Oracle Academy
Introduction to Database Programming with PL/SQL
Instructor Resource Guide

INSTRUCTOR NOTES FOR SLIDES

SECTION 5 LESSON 1 - Introduction to Explicit Cursors

Slide 1: Introduction to Explicit Cursors
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
As most programs will need to handle more than one from a table, we need another way of handling data in PL/SQL programs. This way is explicit cursors.

Slide 4: Tell Me / Show Me – Context Areas and Cursors
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Implicit and Explict Cursors
A SQL cursor is a private Oracle SQL working area. The implicit cursor is used by the server to test and parse the SQL statements. Using these implicit cursor, we can test the outcome of SQL statements in PL/SQL.

An implicit cursor is always called "SQL", and we can access its attributes: SQL%FOUND, SQL%NOTFOUND and SQL%ROWCOUNT.

You can use explicit cursors to name a context area and access its stored data.

Slide 6: Tell Me / Show Me – Limitations of Implicit Cursors
The key point is not whether there **is** more than one row, but that there **can be** more than one row. Programmers must think about the data that is possible as well as the data that actually exists now.

The only SELECT statement that can never return more than one row has WHERE primary_key_or_unique_column_name = value. For example:

```
SELECT …
FROM employees
WHERE employee_id = 101;
```

Group functions without a GROUP BY clause always return exactly one row. For example:

```
SELECT COUNT(*) FROM employees;
```

In all other situations we should declare and use an explicit cursor.

Slide 7: Tell Me / Show Me – Explicit Cursors
The point is not whether there is more than one row, but that there can be more than one row. Programmers must think about the data that is possible as well as the data that actually exists now…so that programs do not have to be updated later.

Explicit cursors are great tools to make programming in PL/SQL easier and faster. For queries that return more than one row, you explicitly declare a cursor to process the rows individually.

It is possible in PL/SQL to write a SELECT statement which returns more than one row, without using explicit cursors. This technique is called **bulk binding**, but it is beyond the scope of this course. Also, bulk binding is not **always** possible, so we still need to understand and use explicit cursors.

Slide 8: Tell Me / Show Me – Example of an Explicit Cursor
Do not explain the detailed code at this stage. Just point out the key words: DECLARE, OPEN, FETCH and CLOSE.

Slide 9: Tell Me / Show Me – Explicit Cursor Operations
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Explicit Cursor Operations (continued)
No instructor notes for this slide.

Slide 11: Tell Me / Show Me – Controlling Explicit Cursors
The FETCH statement retrieves the current row and advances the cursor to the next row either until there are no more rows or until a specified condition is met.

Slide 12: Tell Me / Show Me – Declaring and Controlling Explicit Cursors
No instructor notes for this slide.

Slide 13: Steps for Using Explicit Cursors
Now that you have a conceptual understanding of cursors, review the steps to use them:

1. DECLARE the cursor in the declarative section of a PL/SQL block by naming it and defining the SQL SELECT statement to be associated with it.
2. OPEN the cursor. The OPEN statement executes the SELECT statement and populates the active set with the results of the SELECT.
3. FETCH each row from the active set. After each fetch, you test the cursor to check if the last fetch was successful. If there are no more rows to process, then you must close the cursor.

4. CLOSE the cursor. The CLOSE statement releases the active set of rows. It is now possible to reopen the cursor to establish a fresh active set.

<u>Slide 14: Tell Me / Show Me – Declaring the Cursor</u>
In the executable section, a single-row SELECT statement must have an INTO clause. However, a SELECT statement in a cursor declaration cannot have an INTO clause. This is because you are not retrieving the data into variables yet (we retrieve the data later using a FETCH statement).

<u>Slide 15: Tell Me / Show Me – Declaring the Cursor: Example 1</u>
Note that even if there is currently only one employee in department_id 30, we should still declare a cursor, because in future there may be more than one employee. We want our program to stand the test of time !

<u>Slide 16: Tell Me / Show Me – Declaring the Cursor: Example 2</u>
No instructor notes for this slide

<u>Slide 17: Tell Me / Show Me – Declaring the Cursor: Example 3</u>
No instructor notes for this slide

<u>Slide 18: Tell Me / Show Me – Guidelines for Declaring the Cursor</u>
Remember, the ORDER BY clause is last. SQL SELECT statements can be very in depth…especially with joins and subqueries combined. The code needs to be written were it is readable and understandable. Along with these guidelines, remember to indent, capitalize keywords, and space for readability.

<u>Slide 19: Tell Me / Show Me – Opening the Cursor</u>
No instructor notes for this slide

<u>Slide 20: Tell Me / Show Me – Opening the Cursor (continued)</u>
No instructor notes for this slide

<u>Slide 21: Tell Me / Show Me – Fetching Data from the Cursor</u>
This example fetches only the first row.

<u>Slide 22: Tell Me / Show Me – Fetching Data from the Cursor (continued)</u>
Variables are declared to hold the fetched values from the cursor. The fetch statement only retrieves one row. You have successfully fetched the values from the cursor into the variables. However, there are five employees in department 50. Only one row has been fetched. To fetch all the rows, you have to make use of loops.

As you can see on the slide, we can use explicit cursor attributes to test for the outcome of the cursor, just like we did with implicit cursors. On the slide we are using emp_cursor%NOTFOUND to exit the LOOP.

Students will learn more about explicit cursor attributes such as %NOTFOUND in the next lesson. For now, the key point is that each execution of a FETCH statement returns only one row (the next row from the active set).

Slide 23: Tell Me / Show Me – Guidelines for Fetching Data from the Cursor
Data type mismatching is the most common mistake that programmers make when writing their code. Data types must be compatible between the table and the variables. The variables in the FETCH statement must line up with the columns in the table, so makes sure to have the same number of variables in your INTO statement as you have in your SELECT list. Loops are used all the time to fetch data…using the %NOTFOUND cursor attribute to stop the loop.

Slide 24: Tell Me / Show Me – Fetching Data from the Cursor
Answer: The SELECT statement in the cursor declaration returns three columns, but the FETCH statement references only two variables.

Slide 25: Tell Me / Show Me – Fetching Data from the Cursor (continued)
Answer: The OPEN statement populates the active set with exactly one row. The first time round the loop, this row will be fetched and displayed. The second (and third, and fourth ….) times round the loop, no error is recorded. Therefore the first row will be re-displayed over and over again.

Slide 26: Tell Me / Show Me – Closing the Cursor
Once a CURSOR is CLOSEd, the memory is released. But, the data is still in memory and will be overwritten when a new OPEN statement is executed. So, if the CURSOR is no longer needed, the memory can be released.
A cursor can be reopened only if it is closed.

Slide 27: Tell Me / Show Me – Guidelines for Closing the Cursor
Because a context area is a private memory area, any changes to database data (DML statements) executed by other users will not be visible to our cursor until we close and reopen the cursor.

Slide 28: Tell Me / Show Me –Putting It All Together
Students saw this example briefly earlier in the lesson.

The example declares and processes a cursor to obtain the country name and national holiday for countries in Asia. We have a CURSOR statement being DECLAREd along with other variables to be used. Once the CURSOR is OPENed, a LOOP is started to FETCH the data. When all the data has been FETCHed, then the CURSOR is CLOSEd and memory is released.

