# It is faster to read the notes than to look at the recording

# Les09-PL/SQL part 3 Cursor

## What is cursor?

It is a pointer to a context area.

## What is a context area?

It is a separate memory that holds the metadata results from the currently running SQL statement. Metadata is things like number of rows etc.

For our purposes, a cursor is a private area that contains the SQL results.

➔ Cursor gives you extra control over the context area.

## Execution cycle of an explicit cursor

# Beginning look at Cursor

## First Declare the Cursor

You must declare the cursor in the block before

CURSOR cursor_name IS query

```
DECLARE
-- declare the cursor here

CURSOR first_cursor IS
        SELECT employee_id, last_name
        FROM employees;

-- declare the variables to load data into
v_empid      employees.employee_id%TYPE; ←get datatype, size from column employee_id
v_name       employees.last_name%TYPE;
```

Give the cursor a name and the SQL to run

## Then continue the life cycle of the cursor

```
BEGIN
-- open the cursor

-- FETCH the cursor

-- close the CURSOR

END;
```

## Open Cursor
## Close Cursor

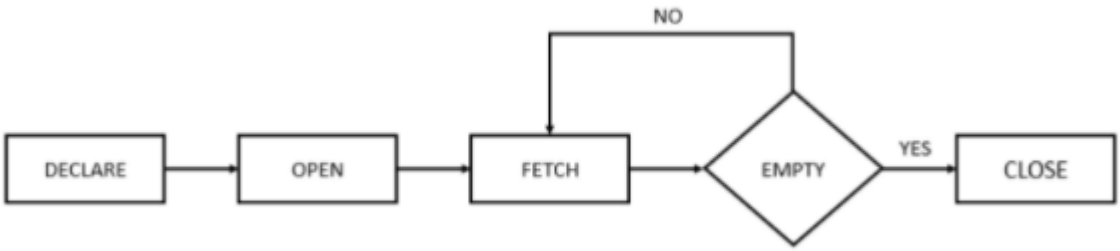
OPEN cursor_name;

CLOSE cursor_name;

```
DECLARE
-- declare the cursor here

CURSOR first_cursor IS
        SELECT employee_id, last_name
        FROM employees;

-- declare the variables to load data into
        v_empid       employees.employee_id%TYPE;
        v_name        employees.last_name%TYPE;

BEGIN
-- open the cursor
        OPEN first_cursor;

-- FETCH the cursor

-- close the CURSOR
        CLOSE first_cursor;

END;
```

At the time of OPEN the SELECT is executed and data stored in cursor area

## Actions to do




```
DECLARE
-- declare the cursor here
CURSOR first_cursor IS
        SELECT employee_id, last_name
        FROM employees;

BEGIN
-- open the cursor
        OPEN first_cursor;

-- FETCH the cursor
        FETCH first_cursor INTO

-- close the CURSOR
        CLOSE first_cursor

END;
```

## Adding the FETCH into portion

```
DECLARE
-- declare the cursor here

CURSOR first_cursor IS
        SELECT employee_id, last_name
        FROM employees;

-- declare the variables to load data into
        v_empid       employees.employee_id%TYPE;
        v_name        employees.last_name%TYPE;

BEGIN
-- open the cursor
        OPEN first_cursor;

-- FETCH the cursor
        FETCH first_cursor INTO v_empid, v_name;

        DBMS_OUTPUT.PUT_LINE (v_empid || v_name);

-- close the CURSOR
        CLOSE first_cursor;

END;
```

FETCH first row from cursor area into variables

Add the way to show the output

**OUTPUT:**

**174Abel** ← Note only returned the first row

PL/SQL procedure successfully completed.

**To improve it we can** cut and paste repeated FETCH to get more rows

```
-- FETCH the cursor
        FETCH first_cursor INTO v_empid, v_name;
        DBMS_OUTPUT.PUT_LINE (v_empid || v_name);
        FETCH first_cursor INTO v_empid, v_name;
        DBMS_OUTPUT.PUT_LINE (v_empid || v_name);
```

And this is the output

**174Abel**
**23Armarillo**

## Adding a LOOP
## Adding an EXIT

We need an EXIT, or the FETCH will continue to loop and cause a load on the server and a problem stopping the loop

```
DECLARE
-- declare the cursor here

CURSOR first_cursor IS
        SELECT employee_id, last_name
        FROM employees;

-- declare the variables to load data into
        v_empid       employees.employee_id%TYPE;
        v_name        employees.last_name%TYPE;

BEGIN
-- open the cursor
        OPEN first_cursor;

-- FETCH the cursor
LOOP

        FETCH first_cursor INTO v_empid, v_name;
                EXIT WHEN first_cursor%NOTFOUND;  --will exit if TRUE meaning not found

        DBMS_OUTPUT.PUT_LINE (v_empid || v_name);

END LOOP;

-- close the CURSOR
        CLOSE first_cursor;

END;
```

