12 Structured Query Language

In This Chapter

12.1 INTRODUCTION

The Structured Query Language (SQL) is a language that enables you to create and operate on relational databases, which are sets of related information stored in tables.

The database world is becoming increasingly integrated, and this has led to a clamor for a standard language that can be used to operate in many different kinds of computer environment. The *SQL* (Structured Query Language) has proved to be a standard language as it allows users to learn one set of commands and use it to create, retrieve, alter, and transfer information regardless of whether they are working on a PC, a workstation, a mini, or a mainframe.

SQL – Structured Query Language

SQL is a simple query language used for accessing, handling and managing data in relational databases.

There are numerous versions of *SQL*. The original version was developed at IBM's *San Jose Research Laboratory* (now the *Almanden Research Center*). This language, originally called *Sequel*, was implemented as part of the System R project in early 1970s. The *Sequel* language has evolved since then, and its name has changed to *SQL*. In 1986, the *American National Standards Institute* (*ANSI*) published an *SQL* standard that was updated again in 1992 ; latest ISO standard of SQL was released in 2008 and named as SQL:2008.

NOTE

Latest SQL standard as of now is SQL:2008, released in 2008.

SQL has clearly established itself as the standard relational database language. In this chapter, we present a briefed survey of *SQL*.

12.2 PROCESSING CAPABILITIES OF SQL

The *SQL* has proved to be a language that can be used by both casual users as well as skilled programmers. It offers a variety of processing capabilities, simpler ones of which may be used by the former and the more complex by the latter class of users.

The various processing capabilities of *SQL* are :

1. ***Data Definition Language*** (***DDL***). The *SQL* DDL provides commands for defining relation schemas, deleting relations, creating indexes, and modifying relation schemas.
2. ***Interactive Data Manipulation Language*** (***DML***). The *SQL* DML includes a query language based on both the relational algebra and the tuple relational calculus. It also includes commands to insert, delete, and modify tuples in the database.
3. ***Embedded Data Manipulation Language.*** The embedded form of *SQL* is designed for use within general-purpose programming languages such as PL/1, Cobol, Fortran, Pascal, and C.
4. ***View Definition***. The *SQL* DDL also includes commands for defining views.
5. ***Authorization***. The *SQL* DDL includes commands for specifying access rights to relations and views.
6. ***Integrity***. The *SQL* provides (limited) forms of integrity checking. Future products and standards of *SQL* are likely to include enhanced features for integrity checking.
7. ***Transaction control***. *SQL* includes commands for specifying the beginning and ending of transactions[1] along with commands to have a control over transaction processing.

12.3 DATA DEFINITION LANGUAGE

Data Dictionary A Data Dictionary is a file that contains "metadata" *i.e.*, "data about data".

DDL – Data Definition Language

The DDL provides a set of definitions to specify the storage structure and access methods used by the database system.

A database scheme is specified by a set of definitions which are expressed by a special language called a *data definition language* (DDL). The result of compilation of DDL statements is a set of tables which are stored in a special file called *data dictionary* (or *directory*).

Whenever data is read or modified in the database system, the *data directory* is consulted.

The DDL (Data Definition Language) provides a set of definitions to specify the storage structure and access methods used by the database system.

An ideal DDL should perform the following functions :

1. It should identify the types of data division such as *data item, segment, record,* and *data-base file.*
2. It should give a unique name to each *data-item-type, record-type, file-type, database,* and *other data subdivision.*
3. It should specify the proper data types.
4. It should specify how the record types are related to make structures.
5. It may define the type of encoding the program uses in the data items (*binary, character, bit, string,* etc.). This should not be confused with the encoding employed in physical representation.
6. It may define the length of the data items.

1. A transaction is a complete logical unit of work. For example, if we say that a file is to be *updated* , then this transaction will include *opening of file, reading of file, making the changes in it* and then finally *storing it back* . Thus this complete unit includes four small steps that make it a transaction.

7. It may define the range of values that a data-item can assume.
8. It may specify means of checking for errors in the data.
9. It may specify privacy locks for preventing unauthorized reading or modification of the data.
10. A logical data definition should not specify addressing, indexing, or searching techniques or specify the placement of data on the storage units, because these topics are in the domain of physical, not logical, organization.

The *SQL* commands covered later in this chapter, include DDL commands also.

12.4 DATA MANIPULATION LANGUAGE

After the database scheme has been specified and the database has been created, the data can be manipulated using a set of procedures which are expressed by a special language called a *data manipulation language* (DML). By data manipulation, we mean :

- the *retrieval* of information stored in the database.
- the *insertion* of new information into the database.
- the *deletion* of information from the database.
- the *modification* of data stored in the database.

Data Manipulation Language

A Data Manipulation Language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model.

The DMLs are basically of *two* types :

(*i*) *Procedural DMLs* require a user to specify *what* data is needed and *how* to get it.

(*ii*) *Non-procedural DMLs* require a user to specify *what* data is needed without specifying how to get it.

12.5 SQL PROCESSING

SQL is a language oriented specifically around relational databases. The *SQL* commands can operate on entire groups of tables as single objects and can treat any quantity of information extracted or derived from them as a single unit as well.

SQL actually consists of *three* sub-languages :

1. **DDL** – *Data Definition Language*. Covered under section 12.3.
2. **DML** – *Data Manipulation Language*. Covered under section 12.4.
3. **DCL** – *Data Control Language*. It is used to control access to the data base (by GRANTing / REVOKing etc.) and therefore essential to the system. We shall not cover DCL here as it is beyond the scope of syllabus.

12.5.1 Concept of Data Types

Since relational-database systems are based on the relationships between pieces of information, the various types of data must be clearly distinguished from one another, so that the appropriate processes and comparisons can be applied. In *SQL*, this is done by assigning each field a *data type* that indicates the kind of value the field will contain. All the values in a given field must be of same data type.

1. What is SQL ? What are various subdivisions of SQL ?
2. Give examples of some DDL commands and some DML commands.

The ANSI *SQL* standard recognizes only text and number types, whereas many commercial programs use other special types as well, for instance, **DATE** and **TIME** types. The exact ANSI data types are being listed below in Table 12.1.

Table 12.1 *ANSI SQL Data Types*

Class	Data Type	Description
Text	CHAR (or CHARACTER)	Values of this type must be enclosed in single quotes such as '*text*', '*example*' etc. Two adjacent single quotes (' ') inside the string will represent one single quote. For instance, *Ram's* will be written as '*Ram''s*'
Exact Numeric	DEC (or DECIMAL)	It can represent a fractional number such as 17.321, 214.003 etc. Here the size argument has two parts : *precision* and *scale*. The ***precision*** indicates how many significant digits the number is to have. The ***scale*** indicates the maximum number of digits to the right of the decimal point. The size given as (5, 2) indicates *precision* as 5 and *scale* as 2. Here the scale cannot exceed the precision.
-do-	NUMERIC	It is same as DECIMAL except that the maximum number of digits may not exceed the precision argument.
-do-	INT (or INTEGER)	It represents a number without a decimal point. Here the size argument is not used ; it is automatically set to an implementation dependent value.
-do-	SMALLINT	Same as INTEGER except that, depending upon the implementation, the default size may (or may not) be smaller than INTEGER.
Approximate Numeric	FLOAT	It represents a floating point number in base 10 exponential notation. The size argument consists of a single number specifying the minimum precision.
-do-	REAL	It is same as FLOAT, except that no size argument is used. The precision is set to an implementation dependent default value.
-do-	DOUBLE PRECISION (or DOUBLE)	Same as REAL, except that the implementation-defined precision for DOUBLE PRECISION must exceed the implementation-defined precision of REAL

In addition to ANSI data types, most implementations of *SQL* support **DATE** and **TIME** types.

Some major date formats you may encounter are :

Standard	Format	Example
International Standards Organization (ISO)	yyyy-mm-dd	2012-02-24
Japanese Industrial Standard (JIS)	yyyy-mm-dd	2012-02-24
IBM European Standard (EUR)	dd.mm.yyyy	24.02.2012
IBM USA Standard (USA)	mm/dd/yyyy	02/24/2012

Some major time formats used across *SQL* implementations are :

Standard	Format	Example
International Standard Organization (ISO)	hh-mm-ss	23-13-41
Japanese Industrial Standard (JIS)	hh-mm-ss	23-13-41
IBM European Standard (EUR)	hh.mm.ss	23.13.41
IBM USA Standard (USA)	hh.mm AM/PM	11.13 PM

Even DATE and TIME can be added, subtracted or compared as it can be done with other data types.

12.5.2 Various SQL Commands and Functions

The *SQL* provides a predefined set of commands that help us work on relational databases. This section deals with various *SQL* commands. But before discussing them, we must be familiar with the conventions and terminology used in *SQL* commands.

Keywords are words that have a special meaning in *SQL*. They are understood to be instructions. Here, the *SQL* keywords have been printed in capital letters. ***Commands***, or ***statements***, are instructions given by you to a *SQL* database. Commands consist of one or more logically distinct parts called ***clauses***. Clauses begin with a keyword for which they are generally named, and consist of keywords and arguments. Examples of clauses are "*FROM sales*" and "*WHERE value = 1500.00*", ***Arguments*** complete or modify the meaning of a clause. In the above examples, '*sales*' is the argument, and FROM is the keyword of FROM clause. Likewise '*value = 1500.00*' is the argument of the WHERE clause. ***Objects*** are structures in the database that are given names and stored in memory. They include *base tables*, *views*, and *indexes*[2].

Now that we are about to start *SQL* commands, following table (Table 12.2) summarises the symbols used in syntax statements.

Table 12.2 *Symbols Used in Syntax of Statements*

Symbol	*Meaning*
|	This is a symbolic way of saying "or". That is, whatever precedes this symbol may optionally be replaced by whatever follows it.
{ }	Everything enclosed in it, is treated as a unit for the purposes of evaluating |, . . , . . or other symbols.
[ ]	This means, everything enclosed in it is optional.
...	This means whatever precedes it may be repeated any number of times.
.,...	Whatever precedes this, may be repeated any number of times with the individual occurrences separated by commas.
< >	*SQL* and other special terms are in angle brackets.

12.5.2A CREATE TABLE Command

Tables are defined with the CREATE TABLE command. When a table is created, its columns are named, data types and sizes are supplied for each column. Each table must have at least one column. The syntax of CREATE TABLE command is :

```
CREATE TABLE <table-name>
   (<column name> <data type> [ ( <size>)],
   <column name> <data type> [ ( <size> ) ... ]) ;
```

To create an *employee* table whose scheme is as follows :

```
employee (ecode, ename, sex, grade, gross)
```

the *SQL* command will be

```
CREATE TABLE employee
(    ecode      integer,
     ename      char(20),
     sex        char(1),
     grade      char(2),
     gross      decimal ) ;
```

2. An index is an ordered list of single or grouped column values that stores the disk locations of the rows containing those values.

Constraint A Constraint is a condition or check applicable on a field or set of fields.