Slide 29: Tell Me / Show Me – Terminology
Context Area – An allocated memory area used to store the data processed by a SQL statement.
Cursor – A label for a context area or a pointer to the context area.
Implicit Cursor – Defined automatically by Oracle for all SQL DML statements, and for

SELECT statements that return only one row.
Explicit Cursor – Declared by the programmer for queries that return more than one row.
Active set – The set of rows returned by a multiple row query in an explicit cursor operation.
FETCH – Statement that retrieves the current row and advances the cursor to the next row either until there are no more rows or until a specified condition is met.
OPEN – Statement that executes the query associated with the cursor, identifies the active set, and positions the cursor pointer to the first row.
CLOSE – Disables a cursor, releases the context area, and undefines the active set.

Slide 30: Summary
No instructor notes for this slide

Slide 31: Try It / Solve It
No instructor notes for this slide

SECTION 5 LESSON 2 - Using Explicit Cursor Attributes

<u>Slide 1: Using Explicit Cursor Attributes</u>
No instructor notes for this slide

<u>Slide 2: What Will I Learn?</u>
No instructor notes for this slide

<u>Slide 3: Why Learn It?</u>
No instructor notes for this slide

<u>Slide 4: Tell Me / Show Me – Cursors and Records</u>
No instructor notes for this slide

<u>Slide 5: Tell Me / Show Me – Cursors and Records (continued)</u>
There are 11 columns in EMPLOYEES, so 11 variables must be declared and populated by the INTO clause in the FETCH statement. And what if a twelfth column was added to the table later?

<u>Slide 6: Tell Me / Show Me – Cursors and Records (continued)</u>
No instructor notes for this slide

<u>Slide 7: Tell Me / Show Me – Cursors and Records (continued)</u>
No instructor notes for this slide

<u>Slide 8: Tell Me / Show Me – Structure of a PL/SQL Record</u>
PL/SQL variables and cursors have *attributes*, which are properties that let you reference the datatype and structure of an item without repeating its definition. A percent sign (%) serves as the attribute indicator.

PL/SQL Record Structure
The whole record is accessed with the name of the record. To reference an individual field, use the dot notation as shown below:

```
record_name.field_name
```

For example, you reference the job_id field in the v_emp_record record as follows:

```
v_emp_record.job_id
```

<u>Slide 9: Tell Me / Show Me – Structure of cursor_name%ROWTYPE</u>
The %ROWTYPE attribute provides a record type that represents a row in a table or a cursor. The record can store an entire row of data selected from the table or fetched from a cursor. %ROWTYPE is used to declare variables with a composite type that matches the type of an existing database cursor or table.

Slide 10: Tell Me / Show Me – Cursors and %ROWTYPE
Point out that:

- the cursor must be declared before the record which references it
- we reference the individual fields in the record as record_name.field_name (look at the DBMS_OUTPUT call).

Ask students: what if in future a new column is added to EMPLOYEES? We would NOT need to change this PL/SQL block at all.

Slide 11: Tell Me / Show Me – Cursors and %ROWTYPE: Another Example
Ask students: how many fields does v_emp_dept_record contain, and what are they?

Answer: three fields

```
v_emp_dept_record.first_name
v_emp_dept_record.last_name
v_emp_dept_record.department_name
```

Slide 12: Tell Me / Show Me – Explicit Cursor Attributes
In order to be able to control the loop we need information about the cursor. The same four built in cursor attributes as we looked at in an earlier lesson for implicit cursors can also be used when we are working with explicit cursors.

The %ISOPEN attribute is never true for implicit cursors, as Oracle is doing all the work for us, it automatically OPENs, FETCHEs and CLOSEs the cursor. %ISOPEN is useful in blocks where we may OPEN and CLOSE the cursor conditionally. If you try to OPEN an OPEN cursor, you will get an error, so we may need to test to see if the cursor is OPEN before we try to OPEN it.

The other three are very useful attributes that help to identify where the pointer is in relation to the data in the CURSOR.

Slide 13: Tell Me / Show Me – %ISOPEN Attribute
%ISOPEN is useful in blocks where we may OPEN and CLOSE the cursor conditionally. For example:

```
…
IF some_condition THEN OPEN cursor_name;
END IF;
-- Maybe the cursor is open now, maybe not
…
-- Now we must have an open cursor whatever, so:
IF NOT cursor_name%ISOPEN THEN
  OPEN cursor_name;
```

Slide 14: Tell Me / Show Me – %ROWCOUNT and %NOTFOUND Attribute
No instructor notes for this slide

Slide 15: Tell Me / Show Me – Example of %ROWCOUNT and %NOTFOUND
In this example, we exit from the loop and close the cursor when one of two conditions is true:

1. The active set contains 11 rows or less and we have fetched all of them
2. The active set contains at least 12 rows but we want to fetch only the first 11.

Slide 16: Tell Me / Show Me – Explicit Cursor Attributes in SQL Statements
Instead, we would copy the cursor attribute value to a separate variable, then use this variable in the SQL statement:

```
DECLARE
  CURSOR emp_cursor IS ...;
  v_emp_record  emp_cursor%ROWTYPE;
  v_count     NUMBER;
  v_rowcount  NUMBER;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO v_emp_record;
    EXIT WHEN emp_cursor%NOTFOUND;
   v_rowcount := emp_cursor%ROWCOUNT;
   INSERT INTO top_paid_emps
      (employee_id, rank, salary)
   VALUES
      (v_emp_record.employee_id, v_rowcount, v_emp_record.salary);
```

Slide 17: Tell Me / Show Me – Terminology
Record – A composite data type in PL/SQL, consisting of a number of fields each with their own name and data type.
%ROWTYPE – Declares a record with the same fields as the cursor on which it is based.
%ISOPEN – Returns the status of the cursor.
%ROWCOUNT – An attribute that processes an exact number of rows or counts the number of rows fetched in a loop.
%NOTFOUND – An attribute used to determine whether a query found any rows matching the query criteria.

Slide 18: Summary
No instructor notes for this slide

Slide 19: Try It / Solve It
No instructor notes for this slide

SECTION 5 LESSON 3 - Cursor FOR Loops

Slide 1: Cursor FOR Loops
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Cursor FOR Loops
Cursors and Loops were made for one another. The majority of your PL/SQL programs will involve working on many rows from the same table, so Oracle made that part easy for you by combining the declaration of a PL/SQL RECORD and the use of a CURSOR in a FOR LOOP. So when using you are using an explicit cursor, simplify your coding by using a cursor FOR loop instead of the OPEN, FETCH, and CLOSE statements. A cursor FOR LOOP implicitly declares its loop counter as a record that represents a row FETCHed from the database. Next, it OPENs a cursor, repeatedly FETCHes rows of values from the result set into fields in the record, then CLOSEs the cursor when all rows have been processed.

Once again the structure of the statement is shown. Remember the indentation for readability, variable naming rules, and spacing. Record_name and cursor_name are identifier names that you make up.

