# UNIT 3 ADVANCED SQL

**Structure** **Page Nos.**



## 3.0 INTRODUCTION

The Structured Query Language (SQL) is a standard query language for database systems. It is considered as one of the factors contributing to the success of commercial database management systems, primarily, because of the availability of this standard language on most commercial database systems. We have described SQL in details in the course MCS-023 Block-2, Unit-1 where we discussed data definition, data manipulation, data control, queries, joins, group commands, sub-queries, etc.

In this unit, we provide details of some of the advanced features of Structured Query Language. We will discuss Assertions and Views, Triggers, Standard Procedure and Cursors. The concepts of embedded and dynamic SQL and SQLJ, which is used along with JAVA, are also been introduced. Some of the advanced features of SQL have been covered. We will provide examples in various sections rather than including a separate section of examples. The examples given here are in a SQL3 standard and will be applicable for any commercial database management system that supports SQL3 standards.

## 3.1 OBJECTIVES

After going through this unit, you should be able to:

- define Assertions and explain how they can be used in SQL;
- explain the concept of views, SQL commands on views and updates on views;
- define and use Cursors;
- discuss Triggers and write stored procedures, and
- explain Dynamic SQL and SQLJ.

## 3.2 ASSERTIONS AND VIEWS

One of the major requirements in a database system is to define constraints on various tables. Some of these simple constraints can be specified as primary key, NOT NULL, check value and range constraints. Such constraints can be specified within

the data or table creation statements with the help of statements like NOT NULL, PRIMARY KEY, UNIQUE, CHECK etc. Referential constraints can be specified with the help of foreign key constraints. However, there are some constraints, which may relate to more than one field or table. These are called assertions.

In this section we will discuss about two important concepts that we use in database systems viz., views and assertions. Assertions are general constraints while views are virtual tables. Let us discuss about them in more detail.

### 3.2.1 Assertions

Assertions are constraints that are normally of general nature. For example, the age of the student in a hypothetical University should not be more than 25 years or the minimum age of the teacher of that University should be 30 years. Such general constraints can be implemented with the help of an assertion statement. The syntax for creating assertion is:

Syntax:

CREATE ASSERTION <Name>

CHECK (<Condition>);

Thus, the assertion on age for the University as above can be implemented as:

```
CREATE ASSERTION age-constraint
CHECK (NOT EXISTS (
                SELECT *
                FROM STUDENT s
                WHERE s.age > 25
                OR s.age > (
                        SELECT MIN (f.age)
                        FROM FACULTY f
                ));
```

The assertion name helps in identifying the constraints specified by the assertion. These names can be used to modify or delete an assertion later. But how are these assertions enforced? The database management system is responsible for enforcing the assertion on to the database such that the constraints stated in the assertion are not violated. Assertions are checked whenever a related relation changes.

Now try writing an assertion for a university system that stores a database of faculty as:

FACULTY (code, name, age, basic salary, medical-allow, other benefits)

MEDICAL-CLAIM (code, date, amount, comment)

Assertion: The total medical claims made by a faculty member in the current financial year should not exceed his/her medial allowance.

```
CREATE ASSERTION med-claim
CHECK (NOT EXISTS (
        SELECT code, SUM (amount), MIN(medical-allow)
        FROM (FACULTY NATRUAL JOIN MEDICAL-CLAIM)
        WHERE date>”31-03-2006”
        GROUP BY code
        HAVING MIN(medical-allow) < SUM(amount)
));
```

**OR**

```
CREATE ASSERTION med-claim
CHECK (NOT EXISTS (
        SELECT  *
        FROM FACULT f
        WHERE (f.code IN
        (SELECT code, SUM(amount)
        FROM MEDICAL-CLAIM m
WHERE date>"31-03-2006" AND f.code=m.code AND
f.medical-allow<SUM(amount)
```

Please analyse both the queries above and find the errors if any.

So, now you can create an assertion. But how can these assertions be used in database systems? The general constraints may be designed as assertions, which can be put into the stored procedures. Thus, any violation of an assertion may be detected.

### 3.2.2 Views

A view is a virtual table, which does not actually store data. But if it does not store any data, then what does it contain?

A view actually is a query and thus has a SELECT FROM WHERE ….. clause which works on physical table which stores the data. Thus, the view is a collection of relevant information for a specific entity. The 'view' has been introduced as a topic in MCS-023, Block 2, Unit-1. Let us recapitulate the SQL commands for creating views, with the help of an example.

