DATABASE SYSTEMS

PLSQL (CURSOR)

By Sana Faiz
Sana.faiz.muet83@gmail.com

INTRODUCTION

- Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

- A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor.
- A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

- You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors.

1. **Implicit cursors**
2. **Explicit cursors**

IMPLICIT CURSORS

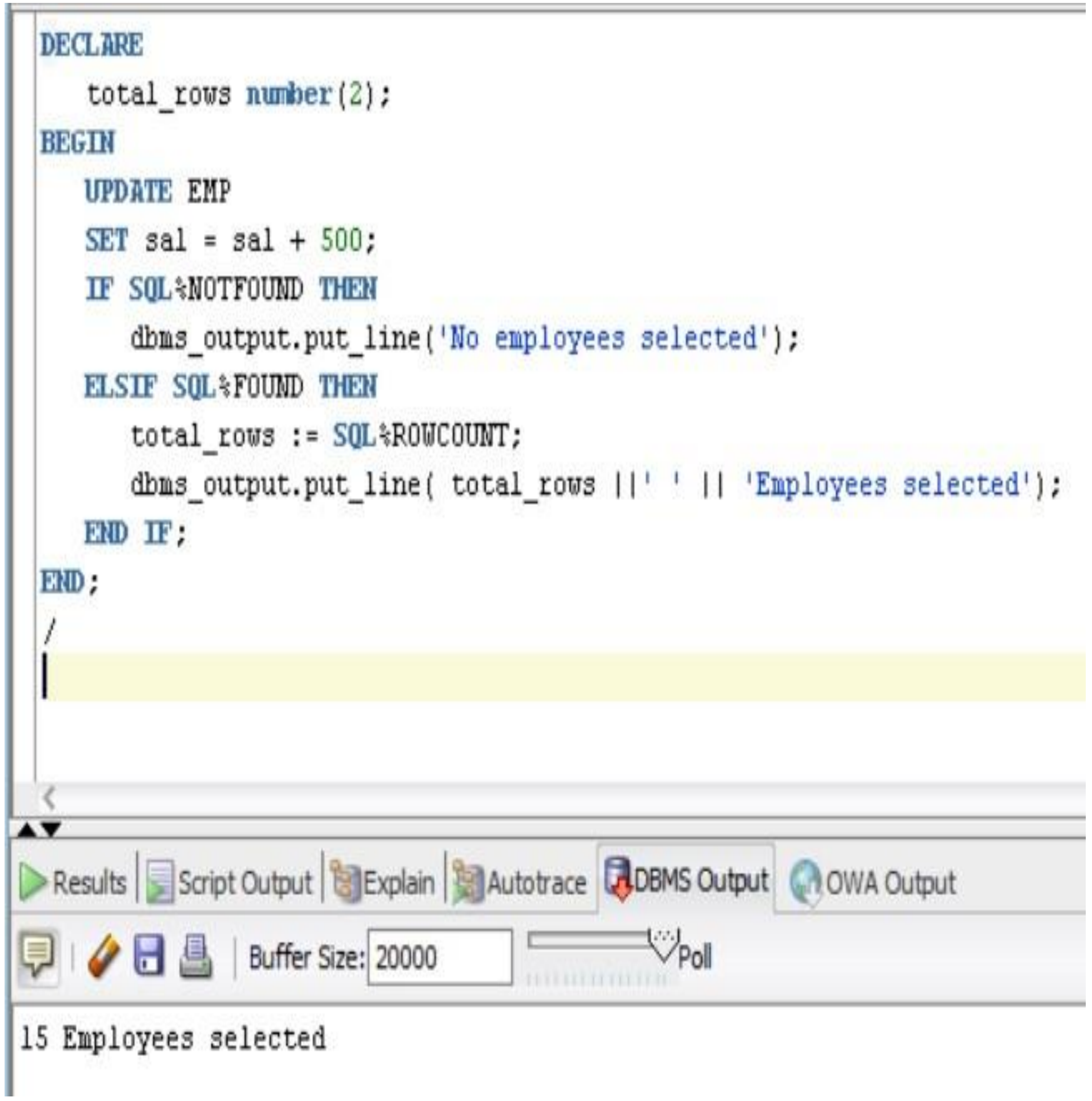
- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.
- Programmers cannot control the implicit cursors and the information in it.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.
- In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.
- There is no need for users to OPEN , FETCH or CLOSE an Implicit cursor.