**OUTPUT:**
**'**
**'**
**'**
**40Whiteduck**
**28Young**
**149Zlotkey**
**180de Man**

Improve it a bit more on next page

# Adding Row Count

```
DECLARE
-- declare the cursor here

CURSOR first_cursor IS
        SELECT employee_id, last_name
        FROM employees;

-- declare the variables to load data into
        v_empid       employees.employee_id%TYPE;
        v_name        employees.last_name%TYPE;

BEGIN
-- open the cursor
        OPEN first_cursor;

-- FETCH the cursor
LOOP
        FETCH first_cursor INTO v_empid, v_name;
                EXIT WHEN first_cursor%NOTFOUND;  --will exit if TRUE
        DBMS_OUTPUT.PUT_LINE (v_empid || v_name);
END LOOP;

        DBMS_OUTPUT.PUT_LINE (first_cursor%ROWCOUNT);

-- close the CURSOR
        CLOSE first_cursor;

END;
```

**OUTPUT:**
**40Whiteduck**
**28Young**
**149Zlotkey**
**180de Man**
**54** ← row count

# What if doing it for the whole row in the table

```
DECLARE
-- declare the cursor here
        CURSOR first_cursor IS SELECT * FROM employees;

        employee_rec        Employees%ROWTYPE;

BEGIN
-- open the cursor
        OPEN first_cursor;

-- FETCH the cursor
LOOP

        FETCH first_cursor INTO  employee_rec;
                EXIT WHEN first_cursor%NOTFOUND; -- will exit if TRUE

        DBMS_OUTPUT.PUT_LINE (employee_rec.employee_id || employee_rec.last_name);

END LOOP;

        DBMS_OUTPUT.PUT_LINE (first_cursor%ROWCOUNT);

-- close the CURSOR
        CLOSE first_cursor;

END;
```

To avoid defining every column in the table you will define a record as shown

Displaying just two parts of record (ID and name), for simplicity

**Watch the output is in different order than you might get**
**36Termede**
**39Testorok**
**40Whiteduck**
**41Montoya**
**54**

# To AVOID the Life Cycle  use FOR

```
DECLARE
-- declare the cursor here
        CURSOR first_cursor IS SELECT * FROM employees;
        E_rec           Employees%ROWTYPE;

BEGIN

FOR E_rec in first_cursor
        LOOP
        DBMS_OUTPUT.PUT_LINE( 'ID: ' || E_rec.employee_id ||' Name: ' || E_rec.last_name);

        END LOOP;

END;
```

**OUTPUT:**

**ID: 36 Name: Termede**
**ID: 39 Name: Testorok**
**ID: 40 Name: Whiteduck**
**ID: 41 Name: Montoya**

# Attributes of Cursor

## 4 Attributes

### %ISOPEN

If the cursor is open the value is TRUE and if not open, then it is FALSE

### %FOUND

This attribute has 4 possible values

| | |
|---|---|
| NULL | before the first fetch has occurred |
| TRUE | if a record was fetched successfully |
| FALSE | if no row was returned by the fetch |
| INVALID_CURSOR | if the cursor was not opened |

### %NOTFOUND

This attribute also has 4 values

| | |
|---|---|
| NULL | before the first fetch |
| TRUE | If no record was fetched |
| FALSE | if a record was fetched |
| INVALID_CURSOR | if cursor was not opened |

### %ROWCOUNT

Returns the number of rows fetched from the cursor

INVALID_CURSOR if cursor was nor opened

PL/SQL Cursor Example

Step 1 – add a column to the CUSTOMERS table

ALTER TABLE customers
ADD credit_limit number (8,2);

Check it was added.

Populate the credit_limit column in customers by a percentage of the customers sales.

Step 2
Create a view to calculate credit value as 5% of total sales revenue

```
CREATE or replace VIEW sales_view AS
SELECT cust_no,
       SUM(price * qty) total,
       ROUND(SUM(price * qty) * 0.05) credit
FROM orderlines
INNER JOIN orders USING (order_no)
WHERE status = 'C'  -- meaning completed and shipped
GROUP BY cust_no;
```

Test the view

```
select * from sales_view;
```

output: 103 rows, just last ones displayed here

```
  CUST_NO      TOTAL     CREDIT
---------- ---------- ----------
      1139       9602        480
      1012       5092        255
      1121      25702       1285
      1026      10213        511

103 rows selected.
```

Now for the problem

# BLOCK to develop

Problem:
1 set all credit limits to zero.

