PL/SQL Cursors

Oracle Academy Study Materials

PL/SQL Cursors

- Every SQL statement executed by the Oracle server has an associated individual cursor.
- A cursor is a pointer to the private memory area allocated by the Oracle server.
- There are two types of cursors:
  - Implicit cursors: Declared and managed by PL/SQL for all DML and PL/SQL SELECT statements
  - Explicit cursors: Declared and managed by the programmer

PL/SQL Cursors

SQL Cursor Attributes for Implicit Cursors

Using SQL cursor attributes, you can test the outcome of your SQL statements.

`SQL%FOUND`	**Boolean attribute that evaluates to `TRUE` if the most recent SQL statement returned at least one row**
`SQL%NOTFOUND`	**Boolean attribute that evaluates to `TRUE` if the most recent SQL statement did not return even one row**
`SQL%ROWCOUNT`	**An integer value that represents the number of rows affected by the most recent SQL statement**

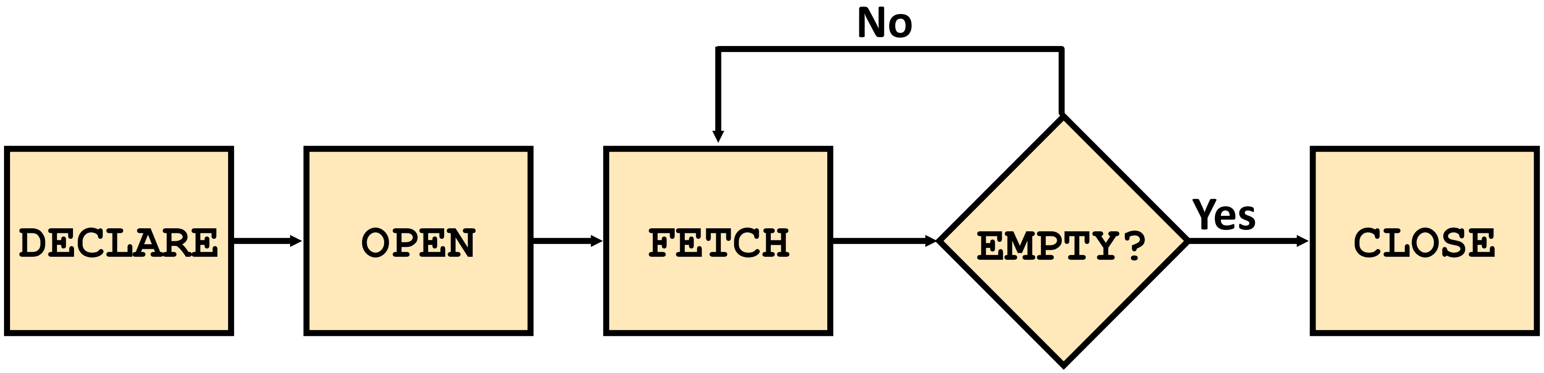
PL/SQL Cursors

SQL Cursor Attributes for Implicit Cursors

- Delete rows that have the specified employee ID from the `employees` table. Print the number of rows deleted.
- Example:

```
DECLARE
  v_rows_deleted VARCHAR2(30)
  v_empno employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM  employees
  WHERE employee_id = v_empno;
  v_rows_deleted := (SQL%ROWCOUNT || ' row deleted.');
  DBMS_OUTPUT.PUT_LINE (v_rows_deleted);
END;
```

Controlling Explicit Cursors



Create a named SQL area.

Identify the active set.

Load the current row into variables.

Test for existing rows.

Return to `FETCH` if rows are found.

Release the active set.