IMPLICIT CURSOR ATTRIBUTES

S.NO	ATTRIBUTE & DESCRIPTION
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as SQL%attribute_name

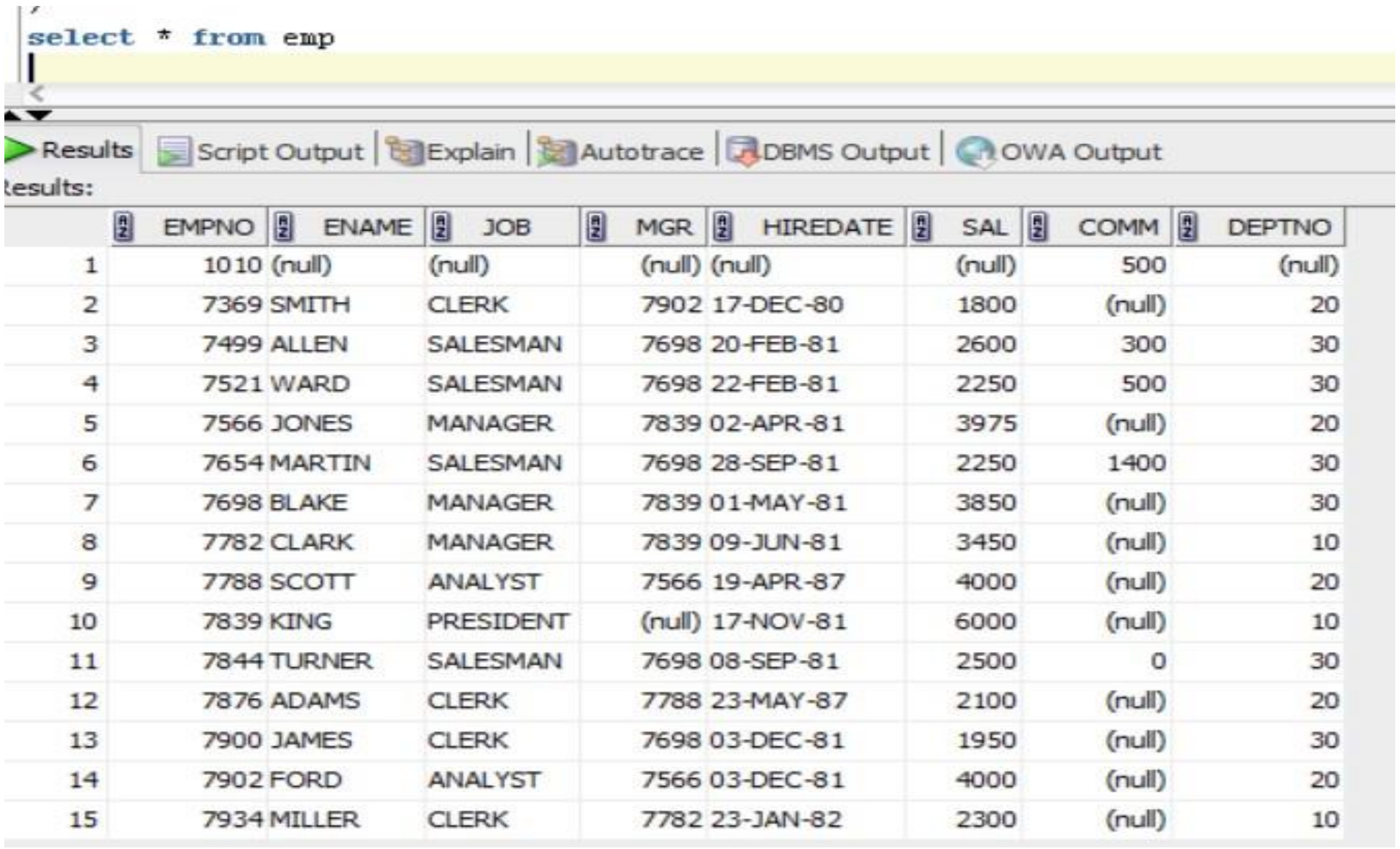
```
DECLARE  total_rows number(2); BEGIN
UPDATE EMP2
  SET sal = sal + 500;
  IF SQL%NOTFOUND THEN
    dbms_output.put_line('No employees
    selected');
  ELSIF SQL%FOUND THEN total_rows :=
    SQL%ROWCOUNT;
    dbms_output.put_line( total_rows || '
    Employees selected ');
  END IF;  END;
/
```



```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE EMP
   SET sal = sal + 500;
   IF SQL%NOTFOUND THEN
      dbms_output.put_line('No employees selected');
   ELSIF SQL%FOUND THEN
      total_rows := SQL%ROWCOUNT;
      dbms_output.put_line( total_rows ||' ' || 'Employees selected');
   END IF;
END;
/
```

