áll rendelkezésre, hogy a nem vizuális komponensek eredményeit megjelenítsük, és közvetlenül szerkeszteni tudjuk. A DataControls panel komponensei láthatóvá és szerkeszthetővé teszik a táblákat (11.6. ábra). Nem látható komponensek az adatkapcsolatok, táblák, lekérdezések eredményei, az eredménytáblák vagy egyéb.

A Data Access paletta komponensei az adatelérést biztosítják (11.6. ábra).



11.6 ábra Adatelérési komponensek.



A DataSource komponens. A Data Access komponens-paletta elemei közül ez a komponens az, amelyik az adateléréshez mindenképpen szükséges. Ez létesít kapcsolatot az adatbázis és az SQL lekérdezések, táblák valamint a megjelenítendő objektum között. A DataSource komponens akkor tud az adatokhoz hozzáférni, ha DataSet tulajdonságában megadjuk az adathalmaz, adatforrás nevét. A State tulajdonságban tárolt érték ad felvilágosítást az adatforrás aktuális állapotáról. A DataSource objektum OnStateChange eseményen keresztül a megjelent adatforrás állapota lekérdezhető.

Itt találhatjuk meg a kliens/server alkalmazásokhoz szükséges két komponenst (ClientDataSet, DataSetProvider).

## Data Controls komponens-paletta

A Data Controls paletta komponensei szolgálnak az adatok megjelenítésére (11.7. ábra).



11.7. ábra A Delphi adatmegjelenítési komponensei

Az adatmegjelenítő komponensek az adatforrásokon keresztül kapcsolódnak az adathozzáférést (`DataSource`) biztosító komponensekhez, ezért azok pillanatnyi állapotát tükrözik. A megjelenítő elemek a `DataSource` tulajdonságaikon keresztül kapcsolódnak a megjelenítendő adatforráshoz (táblához, lekérdezéshez). Az objektumok DB előtagja utal arra, hogy adatforrásból táplálkozik.



A `DBGrid` komponens. A rács komponens táblázatszerűen jeleníti meg a táblákat vagy lekérdezések által visszaadott sorokat. Ezt használjuk legáltalánosabban. Sose felejtjük el beállítani az adatforrással kapcsolatot létrehozó `DataSource` tulajdonságot azért, hogy a rácsban megjelenjen a tábla vagy lekérdezés eredménye. Ez az adatbázisrács minden tekintetben igazán felhasználó-orientált megjelenítést biztosít számunkra.



A `DBNavigator` komponens a táblák kezeléséhez, a táblákon belüli mozgásokhoz nyújt hatékony segítséget.



A `DBEdit` komponens lehetővé teszi, hogy a táblák sorait vagy lekérdezések eredményeit soronként (rekordonként) jelenítsük meg. Ha a rekordok egyes mezőit módosítani kívánja a felhasználó, akkor a `DBEdit` szerkesztődobozon keresztül ezt megteheti. Segítségével a tábla sorainak egy-egy oszlopértéke megváltoztatható.



A `DBText` olyan attributumot jelenít meg, amelyet nem változtathatunk meg. Ez a `Label` komponens adatbázisbeli megfelelője.



A `DBMemo` segítségével hosszabb szöveget adhatunk meg és módosíthatunk. Ez természetesen a szokásos `Memo` komponens adatbázis-kezeléshez rendelt megfelelője.



A `DBImage` segítségével egy képet jeleníthetünk meg.



A `DBListBox` komponens segítségével, előre megadott értékek közül egyiknek kiválasztásával módosíthatjuk egy mező értékét. A megváltoztatandó mező nevét a komponens `DataField` tulajdonságában kell beállítani.



A `DBComboBox` gyakorlatilag azonos a `DBListBox` komponenssel. A legördülő lista adatait kiválaszthatjuk vagy begépelhetjük.



A `DBCheckBox` a szokásos jelölőnégyzethez hasonlóan működik. Kétállású kapcsoló, általában értékek megváltoztatását teszi lehetővé. Általában logikai típusú mezők aktuális értékének megváltoztatására használjuk.



A `DBRadioGroup` segítségével az adatbázishoz kapcsolt választási lehetőségek közül jelölhetünk ki egyet, egyszerre mindig csak egyet.



A `DBRichEdit` segítségével formázni, szerkeszteni tudunk szöveget. Hasonló a WIN32 palettán található `RichEdit` komponenshez.



A `DBChart` a `Chart` komponens adatbázis változata.

## Adatbázis-elérés Delphiben

- BDE (lokális adatbázisok)
- BDE – SQL Links natív driveren keresztül (Dbase, Foxpro, Interbase, Oracle stb)
- BDE – ODBC-n keresztül távoli relációs és nem relációs adatbázisok elérése (régén).
- ADO – OLE DB kapcsolódási felületen keresztül (MsAccess, Oracle és bármilyen relációs, és nem relációs adatbázis-kezelők).
- ODAC – külső program, amely a BDE megkerülésével éri el az ORACLE adatbázist Delphiből.

### A Delphi BDE felülete, lokális adatbázisok elérése

A Delphi előre meg nem határozott adatbázis-szerkezettel dolgozik. Alkalmazásai nem közvetlenül kommunikálnak az adatforrásokkal, hanem a Borland Database Engine segítségével, amely a megfelelő meghajtó-programok útján közvetlen kapcsolatot tud kiépíteni az adatbázisokkal (pl.: dBase, Paradox, Foxpro, Access stb.)

Mivel a Delphi alkalmazások a BDE-n keresztül kommunikálnak az adatbázisokkal, az alkalmazásunkkal együtt a BDE-t is telepíteni kell.

A Borland Adatbázis Motor (Borland Database Engine – BDE) egy adatbázis-elérési kommunikációs eljárás (protokoll), amelyhez számos alkalmazás hozzáfér. A BDE API hívásokat definiál, amivel képes helyi és távoli adatbázis-kiszolgálók elérésére, és azok adatainak módosítására. A BDE egységes felületet kínál a legkülönbözőbb adatbázis-szerverek elérésére úgy, hogy azokhoz meghajtókon (driver) keresztül csatlakozik.

A BDE alapú adatbázis-kezelő alkalmazásokkal együtt terjesztenünk kell a BDE-t is. Ez a többlet növeli alkalmazásunk méretét, de a BDE-t meg tudjuk osztani több BDE alapú alkalmazás között, így a felhasználók gépére csak egy példányát kell telepítenünk.

Az adatbázisok kezelésénél az adatok helyét és formátumát valamilyen módon meg kell adni. A tervezési időben az adatoknak konkrét elérési útjuk van. Ha a programunkba beépítjük az útvonalat, akkor a program más gépen nem fut le, mert a régi (eredeti) helyen keresi az adatokat. Ezt küszöböli ki az aliasnév, az álnév használata. A fejlesztés közben létrehozunk egy alias nevet. Ez az álnév az IDAPI32.cfg konfigurációs állományba kerül. Ebből a konfigurációs állományból olvassa ki a BDE az álneveket. Ha így készítjük el a végrehajtható programunkat, akkor az csak az álnévre hivatkozik, amit a célgépen a telepítés végén elég az új elérési útvonalra átállítani. Új alias a BDE adminisztrátorának, vagy a Delphi Database Explorer programjának (menüpontjának) segítségével készíthetünk.

Az álnév használata az adatbázist helytől és konkrét formájától függetlenné teszi.

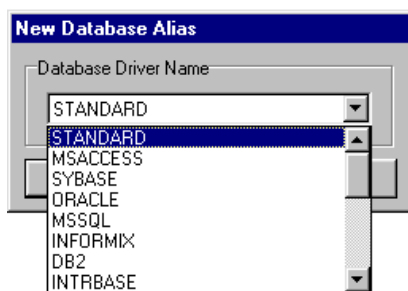
#### Példa

Alias (álnév) létrehozása.

Indítsuk el Start/Beállítások/Vezérlőpult/BDE Administrator programot.

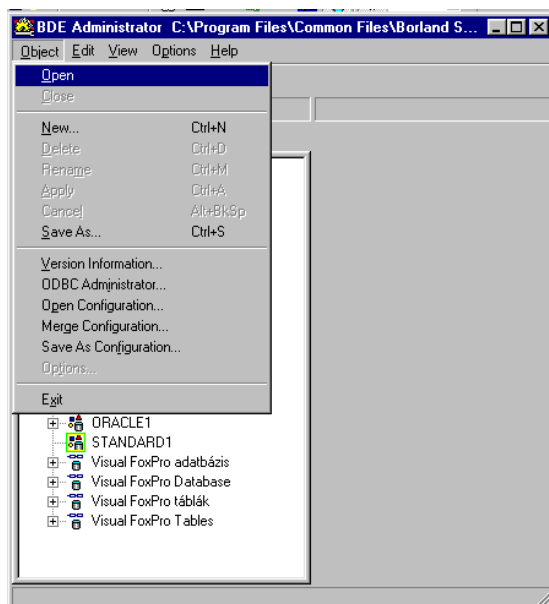
A BDE Administrator Database (Aliases) oldalán az álnevek listája látható. Ha új álnevet szeretnénk léterhozni, akkor az Object/New menüpontot használjuk. Ezután megjelenő New

Database Alias ablakban az új aliasnév formátumát kell megadnunk, illetve kiválasztanunk (11.8 ábra), majd az Apply menüponttal (11.9. ábra) az Idapi32.cfg állományba mentjük.



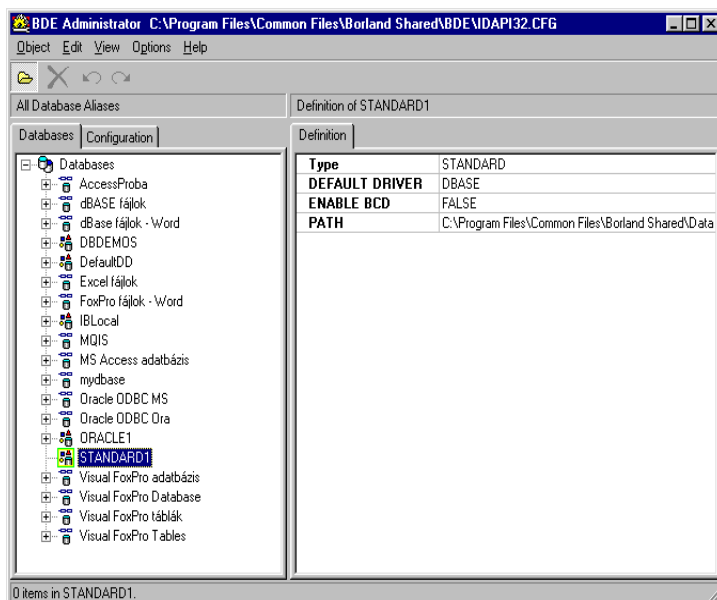
11.8. ábra Alias formátumának kiválasztása.

Mivel a lokális mintaadatbázis Paradox, így válasszuk a STANDARD típust. Létrehoz STANDARD1 néven egy új alias. Ezt a nevet természetesen átírhatjuk.



11.9 ábra Új alias létrehozása

Ezek után az Apply menüre kattintva létrehozza a STANDARD1 alias, amelynek jellemzői a 11.10. ábrán láthatók.



11.10 ábra Standard1 alias jellemzői

## BDE - SQL Links, adatbázis-elérés natív driverrel

### Az SQL Links kapcsolat

A BDE a távoli SQL kiszolgálók eléréséhez az Borland SQL Links programot használja. A Borland SQL Links for Windows programja, a BDE által támogatott meghajtók gyűjteménye adatbázis szerverekhez. Az SQL Links engedélyezi a felhasználóknak, hogy a Borland alkalmazásokból elérjék és manipulálják SQL adatbázisok adatait. Bármely Borland alkalmazás, amely a BDE-n alapul, képes használni az SQL Links-et. Az SQL Links programot csak a Borland cég fejlesztőeszközeinek Enterprise változatai tartalmazzák, úgymint a Delphi és a C++Builder. Az SQL Links termékcsomag tartalmaz adatbázis meghajtókat (database drivers) az Interbase, DB2, Informix, Oracle, Sybase és Microsoft SQL Server adatbázisokhoz. Ezekhez a meghajtókhoz szükséges az adatbázis forgalmazójának szerver kapcsolat API-ja, vagy interfész szoftverje (Oracle adatbázisnál az API, az OCI.DLL). A BDE tartalmaz beépített meghajtókat a Borland standard adatbázisaihoz is.

A Borland SQL Links programját használó alkalmazások adatátviteli sémája:

[Oracle RDBMS] ⇔ [SQL\*Net] ⇔ [OCI] ⇔ [SQL Links] ⇔ [BDE] ⇔ [Alkalmazás]

ahol : (RDBMS a Relációs adatbáziskezelő; OCI az Oracle Call Interface).

Az SQL Links használatának előfeltétele a következő szoftverek megléte.

- Hálózati protokoll az SQL szerverhez való csatlakozáshoz (TCP/IP, SPX/IPX, NetBeui, stb.),

- A szerver szállítójának (Sybase, Oracle, MS, etc.) kliens szoftverje (Oracle adatbázis esetén ez megtalálható a telepítő CD-n),
- Borland BDE és SQL Links (Előbbi minden Delphi verziónál megtalálható, utóbbi csak az Enterprise-nál),
- Futó Oracle szerver a hálózatban (Personal Oracle esetén erre nincs szükség).

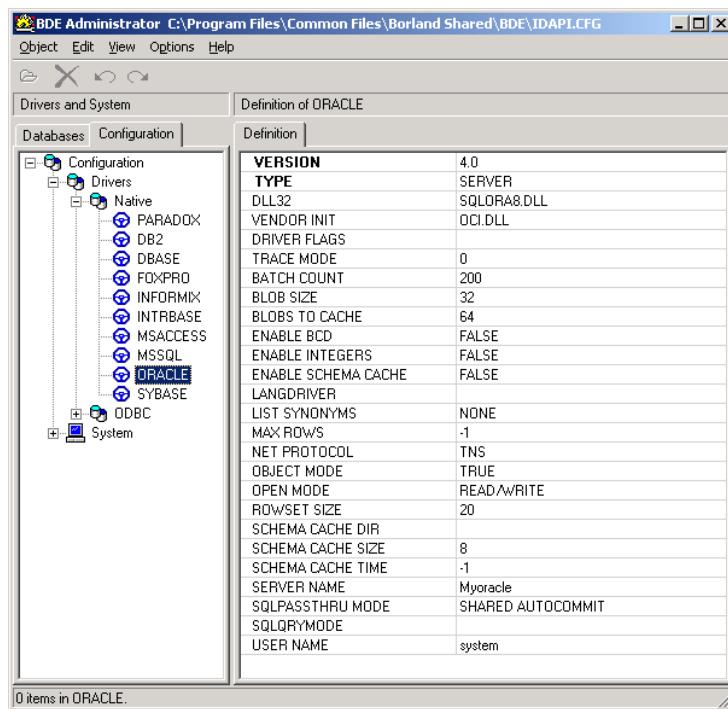
Ellenőrzés:

Nyissuk meg a *Start menü/Beállítások/Vezérlőpult /BDE Administrator* ikont. A Configuration lap *Drivers/Native* könyvtárában kell találnunk egy Oracle nevű oldalt. A 11.13. ábrán látható driver-ek a Delphi natív driver-ei, ezek között az Oracle meghajtója csak akkor szerepel, ha a Delphi Enterprise verzióját telepítettük számítógépünkre.

## A natív driverek

A natív driverek a Delphi beépített meghajtói. Zömükben helyi adatbázisok (adatbázis-állományok) eléréséhez szükséges meghajtók, de találunk köztük nem fájl-alapú szerverhez is meghajtót, mint például a MSSQL, SYBASE stb.

Az Oracle natív driverét minden Oracle verziónál konfigurálnunk kell, mivel a meghajtó 32 bites DLL-je és az inicializáló DLL-je a különböző verzióknál eltér. A driver beállításait Oracle8i-hez a 11.13. ábrán láthatjuk, ahol a SERVER NAME az Oracle installálásakor adott globális adatbázisnév.



11.13 ábra Oracle konfiguráció beállítása

A közvetlen natív driver-es elérés annyiban tér el az SQL Links-es technikától, hogy itt a BDE nem használja sem a hálózati protokollt, sem az adatbázis kliens szoftverjét. A kapcsolódásra a csatoló DLL-ek szolgálnak.

Natív driver-rel való csatlakozás feltételei:

- BDE Administrator eszköz,
- Az adatbázis DLL-jeinek és szerver nevének megfelelő beállítása,
- Futó Oracle szerver számítógépünkön (Personal Oracle esetén erre nincs szükség).

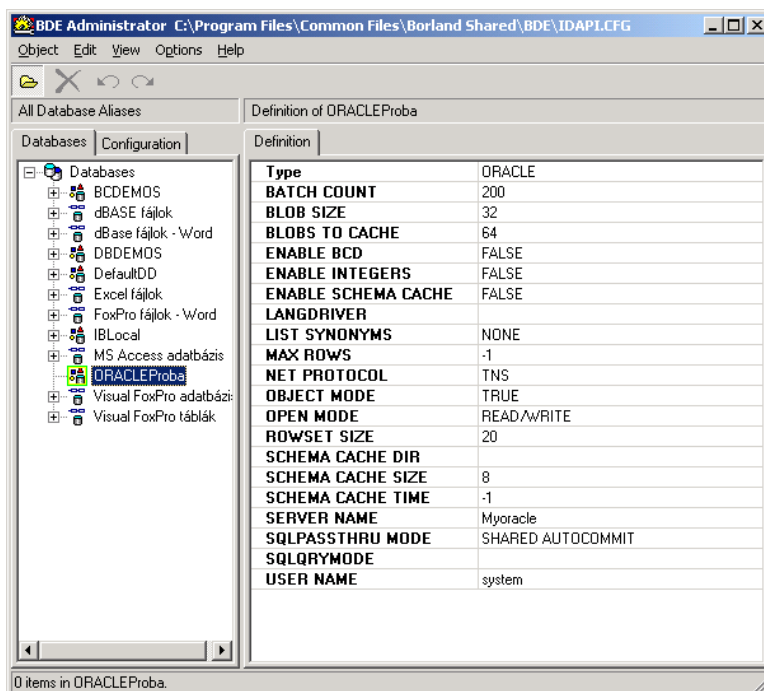
Oracle8i esetén a natív driver az SQLORA8.DLL, amelyik az Oracle adatbázissal az OCI.DLL-en keresztül tartja a kapcsolatot.

Hogyan állíthatjuk be a számunkra szükséges BDE natív drivert Win 2000 alatt? Indítsuk el a Start/Vezérlőpult/BDE Administrator/Configuration lapot (11.13. ábra).

Ebben az ablakban be kell állítanunk a SERVER NAME-t. Az Oracle installálásakor beállított *globális adatbázisnevet* írjuk ide be. Ezután már csak a USER NAME-t kell beírunk. Majd a beállításokat elmenteni az Object/Apply menüpontban.

Új álnév (alias) készítése natív driverhez.

Menjünk a BDE Administrator Databases lapjára, és indítsuk el az Object/New menüpontot. Egy legördülő nyíllal rendelkező választási ablak jelenik meg. Mivel a natív drivert az Oracle adatbázis eléréséhez szeretnénk használni, ezért a felajánlott lehetőségek közül, az új konfigurációra beállított Oracle-t válasszuk ki. Létrehoz egy alias-t, és felajánl egy ORACLE1 nevet. Meghagyhatjuk, de módosíthatjuk is ezt. Most változtassuk meg, legyen az új név ORACLEProba. Mentsük el a beállításokat az Object/Apply menüpontra kattintással.



11.14 ábra. A ORACLEProba alias definíciós panel

Mikor először rákattintunk a Databaseses lapon a létrehozott új aliasra, akkor megjelenik a szokásos jelszóbekérő ablak. Adjuk meg az általunk már jól ismert `system/manager`, illetve a `scott/tiger` felhasználó-jelszópárt. Ekkor, ha a kapcsolat létrejön, megjelenik a definíciós (Definition) panel (11.14 ábra), és itt félkövér betűkkel láthatók a beállított konfigurációk. A félkövér alak jelzi, hogy ezek nem változtathatók meg. A másik oldalon pedig az előtte beállított konfigurációs adatok. Így létrejött `ORACLEProba` alias néven a BDE natív driverén keresztül kapcsolat az Oracle adatbázissal. Most már elérjük a natív driver segítségével Delphi fejlesztői környezetből az Oracle adatbázist. Bármilyen táblát megjeleníthetünk, egyszerű és bonyolult, egy- és többtáblás lekérdezést írhatunk.

A második módszer a futásidőben történő létrehozás. Erre a `Session` komponens `AddAlias` eljárása szolgál. Használatát a későbbiekben példa mutatja.

## BDE – ODBC kapcsolat

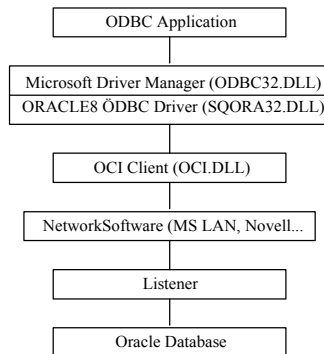
Az ODBC standard interfészt (standard függvényhívásokat) biztosít az alkalmazások számára azokhoz az adatbázisokhoz, amelyek ODBC driverrel (meghajtó programmal) rendelkeznek. Ezeket a drivereket az ODBC Driver Manageren keresztül érhetjük el.

Az ODBC lehetővé teszi egy alkalmazás számára, hogy több különböző típusú adatforráshoz csatlakozzon. Az adatbázis meghajtó (Database Driver) kapcsolja az alkalmazást a meghatározott adatforráshoz. Ez tulajdonképpen egy DLL (Dynamic Link Library).

Alkalmazásunk csakis olyan adatforrások elérésére képes, amihez létezik adatbázis meghajtó, ugyanakkor több meghajtót is használhat egy időben, amelyek különböző adatforrások elérését biztosítják.

Az ODBC interfész tartalmazza a következőket:

- Egy ODBC függvényhívásokat tartalmazó könyvtár, ami képessé teszi alkalmazásunkat az adatforráshoz való csatlakozásra, SQL parancsok futtatására, valamint eredmények fogadására.
- Az SQL szintaxisa standard (az SQL-92 specifikáción alapul).
- a hibakódokat standard táblába gyűjtötték, így azonos a különböző típusú adatbázisoknál.



11.15. ábra: Az ODBC alkalmazás – Oracle adatbázis adatátviteli sémája

Az adatátvitel állomásai az ODBC Driver használatával 11.15. ábra:  
 [Oracle RDBMS] ⇔ [SQL\*Net] ⇔ [OCI] ⇔ [Oracle ODBC Driver] ⇔  
 [MS ODBC Driver Manager] ⇔ [Alkalmazás]

Egy ODBC meghajtóval való csatlakozás feltételei:

- Az adatbázis ODBC meghajtója (ODBC driver),
- Microsoft ODBC Driver Manager installálása számítógépünkre,
- BDE Administrator eszköz.

ODBC drivert a Microsoft és az ORACLE is készít. Elvileg számítógépünkön mindkettőt megtalálhatjuk. A Microsoft-os drivert (Microsoft ODBC for Oracle) a Windows-zal együtt installáljuk, az Oracle driver-jét (Oracle ODBC Driver) pedig az Oracle adatbázissal.

Az ODBC meghajtók paramétereinek jelentését a C függelék 1. táblázata tartalmazza.

## ODBC driver beállítása Oracle adatbázishoz

A művelet elvégezhető a vezérlőpultból, illetve a Delphi/Database/Explorer menüjének segítségével.

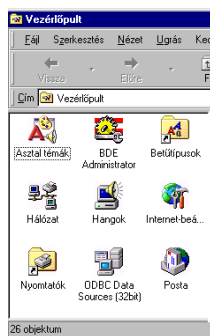
1.a.) Vezérlőpultból: Win 2000 operációs rendszer alatt.

Start/Beállítások/Vezérlőpult/Felügyeleti eszközök Adatforrások (ODBC) ikon. Válasszuk ki a Microsoft ODBC for Oracle drivert (vagy az Oracle ODBC driver), és kattintsunk a Befejezés gombra. Ekkor megjelenő párbeszédablakba írjuk be az adatforrás nevét (mi általunk adott, tulajdonképpen alias név). Mást itt nem kell kitöltenünk. Kattintsunk az OK gombra. Ellenőrizzük, hogy létrejött-e az új meghajtó.

Ugyanez elérhető a Start/Beállítások/Vezérlőpult/ BDE Adminisztrátor Object menü ODBC Administrator almenü segítségével is, vagy a Delphi Database/Explorer/Object, illetve a jobb egérgomb felbukkanó menüjében az ODBC Administrator almenü segítségével.

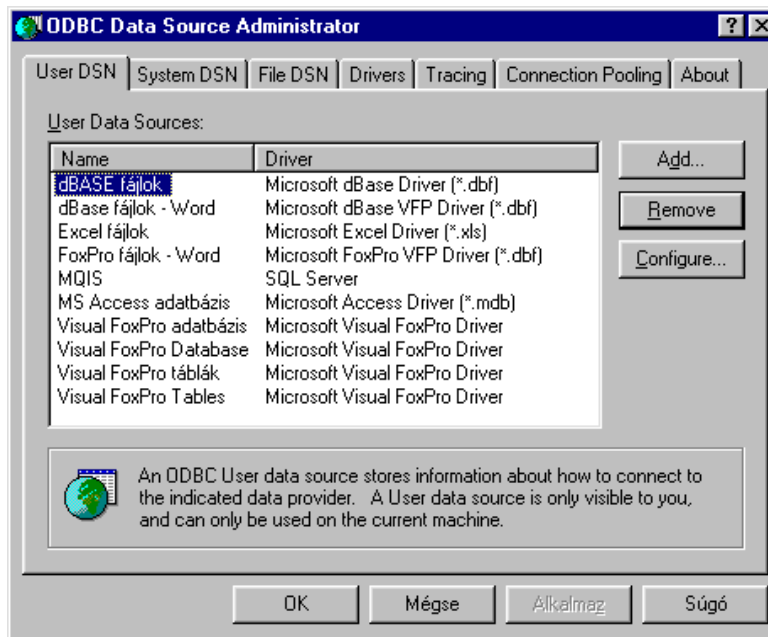
Start/Beállítások/Vezérlőpult/BDE Administrator/Database panel. Ha itt megjelent a mi általunk megadott alias név, akkor kattintsunk rá. Kéri a felhasználó nevét és a jelszót. Legyen ez scott/tiger. Ha a kapcsolat létrejött, akkor a táblák listája megjelenik. Ezután már készíthetünk Delphi fejlesztéseket.

1.b.) Hasonló a beállítás WIN 98 operációs rendszernél, csak abban a Felügyeleti eszközök program helyett, az ODBC DataSources(32) ikonhoz tartozó programot kell elindítani. (11.16. ábra).



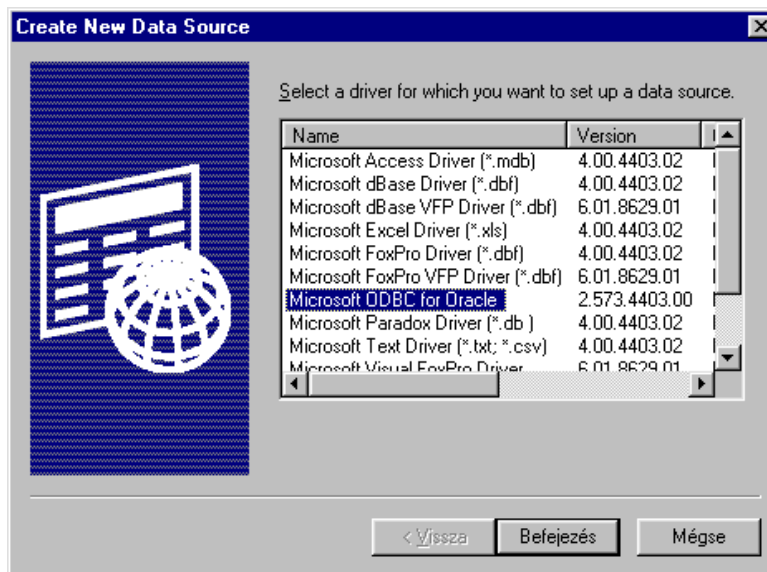
11.16. ábra Vezérlőmenü részlete

Indítsuk el az ODBC DataSources (32bit) programot(11.17. ábra).



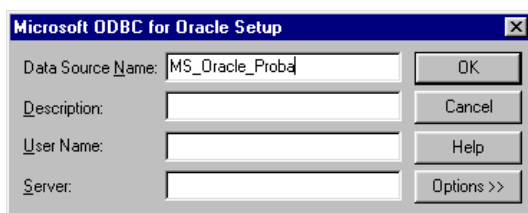
11.17. ábra ODBC Data Source Administrator

Válasszuk az **ADD** (Hozzáadás) gombot. A megjelenő ablakban jelöljük meg, hogy milyen drivert szeretnénk használni. Válasszuk a **Microsoft ODBC for Oracle**-t a megjelenő **Create New Data Source** menüből (11.18. ábra).



11.18. ábra Új adatforrás választása

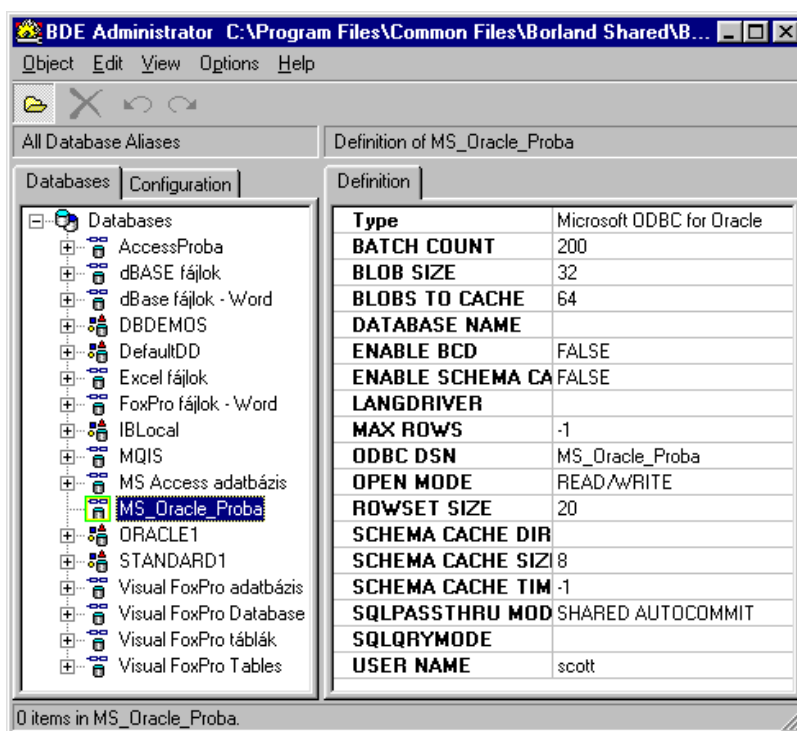
Kattintsunk a **Befejezés** gombra. Megjelenik a következő ablak, ami az adatforrás alias nevét kéri (DataSourceName), adjuk `MS_Oracle_Proba` álnevet (11.19. ábra).



11.19 ODBC adatforrás beállítása

Elég csak a nevet beírni, és újra OK. Ekkor a létrehozni kívánt `MS_Oracle_Proba` alias létrejött (11.20. ábra).

Ellenőrizhetjük a BDE Administrator Databases ablakban, hogy az alias-ok között valóban ott található-e az általunk létrehozott alias. Kattintsunk rá az alias nevére a + jelre, a megjelenő párbeszédablakban adjuk meg a felhasználó nevét és a jelszót. Ha nem ír hibaüzenetet, akkor a beállításunk sikeres volt. Ezzel a létrehozott adatbázis névvel írhatjuk Delphi fejlesztéseinket.



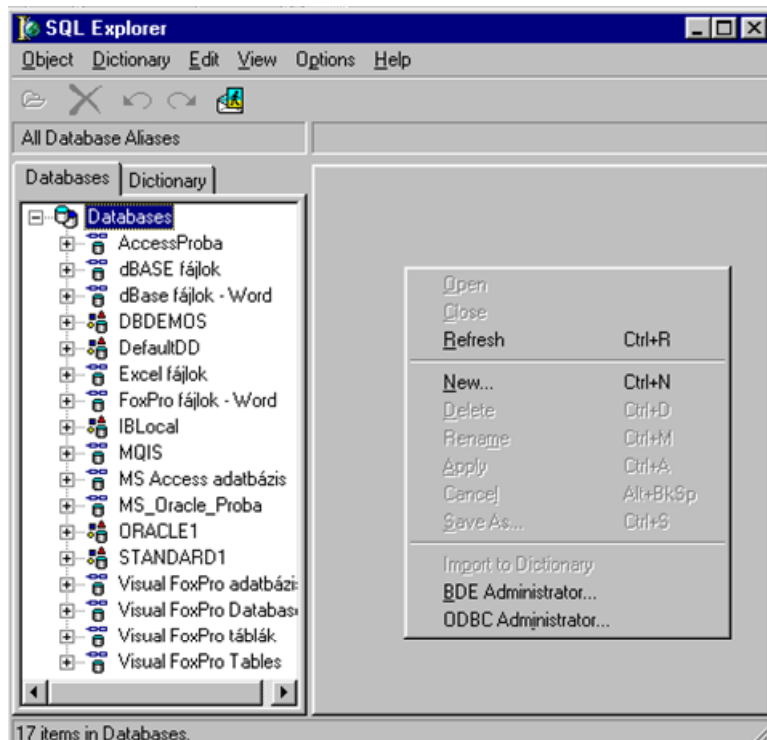
11.20 ábra Működő ODBC alias definíciós panelje

Az (11.20 ábra) ablakban láthatjuk a beállított konfigurációt.

Másik lehetőségünk az Oracle ODBC Driver választása.

## 2. Delphi Database menü Explorer almenüje segítségével.

Indítsuk el a programot. Az egér jobb gombjával előbukkanó menüből (11.21. ábra), vagy az Object menü ODBC adminisztrátorából induljunk el. A végrehajtás folyamata ugyanaz, mintha vezérlőmenüből hajtottuk volna végre. Az ellenőrzés szintén az Object menü, vagy jobb gomb rövid menü BDE adminisztrátor ablakában történik. A Delphi Database menüjének Explorer/Databases almenüjében viszont csak a Delphi újraindítása után jelenik meg az új alias.



11.21 ábra Delphi Database/Explorer ablaka

## ADO – OLE DB kapcsolódási felület

A Microsoft legújabb, és általa maximális támogatást élvező adatelérési technikája az ADO (ActiveX Data Objects). Az előző (DAO – Data Access Objects) és a mostani technika, nemcsak a két betű felcserélésében különbözik. Az ADO az OLE DB csatolófelületet alkalmazza az adatbázisok elérésére. Az OLE DB pedig az ODBC továbbfejlesztett változata, a relációs adatbázisokon kívül képessé teszi az őt használó alkalmazásokat a nem relációs adatbázisok elérésére is.

Az OLE DB az adatelérés egy nyitott szabványa, amely COM interfész-készletet használ a különböző adatbázisok elérésére és módosítására. Ezeket az interfészeket a különböző adatbázisok szállítójától szerezhetjük be.

Az Oracle OLE DB szolgáltató (Oracle Provider for OLE DB) nagy hatékonyságú elérést nyújt az Oracle adatbázisokhoz. Egy szolgáltató visszaadhat táblát az adatbázisból, és megengedi, hogy a tábla formátumát a felhasználó határozza meg, és műveletek is végezhet a tábla adatain.

Minden szolgáltató tartalmazza a COM interfészek gyűjteményét a felhasználótól érkező kérések kezeléséhez, de beépíthetünk opcionális interfészeket is a felhasználó igényeihez mérten. A COM komponenseket majd minden programnyelv eléri, így a Delphi is (C++, VB, Java stb.).