PL/SQL Cursors

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id =30;
  empno employees.employee_id%TYPE;
  lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO empno, lname;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( empno ||' '||lname);
  END LOOP;
  CLOSE emp_cursor;
END;
```

PL/SQL Cursors - 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

PL/SQL Cursors - Cursors and Records

Process the rows of the active set by fetching values into a PL/SQL record.

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id =30;
   v_emp_record   c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                       ||' '||v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END;
```

PL/SQL Cursors - Cursor FOR Loops

Syntax:

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- The cursor FOR loop is a shortcut to process explicit cursors.
- Implicit open, fetch, exit, and close occur.
- Do not declare the record that controls the loop. The record is implicitly declared.
- Test the cursor attributes during the loop, if required.
- Supply the parameters for a cursor, if required, in parentheses following the cursor name in the FOR statement.

PL/SQL Cursors - Cursor FOR Loops

Cursor FOR Loops Using Subqueries

- There is no need to declare the cursor

```
BEGIN
  FOR emp_record IN (SELECT employee_id, last_name
   FROM employees WHERE department_id =30)
  LOOP
   DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
   ||' '||emp_record.last_name);
  END LOOP;
END;
```

PL/SQL Cursors - Cursors with Parameters

Syntax:

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- Pass parameter values to a cursor when the cursor is opened and the query is executed.
- Open an explicit cursor several times with a different active set each time.

```
OPEN  cursor_name(parameter_value,.....) ;
```

PL/SQL Cursors - FOR UPDATE Clause

Syntax:

```
SELECT ...
FROM          ...
FOR UPDATE [OF column_reference][NOWAIT | WAIT n];
```

- If there are multiple sessions for a single database, use explicit locking to deny access to other sessions for the duration of a transaction.
- You see the updated data only when you reopen the cursor.
- The optional NOWAIT keyword tells the Oracle server not to wait if requested rows have been locked by another user. Control is immediately returned to your program.
- If you omit the NOWAIT keyword, the Oracle server waits until the rows are available.
- If the Oracle server cannot acquire the locks on the rows it needs in a SELECT FOR UPDATE operation, it waits indefinitely.
- It is not mandatory for the FOR UPDATE OF clause to refer to a column, but it is recommended for better readability and maintenance.

PL/SQL Cursors - WHERE CURRENT OF Clause

Syntax:

```
WHERE CURRENT OF cursor ;
```

- The WHERE CURRENT OF clause is used in conjunction with the FOR UPDATE clause to refer to the current row in an explicit cursor.
  - Include the `FOR UPDATE` clause in the cursor query to lock the rows first.
- The WHERE CURRENT OF clause is used in the UPDATE or DELETE statement.
  - Use the `WHERE CURRENT  OF` clause to reference the current row from an explicit cursor.

```
UPDATE employees
   SET    salary = ...
   WHERE CURRENT OF c_emp_cursor;
```

PL/SQL Cursors - guidelines

- Do not include the INTO clause in the cursor declaration because it appears later in the FETCH statement.
- If processing rows in a specific sequence is required, use the ORDER BY clause in the query.
- The cursor can be any valid SELECT statement, including joins and subqueries.
- If a query returns no rows when the cursor is opened, PL/SQL does not raise an exception. You can find out the number of rows returned with an explicit cursor by using the `<cursor_name>%ROWCOUNT` attribute.
- You should 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.

PL/SQL Cursors - guidelines

- Test to see whether the cursor contains rows. If a fetch acquires no values, there are no rows left to process in the active set and no error is recorded.
- A cursor can be reopened only if it is closed. If you attempt to fetch data from a cursor after it has been closed, then an INVALID_CURSOR exception will be raised.
- 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 up 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.)

Handling Exceptions

Oracle Academy Study Materials

Handling Exceptions

- An exception is a PL/SQL error that is raised during program execution.
- An exception can be raised:
  - Implicitly by the Oracle server
  - Explicitly by the program
- An exception can be handled:
  - By trapping it with a handler
  - By propagating it to the calling environment
- An exception handler is code that defines the recovery actions to be performed when an exception is raised

Handling Exceptions

Exception Types

Exception	Description	Instructions for Handling
Predefined Oracle server error	Most common PL/SQL errors (about 20 that are named)	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly (automatically).
Non-predefined Oracle server error	Other PL/SQL errors (no name)	Declare within the declarative section and allow the Oracle Server to raise them implicitly (automatically).
User-defined error	Defined by the programmer	Declare within the declarative section, and raise explicitly.