When you create a table, you can place constraints on the values that can be entered into its fields. If this is specified, *SQL* will reject any values that violate the criteria you define.

The two basic types of constraints are *column constraints* and *table constraints*. The difference between the two is that *column constraints* apply only to individual columns, whereas *table constraints* apply to groups of one or more columns. The following is the syntax for the CREATE TABLE command, expanded to include constraints :

```
CREATE TABLE <table name>
(    <column name><data type>[ (<size>) ] <column constraint>,
     <column name><data type>[ (<size>) ] <column constraint> ...
     <table constraint>(<column name>,[,<column name>... ] ) ...) ;
```

The fields given in parenthesis after the table constraint(s) are the fields to which they apply. The column constraints apply to the columns whose definitions they follow.

For example, if you write the keywords NOT NULL immediately after the data type (and size) of a column, this means the column can never have empty values (*i.e.*, NULL[3] values). Otherwise *SQL* will assume that NULLs are permitted. Consider the following *SQL* command :

```
CREATE TABLE employee
(    ecode     integer      NOT NULL ,
     ename     char (20)    NOT NULL ,
     sex       char (1)     NOT NULL ,
     grade     char (2),
     gross     decimal ) ;
```

See, NOT NULL constraint has been applied individually on columns ecode, ename and sex

The above command creates a table called *employee* in which *ecode* column (integer type) can never be employed as its definition is followed by keywords NOT NULL. Similarly, the columns *ename* (char (20)) and *sex* (char (1)) can never have NULL values. Any attempt to put NULL values in these columns will be rejected.

Different Constraints

These constraints ensure database integrity, thus are sometimes called *database integrity constraints*. A few of them are :

- Unique constraint
- Primary key constraint
- Default constraint
- Check constraint

1. Unique Constraint

This constraint ensures that no two rows have the same value in the specified column(s). For example, UNIQUE constraint applied on *ecode* of *employee* table ensures that no rows have the same *ecode* value, as shown below :

```
CREATE TABLE employee
(  ecode     integer      NOT NULL     UNIQUE ,
   ename     char (20)    NOT NULL,
   sex       char (1)     NUT NULL,
   grade     char (2),
   gross     decimal ) ;
```

See, multiple constraints (NOT NULL and UNIQUE constraints here) have been applied on one column by putting space in between. The comma (,) comes in the end of column definition.

3. NULL is a special keyword in SQL that depicts an empty value. A column having NULL is not empty but stores an empty value. Two NULLs cannot be added, subtracted or compared.

This constraint can be applied only to columns that have also been declared NOT NULL, however, this condition is implementation dependent.

2. Primary Key Constraint

This constraint declares a column as the primary key of the table. This constraint is similar to unique constraint except that only one column (or one group of columns) can be applied in this constraint. The primary keys cannot allow NULL values, thus, this constraint must be applied to columns declared as NOT NULL. Consider the following *SQL* statement :

```
CREATE TABLE employee
(   ecode       integer       NOT NULL   PRIMARY KEY ,
    ename       char (20)     NOT NULL,
    sex         char (1)      NUT NULL,
    grade       char (2),
    gross       decimal ) ;
```

3. Default Constraint

A default value can be specified for a column using the DEFAULT clause. When a user does not enter a value for the column (having default value), automatically the defined default value is inserted in the field.

Consider the following *SQL* statement :

```
CREATE TABLE employee
(  ecode        integer       NOT NULL PRIMARY KEY,
   ename        char (20)     NOT NULL,
   sex          char (1)      NOT NULL,
   grade        char (2)      DEFAULT = 'E1' ,
   gross        decimal  ) ;
```

According to above command, if no value is provided for *grade,* the default value of 'E1', will be entered. The datatype of the default value has to be compatible with the datatype of the column to which it is assigned. Insertion of NULL (as default value) is possible only if the column definition permits. (NOT NULL columns cannot have NULL as default). A column can have only one default value.

4. Check Constraint

This constraint limits values that can be inserted into a column of a table. For instance, consider the following *SQL* statement :

```
CREATE TABLE employee
(  ecode        integer       NOT NULL PRIMARY KEY,
   ename        char (20)     NOT NULL,
   sex          char (1)      NUT NULL,
   grade        char (2)      DEFAULT = 'E1',
   gross        decimal       CHECK (gross > 2000) ) ;
```

This statement ensures that the value inserted for *gross* must be greater than 2000.

When a check constraint involves more than one column from the same table, it is specified after all the columns have been defined.

For instance,

See, this constraint was referring to more than one columns, hence it has been defined in the end of table definition, after all the column definitions

```
CREATE TABLE items
(   icode       char (5)        NOT NULL PRIMARY KEY,
    descp       char (20)       NOT NULL,
    ROL         integer,
    QOH         integer,
    CHECK  (ROL<QOH )
) ;
```

This statement compares two columns *ROL* and *QOH*, thus, these two columns must be defined before this CHECK constraint.

NOTE

A *column constraint* is applicable only to a column, whereas a *table-constraint* is applicable to multiple columns.

A *column-constraint* is written along with the column definition ; and a *table-constraint* is written after all the column definitions are over in CREATE TABLE command.

Check constraint can consist of :

- A list of constant expressions specified using IN

  For example,

  ```
  descp char (20) CHECK
  (descp IN ('NUT', 'BOLT', 'SCREW', 'WRENCH', 'NAIL') )
  ```

- Range of constant expressions specified using BETWEEN. The upper and lower boundary values are included in the range.

  For example,

  ```
  price decimal CHECK
  (price BETWEEN 253.00 and 770.00)
  ```

- A pattern specified using LIKE

  For example,

  ```
  ordate char (10) NOT NULL CHECK (ordate LIKE '- -/- -/- - - -')
  ```

- Multiple conditions using OR, AND etc. *For example,*

  ```
  CHECK  (   (discount = 0.15 AND city = 'HISSAR') OR
             (discount = 0.13 AND city = 'JAIPUR') OR
             (discount = 0.17 AND city = 'MOHALI')   )
  ```

1. What is the difference between column constraints and table constraints ? Name some database integrity constraints.
2. How do the following constraints work ?
   (*i*) Unique (*ii*) Primary Key
   (*iii*) Default (*iv*) Check

Note : For the following question consider tables **EMPLOYEE**, **EMP**, **DEPT**, **PROJECT** and **SALGRADE** that have been mentioned in Type B questions 5-7.

3. To create a table DEPT030 to hold the employee numbers, names, jobs and salaries of employed in department with DeptNo = 30.

Applying Table Constraints

When a constraint is to be applied on a group of columns of the table, it is called ***table constraint***. The table constraints appear in the end of table definition. For instance, if you want combination of *icode* and *descp* of table *items* to be unique, you may write it as follows :

These two are table constraints

```
CREATE TABLE items
(   icode       char (5)        NOT NULL,
    descp       char (20)       NOT NULL,
    ROL         integer,
    QOH         integer,
    CHECK       (ROL<QOH),
    UNIQUE      (icode, descp)
) ;
```

The above statement ensures that the combination of *icode* and *desc* in each row must be unique.

Similarly, if you want to define primary key that contains more than one column, you can use PRIMARY KEY table constraint. For instance, if you want to declare a primary key for the table *members* as the combination of columns *firstname* and *lastname*, it can be done as follows :

```
CREATE TABLE members
( firstname     char (15)   NOT NULL,
  lastname      char (15)   NOT NULL,
  city          char (20),
  PRIMARY KEY (firstname, lastname) ) ;      --[4] the table constraint
```

12.5.2B The SELECT Command

The SELECT command of *SQL* lets you make queries on the database. A query is a command that is given to produce certain specified information from the database table(s). There are various ways and combinations, a SELECT statement can be used into. It can be used to retrieve a subset of rows or columns from one or more tables.

In its simplest form, SELECT statement is used as

```
SELECT  <column name> [, <column name>, ... ]
FROM <table name> ;
```

For example, if you want to view only the information of two columns *empno* and *empname* of table *emp*, you may write your query as

```
SELECT Empno, Empname
FROM emp ;
```

If the EMP table is as shown below :

Table 12.3 *Example table namely EMP*

EmpNo	EmpName	Job	Mgr	Hiredate	Sal	Comm	DeptNo
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		NULL
7369	SMITH	CLERK	7902	17-DEC-80	800		NULL
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		NULL

4. A comment can be given using double minus sign *i.e.*, – – .

Then the result of above query will be as shown below :

The data from all the rows appear in the result.

Empno	*EmpName*
7839	KING
7698	BLAKE
7782	CLARK
:	:
:	:

If you want to see the information of columns *empno*, *job*, and *sal* from the table *employee*, you will write

```
SELECT empno, job, sal
FROM emp ;
```

If you want to see the entire table *i.e.*, every column of a table, you need not give a complete list of columns. The asterisk (*) can be substituted for a complete list of columns as follows :

```
SELECT * FROM emp ;
```

This will display all the rows present in the *emp* table.

Reordering Columns in Query Results

While giving a querying, the result can be obtained in any order. For example, if you give

```
SELECT job, empno, sal, FROM emp ;
```

the result will be having *job* as first column, *empno* as second column, and *sal* as third column. You can write the column names in any order and the output will be having information in exactly the same order.

NOTE

The order of selection determines the order of display.

Eliminating Redundant Data (with keyword DISTINCT)

By default, data is selected from all the rows of the table, even if the data appearing in the result gets duplicated. The DISTINCT keyword eliminates duplicate rows from the results of a SELECT statement. For example, if the *Suppliers* table stores the names and cities of the *suppliers* and we want to see the cities where the Suppliers belong to. The table may consist of more than one supplier belonging to the same city but the result of the query should not contain duplicated city names. To do so, we shall write

```
SELECT DISTINCT city
FROM Suppliers ;
```

In the output, there would be no duplicate rows. Whenever DISTINCT is used, only one NULL value is returned in the results, no matter how many NULL values are encountered.

If we consider *Suppliers* table of chapter 11, shown in Fig. 11.3, then the above query will produce the following output :

See the duplicate entry Delhi has not reappeared. This is the property of DISTINCT.

City
Delhi
Mumbai
Jaipur
Banglore

DISTINCT, in effect, applies to the entire output row, not a specific field. The DISTINCT keyword can be specified only once in a given SELECT clause. If the clause selects multiple fields, DISTINCT eliminates rows where all of the selected fields are identical. Rows in which some values are the same and some different will be retained.

Selecting From All the Rows – ALL keyword

If in place of DISTINCT, you give ALL then the result retains the duplicate output rows. It is just the same as when you specify neither DISTINCT nor ALL ; ALL is essentially a clarifier rather than a functional argument. Thus if you give

```
SELECT ALL city FROM suppliers ;
```

it will give values of *city* column from every row of the table without considering the duplicate entries.

