Introduction to PL/SQL Programming
Course Description

Overview
This class will develop database programming skills using PL/SQL. Students will learn PL/SQL programming skills, proper software engineering techniques for database programming, PL/SQL problem solving and performance tuning skills. This class is intended for Oracle analysts, developers, designers, administrators, and managers new to Oracle database programming. Each student will leave with a template toolkit for building PL/SQL applications.

Prerequisites
Students should have a basic understanding of SQL, six months experience working with an Oracle database, and at least six months 3GL programming experience.

Audience: Application developers and database administrators

Objectives
Upon completion of this course, students will understand:

- PL/SQL language basics using variables, datatypes, control and iteration constructs
- PL/SQL datatypes using basic datatypes, records, V2.3 tables, %TYPE, and %ROWTYPE
- How to use SQL in PL/SQL including transaction management
- Implicit and Explicit cursor processing using cursor variables and cursor fetching methods
- Error handling and debugging techniques using exceptions
- Development for interactive PL/SQL programs using SQL*Plus
- Introduction to stored procedures, functions and packages
- Package development techniques using private and public scoped attributes and routines
- Overloading techniques
- Oracle supplied packages
- Dynamic SQL techniques using PL/SQL

Duration
Two days
Introduction to PL/SQL Programming

Chapter Overview................................................................................................................ 1-1
What is PL/SQL? ................................................................................................................... 1-3
Features of PL/SQL ............................................................................................................. 1-4
The Block Structure ............................................................................................................ 1-5
Language Rules ................................................................................................................... 1-6
Using PL/SQL ....................................................................................................................... 1-7
Executing PL/SQL from SQL*Plus ................................................................................... 1-8
Variables and Constants ..................................................................................................... 1-9
Identifiers ........................................................................................................................... 1-10
Customizing SQL*Plus for PL/SQL Scripts ....................................................................... 1-11
SELECT INTO Statement .................................................................................................... 1-12
Using Bind Variables ......................................................................................................... 1-13
Interactive PL/SQL Programs ........................................................................................... 1-14
Oracle Datatypes are supported in PL/SQL .................................................................... 1-15
Numeric Datatypes ............................................................................................................. 1-16
Datatype Conversion ......................................................................................................... 1-17
Implicit and Explicit Conversion ....................................................................................... 1-18
%TYPE and %ROWTYPE ................................................................................................... 1-19
Example: Displaying Dept Values ..................................................................................... 1-20
Subtypes ............................................................................................................................ 1-21
Subtype Declarations ........................................................................................................ 1-22
Labels .................................................................................................................................. 1-23
Variable Scope and Visibility Rules ................................................................................... 1-24
SQL and PL/SQL Reserved Words .................................................................................... 1-25
PL/SQL Reserved Words ................................................................................................... 1-26
Direction of PL/SQL ............................................................................................................ 1-27
External Routines and Java Stored Procedures ................................................................. 1-28
Review Questions ............................................................................................................... 1-29
Summary ............................................................................................................................ 1-30
Exercises ............................................................................................................................. 1-31

Control Flow

Chapter Overview................................................................................................................ 2-1
Operators ............................................................................................................................. 2-3
Built-In Functions .............................................................................................................. 2-4
Handling Exceptions .......................................................................................................... 2-5
Working with Exceptions ................................................................................................... 2-6
SQLCODE and SQLERRM Functions ............................................................................... 2-7
Predefined PL/SQL Exceptions ......................................................................................... 2-8
User-Defined Exceptions .................................................................................................. 2-9
Working with Cursors

Chapter Overview ......................................................... 3-1
Working with Cursors ..................................................... 3-3
Implicit Cursors and Attributes ........................................ 3-4
Explicit Cursors ............................................................ 3-5
Explicit Cursor Attributes ................................................ 3-6
More examples using Cursor Attributes ............................ 3-7
Using Subqueries with Cursors .......................................... 3-8
Using Cursor FOR LOOPS ................................................ 3-9
Example: Cursor FOR LOOP .............................................. 3-10
Using %ROWTYPE with Cursors ....................................... 3-11
Using Aliases and Subqueries ........................................... 3-12
Using Aliases and Subqueries (FOR LOOP) ......................... 3-13
Example: Using Bind Variables ......................................... 3-14
Using Parameters with Explicit Cursors ............................ 3-15
Example: Using Cursor Parameters ................................. 3-16
Modifying Data using Explicit Cursors .............................. 3-17
Example: WHERE CURRENT OF ....................................... 3-18
Managing Transactions in PL/SQL Programs ....................... 3-19
Review Questions .......................................................... 3-20
Summary ........................................................................... 3-21
Exercises .......................................................................... 3-22