Handling Exceptions - Trapping Exceptions

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Handling Exceptions - Trapping Exceptions

Guidelines for Trapping Exceptions

- Begin the exception-handling section of the block with the EXCEPTION keyword.
- Define several exception handlers, each with its own set of actions, for the block.
- Each handler consists of a WHEN clause, which specifies an exception name (exception1, exception 2, etc.), followed by THEN and one or more statements to be executed when that exception is raised (statement1, statement 2, etc.).
- You can include any number of handlers within an EXCEPTION section to handle different exceptions.
- When an exception occurs, PL/SQL processes only one handler before leaving the block.

Handling Exceptions - Trapping Exceptions

Guidelines for Trapping Exceptions

- OTHERS is an optional exception-handling clause that traps any exceptions that have not been explicitly handled.
- Place the OTHERS clause after all other exception-handling clauses.
- You can have only one OTHERS clause.
- Carefully consider whether each exception handler should commit the transaction, roll it back, or let it continue.
- No matter how severe the error is, you want to leave the database in a consistent state and avoid storing any bad data.

Handling Exceptions - Oracle Server Errors

Predefined Oracle server errors:

- Each of these errors has a predefined name, in addition to a standard Oracle error number (ORA-#####) and message.
- For example, if the error ORA-01403 occurs when no rows are retrieved from the database in a SELECT statement, then PL/SQL raises the predefined exception NO_DATA_FOUND.

Non-predefined Oracle server errors:

- Each of these errors has a standard Oracle error number (ORA-#####) and error message, but not a predefined name.
- You declare your own names for these so that you can reference these names in the exception section.

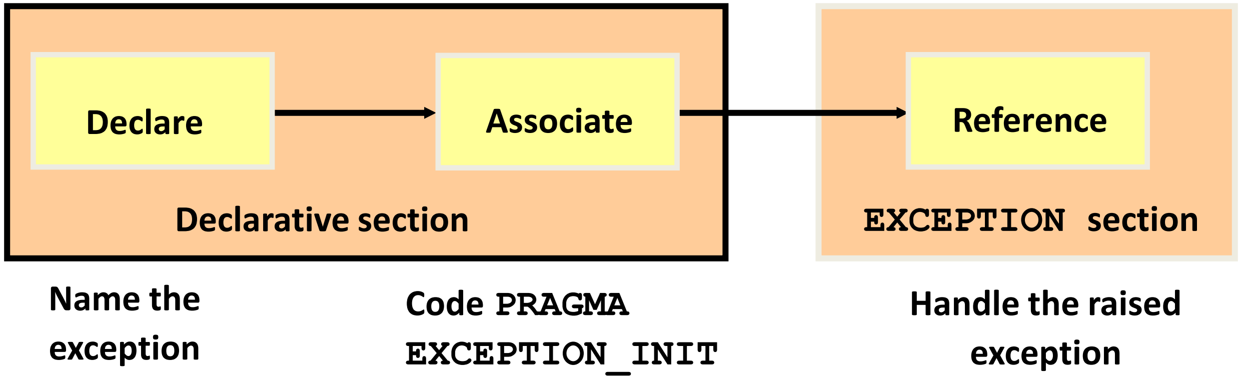
Handling Exceptions - Oracle Server Errors

Exception	Oracle Error	SQLCODE	Description
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.

Handling Exceptions - Oracle Server Errors

Non-predefined Oracle server errors:

- Each of these errors has a standard Oracle error number (ORA-#####) and error message, but not a predefined name.
- You declare your own names for these so that you can reference these names in the exception section.
- You can trap a non-predefined Oracle server error by declaring it first.
- In PL/SQL, the PRAGMA EXCEPTION_INIT tells the compiler to associate an exception name with a specific Oracle error number – this allows you to refer to any Oracle Server exception by a name and to write a specific handler for it.
- The declared exception is raised implicitly.



