# Practical 4 : Cursors

## Learning objectives:

1. Definition of cursor
2. Implicit and Explicit cursor
3. Cursor for loop
4. Cursor Expression – Nested cursor
5. Transaction Control – Locking a Table for Update: SELECT FOR UPDATE and FOR UPDATE Cursors

**References**
PL/SQL Language Reference https://docs.oracle.com/en/database/oracle/oracle-database/18/lnpls/database-pl-sql-language-reference.pdf

https://docs.oracle.com/en/database/oracle/oracle-database/19/tdddg/

## 1. Definition of cursor

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.

## 2. Implicit and Explicit cursor

A SQL (implicit) cursor is opened by the database to process each SQL statement that is not associated with an explicit cursor. Every SQL (implicit) cursor has six attributes, each of which returns useful information about the execution of a data manipulation statement.

Some of the implicit cursor attributes are:

- SQL%FOUND
- SQL%NOTFOUND
- SQL%ROWCOUNT

You can use the implicit cursor attributes to determine the outcome of a SQL statement execution.

```
create or replace procedure adjust_price(in_productcode varchar) is
begin
   UPDATE products
     SET buyprice = buyprice * 1.05
     WHERE productcode = in_productcode;

   IF SQL%NOTFOUND THEN
      dbms_output.put_line('No such product - '|| in_productCode);
   elsif SQL%FOUND THEN
      dbms_output.put_line('Product '|| in_productCode|| ' - buy price is adjusted successfully');
   END IF;
end;
/
```

**What happens if the IF…END IF statement is omitted and a wrong product code is passed in?**

The explicit cursor attributes are:

| | |
|---|---|
| %ROWCOUNT | When its cursor or cursor variable is opened, `%ROWCOUNT` is zeroed. Thereafter, it yields the number of rows fetched so far. The number is incremented if the last fetch returned a row. |
| %FOUND | TRUE when a cursor has some remaining rows to fetch, and FALSE when a cursor has no rows left to fetch |
| %NOTFOUND | TRUE if a cursor has no rows to fetch, and FALSE when a cursor has some remaining rows to fetch. |
| %ISOPEN | TRUE if cursor is opened, or FALSE if cursor has not been opened or has been closed. Only used with explicit cursors. |

| | |
|---|---|
| DECLARE | Declaring an explicit cursor names the cursor with the query associated with the cursor. **CURSOR <cursorname> IS <SELECT statement>;** Can use any legal SELECT statements, including joins and statements with the UNION or MINUS clause. |
| OPEN | **OPEN <cursorname>** causes the SQL commands to parse the SQL Query (i.e. check for syntax errors). The OPEN command causes the cursor to identify the data rows that satisfy SELECT query. However the data values are not actually retrieved. |
| FETCH | Loads the row addressed by the cursor pointer into variables and moves the cursor pointer on to the next row ready for the next fetch. **FETCH <cursorname> INTO <record variable(s)>;** |
| CLOSE | Releases the data within the cursor and closes it. The cursor can be reopened to refresh its data. **CLOSE <cursorname>;** |

Cursors are defined within a DECLARE section of a PL/SQL block.
An example follows:

```
DECLARE
   CURSOR mycur  IS  SELECT  emp_ssn, emp_last_name FROM employee;
   …
```

