Jump2Learn
PUBLICATION

www.jump2learn.com

CONCEPTS of RELATIONAL DATABASE MANAGEMENT SYSTEM

Jump2Learn - The Online Learning Place

Dr. RajeshKumar R. Savaliya | Ms. Sonal B. Shah | Mr. Vaibhav D. Desai

Unit – 5

Cursors and Exception Handling

In this unit, we are going to understand Cursors and Exception Handling features provided by Oracle PL/SQL.

Let us first take a sample database, using which we will understand both the topics in detail in subsequent sections. We have taken a simple scenario of student database like which course he is studying in, what all are the subjects he is studying, what result he is getting. It is assumed here that you all are well aware of table creation, which are having integrity constraints like primary key, foreign key etc. Anyway, table creation scripts along with the sample data is given below.

We have basically taken example of four tables and the data dictionary of these tables is as follows:

1. Table: COURSE

This table is having all the courses offered by a college.

Column	Datatype	Length	Constraint	Description
COURSE_ID	NUMBER	1	Primary Key	Course ID
COUIRSE_NAME	VARCHAR2	20	NOT NULL, CHECK	Courses a student can opt for UG degree (BCOM/BBA/BCA)

Table Creation Script:

```
CREATE TABLE COURSE
(
 COURSE_ID   NUMBER(1),
 COURSE_NAME VARCHAR2(20) NOT NULL
 CHECK ( COURSE_NAME IN ('BBA','BCA','BCOM')
),
CONSTRAINT COURSE_PK PRIMARY KEY ( COURSE_ID ) )
/
```


Data:

Course ID	Course Name
1	BCOM
2	BCA
3	BBA

2. Table: STUDENTS

This table is having details of all the students.

Column	Datatype	Length	Constraint	Description
STUDENT_SPID	NUMBER	5	Primary Key	Students Personal ID allotted by the University
ROLLNO	NUMBER	3	Not Null	Roll number of the student and (Roll Number + Course ID) constitutes a composite unique key
STUDENT_NAME	VARCHAR2	255	Not Null	Name of the student
COURSE_ID	NUMBER	1	Foreign Key, Not Null	Course ID and (Roll Number + Course ID) constitutes a composite unique key
BIRTH_DATE	DATE	-	Not Null	Birth Date

Table Creation Script:

```
CREATE TABLE STUDENTS
(
 STUDENT_SPID          NUMBER(5),
 ROLLNO                NUMBER(3)    NOT NULL,
 STUDENT_NAME          VARCHAR2(255)  NOT NULL,
 COURSE_ID             NUMBER(1) NOT NULL,
 BIRTH_DATE            DATE   NOT NULL,
 CONSTRAINT STUDENTS_PK PRIMARY KEY ( STUDENT_SPID ) ,
 CONSTRAINT STUDENTS_FK1 FOREIGN KEY (COURSE_ID) REFERENCES COURSE,
 CONSTRAINT STUDENTS_UK1 UNIQUE ( ROLLNO, COURSE_ID )
)
/
```

Data:

STUDENT_SPID	ROLLNO	STUDENT_NAME	COURSE_ID	BIRTH_DATE
20211	1	VIMAL	1	01-JAN-80
20212	2	NIDHI	1	01-FEB-80
20213	3	NEHAL	1	01-MAR-80
20214	1	NAINESH	2	01-APR-80
20215	2	EEVA	2	01-MAY-80
20216	3	KRISHNA	2	01-JUN-80
20217	1	AAKANKSHA	3	01-JUL-80
20218	2	JAIMINI	3	01-AUG-80
20219	3	CHIRAG	3	01-SEP-80

3. Table: SUBJECTS

This table is having information about the subjects of a course.

Column	Datatype	Length	Constraint	Description
SUBJECT_ID	NUMBER	3	Primary Key	Subject ID
SUBJECT_NAME	VARCHAR2	20	NOT NULL	Subject Name
COURSE_ID	NUMBER	1	NOT NULL, Foreign Key	Course ID

Table Creation Script:

```
CREATE TABLE SUBJECTS
( SUBJECT_ID          NUMBER(3),
  SUBJECT_NAME        VARCHAR2(20)        NOT NULL,
  COURSE_ID           NUMBER(1)           NOT NULL,
  CONSTRAINT SUBJECTS_PK PRIMARY KEY ( SUBJECT_ID ) ,
  CONSTRAINT SUBJECTS_FK1 FOREIGN KEY (COURSE_ID) REFERENCES COURSE
 )
/
```

Data:

SUBJECT_ID	SUBJECT_NAME	COURSE_ID
101	ACCOUNTANCY	1
102	ECONOMICS	1
201	CPPM	2
202	RDBMS	2
301	MANAGEMENT	3
302	HRM	3

4. Table: RESULT

This table has details of marks scored in each subject by a student.

Column	Datatype	Length	Constraint	Description
STUDENT_SPID	NUMBER	5	NOT NULL, FOREIGN KEY	Students Personal ID allotted by the University
SUBJECT_ID	NUMBER	3	NOT NULL, FOREIGN KEY	Subject ID
MARKS_SCORED	NUMBER	4,1	NOT NULL, CHECK	Marks scored by a student in a subject

Table Creation Script:

```
CREATE TABLE RESULT
( STUDENT_SPID      NUMBER(5)       NOT NULL,
  SUBJECT_ID        NUMBER(3)       NOT NULL,
  MARKS_SCORED      NUMBER(4,1)     NOT NULL
                    CHECK ( MARKS_SCORED BETWEEN 0 AND 100 ),
 CONSTRAINT RESULT_FK1 FOREIGN KEY (STUDENT_SPID) REFERENCES STUDENTS,
CONSTRAINT RESULT_FK2 FOREIGN KEY (SUBJECT_ID) REFERENCES SUBJECTS
)
/
```

Data:

STUDENT_SPID	SUBJECT_ID	MARKS_SCORED
20211	101	75
20211	102	67
20212	101	81
20212	102	61
20213	101	31
20213	102	47
20214	201	94
20214	202	75
20215	201	25
20215	202	29
20216	201	45
20216	202	56
20217	301	52
20217	302	65
20218	301	93
20218	302	88
20219	301	75
20219	302	69

5.1 Concept of Cursors

When any SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. 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.

