CIS 560 – Database System Concepts

Lecture 15

Programming Inside the DBMS (Section 9.4)

September 30, 2013

Credits for slides: Chang, Ullman.

Stored Procedures

- An extension to SQL, called SQL/PSM, or "persistent, stored modules," allows us to store procedures as database schema elements.
- The programming style is a mixture of conventional statements (if, while, etc.) and SQL.
- Let us do things we cannot do in SQL alone.
- Why?
  - Performance (less information sent between server and client -> increased load for the DB server)
  - More secure (operations are properly logged).
- They are harder to develop and maintain.

Basic PSM Format

Procedure:

CREATE PROCEDURE <name> (<parameter list>)
  <optional local declarations>
  <body>;

Example: compute square footage of house lot; then you can query on it

Function:

CREATE FUNCTION <name> (<parameter list>) RETURNS <type>
  <optional local declarations>
  <body>;

Parameters in PSM

- Unlike the usual name-type pairs in standard languages, 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.

Example: Stored Procedure

- Write a procedure that takes three arguments *i*, *n,* and *a*, and adds the following tuple to Supplier (procedure used to add new Supplier more easily)
  - Supplier_ID = *i*, Suplier_Name = *n*, and Address = *a*.

CREATE PROCEDURE <name> (<parameter list>)
    <optional local declarations>
    <body>;

The Procedure

```
CREATE PROCEDURE newSup (
    IN  i     VARCHAR(20),
    IN  n     VARCHAR(50),
    IN  a     VARCHAR(100)
)
INSERT INTO Supplier
VALUES(i, n, a);
```

Parameters are all read-only, not changed

The body - a single insertion

Invoking Procedures/Functions

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

  CALL newSup('s5', 'Bob','1 Main St.');
- Functions used in SQL expressions, where a value of their return type is appropriate.

Types of PSM statements -- 1

- RETURN <expression> sets the return value of a function.
  - Unlike C, etc., RETURN *does not* terminate function execution.
- DECLARE <name> <type> used to declare local variables.
- BEGIN . . . END for groups of statements.
  - Separate by semicolons.

Types of PSM Statements -- 2

- Assignment statements:

  SET <variable> = <expression>;

  - Example: SET b = 'Bud';
- Statement labels: give a statement a label by prefixing it with a name and a colon.

  set1:SET b = 'Bud';

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 suppliers by how many parts they have
  - <=0 parts: 'bad'.
  - < 2 parts: 'average'.
  - >=2 parts: 'good'.
- Function Rate rates supplier's (name).

```
Supplier(Supplier_ID, Supplier_Name, Address)
Catalog(Supplier_ID,Part_ID)
```

Example: IF (continued)

```
CREATE FUNCTION Rate (IN b VARCHAR(50) )
        RETURNS CHAR(10)
        DECLARE pCnt INTEGER;
    BEGIN
        SET pCnt = (SELECT COUNT(*) FROM Supplier,
                    Catalog WHERE Supplier.Supplier_ID =
                    Catalog.Supplier_ID and
                     Supplier_Name= b);
        IF pCnt < 0 THEN RETURN 'bad'
        ELSEIF pCnt < 2 THEN RETURN 'average'
        ELSE RETURN 'good'
        END IF;
    END;
```

Number of the parts of the supplier b

Nested IF statement

Return occurs here, not at one of the RETURN statements

Other Loop Forms

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

Example: Exiting a Loop

```
loop1: LOOP
        . . .
        LEAVE loop1;
        . . .
END LOOP;
```

← If this statement is executed . . .

← Control winds up here

Queries

- General SELECT-FROM-WHERE queries are *not* permitted in PSM.
- There are three ways to get the effect of a query:
  - Queries producing one value can be the expression in an assignment.
  - Single-row SELECT . . . INTO
  - Cursors

Example: Assignment/Query

- If *a* is a local variable and Supplier(Supplier_ID, Supplier_Name, Address) a relation, we can get the address of 'John Smith' by:

```
SET a =(SELECT Address FROM Supplier
WHERE Supplier_Name = 'John Smith');
```

```
SET (a,b)=(SELECT Address,Phone FROM
Supplier WHERE Supplier_Name = 'John
Smith');
```

SELECT . . . INTO

- An equivalent way to get the value of a query that is guaranteed to return a single tuple is by placing INTO <variable> after the SELECT clause.
- Example:

```
SELECT Address INTO a FROM  Supplier
WHERE Supplier_Name = 'John Smith';

SELECT Address, Phone INTO a,b
FROM Supplier WHERE Supplier_Name =
'John Smith';
```

Cursors

- A *cursor* is essentially a tuple-variable that ranges over all tuples in the result of some query.
- Declare a cursor *c* by:

```
DECLARE c CURSOR FOR <query>;
```

Opening and Closing Cursors

- To use cursor *c*, we must issue the command:

  OPEN c;

  - The query of *c* is evaluated, and *c* is set to point to the first tuple of the result.
- When finished with *c*, issue command:

  CLOSE c;

Fetching Tuples From a Cursor

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

  FETCH FROM c INTO x1, x2,…,x*n* ;
- The *x* '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.

Breaking Cursor Loops -- 1

- 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.

Breaking Cursor Loops -- 2

- Each SQL operation returns a *status*, which is a 5-digit number.
  - 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 -- 3

- 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';
```

Breaking Cursor Loops -- 4

- The structure of a cursor loop is thus:

```
cursorLoop: LOOP
   …
   FETCH c INTO … ;
   IF NotFound THEN LEAVE cursorLoop;
   END IF;
   …
END LOOP;
```

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.

Further Readings

- http://dev.mysql.com/doc/refman/5.0/en/stored-routines.html
- http://www.oracle.com/technology/tech/pl_sql/index.html