Slide 5: Tell Me / Show Me – Cursor FOR Loops (continued)
No instructor notes for this slide

Slide 6: Tell Me / Show Me – Cursor FOR Loops (continued)
No instructor notes for this slide

Slide 7: Tell Me / Show Me – Cursor FOR Loops (continued)
Using a cursor FOR loop will NOT improve performance. Although we do not code them explicitly, the OPEN, FETCH, EXIT WHEN …. %NOTFOUND and CLOSE are still happening, so the code will not execute faster.

Do not declare the record that controls the loop because it is declared implicitly. The scope of the implicit record is restricted to the loop, so you cannot reference the record outside the loop. You can access fetched data by r*ecord_name.column_name*.

Slide 8: Tell Me / Show Me – Cursor FOR Loops: A Second Example
Answer: two fields, because the cursor whose %ROWTYPE it is based on SELECTs two table columns.

Slide 9: Tell Me / Show Me – Guidelines for Cursor FOR Loops
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Testing Cursor Attributes
No instructor notes for this slide

Slide 11: Tell Me / Show Me – Cursor FOR Loops Using Subqueries
Ask students to imagine a much bigger program with several hundred lines of PL/SQL code, maybe including many cursor declarations.

Slide 12: Tell Me / Show Me – Cursor FOR Loops Using Subqueries: Example
Note: You cannot reference explicit cursor attributes such as %ROWCOUNT and %NOTFOUND if you use a subquery in a cursor FOR loop because you cannot give the cursor an explicit name

Slide 13: Tell Me / Show Me – Cursor FOR Loops Using Subqueries (continued)
No instructor notes for this slide

Slide 14: Terminology
Cursor FOR loop – Automates standard cursor-handling operations such as OPEN, FETCH, %NOTFOUND and CLOSE, so that they do not need to be coded explicitly

Slide 15: Summary
No instructor notes for this slide

Slide 16: Try It / Solve It
No instructor notes for this slide

SECTION 5 LESSON 4 - Cursors with Parameters

Slide 1: Cursors with Parameters
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Cursors with Parameters
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Cursors with Parameters: Example
In a real program, the region_id would not be hard-coded in the OPEN statement, but would be entered by the user.

Slide 6: Tell Me / Show Me – Defining Cursors with Parameters
We specify the datatype of the cursor parameter, but not the size. For example: VARCHAR2, not VARCHAR2(20). Students will see this again later in the course, when we pass parameters to and from stored subprograms.

When the cursor is opened, the parameter value is passed to the Oracle server, which uses it to decide which rows to retrieve into the active set of the cursor. This means that you can open and close an explicit cursor several times in a block, or in different executions of the same block, returning a different active set on each occasion.

Slide 7: Tell Me / Show Me – Defining Cursors with Parameters (continued)
No instructor notes for this slide

Slide 8: Tell Me / Show Me – Opening Cursors with Parameters
No instructor notes for this slide

Slide 9: Tell Me / Show Me – Cursors with Parameters
No instructor notes for this slide

Slide 10: Tell Me / Show Me – Another Example of a Cursor with a Parameter
The purpose of this code is to fetch and display all employees in the department with the highest department_id. Note that the maximum department_id is stored in the v_deptid variable and is then passed into the empcur cursor. Note also that the SELECT MAX(department_id) …. will always return exactly one row, so an explicit cursor is not needed for this.

Slide 11: Tell Me / Show Me – Cursor FOR Loops with a Parameter
Parameters are placed inside parentheses following the CURSOR. FOR…END LOOP statements let you execute a sequence of statements multiple times. The CURSOR will repeatedly use new value(s) that are passed into the parameter.

Slide 12: Tell Me / Show Me – Cursors with Multiple Parameters
No instructor notes for this slide

Slide 13: Tell Me / Show Me – Cursors with Multiple Parameters: Another Example
No instructor notes for this slide

Slide 14: Summary
No instructor notes for this slide

Slide 15: Try It / Solve It
No instructor notes for this slide

SECTION 5 LESSON 5 - Using Cursors for Update

Slide 1: Using Cursors for Update
While teaching this lesson you may wish to demonstrate locking and lock waits. Because Application Express automatically COMMITs all DML on statement completion (thereby releasing all locks held) you will need to disable the Autocommit feature by unchecking the check box in the top left hand corner of the SQL Commands window.

You can then issue one or more DML statements which will not be committed or rolled back until you execute an explicit COMMIT; or ROLLBACK; SQL statement.

However, please ensure that you remain in the SQL Commands window for the whole transaction. If you leave SQL Commands and go to another Application Express page (for example Home, SQL Scripts or Object Browser) your transaction will automatically be rolled back.

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
An open cursor provides a read-consistent view of the data fetched by the cursor. This means that any updates made by other users since the cursor was opened will not be seen when we fetch the rows, even if the updates were committed. Our session would have to close and re-open the cursor in order to see the committed updates.

Slide 4: Tell Me / Show Me – Declaring a Cursor with the FOR UPDATE Clause
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Declaring a Cursor with the FOR UPDATE Clause (continued)
No instructor notes for this slide

Slide 6: Tell Me / Show Me – NOWAIT Keyword in the FOR UPDATE Clause
No instructor notes for this slide

Slide 7: Tell Me / Show Me – NOWAIT Keyword in the FOR UPDATE Clause (continued)
It may help students to think of a lock as being like a red traffic light. If another session has already locked the rows, the traffic light is red. Using the default, our session is an infinitely patient driver who will wait indefinitely until the traffic light turns green (i.e. the locks are released). If we use NOWAIT, we have no patience at all; we see the red light and immediately back off.

Slide 8: Tell Me / Show Me – FOR UPDATE OF column-name
If we don't specify a column-name, then rows of both tables are locked. This causes unnecessary extra locking when (in this example) we want to lock only the EMPLOYEES rows, not the DEPARTMENTS rows.
Strangely, it doesn't matter which column-name we use. The slide example would work the same if we coded

```
…. FOR UPDATE OF job_id;
or … FOR UPDATE OF last_name;
or … FOR UPDATE OF <any other column of EMPLOYEES>;
```

Individual columns are never locked, only whole rows.

Slide 9: Tell Me / Show Me – WHERE CURRENT OF Clause
No instructor notes for this slide

Slide 10: Tell Me / Show Me – WHERE CURRENT OF Clause (continued)
No instructor notes for this slide

Slide 11: Tell Me / Show Me – WHERE CURRENT OF Clause (continued)
No instructor notes for this slide

Slide 12: Tell Me / Show Me – NOWAIT, FOR UPDATE, and WHERE CURRENT OF Clauses
No instructor notes for this slide

Slide 13: Tell Me / Show Me – A Second Example:
No instructor notes for this slide

Slide 14: Tell Me / Show Me – Terminology
FOR UPDATE – Declares that each row is locked as it is being fetched so other users can not modify the rows while the cursor is open.
NOWAIT – A keyword used to tell the Oracle server not to wait if the requested rows have already been locked by another user.

