

8.előadás: Adatbázisok-1

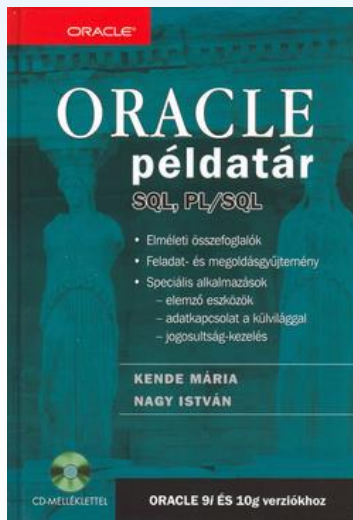
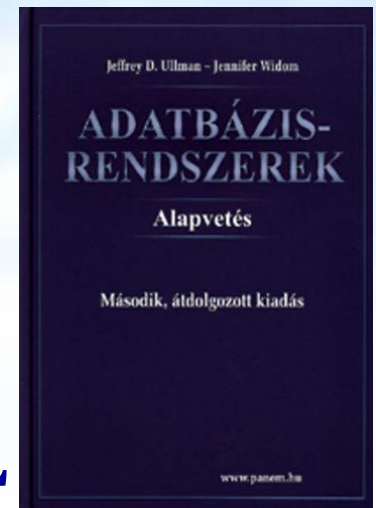
dr. Hajas Csilla (ELTE IK)

<https://sila.hajas.elte.hu/>

Tk-ben: SQL/PSM

9.3. Az SQL és a befogadó nyelv közötti felület

9.4. SQL/PSM Sémában tárolt alprogramok



Gyakorlaton: Oracle PL/SQL

>> Kende Mária-Nagy István:
Oracle példatár (8-10.fejezet)

>> Gábor András-Juhász István:
PL/SQL programozás

Emlékeztető: SQL fő komponensei

- **Az SQL elsődlegesen lekérdező nyelv** (Query Language)
SELECT utasítás (az adatbázisból információhoz jussunk)
- **Adatkezelő nyelv, DML** (Data Manipulation Language)
INSERT, UPDATE, DELETE, SELECT
- **Sémaleíró nyelv, DDL** (Data Definition Language)
CREATE, ALTER, DROP
- **Tranzakció-kezelés**
COMMIT, ROLLBACK, SAVEPOINT
- **Adatvezérlő nyelv, DCL** (Data Control Language)
GRANT, REVOKE
- **Procedurális kiterjesztések**
SQL/PSM és a gyakorlatban Oracle PL/SQL

Háromféle programozási megközelítés

- 1.) **SQL kiterjesztése procedurális eszközökkel**, az adatbázis séma részeként tárolt kódrészekkel, tárolt modulokkal (pl. **PSM** = Persistent Stored Modules, **Oracle PL/SQL**).
- 2.) **Beágyazott SQL** (sajátos előzetes beágyazás EXEC SQL. - Előfordító alakítja át a befogadó gazdanyelvre/host language, pl. C)
- 3.) **Hívásszintű felület**: hagyományos nyelvben programozunk, függvénykönyvtárat használunk az adatbázishoz való hozzáféréshez (pl. CLI = call-level interface, JDBC, PHP/DB)

SQL programnyelvi környezetben

- Milyen problémák merülnek fel, amikor egy alkalmazás részeként, programban használjuk az SQL utasításokat?
- 1.) **Osztott változók használata:** közös változók a nyelv és az SQL utasítás között (ott használható SQL utasításban, ahol kifejezés használható).
- 2.) **A típuseltérés problémája:** Az SQL magját a relációs adatmodell képezi. Reláció: gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. A lekérdezés eredménye hogyan használható fel? Megoldás:

Lekérdezések használata a PSM-ben

- **A típuseltérés problémája:** Az SQL multihalmaz szemlélete hogyan egyeztethető össze a magas-szintű programnyelvekkel? A lekérdezés eredménye hogyan használható fel?
- Három esetet különböztetünk meg attól függően, hogy a `SELECT FROM [WHERE stb]` lekérdezés eredménye skalárértékkel, egyetlen sorral vagy egy listával (multihalmazzal) tér-e vissza.

Lekérdezések használata a PSM-ben

- SELECT eredményének használata:
 1. SELECT eredménye egy **skalárértékkel** tér vissza, **elemi kifejezésként** használhatjuk.
 2. SELECT **egyetlen sorral** tér vissza
SELECT e_1, \dots, e_n **INTO** $vált_1, \dots, vált_n$
 - A végrehajtásnál visszatérő üzenethez az
 - SQL STATE változóban férhetünk hozzá.
 3. SELECT eredménye **több sorból álló tábla**, akkor az eredményt soronként bejárhatóvá tesszük, **kurzor** használatával.

