Persistent Stored Modules (Stored Procedures) : PSM

Murali Mani

Stored Procedures

- What is stored procedure?
  - SQL allows you to define procedures and functions and store them in the database server
  - Executed by the database server

- Advantages
  - Complex application logic executed while “close” to the data: usually implies efficiency
  - Contrast with tuple-at-a time processing by JDBC etc through “cursors”
  - Reuse the application logic

Murali Mani
Stored Procedures in Oracle

- Oracle supports a slightly different version of PSM called PL/SQL
- mySQL support is only in later versions

Defining a stored procedure

CREATE PROCEDURE <procedureName> [(<paramList>)]
  <localDeclarations>
  <procedureBody>;

A parameter in the paramList is specified as:
<name> <mode> <type>
<mode> is one of {IN, OUT, INOUT}
eg: val1 IN int

You can drop procedure by
DROP PROCEDURE <procedureName>

In PL/SQL, you can replace procedure by
CREATE OR REPLACE PROCEDURE <procedureName> …
**Example: Procedure in PSM**

```sql
CREATE PROCEDURE testProcedure
BEGIN
    INSERT INTO Student VALUES (5, 'Joe');
END;
```

Oracle PL/SQL:

```sql
CREATE PROCEDURE testProcedure IS
    BEGIN
        INSERT INTO Student VALUES (5, 'Joe');
    END;
run;
```
More about Procedures

- If there is an error in your procedure, Oracle will give you a warning. Use command `SHOW ERRORS` to show the errors in your procedure.
- Calling Procedures
  call <procedureName> [(<paramList>)];

Example

CREATE PROCEDURE testProcedure (num IN int, name IN varchar) IS
  BEGIN
    /* Insert values */
    INSERT INTO Student VALUES (num, name);
  END;
run;
Local Declarations

Example:
CREATE PROCEDURE testProcedure (num IN int, name
IN varchar) IS
    num1 int;  -- local variable
BEGIN
    num1 := 10;
    INSERT INTO Student VALUES (num1, name);
END;
.
run;

Other PSM features

Assignment statements: PL/SQL
<varName> := <expression>
**Control Structures: IF THEN ELSE**

IF <condition> THEN  
<statementList>  
ELSIF <condition> THEN  
<statementList>  
ELSIF  
...  
ELSE <statementList>  
END IF;

---

**Loops**

LOOP  
<statementList>  
END LOOP;

To exit from a loop use  
EXIT;
Loops: Example
CREATE PROCEDURE testProcedure (num IN int, name IN varchar) IS
  num1 int;
BEGIN
  num1 := 10;
  LOOP
    INSERT INTO Student VALUES (num1, name);
    num1 := num1 + 1;
    IF (num1 > 15) THEN EXIT; END IF;
  END LOOP;
END;
run;

FOR Loops
FOR i in [REVERSE] <lowerBound> .. <upperBound> LOOP
  <statementList>
END LOOP

Example:
FOR i in 1 .. 5 LOOP
  INSERT INTO Student (sNumber) values (10 + i);
END LOOP;
WHILE LOOPS

WHILE <condition> LOOP
  <statementList>
END LOOP;

Functions

CREATE FUNCTION <functionName> [(<paramList>)] RETURNS type IS
  <localDeclarations>
  BEGIN <functionBody>; END;

You can call a function as part of a sql expression

Drop a function:
  drop function <functionName>
Functions: Example

CREATE FUNCTION testFunction RETURN int IS
  num1 int;
BEGIN
  SELECT MAX (sNumber) INTO num1 FROM Student;
  RETURN num1;
END;

SELECT * from Student where sNumber = testFunction();

Other useful tips

- Oracle stores procedures and functions in catalog as relational tables.
  - Check user_procedures
  - Check user_functions
  - You may run queries etc against them such as
    - describe user_procedures;
    - select object_name from user_procedures;
Cursors

When we execute a statement, a relation is returned. It is stored in private work area for the statement. Cursor is a pointer to this area.

To create a cursor

```sql
CURSOR c_customers is
    SELECT * from CUSTOMERS;
```

We can open the cursor.

```sql
OPEN c_customers;
```

We can select data from the cursor.

```sql
FETCH c_customers into customers_rec;
```

And we can close the cursor.

```sql
CLOSE c_customers;
```
Implicit & Explicit Cursors

Every SQL data manipulation statements including queries that return only one row is an implicit cursor. An explicit cursor is what we create. For queries that return more than one row, you must declare an explicit cursor.

```
CREATE OR REPLACE PROCEDURE copyProcedure IS
    stID INT; name VARCHAR (10);
    CURSOR myCursor IS SELECT * FROM STUDENT;
    BEGIN
        OPEN myCursor;
        LOOP
            FETCH myCursor INTO stID, name;
            EXIT WHEN myCURSOR%NOTFOUND;
            INSERT INTO newStudent VALUES (stID, name);
        END LOOP;
        CLOSE myCursor;
    END;
```

Murali Mani
Cursor Attributes

The SQL cursor attributes are:-

- **%ROWCOUNT**: The number of rows processed by a SQL statement.
- **%FOUND**: TRUE if at least one row was processed.
- **%NOTFOUND**: TRUE if no rows were processed.
- **%ISOPEN**: TRUE if cursor is open or FALSE if cursor has not been opened or has been closed. Only used with explicit cursors.

Advanced Explicit Cursor

- Concepts
Cursor that uses parameters

CURSOR c_students
  (p_Department
classes.department%TYPE
  p_Course classes.department%TYPE)
  IS
SELECT * FROM classes
  WHERE department =
    p_Department
  AND course = p_Course;

To call the cursor
OPEN c_students('CS',101);

Murali Mani

---

Cursors for update

The syntax for this parameter in the SELECT
statement is:

SELECT ... FROM ... FOR UPDATE [OF
column_reference] [NOWAIT]

where column_reference is a column in
the table against which the query
is performed. A list of columns
can also be used.

Murali Mani
Example…for update

DECLARE
CURSOR c_AllStudents IS
SELECT *
  FROM students
  FOR UPDATE OF first_name, last_name;

Or the cursor can select every column by not specifying a range

DECLARE
CURSOR c_AllStudents IS
SELECT *
  FROM students
  FOR UPDATE;

NOWAIT

If another session already has locks on the rows in the active set, then the SELECT FOR UPDATE will hang until the other session releases the lock.

To handle this situation the parameter NOWAIT is available, which in case the rows are locked, OPEN will return the error ORA-54 resource busy and acquire with NOWAIT specified.