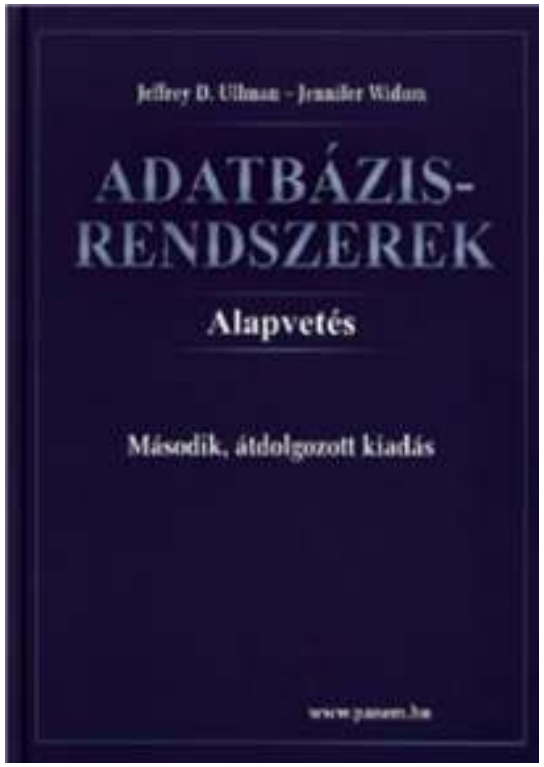
Tankönyv: SQL/PSM + Gyak:PL/SQL

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



10.2. Rekurzió, az „Eljut”-feladat Oracle-ben és PL/SQL-ben

9.3. Az SQL és a befogadó nyelv közötti felület (sormutatók)

9.4. SQL/PSM Sémában tárolt függvények és eljárások

Az „Eljut feladat” SQL-99 szabványban

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) táblában repülőjáratok adatait tároljuk. Mely városokba tudunk eljutni Bp-ről?
-
```
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';
```

Oracle megoldások: WITH utasítással

- Az Oracle SQL a WITH RECURSIVE utasítást nem támogatja, ott másképpen oldották meg WITH utasítással (Oracle 11gR2 verziótól)
- with eljut (honnan, hova) as
  (select honnan, hova from jaratok
  union all
  select jaratok.honnan, eljut.hova
  from jaratok, eljut
  where jaratok.hova=eljut.honnan
)
  SEARCH DEPTH FIRST BY honnan SET SORTING
  CYCLE honnan SET is_cycle TO 1 DEFAULT 0
  select distinct honnan, hova from eljut order by honnan;

Oracle megoldások: connect by

- SELECT DISTINCT hova FROM jaratok
  WHERE HOVA <> 'DAL'
  START WITH honnan = 'DAL'
  CONNECT BY NOCYCLE PRIOR hova = honnan;

- SELECT LPAD(' ', 4*level) || honnan, hova,
  level-1 Atszallasok,
  sys_connect_by_path(honnan||'->'||hova, '/'),
  connect_by_isleaf, connect_by_iscycle
  FROM jaratok
  START WITH honnan = 'SF'
  CONNECT BY NOCYCLE PRIOR hova = honnan;

Rekurzív Eljut feladat PSM-ben ---1

- Az ELJUT feladatot a gyakorlaton oldjuk meg Oracle PL/SQL-ben, itt csak a vázlata PSM-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);

Rekurzív Eljut feladat PSM-ben ---2

- Szamlalo változóknak adunk kiindulási értéket:

```
SET 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;
    SET RegiSzamlalo = UjSzamlalo;
END LOOP;
```

Rekurzív Eljut feladat PSM-ben ---3

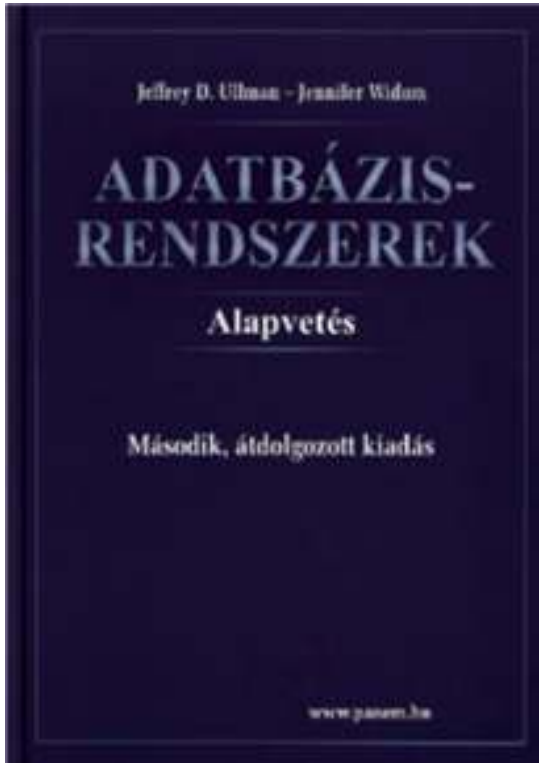
- Az eljut tábla növelése a ciklusban, figyelni kell a duplikátumokra, csak olyan várospárokat vegyünk az eredményhez, ami még nem volt!

```
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

SQL/PSM tárolt modulok

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

-- itt: PSM1modulok: utasítások, modulok, PSM-kivételek
-- később lesz: PSM2kurzorok: lekérdezések PSM-ben

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. Tábla – gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. A lekérdezés eredménye hogyan használható fel? Megoldás: kurzorral, az eredmény soronkénti bejárása.

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)

PSM – Persistent Stored Procedures

- SQL/PSM is a part of the latest revision to the SQL standard, called SQL:2003
- PSM, or "*persistent stored modules*," allows us to store procedures as database schema elements.
- PSM = a mixture of conventional statements (if, while, etc.) and SQL statements.
- Lets us do things we cannot do in SQL alone.
- Each commercial DBMS offers its own extension of PSM, e.g. Oracle: PL/SQL (see, in practice)

PSM tárolt eljárások

- Tárolt eljárások (SQL objektumok)

