



BACHELOR OF COMPUTER APPLICATIONS SEMESTER 3

DCA2102 DATABASE MANAGEMENT SYSTEM

Unit 6

SQL - 1

Table of Contents

SL No	Topic	Fig No / Table / Graph	SAQ / Activity	Page No
1	Introduction to SQL	-	-	3
	1.1 Objectives	-	-	
2	Categories of SQL Commands	-	1	4 - 6
3	Data Definition	-	2	7 - 9
4	Data Manipulation Statements	-	3	10 - 23
	4.1 SELECT – The Basic Form	-	-	
	4.2 Subqueries	-	-	
	4.3 GROUP BY Feature	-	-	
	4.4 Updating the Database	-	-	
	4.5 Data Definition Facilities	-	-	
5	Summary	-	-	24
6	Terminal Questions	1 2	-	24 - 25
7	Answers	-	-	25 - 26

1. INTRODUCTION TO SQL

SQL is an acronym for Structured Query Language. It is available in a number of database management packages based on the relational model of data, for example, in DB2 of the IBM and UNIFY of the UNIFY corporation.

Originally defined by D.D. Chamberlain in 1974, SQL underwent a number of modifications over the years. Today, SQL has become an official ANSI standard.

It allows for data definition, manipulation, and data control for a relational database. The data definition facilities of SQL permit the definition of relations and various alternative views of relations. Further, the data control facility gives features for one user to authorize other users to access his data. This facility also permits assertions to be made about data integrity. All the three major facilities of SQL, namely, data manipulation, data definition, and data control are bound together in one integrated language framework.

1.1 Objectives:

After going through Unit 6, you should be able to:

- ❖ *Differentiate among SQL commands*
- ❖ *List data manipulation commands*
- ❖ *List data definition commands*
- ❖ *Make queries using data manipulation commands*
- ❖ *Understand different views and its definition*
- ❖ *Understand embedded SQL and Transactions*

2. CATEGORIES OF SQL COMMANDS

SQL commands can be roughly divided into three major categories with regard to their functionality. Firstly, there are those used to create and maintain the database structure. The second category includes those commands that manipulate the data in such structures, and thirdly there are those that control the use of the database. To have all this functionality in a single language is a clear advantage over many other systems straightway, and must certainly contribute largely to the rumor of it being easy to use. It's worth naming these three fundamental types of commands for future reference. Those that create and maintain the database are grouped into the class called DDL or Data Definition Language statements and those used to manipulate data in the tables, of which there are four, are the DML or Data Manipulation Language commands. To control usage of the data the DCL commands (Data Control Language) are used, and it is these three in conjunction plus one or two additions that define SQL. There are therefore no environmental statements, as one finds so irritating in COBOL: for example, no statements to control program flow (if/then/else, perform, goto) and of course, no equivalent commands to open and close files, and read individual records. At this level, it is easy to see where SQL gets its send-user- tool and easy-to-use tags.

The Data Definition Statements

To construct and administer the database there are two major DDL statements – CREATE and DROP, which form the backbone of many commands:

CREATE DATABASE to create a database DROP DATABASE to remove a database
CREATE TABLE to create a table DROP TABLE to drop a table
CREATE INDEX to create an index on a column DROP INDEX to drop an index
CREATE VIEW to create a view DROP VIEW to drop a view.

There may be some additional ones, such as ALTER TABLE or MODIFY DATABASE, which are vendor-specific.

The Data Manipulation Statements

To manipulate data in tables directly or through views, we use the four standard DML statements:

SELECT DELETE INSERT UPDATE

These statements are now universally accepted, as is their functionality, although the degree to which these commands support this functionality varies somewhat between products. Compare the functionality of different implementations of UPDATE for example.

Data Control

This deals with three issues:

a) Recovery and Concurrency

Concurrency is concerned with the manner in which multiple users operate upon the database.

Each user can either reflect the updates of a transaction by using the COMMIT or can cancel all the updates of a transaction by using ROLLBACK.

b) Security

Security has two aspects to it.

The first is the VIEW mechanism. A view of a relation can be created which hides the sensitive information and defines only that part of a relation that should be visible. A user can then be allowed to access this view.

```
CREATE VIEW LOCAL AS  
SELECT * FROM SUPPLIER  
WHERE SUPPLIER.CITY = 'Delhi'
```

The above view reveals only the suppliers of Delhi.