**Sample 1 : Prac4_proc1.sql**

**Prac4_proc1.sql** is a stored procedure that prints a report to highlight products running low in stock using cursor construct.

```
/*
CREATE TABLE products(
        productCode          varchar(15) NOT NULL,
        productName          varchar(70) NOT NULL,
        productLine          varchar(50) NOT NULL,
        productScale         varchar(10) NOT NULL,
        productVendor                varchar(50) NOT NULL,
        productDescription   varchar(4000) NOT NULL,
        quantityInStock              number(4) NOT NULL,
        buyPrice             number(7,2) NOT NULL,
        MSRP                 number(7,2) NOT NULL,
        PRIMARY KEY (productCode)
);
*/
CREATE OR REPLACE PROCEDURE prc_Low_Stock(v_lowQty IN NUMBER) IS

 v_prodCode   PRODUCTS.productCode%TYPE;
 v_prodName PRODUCTS.productName%TYPE;
 v_prodLine    PRODUCTS.ProductLine%TYPE;
 v_prodScale  PRODUCTS.productScale%TYPE;
 v_prodVendor         PRODUCTS.productVendor%TYPE;
 v_prodqty     PRODUCTS.quantityInStock%TYPE;
 v_buyPrice    PRODUCTS.buyPrice%TYPE;

 v_indicator    char(5);

CURSOR PROD_CURSOR IS
  SELECT productCode, productName, productLine, productScale, productVendor, quantityInStock,
buyPrice
  FROM PRODUCTS;

BEGIN
 DBMS_OUTPUT.PUT_LINE('PRODUCTS RUNNING LOW ON STOCK');
 DBMS_OUTPUT.PUT_LINE('=============================');
 DBMS_OUTPUT.PUT_LINE('====PLS=====INSERT==SUB-HEADING===HERE==============');

 OPEN PROD_CURSOR;
 LOOP
   FETCH PROD_CURSOR INTO
      v_prodCode, v_prodName, v_prodLine, v_prodScale, v_prodVendor, v_prodQty, v_buyPrice;
   EXIT WHEN PROD_CURSOR%NOTFOUND;
   if v_prodQty < v_lowQty then
     v_indicator := '#<---';
   else
     v_indicator := '     ';
   end if;

   DBMS_OUTPUT.PUT_LINE(v_prodCode||'-'||v_prodName||'-'||v_prodLine||'-'||
     v_prodScale||'-'||v_prodVendor||'-'||v_prodQty||'-'||v_buyPrice||v_indicator);
 END LOOP;

 DBMS_OUTPUT.PUT_LINE('=====================================');
 DBMS_OUTPUT.PUT_LINE('TOTAL PRODUCTS PROCESSED ' || PROD_CURSOR%ROWCOUNT);
 DBMS_OUTPUT.PUT_LINE('# indicates products low in quantity -- less than :'||v_lowQty);
 DBMS_OUTPUT.PUT_LINE('--- END OF REPORT ----');
 CLOSE PROD_CURSOR;
-- cursor by default will be closed when procedure ends
END;
/
```

TASK: The output has not been adjusted/formatted properly. Modify the code to produce a readable report.

**Sample 2 : Prac4_proc2.sql**

Prac4_**proc2.sql** is a stored procedure that uses two cursors to print order details of each customer.

```
/*
CREATE TABLE orders(
orderNumber  number(11) NOT NULL,
orderDate       date NOT NULL,
requiredDate  date NOT NULL,
shippedDate   date DEFAULT NULL,
status             varchar(15) NOT NULL,
comments       varchar(500),
customerNumber       number(11) NOT NULL,
PRIMARY KEY (orderNumber)
);
CREATE TABLE orderdetails(
orderNumber             number(11)  NOT NULL,
productCode              varchar(15) NOT NULL,
quantityOrdered                    number(4)   NOT NULL,
priceEach                  number(7,2) NOT NULL,
orderLineNumber                    number(3)   NOT NULL,
PRIMARY KEY (orderNumber,productCode)
);
*/

CREATE OR REPLACE PROCEDURE prc_order_details AS
  v_orderNo              ORDERS.orderNumber%TYPE;
  v_orderDate            ORDERS.orderDate%TYPE;
  v_requiredDate         ORDERS.requiredDate%TYPE;
  v_shippedDate          ORDERS.shippedDate%TYPE;
  v_custNo               ORDERS.customerNumber%TYPE;
  v_productCode          ORDERDETAILS.productCode%TYPE;
  v_qtyOrd               ORDERDETAILS.quantityOrdered%TYPE;
  v_priceEach            ORDERDETAILS.priceEach%TYPE;

cursor order_cursor is
     select customerNumber, orderNumber, orderDate, requiredDate, shippedDate
     from ORDERS;

cursor orderDetail_cursor is
     select productCode, quantityOrdered, priceEach
     from ORDERDETAILS
     where orderNumber = v_orderNo;

BEGIN
  OPEN order_cursor;
  FETCH order_cursor
      INTO v_custNo, v_orderNo, v_orderDate, v_requiredDate, v_shippedDate;
  WHILE order_cursor%FOUND
  LOOP
    DBMS_OUTPUT.PUT_LINE('Customer No : '||v_custNo);
    DBMS_OUTPUT.PUT_LINE('Order    No : '||v_orderNo);
    DBMS_OUTPUT.PUT_LINE('Order Date    : '||v_orderDate);
    DBMS_OUTPUT.PUT_LINE('Shipped       : '||v_shippedDate);
    DBMS_OUTPUT.PUT_LINE('Required Date  : '||v_requiredDate);
    dbms_output.put_line(chr(10));

    OPEN orderDetail_cursor;
    FETCH orderDetail_cursor
        INTO v_productCode, v_qtyOrd, v_priceEach;

    WHILE orderDetail_cursor%FOUND
    LOOP
      DBMS_OUTPUT.PUT_LINE(v_productCode||'***'||v_qtyOrd||'***'||v_priceEach);

      FETCH orderDetail_cursor
        INTO v_productCode, v_qtyOrd, v_priceEach;
    END LOOP;
    CLOSE orderDetail_cursor;

    DBMS_OUTPUT.PUT_LINE('End of Customer '||v_custNo||'************************');
    dbms_output.put_line(chr(10));

  FETCH order_cursor
      INTO v_custNo, v_orderNo, v_orderDate, v_requiredDate, v_shippedDate;
  END LOOP;
  CLOSE order_cursor;
END;
```

TASK: Suggest how this report can be improved.