Buffer Size: 20000

```
15 Employees selected
```



```
select * from emp
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1010	(null)	(null)	(null)	(null)	(null)	500	(null)
2	7369	SMITH	CLERK	7902	17-DEC-80	1800	(null)	20
3	7499	ALLEN	SALESMAN	7698	20-FEB-81	2600	300	30
4	7521	WARD	SALESMAN	7698	22-FEB-81	2250	500	30
5	7566	JONES	MANAGER	7839	02-APR-81	3975	(null)	20
6	7654	MARTIN	SALESMAN	7698	28-SEP-81	2250	1400	30
7	7698	BLAKE	MANAGER	7839	01-MAY-81	3850	(null)	30
8	7782	CLARK	MANAGER	7839	09-JUN-81	3450	(null)	10
9	7788	SCOTT	ANALYST	7566	19-APR-87	4000	(null)	20
10	7839	KING	PRESIDENT	(null)	17-NOV-81	6000	(null)	10
11	7844	TURNER	SALESMAN	7698	08-SEP-81	2500	0	30
12	7876	ADAMS	CLERK	7788	23-MAY-87	2100	(null)	20
13	7900	JAMES	CLERK	7698	03-DEC-81	1950	(null)	30
14	7902	FORD	ANALYST	7566	03-DEC-81	4000	(null)	20
15	7934	MILLER	CLERK	7782	23-JAN-82	2300	(null)	10

EXPLICIT CURSORS

- Explicit cursors are programmer-defined cursors for gaining more control over the context area.
- An explicit cursor should be defined in the declaration section of the PL/SQL Block.
- It is created on a SELECT Statement which returns more than one row.

SYNTAX:

CURSOR cursor_name **IS** select_statement **;**

- Working with an explicit cursor includes the following steps:

1. Declaring the cursor for initializing the memory.
2. Opening the cursor for allocating the memory.
3. Fetching the cursor for retrieving the data.
4. Closing the cursor to release the allocated memory.

DECLARATION OF EXPLICIT CURSORS

- Cursor declaration is done in the **DECLARE** block.
- There are three types of cursor declaration:

  a) **Cursor without parameters.**

  b) **Cursor with parameters.**

  c) **Cursor with return clause.**

a) CURSOR WITHOUT PARAMETER:

SYNTAX:

```
CURSOR cur_name

       IS

           select_statement ;
```

EXAMPLE:

```
DECLARE

  CURSOR emp_cur
         IS

               SELECT ename
           FROM emp

                WHERE sal > 100 ;
```

OPENING AN EXPLICIT CURSOR

- Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

SYNTAX: OPEN

cur_name ;

SYNTAX:

