Basics of PL/SQL

Writing Your First Program

A SIMPLE PL/SQL CODE BLOCK THAT DISPLAYS THE WORD HELLO

```
SQL> set serveroutput on
SQL> begin
  2   dbms_output.put_line ('Hello');
  3  end;
  4  /
Hello

PL/SQL procedure successfully completed.

SQL>
```

End listing

”. Some important features of the program are:

- The executable portion of a PL/SQL code block starts with the keyword Begin and is terminated with the keyword End.
- PL/SQL code blocks are comprised of statements. Each statement ends with a semi-colon.
- PL/SQL code blocks are followed by a slash (/) in the first position of the following line. This causes the code block statements to be executed.
- The only PL/SQL code block keyword that is followed by a semi-colon is the End keyword.

Executing the PL/SQL Program

Executing a PL/SQL Program

SQL> START C:\BUSINESS\ORACLE~1\PLSQL1\L1.SQL

HELLO

PL/SQL PROCEDURE SUCCESSFULLY COMPLETED

End listing

Practice

1. Create a program that outputs the message “I am soon to be a PL/SQL expert.”

CODE BLOCK COMPONENTS AND BLOCK LABELS

Code Block Sections

There are four types of code block sections. These are:

- Header - This is the optional first section of the code block. It is used to identify the type of code block and its name**.** The code block types are: anonymous procedure, named procedure, and function**.** A header is only used for the latter two types.
- Declaration - This is an optional section of the code block. It contains the name of the local objects that will be used in the code block. These include variables, cursor definitions, and exceptions. This section begins with the keyword Declare.
- Executable - This is the only mandatory section. It contains the statements that will be executed. These consist of SQL statements, DML statements, procedures (PL/SQL code blocks), functions (PL/SQL code blocks that return a value), and built-in subprograms. This section starts with the keyword Begin.
- Exception - This is an optional section. It is used to “handle” any errors that occur during the execution of the statements and commands in the executable section. This section begins with the keyword Exception.

The code block is terminated by the End keyword. This is the only keyword within the construct that is followed by a semi-colon (;). The only required section is the executable section. This means the code block must have the Begin and End keywords. The code block is executed by the slash (/) symbol.

Executing a PL/SQL Program

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2   LOCAL_VARIABLE        VARCHAR2(30);
  3  BEGIN
  4   SELECT 'NUMBER OF
EMPLOYEES'||TO_CHAR(COUNT(LAST_NAME),
'999')
  5     INTO LOCAL_VARIABLE
  6   FROM EMPLOYEE;
  7   DBMS_OUTPUT.PUT_LINE
(LOCAL_VARIABLE);
  8  EXCEPTION
  9   WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('ERROR
OCCURED');
 10  END;
 11  /
NUMBER OF EMPLOYEES  19

PL/SQL PROCEDURE SUCCESSFULLY
COMPLETED.
```

End Listing

Block Labels, Labels, and the Goto Keyword

Some rules to remember are:

- Labels are defined by placing two less than (<<) symbols before the label name and two greater than (>>) symbols after the label name.
- The Goto keyword is used to redirect the focus of the code block. The name of the label is placed after the Goto keyword.

A Block label is similar to a label except that it can be used to qualify the contents of a block. The Block label is placed at the beginning of the block. The label is then placed following the End keyword**.** By placing the label definition and the end label, you can identify the code block and the variables within the labeled block. This can be a useful device when the program contains multiple code blocks.

A Block Label, Label, and Goto Command

```
SQL> BEGIN <<B_LABEL>>
  2    GOTO MIDDLE;
  3    <<TOP>>
  4    DBMS_OUTPUT.PUT_LINE ('TOP
STATEMENT');
  5    GOTO BOTTOM;
  6    <<MIDDLE>>
  7    DBMS_OUTPUT.PUT_LINE ('MIDDLE
STATEMENT');
  8    GOTO TOP;
  9    <<BOTTOM>>
 10    DBMS_OUTPUT.PUT_LINE ('BOTTOM
STATEMENT');
 11  END B_LABEL;
 12  /
MIDDLE STATEMENT
TOP STATEMENT
BOTTOM STATEMENT

PL/SQL PROCEDURE SUCCESSFULLY
COMPLETED.