Slide 15: Summary
No instructor notes for this slide

Slide 16: Try It / Solve It
No instructor notes for this slide

SECTION 5 LESSON 6 - Using Multiple Cursors

Slide 1: Using Cursors for Update
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – A sample Problem Statement
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Problem Solution: Step 1
Answer: Because we will OPEN C_EMP several times (once for each department) and it must fetch a different set of rows each time.

Slide 6: Tell Me / Show Me – Problem Solution: Step 2
No instructor notes for this slide

Slide 7: Tell Me / Show Me – Problem Solution: Step 3
No instructor notes for this slide

Slide 8: Tell Me / Show Me
Go through this carefully with the students. The inner C_EMP loop executes once for each row fetched by the C_DEPT cursor, and fetches a different subset of EMPLOYEES each time.

Slide 9: Tell Me / Show Me – A Second Example
Point out that in these multilevel reports, the tables will usually be related to each other by a foreign key.

Slide 10: Tell Me / Show Me
Go through this carefully with the students. The inner C_DEPT loop executes once for each row fetched by the C_LOC cursor, and fetches a different subset of DEPARTMENTS each time.

Slide 11: Tell Me / Show Me – Using FOR Loops with Multiple Cursors
Point out that this code is functionally identical to that in the previous slide, but is much more compact and easier to maintain.

Slide 12: Tell Me / Show Me – A Final Example
Ask students: which employees will receive a salary increase?
Answer: those whose department is in location_id 1700 and who earn less than 10000.

Slide 13: Summary
No instructor notes for this slide

Slide 14: Try It / Solve It
No instructor notes for this slide

PRACTICE SOLUTIONS

SECTION 5 LESSON 1 – Introduction to Explicit Cursors

Terminology

1. **___Explicit Cursor__________** Declared by the programmer for queries that return more than one row.
2. **___Cursor_________________** A label for a context area or a pointer to the context area.
3. **___CLOSE__________________** Disables a cursor, releases the context area, and undefines the active set.
4. **___Context Area____________** An allocated memory area used to store the data processed by a SQL statement.
5. **___Implicit Cursor___________** Defined automatically by Oracle for all SQL DML statements, and for SELECT statements that return only one row.
6. **___OPEN__________________** Statement that executes the query associated with the cursor, identifies the active set, and positions the cursor pointer to the first row.
7. **___FETCH_________________** Statement that retrieves the current row and advances the cursor to the next row either until there are no more rows or until a specified condition is met.
8. **___Active set_______________** The set of rows returned by a multiple row query in an explicit cursor operation.

Try It/Solve It

1. In your own words, explain the difference between implicit and explicit cursors.

An implicit cursor is declared and controlled automatically by the Oracle server, while an explicit cursor is declared and controlled by the PL/SQL programmer.

2. Which SQL statement can use either an explicit or an implicit cursor, as needed?

SELECT (DML statements always use implicit cursors)

3. List two circumstances in which you would use an explicit cursor.

You need to retrieve more than one row from a table.
You want more programmatic control.

4. Exercise using wf_currencies table:

   A. Write a PL/SQL block to declare a cursor called wf_currencies_cur. The cursor will be used to read and display all rows from the wf_currencies table. You will need to retrieve currency_code and currency_name, ordered by ascending currency_name.

```
DECLARE
  CURSOR wf_currencies_cur IS
    SELECT  currency_code, currency_name
      FROM  wf_currencies
      ORDER BY currency_name;
```

   B. Add a statement to open the wf_currencies_cur cursor.

```
...
BEGIN
  OPEN wf_currencies_cur;
END;
```

   C. Add variable declarations and an executable statement to read ONE row through the wf_currencies_cur cursor into local variables.

```
...
v_curr_code   wf_currencies.currency_code%TYPE;
v_curr_name   wf_currencies.currency_name%TYPE;

...
FETCH wf_currencies_cur
  INTO v_curr_code, v_curr_name;
```

   D. Add a statement to display the fetched row, and a statement to close the wf_currencies_cur cursor.

```
DBMS_OUTPUT.PUT_LINE(v_curr_code || ' ' || v_curr_name);
CLOSE wf_currencies_cur;
```

E. Run your block to check that it works. It should display: AFA Afghani.

By now the block should look like this:

```
DECLARE
  CURSOR wf_currencies_cur IS
    SELECT  currency_code, currency_name
      FROM  wf_currencies
      ORDER BY currency_name;
  v_curr_code   wf_currencies.currency_code%TYPE;
  v_curr_name   wf_currencies.currency_name%TYPE;
BEGIN
  OPEN wf_currencies_cur;
  FETCH wf_currencies_cur
    INTO v_curr_code, v_curr_name;
  DBMS_OUTPUT.PUT_LINE(v_curr_code || ‘ ‘ || v_curr_name);
  CLOSE wf_currencies_cur;
END;
```

F. Your cursor in question 4 fetched and displayed only one row. Modify the block so that it fetches and displays all the rows, using a LOOP and EXIT statement. Test your modified block. It should fetch and display 160 rows. If it displays more or less than 160 rows, check that your EXIT statement is in the correct place in the code.

```
DECLARE
  CURSOR wf_currencies_cur IS
    SELECT  currency_code, currency_name
      FROM  wf_currencies
      ORDER BY currency_name;
  v_curr_code   wf_currencies.currency_code%TYPE;
  v_curr_name   wf_currencies.currency_name%TYPE;
BEGIN
  OPEN wf_currencies_cur;
  LOOP
    FETCH wf_currencies_cur
      INTO v_curr_code, v_curr_name;
    EXIT WHEN wf_currencies_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_curr_code || ' ' || v_curr_name);
  END LOOP;
  CLOSE wf_currencies_cur;
END;
```

G. Write and test a PL/SQL block to read and display all the rows in the wf_countries table for all countries in region 5 (South America region). For each selected country, display the country_name, national_holiday_date, and national_holiday_name. Display only those countries having a national holiday date that is not null. Save your code (you will need it in the next practice).

```
DECLARE
v_country_name          wf_countries.country_name%TYPE;
v_national_holiday_name wf_countries.national_holiday_name%TYPE;
v_national_holiday_date wf_countries.national_holiday_date%TYPE;
CURSOR wf_countries_cur IS
  SELECT country_name, national_holiday_name,
         national_holiday_date
    FROM wf_countries
    WHERE region_id = 5 AND national_holiday_date IS NOT NULL;
BEGIN
  OPEN wf_countries_cur;
  LOOP
    FETCH wf_countries_cur INTO v_country_name,
          v_national_holiday_name,v_national_holiday_date;
    EXIT WHEN wf_countries_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE ('Country: ' || v_country_name
    || ' National holiday: '|| v_national_holiday_name
    || ', held on: '|| v_national_holiday_date );
  END LOOP;
  CLOSE wf_countries_cur;
END;
```

Extension Exercise

1. Write a PL/SQL block to read and display the names of world regions, with a count of the number of countries in each region. Include only those regions having at least 10 countries. Order your output by ascending region name.

```
DECLARE
v_region_name       wf_world_regions.region_name%TYPE;
v_no_of_countries    NUMBER(6);
CURSOR wf_regions_cur IS
  SELECT region_name, COUNT(*)
    FROM wf_world_regions wr, wf_countries c
    WHERE wr.region_id = c.region_id
    GROUP BY region_name
    HAVING COUNT(*) >= 10
    ORDER BY region_name;
BEGIN
  OPEN wf_regions_cur;
  LOOP
    FETCH wf_regions_cur INTO v_region_name, v_no_of_countries;
    EXIT WHEN wf_regions_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (v_region_name || ' contains '||
                       v_no_of_countries || ' countries.');
  END LOOP;
  CLOSE wf_regions_cur;
END;
```

SECTION 5 LESSON 2 – Using Explicit Cursor Attributes

Terminology

1. ___**%ROWTYPE**___________ Declares a record with the same fields as the cursor on which it is based.