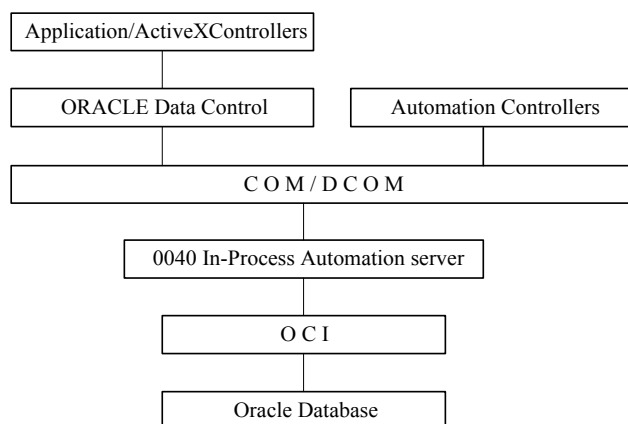
Rendszerkövetelmények:

Az ADO komponensek használatához szükséges eszközök:

- Windows 98 vagy NT 4.0 operációs rendszerek,
- az ADO 2.1 (vagy ennél magasabb verziójú) szoftver installálva legyen számítógépünkre. Az ADO és az OLE DB a Microsoft által támogatott, így a Windows-zal installálásra kerül.
- OLE DB és ODBC meghajtóknak installálva kell lenniük. Ezeket az ORACLE szállítja,
- Oracle adatbázis szerver 7.3.4 vagy magasabb,
- az Oracle kliens szoftverjének szintén fent kell lennie gépünkön. A kliens szoftvert szintén az ORACLE szállítja, és megtalálható az adatbázis telepítőlemezén,
- Net8 Client 8.1.6, az Oracle-lel installálásra kerül.

Az OLE-bázisú alkalmazások támogatását az Oracle adatbázis oldalán az OO4O végzi (11.22. ábra).

Az OO4O (Oracle Objects for OLE) az Oracle adatbázisok könnyebb adateléréséhez tervezett eszköz, amit a Microsoft COM Automation-t és ActiveX technológiát támogató fejlesztőeszközök használnak. Ilyen fejlesztőkörnyezet a Delphi is. (Idetartozik még a VB, VC++, VBA, VBScript és a JavaScript stb.).



11.22. ábra. A OO4O szoftver rétegei

Az ADO által felkínált technika egy a számos adatelérési típus közül, amelyekkel az OLE DB és ODBC meghajtókkal rendelkező adatbázisokat elérhetjük.

## Az Oracle adatbázis külső elérése (ODAC)

Oracle Data Access Components (ODAC) könyvtár Borland Delphi, C++ Builder és Kylix környezetben Oracle adatbázisok elérésére és kezelésére szolgáló komponenseket kínál. A programot a CoreLab nevű cég fejlesztette ki, és a Delphi különböző verzióihoz illesztett verziókat kínál (lásd az Interneten a [www.crlab.com](http://www.crlab.com) helyen). Az ODAC az Oracle Call Interface-t (OCI) közvetlenül használja. Ez a módszer lényegesen gyorsabb adatelérést biztosít, mint az ettől eltérő technikát használó alkalmazások (léteznek más, közvetlenül az OCI-t használó technikák is).

A BDE használatával az adatátvitel:

[Oracle RDBMS] ⇔ [SQL\*Net] ⇔ [OCI] ⇔ [SQL Links] ⇔ [BDE] ⇔ [Alkalmazás]

Az ODAC közvetlenül az Oracle Call Interface-szel dolgozik, így megkerüli a BDE és az SQL Links, az ODBC Driver vagy az OLE DB használatát. Ezzel az átviteli út számottevően csökken.

[Oracle RDBMS] ⇔ [SQL\*Net] ⇔ [OCI] ⇔ [Alkalmazás]

ODAC Net optimális átviteli utat biztosít:

[Oracle RDBMS] ⇔ [TCP/IP] ⇔ [Alkalmazás]

Az Oracle Data Access használatának előnyei:

- nem követeli meg az Oracle szoftverét a kliens oldalon, és nem kellene hozzá meghajtók,
- a Delphi Professional Edition is alkalmas kliens/szerver alkalmazások fejlesztésére (nem csak az Enterprise Edition),
- egyszerűsíti az adatmódosítást, felgyorsítja a rekord lehívását az adatbázisból, automatikusan zárolja és frissíti a rekordokat,
- az ODAC komponensek interfésze (tulajdonságok és eljárások) megfelelnek a standard BDE adathozzáférési komponensek tulajdonságainak, és eljárásaival (TDatabase, TQuery, stb.),
- támogat minden adatfüggő komponenst,
- tartalmazza az SQL Designer-t SQL lekérdezések és PL/SQL blokkok tervezéséhez és futtatásához stb.

## 12. FEJEZET

# Adatbázis-kezelés a gyakorlatban

Az adatbázis-kezelés a Delphi egyik kiemelt területe. Ebben a részben megtanulhatjuk, hogyan kell Delphiben adatbázist kezelni.

## Lokális adatbázisok kezelése

Eddigiekben megismertük a `DataAccess`, a `Data Controls` és a `BDE` komponens-paletta elemeit. Most elkészítünk néhány olyan példát, amelyek a Delphiben lévő minta-adatbázist, illetve mintatáblákat kezelik. A minta-adatbázis és a mintatáblák a `Program Files\Common Files\Borland Share\Data` könyvtárban találhatók.

### Példa (Tábla-megjelenítés, állapot-figyelés)

Első példánkban jelenítsünk meg a Formon egy rácsban egy, a `DBDEMOS` adatbázisban található táblát. A Form fejlécében jelenítsük meg az adatforrás, a `DataSource` (most a tábla) mindenkori állapotát.

Hogyan készíthetjük el ezt az egyszerű feladatot?

Indítsuk el a Delphit. A `New/Application` menüpont indít el egy új alkalmazást.

Helyezzünk a Formra a `DataAccess` komponens-palettáról egy `DataSource` komponenst, a `DataControls` komponens-palettáról egy `DBGrid` (rács) komponenst, és a `BDE` komponens-palettáról pedig egy `Table` komponenst. Az adamegjelenítő vizuális komponensen kívül az ikonok (`Table`, `DataSource`), csak fejlesztési időben látszanak, futási időben nem, így ezeket a Formon bárhova elhelyezhetjük.

Felhelyezés után jelöljük ki a `Table` ikont a Formon (látszanak a kis kijelölő négyzetek). Menjünk az `Object Inspector` (Objektum Felügyelő)-ra, és a `Properties` (Tulajdonság) lapon állítsuk be a következőket:

Tulajdonság beállításánál, a tulajdonság jobb oldalára kell kattintani. Ha megjelenik a nyíl, akkor egyet választanunk kell a legördülő listából, ha nincs nyíl, akkor beírhatjuk azt, amilyen tulajdonságot beállítani szeretnénk.

Kattintsunk a `DatabaseName` tulajdonságra, majd a nyíltra. A megjelenő listából válasszuk ki a `DBDEMOS` adatbázist. Ez az adatbázis a Delphi minta-adatbázisa. A minta-adatbázis tartalmaz `dBase (.dbf)`, valamint `Paradox táblákat (.db)`.

Állítsuk be a `TableName` tulajdonságot is hasonló módon. A legördülő listából válasszunk ki egy táblanevet (`animals.dbf` vagy `country.db` vagy más).

Mivel az adattábla tartalmát szeretnénk megjeleníteni, ezért `DataSource` objektum `DataSet` tulajdonságánál a legördülő listából a `Table1` elemet (most csak ezt ajánlja fel) válasszuk.

Most már meg tudjuk jeleníteni, de hol? Természetesen a rácsban, tehát a `DBGrid`-en keresztül. A `DBGrid` komponens `DataSource` tulajdonságánál válasszuk a `DataSource1`-t.

A `Form Caption` tulajdonságába írjuk azt, hogy *Adattábla megjelenítése*.

Most ezzel a kapcsolati rendszerrel már megjeleníthetjük az adatforrásunkat fejlesztési időben is úgy, hogy a `Table1` objektum `Active` tulajdonságát `True` értékre állítjuk. (12.1. ábra az `animals` adattáblát jeleníti meg a rácsban.)

NAME	SIZE	WEIGHT	AREA	BMP
Angel Fish	2	2	Computer Aquariums	(TYPEDBINARY)
Boa	10	8	South America	(TYPEDBINARY)
Critters	30	20	Screen Savers	(TYPEDBINARY)
House Cat	10	5	New Orleans	(TYPEDBINARY)
Ocelot	40	35	Africa and Asia	(TYPEDBINARY)
Parrot	5	5	South America	(TYPEDBINARY)
Tetras	2	2	Fish Bowls	(TYPEDBINARY)
cinege	12	23	Magyarország	TypedBinary

12.1. ábra Adatbáziskezelő form tervezési időben

Mentsük el a projektünket. A mentés mindig két lépésből áll, először a rendszer menti a unitot, mint Pascal unitot `.pas` kiterjesztéssel, majd a projectet `.dpr` kiterjesztéssel. A Delphi ezen kívül még készít számos állományt, amelyet ugyanebbe a könyvtárba ment. Ezek közül némelyik (`.dfm`, `.res`) feltétlenül szükséges ahhoz, hogy a programunk ismételt lefusszon, illetve más gépekre áthelyezhető legyen.

A feladat még a tábla állapotának lekérdezése, és megjelenítése a fejlécben.


A táblánk a `DataSource` objektumon keresztül jeleníthető meg. Ezért a `DataSource` objektum `Events` (események) lapján találjuk az `OnStateChange` eseményt, amely figyeli az adatforrás mindenkor állapotát. Kattintsunk kétszer az esemény jobb oldalára. Megjelenik az eseményhez rendelhető eljárás.

```
procedure TForm1.DataSource1StateChange(Sender: TObject);
begin

end;
```

Deklaráljunk egy `szoveg` nevű string változót, ennek segítségével írjuk majd ki a lekérdezett állapotot a Form fejlécében, és írjuk meg a következő `case` utasítást a felajánlott eljárásba:

```
procedure TForm1.DataSource1StateChange(Sender: TObject);
var
  szoveg : string;
begin
  case Table1.State of
    dsBrowse: szoveg := 'Tallóz';
    dsEdit   : szoveg := 'Szerkeszt';
    dsInsert: szoveg := 'Beszúrás';
    else
      szoveg := 'Másik állapot';
    end; //case
  Form1.Caption := 'RÁCS ÁLLAPOTA ' + szoveg;
end;
```

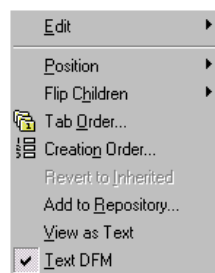
Mentsük el újra, és most már megpróbálhatjuk lefuttatni az elkészített alkalmazást. Futtatás vagy menüből, vagy a  zöld *jobbra mutató nyíl* ikonnal, vagy az *F9* billentyűvel lehetséges. Ha a program hibás, nem fut le, de a fordítási hibát az alkalmazás alatt kiírja, és színesen jelzi az első hibás sort.

Mire képes ez az egyszerű kis program. Képes a tábla lekérdezésére, képes új sorok beszúrására, és sorok törlésére is. Sorok (új rekordok) beszúrása az *Insert* billentyű segítségével történik, míg a rekordok törlése a *CTRL + Delete* billentyűvel. Az *Insert* billentyű hatására, (a kurzor aktuális helye előtt) egy üres sor jelenik meg. A megszorításoknak megfelelő értékeket meg kell adni, és ha ezt helyesen tesszük, a rekord bekerül a tábla sorai közé, mégpedig mindenkor a rendezettségnek megfelelő helyre.

A *CTRL + Delete* billentyű hatására a kurzor alatti rekord törlésre kerül, de csak akkor, ha a megjelenő üzenetre a válaszuk *OK*. (12.2. ábra)



12.2. ábra Rákérdezés a rekord törlésére



12.3. ábra A Form gyorsmenüje

Az első adatbáziskezelő programunk elkészült (40\_adat1). Érdemes mindig megfigyelni a Form szöveges leírását. Ezt megkaphatjuk:

- A Delphi programból, ha a Formon a jobb gombra kattintunk, és választjuk a *View as Text* menüt (12.3. ábra).
- Másik lehetőség a program könyvtárából a `.dfm` kiterjesztésű állomány megnyitása valamilyen text típusú szövegszerkesztővel.

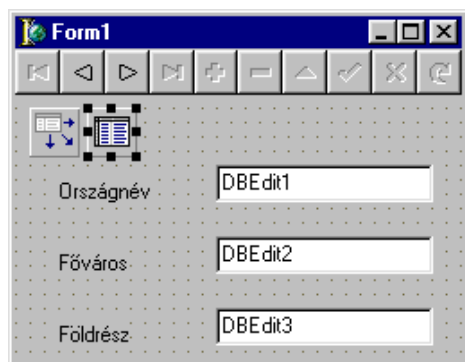
Például ha a `country.db` adattáblát kérdezzük le akkor a `DataSource` és `Table` objektumok leírása a következő lesz:

```
object DataSource1: TDataSource
    DataSet = Table1
    OnStateChange = DataSource1StateChange
    Left = 56
end
object Table1: TTable
    Active = True
    DatabaseName = 'DBDEMOS'
    TableName = 'country.db'
    Left = 104
    Top = 8
End
```

Futtatásnál figyeljük meg, amikor mozgunk a rács sorai között, vagy beszúrunk, vagy átjavítunk egy-egy értéket, milyen üzenetek jelennek meg a fejlécben.

### Példa (Rekordok megjelenítése, navigátor)

41\_adat2 példánkban (12.4. ábra) az előző feladathoz hasonlóan egy adatbázis-tábla tartalmát jelenítjük meg, de nem rácsban, hanem szerkesztődobozokban, és csak azokat az attribútumokat, amelyekre szükségünk van. A rekordok közötti mozgás, beszúrás stb. könnyítése céljából, helyezzük a Formra a gombokból álló ún. `DBNavigator` komponenst. Jelenítsük meg a `country.db` Paradox mintatábla egyes rekordjainak országnévét, fővárosát, és az országot tartalmazó földrészt.



12.4. ábra A 41\_adat2 feladat Formja

Helyezzünk a Formra egy `DataSource` komponenst a Data Access palettáról, egy `Table` komponenst a BDE palettáról, valamint a megjelenítéshez három `DBEdit` szerkesztődobozt, és egy `DBNavigator`-t a Data Controls komponens-palettáról. A Standard palettáról tegyük a három szerkesztődoboz elé három címkét (`Label` komponens), amelyek felirata a szerkesztődobozba kerülő oszlop neve legyen.

Kapcsoljuk össze megfelelően a komponenseket.

Beállítandó tulajdonságok:

```

Table1:      DatabaseName → DBDEMOS;
              TableName→ country vagy animals.dbf stb. lehet.

DataSource1: Dataset → Table1;
DBEdit1:     DataSource → DataSource1;
              DataField → a lenyíló listából válasszuk ki a meg-felelő oszlopnevet.
DBNavigator: DataSource → DataSource1;
              Hint tulajdonságában van az egyes gombok jelentése angolul. Írjuk át
              ezt magyarra.
              ShowHint → True,
              Align (igazítás)→ alTop

```

A DBNavigator gombjainak funkciói:

```

Ugrás az első rekordra
Ugrás az előző rekordra
Ugrás a következő rekordra
Ugrás az utolsó rekordra
Rekord beszúrása
Rekord törlése
Rekord szerkesztése
Az adatok változásainak mentése
Az adatok változásainak elvetése
Adatok frissítése

```

A rekordok egyes mezőit tudjuk módosítani, a rekordot törölni, beszúrni, a változtatásokat jóváhagyni, vagy elvetni, a rekordok között egyesével előre és visszafelé haladni, valamint előre és hátra ugrani.

Ahhoz, hogy a programunk működjön, meg kell nyitni programból a táblát. Ezt azonban egy eseményhez kell rendelni. Legyen ez a Form `OnCreate` eseménye. Tehát kattintsunk az Object Inspectorban a Form eseményénél az `OnCreate`-re kétszer, és írjuk be a megfelelő utasításokat.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    Table1.Open;
end;

```

Eddigi példáinkban az adatbázis táblákat kérdeztük le, és el tudjuk végezni az alapvető módosításokat az adattáblákon. Haladjunk tovább, térjünk rá az egyszerű lekérdezésekre, az egyszerű SQL utasításokra. Ezek a `Query` komponens segítségével valósíthatók meg, amely szintén a BDE komponens-palettán található.

### Példa (SELECT utasítás végrehajtása, eredménytábla megjelenítése)

(42\_adat3 könyvtárban található példa, amelynek futási eredménye a 12.5. ábrán látható). Készítsünk egy olyan Delphi felületet és programot, amely egy `SELECT` utasítás eredménytábláját jeleníti meg még fejlesztési időben. Futtatáskor legyen mód arra, hogy a programba írt `SELECT` utasítást végre tudjuk hajtani (Lekérdezés a programból jelölő négyzet). Képes legyen arra, hogy a lekérdezés feltételét a felhasználó aktuálisan adja meg (Feltétel az

Edit boxból jelölő négyzet). Képes olyan lekérdezésekre, amely azonos eseményekre más-más feltétel alapján adja vissza a sorokat (a két rádiógomb).

Hogy mindezt megvalósítsuk, szükségünk van egy új alkalmazásra (File/New/Application).

Helyezzünk a Formra egy Query komponens a BDE komponens-palettáról, egy DataSource komponens a Data Access komponens-palettáról. Ezeket kapcsoljuk össze a következő módon:

A DataSource1 komponens DataSet tulajdonsága legyen Query1.

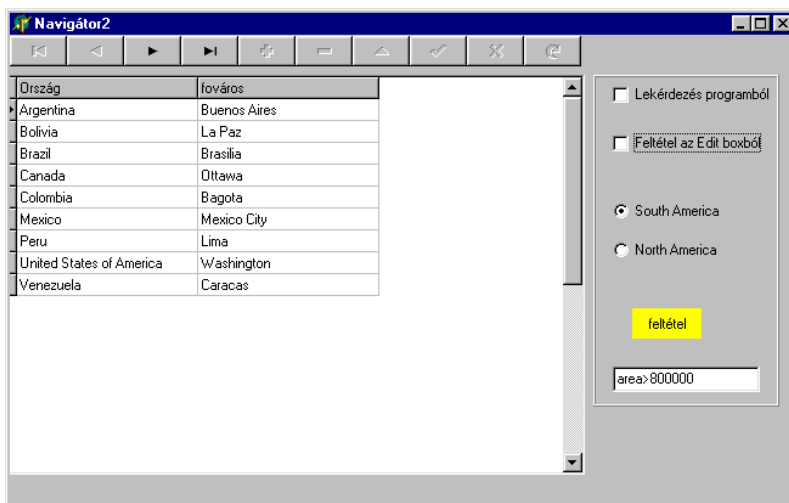
A Query1 komponens tulajdonságai közül a következőket állítsuk be:

DatabaseName legyen DBDEMOS (Ezzel létesítünk kapcsolatot az adatbázissal).

SQL tulajdonságra kétszer kattintva, megjelenik a String List Editor. Ebbe írhatunk SQL utasítást. (Egyszerre mindig csak egyet!). Ha a Query1 objektum Active tulajdonságát True értékre állítjuk, a lekérdezés eredménye azonnal megjelenik a rácspan. Természetesen csak akkor, ha a megítelenítést is, és a kapcsolatokat is beállítottuk.

Helyezzünk a Formra a Data Controls komponens-palettáról egy DBNavigator, egy DBGrid komponens, amelyeknek DataSource tulajdonsága legyen DataSource1.

Tegyük a rác mellé egy Panelt, erre két CheckBox (jelölőnégyzet) komponens, és két RadioButton (rádiógomb) komponens, egy Edit szerkesztődobozt és hozzá egy címkét. Ezeket a Standard palettán találjuk meg. Igazítsuk el a komponenseket szépen a Formon ahogy a 12.5. ábrán látható.



12.5. ábra Többféle feltételű lekérdezés programból

Ha a DBGrid1 komponens összekapcsoltuk a DataSource1-el, akkor a Query1 objektum Active tulajdonságát állítsuk True értékre. Azonnal megjelenik az SQL tulajdonságba írt lekérdezés eredménytáblája.

Írjuk meg a többi lekérdezést is.

Kattintsunk a Lekérdezés a programból jelölőnégyzetre, és írjuk meg a következő eljárást. (CheckBox1 objektum OnClick eseménye.)

A `Query1` komponens tulajdonsága (property `SQL`): `Tstrings` típusú. Ezért az `ADD` eljárással soronként adhatjuk hozzá az `SQL` utasítás (`SELECT` utasítás) sorait.

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    Query1.Close;
    Query1.Sql.Clear;
    Query1.Sql.Add('Select Name as Ország, ' +
                  'Capital as Főváros From Country');
    // Query1.Open;
    Query1.Active:= TRUE;
end;
```

`Query` használata esetén, először a `Queryt` *mindig be kell zárni*, ha szükséges a `Query` objektum `SQL` tulajdonságát töröljük. Következő utasítással töltjük fel az `SQL` tulajdonságot, amely `TString` típusú a `SELECT` lekérdezéssel, mint soronkénti stringgel. Ez a lekérdezés ország, főváros kétoszlopos eredménytáblát ad a `country.db` Delphi Paradox mintatáblából. *Utolsó lépésben nyitjuk meg a Queryt*, illetve hajtjuk végre a `SELECT` lekérdezést. A `Query` objektum lezárása a `Close` eljárással történik, megnyitása pedig az `Open` eljárással. Ugyanígy használható az `Active` tulajdonság `False`, illetve `True` értékre állítása.

Kattintsunk a `Feltétel` az `Edit Boxba` feliratú jelölőnégyzetre, és írjuk meg a `SELECT` utasítást. (`CheckBox2` objektum `OnClick` eseménye).

```
procedure TForm1.CheckBox2Click(Sender: TObject);
begin
    Query1.Close;
    if Edit1.Text <> ''
    then //Ha a szerkesztődoboz nem üres
        begin
            Query1.Sql.Clear;
            Query1.Sql.Add( 'Select Name as Ország, ' +
                          'Capital as főváros From country');
            Query1.Sql.Add('Where ' + Edit1.Text);
        end
    else
        ShowMessage('Adja meg a kiválasztási feltételt!');
    // Query1.Open;
    Query1.Active:= TRUE;
end;
```

Kattintsunk a `South America` feliratú rádiógombra és írjuk meg a harmadik féle lekérdezést. Itt a `WHERE` feltétel jobb oldalát a rádiógomb felirata adja. (`RadioButton2` objektum `OnClick` eseménye.)

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
    Query1.Close;
    Query1.Sql.Clear;
    Query1.Sql.Add('Select * from country ');
    Query1.Sql.Add('Where Continent =''' +
                  (Sender as TRadioButton).Caption +''');
    Query1.Open;
```

```
end;
```

Figyeljük meg nagyon jól az SQL tulajdonságba írt lekérdezésben az aposztrófokat. Az aposztrófok helytelen használatából nagyon sok hiba adódhat.

A North America feliratú rádiógombra nem írunk külön eseményt. Kattintsunk az Objektum felügyelőben a RadioButton3 gomb OnClick eseményére és válasszuk ki a RadioButton2Click eseményt. Így a két gombot egy közös eseményhez kapcsoltuk.

Mentsük el, majd futtassuk a programot. Figyeljük meg, hogy a lekérdezett eredménytáblában csak a rekordok közötti mozgást segítő navigátor-gombok lesznek aktívak, a többi nem. Tehát eredménytáblába beszúrni, törölni, módosítani adatot nem tudunk. Ezért ezeket a gombokat ki lehet szedni a navigátorból a VisibleButtons tulajdonságban. Azoknak a gomboknak a láthatóság tulajdonságát, amelyiket nem akarjuk látni False-ra kell állítani. Próbáljuk ki!

### Példa (Paraméteres lekérdezés.)

Szebb megoldása az előző feladatnak, ha paraméterezett lekérdezéseket használunk. (43\_adat4, és 44\_adat5 könyvtár programjai). A paraméter, olyan változó, amelyet futási időben a felhasználó megváltoztathat. A paraméterre hivatkozás jele a kettőspont (:).

Kérdezzük le a country.db tábla országnév, főváros, földrész oszlopait a kontinensre adott feltételnek, mint paraméternek használatával. A paraméterek most rádiógombok feliratai legyenek. Ha a táblába már írtunk újabb sorokat, amelyekben a mi földrészünk Európa is szerepel, akkor a futási formunk a következőképpen nézhet ki (12.6. ábra).

A megoldás a következő.

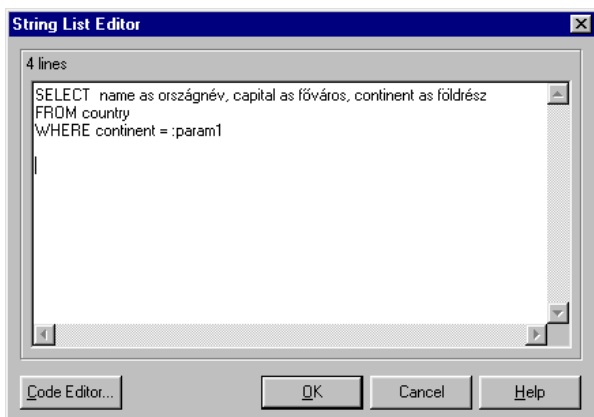
A Data Access komponens-palettáról egy DataSource komponenst, a BDE komponens-palettáról egy Query komponenst, a Data Controls komponens-palettáról egy DBGrid komponenst helyezünk a Formra. Kapcsoljuk ezeket a megfelelő tulajdonságaiknál össze.

országnév	főváros	földrész
Argentina	Buenos Aires	South America
Bolivia	La Paz	South America
Brazil	Brasília	South America
Chile	Santiago	South America
Colombia	Bagota	South America
Ecuador	Quito	South America
Guyana	Georgetown	South America
Paraguay	Asuncion	South America
Peru	Lima	South America
Uruguay	Montevideo	South America

12.6. ábra Paraméterezett lekérdezés eredménye

Legyen még három `RadioButton`, egy `Label` (ezek a Standard komponens-palettáról), valamint az Additional komponens-palettáról egy `BitBtn` képes gomb, amelynek `Kind` tulajdonsága `bkClose` legyen.

`DataSource1` komponens `DataSet` tulajdonsága legyen `Query1`. A `Query1` objektum `DatabaseName` tulajdonsága legyen `DBDEMOS`, és `SQL` tulajdonsága pedig 12.7. ábrán látható SQL tulajdonság szerkesztőjébe beírt lekérdezés. A String List Editorral mindig könnyen megírhatunk lekérdezéseket. A paraméter a feltétel részben szerepel, mégpedig a kontinens neve. A paraméterre hivatkozás a jelen lekérdezésben `:param1`.



12.7.ábra Az SQL tulajdonság szerkesztője

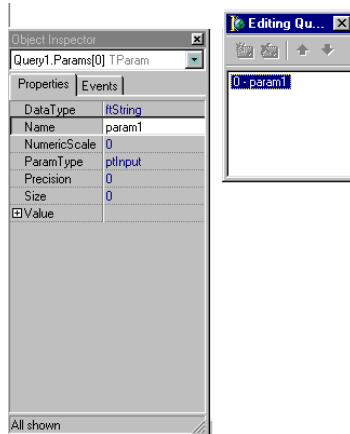
A `Query` objektum `Active` tulajdonsága fejlesztési időben legyen `False`. A `Form` `OnCreate` eseményénél aktivizáljuk, nyitjuk meg futási időben a `Query1` komponenst. Mielőtt megnyitnánk a `Query1` objektumot, használjuk a `Query1` objektum `Prepare` eljárását, amely előkészíti a lekérdezéseket még a megnyitás előtt. Ezzel gyorsítja a paraméterezett lekérdezéseket. A `Prepare` eljárás még a lekérdezés előtt erőforrásokat foglal le a paraméterezett lekérdezések számára (ezeket fel is kell szabadítanunk a munka befejeztével, az `UnPrepare` eljárás aktivizálásával). Az `ExecSQL` eljárás hatására a `Prepare` eljárás aktivizálása automatikusan történik.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Query1.Prepare;
    Query1.Open;
end;
```

Felszabadtítás és a programból való kilépést a Bezár feliratú gomb `OnClick` eseményébe írjuk meg a következőképpen:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    Query1.UnPrepare;
    Query1.Close;
    Application.Terminate;           //Kilépés az alkalmazásból
end;
```

A paramétereket a Query1 paraméterszerkesztőjében definiálhatjuk (12.8. ábra).



Meg kell adnunk a Name tulajdonságot, amelyre az adatbázis-kezelő utasításokban kettősponttal tudunk hivatkozni. A DataType (adattípus) tulajdonságot a megfelelő felajánlott adattípusok közül, a ParamType (paraméter típusát), szintén a felajánlottak közül választhatjuk ki. Megadhatjuk a Size (méret), és a Value (érték) adatokat is, ha alapértelmezett (default), vagy előre ismerjük, és nem változik meg.

12.8. ábra A Paraméter Szerkesztő

A paraméteres lekérdezés aktivizálása a rádiógombok OnClick eseményének hatására következnek be. Mivel három azonos működésű rádiógombunk van, ezért egyikhez írhatjuk az eseményt, és a másik kettő is ezt a közös eseményt használja fel. Írjuk meg az OnClick eseményt az első rádiógombra kattintáshoz.

Ez a következő lesz:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    Query1.Close;
    Query1.ParamByName('param1').Value := (Sender As TRadioButton).Caption;
    // Query1.Params[0].AsString:= (Sender As TRadioButton).Caption;
    Query1.Open;
end;
```

A programrészéből láthatjuk a paraméteres hivatkozásokat. Mind a

Query1.ParamByName('param1').Value := , mind a Query1.Params[0].AsString := hivatkozás megfelelő. Ez utóbbi hivatkozás típusfüggő, az As előtag után a megfelelő típus következik.

A paraméter természetesen interaktív módon is érkezik a felhasználótól, vagy gördülő listából kiválasztás útján (valamilyen ComboBoxból).

Ha a paramétereket Edit szerkesztődobozból kapjuk a következő hivatkozásokat szükséges tennünk (44\_adat5 könyvtár példája, 12.9. ábra):

Bármelyik gombra kattintunk a következő utasítások hajtódnak végre:

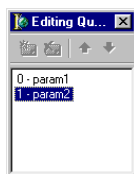
```
procedure TForm1.Button2Click(Sender: TObject);
begin
    Query1.Close;
    Query1.ParamByName('param1').Value:=Edit1.text;
    Query1.ParamByName('param2').Value:=Edit2.text;
    Query1.Open;
end;
```

név	főváros	kontinens
Canada	Ottawa	North America
Cuba	Havana	North America
El Salvador	San Salvador	North America
Jamaica	Kingston	North America
Mexico	Mexico City	North America
Nicaragua	Managua	North America

12.9. ábra Paraméteres lekérdezés

A `Query1` objektum `SQL` tulajdonsága a következő:

```
SELECT name as név, capital as főváros, continent as kontinens
FROM country
WHERE continent = :param1 AND
      population <= :param2
```



12.10. ábra Két paraméter a Paraméter Szerkesztőben

A `Query1` objektum `Params` tulajdonságában, a Paraméter Szerkesztőben (12.10. ábra) tehát két paramétert kell megadni.

### Példa (Tábla létrehozás, rács formázás, számított oszlop)

A következő példánkon hozzunk létre egy új táblát, fűzzünk hozzá egy új számított oszlopot. Formázzuk meg szépen a rácsban megjelenő táblázatunk oszlopait, valamint adjunk segítséget gördülő listákkal az egyes oszlopok attribútumainak beírásához.

Haladjunk végig lépésenként a feladat megoldásán.

Helyezzünk a Formra egy `Panel`t a Standard komponens-palettáról. A `Panel` objektum `Align` tulajdonsága legyen `alTop`. A `Panel` szélére tegyünk fel a Data Access komponens-palettáról egy `DataSource` komponenst, és a BDE komponens-palettáról egy `Table` komponenst. Kapcsoljuk ezeket össze. A `DataSource` objektum `DataSet` tulajdonsága legyen `Table1`. A `Panel`-ra helyezzük el a `DBNavigator` komponenst, amelynek `DataSource` tulajdonsága `DataSource1` legyen. A navigátor nagyban segíti a tábla feltöltését, módosítást, javításokat és az adatbevitel érvényesítését is.

A Form kinézetét a 45b\_adat6b feladatnál láthatjuk a 12.11. ábrán.

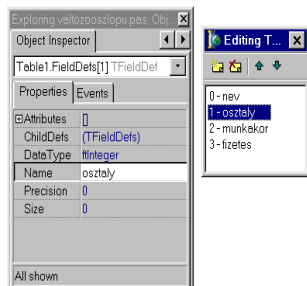
név	osztály	munkakör	fizetés	jutalom
Madár	20	kereskedő	999	499,5
Zubor	10	alkalmazott	900	450
Kovács	15	tanár	1111	555,5
Balogh	30	mérnök	1120	560
Kelemen	20	jogász	1000	500

12.11. ábra A Form futásidőben

Hogy a Formunk igazán felhasználó-orientált legyen, helyezzünk a Formra egy `Splitter` komponenst is az `Additional` komponens-palettáról. A `Splitter` objektum `Align` tulajdonsága is legyen `alTop`. Ezután tegyük rá a `Data Controls` palettáról a `DBGrid` komponenst, amelynek `Align` tulajdonsága legyen `alClient`. Ilyen felhelyezés után futásidőben a `Splitter` segítségével a rács méretét függőleges irányban a `Panel` rováására változtatni tudjuk.

Hogyan hozhatunk létre új paradox táblát a `DBDEMOS` default adatbázisban?

A `Table1` objektum `DatabaseName` tulajdonságát állítsuk `DBDEMOS` adatbázisra, `TableName` tulajdonságánál pedig adjunk egy nevet (például `alkalmazott.db`). Az `alkalmazott.db` táblánk a következő oszlopokat fogja tartalmazni: `nev`, `osztaly`, `munkakor`, `fizetes`. Ezeket a mezőket a `Table1` objektum `FieldDefs` ... tulajdonságánál kell beállítanunk. Kattintsunk rá és nyissuk ki a mezőszerkesztőt. (12.12. ábra)



12.12. ábra Mezőszerkesztő

A jobboldali egérgombra kattintva előbukkan egy gyorsmenü, amelynek `Add` almenüje segítségével új oszlopokat definiálhatunk az új táblánkhöz.

