Adatbázisok

10. labor:

PL/SQL

Bevezetés a PL/SQL-be

- procedurális programnyelv
- blokk részei:
  - deklarációs rész (opcionális)
  - végrehajtható rész
  - kivételkezelő rész (opcionális)

```
[DECLARE
     változók deklarálása]
BEGIN
     utasítások
[EXCEPTION
     kivételkezelés ]
END;
/
```

Első lépések

```
SET serveroutput ON
BEGIN
   dbms_output.put_line('Hello World!');
END;
```

- Output beállítások: egy munkamenetben csak egyszer kell kiadni a parancsot
  - SET serveroutput ON
  - SET verify OFF

Feladat

- Kérd be a felhasználó nevét, majd üdvözöld név szerint!

- ACCEPT!
  - Nem a PL/SQL-hez tartozik az utasítás, hanem az SQL parancssorhoz.
  - ACCEPT nev
  - ACCEPT nev PROMPT 'Kérem a neved:'
  - változó használata: &nev

Megoldás

```
ACCEPT nev
  PROMPT 'Kérem a neved:'
BEGIN
  dbms_output.put_line('Hello '
  || '&nev');
END;
```

Megoldás 2

```
BEGIN
dbms_output.put_line('Hello '||'&n');
END;
```

- Így is bekéri, de így csak lokális!
- Különbség:
  - ACCEPT-tel a teljes munkamenetre megmarad az érték
- Definiálás és lekérdezés: DEFINE
  Törlés: UNDEFINE
- Összefűzés nem szükséges: `'Hello &n'`

Változók

```
ACCEPT x PROMPT 'Kérem a számot:'
DECLARE
  szam NUMBER;
BEGIN
  szam := &x;
  dbms_output.put_line(szam ||
  ' négyzete: ' ||szam*szam);
END;
```

Elágazás

```
IF feltétel THEN
    ezt csináljuk
ELSIF másikfeltétel THEN
    mást csinálunk
ELSE
    különben ezt csináljuk
END IF;
```

Ciklusok

```
DECLARE
  tol NUMBER :=1;
  ig  NUMBER :=10;
BEGIN
  LOOP
      dbms_output.put_line(tol);
      EXIT WHEN tol=ig;
      tol:=tol+1;
  END LOOP;
END;
```

For és While

```
WHILE feltétel
LOOP
   ....
END LOOP;
```

```
FOR i IN tól..ig
LOOP
      ....
END LOOP;
```

Példa

```
DECLARE
  tol NUMBER;
  ig  NUMBER;
BEGIN
  tol:=&mettol;
  ig:=&meddig;
  FOR i in tol..ig
  LOOP
     IF MOD(i,2)=1 THEN
           dbms_output.put(i || ', ');
     END IF;
  END LOOP;
  dbms_output.put_line('');
END;
```

Lekérdezés PL/SQL blokkban

```
ACCEPT rnev PROMPT 'Kérem a részleg nevét:'
DECLARE
  átlag NUMBER;
BEGIN
SELECT AVG(salary) INTO átlag
FROM employees INNER JOIN departments
  USING (department_id)
WHERE LOWER(department_name)=LOWER('&rnev');
dbms_output.put_line('A &rnev részleg átlaga
  ' || átlag);
END;
```

KURZOROK

Implicit kurzor

```
DECLARE
   egysor employees%ROWTYPE;
BEGIN
  FOR egysor IN (SELECT * FROM employees)
  LOOP
    dbms_output.put('Név: ' || egysor.last_name);
    dbms_output.put_line(',  Fizetés: ' ||
   egysor.salary);
  END LOOP;
END;
```

Explicit kurzor

```
DECLARE
  CURSOR kurzor IS select * from employees;
  rekord employees%ROWTYPE;
BEGIN
  FOR rekord IN kurzor
  LOOP
    IF rekord.salary BETWEEN 10000 AND 15000
    THEN
     dbms_output.put_line(rekord.last_name||
  ',   ' || rekord.salary);
    END IF;
  END LOOP;
END;
```

- A rekord típusa kurzor%rowtype is lehetne.

Kurzor használata módosításra

- CURSOR kurzor IS
  SELECT * FROM employees2
  FOR UPDATE [OF salary] [NOWAIT];

  - FOR UPDATE: zárolja az érintett sorokat, a következő COMMIT vagy ROLLBACK utasításkor oldja fel
  - NOWAIT: nem várakozik erőforrásokra

Kurzor használata módosításra

```
...
BEGIN
  FOR rekord IN kurzor
   LOOP
      fizetes := rekord.salary * 1.2;
      UPDATE employees2 SET
      salary = fizetes
      WHERE CURRENT OF kurzor;
  END LOOP;
END;
```

Tárolt eljárások, függvények

- Már ismert koncepció: "metódus"
  - eljárás: nincs visszatérési érték, PROCEDURE
  - függvény: van visszatérési érték, FUNCTION

PROCEDURE

```
CREATE OR REPLACE PROCEDURE
  eljárásnév (paraméterlista)
IS
  lokális változók
BEGIN
  utasítások
EXCEPTION …
END;
```

PROCEDURE

- Futtatása PL/SQL blokkon belül: szokásos
- Futtatása PL/SQL blokkon kívülről:
  EXECUTE eljárásnév;
- Törlés:
  DROP PROCEDURE eljárásnév;
- <OR REPLACE> szerepe: ha már létezik, akkor lecseréli

Példa

```
CREATE PROCEDURE osszead
                (a NUMBER, b NUMBER) IS
BEGIN
  dbms_output.put_line('Összegük: '||
  (a+b));
END;
```

- Teszt:

  execute osszead(2,3)

  begin osszead(2,3); end;

Feladat

- Készítsünk eljárást, amely paraméterként egy dolgozó vezetéknevét kapja, és ha van ilyen dolgozó, akkor annak a fizetését megemeli 10%-kal.

