COMP 3311
Database Management Systems

Lab 5

PL/SQL and Cursors

Lab Objectives

- After this lab you should be able to
  - write simple programs in PL/SQL (section 5.1.3).
  - use cursors in PL/SQL programs (section 5.1.3).
- General information about cursors can be found in the textbook sections indicated above.
- Specific information about Oracle PL/SQL, cursors can be found by following the links given at the top of some of the slides.

SQL and Programming Languages

- There are two approaches for accessing SQL from a programming language.
  - **Embedded SQL** – A preprocessor identifies SQL statements at compile time, submits them to a database server for pre-compilation and optimization and then replaces the SQL statements with appropriate code and function calls before invoking the programming-language compiler.
  - **Dynamic SQL** – A query is constructed as a character string at runtime, submitted to a database server and the result retrieved into program variables one record at a time.
    - Examples: JDBC, ODBC.

PL/SQL (1)

http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/02_funds.htm

- PL/SQL stands for Procedural Language/SQL.
- PL/SQL allows SQL statements to be embedded into a procedural programming language.
- The 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 programming languages.

PL/SQL (2)

- PL/SQL
  - is *case insensitive*.
  - uses C style comments /*...*/.
  - uses := operator to assign values to a variable.
  - uses = operator for comparison.
- ALLOWED SQL statements: select, insert, update and delete.
- NOT ALLOWED SQL statements: create, drop, alter, rename (i.e., DDL statements).

PL/SQL: Basic Structure & Data Types

- Basic structure

declare	**Declarative section:** declaration of variables, types, and local subprograms go here.
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;	

- The data type of a variable can be either
  - a data type used to define the attributes of a table (i.e., number, integer, char, varchar2, date, timestamp, etc.).
  - the same data type as a database attribute.

PL/SQL: Declaring Variables (1)

http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/02_funds.htm#27307

- Declare a variable of type number.

```
declare
    count number;
```

- Declare a variable with the same type as the numberProjectors attribute in the Facility table.

```
declare
    projectors Facility.numberProjectors%type;
```

- Declare a variable that is the same type as a row (tuple).

```
declare
    facilityRecord Facility%rowtype;
```

PL/SQL: Declaring Variables (2)

- Extract data from the Department table into a table called MathDept.

```
declare
    -- deptName is the same type as departmentName in the Department table
    deptName department.departmentName%type;
    -- deptRoom is the same type as roomNo in the Department table
    deptRoom Department.roomNo%type;
begin
    select departmentName, roomNo into deptName, deptRoom
    from department
    where departmentId='MATH';
    insert into MathDept values (deptName, deptRoom);
end;
/
```

The “/” indicates the end of the PL/SQL code and tells the database engine to execute the PL/SQL code.

PL/SQL: Flow of Control Statements (1)

http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/04_struc.htm

- Sequential control
  - goto – branch to a label unconditionally
  - null – pass control to the next statement
- Conditional control
  - if-then, if-then-else, if-then-elseif
  - case - selects one sequence of statements to execute

PL/SQL: Flow of Control Statements (2)

http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/04_struc.htm

- Iterative control
  - loop *statements* end loop;
  - while *condition* loop *statements* end loop;
    - Executes the loop while *condition* is true.
  - for *index* in [reverse] *lower_bound..upper_bound* loop *statements* end loop;
    - Iterates over a range of integers starting either from *lower_bound* to *upper_bound* or in *reverse* order.
  - exit / exit when *condition*
    - Exits *current loop* completely either unconditionally or when *condition* is true.
  - continue / continue when *condition*
    - Exits *current loop iteration* and continues with the next iteration either unconditionally or when *condition* is true.

PL/SQL: If-Then-Else Example (1)

```
declare
    room Department.roomNo%type; -- room is of type roomNo
begin
    select roomNo into room from Department where departmentId='COMP';
    if (room>3000 and room<4000) then
        update Department set roomNo=room+1000 where departmentId='COMP';
    else
        update Department set roomNo=5528 where departmentId='COMP';
    end if;
end;
/
```

This procedure adds 1000 to the room number of the COMP department if the room starts with 3xxx. The room number is set to 5528 if the room number does not start with 3xxx.

PL/SQL: If-Then-Else Example (2)

```
declare
   room Department.roomNo%type; -- room is of type roomNo
begin
⇒  room := select roomNo from Department where departmentId='COMP';
   if (room>3000 and room<4000) then
      update Department set roomNo=room+1000 where departmentId='COMP';
   else
      update Department set roomNo=5528 where departmentId='COMP';
   end if;
end;
/
```

Cannot assign the result of a Select statement to a variable as in line 4 above since the result of a select statement is always a table even if the table contains only one value.