Considering the same table *Suppliers*, now the output will be :

City
Delhi
Mumbai
Delhi
Banglore
Jaipur

Selecting Specific Rows – WHERE clause

In real life, tables can contain unlimited rows. There is no need to view all the rows when only certain rows are needed. *SQL* enables you to define criteria to determine which rows are selected for output. The WHERE clause in SELECT statement specifies the criteria for selection of rows to be returned. The SELECT statement with WHERE clause takes the following general form :

```
SELECT <column name>[, <column name>, ... ]
FROM <table name>
WHERE <condition> ;
```

when a WHERE clause is present, the database program goes through the entire table one row at a time and examines each row to determine if the given condition is true. If it is true for a row, that row is displayed in the output. For example, to display the *empname* and *sal* for employees having their salary more than 2900, the command would be

```
SELECT empname, sal
FROM emp
WHERE sal > 2900 ;
```

*This clause will extract only those rows where **sal** is having value more than 2900*

The above query will produce the following output :

Empname	Sal
KING	5000
JONES	2975
FORD	3000
SCOTT	3000

Only the records having sal > 2900 have appeared in the output.

Relational Operators

To compare two values, a relational operator is used. The result of the comparison is true or false. The *SQL* recognizes following relational operators :

=, >, <, >=, <=, <> (not equal to)

In CHARACTER data type comparisons, < means earlier is the alphabet and > means later in the alphabet. For example $e < f$ and $g > f$. Apostrophes are necessary around all CHAR, DATE and TIME data. For example, to list all the members not from 'DELHI'

```
SELECT * FROM Suppliers
  WHERE city <>'DELHI' ;
```

Logical Operators

The logical operators OR, AND and NOT are used to connect search conditions in the WHERE clause. For example,

1. To list the employees' details having grades 'E2' or 'E3' from table *employee* (not the *EMP* table) :

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (grade = 'E2' OR grade = 'E3') ;
```

2. To list all the employees' details having grades as 'E4' but with gross < 9000

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (grade = 'E4' AND gross < 9000) ;
```

3. To list all the employees' details whose grades are other than 'G1'

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (NOT grade = 'G1') ;
```

when all the logical operators are used together, the order of precedence is NOT, AND, and OR. However, parentheses can be used to override the default precedence.

Condition Based on a Range

The BETWEEN operator defines a range of values that the column values must fall in to make the condition true. The range includes both lower value and the upper value. For example, to list the items whose *QOH* falls between 30 to 50 (both inclusive), the command would be :

```
SELECT icode, descp, QOH
FROM items
WHERE QOH BETWEEN 30 AND 50 ;
```

If the *Items* Table is as shown below :

Table 12.4 *Example table Items*

Icode	*Descp*	*Price*	*QOH*	*ROL*	*ROQ*
I01	Milk	15.00	20	10	20
I02	Cake	5.00	60	20	50
I03	Bread	9.00	40	10	40
I04	Biscuit	10.00	50	40	60
I05	Namkeen	15.00	100	50	70
I06	Cream Roll	7.00	10	20	30

Then the above query will produce the following output :

Only the items having QOH between 30 to 50 have been listed.

Icode	Descp	QOH
I03	Bread	40
I04	Biscuit	50

The operator NOT BETWEEN is reverse of BETWEEN operator, that is, the rows not satisfying the BETWEEN condition are retrieved.

For example,

```
SELECT icode, descp
FROM items
WHERE ROL NOT BETWEEN 100 AND 1000 ;
```

The above query will list the items whose *ROL* is below 100 or above 1000.

Condition Based on a List

To specify a list of values, IN operator is used. The IN operator selects values that match any value in a given list of values.

For example, to display a list of members from 'DELHI', 'MUMBAI', 'CHENNAI' or 'BANGALORE' cities, you may give

```
SELECT * FROM members
WHERE city IN ('DELHI', 'MUMBAI', 'CHENNAI', 'BANGALORE') ;
```

The NOT IN operator finds rows that do not match in the list. So if you write

```
SELECT * FROM members
WHERE city NOT IN ('DELHI', 'MUMBAI', 'CHENNAI') ;
```

It will list members not from the cities mentioned in the list.

Condition Based on Pattern Matches

SQL also includes a string-matching operator, LIKE, for comparisons on character strings using patterns. Patterns are described using two special wildcard characters :

- percent (%). The % character matches any substring.
- underscore (_). The _ character matches any one character.

Patterns are case-sensitive, that is, upper-case characters do not match lower-case characters, or vice-versa.

To illustrate pattern matching, consider the following examples :

- "San%'' matches any string beginning with "San''
- "%idge%'' matches any string containing "idge'' as a substring, for example, "Ridge'', "Bridges'', "Cartridge'', "Ridgeway'' etc.
- "- - - -'' matches any string of exactly 4 characters.
- "- - - -%'' matches any string of at least 4 characters.

(Notice that all patterns are enclosed in double quotes.)

The LIKE keyword is used to select rows containing columns that match a wildcard pattern.

Examples :

1. To list members which are in areas with pin codes starting with 13, the command is :

```
SELECT firstname, lastname, city
FROM members
WHERE pin LIKE "13%" ;
```

LIKE comparison operator for matching patterns

2. To list employees who have four letter first names ending with "D", the command would be :

```
SELECT empno, empname
FROM emp
WHERE empname LIKE "_ _ _D" ;
```

Consider the Table 12.3, the above query will produce the following output :

EmpCode	*EmpName*
7521	WARD
7902	FORD

3. To list members which are not in areas with pin codes starting with 13, the command is :

```
SELECT firstname, lastname, city
FROM members
WHERE  pin NOT LIKE "13%" ;
```

The keyword NOT LIKE is used to select rows that do not match the specified pattern of characters.

In order for patterns to include the special pattern characters (that is, %, _), *SQL* allows the specific of an escape character. The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated as a normal character. We define the escape character for a LIKE comparison using the ESCAPE keyword. To illustrate, consider the following patterns which use a backslash (\) as the escape character.

- LIKE "wx\%yz%" ESCAPE "\" matches all strings beginning with "wx%yz".
- LIKE "wx\\yz%" ESCAPE "\" matches all string beginning with "wx\yz".

The ESCAPE clause can define any character as an escape character. The above two examples use backslash ('\') as the escape character.

Searching for NULL

The NULL value in a column can be searched for in a table using IS NULL in the WHERE clause. (Relational operators like =, <> etc. can't be used with NULL).

For example, to list details of all employees whose departments contain NULL (*i.e.*, novalue), you use the command :

```
SELECT empno, empname, job
FROM emp
WHERE DeptNo IS NULL ;
```

The IS comparison operator for NULLs

Considering the same EMP table, the above query will result in following output :

Only the employees having NULL in their depts have appeard

EmpNo	EmpName	Job
7902	FORD	ANALYST
7369	SMITH	CLERK
7934	MILLER	CLERK

NOTE

Non-NULL values in a table can be listed using IS NOT NULL.

Sorting Results – ORDER BY clause

Whenever a SELECT query is executed, the resulting rows emerge in a predecided order. You can sort the results or a query in a specific order using ORDER BY clause. The ORDER BY clause allows sorting of query results by one or more columns. The sorting can be done either in *ascending* or *descending* order, the default order is ascending. The data in the table is not sorted ; only the results that appear on the screen are sorted. The ORDER BY clause is used as :

```
SELECT <column name> [, <column name>, ... ]
FROM <table name>
[WHERE <predicate> ]
[ORDER BY <column name> ] ;
```

For example, to display the list of employees in the alphabetical order of their names, you use the command :

```
SELECT * FROM employee
ORDER BY ename ;
```

This clause will arrange the output in alphabetical order of ename value

To display the list of employees having salary more than 2500 in the alphabetical order of their names, you may give the command :

```
SELECT empno, empname, job FROM emp
WHERE sal > 2500
ORDER BY ename ;
```

Considering the same EMP table, the above query would produce the following output :

The output appears in the order of Empname.

Empno	EmpName	Job
7698	BLAKE	MANAGER
7902	FORD	ANALYST
7566	JONES	MANAGER
7839	KING	PRESIDENT
7788	SCOTT	ANALYST

To display the list of employees in the descending order of employee code, you use the command :

```
SELECT * FROM employee
ORDER BY ecode DESC ;
```

To specify the sort order, we may specify DESC for descending order or ASC for ascending order. Furthermore, ordering can be performed on multiple attributes. Suppose that we wish to list the entire *employee* relation in descending order of *grade*. If several

employees have the same grade, we order them in ascending order by their names. We express this in *SQL* as follows :

```
SELECT * FROM employee
ORDER BY grade DESC, ename ASC ;
```

Ordering records on multiple fields

See, the multiple fields are separated by commas. In order to fulfill an ORDER BY request, *SQL* must perform a sort. Since sorting a large number of tuples may be costly, it is desirable to sort only when necessary.

How to perform simple calculations ?

Often a simple calculation needs to be done, for example, 4 * 3. The only SQL verb to cause an output to be written to monitor is SELECT. However, a SELECT must have a table name in its FROM clause, otherwise the SELECT fails. You can use **Dual** table for this purpose. For instance, when a calculation is to be performed such as 3 * 4 or 8 * 3 etc., there really is no table being referenced, only numeric literals are being used.

To facilitate such calculations via a SELECT, Oracle provides a dummy table called ***Dual***. **Dual** table is a small worktable, which has just one row and one column. It can be used for obtaining calculation results and also system-date.

The following query :

```
SELECT 4 * 3 FROM dual ;
```

will produce the result as :

```
4 * 3
-----
   12
```

The current date can be obtained from the table ***Dual*** in the required format, using pseudo-column ***sys_date***, as shown below :

```
SELECT sysdate FROM dual ;
```

The output produced by above query will show the current date :

```
SYSDATE
---------
06-Dec-11
```

Aggregate Functions

The summary values are calculated from the data in a particular column using *SQL*'s aggregate functions. Aggregate functions can be applied to all rows in a table or to a subset of the table specified by a WHERE clause.

1. Compare DISTINCT and ALL keywords when used with SELECT command.
2. What is wrong with the following statement ? Write the corrected form of this query :

```
SELECT * FROM employee
WHERE grade = NULL ;
```

Note : For the following questions consider tables **EMPLOYEE**, **EMP**, **DEPT**, **PROJECT** and **SALGRADE** that have been mentioned in Type B questions 5-7.

3. Display names all employee whose names include either of the substring "TH" or "LL".
4. Display data for all CLERKS who earn between 1000 and 2000.
5. Display data for all employees sorted by their department, seniority and salary.
6. Write a SQL statement to list EmpNo, EmpName, DeptNo, for all the employees. This information is should be sorted on EmpName.
7. To find all those employees whose job does not start with 'M'.
8. To display all employees who were hired during 1995.
9. To display DeptNo, Job, EmpName in reverse order of Salary from the EMP table.
10. List EmpName, Job, Sal for all the employees who have a manager.

These functions are called *aggregate functions* because they operate on aggregates of tuples. The results of an aggregate function is a single value.

SQL includes following functions :	
avg	to compute average value
min	to find minimum value
max	to find maximum value
sum	to find total value
stddev	to find the standard deviation
count	to count non-null values in a column
count()	to count total number of rows in a table.
variance	to compute the variance of values in column

Examples :

1. To calculate the total gross for employees of grade 'E2', the command is :