2. ____**Record**________________ A composite data type in PL/SQL, consisting of a number of fields each with their own name and data type.

3. ____**%ISOPEN**_____________ Returns the status of the cursor.

4. ____**%ROWCOUNT**_________ An attribute that processes an exact number of rows or counts the number of rows fetched in a loop.

5. ____**%NOTFOUND**__________ An attribute used to determine whether the most recent FETCH statement successfully returned a row.

Try It/Solve It

1. In your own words, explain the advantage of using %ROWTYPE to declare a record structure based on a cursor declaration.

Using %ROWTYPE we need only one declaration for the whole cursor, not one for each column fetched by the cursor. This is especially useful when the cursor fetches many columns.

2. Write a PL/SQL block to read through rows in the wf_countries table for all countries in region 5 (South America region). For each selected country, display the country_name, national_holiday_date, and national_holiday_name. Use a record structure to hold all the columns selected from the wf_countries table.

   Hint: This exercise is very similar to question 4G in the previous lesson. Use your solution as a starting point for this exercise.

```
DECLARE
  CURSOR wf_countries_cur IS
    SELECT country_name, national_holiday_name,
       national_holiday_date
      FROM wf_countries
      WHERE region_id = 5;
  wf_countries_rec  wf_countries_cur%ROWTYPE;
BEGIN
  OPEN wf_countries_cur;
  LOOP
    FETCH wf_countries_cur INTO wf_countries_rec;
    EXIT WHEN wf_countries_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE ('Country: ' ||
                    wf_countries_rec.country_name
                    || ' National holiday: '||
                    wf_countries_rec.national_holiday_name
                    || ', held on: '||
                    wf_countries_rec.national_holiday_date);
  END LOOP;
  CLOSE wf_countries_cur;
END;
```

3. For this exercise, you use the employees table. Create a PL/SQL block that fetches and displays the six employees with the highest salary. For each of these employees, display the first name, last name, job id and salary. Order your output so that the employee with the highest salary is displayed first. Use %ROWTYPE and the explicit cursor attribute %ROWCOUNT.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT first_name, last_name, job_id, salary
      FROM employees
      ORDER BY salary DESC;
  emp_record  emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
    DBMS_OUTPUT.PUT_LINE(emp_record.first_name ||' '||
                      emp_record.last_name  ||' '||
                      emp_record.job_id ||' '||
                      emp_record.salary);
    EXIT WHEN emp_cursor%ROWCOUNT >= 6;
  END LOOP;
  CLOSE emp_cursor;
END;
```

4. Look again at the block you created in question 3. What if you wanted to display 21 employees instead of 6? There are only 20 rows in the employees table. What do you think would happen?

The loop would try to FETCH a non-existent twenty-first row.

5. In real life we would not know how many rows the table contained. Modify your block from question 3 so that it will exit from the loop when either 21 rows have been fetched and displayed, or when there are no more rows to fetch. Test the block again.

```
 DECLARE
  CURSOR emp_cursor IS
    SELECT first_name, last_name, job_id, salary
      FROM employees
      ORDER BY salary DESC;
  emp_record  emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN emp_cursor%ROWCOUNT > 21
           OR emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_record.first_name ||' '||
                         emp_record.last_name  ||' '||
                         emp_record.job_id ||' '||
                         emp_record.salary);
    END LOOP;
  CLOSE emp_cursor;
END;
```

SECTION 5 LESSON 3 – Cursor FOR Loops

Terminology

1. ___**Cursor FOR loop**_______ Automates standard cursor-handling operations such as OPEN, FETCH, %NOTFOUND and CLOSE, so that they do not need to be coded explicitly

Try It/Solve It

1. Describe two benefits of using a cursor FOR loop

   - **The coding is much shorter and simpler. We don't need to declare a %ROWTYPE record or code OPEN, FETCH, EXIT WHEN and CLOSE statements. We may not even need to declare the cursor.**

   - **All the declaration and manipulation of the cursor is in one place in the code. This makes it much easier to understand what the cursor does, and to modify it later if needed.**

2. Modify the following PL/SQL block so that it uses a cursor FOR loop. Keep the explicit cursor declaration in the DECLARE section. Test your changes.

```
DECLARE
  CURSOR wf_countries_cur IS
    SELECT country_name, national_holiday_name,
           national_holiday_date
      FROM wf_countries
      WHERE region_id = 5;
  wf_countries_rec wf_countries_cur%ROWTYPE;
BEGIN
  OPEN wf_countries_cur;
  LOOP
    FETCH wf_countries_cur INTO wf_countries_rec;
    EXIT WHEN wf_countries_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE ('Country: ' ||
          wf_countries_rec.country_name
          || ' National holiday: '||
          wf_countries_rec.national_holiday_name
          || ', held on: '||
          wf_countries_rec.national_holiday_date);
  END LOOP;
  CLOSE wf_countries_cur;
END;
```

```
DECLARE
  CURSOR wf_countries_cur IS
    SELECT country_name, national_holiday_name,
           national_holiday_date
      FROM wf_countries
      WHERE region_id = 5;
BEGIN
  FOR wf_countries_rec IN wf_countries_cur
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Country: ' ||
          wf_countries_rec.country_name
          || ' National holiday: '||
          wf_countries_rec.national_holiday_name
          || ', held on: '||
          wf_countries_rec.national_holiday_date);
  END LOOP;
END;
```

3. Modify your anwer to question 2 to declare the cursor using a subquery in the FOR … LOOP statement, rather than in the declaration section. Test your changes again.

```
BEGIN
  FOR wf_countries_rec IN
     (SELECT country_name, national_holiday_name,
             national_holiday_date
        FROM wf_countries
        WHERE region_id = 5)
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Country: ' ||
          wf_countries_rec.country_name
          || ' National holiday: '||
          wf_countries_rec.national_holiday_name
          || ', held on: '||
          wf_countries_rec.national_holiday_date);
  END LOOP;
END;
```

4. Using the wf_countries table, write a cursor that returns countries with a highest_elevation greater than 8,000 m. For each country, display the country_name, highest_elevation, and climate. Use a cursor FOR loop, declaring the cursor using a subquery in the FOR … LOOP statement.

