PL/SQL DAY3

PL/SQL Block Types

Anonymous

```
DECLARE
BEGIN
        -statements
EXCEPTION
END;
```

Procedure

```
PROCEDURE <name>
IS
BEGIN
        -statements
EXCEPTION
END;
```

Function

```
FUNCTION <name>
RETURN <datatype>
IS
BEGIN
        -statements
EXCEPTION
END;
```

PL/SQL Variable Types

- Scalar (char, varchar2, number, date, etc)
- Composite (%rowtype)
- Reference (pointers)
- LOB (large objects)

Note: Non PL/SQL variables include bind variables,
host (“global”) variables, and parameters.

Variable Naming Conventions

- Two variables can have the same name if they are in different blocks (bad idea)
- The variable name should not be the same as any table column names used in the block.

PL/SQL is strongly typed

- All variables must be declared before their use.
- The assignment statement

  : =

  is not the same as the equality operator

  =
- All statements end with a ;

Manipulating Character Strings with PL/SQL

- To concatenate two strings in PL/SQL, you use the double bar (||) operator:
  - *new_string := string1 || string2;*
- To remove blank leading spaces use the LTRIM function:
  - *string := LTRIM(string_variable_name);*
- To remove blank trailing spaces use the RTRIM function:
  - *string := RTRIM(string_variable_name);*
- To find the number of characters in a character string use the LENGTH function:
  - *string_length := LENGTH(string_variable_name);*

Manipulating Character Strings with PL/SQL

- To change case, use UPPER, LOWER, INITCAP
- INSTR function searches a string for a specific substring:
  - *start_position := INSTR(original_string, substring);*
- SUBSTR function extracts a specific number of characters from a character string, starting at a given point:
  - *extracted_string := SUBSTR(string_variable, starting_point, number_of_characters);*

COMMON PL/SQL STRING FUNCTIONS

- CHR(asciivalue)
- ASCII(string)
- LOWER(string)
- SUBSTR(string,start,substrlength)
- LTRIM(string)
- RTRIM(string)
- LPAD(string_to_be_padded, spaces_to_pad, |string_to_pad_with|)
- RPAD(string_to_be_padded, spaces_to_pad, |string_to_pad_with|)
- REPLACE(string, searchstring, replacestring)
- UPPER(string)
- INITCAP(string)
- LENGTH(string)

Complex Conditions

- Created with logical operators AND, OR and NOT
- AND is evaluated before OR
- Use () to set precedence

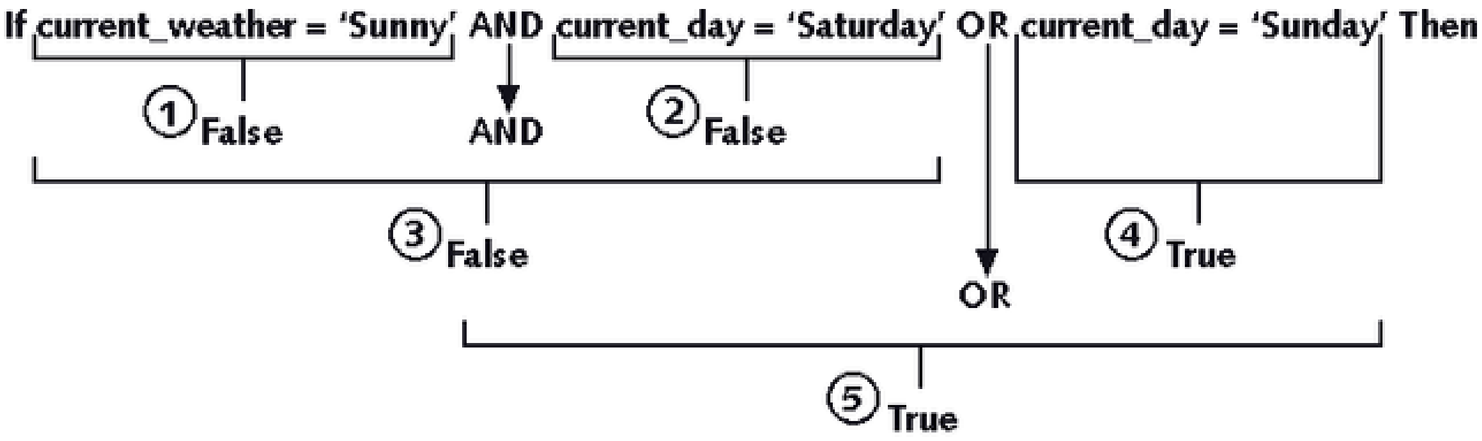

If current_weather = 'Sunny' AND current_day = 'Saturday' OR current_day = 'Sunday' Then
1 False
AND
2 False
3 False
OR
4 True
5 True

Figure 4-19 Evaluating AND and OR in an expression

Reference data types

- In many cases, a PL/SQL variable will be used to manipulate data stored in a existing table. In this case, it is essential that the variable have the same type (compatible is also ok in some situation) as the relation column.
- Directly reference a specific database field or record and assume the data type of the associated field or record
  - **%TYPE: same data type as a database field**
  - **%ROWTYPE: same data type as a database record**

%TYPE vs %ROWTYPE - What's the difference?

- **Description:** Both %TYPE and %ROWTYPE are used to define variables in PL/SQL as it is defined within the database.
- If the data type or precision of a column changes, the program automatically picks up the new definition from the database.

  The %TYPE and %ROWTYPE constructs provide data independence, reduce maintenance costs, and allows programs to adapt as the database changes.

What are the advantages of using these over data types?

- % TYPE provides the data type of a variable or a database column to that variable.
- % ROWTYPE provides the record type that represents a entire row of a table or view or columns selected in the cursor.
- The advantages are :

✓ **Need not know about variable's data type**

✓ **If the database definition of a column in a table changes, the data type of a variable changes accordingly.**

What are % TYPE and % ROWTYPE ?

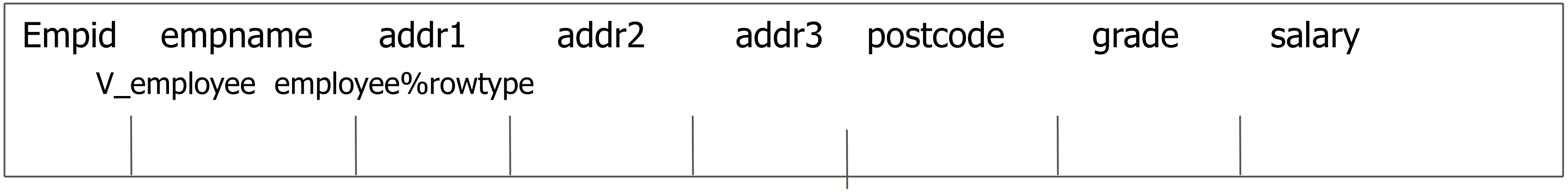
- %TYPE can be used with the column name preceded with table name to decide the data type and length of the variable at runtime. In this case there is no dependency on changes made to the data structure.

