Exceptions

An exception is an error which disrupts the normal flow of program instructions. PL/SQL provides us the exception block which raises the exception thus helping the programmer to find out the fault and resolve it.

Exceptions are used to handle errors or exceptional conditions that might occur during the execution of a program. They allow you to gracefully manage errors and perform actions based on specific error conditions.

There are three types of exceptions

1. Pre defined exceptions

2. User defined exceptions

3. User named exceptions

SQLERRM: It is the return the error message.

SQLCODE: It is return the error code

1. Pre defined exceptions

Predefined exceptions are a set of standard exceptions provided by programming languages or database systems to handle common error conditions that might occur during program execution. These exceptions are already defined and have specific names and meanings assigned to them. They're generally used to catch and handle errors or exceptional situations without having to explicitly define each one.

NO_DATA_FOUND:

It is predefined exception in PL/SQL used in Oracle databases. It's raised when a query, typically a **SELECT** statement, doesn't return any rows. This exception is automatically raised by Oracle when a **SELECT INTO** statement or a cursor fetch operation fails to retrieve data because the query doesn't find any matching rows.
When this exception occurs, it signifies that the operation expected data but found none. This could be due to various reasons such as incorrect query conditions, no matching records in the database, or a programming error causing the absence of expected data.

Syntax:

```
Exception
When no_data_found then
```

Example:

```
DECLARE
V_EMPNO NUMBER;
V_ENAME VARCHAR2 (240);
BEGIN
SELECT EMPNO,ENAME
INTO   V_EMPNO,V_ENAME
FROM EMP
Where deptno=34;
DBMS_OUTPUT.PUT_LINE(V_EMPNO);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE(SQLERRM||SQLCODE||' NO DATA FOUND');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE(SQLERRM||SQLCODE||' MAIN EXCEPTION');
END;
```

TOO_MANY_ROWS:

It is predefined exception in PL/SQL used within Oracle databases. It's raised when a query, usually a **SELECT INTO** statement or a cursor fetch operation, attempts to retrieve data into a single variable or record, but the query returns multiple rows instead of the expected single row.

This exception occurs when the query is designed to fetch or assign data into a variable or record assuming that only one row will be returned, but the query actually retrieves more than one row. In essence, it indicates a violation of the single-row retrieval expectation.

When encountering this exception, it signals that the query's criteria matched multiple rows, which conflicts with the intended operation that expects only a single result.

Syntax:

```
Exception
When no_data_found then
```

Example:

```
DECLARE
V_EMPNO NUMBER;
V_ENAME VARCHAR2 (240);
BEGIN
SELECT EMPNO, ENAME
```

```
INTO   V_EMPNO, V_ENAME
FROM EMP;
DBMS_OUTPUT.PUT_LINE (V_EMPNO||V_ENAME);
EXCEPTION
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE (SQLERRM||SQLCODE||' TOO MANY ROW ');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE (SQLERRM||SQLCODE||' MAIN EXCEPTION ');
END;
```

ZERO_DIVIDE:

It is predefined exception in PL/SQL used within Oracle databases. It's raised when attempting to perform division by zero, which is an arithmetic operation that's undefined and not permissible in mathematics.

When a program tries to divide a number by zero in PL/SQL, the exception is automatically raised. This situation can occur in various scenarios where a calculation or operation attempts to divide a value by zero.

Syntax:

```
Exception
When zero_divide  then
```

Example:

```
DECLARE
L_A NUMBER:=10;
L_B NUMBER:=2;
L_NUMBER NUMBER;
BEGIN
L_NUMBER:=L_A/L_B;
DBMS_OUTPUT.PUT_LINE ('THE RESULT AFTER DIVISION IS '||L_NUMBER);
EXCEPTION
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE (SQLERRM||SQLCODE||'ZERO DIVIDE');
END;
```

VALUE_ ERROR:

It is predefined exception in PL/SQL used in Oracle databases. It's raised when an error occurs during a conversion or assignment assignment of a value that is incompatible with the declared or variable's range.

This exception typically occurs in situations where there's an attempt to assign a value to a variable, perform a conversion, or execute an operation that results in a value that cannot be accommodated within the defined data type or exceeds its range limits.

Syntax:

```
Exception
When value_error then
```

Example:

```
DECLARE
L_NAME NUMBER;
BEGIN
SELECT ENAME
NAME INTO L_NAME
FROM EMP WHERE ENAME='KING';
DBMS_OUTPUT.PUT_LINE (L_NAME);
EXCEPTION
WHEN VALUE_ERROR THEN
DBMS_OUTPUT.PUT_LINE (SQLERRM||SQLCODE||'VALUE ERROR');
END;
```

CURSOR_ALREADY_OPEN:

The exception typically occurs in database-related programming languages, like PL/SQL, when attempting to open a cursor that is already in an open state. Cursors are used to retrieve and process the results of a query in a database.