```
OPEN emp_cur ;
```

FETCHING ROWS FROM AN EXPLICIT CURSOR

- Fetch statement is used to fetch data from a cursor and assign that data to variables.
- The fetch statement retrieves the current row and advances the cursor to the next row to fetch the remaining rows.

SYNTAX:

```
FETCH cursor_name INTO var1,var2,……
```

CLOSING AN EXPLICIT CURSOR

- The close statement is used to disable a cursor and release the memory occupied by it.

SYNTAX:

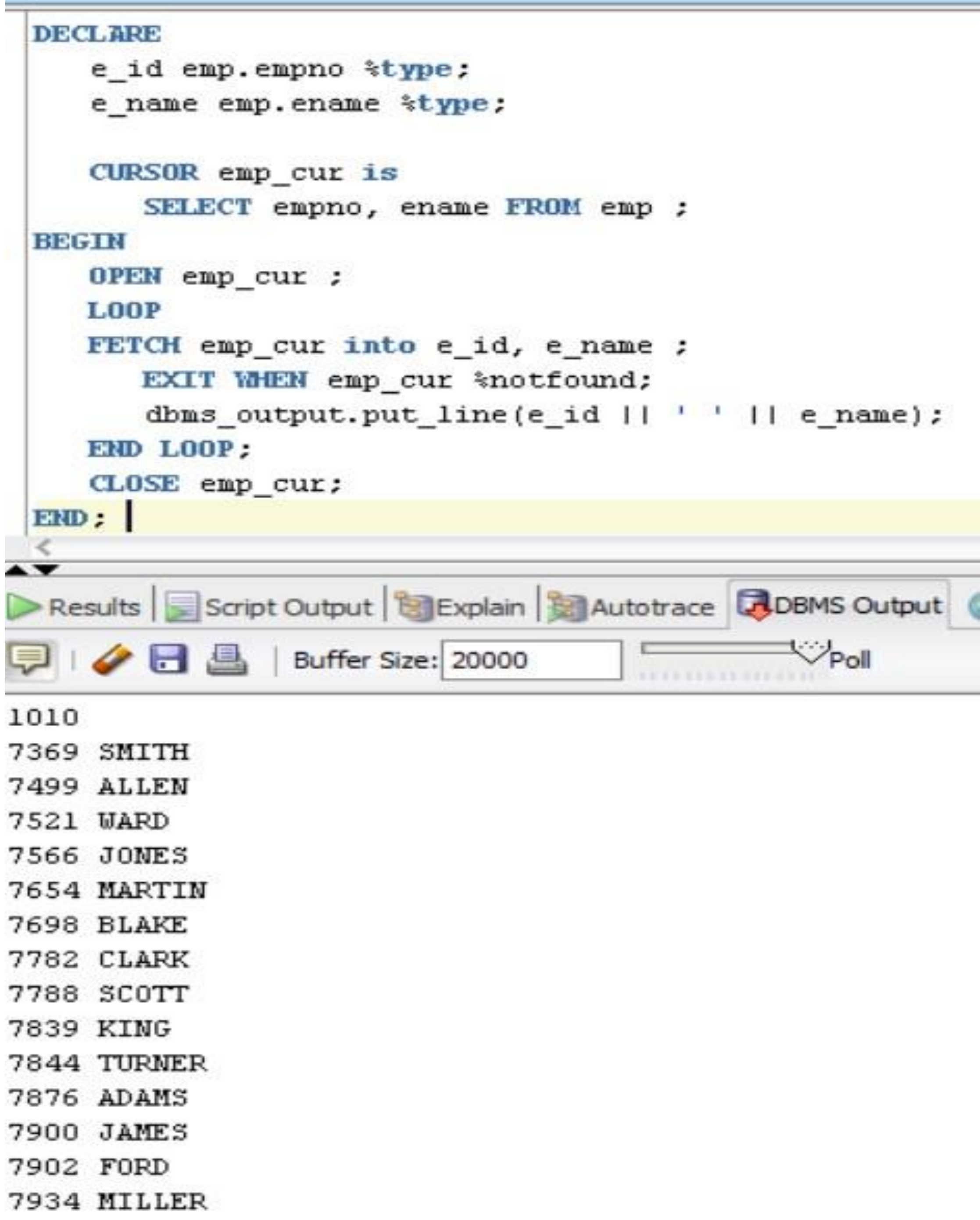
```
CLOSE cur_name ;
```

EXPLICIT CURSOR ATTRIBUTES

S.NO	ATTRIBUTE & DESCRIPTION
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE. The attribute is used after the fetch statement. If the attribute is used after the last row is fetched the attribute returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Checks whether the cursor is open. It returns TRUE if cursor is OPEN else returns FALSE.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. This attribute works like a counter. It returns 0 when the cursor is opened for the first time. This attribute can be used to limit the rows of result set to be returned by cursor.

