## ORACLE 8i

### Oracle8i Server

- **Host alapú,**
- **Kliens/szerver,**
- **Elosztott feldolgozás,**
- **Webes feldolgozás.**

**Procedurális lehetőségek (PL/SQL).**
Tárolt eljárások,
AB-triggerek,
Csomagok.

### Oracle8i Server opciói

- **Oracle Advanced Security**
- **Oracle Advanced replication**
- **Oracle Parallel Server**
- **Oracle Visual Information Retrieval**
- **Oracle Advanced Networking**
- **Oracle interMedia**
- **Oracle Spatial**
- **Oracle ConText**
- **WebServer**
- **OLAP**

### SQL*Plus interfész

**Oracle Forms**
**Oracle Reports (jelentéskészitő )**
**SQL*Loader**
**Oracle Designer**
**Personal Oracle8i**

### Particionálás és párhuzamosság

**Partíció-függetlenség**

### ARCHITEKTÚRA

**Táblaterületek –**
az adatállományok csoportosításának eszközei. Táblaterületek egy vagy több adatállomány együttese.

CREATE TABLE **custmer (first_name varchar2(15), ....)**
**TABLESPACE users;**

**Táblaterületek nevei-**



System táblaterület
Temp táblaterület - ideiglenes táblákat
Tools táblaterület - eszközök (Oracle Report)
Felhasználói táblaterületek
Adat és index áblaterületek
Visszagörgető táblaterület (ROOLBACK, v. RBS)

### VÁLTOZÁSNAPLÓK – a tranzakciónapló

**Minden AB-ban lennie kell legalább két online változásnaplónak.**

**ARCHIVELOG üzémmód – teljes visszaállithatóság**
**NOARCHIVELOG**

**Vezérlőállományok.**
Az AB struktúrájának minden változása megjelenik a vezérlőállományokban.
Ajánlatos legalább 2 példányban haználni a vezérlőállományt.

**PROGRAMOK**
**Felhasználoi (kliens-) folyamatok -**
**SQL*Plus, Oracle Forms, Oracle Reports.**
**Kiszolgáló (szerver-) folyamatok – fogadják a kliensfolyamatok igényeit, és kapcsolatot létesítenek az AB-sal az igények teljesítéséhez.**

**Az AB-t működtető folyamatok**
DBWR-folyamat – a megváltozott adatblokkok adatállományba írását végzi.
SMON-folyamat – indításkor elvégzi a szükséges visszaállítást.

### MEMÓRIASTRUKTÚRÁK

Kétféle struktúrát használ:
SGA (System Global Area) - rendszer globális terület
PGA (Program Global Area) – program globális terület

**SGA - rendszer globális terület**

**A kliens- és szerverfolyamatok kommunikációját lehetővé teszi.**
Adatpuffer-gyorsítótár
Szótár-gyorsítótár
Változásnapló-puffer
* SQL-terület

A program globális területe (PGA) egy folyamat adatait és vezérlési információt tartalmazza.

**ORACLE-példány**: AB, saját SGA, külön kiszolgáló-folyamatok

**PL. Egy tranzakció. Bankautomata**
1. Lekérdezzük a számlaegyenlegünket

```
SELECT account_balance FROM bank_table WHERE
account_number=1111111
   AND account_type= 'SAVINGS';
```

Az SQL-utasitás az SGA-n keresztül a kiszolgálófolyamathoz kerül. Az ellenőrzi, hogy az utasitás futtatható változata az SQL-területen van-e.
Ha nincs, akkor értelmezi az utasitást, végrehajtható formában az SQL-területre helyezi és végrehajtja. A gyaorsitótárból a kliens leolvasa az egyenleget és közli a felhasználóval. Pl. 25000 Ft.