The second is by using GRANT operation. This shall grant one or more access rights to perform the data manipulative operations on the relations.

c) Integrity Constraints

For example, one can specify that an attribute of a relation will not take on null values.

Self-Assessment Questions – 1

1. SQL commands can be roughly divided into_____ major categories with regard to their functionality.
2. To construct and administer the database there are_____major DDL statements.
3. To manipulate data in tables directly or through views we use the four standard _____statements.
4. _____ is concerned with the manner in which multiple users operate upon the data base.
5. A_____of a relation can be created which hides the sensitive information and defines only that part of a relation that should be visible.



3. DATA DEFINITION

Data definition in SQL is via the create statement. The statement can be used to create a table, index, or view (i.e., a virtual table based on existing tables). To create a table, the create statement specifies the name of the table and the names and data types of each column of the table. Its format is:

create table< relation >(< attribute list >)

where the attribute list is specified as:

< attribute list > :: = < attribute name > (< data type >)[not null] < attributelist >

The data types supported by SQL depend on the particular implementation. However, the following data types are generally included: integer, decimal, real (i.e., floating point values), and character strings, both of fixed size and varying length. A number of ranges of values for the integer data type are generally supported, for example, integer and short int. The decimal value declaration requires the specification of the total number of decimal digits for the value and (optionally), the number of digits on the right of the decimal point. The number of fractional decimal digits is assumed to be zero if only the total number of digits is specified.

< data type >:: = < integer > | < short int > | < char (n) > | < float > | < decimal (p[,q]) >

In addition, some implementations can support additional data types such as bit strings, graphical strings, logical, data, and time. Some DBMSs support the concept of date. One possible implementation of date could be as eight unsigned decimal digits representing the data in the yyyyymmdd format. Here yyyy represents the year, mm represents the month and dd represents the day. Two dates can be compared to find the one that is larger and hence occurring later. The system ensures that only the legal date values are inserted (20081016 for the date would be illegal) and functions are provided to perform operations such as adding a number of days to a date to come up with another date or subtracting a date from the current date to find the number of days, months, or years. Date constants are provided in either the format is given above or as a character string in one of the following formats: mm/dd/yy; mm/dd/yyyy; dd-mmm-yy; dd-mmm-yyyy. In this text, we represent a date constant as eight unsigned decimal digits in the format yyyyymmdd.

The employee relation for the hotel database can be defined using the create table statement given below. Here, the Empl_No is specified to be not null to disallow this unique identifier from having a null value. SQL supports the concept of null values and, unless a column is declared with the not null option, it could be assigned a null value.

```
create table EMPLOYEE
(
  Empl_No,   integer not null,
  Name      char (25),
  Skill     char(20),
  Pay-Rate  decimal
)
```

The definition of an existing relation can be altered by using the **alter** statement. This statement allows a row column to be added to an existing relation. The existing tuples of the altered relation are logically considered to be assigned the null value for the added column. The physical alteration occurs to a tuple only during an update of the record.

```
alter table existing-table-name
add column-name-data-type[ ]
alter table EMPLOYEE
add Phone_Number decimal (10)
```

The create index statement allows the creation of an index for an already existing relation. The columns to be used in the generation of the index are also specified. The index is named and the ordering for each column used in the index can be specified as either ascending or descending. The cluster option could be specified to indicate that the records are to be placed in physical proximity to each other. The unique option specifies that only one record could exist at any time with a given value for the column(s) specified in the statement to create the index. (Even though this is just an access aid and a wrong place to declare the primary key). Such columns, for instance, could form the primary key of the relation and hence duplicate tuples are not allowed. One case is the ORDER relation where the key is the combination of the attribute Bill#, Dish#. In the case of an existing relation, an attempt to create an index

with the unique option will not succeed if the relation does not satisfy this uniqueness criterion. The syntax of the create index statement is shown below:

create [unique] index name-of-index

on existing-table-name

(column-name [ascending or descending]

[, column-name[order]....])

[cluster]

The following statement causes an index called emp index to be built on the column Name and Pay_Rate. The entries in the index are ascending by Name value and descending by Pay_Rate. In this example, there are no restrictions on the number of records with the same Name and Pay_Rate.