```
SELECT sum(gross)
FROM employee
WHERE grade = 'E2' ;
```

2. To display the average gross of employees with grades 'E1' or 'E2', the command used is :

```
SELECT avg(gross)
FROM employee
WHERE (grade = 'E1' OR grade = 'E2' ) ;
```

3. To count the number of employees in *employee* table, the *SQL* command is :

```
SELECT count(*)
FROM employee ;
```

4. To count the number of cities, the different members belong to, you use the following command :

```
SELECT count(DISTINCT city)
FROM members ;
```

Here the DISTINCT keyword ensures that multiple entries of the same *city* are ignored. The * is the only argument that includes NULLs when it is used only with COUNT, functions other than COUNT disregard NULLs in any case.

If you want to count the entries including repeats, the keyword ALL is used. The following command will COUNT the numbers of non NULL *city* fields in the *members* table :

```
SELECT count (ALL city)
FROM members ;
```

In general, GROUP functions

- Return a single value for a set of rows
- Can be applied to any numeric values, and some CHAR and DATE values.

Grouping Result – GROUP BY

The GROUP BY clause is used in SELECT statements to divide the table into groups. Grouping can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group. For example, to calculate the number of employees in each grade and average gross for each grade of employees, you use the command

```
SELECT job, count(*), sum(comm)
FROM emp
GROUP BY job ;
```

Grouping of records job wise.

GROUP BY applies the aggregate functions independently to a series of groups that are defined by having a field value in common. The output of this query may look like :

Job	Count(*)	Sum(comm)
PRESIDENT	1	0
MANAGER	3	0
SALESMAN	4	2200
CLERK	4	0
ANALYST	2	0

See each row gives the details of total no. of employee in one job and total gross for each job.

Placing Conditions on Groups – HAVING Clause

The HAVING clause places conditions on groups in contrast to WHERE clause that places conditions on individual rows. While WHERE conditions cannot include aggregate functions, HAVING conditions can do so.

For example, to calculate the average gross and total gross for employees belonging to 'E4' grade, the command would be :

```
SELECT avg(gross), sum(gross)
FROM employee
GROUP BY grade
HAVING grade = 'E4' ;
```

The condition in HAVING clause is applied on groups of records, not on individual records. Only the condition in WHERE clause is applied on individual records

To display the jobs where the number of employee are less than 3, you use the command :

```
SELECT job, count(*)
FROM emp
GROUP BY job
HAVING count(*) < 3 ;
```

This will produce the following output :

Job	Count(*)
PRESIDENT	1
ANALYST	2

Scalar Expressions with Selected Fields

If you want to perform simple numeric computations on the data to put it in a form more appropriate to your needs, *SQL* allows you to place scalar expressions and constants among the selected fields. For example, you might consider it desirable to present your salespeople's commissions as percentage rather than decimal numbers. You may do it as :

```
SELECT salesman_name, comm*100
FROM salesman ;
```

Putting Text in the Query Output

The previous example can be refined by marking the commissions as percentages with the percent sign (%). This enables you to put such items as symbols and comments in the output, as in the following example :

```
SELECT salesman_name, comm*100, '%'
FROM salesman ;
```

A sample output produced by above query is shown below :

Salesman_name	
Ajay	13.00 %
Amit	11.00 %
Shally	07.00 %
Isha	15.00 %

See, the same comment or symbol gets printed with every row of the output, not simply once for the table. You could insert text in your query also, making it more presentable. For example,

```
SELECT salesman_name, 'gets the commission', comm*100, '%'
FROM salesman ;
```

The sample output produced by above query is shown below :

Salesman_name		
Ajay	gets the commission	13.00 %
Amit	gets the commission	11.00 %
Shally	gets the commission	07.00 %
Isha	gets the commission	15.00 %

1. What is the difference between WHERE and HAVING clause ?
2. What is the difference between the working of following functions ?

   count (*), count (<column-name>), count(DISTINCT), count(ALL)

Note : For the following questions consider tables **EMPLOYEE**, **EMP**, **DEPT**, **PROJECT** and **SALGRADE** that have been mentioned in Type B questions 5-7.

3. Write SQL statement for :

   Find all the employees who have no manager.
4. Show the average salary for all departments with more than 3 people for a job.
5. Display only the jobs with maximum salary greater than or equal to 3000.
6. Find out number of employees having "Manager" as Job.
7. Find the average salary and average total remuneration for each job type remember salesman earn commission.

After covering all the clauses, let us now summarise the usage of SELECT.

```
SELECT column list
FROM <table name>
WHERE <predicate>
GROUP BY <column name(s)>
HAVING <search condition>
ORDER BY column-name ;
```

The order of the clauses in the SELECT statement is important.

12.5.2C Creating Table From Existing Table

You can define a table and put data into it without going through the usual data definition process. This can be done by using SELECT statement with CREATE TABLE.

The new table stores the result produced by the SELECT statement. The name of the new table must be unique. Following query illustrates this :

```
CREATE TABLE orditem AS
(    SELECT icode, descp
     FROM items
     WHERE QOH < ROL ) ;
```

This will create a new table called *orditem* that stores two columns : *itemcode* and *description* for the items that have their *QOH* less than *ROL* in the relation ***items***.

The newly created table **orditem** will look as it is shown below :

Table 12.5 *Orditem*

Icode	*Descp*	*Price*	*QOH*	*ROL*	*QOH*
I06	Cream Roll	7.00	10	20	30

The table *orditem* has derived its contents from table **items** (Table 12.4).

If you do not specify the WHERE clause, *icode* & *descp* from all rows of the *items* relation will be copied into *orditem*.

The CREATE TABLE <table> AS SELECT... is useful for creating test tables, new tables as copies of existing tables, and for making smaller tables out of large tables.

NOTE

Desired information may also be extracted from multiple tables. In that case, you need to join the tables. The joins in SQL have been given in ***appendix C.***

In some implementations, a new table, from an existing table can also be created using SELECT INTO as shown below :

```
SELECT icode, descp INTO orditems
FROM Items
WHERE QOH < ROL ;
```

It produces the same result as by previous query.

12.5.2D The INSERT Command

Values are placed in and removed from attributes of a relation with three DML commands : INSERT, DELETE and UPDATE. These are all referred to in *SQL* as *update* commands in a generic sense. In our text, lowercase "update" will indicate these commands generically and the uppercase for the keyword UPDATE.

The rows (tuples) are added to relations using INSERT command of *SQL*. In its simplest form, INSERT takes the following syntax :

```
INSERT INTO <tablename> [ <column list> ]
VALUES ( <value>, <value> ... ) ;
```

For example, to enter a row into *employee* table (defined earlier), you could use the following statement :

```
INSERT INTO employee
VALUES (1001, 'Ravi', 'M', 'E4', 4670.00) ;
```

See the order of values matches the order of columns in the CREATE TABLE command of *employee*. The same can be done with an alternate command as shown below :

```
INSERT INTO employee (ecode, ename, sex, grade, gross)
VALUES (1001, 'Ravi', 'M', 'E4', 4670.00) ;
```

The INSERT statement adds a new row to *employee* giving a value for every column in the row. Note that the data values are in the same order as the column names in the table. Data can be added only to some columns in a row by specifying the columns and their data.

For instance, if you want to insert only *ecode*, *ename* and *sex* columns, you use the command :

```
INSERT INTO employee (ecode, ename, sex)
VALUES (2014, 'Manju', 'F') ;
```

The columns that are not listed in the INSERT command will have their default value, if it is defined for them, otherwise, NULL value.

NOTE

In an INSERT statement, only those columns can be omitted that have either default value defined or they allow NULL values.

If any other column (that does not have a default and is defined NOT NULL), an error message is generated and the row is not added.

Inserting the Results of a Query

INSERT command can also be used to take or derive values from one table and place them in another by using it with a query. To do this, simply replace the VALUES clause with an appropriate query as shown in the following example :

```
INSERT INTO branch1
SELECT * FROM branch 2
WHERE gross > 7000.00 ;
```

It will extract all those rows from *branch2* that have *gross* more than 7000.00 and insert this produced result into the table *branch1*.

To insert using a query, the following conditions must be true :

(*i*) Both the tables must be already created.

(*ii*) The columns of the tables being inserted into, must match the columns output by the subquery.

12.5.2E The DELETE Command

The DELETE command removes rows from a table. This removes the entire rows, not individual field values, so no field argument is needed or accepted. The DELETE statement takes the following general form :

```
DELETE FROM <tablename>
[WHERE <predicate>] ;
```

NOTE

SELECT, INSERT, DELETE and UPDATE are DML commands.

To remove all the contents of *items* table, you use the command :

```
DELETE FROM items ;
```

The table would now be empty and could be destroyed with a DROP TABLE command (covered in section 12.5.4B).

Even some specific rows from a table can also be deleted. To determine which rows are deleted, you use a condition, just as you do for queries. For instance, to remove the tuples from *employee* that have *gross* less than 2200.00, the following command is used :

```
DELETE FROM employee
WHERE gross < 2200.00 ;
```

12.5.2F The UPDATE Command

Sometimes you need to change some or all of the values in an existing row. This can be done using the UPDATE command of *SQL*. The UPDATE command specifies the rows to be changed using the WHERE clause, and the new data using the SET keyword. The new data can be a specified constant, an expression or data from other tables.

For example, to change the reorder level *ROL* of all items to 250, you would write

```
UPDATE items
SET ROL = 250 ;
```

If you want to change *ROL* to 400 only for those items that have *ROL* as 300, you use the command

```
UPDATE items
SET ROL = 400
WHERE ROL = 300 ;
```

Updating Multiple Columns

To update multiple columns, multiple column assignments can be specified with SET clause, separated by commas. All of the said assignments will still be made to the table, a single row at a time. To update the *ROL* and *QOH* for items having icode less than 'I040', we shall write

```
UPDATE items
SET ROL = 400, QOH = 700
WHERE icode < 'I040' ;
```

Using Expressions in Update

Scalar expressions can also be used in the SET clause of the UPDATE command. Suppose, if you want to increase the gross pay of all the employees by Rs. 900/-, you could use the following expression :

```
UPDATE employee
SET gross = gross + 900 ;
```

To double the gross pay of employees of grade 'E3' and 'E4', you use the command :

```
UPDATE employee
SET gross = gross * 2
WHERE (grade = 'E3' OR grade = 'E4') ;
```

Updating to NULL Values

The NULL values can also be entered just as other values. For example, a new grade is to be introduced and all the employees with grade 'E4' have to be promoted to it. But for the time being this grade is not known, thus NULL values are to be inserted for grades 'E4'. This can be done as follows :