TASK: Modify the procedure to accept CustomerCode as input and print all order details for that customer.

Sample 3: Prac4_**proc3.sql** (using **Cursor FOR LOOP**)

Prac4_**proc3.sql** is a stored procedure to list out some employee details, using the FOR…LOOP.

```
CREATE OR REPLACE PROCEDURE prc_emp_list AS
    CURSOR emp_cursor IS
         SELECT employeeNumber, lastName, firstName, email, officeCode
         FROM Employees
          order by employeeNumber;

BEGIN
   FOR emp_rec IN emp_cursor LOOP
       DBMS_OUTPUT.PUT_LINE(  emp_cursor%rowcount ||'. '||emp_rec.employeeNumber||' '||
                  emp_rec.lastName||',' ||emp_rec.firstName ||
                   ' '||emp_rec.email||' '||emp_rec.officeCode);
   END LOOP;
END;
/
```

TASK: What is the fundamental difference between a FETCH… compared to a FOR..LOOP in reading rows from the cursor?

With cursor FOR LOOP, the process of opening, fetching, and closing are implicitly handled.

TASK: Modify the procedure to list the name of the ReportsTo employee.

**Sample 4: Prac4_proc4.sql**

Prac4_**proc4.sql** is a stored procedure using cursor expression to list out all products for each product line.

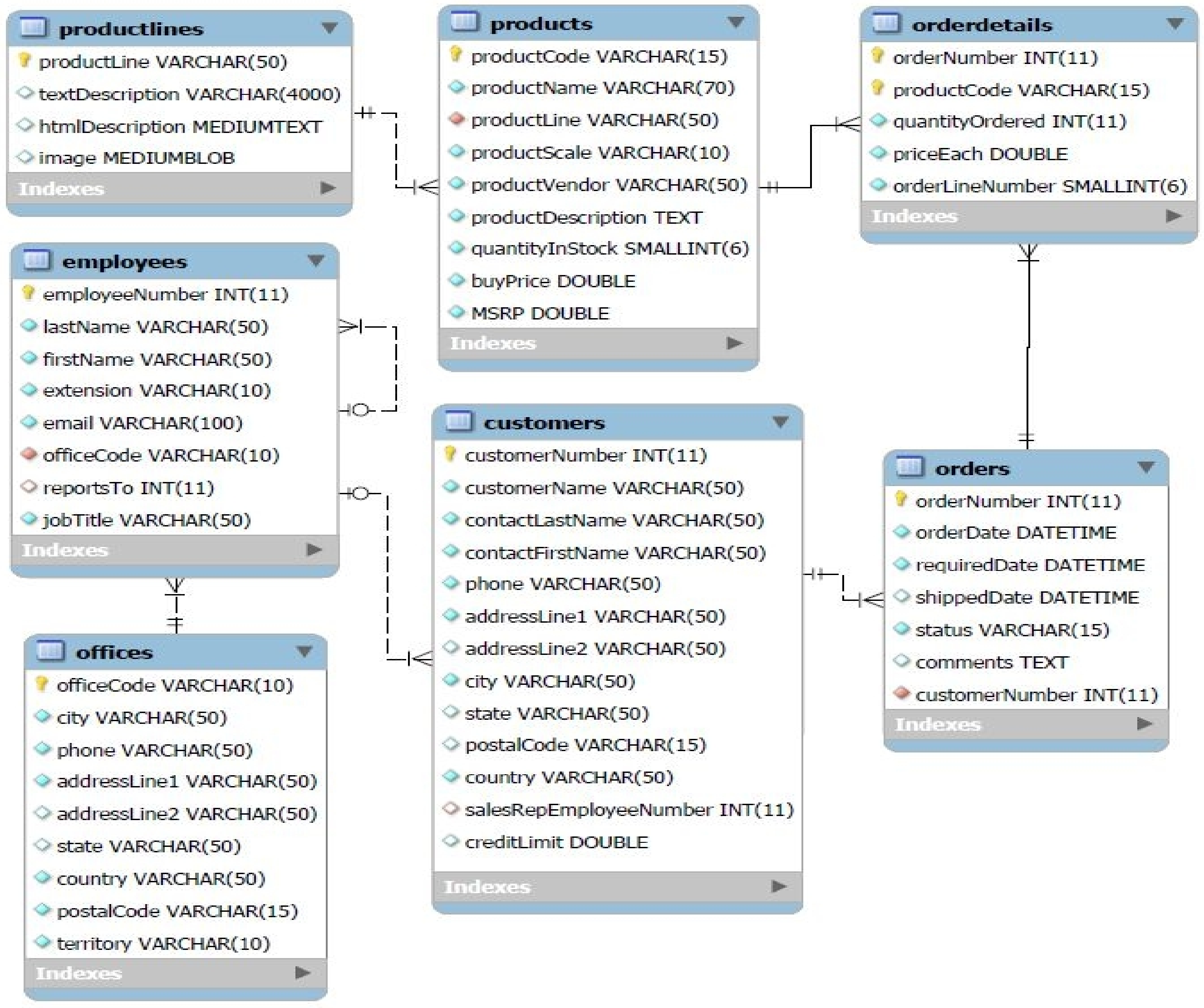
```
DECLARE
  TYPE products_cursor_typ IS REF CURSOR;
  productList_cursor  products_cursor_typ;

  v_productline         productLines.productLine%TYPE;
  v_Desc                productLines.textDescription%TYPE;
  v_productCode         products.productCode%TYPE;
  v_productName         products.productName%TYPE;
  v_MSRP                products.MSRP%TYPE;

  CURSOR c1 IS
    select a.productLine, a.textDescription,
         CURSOR ( select b.productCode, b.productName, b.MSRP
                  from products b
                  where b.productLine = a.productLine
                 ) products_info
    from productLines a;

BEGIN
  OPEN c1;
  LOOP -- Process each row of query result set
     FETCH c1 INTO v_productLine,v_Desc, productList_cursor;
         EXIT WHEN c1%NOTFOUND;
     dbms_output.put_line(chr(10));
    DBMS_OUTPUT.PUT_LINE('Product line: ' || v_productLine);
    DBMS_OUTPUT.PUT_LINE('Description : ' || v_Desc);
    LOOP -- Process each row of subquery result set
        FETCH productList_cursor INTO v_productCode, v_productName, v_MSRP;
          EXIT WHEN productList_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Product Code: ' ||v_productCode||' price is '||v_MSRP);
       DBMS_OUTPUT.PUT_LINE('Product Name: ' ||v_productName);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('No. of products in this line :' || productList_cursor%rowcount);

  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Total Product lines :' || c1%rowcount);
  CLOSE c1;
END;
/
```