SQL>
```

End listing

Comments

Comments can be entered into the code block. Two devices are available. These are:

-- Two dashes placed at the beginning of the line will comment out the entire line.

/* */ The slash-star (**/***) symbol marks the beginning of a commented area. The star-slash (*/) symbol marks the ending. Multiple statements can be included in the commented section.

Practice

2. Create a PL/SQL procedure that has four sections. Each section should output a statement. Use labels and the Goto command to output the section messages in the following order:

   Section 3
   Section 2
   Section 1
   Section 4

DECLARING VARIABLES AND ASSIGNING VALUES

Defining Variables

Variables are defined in the declaration section of the program. The syntax is:

Variable_name datatype(precision);

Oracle treats a variable definition similar to other statements. The definition must end with a semi-colon. The definition statement begins with the variable name and contains the data type. These are the mandatory parts of the definition. A value may also be assigned to the variable during the definition statement. The variable may also be constrained.

Character Definitions

The following are examples of definitions:

first_name varchar2(15);
social_security_number char(11);

The PL/SQL maximum length of the char and varchar2 data types is larger than the length allowed in the Oracle database.

Numeric Definitions

Numeric data definitions can include two parameters. The first parameter is precision and the second is scale. Precision defines the overall length of the value. Scale determines the number of digits to the left or right of the decimal point. The range of scale is –84 to 127.

If a scale is specified, rounding will occur at the end. The following rules apply:

- Positive scale definitions cause rounding to the right of the decimal point.
- Negative scale definitions cause rounding to the left of the decimal point.
- Zero scale definitions cause rounding to the nearest whole number.

The default precision of a number is 38.

age	**integer(3);**
gallons	**number(3);**
salary	**number(8,2);**

Other Definitions

Several other types of definitions are available. These are:

- Boolean — This variable type is used to record a condition. The value can be true, false, or null.
- Date — This variable type is used to record date values.
- Exception — This variable type is used to define a custom named exception or error handler.

The following are several example definitions:

```
yes          boolean;
e_day        date;
big_error    exception;
```

Constrained Definitions

Constraints can be placed on the variables defined in the code block. A constraint is a condition that is placed on the variable. Two common constraints are:

- Constant - This constraint will cause Oracle to ensure the value is not changed after a value is initially assigned to the variable. If a statement tries to change the variable value, an error will occur.
- Not Null - This constraint will cause Oracle to ensure the variable always contains a value. If a statement attempts to assign a null value to the variable, an error will occur.

The following are example of constrained variable definitions:

```
pi            constant number(9,8) := 3.14159265;

birth_date    not null date := '08-APR-53';
```

Aggregate and PL/SQL Record Definitions

An aggregate variable definition is based upon a database or PL/SQL object. They consist of one or more variables and are extremely useful. They have two advantages:

1. The developer can automatically define a variable with the same data specifications as a table column or cursor variable without actually knowing the specifications.

2. The developer can set up an array of variables for a cursor or table record with one statement. The variables will have the same specifications as the table or cursor variables.

Variable_name table_cursor_name.column_name%type;

lname employee.last_name%type;

Array_name **table/cursor_name%rowtype;**

Dept_var **department%rowtype;**

Dept_var.department

Dept_var.department_name

Assigning Values to Variables

A PL/SQL procedure would not be useful unless there is a way to populate the variables with a value. Fortunately, PL/SQL gives us two ways to accomplish this. These are:

- **:=** - The colon/equal sign assigns the argument on the left of the operator to the argument or variable on the right of the sign.
- Into - The Into keyword is used in a Select or Fetch statement. When used in a Select statement, it assigns the values in the Select clause to the variables following the Into keyword. When used with the Fetch statement, it assigns the cursor values to the variables that follow the Into keyword.

Assigning Values to Variables

PL/SQL Record Definiti

Assigning values to variables

```
SQL> declare
  2    retirement_date        date;
  3    emp_var               employee%rowtype;
  4  begin
  5    select min(birth_date)
  6    into emp_var.birth_date
  7    from employee;
  8    retirement_date := add_months(emp_var.birth_date,
12*65);
  9    dbms_output.put_line (to_char(retirement_date));
 10  end;
 11  /
29-DEC-73

PL/SQL procedure successfully completed.