Create index empindex

on **EMPLOYEE** (Name asc, Pay_Rate desc);

An existing relation or index could be deleted from the database by the **drop** SQL statement. The syntax of the drop statement is as follows:

drop table existing-table-name;

drop index existing-index-name;

Self-Assessment Questions – 2

6. Data definition in SQL is via the _____ statement.
7. The data types supported by SQL depend on the particular _____
8. The definition of an existing relation can be altered by using the _____ statement.
9. An existing relation or index could be deleted from the database by the _____ SQL statement.

4. DATA MANIPULATION STATEMENTS

Data manipulation capabilities allow one to retrieve and modify the contents of the database. The most important of these is the SELECT operation which allows data to be retrieved from the database. The relation definitions that shall be used in the rest of the module are given below.

There are parts that are supplied by suppliers. S contains the details about each supplier. Turnover for a supplier is in terms of lakhs of rupees. Information regarding suppliers of specific parts is contained in SP, whereas information about the parts themselves is contained in P.

4.1 SELECT – The Basic Form

The select statement specifies the method of selecting the tuples of the relations(s). The tuples processed are from one or more relations specified by the from clause of the select statement. The basic form of SELECT is

Select < target list >
from < relation list >
[where < predicate >]

SELECT lists the attributes to be selected
FROM relations from which information is to be used
WHERE condition. The rows that qualify are those in which the condition evaluates to true.

Condition is a single predicate or a collection of predicates combined using the Boolean operators AND, OR, and NOT.

The column names following SELECT are to be retrieved from the relations specified in the FROM part. WHERE specifies the condition that the tuples must satisfy in order to be part of the result.

Below we shall state first the retrieval query in English and then specify its SQL equivalent.

Unqualified Retrieval

1. Get the part numbers of all the parts being supplied.

```
SELECT P#
FROM SP
```

```
P#
1
1
2
2
3
3
1
```

Part numbers getting repeated? That's right. SELECT does not eliminate duplicate rows (unlike the project operation of the relational algebra). In order to do that

2. Get the part numbers of all the parts being supplied with no duplication.

```
SELECT DISTINCT P#
FROM SP
```

```
P#
1
2
3
```

If all the columns of the relation are to be retrieved then one needn't list all of them. A* can be specified after SELECT to indicate retrieval of the entire relation.

3. Get full details of all suppliers.

```
SELECT *
FROM S
```

The ORDER BY clause

The result of a query can be ordered either in ascending (ASC) order or in descending (DESC) order.

4. Get the supplier numbers and turnover in descending order of turnover.

```
SELECT S#, TURNOVER
FROM S
ORDER BY TURNOVER DESC
```

S#	TURNOVER
11	100
12	70
10	50
13	20

Instead of a column name, the original position of the column in the result can be used. That is, the above query can be written as

5. Get the details of suppliers who operate from Bombay with a turnover

```
50.SELECT S.*
FROM S
WHERE CITY = 'BOMBAY' AND TURNOVER > 50
```

S#	SNAME	SCITY	TURNOVER
11	NARMADA	BOMBAY	100

The above form is a conjunction of comparison predicates. A comparison predicate is of the form

scalar-expr O scalar-expr

where O is any of the six relational operators

=, <, >, <=, >=

and a scalar expression is an arithmetic expression with operators as +, -

, *, / operands as col., function, constant

BETWEEN Predicate

6. Get the part numbers weighing between 25 and 35

```
SELECT P#, WEIGHT
FROM P
WHERE WEIGHT BETWEEN 25 AND 35
```

P#	WEIGHT
1	25
2	30

The use of BETWEEN gives the range within which the values must lie. If the value should lie outside a range then BETWEEN is to be preceded by NOT. For example,

```
SELECT P#
FROM P
WHERE WEIGHT NOT BETWEEN 25 AND 35
```

P#	WEIGHT
3	45

would retrieve all part numbers whose weight is less than 25 or greater than 35 as shown above.

LIKE Predicate

This predicate is used for pattern matching. A column of type **char** can be compared with a string constant. The use of the word LIKE doesn't look for an exact match but a form of wild string match. A % or - can appear in the string constant where

- % stands for a sequence of n (≥ 0) characters
- stands for a single character

Examples

ADDRESS LIKE '%Bangalore%' – ADDRESS should have Bangalore somewhat as a part of it if the match is to succeed.

```
STRANGE STRING LIKE '\-%' ESCAPE '\'

```

Here, the normal meaning of - is overridden with the use of the escape character. STRANGE above will match any string beginning with –

7. Get the names and cities of suppliers whose names begin with

```
CSELECT      SNAME   SCITY
FROM S
WHERE        SNAME LIKE 'C%'
SNAME       SCITY
CAUVERY     BANGALORE
```

When the data is to be retrieved from more than one relation, then both the relation names are specified in the FROM clause and the join condition in the WHERE part.