  %ROWTYPE can be used to declare the variable having the same no. of variables inside it (ROWTYPE) as no. of columns there in the table. In this case columns selected with SELECT statement **must match** with variables inside the rowtype variable. If not then individually refer these variables inside the ROWTYPE variables

```
-- %TYPE is used to declare a field with the same type as
-- that of a specified table's column:

DECLARE v_EmpName
emp.ename%TYPE;
BEGIN
SELECT ename INTO v_EmpName FROM emp WHERE ROWNUM = 1;
DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName);
END;
/

-- %ROWTYPE is used to declare a record with the same
types as Found in the specified database table, view or
cursor:

DECLARE v_emp emp%ROWTYPE;
BEGIN v_emp.empno := 10;
v_emp.ename := 'XXXXXXX';
END;
/
```

Using SQL in procedures

- Select values into PL/SQL variables
  - using INTO
- **Record.element** notation will address components of tuples *(dot notation)*
- **%rowtype** allows full rows to be selected into one variable



Example (Anonymous Block of Code)

```
Declare
v_employee    employee%rowtype;
Begin
select  *
into v_employee
from employee
  where empid = 65284;

update employee
set salary = v_employee.salary + 1000
   where empid = v_employee.empid;
End;
```

Selects entire row of data into 1 variable called v_employee

Is updating the value of salary based on selected element of a variable

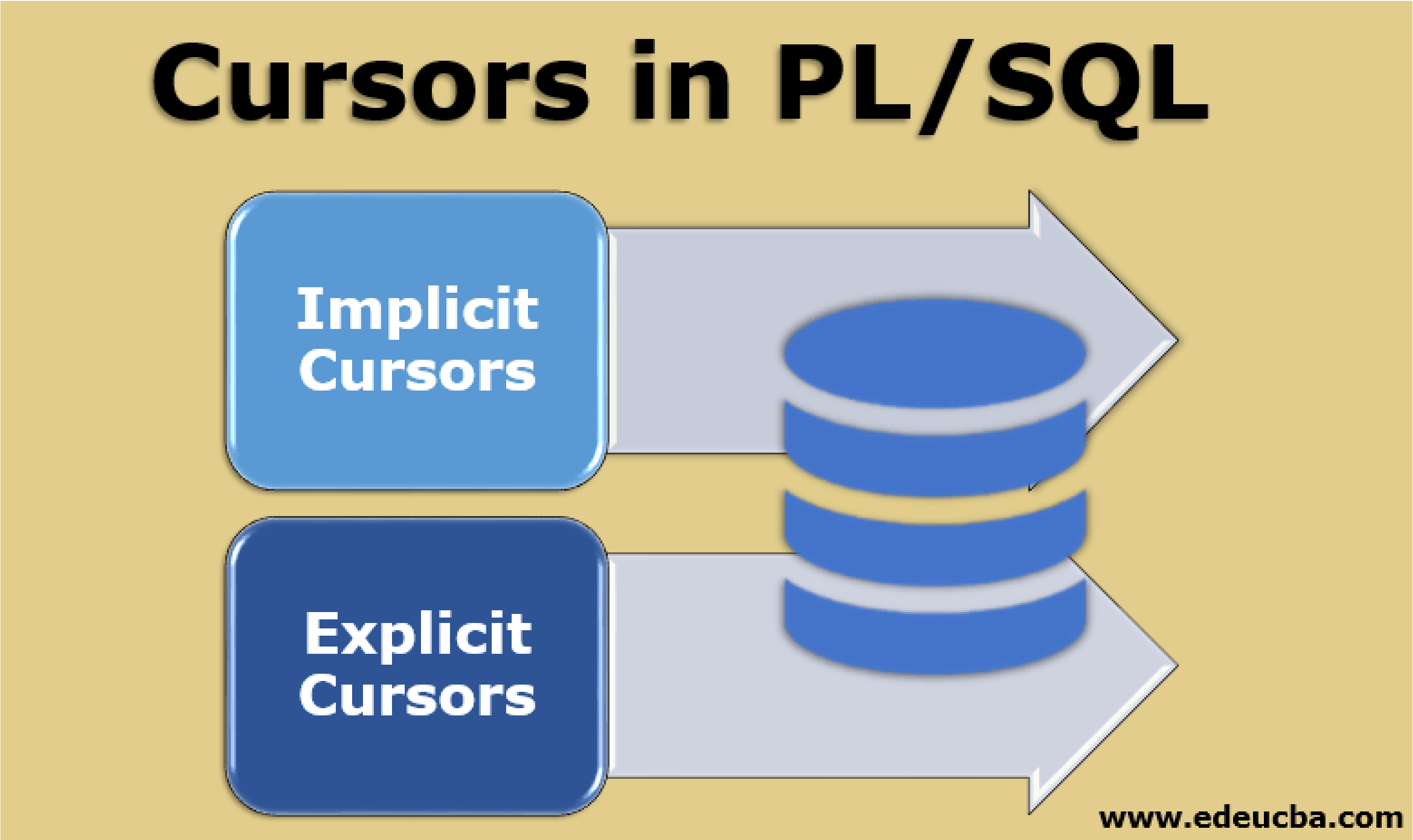
%ROWTYPE

```
Set serveroutput on
DECLARE
  v_student students%rowtype;
BEGIN
  select * into v_student
     from students
     where sid='123456';
  DBMS_OUTPUT.PUT_LINE (v_student.lname);
  DBMS_OUTPUT.PUT_LINE (v_student.major);
  DBMS_OUTPUT.PUT_LINE (v_student.gpa);
END;
/
```

SELECT INTO

```
SET SERVEROUTPUT ON
DECLARE
  v_max_gpa number(3,2);
  v_numstudents number(4);
  v_lname students.lname%type;
  v_major students.major%type;
BEGIN
  select max(gpa) into v_max_gpa
    from students;
  DBMS_OUTPUT.PUT_LINE ('The highest GPA is '||v_max_gpa);
  select count(sid) into v_numstudents
    from students
    where gpa = v_max_gpa;
  IF v_numstudents > 1 then
    DBMS_OUTPUT.PUT_LINE ('There are '||v_numstudents||' with that GPA');
  ELSE
    select lname, major into v_lname, v_major
      from students
      where gpa=v_max_gpa;
    DBMS_OUTPUT.PUT_LINE ('The student name is '||v_lname);
    DBMS_OUTPUT.PUT_LINE ('The student major is '||v_major);
  END IF;
END;
/
```