**Utána kérünk 20000 forintot.** A kliensfolyamat fogadja a kérésünket és SQL-utasitássá alakitja:

```
UPDATE bank_table SET account_balance=23000  WHERE WHERE
account_number=1111111     AND Account_type= 'SAVINGS';
```

**Ezt az utasitást az ORACLE hajtja végre:**

1. A kliensfolyamat az utasitást az SGA-n keresztül eljuttatja a kiszolgálófolyamathoz.
2. A kiszolgálófolyamat ellenőrzi, hogy rendelkezésére áll-e az utasitás végrehajtható formája. Igen- 4, nem –3.
3. Lértehozza az utasitás végrehajtható formáját és az SQL-területre helyezi.
4. Végrehajtja az utasitást.
5. A manipulált adatokat az adatgyorsitoban vannak? Igen - 7, nem – 6.
6. Az adatokat az Oracle beolvassa az adatállományból az adatgyorsitoba.
7. Oracle rögziti az adat régi értékét a visszagörgető szegmensbe (25000).
8. Oracle másolatokat készit a tranzakcióról a váltoásnaplóban.
9. Az adatgyorsító-tárban lévő adatot megváltoztatja az új értékre (23000).
10. A bankautomata jelzi, hogy a tranzakciót befejezte.
11. A váltoásnaplóban rögzitésre kerül, hogy a tranzakció befejeződött.
12. Az Oracle törli a visszagörgető szegmensben tárolt visszaállitási információt.
13. Az automata kiadja a pénzt.

## ADATBÁZIS-objektumok

**Tábla, adatbázis-trigger, nézet, index, szinonima, jogosultságok (privilégiumok), felhasználódefiniált adattipus, adatszotár, szerepkör, távoli adatbázis, pillanatfelvétel, versenyhelyzet (több felhasználó).**

**SQL*Plus-ban:**

```
SQL> CREATE TABLE customer
  2   (customer_id      number(10)  not null,
  3   surname           varchar2(30) not null,
  4   first_name        varchar2(20),
  5   sales_region      char(2),
  6   ytd_sales         number (10.2),
  7   total_sales       number(14.2);

SQL> ALTER TABLE customer
  2   add tax_exemp_ind varchar2(1);

SQL> DESCRIBE customer  - megnézhetjük a tábla szerkezetét

SQL> ALTER TABLE customer
  2   DROP COLUMN  tax_exemp_ind;
```

### Adattípusok

**Új típus  van:  VARRAY  (variable array)**

```
PL.
CREATE TYPE price AS varray(100) of number
CREATE  TABLE car(car_name varchar2(25), car_value price)
```

### Felhasználó-definiált adattípusok

```
SQL> CREATE TYPE indiv_details AS OBJECT
  2   (name            varchar2(40),
  3   Phone            varchar2(20),
  4   Height           number (3),
  5   Weight           number(4) );

SQL> CREATE TABLE vendors
  2   (vendor_name     indiv_details,
  3   ……);
```

### Adatbázis-trigger

Ellenőrizhetjük az adatokat, mielőtt azok az AB kerülnének.
A megadott adatok alapján másik táblában rekordokat generálhatunk.
Üzleti szabályokat definiálhatunk triggerként.

```
SQL>  CREATE OR REPLACE TRIGGER display_new_sales
  2   BEFORE INSERT OR UPDATE ON  customer
  3   FOR EACH ROW
  4   DECLARE
  5        New_sales_amt  number;
  6   BEGIN
  7      New_sales_amt  := :new.total_sales – nvl(:old.total_sales.0);
  8      dbms_output.put_line('New sales amount:  ' + new_sales_amt);
  9   END;
  /
   Trigger created.
```

A trigger neve - display_new_sales.

:old – a mező régi értéke, :new – a mező új (a tranzakció sikeres befejezési utáni) értéke,
dbms_output.put_line – kiir információt a felhasználó számára.

```
SQL> SET SERVEROUTPUT ON      // a megjelenitést (ha lesz!) végrehajtani.
  2  INSERT INTO customer
  3  VALUES
  4  (11, 'a', 'b', 'c', 12);
New sales amount: 12
1 row created.


SQL> SET SERVEROUTPUT ON
  2  UPDATE customer
  3  SET total_sales = 55
  4  WHERE customer_id=11;

New sales amount: 67
1 row updated.
```

**NÉZETEK – adatok speciális formában**

Nézet egy SQL lekérdezést tartalmazz.

```
SQL>      CREATE VIEW v_customer_sales_rgn
  2       AS
  3       SELECT surname, sales_region
  4       FROM customer;
View created.
```

**INDEXEK**

```
SQL>      CREATE TABLE sample_3
  2       (cola varchar(30),
  3       (colb varchar(30),
  4       (colc varchar(30).
Table created.

CREATE INDEX   colc_ind ON samle_3;
```