In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it. Using cursor, a program can fetch and process the rows returned by the SQL statement. These rows can be processed one at a time.

In other words, A cursor is a pointer that points to a result of a query. The resultant rows of a query can be processed one at a time.

5.1.1 Types of Cursor

There are two types of cursors:

1. Implicit cursors
2. Explicit cursors

1. Implicit Cursors

As the name suggests, the implicit cursors are automatically generated by Oracle while an SQL statement is processed or executed. They are created automatically, if you don't use an explicit cursor for the statement.

Implicit cursors are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed. 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.

Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

2. Explicit Cursors

An explicit cursor is a SELECT statement declared in declaration section of a PL/SQL block. An explicit cursors are defined to gain more control over the context area created by the cursors. Just like variables and constants, the cursors are also need to be defined in declaration area. The explicit cursor is created based on a SELECT statement which returns one or more rows.

Following is the syntax to create an explicit cursor:

```
CURSOR cursor_name IS select_statement;
```

There are specific steps which need to be followed to use features of explicit cursors.

5.1.2 Declare, open, fetch and close cursors.

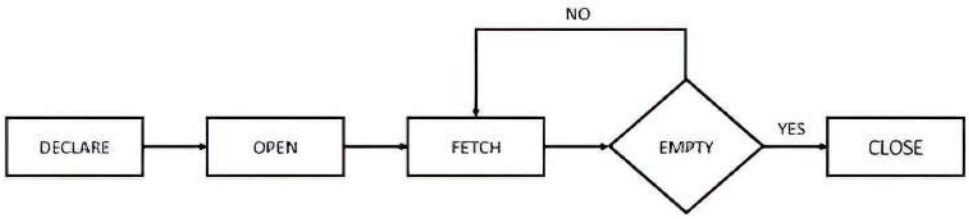

Fig 5.1 Execution Cycle of Explicit Cursor

Above figure explains the execution cycle for explicit cursors. It is having following main steps.

1. Declaring the cursor

This step causes the initialization of the memory. The cursor is declared using user defined name and a select statement associated with it. The cursor is declared in the declaration section of a PL/SQL block. The syntax to declare a cursor is already discussed earlier.

2. Opening the cursor

Opening a cursor causes allocation of the memory by executing the associated SQL and storing the resultant rows in the memory. The cursor is generally opened before starting its use. The cursor is opened in execution section of a PL/SQL block and the syntax is as follows:

```
OPEN cursor_name;
```

3. Fetch rows from the cursor (one by one)

Once the cursor is opened successfully, the rows in the cursors could be fetched in to various containers one by one using FETCH statement. The FETCH statement places the contents of the current row into variables or PL/SQL records. The syntax of FETCH statement is as follows:

```
FETCH cursor_name INTO variable_list/PLSQL_Record;
```


To retrieve all rows in a result set, you need to fetch each row till the last one.

4. Close the cursor

After fetching all rows, you need to close the cursor with the CLOSE statement:

```
CLOSE cursor_name;
```

Closing a cursor instructs Oracle to release allocated memory at an appropriate time.

5.2 Cursor Attributes :(%FOUND,%NOTFOUND,%ISOPEN,%ROWCOUNT)

Cursor attributes help a programmer to control features of an explicit cursor in a better way. These attributes have certain values based on the current status of the explicit cursor. The value of a cursor attribute could be used by following syntax:
<CursorName>%<AttributeName>

Following are main four attributes we are about to study.

Attribute	Description	Value	Meaning
%ISOPEN	It defines whether a cursor is in open or close state at present.	TRUE	The cursor is open.
		FALSE	The cursor is closed.
%FOUND	It defines whether the cursor has any rows remaining for further rows to be processed or it has reached to an end. (by processing one by one rows)	NULL	Before first fetch of a row from a cursor. (After the cursor is opened)
		TRUE	A record is fetched successfully.
		FALSE	The cursor has not returned any record and all the records are processed.
		INVALID_CURSOR	If the cursor is not opened and an attempt is made to fetch record from a cursor
%NOTFOUND	It's exactly opposite to %FOUND attribute. It defines whether the cursor has any rows remaining for further rows to be processed or it has reached to an end. (by processing one by one rows)	NULL	Before first fetch of a row from a cursor. (After the cursor is opened)
		FALSE	A record is fetched successfully.
		TRUE	The cursor has not returned any record and all the records are processed.
		INVALID_CURSOR	If the cursor is not opened and an attempt is made to fetch record from a cursor

Attribute	Description	Value	Meaning
%ROWCOUNT	It returns the number of rows fetched from the cursor.	<Number n>	N rows are fetched from a cursor
		INVALID_CURSOR	If the cursor is not opened and an attempt is made to use this attribute.

Let us understand execution cycle of a cursor by a simple example, which lists all the students studying in BCOM stream and the output is sorted by name. The comments in the PL/SQL block describes all the cursor execution steps. (TEST51.SQL)

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        /* Declaring a cursor C_STUDENTS */
        CURSOR C_STUDENTS
        IS
        Select  ROLLNO, STUDENT_SPID, STUDENT_NAME
        From    STUDENTS
        WHERE           COURSE_ID     =         1
        Order by STUDENT_NAME ;

        /* Declaring variables with appropriate data types */
        V_ROLLNO      NUMBER(3);
        V_STUDENT_SPID       NUMBER(5);
        V_STUDENT_NAME    VARCHAR2(255);
BEGIN
        /* Opening the cursor C_STUDENTS */
        OPEN  C_STUDENTS;
        /*Using LOOP-END LOOP iterative statement to process the cursor one by one
records */
        LOOP
                /* Fetching the cursor in variables */
                FETCH C_STUDENTS INTO V_ROLLNO, V_STUDENT_SPID,
V_STUDENT_NAME;
                /* Using %FOUUND attribute to check whether */
                /* the FETCH statement was able to fetch record or not */
```

```
                IF C_STUDENTS%FOUND THEN
                        DBMS_OUTPUT.PUT_LINE(V_STUDENT_NAME||' '||V_ROLLNO||'
'||V_STUDENT_SPID);
                ELSE
                        /* Exiting the loop when all the records are displayed */
                        EXIT;
                END IF;

        END LOOP;
        /* Closing the cursor at last */
        CLOSE  C_STUDENTS;
END;
/
```

(TEST51.SQL)

OUTPUT

```
SQL> @TEST51

PL/SQL procedure successfully completed.

