Exception Handling in PL/SQL

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.

There are two types of exceptions defined in PL/SQL

- User defined exception.
- System defined exceptions.

Syntax to write an exception

```
WHEN exception THEN
   statement;

DECLARE

  declarations section;

BEGIN

  executable command(s);
```

```
EXCEPTION

   WHEN exception1 THEN

     statement1;

   WHEN exception2 THEN

     statement2;

   [WHEN others THEN]

   /* default exception handling code */
END;
```

Note:

When other keyword should be used only at the end of the exception handling block as no exception handling part present later will get executed as the control will exit from the block after executing the WHEN OTHERS.

System defined exceptions:

These exceptions are predefined in PL/SQL which get raised WHEN certain database rule is violated.
System-defined exceptions are further divided into two categories:

1. Named system exceptions.
2. Unnamed system exceptions.

Named system exceptions: They have a predefined name by the system like ACCESS_INTO_NULL, DUP_VAL_ON_INDEX, LOGIN_DENIED etc. the list is quite big.

NO_DATA_FOUND: It is raised WHEN a SELECT INTO statement returns *no* rows. For eg

TOO_MANY_ROWS:It is raised WHEN a SELECT INTO statement returns *more* than one row.

VALUE_ERROR:This error is raised WHEN a statement is executed that resulted in an arithmetic, numeric, string, conversion, or constraint error. This error mainly results from programmer error or invalid data input.

ZERO_DIVIDE = raises exception WHEN dividing with zero

Eg

```
declare
a float;
b float;
c float;
begin
a:=&a;
b:=&b;
c:=a/b;
dbms_output.put_line('Divide'||c);
exception
when ZERO_DIVIDE then
dbms_output.put_line('the value of b should not be zero');
end;
```

/

DUP_VAL_ON_INDEX=raises exception when we try to insert duplicate value if it has key constraints.

Unnamed system exceptions: Oracle doesn't provide name for some system exceptions called unnamed system exceptions.These exceptions *don't* occur frequently.These exceptions have two parts *code and an associated message*.
The way to handle to these exceptions is to *assign name* to them using Pragma EXCEPTION_INIT
Syntax:

PRAGMA EXCEPTION_INIT(exception_name, -error_number);

error_number are pre-defined and have negative integer range from -20000 to -20999.

Example:

```
DECLARE
   exp exception;
   pragma exception_init (exp, -20015);
   n int:=10;

BEGIN
   FOR i IN 1..n LOOP
      dbms_output.put_line(i*i);
         IF i*i=36 THEN
            RAISE exp;
         END IF;
   END LOOP;

EXCEPTION
```

```
  WHEN exp THEN
    dbms_output.put_line('Welcome to GeeksforGeeks');

END;
```

Cursor:-

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. The SQL cursor has additional attributes, %BULK_ROWCOUNT and %BULK_EXCEPTIONS, designed for use with the FORALL statement. The following table provides the description of the most used attributes −

S.No Attribute & Description

	%FOUND	
1	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.	
	%NOTFOUND	
2	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.	
	%ISOPEN	
3	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.	
	%ROWCOUNT	
4	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.	

Any SQL cursor attribute will be accessed as sql%attribute_name as shown below in the example.

Example

We will be using the CUSTOMERS table we had created and used in the previous chapters.

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
```

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

+----+----------+-----+-----------+----------+

The following program will update the table and increase the salary of each customer by 500 and use the SQL%ROWCOUNT attribute to determine the number of rows affected −

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
6 customers selected


PL/SQL procedure successfully completed.
```

If you check the records in customers table, you will find that the rows have been updated −

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```


Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
```

```
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

1 Ramesh Ahmedabad

2 Khilan Delhi

3 kaushik Kota

4 Chaitali Mumbai

5 Hardik Bhopal

6 Komal MP

PL/SQL procedure successfully completed.

PL/SQL Triggers

PL/SQL stands for Procedural Language/ Structured Query Language. It has block structure programming features.PL/SQL supports SQL queries. It also supports the declaration of the variables, control statements, Functions, Records, Cursor, Procedure, and Triggers.PL/SQL

contains a declaration section, execution section, and exception-handling section. Declare and exception handling sections are optional.

Syntax:

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

PL/SQL Triggers

PL/SQL triggers are block structures and predefined programs invoked automatically when some event occurs. They are stored in the database and invoked repeatedly in a particular scenario. There are two states of the triggers, they are enabled and disabled. When the trigger is created it is enabled. CREATE TRIGGER statement creates a trigger. A triggering event is specified on a table, a view, a schema, or a database.BEFORE and AFTER are the trigger Timing points.DML triggers are created on a table or view, and triggers.

Why are Triggers important?

The importance of Triggers are:

- Automated Action: It helps to automate actions in response to events on table or views.
- Data integrity: Constraint can be applied to the data with the help of trigger. It is used to ensure referential integrity.

- Consistency: It helps to maintain the consistency of the database by performing immediate responses to specific events.
- Error handling: It helps in error handling by responding to the errors. For example, If specific condition is not met it will provide an error message.