SQL>
```

End Listing

Using the Into Assignment Keyword With PL/SQL Records

Assigning Values to Variables

```
SQL> declare
  2    retirement_date        date;
  3    emp_var               employee%rowtype;
  4  begin
  5    select *
  6    into emp_var
  7    from employee where last_name = 'ANTHONY';
  8    retirement_date := add_months(emp_var.birth_date,
12*65);
  9    dbms_output.put_line (to_char(retirement_date));
 10  end;
 11  /
15-FEB-85

PL/SQL procedure successfully completed.

SQL>
```

End listing

Practice

3. Create a PL/SQL procedure that computes the retirement age of the youngest employee. You should also list the employee's name.

4. Modify the program in #3 to compute the number of days between today and the employee's retirement date.

5. Identify the number of tool purchases for Harry Truman and George Bush. Output the name of the employee with the greater number of tool purchases.

THE IF-THEN-ELSE STRUCTURE

To be effective, a code block or procedure needs to have commands that allow the developer to document the logic necessary to determine the behavior. Oracle uses conditional logic statements to form the procedure's behavior. The logic statements come in two forms. These are:

If-then-else structures

Elsif statements

The If-Then-Else Structure

The basic structure is as follows:

If (*conditional expression*) then
_____ *Statements;*
Else
_____ *Statements;*
End if;

An If-Then-Else Structure Example

```
SQL> declare
  2    male_avg       number;
  3    female_avg     number;
  4  begin
  5    select avg(months_between(employment_date, birth_date)/12)
  6      into male_avg
  7    from employee
  8    where gender = 'M';
  9    select avg(months_between(employment_date, birth_date)/12)
 10      into female_avg
 11    from employee
 12    where gender = 'F';
 13    if (male_avg > female_avg) then
 14     dbms_output.put_line ('Males have the greatest avg hiring age');
 15     dbms_output.put_line ('With and avg age of '||to_char(male_avg));
 16    else
 17     dbms_output.put_line ('Females have the greatest avg hiring age');
 18     dbms_output.put_line ('With and avg age of '||to_char(female_avg));
 19    end if;
 20  end;
 21  /
Males have the greatest avg hiring age
With and avg age of 55.9176154332700822264389626818469323214 1

PL/SQL procedure successfully completed.

SQL>
```

End Listing

Nested If-Then-Else Structures

An If-Then-Else Structure With a Nested If Statement

Outer If Condition

Inner or Nested If Condition

```
SQL> declare
  2    male_avg       number;
  3    female_avg     number;
  4  begin
  5    select avg(months_between(employment_date, birth_date)/12)
  6      into male_avg
  7    from employee
  8    where gender = 'M';
  9    select avg(months_between(employment_date, birth_date)/12)
 10      into female_avg
 11    from employee
 12    where gender = 'F';
 13    if (male_avg > female_avg) then
 14     dbms_output.put_line ('Males have the greatest avg hiring age');
 15     dbms_output.put_line ('With and avg age of '||to_char(male_avg));
 16     if (male_avg > female_avg + 10) then
 17      dbms_output.put_line ('The male average is greater than 10 years');
 18     end if;
 19    else
 20     dbms_output.put_line ('Females have the greatest avg hiring age');
 21     dbms_output.put_line ('With and avg age of '||to_char(female_avg));
 22    end if;
 23  end;
 24  /
Males have the greatest avg hiring age
With and avg age of 55.9176154332700822264389626818469323214 1
The male average is greater than 10 years

PL/SQL procedure successfully completed.

SQL>
```

End Listing

The Elsif/Then Structure

An If-Then-Elsif Structure

```
SQL> declare
  2    current_month       char(3);
  3  begin
  4    select to_char(sysdate, 'MON') into current_month from dual;
  5    if current_month = 'JAN' then
  6      dbms_output.put_line ('My daughter Jane was born in January');
  7    elsif current_month = 'FEB' then
  8      dbms_output.put_line ('My good friend Ron was born in February');
  9    elsif current_month = 'MAR' then
 10      dbms_output.put_line ('My father was born in March');
 11    elsif current_month = 'APR' then
 12      dbms_output.put_line ('I was born in April');
 13    elsif current_month = 'MAY' then
 14      dbms_output.put_line ('My son Matt was born in May');
 15    elsif current_month = 'OCT' then
 16      dbms_output.put_line ('My wife was born in October');
 17    else
 18      dbms_output.put_line ('I do not have any relatives
 19                             born in '||current_month);
 20    end if;
 21  end;
 22  /
I do not have any relatives born in JUN

PL/SQL procedure successfully completed.