Example: A student's database may have the following tables:

STUDENT (name, enrolment-no, dateofbirth)
MARKS (enrolment-no, subjectcode, smarks)

For the database above a view can be created for a Teacher who is allowed to view only the performance of the student in his/her subject, let us say MCS-043.

```
CREATE VIEW SUBJECT-PERFORMANCE AS

        (SELECT  s.enrolment-no, name, subjectcode, smarks
        FROM STUDENT s, MARKS m
        WHERE s.enrolment-no = m.enrolment-no AND
        subjectcode 'MCS-043' ORDER BY s.enrolment-no;
```

A view can be dropped using a DROP statement as:

```
DROP VIEW SUBJECT-PERFORMANCE;
```

The table, which stores the data on which the statement of the view is written, is sometimes referred to as the base table. You can create views on two or more base tables by combining the data using joins. Thus, a view hides the logic of joining the tables from a user. You can also index the views too. This may speed up the performance. Indexed views may be beneficial for very large tables. Once a view has been created, it can be queried exactly like a base table. For example:

```
SELECT   *
FROM  STUDENT-PERFORMANCE
WHERE   smarks >50
```

How the views are implemented?

There are two strategies for implementing the views. These are:

- Query modification
- View materialisation.

In the query modification strategy, any query that is made on the view is modified to include the view defining expression. For example, consider the view STUDENT-PERFORMANCE. A query on this view may be: The teacher of the course MCS-043 wants to find the maximum and average marks in the course. The query for this in SQL will be:

```
SELECT  MAX(smarks), AVG(smarks)
FROM SUBJECT-PERFORMANCE
```

Since SUBJECT-PERFORMANCE is itself a view the query will be modified automatically as:

```
SELECT MAX (smarks), AVG (smarks)
FROM STUDENT s, MARKS m
WHERE s.enrolment-no=m.enrolment-no AND subjectcode= "MCS-043";
```

However, this approach has a major disadvantage. For a large database system, if complex queries have to be repeatedly executed on a view, the query modification will have to be done each time, leading to inefficient utilisation of resources such as time and space.

The view materialisation strategy solves this problem by creating a temporary physical table for a view, thus, materialising it. However, this strategy is not useful in situations where many database updates are made on the tables, which are used for view creation, as it will require suitable updating of a temporary table each time the base table is updated.

Can views be used for Data Manipulations?

Views can be used during DML operations like INSERT, DELETE and UPDATE. When you perform DML operations, such modifications need to be passed to the underlying base table. However, this is not allowed on all the views. Conditions for the view that may allow Data Manipulation are:

A view allows data updating, if it follows the following conditions:

1) If the view is created from a single table, then:
    - For INSERT operation, the PRIMARY KEY column(s) and all the NOT NULL columns must be included in the view.
    - View should not be defined using any aggregate function or GROUP BY or HAVING or DISTINCT clauses. This is due to the fact that any update in such aggregated attributes or groups cannot be traced back to a single tuple of the base table. For example, consider a view avgmarks (coursecode, avgmark) created on a base table student(st_id, coursecode, marks). In the avgmarks table changing the class average marks for coursecode "MCS 043" to 50 from a calculated value of 40, cannot be accounted for a single tuple in the Student base table, as the average marks are computed from the marks of all the Student tuples for that coursecode. Thus, this update will be rejected.
2) The views in SQL that are defined using joins are normally NOT updatable in general.
3) WITH CHECK OPTION clause of SQL checks the updatability of data from views, therefore, must be used with views through which you want to update.

**Views and Security**

Views are useful for security of data. A view allows a user to use the data that is available through the view; thus, the hidden data is not made accessible. Access privileges can be given on views. Let us explain this with the help of an example. Consider the view that we have created for teacher-STUDENT-PERFORMANCE. We can grant privileges to the teacher whose name is 'ABC' as:

GRANT SELECT, INSERT, DELETE ON STUDENT-PERFORMANCE TO ABC WITH GRANT OPTION;

Please note that the teacher ABC has been given the rights to query, insert and delete the records on the given view. Please also note s/he is authorised to grant these access rights (WITH GRANT OPTION) to any data entry user so that s/he may enter data on his/her behalf. The access rights can be revoked using the REVOKE statement as:

REVOKE ALL ON STUDENT-PERFORMANCE FROM ABC;

### ☞ Check Your Progress 1

1) Consider a constraint – the value of the age field of the student of a formal University should be between 17 years and 50 years. Would you like to write an assertion for this statement?