```
BEGIN
  FOR countries_rec IN
    (SELECT country_name, highest_elevation, climate
       FROM wf_countries
         WHERE highest_elevation > 8000)
  LOOP
    DBMS_OUTPUT.PUT_LINE('Country name: '||
                      countries_rec.country_name ||
                      ' the highest elevation is: ' ||
                      countries_rec.highest_elevation ||
                      ' and the climate is: '||
                      countries_rec.climate);
  END LOOP;
END;
```

5. This question uses a join of the wf_spoken_languages and wf_countries tables with a GROUP BY and HAVING clause.

   Write a PL/SQL block to fetch and display all the countries that have more than six spoken languages. For each such country, display country_name and the number of spoken languages. Use a cursor FOR loop, but declare the cursor explicitly in the DECLARE section. After all the rows have been fetched and displayed, display an extra row showing the total number of countries having more than six languages. (Hint: Declare a variable to hold the value of %ROWCOUNT.)

```
DECLARE
  v_num_countries NUMBER(4);
  CURSOR languages_cur IS
    SELECT country_name, COUNT(*) AS no_of_countries
      FROM wf_countries c, wf_spoken_languages sl
      WHERE c.country_id = sl.country_id
      GROUP BY country_name
      HAVING COUNT(*) > 6;
BEGIN
  FOR languages_rec IN languages_cur
  LOOP
    DBMS_OUTPUT.PUT_LINE
     (languages_rec.country_name || ' has ' ||
      languages_rec.no_of_countries ||
      ' spoken languages');
    v_num_countries := languages_cur%ROWCOUNT;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(v_num_countries ||
        ' countries speak more than 6 languages');
END;
```

6. Why did your block in question 4 need to declare the cursor explicitly, instead of declaring it as a subquery in the FOR … LOOP statement?

Because we needed to access its %ROWCOUNT attribute. We can access cursor attributes only if the cursor is explicitly declared with a cursor name.

SECTION 5 LESSON 4 – Cursors with Parameters

Terminology
No new vocabulary for this lesson.

Try It/Solve It
1. Describe the benefit of using one or more parameters with a cursor.

Using parameters in the WHERE or HAVING clause of a cursor definition allows different subsets of data to be fetched using a single cursor definition.

2. Write a PL/SQL block to display the country_name and area of all countries in a chosen region. The region_id should be passed to the cursor as a parameter. Test your block using two region_ids: 5 (South America) and 30 (Eastern Asia) . Do not use a cursor FOR loop.

```
DECLARE
  CURSOR country_curs
    (p_region_id wf_countries.region_id%TYPE) IS
    SELECT country_name, area FROM wf_countries
      WHERE region_id = p_region_id;
  country_rec  country_curs%ROWTYPE;
BEGIN
  OPEN country_curs(5); -- retest with (30)
  LOOP
    FETCH country_curs INTO country_rec;
    EXIT WHEN country_curs%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name: ' ||
                     country_rec.country_name ||
                     ' Area: ' ||
                     country_rec.area);
  END LOOP;
  CLOSE country_curs;
END;
```

3. Modify your answer to question 2 to use a cursor FOR loop. You must still declare the cursor explicitly in the DECLARE section. Test it again using regions 5 and 30.

```
DECLARE
  CURSOR country_curs
    (p_region_id wf_countries.region_id%TYPE) IS
    SELECT country_name, area FROM wf_countries
      WHERE region_id = p_region_id;
BEGIN
  FOR country_rec in country_curs(5) -- retest with (30)
    LOOP
      DBMS_OUTPUT.PUT_LINE('Name: ' ||
                   country_rec.country_name ||
                   ' Area: ' ||
                   country_rec.area);
    END LOOP;
END;
```

4. Modify your answer to question 3 to display the country_name and area of all countries in a chosen region which have an area greater than a specific value. The region_id and specific area should be passed to the cursor as two parameters. Test your block twice using region_id 5 (South America): the first time with area = 200000 and the second time with area = 1000000.

```
DECLARE
  CURSOR country_curs
    (p_region_id wf_countries.region_id%TYPE,
     p_area      wf_countries.area%TYPE) IS
    SELECT country_name, area FROM wf_countries
      WHERE region_id = p_region_id
      AND   area      > p_area;
BEGIN
  FOR country_rec in country_curs(5,200000)
      /* retest using(5,1000000) */
    LOOP
      DBMS_OUTPUT.PUT_LINE('Name: ' ||
                   country_rec.country_name ||
                   ' Area: ' ||
                   country_rec.area);
    END LOOP;
END;
```

Extension Exercise

1. Modify your answer to question 4 to fetch and display two sets of countries in a single execution of the block. You should open and close the cursor twice, passing different parameter values to it each time. Before each set of output rows, display the message “Region: <region_id>   Minimum Area: <area> ”., for example “Region: 5 Minimum Area: 200000”. Test your changes using (5, 200000) and (30, 500000).

```
DECLARE
  CURSOR country_curs
    (p_region_id wf_countries.region_id%TYPE,
     p_area      wf_countries.area%TYPE) IS
    SELECT country_name, area FROM wf_countries
      WHERE region_id = p_region_id
      AND   area      > p_area;
  v_region_id   wf_countries.region_id%TYPE;
  v_min_area    wf_countries.area%TYPE;
BEGIN

  /* First set of rows */
  v_region_id := 5;
  v_min_area  := 200000;
  DBMS_OUTPUT.PUT_LINE('Region: ' || v_region_id ||
                   ' Minimum Area: ' || v_min_area);
  FOR country_rec in country_curs(v_region_id, v_min_area)
    LOOP
      DBMS_OUTPUT.PUT_LINE('Name: ' ||
                      country_rec.country_name ||
                      ' Area: ' ||
                      country_rec.area);
    END LOOP;

  /* Second set of rows */
  v_region_id := 30;
  v_min_area  := 500000;
  DBMS_OUTPUT.PUT_LINE('Region: ' || v_region_id ||
                   ' Minimum Area: ' || v_min_area);
  FOR country_rec in country_curs(v_region_id, v_min_area)
    LOOP
      DBMS_OUTPUT.PUT_LINE('Name: ' ||
                      country_rec.country_name ||
                      ' Area: ' ||
                      country_rec.area);
    END LOOP;

END;
```

SECTION 5 LESSON 5 – Using Cursors FOR UPDATE

Terminology

1. ____**FOR UPDATE**_________ Declares that each row is locked as it is being fetched so other users can not modify the rows while the cursor is open.
2. ____**NOWAIT**_____________ A keyword used to tell the Oracle server not to wait if the requested rows have already been locked by another user.

Try It/Solve It

In this Practice you will INSERT and later UPDATE rows in a new table: proposed_raises, which will store details of salary increases proposed for suitable employees. Create this table by executing the following SQL statement:

```
CREATE TABLE proposed_raises
 (date_proposed       DATE,
  date_approved       DATE,
  employee_id         NUMBER(6),
  department_id       NUMBER(4),
  original_salary     NUMBER(8,2),
  proposed_new_salary  NUMBER(8,2));
```

1. Write a PL/SQL block which inserts a row into proposed_raises for each eligible employee. The eligible employees are those whose salary is below a chosen value. The salary value is passed as a parameter to the cursor. For each eligible employee, insert a row into proposed_raises with date_proposed = today's date, date_appoved null, and proposed_new_salary 5% greater than the current salary. The cursor should LOCK the employees rows so that no-one can modify the employee data while we the cursor is open. Test your code using a chosen salary value of 5000.