SQL>
```

End Listing

Practice

6. Use a nested-if statement to output whether the highest employee in #5 had two or more than the lower.

7. Output which decade of the twentieth century Bill Clinton was born in.

8. Create a PL/SQL procedure that computes and displays the average starting age of the set of employees in the Employee database.

CURSORS

A **cursor** is a device that is used to retrieve a set of records from a table/view into memory. Cursors allow each of the records to be read into the code block and processed one-at-a-time. A cursor can be compared to a book containing a page mark. Each of the pages is a record in the set of records retrieved when the cursor is executed. The bookmark indicates the current page. When using a cursor**,** Oracle always knows the current record. As one record is read into the code block**,** the current record is changed just as the bookmark is changed as a page is read. Cursors are important tools for the processing of records. They allow the developer to bring records into the code block and to process them using a complex set of statements

Declaring the Cursor

Cursors are defined in the Declaration section of the code block. The definition consists of the keywords Cursor and Is, the name of the cursor, and the Select statement used to retrieve the record set. The following is an example of the cursor definition structure:

Cursor *cusor name* is *select statement;*

Cursor Commands

There are three commands that are used in conjunction with cursors. These commands are contained in Table:

Cursor Commands

Command	Example	Description
Open	Open *cursor_name;*	This command executes the cursor's Select statement and places the records into memory. The first record in the set is the current set.
Fetch/into	Fetch *cursor_name* into *variables;*	This command assigns the values from the current cursor record to the listed local variables or PL/SQL record. It also makes the next record in the set the current record.
Close	Close *cursor_name;*	Terminates the cursor and frees the memory used by the cursor for other uses.

Several items to remember about cursor commands are:

- The commands end with a semi-colon.
- Issuing the Open command when the cursor is currently open will cause an error and terminate the procedure.
- Issuing the Close command when the Cursor is not open will cause an error and terminate the procedure.
- Issuing the Fetch/into command when the cursor is not open will cause an error and terminate the procedure.
- Issuing the Fetch/Into command after the last record has been fetched will not cause an error. The values from the last record will be reassigned to the local variables.

Using Cursors and Cursor Commands

```
SQL> declare
  2    oldest_birth_date     date;
  3    lname                 employee.last_name%type;
  4    fname                 employee.first_name%type;
  5    cursor find_old_b_day is select min(birth_date) from
employee;
  6    cursor id_employee is select last_name, first_name
  7                          from employee
  8                          where birth_date =
oldest_birth_date;
  9  begin
 10    open find_old_b_day;
 11    fetch find_old_b_day into oldest_birth_
 12    close find_old_b_day;
 13    open id_employee;
 14    fetch id_employee into lname, fname;
 15    close id_employee;
 16   dbms_output.put_line ('The Oldest Employee Is'
 17                          ||lname||', '||fname);
 18  end;
 19  /
The Oldest Employee Is JOHNSON, ANDREW

PL/SQL procedure successfully completed.

SQL>
```

Value from the old_b_day cursor is used as an argument

Cursor commands

End Listing

Using Aggregate Variables With Cursors

In the previous example local variables were defined for each of the columns retrieved by the cursors. The developer had to declare each of the variables used to assign cursor values and also had to include them in the fetch statements. There are two potential problems with this method. These are:

1. The developer must document the local variable's size and type. If the size of the cursor variable is larger than the size of the local variable its value is assigned to, an error will occur and the procedure will terminate. The procedure will also terminate if the data types are different.

2. If the size of the column is changed, the procedure variables will also need to be changed. Failure to change the procedure may cause the procedure to terminate when run.

Using %rowtype to Define Cursor Variables

```
SQL> declare
  2   cursor find_old_b_day is select min(birth_date) day
  3   from employee;
  4    old_date              find_old_b_day%rowtype;
  5    cursor id_employee is select last_name, first_name
  6                          from employee
  7                          where birth_date =
old_date.day;
  8    id                    id_employee%rowtype;
  9  begin
 10    open find_old_b_day;
 11    fetch find_old_b_day into old_date;
 12    close find_old_b_day;
 13    open id_employee;
 14    fetch id_employee into id;
 15    close id_employee;
 16   dbms_output.put_line ('The Oldest Employee Is '
 17                          ||id.last_name||',
      '||id.first_name);
 18  end;
 19  /
The Oldest Employee Is JOHNSON, ANDREW

PL/SQL procedure successfully completed.