NEHAL 3 20213
NIDHI 2 20212
VIMAL 1 20211

PL/SQL procedure successfully completed.

SQL>
```

We could write and shorten above script by using "EXIT WHEN ..." statement as shown in TEST52.SQL.

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        CURSOR C_STUDENTS
        IS
        Select  ROLLNO, STUDENT_SPID, STUDENT_NAME
        From    STUDENTS
        WHERE           COURSE_ID     =         1
        Order by STUDENT_NAME ;
```

```
        V_ROLLNO      NUMBER(3);
        V_STUDENT_SPID      NUMBER(5);
        V_STUDENT_NAME    VARCHAR2(255);
BEGIN
        OPEN  C_STUDENTS;

        LOOP
                FETCH C_STUDENTS INTO V_ROLLNO, V_STUDENT_SPID,
V_STUDENT_NAME;
                EXIT WHEN C_STUDENTS%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(V_STUDENT_NAME||' '||V_ROLLNO||
                                        ' '||V_STUDENT_SPID);
        END LOOP;
        CLOSE  C_STUDENTS;
END;
/
```

(TEST52.SQL)

Let us now write above script using WHILE loop (TEST53.SQL).

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        CURSOR C_STUDENTS
        IS
        Select  ROLLNO, STUDENT_SPID, STUDENT_NAME
        From   STUDENTS
        WHERE          COURSE_ID     =        1
        Order by STUDENT_NAME ;

        V_ROLLNO      NUMBER(3);
        V_STUDENT_SPID      NUMBER(5);
        V_STUDENT_NAME    VARCHAR2(255);
BEGIN
        OPEN C_STUDENTS;
        FETCH C_STUDENTS INTO V_ROLLNO, V_STUDENT_SPID, V_STUDENT_NAME;
        WHILE C_STUDENTS%FOUND
        LOOP
                DBMS_OUTPUT.PUT_LINE(V_STUDENT_NAME||' '||V_ROLLNO||
```

```
                                        ''||V_STUDENT_SPID);
            FETCH C_STUDENTS INTO V_ROLLNO, V_STUDENT_SPID,
V_STUDENT_NAME;
    END LOOP;
END;
/
```

(TEST53.SQL)

Before writing above PL/SQL script using FOR loop, we have to understand a new data structure called PL/SQL record.

PL/SQL Record

A PL/SQL record is a composite data structure that is a group of related data stored in fields. Each field in the PL/SQL record has its own name and data type. A cursor can have one or more records in it, whereas the PL/SQL record can have only one record at a time. So you may write a loop which fetches one by one record from a cursor into PL/SQL record. So you may replace declaration of multiple variables to declaration of single PL/SQL record.

You can declare a PL/SQL record in three ways:

1. Declaring table based PL/SQL record

To declare a PL/SQL record based on a table one has to use %ROWTYPE attribute. The fields in the PL/SQL records will be same as the fields of the tables on which it is based. Let us write out script using table based PL/SQL record (TEST54.SQL).

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
    CURSOR C_STUDENTS
    IS
    Select *
    From  STUDENTS
    WHERE      COURSE_ID   =     1
    Order by STUDENT_NAME ;

    R_STUDENTS  STUDENTS%ROWTYPE;
BEGIN
    OPEN C_STUDENTS;
    LOOP
```

```
                /* Fetch one by one records from
                the cursor C_STUDENTS to PL/SQL record R_STUDENTS */
                FETCH C_STUDENTS INTO R_STUDENTS;
                EXIT WHEN C_STUDENTS%NOTFOUND;
                /* Refer individual fields in a record using RecordName.FieldName format
*/
                DBMS_OUTPUT.PUT_LINE(R_STUDENTS.STUDENT_NAME||' '||
                                        R_STUDENTS.ROLLNO||' '||
                                        R_STUDENTS.STUDENT_SPID);
        END LOOP;
        CLOSE C_STUDENTS;
END;
/
```

(TEST54.SQL)

2. Declaring programmer defined record

To declare programmer-defined record, first you have to define a record type by using TYPE statement with the fields of record explicitly. Then, you can declare a record based on record type that you've defined.

The following illustrates the syntax of the defining programmer-defined record with TYPE statement:

```
TYPE type_name IS RECORD
  (field1 data_type1 [NOT NULL] := [DEFAULT VALUE],
   field2 data_type2 [NOT NULL] := [DEFAULT VALUE],
   ...
   fieldn data_type3 [NOT NULL] := [DEFAULT VALUE]
   );
```

Let us again write our PL/SQL script using above feature (TEST55.SQL). Please note that in this case, a user defined record type is declared first in declaration section and then a PL/SQL record is declared by assigning it above user defined record type.

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        CURSOR C_STUDENTS
        IS
        Select  STUDENT_NAME, ROLLNO, STUDENT_SPID
        From    STUDENTS
```

```
        WHERE          COURSE_ID    =        1
        Order by STUDENT_NAME ;

        /* User defined PL/SQL record type TY_STUDENTS is being defined */
        TYPE TY_STUDENTS
        IS RECORD
        ( STUDENT_NAME      VARCHAR2(255),
          ROLLNO      NUMBER(3),
      STUDENT_SPID  NUMBER(5) );