CURSORS


Cursors in PL/SQL
Implicit Cursors
Explicit Cursors
www.edeucba.com

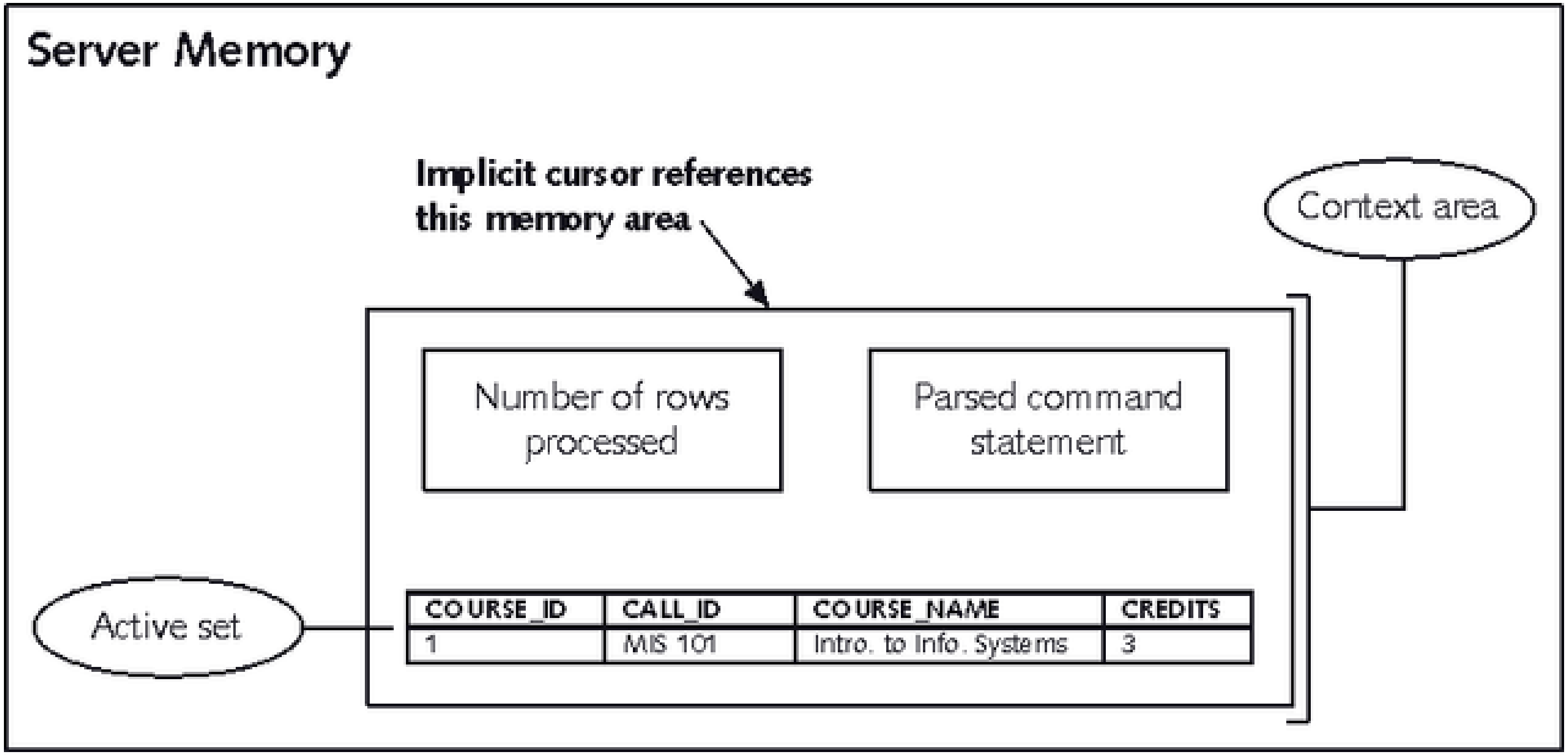
CURSOR

- PL/SQL creates an implicit cursor (**a private work area for that statement**) when an SQL statement is executed from within the program block.
- This work area stores the statement and the results returned by execution of that statement.
- If a cursor is not declared, it is created automatically and is known as an **implicit cursor**. An implicit cursor is used when the embedded SQL statement returns no more than one row.
- If used in such examples, a **TOO_MANY_ROWS** exception. On the other hand, if the SQL statement returns more than one row, an explicit cursor is needed.

Cursors in SQL

- Enables users to loop around a selection of data.
- Stores data selected from a query in a temp area for use when opened.
- Use complex actions which would not be feasible in standard SQL selection queries
- Pointer to a memory location that the DBMS uses to process a SQL query
- Use to retrieve and manipulate database data

Implicit Cursor



COURSE_ID	CALL_ID	COURSE_NAME	CREDITS
1	MIS 101	Intro. to Info. Systems	3

Figure 4-26 Implicit cursor

Using an Implicit Cursor

- Executing a SELECT query creates an implicit cursor
- To retrieve it into a variable use INTO:
  - SELECT *field1*, *field2*, ...

    INTO *variable1*, *variable2*, ...

    FROM *table1*, *table2*, ...

    WHERE *join_ conditions*

    AND

    search_condition_to_retrieve_1_record;
- Can only be used with queries that **return exactly one record**

Explicit Cursor

- Use for queries that **return multiple records** or no records
- **Must be explicitly declared and used**
- Four actions can be performed on an explicit cursor:

1. Declare a cursor with a name
2. Open the cursor
3. Fetch row(s) from the cursor
4. Close the cursor

Implicit cursors

- Whenever Oracle executes an SQL statement such as SELECT INTO, INSERT, UPDATE, and DELETE, it automatically creates an implicit cursor.

- Oracle internally manages the whole execution cycle of implicit cursors and reveals only the cursor's information and statuses such as SQL%ROWCOUNT, SQL%ISOPEN, SQL%FOUND, and SQL%NOTFOUND.

- The implicit cursor is not elegant when the query returns zero or multiple rows which cause NO_DATA_FOUND or TOO_MANY_ROWS exception respectively.