……

……

……

2) Create a view for finding the average marks of the students in various subjects, for the tables given in section 3.2.2.

……

……

……

3) Can the view created in problem 2 be used to update subjectcode?

……

……

……

## 3.3 EMBEDDED SQL AND DYNAMIC SQL

SQL commands can be entered through a standard SQL command level user interface. Such interfaces are interactive in nature and the result of a command is shown immediately. Such interfaces are very useful for those who have some knowledge of SQL and want to create a new type of query. However, in a database application where a naïve user wants to make standard queries, that too using GUI type interfaces, probably an application program needs to be developed. Such interfaces sometimes require the support of a programming language environment.

Please note that SQL normally does not support a full programming paradigm (although the latest SQL has full API support), which allows it a full programming interface. In fact, most of the application programs are seen through a programming interface, where SQL commands are put wherever database interactions are needed. Thus, SQL is embedded into programming languages like C, C++, JAVA, etc.

Let us discuss the different forms of embedded SQL in more detail.

### 3.3.1 Embedded SQL

The embedded SQL statements can be put in the application program written in C, Java or any other host language. These statements sometime may be called static. Why are they called static? The term 'static' is used to indicate that the embedded SQL commands, which are written in the host program, do not change automatically during the lifetime of the program. Thus, such queries are determined at the time of database application design. For example, a *query statement* embedded in C to determine the status of train booking for a train will not change. However, this query may be executed for many different trains. Please note that it will only change the input parameter to the query that is train-number, date of boarding, etc., and not the query itself.

But how is such embedding done? Let us explain this with the help of an example.

Example: Write a C program segment that prints the details of a student whose enrolment number is input.

Let us assume the relation

STUDENT (enrolno:char(9), name:Char(25), phone:integer(12), prog-code:char(3))

```
/* add proper include statements*/
/*declaration in C program */
EXEC SQL BEGIN DECLARE SECTION;
        Char enrolno[10], name[26], p-code[4];
        int phone;
        int SQLCODE;
        char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;

/* The connection needs to be established with SQL*/
/* program segment for the required function */
printf ("enter the enrolment number of the student");
scanf ("% s", &enrolno);
EXEC SQL
        SELECT  name,  phone,  prog-code INTO
                :name, :phone, :p-code
        FROM STUDENT
        WHERE  enrolno = :enrolno;
        If (SQLCODE ==0)
            printf  ("%d,  %s, %s, %s", enrolno, name, phone, p-code)
            else
            printf ("Wrong Enrolment Number");
```

Please note the following points in the program above:

- The program is written in the host language 'C' and contains embedded SQL statements.
- Although in the program an SQL query (SELECT) has been added. You can embed any DML, DDL or views statements.
- The distinction between an SQL statement and host language statement is made by using the key word EXEC SQL; thus, this key word helps in identifying the Embedded SQL statements by the pre-compiler.
- Please note that the statements including (EXEC SQL) are terminated by a semi-colon (;),
- As the data is to be exchanged between a host language and a database, there is a need of shared variables that are shared between the environments. Please note that enrolno[10], name[20], p-code[4]; etc. are shared variables, colon (:) declared in 'C'.

- Please note that the shared host variables enrolno is declared to have char[10] whereas, an SQL attribute enrolno has only char[9]. Why? Because in 'C' conversion to a string includes a '\ 0' as the end of the string.
- The type mapping between 'C' and SQL types is defined in the following table:

| 'C' TYPE | SQL TYPE |
|---|---|
| long | INTEGER |
| short | SMALLINT |
| float | REAL |
| double | DOUBLE |
| char [ i+1] | CHAR (i) |

- Please also note that these shared variables are used in SQL statements of the program. They are prefixed with the colon (:) to distinguish them from database attribute and relation names. However, they are used without this prefix in any C language statement.
- Please also note that these shared variables have almost the same name (except p-code) as that of the attribute name of the database. The prefix colon (:) this distinguishes whether we are referring to the shared host variable or an SQL attribute. Such similar names is a good programming convention as it helps in identifying the related attribute easily.
- Please note that the shared variables are declared between BEGIN DECLARE SECTION and END DECLARE SECTION and there typed is defined in 'C' language.