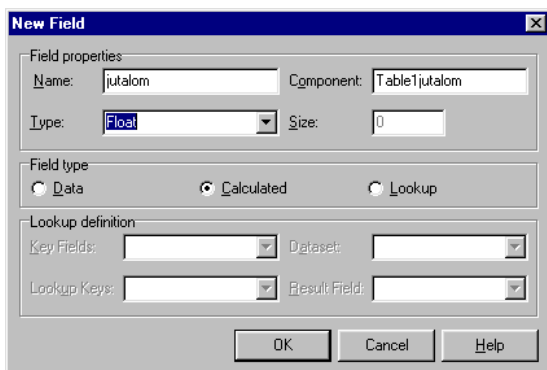
A megjelenő első mező `0-Table1.Field1` névre kattintva az Object Inspectorban szerkeszthető, definiálható a mező. Meg kell adnunk a `Name`, valamint a `DataType` tulajdonságokat, illetve ha tudjuk a többi tulajdonságot is. Ha ezzel elkészültünk a `Table1` objektum `StoreDef` tulajdonságát ellenőrizzük, hogy a tulajdonság `True` értékű legyen. Ezek után, ha a mezőket beállítottuk, írjuk meg a `Form OnCreate` eseményéhez a következőket: ha a táblánk nem létezik, hozzuk létre és nyissuk meg. Ezt az alábbi részlet szemlélteti.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    if not Table1.Exists
    then
        Table1.CreateTable;
    Table1.Open;
end;
```

Ezután fűzzük a már meglévő táblánkhoz egy új számított oszlopot. Az oszlop legyen a `jutalom`, és ezt a `fizetes` oszlopból számoljuk ki. Jelöljük ki a `Formon` levő `Table1` objektumot, és kattintsunk az egér jobb gombjára, majd kattintsunk a `Fields Editor ...` menüpontra. Ekkor jelenik meg a tábla mezőszerkesztője. (12.13. ábra) Az egér jobb gombjára kattintva bukkannak elő a menüpontok. Adjuk hozzá egyesével (`Add Fields...`), vagy egyszerre (`Add all Fields`) menüpontok segítségével az eddigi összes oszlopot. Ezután adjuk hozzá az új mezőt a `New Field` menüpontra kattintással (12.14. ábra.)



12.13. ábra Tábla mezőszerkesztője

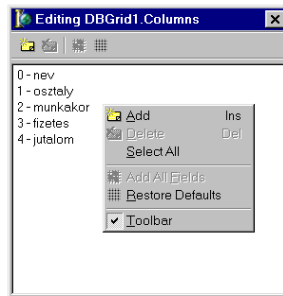


12.14. ábra Már létező táblához új oszlop adatainak definiálása

Legyen az új oszlop neve `jutalom` és típusa `Float`. A mező számított lesz, ezért kattintsunk a `Calculated` rádiógombra, majd az `OK`-ra. A `jutalom` legyen a `fizetes` fele. Írjuk be a `Table1` objektum `OnCalcFields` eseményéhez a következőket:

```
procedure TForm1.Table1CalcFields(DataSet: TDataSet);
begin
    Table1jutalom.Value:= Table1fizetes.Value/2;
end;
```

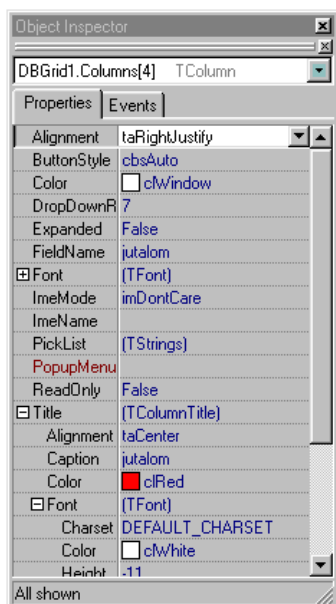
Most nézzük meg, milyen lehetőségek vannak a `DBGrid` objektum formázására. A rács `DataSource` tulajdonságát állítsuk `DataSource1`-re. Az oszlopfejlécek formázása a `DBGrid` objektum `Columns` tulajdonságában történik. Ha rákattintunk megjelenik az oszlopszerkesztő (12.15. ábra), és erre az egér jobb gombjával kattintva megjelenik az előbukkanó menü, amelynek `Add` almenüjére kattintsunk, akkor megjelenik `0-TColumn`. Kattintsunk erre, megjelenik az `Object Inspektor` oszlopszerkesztője (12.16. ábra). Válasszuk ki a `FieldName` tulajdonságban a megjelenő oszlop nevét. A `Title` (oszlopfejléc) tulajdonságot nyissuk ki, a `Caption` tulajdonságnál írjunk be közérthető magyar nevet. Változtathatjuk az oszlopfejléc színét a `Color`, betűtípusát is a `Font` tulajdonságnál, valamint az oszlopfejléceket megfelelően igazíthatjuk az `Alignment` tulajdonságnál. Segítsük a tábla feltöltését azzal, hogy olyan oszlopokban, ahol a mezők adatai, az attribútum értékek csak néhány közül kiválaszthatóak, egy gördülő listába helyezzük.



12.15. ábra A rács oszlopainak felvétele, oszlopszerkesztő

A `PickList` tulajdonságra kattinsunk, és írjuk be a választható adatokat. Például a `osztaly` lehet 10, 15, 20, 25 és 30. Ezeket írjuk be egymás alá. Ugyanígy tehetünk a `munkakor` oszlopban is, ahol az alkalmazottak foglalkozásait írhatjuk be. Ha ezt megtettük, futtatáskor a sor beírásnál ebbe a mezőbe kerülve, megjelenik a nyíl, amire kattintva a gördülő listában a beírt választási lehetőségek megjelennek.

Most már színezhetjük, betűtípust választhatunk magának az oszlopoknak a megjelenítéséhez. Ezt már az Olvasóra bizzuk.



12.16. ábra Az oszloptulajdonságainak beállítása

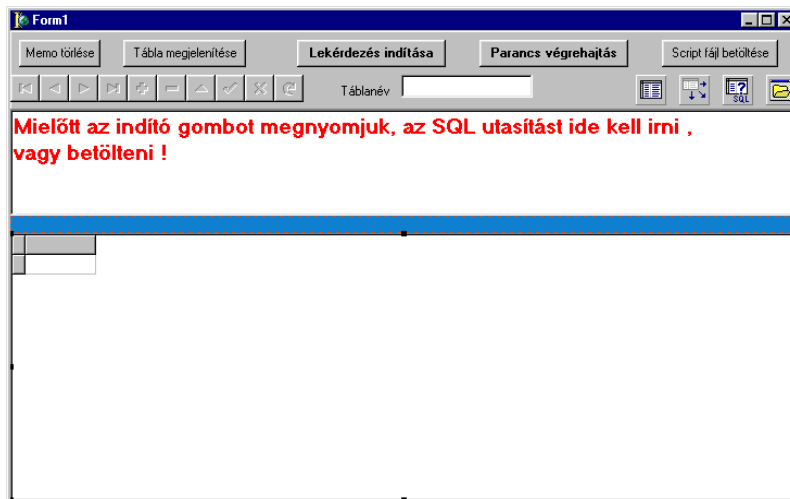
### Példa

Az Oracle beágyazásokhoz talán legjobban használható mintapélda lesz a következő. Készítsünk egy olyan Delphi felhasználói felületet, amelyik egy táblát jelenít meg egy rácsban, valamint a felhasználó által a futás közben megadott SQL utasításokat hajt végre. Ha az SQL utasításokat a felhasználó már megírta script (.sql) programba, a Delphi program akkor is végre tudja hajtani az SQL utasításokat.

Az SQL utasítások lehetnek lekérdezések, ezeket a `Query` komponens `Open` eljárásával tudjuk végrehajtani, lehetnek DDL és DML utasítások, azaz rekordok (sorok) beszúrása, módosítása, törlése, táblák létrehozása, módosítása, törlése, amelyeket a `Query` komponens `ExecSQL` eljárásával hajthatjuk végre. Ezek az utasítások nem adnak vissza a végrehajtás után sorokat, így közvetlenül nem tudjuk megjeleníteni, csak – az SQL utasítások végrehajtása után – a módosított táblát lehet a rácsban megjeleníteni. Ezért szükséges megadnunk a tábla nevét (például egy `Edit` szerkesztődobozban, vagy esetleg másban), hogy mindig azt a táblát jelenítsük meg, amelyiket szeretnénk. Mindezeket megoldja a következő bemutatandó példa, amelynek Formja a 12.17. ábrán látható.

A Formon található egy `Panel` a Standard komponens-palettáról (`Align` tulajdonsága legyen `alTop`). Ezen a `Panel`-en helyeztünk el az öt normál gombot. A feliratok egyértelműek. Itt található még egy `DBNavigator` komponens is a Data Controls komponens-palettáról, amely a tábla megjelenítése esetén segíti a mozgást, esetleges változtatásokat, és egy `Táblanév` feliratú `Edit` szerkesztőmező is. Alatta egy `Memo` komponent helyezünk el a Standard komponenspalettáról (`Align` tulajdonsága legyen `alTop`), és ebbe írhatja be a felhasználó az SQL utasításokat, vagy töltheti be az SQL utasításokat tartalmazó script programokat. Következő komponens a Formon egy `Splitter` az Additional komponens-palettáról (`Align` tulajdonsága szintén `alTop`). Végül helyezzük fel a tábla vagy lekérdezések eredményeinek megjelenítésére szolgáló `DBGrid` komponent

(Data Control paletta), amelynek `Align` tulajdonsága `alClient`. Ilyen elrendezés esetén futásidőben a rács és `memo` mérete változtatható.



12.17. ábra A Form felülete

Helyezzünk egy `DataSource` komponenst a Data Access komponens-palettáról, egy `Table` és egy `Query` komponenst a BDE komponens-palettáról, és egy `FileDialog` komponens a Dialogs komponens-palettáról.

Kapcsoljuk össze az elemeket. A `DataSource` `Dataset` tulajdonsága legyen először `Table1`, a `Table1` objektum `DatabaseName` tulajdonsága legyen `DBDEMOS`, `TableName` tulajdonsága pedig az előző feladatban létrehozott tábla neve, azaz `alkalmazott.db` vagy természetesen más is lehet (`country.db`, `stb.`). Ha a `Table1` objektum `Active` tulajdonságát `True` értékre állítjuk, a tábla sorai azonnal megjelennek a rácsban.

Írjuk meg a programrészeket az egyes gombok `OnClick` eseményeihez.

Első gombunk a Memo üritése. Kattintsunk kétszer a Memo törlése feliratú gombra, és írjuk be az eljárásba a Memo törlési utasítást, és a Focust helyezzük a Memo-ba:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    // Memo üritése
    Memo1.Clear;
    Memo1.SetFocus;
end;
```

Kattintsunk kétszer a Tábla megjelenítése feliratú gombra, és írjuk meg az eljárás törzsét. A program legyen képes bármelyik, az `Edit` szerkesztőmezőben megadott nevű tábla megjelenítésére. A `Table` objektumot először mindig be kell zárni, majd ha megadtuk a tábla nevét, akkor lehet megnyitni.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    // Az Editben megadott paradox tábla megjelenítése
    Table1.Close;
    DataSource1.DataSet := Table1;
```

```

Table1.TableName := Edit1.Text;
Table1.Open;
end;

```

A Form harmadik gombjának felirata `Lekérdezés indítása`. Ha erre a gombra kattintunk, akkor a program képes olyan SQL utasítások végrehajtására, amelyek sorokat adnak vissza. A program legyen alkalmas tehát `SELECT` utasítások végrehajtására, és a lekérdezések sorai jelenjenek meg a rácsban. Ezért a megjelenítendő adatforrás a lekérdezés eredménye, azaz a `Query` eredménye legyen. Fejlesztési időben viszont a `Table1` komponenst jelenítettük meg, ezért futásidőben át kell állítanunk a megjelenítendő komponenst a `DataSource` objektumban. Ennek `DataSet` tulajdonságához a `Query1` objektumot kell rendelni. Mivel a `Query` most csak `Select` utasításokat fog végrehajtani, ezért a végrehajtás az `Open` eljárással történhet.

A gomb `OnClick` eseménye tehát a következő:

```

procedure TForm1.Button3Click(Sender: TObject);
begin
    Query1.Close;
    Query1.Sql.Clear;
    DataSource1.DataSet :=Query1;          // Megjelenítés hozzárendelése
    // Query1.Sql:= Memo1.Lines;          Ez is jó.
    Query1.Sql.Assign(Memo1.Lines);
    Query1.Open;
end;

```

Ez az `OnClick` eljárás egy-, vagy többtáblás, egyszerű, vagy bonyolultabb `SELECT` utasításokat végrehajt.

Az SQL utasításoknak van olyan fajtája is, amely nem ad vissza sorokat, ezek a DDL, és DML utasítások, tehát a módosítások, beszúrások, törlések. Ezeknek az utasításoknak a végrehajtása a `Query` objektum `ExecSQL` eljárásával történik. Írjuk meg a `Parancs végrehajtása` feliratú gomb `OnClick` eseményéhez a `Memo`-ba betöltött, sorokat vissza nem adó SQL utasítások végrehajtását. (Az `ember.db` egy kétszlopos tábla, amelynek oszlopai. `nev`, `fiz`)

```

procedure TForm1.Button4Click(Sender: TObject);
begin
    Query1.Close;
    Query1.Sql.Clear;
    // Query1.Sql:= Memo1.Lines;          Ez is jó.
    Query1.Sql.Assign(Memo1.Lines);
    Query1.ExecSQL;
    // például a következő utasításokat lehet végrehajtani
    // insert into ember values('Vasorrú Bába', 666)
    // insert into ember values('Komisz Varga', 777)
    // update ember set fiz=888 where nev='KomiszVarga'
end;

```

Végezetül, hogy a programunk teljes legyen, és hogy script programból a `Memo`-ba töltött SQL utasításokat is végre tudjuk hajtani, ezért az utolsó `Script` program betöltése feliratú gomb `OnClick` eseménye a következő lesz:

```

procedure TForm1.Button5Click(Sender: TObject);
var
    f:   TextFile;           // script program beolvasása
    sor: string;
begin
    Memo1.Clear;
    Opendialog1.Filter:='SQL file(*.sql)|*.sql|All files (*.*)|*.*' ;
    Opendialog1.FileName:='';
    If Opendialog1.Execute
    then
        begin
            AssignFile(f,Opendialog1.FileName);
            Try
                Reset(f);
                While not EOF(f) do
                    begin
                        readln(f,sor);
                        Memo1.Lines.Add(sor);
                    end;
            Finally
                closefile(f);
            end;
        end;
    end;
end;

```

Deklaráltuk lokális változónak az `f` logikai fájlt, és a `sor` nevű string típusú segédváltozót. A `Memo` tartalmát először töröljük, így már betölthető a script program tartalma. Ehhez meg kell nyitnunk az állományt, amelynek szűrését is beállítottuk. Megnyitás előtt az állomány-névnek "üres" értéket adtunk, hogy ha a megnyitás párbeszédablakban a Cancel gombra kattintunk, akkor a program hibátlanul futhasson tovább. Megnyitás után az `f` logikai állománynévhez a megnyitandó állomány nevét rendeljük. Ezután `Try...Finally` blokkban nyissuk meg az állományt (az állományt mindig be kell zárni), olvassunk belőle az állomány végéig sorokat, és ezeket a sorokat helyezzük a `Memo`-ba.

A `Memo` komponensből pedig a `Lekérdezés` indítása vagy a `Parancs` végrehajtása gombbal végrehajtható a betöltött script program, attól függően, hogy DQL vagy DML, DDL utasítást tartalmaz.

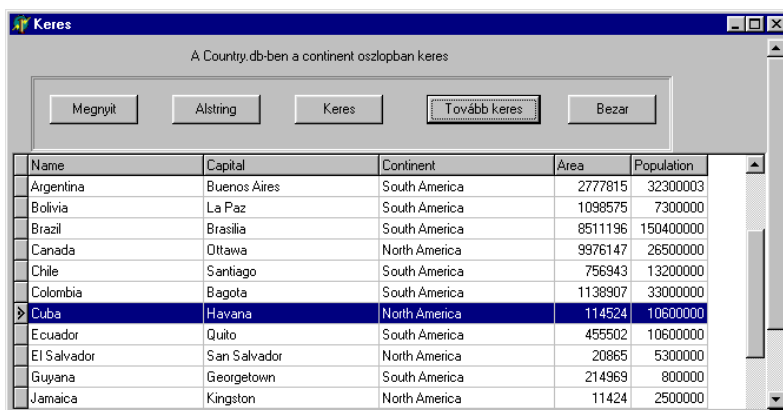
Tudnunk kell azt, hogy a `Query` komponens egyszerre csak egy SQL utasítást képes végrehajtani.

## Adatfeldolgozás

### Példa

Példánkban bemutatjuk, hogyan lehet feldolgozni a rácsban megjelenített táblát. A programunk a rácsba betöltött tábla egyik string típusú oszlopában keres soronként egy alsztringet, karaktersorozatot. A találati sort jelölje meg. Először kérjük be az alsztringet, majd legyen egy gomb az első találat megkeresésére és kijelölésére. Ezután keressünk majd tovább az adathalmaz végéig, és mindig az aktuális találatot jelöljük ki.

A feladat `Keres` Formja a 12.18. ábrán látható.



12.18. ábra Keres feladat futási képe

A Formon egy Panel komponensen öt gomb található. A Megnyit és Bezár feliratú gomb szolgál a Table objektum megnyitására, megjelenítésére illetve bezárására. Az Alstring feliratú gomb bekéri a keresendő alsztringet. A Keres gomb megkeresi, és kijelöli az első olyan rekordot, amelyben a keresett alsztring megtalálható. A Tovább keres feliratú gomb, megkeresi, és aktuálisan kijelöli a keresett alsztringet tartalmazó sorokat a betöltött tábla végéig vizsgálva.

A Formon elhelyeztünk egy Table és egy DataSource komponenst. Ezek futásidőben nem látszanak. Kapcsoljuk össze a komponenseket, az eddig megismert tulajdonságok megfelelő beállításával. Ezek után írjuk meg a gombok onClick eseményeit. A Megnyit feliratú gomb eseménye a tábla aktivizálása:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Table1.Open;
end;
```

A Bezárás gomb onClick eseménye pedig:

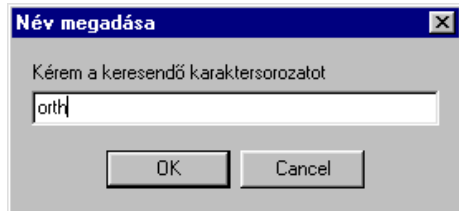
```
procedure TForm1.Button5Click(Sender: TObject);
begin
    Table1.Refresh;
    Table1.Close;
end;
```

Hogy programunk működjön, az egyes eljárások elérjék a számukra szükséges azonos adatokat, ezért ezeket a unit interface részében deklaráljuk. Ilyenek a szoveg és a nev elnevezésű string típusú változók, valamint a talal elnevezésű logikai változó.

```
var
    Form1: TForm1;
    szoveg, nev : string;
    talal : boolean;
```

Az `Alstring` feliratú gomb (`Name` tulajdonság legyen `alstring`) `OnClick` eseménye eredményezi a keresendő karaktersorozat megadását (12.19. ábra), a `szoveg` változóba tároljuk el a keresendő karaktersorozatot.

```
procedure TForm1.alstringClick(Sender: TObject);
begin
    szoveg:=Inputbox('Név megadása',
                    'Kérem a keresendő karaktersorozatot','');
end;
```



12.19. ábra A bekérési `Inputbox` párbeszédablak

Ha bekértük a keresendő karaktersorozatot, akkor indíthatjuk el az első keresét.

A `Keres` feliratú gomb `onClick` eseménye tehát az alábbi:

A `Keres` feliratú gomb `Name` tulajdonságát áttértük `Keres`-re, hogy biztosan eligazodjunk a megfelelő eljárásokban.

Az eljárás törzsében a kezdeti értékeket állítjuk be legelőször. A `Talal` változó kezdetben legyen `False`, mivel nincs találat. A `DBGrid` objektum esetleges előző kijelöléseit szüntessük meg, mégpedig úgy, hogy a kijelölt sor aktuális-sorkijelölés tulajdonsága legyen `False`.

```
SelectedRows.CurrentRowSelected:=False;
```

Mivel a keresés folyamatában, minden lépés alatt az adatforrást folyamatosan jeleníti meg a rács, ezért minden keresés után is újra és újra megjeleníti. Ez borzasztóan lelassítja a program működését. Kapcsoljuk ki a folyamatos megjelenítést (`DisableControls:=False`), és majd találat után jelenítsük meg a rácsban a rekordokat.

Mielőtt a kereső ciklust elindítanánk, állítsuk a kurzort az első rekordra a `Table1` objektum `First` eljárásával. Ezután indíthatjuk a keresési ciklust mindaddig, amíg az adatforrás (adattábla) végére nem érünk. Ezt vizsgálhatjuk a `Table1` objektum `EOF` függvényének `True` állapotával. Az `EOF` függvény, vagyis az `End of File` függvény, amelynek értéke igaz, ha megtalálta az adattábla végét. A `BOF` (`Beginning of File`) függvény az adattábla elején ad vissza igaz értéket.

Az átláthatóság kedvéért azt az oszlopértéket helyezzük egy `Nev` elnevezésű string típusú változóba, amelyikben az alsztringet keressük. Keresésre a Delphi `pos` függvényét használjuk. A `pos` függvény visszatérési értéke adja meg a találat helyét. Ha ez a hely, pozíció nulla, akkor nem találta meg a keresendő karaktersorozatot. Ha a `pos` keresőfüggvény megtalálta a keresett karaktersorozatot, állítsuk a `talal` változót `True` értékre, és ugorjunk ki a keresési ciklusból (`Break`). Ha nem találta meg keresendő karaktersorozatot, akkor lépünk tovább, keressük az alsztringet a következő rekordban. Ne feledkezzünk meg a kurzor továbbléptetéséről. Ez a `Table1` objektum `Next` eljárásával történik.



```

TRY
  while not EOF do
    begin
      // Ujrakeresés estén a kijelölés megszüntetése
      DBGRID1.SelectedRows.CurrentRowSelected:=False;
      Next;          // következő
      nev := fieldbyname('continent').Value;
      if Pos(szoveg,nev) >0 then
        begin
          talal:= True;
          DBGRID1.SelectedRows.CurrentRowSelected:=True;
          Break;
        end;
      end;
    end;
  Finally
    EnableControls; // Adatmegjelenítés bekapcsolása
  end;
  if not talal
  then
    if EOF
    then
      Showmessage('Nincs több') ;
    end;
  end;
end;

```

## Paradox táblalétrehozása és kezelése

### Példa

Utolsó kidolgozott feladatunk a Delphi paradox (default adatbázisában) a Partnerkeresés. Hozzunk létre, ha még nincs (nem létezik) egy egész egyszerű dolgozo.db nevű paradox adattáblát. A táblának legyen három oszlopa, amelyeknek elnevezése nev (string), fizetes (integer) és partner (string) oszlop. Töltsük fel adatokkal a nev és fizetes oszlopot, és töltsük fel a partner oszlopot „\*\*\*”-gal. Keressünk minden dolgozóhoz partnert (ha még nincs), vagyis azt a kollegáját, akinek fizetése a saját fizetésének 0,9 és 1,1-szeres tartományába esik, és még senkinek sem partnere. Ha van ilyen, akkor írjuk be az első feltételnek megfelelő dolgozót a partner oszlopba, a megtalált név partner oszlopába pedig a kereső dolgozó nevét.

A feladat megoldása a 49\_Partner+ könyvtárban található.

A feladat Formja a 12.20. ábrán látható.

nev	fizetes	partner
Kukás Bácsi	200	XXXX
Kovács	130	XXXX
Kelemen	135	XXXX
Balogh	126	XXXX
Magyar	132	XXXX
Kalapos	121	XXXX
Pintér	130	XXXX
Mari Nénje	201	XXXX
Lázár	202	XXXX

12.20. ábra A Partnerkeresés Formja

A partnerkereső Delphi alkalmazás unitja:

```
unit PartnerU;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,
  Dialogs, StdCtrls, DB, DBTables, Grids, DBGrids, ExtCtrls, DBCtrls;

type
  TForm1 = class(TForm)
    DataSource1: TDataSource;
    AdatTabla: TTable;
    Letrehozogomb: TButton;
    DBGrid1: TDBGrid;
    InicializaloGomb: TButton;
    NevSzerkesztoAblak: TEdit;
    FizetesSzerkesztoAblak: TEdit;
    Hozzafuzogomb: TButton;
    NevCimke: TLabel;
    FizetesCimke: TLabel;
    Keresogomb: TButton;
    DBNavigator1: TDBNavigator;
    procedure LetrehozogombClick(Sender: TObject);
    procedure InicializaloGombClick(Sender: TObject);
    procedure HozzafuzogombClick(Sender: TObject);
    procedure KeresogombClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
end;
```

```

var
    Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.LetrehozoGombClick(Sender: TObject);
begin
    AdatTabla.Close;
    If not AdatTabla.Exists
    then
        AdatTabla.CreateTable;
    AdatTabla.Open;
    if Not AdatTabla.Active
    then
        AdatTabla.Active := true;
end;

```

Az inicializálás itt a `partner` oszlop feltöltését jelenti három csillaggal (\*\*\*) . A tábla egy adott mezejére, oszlopára hivatkozni többféleképpen lehet, amint ezt a programrészletben láthatjuk. Ha beszúrtunk egy mezőbe egy-egy adatot (csillagokat), érvényesítsük ezt azonnal az eredeti táblán is a `Table1` objektum `Post` eljárásával.

```

procedure TForm1.InicializaloGombClick(Sender: TObject);
// A partner mező inicializálása
begin
    AdatTabla.First;
    While not AdatTabla.EOF do // amíg nincs vége az adathalmaznak
    begin
        AdatTabla.Edit; // legyen az adattábla szerkeszthető módban
        //AdatTabla.FieldName('partner').Value := '***';
        //AdatTabla.FieldName('partner').AsString := '***';
        //AdatTabla.FieldValues['partner'] := '***';
        AdatTabla['partner'] := '***';
        AdatTabla.Post;
        AdatTabla.Next;
    end;
    AdatTabla.First;
end;

```

A `Hozzáfuz` gomb `OnClick` eseménye, egy – egy új sort, új alkalmazottat fűz hozzá az adathalmazhoz. Az új sor mindkét attribútumának értéke az `Edit` szerkesztődobozból, mint paraméter kerül be az adattáblába. Ellenőrizzük le, mielőtt a beszúrást végrehajtanánk, hogy az `Edit` szerkesztőmezőkben vannak-e beviendő adatok, és hogy a név szerkesztődoboz karakterrel kezdődik-e. Ehhez két lokális változót veszünk fel a `Fizetes` (integer) és a `Hiba` (logikai) változót. A `Fizetes` integer változó konvertálását ellenőrizzük, ha hiba történt, ugorjunk ki az új adat bevitelének ciklusából. Amennyibe a két `Edit` szerkesztőmezőből bevitt adat helyes, fűzzük hozzá ezt a tábla már létező soraihoz (`Table` objektum `Append` eljárása), és azonnal érvényesítsük ezt a valódi adatbázisunkban. A hibákat hibaüzenettel jelenítsük meg.

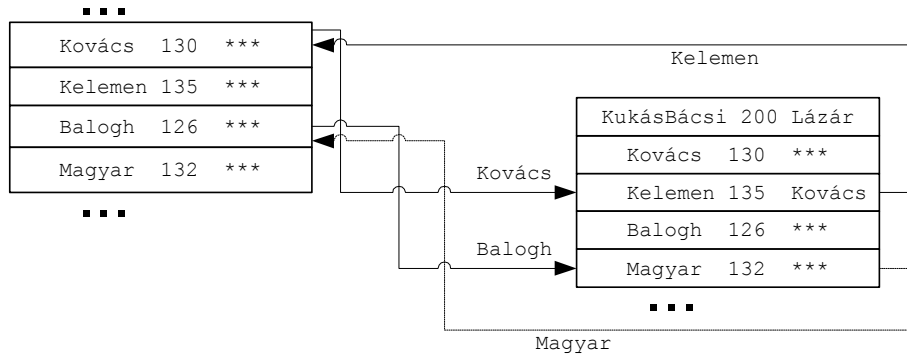
```

procedure TForm1.HozzafuzoGombClick(Sender: TObject);
label VEGE;
var
  Fizetes : Integer;
  Hiba     : Boolean;
begin
  Hiba := FALSE;                                // Hiba kezdőértéke
  if not ((NevSzerkesztoAblak.Text = '') or
    (FizetesSzerkesztoAblak.Text = ''))
  then
    If not (NevSzerkesztoAblak.Text[1] in ['0','1','2','3','4',
      '5','6','7','8','9'])
    then
      begin
        try
          // fizetes számmá alakítása
          Fizetes := StrToInt(FizetesSzerkesztoAblak.Text);
        except
          on EConvertError do                // konvertálási hiba
            begin
              MessageDlg('A Fizetés érték hibás!', mtError, [mbOK], 0);
              FizetesSzerkesztoAblak.Clear;
              FizetesSzerkesztoAblak.SetFocus;
              Hiba := TRUE;
            end;
          end;
        if Hiba                // ha hiba történt a konvertálásnál ugorjunk ki
        then
          goto VEGE;
        AdatTabla.Append; // fűzzük hozzá az adattáblához
        AdatTabla['nev'] := NevSzerkesztoAblak.text;
        AdatTabla['fizetes'] := Fizetes;
        AdatTabla['partner'] := '***';
        AdatTabla.Post;
        Showmessage('A(z) '+AdatTabla['nev']+
          ' a '+IntToStr(AdatTabla.RecordCount)+'-ik');
        NevSzerkesztoAblak.Clear;
        FizetesSzerkesztoAblak.Clear;
        NevSzerkesztoAblak.SetFocus;
      end
    else
      begin
        MessageDlg('Hibás a dolgozónév!', mtError, [mbOK], 0);
        NevSzerkesztoAblak.Clear;
        NevSzerkesztoAblak.SetFocus;
      end
    else
      MessageDlg('Érvénytelen az adatbevitel, ha valamelyik adatmező üres!',
        mtError, [mbOK], 0);
  VEGE:
  end;

```

A partner megkeresése a `Keresogomb` `OnClick` eseményénél történik. A feladat megoldásához két egymásbaágyazott ciklus szükséges. A belső ciklust egy önálló `Keres` nevű paraméterezett logikai függvénybe tettük. A keresés lényege:

Elindulunk az adattábla első rekordjától, ha még nincs partnere, akkor hívjuk a `Keres` függvényt, amely szintén előről végignézi az adattábla minden rekordját, amíg olyan sort nem talál önmagán kívül, amelynek partner mezejében még nincs senkinek sem a neve, és a fizetése a partnerkereső személy fizetésének 0,9 és 1,1 tartományába esik. Ha megtalálta, a `talalt` logikai változó legyen `True`, tegyük az adattáblát szerkeszthető (`Edit`) módba, és írjuk be a megtalált sorba a paraméterként átvett, hívó ciklusban aktuális nevet, és azonnal érvényesítsük (`Post`) a változást az eredeti adatbázisban is.



Például az első rekordot már megvizsgáltuk, most a következő vizsgálandó rekord legyen a Kovács, hívjuk a `Keres` függvényt, Kukás Bácsinak már van partnere, ezért Őt nem vizsgáljuk, önmagával nem vizsgáljuk, következő a Kelemen, aki megfelel a feltételnek, ezért a Kelemen `partner` mezejébe beírjuk a Kovács nevet. Kilépünk a `Keres` függvényből, és beírjuk Kovács `partner` mezejébe a Kelemen nevét, és így tovább.

A `Keres` függvény bemenő paraméterei : SajátNév, SajátFizetés,  
kimenő paraméterei : PartnerNév, PartnerSorszám.

```
procedure TForm1.KeresoGombClick(Sender: TObject);
var
  SajatSorszam    : Integer;
  SajatNev        : String;
  SajatFizetes    : Integer;
  PartnerSorszam  : Integer;
  PartnerNev      : String;

Function Keres(SajatNev      : String;
               SajatFizetes  : Integer;
               var PartnerNev : String;
               var PartnerSorszam : Integer): Boolean;

label VEGE;
var
  talalt : Boolean;
begin
  talalt := FALSE;
  AdatTabla.First;
  While not AdatTabla.EOF do
```

```

begin
  If (AdatTabla['nev'] <> SajatNev) and
    (AdatTabla['partner'] = '****') and
    (AdatTabla['fizetes'] >= 0.9*SajatFizetes) and
    (AdatTabla['fizetes'] <= 1.1*SajatFizetes)
  then
    begin
      talalt := TRUE;
      AdatTabla.Edit;
      AdatTabla['partner'] := SajatNev;
      AdatTabla.Post;
      PartnerNev := AdatTabla['nev']; // belső név megjegyzése
      PartnerSorszam := AdatTabla.RecNo; // belső sorszám megjegyz.
      // ShowMessage('>>Keres <<'+SajatNev+' '
      // +IntToStr(SajatFizetes));
      goto VEGE; // kiugrás
    end;
    AdatTabla.Next;
  end;
VEGE:
  Keres := talalt;
end;

begin { A TForm1.Button4Click [a Keres gomb] blokkja }
  AdatTabla.First;
  While not AdatTabla.EOF do
    begin
      if AdatTabla['partner'] = '****'
      then
        begin
          SajatSorszam := AdatTabla.RecNo; //Külső rekordszám megjegyzése
          SajatNev := AdatTabla['nev']; //Külső név megjegyzése
          SajatFizetes := AdatTabla['fizetes']; //Külső fizetés megjegyzése
          if Keres(SajatNev, SajatFizetes, //Keres függvény hívása
            PartnerNev, PartnerSorszam)
          then
            begin
              //A külső vizsgálatot indító rekordra mutasson az aktuális kurzor,
              //visszatalálás
              AdatTabla.MoveBy(SajatSorszam - PartnerSorszam);
              AdatTabla.Edit;
              AdatTabla['partner'] := PartnerNev; // Partner beírása
              AdatTabla.Post;
              // ShowMessage(' >>Visszatért!<<' + PartnerNev);
            end
          else
            // Ha nem talált megfelelőt, menjen tovább
          end;
          AdatTabla.Next;
        end;
      AdatTabla.First;
    end;
  end.

```

## Példák ADO lokális adatbáziskezelésre (Access)

A Microsoft Access adatbázis-kezelő programja minden olyan számítógépen megtalálható, ahol az Office programcsomag Professional változatát telepítették. Segítségével minden különösebb előtanulmány nélkül, némi próbálkozás után létrehozhatunk adatbázist és táblát.

Készítsünk például az Access-ben egy adatbázist, és ebben egy két-oszlopos táblát, amelynek oszlopnevei: *nev* (string), *fizetes* (integer). A tábla neve *Tábla1.mdb* lesz.

Az ADO komponens-paletta a 12.21. ábrán látható.



12.21. ábra Az ADO komponens-paletta

A Delphi ADO komponensei segítségével kérdezzük le (Tábla megjelenítés gomb) a létrehozott táblánkat, válogassuk ki azokat (Lekérdezés gomb), akiknek fizetése > 120. Vegyünk fel új rekordokat a táblába (Hozzáadás gomb, és Edit1, Edit2 szerkesztődoboz). Legyen a Formon egy Memo komponens, amelyikbe SQL utasításokat írhatunk, és ezt a DDL és DML utasítások végrehajtása feliratú gombbal a program hajtja végre.

A feladat megoldásának Formja a 12.22. ábrán látható (60\_ADO\_McAccess\_memo\_SQL+). Ehhez szükségesek a következő komponensek:



Az ADOConnection. Beállítandó tulajdonságai: *ConnectionString*. Az előbukkanó panel Build gombjára kattintva a kapcsolat felépíthető. A Data Link Properties panel Provider lapján válasszuk a Microsoft Jet 4.0 OLE DB Provider kapcsolatot. A Next gombra kattintva a Connection lap jelenik meg. Itt állítsuk be a létrehozott .mdb adatbázisunk elérési útját. Ha nem akarunk mást, fogadjuk el a felajánlott felhasználót és jelszót. Teszteljük a kapcsolatot a Test connection gombbal. Ha a kapcsolat felépíthető, akkor léphetünk tovább.



Az ADODataSet. Beállítandó tulajdonságai: *Connection* - ADOConnection1. Mivel ezt a komponenst mind a tábla megjelenítésre, mind egy egyszerű SELECT lekérdezésre használjuk, ezért más tulajdonságot most nem állítunk be.



Az ADOCommand. (Olyan SQL utasítások végrehajtására, amelyek nem adnak vissza sorokat. Végrehajtó eljárása Execution.). Beállítandó tulajdonságai: *Connection* - ADOConnection1.



A DataSource komponens, amely mindig szükséges, ha valamit meg akarunk jeleníteni. Beállítandó tulajdonságai: *DataSet* - ADODataSet1.

DBGrid, és DBNavigator komponens:

Beállítandó tulajdonságai: *DataSource* - DataSource1.

nev	fizetes
Kelemen	100
Kalap	120
Tódor	111
Lázár	115
Molnár	121
Zslya	200
Salya	444
Petya	222
Vasorrúbába	666
Banya	555

12.22. ábra Az ADO MsAcces feladat Formja

Jelenítsük meg a létrehozott táblát. Ehhez meg kell írunk a programot a Tábla megjelenítés gomb `OnClick` eseményében.

```
procedure TForm1.MegjelenitesClick(Sender: TObject);
begin
    // tábla megjelenítése
    ADODataSet1.Active := False;
    ADODataSet1.CommandType:= cmdTable; // CommandType tulajdonság
    ADODataSet1.CommandText:='Táblal'; // CommandText tulajdonság
    ADODataSet1.Active:= TRUE;
end;
```

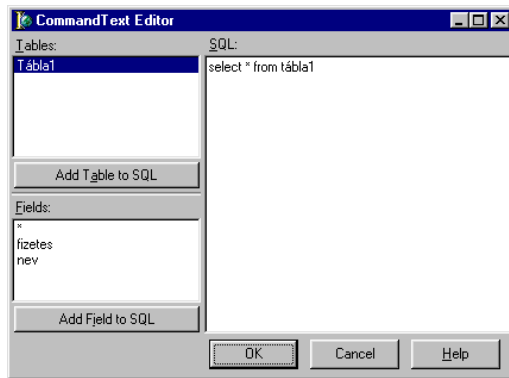
Zárjuk be az adathalmazt. Állítsuk be az `ADODataSet` objektum tulajdonságait. Legyen a tábla megjelenítésénél a `CommandType:= cmdTable`, és a `CommandText` tulajdonság egyszerűen csak a tábla neve `Táblal`. Mindezeket a tulajdonságokat az Objektum Felügyelőben is beállíthatjuk. De mivel egy programon belül kétféle adatforrással dolgozunk (tábla és lekérdezés), célszerűbb programból beállítani ezeket a tulajdonságokat. Ha a gombra kattintunk az Accessben létrehozott táblánk megjelenik. Ezután már feltölthetjük akárhány adattal Delphiből. (Persze magát az adatbázist Accessben kell létrehozni.)

A `Lekérdezés` gombra kattintva `SELECT` utasítások hajthatók végre. A fejlesztési időben a `Command Text Editor` segítségével írjuk be a `SELECT` utasítást (12.23. ábra). Ennek a lekérdezésnek az eredménye az `ADODataset` megnyitásakor megjelenik a rácsban.

