COMP 3311 Database Management Systems Spring 2015

Lab 5. PL/SQL, cursors, and triggers

Objectives of the Lab

- After this lab you should be able to
  - Know basic PL/SQL programming knowledge,
  - Know applying PL/SQL to build cursors,
  - Know applying PL/SQL to build triggers.

Getting the lab SQL script file

- Download the lab5.sql file as follows
  - 1. login to an arbitrary machine csl2wkxx.cse.ust.hk where xx=01-40
  - 2. at the command prompt type

    csl2wk01:lamngok:220> cd ~

    csl2wk01:lamngok:221> wget \

    ?http://course.cse.ust.hk/comp3311/labs/lab5.sql

- Log into Oracle database server using SQL*Plus with your password.

Running the lab SQL script

- Execute the script lab5.sql at the prompt

  SQL> @lab5.sql

- The tables created last time were dropped.
- Some new tables are created.

PL/SQL 1

- PL/SQL stands for Procedural Language/SQL.
- Basic unit in PL/SQL is called a block.
- PL/SQL extends the capabilities of SQL by adding to it the functionalities that are supported by procedural languages.

PL/SQL 2

- Basic structure of PL/SQL

```
DECLARE
 /* Declarative section: variables, types, and local subprograms. */

BEGIN
 /* Executable section: procedural and SQL statements go here. */
 /* This is the only section of the block that is required. */

EXCEPTION
/* Exception handling section: error handling statements go here. */

END;
```

PL/SQL 3

- PL/SQL is case *insensitive*
- C style comments /*…*/
- The SQL statements allowed in a PL/SQL program are SELECT, INSERT, UPDATE and DELETE.
- Data definition language like CREATE, DROP, ALTER are not allowed.
- In PL/SQL we used the ":=" operator to assign values to a variable.
- The "=" operator is for comparison.

PL/SQL 4

- Data type supported in PL/SQL
  - One of the types supported by SQL for defining the columns (i.e. NUMBER, INTEGER, CHAR, VARCHAR2, DATE, TIMESTAMP, etc).
  - Types declared to be of the same types as some database columns.
  - Some generic types.
  - For details of data type supported, you may refer to:

    http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/03_types.htm

PL/SQL 5

- Declaring variables
  - Declares a variable of the type number

    ```
    DECLARE
    count NUMBER;
    ```
  - Declares a variable with the same type as the no_of_projectors column in the facility table.

    ```
    DECLARE
    projectors facility.no_of_projectors%TYPE;
    ```
  - Declares a variable which is the same type as a row (record).

    ```
    DECLARE
    facility_record facility%ROWTYPE;
    ```

PL/SQL 6

- A simple PL/SQL that extract information from the departments table to a table called math_dept (lab5_plsql1.sql):

```
DECLARE
dept_name departments.name%TYPE;
dept_room departments.room_number%TYPE;
BEGIN
SELECT name, room_number INTO
dept_name,dept_room FROM departments
WHERE department_id='MATH';
INSERT INTO math_dept VALUES (dept_name,dept_room);
END;
.
RUN;
```

- The dot "." indicates the end of the PL/SQL code.
- The statement "run;" tells the database engine to execute the PL/SQL codes defined, we can use "/" to replace "." +"run;"

PL/SQL 7

- Flow Control: IF-THEN-ELSE-END IF (lab5_plsql2.sql)

```
DECLARE
    room departments.room_number%TYPE;
  BEGIN
    SELECT room_number INTO room FROM departments
    WHERE department_id='COMP';
    IF (room>3000 and room<4000) THEN
          UPDATE departments SET room_number=room+2000
          WHERE department_id='COMP';
    ELSE
          UPDATE departments SET room_number=5528
          WHERE department_id='COMP';
    END IF;
  END;
   /
```

(The above procedure adds 2000 to the room number of COMP dept if the room starts with 3xxx. The room number will be 5528 if the room number does not start with 3xxx.)

PL/SQL 8

- Flow control: LOOP (lab5_plsql3.sql)

```
DECLARE
  i testloop.i%TYPE :=1;
BEGIN
  LOOP
    INSERT INTO testloop VALUES (i);
    i:=i+1;
    EXIT WHEN i>10;
  END LOOP;
END;
  /
```

- A LOOP can be terminated by the EXIT WHEN keyword

PL/SQL 9

- Flow control: FOR LOOP (lab5_plsql4.sql)

```
DECLARE
   i NUMBER(2):=1;
 BEGIN
   FOR VAR IN (
          SELECT * FROM facility order by department_id
    )LOOP
          UPDATE facility SET
          no_of_computers = no_of_computers+i
          WHERE department_id=VAR.department_id;
           i:=i+1;
  END LOOP;
 END;
  /
```

- VAR is a variable local to the for-loop and need not be declared.

Cursor 1

- The SELECT statement in PL/SQL can only fetch a single record.
- If the query returns more than one records, a cursor is needed.
- A cursor is like a pointer that points to a single record each time.
- Using the cursor, the records can be fetched in a one-by-one manner.

Cursor 2

- A cursor should be defined at the DECLARE section of the PL/SQL codes.
- It needs to be activated by the OPEN command.
- Then we can FETCH the records in a one-by-one manner.
- When all the records are fetched, "%NOTFOUND" will return a true (will see its details 2 slides later) .
- We need to CLOSE the cursor after using it, so as to free up the resources.