PL/SQL

Oracle PL/SQL **9-11.gyakorlatok** témaköre:

- Blokkos szerkezet
- Változók, Típusok
- Vezérlési szerkezetek
- Kurzorok, kurzorváltozók
- Alprogramok, Tárolt eljárások és függvények
- Hiba-és kivételkezelés
- Triggerek
- Monoton rekurzív lekérdezések (Eljut-feladat)

PL/SQL – I.rész az alapok

- ELTE Adatbázisok gyakorlaton: Oracle PL/SQL
- **Oracle® Database PL/SQL Language Reference**
- PL/SQL
- Procedurális nyelv
- Az SQL DML-t kombinálja a procedurális nyelvi feldolgozással (adatbázis + programozás)

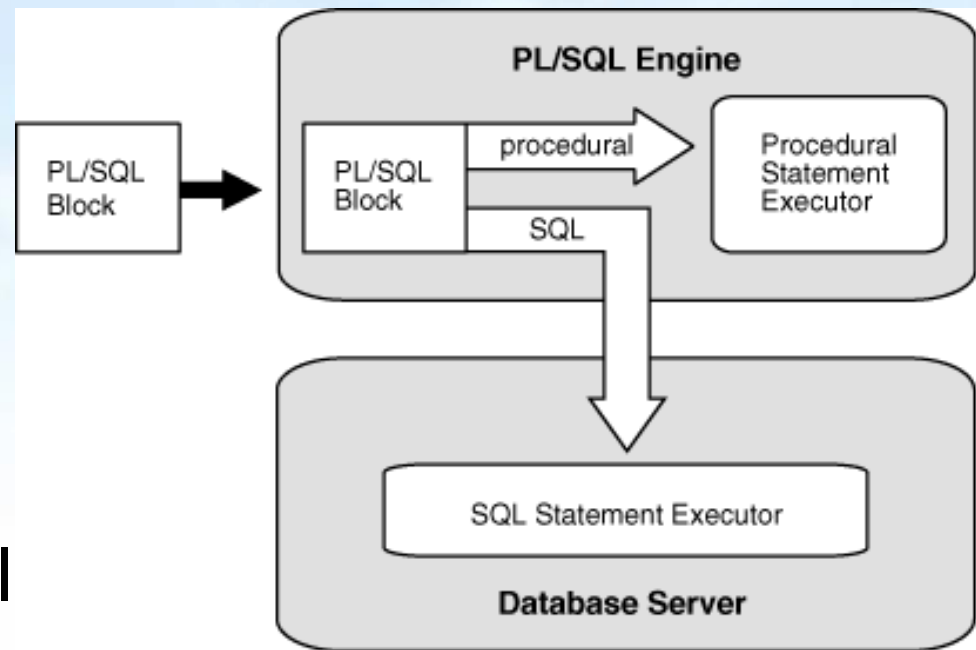


Figure 1-1 PL/SQL Engine

PL/SQL

- Egy PL/SQL blokk szerkezete:

[címke]

[DECLARE

deklarációs utasítások]

BEGIN

végrehajtandó utasítások

[EXCEPTION

kivételkezelés]

END [név];

PL/SQL

➤ Eljárás deklaráció

```
PROCEDURE eljárás_neve [(formális  
paraméterlista)]  
IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név];
```

PL/SQL

➤ Függvény deklaráció

```
FUNCTION függvény_neve [(formális  
paraméterlista)]  
RETURN típus IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név];
```

PL/SQL

- A különbség az eljárás és a függvény között
 - Eljárás: direkt módon nem adnak vissza értéket, általában utasítások lefuttatása a cél (illetve logikailag egy egységbe tartozó utasítások egy helyen kezelése)
 - Függvény: visszaad egy értéket, általában arra használják, hogy kiszámítsanak valamit és azt visszaadják.

PL/SQL – Deklarációs rész ---1

- Tartalma lehet
 - Típus definíció
 - Változó deklaráció

Név típus [[NOT NULL] {:= | DEFAULT} kifejezés];

Példák: belépési idő változó, illetve dolgozók száma változó és az alapértelmezett értéke 0.

```
belepesi_ido DATE;
```

```
dolg_szama NUMBER NOT NULL DEFAULT 0;
```

```
dolg_fizetes NUMBER NOT NULL := 1000;
```

PL/SQL – Deklarációs rész ---2

- Tartalma lehet
 - Nevesített konstans deklaráció
 - Név CONSTANT típus [NOT NULL] {:= | DEFAULT} kifejezés;

Példa: fizetés konstans, melynek értéke 1000.

```
fizetes CONSTANT NUMBER := 1000;
```

- Kivétel deklaráció
- Kurzor definíció
- Alprogram definíció

PL/SQL – Adattípusok ---1

- Logikai (ez új, nem volt a create table esetén)
 - BOOLEAN --- 3-értékű logika

- Numerikus
 - NUMBER – ez így lebegőpontos
 - NUMBER(3) – ez így fixpontos
 - FLOAT – nem korlátozott lebegőpontos
 - INT, INTEGER, SMALLINT – korlátozott fixpontos
 - stb ...

