# Database Design and Programming

Tahaluf Training Center 2021

# Chapter 08

1. Conditional control
2. Iterative processing with loops
3. **Exception**
4. Records
5. Cursors

# Exception

PL/SQL treats all errors that occur in an anonymous block, procedure, or function as **exceptions**.

The exceptions can have different causes such as coding mistakes, bugs, even hardware failures.

It is not possible to anticipate all potential exceptions, however, you can write code to handle exceptions to enable the program to continue running as normal.

# Exception

The code that you write to handle exceptions is called an **exception handler.**

```
BEGIN
   -- executable section
   ...
   -- exception-handling section
   EXCEPTION
      WHEN e1 THEN
         -- exception_handler1
      WHEN e2 THEN
         -- exception_handler1
      WHEN OTHERS THEN
         -- other_exception_handler
END;
```

# Exception



```
DECLARE
c_id student.id%type := 2;
c_name student.Name%type;

BEGIN
SELECT name, id INTO c_name, c_id
FROM student
WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '|| c_name);
DBMS_OUTPUT.PUT_LINE ('id: ' || c_id);
EXCEPTION
WHEN no_data_found THEN
dbms_output.put_line('No such student!');
WHEN others THEN
dbms_output.put_line('Error!');
END;
```

# Chapter 08

1. Conditional control
2. Iterative processing with loops
3. Exception
4. **Records**
5. Cursors

A PL/SQL **record** is a composite data structure which consists of multiple fields; each has its own value. The following picture shows an example record that includes first name, last name, email, and phone number:

# Records



PL/SQL **record** helps you simplify your code by shifting from field-level to record-level operations.

PL/SQL has three types of records: **table-based**, **cursor-based**, programmer-defined.

Before using a record, you must declare it.

```
DECLARE
  record_name table_name%ROWTYPE;
```

```
CREATE TABLE persons  (
   person_id NUMBER GENERATED BY DEFAULT
AS IDENTITY,
   first_name VARCHAR2( 50 ) NOT NULL,
   last_name VARCHAR2( 50 ) NOT NULL,
   primary key (person_id)
);
```

```
DECLARE
  r_person persons%ROWTYPE;

BEGIN
  -- assign values to person record
  r_person.person_id  := 1;
  r_person.first_name := 'John';
  r_person.last_name  := 'Doe';

  -- insert a new person
  INSERT INTO persons VALUES r_person;
END;
```

# Records

```
DECLARE
  r_person persons%ROWTYPE;

BEGIN
  -- get person data of person id 1
  SELECT * INTO r_person
  FROM persons
  WHERE person_id = 1;

  -- change the person's last name
  r_person.last_name  := 'Smith';

  -- update the person
  UPDATE persons
  SET ROW = r_person
  WHERE person_id = r_person.person_id;
END;
```

# Chapter 08

1. Conditional control
2. Iterative processing with loops
3. Exception
4. Records
5. **Cursors**

# Cursors



A **cursor** is a pointer that points to a result of a query. PL/SQL has two types of cursors: **implicit** cursors and **explicit** cursors.

# 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**
- **SQL%NOTFOUND.**

# Implicit cursors



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.

# Explicit cursors

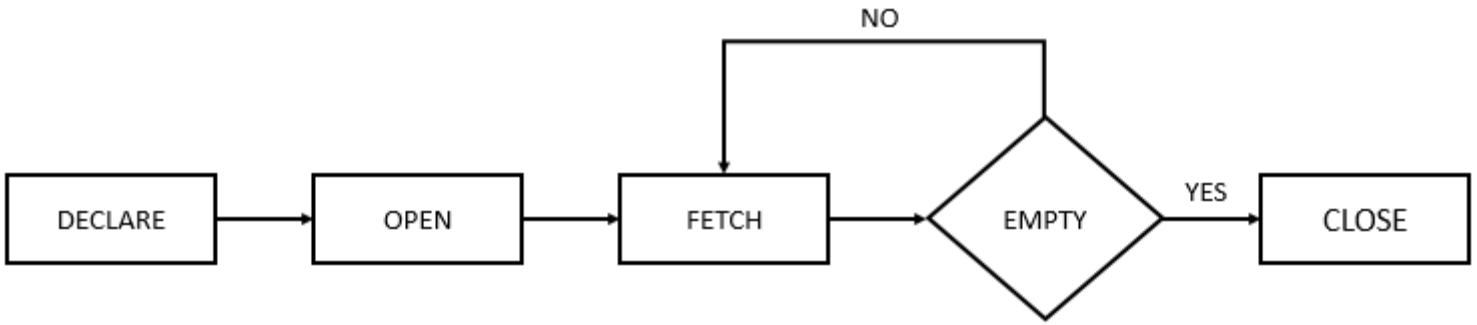


An **explicit** cursor is an SELECT statement declared explicitly in the declaration section of the current block or a package specification.