8. Get pairs of supplier numbers such that both operate from the same city.

```
SELECT FIRST.S#, SECOND.S#
FROM   S FIRST, S SECOND
WHERE  FIRST.CITY = SECOND.CITY
AND    FIRST.S# < > SECOND.S#
```

FIRST and SECOND are tuple variables, both ranging over S. The last line eliminates a supplier getting compared with him.

```
S#   S#
11   13
13   11
```

But, we see that suppliers with numbers 11 and 13 are getting compared twice. Can that be avoided? How about < instead of < >?

```
SELECT      FIRST.S#, SECOND.S#
FROM        S FIRST, S SECOND
WHERE       FIRST.CITY = SECOND.CITY
AND         FIRST.S# < SECOND.S#
```

IN Predicate

This is to be used whenever you want to test whether an attribute value is one of a set of values. For example,

9. Get the part numbers that cost 20, 30, or 40

```
rupees.SELECT      P.P#, SELLING PRICE
FROM              P
WHERE             SELLING PRICE IN (20,40,45)
```

P#	SELLING PRICE
2	45
3	45

It's a quicker way of specifying comparison. The format of the predicate is scalar-expr [NOT] IN (atom list)

4.2 Subqueries

The expression following WHERE can be either a simple predicate as explained above or it can be a query itself! This part of the query following WHERE is called a Subquery.

A subquery, which in turn is a query, can have its own subquery and the process of specifying subqueries can continue ad infinitum! More practically, the process ends once the query has been fully expressed as an SQL statement.

Subqueries can appear when using the comparison predicate, the IN predicate and when quantifiers are used.

Comparison Predicate

10. Get the supplier numbers of suppliers who are located in the same city as TAPI.

```
SELECT S.S#, SNAME
FROM S
WHERE S.CITY = (SELECT CITY
                FROM S
                WHERE SNAME = 'TAPI')
```

The inner select (subquery) retrieves the city of the supplier named TAPI. The outer select (the main one) then compares the city of each supplier in the supplier relation and picks up those where the comparison succeeds.

S#	SNAME
11	NARMADA
13	TAPI

Notice that the subquery appears after the comparison operator. The format of this form of expression

scalar-expr operator subquery

IN Predicate

In this form, the subquery selects a set of values. The outer query checks whether the value of a specified attribute is in this set.

11. Get the names of suppliers who supply part 2

```
SELECT  S.SNAME
FROM    S
WHERE   S.S# IN
        (SELECT  SP.S#
         FROM     SP
         WHERE    SP.P#= 2)
        SNAME
        CAUVERY
        NARMADA
```

The above query can be equivalently expressed as

```
SELECT  S.SNAME
FROM    S
WHERE   2 IN
        (SELECT  P#
         FROM     SP
         WHERE    S.S#=S#)
```

S# is unqualified and therefore, refers to SP. That is because every unqualified attribute name is implicitly qualified with the relation name from the nearest applicable FROM clause.

Quantified predicates

The two quantifiers that can be used are ALL and ANY. ANY stands for the existential quantifier and ALL for the universal quantifier.

Let's first look at ANY. It can be specified in a comparison predicate just following the comparison operator. That is,

scalar-expr O ANY subquery

The subquery is first evaluated to give a set of values. The above expression is true if the scalar-expr is O comparable with any of the values that form the result of the subquery.

12. Get the part numbers for parts whose cost is less than the current maximum cost.

```
SELECT P#, COST
FROM   P
WHERE  COST < ANY
      (SELECT COST
       FROM   P)
P#      COST
1        10
2        15
```

The inner select gets the cost of all the parts. In the outer select, a P# is selected if its cost is less than some element of the set selected in the earlier step.