SQL>
```

PL/SQL records

PL/SQL record variable

End Listing

Several things to remember when using %rowtype are:

- The cursor must be defined before the PL/SQL record definition.
- All cursor columns must have a name. When expressions are included such as the case of group functions, you must include a column alias.

Practice

9. Create a PL/SQL procedure that computes the hiring age of the first employee hired by the “WEL” department.

Cursor Attributes

Cursor Attributes

Name	Description
%found	This attribute is true if the last fetch statement returned a record. It is false if it did not.
%notfound	This attribute is true if the last fetch statement did not return a record. It is false if it did.
%rowcount	This attribute returns the number of fetch commands that have been issued for the cursor.
%isopen	This attribute is true if the indicated cursor is currently open. It is false if the cursor is currently closed.

These commands are used in a condition within the procedure. They are used to evaluate the condition of a cursor. Based upon this condition, an action will occur. The syntax of the expression is as follows:

***Cursor_name*%isopen**

Using %isopen Cursor Attribute to Control Errors

```
CHAPTER 1 SQL> DECLARE
  2    CURSOR NAME IS SELECT
MAX(FIRST_NAME) FNAME,
CHAPTER 2   3                MAX(LAST_NA
LNAME
  4              FROM EMPLOYEE;
  5    NAMES      NAME%ROWTYPE;
CHAPTER 3   6  BEGIN
CHAPTER 4   7    IF NOT NAME%ISOPEN THEN
CHAPTER 5   8      OPEN NAME;
CHAPTER 6   9    END IF;
CHAPTER 7  10    FETCH NAME INTO NAMES;
CHAPTER 8  11    DBMS_OUTPUT.PUT_LINE
(NAMES.FNAME||' '||NAMES.LNAME);
CHAPTER 9  12    IF NAME%ISOPEN THEN
CHAPTER 10  13      CLOSE NAME;
CHAPTER 11  14    END IF;
CHAPTER 12  15  END;
CHAPTER 13  16  /
CHAPTER 14 WOODROW WILSON
CHAPTER 15
CHAPTER 16 PL/SQL PROCEDURE
SUCCESSFULLY COMPLETED.
CHAPTER 17
CHAPTER 18 SQL>
CHAPTER 19
```

%Isopen cursor variable

End listing

Differences Between a Cursor and a Select/Into Statement

A cursor and a Select/Into statement are similar in that they both can be used to retrieve values for local variables. There are two shortcomings with the Select/Into statement. These are:

1. The Select/Into statement cannot be used to process multiple database records. If the Select command retrieves more than one record, an error will occur.

2. If the Select/Into statement does not return a record from the database, an error will occur.

A cursor does not have these limitations. Cursors can process multiple records. In addition, failure of the cursor to retrieve a record will not cause an error to occur. Null values will be brought into the procedure variables by the fetch command. For these two reasons, a cursor is preferable to the Select/Into statement.

Practice

10. Cause a “cursor already open” error to occur.

11. Fix the error produced in #10 using the %isopen cursor attribute.

LOOPS

There are three types of **looping structures**. These are the Loop structure, While structure, and For structure. The former two structures will be discussed in this section. The For looping structure will be discussed in the next section.

Each of the loop structures has three things in common:

1. The structure contains the Loop keyword.
2. Each structure ends with the End loop keywords.
3. Each structure uses a conditional expression to determine whether to stop the looping.

The Loop Structure

The following is a template of the Loop structure:

Loop

Statements;

When *break_out condition* then exit;

Statements;

End loop;

The Loop Structure

```
SQL> declare
  2    counter_variable   number := 1;
  3    cursor a is select last_name from employee;
  4    cur_var            a%rowtype;
  5  begin
  6    open a;

  7    loop
  8      exit when counter_variable = 7;
  9      fetch a into cur_var;
 10      dbms_output.put_line (cur_var.last_name);
 11      counter_variable := counter_variable +1;
 12    end loop;
 13  end;
 14  /
COOLIDGE
JOHNSON
REAGAN
BUSH
JOHNSON
CLINTON