PL/SQL Trigger Structure

Triggers are fired on the tables or views which are in the database. Either table, view ,schema, or a database are the basic requirement to execute a trigger. The trigger is specified first and then the action statement are specified later.

Syntax:

CREATE OR REPLACE TRIGGER trigger_name

BEFORE or AFTER or INSTEAD OF //trigger timings

INSERT or UPDATE or DELETE // Operation to be performed

of column_name

on Table_name

FOR EACH ROW

DECLARE

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

/

Query operation to be performed i.e INSERT,DELETE,UPDATE.

- CREATE [OR REPLACE] TRIGGER trigger_name is used to create a trigger or replace the existing trigger.|
- BEFORE | AFTER | INSTEAD OF specifies trigger timing.
- INSERT | UPDATE | DELETE are the DML operations performed on table or views.
- OF column_name specifies the column that would be updated.
- ON table_name species the table for the operation.
- FOR EACH ROW specify that trigger is executed on each row .

Types of PL/SQL Triggers

1. Row Trigger
2. Statement Trigger
3. Before V/s After Trigger

Enabling and Disabling Triggers:-

Database triggers are procedures that are stored in the database and activated ("fired") when specific conditions occur, such as adding a row to a table. You can use triggers to supplement the standard capabilities of the database to provide a highly customized database management system. For example, you can create a trigger to restrict DML operations against a table, allowing only statements issued during regular business hours.

Database triggers can be associated with a table, schema, or database. They are implicitly fired when:

- DML statements are executed (INSERT, UPDATE, DELETE) against an associated table
- Certain DDL statements are executed (for example: ALTER, CREATE, DROP) on objects within a database or schema
- A specified database event occurs (for example: STARTUP, SHUTDOWN, SERVERERROR)

Create triggers with the CREATE TRIGGER statement. They can be defined as firing BEFORE or AFTER the triggering event, or INSTEAD OF it. The following statement creates a trigger scott.emp_permit_changes on table scott.emp. The trigger fires before any of the specified statements are executed.

```
CREATE TRIGGER scott.emp_permit_changes
    BEFORE
    DELETE OR INSERT OR UPDATE
    ON scott.emp
    .
    .
```

.

pl/sql block

.

.

.

You can later remove a trigger from the database by issuing the DROP TRIGGER statement.

trigger can be in either of two distinct modes:

- Enabled

An enabled trigger executes its trigger body if a triggering statement is issued and the trigger restriction, if any, evaluates to true. By default, triggers are enabled when first created.

- Disabled

A disabled trigger does not execute its trigger body, even if a triggering statement is issued and the trigger restriction (if any) evaluates to true.

To enable or disable triggers using the ALTER TABLE statement, you must own the table, have the ALTER object privilege for the table, or have the ALTER ANY TABLE system privilege. To enable or disable an individual trigger using the ALTER TRIGGER statement, you must own the trigger or have the ALTER ANY TRIGGER system privilege.

Enabling Triggers

You enable a disabled trigger using the ALTER TRIGGER statement with the ENABLE option. To enable the disabled trigger named reorder on the inventory table, enter the following statement:

ALTER TRIGGER reorder ENABLE;

To enable all triggers defined for a specific table, use the ALTER TABLE statement with the ENABLE ALL TRIGGERS option. To enable all triggers defined for the INVENTORY table, enter the following statement:

ALTER TABLE inventory

ENABLE ALL TRIGGERS;

Disabling Triggers

Consider temporarily disabling a trigger if one of the following conditions is true:

- An object that the trigger references is not available.
- You must perform a large data load and want it to proceed quickly without firing triggers.
- You are loading data into the table to which the trigger applies.

You disable a trigger using the ALTER TRIGGER statement with the DISABLE option. To disable the trigger reorder on the inventory table, enter the following statement:

ALTER TRIGGER reorder DISABLE;

You can disable all triggers associated with a table at the same time using the ALTER TABLE statement with the DISABLE ALL TRIGGERS option. For example, to disable all triggers defined for the inventory table, enter the following statement:

ALTER TABLE inventory

DISABLE ALL TRIGGERS;

What is a PL/SQL Index?

An index in PL/SQL is a database object designed for instant access to data rows from the database. It functions the same as an index of a book, enabling easy navigation to specific information without wasting the time to scan the entire database for specific data.

Types of PL/SQL Indexes

Indexes can be classified into several types, each serving different purposes to optimize database performance:

- Single Column Index: Created on a single column of a table
- Composite Index: An index on multiple columns
- Unique Index: Ensures that the indexed column has unique values.
- Clustered Index: Arranges the data rows in the table according to the index order.

How to Create an Index in PL/SQL

In procedural language/structured query language the syntax of creating the index is as follows

Syntax:

CREATE INDEX Index_Name ON Table_Name (Column_Name);

Here,

- Index_Name: This is just the name we give to the index we want to create.
- Table_Name: This refers to the name of the table in a database where you want to create the index.
- Column_Name: This is the specific column within the table where you want to create the index.