Two more shared variables have been declared in 'C'. These are:

- SQLCODE as int
- SQLSTATE as char of size 6
- These variables are used to communicate errors and exception conditions between the database and the host language program. The value 0 in SQLCODE means successful execution of SQL command. A value of the SQLCODE =100 means 'no more data'. The value of SQLCODE if less than 0 indicates an error. Similarly, SQLSTATE is a 5 char code the 6$^{th}$ char is for '\0' in the host language 'C'. Value "00000" in an SQLSTATE indicate no error. You can refer to SQL standard in more detail for more information.
- In order to execute the required SQL command, connection with the database server need to be established by the program. For this, the following SQL statement is used:

  CONNECT <name of the server> AS <name of the connection>
  AUTHORISATION <username, password>,

  TO DISCONNECT we can simply say

  DISCONNECT <name of the connection>;

However, these statements need to be checked in the commercial database management system, which you are using.

*Execution of SQL query in the given program*: To create the SQL query, first, the given value of enrolment number is transferred to SQL attribute value, the query then is executed and the result, which is a single tuple in this case, is transferred to shared host variables as indicated by the key word INTO after the SELECT statement.

The SQL query runs as a standard SQL query except the use of shared host variables. Rest of the C program has very simple logic and will print the data of the students whose enrolment number has been entered.

Please note that in this query, as the enrolment number is a key to the relation, only one tuple will be transferred to the shared host variables. But what will happen if more than one tuple results on the execution of embedded SQL query. Such situations are handled with the help of a cursor. Let us discuss such queries in more detail.

### 3.3.2 Cursors and Embedded SQL

Let us first define the terminate 'cursor'. The Cursor can be defined as a pointer to the current tuple. It can also be defined as a portion of RAM allocated for the internal processing that has been set aside by the database server for database interactions. This portion may be used for query processing using SQL. But, what size of memory is to be allotted for the cursor? Ideally, the size allotted for the cursor should be equal to the memory required to hold the tuples that result from a query. However, the available memory puts a constraint on this size. Whenever a query results in a number of tuples, we can use a cursor to process the currently available tuples one by one. How? Let us explain the use of the cursor with the help of an example:

Since most of the commercial RDBMS architectures are client-server architectures, on execution of an embedded SQL query, the resulting tuples are cached in the cursor. This operation is performed on the server. Sometimes the cursor is opened by RDBMS itself – these are called **implicit** cursors. However, in embedded SQL you need to declare these cursors explicitly – these are called **explicit cursors.** Any cursor needs to have the following operations defined on them:

DECLARE – to declare the cursor
OPEN AND CLOSE - to open and close the cursor
FETCH – get the current records one by one till end of tuples.

In addition, cursors have some attributes through which we can determine the state of the cursor. These may be:

ISOPEN – It is true if cursor is OPEN, otherwise false.
FOUND/NOT FOUND – It is true if a row is fetched successfully/not successfully.
ROWCOUNT – It determines the number of tuples in the cursor.

Let us explain the use of the cursor with the help of an example:

Example: Write a C program segment that inputs the final grade of the students of MCA programme.

Let us assume the relation:

STUDENT (enrolno:char(9), name:Char(25), phone:integer(12),
prog-code:char(3)); grade: char(1));

The program segment is:

```
/* add proper include statements*/
/*declaration in C program */

EXEC SQL BEGIN DECLARE SECTION;
        Char enrolno[10], name[26], p-code[4], grade /* grade is just one character*/
        int phone;
        int SQLCODE;
        char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;

/* The connection needs to be established with SQL*/
/* program segment for the required function */
```

```
printf ("enter the programme code);
scanf ("%s, &p-code);
EXEC SQL DECLARE CURSOR GUPDATE
        SELECT enrolno, name, phone, grade
        FROM STUDENT
        WHERE progcode =: p-code
        FOR UPDATE OF grade;
EXEC SQL OPEN GUPDATE;
EXEC SQL FETCH FROM GUPDATE
        INTO :enrolno, :name, :phone, :grade;
WHILE (SQLCODE==0) {
        printf ("enter grade for enrolment number, "%s", enrolno);
        scanf ("%c", grade);
        EXEC SQL
                UPDATE STUDENT
                SET grade=:grade
                WHERE CURRENT OF GUPDATE
        EXEC SQL FETCH FROM GUPDATE;
}
EXEC SQL CLOSE GUPDATE;
```