Non-predefined Oracle server errors

```
DECLARE
        e_insert_excep EXCEPTION;
        PRAGMA EXCEPTION_INIT (e_insert_excep, -01400);
BEGIN
        INSERT INTO departments (department_id, department_name)
               VALUES (280, NULL);
EXCEPTION
        WHEN e_insert_excep THEN
               DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

Handling Exceptions

Functions for Trapping Exceptions

- SQLCODE: Returns the numeric value for the error code
- SQLERRM: Returns the message associated with the error number

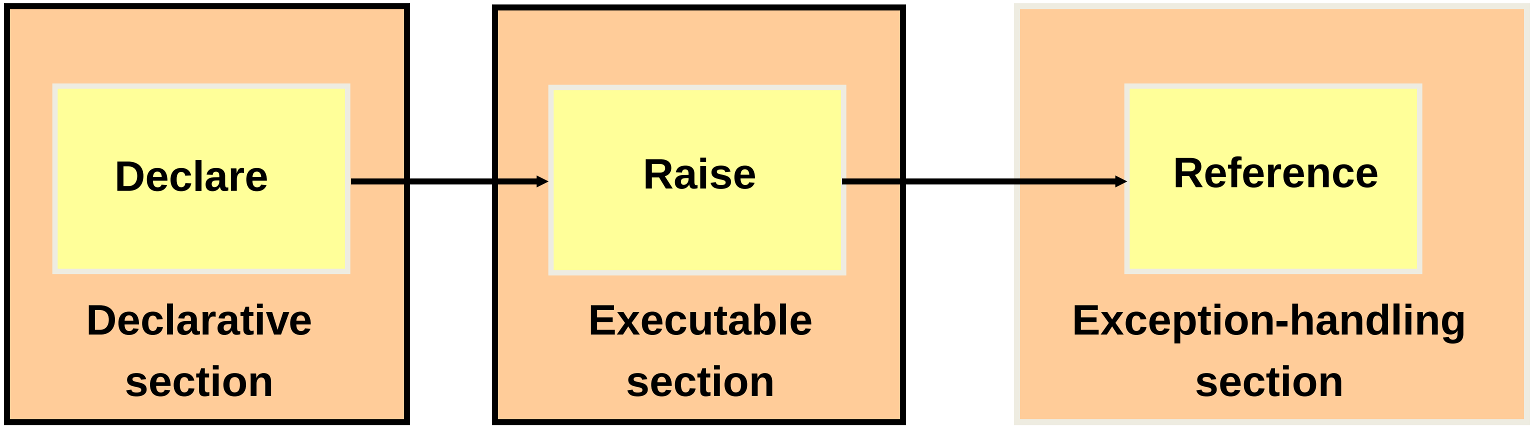
You cannot use SQLCODE or SQLERRM directly in a SQL statement. Instead, you must assign their values to local variables and then use the variables in the SQL statement

SQLCODE Value	Description
0	No exception encountered
1	User defined exception
+100	NO_DATA_FOUND exception
Negative number	Another Oracle Server error number

Handling Exceptions

Trapping User-Defined Exceptions

- PL/SQL allows you to define your own exceptions.