Using an Explicit Cursor

- Declare the cursor
  - *CURSOR cursor_name **IS** select_query;*
- Open the cursor
  - *OPEN cursor_name;*
- Fetch the data rows

  LOOP

  FETCH cursor_name INTO variable_name(s);

  EXIT WHEN cursor_name%NOTFOUND;
- Close the cursor
  - *CLOSE cursor_name;*

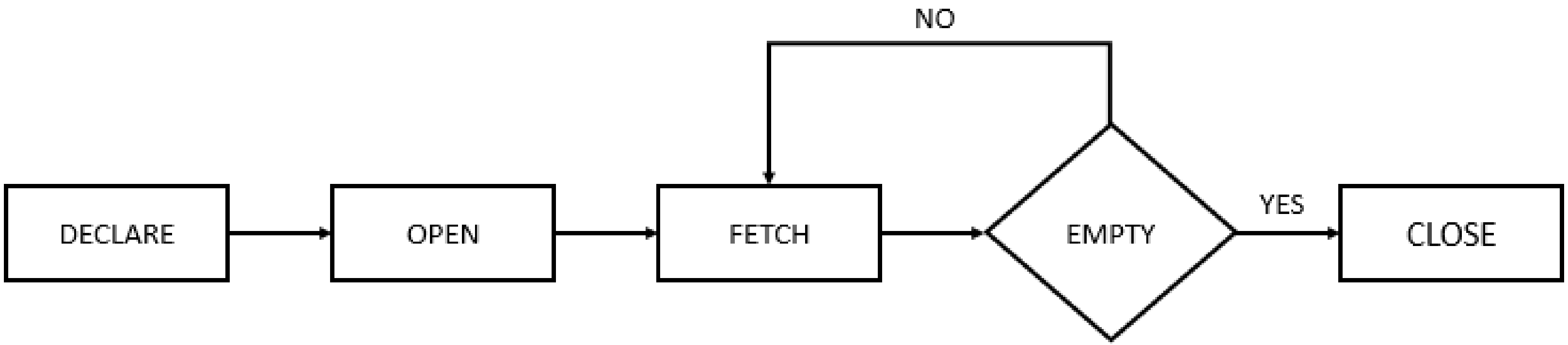
Explicit cursors

- An explicit cursor is an SELECT statement declared explicitly in the declaration section of the current block or a package specification.

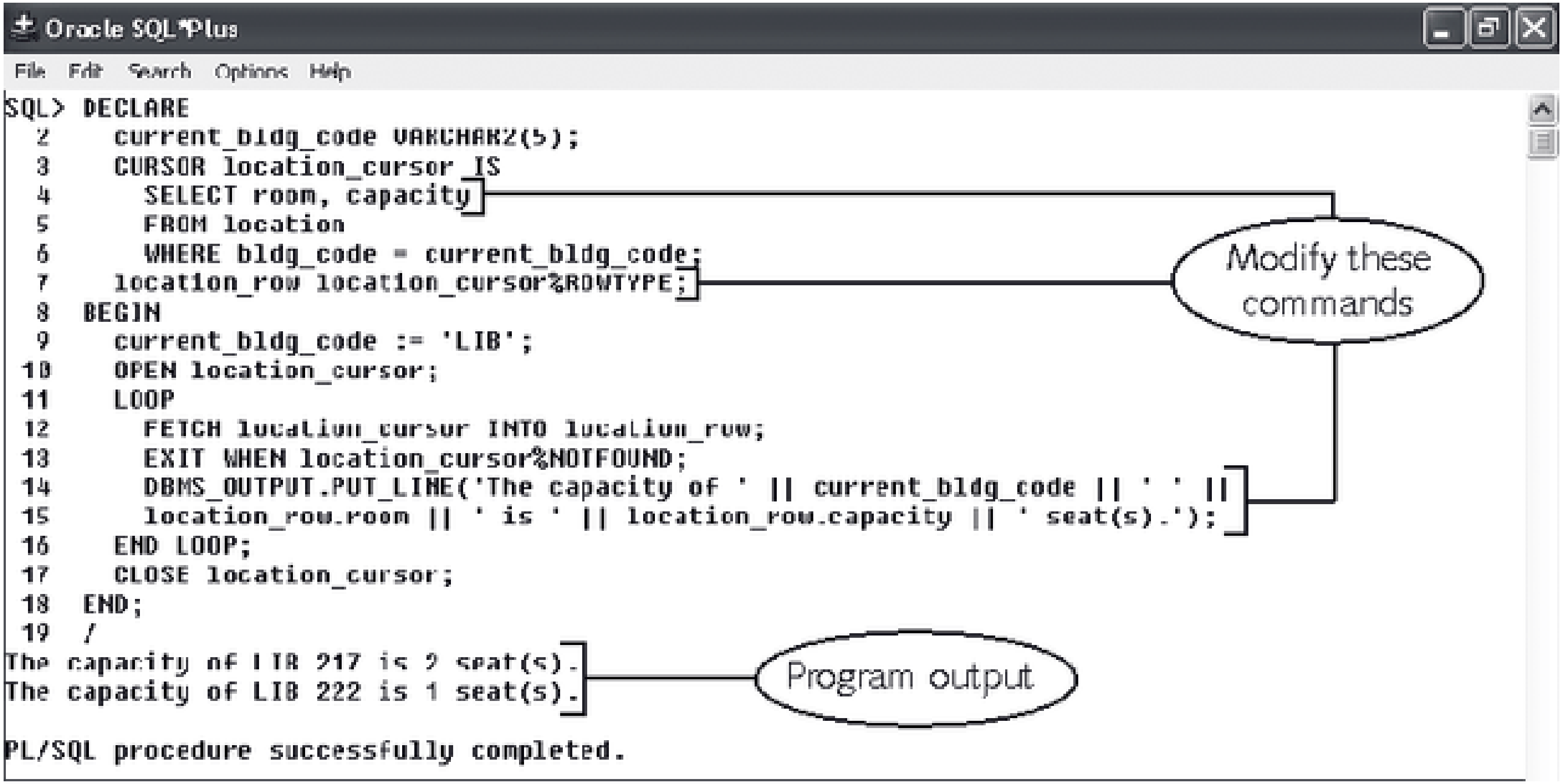
- For an explicit cursor, you have control over its execution cycle from OPEN, FETCH, and CLOSE.

- Oracle defines an execution cycle that executes an SQL statement and associates a cursor with it.