- Please note that the declared section remains almost the same. The cursor is declared to contain the output of the SQL statement. Please notice that in this case, there will be many tuples of students database, which belong to a particular programme.
- The purpose of the cursor is also indicated during the declaration of the cursor.
- The cursor is then opened and the first tuple is fetch into shared host variable followed by SQL query to update the required record. Please note the use of CURRENT OF which states that these updates are for the current tuple referred to by the cursor.
- WHILE Loop is checking the SQLCODE to ascertain whether more tuples are pending in the cursor.
- Please note the SQLCODE will be set by the last fetch statement executed just prior to while condition check.

How are these SQL statements compiled and error checked during embedded SQL?

- The SQL pre-compiler performs the type of checking of the various shared host variables to find any mismatches or errors on each of the SQL statements. It then stores the results into the SQLCODE or SQLSTATE variables.

Is there any limitation on these statically embedded SQL statements?

They offer only limited functionality, as the query must be known at the time of application development so that they can be pre-compiled in advance. However, many queries are not known at the time of development of an application; thus we require dynamically embedded SQL also.

### 3.3.3 Dynamic SQL

Dynamic SQL, unlike embedded SQL statements, are built at the run time and placed in a string in a host variable. The created SQL statements are then sent to the DBMS for processing. Dynamic SQL is generally slower than statically embedded SQL as they require complete processing including access plan generation during the run time.

However, they are more powerful than *embedded SQL* as they allow run time application logic. The basic advantage of using dynamic embedded SQL is that we need not compile and test a new program for a new query.

Let us explain the use of dynamic SQL with the help of an example:

Example: Write a dynamic SQL interface that allows a student to get and modify permissible details about him/her. The student may ask for subset of information also. Assume that the student database has the following relations.

STUDENT (enrolno**,** name, dob)

RESULT (enrolno, coursecode, marks)

In the table above, a student has access rights for accessing information on his/her enrolment number, but s/he cannot update the data. Assume that user names are enrolment number.

Solution: A sample program segment may be (please note that the syntax may change for different commercial DBMS).

```
/* declarations in SQL */
EXEC SQL BEGIN DECLARE SECTION;
        char inputfields (50);
        char tablename(10)
        char sqlquery ystring(200)
EXEC SQL END DECLARE SECTION;
        printf  ("Enter the fields you want to see \n");
        scanf ("SELECT%s", inputfields);
        printf ("Enter the name of table STUDENT or RESULT");
        scanf ("FROM%s", tablename);
        sqlqueryystring = "SELECT" +inputfields +" "+
                "FROM" + tablename
                + "WHERE enrolno + :USER"
/*Plus is used as a symbol for concatenation operator; in some DBMS it may be ||*/
/* Assumption: the user name is available in the host language variable USER*/

EXEC SQL PREPARE sqlcommand FROM :sqlqueryystring;
EXEC SQL EXECUTE sqlcommand;
```

Please note the following points in the example above.

- The query can be entered completely as a string by the user or s/he can be suitably prompted.
- The query can be fabricated using a concatenation of strings. This is language dependent in the example and is not a portable feature in the present query.
- The query modification of the query is being done keeping security in mind.
- The query is prepared and executed using a suitable SQL EXEC commands.

### 3.3.4 SQLJ

Till now we have talked about embedding SQL in C, but how can we embed SQL statements into JAVA Program? For this purpose we use SQLJ. In SQLJ, a preprocessor called SQLJ translator translates SQLJ source file to JAVA source file. The JAVA file compiled and run on the database. Use of SQLJ improves the productivity and manageability of JAVA Code as:

- The code becomes somewhat compact.
- No run-time SQL syntax errors as SQL statements are checked at compile time.
- It allows sharing of JAVA variables with SQL statements. Such sharing is not possible otherwise.

Please note that SQLJ cannot use dynamic SQL. It can only use simple embedded SQL. SQLJ provides a standard form in which SQL statements can be embedded in

JAVA program. SQLJ statements always begin with a #sql keyword. These embedded SQL statements are of two categories – Declarations and Executable Statements.