```
DECLARE e_id emp.empno %TYPE; e_name
  emp.ename %TYPE;

  CURSOR emp_cur IS
    SELECT empno, ename FROM emp ;
BEGIN
  OPEN emp_cur ;
  LOOP
  FETCH emp_cur INTO e_id, e_name ;
    EXIT WHEN emp_cur %NOTFOUND ;
   DBMS_OUTPUT.PUT_LINE (e_id || ' ' ||  e_name)
;
  END LOOP ;
  CLOSE emp_cur ; END;
```



```
DECLARE
    e_id emp.empno %type;
    e_name emp.ename %type;

    CURSOR emp_cur is
       SELECT empno, ename FROM emp ;
BEGIN
    OPEN emp_cur ;
    LOOP
    FETCH emp_cur into e_id, e_name ;
       EXIT WHEN emp_cur %notfound;
       dbms_output.put_line(e_id || ' ' || e_name);
    END LOOP;
    CLOSE emp_cur;
END;
```

```
1010
7369 SMITH
7499 ALLEN
7521 WARD
7566 JONES
7654 MARTIN
7698 BLAKE
7782 CLARK
7788 SCOTT
7839 KING
7844 TURNER
7876 ADAMS
7900 JAMES
7902 FORD
7934 MILLER
```

CURSOR FOR LOOP

- The cursor FOR LOOP statement is an elegant extension of the numeric FOR LOOP statement.
- The numeric FOR LOOP executes the body of a loop once for every integer value in a specified range. Similarly, the cursor FOR LOOP executes the body of the loop once for each row returned by the query associated with the cursor.
- A nice feature of the cursor FOR LOOP statement is that it allows you to fetch every row from a cursor without manually managing the execution cycle i.e., OPEN, FETCH, and CLOSE.
- The cursor FOR LOOP implicitly creates its loop index as a record variable with the row type in which the cursor returns and then opens the cursor.
- In each loop iteration, the cursor FOR LOOP statement fetches a row from the result set into its loop index. If there is no row to fetch, the cursor FOR LOOP closes the cursor.
- The cursor is also closed if a statement inside the loop transfers control outside the loop, e.g., EXIT and GOTO, or raises an exception.

SYNTAX:

```
FOR record IN cursor_name
LOOP
        statements ;
END LOOP ;
```

- The record is the name of the index that the cursor FOR LOOP statement declares implicitly as a %ROWTYPE record variable of the type of the cursor.

- The record variable is local to the cursor FOR LOOP statement. It means that you can only reference it inside the loop, not outside. After the cursor FOR LOOP statement execution ends, the record variable becomes undefined.

```
DECLARE
 CURSOR c_product
 IS
  SELECT product_name,
    list_price
  FROM
    products
  ORDER BY  list_price DESC ;
BEGIN
 FOR r_product IN c_product
 LOOP
  DBMS_OUTPUT.PUT_LINE( r_product.product_name || ': $' ||  r_product.list_price ) ;
END LOOP ; END
```