        /* R_STUDENT record is declared by assigning it TY_STUDENTS record type */
        R_STUDENTS   TY_STUDENTS;
BEGIN
        OPEN C_STUDENTS;
        LOOP
                /* Fetch one by one records from
                the cursor C_STUDENTS to PL/SQL record R_STUDENTS */
                FETCH C_STUDENTS INTO R_STUDENTS;
                EXIT WHEN C_STUDENTS%NOTFOUND;
                /* Refer individual fields in a record using RecordName.FieldName format
*/
                DBMS_OUTPUT.PUT_LINE(R_STUDENTS.STUDENT_NAME||' '||
                      R_STUDENTS.ROLLNO||' '||
                                R_STUDENTS.STUDENT_SPID);
        END LOOP;
        CLOSE C_STUDENTS;
END;
/
```

(TEST55.SQL)

3. Declaring cursor based record

You can define a record based on a cursor. First, you must define a cursor. And then you use %ROWTYPE with the cursor variable to declare a record. The fields of the record correspond to the columns in the cursor SELECT statement.
TEST56.SQL gives an example on how to declare a record based on a cursor.

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        CURSOR C_STUDENTS
        IS
        Select  ROLLNO, STUDENT_SPID, STUDENT_NAME
        From   STUDENTS
        WHERE          COURSE_ID    =       1
        Order by STUDENT_NAME ;

        /* Declaring a record based on cursor using %ROWTYPE attribute */
        R_STUDENTS  C_STUDENTS%ROWTYPE;
BEGIN
        OPEN  C_STUDENTS;

        LOOP
                FETCH C_STUDENTS INTO R_STUDENTS;
                EXIT WHEN C_STUDENTS%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(R_STUDENTS.STUDENT_NAME||' '||
                                        R_STUDENTS.ROLLNO||' '||
                                        R_STUDENTS.STUDENT_SPID);

        END LOOP;
        CLOSE  C_STUDENTS;
END;
/
```

(TEST56.SQL)

Let us now use FOR loop to write our PL/SQL to display student details studying in BCOM course. As we are aware FOR is a fixed number of iterations type of loop. And the number of records in a cursor is also fixed. We can define a cursor and a PL/SQL record in declaration section and then we can use FOR loop, which fetches once by one record from a cursor into the PL/SQL record (TEST57.SQL).

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        CURSOR C_STUDENTS
        IS
        Select  ROLLNO, STUDENT_SPID, STUDENT_NAME
        From   STUDENTS
        WHERE          COURSE_ID    =       1
        Order by STUDENT_NAME ;
BEGIN
        FOR R_STUDENTS IN C_STUDENTS
        LOOP
                DBMS_OUTPUT.PUT_LINE(R_STUDENTS.STUDENT_NAME||' '||
                                        R_STUDENTS.ROLLNO||' '||
                                        R_STUDENTS.STUDENT_SPID);
        END LOOP;
END;
/
```

(TEST57.SQL)

It is supposed to be the most simple and short script amongst all. Here, we need to make few interesting observations:

1. PL/SQL record R_STUDENTS is not defined explicitly in declaration section. It is implicitly defined when the first iteration of FOR loop is stared.

2. Cursor C_STUDENTS is never opened or closed, when we are using FOR loop. Again, its opened implicitly with the first iteration of the FOR loop and its closed implicitly with the last iteration of FOR loop.

3. As we are aware, FOR is a fixed iteration loop and hence there is no need to write EXIT statement to come out of the loop.

As we are having very good idea about cursors and its attributes, we now write a PL/SQL block (TEST58.SQL) which lists department wise students list. The output should have following details:

1. Course Name (BCOM/BBA/BCA from COURSE table)

2. Roll Number of the student (from STUDENTS table)

3. Name of the student (from STUDENTS table)

As we need to fetch fields from two different tables, we have to write a join query to declare a cursor and then we will be declaring a PL/SQL records based on this cursor. We will use FOR loop for simplicity.

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        CURSOR C_STUD_DETAILS
        IS
        Select  C.COURSE_NAME,
                S.ROLLNO,
                S.STUDENT_SPID,
                S.STUDENT_NAME
        From    COURSE                  C,
                STUDENTS        S
        WHERE           C.COURSE_ID  =          S.COURSE_ID
        Order by C.COURSE_NAME, S.ROLLNO ;
BEGIN
        FOR R_STUD_DETAILS IN C_STUD_DETAILS
        LOOP
                DBMS_OUTPUT.PUT_LINE(R_STUD_DETAILS.COURSE_NAME||' '||
                                        R_STUD_DETAILS.ROLLNO||' '||
                                        R_STUD_DETAILS.STUDENT_SPID||' '||
                                        R_STUD_DETAILS.STUDENT_NAME );
        END LOOP;
END;
/
```

(TEST58.SQL)

OUTPUT:

```
SQL> @TEST58

PL/SQL procedure successfully completed.

BBA 1 20217 AAKANSHA
BBA 2 20218 JAIMINI
BBA 3 20219 CHIRAG
BCA 1 20214 NAINESH
BCA 2 20215 EEVA
```

```
BCA 3 20216 KRISHNA
BCOM 1 20211 VIMAL
BCOM 2 20212 NIDHI
BCOM 3 20213 NEHAL

PL/SQL procedure successfully completed.
SQL>
```

5.3 Exception Handling in PL/SQL

In PL/SQL, a warning or error condition is called an exception. An exception occurs when the PL/SQL engine encounters an instruction or statement which is not executable due to an error that occurs at run-time. (Although the code is syntax wise correct.) These errors will not be captured at the time of compilation and hence these needed to handle only at the run-time.

For example, if PL/SQL engine receives an instruction to divide any number by '0', then the PL/SQL engine will throw it as an exception. The exception is only raised at the run-time by the PL/SQL engine.

Exceptions will stop the program from executing further, so to avoid such condition, they need to be captured and handled separately. This process is called as Exception-Handling, in which the programmer handles the exception that can occur at the run time. If such exceptions are not handled then the program will terminate with an error.

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle.
The default exception will be handled using WHEN others THEN –

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
```

```
  WHEN others THEN
    exception3-handling-statements
END;
```

Although we have not gone through the system defined exception yet, we can understand exception handling by a simple code given below. On the left side a simple PL/SQL code written in which there is no exception handling mechanism, which would result in a runtime error. On the right side the PL/SQL code is having exception handling mechanism, which can handle this erroneous situation. The code us fetching a name of the student who is studying in BCOM and whose roll number is 77. We know that there is no record for his criteria and the query is going to return no rows.

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        V_STUDENT_NAME
        STUDENTS.STUDENT_NAME%TYPE;
BEGIN
        SELECT  STUDENT_NAME
        INTO    V_STUDENT_NAME
        FROM    STUDENTS
        WHERE   COURSE_ID       =       1
        AND     ROLLNO          =       77;

DBMS_OUTPUT.PUT_LINE(V_STUDENT_NAME );
END;
/
```

(TEST59A.SQL)

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
        V_STUDENT_NAME
        STUDENTS.STUDENT_NAME%TYPE;
BEGIN
        SELECT  STUDENT_NAME
        INTO    V_STUDENT_NAME
        FROM    STUDENTS
        WHERE   COURSE_ID       =       1
        AND     ROLLNO          =       77;

DBMS_OUTPUT.PUT_LINE(V_STUDENT_NAME );

EXCEPTION WHEN NO_DATA_FOUND
          THEN
        DBMS_OUTPUT.PUT_LINE('ROLL NO 77
NOT FOUND.');
END;
/
```

(TEST59B.SQL)

Output:

```
SQL> @TEST59a

DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 4


SQL>
```

Output:

```
SQL> @TEST59B

PL/SQL procedure successfully completed.