2 Fetch customers in descending order and give them new credit limits.
The total budget for credit limits can not exceed 100,000 dollars

```
DECLARE
  l_budget NUMBER := 100000;                    -- declare budget and set value
   -- cursor
  CURSOR cust_sales IS
  SELECT  *  FROM sales_view                     -- using view
  ORDER BY total DESC;
   -- record
   r_sales  cust_sales%ROWTYPE;                 -- define a record like the cust_sales

BEGIN
  -- reset credit limit of all customers to start the process
  UPDATE customers
    SET credit_limit = 0;

  OPEN cust_sales;

  LOOP
    FETCH  cust_sales  INTO   r_sales;          -- getting a row at a time
       EXIT WHEN cust_sales%NOTFOUND;

    -- update credit for the current customer since did not EXIT in the above line
    UPDATE  customers
    SET credit_limit =          CASE WHEN l_budget > r_sales.credit
                                          THEN r_sales.credit
                                   ELSE l_budget
                                END
    WHERE  cust_no = r_sales.cust_no;

    --  reduce the budget for credit limit
    l_budget := l_budget - r_sales.credit;

    DBMS_OUTPUT.PUT_LINE( 'Customer id: ' ||r_sales.cust_no ||
' Credit: ' || r_sales.credit || ' Remaining Budget: ' || l_budget );

    -- check the budget
    EXIT WHEN l_budget <= 0;
  END LOOP;
    dbms_output.put_line ('Row Count ' ||cust_sales %rowcount);

  CLOSE cust_sales;

END;
```

OUTPUT:
Customer id: 1102 Credit: 3649 Remaining Budget: 96351
Customer id: 1056 Credit: 3477 Remaining Budget: 92874
Customer id: 1038 Credit: 3273 Remaining Budget: 89601
Customer id: 1130 Credit: 3067 Remaining Budget: 86534
Customer id: 1008 Credit: 3050 Remaining Budget: 83484
Customer id: 1120 Credit: 2962 Remaining Budget: 80522
Customer id: 1062 Credit: 2579 Remaining Budget: 77943

Removed rows at this point to show output on one page

Customer id: 1093 Credit: 989 Remaining Budget: 11977
Customer id: 1137 Credit: 971 Remaining Budget: 11006
Customer id: 1080 Credit: 950 Remaining Budget: 10056
Customer id: 1090 Credit: 932 Remaining Budget: 9124
Customer id: 1111 Credit: 820 Remaining Budget: 8304
Customer id: 1105 Credit: 790 Remaining Budget: 7514
Customer id: 1064 Credit: 786 Remaining Budget: 6728
Customer id: 1025 Credit: 751 Remaining Budget: 5977
Customer id: 1037 Credit: 750 Remaining Budget: 5227
Customer id: 1112 Credit: 739 Remaining Budget: 4488
Customer id: 1108 Credit: 720 Remaining Budget: 3768
Customer id: 1075 Credit: 713 Remaining Budget: 3055
Customer id: 1129 Credit: 710 Remaining Budget: 2345
Customer id: 1069 Credit: 706 Remaining Budget: 1639
Customer id: 1076 Credit: 703 Remaining Budget: 936
Customer id: 1135 Credit: 683 Remaining Budget: 253
Customer id: 1018 Credit: 665 Remaining Budget: -412
Row Count 66

PL/SQL procedure successfully completed.

## Did you see anything wrong with the code?

If YES
Then fix it.

## How do you check if it worked?

```
SELECT cust_no,
      cname,
      credit_limit
FROM customers
ORDER BY credit_limit DESC;
```

# Just a bit more

## 2 styles with the cursor

A) PL/SQL cursor FOR LOOP example
The following example declares an explicit cursor and uses it in the cursor FOR LOOP statement.

```
DECLARE
  CURSOR c_product
  IS
    SELECT prod_name, prod_sell
    FROM    products
    ORDER BY prod_sell DESC;

BEGIN
  FOR r_product IN c_product
  LOOP
    dbms_output.put_line( r_product.prod_name || ':    $' ||  r_product.prod_sell );
  END LOOP;
END;
```

B) Cursor with an SQL statement

```
BEGIN
  FOR r_product IN (
        SELECT prod_name, prod_sell
        FROM    products
        ORDER BY prod_sell DESC
    )
  LOOP
     dbms_output.put_line( r_product.prod_name ||
        ':     $' ||
        r_product.prod_sell );
  END LOOP;
END;
```