```
UPDATE employees
SET grade = NULL
WHERE grade = 'E4' ;
```

Note : For the following questions consider tables **EMPLOYEE**, **EMP**, **DEPT**, **PROJECT** and **SALGRADE** that have been mentioned in Type B questions 5-7.

1. Insert a record with suitable data in the table EMP, tabing system date as the Hiredate.
2. Update table Empl by setting Dept as Null for all employees with Department no as 400.
3. Delete all the employees that earn less than Rs. 5000.

12.5.2G The CREATE VIEW Command

As you know about views that a view is a virtual table with no data, but can be operated like any other table. It is like a window through which you can view the data of another table, which is called the *base table*.

The *SQL* provides a statement for creating views which is CREATE VIEW command. This command consists of the words CREATE VIEW, the name of the view to be created, the word AS, and then a query, as in the following example :

```
CREATE VIEW taxpayee
AS SELECT *
FROM employee
WHERE gross >8000 ;
```

This view will be having details of those employees that have gross more than Rs. 8000/. Now this view can be used just like any other table. It can be queried, updated (if allowed), inserted into (if allowed), deleted from (if allowed) etc. This view can be queried as follows :

```
SELECT * FROM taxpayee ;
```

If any column in the view is to be given a different name other than the name of the column from which it is derived, it is done by specifying the new column names as shown below :

```
CREATE VIEW taxpayee (empcode, empname, sex, empgrade, empgross)
AS SELECT * FROM employee
WHERE gross >8000 ;
```

NOTE

The SELECT statement used in a view definition cannot include :

- ORDER BY clause
- INTO clause

A view can contain columns having calculated values, but you must specify the name for that column. For example, the following command

```
CREATE VIEW taxpayee (ecode, ename, tax)
AS  SELECT ecode, ename, gross * 0.1
FROM employee
WHERE gross >8000 ;
```

The above created view has an additional column called *tax* which is 10% of the *gross*.

Whether you can use update commands (INSERT, DELETE, and UPDATE) on views, depends upon the fact whether the view is updatable or not. A view is updatable if it has been defined from a single relation and the update query can be mapped on to the base table successfully. For instance, if you create a view as follows :

```
CREATE VIEW sample
AS SELECT ename, gross
FROM employee ;
```

In nutshell, we can say,

(*i*) A view is like a window through which you can view or change information in a table.

(*ii*) A view is a virtual table *i.e.*,

- it looks like a table, but it does not exist as such
- its data are derived from base table(s).
- it only stores its definition ; it does not contain any copy of the data.

(*iii*) A view provides

- *Simplicity* – you can see exactly what you need
- *Security* – it prevents unauthorised users from seeing unrelevant information.

You cannot use the following command

```
INSERT INTO sample
VALUES ('Rohan', 4900.00) ;
```

The above statement cannot be mapped onto the base table *employee*, as the value of the primary key *ecode* is missing and also the *sex* column does not have any default value and it cannot have NULL at the same time. Thus, no such row can be inserted that has primary key value missing. Therefore, the view *sample* is not updatable.

12.5.3 Some Built-In Functions

Most SQL implementations provide several types of built-in functions[5] that return different kinds of information from the database. We are considering here a few of them.

1. **Lower** (***Character-expression***) converts the given character-expression in to lower case.
2. **Upper** (***Character-expression***) converts the given character-expression into upper case.

5. These functions may vary from one implementation to another.

3. **Replicate** (***char-expn, no-of-times***) repeats the given *char-expn* (character expression) the specified *no-of-times*. The *no-of-times* has to be an integer.
4. **Substr** (***expn, startpos, no-of-chars***) returns the given *no-of-chars* (integer) from a character string *expn* starting at the specified *startpos* (integer).
5. **getdate( )** returns the current system date.

See the following examples for better understanding of these functions.

The query	will return
SELECT lower ("HELLOW") FROM Dual ;	hellow
SELECT upper ("friends") FROM Dual ;	FRIENDS
SELECT replicate ("*#", 4) FROM Dual ;	*#*#*#*#
SELECT substr ("Pointer", 3, 2) FROM Dual ;	in

The query

```
SELECT getdate ( ) FROM Dual ;
```

will return the current system date of your computer.

12.5.4 Joins

Table Structures

A. Billed
- Bill_No
- Patient_No
- Item_Code
- Charge

B. Item
- Item_Code
- Description
- Normal_Charge

A **join** is a query that combines rows from two or more tables[6]. In a *join-query*, more than one table are listed in FROM clause. The function of combining data from multiple tables is called *joining*.

SQL can produce data from several related tables by performing either a physical or virtual join of the tables.

The WHERE clause is most often used to perform the JOIN function where two or more tables have common columns. Consider the following example :

```
SELECT patient_no, description, normal_charge, charge
FROM billed, item
WHERE billed.item_code = item.item_code ;
```

tables being joined

join condition

The above query is joining two tables *billed* and *item* (*see their structures on the left*) by equating their *item_codes*. The select_list consists of fields coming from both the tables. When two or more tables have a column with the *same name*, the name can be *qualified* by using the table name combined with the period (.) in referring to the column, *e.g., billed.patient_no*. Note that columns with unique names *do not* have to be qualified by the table name.

12.5.4A Cartesian Product

Cartesian Product

In unrestricted join or *Cartesian product* of two tables, all possible concatenations are formed of all rows of both the tables.

Consider the following query :

```
SELECT *
FROM EMP, DEPT ;
```

This query will give you *the Cartesian product i.e.*, all possible concatenations are formed of all rows of both the tables EMP and DEPT. That is, when no particular rows (using WHERE clause) and columns (through SELECT list) are selected. Such an operation is also known as **Unrestricted Join**. It returns **n1** × **n2** rows where **n1** is number of rows in first table and **n2** is number of rows in second table.

6. A join can combine views also. A view is a stored query that appears like a table and can be used like a table.

12.5.4B Table Aliases

A **Table Alias** is a temporary label given along with table name in FROM clause.

To cut down on the amount of typing required in your queries you can use aliases for table names in the SELECT and WHERE clauses. Consider the following join query. Consider the example 12.1.

Table Structures

C. Treats

- Phy_Id
- Patient_No
- Procedure_No
- Date_Treated
- Treat_Result

D. Physicians

- Phy_Id
- Phy_Phone
- Phy_Name

EXAMPLE 12.1. *List the patients who had either Dr. N. Pandya, Dr. A. Sapra, Dr. Suleman Rashid, Dr. Keith John as physician. [Dr. N. Pandya has physician id as 8883, Dr. Sapra as 8887, Dr. Rashid as 8886 and Dr. John as 8882].* (*see table structures on the left*)

SOLUTION.
```
SELECT DISTINCT patient_no, phy_id
FROM treats
WHERE phy_id IN (8887, 8886, 8883, 8882) ;
```

The above example 12.1 where the query listed the *PATIENT_NO* and *PHY_ID*, but not the PAT_NAME of *PHY_NAME* fields. The following query joins the *PATIENT*, *PHYSICIAN*, and *TREATS* tables to produce the desired information. The query also demonstrates the use of an ALIAS for a table name – here we have three aliases (*PA*, *PH*, and *TR*), one for each table.

EXAMPLE 12.2. *List which patient was treated by which physician. Also list their ids along with their names.* (*see table structures on the left*)

SOLUTION.
```
SELECT DISTINCT PA.patient_no, pat_name, PH.phy_id, phy_name
FROM patient PA, physician PH, treats TR
WHERE  PA.patient_no = TR.patient_no AND
PH.phy_id = TR.phy_id AND
PH.phy_id IN (8887, 8886, 8883, 8882) ;
```

12.5.4C Equi-Join and Natural Join

Equi-Join The Join, in which columns are compared for equality, is called *Equi-Join*.

Natural Join The Join in which only one of the identical columns (coming from joined tables) exists, is called *Natural Join*.

The Join, in which columns are compared for equality, is called **Equi-Join.**

A *non-equi-join* is a query that specifies some relationship other than equality between the columns.

The Join in which only one of the identical columns (coming from joined tables) exists, is called **Natural Join.**

The Equi-Join and Natural Join are equivalent except that duplicate columns are eliminated in the Natural Join that would otherwise appear in the Equi-Join.

EXAMPLE 12.3. *Display patient_no, the date when he/she is discharged and the charge paid by him/her.*

SOLUTION.
```
SELECT billed.patient_no, date_discharged, charge
FROM billed, patient
WHERE patient.patient_no = billed.patient_no ;
```

The above query shows the *PATIENT_NO* and *DATE_DISCHARGED* from the *PATIENT* table and the associated *CHARGE* from the *BILLED* table.

12.5.5 UNION

Multiple queries can be combined into one by forming a union of them. The *SQL* UNION operator allows manipulation of results returned by two or more queries by combining the results of each query into a single result set. Consider the data given below that contains rows from relations *A* and *B*.

Relation A

Rollno.	*Name*	*Age*
101	Sidharth	19
102	Kushagra	18
103	Usman	18

Relation B

Rollno.	*Name*	*Age*
301	Anubha	17
102	Kushagra	18
303	Sba	18

The UNION of these two relations can be created as follows :

```
SELECT * FROM A
UNION
SELECT * FROM B ;
```

The result produced will be as follows :

Rollno.	*Name*	*Age*
101	Sidharth	19
102	Kushagra	18
103	Usman	18
301	Anubha	18
303	Sba	18

See the duplicate row (102 Kushagra 18) has automatically been removed. By default, the UNION operator removes duplicate rows from the result. If the ALL option is used, all rows, including duplicates, are included in the results. The general form of using UNION operator is :

```
SELECT statement
UNION [ALL]
SELECT statement ;
```

Rules of Using UNION Operator

1. Both the SELECT statements must be UNION compatible, that is, the *select-lists* (column-lists) of SELECT statement must have same number of columns having similar data types and order. For instance, the following union query is *invalid* :

   ```
   SELECT ecode, ename FROM branch1
   UNION
   SELECT ename, ecode FROM branch2 ;
   ```

   It is because the order of columns is different. Also, the following union is invalid because the SELECT statements do not contain compatible *select-lists*.

   ```
   SELECT ecode, ename, sex FROM branch1
   UNION
   SELECT ecode, ename, sex, grade FROM branch2 ;
   ```

2. The ORDER BY clause can occur only at the end of the UNION statement. It can't be used within the individual queries that make the UNION statement.
3. The GROUP BY and HAVING clauses are allowed only within individual queries. These clauses cannot be used to affect the final result set.

12.5.6 Some More DDL Commands

AFTER discussing DML commands like SELECT, INSERT, DELETE and UPDATE, let us now move on to discussion of some more DDL commands.

12.5.6A The ALTER TABLE Command

The ALTER TABLE command is not part of the ANSI standard, but it is widely available, and its form is fairly consistent, although its capabilities vary considerably. It is used to change the definitions of existing tables. Usually, it can add columns to a table. Sometimes it can delete columns or change their sizes. Typically, the syntax to add a column to a table is as follows :