Procedures, Functions, and Packages

Chapter Overview .......................................................... 4-1
Working with Subprograms ............................................. 4-3
Named Subprograms ...................................................... 4-4
### Collections

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>5-1</td>
</tr>
<tr>
<td>Records</td>
<td>5-3</td>
</tr>
<tr>
<td>Creating a PL/SQL Record</td>
<td>5-4</td>
</tr>
<tr>
<td>PL/SQL2 Tables</td>
<td>5-5</td>
</tr>
<tr>
<td>Defining PL/SQL2 Tables</td>
<td>5-6</td>
</tr>
<tr>
<td>Table Attributes</td>
<td>5-7</td>
</tr>
<tr>
<td>PL/SQL8 Tables</td>
<td>5-8</td>
</tr>
<tr>
<td>Instantiating PL/SQL Tables</td>
<td>5-9</td>
</tr>
<tr>
<td>Coding Examples</td>
<td>5-10</td>
</tr>
<tr>
<td>Create a Type Specification</td>
<td>5-11</td>
</tr>
<tr>
<td>CAST Operator</td>
<td>5-12</td>
</tr>
<tr>
<td>Operator “THE”</td>
<td>5-13</td>
</tr>
<tr>
<td>Queries on Nested Tables</td>
<td>5-14</td>
</tr>
<tr>
<td>Review Questions</td>
<td>5-15</td>
</tr>
<tr>
<td>Summary</td>
<td>5-16</td>
</tr>
<tr>
<td>Exercises</td>
<td>5-17</td>
</tr>
</tbody>
</table>
Working with Cursors

Chapter Overview

PL/SQL programs often access a large number of records. Cursors are used when a SQL command may manipulate a number of records. This chapter will show how to manipulate data using cursors.

At the end of this chapter, students will understand:

♦ Implicit cursors
♦ Explicit cursors
♦ DECLARE, OPEN, FETCH, and CLOSE cursor statements
♦ Using cursors in FOR loops
♦ Cursor variables
Working with Cursors

Cursor attributes contain information about the processing of SELECT, INSERT, UPDATE, and DELETE statements.

Cursors are used to access data. A cursor is an area of memory containing SQL statements and information for processing the statements.

- There are implicit and explicit cursors.

Implicit cursors are created automatically for SQL statements executed in PL/SQL.

Explicit cursors can be defined to control how data is processed by SQL commands.
Implicit Cursors and Attributes

Implicit cursors are created in PL/SQL by any DML statements that do not have an explicit cursor defined with the statement. Implicit cursors are created for the following statements:

- SELECT ... INTO
- INSERT
- UPDATE
- DELETE

Implicit Cursor Attributes return information on INSERT, UPDATE, and DELETE statements.

- SQL%FOUND returns TRUE if a row is modified, FALSE if no records are affected.
- SQL%NOTFOUND returns TRUE if no records are affected, FALSE if any rows are modified.
- SQL%ROWCOUNT returns how many records were affected by the SQL command.

Example:

```sql
UPDATE employee
  SET salary = salary * 1.2
WHERE id = vempid;
IF SQL%FOUND THEN
  COMMIT;
END IF;
...
```
Explicit Cursors

Explicit cursors are used when a query or data modification may access multiple records. Cursors are processed in four steps.

Declaring the cursor associates the cursor name with the query:

- The DECLARE cursor statement is used to define the cursor using a SQL statement.
- Cursor variable scopes are similar to the scopes of other PL/SQL variables.
- The SQL command in a cursor statement cannot contain a SELECT INTO statement.

Opening the cursor executes the cursor:

- The OPEN cursor statement processes the SQL statement, allocates needed resources, and puts a pointer at the first record in the result set.

Fetching records from the cursor returns one record from the result set:

- The FETCH cursor command will read one record at a time and move the pointer to the next record.

