EXCEPTIONS

Errors

- Two types of errors can be found in a program: compilation errors and runtime errors.
- There is a special section in a PL/SQL block that handles the runtime errors.
- This section is called the *exception-handling section*, and in it, runtime errors are referred to as *exceptions*.
- The exception-handling section allows programmers to specify what actions should be taken when a specific exception occurs.

Exception Handling

- In order to handle run time errors in the program, an exception handler must be added.
- The exception-handling section has the following structure:

```
EXCEPTION
    WHEN EXCEPTION_NAME
    THEN
        ERROR-PROCESSING STATEMENTS;
```

- The exception-handling section is placed after the executable section of the block.

Predetermined Internal PL/SQL Exceptions(Built –in Exceptions)

1. **DUP_VAL_ON_INDEX**: Raised when an insert or update attempts to create two rows with duplicate values in columns constrained by a unique index.

2. **LOGIN_DENIED**: Raised when an invalid username/password was used to log onto Oracle.

3. **NO_DATA_FOUND**: Raised when a select statement returns zero rows.

4. **NOT_LOGGED_ON**:Raised when PL/SQL issues an oracle call without being logged onto Oracle.

5. **PROGRAM_ERROR**: Raised when PL/SQL has an internal problem.

Predetermined Internal PL/SQL Exceptions(Built –in Exceptions)

6. **TIMEOUT_ON_RESOURCE**: Raised when Oracle has been waiting to access a resource beyond the user-defined timeout limit.

7. **TOO_MANY_ROWS**: Raised when a select statement returns more than one row.

8. **VALUE_ERROR**: Raised when the data type or data size is invalid.

9. **OTHERS**: stands for all other exceptions not explicitly named

10.ZERO DIVIDE:Raised when number is divided by zero

```
DECLARE
  v_cointosses integer := &v_cointosses;
  v_Head integer := &v_Head;
  v_Prob number;
BEGIN
  v_Prob := v_Head / v_cointosses;
  DBMS_OUTPUT.PUT_LINE ('Probablity is : '|| v_Prob);
EXCEPTION
  WHEN ZERO_DIVIDE   THEN
        DBMS_OUTPUT.PUT_LINE
       ('number of Cointosses can''t be  zero.');
END;
/
```

Exp1.sql

```
DECLARE
   v_empno  emp.empno%TYPE;
   v_ename  emp.ename%TYPE;
   v_salary  emp.salary%TYPE;
BEGIN
   v_empno:=&v_empno;
   SELECT ename,salary INTO v_ename,v_salary FROM emp
   WHERE empno=v_empno;
   DBMS_OUTPUT.PUT_LINE(v_ename||' draws '||v_salary||'
    as salary');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('NO such employee found');
END;
/
```

Exp2.sql

User Defined Exceptions

- For example, your program asks a user to enter a value for emp_id. This value is then assigned to the variable v_empid that is used later in the program.
- Generally, you want a positive number for an emp_id. By mistake, the user enters a negative number.
- However, no error has occurred because emp_id has been defined as a number, and the user has supplied a legitimate numeric value.
- Therefore, you may want to implement your own exception to handle this situation.

User Defined Exceptions

- This type of an exception is called a *user-defined exception* because it is defined by the programmer.
- Before the exception can be used, it must be declared.
- A user-defined exception is declared in the declarative part of a PL/SQL block as shown below:

DECLARE

***exception_name* EXCEPTION;**

- Once an exception has been declared, the executable statements associated with this exception are specified in the exception-handling section of the block.
- The format of the exception-handling section is the same as for built-in exceptions.

```
DECLARE
   exception_name EXCEPTION;
BEGIN
  IF CONDITION   THEN
       RAISE exception_name;
   ELSE
        …
  END IF;
EXCEPTION
     WHEN exception_name  THEN
          ERROR-PROCESSING STATEMENTS…..
END;
```

```
DECLARE
  v_empno  emp.empno%TYPE;
  v_ename  emp.ename%TYPE;
  v_salary  emp.sal%TYPE;
  ex_invalid_id EXCEPTION;
BEGIN
 v_empno:=&v_empno;
 IF v_empno <= 0 THEN
     RAISE ex_invalid_id;
ELSE
  SELECT ename,sal INTO v_ename,v_salary FROM emp WHERE
  empno=v_empno;
  DBMS_OUTPUT.PUT_LINE(v_ename||' draws '||v_salary||' as
  salary');
END IF;
```