ROLL NO 77 NOT FOUND.

PL/SQL procedure successfully completed.

SQL>
```

It is obvious that TEST59A.SQL has resulted in a runtime error with error number ORA-01403: No data found, which means the SELECT statement has not returned any record. The same SELECT statement is accompanied by exception handling mechanism in TEST59B.SQL. The exception NO_DATA_FOUND is a named system exception, which is ***raised*** by SELECT statement automatically as it returns no record. The control directly passes to exception handling section after executing SELECT statement, ignoring DBMS_OUTPUT.PUT_LINE statement. After executing exception handling section, the PL/SQL script competes the execution without any runtime error.

5.3.1 Types of Exceptions:

There are mainly two types of exceptions:

1. Internally system defined (by the run-time system) exceptions

2. User defined exceptions

Let us now understand internally system defined exceptions.

5.3.1.1 Named System Exceptions

Named system exceptions are exceptions that have been given names by PL/SQL. They are named in the STANDARD package in PL/SQL and do not need to be defined by the programmer. PL/SQL provides many pre-defined named exceptions, which are executed when any database rule is violated by a program. The following table lists a few of the important pre-defined exceptions –

Oracle Exception Name	Oracle Error	Description
DUP_VAL_ON_INDEX	ORA-00001	You tried to execute an INSERT or UPDATE statement that has created a duplicate value in a field restricted by a unique index.
TIMEOUT_ON_RESOURCE	ORA-00051	You were waiting for a resource and you timed out.
INVALID_CURSOR	ORA-01001	You tried to reference a cursor that does not yet exist. This may have happened because you've executed a FETCH cursor or CLOSE cursor before OPENing the cursor.
NOT_LOGGED_ON	ORA-01012	You tried to execute a call to Oracle before logging in.

LOGIN_DENIED	ORA-01017	You tried to log into Oracle with an invalid username/password combination.
NO_DATA_FOUND	ORA-01403	You tried one of the following: 1. You executed a SELECT INTO statement and no rows were returned. 2. You referenced an uninitialized row in a table. 3. You read past the end of file with the UTL_FILE package.
TOO_MANY_ROWS	ORA-01422	You tried to execute a SELECT INTO statement and more than one row was returned.
ZERO_DIVIDE	ORA-01476	You tried to divide a number by zero.
INVALID_NUMBER	ORA-01722	You tried to execute a SQL statement that tried to convert a string to a number, but it was unsuccessful.
STORAGE_ERROR	ORA-06500	You ran out of memory or memory was corrupted.
PROGRAM_ERROR	ORA-06501	This is a generic "Contact Oracle support" message because an internal problem was encountered.
VALUE_ERROR	ORA-06502	You tried to perform an operation and there was a error on a conversion, truncation, or invalid constraining of numeric or character data.
CURSOR_ALREADY_OPEN	ORA-06511	You tried to open a cursor that is already open.

We have already seen an example which explains using NO_DATA_FOUND exception.
Let us now understand another exception DUP_VAL_ON_INDEX in TEST510.SQL

```
execute dbms_output.enable(10000);
set serveroutput on;
BEGIN
 INSERT INTO STUDENTS
 ( STUDENT_SPID, ROLLNO, STUDENT_NAME, COURSE_ID, BIRTH_DATE)
 VALUES
 ( 20221, 1, 'VISHAL PATEL', 1, '01-SEP-99');
```

```
DBMS_OUTPUT.PUT_LINE('One row inserted.');

EXCEPTION WHEN NO_DATA_FOUND
        THEN
          DBMS_OUTPUT.PUT_LINE('ROLL NO 1 NOT FOUND.');
      WHEN DUP_VAL_ON_INDEX
      THEN
        DBMS_OUTPUT.PUT_LINE('PRIMARY KEY VIOLATED.');
        DBMS_OUTPUT.PUT_LINE('ROW NOT INSERTED.');
END;
/
```

(TEST510.SQL)

Output:

```
SQL> @TEST510

PL/SQL procedure successfully completed.

PRIMARY KEY VIOLATED.
ROW NOT INSERTED.

PL/SQL procedure successfully completed.

SQL>
```

Above script is having an INSERT statement which tries to insert a record in STUDENTS table with COURSE_ID= 1 and ROLLNO=1. As we are aware, COURSE_ID+ROLLNO is the primary key on STUDENTS table and it is already having a record with COURSE_ID=1 and ROLLNO=1. So when we try to insert another record with the same key values, the exception DUP_VALUE_ON_INDEX is raised.

5.3.1.2 Unnamed System Exceptions

The system exceptions for which Oracle does not have a name are known as unnamed system exceptions. These types of exceptions have a predefined unique error code and an error message but without a name. These exceptions occur in a rare situation and that is why these are not named.

These exceptions can be handled using the below two methods,

1. By handling them using the OTHERS handler.

2. By associating a name to its error code explicitly using the PRAGMA compiler called as EXCEPTION_INIT.

1. By handling them using the OTHERS handler.

The WHEN OTHERS clause is used to trap all remaining exceptions that have not been handled by your Named System Exceptions and Named Programmer-Defined Exceptions.

Let us understand WHEN OTHERS clause with the help of following example (TEST511.SQL).