EXPLICIT CURSOR



Explicit Cursor with %ROWTYPE



```
SQL> DECLARE
       current_bldg_code VARCHAR2(5);
       CURSOR location_cursor IS
         SELECT room, capacity
         FROM location
         WHERE bldg_code = current_bldg_code;
       location_row location_cursor%ROWTYPE;
     BEGIN
       current_bldg_code := 'LIB';
       OPEN location_cursor;
       LOOP
         FETCH location_cursor INTO location_row;
         EXIT WHEN location_cursor%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('The capacity of ' || current_bldg_code || ' ' ||
         location_row.room || ' is ' || location_row.capacity || ' seat(s).');
       END LOOP;
       CLOSE location_cursor;
     END;
     /
The capacity of LIB 217 is 2 seat(s).
The capacity of LIB 222 is 1 seat(s).

PL/SQL procedure successfully completed.
```

Modify these commands

Program output

Figure 4-31 Processing an explicit cursor using a %ROWTYPE variable

Syntax for Cursors

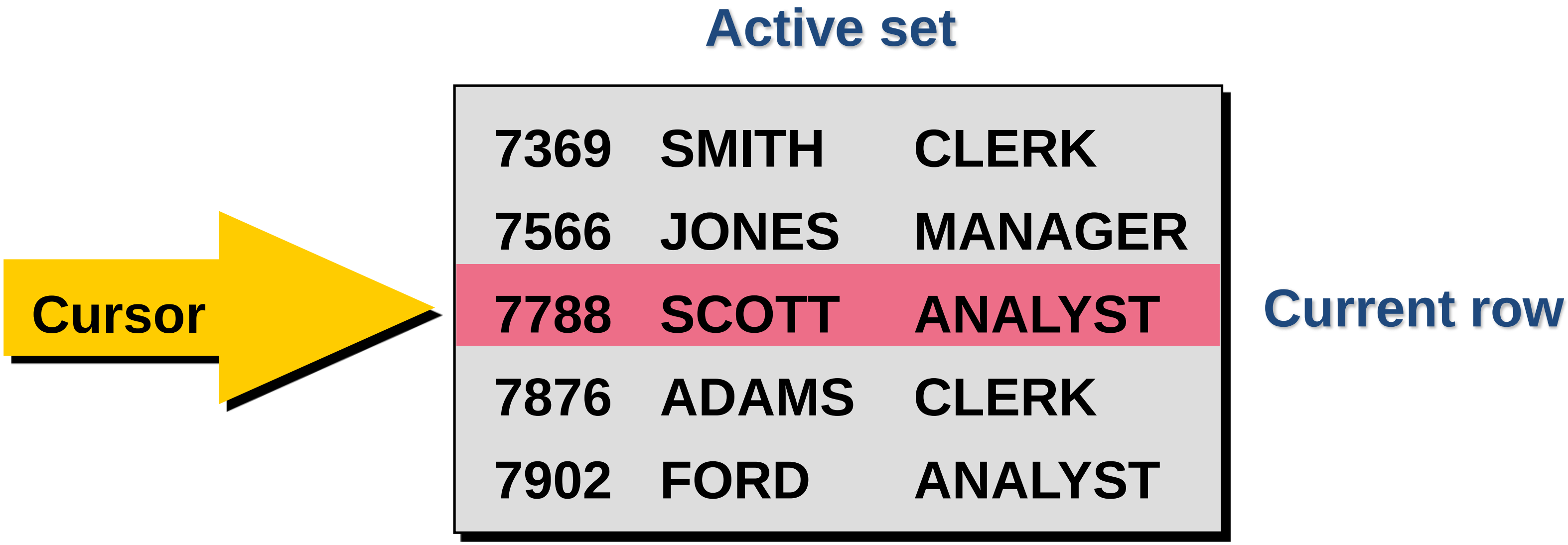
- Declared as a variable in the same way as standard variables
- Identified as cursor type
- SQL included
- E.g.

```
Cursor cur_emp is
    Select emp_id, surname name, grade, salary
         From employee
           Where regrade is true;
```

Cursors

- A cursor is a temp store of data.
- The data is populated when the cursor is opened.
- Once opened the data must be moved from the temp area to a local variable to be used by the program. These variables must be populated in the same order that the data is held in the cursor.
- The data is looped round till an exit clause is reached.

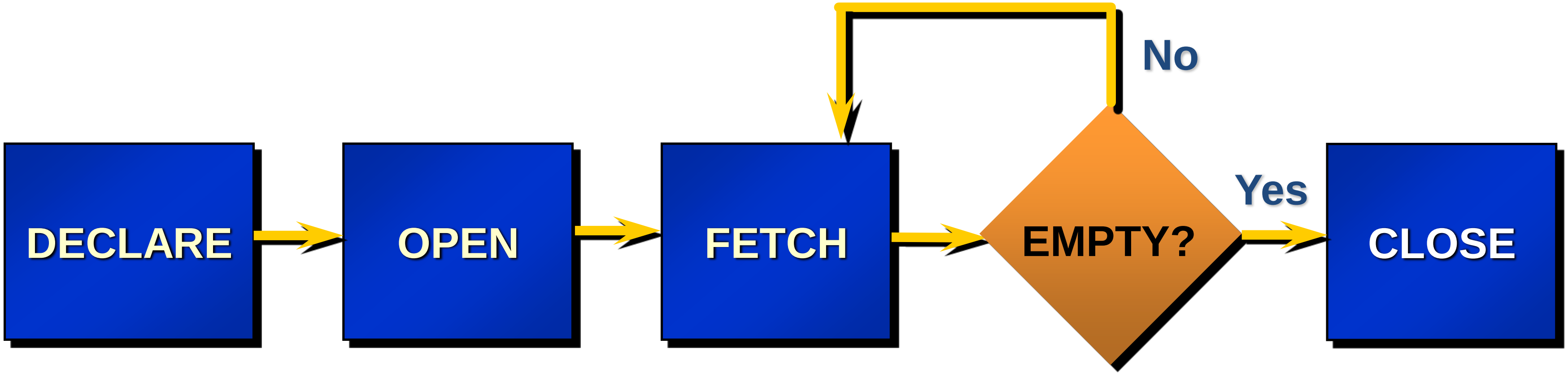
Cursor Functions



Active set

7369	SMITH	CLERK
7566	JONES	MANAGER
7788	SCOTT	ANALYST
7876	ADAMS	CLERK
7902	FORD	ANALYST

Controlling Cursor



DECLARE

OPEN

FETCH

EMPTY?