PL/SQL – Adattípusok---2

- Karakteres
 - CHAR
 - VARCHAR2
 - NVARCHAR2
 - stb...
- Dátum
 - DATE
 - TIMESTAMP (p)
 - INTERVAL

PL/SQL – Adattípusok---3

- A deklarációban a típus lehet
 - Skalár adattípus
 - Hivatkozási típus: %TYPE, %ROWTYPE
 - Változónév / rekordnév /
adatbázis_tábla_név.oszlopnév /
kurzorváltozó_név / kollekciónév /
objektumnév%TYPE
 - Adatbázis_táblanév /
kurzornév%ROWTYPE

PL/SQL – Adattípusok---4

- PL/SQL Ref.: Example 2-24 Assigning Values

DECLARE -- You can assign initial values here

counter NUMBER := 0;

done BOOLEAN;

emp_rec employees%ROWTYPE;

BEGIN -- You can assign values here too

done := (counter > 100);

emp_rec.first_name := 'Antonio';

emp_rec.last_name := 'Ortiz';

END;

/

PL/SQL – Adattípusok---5

- Rekord típus deklaráció

- TYPE név IS RECORD (
mezőnév típus [[NOT NULL] {:= |
DEFAULT} kifejezés], ...);

- Példa: telefonkönyv rekord
TYPE telkonyv IS RECORD (
szam NUMBER,
nev VARCHAR2(20));

- Rekord deklaráció
telefonkonyv telkonyv;

- Rekord mezőjének elérése
telefonkonyv.nev;

PL/SQL – Adattípusok---6

➤ Altípusok

➤ SUBTYPE név IS
alaptípus_név[(korlát)] [NOT NULL];

➤ Példa: beépített altípus az INTEGER
SUBTYPE INTEGER IS NUMBER(38,0);

➤ Tömbök

➤ TYPE típusnév IS VARRAY(n) OF
<elemek típusa>;

➤ Példa: TYPE szamtomb IS VARRAY(10)
OF NUMBER;

PL/SQL - Típuskonverzió

- Implicit a skalártípusok között
- Explicit a beépített függvények használatával
 - TO_DATE
 - TO_NUMBER
 - TO_CHAR

PL/SQL – Kiírás a konzolra

- A PL/SQL nem tartalmaz I/O utasításokat.
 - A DBMS_OUTPUT csomag segítségével üzenetet helyezhetünk el egy belső pufferbe.
 - PUT_LINE eljárás üzenetet ír a pufferbe
 - A puffer tartalmát a SET SERVEROUTPUT ON utasítással jeleníthetjük meg a képernyőn
-
- Példa: Hello World!

```
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
/
```

PL/SQL – Utasítások

- Üres
 - NULL;
- Értékadó
 - $X := 0;$
- Ugró
 - GOTO címke;

PL/SQL – Utasítások

- Elágazás
 - IF
 - CASE
- Ciklusok
 - Végtelen
 - WHILE
 - FOR
 - Kurzor FOR (később)
- SQL utasítások

PL/SQL – IF utasítás

➤ Szintaxis:

IF (feltétel)

 THEN utasítás [utasítás] ...

 [ELSIF (feltétel)

 THEN utasítás [utasítás] ...] ...

 [ELSE utasítás [utasítás] ...]

END IF;

PL/SQL – IF utasítás

```
SET SERVEROUTPUT ON
DECLARE
  a number(3) := 100;
BEGIN
  IF ( a = 10 ) THEN
    dbms_output.put_line('Value of a is 10' );
  ELSIF ( a = 20 ) THEN
    dbms_output.put_line('Value of a is 20' );
  ELSIF ( a = 30 ) THEN
    dbms_output.put_line('Value of a is 30' );
  ELSE
    dbms_output.put_line('None of the values is matching');
  END IF;
  dbms_output.put_line('Exact value of a is: '|| a );
END;
/
```


PL/SQL – CASE utasítás

➤ Szintaxis:

CASE kifejezés

WHEN érték1 THEN utasítás1;

...

ELSE utasítás

END CASE;

PL/SQL – CASE utasítás

```
SET SERVEROUTPUT ON
DECLARE
  grade char(1) := 'A';
BEGIN
  CASE grade
    when 'A' then dbms_output.put_line('Excellent');
    when 'B' then dbms_output.put_line('Very good');
    when 'C' then dbms_output.put_line('Well done');
    when 'D' then dbms_output.put_line('You passed');
    when 'F' then dbms_output.put_line('Better try
again');
    else dbms_output.put_line('No such grade');
  END CASE;
END;
```

PL/SQL – LOOP utasítás

- Végtelen ciklus
- Szintaxis:

```
LOOP  
utasítás(ok);  
END LOOP;
```

- EXIT-re lép ki
 - Ehelyett használható EXIT WHEN (feltétel) is

PL/SQL – LOOP utasítás

```
➤ SET SERVEROUTPUT ON
DECLARE
  x number := 10;
BEGIN
  LOOP
    dbms_output.put_line(x);
    x := x + 10;
    IF x > 50 THEN
      exit; -- itt lep majd ki
    END IF;
  END LOOP;
  dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

PL/SQL – WHILE utasítás

- Előltesztelős ciklus
- Szintaxis:

```
WHILE feltétel LOOP  
utasítás(ok);  
END LOOP;
```

PL/SQL – WHILE utasítás

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
  a number(2) := 10;
```

```
BEGIN
```

```
  WHILE a < 20 LOOP
```

```
    dbms_output.put_line('value of a: ' || a);
```

```
    a := a + 1;
```

```
  END LOOP;
```

```
END;
```


PL/SQL – FOR utasítás

- Számlálós ciklus
- Szintaxis:

```
FOR számláló IN [REVERSE] kezdőérték ..  
Végérték LOOP  
utasítás(ok);  
END LOOP;
```

PL/SQL – FOR utasítás

```
SET SERVEROUTPUT ON
DECLARE
  a number(2);
BEGIN
  FOR a in 10 .. 20 LOOP
    dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
```

SQL utasítások PL/SQL-ben

- **Nem használható SELECT**, csak spec.esetben
 - amikor egy sort ad vissza kiegészül egy INTO (ill. ált. BULK COLLECT INTO) utasításrészsel
- **DML utasítások**: INSERT, DELETE, UPDATE
 - kiegészülnek egy RETURNING utasításrészsel, segítségével az érintett sorok alapján számított értéket kaphatunk meg
- MERGE
 - „UPSERT” funkcionalitás
- **Tranz.kez.:** COMMIT, ROLLBACK, SAVEPOINT

SQL utasítások PL/SQL-ben

- SELECT értékének kiválasztása egy változóba
 - **SELECT** select_kifejezés **INTO** változónév
FROM táblanév;
- Példa: King adatainak tárolása a dolg változóban:
 - DECLARE
 dolg dolgozo%ROWTYPE;
BEGIN
 SELECT * INTO dolg
 FROM dolgozo
 WHERE dnev='KING';
END;

SQL utasítások PL/SQL-ben

➤ PL/SQL Ref: Example 2-25 SELECT INTO
DECLARE

```
bonus NUMBER(8,2);
```

BEGIN

```
SELECT salary * 0.10 INTO bonus
```

```
FROM employees
```

```
WHERE employee_id = 100;
```

```
DBMS_OUTPUT.PUT_LINE('bonus = ' || TO_CHAR(bonus));
```

```
END;
```

```
/
```

SQL utasítások PL/SQL-ben

- Törlés egy táblából

```
DELETE [FROM] táblahivatkozás  
[WHERE feltétel]  
[returning utasításrész];
```

- A RETURNING formája

```
RETURNING  
egysoros select kifejezés[, ...]  
INTO {változó[, ...] | rekord};
```

SQL utasítások PL/SQL-ben

- Beszúrás egy táblába

INSERT INTO táblahivatkozás
[(oszlop, ...)]

VALUES

{(sql_kifejezés, [...]) | rekord}
[returning utasításrész];

SQL utasítások PL/SQL-ben

- Táblában érték módosítása

```
UPDATE táblahivatkozás  
SET oszlop=sql_kifejezés [, ...]  
[WHERE feltétel]  
[returning utasításrész];
```


SQL utasítások PL/SQL-ben

```
DECLARE          -- PL/SQL Ref.: Example 6-1 Static SQL Statements
emp_id    employees.employee_id%TYPE := 299;
emp_first_name  employees.first_name%TYPE := 'Bob';
emp_last_name   employees.last_name%TYPE  := 'Henry';
BEGIN
  INSERT INTO employees (employee_id, first_name, last_name)
    VALUES (emp_id, emp_first_name, emp_last_name);
  UPDATE employees
    SET first_name = 'Robert'
    WHERE employee_id = emp_id;
  DELETE FROM employees
    WHERE employee_id = emp_id
    RETURNING first_name, last_name
    INTO emp_first_name, emp_last_name;
  COMMIT;
  DBMS_OUTPUT.PUT_LINE (emp_first_name || ' ' || emp_last_name);
END;
/
```

SQL utasítások PL/SQL-ben

- PL/SQL Ref.: Example 6-4 **SQL%ROWCOUNT**

```
DROP TABLE emp_temp;
CREATE TABLE emp_temp AS
    SELECT * FROM employees;
DECLARE
    mno NUMBER(6) := 122;
BEGIN
    DELETE FROM emp_temp WHERE manager_id = mno;
    DBMS_OUTPUT.PUT_LINE ('Number of employees
        deleted: ' || TO_CHAR(SQL%ROWCOUNT));
END;
/
```

PL/SQL – II.rész: Kurzorok

- ~Iterátorok ahhoz, hogy adatbázisok sorait tudjuk kezelni PL/SQL-ben
- Két típus:
 - Implicit
 - Explicit

PL/SQL - Kurzorok

- Implicit kurzort az Oracle hoz létre, amennyiben SQL utasítást futtatunk és nincs hozzá explicit kurzor. Ilyen például a következő dián lévő FOR-ban SELECT, de lehet bármelyik DML utasítás is.
- Explicit kurzort mi tudunk létrehozni

PL/SQL - Kurzorok

- Implicit kurzor FOR ciklushoz

```
FOR ciklusváltozó_név IN (SELECT  
utasítás)
```

```
  LOOP
```

```
    utasítások;
```

```
  END LOOP;
```