```
execute dbms_output.enable(10000);
set serveroutput on;

BEGIN
 INSERT INTO STUDENTS
 ( STUDENT_SPID, ROLLNO, STUDENT_NAME, COURSE_ID, BIRTH_DATE)
 VALUES
 ( 20222, 5, 'CHINMAY MODI', 1, '31-SEP-98');

DBMS_OUTPUT.PUT_LINE('One row inserted.');

EXCEPTION
       WHEN DUP_VAL_ON_INDEX
       THEN
      raise_application_error (-20001,'PRIMARY KEY VIOLATED.');
          WHEN OTHERS
       THEN
      raise_application_error (-20002,'An error has occurred inserting a student.');
END;
/
```

(TEST511.SQL)

Output:

```
SQL> @TEST511

PL/SQL procedure successfully completed.

BEGIN
*
ERROR at line 1:
ORA-20002: An error has occurred inserting a student.
ORA-06512: at line 15

SQL>
```

As you can observe, the birth date of the newly added student is kept incorrect knowingly. (31-SEP-98). As any of the named exception is not able to handle this situation/error, the OTHERS exception is useful and it handles this unnamed system exception. The procedure raise_application_error allows you to issue an user-defined error from a code block or stored program. The raise_application_error has the following syntax:

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

In this syntax:

- The error_number is a negative integer with the range from -20999 to -20000.
- The message is a character string that represents the error message. Its length is up to 2048 bytes.
- If the third parameter is FALSE, the error replaces all previous errors. If it is TRUE, the error is added to the stack of previous errors.

2. By associating a name to its error code explicitly using the PRAGMA compiler called as EXCEPTION_INIT.

As we can understand easily, all named system defined exceptions do not suffice the requirements to handle all Oracle error conditions. To handle such error conditions (typically ORA- messages) that have no predefined name, you must use the OTHERS handler or the pragma EXCEPTION_INIT. A pragma is a compiler directive that is processed at compile time, not at run time.

In PL/SQL, the pragma EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number. That lets you refer to any internal exception by name and to write a specific handler for it. When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.

You code the pragma EXCEPTION_INIT in the declarative part of a PL/SQL block, subprogram, or package using the syntax

```
PRAGMA EXCEPTION_INIT(exception_name, -Oracle_error_number);
```

Following scrip help us understanding PRAGMA EXCEPTION_INIT (TEST512.SQL). We have tried to associate Oracle error number ORA-00060 (deadlock related error) with our exception.

```
execute dbms_output.enable(10000);
set serveroutput on;

DECLARE
  deadlock_detected EXCEPTION;
  PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
 INSERT INTO STUDENTS
 ( STUDENT_SPID, ROLLNO, STUDENT_NAME, COURSE_ID, BIRTH_DATE)
 VALUES
 ( 20223, 6, 'TEJAS PATEL', 1, '31-OCT-98');

DBMS_OUTPUT.PUT_LINE('One row inserted.');

EXCEPTION
       WHEN DUP_VAL_ON_INDEX
       THEN
      raise_application_error (-20001,'PRIMARY KEY VIOLATED.');
          WHEN deadlock_detected
       THEN
      DBMS_OUTPUT.PUT_LINE('Deadlock occurred while creating a student.');
END;
/
```

(TEST512.SQL)

Output (Assuming a deadlock occurred while running the script)

```
SQL> @TEST512

PL/SQL procedure successfully completed.

Deadlock occurred while creating a student.

PL/SQL procedure successfully completed.

SQL>
```

Oracle error 60 is for deadlock encountered while executing a DML statement. The exact error code and error message is : "ORA-00060: deadlock detected while waiting for resource.".
In above script, "deadlock_detected" exception is declared in exception section. And by PRAGMA EXCEPTION_INIT, it is initialized by error code 60. Hence, whenever the script encounters an Oracle error 60, it will be named as "deadlock_detected". And in the exception handling section, it is handled with "EXCEPTION WHEN DEADLOCK_DETECTED" statement

5.3.1.3 User-defined Exceptions

User defined exceptions are the exceptions defined and named by the user, who is writing PL/SQL block. This type of exceptions are generally used for business logic. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure RAISE_APPLICATION_ERROR. As we have already seen use of RAISE_APPLICATION_ERROR, we will see an example of user defined exception using RAISE statement (TEST513.SQL).

```
execute dbms_output.enable(10000);
set serveroutput on;

ACCEPT V_ROLLNO NUMBER PROMPT 'ENTER ROLL NUMBER :';

DECLARE
  NEGATIVE_ROLLNO EXCEPTION;
BEGIN
IF &V_ROLLNO < 0 THEN
  RAISE NEGATIVE_ROLLNO;
ELSE
 INSERT INTO STUDENTS
 ( STUDENT_SPID, ROLLNO, STUDENT_NAME, COURSE_ID, BIRTH_DATE)
 VALUES
 ( 20224, &V_ROLLNO, 'VIRAL', 1, '31-OCT-98');
END IF;

DBMS_OUTPUT.PUT_LINE('One row inserted.');

EXCEPTION
      WHEN DUP_VAL_ON_INDEX
      THEN
```

```
      raise_application_error (-20001,'PRIMARY KEY VIOLATED.');
          WHEN NEGATIVE_ROLLNO
       THEN
      DBMS_OUTPUT.PUT_LINE('ERROR: A student can NOT have a negative roll
number.');
END;
/
```

(TEST513.SQL)

Output:

```
SQL> @TEST513

PL/SQL procedure successfully completed.

ENTER ROLL NUMBER :-5
old   4: IF &V_ROLLNO < 0 THEN
new   4: IF       -5 < 0 THEN
old  10:  ( 20224, &V_ROLLNO, 'VIRAL', 1, '31-OCT-98');
new  10:  ( 20224,       -5, 'VIRAL', 1, '31-OCT-98');
ERROR: A student can NOT have a negative roll number.

PL/SQL procedure successfully completed.

SQL>
```

In above script, we have defined an exception "NEGATIVE_ROLLNO" in declaration section. Please note that it is not associated with any Oracle error. The user is prompted to enter the roll number at the start of the script by SQL command ACCEPT. Now the script has to validate that if the entered roll number is positive then only the record is to be added to STUDENTS table. If it is negative, the user must be informed that the roll number is negative and hence, the records could not be inserted. The exception NEGATIVE_ROLLNO is explicitly raised by RAISE NEGATIVE_ROLLNO command within conditional statement to check if it is negative or not.

5.3.2 Exception Handling

We have discussed this part earlier in this unit only with the help of some of the basic kind of example. We are also aware about how the exceptions are handled in the PL/SQL script and how the control of the program is directly transferred to exception handling section in case of an exception is raised in main/execution section of the PL/SQL.

Let us not discuss both cursors and exception handling with bit difficult PL/SQL examples.

Example-1: Write a PL/SQL script which shows subject wise marks scored by every student studying in the course entered by a user at runtime.