**A WHERE utasításrész és az indexek**
**A lekérdezés WHERE felépítése alapján határozza meg a felhasználható indexet, ami a leggyorsabb eredményt biztosit.**

**PRIVILÉGIUMOK**



**Privilégiumok** – engedélyek, amelyek lehetővé teszik az egyes felhasználók számára, hogy más felhasználók adataival dolgozzanak.

```
GRANT SELECT ON samle_3 TO public;
GRANT SELECT ON samle_3 TO x, y;
```

**SZEREPKÖRÖK** – mint lehetőség a felhasználók csoportositására.

```
CREATE ROLE nurse;

GRANT INSERT ON        table_a TO nurse;
GRANT INSERT ON    table_b TO nurse;
GRANT INSERT, DELETE  ON table_c  TO nurse;
GRANT UPDATE ON table_d TO nurse;
GRANT DELETE ON  table_e TO nurse;
GRANT SELECT  ON table_f TO nurse;

GRANT nurse TO x;
GRANT nurse TO y;
```

**TÁROLT OBJEKTUMOK: Eljárások, Csomagok, Függvények.**

```
GRANT EXECUTE  ON my_package TO public;
GRANT EXECUTE  ON my_func TO nurse;
GRANT EXECUTE  ON my_proc TO x;
```

**SZEKVENCIÁK**

**Szekvenciák – kulcsként használható numerikus értékek.**

**KLASZTEREK**

Ha állandóan két vagy több táblából álló tábla-együttessel dolgoznak, akkor a DBA dönthet, hogy a táblákat klaszterben tárolják.

**SQL-UTASITÁSOK**

**DDL (Data definition Language)**

**ALTER PROCEDURE**
**ALTER TABLE**
**ANALYZE** (teljesitménystatisztikát gyújt)
**ALTER TABLE ADD CONSTRAINT**
**CREATE TABLE**
**CREATE INDEX**
**DROP INDEX**
**DROP TABLE**
**GRANT**
**TRUNCATE** (törli egy tábla összes sorát)
**REVOKE** (priviligiumot von vissza)

DML

INSERT
DELETE
UPDATE
SELECT
COMMIT WORK (a tranzakciók változtatásait írja lemezre)
ROLLBACK

### SQL*Plus indítása

**Sqlplus név/jelszó**
**SQL>**

### ADATTÍPUSOK

**char**(méret) - fix , 2000 karakterig,
**nchar**(méret) - azonos a char tipussal, de a max a karakterkészletétől függ,
**varchar2**(méret) – változó hosszúságú, max 4000,
**nvarchar2**(méret) – mint a varchar2, de a max a karakterkészletétől függ,
**varchar** - azonos a varchar tipussal (jelenjeg),
**number**(h,t) – h a szám hossza, t – tizedesjegyek száma,
**blob** – bináris nagyméretú objectum, max 4 GB,
**raw**(méret) – bináris adat, max 2000 bájt,
**date** – dátum,
**long** – változó hosszuságú karakteres adat, max 2 GB.

### DESCRABE parancs

Objektumokról kérhetünk le összefoglaló információt. Pl.

**SQL> DESCRIBE customer**

| Name | Null? | Type |
|---|---|---|
| LAST_NAME | NOT NULL | VARCHAR2(50) |
| STATE_CD | NOT NULL | VARCHAR2(2) |
| SALES | NOT NULL | NUMBER |

### Függvények alkalmazása numerikus adatokon

| | | |
|---|---|---|
| **Ceil(n)** | **ceil(10.7)** | **11** |
| **floor(n)** | **floor(10.7)** | **10** |
| **mod(m,n)** | **mod(7,5)** | **2** |
| **power(m,n)** | **power(3,2)** | **9** |
| **sign(n)** | **-1, 0, 1** | |
| **sqrt(n)** | **sqtr(25)** | **5** |



### Függvények alkalmazása karakteres adatokon

| | | |
|---|---|---|
| **Initcap(s)** | **Initcap('szabó józsef')** | **Szabó József** |
| **Lower(s)** | **Lower('ABC')** | **abc** |
| **Replace(S,s1,s2)** | **Replace('sajto','s','szak')** | **szakjato** |
| **Substr(s,n,m)** | **Substr('abcdef', 3, 2)** | **cd** |
| **Length(s)** | **Length('Oracle')** | **6** |

A

### SHOW ALL

parancs által az SQL*Plus összes beállitását megtekinthetjük.