Declarations have the following syntax:

```
#sql <modifier> context context_classname;
```

The executable statements have the following syntax:

```
#sql {SQL operation returning no output};
        OR
#sql result = {SQL operation returning  output};
```

**Example:**

Let us write a JAVA function to print the student details of student table, for the student who have taken admission in 2005 and name are like 'Shyam'. Assuming that the first two digits of the 9-digit enrolment number represents a year, the required input conditions may be:

- The enrolno should be more than "05000000" and
- The name contains the sub string "Shyam".

Please note that these input conditions will not be part of the Student Display function, rather will be used in the main ( ) function that may be created separately by you. The following display function will accept the values as the parameters supplied by the main ( ).

```
Public void  DISPSTUDENT (String enrolno, String name, int phone)
{
    try {
                if ( name equals ( " " ) )
                    name =  "%";
                if (enrolno  equals (" ") )
                    enrolno  = "%";
                 SelRowIter       srows = null;
                # sql srows = { SELECT Enrolno, name, phone
                        FROM STUDENT
                 WHERE   enrolno > :enrolno AND name like :name
                  };
                 while ( srows.next ( ) ) {
                        int enrolno          =          srows. enrolno ( );
                        String name          =          srows.name ( );

                System.out.println ( "Enrollment_No  =  " + enrolno);
                System.out.println ("Name =" +name);
                System.out.println ("phone =" +phone);
            }
    } Catch (Exception e) {
            System.out.println ( " error accessing database" + e.to_string);
    }
}
```

### ☞ Check Your Progress 2

1) A University decided to enhance the marks for the students by 2 in the subject MCS-043 in the table: RESULT (enrolno, coursecode, marks). Write a segment of embedded SQL program to do this processing.

……

……

……

2) What is dynamic SQL?

……

……

……

3) Can you embed dynamic SQL in JAVA?

……

……

……

## 3.4 STORED PROCEDURES AND TRIGGERS

In this section, we will discuss some standard features that make commercial databases a strong implementation tool for information systems. These features are triggers and stored procedures. Let us discuss about them in more detail in this section.

### 3.4 .1 Stored Procedures

Stored procedures are collections of small programs that are stored in compiled form and have a specific purpose. For example, a company may have rules such as:

- A code (like enrolment number) with one of the digits as the check digit, which checks the validity of the code.
- Any date of change of value is to be recorded.

These rules are standard and need to be made applicable to all the applications that may be written. Thus, instead of inserting them in the code of each application they may be put in a stored procedure and reused.

The use of procedure has the following advantages from the viewpoint of database application development.

- They help in removing SQL statements from the application program thus making it more readable and maintainable.
- They run faster than SQL statements since they are already compiled in the database.

Stored procedures can be created using CREATE PROCEDURE in some commercial DBMS.

**Syntax:**

CREATE [or replace] PROCEDURE [user]PROCEDURE_NAME

[(argument datatype

[, argument datatype]….)]

BEGIN

Host Language statements;

END;

For example, consider a data entry screen that allows entry of enrolment number, first name and the last name of a student combines the first and last name and enters the complete name in upper case in the table STUDENT. The student table has the following structure:

STUDENT (enrolno:char(9), name:char(40));

The stored procedure for this may be written as:

```
CREATE PROCEDURE studententry (
        enrolment   IN  char (9);
        f-name      INOUT  char (20);
        l-name      INOUT   char (20)
BEGIN
 /* change all the characters to uppercase and trim the length */
        f-name  TRIM =  UPPER (f-name);
        l-name   TRIM =  UPPER (l-name);
        name    TRIM  =  f-name || . . || l-name;
        INSERT INTO CUSTOMER
        VALUES (enrolment, name);
END;
```

INOUT used in the host language indicates that this parameter may be used both for input and output of values in the database.

While creating a procedure, if you encounter errors, then you can use the ***show errors*** command. It shows all the error encountered by the most recently created procedure object.

You can also write an SQL command to display errors. The syntax of finding an error in a commercial database is:

```
SELECT  *
FROM USER_ERRORS
WHERE Name='procedure name' and type='PROCEDURE';
```

Procedures are compiled by the DBMS. However, if there is a change in the tables, etc. referred to by the procedure, then the procedure needs to be recompiled. You can recompile the procedure explicitly using the following command:

ALTER PROCEDURE procedure_name COMPILE;

You can drop a procedure by using DROP PROCEDURE command.

### 3.4.2 Triggers

Triggers are somewhat similar to stored procedures except that they are activated automatically. When a trigger is activated? A trigger is activated on the occurrence of a particular event. What are these events that can cause the activation of triggers?