4.3 Aggregate Functions

Some standard functions are defined in SQL and can be used when framing queries. There are five aggregate functions. These are

COUNT	–	number of values of a column
SUM	–	Sum of values in a column
AVG	–	Average of values in a column
MAX	–	Maximum of all the values in a column
MIN	–	Minimum of all the values in a column

If the function is followed by the word DISTINCT then unique values are used. On the other hand, if ALL follows the functions then all the values are used for evaluating the function. ALL is default.

COUNT(*) has a special meaning in that it counts the number of rows of a relation. COUNT in any other form must make use of DISTINCT. In other words, except when rows are counted, COUNT always returns the number of distinct values in a column.

13. Get the total number of suppliers

```
SELECT  COUNT(*)  
FROM    S
```

COUNT (*) counts the number of tuples of S and hence, the number of suppliers.

14. Get the total quantity of Part 2 that is supplied

```
SELECT      SUM(SP.QTY)  
FROM        SP  
WHERE       SP.P#=2
```

The answer is one of

a) 25 b) 40 c) 80 d) 100

15. Get the part numbers whose cost is greater than the average cost.

```
SELECT      P#  
FROM        P  
WHERE       COST >  
(SELECT     AVG (COST)  
FROM        P)
```

16. Get the names of suppliers who supply from a city where there is atleast one more supplier

```
SELECT      SNAME  
FROM S      FIRST  
WHERE       2 >=  
(SELECT     COUNT (CITY)  
FROM        S  
WHERE       CITY = FIRST.CITY)
```


4.4 GROUP BY Feature

This feature allows one to partition the result into a number of groups such that all rows of the group have the same value in some specified column.

17. Get the part number and the total quantity.

```
SELECT      P#, SUM(QTY)
FROM        SP
GROUP BY    P#
P#          SUM(QTY)

1           125
2           80
3           110
```

GROUP BY groups together all the rows which have the same value for P#. The function SUM is then applied to each group. That is, the result consists of a part number along with the total quantity in which it is supplied.

Whenever GROUP BY is used then the phrase WHERE is to be replaced by HAVING. The meaning of HAVING is the same as WHERE except that the condition is now applicable to each group.

18. Get the part numbers for parts supplied by more than one supplier.

```
SELECT      P#
FROM        SP
GROUP BY    P#
HAVING      COUNT(*) > 1
```

Each group contains one or more tuples which have the same part number. COUNT(*) is applied to each such group.

The result before COUNT(*) is applied is

```
P#
1
2
3
```

4.5 Updating the Database

The contents of the database can be modified by inserting a new tuple, deleting an existing tuple or changing the values of attributes of one or more tuples.

INSERT

The insertion facility allows new tuples to be inserted into given relations. Attributes which are not specified by the insertion statement are given null values. Consider

1. Add a part with 14, weight 10, colored red, with the cost and selling price as 20 and 60 respectively

```
INSERT INTO P:      < 14, 10, 'red', 20, 60 >
```

The tuple is inserted into P.

If all the fields are not known then a tuple can still be added. The attributes whose values are not specified will have a null value.

The tuple is inserted into P.

If all the fields are not known then a tuple can still be added. The attributes whose values are not specified will have a null value.

```
INSERT INTO P:  
< 15, 'green' >
```

The values for fields the weight, cost and the selling price which are not specified are assumed to be null.

2. Let us assume that there is a relation called RED-PART with one column P#.

```
INSERT INTO RED-PART  
SELECT      P#  
FROM        P  
WHERE       COLOUR = 'red'
```

The various attributes of P having red color are identified and inserted into the relation RED-PART.

DELETE

The deletion facility removes specified tuples from the database. Consider

1. Delete supplier 13

```
DELETE    S
WHERE     S# = 13
```

Since S# is the primary key only one tuple will be deleted from S.

2. Delete all suppliers who supply from Bangalore

```
DELETE    S
WHERE     SCITY = 'BANGALORE'
```

Here, more than one supplier can get deleted.

3. Delete all suppliers

```
DELETE    S
```

The definition of S exists but the relation is empty.

UPDATE

When columns are to be modified, SET clause is used. This clause specifies the update to be made to selected tuples.

1. Change the city of supplier 13 to Bangalore and increase the turnover by 20 lakhs.

```
UPDATE    S
SET CITY = 'BANGALORE'
TURNOVER = TURNOVER + 20
WHERE     S# = 13
```

2. Increase quantity by 10 for all supplies of red coloured parts.

```
UPDATE    SP
SET QTY = QTY + 10 WHERE P# IN
(SELECT   P#
FROM      P
WHERE     COLOUR = 'RED')
```

4.6 Data Definition Facilities

Data definition facilities permit users to create and drop relations, and define alternative views of relations.