- A ciklusváltozó kurzornév%ROWTYPE típusú lesz
- Megnyitja, betölti az aktív halmaz összes sorát, majd bezárja a kurzort

PL/SQL - Kurzorok

- Példa: az alábbi program kiírja minden dolgozó kódját és nevét PL/SQL-ből implicit kurzort használva.

```
SET SERVEROUTPUT ON
```

```
BEGIN
```

```
    FOR cikl IN (SELECT * FROM Dolgozo)
```

```
    LOOP
```

```
        dbms_output.put_line('Kod: ' ||  
cikl.dkod || ', nev: ' || cikl.dnev);
```

```
    END LOOP;
```

```
END;
```

PL/SQL - Kurzorok

- Explicit kurzor létrehozás (a deklarációs részben):

```
CURSOR név [(paraméterlista)]  
    [RETURN sortípus]  
IS  
    select utasítás;
```

- Ha nem adunk meg sortípust, akkor az Oracle kikövetkezteti a legtöbb esetben.

PL/SQL - Kurzorok

- Használathoz a kurzort meg kell nyitni. Erre az OPEN utasítás szolgál:

OPEN kurzornév [aktuális
paraméterlista];

PL/SQL - Kurzorok

- A kurzorból a sorokat változókba kell betölteni, erre a FETCH utasítást használjuk:

```
FETCH {kurzornév | kurzorváltozó név}  
    { INTO {rekordnév | változónév lista}  
    |  
    BULK COLLECT INTO kollekciónév lista  
    LIMIT sorok};
```

PL/SQL - Kurzorok

- Használat után a kurzort be kell zárni a CLOSE utasítással:

```
CLOSE {kurzornév | kurzorváltozó  
név};
```

PL/SQL - Kurzorok

- Példa: az alábbi program kiírja minden dolgozó kódját és nevét PL/SQL-ből explicit kurzort használva.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR curs IS SELECT * FROM Dolgozo;
    dolg Dolgozo%ROWTYPE;
BEGIN
    OPEN curs;
    LOOP
        FETCH curs into dolg;
        EXIT WHEN curs%NOTFOUND;
        dbms_output.put_line('Kod: ' ||
dolg.dkod || ', nev: ' || dolg.dnev);
    END LOOP;
END;
```

PL/SQL - Kurzorok

- Kurzorattribútumok
 - %FOUND
 - Megnyitás után, de az első betöltés előtt értéke NULL
 - Sikeres betöltés esetén értéke TRUE
 - Sikertelen betöltés esetén értéke FALSE
 - %NOTFOUND
 - A fentebbi negáltja

PL/SQL - Kurzorok

- Kurzorattribútumok
 - %ISOPEN
 - Amennyiben a kurzor meg van nyitva, értéke TRUE
 - Ellenkező esetben FALSE
 - %ROWCOUNT
 - Megnyitás után, de az első betöltés előtt értéke 0
 - Minden sikeres betöltés esetén eggyel nő az értéke

PL/SQL - Kurzorok

```
DECLARE    -- PL/SQL Ref.: Example 6-14 %ROWCOUNT Attribute
    CURSOR c1 IS
        SELECT last_name FROM employees;
    name employees.last_name%TYPE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO name;
        EXIT WHEN c1%NOTFOUND OR c1%NOTFOUND IS NULL;
        DBMS_OUTPUT.PUT_LINE(c1%ROWCOUNT || ' ' || name);
        IF c1%ROWCOUNT = 5 THEN
            DBMS_OUTPUT.PUT_LINE('--- Fetched 5th record ---');
        END IF;
    END LOOP;
    CLOSE c1;
END;
```

PL/SQL - Kurzorok

- Amennyiben UPDATE vagy DELETE utasítást szeretnénk használni explicit kurzorral hasznos lehet a WHERE CURRENT OF kurzornév utasítás, mellyel a kurzorba a legutóbbi FETCH által betöltött sor módosítható / törölhető, explicit zárolást eredményez.

PL/SQL - Kurzorok

- Példa: ha valakinek a foglalkozása manager és a fizetése még nem éri el az 5000-et, akkor állítsuk 5000-re. Csak a ciklust leírva:

LOOP

```
    FETCH curs INTO v_curs;  
    EXIT WHEN curs%NOTFOUND;  
    IF v_curs.foglalkozas='MANAGER'  
AND v_curs.fizetes<5000 THEN  
        UPDATE dolgozo SET fizetes=5000  
            WHERE CURRENT OF curs;  
    END IF;  
END LOOP;
```


PL/SQL - Kurzorok

```
DECLARE -- PL/SQL Ref.: Example 6-43 FOR UPDATE Cursor
my_emp_id employees.employee_id%type;
my_job_id employees.job_id%type;
my_sal employees.salary%type;
CURSOR c1 IS
    SELECT employee_id, job_id, salary
    FROM employees FOR UPDATE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO my_emp_id, my_job_id, my_sal;
        IF my_job_id = 'SA_REP' THEN
            UPDATE employees
            SET salary = salary * 1.02
            WHERE CURRENT OF c1;
        END IF;
        EXIT WHEN c1%NOTFOUND;
    END LOOP;
    CLOSE c1;
END;
```

PL/SQL - Kurzorok