### Irányítás a kimenet állományba

**SPOOL c:\...**

Az irányítást a **SPOOL OFF** vagy **SPOOL OUT** parancs kikapcsolja.

## PL/SQL

Alkalmazzák a
**Oracle Forms, Oracle Reports, SQL*Plus,**
**Oracle Graphics, Oracle Application Server.**

**Aritmetikai operátorok.**

**Relációs operátorok.**
!= - nem egyenlő

**Szimbólumok**
( ) - listában,
:= - értékadás,
|| - konkatenáció,
--- - megjegyzés,
/* */ - megjegyzés.

## PL/SQL komponensek

## SOROZATOK

**CREATE SEQUENCE sor**

**CURRVAL és NEXTVAL**

```
CREATE SEQUENCE student_sequence
   START WITH 10000;

INSERT INTO students (id, first_name)
   VALUES (student_sequence.NEXTVAL, 'Scott');
```

## TÁBLÁK

```
CREATE TABLE students (
 id            NUMBER(5) PRIMARY KEY,
 first_name    VARCHAR2(20),
 last_name     VARCHAR2(20),
 major         VARCHAR2(30),  -- szak
 current_credits  NUMBER(3)  -- leadott tantárgyak száma
 );

INSERT INTO students (id, first_name, last_name, major, current_credits)
  VALUES (student_sequence.NEXTVAL, 'Scott', 'Smith', 'Computer Science', 0);

INSERT INTO students (id, first_name, last_name, major, current_credits)
  VALUES (student_sequence.NEXTVAL, 'Margaret', 'Mason', 'History', 0);
```



```
INSERT INTO students (id, first_name, last_name, major, current_credits)
  VALUES (student_sequence.NEXTVAL, 'Joanne', 'Junebug', 'Computer Science',
0);

INSERT INTO students (id, first_name, last_name, major, current_credits)
  VALUES (student_sequence.NEXTVAL, 'Manish', 'Murgratroid', 'Economics', 0);
```

## Blokk

A PL/SQL-programok blokk-szerkezetűek.
Lehetnek:

**Névtelen blokk (anonymous block),**
**Megnevezett blokk (Named block),**
**Alprogram (subprogram),**
**Trigger (trigger).**

## Névtelen blokk

```
DECLARE
 v_Num1     NUMBER := 1;
 v_Num2     NUMBER := 2;
 v_String1  VARCHAR2(50) := 'Hello World!';
 v_String2  VARCHAR2(50) := '-- ';
 v_OutputStr VARCHAR2(50);

BEGIN
 INSERT INTO temp_table (num_col, char_col)
      VALUES (v_Num1, v_String1);
 INSERT INTO temp_table (num_col, char_col)
      VALUES (v_Num2, v_String2);

 SELECT char_col
  INTO v_OutputStr
       FROM temp_table
       WHERE num_col = v_Num1;
 DBMS_OUTPUT.PUT_LINE(v_OutputStr);

 SELECT char_col
  INTO v_OutputStr
       FROM temp_table
       WHERE num_col = v_Num2;
 DBMS_OUTPUT.PUT_LINE(v_OutputStr);
```

```
END;
```

**Megnevezett blokk**

```
<<l_InsertIntoTemp>>
DECLARE
 v_Num1     NUMBER := 3;
 v_Num2     NUMBER := 4;
 v_String1  VARCHAR2(50) := 'Hello World!';
 v_String2  VARCHAR2(50) := '-- ';
 v_OutputStr VARCHAR2(50);

BEGIN
 INSERT INTO temp_table (num_col, char_col)
  VALUES (v_Num1, v_String1);
 INSERT INTO temp_table (num_col, char_col)
  VALUES (v_Num2, v_String2);

 SELECT char_col
  INTO v_OutputStr
       FROM temp_table
       WHERE num_col = v_Num1;
 DBMS_OUTPUT.PUT_LINE(v_OutputStr);

 SELECT char_col
  INTO v_OutputStr
       FROM temp_table
       WHERE num_col = v_Num2;
 DBMS_OUTPUT.PUT_LINE(v_OutputStr);
END l_InsertIntoTemp;
```