```
CREATE PROCEDURE eljárás-név (
      paraméter-lista )
[DECLARE … deklarációk]
BEGIN
      az eljárás utasításai;
 END;
```

- Paraméter lista (tárolt eljárásban)

```
[IN | OUT | INOUT] paraméternév  értéktípus
```

PSM eljárások paraméterei

- Mód – Név – Típus hármas
- The parameters of a PSM procedure:
  Unlike the usual name-type, PSM uses mode-name-type triples, where the *mode* can be:
  - IN = procedure uses value, does not change value.
  - OUT = procedure changes, does not use.
  - INOUT = both.

PSM tárolt függvények

- Tárolt függvények

  CREATE FUNCTION függvény-név (
  paraméter-lista) RETURNS értéktípus
  [DECLARE … deklarációk]
  BEGIN
  utasítások …
  END;

- Függvények paraméterei:
  - may only be of mode IN

    (PSM forbids side-effects in functions)

Példa: Stored Procedure

- Let's write a procedure that takes two arguments *b* and *p*, and adds a tuple to Sells(bar, beer, price) that has bar = 'Joe''s Bar', beer = *b*, and price = *p*.
  - Used by Joe to add to his menu more easily.

```
CREATE PROCEDURE JoeMenu (
IN  b  CHAR(20),
IN  p  REAL
)
INSERT INTO Sells
VALUES('Joe''s Bar', b, p);
```

Parameters are both read-only, not changed

The body a single insertion

Legfontosabb utasítások --- 1

1. Eljáráshívás: The call statement

```
CALL <procedure name>(<argument
list>);
```

Use SQL/PSM statement CALL, with the name of the desired procedure and arguments.

- Example:

```
CALL JoeMenu('Moosedrool', 5.00);
```

Legfontosabb utasítások --- 2

2. The return statement

Függvényhívás: Functions used in SQL expressions wherever a value of their return type is appropriate.

RETURN <expression> sets the return value of a function.

- Unlike C, etc., RETURN *does not* terminate function execution.

3. Változók deklarálása

DECLARE <name> <type>

used to declare local variables.

Legfontosabb utasítások --- 3

4. Értékadás - Assignment statements

SET <variable> = <expression>;

- Example: `SET b = 'Bud';`

5. Statement group

BEGIN . . . END for groups of statements.

- Separate statements by semicolons.

6. Statement labels

- give a statement a label by prefixing a name and a colon.

7. SQL utasítások

- DELETE, UPDATE, INSERT, MERGE
- (de SELECT nem, azt később nézzük)

IF Statements

- Simplest form:
  IF <condition> THEN
  <statements(s)>
  END IF;
- Add ELSE <statement(s)> if desired, as
  IF . . . THEN . . . ELSE . . . END IF;
- Add additional cases by ELSEIF <statements(s)>:
  IF … THEN … ELSEIF … THEN …
  ELSEIF … THEN … ELSE … END IF;

Example: IF

- Let's rate bars by how many customers they have, based on Frequents(drinker,bar).
  - <100 customers: 'unpopular'.
  - 100-199 customers: 'average'.
  - >= 200 customers: 'popular'.
- Function Rate(b) rates bar b.

Example: IF (continued)

```
CREATE FUNCTION Rate (IN b CHAR(20) )
        RETURNS CHAR(10)
        DECLARE cust INTEGER;
    BEGIN
        SET cust = (SELECT COUNT(*)
                    FROM  Frequents    WHERE bar = b);
        IF cust < 100 THEN RETURN 'unpopular'
        ELSEIF cust < 200 THEN RETURN 'average'
        ELSE RETURN 'popular'
        END IF;
END;
```

-- Number of
-- customers of
-- bar b

-- Return occurs here,
-- not at one of the RETURN --
statements

-- Nested
-- IF statement

Ciklusok

- Basic form:

  <loop label>: LOOP <statements>
  END LOOP;

- Exit from a loop by:

  LEAVE <loop label>

Example: Exiting a Loop