These events may be database update operations like INSERT, UPDATE, DELETE etc. A trigger consists of these essential components:

- An event that causes its automatic activation.
- The condition that determines whether the event has called an exception such that the desired action is executed.
- The action that is to be performed.

Triggers do not take parameters and are activated automatically, thus, are different to stored procedures on these accounts. Triggers are used to implement constraints among more than one table. Specifically, the triggers should be used to implement the constraints that are not implementable using referential integrity/constraints. An instance, of such a situation may be when an update in one relation affects only few tuples in another relation. However, please note that you should not be over enthusiastic for writing triggers – if any constraint is implementable using declarative constraints such as PRIMARY KEY, UNIQUE, NOT NULL, CHECK, FOREIGN KEY, etc. then it should be implemented using those declarative constraints rather than triggers, primarily due to performance reasons.

You may write triggers that may execute once for each row in a transaction – called Row Level Triggers or once for the entire transaction Statement Level Triggers. Remember, that you must have proper access rights on the table on which you are creating the trigger. For example, you may have all the rights on a table or at least have UPDATE access rights on the tables, which are to be used in trigger. The following is the syntax of triggers in one of the commercial DBMS:

CREATE TRIGGER <trigger_name>
[BEFORE | AFTER|
<Event>
ON <tablename>
[WHEN <condition> | FOR EACH ROW]

<Declarations of variables if needed is – may be used when creating trigger using host language>

BEGIN
    <SQL statements OR host language SQL statements>

[EXCEPTION]

    <Exceptions if any>
END:

Let us explain the use of triggers with the help of an example:

**Example:**

Consider the following relation of a students database

STUDENT(enrolno, name, phone)
RESULT (enrolno, coursecode, marks)
COURSE (course-code, c-name, details)

Assume that the marks are out of 100 in each course. The passing marks in a subject are 50. The University has a provision for 2% grace marks for the students who are failing marginally – that is if a student has 48 marks, s/he is given 2 marks grace and if a student has 49 marks then s/he is given 1 grace mark. Write the suitable trigger for this situation.

Please note the requirements of the trigger:

Event: UPDATE of marks

OR
INSERT of marks
Condition: When student has 48 OR 49 marks

Action: Give 2 grace marks to the student having 48 marks and 1 grace mark to the student having 49 marks.

The trigger for this thus can be written as:

```
CREATE TRIGGER grace
AFTER INSERT OR UPDATE OF marks ON RESULT
WHEN (marks = 48 OR marks =49)
UPDATE RESULT
SET marks =50;
```

We can drop a trigger using a DROP TRIGGER statement

```
DROP TRIGGER trigger_name;
```

The triggers are implemented in many commercial DBMS. Please refer to them in the respective DBMS for more details.

## 3.5 ADVANCED FEATURES OF SQL

The latest SQL standards have enhanced SQL tremendously. Let us touch upon some of these enhanced features. More details on these would be available in the sources mentioned in 'further readings'.

**SQL Interfaces:** SQL also has a good programming level interfaces. The SQL supports a library of functions for accessing a database. These functions are also called the Application Programming Interface (API) of SQL. The advantage of using an API is that it provides flexibility in accessing multiple databases in the same program irrespective of DBMS, while the disadvantage is that it requires more complex programming. The following are two common functions called interfaces:

SQL/CLI (SQL – call level interface) is an advanced form of Open Database Connectivity (ODBC).

Java Database Connectivity (JDBC) – allows object-oriented JAVA to connect to the multiple databases.

**SQL support for object-orientation:** The latest SQL standard also supports the object-oriented features. It allows creation of abstract data types, nested relations, object identifies etc.

Interaction with Newer Technologies: SQL provides support for XML (eXtended Markup Language) and Online Analytical Processing (OLAP) for data warehousing technologies.

### ☞ Check Your Progress 3

1) What is stored procedure?

……

……

……

2) Write a trigger that restricts updating of STUDENT table outside the normal working hours/holiday.

……

……

……

## 3.6 SUMMARY

This unit has introduced some important advanced features of SQL. The unit has also provided information on how to use these features.

An assertion can be defined as the general constraint on the states of the database. These constraints are expected to be satisfied by the database at all times. The assertions can be stored as stored procedures.

Views are the external schema windows of the data from a database. Views can be defined on a single table or multiple tables and help in automatic security of hidden data. All the views cannot be used for updating data in the tables. You can query a view.