```
DECLARE
  CURSOR emp_curs (p_min_salary employees.salary%TYPE)
    IS SELECT employee_id, department_id, salary
         FROM employees
         WHERE salary < p_min_salary
         FOR UPDATE;
BEGIN
  FOR emp_rec IN emp_curs(5000) LOOP
    INSERT INTO proposed_raises (date_proposed, employee_id,
          department_id, original_salary, proposed_new_salary)
      VALUES (SYSDATE, emp_rec.employee_id,
            emp_rec.department_id, emp_rec.salary,
            emp_rec.salary * 1.05);
  END LOOP;
END;
```

2. SELECT from the proposed_raises table to see the results of your INSERT statements. There should be six rows. If you run your block in question 1 more than once, make sure the proposed_raises table is empty before each test.

   SELECT * FROM proposed_raises;

   [DELETE FROM proposed_raises;]

3. Before starting this question, ensure that there are six rows in proposed_raises. Now imagine that these proposed salary increases have been approved by company management.

   A. Write and execute a PL/SQL block to read each row from the proposed_raises table. For each row, UPDATE the date_approved column with today's date. Use the WHERE CURRENT OF... syntax to UPDATE each row.

```
DECLARE
  CURSOR raises_cur IS
    SELECT employee_id
      FROM proposed_raises
      FOR UPDATE;
BEGIN
  FOR raises_rec in raises_cur LOOP
    UPDATE proposed_raises
      SET date_approved = SYSDATE
      WHERE CURRENT OF raises_cur;
  END LOOP;
END;
```

   B. SELECT from the proposed_raises table to view the updated data.

   SELECT * FROM proposed_raises;

   C. Management has now decided that employees in department 50 cannot have a salary increase after all. Modify your code from question 3 to DELETE employees in department 50 from proposed_raises. This could be done by a simple DML statement (DELETE FROM proposed_raises WHERE department_id = 50;) but we want to do it using a FOR UPDATE cursor. Test your code, and view the proposed_raises table again to check that the rows have been deleted.

```
DECLARE
  CURSOR raises_cur IS
    SELECT employee_id
      FROM proposed_raises
      WHERE department_id = 50
      FOR UPDATE;
BEGIN
  FOR raises_rec in raises_cur LOOP
    DELETE FROM proposed_raises
      WHERE CURRENT OF raises_cur;
  END LOOP;
END;
```

SELECT * FROM proposed_raises;

You should see 2 remaining rows in the table.

D. We are going to set up two sessions into the same schema. From one of the sessions we will manually update an employee row *NOT COMMITING*. From the other session we will try to update everyone's salary , again *NOT COMMITING*. You should see the difference between NOWAIT and WAIT when using FOR UPDATE.

IMPORTANT NOTE: in each of these sessions, do NOT leave the SQL Commands screen to visit another Application Express page (for example Object Browser or Home). If you leave SQL Commands, your updates will automatically be rolled back, releasing all locks being held.

a. In preparation, create a copy of the employees table by executing the following SQL statement. You should use the upd_emps table for the rest of this exercise.

```
CREATE TABLE upd_emps AS SELECT * FROM employees;
```

b. Open a second Application Express session in a new browser window and connect to your schema. Ensure that Autocommit is disabled in BOTH your sessions (uncheck the check box in the top left corner of the SQL Commands window).

c. In your first session, update employee_id 200 (Jennifer Whalen)'s first name to Jenny.

DO NOT COMMIT. You now have a lock on row 200 that will last indefinitely.

UPDATE upd_emps SET first_name = 'Jenny' WHERE employee_id=200;

d. In your second session, write a PL/SQL block to give every employee in upd_emps a $1 salary raise. Your cursor should be declared FOR UPDATE NOWAIT. Execute your code. What happens?

```
DECLARE
  CURSOR upd_emps_cur IS
    SELECT employee_id, salary
      FROM upd_emps
      FOR UPDATE NOWAIT;
BEGIN
  FOR upd_emps_rec IN upd_emps_cur LOOP
    UPDATE upd_emps
      SET salary = salary + 1
      WHERE CURRENT OF upd_emps_cur;
  END LOOP;
END;
```

The second session returns the error message:

ORA-00054: resource busy and acquire with NOWAIT specified

e. Still in your second session, modify your block to remove the NOWAIT attribute from the cursor declaration. Re-execute the block. What happens this time?

```
DECLARE
  CURSOR upd_emps_cur IS
    SELECT employee_id, salary
      FROM upd_emps
      FOR UPDATE;
BEGIN
  FOR upd_emps_rec in upd_emps_cur LOOP
    UPDATE upd_emps
      SET salary = salary + 1
      WHERE CURRENT OF upd_emps_cur;
  END LOOP;
END;
```

The second session hangs indefinitely, waiting for the first session to release its lock on Jennifer Smith.

f. After waiting a minute or so, switch to your first session and COMMIT the update to Jennifer Smith's row. Then switch back to your second session. What happened?

As soon as the first session committed the update and released its lock, the second session sprang back into life and continued executing. It has now updated all the rows … but not COMMITTed.

g. Clean up by COMMITTing the updates in your second session.

Extension Exercise

1. For this question you will also need two separate Application Express sessions in two browser windows, with Autocommit turned off in both sessions. You will also need a copy of the departments table. Create this copy by executing the following SQL statement:

   CREATE TABLE upd_depts AS SELECT * FROM departments;

   A. Modify your $1 salary raise block from the previous question so that the cursor SELECTS from a join of upd_emps and upd_depts.

```
DECLARE
  CURSOR upd_emps_cur IS
    SELECT employee_id, salary, department_name
      FROM upd_emps e, upd_depts d
      WHERE e.department_id = d.department_id
      FOR UPDATE;
BEGIN
  FOR upd_emps_rec in upd_emps_cur LOOP
    UPDATE upd_emps
      SET salary = salary + 1
      WHERE CURRENT OF upd_emps_cur;
  END LOOP;
END;
```

   B. Run the block in your second session. It should execute successfully, updating all the upd_emps rows but not committing.

   C. Switch to your first session and try to update an upd_depts row using the following statement. What happens and why ?

```
UPDATE upd_depts SET department_name = 'Accounting'
  WHERE department_id = 50;
```

   The UPDATE statement hangs indefinitely waiting for a lock. This is because the second session's FOR UPDATE cursor has locked both upd_emps rows and upd_depts rows. upd_depts has been locked unnecessarily (the cursor is not updating it).

   D. Release all the locks by committing in your second session and then in your first session.

   E. How would you prevent your cursor from locking upd_depts rows unnecessarily? Modify your block to avoid the unnecessary locking, and rerun the test you did in steps b and c. What happens this time?