Closing the cursor releases resources held by the cursor:

- The CLOSE cursor command will release all the resources allocated for processing the cursor. This means, the cursor must be opened again before records can be fetched again.
Explicit Cursor Attributes

Explicit Cursor Attributes return information on cursor processing.

- `cursorname%ISOPEN` returns TRUE if the cursor is opened, FALSE if the cursor is closed.
- `cursorname%FOUND` returns TRUE if a row is returned, FALSE if no records were returned.
- `cursorname%NOTFOUND` returns TRUE if no records are returned, FALSE if a row is returned from a fetch.
- `cursorname%ROWCOUNT` gets incremented by one each time a row is returned from a FETCH statement.

```sql
-- empcur.sql
DECLARE
    CURSOR emp_cur IS SELECT last_name, salary,
        NVL(commission_pct, 0) FROM employee
        WHERE start_date < '01-JAN-99';
    vlast_name employee.last_name%TYPE;
    vsalary employee.salary%TYPE;
    vcomm    employee.commission_pct%TYPE;
BEGIN
    OPEN emp_cur;
    LOOP
        FETCH emp_cur INTO vlast_name, vsalary, vcomm;
        EXIT WHEN emp_cur%NOTFOUND;
        dbms_output.put_line(vlast_name || ' ' || vsalary || ' ' || vcomm);
    END LOOP;
    CLOSE emp_cur;
END;
```

Notes

The cursor attribute using %FOUND or %NOTFOUND returns FALSE before the first FETCH statement is executed. %ROWCOUNT is set to zero before the FETCH statement is executed.
More examples using Cursor Attributes

```sql
-- c_increase.sql
DECLARE
    c_increase NUMBER(7,2) := 1.2;
    CURSOR emp_cur IS SELECT last_name, salary * c_increase, title
        FROM employee
        WHERE start_date < '01-JAN-99';
    vlast_name employee.last_name%TYPE;
    vsalary  employee.salary%TYPE;
    vtitle     employee.title%TYPE;
BEGIN
    OPEN emp_cur;
    LOOP
        FETCH emp_cur INTO vlast_name, vsalary, vtitle;
        IF emp_cur%FOUND THEN
            dbms_output.put_line(vlast_name || ' ' || vsalary || ' ' || vtitle);
        ELSE
            EXIT;
        END IF;
        c_increase := c_increase + .05; -- does not affect fetch
    END LOOP;
    CLOSE emp_cur;
END;
```

```sql
LOOP
...
    IF emp_cur%ISOPEN THEN -- cursor is open
        FETCH emp_cur INTO vlast_name, vsalary, vtitle;
        EXIT WHEN emp_cur%NOTFOUND;
        dbms_output.put_line(vlast_name || ' ' || vsalary || ' ' || vtitle);
    ...
    ELSE -- cursor is closed
        OPEN emp_cur;
    END IF;
    ...
END LOOP;
```

Note

A cursor must be closed before it can be reopened.
Using Subqueries with Cursors

-- subquery.sql
DECLARE
    CURSOR emp_cur IS SELECT last_name, salary, title FROM employee
        WHERE start_date < '01-JAN-99'
            AND dept_id = (SELECT dept_id FROM employee
                                WHERE last_name = 'WONDER')
        ORDER BY salary DESC;
    vlast_name    employee.last_name%TYPE;
    vsalary    employee.salary%TYPE;
    vtitle       employee.title%TYPE;
    c_maxemps CONSTANT NUMBER(3) := 10;
    MAX_EMPS_EXCEEDED  exception;
BEGIN
    OPEN emp_cur;
    LOOP
        FETCH emp_cur INTO vlast_name, vsalary, vtitle;
        EXIT WHEN emp_cur%NOTFOUND;
        IF emp_cur%ROWCOUNT > c_maxemps THEN
            RAISE max_emps_exceeded;
        END IF;
        dbms_output.put_line(vlast_name || ' ' || vsalary || ' ' || vtitle);
    END LOOP;
    CLOSE emp_cur;
EXCEPTION
    WHEN max_emps_exceeded THEN
        dbms_output.put_line('exceeded max employees');
    WHEN OTHERS THEN
        dbms_output.put_line('abnormal termination: ');
        dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;