PL/SQL procedure successfully completed.
SQL>
```

Beginning of loop

Breakout statement

End of the loop structure

End listing

The Loop Structure Using the If-Then Structure to Terminate the Loop

```
SQL> declare
  2    counter_variable   number := 1;
  3    cursor a is select last_name from employee;
  4    cur_var            a%rowtype;
  5  begin
  6    open a;
  7    loop
  8      if counter_variable = 7 then exit; end if;
  9      fetch a into cur_var;
 10      dbms_output.put_line (cur_var.last_name);
 11      counter_variable := counter_variable +1;
 12    end loop;
 13  end;
 14  /
COOLIDGE
JOHNSON
REAGAN
BUSH
JOHNSON
CLINTON

PL/SQL procedure successfully completed.

SQL>
```

If statement used to breakout of

End listing

Practice

12. Determine the hiring date for Ronald Reagan and how many tool and eyeglass purchases he made.

13. Use a simple loop to list the first 12 records of the Emp_tools table. Use the When keyword to construct the loop breakout.

14. Modify your procedure in #13. Use the If-then structure to construct the loop breakout.

The While Loop

. The following is a syntax template of the While looping structure:

While *breakout_condition*
Loop
Statements;
End loop;

The While loop Structure

```
SQL> declare
  2    counter_variable   number := 1;
  3    cursor a is select last_name from employee;
  4    cur_var            a%rowtype;
  5  begin
  6    open a;
  7    while counter_variable != 7
  8    loop
  9      fetch a into cur_var;
 10      dbms_output.put_line (cur_var.last_name);
 11      counter_variable := counter_variable +1;
 12    end loop;
 13  end;
 14  /
COOLIDGE
JOHNSON
REAGAN
BUSH
JOHNSON
CLINTON

PL/SQL procedure successfully completed.

SQL>
```

While loop condition

End listing
Using the %found

Using the %found cursor attribute with loops

Open *cursor_name;*
Fetch *cursor_attributes* into *local_variables;*
While (*cursor_name*%found)
Loop
    Statements;
    Fetch *cursor_attributes* into *local_variables;*
End loop;
Close *cursor_name;*

Illustrates a Loop Using the %found Cursor Attribute.

A While loop using the %found attribute

```
SQL> declare
  2    cursor a is select last_name from employee;
  3    cur_var            a%rowtype;
  4  begin
  5    open a;
  6    fetch a into cur_var;
  7    while a%found
  8    loop
  9      dbms_output.put_line (cur_var.last_name);
 10      fetch a into cur_var;
 11    end loop;
 12  end;
 13  /
COOLIDGE
.
.
.
ROOSEVELT
ANTHONY
ROOSEVELT

PL/SQL procedure successfully completed.

SQL>
```

A record is fetched before the loop

Another fetch command is used at the end of

End listing

Nested Loops

A Nested While Loop Used to Compute the Highest Priced Tool

```
SQL> declare
  2    cursor a is select payroll_number, last_name from employee
  3                where fk_department = 'WEL';
  4    a_var       a%rowtype;
  5    cursor b is select tool_name, tool_cost from emp_tools
  6                where fk_payroll_number = a_var.payroll_number;
  7    b_var              b%rowtype;
  8    hi_tool_name      emp_tools.tool_name%type;
  9    hi_tool_cost      emp_tools.tool_cost%type;
 10  begin
 11    open a; fetch a into a_var;
 12    while a%found loop
 13      open b; fetch b into b_var;
 14      while b%found loop
 15        if b_var.tool_cost > hi_tool_cost or b_var.tool_cost is null then
 16           hi_tool_name := b_var.tool_name;
 17           hi_tool_cost := b_var.tool_cost;
 18        end if;
 19        fetch b into b_var;
 20       end loop;
 21       close b;
 22       dbms_output.put_line (a_var.last_name||' '||b_var.tool_name);
 23       hi_tool_name := null;
 24       hi_tool_cost := null;
 25       fetch a into a_var;
 26    end loop;
 27    close a;
 28  end;
 29  /
REAGAN Tool Chest
CARTER Tool Chest
HOOVER TIN SNIPS
TAFT FOUNTAIN PEN
ANTHONY STAPLER
ROOSEVELT PLIERS

PL/SQL procedure successfully completed.