- CREATE TABLE dolgozo AS
- SELECT * FROM employees;

Feladat

- Készítsünk eljárást, amely egy szöveges paraméterrel rendelkezik, ez legyen egy részlegnév.
  Az eljárás írja ki az adott részleg összfizetését, illetve ha nincs ilyen nevű részleg, akkor pedig a 'Nem létező részleg!' szöveget.

FUNCTION

```
CREATE OR REPLACE FUNCTION
     függvénynév (paraméterlista)
RETURN típus
IS
  lokális változók
BEGIN
  utasítások
EXCEPTION …
END;
```

Példa

```
CREATE OR REPLACE FUNCTION maxfizu
(dn NUMBER) RETURN NUMBER IS
m NUMBER;
BEGIN
SELECT MAX(salary) INTO m
FROM employees WHERE department_id=dn;
RETURN m;
END;
```

Teszt:

```
select department_name, maxfizu(department_id)
from departments;
```

Feladat

- Készítsünk függvényt, amelynek a paramétere egy dolgozó vezetékneve, a visszatérési értéke pedig a dolgozó részlegének neve. Ha nincs megadott nevű dolgozó, akkor a 'Nincs ilyen' szöveget adja vissza.

Feladat

- Töröljük a létrehozott eljárásokat és függvényeket.

```
DROP PROCEDURE osszead;
DROP FUNCTION maxfizu;
...
```

Triggerek

- Adott esemény bekövetkezésére reagálunk (tárolt eljárást futtat)
- Milyen műveletek:
  INSERT, UPDATE, DELETE, CREATE, ALTER, DROP
- Mikor: BEFORE, AFTER
  - INSTEAD OF (csak nézetekre, csak soronként; ezzel oldható meg DML azokra a nézetekre, amikre amúgy nem lehetne)
- Az ALTER TRIGGER a nézethez hasonlóan nem használható a definíció megváltoztatására (csak enable, disable, compile)

Triggerek

```
CREATE OR REPLACE TRIGGER  {név}
   [BEFORE | AFTER | INSTEAD OF]
   {esemény} [OR {esemény} …]
   ON {tábla}
[FOR EACH ROW [WHEN {feltétel}]]
[DECLARE {változók}]
BEGIN
   {utasítások}
[EXCEPTION …]
END;
```

Egyszerű trigger

```
CREATE TRIGGER Jelzo
AFTER INSERT ON departments
BEGIN
  dbms_output.put_line('Új részleg
  beszúrva!');
END;
```

Trigger példa

- Készítsünk egy triggert, amely
  - az employees táblán végzett
  - beszúrás és törlés előtt fut le;
  - minden érintett sorra
  - kiírja az érintett dolgozó vezetéknevét
  - és a rajta végzett műveletet (beszúrás vagy törlés)

Trigger példa

```
CREATE OR REPLACE TRIGGER Naplozo
BEFORE DELETE OR INSERT ON employees
FOR EACH ROW BEGIN
  IF INSERTING THEN
     dbms_output.put_line('Uj: '||
     :NEW.last_name);
  ELSIF DELETING THEN
     dbms_output.put_line('Töröl: '||
     :OLD.last_name);
  END IF;
END;
```

Magyarázat a példához

- A :OLD és :NEW csak sor szintű triggerben érhetők el;
  ha nincs értelmük, akkor az értékük NULL (pl. beszúrásnál az OLD, törlésnél a NEW értékei)
- A WHEN feltételben is használhatók, de ott nem kell eléjük kettőspont.

Feladat

- Készítsünk triggert, amely UPDATE művelet után kiírja az érintett dolgozók vezetéknevét, régi és új fizetésüket.
  - Vigyázat: egy UPDATE művelet több sort is módosíthat egyszerre, a kiírást minden sorra el kell végezni!

Megoldás

```
CREATE OR REPLACE TRIGGER Naplozo2
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
  dbms_output.put_line('Dolgozó: '||
  :OLD.last_name);
  dbms_output.put_line('Régi és új
  fizetése: '|| :OLD.salary || ' - '
  || :NEW.salary);
END;
```

Feladat

- Továbbfejlesztés: csak akkor írjon ki bármit, ha a fizetés értéke megváltozott.
  - TIPP: a fizetés régi és új értékét a FOR EACH ROW WHEN feltételben is vizsgálhatjuk

Feladat

- Módosítsuk az előző példát úgy, hogy az UPDATE előtt fusson le, és amennyiben december van, akkor írja ki azt is, hogy „Kellemes ünnepeket!”
  - TIPP: az aktuális dátum lekérdezhető a **sysdate** változó segítségével