Exp5.sql

```
EXCEPTION
-- user defined
WHEN ex_invalid_id THEN
  DBMS_OUTPUT.PUT_LINE('Emp no must be greater than zero’);
-- System defined Named Exception
WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('NO such employee found');
  END;
/
```

Exp5.sql

Using OTHERS Exception

OTHERS exception takes care of any other exception raised apart from the defined exceptions in the

Example: In the following example, user accepts employee number and using select statement we try to find salary of that employee. If entered employee number do not exists, first system check for is there any WHEN NO_DATA_FOUND THEN, i.e. exception handler for NO_DATA_FOUND system exception. If WHEN NO_DATA_FOUND THEN is not found then system looks for OTHERS. If OTHERS is also not there in PL-SQL block then System handles by displaying **No Rows Found System defined** error message.

Example: OTHERS

```
DECLARE
    ENO EMP.EMPNO%TYPE;
    salary emp.sal%type;
BEGIN
    ENO:=&ENO;
    SELECT SAL INTO SALARY FROM EMP WHERE
    EMPNO=ENO;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('Some Error occurred ...');
END;
/
```

Using System Defined Numbered Exception

Assume that following table is created with check constraint on supplier_id.

```
CREATE TABLE suppliers
( supplier_id number(4),  supplier_name varchar2(50),
CONSTRAINT check_supplier_id
CHECK (supplier_id BETWEEN 100 and 999)
);
```

Whenever check constraint on supplier_id is violated **ORA -02290 exception number is raised**, we can associate an Exception name to this error number ORA -02290 and use in a PL SQL block to handle this exception.

```
DECLARE
    Supplier_ID_Range EXCEPTION;
    pragma EXCEPTION_INIT(Supplier_ID_Range,-02290);
BEGIN
    INSERT INTO suppliers ( supplier_id, supplier_name )
                    VALUES ( 1, 'IBM' );
EXCEPTION
 WHEN Supplier_ID_Range THEN
    DBMS_OUTPUT.PUT_LINE(' Supplier ID entered must be in
    the range 100 t0 9999');
END:
/
```

```
-- In Built Error messages with Error Number
DECLARE
  MONTH_Range exception;
  pragma EXCEPTION_INIT(MONTH_Range,-01843);

BEGIN
  insert into emp1(empno, Bdate) vALUES(9007,TO_DATE('10-41-2018','DD-MM-YYYY'));
  EXCEPTION
   WHEN MONTH_Range THEN
      dbms_output.put_line('Valid values for Month is 1 to 12');
END;
/
```

Example

```
--outer block
DECLARE
    e_exception1 EXCEPTION;
    e_exception2 EXCEPTION;
BEGIN
    -- inner block
    BEGIN
        RAISE e_exception1;
    EXCEPTION
    WHEN e_exception1
    THEN
        RAISE e_exception2;
```

Example contd.

```
    WHEN e_exception2
    THEN
        DBMS_OUTPUT.PUT_LINE ('An error has occurred in the    inner'|| 'block');
    END;
EXCEPTION
WHEN e_exception2
THEN
    DBMS_OUTPUT.PUT_LINE ('An error has occurred in the program');
END;
```

Example

Write a PL/SQL block to accept, Principle, Interest rate and duration (in years) to calculate Interest to be paid. Handle the exceptions if Principle <=1000, interest rate <5, year <1 and display proper error message for each.