- You define exceptions depending on the requirements of your application.



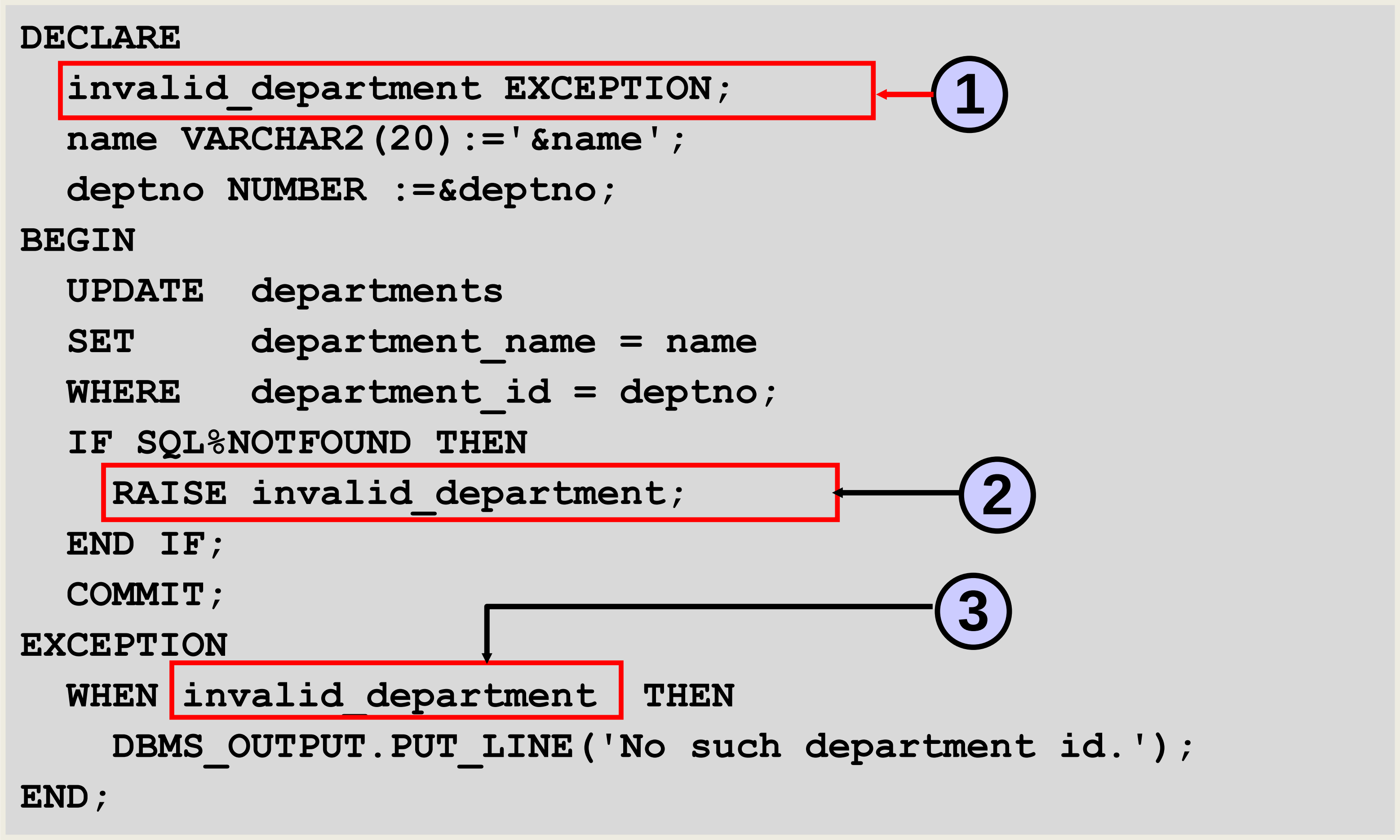
Name the exception.

Explicitly raise the exception by using the `RAISE` statement.

Handle the raised exception.

Handling Exceptions

Trapping User-Defined Exceptions



```
DECLARE
  invalid_department EXCEPTION;
  name VARCHAR2(20):='&name';
  deptno NUMBER :=&deptno;
BEGIN
  UPDATE  departments
  SET     department_name = name
  WHERE   department_id = deptno;
  IF SQL%NOTFOUND THEN
    RAISE invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN invalid_department  THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

Handling Exceptions

The RAISE_APPLICATION_ERROR Procedure

Syntax:

```
raise_application_error (error_number,
                         message[, {TRUE | FALSE}]);
```

You can use this procedure to issue user-defined error messages from stored subprograms.

You can report errors to your application and avoid returning unhandled exceptions.