SQL>
```

Nested

End listing

Practice

15. Create a procedure that displays the employees in the “INT” department. Use a While loop.

16. Create a procedure that determines the number of tool purchases and the number of eyeglass purchases per employee. Use the %rowcount cursor attribute to number the displayed rows.

17. Recreate #15 using the %notfound cursor attribute.

Locking Records With the For Update Of Option

A Cursor Select Statement Using the For Update Option

```
SQL> declare
  2    cursor a is select last_name, first_name from employee
  3                where fk_department = 'WEL'
  4                for update;
  5    a_var      a%rowtype;
  6  begin
  7    open a;
  8    fetch a into a_var;
  9    while a%found loop
 10      dbms_output.put_line (a_var.last_name);
 11      fetch a into a_var;
 12    end loop;
 13  end;
 14  /
REAGAN
.
ROOSEVELT
PL/SQL procedure successfully completed.
SQL>
```

For Update Keyword That Locks the Cursor Records

End Listing

The For Update Of Option

A Cursor With the For Update Of Option

```
SQL> declare
  2    cursor a is select last_name, first_name from employee
  3                where fk_department = 'WEL'
  4                for update of wages;
  5    a_var      a%rowtype;
  6  begin
  7    open a;
  8    fetch a into a_var;
  9    while a%found loop
 10      dbms_output.put_line (a_var.last_name);
 11      fetch a into a_var;
 12    end loop;
 13  end;
 14  /
REAGAN
CARTER
HOOVER
TAFT
ANTHONY
ROOSEVELT

PL/SQL procedure successfully completed.

SQL>
```

This For Update Statement Will Not Lock Records Since the Wages Columns is Not in the Select

End listing

The Where Current Of Option

This has two important benefits:

1. Performance. Oracle always knows the current record. When the record is modified, Oracle can go directly to the record without having to locate the record in the table. If the option is missing, the Update and Delete statements will need a Where clause to locate the proper record. This will require some I/O. The Where Current Of option can dramatically increase performance of data modification procedures.

2. Code Simplification. The option eliminates the need to create a Where clause for the DML commands. This eliminates the need to create local variables, fetch values for the variables, and include them in the Where clause. The option will reduce the size of the procedure.

Update *tablename* set *column_name* = *value*
Where current of *cursor_name*;

An Update Statement Using the Where Current Of Option

```
SQL> declare
  2    cursor a is select last_name, first_name, wages from employee
  3                where fk_department = 'WEL'
  4                for update;
  5    a_var      a%rowtype;
  6  begin
  7    open a;
  8    fetch a into a_var;
  9    while a%found loop
 10      dbms_output.put_line (a_var.last_name||' '||to_char(a_var.wages));
 11      update employee set wages = wages * 1.03
 12        where current of a;
 13      fetch a into a_var;
 14    end loop;
 15    close a;
 16    open a;
 17    fetch a into a_var;
 18    while a%found loop
 19      dbms_output.put_line (a_var.last_name||' '||to_char(a_var.wages));
 20      fetch a into a_var;
 21    end loop;
 22    close a;
 23  end;
 24  /
REAGAN 14500
CARTER 14000
HOOVER 10000
TAFT 8500
ANTHONY 7000
ROOSEVELT
REAGAN 14905
CARTER 14390
HOOVER 10300
TAFT 8755
ANTHONY 7210
ROOSEVELT

PL/SQL procedure successfully completed.

SQL>
```

This clause tells Oracle exactly where the record to be updated

End Listing

Practice

18. Create a procedure that updates the Absences column in the Employee table. The value should be set to 0. Use the Where Current Of option.

FOR LOOPS

Numeric For loops

A syntax template for the structure follows:

For *counting_variable*
in *lower_range_number* .. *highest_range_number*
Loop
Statements;
End loop;

Using the numeric For loop

```
SQL> declare
  2    cursor a is select first_name, last_name from employee;
  3    emp_var       a%rowtype;
  4  begin
  5    open a;
  6    for cnt_var in 1..10
  7     loop
  8      fetch a into emp_var;
  9      dbms_output.put_line(to_char(cnt_var)||' '||emp_var.last_name);
 10     end loop;
 11    close a;
 12  end;
 13  /
1 COOLIDGE
2 JOHNSON
3 REAGAN
4 BUSH
5 JOHNSON
6 CLINTON
7 CARTER
8 FORD
9 NIXON
10 KENNEDY