```
ALTER TABLE <table name>
ADD <column name> <data type> <size> ;
```

The new column will be added with NULL values for all rows currently in the table.

It is generally possible to add several new columns, separated by commas, in a single command. It may be possible to drop or alter columns through ALTER TABLE command depending upon the SQL being supported by your dbms.

Most often, altering columns will simply be a matter of increasing their size. Because of the non-standard nature of the ALTER TABLE command, you must refer to your system documentation for exact details of ALTER TABLE command.

To modify existing columns of table, ALTER TABLE command can be used according to following syntax :

```
ALTER TABLE <tablename>
MODIFY (columname newdatatype (newsize) ) ;
```

For instance, to add a new column *tel_number* of type *integer* in table *Emp* you may give :

```
ALTER TABLE Emp
ADD (tel_number integer) ;
```

To modify column *Job* of table *Emp* to have new width of 30 characters, you may give :

```
ALTER TABLE Emp
MODIFY (Job char(30) ) ;
```

12.5.6B The DROP TABLE Command

The DROP TABLE command of *SQL* lets you drop a table from the database. The *SQL* requires you to empty a table before you eliminate from the database. But there is a condition for dropping a table ; it must be an empty table.

NOTE

A table with rows in it cannot be dropped.

To remove all the rows from your table, you use DELETE command (as discussed in section 12.5.2E). For instance, if you want to drop *items* table, you first remove all the rows using the command :

```
DELETE FROM items ;
```

Then you can drop the empty table *items* as follows :

```
DROP TABLE items ;
```

Once this command is given, the table name is no longer recognized and no more commands can be given on that object.

When a table is dropped, any request to access its dependent view results into an error because by default dependent entities such as foreign keys and views etc. are not

1. What is the difference between SELECT INTO and CREATE VIEW commands ?
2. What are views ? When can a view be updated ?
3. What is the condition of dropping a table ?

Note : For the following questions consider tables **EMPLOYEE**, **EMP**, **DEPT**, **PROJECT** and **SALGRADE** that have been mentioned in Type B questions 5-7.

4. What happens if you try to drop a table on which a view exists.
5. Create a view with one of the columns Salary * 12. Try updating columns of this view.
6. Can you create view of a view ?
7. Compare Join and Cartesian product.
8. What is the difference between a Cartesian product and a Union ?

dropped with table. For example, if the table *employee* is dropped and then a request is made to select data from view *taxpayee* which was dependent upon *employee* table, it will result in an error.

If the dropped table is a base table for a view, then the DBMS like Oracle invalidates these dependent views but does not drop them. You cannot use these views unless you re-create the table or drop and re-create the objects so that they no longer depend on the table.

12.5.6C The DROP VIEW Command

To delete a view from the database the DROP VIEW command is used. For example,

```
DROP VIEW taxpayee
```

drops the view *taxpayee* from the database. When a view is dropped, it does not cause any change in its base table. After the removal of view *taxpayee*, its base table *employee* remains intact.

Let Us Revise

- *SQL is a language that enables you to create and operate on relational databases.*
- *The various processing capabilities of SQL are : data definition language (DDL), interactive and embedded data manipulation language (DML), view definition, authorization, integrity and transaction control.*
- *The DDL provides statements for the creation and deletion of tables and indexes.*
- *The DML provides statements to enter, update, delete data and perform complex queries on these tables.*
- *The ANSI standard supports these data types : CHAR, DECIMAL (DEC), NUMERIC, INT(INTEGER), SMALLINT FLOAT, REAL, DOUBLE PRECISION (DOUBLE).*
- *Most implementations also support DATE and TIME types.*
- *The CREATE TABLE command creates a new table.*
- *The SELECT command lets you make queries on the database. Rows returned are restricted using WHERE clause.*
- *Results are sorted using ORDER BY clause and the GROUP BY clause divides the result obtained into groups and the HAVING clause sets condition for the GROUP BY clause.*
- *SQL aggregate functions (avg, min, max, sum, count) calculate the summary values from the data in a particular column.*
- *SELECT INTO creates a new table by extracting data from another table.*
- *The rows are added to relations using INSERT command.*
- *The rows are removed from a relation using DELETE command.*
- *The UPDATE command lets you change some or all of the values in an existing row.*
- *The CREATE VIEW creates view from a table.*
- *In unrestricted join or Cartesian product of two tables, all possible concate- nations are formed of all rows of both the tables.*
- *The Join, in which columns are compared for equality, is called Equi-Join.*
- *The ALTER TABLE changes the definition of an existing table.*
- *The DROP TABLE drops a table from the database.*
- *The DROP VIEW drops a view from the database.*

SOLVED PROBLEMS

1 ■ *Why can you not ask to see the first seven rows of a table ?*

SOLUTION. Because the rows are, by definition, in no particular order.

2 ■ *Which subdivision of SQL is used to put values in tables and which one to create tables ?*

SOLUTION. Data Manipulation Language (DML) is used to put values in tables and Data Definition Language (DDL) is used to create tables.

3 ■ *What are DDL and DML ?* (Outside Delhi 2006)

SOLUTION. The DDL provides statements for the creation and deletion of tables, indexes, views etc.

The DML provides statements to enter, update, delete data and perform complex queries on these tables.

4 ■ *Write a query that produces the salesman table with the columns in the following order : city, salesman_name, salesman_code, commission.*

SOLUTION.

```
SELECT city, salesman_name, salesman_code, commission
FROM salesman ;
```

5 ■ *Write a query that will produce the salesman-code values of all sales people with orders currently in the Orders table without any rates.*

SOLUTION.

```
SELECT DISTINCT salesman_code
FROM Orders ;
```

6 ■ *Write a query on the customers table whose output will exclude all customers with a rating <=100, unless they are located in Shimla.*

SOLUTION.

```
SELECT *
FROM customers
WHERE rating > 100
      OR city = 'Shimla' ;
SELECT *
FROM customers
WHERE NOT rating <=  100
      OR city = 'Shimla' ;
SELECT *
FROM customers
WHERE NOT (rating <= 100
            AND city < >'Shimla') ;
```

There may be other solutions as well.

7 ■ *Write a query that selects all orders except those with zeros or NULLs in the amt field.*

SOLUTION.

```
SELECT * FROM Orders
WHERE amt < >0 AND (amt IS NOT NULL) ;
```

8 ■ *Write a query that counts the number of salespeople registering orders for each day. (If a salesperson has more than one order on a given day, he or she should be counted only once.).*

SOLUTION.