No

Yes

CLOSE

- Create a named SQL area

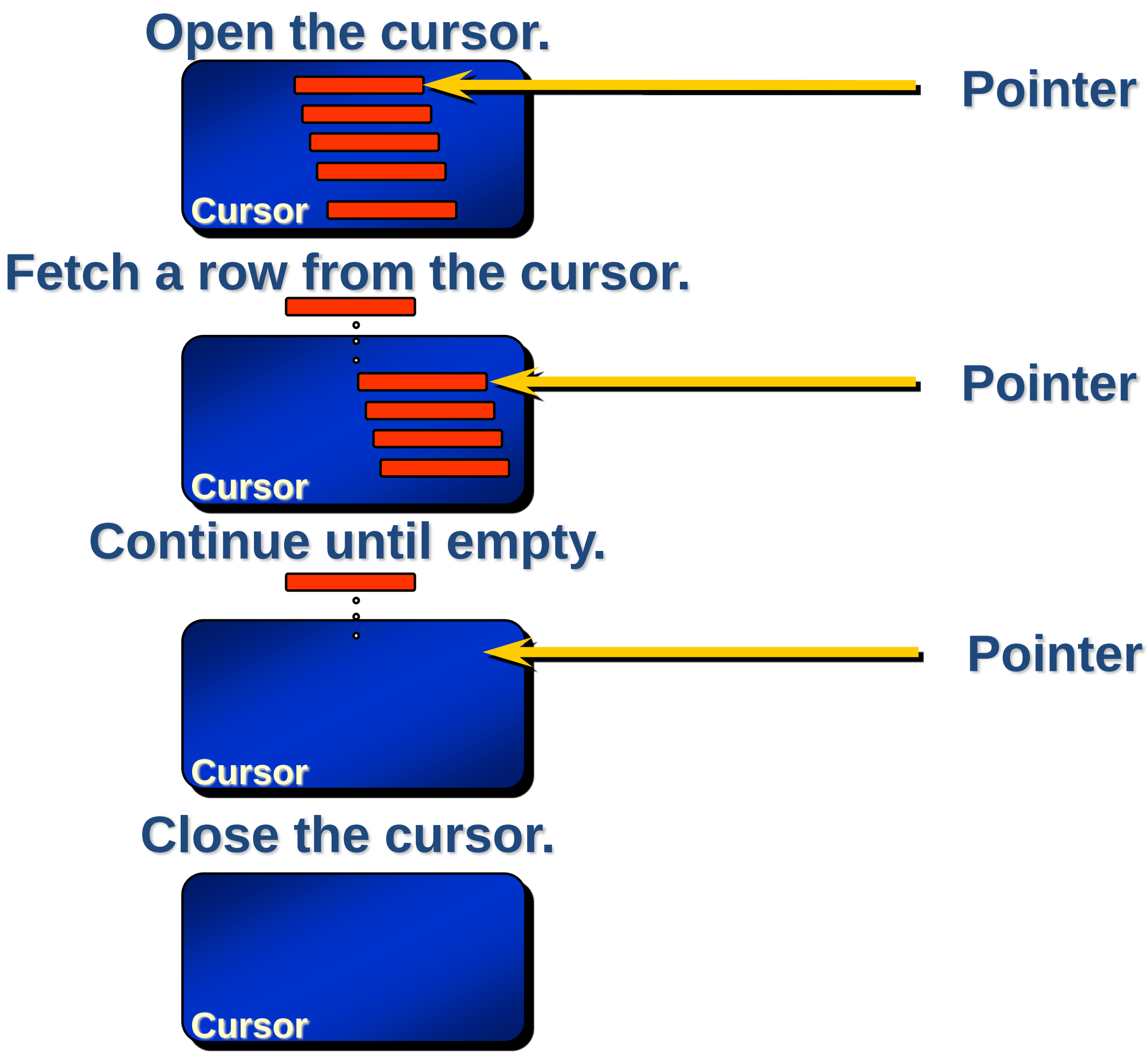
- Identify the active set

- Load the current row into variables

- Test for existing rows
- Return to FETCH if rows found

- Release the active set

Controlling Cursor…



Open the cursor.

Fetch a row from the cursor.

Continue until empty.

Close the cursor.

Cursor Attributes

Obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far

The %ISOPEN Attribute

- Fetch rows only when the cursor is open.
- Use the %ISOPEN cursor attribute before performing a fetch to test whether the cursor is open.
- Example

```
IF NOT cur_sample%ISOPEN THEN
    OPEN cur_sample;
END IF;
LOOP
  FETCH cur_sample...
```

Cursors and Records

- Process the rows of the active set conveniently by fetching values into a PL/SQL RECORD.
- Example

```
DECLARE
  CURSOR emp_cursor IS
    SELECT  empno, ename
    FROM    emp;
  emp_record   emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
  ...
```

Cursor FOR Loops

- Syntax

```
FOR record_name IN cursor_name LOOP
   statement1;
   statement2;
   . . .
END LOOP;
```

- The cursor FOR loop is a shortcut to process cursors.
- Implicitly opens, fetches, and closes cursor.
- The record is implicitly declared.

Cursor FOR Loops: An Example

- Retrieve employees one by one until no more are left.
- Example

```
DECLARE
  CURSOR emp_cursor IS
    SELECT ename, deptno
    FROM   emp;
BEGIN
  FOR emp_record IN emp_cursor LOOP
         -- implicit open and implicit fetch occur
    IF emp_record.deptno = 30 THEN
      ...
  END LOOP; -- implicit close occurs
END;
```

```
SET SERVEROUTPUT ON
SET VERIFY OFF

DECLARE
    hdate DATE;
CURSOR Temp_Cursor is
SELECT hdate from employee;
BEGIN
OPEN Temp_Cursor;
LOOP
FETCH Temp_Cursor INTO hdate;
EXIT WHEN Temp_Cursor%NOTFOUND;
    IF hdate > '01-JAN-15' THEN
        DBMS_OUTPUT.PUT_LINE ('The employee is JUNIOR');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('The employee is SENIOR');
    END IF;
END LOOP;
COMMIT;
    IF Temp_Cursor%ISOPEN THEN CLOSE Temp_Cursor;
    END IF;
EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('Error in Seniority Dtermination');
    IF Temp_Cursor%ISOPEN THEN CLOSE Temp_Cursor;
    END IF;
ROLLBACK;
```

If the salary of the employe is <20000, update the salary by 5000

```
DECLARE
c_id customers.id%type := 1;
c_sal customers.salary%type;

BEGIN
SELECT salary
INTO c_sal
FROM customers
WHERE id = c_id;

IF (c_sal <= 2000) THEN
UPDATE customers
SET salary = salary + 5000
WHERE id = c_id;
dbms_output.put_line ('Salary updated');
END IF;
END;
/
```