PL/SQL procedure successfully completed.
SQL>
```

Numeric For Loop Header

End listing

Using the Numeric For Loop Using the Reverse Option

```
SQL> declare
  2    cursor a is select first_name, last_name from employee;
  3    emp_var       a%rowtype;
  4  begin
  5    open a;
  6    for cnt_var in reverse 3..10
  7     loop
  8      fetch a into emp_var;
  9      dbms_output.put_line(to_char(cnt_var)||' '||emp_var.last_name);
 10     end loop;
 11    close a;
 12  end;
 13  /
10 COOLIDGE
9 JOHNSON
8 REAGAN
7 BUSH
6 JOHNSON
5 CLINTON
4 CARTER
3 FORD

PL/SQL procedure successfully completed.

SQL>
```

Same list of last_name values as in Listing 14.21. Only the corresponding counter variable

End listing

Practice

19. Create a procedure that displays the five oldest employees. Use a numeric For loop in your procedure and number each record.

20. Modify the procedure you built in #19 to number the records in reverse order.

The Basic Cursor For Loop

The basic Cursor For loop eliminates the shortcomings of the Numeric For loop when the Numeric For loop is used with cursors. The Cursor For loop is similar to the Numeric For loop, but has four main differences:

1. The high and low range values in the header are changed to the name of the cursor. This in effect tells Oracle to use an implied %notfound cursor attribute to denote the cursor records have been processed.

2. The structure does not have a counting variable. Since the range values are not needed, a counting variable is not created or needed.

3. The cursor commands Open, Fetch, and Close are not needed. These commands are implicitly issued by the Loop structure.

4. The local variables used within the loop do not have to be defined. Oracle will create a PL/SQL record for the cursor's fetched attributes. These variables are qualified by the name of the Cursor For loop**.**

The Basic Cursor For loop

PL/SQL record

Cursor name

```
SQL> declare
  2    cursor a is select first_name, last_name from employee;
  3  begin
  4    for cnt_var in a
  5     loop
  6      dbms_output.put_line(to_char(a%rowcount)||' '||cnt_var.last_name);
  7     end loop;
  8  end;
  9  /
1 COOLIDGE
.
.
19 ROOSEVELT

PL/SQL procedure successfully completed.
```

End Listing

Defining the Cursor in the Cursor For Header

The Cursor For loop with the cursor defined within the loop

```
SQL> begin
  2     for cnt_var in (select first_name, last_name from employee)
  3      loop
  4        dbms_output.put_line(cnt_var.last_name);
  5      end loop;
  6  end;
  7  /
COOLIDGE
JOHNSON
.
.
ANTHONY
ROOSEVELT

PL/SQL procedure successfully completed.

SQL>
```

End Listing

Nested For loops

A nested cursor for loop

Outer loop variable used in the inner loop cursor

```
SQL> declare
  2    hi_tool_name      emp_tools.tool_name%type;
  3    hi_tool_cost      emp_tools.tool_cost%type;
  4  begin
  5    for outer_loop in (select payroll_number, last_name from employee
  6                       where fk_department = 'WEL')
  7      loop
  8        for inner_loop in (select tool_name, tool_cost from emp_tools
  9                           where fk_payroll_number = outer_loop.payroll_number)
 10        loop
 11        if (inner_loop.tool_cost > hi_tool_cost
 12            or hi_tool_cost is null) then
 13           hi_tool_name := inner_loop.tool_name;
 14           hi_tool_cost := inner_loop.tool_cost;
 15        end if;
 16        end loop;
 17      dbms_output.put_line (outer_loop.last_name||' '||hi_tool_name);
 18      hi_tool_name := null;
 19      hi_tool_cost := null;
 20    end loop;
 21  end;
 22  /
REAGAN Tool Chest
CARTER
HOOVER TIN SNIPS
TAFT FOUNTAIN PEN
ANTHONY BRIEF CASE
ROOSEVELT CALCULATOR

PL/SQL procedure successfully completed.

SQL>
```

End listing

Practice

21. Create a procedure to list the employees in the “INT” and “POL” departments. Use a Cursor For loop in this procedure. The procedure should define a cursor.

22. Modify the procedure in #21. Define the select statement used in the Cursor For loop within the Cursor For structure.

23. Create a procedure that determines the date of the highest priced tool and the date of the highest priced eyeglass purchase for each employee. Use nested Cursor For loops