```
SELECT ord_date, count (DISTINCT salesman-code)
FROM orders
GROUP BY ord_date ;
```

9 ■ *Write a query on the customers table that will find the highest rating in each city. Put the output in this form :*

For the city (city), the highest rating is : (rating).

SOLUTION.
```
SELECT 'For the city', city,', the highest rating is', MAX (rating)
FROM customers
GROUP BY city ;
```

10 ■ *Write a query that lists customers in descending order of rating. Output the rating field first, followed by the customer's name and number.*

SOLUTION.
```
SELECT rating, cust_name, cust_num
FROM customers
ORDER BY rating DESC ;
```

11 ■ *Write a query that tables the orders for each day and places the results in descending order of the date.*

SOLUTION.
```
SELECT ord_date, sum (amt)
FROM orders
GROUP BY ord_date
ORDER BY ord_date DESC ;
```

12 ■ *Write a command that puts the following values, in their given order, into the salesman table : city-Manali, cust_name-Manisha, comm-NULL, cust-num-1901.*

SOLUTION.
```
INSERT INTO salesman(city, cust_name, comm, cust_num)
VALUES ('Manali', 'Manisha', NULL, 1901) ;
```

13 ■ *Write a command that removes all orders from customer Sohan from Orders table.*

SOLUTION.
```
DELETE FROM Orders
WHERE cust_name = 'Sohan' ;
```

14 ■ *Write a command that increases the ratings of all customers in Shimla by 100.*

SOLUTION.
```
UPDATE Customers
SET rating = rating + 100
WHERE city = 'Shimla' ;
```

15 ■ Observe the following PARTICIPANTS and EVENTS tables carefully and write the name of the RDBMS operation which will be used to produce the output as shown in RESULT. Also, find the Degree and Cardinality of the RESULT. (Outside Delhi 2016)

Table : PARTICIPANTS

PNO	NAME
1	Aruanabha Tariban
2	John Fedricks
3	Kanti Desai

Table : EVENTS

EVENTCODE	EVENTNAME
1001	IT Quiz
1002	Group Debate

Table : RESULT

PNO	NAME	EVENTCODE	EVENTNAME
1	Aruanabha Tariban	1001	IT Quiz
1	Aruanabha Tariban	1002	Group Debate
2	John Fedricks	1001	IT Quiz
2	John Fedricks	1002	Group Debate
3	Kanti Desai	1001	IT Quiz
3	Kanti Desai	1002	Group Debate

SOLUTION. **RDBMS Operation :** Cartesian Product
Degree : 4
Cardinality : 6

16 ■ *Consider the following tables STORE and ITEM and answer (a) and (b) parts of this question.*
(Delhi 2014)

Table : STORE

SNo	SName	Area
S01	ABC Conputronics	GK II
S02	All Infotech Media	CP
S03	Tech Shoppe	Nehru Place
S04	Geeks Tecno Soft	Nehru Place
S05	Hitech Tech Store	CP

Table : ITEM

INo	Iname	Price	SNo
T01	Mother Board	12000	S01
T02	Hard Disk	5000	S01
T03	Keyboard	500	S02
T04	Mouse	300	S01
T05	Mother Board	13000	S02
T06	Key Board	400	S03
T07	LCD	6000	S04
T08	LCD	5500	S05
T09	Mouse	350	S05
T10	Hard Disk	4500	S03

(*a*) Write the SQL queries ((*i*) to (*iv*))

(*i*) To display IName and Price of all the Items in ascending order of their price.

(*ii*) To display SNo and SName of all Stores located in CP.

(*iii*) To display minimum and maximum price of each Iname from the table Item.

(*iv*) To display IName, Price of all items and their respective SName where they are available.

(*b*) Write the output of the following SQL commands ((*i*) to (*iv*)) :

(*i*) SELECT DISTINCT INAME FROM ITEM WHERE PRICE >= 5000 ;

(*ii*) SELECT AREA, COUNT (*) FROM STORE GROUP BY AREA ;

(*iii*) SELECT COUNT(DISTINCT AREA) FROM STORE ;

(*iv*) SELECT INAME, PRICE * 0.05 DISCOUNT FROM ITEM WHERE SNO IN ('S02', 'S03') ;

SOLUTION. (*a*)

(*i*)
```
SELECT IName, Price
FROM ITEM
ORDER BY Price ;
```

(*ii*)
```
SELECT SNo, SName
FROM STORE
WHERE Area = "CP";
```

(*iii*)
```
SELECT IName, Min(Price), Max(Price)
FROM ITEM
GROUP BY IName ;
```

(*iv*)
```
SELECT IName, Price, SName
FROM ITEM, STORE
WHERE ITEM.SNO = STORE.SNO ;
```

(*b*) (*i*)
```
Mother Board
Hard Disk
LCD
```

(*ii*)
```
GK II          1
CP             2
Nehru Place    2
```

(*iii*) 3

(*iv*)

INAME	DISCOUNT
Keyboard	25
Mother Board	650
Keyboard	20
Hard Disk	225

17 ■ *Write SQL queries for (i) to (iv) and find outputs for SQL queries (v) to (viii), which are based on the tables.* (CBSE D 2016)

Table : VEHICLE

CODE	VTYPE	PERKM
101	VOLVO BUS	160
102	AC DELUXE BUS	150
103	ORDINARY BUS	90
105	SUV	40
104	CAR	20

Note.

- PERKM is Freight Charges per kilometre
- VTYPE is Vehicle Type

Table : TRAVEL

NO	NAME	TDATE	KM	CODE	NOP
101	Janish Kin	2015-11-13	200	101	32
103	Vedika Sahai	2016-04-21	100	103	45
105	Tarun Ram	2016-03-23	350	102	42
102	John Fen	2016-02-13	90	102	40
107	Ahmed Khan	2015-01-10	75	104	2
104	Raveena	2016-05-28	80	105	4
106	Kripal Anaya	2016-02-06	200	101	25

Note.

- NO is Traveller Number
- KM is Kilometer travelled
- NOP is number of travellers travelled in vehicle
- TDATE is Travel Date

(*i*) *To display NO, NAME TDATE from the table TRAVEL in descending order of NO.*

(*ii*) *To display the NAME of all the travellers from the table TRAVEL who are travelling by vehicle with code 101 or 102.*

(*iii*) *To display the NO and NAME of those travellers from the table TRAVEL who travelled between '2015-12-31' and '2015-04-01'.*

(*iv*) *To display all the details from table TRAVEL for the travellers, who have travelled distance more than 100 KM in ascending order of NOP.*

(*v*)
```
SELECT COUNT(*), CODE FROM TRAVEL
GROUP BY CODE HAVING COUNT(*)>1 ;
```

(*vi*)
```
SELECT DISTINCT CODE FROM TRAVEL ;
```

(*vii*)
```
SELECT A.CODE, NAME, VTYPE
FROM TRAVEL A, VEHICLE B
WHERE A.CODE = B.CODE AND KM < 90 ;
```

(*viii*)
```
SELECT NAME, KM * PERKM
FROM TRAVEL A, VEHICLE B
WHERE A.CODE = B.CODE AND A.CODE = '105' ;
```

SOLUTION.

(*i*)
```
SELECT NO, NAME, TDATE
FROM TRAVEL
ORDER BY NO DESC ;
```

(*ii*)
```
SELECT NAME
FROM TRAVEL
WHERE CODE IN(101, 102) ;
```

(*iii*)
```
SELECT NO, NAME
FROM TRAVEL
WHERE TDATE >= '2015-04-01' AND TDATE <= '2015-12-31' ;
```

(*iv*)
```
SELECT *
FROM TRAVEL
WHERE KM > 100
ORDER BY NOP ;
```

(*v*)
```
2    101
2    102
```

(*vi*)
```
101
103
102
104
105
```

(*vii*)
```
104    Ahmed Khan CAR
105    Raveena SUV
```

(*viii*)
```
Raveena    3200
```

18 ■ *Consider the following tables Stock and Dealers and answer (a1) and (a2) parts of this question :*

(Outside Delhi 2010)

Table : Stock

ItemNo	*Item*	*Dcode*	*Qty*	*UnitPrice*	*StockDate*
5005	Ball Pen 0.5	102	100	16	31-Mar-10
5003	Ball Pen 0.25	102	150	20	01-Jan-10
5002	Gel Pen Premium	101	125	14	14-Feb-10
5006	Gel Pen Classic	101	200	22	01-Jan-09
5001	Eraser Small	102	210	5	19-Mar-09
5004	Eraser Big	102	60	10	12-Dec-09
5009	Sharpener Classic	103	160	8	23-Jan-09

Table : Dealers

Dcode	Dname
101	Reliable Stationers
103	Classic Plastics
102	Clear Deals

(a1) *Write SQL commands for the following statements :*

(*i*) To display details of all Items in the Stock table in ascending order of StockDate.

(*ii*) To display ItemNo and Item name of those items from Stock table whose UnitPrice is more than Rupees 10.

(*iii*) To display the details of those items whose dealer code (Dcode) is 102 or Quantity in Stock (Qty) is more than 100 from the table Stock..

(*iv*) To display Maximum UnitPrice of items for each dealer individually as per Dcode from the table Stock.

(a2) *Give the output of the following SQL queries :*

(*i*) SELECT COUNT(DISTINCT Dcode) FROM Stock ;

(*ii*) SELECT Qty*UnitPrice FROM Stock
WHERE ItemNo = 5006 ;

(*iii*) SELECT Item, Dname FROM Stock S, Dealers D
WHERE S.Dcode = D.Dcode AND ItemNo = 5004 ;

(*iv*) SELECT MIN(StockDate) FROM Stock ;

SOLUTION. (a1)

(*i*)
```
SELECT *
FROM Stock
ORDER BY StockDate ;
```

(*ii*)
```
SELECT ItemNo, Item
FROM Stock
WHERE UnitPrice > 10 ;
```

(*iii*)
```
SELECT *
FROM Stock
WHERE Dcode = 102 OR Qty > 100 ;
```

(*iv*)
```
SELECT Dcode, MAX (UnitPrice)
FROM Stock
GROUP BY Dcode ;
```

(a2) (*i*) 3 (*ii*) 4400 (*iii*) Eraser Big Clear Deals (*iv*) 01-Jan-09

19 ■ *Consider the following tables EMPLOYEE and SALGRADE and answer (A1) and (A2) parts of this question :* (Outside Delhi 2011)

Table : EMPLOYEE

ECODE	NAME	DESIG	SGRADE	DOJ	DOB
101	Abdul Ahmad	EXECUTIVE	S03	23-Mar-2003	13-Jan-1980
102	Ravi Chander	HEAD-IT	S02	12-Feb-2010	22-Jul-1987
103	John Ken	RECEPTIONIST	S03	24-Jun-2009	24-Feb-1983
105	Nazar Ameen	GM	S02	11-Aug-2006	03-Mar-1984
108	Priyam Sen	CEO	S01	29-Dec-2004	19-Jan-1982

Table : SALGRADE

SGRADE	SALARY	HRA
S01	56000	18000
S02	32000	12000
S03	24000	8000

(A1) *Write SQL commands for the following statements :*

(*i*) To display the details of all EMPLOYEEs in descending order of DOJ.

(*ii*) To display NAME and DESIG of those EMPLOYEEs, whose SALGRADE is either S02 or S03.

(*iii*) To display the content of all the EMPLOYEEs table, whose DOJ is in between '09-Feb-2006' and '08-Aug-2009'.

(*iv*) To add a new row with the following : 19, 'Harish Roy', 'HEAD-IT', 'S02', '09-Sep-2007', '21-Apr-1983'

(A2) *Give the output of the following SQL queries :*

(*i*) SELECT COUNT (SGRADE), SGRADE FROM EMPLOYEE GROUP BY SGRADE;

(*ii*) SELECT MIN(DOB), MAX(DOJ) FROM EMPLOYEE;

(*iii*) SELECT NAME, SALARY FROM EMPLOYEE E,
SALGRADE S WHERE E.SGRADE = S.SGRADE AND E.ECODE<103;

(*iv*) SELECT SGRADE, SALARY + HRA FROM SALGRADE WHERE SGRADE = 'S02';

SOLUTION. **(A1)**

(*i*)
```
SELECT * FROM EMPLOYEE ORDER BY DOJ DESC ;
```

(*ii*)
```
SELECT NAME, DESIG FROM EMPLOYEE WHERE SALGRADE IN('S02', S03') ;
```

(*iii*)
```
SELECT * FROM EMPLOYEE WHERE DOJ
BETWEEN '09-Feb-2006' AND '08-Aug-2009' ;
```

(*iv*)
```
INSERT INTO EMPLOYEE
VALUES(19,'Harish Roy','HEAD-IT','S02','09-Sep-2007','21-Apr-1983') ;
```

(A2)

(*i*)
```
COUNT    SGRADE
2        S03
2        S02
1        S01
```

(*ii*)
```
13-Jan-1980    12-Feb-2010
```

(*iii*)
```
NAME                SALARY
Abdul Ahmad         24000
Ravi Chander        32000
```

(*iv*)
```
SGRADE      SALARY + HRA
S02         44000
```

20 ■ *Differentiate between Data Definition Language and Data Manipulation Language.* (OD 2002)

SOLUTION. The SQL DDL (***Data Definition Language***) provides commands for defining relation schemas, deleting relations, creating indexes and modifying relation schemas.

The SQL DML *(**Data Manipulation Language**)* includes a query language to insert, delete and modify tuples in the database.

DML is used to put values and manipulate them in tables and other database objects and DDL is used to create tables and other databse objects.