```
DECLARE
  CURSOR upd_emps_cur IS
    SELECT employee_id, salary, department_name
      FROM upd_emps e, upd_depts d
      WHERE e.department_id = d.department_id
      FOR UPDATE OF salary;
BEGIN
  FOR upd_emps_rec in upd_emps_cur LOOP
    UPDATE upd_emps
      SET salary = salary + 1
      WHERE CURRENT OF upd_emps_cur;
  END LOOP;
END;
```

The cursor is no longer locking upd_depts, so the first session's UPDATE statement should execute immediately without waiting for a lock.

F. Clean up by COMMITTing your updates in both sessions and re-enabling Autocommit.

SECTION 5 LESSON 6 – Using Multiple Cursors

Terminology
No new vocabulary for this lesson.

Try It/Solve It

1. Write and run a PL/SQL block which produces a listing of departments and their employees. Use the departments and employees tables. In a cursor FOR loop, retrieve and display the department_id and department_name for each department, and display a second line containing '----------' as a separator. In a nested cursor FOR loop, retrieve and display the first_name, last_name and salary of each employee in that department., followed by a blank line at the end of each department. Order the departments by department_id, and the employees in each department by last_name.

   You will need to declare two cursors, one to fetch and display the departments, the second to fetch and display the employees in that department, passing the department_id as a parameter.

   Your output should look something like this (only the first few departments are shown):

```
10 Administration
-----------------------------
Jennifer Whalen 4400

20 Marketing
-----------------------------
Pat Fay 6000
Michael Hartstein 13000

50 Shipping
-----------------------------
Curtis Davies 3400
Randall Matos 2600
Kevin Mourgos 5800
Trenna Rajs 3500
Peter Vargas 2500
```

```
DECLARE
  CURSOR dept_cursor IS
    SELECT department_id,department_name
      FROM     departments
      ORDER BY department_id;
  CURSOR emp_cursor(v_deptno departments.department_id%TYPE) IS
      SELECT first_name, last_name, salary
        FROM    employees
        WHERE department_id = v_deptno
        ORDER BY last_name;

BEGIN

  /* Outer FOR loop: executed once for each department */
  FOR dept_record IN dept_cursor LOOP
    DBMS_OUTPUT.PUT_LINE(dept_record.department_id ||
                     ' ' || dept_record.department_name);
    DBMS_OUTPUT.PUT_LINE('-----------------------------');

    /* Inner FOR loop: executed once for each employee
         in a department */
    FOR emp_record IN emp_cursor(dept_record.department_id) LOOP
      DBMS_OUTPUT.PUT_LINE (emp_record.first_name || ' ' ||
                       emp_record.last_name || ' ' ||
                       emp_record.salary);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('    ');
  END LOOP;
END;
```

2. Write and run a PL/SQL block which produces a report listing world regions and countries in those regions. You will need two cursors: an outer loop cursor which fetches and displays rows from wf_world_regions, and an inner loop cursor which fetches and displays rows from wf_countries for countries in that region, passing the region_id as a parameter. Restrict your regions to those in America (region_name like '%America%'). Order your output by region_name, and by country_name within each region.

   Your output should look something like this (only the first two regions are shown):

```
13 Central America
-----------------------------
Belize 22966 287730
Republic of Costa Rica 51100 4075261
Republic of El Salvador 21040 6822378
Republic of Guatemala 108890 12293545
Republic of Honduras 112090 7326496
Republic of Nicaragua 129494 5570129
Republic of Panama 78200 3191319
United Mexican States 1972550 107449525

21 Nothern America
-----------------------------
Bermuda 53 65773
Canada 9984670 33098932
Greenland 2166086 56361
Territorial Collectivity of Saint Pierre and Miquelon 242 7026
United States of America 9631420 298444215
```

```
DECLARE
  CURSOR region_curs IS
    SELECT region_id, region_name
      FROM      wf_world_regions
      WHERE region_name like '%America%'
      ORDER BY region_name;
  CURSOR country_curs
        (p_region_id wf_world_regions.region_id%TYPE) IS
      SELECT country_name, area, population
        FROM      wf_countries
        WHERE     region_id = p_region_id
        ORDER BY country_name;

BEGIN

  /* Outer FOR loop: executed once for each region */
  FOR region_rec IN region_curs LOOP
    DBMS_OUTPUT.PUT_LINE(region_rec.region_id ||
                    ' ' || region_rec.region_name);
    DBMS_OUTPUT.PUT_LINE('-----------------------------');

    /* Inner FOR loop: executed once for each country
        in a region */
    FOR country_rec IN country_curs(region_rec.region_id) LOOP
      DBMS_OUTPUT.PUT_LINE(country_rec.country_name || ' ' ||
                      country_rec.area || ' ' ||
                      country_rec.population);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('    ');
  END LOOP;
END;
```

Extension Exercise

1. Modify your block from question 2 to display the names of official spoken languages in each country. You will need three cursors and three loops. The first two cursors should fetch and display regions and countries, as in question 2. The innermost loop should accept a country_id as a parameter, and fetch and display the name of each official language, using a join of wf_spoken_languages and wf_languages. Within each country, the languages should be ordered by language_name. Test your block, restricting regions to those in America.

   Your output should look something like this (only the first two regions are shown):

```
13 Central America
-----------------------------
Belize 22966 287730
--- English
Republic of Costa Rica 51100 4075261
--- Spanish
Republic of El Salvador 21040 6822378
Republic of Guatemala 108890 12293545
Republic of Honduras 112090 7326496
Republic of Nicaragua 129494 5570129
--- Spanish
Republic of Panama 78200 3191319
--- Spanish
United Mexican States 1972550 107449525

21 Nothern America
-----------------------------
Bermuda 53 65773
--- English
Canada 9984670 33098932
--- English
--- French
Greenland 2166086 56361
Territorial Collectivity of Saint Pierre and Miquelon 242 7026
--- French
United States of America 9631420 298444215
--- English
```

```
DECLARE
  CURSOR region_curs IS
    SELECT region_id, region_name
      FROM      wf_world_regions
      WHERE region_name like '%America%'
      ORDER BY region_name;
  CURSOR country_curs
         (p_region_id wf_world_regions.region_id%TYPE) IS
      SELECT country_id, country_name, area, population
        FROM    wf_countries
        WHERE region_id = p_region_id
        ORDER BY country_name;
  CURSOR language_curs
         (p_country_id wf_countries.country_id%TYPE) IS
      SELECT language_name
        FROM wf_spoken_languages sl, wf_languages l
        WHERE sl.language_id = l.language_id
        AND   official = 'Y'
        AND   country_id = p_country_id
        ORDER BY language_name;

BEGIN

  /* Outer FOR loop: executed once for each region */
  FOR region_rec IN region_curs LOOP
    DBMS_OUTPUT.PUT_LINE(region_rec.region_id ||
                       ' ' || region_rec.region_name);
    DBMS_OUTPUT.PUT_LINE('----------------------------');

    /* Inner FOR loop: executed once for each country
         in a region */
    FOR country_rec in country_curs(region_rec.region_id) LOOP
      DBMS_OUTPUT.PUT_LINE (country_rec.country_name || ' ' ||
                       country_rec.area || ' ' ||
                       country_rec.population);
      /* Innermost FOR loop: executed once for each official
         language in a country */
      FOR language_rec IN language_curs(country_rec.country_id)
        LOOP
          DBMS_OUTPUT.PUT_LINE('--- ' ||
                         language_rec.language_name);
        END LOOP;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('    ');
  END LOOP;
END;
```