- You can also create composite indexes by specifying multiple columns.
- Composite Index Syntax:
- CREATE INDEX index_name ON table_name (column1, column2, ...);
- Examples of Index in PL/SQL
- Example 1: Creating, Displaying, and Deleting and Managing Index in PL/SQL
- Let's create an index on the Students table to improve the performance of queries that filter by Stud_ID:

Stud_ID	Stud_Name	Stud_Gender	Stud_Class	Stud_City	Stud_State
2001	Sahil	M	5TH	Pune	Maharashtra
2002	Kunal	M	5TH	Pune	Maharashtra
2003	Purva	F	10TH	Mumbai	Maharashtra
2004	Rahul	M	9TH	Karmala	Maharashtra

- Creating Index: We will create the index with name idx_stud_ID on the column Stud_ID of table students of the database.
- CREATE INDEX idx_Stud_ID ON Students(Stud_ID);
- Output:

```
mysql> select * from students;
+---------+-----------+-------------+------------+-----------+-------------+
| Stud_ID | Stud_Name | Stud_Gender | Stud_Class | Stud_City | Stud_State  |
+---------+-----------+-------------+------------+-----------+-------------+
|    2001 | Sahil     | M           | 5TH        | Pune      | Maharashtra |
|    2002 | Kunal     | M           | 5TH        | Pune      | Maharashtra |
|    2003 | Purva     | F           | 10TH       | Mumbai    | Maharashtra |
|    2004 | Rahul     | M           | 9TH        | Karmala   | Maharashtra |
+---------+-----------+-------------+------------+-----------+-------------+
4 rows in set (0.00 sec)

mysql> CREATE INDEX idx_Stud_ID ON Students(Stud_ID);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- Creating Index on students table

- Display Created Index from Database: We can see the created Index by using following query in PL/SQL.
- SHOW INDEX FROM Students;
- Output:

```
mysql> SHOW INDEX FROM Students;
+----------+------------+------------+--------------+-------------+-----------+
| Table    | Non_unique | Key_name   | Seq_in_index | Column_name | Collation |
+----------+------------+------------+--------------+-------------+-----------+
| students |          0 | PRIMARY    |            1 | Stud_ID     | A         |
| students |          1 | idx_Stud_ID |            1 | Stud_ID     | A         |
+----------+------------+------------+--------------+-------------+-----------+
2 rows in set (0.02 sec)

mysql>
```

- Display the created index

- Deleting the Created Index: Sometimes user need to delete the index, So index can be deleted/dropped using following PL/SQL query.
- DROP INDEX idx_Stud_ID ON Students;
- Output:

Explanation: In this example, we show how to create, view, and delete an index called idx_stud_ID on Stud_ID column in the “Students” table of a database. The following PL/SQL queries show how to improve database performance by index management, and provide insight into how indexing works in Oracle Database.

Example 2: Index Management in the Employee Table

Consider the following Student Table of the database:

Emp_ID	Emp_Name	Emp_Gender	Emp_Department	Emp_City	Emp_State
1001	Sahil	M	IT	Nagpur	Maharashtra
1002	Sam	F	HR	Barshi	Maharashtra
1003	Mia	M	Finance	Baramati	Maharashtra
1004	Ajay	F	Marketing	Pune	Maharashtra

Creating Index: This query will create an index named idx_Emp_ID on the Emp_ID column of the Employee table.

CREATE INDEX idx_Emp_ID ON Employee(Emp_ID);

Output:

```
mysql> CREATE INDEX idx_Emp_ID ON Employee(Emp_ID);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Ex. 2 Creating Index in Employee DB

Deleting the Created Index: Sometimes user need to delete the index, So index can be deleted/dropped using following PL/SQL query.

DROP INDEX idx_Emp_ID ON Employee;

Output:

```
mysql> DROP INDEX idx_Emp_ID ON Employee;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Ex.2 Dropping the Created index

Explanation: In the following example, we show how to create and delete an index called idx_Empire_ID on Emp_ID in the "Employee" column of a database. These PL/SQL queries illustrate how to optimize database performance through effective index management, and provide practical insights into how indexing works in Oracle Database.

Advantages of Using Indexes in PL/SQL

1. Improves Query Performance: Indexes helps the database to find the rows that matches the given requirement/query, which can significantly improve the performance of search queries.
2. Easier Data Maintenance: Indexes helps the database to maintain specific type of data in databases such as sorted and unique data values.
3. Improve Concurrency: Indexes improve the concurrency of databases as they reduce the amount of locking and blocking that occurs while multiple transactions request/access the same data.

Disadvantages of Using Indexes in PL/SQL

1. Increases Storage of Database: Indexes increases the storage overhead of database as they requires the additional storage to save the indexes data. This additional data results into increase in storage of the database.
2. Increases Maintenance of Database: Indexes need to be regularly checked and updated to make sure they work well with the queries used in a database. This extra work results into the increase in Maintenance of database.
3. Increased query complexity: Sometimes Indexes can make the Query execution complex due to the considering the multiple execution plans and choosing shortest one from database to achieve the result.

@%oracle_home%\sqlplus\admin\plustrce.sql;