Írhatunk a `CommandText` tulajdonságba a programból is `SELECT` utasításokat. Ezt írtuk meg a `Lekérdezés` gomb `OnClick` eseményébe. Először zárjuk be az adathalmazt, állítsuk be a `CommandType` és `CommandText` tulajdonságot, majd nyissuk meg az `ADODataset` (adathalmazt) objektumot.

```
procedure TForm1.LekerdezesClick(Sender: TObject);
begin
    // Select utasítás végrehajtása
    ADODataSet1.Active:= False;
    ADODataSet1.CommandType := cmdText;
    ADODataSet1.CommandText:= 'SELECT * from Táblal Where fizetes > 120';
```

```
ADODataset1.Active := True;
end;
```



12.23. ábra A Command Text Editor

A tábla feltöltését láthatjuk a következő Hozzáadás gombra kattintással. Az adatbevitel az Insert SQL utasítással történik. Az értékek bevitelét *paraméterátadással* valósítjuk meg.

Állítsuk be a paramétereket az ADOCommand objektum Parameters tulajdonságának AddParameter eljárása segítségével. A két paraméter a nev és fizetes, hiszen ez minden egyes sornál várhatóan különböző értékű, és futásidőben a felhasználó adja meg. A paramétereknek be kell állítani a Name tulajdonságát, DataType (adattípus) tulajdonságát, valamint a paraméter Direction (irány) tulajdonságát, amely lehet bemenő, kimenő és bekimenő együtt. Jelen eljárás a programból definiálja a paramétereket, de ugyanez megvalósítható az Objektum Felügyelő Parameters tulajdonság Paraméter Szerkesztőjében is. Ha a paramétereket már definiáltuk, akkor az Edit szerkesztődobozokból már értéket vehetnek fel. A CommandType tulajdonság legyen cmdText, CommandType tulajdonságába pedig írjuk be a paraméterezett értékeket.

```
procedure TForm1.HozzaadasClick(Sender: TObject);
begin
    ADOCommand1.CommandType := cmdText;
    // paraméterek beállítása az ADOCommand komponens segítségével
    with ADOCommand1.Parameters.AddParameter do
    begin
        Name := 'nev';
        DataType := ftString;
        Direction := pdInputOutput;
        //Value := 'Marosi';
    end;
    with ADOCommand1.Parameters.AddParameter do
    begin
        Name := 'szam';
        DataType := ftInteger;
        Direction := pdInputOutput;
        //Value := 111;
    end;
    // a nev paraméter értéke
    ADOCommand1.Parameters.ParamValues['nev'] := Edit1.Text
```

```
//szam (fizetes) paraméter értékét az Edit2 syerkesztőmezőből olvassa ki a
//program. Alapértelmezett konverzióval
ADOCommand1.Parameters.ParamValues['szam']:= Edit2.Text;

// A Paraméterezett lekérdezés
ADOCommand1.CommandText:= 'INSERT INTO Táblal VALUES(:nev, :szam)';

//Paraméter nélkül, de Edit boxokon keresztül új rekord felvétele
//ADOCommand1.CommandText:= 'INSERT INTO Táblal
VALUES(''+Edit1.Text+'',''+ Edit2.Text+'')';

ADOCommand1.Execute; // ADOCommand végrehajtása

Edit1.Clear;
Edit2.Clear;
Edit1.SetFocus;
end;
```

A Hozzáadás gombbal beszúrhatunk új sorokat a táblába. Ez az ADODataSet objektum Execute eljárásának aktivizálásával hajtódik végre. Hogy ez valóban végrehajtódott, kérdezzük le a táblát, azaz kattintsunk a Tábla megjelenítés gombra.

A DDL és DML parancs végrehajtása feliratú gombra kattintással a Memo komponensbe írt DDL és DML utasítások hajthatók végre az ADODataSet objektum Execute eljárásával. Mivel ezek az SQL utasítások nem adnak vissza sorokat, az eredmény csak a tábla ismételt megjelenítésével láthatjuk.

```
procedure TForm1.DDLesDMLClick(Sender: TObject);
begin
  ADOCommand1.CommandType := cmdText;
  ADOCommand1.CommandText:= Memo1.Text;
  //ADOCommand1.CommandText:= Memo1.Lines[0];
  ADOCommand1.Execute;

  // insert into táblal values('Petya', 222)
  // Insert into táblal values('VasorrúBába', 666)
  // Alter table táblal add column partner char(20) -- új oszlop beszúrás
  // Alter table táblal drop column partner -- oszlop eldobás
  // Update táblal set fiz =444 where nev := 'Satya' // módopsítás
  // C:\AA_Munkak\DELPHI\Laborok\60_ADO_McAccess_memo_SQL
  // C:\Dokumentumok

end;
```

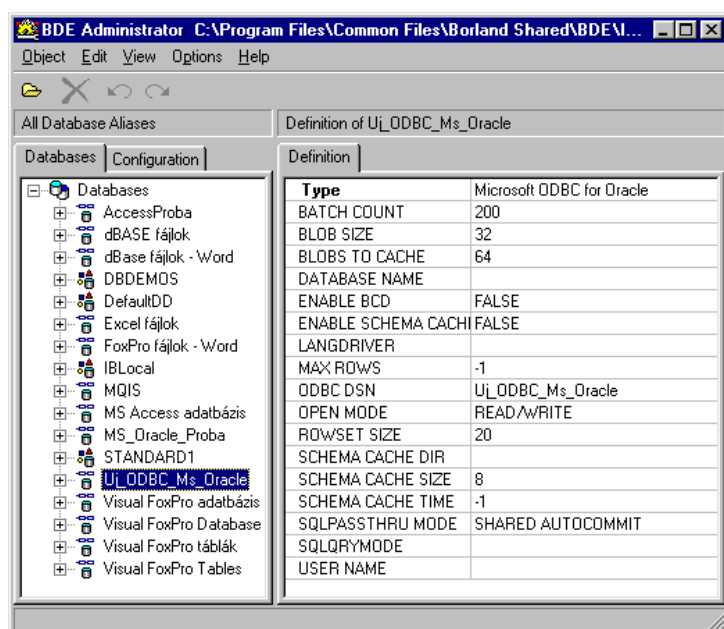
A programrészben megjegyzésben láthatók azok az SQL utasítások, amelyek végrehajthatók Memoból is az Execute eljárással. (Insert, Alter, Update utasítások)

## 13 FEJEZET

# Oracle adatbázisok használata

## ODBC – Oracle

Az ODBC-Oracle kapcsolat létrehozása a már ismert módon jön létre. Tehát az ODBC adminisztrátor (vezérlőpult önálló vagy BDE adminisztrátor Object menü almenüje) segítségével hozzáadjuk az új aliaszt. Válasszuk a Microsoft ODBC for Oracle drivert és adjunk egy új adatbázis alias nevet nevet (Uj\_ODBC\_Ms\_Oracle).



13.1. ábra Az ODBC-Oracle kapcsolat database jellemzői.

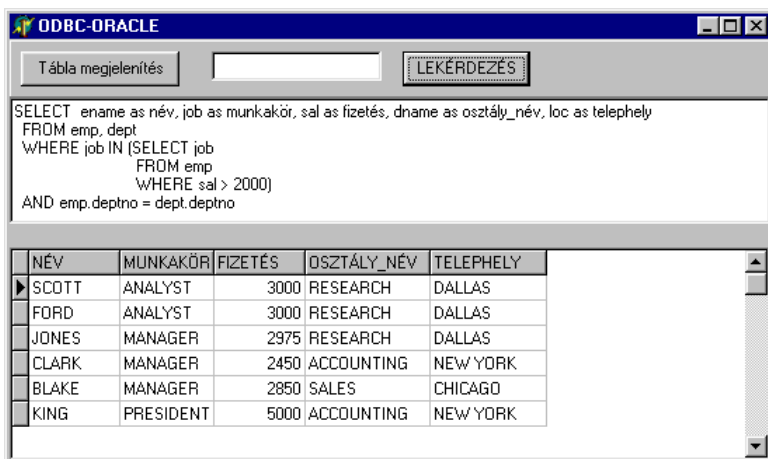
Miután a kapcsolat, az adatbázis létrejött (13.1. ábra), írhatunk Delphi felhasználói felületet az Oracle adatbázis kezeléséhez. A példa megmutatja a kapcsolat létrehozását, és mivel az ODBC driveren keresztüli kapcsolat a BDE-n keresztül történik, ezért az adatbázis-kezelésben is lényegében azonos módon az eddig ismertetett alavető komponensek használatát.

### Példa

A példánkban felépítjük az adatbázis elérést. Listázzuk az általunk ismert `scott` felhasználó tábláit, és írjunk lekérdezéseket. A feladat megoldása a 70\_Odbc1 könyvtárban található, amelynek futási formja a 13.2. ában látható. Legyen a Formon egy `Database`, egy `Table`, és egy `Query` komponens a BDE komponens-palettáról, és egy `DataSource` komponens a Data Access komponens-palettáról. (Ezek a komponensek futási időben nem látszanak.) Kapcsoljuk össze a komponenseket a következőképpen:

A `Database1` objektum `AliasName` tulajdonsága legyen a gördülő mezőből kiválasztva léterhozott ODBC meghajtóhoz választott alias név, az `Uj_ODBC_Ms_Oracle`.

A `DatabaseName` tulajdonsága legyen egyszerűen csak mi általunk beírt `uj`.



13.2. ábra Az ODBC-ORACLE feladat futási formja

A `Table1` objektum `DatabaseName` tulajdonsága legyen már a gördülő listából választható új, `TableName` tulajdonságnál válasszunk a gördülő listából egy létező `scott` tulajdonában levő táblát, például `emp`, vagy `dept` táblát.

A `Query1` komponens `DatabaseName` tulajdonsága szintén legyen új az SQL tulajdonságba most ne írjunk semmit, mert a felhasználó futási időben adja meg a lekérdezéseket.

A `DataSource1` objektum `DataSet` tulajdonsága legyen `Table1`, és a `Table1` objektum `Active` tulajdonságát állítsuk `True` értékre, akkor az adattábla megjelenik a rácsban.

Legyen a Formon egy `Panel`, és ezen két gomb, valamint egy `Edit1` szerkesztőmező. Alatta legyen a `Splitter`, és ez alatt pedig a `DBGrid` komponens.

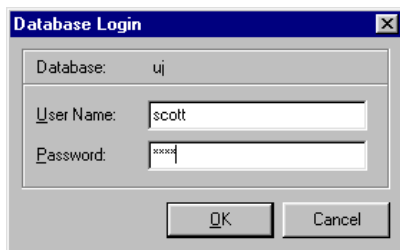
Írjuk meg a `Tábla megjelenítés` gomb `OnClick` eseményéhez tartozó programot, amelyben az `Edit1` szerkesztőmezőbe helyezzük a fókuszot. A tábla adatforrást először lezárjuk, ezután rendeljük a `Table1` objektum `TableName` tulajdonságához a felhasználó által megadott `Edit1` szerkesztőmezőbe beírt táblanevet, azonban csak akkor, ha az `Edit1` szerkesztőmező nem üres, különben hibakezelés és hibaüzenet jelenjen meg. Ezután nyissuk meg a táblát.

```
procedure TForm1.MegjelenitesClick(Sender: TObject);
begin
    Edit1.SetFocus;
    Table1.Active := False;
    Try
        If Edit1.Text <> ''
        then
            begin
                Table1.TableName := Edit1.Text;
                Table1.Open;
            end;
        except
            ShowMessage('Nincs megegyeztetendő táblanév');
        end;
    end;
end;
```

A `LEKÉRDEZÉS` gombra kattintva írjuk meg az `OnClick` eseményhez tartozó utasításokat. Mivel ezzel a gombbal csak `SELECT` utasításokat tudunk végrehajtani, ezért a `Query1` objektum `Open` eljárását használjuk. A lekérdezés eredményét azonban látni is szeretnénk, ezért a megjelenítést irányítsuk a `Query1`-re (`DataSource1.DataSet := Query1`). A teljes eljárás a következő:

```
procedure TForm1.LEKERDEZClick(Sender: TObject);
begin
    Query1.Close;
    Query1.SQL.Clear;
    DataSource1.DataSet := Query1;
    Query1.SQL.Assign(Memo1.Lines);
    Query1.Open;
end;
```

Indításkor megjelenik a jelszóbekérő ablak, amely a 13.3. ábrán látható. A helyes jelszó után a program működik. Egy-egy táblát képes megjeleníteni, és egy- vagy többtáblás lekérdezéseket végrehajtani, amint azt a futási képen is láthatjuk (13.2. ábra).



13.3. ábra A futáskor megjelenő jelszót bekérő ablak

## Oracle adatbázis-kezelés natív driverrel

### Példa (71\_Oracle\_nativ)

Adatbázis táblájának megjelenítése natív driverrel (13.4. ábra).

Készítsünk egy olyan alkalmazást, amely az Oracle adatbázis egy táblájának adatait jeleníti meg egy adatrácsban, egy DBGrid komponensben a Table komponens segítségével. A kapcsolatot az Oracle natív drivere segítségével vegyük fel.

Készítsünk egy új alkalmazást a Delphi File/New/Application menüpontjával.

Helyezzünk a Formra egy Database komponenset. A DriverName tulajdonságát állítsuk ORACLE-re. Az Oracle drivere ettől eltérő nevű is lehet, ellenőrizni a Vezérlőpult/BDE Administrator/Configuration/Drivers/Native elérési úton tudjuk. A Database komponens DatabaseName tulajdonságában kell megadnunk az adatbázis nevét. Ennek az értéknek nem kell megegyeznie az Oracle adatbázisunk nevével, csak arra szolgál, hogy felépítse a kapcsolatot a Database és az adatkészlet (Query, Table) komponens között. Állítsuk tehát a DatabaseName tulajdonságot MyOracle-re. A kapcsolat felvételéhez állítsuk a Connected tulajdonságot True érték-re.

13.4. ábra Oracle adtbázis elérése natív driverrel

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
1000	SMITH	CLERK	7902	1980. 12. 17.	
7499	ALLEN	SALESMAN	7698	1981. 02. 20.	
7521	WARD	SALESMAN	7698	1981. 02. 22.	
7566	JONES Jr.	MANAGER	7839	1981. 04. 02.	
7654	MARTIN	SALESMAN	7698	1981. 09. 28.	
7698	BLAKE	MANAGER	7839	1981. 05. 01.	
7751	MELOS	CLERK	7902	1998. 01. 01.	
7777	MULA	SALESMAN	7698	1999. 02. 01.	
7782	CLARK	MANAGER	7839	1981. 06. 09.	
7788	SCOTT	ANALYST	7566	1987. 04. 19.	
7839	KING	PRESIDEN		1981. 11. 17.	
7876	ADAMS	CLERK	7788	1987. 05. 23.	
7900	JAMES	CLERK	7698	1981. 12. 03.	
7902	FORD	ANALYST	7566	1981. 12. 03.	
7934	MILLER	CLERK	7782	1982. 01. 23.	

Rakjunk fel egy Table komponenset, ami az adatforrásunkat szolgáltatja. DatabaseName tulajdonságát állítsuk a Database komponens DatabaseName tulajdonságának megfelelő értékre, ekkor a TableName értékét az Object Inspectorban egy legördülő listából választhatjuk ki. Válasszuk az EMP táblát.

Az adatkészlet megjelenítéséhez szükségünk van egy adatforrás (DataSource) és egy DBGrid komponensre is. Tegyük láthatóvá az adatkészletet: állítsuk a DataSource1 nevű adatforrás objektum DataSet tulajdonságát Table1-re, valamint a DBGrid objektum DataSource tulajdonságát DataSource1-re.

Az adatok nem jelennek meg azonnal az adatrácsban. Miért? Ehhez először meg kell nyitnunk az adatforrást. A Table1 komponensnél ez az Active tulajdonság True-ra állításával, vagy az Open eljárás hívásával történik. Ekkor az adatok megjelennek az adatrácsban.

Az adatrács adatai szerkeszthetők, ezt háromféleképpen állíthatjuk át:

- az adatkészlet zárolásával,
- az adatforrás módosító tevékenységének tiltásával, vagy
- az adatfüggő komponens módosítási képességének letiltásával.

Esetünkben: a Table1 komponens ReadOnly tulajdonságának True értékre állításával, a DataSource objektum AutoEdit tulajdonságának False értékre állításával, vagy a DBGrid objektum ReadOnly jellemzőjének True értékre állításával.

Kényelmesebbé teszi a rekordok kezelését a DBNavigator komponens, bár minden rajta lévő gomb funkcióját el tudjuk érni csupán a billentyűzet kezelésével: a pozicionálást a <↑> és <↓> billentyűkkel, a rekordkészlet elejére és végére ugrást a <Ctrl+Home> és <Ctrl+End>-del. Beszúrni az <Insert>, törölni a <Ctrl+Del> billentyűkkel tudunk.

A DBNavigator komponens VisibleButtons tulajdonságánál be tudjuk állítani, hogy mely gombok legyenek láthatóak.

Hogyan lehet álneveket készíteni programból? Az alábbi példán keresztül ezt mutatjuk meg.

### Példa (72\_alias\_nativ)

Álnevek (alias nevek) készítése és törlése natív és ODBC driver-hez.

Készítsünk álnévet (alias) az Oracle adatbázishoz a natív és ODBC driverre futásidőben. A felhasználónak engedélyezzük az álnévek törlését is: Legyen lehetősége az álnév nevének beírására a létrehozáskor, és törléskor a felhasználó egy listából válassza ki a törölni kívánt álnévet.

Készítsünk egy új alkalmazást a Delphi File/New/Application menüpontjával.

A program megírásához szükségünk lesz, két Edit szerkesztődoboz, egy ListBox, egy ComboBox és két Button (normál gomb) komponensre. Persze helyezzünk fel a Formra Label (cimke) komponenseket is.

A ComboBox objektumban választjuk ki majd a használni kívánt driver nevét.

A ListBox szolgál a már létező álnévek megjelenítésére: tartalmát frissítenünk kell a Form OnCreate eljárásában, mégpedig új álnév létrehozásakor és törléskor. A frissítés azt jelenti, hogy lekérjük a ListBox objektum Items tulajdonságába az álnéveket.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    // beírjuk az aliasneveket a ListBox-ba
    Session.GetAliasNames(ListBox1.Items);
end;
```

Az új ODBC álnév létrehozásához szükséges az álnév neve, a natív aliashoz pedig ezen felül az Oracle adatbázis neve. Mindkettőt egy-egy Edit szerkesztőmezőben kérjük be, és a ComboBox tartalmától függően hozzáadjuk vagy nem a paraméterekhez. Az új alias létrehozására a Session objektum AddAlias eljárása szolgál. Ha ki van töltve mindkét Edit szerkesztőmező, akkor összeállítjuk az AddAlias paramétereit, majd hibaellenőrzéssel meghívjuk az alias előállító függvényt.

Mindez az Add feliratú gomb OnClick eljárásában történik:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    ParamList: TStringList;
begin
    if (Edit1.Text<> '') and (Edit2.Text<> '') then
    begin
        ParamList := TStringList.Create;
        with ParamList do
        begin
            Add('USER NAME=system');
            if ComboBox1.ItemIndex=0
                // a SERVER NAME paraméter csak natív driver álnéve esetén kell:
            then Add('SERVER NAME=' + Edit2.Text);
        end;
        Try
            // megpróbáljuk létrehozni az alias:
            Session.AddAlias(Edit1.Text, ComboBox1.Items[ComboBox1.ItemIndex],
                ParamList);
        Finally
            ParamList.Free;
            // a ListBox-ban frissítjük az aliasneveket:
            Session.GetAliasNames(ListBox1.Items);
            ShowMessage('A(z) ' + Edit1.Text + ' alias létrejött.');
```

Törléskor a felhasználó a ListBox legördülő listájából tudja kiválasztani a törlendő álnévet.

A törlés nem íródik be a BDE konfigurációjába, csak a memóriában tárolódik. A konfigurációba való mentéshez a törlés után meg kellene hívnunk a Session objektum SaveConfigFile eljárását is. Ezt most nem tettük meg.

A törlés a DeleteAlias eljárással történik a felhasználó megerősítése után, a Del (Törlés) feliratú Button2 gomb megnyomása után:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    if MessageDlg('Biztos, hogy törli a(z)
        'ListBox1.Items[ListBox1.ItemIndex] + ' álnévet?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes
    then
        begin
            Try
```

```

// csak ebből a Session-ből töröljük az álnevet. A konfigurációból
// való törléshez ezt a SaveConfigFile-lal menteni kellene:
Session.DeleteAlias(ListBox1.Items[ListBox1.ItemIndex]);
except
    MessageDlg('Az álnév törlése nem sikerült.', mtError, [mbOk], 0)
end;
// frissítjük az aliasneveket:
Session.GetAliasNames(ListBox1.Items);
end;
end;

```

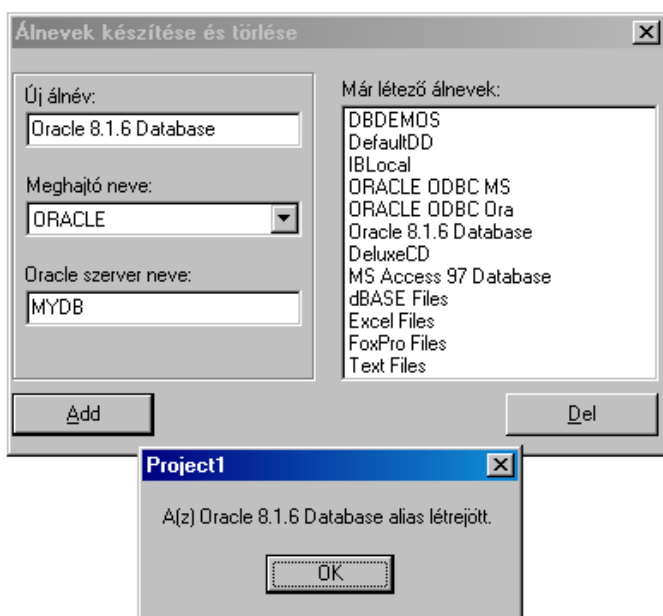
A ComboBox objektum OnChange eseményéhez tartozó eljárás:

```

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
    // ODBC driver esetén nem kell SERVER NAME paramétert megadnunk, ezért
    // letiltjuk a beadására szolgáló Edit szerkesztőmezőt:
    if ComboBox1.ItemIndex=1
    then Edit2.Enabled:=False
    else Edit2.Enabled:=True;
end;

```

A futás eredménye a 13.5. ábrán látható.



13.5. ábra Aliast létrehozó program futása

### Példa (73\_módosít\_natív)

Az UpdateSQL használata Query komponenssel.

Készítsünk egy olyan alkalmazást, amellyel egy Query komponens adatkészletét tudjuk módosítani egy UpdateSQL komponens segítségével.

A felhasználó egyenként tudja módosítani az aktuális rekord adatait, törölheti azt, vagy új rekordot vehet fel. Módosítást a Szerkesztés felíratú gombbal tudja engedélyezni.

Készítsünk egy új alkalmazást a Delphi File/New/Application menüpontjával.

Rakjunk a Formra egy Database, Query, UpdateSQL komponenst a BDE komponens-palettáról, DataSource, és az adatáramláshoz DBNavigator komponenseket a Data Control komponens-palettáról. A Query által szolgáltatott adatkészlet rekordjainak mezőit DBEdit-mezőszerkesztőkben jelenítsük meg.

Kapcsoljuk össze a komponenseket: állítsuk a Query1 objektum UpdateObject tulajdonságát az UpdateSQL objektum nevére (UpdateSQL1), a DBEdit-szerkesztő objektumok DataSource tulajdonságát DataSource1-re, DataField tulajdonságát pedig a megjeleníteni kívánt mező nevére. A többiek összekapcsolása a szokásos módon zajlik.

Írjuk meg a Query objektum SQL tulajdonságába kerülő lekérdező utasítást. Legyen ez például az alábbi:

```

SELECT empno, ename, job, sal, deptno
FROM Scott.emp

```

Ezután generáljunk tartalmat az UpdateSQL-be. A Formon szükség van még, az új rekord beszűrő, törlő, változtatásokat alkalmazó és visszavonó Button (gomb) komponensekre. Ezek OnClick eljárásukban hívják a Query1 objektum Insert, Delete, ApplyUpdate és CancelUpdate eljárásait.

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    Query1.ApplyUpdates;           // A változtatások mentése
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Query1.CancelUpdates;         // A változtatások visszavonása
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
    Query1.Insert;                // új rekord beszűrése
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
    Query1.Delete;                // rekord törlése
end;

```

Csak a szerkesztést engedélyező gomb OnClick eseménye más, mert itt a Query objektum CachedUpdates tulajdonságát állítjuk – ezzel engedélyezve a csak olvasható adatforrás módosításainak tárolását -, és állítsuk be a gomb Hint tulajdonságát azért, hogy tudjuk a cache-elés aktuális állapotát.

```

procedure TForm1.Button3Click(Sender: TObject);
begin
    if Query1.CachedUpdates
    then
        begin
            Query1.CachedUpdates:= False;
            Button3.Hint:='CachedUpdate = FALSE';
        end
    else
        begin
            Query1.CachedUpdates:=True;
            Button3.Hint:='CachedUpdate = TRUE';
        end;
end;

```

Figyeljük meg a DBNavigator-t, miként viselkedik nem cache-elt módosítások esetén: a beszűrés és módosítás funkciót engedélyezi, míg a törlést letiltja. A futtatás eredménye a 13.6. ábrán látható.

13.6. ábra UpdateSQL használata



kurzorhasználattal kell megnyitnunk (a `CursorLocation` tulajdonságot `clUseClient`-re kell állítani). Ellenkező esetben kivétel képződik.

A `Filter` tulajdonság használatával tudjuk szűrni adatkészletünket. A szűrés érvénybe léptetéséhez a `Filtered` tulajdonságot `True`-ra kell állítanunk.



Az `ADOTable` szolgál egy egyszerű adatbázis-tábla elérésére. Az `ADOTable` komponens közvetlen hozzáférést biztosít minden rekordhoz és mezőhöz a hozzárendelt adatbázis-táblában. Ugyanakkor jelentheti a tábla rekordjainak egy részhalmazát is.



Az `ADOQuery` komponenssel egy vagy több táblát érhetünk el az adatbázisban. Képes SQL utasítások (DQL, DDL és DML) és tárolt eljárások futtatására.

A `ADOQuery` objektum `SQL` tulajdonságában állíthatjuk be a futtatandó lekérdezés szövegét. Futtatása kétféleképpen történhet: az `ExecSQL` és az `Open` eljárások hívásával (utóbbival megegyező eredményt kapunk, ha az `Active` tulajdonságot `True` értékre állítjuk). Az `ExecSQL` és az `Open` eljárások között a különbség az, hogy csak az `Open` képes adatokat fogadni az adatbázisból. Ezért a sorokat (kurzort) vissza nem adó lekérdezések (`INSERT`, `DROP` stb.) futtatására az `ExecSQL`, míg az eredményhalmazt visszaadókhöz (`SELECT`) az `Open` eljárást használjuk.

A teljesítmény növelésének érdekében az alkalmazásnak ajánlatos előkészítenie egy lekérdezés futtatását. Megtehetjük ezt a `Prepared` tulajdonság `True` értékre állításával. Kis adathalmazzal rendelkező alkalmazásoknál nincs észrevehető időbeli különbség.

A Delphi nem ellenőrzi a `ADOQuery` objektumon keresztül futtatandó lekérdezések helyességét, mivel szintaktikájuk a különböző adatbázisoknál eltérő lehet. A visszakapott hibaüzenetek kódja és szövege ennek megfelelően specifikus.



Az `ADOStoredProc` akkor használatos, mikor a kliens-alkalmazásnak az adatbázis tárolt eljárását kell használnia. `ADOStoredProc` objektum `Parameters` tulajdonsága hordozza a tárolt eljárások visszatérési értékeit. A szerver implementációjától függően egy tárolt eljárás visszatérhet egy egyszerű értékhalmazzal, vagy egy, a lekérdezésekéhez hasonló eredményhalmazzal.



Az `ADOCommand` komponens továbbítja a parancsokat az adattároló felé, amit az ADO szolgáltatón keresztül érünk el. Egyszerre csak egy parancs futtatására alkalmas ez az objektum is, és az utasítást a `CommandText` tulajdonságban adjuk meg. Az `ADOCommand` objektum az adattárolóhoz való csatlakozásra (a `Connection` tulajdonságán keresztül) használhatja az `ADOConnection` objektumot is, vagy közvetlen módon is képes a csatlakozásra, ha a szükséges információkat a `ConnectionString` tulajdonságánál beállítjuk.

Ezt a komponenst legtöbbször DDL SQL parancsok vagy tárolt eljárások futtatására használjuk, amelyek nem adnak vissza eredményhalmazt. Ugyan képes sorokat visszaadó SQL utasítások végrehajtására is, de ekkor egy `ADODataset` alkalmazása is szükséges.

Mind az `ADOCommand` mind az `ADOQuery` esetében – mint bármilyen lekérdezésnél – lehetőségünk van paraméterezésre. A paraméterezés a BDE Query komponensének paraméterezési eljárásával megegyezik. Az `ADOCommand` objektum `CommandText` tulajdonságában határozzuk meg a végrehajtandó utasítás szövegét. Ez lehet SQL lekérdezés, táblanév, vagy egy futtatandó tárolt eljárás neve. A `CommandType` jellemző megadja, hogy a `CommandText` tulajdonságban megadott utasítás milyen jellegű. Például ha ott egy tábla nevét találjuk, akkor a `CommandType` értékét `cmdTable`, vagy `cmdTableDirect` értékre kell állítanunk.

Az `ADOCommand` objektum `CommandTimeout` tulajdonságával állíthatjuk be azt az időt (másodpercekben), ameddig alkalmazásunk kísérletezik a parancs végrehajtásával. Az alapérték 30 másodperc.

A parancs futtatása az objektum `Execute` eljárásával történik.

### Példa (61\_ADO\_Oracle\_tablamegjelenítés)

Adatbázis-tábla megjelenítése ADO komponensekkel. (13.8. ábra)

Nézzünk egy nagyon egyszerű alkalmazást, amely csatlakozik az ORACLE adatbázisunkhoz, valamint egy adatrácsban megjeleníti egy kiválasztott Oracle tábla sorait. Az alkalmazással képesek vagyunk az adattábla, az adatbázis adatainak módosítására.

Készítsünk egy új alkalmazást a Delphi `File/New/Application` menüpontjával.

Helyezzünk el a Formon egy `Panel`-t, erre egy `DBNavigator` komponenst, egy `ADOConnection`, `ADODataset` (`ADOTable`) komponenst az ADO komponens-palettáról, egy `DataSource` komponenst a Data Access komponens-palettáról, és egy `DBGrid` komponenst a Data Controls komponens-palettáról. Építsük fel a komponens `ConnectionString` tulajdonságát a fent leírt módon a `scott` felhasználónév és a `tiger` jelszó megadásával. Ellenőrizzük a kapcsolatot a `Test Connection` feliratú gomb megnyomásával.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7777	KELEMEN	ANALYST	2222	2001.09.26.	2222	1000	10
7499	ALLEN	SALESMAN	7698	1981.02.20.	1600	300	30
7521	WARD	SALESMAN	7698	1981.02.22.	1250	500	30
7566	JONES	MANAGER	7839	1981.04.02.	2975		20
7654	MARTIN	SALESMAN	7698	1981.09.28.	1250	1400	30
7698	BLAKE	MANAGER	7839	1981.05.01.	2850		30
7782	CLARK	MANAGER	7839	1981.06.09.	2450		10
7788	SCOTT	ANALYST	7566	1987.12.09.	3000		20
7839	KING	PRESIDENT		1981.11.17.	5000		10
7844	TURNER	SALESMAN	7698	1981.09.08.	1500	0	30
7876	ADAMS	CLERK	7788	1987.05.23.	1100		20
7900	JAMES	CLERK	7698	1981.12.03.	950		30
7902	FORD	ANALYST	7566	1981.12.03.	3000		20
7934	MILLER	CLERK	7782	1982.01.23.	1300		10

13.8. ábra ADO-Oracle táblamegjelenítés

Kapcsoljuk össze a komponenseket: Állítsuk be az `ADODataset1` nevű `ADODataset` komponens `Connection` tulajdonságát `ADOConnection1`-re (`ADOTable1` objektum `Connection` tulajdonságát `ADOConnection1`-re), a `DataSource1` objektum `DataSet` tulajdonságát `ADODataset1` vagy `ADOTable1`-re, és a `DBGrid`, és `DBNavigator` objektumok `DataSource` tulajdonságát `DataSource1`-re.

Helyezzünk a `Panelra` egy `Megjelenít` feliratú gombot, amely `OnClick` eseményében egy Oracle adattábla sorait jelentessük meg.

```
procedure TForm1.MegjelenitClick(Sender: TObject);
begin
    // ADODataset objektum használata esetén
    {
        ADODataset1.Close;
        ADODataset1.CommandType:=cmdTable;
        ADODataset1.CommandText:= 'EMP1';
        ADODataset1.Open;
    }

    //Mindkettő megoldja a feladatot

    //ADOTable használata esetén
    ADOTable1.Close;
    ADOTable1.TableName:= 'EMP1';
    ADOTable1.Open;
end;
```

A rácsban megjelenő adatok szerkeszthetők: módosíthatjuk egy mező értékét, ha az egérrel rákattintunk, vagy ha az a mező aktív, egyszerűen az érték begépelésével. Törölhetjük az egész rekordot a `<Ctrl+Del>` billentyűkombinációval, vagy új rekordot szúrhatunk be az `<Insert>` billentyű megnyomásával.

### Példa (62\_ADO\_Oracle\_tablalistazas)

Adatbázis tábláinak listázása ADO-val. (13.9. ábra)

Készítsünk egy olyan alkalmazást, amellyel a felhasználó képes új tábla létrehozására és törlésére, valamint képes listázni az ORACLE adatbázis tábláit, és módosítani a kiválasztott tábla adatait.

Helyezzünk a `Formra` egy `ADOConnection` komponenst, majd állítsuk be a `ConnectionString` tulajdonságát. A tábla létrehozását és törlését egy-egy `ADOCommand` objektumon segítségével hajtsuk végre. Állítsuk be `Connection` tulajdonságaikat `ADOConnection1`-re. A táblanevek listáját egy `ListBox`-ból válasszuk ki, és egy `ADOQuery` objektumon keresztül kérdezzük le a kiválasztott tábla adatait, amelyeket egy `DataSource` objektumon keresztül egy `DBGrid` komponens jelenítsen meg.

Első lépésben fel kell töltenünk a `ListBox` objektumot az adatbázis tábláinak neveivel. Ez az `ADOConnection` objektum `GetTableNames` eljárásával történik, amelynek két paramétere van. A eljárás első paramétere az eredmény visszaadására szolgál és `TString` (lista) típusú, a második paraméter arra vonatkozik, hogy a függvény visszaadja-e a rendszertáblák neveit is, ez logikai típusú, alapértelmezés `False` értékű. A lekérdezett táblák neveit (az első paraméter) vegye fel a `ListBox` objektum `Items` tulajdonsága, mert abba szeretnénk helyezni a táblaneveket, a második (bemenő paraméter) pedig a rendszertáblák lekérdezésére vonatkozik, és ez lehet igaz vagy hamis érték. Ezt az értéket híváskor egy `CheckBox` objektum `Checked` tulajdonságának megváltozása adja. Mivel a `ListBox` adatait többször is frissítenünk is kell a létrehozás és törlés miatt, hozzunk ehhez létre egy új eljárást `ListRefresh` néven.

A `ListRefresh` eljárás:

```

procedure TForm1.ListRefresh(SysTables: Boolean);
begin
    // lekérjük a táblák neveit
    ADOConnection1.GetTableNames(ListBox1.Items, SysTables);
end;

```

A jelölőnégyzet (CheckBox) megváltozásának eseményéhez a lista frissítését rendeljük. A ListRefresh eljárás paramétere egyben a GetTableNames eljárás bemenő paramétere is.

```

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    // (csak) ezzel a CheckBox-szal tudjuk a ListRefresh paraméterét változtatni.
    ListRefresh(CheckBox1.Checked); // Lehet True, vagy False
end;

```

A Form OnCreate eseményében történik a ListBox első feltöltése.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    // induláskor feltöltjük a ListBox-ot:
    ListRefresh(CheckBox1.Checked);
end;

```

Engedjük meg a felhasználónak, hogy ő írja be a létrehozandó, illetve törlendő tábla nevét. A létrehozandó új táblában egy *szam* (integer típusú), egy *nev* (VARCHAR2 típusú), és egy *kod* (VARCHAR2 típusú) oszlopot veszünk fel. Ezt úgy érjük el, hogy a Formra még egy Edit szerkesztődoboz komponenst is helyezünk. Ha a felhasználó változtatja a szerkesztőmező tartalmát, meg kell vizsgálnunk, hogy a tábla létezik-e vagy sem: ha létezik, akkor törölni engedjük, ha nem létezik, akkor a létrehozást engedélyezzük. Mindezt az Edit1 szerkesztődoboz OnChange eseményének bekövetkeztekor hajtjuk végre.