```
címke: LOOP
        . . .
        LEAVE címke;
        . . .
END LOOP;
```

Other Loop Forms

- WHILE <condition>
  DO <statements>
  END WHILE;

- REPEAT <statements>
  UNTIL <condition>
  END REPEAT;

PSM kivételek

- Az SQL-rendszer a hibákat egy ötjegyű SQLSTATE nevű karakterlánc beállításával jelzi, például ‘02000’ jelzi, hogy nem talált sort.
- Kivételek kezelése, kivételek nem feltétlen hiba, hanem a normálistól való eltérés kezelése
- DECLARE <hova menjen>

  HANDLER FOR <feltétel lista>

  <utasítás>
- <hova menjen> lehetőségek:

  CONTINUE, EXIT, UNDO

Tankönyv 9.16 példa (9.18 ábra)

PL/SQL

- Oracle uses a variant of SQL/PSM which it calls PL/SQL.
- PL/SQL not only allows you to create and store procedures or functions, but it can be run from the *generic query interface* (sqlplus), like any SQL statement.
- Triggers are a part of PL/SQL.

Trigger Differences

- Compared with SQL standard triggers, Oracle has the following differences:
  1. Action is a PL/SQL statement.
  2. New/old tuples referenced automatically.
  3. Strong constraints on trigger actions designed to make certain you can't fire off an infinite sequence of triggers.

SQLPlus

- In addition to stored procedures, one can write a PL/SQL statement that looks like the body of a procedure, but is executed once, like any SQL statement typed to the generic interface.
  - Oracle calls the generic interface “sqlplus.”
  - PL/SQL is really the “plus.”

Form of PL/SQL Statements

```
DECLARE
   <declarations>
BEGIN
   <statements>
END;
.
run
```

- The DECLARE section is optional.

Form of PL/SQL Procedure

```
CREATE OR REPLACE PROCEDURE
   <name> (<arguments>) AS
   <optional declarations>
   BEGIN
        <PL/SQL statements>
   END;
.
run
```

Notice AS needed here

Needed to store procedure in database; does not really run it.

PL/SQL Declarations és Assignments

- The word DECLARE does not appear in front of each local declaration.
  - Just use the variable name and its type.
- There is no word SET in assignments, and := is used in place of =.
  - Example: x := y;

PL/SQL Procedure Parameters

- There are several differences in the forms of PL/SQL argument or local-variable declarations, compared with the SQL/PSM standard:
  1. Order is name-mode-type, not mode-name-type.
  2. INOUT is replaced by IN OUT in PL/SQL.
  3. Several new types.

PL/SQL Types

- 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.

Example:JoeMenu

- Recall the procedure JoeMenu(b,p) that adds beer *b* at price *p* to the beers sold by Joe (in relation Sells).
- Here is the PL/SQL version.

```
CREATE OR REPLACE PROCEDURE JoeMenu (
    b IN Sells.beer%TYPE,
    p IN Sells.price%TYPE
) AS
BEGIN
    INSERT INTO Sells
    VALUES ('Joe''s Bar', b, p);
END;
run
```

PL/SQL Branching Statements

- Like IF … in SQL/PSM, but:
- Use ELSIF in place of ELSEIF.
- Viz.: IF … THEN … ELSIF … THEN … ELSIF … THEN … ELSE … END IF;

PL/SQL Loops

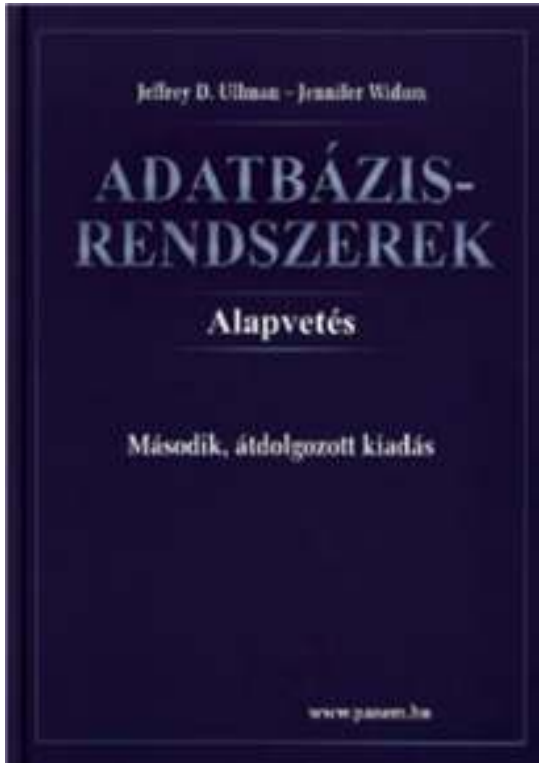
- LOOP … END LOOP as in SQL/PSM.
- Instead of LEAVE … , PL/SQL uses
  EXIT WHEN <condition>
- And when the condition is that cursor *c* has found no tuple, we can write c%NOTFOUND as the condition.

Tuple-Valued Variables

- PL/SQL allows a variable *x* to have a tuple type.
- x 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*.

SQL/PSM kurzorok

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, …, e_n INTO vált$_1$, … 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.

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 v1, …,v*n*;

- 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;
        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;
```

Check if the recent FETCH failed to get a tuple

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 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