```
DECLARE --PL/SQL REF: Example 6-17 Parameters to Explicit Cursors
emp_job    employees.job_id%TYPE := 'ST_CLERK';
emp_salary employees.salary%TYPE := 3000;
my_record  employees%ROWTYPE;
CURSOR c1 (job VARCHAR2, max_wage NUMBER) IS
  SELECT * FROM employees
  WHERE job_id = job AND salary > max_wage;
BEGIN
  OPEN c1(emp_job, emp_salary);
  LOOP
    FETCH c1 INTO my_record;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE
      ('Name = ' || my_record.last_name || ', salary = ' ||
       my_record.salary || ', Job Id = ' || my_record.job_id );
  END LOOP;
END;
```

PL/SQL - Kurzorok

- Kurzorváltozók
 - Nem kell fordítási időben ismerni a SELECT utasítást
 - Referencia típusú változó
 - Két lépéses létrehozás

1. REF CURSOR típus létrehozása

```
TYPE név IS REF CURSOR [RETURN  
{táblanév|kuzornév|kuzorvá]tozónév}  
%ROWTYPE | rekordnév%TYPE |  
rekordtípusnév |  
kuzorreferenciatípus_név];
```

PL/SQL - Kurzorok

1. Kurzorváltozó deklarációja

```
kurzorváltozó_neve  
ref_cursor_típus_neve;
```

PL/SQL - Kurzorok

- Kurzorreferencia típus lehet
 - Erős, amennyiben szerepel RETURN rész, ekkor a fordító majd ellenőrzi a később kapcsolt SELECT típuskompatibilitását.
 - Gyenge, melyhez bármilyen lekérdezés hozzákapcsolható.
- Megnyitására az OPEN . . . FOR utasítás használandó
OPEN kurzorváltozó_név FOR select utasítás;

PL/SQL - Alprogramok

- Deklarálhatóak
 - Blokkba ágyazva
 - Séma szinten
 - Csomagban

PL/SQL - Alprogramok

- A különbség az eljárás és a függvény között
 - Eljárás: direkt módon nem adnak vissza értéket, általában utasítások lefuttatása a cél (illetve logikailag egy egységbe tartozó utasítások egy helyen kezelése)
 - Függvény: visszaad egy értéket, általában arra használják, hogy kiszámítsanak valamit és azt visszaadják.

PL/SQL - Alprogramok

- Miért használjuk?
 - Átláthatóbbá teszi a kódot
 - Támogatja az újrafelhasználást
 - OOP-szerű

PL/SQL - Alprogramok

➤ Eljárás deklaráció

```
PROCEDURE eljárás_neve [(formális  
paraméterlista)]  
IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név];
```

PL/SQL - Alprogramok

➤ Függvény deklaráció

```
FUNCTION függvény_neve [(formális  
paraméterlista)]  
RETURN típus IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név];
```

PL/SQL - Alprogramok

- Példa: PL/SQL blokkban deklarált eljárás (koszon) és függvény(fix_szam), melyeket meghívunk a PL/SQL programból.

```
SET SERVEROUTPUT ON
DECLARE
    szam NUMBER(2);
    PROCEDURE koszon IS
    BEGIN
        dbms_output.put_line('Hello!');
    END koszon;
    function fix_szam RETURN NUMBER is
    BEGIN
        RETURN 10;
    END fix_szam;
BEGIN
    koszon;
    szam := fix_szam;
    dbms_output.put_line(szam);
END;
```

PL/SQL - Alprogramok

- Formális paraméterlista

név [{IN|OUT|IN OUT} [NO COPY]] típus
[{:|=|DEFAULT} kifejezés];

- IN: érték szerinti paraméterátadás
- OUT: eredmény szerinti paraméterátadás
- IN OUT: érték-eredmény szerinti paraméterátadás
- NOCOPY: hint a fordítónak, hogy IN OUT esetben se másoljon értéket

PL/SQL - Alprogramok

- A paraméterösszerendelés történhet pozíció, és/vagy név alapján
 - Keverhetjük a kettő módszert, ekkor először a pozíció, utána a név szerintiek jönnek
- A lokális és csomagbeli nevek túlterhelhetőek
- Példa: különféle formális paraméterek használata. Az inp paramétert csak beolvassuk és értékét használjuk, az outp paraméterbe csak eredményt írunk, az inout paraméterből olvasunk is és írunk is bele. A példában pozíció szerinti paraméter-összerendelés történik.

PL/SQL - Alprogramok