**Alprogramok**

```
CREATE OR REPLACE PROCEDURE InsertIntoTemp AS
 v_Num1     NUMBER := 5;
 v_Num2     NUMBER := 6;
 v_String1  VARCHAR2(50) := 'Hello World!';
 v_String2  VARCHAR2(50) := '-- ';
 v_OutputStr VARCHAR2(50);
BEGIN
 INSERT INTO temp_table (num_col, char_col)
  VALUES (v_Num1, v_String1);
 INSERT INTO temp_table (num_col, char_col)
  VALUES (v_Num2, v_String2);

 SELECT char_col
  INTO v_OutputStr
       FROM temp_table
       WHERE num_col = v_Num1;
 DBMS_OUTPUT.PUT_LINE(v_OutputStr);

 SELECT char_col
  INTO v_OutputStr
       FROM temp_table
       WHERE num_col = v_Num2;
 DBMS_OUTPUT.PUT_LINE(v_OutputStr);
END InsertIntoTemp;
```



**Trigger.**

```
CREATE OR REPLACE TRIGGER OnlyPositive
 BEFORE INSERT OR UPDATE OF num_col
 ON temp_table
 FOR EACH ROW
BEGIN
 IF :new.num_col < 0 THEN
  RAISE_APPLICATION_ERROR(-20100, 'Please insert a positive value');
 END IF;
END OnlyPositive;
```

**Blokk általános struktúrája**

```
DECLARE
....
BEGIN
....
EXCEPTION
....
END;
```

**Pl.:**

```
DECLARE
 v_StudentID NUMBER(5) := 10000;
 v_FirstName VARCHAR2(20);

BEGIN
 SELECT first_name
  INTO v_FirstName
  FROM students
  WHERE id = v_StudentID;

EXCEPTION
 WHEN NO_DATA_FOUND THEN
```

```
    INSERT INTO log_table (info)
     VALUES ('Student 10000 does not exist!');
END;
```

**VÁLTOZÓK DEKLARÁLÁSA**

**DECLARE**

**Új típus:**

```
CREATE OR REPLACE TYPE StudentObj AS OBJECT (
 ID            NUMBER(5),
 first_name    VARCHAR2(20),
 last_name     VARCHAR2(20),
 major         VARCHAR2(30),
 current_credits NUMBER(3);
```

**Név típus [CONSTANT] [NOT NULL] [:= érték];**

**TÍPUSOK**

**Leggyakoribb adattípusok: Varchar2, number, date, boolean.**

**Numerikus típusok:**

**DEC,**
**DECIMAL,**
**DOUBLE PRECISION,**
**INTEGER,**
**INT,**
**NUMERIC,**
**REAL,**
**SMALLINT,**
**BINARY_INTEGER <= +/- 2147483647**
**NUMBER (m, n)**

| | | |
|---|---|---|
| **NUMBER** | **12.36** | **12.36** |
| **NUMBER (3)** | **123** | **123** |
| **NUMBER (3)** | **1234** | **HIBA** |
| **NUMBER (4,3)** | **1.234567** | **1.235** |
| **NUMBER (3,-3)** | **1234** | **1000** |
| **NUMBER (3,-1)** | **1234** | **1230** |

**Karakteres:**
**VARCHAR2(hossza)** **változó hossza 32767.**
**CHAR(hossza)** **fix hossza**



**DATE**

**BOOLEAN TRUE, FALSE, NULL**

**LOB (Large OBject )**

**%TYPE**
**Name student.first_name%TYPE**

**FÜGGVÉNYEK**

| | | |
|---|---|---|
| **TO_CHAR** | **számokból, dátumból** | **karakterbe** |
| **TO_DATE** | **karakterből** | **dátumba** |
| **TO_NUMBER** | **karakterből** | **számba** |

