Date:

Page:

Chapter / Unit - 7 (last Unit)

Handling PL/SQL Errors

Run-time Error Handling in PL/SQL

Every PL/SQL block executed at client-side may generate some run-time errors called exception.

To recognize and handle those exception ORACLE engine provides exception condition and exception handler.

An exception handler is a code block in memory that will attempt to resolve the current exception condition. The ORACLE engine can recognize every exception condition that occurs in memory.

To handle very common and repeatitative exception conditions the ORACLE engine uses pre-defined PL/SQL exceptions also known as Named Exception Handler.

Date:
Page:

Named Exception Handler:-

The ORACLE engine has a set of pre-defined ORACLE error handlers known as Named Exception. These Error Handlers are referenced by their names. The ORACLE engine has near about 15 to 20 named exception handler. In addition to this ORACLE engine identifies those exception handlers by 4 digits integer numbers precedded by a dash (–), like (–1414)

Few commonly used pre-defined exceptions are:-

- DUP_VAL_ON_INDEX:
- LOGIN_DENIED
- NO_DATA-FOUND
- NOT_LOGGED_ON
- PROGRAM-ERROR
- TIMEOUT-ON-RESOURCE
- TOO-MANY_ROWS
- VALUE_ERROR

- DUP_VAL_ON-INDEX:

It is raised when an INSERTOR UPDATE attempt to create two or more records with the duplicate values in columns restricted by PRIMARY KEY or UNIQUE Constraints.

Date:

Page:

- LOGIN_DENIED:

It is raised when an invalid username or password is used to login.

Scanned with CamScanner

PL/SQL

Date: ______

Page: ______

Continued ...

NO-DATA-FOUND: It is raised when a SQL statement like a SELECT statement returns zero records.

NOT_LOGGED_ON: It is raised when PL/SQL issues a ORACLE call without being logged in.

PROGRAM_ERROR: It is raised when PL/SQL has an internal problem.

TIMEOUT_ON_RESOURCE: It is raised when an ORACLE has been waiting to access the resource beyond the user defined time out limit.

TOO_MANY_ROWS:- It is raised when a SELECT statement returns more than ~~0~~ one rows.

VALUE_ERROR:- It is raised when a datatype or size is invalid.

OTHERS:- It indicates all other exceptions not explictly named.

Date: ________

Page: ________

Syntax:

```
EXCEPTION
        WHEN Exception Name THEN
            user-defined-action-to-be-carried-out;
```

Fer example:-

Consider the Emp table and write a PL/SQL block that takes employee number as input and find the information of that employee. If the user enters the employee number that is not in the emp table then display an appropriate message using Exception.

Emp (EmpNo, EName, ...)

```
DECLARE
    eno  Emp.EmpNo % TYPE;
BEGIN
      emp := :emp;
    SELECT * FROM Emp
      WHERE EmpNo = eno;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
     dbms_output.put_line ('EmpNo' || eno ||
                       'does not exist');
```

Date: ______

Page: ______

END;

User Named Exception Handler:

The technique that is used to bind a numbered exception handler a user-defined name using the statement
PRAGMA EXCEPTION_INIT()
is known as User Named Exception.
The PRAGMA keyword is used to call a pre compiler, to bind a numbered exception to the named exception.
The function ~~exep~~ EXCEPTION-INIT( ) takes two parameters. The first parameter is the user-defined exception name and a second parameter is the ORACLE engine exception number. This function must be included within the DECLARE section and used in EXCEPTION section.

Continued in next page:-

PL/SQL

Date:

Page:

Continued:-

SYNTAX

```
DECLARE
    ------
    exceptionName EXCEPTION;
    PRAGMA EXCEPTION_INIT(exceptionName,
                          exception_code);
BEGIN
    ------
    ------
EXCEPTION
    WHEN exceptionName THEN
        Action;
END;
```

for example:

Write a PL/SQL block that accepts employee number and find the salary of that employee number and increase the salary by 2000. Raise an exception -0054 with a user defined name to indicate that the resource is busy.

```
DECLARE
    eno emp.empno % TYPE;
    sal emp. salary % TYPE;
    RESOURCE_BUSY EXCEPTION;
    PRAGMA EXCEPTION_INIT (RESOURCE_
                BUSY, -0054);
BEGIN
    eno :=: eno;
    SELECT Salary INTO Sal FROM emp
    WHERE empno = ~~em~~ eno;
    sal := sal + 2000;
    UPDATE emp SET salary = Sal
    WHERE emp = eno;
EXCEPTION
    WHEN RESOURCE_BUSY THEN
    dbms_output.put_line ('The Row is in
                        Use');
    END;
```

---

lock Acquire/Release.

Read (Shared) / Write (Exclusive) → Locking T ⟶

} Mostly Server Busy for case (H1)

Date:

Page:

(a) User defined Exception Handler :-

We can create user defined exceptions for business rules validity. If the data provided by user violates the business rules, the entire record must be rejected.

User defined exception conditions must be declared in the declarative part of PL/SQL block. In the executable part, condition is checked for raising the exception. If the condition is satisfied, the user defined exception is raised by using a ~~raised~~ RAISE statement. The exception once raised is then handled in the exception handling section.

PL/SQL

Continued:-

Syntax:

```
    DECLARE
        exceptionName  EXCEPTION;
        --------
    BEGIN
        ------
        ----
      IF Condition THEN
         RAISE exceptionName;
      ENDIF;
    EXCEPTION
       WHEN exceptionName THEN
          user-defined-action;
    END;
```

Eg:-
Write a PL/SQL Block to take account number and amount as input to withdraw and raise an exception if the amount is greater then the balance in account (AccNo, AcName, AcType, Balance)

Date:

Page:

```
DECLARE
    acno Account.AccNo %TYPE;
    amt FLOAT; bal Account.Balance %TYPE;
  NOT_SUFFICIENT_BALANCE EXCEPTION;
  BEGIN
      acno:=:acno;
      amt:=:amt;
      SELECT balance INTO bal FROM Account
      WHERE Accno=acno;
 IF amt< bal THEN
     UPDATE Account SET balance =
                balance-amt WHERE Accno=acno;
   ELSE
     RAISE NOT_SUFFICIENT_BALANCE;
  END IF;
EXCEPTION
    WHEN NOT_SUFFICIENT_BALANCE THEN
    dbms_output.put_line('Your Balance is
                                      low');
END;
```