```
➤ SET SERVEROUTPUT ON
DECLARE
    szam1 NUMBER(2) := 1;
    szam2 NUMBER(2);
    szam3 NUMBER(2) := 3;
    PROCEDURE muvelet (inp IN NUMBER, outp OUT
NUMBER, inout IN OUT NUMBER) IS
    BEGIN
        dbms_output.put_line('in parameter: '
|| inp || ', in out parameter: ' || inout);
        outp := inp + inout;
        inout := outp + inp;
    END muvelet;
BEGIN
    muvelet(szam1, szam2, szam3);
    dbms_output.put_line('out parameter: ' ||
szam2 || ', in out parameter: ' || szam3);
END;
```

PL/SQL - Alprogramok

- Hatáskör-, és élettartamkezelés
 - Statikus (egy név csak a deklarációjától kezdve él)
 - Dinamikus (alprogramok és blokkok esetén)

PL/SQL - Alprogramok

- Tárolt alprogramok
 - Van lehetőség arra, hogy létrehozzunk tárolt eljárást/függvényt
 - Ekkor azt az sqldeveloper eltárolja, később hívható lesz
 - Ez jó az újrafelhasználhatóság szempontjából

PL/SQL - Alprogramok

➤ Tárolt eljárás létrehozása

```
CREATE [OR REPLACE] PROCEDURE név  
[formális paraméterlista]  
IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név];
```

PL/SQL - Alprogramok

➤ Tárolt függvény létrehozása

```
CREATE [OR REPLACE] FUNCTION név  
[formális paraméterlista]  
RETURN típus IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név];
```

PL/SQL - Alprogramok

- Tárolt alprogram újrafordítása

```
ALTER {PROCEDURE | FUNCTION} név  
    COMPILE [DEBUG];
```

- Tárolt alprogram törlése

```
DROP {PROCEDURE | FUNCTION} név;
```

PL/SQL - Alprogramok

- Tárolt alprogram meghívása

```
CALL alprogram_név([aktuális  
paraméterlista])  
    [INTO változó];
```

PL/SQL - Kivételkezelés

- Futás közbeni hibák kezelésére
- Két fajta kivétel
 - Beépített
 - Felhasználó által definiált

PL/SQL - Kivételkezelés

- Kivételkezelés szintaxisa
[DECLARE deklarációs utasítások]
BEGIN végrehajtandó utasítások
EXCEPTION
 WHEN exception1 THEN végrehajtandó
 utasítások exception1 esetén
 WHEN exception2 THEN végrehajtandó
 utasítások exception2 esetén
 WHEN exception3 THEN végrehajtandó
 utasítások exception3 esetén
 .
 .
 .
 WHEN others THEN végrehajtandó
 utasítások egyéb esetben
END;

PL/SQL - Kivételkezelés

- Példa: Lekérdezzük a dolgozó nevét, amennyiben nincs ilyen kódú: 'Nincs ilyen dolgozo', egyéb hiba esetén a 'Hiba' hibaüzenetet adjuk.

```
SET SERVEROUTPUT ON
DECLARE
    kod Dolgozo.dkod%TYPE;
    nev Dolgozo.dnev%TYPE;
BEGIN
    SELECT dkod, dnev
    INTO kod, nev
    FROM Dolgozo
    WHERE dkod=kod;
    dbms_output.put_line(kod);
    dbms_output.put_line(nev);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Nincs ilyen kodu
dolgozo');
    WHEN OTHERS THEN
        dbms_output.put_line('Hiba');
END;
```

PL/SQL - Kivételkezelés

- Saját kivétel definiálása

```
DECLARE  
    saját_kivete1 EXCEPTION;
```

- Kivétel hívás

```
RAISE saját_kivete1_neve;
```


PL/SQL - Kivételkezelés

- Példa: amennyiben a bekért változó értéke negatív, dobunk egy `negativ_ertek` kivételt, majd kezeljük azt egy üzenettel. Ha nem történt hiba, kiírjuk a számot.

```
SET SERVEROUTPUT ON
DECLARE
    negativ_ertek EXCEPTION;
    szam NUMBER := &szam;
BEGIN
    IF (szam < 0) THEN
        RAISE negativ_ertek;
    END IF;
    dbms_output.put_line(szam);
EXCEPTION
    WHEN negativ_ertek THEN
        dbms_output.put_line('A szam nem lehet negativ!');
    WHEN OTHERS THEN
        dbms_output.put_line('Hiba');
END;
```

Eljut-feladat (monoton rekurzió) megoldása PL/SQL programmal

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

➤ **Jaratok**(legitarsasag, honnan, hova, koltseg, indulas, erkezes) táblában repülőjáratok adatait tároljuk.

➤ A járatok táblát létrehozó script:

http://people.inf.elte.hu/sila/eduAB/jaratok_tabla.txt

➤ **Mely (x,y) párokra lehet eljutni x városból y városba?**

➤ Ezt egy relációs algebrai kifejezésként nem tudjuk megadni zárt alakban, klasszikus SQL SELECT utasítással sem tudjuk kifejezni, csak azt tudjuk, hogy átszállás nélkül, egy, két, stb... átszállással:

Az Eljut-feladatnak nincs algebrai megoldása

```
select distinct honnan, hova  
  from jaratok
```

union

```
select j1.honnan, j2.hova  
  from jaratok j1, jaratok j2  
  where j1.hova=j2.honnan
```

union

