Chapter 7 - Using Explicit Cursor:

Cursors (Explicit Cursor) can be used to name a Private SQL Area (aka Context Area) (also a pointer to the memory or rows).

Cursors are used for SELECT and DML (INSERT, UPDATE, DELETE) and to fetch multiple rows

Cursor:

1. Implicit: Managed directly
2. Explicit: Managed by Programmer

4 Steps for using Cursor:

1. Declaration: → (Create a named SQL area)
    a. Cursor Name
    b. SQL Statement
    c. Variable_Name, Record
    d. Note: INTO clause can not be used
2. Open: → (Identify the active set)
    a. Processing of a SQL Statement
        i. Bind → For Cursor
        ii. Parse (Checking Syntax and semantics)
    b. Allocating Memory (Private SQL Area)
    c. Identifying the rows to be fetched → aka **Active set**
    d. Position the pointer to the first row in active set
3. Fetch: →(Load the current row into variables)
    a. Pull/Read the row data into variables
    b. Advances the pointer to the next row in the active set
    c. If there are no more rows to process then close the cursor

Fetching Data from the Cursor (continued)

You can include the same number of variables in the `INTO` clause of the `FETCH` statement as there are columns in the `SELECT` statement; be sure that the data types are compatible. Match each variable to correspond to the columns positionally. Alternatively, you can also define a record for the cursor and reference the record in the `FETCH INTO` clause. Finally, test to see

4. Close: → Release the active set
    a. Deallocating the memory
    b. Releases the active set rows → ready to reopen cursor to establish a fresh active set

Closing the Cursor

The CLOSE statement disables the cursor, releases the context area, and "undefines" the active set. Close the cursor after completing the processing of the FETCH statement. You can reopen the cursor if required. A cursor can be reopened only if it is closed. If you attempt to fetch data from a cursor after it is closed, an INVALID_CURSOR exception is raised.

Note: Although it is possible to terminate the PL/SQL block without closing cursors, you should make it a habit to close any cursor that you declare explicitly to free resources.

There is a maximum limit on the number of open cursors per session, which is determined by the OPEN_CURSORS parameter in the database parameter file. (OPEN_CURSORS = 50 by default.)

Syntax:

```
CURSOR cursor_name IS
     select_statement;
```

Declaring the Cursor

The syntax to declare a cursor is shown in the slide. In the syntax:

cursor_name	Is a PL/SQL identifier
select_statement	Is a SELECT statement without an INTO clause

Cursor FOR Loop:

- Shortcut to process explicit cursors
- Implicit cursor → open, fetch, exit and close occurs
- Record is implicitly declared
- When we use this cursor FOR Loop → it will open cursor, row is fetched for each iteration , loop exits when the last rows is processed and cursor is closed automatically.

Explicit Cursor Attributes

Use explicit cursor attributes to 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

1. %ISOPEN:
    a. Rows can be fetched only when the cursor is open
    b. Use this before performing fetch of data
2. Example for %ROWCOUNT and %NOTFOUND:

```
OPEN c_emp_cursor;
LOOP
 FETCH c_emp_cursor INTO v_emp_record;
 EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
                   c_emp_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                ||' '||v_emp_record.last_name);
END LOOP;
CLOSE c_emp_cursor;
```

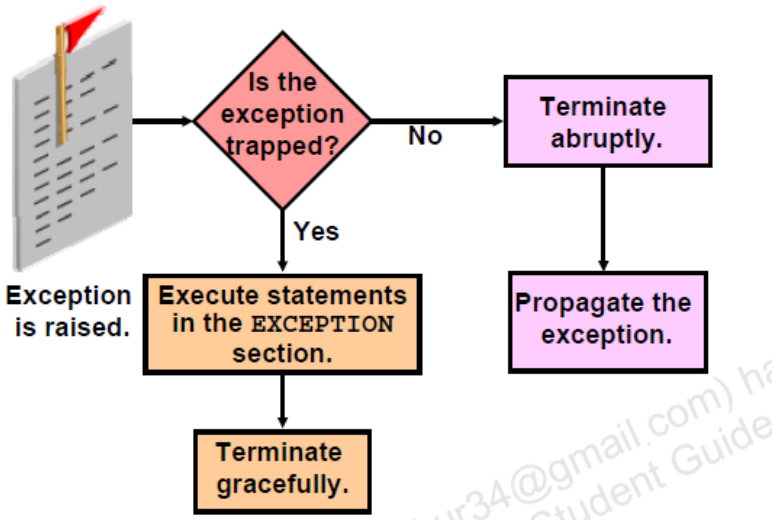
Locking Rows and referencing the current row:

- Use explicit locking to deny access to other sessions for the duration of a transaction
- Lock the rows before the update or delete
- When you are working and you don't want other on that database
- Explicit lock can be removed/ using **COMMIT** or **ROLLBACK**. And if the user doesn't remove lock then DBA Intervention is needed using **KILL SESSION**

Chapter 8 – Exceptions:

- It's not an error but it's kind of an error
- Types:
    1. User defined
    2. Oracle Defined
- Exceptions shows us cause (what caused an exceptions) and actions (what can you do to remove exceptions)
- An exception is a PL/SQL error that is raised during program execution
- Exception raised in two ways:
    - Implicitly
    - Explicitly
- An exception can be handled:
    - By trapping it with handler
    - By Propagating it to the calling environment

Handling Exceptions



Exception Types

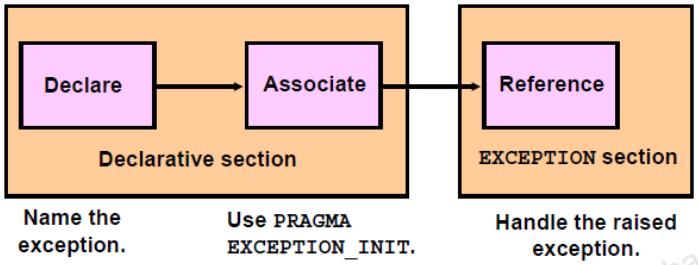
- Predefined Oracle Server
- Non-predefined Oracle Server

} **Implicitly raised**

- User-defined

Explicitly raised

Trapping Non-Predefined Oracle Server Errors



Exception Types

There are three types of exceptions.

Exception	Description	Directions for Handling
Predefined Oracle Server error	One of approximately 20 errors that occur most often in PL/SQL code	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly.
Non-predefined Oracle Server error	Any other standard Oracle Server error	You need to declare these within the declarative section; the Oracle server raises the error implicitly, and you can catch the error in the exception handler.
User-defined error	A condition that the developer determines is abnormal	You need to declare in the declarative section and raise explicitly.

Processing Parts:

i. Bind → For Cursor
ii. Parse (Checking Syntax and semantics)
iii. Define
iv. Describe
v. Parallelise
vi. Execute and Fetch (Not done in Cursor)

For example:

```
WHEN NO_DATA_FOUND THEN
 statement1;
 ...
WHEN TOO_MANY_ROWS THEN
 statement1;
 ...
WHEN OTHERS THEN
 statement1;
```