```

procedure TForm1.Edit1Change(Sender: TObject);
var
    letezik: Boolean;
    i:      Byte;
begin
    if Edit1.Text=''
    then ShowMessage('Kérem adja meg a táblanevet! ')
    else
        begin
            letezik:=False;
            // a ListBox-ban keressük a beírt nevet:
            for i:=0 to (ListBox1.Items.Count-1) do
                begin
                    if (ListBox1.Items[i]=Edit1.Text) or
                       (ListBox1.Items[i]=Uppercase(Edit1.Text))
                    then letezik:=True;
                end;
            if letezik
            then
                begin
                    Button1.Enabled:=False;
                    Button2.Enabled:=True;
                end
            else
                begin
                    Button1.Enabled:=True;
                    Button2.Enabled:=False;
                end;
            end;
        end;
end;

```

A Létrehozás gomb OnClick eseményének hatására végrehajtandó program:

```

procedure TForm1.LetrehozasClick(Sender: TObject);
begin
    // A "Létrehozás" gombra kattintva létrehozuk az Edit1 Text
    // tulajdonságával megegyező nevű táblát. Az egyszerűség kedvéért a

```

```
// táblastruktúra mindig ugyanaz:
ADOCommand1.CommandText:='CREATE TABLE '+ Edit1.Text +
    ' (szam INTEGER, nev VARCHAR2(20), kod VARCHAR2(10))';
ADOCommand1.Execute;

ListRefresh(CheckBox1.Checked);
ShowMessage('A "' + Edit1.Text +'" tábla létrehozása megtörtént.');
```

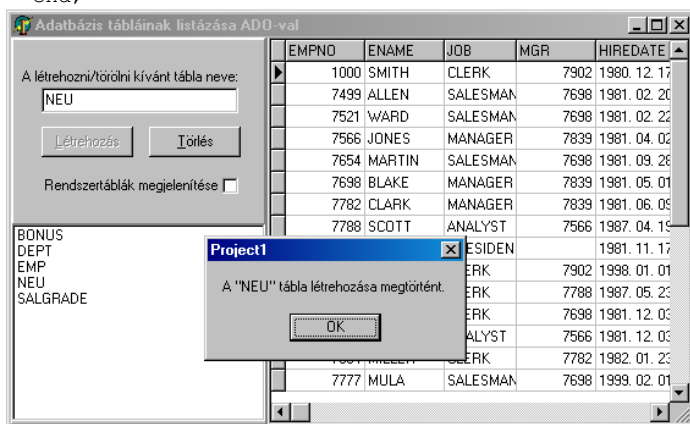
Button1.Enabled:=False;  
Button2.Enabled:=True;  
end;

A létrehozás és a beillesztés művelete történjen a Button1 **Létrehozás** nevű gomb megnyomásakor. A tábla mezői mindig ugyanazok legyenek, az ADOCommand objektum CommandText tulajdonságában adjuk meg futás közben, mégpedig sorszám legyen *szam* (integer típusú), a név (*nev*) max. 20 karakter, és kód (*kod*) max. 10 karakteres string. A tábla törlését a Törlés gomb **OnClick** eseményének bekövetkeztekor, az ADOCommand2 objektum **Execute** eljárásával érjük el. Előzőleg meg kell adnunk az ADOCommand objektumnak, hogy melyik táblát kívánjuk törölni: a tábla nevét az Edit szerkesztődobozból olvassuk ki. A törlést jelezzük egy üzenettel. A két gomb **OnClick** eljárása csak az ADOCommand objektum CommandText tulajdonságának szövegében, és a gombok engedélyezésében tér el. Törlés után (mivel az Edit szerkesztőmezőt nem töröljük, így a törölt tábla neve benne marad) a létrehozást engedélyezzük, létrehozás után pedig a törlést végző gombot engedélyezzük.

```
procedure TForm1.TorlesClick(Sender: TObject);
begin
    // A "Törlés" gombra kattintva töröljük az Edit1.Text
    // tulajdonságával megegyező nevű táblát
    ADOCommand2.CommandText:='DROP TABLE '+Edit1.Text;
    ADOCommand2.Execute;

    ListRefresh(CheckBox1.Checked);
    ShowMessage('A "' +Edit1.Text+'" tábla törlése megtörtént.');
```

Button1.Enabled:=True;  
Button2.Enabled:=False;  
end;



13.9. ábra Tábla létrehozása és törlése

Megvan tehát a tábla létrehozása, törlése, és a táblanevek listázása, ami hiányzik az a tábla adatainak megjelenítése. Ennél mi sem egyszerűbb, mert a ListBox komponens **OnClick** eseményében az ADOQuery1 komponens SQL tulajdonságának állításával, majd a lekérdezés futtatásával megjelenítjük a kiválasztott tábla adatait. A lekérdezés szövegéhez hozzáfűzzük a ListBox-ban kijelölt tábla nevét:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
    ADOQuery1.Close;
    ADOQuery1.SQL.Text:='SELECT * FROM '+ListBox1.Items[ListBox1.ItemIndex] ;
    ADOQuery1.Open;
end;
```

Gondoskodnunk kell még arról is, hogy a ListBox-ban megjelenő táblanevek az adatbázis tényleges adatait tükrözzék. Ha a CheckBox1 jelölőnégyzetre kattintunk, meg kell hívnunk a ListRefresh eljárást. A felhasználó szabadon módosíthatja a kiválasztott tábla adatait. A módosítások azonnal bekerülnek a táblába.

A program futási Formja a 13.9. ábrán látható.

### Példa (63\_ADO\_Oracle\_param)

Paraméteres lekérdezés használata az ADO – Oracle kapcsolatban

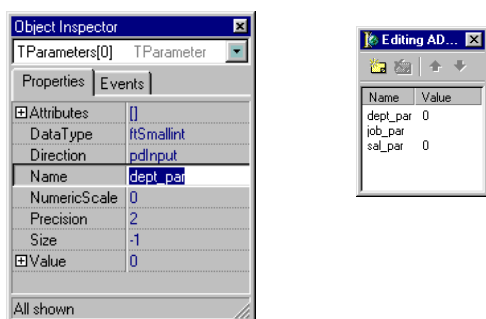
Írjunk egy olyan alkalmazást, amely a paraméteres lekérdezéssel válogat az adatbázis egyik táblájából. A paraméterek értékadásának szemléltetése céljából most három paraméterrel dolgozunk.

A felhasználónak az a kívánsága, hogy megtalálja adott részlegben, adott foglalkozású dolgozót, aki kevesebbet keres, mint egy bizonyos összeg. A három paraméter tehát a részleg (*dept\_par*), a foglalkozás (*ob\_par*) és a fizetés (*sal\_par*).

Készítsünk egy új alkalmazást. Helyezzünk el egy *ADOConnection* komponenst, *ADOQuery* komponenst, *DataSource* komponenst és egy *DBGrid* komponenst a Formon. Kapcsoljuk össze a komponenseket. Az adatok bekéréséhez szükségünk van egy gombra és három *Edit* szerkesztőmezőre is.

Írjuk meg a lekérdezés szövegét az *ADOQuery* objektum *SQL* tulajdonságába, a *Strig List Editor* segítségével:

```
SELECT ename, hiredate
FROM Scott.Emp
WHERE deptno = :dept_par
    and job = :job_par
    and sal < :sal_par // nem kell a végére a pontosvessző(; )!!!
```



13.10. ábra Paraméterszerkesztő

Ha ennek mentése után megnézzük a *Parameters* tulajdonságát, ott mindhárom paraméter megjelenik. Ha rákattintunk a paraméter nevére az Object Inspector-ban megjelenik a kitöltendő tulajdonságok listája, amint ezeket a 13.10. ábrán láthatjuk is. A paraméterek értékét futási időben adjuk meg, amit az alábbi programrészletben látható. Be kell állítanunk mindenképpen a paraméterek típusát és a méretet/pontosságot a string/numerikus adatok esetén (ha nem konvertáljuk őket a lekérdezésben - /,\*,% stb. - akkor az adatbázis mezőinek típusával egyezően). A paraméter jellege (*Direction*) alapértelmezetten bemenő (*pdInput*). A *Futtatás* gomb *OnClick* eljárásában olvassuk be az *Edit* szerkesztőmezők tartalmát, és ez az érték legyen *ADOQuery* objektum *Parameters* tulajdonság értéke. A paraméterek értékének állítása háromféle módon lehetséges. Két eset a *Parameters* objektum elérésében különbözik, egy pedig közvetlen módon a paraméter értékét módosítja:

```
procedure TForm1.FuttatasClick(Sender: TObject);
begin
    ADOQuery1.Close;
    try
        // A paraméterek értékét háromféleképpen adhatjuk meg:
        ADOQuery1.Parameters.ParamByName('dept_par').Value:=Edit1.Text;
        ADOQuery1.Parameters[1].Value:=Edit2.Text;
        ADOQuery1.Parameters.ParamValues['sal_par']:=Edit3.Text;
        ADOQuery1.Open;
    except
        MessageDlg('Nem megfelelő paraméterek, vagy'#13#10
            'nincs minden paraméter megadva.',mtError,[mbOk],0);
    end;
end;
```

Természetesen hibaellenőrzést, kivételkezelést is kell végeznünk a helytelen paraméterek kiszűrésére. Futtatáskor a paraméterek beírása után a *Futtatás* gombra kattintunk, s ekkor jelenik meg a jelszóbekérő ablak. A futási kép a 13.11. ábrán látható.

ENAME	HIREDATE
ALLEN	1981.02.20.
WARD	1981.02.22.
MARTIN	1981.09.28.
TURNER	1981.09.08.

13.11. ábra A paraméteres lekérdezés futási képe

**Példa (64\_ADO\_ORACLE\_2tabla)**

Készítsünk olyan alkalmazást, amellyel összehasonlítható az `ADODataset` objektum `SELECT` lekérdezésen alapuló, illetve táblaspecifikáción alapuló adatforrás-meghatározása.

- a felhasználónak lehetősége legyen lekérdezések beírására, amellyel meghatározzuk az adatforrást,
- egy másik lehetőség a megjelenített adattábla válogatására, rendezésére.

A két `ADODataset` objektummal visszaadott adatforrást két külön adatrácsban jelenítsük meg, hogy legyen lehetőség az eredményhalmazok összehasonlítására.

Helyezzünk a Formra egy `ADOConnection` komponenst a csatlakozáshoz, egy `ADODataset`, `DataSource` és egy `DBGrid` komponens-hármaszt a lekérdezés alapú és egy másik hármaszt a tábla alapú adatkészlet felhasználásához. Kapcsoljuk össze ezeket a komponenseket: Állítsuk be az `ADODataset1` és `ADODataset2` nevű komponensek `Connection` tulajdonságát `ADOConnection1`-re, a `DataSource` komponensek `DataSet` tulajdonságát a hozzájuk tartozó `ADODataset` nevének megfelelően, és a két `DBGrid` komponens `DataSource` tulajdonságát a megfelelő `DataSource` komponensre. Válasszuk el a két rács komponenst függőlegesen egy `Splitter` komponenssel.

Szükségünk lesz még egy `Edit` és egy `Button` komponensre a lekérdezés bekéréséhez, illetve futtatásához, valamint három `Edit` szerkesztőmezőre és egy gombra a tábla adathalmazának válogatásához és rendezéséhez.

Ellenőrizzük le a kapcsolat létrejöttét.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  try
    ADOConnection1.Open;
  except
    MessageDlg('Az Oracle-höz való csatlakozás nem történt meg.',
      mtWarning, [mbOk], 0);
  end;
end;
```

Ahhoz hogy az `ADODataset1` nevű objektumot lekérdezés adatforrásaként tudjuk használni, meg kell írunk a hozzá kapcsolódó SQL lekérdezést (a `CommandText` tulajdonságának értékeként). Az `ADODataset1` objektum `CommandType` jellemzőjének alapértéke `cmdText`, ez SQL lekérdezések esetén megfelelő. Az adatkészlet megnyitását, és ezen keresztül a lekérdezés futtatását a `Lekérdezés` feliratú gomb komponens `OnClick` eseményében végezzük el.

Gondolnunk kell azonban arra az eshetőségre, hogy a felhasználó rossz lekérdezést ír be. Ennek érdekében ellenőriznünk kell a lekérdezés helyességét, amely úgy történik, hogy: eltároljuk a `CommandText` tulajdonságba írt lekérdezés szövegét, majd megvizsgáljuk az új szöveget egy `try...except` blokkban. Ha a lekérdezés hibás, adjon a program hibaüzenetet, és állítsuk vissza a régi lekérdezést. Az `OnClick` eljárás teljes szövege:

```
procedure TForm1.LekerdezesClick(Sender: TObject);
var
  CommandStr: WideString;
begin
  // elmentjük a régi CommandText-et, hiba esetére:
  CommandStr:=ADODataset1.CommandText;
  ADODataset1.Close;
  // átadjuk az új CommandText-et:
  ADODataset1.CommandText:=Edit1.Text;
```

```

try
  // megpróbáljuk megnyitni:
  ADODataset1.Open;
except
  MessageDlg('Nem megfelelő SQL lekérdezés!', mtError, [mbOk], 0 );
// hiba esetén visszaállítjuk az előzőt:
ADODataset1.CommandText:=CommandStr;
if CommandStr<>''
then
  ADODataset1.Open;
end;
end;

```

Ebben az esetben - ha a beírt lekérdezés megfelelő – az adatforrás aktívvá válik. Ha azonban a lekérdezés szövegében hibát vétünk, nem tudjuk ellenőrizni, hogy mit rontottunk el. Ebből a szempontból előnyösebb a tábla alapú rekordkészlet használata.

Természetesen a felhasználó csak `SELECT` utasítást, lekérdezést adhat, egyéb (pl. összes DDL, DML) utasítás hibás lekérdezésnek minősül, mivel nem ad vissza sorokat.

A három fennmaradó `Edit` szerkesztőmezőben tudjuk meghatározni az adatkészlethez alapul veendő tábla nevét (`Edit2` szerkesztőmező), a válogatáshoz használt kritériumot (`Edit3` szerkesztőmező), és az adatkészlet rendezéséhez alapot nyújtó mező(k) nevét (`Edit4` szerkesztőmező). Mindezeket a `Tábla_megnyitás` feliratú gomb `OnClick` eseményében olvassuk be.

A legfontosabb paraméter a tábla neve, amelyet ha nem adunk meg, az adatkészlet megnyitását nem szabad engedélyeznünk (ezt a `Tablanyitas` nevű gomb `Enabled` tulajdonságának állításával tudjuk szabályozni, amit az `Edit2` szerkesztőmező `OnChange` eseményében végzünk). Ha rosszul adjuk meg a tábla nevét, az alkalmazásnak hibaüzenetet kell adnia.

Ezután jön a válogatási és rendezési szempont ellenőrzése. A válogatási szempont megfelel egy lekérdezés `WHERE`, a rendezési pedig az `ORDER BY` záradéka *utáni* értéknek. Ezeket az `ADODataset` objektum `Filter` és `IndexFieldNames` tulajdonságaként találjuk meg, értéküket futásidőben is változtathatjuk. A `Filter` értéke mindig valamely mező és egy hozzá kapcsolódó érték, míg az `IndexFieldNames` értéke egy mező, vagy több mező neve vesszővel elválasztva.

Ellenőrizzük a `Filter`- és a `Filtered` tulajdonság `True` értékre állításával, ha hiba lép fel, visszaállítjuk a `Filtered` értékét `False` értékre. Az `IndexFieldNames` tulajdonságnak adott érték helyességét csak úgy tudjuk ellenőrizni, ha azt azonnal érvénybe is léptetjük.

Ezek alapján a tábla alapú rekordkészletet előállító `Tablanyitas` nevű komponens `OnClick` eseménye:

```

procedure TForm1.TablanyitasClick(Sender: TObject);
var
  TableStr: WideString;
begin
  // elmentjük a régi táblanevet:
  TableStr:=ADODataset2.CommandText;
  ADODataset2.Close;
  ADODataset2.CommandText:=Edit2.Text;    // átadjuk az újat:
  try
    ADODataset2.Open;                    // próbáljuk megnyitni:
  except
    MessageDlg('Nem létező tábla!', mtError, [mbOk], 0 );
    // hiba esetén visszaállítjuk az előzőt:
    ADODataset2.CommandText:=TableStr;
    if TableStr<>''
    then ADODataset2.Open;
  end;
  // ha sikerült megnyitni, bekérjük a másik két tulajdonságot is
  // (szűrés, rendezés)
  if ADODataset2.Active
  then
    begin
      if Edit3.Text<>'' then
        begin
          try
            ADODataset2.Filter:=Edit3.Text;
            ADODataset2.Filtered:=True;
          except

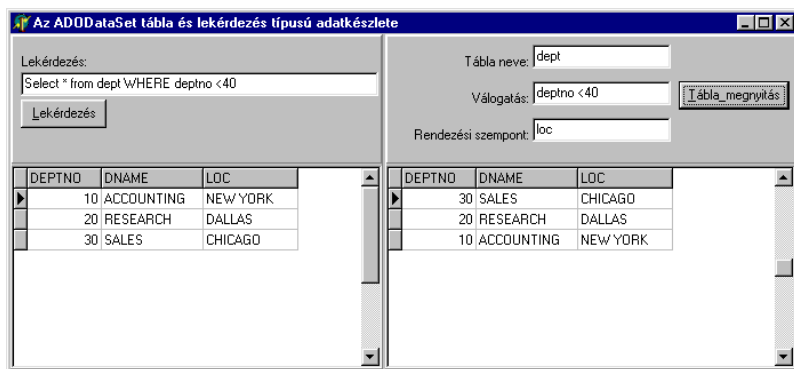
```

```

    MessageDlg('Hibás filter!', mtError, [mbOk], 0 );
    ADODataSet2.Filtered:=False;
end;
end;
if Edit4.Text<>''
then
begin
try
ADODataSet2.IndexFieldNames:=Edit4.Text;
except
MessageDlg('Hibás rendezési paraméter!',mtError, [mbOk], 0 );
ADODataSet2.IndexFieldNames:='';
end;
end;
end;
end;
end;

```

A program futási eredménye a 13.12. ábrán látható



13.12. ábra Az ADODataSet kétféle használata (Táblamegjelenítés és Lekérdezés)

Ilyen, az ADODataSet komponensre hasonlító (kétfajta adatkészletet létrehozni képes) komponens, nincs a BDE ben.

### Példa

Készítsünk egy adatbáziskezelő programot Delphi környezetben, amely az alább részletezett feladatot oldja meg az ADO kapcsolat segítségével az Oracle rendszerben.

(Először figyelmesen olvassa végig a feladatot!)

1. A program első lépésként az emp táblából hozzon létre egy alkalmazott nevű táblát, amelyet jelenítsen meg egy rácsban.
2. A rács fölött helyezkedjen el egy navigátor lécs, amelynek segítségével új rekord legyen bevihető az alkalmazott táblába, illetve abban a felhasználó kedve szerint tallózhasson.
3. A felhasználói felületen legyen egy Memo ablak, amelyhez rendelt gomb segítségével a Memo ablakba legyen betölthető egy lekérdezést vagy DDL utasítást tartalmazó script file.
4. A Memo ablak mellett legyen egy másik gomb is, amelynek segítségével a Memo ablakba betöltött, vagy oda kézzel beírt lekérdező, és DDL utasítás futtatható legyen.
5. A lefutott lekérdezés eredménye jelenjen meg a rácsban.
6. A státuszsorban jelenítsük meg az aktuális funkciót, a dátumot és időt.

Megoldás (65\_ADO\_Oracle\_script)

A feladat futási formja a 13.13. ábrán látható.

A Formra mint az ábrán is látható, két panel, egy memo komponens (a Standard komponens-palettáról), egy rács, egy navigátor (a Data Control komponens-palettáról), két splitter (az Additional komponens-palettáról) és egy státuszpanel (a Win32 komponens-palettáról) vizuális (futási időben látható) komponenst helyeztünk.

ENAME	JOB	SAL	DEPTNO
JONES	MANAGER	2975	20
BLAKE	MANAGER	2850	30
CLARK	MANAGER	2450	10
SCOTT	ANALYST	3000	20
KING	PRESIDEN	5000	10
FORD	ANALYST	3000	20

SELECT ename, job, sal, deptno FROM alkalmazott  
WHERE sal > 2000

Select utasítás végrehajtása    2001.11.28.    13:37:01

13.13. ábra Futási Form

Futási időben nem látható komponensek:

- DataSource komponens a Data Access komponens-palettáról,
- ADOConnection komponens az ADO komponens-palettáról,
- ADODataSet komponens az ADO komponens-palettáról,
- ADOCommand komponens az ADO komponens-palettáról,
- ADOQuery komponens az ADO komponens-palettáról,
- Timer komponens a System komponens-palettáról,
- OpenFileDialog komponens a Dialog komponens-palettáról.

A komponenseket a már ismert módon kapcsoljuk össze.

Az eljárások programjai:

#### A Memo törlése

```
procedure TForm1.memotorolClick(Sender: TObject);
begin
    Memo1.Clear;
end;
```

A létrehozásnál ellenőrizzük, hogy a tábla már létezik-e, van-e aktív rekordja. Ha van, adjunk üzenetet, ha nincs, hozzuk létre az új táblát az emp táblából allekérdezéssel. (Ez utóbbit az AdoQuery1 objektum Locate eljárásával ellenőrizhetjük, mivel nincs Exists eljárása.)

```
procedure TForm1.LetrehozClick(Sender: TObject);
begin
    AdoQuery1.Close;
    AdoQuery1.SQL.Add('Select table_name from user_tables');
    AdoQuery1.Open;
    if not AdoQuery1.Locate('table_name', 'alkalmazott', [])
    then
        begin
            ADOCommand1.CommandType := cmdText;
            ADOCommand1.CommandText := 'create table alkalmazott ' +
                'as (select * from emp) ' ;
            ADOCommand1.Execute;
            ShowMessage('Az alkalmazott tábla létrejött') ;
        end
    else
        ShowMessage('Az alkalmazott tábla létezik már');
end;
```

*Tábla megjelenítése*

```

procedure TForm1.MegjelenitClick(Sender: TObject);
begin
  ADODataSet1.Active := False;
  ADODataSet1.CommandType := cmdTable;
  ADODataSet1.CommandText := 'alkalmazott';
  DataSource1.DataSet := ADODataSet1;
  ADODataSet1.Active := True;
end;

```

A **Select** utasítás végrehajtása, az **ADODataSet** objektum **Active** tulajdonságának **True** értékre állításával vagy az **Open** eljárásával történhet. A lekérdezés eredménye megjelenik a rácsban.

```

procedure TForm1.LekerdezClick(Sender: TObject);
begin
  ADODataSet1.Active := False;
  ADODataSet1.CommandType := cmdText;
  ADODataSet1.CommandText := Memo1.Text;
  ADODataSet1.Active := True;
end;

```

Megnyitunk egy script programot (.sql kiterjesztésű állományt), és soronként betöltjük a tartalmát a **Memo** szerkesztőbe.

```

procedure TForm1.beolvasasClick(Sender: TObject);
var
  f: TextFile;
  sor : string;
begin
  OpenFileDialog1.Filter:='Sql file (*.sql)|*.sql|ALL files (*.all)|*.*';
  OpenFileDialog1.FileName := '';
  If OpenFileDialog1.Execute
  then
    begin
      AssignFile(f, OpenFileDialog1.FileName);
      Try
        reset(f);
        while not eof(f) do
          begin
            readln(f, sor);
            Memo1.Lines.Add(sor);
          end;
        Finally
          CloseFile(f);
        end;
      end;
    end;
end;

```

A státuszsorban való megjelenítéshez írjunk egy saját **GlobalEnter** eseményt. A **Sender** paraméter azonosítja, ellenőrzi a küldőt. A státuszpanel első részében a gombok **Hint** tulajdonságát jelenítsük meg, második részében a dátumot, a harmadikban pedig az időt. Minden gombnak meg kell adni a **Hint** tulajdonságát, valamint az **OnEnter** eseményéhez az általunk írt **GlobalEnter** eseményt, mint közös eseményt, kell a gördülő listából kiválasztani, beírni.

```

procedure TForm1.GlobalEnter(Sender: TObject);
//Minden Tcontrol OnEnter eseményéhez a GlobalEnter eseményt választjuk
begin
  StatusBar1.Panels[0].Text := (Sender as TControl).Hint;
  StatusBar1.Panels[1].Text := DateToStr(now);
  StatusBar1.Panels[2].Text := TimeToStr(now);
end;

```

A **DDL** parancsok végrehajtása az **ADOCommand** objektum **Execute** eljárásával történik.

```

procedure TForm1.VegrehajtClick(Sender: TObject);
begin
  ADOCommand1.CommandType := cmdText;
  ADOCommand1.CommandText := Memo1.Text;
  ADOCommand1.Execute;
end;

```

```
    ShowMessage('A parancs végrehajtva')  
end;
```

## Oracle adatbázis-kezelése ODAC komponensekkel

### Az ODAC komponensek

Az ODAC program az installálása egy Oracle Access nevű komponens-palettával bővíti eszközkészletünket. A komponens-paletta legvégére helyezi a hozzáadott új komponens-palettát, de ezt az egér jobb gombjának lenyomására előbukkanó menü Properties almenüjének segítségével átrendezhetjük. Ezt a 13.14. ábrán láthatjuk.



13.14. ábra: Az ODAC komponens-paletta

A komponensek közül az OraQuery, a SmartQuery, az OraStoredProc, az OraSQL, az OraUpdateSQL és OraScript képesek arra, hogy velük lekérdezéseket továbbítsunk az Oracle adatbázisunk felé.

Egyetlen lekérdezés használatát engedélyezi az OraQuery, a SmartQuery és az eredményhalmaz visszaadására képes az OraQuery, a SmartQuery és az OraStoredProc. Módosításokat egy kijelölt adatkészleten az OraQuery, a SmartQuery, az OraSQL, az OraStoredProc és az OraUpdateSQL tud végrehajtani. A PL/SQL blokkokat az OraQuery, az OraSQL, és az OraScript képes futtatni; több, egymás után írt lekérdezés futtatására pedig az OraScript képes.

### A KAPCSOLAT FELÉPÍTÉSE



Az OraSession komponens beállítja és vezérli a kapcsolatot az Oracle adatbázissal. Ez a komponens mindenképpen szükséges a program működéséhez. Minden komponens, amely adathoz akar hozzáférni, mint például az OraQuery, az OraSQL, vagy az OraScript, használja az OraSession komponenst. Mindegyiküknél megtalálható a Session tulajdonság, amely a saját Session-re mutat.

Az adatbázishoz való csatlakozásra az OraSession objektum Connect vagy Open eljárását kell meghívunk. Ha már tervezési időben látni szeretnénk az eredménytáblát, akkor a Connected tulajdonságot True értékre állítunk. Nem túl nagy hiba, de érdemes kikerülni, hogy ne állítsuk az OraSession objektum Connected (bejelentkezve) és ConnectPrompt (bejelentkező ablak megjelenítése) tulajdonságát együtt True értékre. Ilyenkor, ha bejelentkezéskor a Mégse gombra kattintunk, a program hibával leáll. Érdemes a Connected tulajdonságot tervezési időben False értékre állítani. Ha nem szeretnénk látni az adatbázis adatait tervezési időben, akkor használjuk az OraSession objektum Options/ NeverConnect tulajdonságát. Ennek True értékre állításával azt érjük el, hogy futásidőben induláskor az alkalmazás soha nem lesz csatlakozva a szerverhez, a Connected tulajdonság tervezési időben beállított értékétől függetlenül.

A ConnectPrompt tulajdonság a bejelentkező párbeszédablak megjelenítését (True) illetve nem megjelenítését (False) jelzi.



A ConnectDialog komponensnek *csak tulajdonságai* vannak, nincsenek eljárásai. Ez a komponens mindössze a csatlakozáshoz használt párbeszédablak tulajdonságainak beállítására használható. Egy Session csatlakozó ablakának tulajdonságait akkor tudjuk beállítani, ha elhelyezünk a Formon egy ConnectDialog komponenst, majd beállítjuk a Session objektum ConnectDialog tulajdonságához a használni kívánt csatlakozó ablak nevét.

A ConnectDialog komponens LabelSet tulajdonságában beállíthatjuk a csatlakozó ablakunk címkéinek nyelvét. Magyart sajnos nem találunk, de az IsCustom értéket beállítva a címkéket egyenként módosíthatjuk (UsernameLabel, ServerLabel stb.).



A BDESession ODAC komponens segítségével tudunk a BDE-n keresztül (nem csak Oracle) adatbázisokhoz csatlakozni.

### ADATFORRÁS MEGHATÁROZÁSA

Az adatbázis adatainak olvasásához nem elég csak a Session komponens, ez csak a kapcsolat vezérlésére szolgál. Következő lépésként meg kell határoznunk a használni kívánt adatforrást. Ez történhet az OraQuery, a SmartQuery, az OraTable és az OraStoredProc komponenssel.



A legegyszerűbb módszer, amivel az adatbázis egy táblájának adatait elérhetjük, az `OraTable` komponens használata. Ennél az összetevőnél mindössze a `Session` tulajdonságot kell beállítanunk, majd a `TableName` tulajdonságnál kiválasztanunk az elérni kívánt tábla nevét.

Az `OraTable`, mint nevéből is kitűnik, az Oracle adatbázis egy táblájának megjelenítésére képes, tulajdonképpen egyetlen `SELECT` lekérdezés eredményét jeleníti meg. Ez a komponens többé-kevésbé egyenértékű a Delphi `Table` komponensével, annak legtöbb tulajdonsága és eljárása alkalmazható rá. Legnagyobb hiányosság, amit felleltünk, hogy nem találhatók meg a `Table` objektumnál használatos `FindNearest` és `FindKey` eljárások. Azonban, mivel a `TOraTable` osztály a `TCustomSmartQuery`-ből származtatott, megvannak az ennek megfelelő tulajdonságai, mégpedig a `KeyFields`, amely egy vesszővel elválasztott mezőlista. Az `OraTable` objektum használ elsődleges és egyedi kényszerű, valamint `ROWID` pseudooszlop mezőket.

A `FilterSQL`, az `AddWhere`, illetve a `SetOrderBy` eljárások szintén a tábla lekérdezés jellegét használják ki.

A `FilterSQL` értékének módosításakor megváltozik a lekérdezés `WHERE` záradéka, majd a lekérdezés automatikusan frissül. Ezt a módosítást visszaállítani úgy tudjuk, hogy a `FilterSQL` értékét üres sztringre állítjuk. Érdekesség: a `FilterSQL` értéket nem tudjuk visszaolvasni a tábla lekérdezés szövegéből, de ha `Debug` tulajdonsággal ellenőrizzük, ott megjelenik.

Ugyancsak a `WHERE` záradékot módosítja, azonban nem törli, hanem hozzáfűz az `AddWhere` eljárás. Ha még nincs `WHERE` záradék, az `AddWhere` eljárás létrehozza.

A `SetOrderBy` eljárás felépíti a lekérdezés `ORDER BY` záradékát. A mezőket pontosvesszővel válasszuk el.

A `RestoreSQL` eljárás visszaállítja az `AddWhere` és a `SetOrderBy` eljárások módosításait a lekérdezés szövegében.

Természetesen, ugyanúgy mint a lekérdezéseknél általában, a lekérdezés szövegének módosítása előtt a táblát be kell zárni, majd a módosítás után aktiválni. Ez alól a fent említett eljárások közül csak a `FilterSQL` kivétel.

A táblában megjelenő adatokat módosíthatjuk is, nem kell lekérdezéseket írunk, mint például az `OraQuery` komponensnél.



Az `OraQuery` objektum SQL lekérdezéseket alkalmaz az Oracle tábla vagy táblák adatainak eléréséhez, és szolgáltatja ezeket az adatokat egy vagy több adatfüggő komponensnek az adatforrás komponensen keresztül. Az SQL lekérdezések előre definiáltak az `SQLInsert`, az `SQLDelete`, az `SQLUpdate` tulajdonságokkal (ezek akármilyen SQL vagy PL/SQL lekérdezések lehetnek). Ezen kívül hívhatók még ezzel a komponenssel a tárolt eljárások is.

Az `OraQuery` komponens csak olvasható, mikor nincs definiálva az `SQLInsert`, `SQLDelete` vagy `SQLUpdate` tulajdonság valamelyike. Ezek a tulajdonságok tervezési időben definiálhatók az `OraQuery` objektum SQL tulajdonságának szerkesztőjével (dupla kattintás a komponensen), vagy az Object Inspectorban, valamint futási időben az azonos nevű tulajdonságok változtatásával.

Ismerkedjünk meg közelebbről az `OraQuery` komponenssel. Az `OraQuery` fő jellemzője az SQL tulajdonság értéke, a többi Delphi lekérdezést végrehajtó komponenshez hasonlóan (`Query`, `ADOQuery`). Amikor az `OraQuery` objektumot futtatjuk, mindig a tervezési időben az SQL tulajdonságba beírt alaplekérdezést aktiváljuk. A futtatás történhet az `Active` tulajdonság `True` értékre állításával, illetve az `Open`, az `Execute` vagy az `ExecSQL` eljárások hívásával. Az első kettő teljesen egyenértékű, míg az `ExecSQL` csak akkor használható, ha a lekérdezés nem ad vissza sorokat (nem `SELECT`, hanem `INSERT`, `UPDATE`, `DELETE` stb. utasítás). Az `ExecSQL` előkészíti a lekérdezést futtatás előtt (ld. később: `Prepare` eljárás).

Mint tudjuk, az `OraQuery` automatikusan végzi a beszúrás, módosítás és törlés műveleteket, azonban csak az SQL tulajdonságba a felhasználó által megírt lekérdezés futtatását végzi automatikusan. A vonatkozó lekérdezéseket és ezek paramétereit a komponens szerkesztőjének `UpdateSQL` lapjára írhatjuk (vagy ugyanezt megtehetjük az Object Inspectorban is). A megfelelő műveletet a `Statement Type` rádiógombokkal választhatjuk ki, például az új rekord beszúrását az `Insert` műveletnél (vagy az Object Inspector-ban `SQLInsert` tulajdonságnál).

A lekérdezés futtatásának megszakítására az `OraQuery` objektum `BreakExec` eljárását (multiprogramozás esetén), a lekérdezés bezárásához a `Close` eljárását használjuk. A `Prepare` eljárás meghívásakor az adatbázis szerver erőforrásokat allokál a lekérdezéshez, előkészíti a futtatást. Ha alkalmazzuk ezt a eljárást a lekérdezés hívása előtt, az növeli az alkalmazás hatékonyságát. Ha már nem hívjuk többé a lekérdezést, használjuk az `UnPrepare` eljárást, ez felszabadítja az erőforrásokat (az előbbi mintára készíthetünk ide is egy kaméleon gombot). Azonban vigyázzunk ezek ésszerű használatára, mivel túl sokszori alkalmazásukkal ugyanazt érjük el, mint a használatuk nélkül. Ha előkészítés vagy futtatás előtt ellenőrizni akarjuk a lekérdezés szövegét, állítsuk az `OraQuery` objektum `Debug` tulajdonságát `True` értékre.

Az `OraQuery` objektum `Macros` tulajdonsága egy makró értékét reprezentálja. A makró egy változó, ami sztring értéket képvisel. Csak be kell illeszteni a makró nevét (az `&` karakterrel megelőzve) a lekérdezés szövegébe, és megváltoztatni az értékét az `OraQuery` szerkesztő `Macros` lapján tervezési időben, vagy a makró `Value` tulajdonságának

megváltoztatásával futásidőben. A *lekérdezés megnyitásakor* a makró az értékével helyettesítődik. Futási időben a *makrónak mindig kell, hogy legyen értéke!*



A SmartQuery komponenst könnyebb használni, mint OraQuery komponenst, automatikusan végzi az INSERT, DELETE, UPDATE utasításokat, valamint automatikusan zárolja és frissíti a rekordokat. A SmartQuery és az OraQuery között az alábbi különbségek vannak:

- A SmartQuery csak egy táblát tud módosítani, míg az OraQuery többet is. A SmartQuery mindig az első táblát módosítja a használtak közül.
- Ahhoz, hogy rekordokat tudjunk módosítani a SmartQuery komponens segítségével, be kell állítanunk a kulcsmezőket a KeyFields tulajdonságnál, vagy a SELECT utasításnak az SQL tulajdonságban vissza kell adnia a módosítandó tábla Rowid értékét.
- A SmartQuery csak SQL lekérdezések és tárolt eljárások hívására alkalmas.



Az OraStoredProc az OraQuery osztályból származtatott, tárolt eljárások futtatására alkalmas. Csak az eljárásnevet kell megadnunk SQL lekérdezés írása nélkül.



A VirtualTable komponens az adatokat a memóriában tárolja, nincsenek mögötte fizikailag létező adatbázisok. Aki dolgozott már nagy mennyiségű adattal, amire csak futásidőben volt szükség, tudja hogy ez mekkora előny, mert használhatjuk az adatbázis-kezelő komponensek összes hasznos tulajdonságát.

## AZ ADATFORRÁS MEGJELENÍTÉSE



Az adatkészlet és az adatfüggő komponensek közti kapcsolat felépítésére a standard Delphi komponensek közül a DataSource, az ODAC készletből pedig az OraDataSource komponens szolgál. A kettő ugyanazokkal a tulajdonságokkal és eljárásokkal rendelkezik.

Egyéb ODAC komponensek



Az OraSQL egy SQL lekérdezést vagy egy PL/SQL blokkot képes használni, vagy tárolt eljárást hívni az adatbázis szerveren. A komponens lekérdezése nem ad vissza kurzort vagy sorokat, így a komponens nem szolgáltatthat adatkészletet.



Az OraUpdateSQL komponens – mint nevéből is kitűnik – a táblák adatainak módosítását végzi. Nincs saját kiválasztó lekérdezése, és nem képes PL/SQL blokkok vagy tárolt eljárások futtatására. A feladata kizárólag az adatfüggő komponenseken módosított adatok regisztrálása. Az OraSQL komponensen kívül minden adatkészletet meghatározó komponenshez kapcsolható, azok OraUpdateSQL tulajdonságának beállításával. Képes az INSERT, a DELETE, az UPDATE és a REFRESH műveletek végrehajtására, de a LOCK műveletre nem.



Az OraScript komponens gyakran szükséges egyedi SQL lekérdezések, vagy névtelen PL/SQL blokkok futtatása, de ez utóbbiban nem alkalmazhatunk DDL lekérdezéseket. Lehetőségünk van ilyen esetekben sok komponens, mint például az OraSQL használatára. Az OraScript alkalmazásával egymás utáni lekérdezéseket, mint egységeket futtathatunk. Az egyes lekérdezések elkülönítésére a pontosvesszőt (;) használjuk. Az OraScript komponenssel azonban nem tudunk önálló SQL DML parancsokat végrehajtani, illetve olyan lekérdezéseket futtatni, amelyek kurzort vagy sorokat adnak vissza.

### Példa (81\_ora1\_ODAC)

(Adatbázis táblájának megjelenítése ODAC komponensekkel.)

Legelső lépésként itt is a legegyszerűbb alkalmazást nézzük meg, azaz hogyan csatlakozhatunk az adatbázisunkhoz, hogyan nyerhetünk belőle egyszerűen információkat és módosíthatjuk azokat.

Készítsünk egy új alkalmazást.

Helyezzünk el a Formon egy OraSession komponenst az Oracle Access komponens-palettáról. Ha duplán kattintunk rá, megjelenik egy szerkesztőablak, ami megkönnyíti a beállítást (ugyanezeket az információkat beállíthatjuk az Object Inspector-ban is). Ez a szerkesztő a legtöbb komponensnél megtalálható.

A felhasználónév, a jelszó és a szerver információk beállítása után ellenőrizhetjük a kapcsolatot a Connect gombra kattintással (vagy a Connected tulajdonság True-ra állításával). A felhasználónév és a jelszó megadása nem kötelező,

gyakorlati jelentősége abban rejlik, hogy ha megadjuk ezeket az információkat, akkor a kapcsolatot a program futtatása nélkül is tudjuk ellenőrizni. Ha nem szeretnénk minden egyes alkalommal beírni ezeket, írjuk be az `OraSession` komponens `ConnectionString` tulajdonságához a felhasználó-nevet és a jelszót / jellel elválasztva, esetünkben legyen ez `scott/tiger`.

Adatforrás elérésére az egyszerűség kedvéért az `OraTable` komponenst használjuk. Hogy az adatokat láthatóvá tegyük, helyezzünk el a Formon egy `OraDataSource`, egy `DBGrid`, és egy `DBNavigator` komponenst is.

Állítsuk az `OraTable1` objektum `Session` tulajdonságát `OraSession1`-re, a `TableName` tulajdonságot a használni kívánt tábla nevére (legyen például a `Scott.Empl`). Az `OraDataSource1` objektum `DataSet` tulajdonságát `OraTable1`-re, majd az `DBGrid1` komponens `DataSource` tulajdonságát `OraDataSource1`-re. Ha ezek után az `OraTable1` objektum `Active` tulajdonságát `True` értékre állítjuk, megjelennek a rácsban a tábla adatai.

Háromféleképpen érhetjük el, hogy a tábla adatait közvetlenül *ne* lehessen módosítani: A `DBGrid1`, vagy az `OraTable1` objektum `ReadOnly` tulajdonságát állítsuk `True` értékre, vagy az `OraDataSource1` objektum `AutoEdit` tulajdonságát `False` értékre. A különbség köztük mindössze annyi, hogy az utolsó kettőnél explicit módon meghívhatjuk az `OraTable1` objektum `Edit` eljárását, míg az elsőnél érthető okokból a `Cannot modify a read-only dataset` hibaüzenetet kapjuk.

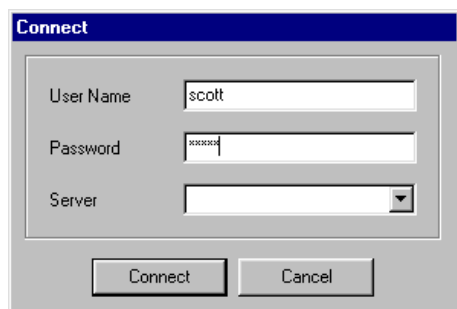
A másik módszer a csatlakozásra forráskód írása. Ha a következő programrészt elhelyezzük a Form `OnCreate` eljárásában, az előző eredményt érjük el:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  With Orasession1 do
    begin
      Username := 'scott';
      Password := 'tiger';
      Connect;
    end;
    OraTable1.Open;
  end;
```