CREATE statement allows to define a relation. The name of the relation to be created and its various fields together with their data types must be specified. If a certain attribute is barred from containing null values then a NONNULL specification must be made for it.

It must be noted that the word TABLE is used in this syntax instead of RELATION.

Example

```
CREATE TABLE DEPT  
(DNO CHAR (2) NONNULL,  
DNAME VARCHAR (12),  
LOC VARCHAR (12))
```

VIEW

A very important aspect of data definition is the ability to define alternative views of data. The process of specifying an alternative view is very similar to that of framing a query. The derived relation is stored and can be used thereafter as an object of the various commands. It is also possible to define other views on top of the newly created relation.

Example

```
DEFINE VIEW D50 AS  
SELECT EMPNO, NAME, JOB  
FROM EMP  
WHERE DNO = 50
```

D50 contains the employee number, name, and job of those employees who are in department 50.

Self-Assessment Questions – 3

10. Data manipulation capabilities allow one to _____ and modify contents of the data base.
11. The _____ statement specifies the method of selecting the tuples of the relations(s).
12. SELECT does not eliminate _____ rows.
13. The use of _____ gives the range within which the values must lie.
14. Like predicate is used for pattern _____.
15. Subqueries can appear when using the _____ predicate, the IN predicate and _____ when quantifiers are used.
16. ANY stands for the existential quantifier and ALL for the _____ quantifier.
17. There are _____ built-in functions in SQL.
18. In SQL _____ function has a special meaning in that it counts the number of rows of a relation.
19. When columns are to be modified _____ clause is used.



5. SUMMARY

This unit discussed that most commercial relational DBMSs support some form of the SQL data manipulation language, and this creates different dialects of SQL. We dealt that SQL commands can be roughly divided into three major categories: (i) commands to create and maintain the database structure (ii) commands that manipulate the data in such structures (iii) commands that control the use of the database. We discussed about subqueries, functions, GROUP BY features, updating the database, and data definition facilities.

6. TERMINAL QUESTIONS

1. Consider the insurance database of Figure 6.1, where the primary keys are underlined. Construct the following SQL queries for this relational database.
 - a) Find the total number of people who owned cars that were involved in accidents in 1989.
 - b) Find the number of accidents in which the cars belonging to "JohnSmith" were involved.
 - c) Add a new accident to the database; assume any values for required attributes.
 - d) Delete the Mazda belonging to "John Smith".

Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to \$3000.

person (driver-id#, name, address)
car (license, model, year)
accident (report-number, date, location)
owns (driver-id#, license)
participated (driver-id, car, report-number, damage-amount)

Fig 6.1: Insurance database

employee (employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)

Fig 6.2: Employee database

2. Consider the employee database of Figure 6.2, where the primary keys are underlined. Give an expression in SQL for each of the following queries.
- Find the names of all employees who work for First Bank Corporation.
 - Find the names and cities of residence of all employees who work for First Bank Corporation.
 - Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than \$10,000.
 - Find all employees in the database who live in the same cities as the companies for which they work.
3. Consider the relational database of Figure 6.2. Using SQL, define a view consisting of *manager-name* and the average salary of all employees who work for that manager. Explain why the database system should not allow updates to be expressed in terms of this view.
4. Describe the circumstances in which you would choose to use embedded SQL rather than SQL alone or only a general-purpose programming language.

7. ANSWERS

Self-Assessment Questions

- Three
- Two
- DML
- Concurrency
- View
- Create
- Implementation
- Alter
- Drop
- Retrieve
- Select
- Duplicate
- BETWEEN
- Matching
- Comparison

Terminal Questions

1. Refer the whole unit to construct SQL queries of the given insurance database and check it practically to confirm results.
2. Updates should not be allowed in view because there is no way to determine how to change the underlying data and define alternative views of data. (Refer section 6.4 for detail)
3. Writing queries in SQL is easier than coding the same queries in a general-purpose programming language. Embedded SQL is less complicated since it avoids the clutter of the ODBC or JDBC function calls, but requires a specialized pre-processor. (Refer whole unit for detail)