```
execute dbms_output.enable(10000);
set serveroutput on;

ACCEPT V_COURSE CHAR PROMPT 'Enter the Course (BCOM/BBA/BCA) : ';

DECLARE
  CURSOR C_RESULT
  IS
  SELECT  C.COURSE_NAME, S.ROLLNO, S.STUDENT_NAME,
          SB.SUBJECT_NAME, R.MARKS_SCORED
  FROM    COURSE   C,
          STUDENTS  S,
          RESULT   R,
          SUBJECTS  SB
  WHERE C.COURSE_NAME   =   '&V_COURSE'
  AND    C.COURSE_ID    =   S.COURSE_ID
  AND    R.STUDENT_SPID   = S.STUDENT_SPID
  AND    SB.SUBJECT_ID   =   R.SUBJECT_ID
  AND    S.COURSE_ID   =   SB.COURSE_ID
  ORDER BY C.COURSE_NAME, S.ROLLNO, SB.SUBJECT_NAME;

  NEGATIVE_MARKS EXCEPTION;
BEGIN
  FOR R_RESULT IN C_RESULT
  LOOP
    IF R_RESULT.MARKS_SCORED < 0 THEN
       RAISE NEGATIVE_MARKS;
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE(R_RESULT.COURSE_NAME||' '||
                    R_RESULT.ROLLNO||' '||
                    R_RESULT.STUDENT_NAME||' '||
                    R_RESULT.SUBJECT_NAME||' '||
                    R_RESULT.MARKS_SCORED );
    END IF;
 END LOOP;
DBMS_OUTPUT.PUT_LINE('Listing Completed Successfully !!!');

EXCEPTION
        WHEN NEGATIVE_MARKS
      THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: A student can NOT have negative marks.');
      WHEN OTHERS
      THEN
        DBMS_OUTPUT.PUT_LINE('SQLCODE : '||SQLCODE);
        DBMS_OUTPUT.PUT_LINE('Message : '||SQLERRM);
END;
/
```

(TEST514.SQL)

Outpupt:

```
SQL> @TEST514

PL/SQL procedure successfully completed.

Enter the Course (BCOM/BBA/BCA) : BCOM
old  9:   WHERE   C.COURSE_NAME   =   '&V_COURSE'
new  9:   WHERE   C.COURSE_NAME   =   'BCOM'
BCOM 1 VIMAL ACCOUNTANCY 75
BCOM 1 VIMAL ECONOMICS 67
BCOM 2 NIDHI ECONOMICS 61
BCOM 2 NIDHI ACCOUNTANCY 81
BCOM 3 NEHAL ECONOMICS 47
BCOM 3 NEHAL ACCOUNTANCY 31
Listing Completed Successfully !!!

PL/SQL procedure successfully completed.

SQL>
```

In TEST514.SQL, the cursor C_RESULT is created based on a join query of four tables. If we understand all the tables' integrity constraints, writing this query is not so difficult. A user defined exception NEGATIVE_MARKS is declared and raised in the main/execution section based on a condition checking if the marks are negative. The FOR loop is used, where there is no need to OPEN or CLOSE the cursor explicitly. Also there is no need to declare PL/SQL record R_RESULT. The Oracle SQLERRM function returns the error message associated with the most recently raised error exception. The Oracle SQLERRM function should only be used within the Exception Handling section of your PL/SQL script. SQLCODE returns the error number for which the exception was raised. As they both are written within exception OTHERS part, they will show information regarding unnamed system exceptions.

We can write above PL/SQL script in a slightly different way, where we will use two features: 1. Parent and Child cursor 2. Nested Loop. (TEST515.SQL)

```
execute dbms_output.enable(10000);
set serveroutput on;

ACCEPT V_COURSE CHAR PROMPT 'Enter the Course (BCOM/BBA/BCA) : ';

DECLARE
  V_STUDENT_SPID STUDENTS.STUDENT_SPID%TYPE;
  V_COURSE_ID   COURSE.COURSE_ID%TYPE;

  CURSOR  C_STUDENTS
  IS
  SELECT  C.COURSE_NAME, S.ROLLNO, S.STUDENT_NAME, S.STUDENT_SPID,
C.COURSE_ID
  FROM    COURSE    C,
          STUDENTS  S
  WHERE   C.COURSE_NAME   =   '&V_COURSE'
  AND     C.COURSE_ID     =   S.COURSE_ID
  ORDER BY C.COURSE_NAME, S.ROLLNO;

  CURSOR  C_RESULT
  IS
  SELECT  S.SUBJECT_NAME, R.MARKS_sCORED
  FROM    RESULT    R,
          SUBJECTS  S
  WHERE   R.STUDENT_SPID = V_STUDENT_SPID
  AND     S.COURSE_ID   = V_COURSE_ID
```

```
  AND     S.SUBJECT_ID  = R.SUBJECT_ID
  ORDER BY  S.SUBJECT_NAME, R.MARKS_SCORED;

  NEGATIVE_MARKS EXCEPTION;

BEGIN

  FOR R_STUDENTS IN C_STUDENTS
  LOOP
    DBMS_OUTPUT.PUT_LINE('COURSE : '||R_STUDENTS.COURSE_NAME||' '||
                  'ROLL NO : '||R_STUDENTS.ROLLNO||' '||
                  'NAME : '||R_STUDENTS.STUDENT_NAME );

    V_COURSE_ID   := R_STUDENTS.COURSE_ID;
    V_STUDENT_SPID := R_STUDENTS.STUDENT_SPID;


    FOR R_RESULT IN C_RESULT
    LOOP

      IF R_RESULT.MARKS_SCORED < 0 THEN
          RAISE NEGATIVE_MARKS;
      ELSE
        DBMS_OUTPUT.PUT_LINE(R_RESULT.SUBJECT_NAME||' '||
                    R_RESULT.MARKS_SCORED );
      END IF;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('---------------------------------');

  END LOOP;

DBMS_OUTPUT.PUT_LINE('Listing Completed Successfully !!!');

EXCEPTION
         WHEN NEGATIVE_MARKS
      THEN
         DBMS_OUTPUT.PUT_LINE('ERROR: A student can NOT have negative marks.');
      WHEN OTHERS
      THEN
         DBMS_OUTPUT.PUT_LINE('SQLCODE : '||SQLCODE);
```

```
        DBMS_OUTPUT.PUT_LINE('Message : '||SQLERRM);
END;
/
```

(TEST515.SQL)

Output :

```
SQL> @TEST515

PL/SQL procedure successfully completed.