Az alkalmazás által felépített kapcsolat kétirányú, mert a táblában megjelenő adatokat nem csak olvasni, hanem módosítani is tudjuk a `DBNavigator` komponens segítségével (`DataSource` tulajdonságát `DataSource1`-re állítjuk). Ha a tábla megjelenítését a rácsban csak a futási időben szeretnénk látni, akkor helyezzünk fel két gombot (Megnyit, Bezár), és írjuk meg a hozzájuk tartozó `OnClick` eseményt. Ekkor fejlesztési időben a `Table1` objektum `Active` tulajdonsága `False` értékű.

```
procedure TForm1.MegnyitClick(Sender: TObject);
begin
  //OraTable1.Open;           // Mindhárom megfelel
  OraTable1.Active:=True;
  //Oratable1.Execute;
end;
```

Futtatáskor megjelenik a jelszóbekérő ablak (13.15. ábra)



13.15. ábra Az ODAC komponensekkel indított program alapértelmezett jelszóbekérő ablaka

A Bezár gomb `OnClick` eseményének eljárása:

```
procedure TForm1.BezarClick(Sender: TObject);
begin
  //OraTable1.Active:= False;      // Mindkettő jó.
  OraTable1.Close;
end;
```

A Close ikonos (bitképpel ellátott) gomb az alkalmazás bezárására szolgál. (13.16. ábra)

Az adatforrás State állapotát megfigyelhetjük, és a Form fejlécébe kiírhatjuk a DataSource objektum OnStateChange eseményének eljárásában (82\_oral\_ODAC). A programban a rácsot felhasználók számára jól kezelhetővé tettük (oszlopfejlécek formázása, szín stb.).

```
procedure TForm1.DataSource1StateChange(Sender: TObject);
var
  szoveg : string;
begin
  case OraTable1.State of
    dsBrowse:   szoveg := 'Tallóz';
    dsEdit :    szoveg := 'Szerkeszt';
    dsInsert:   szoveg := 'Beszúrás';
  else
    szoveg := 'Másik állapot';
  end; //case
  Form1.Caption := 'RÁCS ÁLLAPOTA ' + szoveg;
end;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7777	KELEMEN	ANALYST	2222	2001.09.26.	2222	
7499	ALLEN	SALESMAN	7698	1981.02.20.	1600	
7521	WARD	SALESMAN	7698	1981.02.22.	1250	
7566	JONES	MANAGER	7839	1981.04.02.	2975	
7654	MARTIN	SALESMAN	7698	1981.09.28.	1250	
7698	BLAKE	MANAGER	7839	1981.05.01.	2850	
7782	CLARK	MANAGER	7839	1981.06.09.	2450	
7788	SCOTT	ANALYST	7566	1987.12.09.	3000	
7839	KING	PRESIDEN		1981.11.17.	5000	
7844	TURNER	SALESMAN	7698	1981.09.08.	1500	
7876	ADAMS	CLERK	7788	1987.05.23.	1100	
7900	JAMES	CLERK	7698	1981.12.03.	950	
7902	FORD	ANALYST	7566	1981.12.03.	3000	

13.16. ábra Tábla megjelenítése az ODAC komponensekkel

### Példa (90\_oracle).

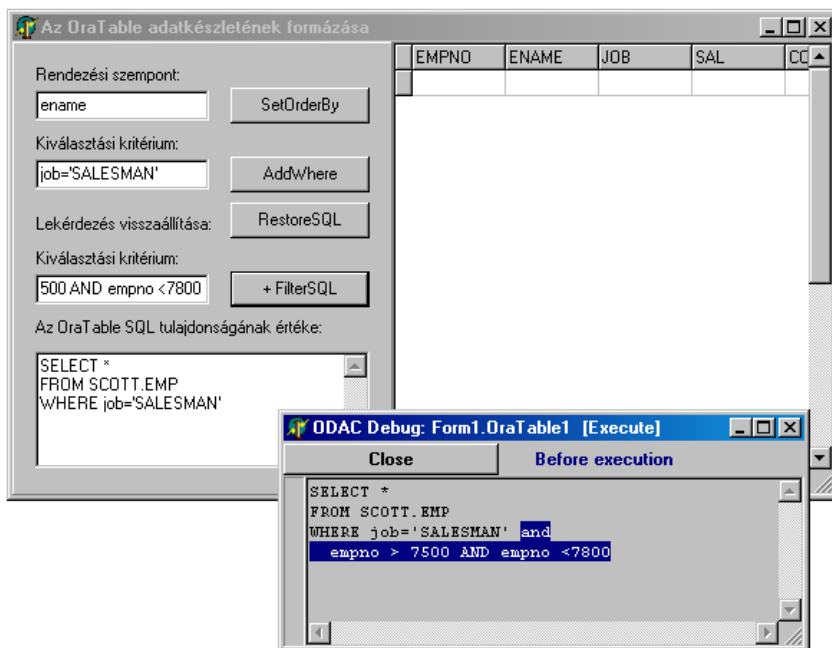
(Az OraTable lekérdezési jellegét, tulajdonságait, eljárásait mutatja be.)

Az alkalmazást használva a felhasználó képes lesz a lekérdezés szövegének alakítására úgy, hogy annak egyes záradékainak értékét külön-külön tudja beírni.

Készítsünk egy új alkalmazást. Helyezzünk egy üres Formra egy OraSession, egy ConnectDialog, egy OraTable és egy OraDataSource komponenst. Állítsuk az OraSession objektum ConnectDialog tulajdonságát ConnectDialog1-re, ezután a ConnectDialog objektum tulajdonságait állítsuk be ízlésünknek megfelelően, azaz magyar jelszóablakot készítsünk. (13.17. ábra). Állítsuk továbbá az OraTable objektum OraSession tulajdonságát OraSession1-re, TableName tulajdonságát SCOTT.EMP-re, és az OraDataSource objektum DataSet tulajdonságát OraTable1-re. Tegyük fel egy Panel komponenst Form bal- és egy adatrácsot a jobb oldalára. Állítsuk a Panel objektum Align jellemzőjét alLeft-re, az adatrács Align tulajdonságát alClient-re, a DataSource tulajdonságát pedig OraDataSource1-re. Ha ezek után aktiváljuk az OraSession és az OraTable objektumokat, a tábla sorai megjelennek az adatrácsban.

13.17. ábra Magyar jelszóablak

A Form futási képe a 13.18. ábrán látható.



13.18. ábra Futási kép

Rakjunk fel egy-egy gomb komponenst a SetOrderBy, az AddWhere, a RestoreSQL és a FilterSQL eljárásokhoz, és adjuk a gombok Caption tulajdonságának a hívott eljárások nevét.

Az első gomb legyen a SetOrderBy eljárásé. Az eljárás hívása előtt be kell zárnunk a táblát. Nyitáskor ellenőriznünk kell, hogy történt-e nem megfelelő esemény (kivétel). Ha történt, a SetOrderBy eljárással töröljük a rendezési szempontot:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    OraTable1.Close;
    //A rendezés alapját képező mezőt a SetOrderBy paramétereként is megadhatjuk.
    OraTable1.SetOrderBy(Edit1.Text);
    TRY
        // Ez a tábla megnyitásakor lép érvénybe, ezért a hiba-ellenőrzést a
        // megnyitáskor végezzük:
        OraTable1.Open;
    EXCEPT
        MessageDlg('Hibás rendezési paraméter!', mtError, [mbOk], 0);
        // Hiba esetén visszaállítjuk a rendezést az alapértelmezett értékre
        OraTable1.SetOrderBy('');
        OraTable1.Open;
    end;

    // Megjelenítjük a tábla SQL tulajdonságát a Memo-ban. A lekérdezés
    // módosul (kiegészíti egy ORDER BY záradék):
    Memo1.Lines.Clear;
    Memo1.Lines.Add (OraTable1.SQL.Text);
end;
```

Egy időben több mezőt is megadhatunk a rendezés alapjául, azokat pontosvesszővel (;) elválasztva.

Az AddWhere hibás felhasználói paraméterezése esetén nem tudjuk visszaállítani az eredeti lekérdezést, csak a RestoreSQL eljárással. Ez azonban visszaállítaná az esetleg beállított ORDER BY záradékot is, ezért el kell tárolnunk a lekérdezés eredeti szövegét, és kivétel esetén visszaállítani:

```
procedure TForm1.Button2Click(Sender: TObject);
var
```

```

    SQLText: String;
begin
    OraTable1.Close;
    { Elmentjük a tábla SQL tulajdonságát. Erre azért van szükség, mert
      nincs lehetőségünk az előző módszerhez hasonló visszaállításra hiba
      esetén ( ADDWhere('') ).
      Az AddWhere mindig megtoldja az SQL tulajdonságot. }

    SQLText:=OraTable1.SQL.Text;
    OraTable1.AddWhere(Edit2.Text);
    TRY
        OraTable1.Open;
    EXCEPT
        MessageDlg('Hibás kiválasztási paraméter!', mtError, [mbOk], 0);
        // visszaállítjuk a hívás előtti SQL tulajdonságot:
        OraTable1.SQL.Text:=SQLText;
        OraTable1.Open;
    end;
    // megjelenítjük a tábla SQL tulajdonságát a Memo-ban. A lekérdezés
    // módosul (hozzájön egy WHERE záradék):
    Memo1.Lines.Clear;
    Memo1.Lines.Add (OraTable1.SQL.Text);
end;

```

Az OraTable1 objektum RestoreSQL eljárásának végrehajtása a Button3 (RestoreSQL feliratú) komponens onClick eljárásából történik. Ez visszaállítja az AddWhere és SetOrderBy módosításait.

```

procedure TForm1.Button3Click(Sender: TObject);
begin
    OraTable1.Close;
    //A RestoreSQL visszaállítja a tervezési időben beállított SQL értéket:
    OraTable1.RestoreSQL;
    OraTable1.Open;
    Memo1.Lines.Clear;
    Memo1.Lines.Add (OraTable1.SQL.Text);
end;

```

A FilterSQL hívásánál annyi eltérés van, hogy ennek módosításait a RestoreSQL eljárás nem állítja vissza, így azt nekünk kell megtennünk. Tegyük ezt egy kaméleongommbal. Védekeznünk kell a rossz filter beállítása ellen is. kivétel esetén az alkalmazás írjon ki egy hibaüzenetet, majd törölje a filter értékét. A filter jelenlétét jelezze a + Filter felirat a Button4 nevű komponensen.

```

procedure TForm1.Button4Click(Sender: TObject);
begin
    // Ha a FilterSQL tulajdonság üres, vagy eltér az Edit3 nevű EditBox
    // Text értékétől, akkor módosítjuk, hiba esetén töröljük azt:
    if (OraTable1.FilterSQL='') or (OraTable1.FilterSQL<>Edit3.Text)
    then
        begin
            try
                OraTable1.FilterSQL:= Edit3.Text;
                Button4.Caption:='- FilterSQL';
            except
                MessageDlg('Hibás filter!', mtError, [mbOk], 0);
                OraTable1.FilterSQL:='';
                OraTable1.Open;
            end;
        end
    else
        // egyébként töröljük:
        begin
            Button4.Caption:='+ FilterSQL';
            OraTable1.FilterSQL:='';
        end;
    // FilterSQL tulajdonság állítása nem jelenik meg az SQL tulajdonságban:
    Memo1.Lines.Clear;
    Memo1.Lines.Add (OraTable1.SQL.Text);
end;

```

Ha ezt elvégeztük, akkor futás közben gombokkal tudjuk a lekérdezést formázni. Adjunk az alkalmazáshoz egy olyan lehetőséget is, amellyel a felhasználó képes a tábla SQL tulajdonságának követésére. Hogy a lekérdezés szövegét megfigyeljük, tegyünk a Panel-ra egy Memo komponenst. Megtehetnénk ezt a tábla Debug tulajdonságának True értékre állításával is, de mint megfigyelhetjük, a két érték a FilterSQL-nél eltérő lesz. Minden nyomógomb OnClick eseményének utolsó két sora legyen a lekérdezés szövegének Memo1-be helyezése.

A táblák lekérdezés jellegét csak az OraTable használja ki (sem a BDE Table, sem az ADOTable nem rendelkezik AddWhere, SetOrderBy, FilterSQL és RestoreSQL eljárásokkal).

A példa BDE-s és ADO-s változatában azonban az FilterSQL eljárás (ami a kiválasztás egyik módja) kiváltható a Filter, illetve a SetOrderBy az IndexFieldNames tulajdonsággal.

### Példa

Készítsünk egy adatbáziskezelő programot Delphi környezetben, amely az alább részletezett feladatot oldja meg az ODAC kapcsolat segítségével az Oracle rendszerben.

(Mielőtt a feladatot megoldjuk, figyelmesen olvassuk végig a kiírást.

- A program első lépésként az emp táblából hozzon létre egy dolgozo nevű táblát, amelyet jelenítsen meg egy rácsban.
- A rács fölött helyezkedjen el egy navigátor lécz, amelynek segítségével új rekord legyen bevihető a dolgozo táblába, illetve abban a felhasználó kedve szerint tallózhasson.
- A felhasználói felületen legyen egy memo ablak, amelyhez rendelt gomb segítségével a memo ablakba legyen betölthető egy módosítást tartalmazó script program.
- A memo ablak mellett legyen gomb is, amelyeknek segítségével a memo ablakba betöltött, vagy oda kézzel beírt DML utasítás futtatható legyen.
- A lefutott SQL utasítások eredményét jelenítse meg a rácsban.

A program futási képe a 13.19.ábrán látható.

13.19. ábra Futási Form

A Formon a látható komponensek a következők:

- 2 panel, 4 gomb egy memo a Standard komponens-palettáról,
- Splitter Additional komponens-palettáról,
- Egy rács és felette egy navigátor a Data Controls komponens-palettáról.

Az alábbi nem vizuális komponensek találhatók a Formon az Oracle Access komponens-palettáról:

- OraSession,
- OraDataSource,
- OraTable,
- OraQuery.

Kapcsoljuk a komponenseket megfelelően össze.

A Létrehoz feliratú gomb `OnClick` eseménye, létrehozza a táblát, ha az még nem létezik. Az ellenőrzéshez először lekérdezzük a felhasználói rendszertáblából a táblaneveket. Ezeket az ODAC egy átmeneti, ún. LOB locator területen tárolja. Itt ellenőrizhetjük, hogy az általunk létrehozni kívánt tábla létezik-e, azaz van-e aktív rekordja, ha nincs, akkor hozzuk létre allekérdezéssel az új táblát az `OraQuery` objektum `SQL` tulajdonságába beírt `Create` paranccsal, és az objektum `Execute` eljárásával.

```
procedure TForm1.LetrehozClick(Sender: TObject);
begin
    OraQuery1.Close;                                //Bezárjuk
    // Töröljük a tervezési időben beírt SQL tulajdonságot
    OraQuery1.SQL.Clear
    // Lekérdezzük, a felhasználói táblák listáját, ezt az Odac egy átmenti
    // ún. LOB locator területen tárolja
    OraQuery1.SQL.Add('SELECT table_name from user_tables');
    OraQuery1.Open;
    // Ellenőrizzük van-e ezen a területen aktív rekord az általunk
    //létrehozni kívánt táblában, ha nincs, azaz a tábla nem létezik,
    //létrehozzuk a táblánkat.
    if not OraQuery1.Locate('table_name', 'dolgozo', [])
    then
        begin
            OraQuery1.SQL.Clear;
            OraQuery1.SQL.Add('CREATE Table dolgozo as SELECT * From emp');
            OraQuery1.Execute;
        end
    else
        ShowMessage('Ilyen tábla már van');
end;
```

A Megjelenít gomb a dolgozo táblát jeleníti meg a `Table` objektum megnyitásával.

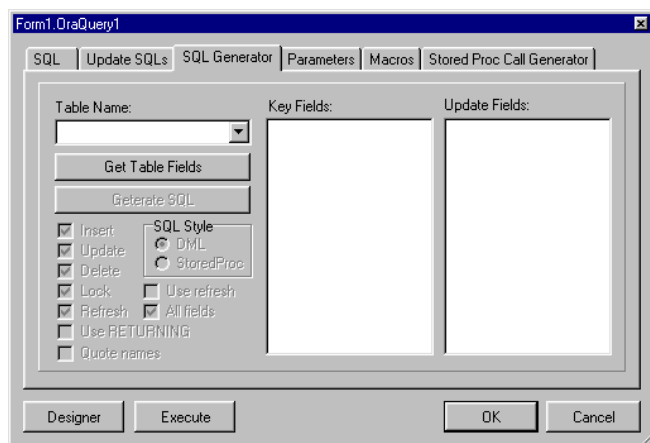
```
procedure TForm1.MegjelenitClick(Sender: TObject);
begin
    OraTable1.Close;                                // bezár
    OradataSource1.DataSet := OraTable1;           // OraTable1 a megjelenítendő
    OraTable1.TableName := 'dolgozo';               // A tábla neve
    OraTable1.Open;                                  // Megnyitás
end;
```

A Betölt feliratú gomb `OnClick` eseményében a felhasználó képes a már meglévő script programot (egy – egy SQL DML, vagy DDL utasítást) betölteni a `memo` objektumban. Itt módosítani, vagy átírni is módja van. Ehhez szükséges az `OpenDialog` komponens. Az erőforrásokat (az állományt is) mindig le kell zárni, ezért ilyen esetben feltétlenül szükséges a `Try Finally end` kivételkezelő blokk használata.

```
procedure TForm1.BetoltClick(Sender: TObject);
var
    f: TextFile;                                     // fájlváltozó
    sor : string;
begin
    OpenDialog1.Filter:='SQL file (*.sql)|*.sql|ALL files (*.all)|*.*';
    OpenDialog1.FileName := '';
    If OpenDialog1.Execute then
    begin
        AssignFile(f, OpenDialog1.FileName);
        Try
            reset(f);
            while not eof(f) do
            begin
                readln(f, sor);
                Memo1.Lines.Add(sor);
            end;
        Finally
            CloseFile(f);
        end;
    end;
end;
```

A Futtat\_DML feliratú gomb OnClick eseménye képes a Memo-ban levő SQL utasítás (egyszerre csak egy) futtatására. Ehhez jelen programrészben a már a Formon levő OraQuery objektumot használtuk fel. (Használható bármelyik ismertetett objektum, amelyik képes DML utasítások végrehajtására.) Az OraQuery objektum SQL tulajdonságának tartalmát a bezárás után törölni kell. Ezután adjuk meg az SQL tulajdonságnak vagy tervezési időben közvetlenül (bármilyen típusú az utasításunk: SELECT, INSERT, UPDATE, DELETE), vagy futáskor a Memo tartalmát. A Memo tartalmát többféle eljárással adhatjuk át az SQL tulajdonságnak (Add, Append, AddString, Assign). Vigyázzunk mindig a megfelelő típusegyeztetésekre. Az SQL tulajdonságba írt utasítások értelmezése és elemzése az OraQuery objektum feladata. Ezt megfigyelhetjük a Params tulajdonságra való kettős kattintásnál. Ha paraméterezett SQL utasítást szeretnénk használni, akkor azt az SQL Generator lap segítségével a paraméter-szerkesztőben generáltatjuk (13.20. ábra), ahol meg kell adnunk a táblanevet, majd a Generate gombra kattintva, a rendszer elkészíti a paraméterezett INSERT, UPDATE stb. utasítást illetve makrót. Ebben az esetben aztán a megfelelő ablak-fülekre kattintva, megjelenik a paraméterezett utasítás, amit a felhasználó által megadott paraméterekkel végre tudunk hajtani. Természetesen ehhez a Formon valamilyen szerkesztő, beviteli mező szükséges. A paraméterek jellemzőit a Parameters lapon adjuk meg. Az utasítás végrehajtását ellenőrizhetjük a fejlesztési időben az Execute gombra kattintással.

```
procedure TForm1.Futtat_DMLClick(Sender: TObject);
begin
  TRY
    OraQuery1.SQL.Clear;
    OraQuery1.SQL.AddStrings (Memo1.Lines);
    OraQuery1.Execute;
  Except
    ShowMessage ('HIBA!!!');
  end;
end;
```



13.20. ábra. OraQuery objektum Params tulajdonságának szerkesztője

**Példa (84\_oraz\_ODAC\_ciklusban\_keres)**

(Bemutatja az adattábla rekordonkénti feldolgozását.)

Feladatunk egy karaktersorozat megkeresése egy adattábla egyik karakter típusú oszlopában. A megtalált rekordot jelöljük ki.

A Form megjelenő képe a 13.21. ábrán látható.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7777	KELEMEN	ANALYST	2222	2001.09.26.	2222		10
7499	ALLEN	SALESMAN	7698	1981.02.20.	1600	300	30
7521	WARD	SALESMAN	7698	1981.02.22.	1250	500	30
7566	JONES	MANAGER	7839	1981.04.02.	2975		20
7654	MARTIN	SALESMAN	7698	1981.09.28.	1250	1400	30
7698	BLAKE	MANAGER	7839	1981.05.01.	2850		30
7782	CLARK	MANAGER	7839	1981.06.09.	2450		10
7788	SCOTT	ANALYST	7566	1987.12.09.	3000		20
7839	KING	PRESIDEN		1981.11.17.	5000		10
7844	TURNER	SALESMAN	7698	1981.09.08.	1500	0	30
7876	ADAMS	CLERK	7788	1987.05.23.	1100		20
7900	JAMES	CLERK	7698	1981.12.03.	950		30
7902	FORD	ANALYST	7566	1981.12.03.	3000		20
7934	MILLER	CLERK	7782	1982.01.23.	1300		10
7369	SMITH	CLERK	7902	1980.12.17.	800		20
8888	MAGYAR	ANALYST		0001.09.20.	3322		10
8765	KALAPOS	CLERK		2001.10.18.	1234		10

13.21. ábra A keresési feladat Formja

Az alkalmazás képes egy táblát megjeleníteni, abba új rekordokat Edit szerkesztőmezőből beszúrni BESZÚR gomb, rekordokat törölni a TÖRÖL gomb segítségével, valamint a keresést elvégezni.

A Formon található egy OraSession, egy OraTable és egy OraDataSource komponens az Oracle Access komponens-palettáról. Legyen egy Panel, rajta három gomb, valamint az Edit szerkesztőmezők és címkék. (Minden olyan adatot be kell vinnünk egy sorba, amelynek NOT NULL kényszere van.) Utána van egy Splitter, és egy rács komponens.

A komponenseket a már megismert módon kapcsoljuk össze, és az OraTable1 objektum Active tulajdonságát állítsuk True értékre. Ekkor a tábla a 13.21. ábrán látható módon jelenik meg.

A TÖRÖL gomb OnClick eljárása:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    OraTable1.Delete;      // az aktuális sort törli
end;
```

A BESZÚR gomb OnClick eljárása:

```
procedure TForm1.BeszurClick(Sender: TObject);
begin
    With OraTable1 do
        begin
            Insert;
            // Közvetlen érték megadással programból
            // FieldByName('Empno').Value:= 7777;
            // FieldByName('Ename').Value:= 'MAGYAR';
            // FieldByName('job').Value:= 'CLERK';
            // FieldByName('sal').Value:= 1111;
            // FieldByName('deptno').Value:=40;

            // Az értékek megadása futás közben a felhasználótól.
            FieldByName('Empno').Value := Edit1.Text;
            FieldByName('Ename').Value := Edit2.Text;
            FieldByName('job').Value := Edit3.Text;
            FieldByName('sal').Value := Edit4.Text;
            FieldByName('deptno').Value:= Edit5.Text;
            Post;      // beírja a módosított rekordot az adattáblába
```

```

    end;
    Edit1.Clear;
    Edit2.Clear;
    Edit3.Clear;
    Edit4.Clear;
    Edit5.Clear;
    Edit1.SetFocus;
end;

```

#### A Keres eljárás:

```

procedure TForm1.KeresClick(Sender: TObject);
var
    szoveg, nev : string;
    talal : boolean;
begin
    Talal := False;
    szoveg :=Inputbox('Név megadása', 'Kérem a keresendő nevet','');
    // Újrakeresés estén a kijelölés megszüntetése
    DBGrid1.SelectedRows.CurrentRowSelected:=False;
    with OraTable1 do
    begin
        DisableControls; // Adatmegjelenítés kikapcsolása. Rács léptetés
                           // esetén minden egyes rekordnál újrarajzol
    TRY
        First;           // első rekordra mutat a kurzor
        while not EOF do
        begin
            nev := FieldByName('ename').Value;
            // ShowMessage(nev);
            if pos(szoveg,nev) >0           // keresés pos függvénnnyel
            then
                begin
                    talal:= True;
                    Break;
                end;
            Next;           // A kurzor a következő rekordra mutat
        end;
        // ShowMessage(INTTOSTR(i));
        DBGRID1.SelectedRows.CurrentRowSelected:= True;
    Finally
        EnableControls;           // Adatmegjelenítés bekapcsolása
    end;
    if not talal
    then
        Showmessage('Nem talált') ;
    end;
end;

```

#### Példa (91\_ora\_oracle)

(Többtáblás lekérdezés.)

Készítsünk egy olyan alkalmazást, amely az `OraQuery` komponenst felhasználva összekapcsol két táblát. Jelenítse meg egy vállalat dolgozóit, továbbá azt, hogy melyik részlegen dolgoznak, és milyen beosztásban. Legyen képes a felhasználó az első tábla adatainak módosítására is.

A Form a 13.20. ábrán látható.

Helyezzünk el a Formon egy `OraSession`, egy `ConnectDialog`, egy `OraQuery`, és egy `OraDataSource` komponenst az Oracle Access komponens-palettáról, valamint egy `DBGrid` komponenst, majd kapcsoljuk össze a megfelelő tulajdonságaikat.

Állítsuk be az `OraSession` komponens `ConnenctString` tulajdonságát a `scott/tiger` értékre, és formázzuk a kapcsolódás párbeszédablakát a `ConnectDialog` tulajdonságainak állításával. A `LabelSet` tulajdonságnál található néhány előre definiált címkekészlet, de sajnos magyar nincs, így ezeket nekünk kell begépelnünk.

Ezután meg kell írunk a lekérdezést, ami a táblák adatait összekapcsolja. Ez lesz az „általános” alapértelmezett `SQL` tulajdonság:

```

SELECT  e.EMPNO, e.ENAME, e.JOB, e.DEPTNO, d.DNAME
FROM    Scott.Emp e, Scott.Dept d

```

```
WHERE e.DEPTNO = d.DEPTNO
ORDER BY e.DEPTNO
```

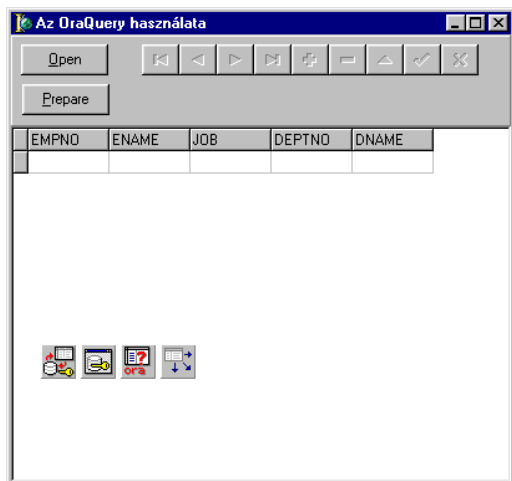
Ezt írjuk be az OraQuery szerkesztő (dupla kattintás a komponensen) SQL tulajdonságának szövegmezőjébe. A lekérdezés futtatásához tegyünk a Formra egy gomb komponenst, nevezzük btOpen-nek. Legyen a gomb úgynevezett kaméleon gomb, és az OraQuery állapotától függően nyitást vagy zárást hajtson végre. Az állapotát jelezzük a gomb feliratán (Caption). A műveletet a gomb OnClick eljárásában végezzük el, itt hívjuk meg az OraQuery objektum Open és Close eljárását.

```
procedure TForm1.btOpenClick(Sender: TObject);
begin
    // egyetlen gombbal végezzük a lekérdezés megnyitását és zárását (Open
    // és Close eljárásokkal). A Query állapotát a Button komponens Caption
    // tulajdonságával ellenőrizzük:
    if btOpen.Caption='&Open'
    then
        begin
            OraQuery.Open;
            btOpen.Caption:='&Close';
        end
    else
        begin
            OraQuery.Close;
            btOpen.Caption:='&Open';
        end;
end;
```

Ha futtatjuk a programot, az Open gombra kattintva megjelennek az adatrácsban a kívánt adatok, ezeket azonban csak olvasni tudjuk, módosítani nem. A tábla adatainak a felhasználó által kívánt aktuális módosításához (paraméterezett utasítások) meg kell adnunk az OraQuery komponens SQLInsert, SQLDelete, és SQLUpdate tulajdonságait is. Az Insert művelethez tartozó paraméterezett utasítások a String List Editorban

```
INSERT INTO Scott.Emp (EMPNO, ENAME, JOB, DEPTNO)
VALUES (:EMPNO, :ENAME, :JOB, :DEPTNO)
```

Láthatjuk, hogy a beszúrandó új értékeket a komponens automatikusan kapja meg, ennek feltétele csak az, hogy a paraméter neve a vonatkozó mező nevével egyezzen meg.



13.22. ábra Az OraQuery használata

Írjuk meg a törlést az SQLDelete tulajdonság a Strig List Editorába, és a módosítást az SQLUpdate tulajdonság String List Editorába.