**Sample 5 : Locking the table while updating a record using cursor**

When a SELECT...FOR UPDATE statement is issued, the DBMS automatically obtains exclusive row-level locks on all the rows identified by the SELECT statement, holding the records "for your changes only" as you move through the rows retrieved by the cursor. No one else will be able to change any of these records until you perform a ROLLBACK or a COMMIT.

```
DECLARE
--Cursor1 is called a FOR UPDATE cursor
  CURSOR cursor1 IS
    SELECT * FROM PRODUCTS
    FOR UPDATE;

  v_new_price Products.MSRP%TYPE;

BEGIN
  FOR cursor1_rec IN cursor1 LOOP
    IF (cursor1_rec.MSRP < 100.00) THEN
      UPDATE PRODUCTS
      SET MSRP = ROUND((1.2*MSRP),2)
      WHERE CURRENT OF cursor1;
--Only a FOR UPDATE cursor can appear in the CURRENT OF clause of an UPDATE or DELETE statement.

      v_new_price := ROUND((1.2* cursor1_rec.MSRP),2);

      DBMS_OUTPUT.PUT_LINE('Price of Product '||cursor1_rec.productCode||' '||
          RPAD(cursor1_rec.ProductName,40, ' ')||' changed from $'||
          cursor1_rec.MSRP ||' to $' || v_new_price);
    END IF;
  END LOOP;
END;
/
```

## Questions



1. Write a procedure to print all products for a given product line. The procedure will receive productLine as input. You should print useful relevant information.

2. Write a procedure to list all order details for each order for a particular date range of OrderDate. Calculate and print the total value of all the orders for that date range.

3. Produce a report to show all customers whose last order was more than 6 months ago (i.e. have not been active for the last 6 months). Indicate the value of their last order.

4. Produce a report to show the profit margin (in percentage) for each product. At the end of the report, calculate and print the average profit margin. List the record from the highest to the lowest profit margin.

5. Print a report with the title “Product Total Unit Sales Report”. List the records from the highest to the lowest total units sold. Include also the total sales value of each product.