The embedded SQL helps in providing a complete host language support to the functionality of SQL, thus making application programming somewhat easier. An embedded query can result in a single tuple, however, if it results in multiple tuple then we need to use cursors to perform the desired operation. Cursor is a sort of pointer to the area in the memory that contains the result of a query. The cursor allows sequential processing of these result tuples. The SQLJ is the embedded SQL for JAVA. The dynamic SQL is a way of creating queries through an application and compiling and executing them at the run time. Thus, it provides dynamic interface and hence the name.

Stored procedures are the procedures that are created for some application purpose. These procedures are precompiled procedures and can be invoked from application programs. Arguments can be passed to a stored procedure and it can return values. A trigger is also like a stored procedure except that it is invoked automatically on the occurrence of a specified event.

You should refer to the books further readings list for more details relating to this unit.

## 3.7 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) The constraint is a simple Range constraint and can easily be implemented with the help of declarative constraint statement. (You need to use CHECK CONSTRAINT statement). Therefore, there is no need to write an assertion for this constraint.

2)
```
CREATE VIEW avgmarks AS (
  SELECT subjectcode, AVG(smarks)
  FROM MARKS
  GROUP BY subjectcode);
```

3) No, the view is using a GROUP BY clause. Thus, if we try to update the subjectcode. We cannot trace back a single tuple where such a change needs to take place. Please go through the rules for data updating though views.

## Check Your Progress 2

1)

```
/*The table is RESULT (enrolno.coursecode, marks). */
EXEC SQL BEGIN DECLARE SECTION;
        char enrolno [10], coursecode[7]; /* grade is just one character*/
        int marks;
        char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/*The connection needs to be established with sQL*/
/* program segment for the required function*/
        printf ("enter the course code for which 2 grace marks are to be added")
        scanf ("%s", &coursecode);
EXEC SQL DECLARE CURSOR GRACE
            SELECT enrolno, coursecode, marks
            FROM RESULT
            WHERE coursecode=:coursecode
            FOR UPDATE OF marks;
EXEC SQL OPEN GRACE;
EXEC SQL FETCH FROM GRACE
            INTO :enrolno, :coursecode, :marks;
WHILE (SQL CODE==0) {
            EXEC SQL
                    UPDATE RESULT
                    SET marks = marks+2
                    WHERE CURRENT OF GRACE;
            EXEC SQL FETCH FROM GRACE;
}
EXEC SQL CLOSE GRACE;
```

An alternative implementation in a commercial database management system may be:

```
DECLARE CURSOR grace IS
            SELECT enrolno, coursecode, marks
            FROM RESULT
            WHERE coursecode ='MCS043';
str_enrolno RESULT.enrolno%type;
str_coursecode RESULT.coursecode%type;
str_marks RESULT.marks%type;
BEGIN
            OPEN  grace;
IF GRACE %OPEN THEN
            LOOP
                FETCH grace INTO str_enrolno, str_coursecode, str_marks;
                Exit when grace%NOTFOUND;
                UPDATE student SET marks=str_marks +2;
                INSERT INTO resultmcs-43 VALUES (str_enrolno,
                str_coursecode, str_marks);
            END LOOP;
            COMMIT;
            CLOSE grace;
            ELSE
                Dbms_output.put_line ('Unable to open cursor');
            END IF;
END;
```

2) Dynamic SQL allows run time query making through embedded languages. The basic step here would be first to create a valid query string and then to execute that query string. Since the queries are compiled and executed at the run time thus, it is slower than simple embedded SQL.

3) No, at present JAVA cannot use dynamic SQL.

## Check Your Progress 3

1) Stored procedure is a compiled procedure in a host language that has been written for some purpose.

2) The trigger is some pseudo DBMS may be written as:

```
CREATE  TRIGGER resupdstrudent
        BEFORE INSERT OR UPDATE OR DELETE ON STUDENT
BEGIN
    IF (DAY ( SYSDATE) IN ('SAT', 'SUN')) OR
        (HOURS (SYSDATE) NOT BETWEEN  '09:00' AND 18:30')
    THEN
        RAISE_EXCEPTION AND OUTPUT ('OPERATION NOT
                                    ALLOWED AT THIS TIME/DAY');
    END IF;
END
```

Please note that we have used some hypothetical functions, syntax of which will be different in different RDBMS.