STRINGS

```
DECLARE
greetings varchar2(11) := 'hello world';
BEGIN
dbms_output.put_line(UPPER(greetings));
dbms_output.put_line(LOWER(greetings));
dbms_output.put_line(INITCAP(greetings));
        /* retrieve the first character in the string */
dbms_output.put_line ( SUBSTR (greetings, 1, 1));
        /* retrieve the last character in the string */
dbms_output.put_line ( SUBSTR (greetings, -1, 1));
        /* retrieve five characters, starting from the seventh position. */
dbms_output.put_line ( SUBSTR (greetings, 7, 5));
        /* retrieve the remainder of the string, starting from the second position. */
dbms_output.put_line ( SUBSTR (greetings, 2));
        /* find the location of the first "e" */
dbms_output.put_line ( INSTR (greetings, 'e'));
END;
/
```

STRINGS

```
DECLARE
greetings varchar2(30) := '......Hello World.....';
BEGIN
dbms_output.put_line(RTRIM(greetings,'.'));
dbms_output.put_line(LTRIM(greetings, '.'));
dbms_output.put_line(TRIM( '.' from greetings));
END;
/
```

CURSORS

EXAMPLES

Explicit cursors

```
SQL> edit cursor1
```

```
DECLARE
  v_name      donor.name%TYPE;
  v_yrgoal    donor.yrgoal%TYPE;
  v_state     donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

```
SQL> @ cursor1

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME               YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Carl Hersey               RI
Susan Ash             100 MA
Nancy Taylor           50 MA
Robert Brooks          50 MA
```

The table donor_part was empty before cursor1 was executed. After running the anonymous block, there are now six records in the table. They correspond to the six records that were in the donor table.

Explicit cursor

```
SQL> SELECT * FROM donor;

IDNO  NAME            STADR           CITY       ST ZIP   DATEFST       YRGOAL CONTACT
----- --------------- --------------- ---------- -- ----- --------- --------- ------------
11111 Stephen Daniels 123 Elm St      Seekonk    MA 02345 03-JUL-98       500 John Smith
12121 Jennifer Ames   24 Benefit St   Providence RI 02045 24-MAY-97       400 Susan Jones
22222 Carl Hersey     24 Benefit St   Providence RI 02045 03-JAN-98           Susan Jones
23456 Susan Ash       21 Main St      Fall River MA 02720 04-MAR-92       100 Amy Costa
33333 Nancy Taylor    26 Oak St       Fall River MA 02720 04-MAR-92        50 John Adams
34567 Robert Brooks   36 Pine St      Fall River MA 02720 04-APR-98        50 Amy Costa

6 rows selected.
```

```
SQL> SELECT * FROM donor_part;

NAME                YRGOAL ST
--------------- ---------- --
Stephen Daniels        500 MA
Jennifer Ames          400 RI
Carl Hersey                RI
Susan Ash              100 MA
Nancy Taylor            50 MA
Robert Brooks           50 MA
```

Initial FETCH got the first record from the table and put the data into the variables. The INSERT inside the loop put the data from the variables into the new table.

The FETCH after the INSERT (the last command in the loop) got the second record from the table and put the data in the variables. The INSERT inside the loop put the data from the variables into the new table.

The FETCH after the INSERT (the last command in the loop) got the third record from the table and put the data in the variables. The INSERT inside the loop put the data from the variables into the new table.

Explicit cursor

These are the variable names declared to receive the data from the table.

```
DECLARE
  v_name      donor.name%TYPE;
  v_yrgoal    donor.yrgoal%TYPE;
  v_state     donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
  OPEN donor_cursor;
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
  WHILE donor_cursor%FOUND LOOP
    INSERT INTO donor_part
      VALUES(v_name, v_yrgoal, v_state);
    FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
  END LOOP;
  CLOSE donor_cursor;
END;
/
```

The cursor is created with a select statement. The select statement will be processed by the cursor providing the rows to be processed in the block.

The OPEN statement opens or activates the cursor - this means the select is executed to fill the cursor with rows.

The FETCH statement gets the first record in the cursor and moves the data to the defined variables. This is the initial FETCH.

The FETCH which is the last statement in the loop will get all other records.

When the loop is complete the cursor is closed.

The WHILE loop will continue to execute while there is still data in the cursor. This is tested with the %FOUND. Note that when the loop is entered, the FETCH of the initial record has already been done. The INSERT statement will insert the data from that record into the table named donor_part. Then it will execute the FETCH which is the last statement in the loop to get the next record. As long as a record is found, the INSERT will be done followed by another FETCH. When the FETCH is unsuccessful, the WHILE will terminate because of donor_cursor%FOUND.

INSERT puts a record into donor_part containing the information that the FETCH put into the variables.

Explicit cursor

```
SQL> edit cursor2
```

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  IF v_yrgoal > 50 THEN
    INSERT INTO donor_part
      VALUES(v_name, v_yrgoal, v_state);
  END IF;
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

```
SQL> @ cursor2

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME                YRGOAL ST
--------------- ---------- --
Stephen Daniels        500 MA
Jennifer Ames          400 RI
Susan Ash              100 MA
```

The IF statement only INSERTs records where the year goal is greater than 50. Only the three records shown met the criteria.

Explicit cursor

```
SQL> edit cursor2a
```

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor
      WHERE yrgoal> 50;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

```
SQL> @ cursor2a

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME                YRGOAL ST
--------------- ---------- --
Stephen Daniels        500 MA
Jennifer Ames          400 RI
Susan Ash              100 MA
```

Instead of selecting the record after they have been FETCHed with the IF, you can SELECT the records that meet the condition in the CURSOR with the WHERE clause.

Explicit cursor

```
SQL> edit cursor2b
```

```
DECLARE
  v_name      donor.name%TYPE;
  v_yrgoal    donor.yrgoal%TYPE;
  v_state     donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor
      WHERE yrgoal> 50;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
  EXIT WHEN donor_cursor%NOTFOUND;
END LOOP;
CLOSE donor_cursor;
END;
/
```