**ÉRTÉKADÁS**

```
V:=kif;
String_1 := 'Hello' || 'World' || '!';
```

**VEZÉRLÉSI STRUKTURÁK**

**IF - THEN - ELSE**

```
IF log_kif THEN
  Operátorok;
[ELSEIF log_kif THEN
 Operátorok; ]
 .....
[ELSE
 Operátorok;]
END IF;
```

```
DECLARE
 v_NumberSeats rooms.number_seats%TYPE;
 v_Comment VARCHAR2(35);
BEGIN
  SELECT number_seats      INTO v_NumberSeats
      FROM rooms    WHERE room_id = 99999;
 IF v_NumberSeats < 50 THEN
  v_Comment := 'Fairly small';

 ELSIF v_NumberSeats < 100 THEN
  v_Comment := 'A little bigger';
 ELSE
  v_Comment := 'Lots of room';
 END IF;
```

```
END;
```

Vagy-

```
DECLARE
 v_NumberSeats rooms.number_seats%TYPE;
 v_Comment VARCHAR2(35);
BEGIN
 SELECT number_seats
  INTO v_NumberSeats
  FROM rooms
  WHERE room_id = 99999;
 IF v_NumberSeats < 50 THEN
  v_Comment := 'Fairly small';
  INSERT INTO temp_table (char_col)
   VALUES ('Nice and cozy');
 ELSIF v_NumberSeats < 100 THEN
  v_Comment := 'A little bigger';
  INSERT INTO temp_table (char_col)
   VALUES ('Some breathing room');
 ELSE
  v_Comment := 'Lots of room';
 END IF;
END;
```

**CIKLUSOK**

**1. Egyszerű ciklusok**

```
LOOP
 Operátorok;
EXIT [WHEN feltététel]
END LOOP;
```

```
DECLARE
 v_LoopCounter BINARY_INTEGER := 1;
BEGIN
  LOOP
   INSERT INTO temp_table (num_col)
       VALUES (v_LoopCounter);
    v_LoopCounter := v_LoopCounter + 1;
   EXIT WHEN v_LoopCounter > 50;
  END LOOP;
END;
```

Vagy:

```
BEGIN
 FOR v_LoopCounter IN 1..50 LOOP
   INSERT INTO temp_table (num_col)
    VALUES (v_LoopCounter);
 END LOOP;
END;
```

```
DECLARE
 v_Counter BINARY_INTEGER := 1;
BEGIN
 LOOP
  INSERT INTO temp_table
   VALUES (v_Counter, 'Loop index');
  v_Counter := v_Counter + 1;

  IF v_Counter > 50 THEN
     EXIT;
  END IF;
 END LOOP;
END;
```

**Vagy-**

```
DECLARE
 v_Counter BINARY_INTEGER := 1;
BEGIN
LOOP
  INSERT INTO temp_table
   VALUES (v_Counter, 'Loop index');
  v_Counter := v_Counter + 1;

   EXIT WHEN v_Counter > 50;
END LOOP;
END;
```

**2. WHILE ciklusok**

```
WHILE feltétel LOOP
     Operátorok;
END LOOP;
```

```
DECLARE
 v_Counter BINARY_INTEGER := 1;
BEGIN
WHILE v_Counter <= 50 LOOP
  INSERT INTO temp_table
   VALUES (v_Counter, 'Loop index');
  v_Counter := v_Counter + 1;
 END LOOP;
END;
```

**3. FOR ciklusok.**

```
FOR i IN [REVERSE] i_min .. i_max    LOOP
      Operátorok;
END LOOP;
```

```
BEGIN
 FOR v_Counter IN 1..50 LOOP
  INSERT INTO temp_table
   VALUES (v_Counter, 'Loop Index');
 END LOOP;
END;
```

```
BEGIN
 FOR v_Counter IN REVERSE 1..50 LOOP
  INSERT INTO temp_table
   VALUES (v_Counter, 'Loop Index');
 END LOOP;
END;
```

**Kurzorok**

```
DECLARE
 v_FirstName VARCHAR2(20);
 v_LastName  VARCHAR2(20);
 CURSOR c_Students IS
  SELECT first_name, last_name
   FROM students;
BEGIN
 OPEN c_Students;
 LOOP
  FETCH c_Students INTO v_FirstName, v_LastName;
  EXIT WHEN c_Students%NOTFOUND;
  /* Process data here */
 END LOOP;
 CLOSE c_Students;
END;
```

