

# SQL/PSM kurzorok --- 2.rész

Tankönyv: Ullman-Widom:  
Adatbázisrendszerek Alapvetés  
Második, átdolgozott kiadás,  
Panem, 2009

---

---



9.3. Az SQL és a befogadó nyelv  
közötti felület (sormutatók)  
9.4. SQL/PSM Sémában  
tárolt függvények és eljárások

-- volt: PSM1modulok: utasítások, modulok, PSM-kivételek  
-- most: PSM2kurzorok: lekérdezések PSM-ben

# Lekérdezések használata a PSM-ben

- **A típuseltérés problémája:** Az SQL magját a relációs adatmodell képezi. Tábla – gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. 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<sub>1</sub>, ... vált<sub>n</sub>  
--- 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.

# 1. Example: Assignment/Query

- Using local variable  $p$  and **Sells(bar, beer, price)**, we can get the price Joe charges for Bud by:

```
SET p = (SELECT price FROM Sells
        WHERE bar = 'Joe''s Bar' AND
               beer = 'Bud');
```

## 2. SELECT ... INTO

- Another way to get the value of a query that returns one tuple is by placing **INTO <variable>** after the SELECT clause.
- **Example:**

```
SELECT price INTO p FROM Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

## 3. Cursors

- Ha a SELECT eredménye több sorral tér vissza, akkor valamilyen ciklussal járjuk be az eredmény sorait
- A *cursor* is essentially a tuple-variable that ranges over all tuples in the result of some query.
- Declare a cursor *c* by:  
**DECLARE** sormutató **CURSOR**  
**FOR** (lekérdezés);

# Opening and Closing Cursors

- To use cursor *c*, we must issue the command:  
**OPEN** sormutató;
- Hatására a rendszer a lekérdezést kiértékeli és hozzáférhető lesz a lekérdezés eredménye, ehhez a bejáráshoz egy ciklust kell indítani, és a sormutató az eredmény első sorára mutat
- (ezt a ciklust lásd a következő oldalon)
- When finished with *c*, issue command:  
**CLOSE** sormutató;

# Fetching Tuples From a Cursor

## I: LOOP

- To get the next tuple from cursor *c*, issue command:

**FETCH FROM** sormutató **INTO** *v*<sub>1</sub>, ..., *v*<sub>*n*</sub>;

- The *v*'s are a list of variables, one for each component of the tuples referred to by *c*.
- *c* is moved automatically to the next tuple.

**IF** „ellenőrzés: kaptunk-e új sort?”

**THEN LEAVE I**

**END IF;**

**ENDLOOP;**



# Breaking Cursor Loops

- The usual way to use a cursor is to create a loop with a **FETCH** statement, and do something with each tuple fetched.
- A tricky point is how we get out of the loop when the cursor has no more tuples to deliver.
- Each SQL operation returns a *status*, which is a 5-digit character string.
  - For example, 00000 = “Everything OK,”  
and 02000 = “Failed to find a tuple.”
- In PSM, we can get the value of the status in a variable called **SQLSTATE**.

# Breaking Cursor Loops

- We may declare a *condition*, which is a boolean variable that is true if and only if SQLSTATE has a particular value.
- **Example:** We can declare condition `NotFound` to represent 02000 by:

```
DECLARE NotFound CONDITION FOR  
SQLSTATE '02000';
```

```
DECLARE <name> CONDITION FOR  
SQLSTATE <value>;
```

# Breaking Cursor Loops

- The structure of a cursor loop is thus:

```
cursorLoop: LOOP
```

```
...
```

```
  FETCH c INTO ... ;
```

```
  IF NotFound THEN LEAVE cursorLoop;
```

```
  END IF;
```

```
...
```

```
END LOOP;
```

# Example: Cursor

- Let's write a procedure that examines `Sells(bar, beer, price)`, and raises by \$1 the price of all beers at Joe's Bar that are under \$3.
- Yes, we could write this as a simple UPDATE, but the details are instructive anyway.

# The Needed Declarations

```
CREATE PROCEDURE JoeGouge( )  
  DECLARE theBeer CHAR(20);  
  DECLARE thePrice REAL;  
  DECLARE NotFound CONDITION FOR  
    SQLSTATE '02000';  
  DECLARE c CURSOR FOR  
    (SELECT beer, price FROM Sells  
     WHERE bar = 'Joe''s Bar');
```

-- Used to hold  
-- beer-price pairs  
-- when fetching  
-- through cursor c

-- Returns Joe's menu

# The Procedure Body

BEGIN

OPEN c;

menuLoop: LOOP

FETCH c INTO theBeer, thePrice;

Check if the recent  
FETCH failed to  
get a tuple

IF NotFound THEN LEAVE menuLoop END IF;

IF thePrice < 3.00 THEN

UPDATE Sells SET price = thePrice + 1.00

WHERE bar = 'Joe's Bar' AND beer = theBeer;

END IF;

END LOOP;

CLOSE c;

END;

If Joe charges less than \$3 for  
the beer, raise its price at  
Joe's Bar by \$1.

# PL/SQL különbségek

- In addition to the SQL types, NUMBER can be used to mean INT or REAL, as appropriate.
- You can refer to the type of attribute  $x$  of relation  $R$  by  $R.x\%TYPE$ .
  - Useful to avoid type mismatches.
  - Also,  $R\%ROWTYPE$  is a tuple whose components have the types of  $R$ 's attributes.

# PL/SQL Cursors

- The form of a PL/SQL cursor declaration is:  
CURSOR <name> IS <query>;
- To fetch from cursor c, say:  
FETCH c INTO <variable(s)>;



# Example: JoeGouge() in PL/SQL

- Recall `JoeGouge()` sends a cursor through the Joe's-Bar portion of `Sells`, and raises by \$1 the price of each beer Joe's Bar sells, if that price was initially under \$3.

# Example: JoeGouge() Declarations

```
CREATE OR REPLACE PROCEDURE
    JoeGouge () AS
    theBeer Sells.beer%TYPE;
    thePrice Sells.price%TYPE;
    CURSOR c IS
        SELECT beer, price FROM Sells
        WHERE bar = 'Joe''s Bar';
```

# Example: JoeGouge() Body

```
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO theBeer, thePrice;
    EXIT WHEN c%NOTFOUND;
    IF thePrice < 3.00 THEN
      UPDATE Sells SET price = thePrice + 1.00;
      WHERE bar = Joe''s Bar' AND beer = theBeer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

How PL/SQL  
breaks a cursor  
loop

Note this is a SET clause  
in an UPDATE, not an assignment.  
PL/SQL uses := for assignments.

# Tuple-Valued Variables

- PL/SQL allows a variable  $x$  to have a tuple type.
- $x \text{ R}\%ROWTYPE$  gives  $x$  the type of  $R$ 's tuples.
- $R$  could be either a relation or a cursor.
- $x.a$  gives the value of the component for attribute  $a$  in the tuple  $x$ .

# Example: Tuple Type

- Repeat of JoeGouge() declarations with variable *bp* of type beer-price pairs.

```
CREATE OR REPLACE PROCEDURE
    JoeGouge () AS
    CURSOR c IS
    SELECT beer, price FROM Sells
    WHERE bar = 'Joe''s Bar';
    bp c%ROWTYPE;
```

# JoeGouge() Body Using *bp*

```
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO bp;
    EXIT WHEN c%NOTFOUND;
    IF bp.price < 3.00 THEN
      UPDATE Sells SET price = bp.price + 1.00
      WHERE bar = 'Joe''s Bar' AND beer =bp.beer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

Components of bp are obtained with a dot and the attribute name