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.

**The following illustration shows the execution cycle of an explicit cursor:**



## Declare a cursor

Before using an explicit cursor, you must declare it in the declaration section of a block or package as follows:

```
CURSOR cursor_name IS query;
```

## In this syntax:

- **First**, specify the name of the cursor after the CURSOR keyword.
- **Second**, define a query to fetch data after the IS keyword.

## Open a cursor

Before start fetching rows from the cursor, you must open it. To open a cursor, you use the following syntax:

```
OPEN cursor_name;
```

**In this syntax**, the cursor_name is the name of the cursor declared in the declaration section.

## Fetch from a cursor

The FETCH statement places the contents of the current row into variables. The syntax of FETCH statement is as follows:

**FETCH cursor_name INTO variable_list;**

To retrieve all rows in a result set, you need to fetch each row till the last one.

## Closing a cursor

After fetching all rows, you need to close the cursor with the CLOSE statement:

```
CLOSE cursor_name;
```

Closing a cursor instructs Oracle to release allocated memory at an appropriate time.

# Explicit Cursor Attributes



A cursor has four attributes to which you can reference in the following format:

**cursor_name%attribute**

where cursor_name is the name of the explicit cursor.

# Explicit Cursor Attributes



- **%ISOPEN**

This attribute is TRUE if the cursor is open or FALSE if it is not.

- **%FOUND**

This attribute has four values:

1. **NULL** before the first fetch
2. **TRUE** if a record was fetched successfully
3. **FALSE** if no row returned
4. **INVALID_CURSOR** if the cursor is not opened

# Explicit Cursor Attributes

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

- **%NOTFOUND**

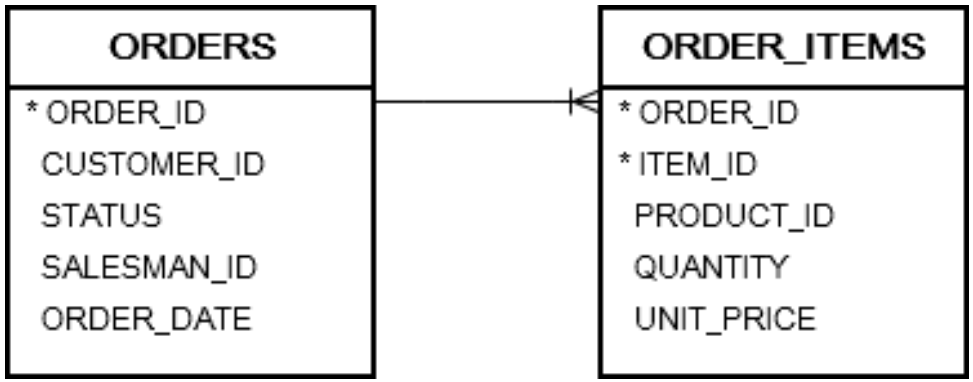
This attribute has four values:

1. **NULL** before the first fetch
2. **FALSE** if a record was fetched successfully
3. **TRUE** if no row returned
4. **INVALID_CURSOR** if the cursor is not opened

- **%ROWCOUNT**

The %ROWCOUNT attribute returns the number of rows fetched from the cursor. If the cursor is not opened, this attribute returns **INVALID_CURSOR**.

We will use the orders and order_items tables from the sample database for the demonstration.



creates a view that returns the sales revenues by customers:

```
CREATE VIEW sales AS
SELECT customer_id,
      SUM(unit_price * quantity) total,
      ROUND(SUM(unit_price * quantity) * 0.05)
credit
FROM order_items
INNER JOIN orders USING (order_id)
WHERE status = 'Shipped'
GROUP BY customer_id;
```

Suppose you need to develop a anonymous block that:

1. Reset credit limits of all customers to zero.
2. Fetch customers sorted by sales in descending order and gives them new credit limits from a budget of 1 million.

The following anonymous block illustrates the logic:

In the declaration section, we declare three variables.

The first one is **l_budget** whose initial value is 1,000,000.

The second variable is an explicit cursor variable named **c_sales** whose SELECT statement retrieves data from the sales view:

```
CURSOR c_sales IS
    SELECT  *  FROM sales
    ORDER BY total DESC;
```

**In the execution section, we perform the following:**

1. First, reset credit limits of all customers to zero using an UPDATE statement.
2. **Second**, open the c_sales cursor.
3. **Third**, fetch each row from the cursor. In each loop iteration, we update the credit limit and reduced the budget. The loop terminates when there is no row to fetch or the budget is exhausted.
4. **Finally**, close the cursor.