Notes

The SQL commands used to define a cursor can be complex queries containing joins, subqueries, functions, expressions, GROUP BY statements, and ORDER BY statements.
Using Cursor FOR LOOPS

The FOR LOOP syntax can be used with cursors. Using FOR LOOPS with cursors offers a number of advantages:

- Eliminates extra code used to perform cursor operations
- Easier to maintain than writing individual statements to manage cursors

A cursor FOR LOOP:

- Defines the record variable with attributes for each column returned in the CURSOR SELECT statement.
- Opens the cursor the first time through the loop.
- Fetches the data into the record variable each loop iteration.
- Checks for the NOTFOUND condition after each fetch.
- Closes the cursor after the last iteration.

Syntax:

```
FOR record_variable IN cursor_name LOOP
  statements
END LOOP;
```
Example: Cursor FOR LOOP

```
--cursorloop.sql
DECLARE
  CURSOR dept_cur IS SELECT id, name, region_id FROM dept
  ORDER BY name;
BEGIN
  FOR vdept_rec IN  dept_cur LOOP
    dbms_output.put_line('departments: ' || vdept_rec.id || ' ' || vdept_rec.name || ' ' || vdept_rec.region_id);
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    dbms_output.put_line('abnormal termination:');
    dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;
/
```

Notes

This example eliminates 5 unnecessary lines from the example without the FOR LOOP. Possibly, the best feature is the implicit creation of the record variable in the FOR LOOP. This correctly defines each attribute with the appropriate datatype.

Using FOR LOOPS with cursors eliminates all the redundant steps of OPEN, FETCH, EXIT WHEN cursor%NOTFOUND, and CLOSE when working with cursors.
Using %ROWTYPE with Cursors

It is recommended to use %ROWTYPE when a cursor will be working with all columns from a table.

```
-- rowtype.sql
DECLARE
  CURSOR dept_cur IS SELECT id, name, region_id FROM dept
  ORDER BY name;
  vdept_rec   dept%ROWTYPE;
BEGIN
  OPEN dept_cur;
  LOOP
    FETCH dept_cur INTO vdept_rec;
    EXIT WHEN dept_cur%NOTFOUND;
    dbms_output.put_line('departments: ' || vdept_rec.id || '
      ' || vdept_rec.name || ' ' || vdept_rec.loc);
  END LOOP;
  CLOSE dept_cur;
EXCEPTION
  WHEN OTHERS THEN
    dbms_output.put_line('abnormal termination:');
    dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;
/
```
Using Aliases and Subqueries

```sql
-- aliasex.sql

DECLARE
    CURSOR dept_emp_cur IS
        SELECT d.id, d.name, "NO_EMPS", "SUM_SAL"
        FROM dept d,
            (SELECT dept_id, count(*) "NO_EMPS",
                sum(salary) "SUM_SAL"
            FROM employee GROUP BY dept_id) dcnt
        WHERE d.id = dcnt.dept_id AND "SUM_SAL" >= 8000
        ORDER BY "SUM_SAL";
    vdeptid      dept.id%TYPE;
    vname        dept.name%TYPE;
    vno_emps     NUMBER(4);
    vsum_sal     NUMBER(7,2);
BEGIN
    OPEN dept_emp_cur;
    LOOP
        FETCH dept_emp_cur INTO vdeptid, vname, vno_emps, vsum_sal;
        EXIT WHEN dept_emp_cur%NOTFOUND;
        dbms_output.put_line('department: ' || vdeptid || ' ' ||
            rpad(vname,12,' ') || ' ' || vno_emps || ' ' ||
            vsum_sal);
    END LOOP;
    CLOSE dept_emp_cur;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('abnormal termination:');
        dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;
/
```

Notes

Notice a variable must be defined for each column returned from the query defined in the cursor definition.
Using Aliases and Subqueries (FOR LOOP)

```sql
-- aliasforloop.sql
DECLARE
    CURSOR dept_emp_cur IS
        SELECT d.id, d.name, "NO_EMPS", "SUM_SAL"
        FROM dept d, (SELECT dept_id, count(*) "NO_EMPS",
                        sum(salary) "SUM_SAL"
                        FROM employee GROUP BY dept_id) dcnt
        WHERE d.id = dcnt.dept_id AND "SUM_SAL" >= 8000
        ORDER BY "SUM_SAL";
BEGIN
    FOR drec  IN dept_emp_cur LOOP
        dbms_output.put_line('department: '||drec.id ||'
                        '||rpad(drec.name,12,' ') ||' '|| drec.no_emps ||'
                        '|| drec.sum_sal );
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('abnormal termination:');
        dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;
/
```