```
select j1.honnan, j3.hova  
  from jaratok j1, jaratok j2, jaratok j3  
  where j1.hova=j2.honnan  
  and j2.hova=j3.honnan
```

--- union stb... Ezt így nem lehet felírni...

Az Eljut feladat SQL-99 szabványban

- Datalog **LINEÁRIS, MONOTON** rekurzió átírható:
Eljut(x, y) <- Jaratok(l, x, y, k, i, e)
Eljut(x, y) <- Eljut(x, z) AND Jaratok(l, z, y, k, i, e)
- Hova, mely városokba tudunk eljutni Budapestről?

WITH RECURSIVE Eljut AS

(SELECT honnan, hova FROM Jaratok

UNION

SELECT Eljut.honnan, Jaratok.hova

FROM Eljut, Jaratok

WHERE Eljut.hova = Jaratok.honnan)

SELECT hova **FROM Eljut** WHERE honnan='Bp';

Rekurzív lekérdezések

- **Datalog rekurzió** segít megérteni az SQL-99 szabványban bevezetett **rekurzív lekérdezések WITH RECURSIVE** záradékát.
- A BSc-n **csak MONOTON rekurziót** vesszük, vagyis nem használjuk nem-monoton különbség műveletet, nincs csoportosítás-aggregálás (ugyanis az olyan lekérdezések, amelyek nem-monotonok, megengedik a negációt és aggregálást az olyan különös hatással van a rekurzióra, ezt csak MSc kurzusokon vesszük).
- **Nézzük meg a monoton rekurzív Eljut-feladatnak a megvalósítását Oracle PL/SQL programmal**

Eljut feladat PL/SQL-ben ---1

- **Rek1.feladat:** Mely (x, y) várospárokra lehet egy vagy több átszállással eljutni x városból y városba?
- Ehhez hozzuk létre eljut(honnan,hova) táblát,
DROP TABLE eljut;
CREATE TABLE eljut(
 honnan VARCHAR2(10),
 hova VARCHAR2(10));
- Írjunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát a sorait a járatok tábla alapján (ehhez ciklust szervezni, az insert több sor felvitele 2.alakja alkérdéssel járatok és eljut táblák alapján)

Eljut feladat PL/SQL-ben ---2

- Az ELJUT feladat megoldása Oracle PL/SQL-ben
- A ciklus során ellenőrizni kell, hogy addig hajtsuk végre a ciklust, amíg növekszik az eredmény (Számláló)
- **DECLARE** RegiSzamlalo Integer;
UjSzamlalo Integer;
- Deklarációs rész után **BEGIN ... END;** között az utasítások, először az eljut táblának kezdeti értéket adunk (a megvalósításnál az INSERT-nél figyelni, hogy ne legyenek ismétlődő sorok: `select distinct delete from eljut;`
`insert into eljut (SELECT distinct honnan, hova FROM jaratok);`

Eljut feladat PL/SQL-ben ---3

- Szamlalo változóknak adunk kiindulási értéket:

```
RegiSzamlalo := 0;
```

```
select count(*) into UjSzamlalo from eljut;
```

- A ciklust addig kell végrehajtani, amíg növekszik az eredmény (Szamlalo) duplikátumokra figyelni!

LOOP

```
insert into eljut (lásd a köv.oldalon...)
```

```
select count(*) into UjSzamlalo from eljut;
```

```
EXIT WHEN UjSzamlalo = RegiSzamlalo;
```

```
RegiSzamlalo := UjSzamlalo;
```

```
END LOOP;
```

```
commit;
```


Eljut feladat PL/SQL-ben ---4

- Az eljut tábla növelése a ciklusban, figyelni kell a duplikátumokra, csak olyan várospárokat vegyünk az eredményhez, ami még nem volt!

insert into eljut

```
(select distinct eljut.honnan, jaratok.hova  
from eljut, jaratok --- *from (lineáris rekurzió)  
where eljut.hova = jaratok.honnan  
and (eljut.honnan,jaratok.hova)  
    NOT IN (select * from eljut));
```

- Megjegyzés: PSM-ben a **nem-lineáris rekurzió** is megengedett: **from eljut e1, eljut e2 ---*from-ban**

Eljut feladat PL/SQL-ben ---5

- **Rek2.feladat:** Mely (x,y) város párokra hány darab átszállással és milyen költségekkel lehetséges egy vagy több átszállással eljutni x városból y városba?
- Ehhez készítsünk Eljut2(honnan, hova, atszallas, koltseg) táblát. Írjunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát.
- **Rek3.feladat:** Tegyük fel, hogy nemcsak az érdekel, hogy el tudunk-e jutni az egyik városból a másikba, hanem az is, hogy utazásunk során az átszállások is ésszerűek legyenek, ami azt jelenti, hogy ha több járattal utazunk, akkor nézni kell átszálláskor az érkező járatnak legalább egy órával a rákövetkező indulás előtt meg kell érkeznie, és 6 óránál ne kelljen többet várnia.

Kérdés/Válasz

- Köszönöm a figyelmet!
- Kérdés/Válasz?

- Gyakorlás az Oracle Példatár feladatai:
 - 8-10.fejezet PL/SQL példái és feladatai