```
SQL> @ cursor2b

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME                YRGOAL ST
--------------- ---------- --
Stephen Daniels        500 MA
Jennifer Ames          400 RI
Susan Ash              100 MA
```

This code changes to a simple LOOP with an exit based on %NOTFOUND instead of %FOUND.

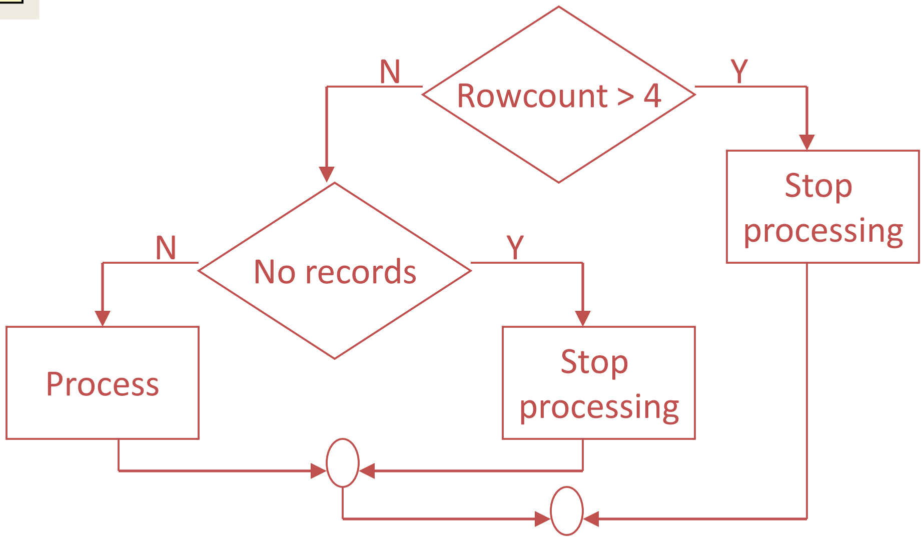
Explicit cursor

```
DECLARE
  v_name      donor.name%TYPE;
  v_yrgoal    donor.yrgoal%TYPE;
  v_state     donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%ROWCOUNT < 5 AND donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

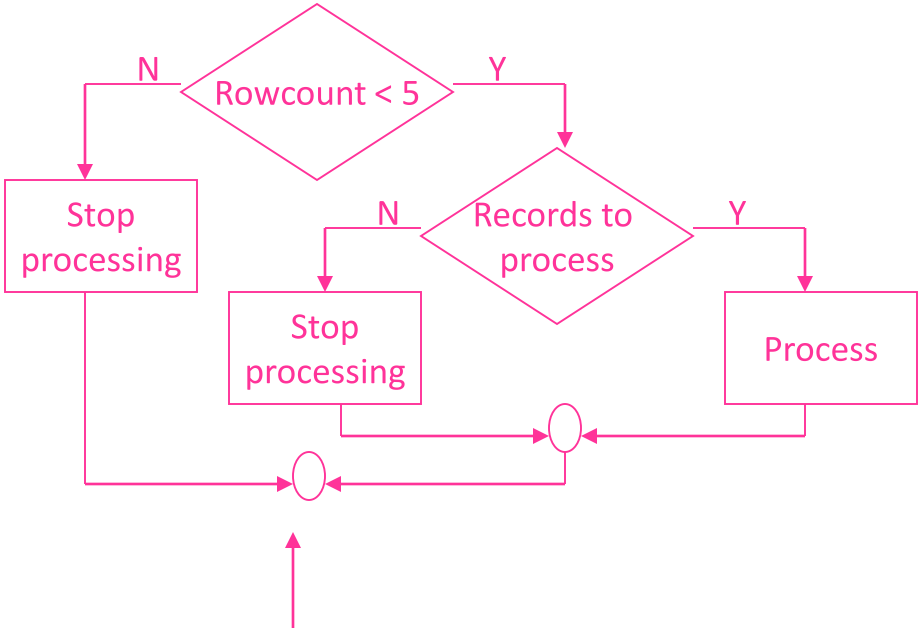
```
SQL> @ cursor3

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME               YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Carl Hersey               RI
Susan Ash             100 MA
```

The while loop will terminate after 4 records have been processed or when no more records are in the cursor. %ROWCOUNT is used to determine when 4 records have been processed.

Logic



OR LOGIC: Logic for if row count is > 4 OR there are no more records stop processing. Otherwise process the records.



AND LOGIC: Logic for if row count is < 5 and there are records to process, process. If either condition is false, do not process.

```
WHILE donor_cursor%ROWCOUNT < 5 AND donor_cursor%FOUND LOOP
```

A variation using WHILE Loop and %FOUND

```
DECLARE
  CURSOR low_pay IS SELECT surname,salary FROM
  Personnel where salary < 30000;
v_surname       personnel.surname%TYPE;
v_salary        personnel.salary%TYPE;
BEGIN
  OPEN low_pay;
   FETCH low_pay INTO v_surname, v_salary;
     WHILE low_pay%FOUND LOOP
       DBMS_OUTPUT.PUT_LINE(v_surname ||' '||
  v_salary);
       FETCH low_pay INTO v_surname, v_salary;
     END LOOP;
  CLOSE low_pay;
END;
```

Note 2 FETCH commands

Parameters in Cursors

Block2

```
DECLARE
  CURSOR c_salary (p_min number,p_max number)
   IS SELECT surname,salary FROM Personnel
      where salary between p_min and p_max;
v_surname                  Personnel.surname%TYPE;
v_salary                   Personnel.salary%TYPE;
BEGIN
  OPEN c_salary(&p_min, &p_max);
   LOOP
      FETCH c_salary INTO v_surname, v_salary;
      EXIT WHEN c_salary%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_surname||' '||v_salary);
   END LOOP;
  CLOSE c_salary;
END;
```

These would be in quotes for VARCHAR2 variables

FOR LOOP requires no CURSOR OPEN, FETCH, CLOSE

```
DECLARE
   CURSOR c_salary IS SELECT surname,salary
        FROM Personnel
        where salary < 30000;
BEGIN
   FOR counter in c_salary LOOP
        DBMS_OUTPUT.PUT_LINE(counter.surname
               ||' '||counter.salary);
   END LOOP
END;
```

Useful when updating or deleting each row fetched in a cursor otherwise all would be updated at once

SELECT FOR UPDATE Cursors

```
DECLARE
  CURSOR c_salary IS SELECT surname,salary FROM Personnel
      FOR UPDATE;
v_surname         personnel.surname%TYPE;
v_salary          personnel.salary%TYPE;
BEGIN
  OPEN c_salary;
   LOOP
      FETCH c_salary INTO v_surname, v_salary;
      EXIT WHEN c_salary%NOTFOUND;
      UPDATE Personnel SET BONUS=v_salary*0.05 WHERE
            CURRENT of c_salary;
   END LOOP;
  CLOSE c_salary;
END;
```