When working with cursors, they must be properly managed through their lifecycle, including opening, fetching data, and closing them after use. If a cursor is already open and an attempt is made to open it again without first closing it, the cursor_already_open exception is raised.

Syntax:

```
Exception
When CURSOR_ALREADY_OPEN THEN
```

Example:

```
DECLARE
CURSOR C_1 IS
SELECT EMPNO, ENAME FROM EMP;
L_EMPNO NUMBER;
L_ENAME VARCHAR2 (240);
BEGIN
OPEN C_1; LOOP
FETCH C_1 INTO L_EMPNO,L_ENAME;
DBMS_OUTPUT.PUT_LINE ('EMPLOYEE: '||L_EMPNO||','||L_ENAME);
OPEN C_1;
END LOOP;
CLOSE C_1;
EXCEPTION
WHEN CURSOR_ALREADY_OPEN THEN
DBMS_OUTPUT.PUT_LINE ('CURSOR IS ALREADY OPEN.');
END;
```

INVALID_CURSOR:

It is predefined exception in PL/SQL used in Oracle databases. It's raised when an attempt is made to use a cursor that is in an invalid state or when a cursor is used incorrectly.

Cursors in databases are pointers to result sets from SQL queries. They have a lifecycle - they are typically opened, data is fetched or processed, and then they are closed to release resources. The Invalid_cursor exception occurs when a cursor is not in a state where it can be used for operations like fetching data or closing.

Syntax:

```
Exception
When INVALID_CURSOR THEN
```

Example:

```
DECLARE
CURSOR C_1 IS
SELECT EMPNO, ENAME FROM EMP;
L_EMPNO NUMBER;
L_ENAME VARCHAR2 (240);
BEGIN
OPEN C_1;
```

```
FETCH C_1 INTO L_EMPNO,L_ENAME;
DBMS_OUTPUT.PUT_LINE ('EMPLOYEE: '||L_EMPNO||','||L_ENAME);
CLOSE C_1;
FETCH C_1 INTO L_EMPNO, L_ENAME;
EXCEPTION
WHEN INVALID_CURSOR THEN
DBMS_OUTPUT.PUT_LINE ('INVALID CURSOR.');
END;
```

2. User defined exceptions

User-defined exceptions in programming languages like PL/SQL enable developers to create custom exception conditions to handle specific error scenarios that may not be covered by the predefined exceptions. These exceptions are defined by the programmer based on the requirements of the application.

In PL/SQL, for instance, you can define your own exceptions using the DECLARE block and associate them with specific error conditions or events that might occur during the execution of your code.

Syntax:

```
DECLARE
EXP EXCEPTION;
BEGIN
ANY CONDITION
RAISE EXP:
END;
```

Example:

```
DECLARE
 L_EXCEPTION EXCEPTION;
 N NUMBER: =10;
BEGIN
 FOR I IN 1..N LOOP
DBMS_OUTPUT.PUT_LINE (I*I);
 IF I*I=49 THEN
 RAISE L_EXCEPTION;
 END IF;
 END LOOP;
 END;
```

3. User named exception

A user-named exception in programming languages, such as PL/SQL, refers to an exception that is explicitly named and defined by the developer rather than relying solely on the predefined exceptions provided by the language or system.

In PL/SQL, developers have the capability to create their own named exceptions to handle specific error conditions or scenarios that are unique to their application. These exceptions are given a custom name and can be raised and handled within the code just like the predefined exception.

Syntax:

```
DECLARE
EXP EXCEPTION;
PRAGMA EXCEPTION_INIT (L_EXCEPTION, -20015);
RAISE EXP;
EXCEPTION
WHEN EXP THEN
END;
```

Example:

```
DECLARE
L_EXCEPTION EXCEPTION;
PRAGMA EXCEPTION_INIT (L_EXCEPTION, -20015);
N NUMBER:=10;
BEGIN
FOR I IN 1..N LOOP
DBMS_OUTPUT.PUT_LINE (I*I);
IF I*I=36 THEN
RAISE L_EXCEPTION;
END IF;
END LOOP;
EXCEPTION
WHEN L_EXCEPTION THEN
DBMS_OUTPUT.PUT_LINE ('USER NAMED EXCEPTION');
END;
```