Enter the Course (BCOM/BBA/BCA) : BCOM
old 10:   WHERE   C.COURSE_NAME   =   '&V_COURSE'
new 10:   WHERE   C.COURSE_NAME   =   'BCOM'

COURSE : BCOM ROLL NO : 1 NAME : VIMAL
ACCOUNTANCY 75
ECONOMICS 67
---------------------------------
COURSE : BCOM ROLL NO : 2 NAME : NIDHI
ACCOUNTANCY 81
ECONOMICS 61
---------------------------------
COURSE : BCOM ROLL NO : 3 NAME : NEHAL
ACCOUNTANCY 31
ECONOMICS 47
---------------------------------
Listing Completed Successfully !!!

PL/SQL procedure successfully completed.

SQL>
```


In above script, C_STUDENT is parent cursor and C_RESULT is child cursor. C_STUDENT cursor is opened for each record of C_STUDENT i.e. for each different STUDEN_SPID of C_STUDENT, C_RESULT will be opened and then closed. The opening of C_RESULT is controlled by two variables V_STUDENT_SPID and V_COURSE_ID, which are initialized for each record of C_STUDENT. This feature of parent and child cursor is handled by nested FOR loops. (which we studied in Unit 4).

Now, we will understand a PL/SQL which gives additional 10 marks to the student of BCA in the subject of RDBMS. (TEST516.SQL)

```
execute dbms_output.enable(10000);
set serveroutput on;

ACCEPT V_COURSE  CHAR PROMPT 'Enter the Course (BCOM/BBA/BCA) : ';
ACCEPT V_SUBJECT CHAR PROMPT 'Enter the Subject : ';

DECLARE
  deadlock_detected EXCEPTION;
  PRAGMA EXCEPTION_INIT(deadlock_detected, -60);

  NEGATIVE_MARKS EXCEPTION;

  CURSOR C_RESULT
  IS
  SELECT  C.COURSE_NAME, S.ROLLNO, S.STUDENT_NAME,
       SB.SUBJECT_ID, SB.SUBJECT_NAME, R.MARKS_sCORED,
       S.STUDENT_SPID
  FROM    COURSE    C,
          STUDENTS  S,
          RESULT    R,
          SUBJECTS  SB
  WHERE  C.COURSE_NAME   =   '&V_COURSE'
  AND    SB.SUBJECT_NAME =   '&V_SUBJECT'
  AND    C.COURSE_ID     =   S.COURSE_ID
  AND    R.STUDENT_SPID  =   S.STUDENT_SPID
  AND    SB.SUBJECT_ID   =   R.SUBJECT_ID
  AND    S.COURSE_ID     =   SB.COURSE_ID
  ORDER BY C.COURSE_NAME, S.ROLLNO, sb.SUBJECT_NAME;

  V_CNT   NUMBER(2) := 0;
BEGIN
  FOR R_RESULT IN C_RESULT
  LOOP
    IF R_RESULT.MARKS_SCORED < 0 THEN
       RAISE NEGATIVE_MARKS;
    ELSIF R_RESULT.MARKS_SCORED <= 90 THEN
       UPDATE   RESULT
       SET      MARKS_SCORED = MARKS_sCORED + 10
```

```
        WHERE    STUDENT_SPID = R_RESULT.STUDENT_SPID
        AND      SUBJECT_ID   = R_RESULT.SUBJECT_ID ;

        V_CNT := V_CNT + 1;
    END IF;

  END LOOP;

DBMS_OUTPUT.PUT_LINE('Total '||V_CNT||' students are given grace marks.');
EXCEPTION
      WHEN NEGATIVE_MARKS
      THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: A student can NOT have negative marks.');
      WHEN deadlock_detected
      THEN
        DBMS_OUTPUT.PUT_LINE('Deadlock occurred while updating RESULT table.');
      WHEN OTHERS
      THEN
          DBMS_OUTPUT.PUT_LINE('SQLCODE : '||SQLCODE);
          DBMS_OUTPUT.PUT_LINE('Message : '||SQLERRM);
END;
/
```

(TEST516.SQL)

Output:

```
SQL> @TEST516
PL/SQL procedure successfully completed.

Enter the Course (BCOM/BBA/BCA) : BCA
Enter the Subject : RDBMS
old 16:   WHERE   C.COURSE_NAME   =   '&V_COURSE'
new 16:   WHERE   C.COURSE_NAME   =   'BCA'
old 17:   AND     SB.SUBJECT_NAME =   '&V_SUBJECT'
new 17:   AND     SB.SUBJECT_NAME =   'RDBMS'
Total 3 students are given grace marks.

PL/SQL procedure successfully completed.

SQL>
```

The marks of BCA students in RDBMS subject before running the script was:

STUDENT_NAME	SUBJECT_NAME	MARKS_SCORED
NAINESH	RDBMS	75
EEVA	RDBMS	29
KRISHNA	RDBMS	56

And after executing script TEST516.SQL it becomes:

STUDENT_NAME	SUBJECT_NAME	MARKS_SCORED
NAINESH	RDBMS	85
EEVA	RDBMS	39
KRISHNA	RDBMS	66

In above script, all the records of RESULT table has been updated where marks of a student in RDBMS subject is less or equal to 90. All such records were updated to add 10 gracing marks. The script is having an UPDATE statement to do this updation. A number variable V_CNT has been used just to show how many records were updated by the script. (In our case, 3 records were updated.)

****