```
DELETE FROM Scott.Emp
WHERE
    EMPNO=:EMPNO
```

Valamint a módosítás:

```

UPDATE Scott.Emp
SET
  EMPNO=:EMPNO,
  ENAME=:ENAME,
  JOB=:JOB,
  DEPTNO=:DEPTNO
WHERE
  EMPNO = :EMPNO

```

Ebben az esetben egy `OraQuery` objektumban módosítás, törlés, beszúrás és lekérdezés is végrehajtható. Az utasításokban paraméterek szerepelnek, amelyek a tábla megfelelő oszlopaiból értéket kapnak.

Az `OraQuery` csak egy tábla módosítására használható. Ha a másik táblát is módosítani akarjuk, máshogy kell a feladatot megoldanunk. (Például `OraUpdateSQL` komponens használatával, amelyet a Formra helyezünk, ezután az `OraQuery` objektum `UpdateObject` tulajdonságánál hozzárendeljük, és a módosító utasításokat a tulajdonság megnyitásával beírjuk a megfelelőhelyekre).

Állítsuk a `DBGrid` objektum `ReadOnly` tulajdonságát `False` értékre az `Object Inspector`-ban, majd a `dept` tábla oszlopainak (`DEPTNO`, `DNAME`) ugyanezen tulajdonságát `True` értékre (Ezt úgy tudjuk megtenni, ha az `OraQuery` objektum `Active` tulajdonságát `True` értékre állítjuk, majd a `DBGrid` objektum `Columns Editor` szerkesztőjében hozzáadjuk az összes mezőt. Ezután a megfelelő oszlopok tulajdonságait az `Object Inspector`-ban tudjuk állítani). Ezután az `emp` tábla adatai szerkeszthetővé válnak.

A példa az `OraQuery` komponens képességeinek bemutatását célozza (`Update`, `Insert`...). A másik két komponenscsomag `Query` típusú komponensei (`Query` és `ADOQuery`) nem képes önállóan ilyen feladatok ellátására.

### Példa (92\_oracle)

(Az `OraQuery` és az `OraScript` használata.)

Készítsünk olyan alkalmazást, amely képes arra, hogy egy vállalat munkaerő-gazdálkodási osztályának megmutassa a jelenleg foglalkoztatott összes személyt, majd ezeket rendezni lehessen különböző szempontok szerint.

Az alkalmazás az indulásakor nincs csatlakozva az adatbázishoz, ezt a `Csatlakozás` gombbal teheti meg a felhasználó. Ekkor megjelenik a vállalat összes dolgozója az adatrácsban. A munkavállalók adatait egy külön formon egyenként módosíthatjuk, új tagot vehetünk fel, vagy törölhetünk. Ezt a formot a `Módosítás` gombbal lehet megnyitni.

Helyezzük el a Formon kapcsolódáshoz szükséges komponenseket, majd egy nyomógombot, aminek neve legyen `btConnect`, a felirata `Kapcsolódás`. Ezzel fogunk csatlakozni az adatbázishoz, és lekapcsolódni is. Az adatok megjelenítésére használjuk az `OraQuery` komponenst, mellé helyezzünk `DBGrid`, valamint egy `OraDataSource` komponenst. Kapcsoljuk össze a megfelelő tulajdonságokat. A gomb `OnClick` eseményéből kell hívunk a `Session` objektum `Connected` tulajdonságától függően a `Connect` vagy `Disconnect` eljárásait.

Legyen lehetsége a felhasználónak az adatbázis tábláinak létrehozására első futtatáskor. A tábla létrehozásához szükség van egy `OraScript`, és egy `Táblák létrehozása` feliratú gomb komponensre. Az `OraScript` komponenssel több önálló lekérdezést tudunk egymás után futtatni, a lekérdezéseket pontosvesszővel elválasztva az `SQL` tulajdonságba kell írunk. A `btCreate` nevű, `Táblák létrehozása` feliratú gomb `OnClick` eljárásából hívjuk az `OraScript` objektum `Execute` eljárását, amellyel végrehajtjuk a két táblát törölő, létrehozó és adatokkal feltöltő `SQL` utasításokat. Ha nem léteznek a táblák, az Oracle-től a szokásos hibaüzenetet kapjuk vissza, de tovább futtatva a további `SQL` utasításokat végrehajtnak. A táblák törlésére, létrehozására és feltöltésére szolgálnak a következő `SQL` parancsok:

```

DROP TABLE empdata;
DROP TABLE BonusData;

CREATE TABLE EmpData
  (empno NUMBER, fname VARCHAR2(15), sname VARCHAR2(15), sal NUMBER,
   job VARCHAR2(15), offno INTEGER);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (1, 'KOVACS', 'PÉTER', 80000, 'TITKÁR', 10);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (2, 'KISS', 'PÁL', 60000, 'KARBANTARTÓ', 20);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (3, 'SZÉP', 'EDINA', 70000, 'KARBANTARTÓ', 20);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (4, 'BEDE', 'FERENC', 87000, 'TITKÁR', 20);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (5, 'ARADI', 'PÉTER', 63000, 'IG. HELYETTES', 20);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (6, 'PELO', 'ISTVÁN', 110000, 'IGAZGATO', 10);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
  VALUES (7, 'PROLÓG', 'ISTVÁN', 110000, 'IGAZGATO', 20);

```

```

INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
VALUES (8, 'ANGYAL', 'ZSUZSA', 110000, 'IGAZGATO', 30);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
VALUES (9, 'TÁRKONYI', 'FERENC', 110000, 'FEJLESZTŐ', 40);
INSERT INTO EmpData (empno, fname, sname, sal, job, offno)
VALUES (10, 'KAÁL', 'FRANCISKA', 90000, 'TITKÁR', 30);

CREATE TABLE BonusData (offno NUMBER, BonusData_val VARCHAR2(40));
INSERT INTO BonusData (offno, BonusData_val) VALUES (10, 15);
INSERT INTO BonusData (offno, BonusData_val) VALUES (20, 10);
INSERT INTO BonusData (offno, BonusData_val) VALUES (30, 8);
INSERT INTO BonusData (offno, BonusData_val) VALUES (40, 15);

```

Az `OraSession` komponens `Options` tulajdonságában lévő `NeverConnect` altulajdonság a fejlesztőknek azért hasznos, mert ha ezt `True` értékre állítjuk, akkor annak ellenére, hogy tervezési időben a `Connected` tulajdonság `True` értékű, a kapcsolódás nem jön létre, csak futási időben, és akkor is csak annyi ideig kapcsolódik a szerverhez, amíg az esemény lezajlik, azaz például egy lekérdezés, vagy tábla megjelenítésének idejéig.

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    Orasession1.Connected := True;
    // OraSession1.Connect;
    OraTable1.Open;
end;

```

Ennek az a jelentősége, hogy ha futási időben szeretnénk az adatokat vizsgálni, akkor ezt anélkül tehetjük meg, hogy a futtatás előtt és után át kelljen állítani a `Connected` tulajdonságot.

Annak érdekében, hogy a felhasználó keresni tudjon a táblában, és az adatait rendezni, tegyünk a Formra egy `Panel` komponens, rajta két `ComboBox` lenyitható szerkesztődobozzal és egy `Edit` szerkesztőmezővel. A `ComboBox2` objektum és az `Edit1` szerkesztőmező segítségével valósítja meg az alkalmazásunk a karakter típusú oszloptartalom keresését.

Az Oracle szerverhez való kapcsolódás a `btConnect` nevű gomb komponens révén történik. Az `OnClick` esemény állítja be a `ComboBox` objektumok `Items` tulajdonságait is:

```

procedure TfrmMain.btConnectClick(Sender: TObject);
var
    i: byte;
begin
    // ha kapcsolódva vagyunk, leválunk, ellenkező esetben kapcsolódunk
    if not (OraSession1.Connected)
    then
        begin
            OraSession1.Connect;
            ComboBox1.Items.Clear;
            ComboBox1.Items.Create;
            // az összes oszlop felvétele:
            // for i:= 0 to DBGrid1.Columns.Count-1 do
            //csak a karakter típusú oszlopok felvétele:
            for i:= 1 to 3 do
                ComboBox1.Items.Add(DBGrid1.Columns.Items[i].Title.Caption);
            ComboBox1.ItemIndex:=0;
            ComboBox2.Items.Create;
            ComboBox2.Items :=ComboBox1.Items;
            ComboBox2.ItemIndex:=0;
            OraQuery1.Open;
        end
    else
        OraSession1.Disconnect;
end;

```

Ahhoz, hogy az adatokat látni lehessen az adatrácsban, az `OraQuery` komponens `Active` tulajdonságának `True` értéket kell adnunk, ehhez azonban meg kell írunk a szükséges lekérdezéseket. Első lépésben írassuk ki az összes dolgozó minden adatát, vagyis az `SQL` tulajdonsága legyen:

```

SELECT * FROM EmpData

```

Két módszer adódik a lekérdezés által visszaadott adatkészlet további válogatására: a sztringkezelés elvével (a lekérdezés szövegének direkt módosításával) és az *OraQuery* tulajdonságait és eljárásait használva (*FilterSQL*-t, *SetOrderBy*-t, stb.). A legcélravezetőbb, ha mindkét módszert alkalmazzuk. Rendezni mindenképpen kell a kapott adatkészletet, amit célszerű makróval végrehajtani (a *Macros* az *OraQuery* objektum egyik tulajdonsága).

A standard Delphi SQL komponenseinek körében nem található meg a makró, pedig mint látni fogjuk, hasznos dolog. Írjunk hozzá lekérdezést.

```
SELECT D1.EMPNO, D1.FNAME, D1.SNAME, D1.JOB, D1.OFFNO, D1.SAL,
       D2.BONUSDATA_VAL, D1.SAL*D2.BONUSDATA_VAL/100
FROM SCOTT.EmpData D1, SCOTT.BonusData D2
WHERE D1.OFFNO=D2.OFFNO
ORDER BY D1.&Ord
```

Ezzel azt érjük el, hogy a rendezés az *Ord* makró értékével megegyező mező alapján történik. Az *OraQuery* objektum *Macros* tulajdonságai között automatikusan megjelenik az *Ord* makró, a *Value* ablakban adjuk meg a kezdőértéket, például az *EMPNO*-t. Ne felejtjük el, hogy *futtatáskor a makrónak mindig kell, hogy legyen értéke!* Arra figyeljünk, hogy legyünk következetesek, mert a lekérdezés eredményének rendezése ennek alapján történik, de a *ComboBox1* értéke mást is mutathat a felhasználónak. Nem hiba, de kerüljük el, tehát állítsuk a *ComboBox1* objektum *ItemIndex* tulajdonságát a makró kezdőértékének megfelelő értékre. A *ComboBox1* változtatásakor adjuk át az értéket a makrónak:

```
procedure TfrmMain.ComboBox1Change(Sender: TObject);
begin
  OraQuery1.Close;
  // a ComboBox1-gyel az OraQuery1 "Ord" nevű makróját tudjuk állítani,
  // ami a rendezési mező helyén áll a lekérdezés szövegében:
  OraQuery1.MacroByName('Ord').Value :=
    DBGrid1.Columns.Items[ComboBox1.ItemIndex].FieldName;
  OraQuery1.Open;
end;
```

Nem mindig válogatjuk azonban az adatkészletet, így a kritériumhoz tartozó mező neve üres sztring lenne a makró alkalmazása esetén, ami hiba. Itt tehát a válogatáshoz használt mező nevét és értékét a szöveghez közvetlenül hozzáfűzzük. A mező nevét a *ComboBox2* objektum *Items* tulajdonságából állapítjuk meg:

```
procedure TfrmMain.ComboBox2Change(Sender: TObject);
begin
  // a FilterSQL eljárással kiválogatjuk azokat a rekordokat, amelyek
  // megfelelnek a ComboBox2-ben megjelenő mező EditBox1-ben megjelenő
  // értékének:
  if Edit1.Text=''
  then OraQuery1.FilterSQL:= ''
  else
    OraQuery1.FilterSQL:=DBGrid1.Columns.Items[ComboBox2.ItemIndex].
      FieldName + '=' + Edit1.Text + '';
end;
```

Ezeket a lépéseket végigkövetve az adatbázis adatait még nem tudjuk szerkeszteni. Használjunk erre egy új Formot.

Vegyük ki a *DBNavigator* gombokból álló komponensből a *VisibleButtons* tulajdonságának állításával a szerkesztő gombokat, ezek úgyis feleslegesek.

Tegyünk az alkalmazáshoz egy *btEdit* nevű, *Módosítás* feliratú gombot, és egy új Formot aminek *frmEdit* lesz a neve.

A *Módosítás* gomb *OnClick* eseményében hívjuk meg a *frmEdit* objektum *Show* eljárását. Ezen az új formon az adatkészlet egyetlen rekordját módosíthatjuk egy időben, ezt a következőképpen érjük el:

```
procedure TfrmMain.btChangeClick(Sender: TObject);
begin
  // megjelenítjük a módosító formot:
  frmEdit.Show;
end;
```

Az első form futtatási képe (13.23. ábra)

Szám	Vezetéknév	Keresztnev	Beosztás	Iroda	Bruttó bér	Prémium
4	BEDE	FERENC	TITKÁR	20	87000	8700
10	KAÁL	FRANCISKA	TITKÁR	30	90000	7200
1	KOVACS	PÉTER	TITKÁR	10	80000	12000

13.23. ábra Az indító Form futási képe

Szükségünk van egy adathalmazt meghatározó komponensre – ez legyen egy `OraQuery`. Az `OraQuery` objektum `SQL` tulajdonságába írjuk meg a lekérdezést:

```
SELECT D1.EMPNO, D1.FNAME, D1.SNAME, D1.JOB, D1.OFFNO, D1.SAL
FROM SCOTT.EmpData D1
ORDER BY D1.EMPNO
```

Generáljuk az `INSERT`, `UPDATE`, `DELETE` és `Refresh` műveletekhez tartozó kódot az `OraQuery` szerkesztőjének `SQL` oldalán (`Get Table Fields`, `Generate SQL`).

A rekord mezőinek megjelenítéséhez és módosításához helyezzünk fel `DBEdit`-et az `EMPNO` kivételével, ide helyezzünk `DBText` szövegmezőt (A felhasználó ezt nem változtathatja meg). Értékét az alkalmazásnak kell kiszámolnia (lásd. később).

Szükségünk lesz egy `DBNavigator` komponensre is a rekordok közti tallózáshoz, aminek csak a navigációs gombjait hagyjuk meg.

Kényelmetlen lenne, hogy módosításkor a `DBNavigator` gombjaival kelljen kiválasztani a rekordunkat, ezért helyezzünk fel még egy `Edit` szerkesztőmezőt is, azért, hogy ebben a dolgozók neve alapján tudunk pozicionálni az adatbázisban. A Form képe a 13.24. ábrán látható.

Szükség lesz még ezen kívül a Szerkesztés feliratú (a neve legyen `btEdit`), a Törlés feliratú (`btDelete`), és a Felvétel feliratú (`btInsert`) gombokra a feliratuk szerinti műveletekhez, valamint a szokásos `Ok` (`btOk`), `Mégse` (`btCancel`) és `Alkalmaz` (`btApply`) gombokra.

Alapállapotban állítsuk a `DBEdit` szerkesztők `ReadOnly` tulajdonságát `True` értékre, színét pedig `clBtnFace` színűre, így látszik, hogy csak-olvashatóak. Ezt a `btEdit` gombbal oldjuk fel. A vezérlők automatikusan meghívják az adatkészlet `Edit` eljárását, ezzel nem kell törödnünk.

#### Törlés:

A törlés az adatbázisok szempontjából egy nagyon érdekes, és fontos művelet. Soha nem tudhatjuk, hogy egy törölt adat nem lesz-e később fontos valamiért. Ebből az okból a valódi törlés helyett inkább egy "áltörlést", vagy "logikai törlést" szokás alkalmazni. Ez lehet egy másik táblába mozgatás, egy flag állítása stb. Ebben a példában az utóbbit alkalmazzuk, külön kialakított flag nélkül (erre a célra az `empno` mezőt használtuk). Ha az adatbázisban túl sok a felesleges adat, akkor véglegesen törölhetjük, vagy archiválhatjuk őket.

Legyen az adatbázis karbantartásának elve: Adatokat akkor törölünk az adatbázisból (vagy archiválunk), mikor az megtelik. Állapítsuk meg a maximális bejegyzések számát 5000-ben. Amíg az adattábla rekordjainak száma ezt el nem éri, addig a törölni kívánt adatokat ne távolítsuk el véglegesen, hanem `empno` mezőjük értékét növeljük 5000-rel. Fontos, hogy az ily módon logikailag törölt dolgozók számát ne kaphassák meg mások. Ennél a módszernél keletkezhetnek lyukak két dolgozói szám között. Egy-egy archiválásnál gondoskodnunk kell arról, hogy egy új dolgozó a legkisebb üres dolgozói számot kapja meg. Ehhez nemcsak az aktív dolgozókat kell megnéznünk (`empno < 5000`), hanem a törölteket is (`empno > 5000`).

A rekord törlését a `btDelete` gombbal valósítjuk meg, aminek `OnClick` eljárásában az `OraQuery` objektum `Delete` eljárását hívjuk meg. A fent ismertetett elv alapján módosítsuk a `OraQuery` objektum `SQLDelete` tulajdonságát a következőképpen:

```
UPDATE Scott.EmpData
SET EMPNO = EMPNO+5000
WHERE EMPNO = :EMPNO
```

#### Felvétel:

Az új rekord felvételére a `btInsert` gomb szolgál, annak `OnClick`-eseményéből meg kell hívnunk az `OraQuery1` nevű `OraQuery` objektum `Insert` eljárását, ki kell számolnunk a dolgozó leendő számát és beírni az `EMPNO` mezőhöz tartozó vezérlőbe. A kereséshez egy `OraTable` komponenst, a `Findkey` eljárást és a `samlalo` segédváltozót használunk, a következő módon:

```
procedure TfrmEdit.btInsertClick(Sender: TObject);
var
  szamlalo: Integer;
begin
  OraQuery1.Insert;
  szamlalo:=0;
  OraTable1.KeyFields:='empno';
  OraTable1.Open;
  repeat
    inc(szamlalo)
  until ( not (OraTable1.FindKey([szamlalo]))
    and not (OraTable1.FindKey([szamlalo+5000])) ) or (szamlalo=5000);
  OraTable1.Close;

  if szamlalo=5000
  then ShowMessage('Az adatbázis megtelt.')
  else
    begin
      DBText1.Field.Value:=szamlalo;
      ...
    end;
  ...
end;
```

EMPNO	FNAME	SNAME	SAL	JOB
5004	BEDE	FERENC	87000	TITKÁI

13.24. ábra A Módosítás elvégző Form

**Szerkesztés:**

Az adatok szerkesztését a `Name = btEdit` tulajdonságú gomb megnyomása után végezhetjük. Ez a gomb csupán az `OraQuery1` objektum `Edit` eljárását hívja.

Ez a programrészlet a `FindKey` eljárás segítségével megkeresi a legkisebb nem használt dolgozói számot úgy, hogy addig növeli 1-ről a `samlalo` értékét, míg nem talál egy olyan számot, amely nincs a rekordok `empno` mezőjében. Vizsgálja a törölt adatokat is, úgy hogy a `samlalo` 5000-rel emelt értékét is nézi. Ha eléri a maximális 5000 bejegyzést, szintén megáll a ciklus.

Ha a felhasználó végzett az adatbevitellel, mentheti azt a `btOk` gombbal. Ennek `OnClick` eljárásában végezzük el a kiválasztott rekord módosítását az `OraQuery1` objektum `Update` eljárás hívásával, és a Form bezárását. Ugyanezt teszi a `btApply` a form bezárása nélkül. A `btCancel` ugyan bezárja a formot, de a `Cancel` eljárást hívja meg, így a módosítások nem íródnak be az adatforrásba.

**Tallózás:**

Erre a `DBNavigator` gombjai alkalmasak, illetve a felhelyezett `Edit1` szerkesztőmező. Ehhez azonban az `Edit1` objektum `OnChange` eljárását módosítanunk kell a következőképpen:

```
procedure TfrmEdit.Edit1Change(Sender: TObject);
begin
  if Edit1.Text<>''
  then
    begin
      OraQuery1.FilterSQL:='fname LIKE '''+Edit1.Text+'%''';
      OraQuery1.Filtered:=True;
    end
  else
    begin
      OraQuery1.FilterSQL:='';
      OraQuery1.Filtered:=False;
    end;
end;
```

Ennek használatával közelítő keresést hajthatunk végre az adatkészleten.

A törölt munkavállalók közti tallózást is egyszerűen megvalósíthatjuk. A `Panel`-en elhelyezünk egy `DBGrid` komponenst, amelyben a törölt munkavállalók adatait jelenítjük meg. A legegyszerűbben az ilyen rekordokat egy `SmartQuery` komponenssel válogathatjuk ki, amelynek `SQL` tulajdonsága:

```
SELECT * FROM EmpData WHERE empno > 5000
```

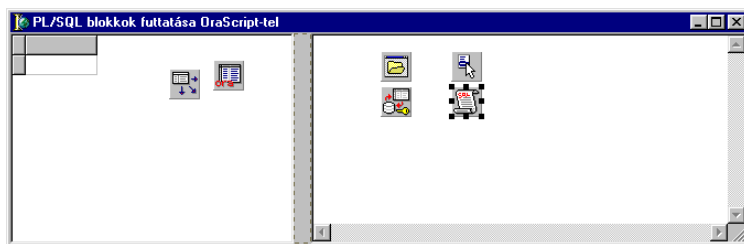
Egy gomb komponenssel változtatjuk a `Panel` komponens láthatóságát és növeljük/csökkentjük a Form magasságát. Így ha nem akarjuk a törölt adatokat látni, azok rejtve maradnak.

**Példa (93\_ora\_oracle2)**

(Egy PL/SQL blokk beolvasása és futtatása.)

Készítsünk olyan programot, amely lehetővé teszi tetszőleges a felhasználónak `DDL` parancsok írását és futtatását, PL/SQL fájl beolvasását, és a benne szereplő script program vagy PL/SQL blokk futtatását. A program megengedi, hogy a felhasználó módosítsa a beolvasott adatokat.

Készítsünk egy új alkalmazást. A látható komponensek közül egy `Memo` komponenst, és egy `DBGrid` komponenst helyezünk a Formra. A kettőt egy `Splitter` komponens választja el. A `Memo` komponensben szerkeszthetjük is a lekérdezésünket, PL/SQL blokkunkat, vagy megjeleníthetjük a beolvasott fájl tartalmát. Állítsuk a rács `Align` tulajdonságát `alLeft` értékre, a `Splitter` igazítása szintén `alLeft` értékű legyen, és a `Memo` objektum igazítása `alClient`-értékű legyen. Ebben az esetben a rács, illetve a `Memo` objektum mérete futás közben változtatható. A rácsban a létrehozott táblát jeleníthetjük meg. (13.25. ábra)

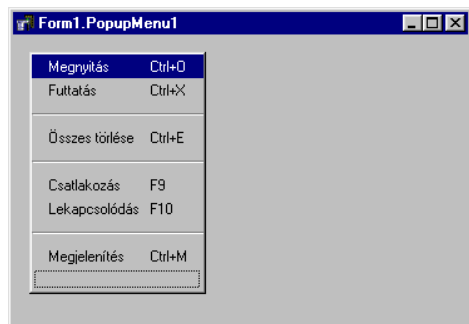


13.25. ábra A Form tervezési időben

A nem-vizuális komponensek az alkalmazásban az alábbiak: `OpenDialog`, `PopupMenu`, `OraSession` és `OraScript`, valamint egy `OraDataSource`, és egy `OraTable` komponens. Építsük fel az `OraSession` kapcsolatát, majd kapcsoljuk hozzá az `OraScript` komponenst. Szükséges még a `PopupMenu` a `Memo` komponenshez kapcsolása, valamint a `Memo`, esetleg a rács objektum `PopupMenu` tulajdonságát `PopupMenu1`-re állítani. A megjeleníteni kívánt objektum az `OraTable`, kapcsoljuk össze a rácsot és az `OraDataSource` objektumokat is.

Be kell állítanunk az `OpenDialog` állomán- megnyitási objektum `Filter` tulajdonságát is úgy, hogy csak `.sql` és `.txt` kiterjesztésű állományokat engedjünk beolvasni.

Milyen menüpontokra lesz szükségünk? (13.26. ábra) A Megnyitás menüpontra az állomány beolvasásához, a Futtatás menüpontra a `Memo` tartalmának futtatására, és az Összes törlése menüpontra a `Memo` tartalmának ürítésére. Vegyük még ide a Csatlakozás és a Lekapcsolódás pontokat is, a szerverhez való csatlakozáshoz, és Megjelenítés menüpontot a létrehozott tábla megjelenítéséhez.



13.26. ábra A Form repülőmenüje

A menüpontokat a `PopupMenu` szerkesztőjébe írhatjuk (vagy az `Items` tulajdonságához az `Object Inspector`-ban), ott definiálhatunk hozzájuk gyorsbillentyűket is. Duplán kattintva a menüpontra előbukkan annak `OnClick` eseménye.

A Megnyitás menüpontban egyszerű fájlkezelést végzünk, az állomány sorait átmásoljuk a `Memo` komponensbe (az `x` segédváltozó segítségével):

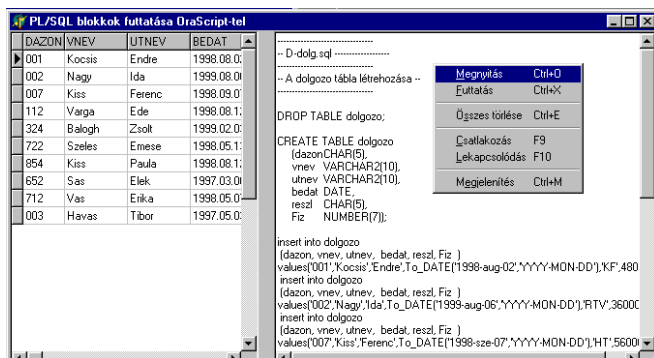
```
procedure TForm1.OpenScript(Sender: TObject);
var
  F: TextFile;
  x: string;
begin
    // "megnyitás" menüpont:
    if OpenDialog1.Execute
    then
        begin
            // hozzárendeljük a fájlt az F változóhoz:
            AssignFile(F, OpenDialog1.FileName);
            Reset(F) ;           // megnyitjuk a Fájlt-et:
            Memo1.Lines.Clear;    // töröljük a Memo-t:
            While not Eof(F) do
            begin
                ReadLn(F, x);     // beolvassuk a tartalmát a Memo-ba:
                Memo1.Lines.Add(x);
            end;
            CloseFile(F);         // bezárjuk a fájlt:
        end;
end;
```

A futtatás hasonlóan egyszerű műveleteket igényel; a `Memo` objektum `Lines` tulajdonságát egyenlővé kell tennünk az `OraScript` objektum `SQL` tulajdonságával, majd futtatnunk kell hibaellenőrzéssel.

A csatlakozás és lekapcsolódás menüpontok mindössze az `OraSession` objektum `Connect` és `Disconnect` eljárásait hívják meg.

Futtassunk egy PL/SQL blokkot!

Futtassuk programunkat, majd a `Memo` objektum területén a jobb egérgombbal kattintva válasszuk az előbukkanó menüből a Megnyitás menüpontot. Válasszuk az alkalmazás könyvtárában lévő `d-dolg.sql` nevű fájlt. A párbeszédablak `OK` feliratú gombjára kattintva megjelenik a fájl tartalma a `Memo`-ban, ezt már tudjuk módosítani és futtatni (csatlakozni tudunk a lekérdezés futtatásakor, vagy a Csatlakozás menüponttal előtte). Ez a blokk létrehoz egy táblát úgy, hogy előtte törli azt, és beszúr néhány új rekordot. Ha még nem létezik a tábla, akkor az Oracle hibaüzenetet küld.



13.27. ábra Az Orascrip program futási képe

A futtatási kép a 13.27. ábrán látható.

A Megjelenítés menüponthoz tartozó program:

```
procedure TForm1.Megjelenites1Click(Sender: TObject);
begin
    Oratable1.Close;
    Oratable1.TableName := 'dolgozo';
    Oratable1.Active := True;
end;
```

Sem a BDE, sem az ADO nem rendelkezik script futtatására alkalmas komponenssel.

### Példa (94\_ora\_oracle)

(Kurzorkezelés)

A dolgozó táblát bővítjük egy új, oszloppal, amelynek neve legyen `partner`. Futtassunk egy olyan PL/SQL blokkot, amely blokk a felhasználó által megadott részleg minden dolgozójának a partner nevű oszlopába beírja egy olyan azonos részlegbeli kollégájának a nevét, akinek fizetése bizonyos összeggel tér el az övétől, és hozzá partnerként még nincs senki sem párosítva. (Természetesen saját maga senkinek se lehessen partnere.)

A részleg nevét és a fizetések eltérésének mértékét, mint paramétert, a felhasználó adja meg. Jelenítsük meg az eredményt egy adatrácsban.

Készítsünk egy új alkalmazást. A feladat futási formja a 13.28. ábrán látható.

Helyezzünk a Formra egy `OraSession` komponenszt a csatlakozáshoz, két `OraSQL` komponenszt, egyet az oszlopok hozzáfűzéséhez, egyet pedig a partnerek kereséséhez. A tábla megjelenítését egy `OraQuery`, egy `OraDataSource` és egy `DBGrid` végezze. A végrehajtandó utasításokat nyomógombokkal indítsuk.

A `ComboBox` kiválasztott elemével tudjuk kiválasztani és megadni a kereső algoritmusnak a kívánt részleg nevét. A `ComboBox1` lista feltöltését tervezési időben végezzük, az `Items` tulajdonság beállításával.

Az első nyomógomb `Oszlop hozzáfűzése` feliratú (`Button1`) az `OraSQL1` objektum SQL utasítását hajtja végre, tehát hozzáfűzi a `partner` oszlopot a `dolgozo` táblához.

Az `OraSQL1` objektum SQL tulajdonsága:

```
ALTER TABLE dolgozo ADD partner VARCHAR2(10) DEFAULT '---'
```

A `Button2` `Partner keresése` feliratú gomb `OnClick` eseményében paraméterként (`reszl`) átadjuk a `ComboBox1` objektumban kiválasztott részleg nevét az `OraSQL2`-nek, és a partnerek fizetése közti maximális különbséget (`ber`). Mivel utóbbit egy `Edit` szerkesztőmezőben kérjük be, hibellenőrzést kell végeznünk; hibás érték esetén 10000 értékre állítsuk be a paramétert:

```
procedure TForm1.Partner_keresoClick(Sender: TObject);
beginbegin
    OraQuery1.Close;
    // megadjuk az algoritmus számára szükséges két paramétert. A részleg
    // nevét hordozó paraméter értéke nem lehet hibás, mivel egy ComboBox-
    // ből választjuk ki...:
    OraSQL2.ParamByName('reszl').Value :=
        ComboBox1.Items[ComboBox1.ItemIndex];
    TRY
        // viszont a bért ellenőriznünk kell. Ha rossz a paraméter,
        // beállítjuk a 10000 alapértelmezett értéket:
        OraSQL2.ParamByName('ber').Value:=StrToInt(Edit1.Text);
    EXCEPT
```

```

OraSQL2.ParamByName('ber').Value:=10000;
Edit1.Text:='10000';
end;
// futtatjuk az OraSQL2-t, ami megkeresi a partnereket:
OraSQL2.Execute;           // és végrehajtja a DML utasításokat
OraQuery1.Open;
end;

```

Az Alapállapot gomb visszaállítja a partner oszlop alapértékét, ha újra akarjuk futtatni a partnerkereső PL/SQL blokkot. Az Alapállapot visszaállításához a Formra helyeztünk egy SmartQuery komponenst. Ennek SQL tulajdonságába írtuk bele az alapállapothoz módosító utasítást, amelyet azután az ExecSql eljárással hajtunk végre.

```

procedure TForm1.AlapallapotClick(Sender: TObject);
begin
  OraQuery1.Close;
  // A partner oszlopot alapállapotba állítja vissza
  SmartQuery1.SQL.Clear;
  SmartQuery1.SQL.Append('Update dolgozo SET partner = ''---''');
  SmartQuery1.ExecSQL;           // DML utasítások végrehajtása
  // Megjeleníti a módosított lekérdezést, a saját előre beállított
  // lekérdezésnek megfelelően.
  OraQuery1.Open;
end;

```

DAZON	VNEV	UTNEV	BEDAT	RESZL	FIZ	PARTNER
001	Kocsis	Endre	1998.08.02.	KF	48000	---
002	Nagy	Ida	1999.08.06.	RTV	36000	---
007	Kiss	Ferenc	1998.09.07.	HT	56000	---
112	Varga	Ede	1998.08.12.	RTV	72000	Balogh
324	Balogh	Zsolt	1999.02.03.	RTV	65000	Varga
722	Szeles	Emese	1998.05.13.	HT	68000	---
854	Kiss	Paula	1998.08.12.	HT	42000	---
652	Sas	Elek	1997.03.06.	RTV	70000	---
712	Vas	Erika	1998.05.07.	FF	66000	---

13.28. ábra A partnerkereső program formja

Az OraSQL2 egy PL/SQL blokkot tartalmaz. A eredeti feladat megoldása az Oracle PL/SQL blokkban az alábbi listán látható. A programban a nyomtatási programsorok itt megjegyzésként jelennek meg.

```

DECLARE
  v_reszlegnev      dolgozo.reszl%TYPE;
  v_dolg_partner    dolgozo.partner%TYPE;
  v_part_partner    dolgozo.partner%TYPE;
  CURSOR dolgozo_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
    SELECT vnev, fiz, ROWID sorazonosito
    FROM dolgozo
    WHERE (reszl = p_reszleg) and
          (partner = '---')
  FOR UPDATE OF fiz NOWAIT;
  dolgozo_rekord    dolgozo_kurzor%ROWTYPE;
  CURSOR partner_kurzor (p_reszleg dolgozo.reszl%TYPE) IS
    SELECT vnev, fiz, ROWID sorazonosito
    FROM dolgozo
    WHERE (reszl = p_reszleg) and
          (partner = '---')
  FOR UPDATE OF fiz NOWAIT;
  partner_rekord    dolgozo_kurzor%ROWTYPE;

BEGIN

```

```

v_reszlegnev := UPPER(:reszl);                                // részleg paraméter
OPEN dolgozo_kurzor(v_reszlegnev);
LOOP
    FETCH dolgozo_kurzor
    INTO dolgozo_rekord;
    EXIT WHEN dolgozo_kurzor %NOTFOUND;
    --DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||' ciklusa: ');
    SELECT partner
    INTO v_dolg_partner
    FROM dolgozo
    -- WHERE CURRENT OF dolgozo_kurzor; -- Csak a ROWID használható!!!
    WHERE ROWID = dolgozo_rekord.sorazonosito;
    IF v_dolg_partner = '---'
    THEN
        OPEN partner_kurzor(v_reszlegnev);
        LOOP
            FETCH partner_kurzor
            INTO partner_rekord;
            EXIT WHEN partner_kurzor %NOTFOUND;
            SELECT partner
            INTO v_part_partner
            FROM dolgozo
            WHERE ROWID = partner_rekord.sorazonosito;
            IF (partner_rekord.sorazonosito <>
                dolgozo_rekord.sorazonosito) and
                ((dolgozo_rekord.fiz - partner_rekord.fiz)
                 BETWEEN -:ber and :ber)           // bér paraméter          and
            (v_part_partner = '---')
            THEN
                --DBMS_OUTPUT.PUT_LINE(dolgozo_rekord.vnev||' -
                '||partner_rekord.vnev);
                UPDATE dolgozo
                SET partner = partner_rekord.vnev
                WHERE CURRENT OF dolgozo_kurzor;
                UPDATE dolgozo
                SET partner = dolgozo_rekord.vnev
                WHERE CURRENT OF partner_kurzor;
                EXIT;
            END IF;
        END LOOP;
        CLOSE partner_kurzor;
    END IF;
    -- DBMS_OUTPUT.PUT_LINE('----- '||dolgozo_rekord.vnev||
    -- ' ciklusának vége');
END LOOP;
CLOSE dolgozo_kurzor;
COMMIT;
END;

```

# Tárgymutató

- 255  
 -- 259  
 # 342  
 \$ 342  
 % 266  
 %FOUND 425  
 %ISOPEN 425  
 %NOTFOUND 425  
 %ROWCOUNT 425  
 %ROWTYPE 391  
 %ROWTYPE 416  
 %TYPE 391  
 %TYPE 416  
 & 318, 562  
 && 318, 394  
 \* 242  
 \*\* (PL/SQL) 398  
 / 242, 255  
 /\* \*/ 259  
 : (PL/SQL) 393  
 : (Delphi) 512  
 ; 255  
 @ 236, 258, 260  
 @@ 260  
 [ ] 233  
 \_ 266, 342  
 {} 233  
 | 233  
 || 248  
 „ ” 244  
 ... 233  
 0NF 219  
 1NF 52, 220  
 2NF 52, 221  
 3NF 52, 222  
 4GL 57  
 a[ppend] 256, 258  
 ABC-elemzés 59  
 ACCEPT 323, 394  
 adatbányászat 69  
 adatbázis  
   ~ adminisztrátor 375  
   deduktív ~ 69  
   felügyelő 46  
   ~ modell 118  
   ~ objektumok 342  
   objektumorientált ~ 67  
   osztott ~ 58  
   ~ok összekapcsolása 71  
   ~ séma 118  
   ~ felhasználói nézete 46  
   ~ fizikai modell 45  
   ~ implementációs modell 45  
   ~ konceptuális modell 45  
   ~ minivilág 45  
   ~ szövetség 71  
   ~ tranzakciók 337

adatdefiniáló nyelv (DDL) 46, 241  
 adatfüggetlenség  
   fizikai ~ 46  
   logikai ~ 46  
 adatkezelő nyelv (DML) 48, 241, 330  
 adatkinyerő 71  
 adatleíró nyelv (DDL) 46  
 adatlekérdező nyelv (DQL) 241  
 adatmanipuláló nyelv (DML) 48  
 adatszótár 238, 342  
   lekérdezése 343  
 adattárház 71  
 adattípus (PL/SQL)  
   egyszerű ~ 388  
   összetett ~ 415  
 adattípus (SQL) 239, 343  
 adatvédelem  
   algoritmikus 60  
   fizikai 60  
   ügymiteli 60  
 adatvezérlő nyelv (DCL) 242, 375  
 ADD 345, 346, 361  
 ADD\_MONTHS 280  
 afiedt.buf 255  
 alapmodell 217  
 alaptábla 364  
 algoritmikus védelem 60  
 alias név 149  
 ALL 314  
 all\_ 343  
 all\_synonyms 374  
 all\_views 364  
 allekérdezés 310  
   egysoros ~ 311  
   ~ a FROM utasításrészben 316  
   korrellált ~ 317  
   többsoros ~ 311, 313  
 állomány 28  
 állománystruktúrák 30  
   direkt ~ 40  
   hálós ~ 37  
   hierarchikus ~ 34  
   indexelt ~ 38  
   szekvenciális ~ 30  
 alprogram 386, 465  
   ~ deklarációja 471  
   ~ elődeklarációja 472  
   rekurzív ~ 479  
   tárolt ~ 475  
 alséma 46  
 ALTER 241  
 ALTER SESSION 232  
 ALTER TABLE 345  
 ALTER TABLE 361  
 ALTER USER 376  
 AND 267