Cursor 3

- Syntax of a cursor

```
DECLARE
        CURSOR cursor_name
        IS select_statement;
```

- An example:

```
DECLARE
  CURSOR facility_cursor
  IS SELECT department_id, name,
                  no_of_projectors, no_of_computers
  FROM facility;
```

- The above cursor fetches all the records from the facility table.

Cursor 4

- Getting the status of a cursor:
  - cursor_name%NOTFOUND
    Whether the previous fetch has failed.
  - cursor_name%FOUND
    Whether the previous fetch is successful.
  - cursor_name%ROWCOUNT
    Number of records fetched so far.
  - cursor_name%ISOPEN
    Is the cursor still open.

Cursor 5

- Incorporating the Cursor to PL/SQL codes (lab5_cursor1.sql) :

```
DECLARE
        var_deptid facility.department_id%TYPE;
        var_name  facility.name%TYPE;
        CURSOR facility_cursor
        IS SELECT department_id, name FROM facility;
BEGIN
      OPEN facility_cursor;
      LOOP
                FETCH facility_cursor INTO var_deptid,var_name;
                EXIT WHEN facility_cursor%NOTFOUND;
                INSERT INTO test VALUES (var_deptid,var_name);
      END LOOP;
      CLOSE facility_cursor;
END;
/
```

- The above cursor fetches records from the facility table, and insert the values one by one into another table called test.

Cursor 6

- Cursor loop (lab5_cursor2.sql) :

```
DECLARE
      var_deptid facility.department_id%TYPE;
      var_name  facility.name%TYPE;
      CURSOR facility_cursor
      IS SELECT department_id, name FROM facility;
BEGIN
     FOR rec in facility_cursor
           LOOP
                var_deptid:=rec.department_id;
                  var_name:=rec.name;
              INSERT INTO test VALUES (var_deptid,var_name);
           END LOOP;
END;
/
```

Cursor 7

- The facility_cursor on the previous slide is automatically opened by the FOR LOOP.
- The rec is a rowtype data, but there is no need for us to declare it.
- Codes inside the LOOP are execute once for each row of the cursor, and each time the two columns department_id, and name are copied into rec.
- We can access the data in rec directly (as shown in the codes).
- LOOP terminates automatically once all the records in the cursor are fetched.
- The cursor is then closed automatically.

Trigger 1

- Triggers are procedures that are stored in the database and are implicitly run, or **fired**, when something happens.
- Traditionally, triggers supported the execution of a PL/SQL block when an INSERT, UPDATE, or DELETE occurred on a table or view. Triggers support system and other data events on DATABASE and SCHEMA. Oracle Database also supports the execution of PL/SQL or Java procedures.

Trigger 2

- The syntax for creating a trigger in Oracle is listed below:

```
CREATE [OR REPLACE] TRIGGER trigger_name
[BEFORE | AFTER | INSTEAD OF] database_event
[REFERENCING [OLD AS old_name] [NEW AS
new_name]]
FOR EACH ROW
[WHEN criteria]
BEGIN
trigger body [PL/SQL blocks]
END;
```

Trigger 3

- The semantics
  - BEFORE: if this keyword presents, the trigger will be started before each of the affected rows has been changed.
  - AFTER: if this keyword presents, the trigger will be started after each of the affected rows has been changed.
  - INSTEAD OF: if this keyword presents, the trigger will be started instead of performing the DML on the view
  - Database_event:indicates the specific database events that will start the trigger
  - FOR EACH ROW: the trigger will be started once for each row (record).
  - WHEN: specifies the trigger condition.
  - NEW: this keyword refers to a new record retrieved
  - OLD: this keyword refers to an exisiting record.

Trigger 4

- The following trigger adds a prefix to the email address for the CS students when a new CS student record is being inserted. (lab5_trigger1.sql)

```
CREATE OR REPLACE TRIGGER chk_email
BEFORE INSERT ON students
FOR EACH ROW
WHEN (NEW.department_id = 'COMP')
DECLARE
      prefix CHAR(3) := 'cs_';
BEGIN
      :NEW.email := prefix || :NEW.email;
END;
/
```

- Note that the red part of the codes is a PL/SQL block.

Trigger 5

- The following example backs up the record for the facility table in the old_facility table, if the record is to be removed from the facility table. (lab5_trigger2.sql)

```
CREATE OR REPLACE TRIGGER backup_facility
BEFORE DELETE ON facility
FOR EACH ROW
DECLARE
  id_null EXCEPTION;
BEGIN
   INSERT INTO old_facility
   VALUES (:old.department_id, :old.name, :old.no_of_projectors,
                :old.no_of_computers);
   IF (:old.department_id IS NULL) THEN
      RAISE id_null;
  END IF;
  EXCEPTION
     WHEN id_null THEN
     DBMS_OUTPUT.PUT_LINE('Department ID missing');
 END;
/
```

- Note that the red part is also a PL/SQL block.

Conclusion

- We covered the following topics in this lab:
  - Simple PL/SQL syntax.
  - Building Cursors with PL/SQL.
  - Building triggers with PL/SQL.