Notes

This code eliminates a number of lines from the previous example. The record variable in the loop properly creates all the necessary record attributes from the query defined with the cursor.
Example: Using Bind Variables

```sql
--bindvar.sql
DECLARE
  vmax_sal   NUMBER(8,2);
  vdeptid    employee.dept_id%TYPE;
  max_deptid employee.dept_id%TYPE;
  max_sumsal NUMBER(8,2) := 0;
  CURSOR  dept_cur IS select dept_id, sum(salary) "SUM_SAL"
               FROM employee GROUP BY dept_id;
  CURSOR emp_cur IS SELECT dept_id, last_name, salary, title
               FROM employee
               WHERE dept_id = vdeptid;
BEGIN
  -- find the dept_id with the largest payroll
  FOR dept_rec IN dept_cur LOOP
    IF dept_rec.sum_sal > max_sumsal THEN
      max_deptid := dept_rec.dept_id;
      max_sumsal := dept_rec.sum_sal;
    END IF;
  END LOOP;

  -- get employees determined by max_deptid
  vdeptid := max_deptid;
  max_sumsal := 0;
  FOR e_rec IN emp_cur LOOP
    dbms_output.put_line(rpad(e_rec.last_name, 10, ' ') ||
                         e_rec.dept_id || e_rec.salary || e_rec.title);
    max_sumsal := e_rec.salary + max_sumsal;
  END LOOP;
  dbms_output.put_line('The total payroll is: ' || max_sumsal);
EXCEPTION
  WHEN OTHERS THEN
    dbms_output.put_line('abnormal termination:');
    dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;
```
Using Parameters with Explicit Cursors

Cursor parameters pass values to cursors when they are opened. The values of the parameters determine the result set returned when the cursor is opened.

- The scope of the parameter is the cursor definition. This makes sure the parameter value can only be set when the cursor is opened.
- The cursor can return a different result set each time it is opened by giving the parameter a different value.

Syntax:

```sql
DECLARE
    CURSOR cursor_name (parameter1, parameter2, ...) IS
    SELECT ...

Example:

CURSOR emp_cur (p_deptid employee.dept_id%TYPE) IS
    SELECT dept_id, last_name, salary, title FROM employee
    WHERE dept_id = p_deptid;
BEGIN

Opening a cursor:

OPEN emp_cur(vdeptid);

Opening a cursor using a FOR LOOP:

FOR e_rec IN emp_cur(vdeptid) LOOP
    ...
END LOOP;
```
Example: Using Cursor Parameters

```
--curparam.sql
DECLARE
    vmax_sal  NUMBER(8,2);
    max_deptid employee.dept_id%TYPE;
    max_sumsal NUMBER(8,2) := 0;
    CURSOR  dept_cur IS select dept_id, sum(salary) "SUM_SAL"
               FROM  employee GROUP BY dept_id;
    CURSOR emp_cur (p_deptid employee.dept_id%TYPE) IS
      SELECT dept_id, last_name,salary, title FROM employee
      WHERE dept_id = p_deptid;
BEGIN
    -- find the dept_id with the largest payroll
    FOR dept_rec IN dept_cur LOOP
      IF dept_rec.sum_sal > max_sumsal THEN
        max_deptid := dept_rec.dept_id;
        max_sumsal := dept_rec.sum_sal;
      END IF;
    END LOOP;

    -- get employees determined by max_deptid
    max_sumsal := 0;
    FOR e_rec IN emp_cur(max_deptid) LOOP
        dbms_output.put_line(rpad(e_rec.last_name, 10, ' ')
                           || e_rec.dept_id || e_rec.salary
                           || e_rec.title);
        max_sumsal := e_rec.salary + max_sumsal;
    END LOOP;

    dbms_output.put_line('The total payroll is: '||max_sumsal);
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('abnormal termination:');
        dbms_output.put_line(SUBSTR(SQLERRM,1,100));
END;
```

Notes

Compare this code to the example using bind variables.
Modifying Data using Explicit Cursors

Explicit cursors can be used to modify data. Defining explicit cursors to modify data is done in two steps:

- The FOR UPDATE clause is used with the cursor definition.
- The WHERE CURRENT OF clause is used with the UPDATE or DELETE statement.