**GOTO és cimkék**

```
GOTO << cimke >>;
```

```
DECLARE
 v_Counter  BINARY_INTEGER := 1;
BEGIN
 LOOP
  INSERT INTO temp_table
   VALUES (v_Counter, 'Loop count');
```



```
  v_Counter := v_Counter + 1;
  IF v_Counter > 50 THEN
   GOTO l_EndOfLoop;
  END IF;
 END LOOP;

 <<l_EndOfLoop>>
 INSERT INTO temp_table (char_col)
  VALUES ('Done!');
END;
```

**NULL operátor**

```
DECLARE
 v_TempVar  NUMBER := 7;
BEGIN
 IF v_TempVar < 5 THEN
  INSERT INTO temp_table (char_col)
   VALUES ('Too small');
 ELSIF v_TempVar < 10 THEN
  INSERT INTO temp_table (char_col)
   VALUES ('Just right');
 ELSE
        NULL;
 END IF;
END;
```

---

**RECORDOK**

```
TYPE  t_StudentRecord  IS  RECORD (
   StudentID NUMBER (5),
   FirstName VARCHAR2 (20),
  LastName VARCHAR2 (20) );

V_StudentInfo t_StudentRecord;
```

```
DECLARE
 TYPE t_Rec1Type IS RECORD (
    Field1 NUMBER,
    Field2 VARCHAR2(5));
 TYPE t_Rec2Type IS RECORD (
  Field1 NUMBER,
  Field2 VARCHAR2(5));
 v_Rec1 t_Rec1Type;
 v_Rec2 t_Rec2Type;
BEGIN
 v_Rec1 := v_Rec2;     /*  különböző típusok,  HIBA !!!  */

 v_Rec1.Field1 := v_Rec2.Field1;
```

```
 v_Rec2.Field2 := v_Rec2.Field2;
END;



DECLARE
TYPE t_StudentRecord IS RECORD (
   FirstName  students.first_name%TYPE,
   LastName   students.last_name%TYPE,
   Major      students.major%TYPE);

v_Student  t_StudentRecord;

BEGIN
SELECT first_name, last_name, major
   INTO v_Student
   FROM students
   WHERE ID = 10000;
END;
```

**%ROWTYPE alkalmazása**

```
DECLARE
  V_RoomRecord    rooms%ROWTYPE
```

**TÁBLÁK (tömbök)**

```
TYPE tábla_tipus IS  TABLE  OF tipus
    INDEX BY BINARY_INTEGER;
(nem kötelező!)

TYPE  t_CharacterTable  IS  TABLE  OF  VARCHAR2(10)
    INDEX BY BINARY_INTEGER;

V_ Characters     t_CharacterTable;

DECLARE
TYPE  t_NameTable  IS  TABLE  OF students.first_name%TYPE
    INDEX BY BINARY_INTEGER;
TYPE  t_DateTable  IS  TABLE  OF DATE
    INDEX BY BINARY_INTEGER;
V_ Names     t_NameTable;
V_ Dates     t_DateTable;

BEGIN
   V_ Names(1):= 'Scott';
   V_ Dates(4):=  SYSDATE - 1;
END;
```



```
SET serveroutput on
DECLARE
  TYPE t_StudentTable IS TABLE OF students%ROWTYPE
    INDEX BY BINARY_INTEGER;
v_Students t_StudentTable;

BEGIN
  SELECT *
    INTO v_Students(10001)
    FROM students
    WHERE id = 10001;

  v_Students(10001).first_name := 'Larry';
  DBMS_OUTPUT.PUT_LINE(v_Students(10001).first_name);
END;
```

**Tábla atributúmai** (tulajdonságai )

```
COUNT      -> NUMBER
DELETE     -
EXISTS     -> BOOLEAN
FIRST      -> BINARY_INTEGER
LAST       -> BINARY_INTEGER
NEXT       -> BINARY_INTEGER
PRIOR      -> BINARY_INTEGER

DECLARE
  TYPE t_NumberTable IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_Numbers t_NumberTable;
  v_Total NUMBER;
BEGIN
FOR v_Counter IN 1..50 LOOP
    v_Numbers(v_Counter) := v_Counter;
  END LOOP;

  v_Total := v_Numbers.COUNT;
END;

DELETE
DELETE (5)
DELETE(5, 8)        5,6,7,8


SET serveroutput on
DECLARE
  TYPE t_ValueTable IS TABLE OF VARCHAR2(10)
    INDEX BY BINARY_INTEGER;
```