PL/SQL: Loop Example

- Insert values 1 to 10 into table Testloop.

```
declare
    i Testloop.testValue%type := 1; -- i is of type testValue and is initialized to 1
begin
    loop
        insert into Testloop values (i);
        i := i + 1;
        exit when i>10;
    end loop;
end;
/
```

Note: A loop can be terminated by the exit or exit when keywords.

PL/SQL: Loop Label

- A loop label appears at the beginning of a loop statement enclosed by double angle brackets.

```
declare
    i Testloop.testValue%type := 1; -- i is of type testValue and is initialized to 1
begin
    <<myLoop>>
    loop
        insert into Testloop values (i);
        i := i + 1;
        exit myLoop when i>10;
    end loop;
end;
/
```

Note: Several levels of nested loops can be terminated using a loop label.

PL/SQL: For-Loop Example

- Increase the number of projectors in a department based on a loop counter i.

```
declare
    i number(2) := 1;
begin
    for var in (select * from Facility order by departmentId) loop
        update Facility set numberComputers = numberComputers + i
            where departmentId=var.departmentId;
        i := i + 1;
    end loop;
end;
/
```

Note: In this example the for loop acts like a for each loop where var is assigned the next record in the result of the select statement in each iteration of the for loop.

Cursors

http://www.techonthenet.com/oracle/cursors/

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

Cursors: How to Use

- A cursor is defined in the declare section of a PL/SQL program.
- It needs to be activated by the open command.
- The fetch command gets the records one-at-a-time.
- When all the records are fetched, %notfound will return true.
- It is necessary to close the cursor after using it so as to free up resources.

Cursors: Declaration

- A cursor is declared using the syntax:

```
declare
    cursor cursor_name is select_statement;
```

- Example: Fetch all the records from the Facility table.

```
declare
    cursor facilityCursor is select departmentId, name, numberProjectors,
        numberComputers from Facility;
```

Cursors: Status

- The possible values of a cursor status are:
  - Determine whether the previous fetch failed.

    cursor_name%notfound
  - Determine whether the previous fetch succeeded.

    cursor_name%found
  - Determine the number of records fetched so far.

    cursor_name%rowcount
  - Determine whether the cursor is still open.

    cursor_name%isopen

Cursors: Example With PL/SQL (1)

```
declare
    varDeptId Facility.departmentId%type;
    varName Facility.departmentName%type;
    cursor facilityCursor is select departmentId, departmentName from Facility;
begin
    open facilityCursor;
    loop
        fetch facilityCursor into varDeptId, varName;
        exit when facilityCursor%notfound;
        insert into ResultTable values (varDeptId, varName);
    end loop;
    close facilityCursor;
end;
/
```

The facilityCursor retrieves records from the Facility table and inserts the values one-by-one into another table called ResultTable.

Cursors: Example With PL/SQL (2)

```
declare
    varDeptId Facility.departmentId%type;
    varName Facility.departmentName%type;
    cursor facilityCursor is select departmentId, departmentName from Facility;
begin
    for record in facilityCursor loop
        varDeptId := record.departmentId;
        varName := record.departmentName;
        insert into ResultTable values (varDeptId,varName);
    end loop;
end;
/
```

Same example as on previous slide, but using a for loop automatically opens the cursor, exits the loop when there are no more records in the cursor and closes the cursor.

Cursors: Example With PL/SQL (3)

- The FacilityCursor on the previous slide is automatically opened by the for loop.
- The variable record is of data type rowtype, but there is no need to declare it.
- Code inside the loop is executed once for each row of the cursor, and each time the two attributes departmentId, and departmentName are copied into record.
- The data in record can be accessed directly (as shown in the code).
- The loop terminates automatically once all the records in the cursor are fetched.
- The cursor is then closed automatically.

Summary

- This lab covered the following topics:
  - Simple PL/SQL syntax and usage.
  - Using cursors with PL/SQL.

Lab Exercise

- You must complete the lab exercise and upload the result to Canvas by **11:59 p.m. this Friday**.

Ask for help if you need it!

IMPORTANT NOTE

If you want to save your modified SQL script files, copy them to the M drive or to a USB drive as any personal files on the lab computers will be automatically deleted periodically.