The FOR UPDATE clause will lock records returned by the cursor.

Syntax:

```
DECLARE
    ...
    CURSOR cursor_name IS
        SELECT ...
        FOR UPDATE;
BEGIN
    ...
    FOR var_rec IS cursor_name LOOP
        UPDATE ...
        WHERE CURRENT OF cursor_name;

    END LOOP;
    ...
END;
```

Notes

Do not execute the COMMIT command between fetches when using the WHERE CURRENT OF clause.
Example: WHERE CURRENT OF

```sql
--where_current.sql
DECLARE
    min_comm employee.commission_pct%TYPE := 10;
    CURSOR emp_cur IS
        SELECT id, last_name, salary, commission_pct, title FROM employee
        FOR UPDATE;
BEGIN
    FOR emp_rec IN emp_cur LOOP
        IF emp_rec.title = 'Sales Representative' AND emp_rec.commission_pct < min_comm THEN
            DELETE FROM employee
            WHERE CURRENT OF emp_cur;
            dbms_output.put_line('Deleted employee: ' || emp_rec.id || ' ' || emp_rec.last_name);
        END IF;

        IF emp_rec.title = 'Warehouse Manager' THEN
            UPDATE employee
            SET salary = salary * 1.15
            WHERE CURRENT OF emp_cur;
            dbms_output.put_line('15% raise given to: ' || emp_rec.id || ' ' || emp_rec.last_name);
        END IF;
    END LOOP;
END;
```
Managing Transactions in PL/SQL Programs

PL/SQL programs can execute the following DML commands that impact transaction processing.

- COMMIT
- ROLLBACK
- SAVEPOINT
- ROLLBACK TO SAVEPOINT

```sql
DECLARE
BEGIN
...
  INSERT INTO DEPT VALUES(50, 'WEB SERVICES', 1);
  SAVEPOINT A;
...
  INSERT INTO DEPT VALUES(60, 'NETWORK', 3);
  SAVEPOINT B;
...
  INSERT INTO DEPT VALUES(70, 'TELECOM', 2);
  SAVEPOINT C;
...
  ROLLBACK TO SAVEPOINT B;
  COMMIT;
END;
/```
Review Questions

1. What is the difference between an implicit cursor and an explicit cursor?

2. When should one use an explicit cursor?

3. Which of the following are implicit cursor attributes?
   a. SQL%NOTFOUND
   b. EMP_CUR%FOUND
   c. EMP_CUR%ROWCOUNT
   d. SQL%ROWCOUNT

4. What are four steps used to process a cursor?

5. What are the advantages of using a cursor FOR LOOP over an ordinary cursor without a cursor FOR LOOP?

6. When do you use FOR UPDATE and WHERE CURRENT OF statements?
Summary

Cursors are used to access data. A cursor is an area of memory containing SQL statements and information for processing the statements.

- There are implicit and explicit cursors.

```
-- empcur.sql
DECLARE
    CURSOR emp_cur IS SELECT last_name, salary,
                 NVL(commission_pct,0) FROM employee
                 WHERE start_date < '01-JAN-99';
    vlast_name    employee.last_name%TYPE;
    vsalary  employee.salary%TYPE;
    vcomm     employee.commission_pct%TYPE;
BEGIN
    OPEN emp_cur;
    LOOP
        FETCH emp_cur INTO vlast_name, vsalary, vcomm;
        EXIT WHEN emp_cur%NOTFOUND;
        dbms_output.put_line(vlast_name || ' ' || vsalary || ' ' || vcomm);
    END LOOP;
    CLOSE emp_cur;
END;
```

```
FOR record_variable IN cursor_name LOOP
    statements
END LOOP;
```
Exercises

1. Write a program to determine the seniority of employees with the same title. Use an explicit cursor with open, fetch, and close statements to display the employee with the most seniority for each title.

2. Modify the program to use a cursor FOR LOOP.