- anomáliák 226
  - bővítési ~ 227
  - módosítási ~ 226
  - törlési ~ 227
- ANY 314
- argumentum 75
- Armstrong-bővítés 201
- Armstrong-szabályok 199
- ARRAY[SIZE] 325
- AS 242
- ASC 270
- asszociativitás 139
- Átírányítás menüpont 262
- átnevezés 149, 350
- atomok 55
- attribútum 110
  - elsődleges ~ 210
  - másodlagos ~ 210
- attribútumvektor 110
- AUTOCOMMIT 338
- AVG 284
- azonosítók (PL/SQL) 397
- B+-fa 38
- BACKUP ANY TABLE 375
- BCNF 223
- BEGIN...END 384
- belső függvények (PL/SQL) 398
- belső mutató eljárások
  - hálós struktúrák 37
  - hierarchikus struktúrák 34
- beszúrás 330
- BETWEEN...AND 264
- B-fa 36
- BFILE 343, 390
- bináris fák 35
- bináris keresés 31
- BINARY\_INTEGER 389, 421
- biztos kifejezések 55
- BLOB 343, 390
- blokk 384, 385, 456
  - nevesített ~ 386
  - névtelen ~ 386
  - ~ szerkezete 384
  - ~ típusok 385
- BOOLEAN 389
- borítékoló 72
- Boyce–Codd normálforma (BCNF) 52, 223
- bővítés
  - ekvivalens ~ 195
  - halmaz ~ 76
  - logikai struktúra ~ 195
- BRE[AK] 289, 326
- BTI[TLE] 326
- burokséma 197
- burokséma-halmaz 197
- burokstruktúra 196
- c[hange] 256
- CASCADE 355, 361
- CASCADE CONSTRAINTS 381
- CHAR 343, 389
- CHECK 352, 356
- ciklus 405, 409
- cylinderindex 39
- cl[ear] 256
- CLEAR 327
- CLEAR BREAKS 290
- CLOB 343, 390
- CLOSE kurzornév 427, 430
- CODASYL bizottság 50
- COLSEP 325
- COLUMN 326, 327
- COMMENT 351
- COMMIT 241, 337, 400
- CONCAT 276
- CONNECT 234, 377, 379
- CONSTANT 388
- CONSTRAINT 353
- COUNT 284, 422
- CREATE 241
- CREATE FUNCTION 476
- CREATE INDEX 372
- CREATE PROCEDURE 377, 476
- CREATE ROLE 378
- CREATE SESSION 377
- CREATE SYNONIM 373
- CREATE TABLE 252, 344, 377
- CREATE USER 375
- CREATE VIEW 365, 377
- CURRENT OF 441
- CURSOR...IS 428, 438
- csaknem fizikai szekvenciális struktúrák 33
- csomag 386
- csoportfüggvény 284, 368
- Data Control Language *lásd* DCL
- Data Definition Language *lásd* DDL
- Data Manipulation Language *lásd* DML
- Database Administrator 375
- DATE 240, 343, 389
- dátum 279
  - ~ formátum 232
  - ~ formátummaszk 281
  - ~kezelő függvények 280
- DBA 375
- dba\_ 343
- DBMS\_OUTPUT 326, 395
- DBMS\_OUTPUT.PUT\_LINE 396
- DBTG (Data Base Task Group) 50
  - jelentés 50, 52
- DCL (Data Control Language) 56, 242, 375
- DDL (Data Definition Language) 56, 241, 342, 344
- deadlock 47
- DEC 389
- DECIMAL 389
- DECLARE 384, 388
- deduktív adatbázisok 68
- DEF[INE] 323, 326, 394
- DEFAULT kifejezés 252

- deklarációs szegmens 384, 387
- deklaratív nyelv 48
- dekompozíció 52, 152, 211, 212
  - függőségörző ~ 214
  - veszteségmentes ~ 212
- del 256
- DELETE 241, 335, 401, 422
- Delphi 483
  - & 562
  - .db 504
  - .dbf 504
  - .dfm 505
  - .dpr 505
  - .mdb 531
  - .pas 505
  - .res 505
  - .sql 518
  - : 512
  - Access adatbázis-kezelő program 485
  - Active tulajdonság 490
  - ActiveX Data Objects 485
  - adatbázis
    - adminisztrátor 487
    - elérés Delphiben 492
    - kezelő komponensek 488
    - lokális ~ elérése 492
    - lokális ~ kezelése 504
    - távoli ~ elérése 486
  - Add eljárás 510
  - Add Fields menüpont 517
  - AddAlias eljárás 485, 489, 540
  - Additional komponenspaletta 512
  - AddParameter eljárás 533
  - AddStandardAlias 485, 489
  - AddStrings eljárás 573
  - AddWhere eljárás 561
  - ADO 485
    - komponenspaletta 531
    - OLE DB kapcsolat 502
    - Oracle kapcsolat 544
  - ADOCommand komponens 532, 546
  - ADOConnection komponens 532, 544
  - ADODataset komponens 532, 545
  - ADOQuery komponens 546
  - ADOStoredProc komponens 546
  - ADOTable komponens 546
  - alias név létrehozása 493
  - AliasName tulajdonság 489, 537
  - Align tulajdonság 508
  - Alignment tulajdonság 518
  - Append eljárás 528, 589
  - Application objektum 504
  - ApplyUpdates eljárás 543
  - Assign eljárás 520
  - AssignFile eljárás 521
  - AsString tulajdonság 513
  - AutoEdit tulajdonság 564
- Delphi (folytatás)
  - BDE 485
    - felület 492
    - komponenspaletta 489
      - ODBC kapcsolat 497
      - SQL links kapcsolat 494
  - BDESession komponens 561
  - BitBtn komponens 512
  - BOF függvény 524
  - Break utasítás 524
  - BreakExec eljárás 562
  - Button komponens 514, 540
  - CacheUpdates tulajdonság 543
  - CancelUpdates eljárás 543
  - Caption tulajdonság 505
  - CheckBox komponens 509
  - Checked tulajdonság 549
  - Clear eljárás 520
  - CliensDataSet komponens 490
  - Close eljárás 510
  - CloseFile eljárás 521
  - Color tulajdonság 518
  - Columns tulajdonság 517
  - ComboBox komponens 579
  - CommandText tulajdonság 532, 545
  - CommandType tulajdonság 532, 545
  - Connect eljárás 565
  - ConnectDialog komponens 560
  - Connected tulajdonság 489
  - Connection tulajdonság 546
  - ConnectionTimeout tulajdonság 545
  - ConnectPrompt tulajdonság 560
  - ConnectString tulajdonság 545
  - Create eljárás 579
  - CreateTable eljárás 516
  - CurrentRowSelected tulajdonság 523
  - CursorLocation tulajdonság 545
  - Data Access komponenspaletta 490
  - Data Controls komponenspaletta 491
  - Database komponens 489
  - DataBaseName tulajdonság 489, 537
  - DataField tulajdonság 491, 508, 542
  - DataSet tulajdonság 489, 490
  - DataSetProvider komponens 491
  - DataSource komponens és tulajdonság 490, 532
  - DataType tulajdonság 513
  - dBase adatbázis-kezelő program 485
  - DBChart komponens 492
  - DBCheckBox komponens 492
  - DBComboBox komponens 492
  - DBDEMOS Delphi adatbázis gyűjtemény 504
  - DBEdit komponens 491
  - DBGrid komponens 491
  - DBImage komponens 491
  - DBListBox komponens 491
  - DBMemo komponens 491
  - DBNavigator komponens 491
  - DBRadioGroup komponens 492

## Delphi (folytatás)

- DBRichEdit komponens 492
- DBText komponens 491
- Debug tulajdonság 567
- Delete eljárás 543
- DeleteAlias eljárás 489, 541
- Dialogs komponenspaletta 519
- Direction tulajdonság 534
- DisableControls tulajdonság 524
- Disconnect eljárás 578
- Edit eljárás 527
- Edit komponens 509
- EnableControls tulajdonság 524
- Enabled tulajdonság 542
- EOF függvény 521
- események (Events) 489
- Events (események) 489
- ExecSQL eljárás 490
- Execute eljárás 490, 534
- Exists eljárás 516
- FieldDef tulajdonság 516
- FieldName tulajdonság 518
- Fields Editor menüpont 517
- FieldValues tulajdonság 528
- FileDialog komponens 519
- FileName tulajdonság 521
- Filter tulajdonság 521, 546
- Filtered tulajdonság 546
- FilterSQL eljárás 561
- Finally 521
- FindKey eljárás 561
- FindNearest eljárás 561
- First eljárás 524
- Font tulajdonság 518
- Form ablak 504
- Free eljárás 541
- GetAliasNames eljárás 540
- GetTableNames eljárás 549
- Hint tulajdonság 508
- IDAPI32 493
- IndexFieldName tulajdonság 545
- InputBox függvény 523
- Insert eljárás 543
- Interbase adatbázis-kezelő program 485
- IntToStr függvény 529
- ItemIndex tulajdonság 541
- Items tulajdonság 541
- KeyFields tulajdonság 583
- Kind tulajdonság 512
- komponenspaletta 489
  - Additional 512
  - ADO 531
  - BDE 489
  - Data Access 490
  - Data Controls 491
  - Dialogs 519
  - ODAC 560
  - Oracle Access 559

## Delphi (folytatás)

- komponenspaletta (folytatás)
  - Standard 507
  - System 557
- Label komponens 507
- LabelSet tulajdonság 575
- Last eljárás 525
- Lines tulajdonság 520, 586
- ListBox komponens 540
- Locate tulajdonság 557, 571
- lokális adatbázis 492
- MacroByName tulajdonság 580
- Macros tulajdonság 562
- MDAC 488
- Memo komponens 519
- MessageDlg függvény 528
- MoveBy eljárás 525, 531
- Name tulajdonság 513
- natív driver 494, 539
- natív driver – Oracle kapcsolat 539
- NeverConnect tulajdonság 560
- New Fields menüpont 517
- Next eljárás 524
- Object Inspector 489
- OCI.DLL 495, 503
- ODAC 503
  - komponensek 559
  - komponenspaletta 560
    - Oracle kapcsolat 559
- ODBC 486, 497
  - administrator 499
  - Microsoft ODBC for Oracle 498
    - Oracle kapcsolat 536
  - Oracle ODBC driver 498
- OLE DB 488, 502
  - Microsoft OLE DB Provider for Oracle 544
  - Oracle Provider for OLE DB 544
- OnCalcFields esemény 517
- OnChange esemény 541, 555
- OnClick esemény 510
- OnCreate esemény 508
- OnEnter esemény 559
- OnStateChange esemény 491, 565
- Open eljárás 490
- OpenDialog komponens 521
- Options tulajdonság 560
- Oracle Access komponenspaletta 559
- Oracle adatbázis-kezelő program 486
- Oracle Developer 483
- OraDataSource komponens 563
- OraQuery komponens 562
- OraScript komponens 563
- OraSession komponens 560
- OraSQL komponens 563
- OraStoredProc komponens 563
- OraTable komponens 561
- OraUpdateSQL komponens 563
- Panel komponens 509

## Delphi (folytatás)

- Panels tulajdonság 559
- Paradox adatbázis-kezelő program 485
- ParamByName tulajdonság 513
- Paraméter Szerkesztő 514
- Parameters tulajdonság 533
- ParamList eljárás 541
- Params tulajdonság 490, 513
- ParamType tulajdonság 513
- ParamValues tulajdonság 534, 552
- Password tulajdonság 565
- PickList tulajdonság 518
- PopupMenu komponens 585
- Pos függvény 524
- Post eljárás 528
- Prepare eljárás 513
- Prior eljárás 525
- Properties (tulajdonságok) 489
- Query komponens 489
- RadioButton komponens 509
- ReadOnly tulajdonság 564
- RecNo tulajdonság 530
- Refresh eljárás 523
- rekord megjelenítése 507
- Reset eljárás 521
- RestoreSQL eljárás 561
- scott/tiger 497
- SelectedRows tulajdonság 523
- Sender paraméter 506, 559
- Session komponens 489
- SetFocus eljárás 520
- SetOrderBy eljárás 561
- Show eljárás 581
- ShowHint tulajdonság 508
- ShowMessage eljárás 524
- Size tulajdonság 513
- SmartQuery komponens 563
- Splitter komponens 515
- SQL Links 486, 494
- SQL Server adatbázis-kezelő program 486
- SQL tulajdonság 490
- SQLDelete tulajdonság 562
- SQLInsert tulajdonság 562
- SQLUpdate tulajdonság 562
- Standard komponenspaletta 507
- Statusbar komponens 559
- StoreDef tulajdonság 516
- StoredProcedure komponens 490
- String List Editor 509
- StrToInt függvény 528
- Sybase adatbázis-kezelő program 486
- System komponenspaletta 557
- system/manager 497
- tábla állapotmegfigyelése 504
- tábla megjelenítése 504
- TableName tulajdonság 489
- Table komponens 489
- TableName tulajdonság 537

## Delphi (folytatás)

- távoli adatbázis 486
- TDataSet 489
- Text tulajdonság 528, 551
- TextFile adattípus 521
- Timer komponens 557
- Title tulajdonság 518, 579
- Try...Except...End kivételkezelés 528
- Try...Finally...End kivételkezelés 521
- tulajdonságok (properties) 489
- UDA 488
- UnPrepare eljárás 513
- UpdateObject tulajdonság 542
- UpdateSQL komponens 490, 542
- Uppercase tulajdonság 550
- UserName tulajdonság 564
- Value tulajdonság 513
- VirtualTable komponens 563
- VisibleButtons tulajdonság 511
- dept tábla 238
- DESC 239, 270
- Descartes-szorzás 95, 96, 153, 303
- digitális kézjegy 64
- direkt állományszervezés 40
- DISABLE CONSTRAINT 362
- DISTINCT 250, 273, 368
- DML (Data Manipulation Language) 48, 56, 241, 330, 368
- Dom függvény 55
- DQL 241, 400
- DROP 241, 254, 347, 361
- DROP ANY TABLE 349, 375
- DROP COLUMN 347
- DROP INDEX 373
- DROP SYNONYM 374
- DROP TABLE 254, 349
- DROP USER 375, 382
- DROP VIEW 368
- DUAL 232, 278
- DUBLE PRECISION 389
- DUMMY 278
- ed[it] 236, 255, 258
- EER modell 65
- egyed (entity) 28, 49
- egyedi érték 352
- egyesítés 78, 82, 91, 296
- egysoros függvények 275
- ekvivalencia 186, 187, 189
- ekvivalencia-reláció 142
- elem
  - egységelem 140
  - egyszerű ~ 90
  - összetett ~ 90
- eljárás 386, 465
- ellentmondás-mentesség 337
- előrehivatkozás 472
- elsődleges példány módszer 59
- emp tábla 234, 237

ENABLE CONSTRAINT 362  
 END 264, 384  
 ER modell 49  
 eredménytábla 242  
 erős konzisztencia 59  
 értékadás 392, 417  
 érzékeny szegmens 49  
 érzékenységelemzés 59  
 EXCEPTION 456  
 EXISTS 314, 422  
 exit 258  
 EXIT 409  
 EXIT WHEN 409  
 EXTEND 422  
 Fájl menüpont 260  
 FALSE 268, 390  
 FEED[BACK] 326  
 felbontás *(lásd dekompozíció)*  
 felhasználó-azonosítás 63  
 felosztás  
     belső ~ 177  
     egység ~ 147  
     elemi-séma ~ 147  
     függősségi ~ 177  
     halmaz ~ 143, 145  
     külső ~ 191  
     ortogonális ~ 191  
     reláció ~ 146  
     séma ~ 146  
 Felső-N analízis 370  
 feltétel 262  
 FETCH...INTO 392, 427, 430  
 FIRST (PL/SQL) 422  
 fizikai  
     adatfüggetlenség 46  
     modell 45  
     szekvenciális állománystruktúrák 30  
     védelem 60  
 FLOAT 389  
 fogalomhierarchia 206  
 FOR 411  
 FOR UPDATE 440  
 FOR[MAT] 323, 327  
 FORCE 365  
 fordítási szakadék 69  
 FOREIGN KEY 352, 355  
 formátumkód 233  
 formátummaszk 281  
 formázási utasítások (SQL\*Plus) 326  
 formula 55  
 forrás-nyelvi elemzés 59  
 főindex 39  
 FROM 242  
 FUNCTION 468  
 függőség 173  
     ~ felosztása 208  
     belső ~ 211  
     ~ család 202  
     ~ egyesítés 208

függőség (folytatás)  
     egység~ 174  
     egyszerű ~ 174  
     funkcionális ~ 174, 181  
     hatvány~ 173  
     összetett ~ 174  
     ~ magja 174  
     speciális ~ 175  
     ~ szorzás 208  
     teljes ~ 210  
     tranzitív ~ 210  
     triviális ~ 174  
     zérus~ 174  
 függőség-generátor 204  
     irredundáns ~ 205  
     minimális ~ 205  
 függvény (PL/SQL) 386, 468  
 függvény 118  
     bijektív ~ 122  
     egyszerű ~ 121  
     halmaz~ 134  
     hipervektor~ 132  
     komponenshalmaz ~ 85, 94  
     vektor~ 129, 176  
     vektorhalmaz~ 135  
 gazdanyelv (Host Language) 48  
 generalizáció 66  
 get 258  
 GRANT 242, 377, 379  
 GROUP BY 287, 368  
 gyenge konzisztencia 59  
 gyűjtőtábla (PL/SQL) 415, 420  
     ~ hivatkozás 421  
     ~ metódusok 422  
     rekord típusú ~ 423  
 halmaz  
     hatvány~ 76  
     hiper~ 91  
     mono~ 79  
     multi~ 78  
     rész~ 76  
     üres ~ 76, 110  
     ~rendszer 90  
 halmazműveletek 296  
 halmazvektor 92  
 hálós állománystruktúrák 37  
 hashing algoritmus 40  
 hasonló  
     ~ rekordok 100, 110  
     ~ vektorhalmazok 103, 110  
     ~ vektorok 87, 109  
 hasonlóság 109  
 hatványozás 98  
 HAVING 287, 295  
 HEA[DING] 326, 327  
 HIDE 323  
 hierarchikus állománystruktúrák 34-37  
 holtpon 47  
 hozzáférés-védelem 65

- hozzárendelt változó 392
- i[ntput] 256
- idegen kulcs 352
- IDENTIFIED BY 376
- IF 406
- igazítás 243
- implementációs modell 45
- IN 265, 314, 387, 465, 468, 476, 479
- IN OUT 387, 465, 468, 478, 479
- index 83, 342, 371
  - felhasználói ~ 372
  - ~ létrehozás 372
  - ~ törlése 373
- indexelés
  - indexszekvenciális szervezés 38
  - ritka ~ 38
  - sűrű ~ 38
- indexelt állománystruktúrák 38
- infix jelölés 137
- inicializált változó 478
- INITCAP 275
- INLINE nézet 311, 369
- INSERT 241, 253, 330, 401
- INSTR 276
- INT 389
- INTEGER 389
- INTERSECT 296
- INTO 253
- invertálhatóság 140
- inverz 120, 121, 130
- IS 465
- IS NULL 267
- IS RECORD 416
- jelszó 231, 376
- jogosultság 375
  - ~ ellenőrzése 380
  - futtatási ~ 379
  - objektumkezelési ~ 379
  - rendszer~ 375
  - felhasználói ~ 376
  - ~ visszavonása 381
- join 155
  - natural ~ 154
- joker karakter 266
- journal állományok 47
- JUSTIFY 327
- kapcsolatok (relationship) 49
- karakterkezelő függvények 275
- karakterlánc 248
- kép
  - ~ elem képhalmaza 120, 127, 132, 134, 135
  - függőség ~-e 174
- keresés 29
  - bináris 31
  - kupacos 33
  - logaritmikus 31
  - Peterson-féle 31
- keret
  - parciális ~ 106
  - reláció ~-e 113
  - relációséma ~-ei 111
  - teljes ~ 106
  - vektorhalmaz ~-ei 106
- kifejezés 242
- kiválasztás 150, 405
- kivétel 458
  - ~ definiálása 457
  - ~ fajtái 457
  - ~ működése 458
  - ~ továbbadása 464
  - ~kezelő szegmens 385, 456
- kivonás 296
- kliens-szerver architektúrák 57
- koherencia 59
- kommutativitás 140
- kompatibilitás 173, 175, 184, 185, 186
- komponens 83
- kompozíció 164
- konceptuális modell 45
- konkurens folyamatok 46
- konvencionális kódolás 61
- konverzió 390
- konverziós függvények 281
- konzisztens 337
- kölcsönös kizárás 47
- következtető rendszer 69
- közös gyök 87
- központi zárellenőrzés 59
- központosított protokollok 59
- közvetítés 72
- kulcs 52, 194, 351, 352
  - elsődleges ~ 194
  - generálás 62
  - gondozás 62
  - idegen ~ 194
  - ~jelölt 194
  - kiosztás 62
  - ~ meghatározása 201
  - szuper~ 195
  - tárolás 63
- kupacos keresés 33
- kurzor 424
  - ~ allelkérdezéssel 444
  - ~ bezárása 430
  - ~ deklarációja 428
  - explicit ~ 425, 427
  - ~ FOR ciklusban 436
  - ~ függvények 425, 431
  - implicit ~ 425
  - ~ kapcsolata rekorddal 432
  - ~ megnyitása 429
  - paraméterezett ~ 437
- különbség-halmaz 77, 80, 82
- különbségi állományok 47

- külső mutatós eljárás
  - hálós struktúrák ~-ai 37
  - hierarchikus struktúrák ~-ai 35
- l[ist] 256
- láncolás 89
- LAST (PL/SQL) 422
- LAST\_DAY 280
- leképezés 118, 120
  - bináris vektor~ 125
  - egyrétegű vektor~ 125
  - egyszerű ~ 119
  - elem ~-e 127
  - halmaz~ 132, 133
  - hipervektor~ 130
  - közvetlen 40
  - n*-változós vektor~ 125
  - összetett ~-ek 123
  - reláció ~-e 126
  - unáris vektor~ 125
  - vektorhalmaz~ 134
  - vektor~ 123
  - ~ vetületei 120, 124, 129, 131, 133, 135
- lekérdezés 56, 164
- LENGTH 276
- lezárás (adat) 47
- LIKE 266
- LIN[ESIZE] 235, 326
- literál 249, 397
- LOCK TABLE 337
- logaritmus keresés 31
- logikai adatfüggetlenség 46
- logikai operátor 267
- logikai szekvenciális állománystruktúrák 32
- logikai változók deklarálása (PL/SQL) 390
- logikai-struktúra 183
  - ~ buroksémája 197
  - ~ relációsémája 185
  - ~ bővítése 195
  - ~-k ekvivalenciája 187, 204
- login.sql 236, 326
- LONG 343, 389
- LONG RAW 343, 389
- LOOP 409
- LOWER 275
- LPAD 276
- másodlagos
  - attribútumnév 149
  - oszlopnév 242
  - táblanevek 306
- MAX 284
- megjegyzés 351, 397
- megszorítás 351
  - ~ engedélyezése 363
  - ~ hozzáadása 361
  - integritási ~ 351
  - ~ lekérdezése 359
  - ~ módosítása 361
  - oszlopszintű ~ 352
  - táblaszintű ~ 352
- megszorítás (folytatás)
  - ~ tiltása 362
  - tovagyűrűző ~ 363
  - ~ törlése 361
- Mentés menüpont 262
- mentési pont 338
- metszet 77, 80, 82, 91, 296
- mező 52, 416
- minivilág 45
- minősített név 78, 241, 374
- MINUS 296
- MOD 278
- MODIFY 346
- módosítás 333
- monohalmaz 79, 296
- monoreláció 112
- monounió 296
- MONTHS\_BETWEEN 280
- multihalmaz 78, 296
  - valódi ~ 80
- multiplicitás 79
- multiplikált halmaz 82
- multiplikátor 79
- multireláció 112
- multiúnió 81, 296
- művelet 136
  - bináris ~ 137
  - egyszerű ~ 137
  - multihalmaz ~-ei 78
  - osztályozó ~ 157, 162
  - összetett ~ 138
  - redukáló ~ 82
  - relációalgebra ~-ei 148
  - szerkezetahagyó ~-ek 149
  - szerkezetmódosító ~-ek 151
  - unáris ~ 137
  - vektorhalmaz ~-ei 94, 101
- NCLOB 390
- nemzeti nyelvi paraméterek 281
- Nested Relational Model 66
- NESTED TABLE adattípus 391
- NEW\_LINE 396
- NEW[FORMAT] 326
- NEW[AGE] 326
- NEXT (PL/SQL) 422
- NEXT\_DAY 280
- nézet 238, 342, 316, 363
  - adatszótárbeli ~ 342
  - ~ törlése 369
- NLS\_DATE\_FORMAT 232
- NLS\_DATE\_LANGUAGE 232
- NOFORCE 365
- NOPRINT 327
- normálforma 52
  - Boyce-Codd ~ 223
  - első ~ 220
  - harmadik ~ 222
  - második ~ 221

- normálforma (folytatás)
  - nulladik ~ 219
  - ösmodell 217
- normalizálás 52, 216
- NOT 267
- NOT NULL 240, 352, 353
- NOWAIT 440
- NULL érték 247, 286
- NUM[WIDTH] 326
- NUMBER 240, 343, 389
- NUMERIC 389
- numerikus függvények 278
- NVL függvény 248, 284, 286
- nyelv megadás 232
- nyilvános kulcsú kódolás 62
- objektum 52, 342
- objektumorientált adatbázis-kezelés 67
- ON DELETE CASCADE 355
- Opciók menüpont 235
- OPEN kurzornév 427, 429
- operandus 75
- operátor
  - egysoros ~311
  - többsoros ~ 11
  - PL/SQL-beli ~398
- OR 267
- ORDER BY 270, 370
- ortogonalitás 187
- oszlop 242
  - ~ bővítés 346
  - ~ fejléc 243
  - ~ módosítás 346
  - ~ törlés 347
- osztálykritérium 143
- osztályozás 142, 145
- osztott
  - ~ adatbázisok 58
  - ~ protokollok 60
- OUT 387, 465, 468, 477, 479
- önálló nyelv (Self Contained Language) 48
- öröklődés 68
- ösmodell 217
- összefűzés operátor 248
- összehasonlító operátor 264
- összekapcsolás 302
  - általános ~ 155
  - belső ~ 303, 307
  - egyen~ 303
  - egyszerű ~ 303
  - külső ~ 303, 308
  - nem egyen~ 303, 306
  - tábla ~ önmagával 303, 309
  - természetes ~ 154
  - Théta ~ 155
- pages[ize] 235, 326
- paraméter
  - aktuális ~ 476
  - alapértelmezett ~ 476
- paraméter (folytatás)
  - formális ~ 476
  - IN (érték) ~ 476
  - IN OUT (érték-eredmény) ~ 478
  - OUT (eredmény) ~ 477
  - ~ tábla 242
- particionálás 143
- partnerazonosítás 64
- patthelyzet 47
- Personal Oracle 231
- Peterson-féle keresés 31
- PL/SQL 254, 326, 383
  - függvények 398
  - rekord 415
  - tábla 415, 420
- PLS\_INTEGER 389
- pontozott egyesítés 78
- POWER 278
- PRAGMA EXCEPTION\_INIT 457
- PRIMARY KEY 352, 354
- PRINT 327, 393
- PRIOR (PL/SQL) 422
- privilegium 375
- procedurális nyelv 48
- PROCEDURE 465
- program típusok 386
- projekció 151
- PROMPT 323
- prompt jel 242
- pszeudooszlop 246, 439
- PUBLIC 373, 379, 382
- PUT 396
- PUT\_LINE 396
- QBE (Query by Example) 56
- Query Language 56
- r[un] 242, 256
- RAISE 457
- RAW 343, 389
- REAL 389
- redukálás
  - halmaz ~-a 79
  - reláció ~-a 150
- REFERENCES 355
- rejtjelezés 60
- rejtjelötvözés 61, 62
- rekord 28, 52, 95, 114, 391, 415
- rektanguláris 91, 111
- rekurzív alprogramok 479
- reláció 52, 112, 113
  - ~ algebra 53, 75
  - ~ alrelációja 115
  - ~ kalkulus 55
  - ~ részrelációja 117
  - triviális ~ 113
  - ~ mint tábla 116
  - ~-k egyesítése 150
  - ~-k különbsége 150
  - ~-k metszete 150
  - ~-k osztása 154

- relációs
  - adatmodell 52, 73
  - rendszer lekérdezése 56
- relációséma 110, 111
  - ~ struktúrája 185
  - beágyazott ~ 219
  - irreducibilis ~ 175
  - ortogonális ~ 187, 190
  - triviális ~ 110
- relatív azonos rekordok 100
- rem[ark] 259
- RENAME 241, 350
- rendezés 270, 291
- rendszermérnök 69
- rep[lace] 258, 365
- részhalmoz 76, 80, 82
  - relatív ~ 77
- résztartomány 93
- RETURN utasítás 471
- RETURN záradék 468
- REVERSE (PL/SQL) 411
- REVOKE 242, 381
- ritka indexelés 38
- ROLE 378
- ROLE\_SYS\_PRIVS 380
- ROLE\_TAB\_PRIVS 380
- ROLLBACK 241, 335, 337, 400
- ROUND 278, 280
- ROWID 246, 356, 389, 439
- ROWNUM 246, 356, 368, 370, 439
- RPAD 276
- salgrade tábla 238
- sav[e] 258
- SAVEPOINT 241, 337, 400
- sávindex 39
- scott/tiger 231
- script program 236, 254
- SELECT 232, 241, 242
- SELECT...INTO (PL/SQL) 400
- séma 46 (*lásd még relációséma*)
  - alséma 115
  - részséma 117
- SERVEROUT[PUT] 326
- SET 50, 325, 333
- SET SERVEROUTPUT 396
- SET VERIFY 320
- SHOW ALL 234, 325
- SHOW VERIFY 325
- skaláris adattípusok (PL/SQL) 389
- skaláris változók deklarálása (PL/SQL) 390
- SMALLINT 389
- specializáció 65
- spo[ol] 258
- SQL nyelv (Structured Query Language) 56, 231
- SQL prompt 232, 255
- SQL%FOUND 425
- SQL%ISOPEN 426
- SQL%NOTFOUND 425
- SQL%ROWCOUNT 426
- SQL\*Plus
  - környezet 231, 254
  - szerkesztési utasítások 255
  - változók (PL/SQL)
- SQL-függvények 398
- sta[rt] 258
- standard csomag 457, 458
- STDDEV 284
- STRING 389
- struktúra 136, 138 (*lásd még logikai struktúra*)
  - egyszerű ~ 139
  - összetett ~ 141
  - részstruktúra 184
  - triviális ~ 184
  - ~ tulajdonságok 139
- subclass 65
- SUBSTR 276
- SUM 284
- superclass 65
- sűrű indexelés 38
- sys/change\_on\_install 231
- SYS\_Cn 352
- sysdate 356
- system/manager 231
- szakértő rendszerek 68
- szegmens 49
  - érzékeny 49
- szekvencia 405
- szekvenciális állománystruktúrák 30
- szelekció 151
- szerepkör 377
- Szerkesztés menüpont 261
- szimmetrikus differencia 78
- szinkronizációs protokoll 59
- szinonima 342, 373
- szinonimok 40
- szűrés 150
- tábla, táblázat 52, 116, 342
  - adatszótárbeli ~ 342
  - altábla 117
  - felhasználói ~ 342
  - ~ feltöltése 253
  - gyermek ~ 352, 355
  - ~ létrehozása 252, 344
  - ~ módosítás 345
  - ~ összekapcsolás
  - résztabla 117
  - szülő ~ 352, 355
  - ~ törlése 254, 349
- táblavázak (skeleton) 56
- TABLE (PL/SQL) 415, 420
- tag (member) 50
- tárolt alprogram 386, 475
- TERM[OUT] 326
- terület (area) 51
- területi szakértő 69
- TO 377

TO\_CHAR 281  
 TO\_CHAR függvény 232  
 TO\_DATE 281  
 TO\_NUMBER 281  
 token módszer 59  
 TOP\_N 370  
 többbretűség 68  
 többsoros utasítás 242  
 törlés 335  
 törlésre kijelölés 348  
 TRANSACTION 337  
 transzvertálás 105, 109  
 tranzakció 337  
 trigger 386  
 TRIM (PL/SQL) 422  
 TRUE 268, 390  
 TRUNC 278, 280  
 TRUNCATE 241  
 TTI[TLE] 326  
 tudásbázis-kezelő rendszerek 68  
 tudásmérnök 69  
 tulajdonos (owner) 50  
 tulajdonság 52  
 type 391, 416  
 UID 246, 356  
 UNDEFINE 325, 394  
 únió 78  
 UNION 296  
 UNION ALL 296  
 UNIQUE 352, 353  
 UNUSED 348  
 UPDATE 241, 333, 401  
 UPPER 275  
 USER\_COL\_PRIVS\_MADE 381  
 USER\_COL\_PRIVS\_RECD 381  
 USER\_ROLE\_PRIVS 380  
 USER\_TAB\_PRIVS 380  
 USER\_TAB\_PRIVS\_RECD 380  
 user 246, 356  
 user\_343  
 user\_catalog 239, 343, 364  
 user\_con\_columns 359  
 user\_constraints 359  
 user\_objects 343  
 user\_synonyms 374  
 user\_tables 238, 343  
 user\_views 364  
 ügyviteli védelem 60  
 üres? 240  
 üzenethitelesítés 64  
 változó  
     felhasználói ~ 323  
     helyettesítő ~ 318  
     ~ kiírása (PL/SQL) 395  
     ~ név 318  
 VALUES 253  
 VAR[iable] 393  
 várakozási gráf 47  
 VARCHAR 389  
 VARCHAR2 240, 343, 389  
 VARIANCE 284  
 VARRAY adattípus 391  
 végrehajtható szegmens 384, 397  
 vektor 83, 85  
     alvektor 86  
     ~láncolás 89  
     monovektor 83  
     multivektor 83  
 vektorhalmaz 94, 101  
     alvektorhalmaz 102  
 VER[IFY] 326  
 vetítés 107, 151  
     halmazvektor ~ 92, 93, 107  
     halmazvetítés ~ 77, 107  
     hasonlósági ~ 103, 108  
     rekord ~ 99, 108  
     transzvertált ~ 104, 109  
     vektor ~ 87, 107  
     vektorhalmaz ~ 94, 101, 108  
 vetület  
     halmazleképezés ~-e 133  
     hipervektor-leképezés ~-e 131  
     leképezés ~-e 119, 120, 129  
     vektorhalmaz-leképezés ~-e 135  
     vektorleképezés ~-e 124  
 vezérlési szerkezetek 405  
 view (lásd nézet) 363  
 virtuális tábla (lásd nézet) 363  
 visszagörgetés (lásd ROLLBACK) 47, 338  
 WHEN 457  
 WHEN OTHERS 457  
 WHERE 262  
 WHILE 413  
 WITH CHECK OPTION 365  
 WITH GRANT OPTION 379  
 WITH READ ONLY 365, 368  
 zárolás 337  
 zártság 67