HOTS Higher Order Thinking Skills

Solved Problems

1. *Differentiate between SQL commands DROP TABLE and DROP VIEW. Define Second Normal form.* **(CBSE Outside Delhi 2000)**

Solution. DROP TABLE command removes a table from a database and DROP VIEW command removes a view from the data base.

2. *Given the following student relation :* **(CBSE Outside Delhi 1999)**

relation Student

No.	Name	Age	Department	Dateofadm	Fee	Sex
1.	Pankaj	24	Computer	10/01/97	120	M
2.	Shalini	21	History	24/03/98	200	F
3.	Sanjay	22	Hindi	12/12/96	300	M
4.	Sudha	25	History	01/07/99	400	F
5.	Rakesh	22	Hindi	05/09/97	250	M
6.	Shakeel	30	History	27/06/98	300	M
7.	Surya	34	Computer	25/02/97	210	M
8.	Shikha	23	Hindi	31/07/97	200	F

Write SQL commands for (a) to (f) and write output for (g).

(*a*) *To show all information about the students of History department*

(*b*) *To list the names of female students who are in Hindi department*

(*c*) *To list names of all students with their date of admission in ascending order.*

(*d*) *To display student's Name, Fee, Age for male Students only.*

(*e*) *To count the number of student with Age < 23.*

(*f*) *To inset a new row in the STUDENT table with the following data :*

9, "Zaheer'', 36, "Computer'', {12/03/97}, 230, "M''

(*g*) *Give the output of following SQL statements :*

(*i*) Select COUNT (distinct department) from STUDENT ;

(*ii*) Select MAX (Age) from STUDENT where Sex = "F'' ;

(*iii*) Select AVG (Fee) from STUDENT where Dateofadm < {01/01/98} ;

(*iv*) Select SUM (Fee) from STUDENT where Dateofadm < {01/01/98} ;

Solution.

(*a*)
```
SELECT * FROM Student
WHERE Department = "History" ;
```

(*b*)
```
SELECT Name FROM Student
WHERE sex = "F" and
Department = "Hindi" ;
```

(*c*)
```
SELECT name FROM Student
ORDER BY Dateofadm ;
```

(*d*)
```
SELECT Name, Fee, Age
FROM Student
WHERE sex = "M" ;
```

(*e*)
```
SELECT COUNT (*) FROM Student
WHERE Age < 23 ;
```

(*f*)
```
INSERT INTO Student
VALUES (9, "Zaheer", "Computer", "12/03/97", 230, "M") ;
```

(*g*) (*i*) 3 (*ii*) 25 (*iii*) 216 (*iv*) 1080

3. *Given the following tables for a database LIBRARY :* **(CBSE Delhi 2004)**

Table : BOOKS

Book_Id	Book_Name	Author_Name	Publishers	Price	Type	Qty.
C0001	Fast Cook	Lata Kapoor	EPB	355	Cookery	5
F0001	The Tears	William Hopkins	First Publ.	650	Fiction	20
T0001	My First C++	Brian & Brooke	EPB	350	Text	10
T0002	C++ Brainworks	A.W. Rossaine	TDH	350	Text	15
F0002	Thunderbolts	Anna Roberts	First Publ.	750	Fiction	50

Table : ISSUED

Book_Id	Quantity_Issued
T0001	4
C0001	5
F0001	2

Write SQL queries for (a) to (f) :

(*a*) *To show Book name, Author name and Price of books of First Publ. publishers.*

(*b*) *To list the names from books of Text type.*

(*c*) *To display the names and price from books in ascending order of their price.*

(*d*) *To increase the price of all books of EPB Publishers by 50.*

(*e*) *To display the Book_Id, Book_Name and Quantity_Issued for all books which have been issued. (The query will require contents from both the tables.)*

(*f*) *To insert a new row in the table Issued having the following data : "F0003", 1*

(*g*) *Give the output of the following queries based on the above tables :*

(*i*) SELECT COUNT(*) FROM Books ;

(*ii*) SELECT MAX(Price) FROM Books WHERE Quantity >=15 ;

(*iii*) SELECT Book_Name, Author_Name FROM Books WHERE Publishers = "EPB" ;

(*iv*) SELECT COUNT (DISTINCT Publishers) FROM Books WHERE Price >= 400 ;

SOLUTION.

(*a*)
```
SELECT Book_Name, Author_Name, Price
FROM Books
WHERE Publishers = "First Publ." ;
```

(*b*)
```
SELECT Book_Name
FROM Books
WHERE Type = "Text" ;
```

(*c*)
```
SELECT Book_Name, Price
FROM Books
ORDER BY Price ;
```

(*d*)
```
UPDATE Books
SET Price = Price + 50
WHERE Publishers = "EPB" ;
```

(*e*)
```
SELECT Books.Book_Id, Book_Name, Quantity_Issued
FROM Books, Issued
WHERE Books.Book_Id = Issued.Book_Id ;
```

(*f*)
```
INSERT INTO Issued
VALUES("F0003", 1) ;
```

(*g*) (*i*) 5 (*ii*) 750 (*iii*) Fast Cook Lata Kapoor
My First C++ Brian & Brooke (*iv*) 1

4. *Consider the following DEPT and WORKER tables. Write SQL queries for (i) to and (iv) find outputs for SQL queries (v) to (viii) :* **(CBSE Delhi 2015)**

Table : DEPT

DCODE	DEPARTMENT	CITY
D01	MEDIA	DELHI
D02	MARKETING	DELHI
D03	INFRASTRUCTURE	MUMBAI
D05	FINANCE	KOLKATA
D04	HUMAN RESOURCE	MUMBAI

Table : WORKER

WNO	NAME	DOJ	DOB	GENDER	DCODE
1001	George K	2013-09-02	1991-09-01	MALE	D01
1002	Ryma Sen	2012-12-11	1990-12-15	FEMALE	D03
1003	Mohitesh	2013-02-03	1987-09-04	MALE	D05
1007	Anil Jha	2014-01-17	1984-10-19	MALE	D04
1004	Manila Sahai	2012-12-09	1986-11-14	FEMALE	D01
1005	R Sahay	2013-11-18	1987-03-31	MALE	D02
1006	Jaya Priya	2014-06-09	1985-06-23	FEMALE	D05

Note. *DOJ refers to date of joining and DOB refers to date of birth of workers.*

(*i*) *To display Wno, Name, Gender from the table WORKER in descending order of Wno.*

(*ii*) *To display the Name of all the FEMALE workers from the table WORKER.*

(*iii*) *To display the Wno and Name of those workers from the table WORKER who are born between '1987-01-01' and '1991-12-01'.*

(*iv*) *To count and display MALE workers who have joined after '1986-01-01'.*

(*v*)
```
SELECT COUNT(*), DCODE FROM WORKER
GROUP BY DCODE HAVING COUNT(*) > 1 ;
```

(*vi*)
```
SELECT DISTINCT DEPARTMENT FROM DEPT ;
```

(*vii*)
```
SELECT NAME, DEPARTMENT, CITY FROM WORKER W, DEPT D WHERE
W.DCODE = D.DCODE AND WNO < 1003 ;
```

(*viii*)
```
SELECT MAX(DOJ), MIN(DOB) FROM WORKER ;
```

SOLUTION.

(*i*)
```
SELECT WNO, NAME, Gender,
FROM WORKER
ORDER BY WNO Desc ;
```

(*ii*)
```
SELECT NAME
FROM WORKER
WHERE GENDER = 'FEMALE' ;
```

(*iii*)
```
SELECT WNO, NAME
FROM WORKER
WHERE  DOB BETWEEN '1987-01-01' AND '1991-12-01' ;
```

(*iv*)
```
SELECT count(*)
FROM WORKER
WHERE GENDER = 'MALE' AND DOJ > '1986-01-01' ;
```

(*v*)

COUNT(*)	DCODE
2	D01
2	D05

(*vi*)

DISTINCT DEPARTMENT
MEDIA
MARKETING
INFRASTRUCTURE
FINANCE
HUMAN RESOURCE

(*vii*)

NAME	DEPARTMENT	CITY
George K	MEDIA	DELHI
Ryma Sen	INFRASTRUCTURE	MUMBAI

(*viii*)

Max(DOJ)	Min(DOB)
2014-06-09	1984-10-19

UNSOLVED PROBLEMS

1. Consider the following tables Employee and Salary. Write SQL commands for the statements (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*vii*)

Table : Employee

Eid	Name	Depid	Qualification	Sec
1	Deepali Gupta	101	MCA	F
2	Rajat Tyagi	101	BCA	M
3	Hari Mohan	102	B.A.	M
4	Harry	102	M.A.	M
5	Sumit Mittal	103	B.Tech.	M
6	Jyoti	101	M.Tech.	F

Table : Salary

Eid	Basic	D.A.	HRA	Bonus
1	6000	2000	2300	200
2	2000	300	300	30
3	1000	300	300	40
4	1500	390	490	30
5	8000	900	900	80
6	10000	300	490	89

(*i*) To display the frequency of employees department wise.

(*ii*) To list the names of those employees only whose name starts with 'H'

(*iii*) To add a new column in salary table. The column name is Total_Sal.

(*iv*) To store the corresponding values in the Total_Sal column.

(*v*) Select max(Basic) from Salary where Bonus > 40 ;

(*vi*) Select count(*) from Employee group by Sex ;

(*vii*) Select Distinct Depid from Employee ;

2. With reference to following relations PERSONAL and JOB answer the questions that follow :

Create following tables such that *Empno* and *Sno* are not null and unique, *date of birth* is after '12-Jan-1960' , *name* is never blank, *Area* and *Native place* is valid, *hobby, dept* is not empty, salary is between 4000 and 10000.

Table : Personal

Empno	Name	Dobirth	Native-place	Hobby
123	Amit	23-Jan-1965	Delhi	Music
127	Manoj	12-dec-1976	Mumbai	Writing
124	Abhai	11-aug-1975	Allahabad	Music
125	Vinod	04-apr-1977	Delhi	Sports
128	Abhay	10-mar-1974	Mumbai	Gardening
129	Ramesh	28-oct-1981	Pune	Sports

Table : Job

Sno	Area	App_date	Salary	Retd_date	Dept
123	Agra	25-jan-2006	5000	25-jan-2026	Marketing
127	Mathura	22-dec-2006	6000	22-dec-2026	Finance
124	Agra	19-aug-2007	5500	19-aug-2027	Marketing
125	Delhi	14-apr-2004	8500	14-apr-2018	Sales
128	Pune	13-mar-2008	7500	13-mar-2028	Sales

1. Show empno, name and salary of those who have Sports as hobby.
2. Show name of the eldest employee.
3. Show number of employee area wise.
4. Show youngest employees from each Native place.
5. Show Sno, Name, Hobby and Salary in descending order of Salary.
6. Show the hobbies of those whose name pronounces as 'Abhay'.
7. Show the appointment date and native place of those whose name starts with 'A' or ends in 'd'.
8. Show the salary expense with suitable column heading of those who shall retire after 20-jan-2006.
9. Show additional burden on the company in case salary of employees having hobby as sports, is increased by 10%.
10. Show the hobby of which there are 2 or more employees.
11. Show how many employee shall retire today if maximum length of service is 20 years.
12. Show those employee name and date of birth who have served more than 17 years as on date.
13. Show names of those who earn more than all of the employees of Sales dept.
14. Increase salary of the employees by 5 % of their present salary with hobby as Music or they have completed atleast 3 years of service.

Write the output of :

1. Select distinct hobby from personal ;
2. Select avg(salary) from personal, job where Personal.Empno = Job.Sno and Area in ('Agra','Delhi') ;
3. Select count(distinct Native_place) from personal.
4. Select name, max(salary) from personal, job where Personal.Empno = Job.Sno ;

Write SQL statements for the following :

1. Add a new tuple in the table essentially with hobby as Music.
2. Insert a new column email in job table
3. Create a table with values of columns empno, name, and hobby.
4. Create a view of personal and job details of those who have served less than 15 years.
5. Erase the records of employee from job table whose hobby is not Sports.
6. Remove the table personal.

3. With reference to the table below, answer the questions that follow :

Table : Employees

Empid	Firstname	Lastname	Address	City
010	Ravi	Kumar	Raj nagar	GZB
105	Harry	Waltor	Gandhi nagar	GZB
152	Sam	Tones	33 Elm St.	Paris
215	Sarah	Ackerman	440 U.S. 110	Upton
244	Manila	Sengupta	24 Friends street	New Delhi
300	Robert	Samuel	9 Fifth Cross	Washington
335	Ritu	Tondon	Shastri Nagar	GZB
400	Rachel	Lee	121 Harrison St.	New York
441	Peter	Thompson	11 Red Road	Paris

Table : EmpSalary

Empid	Salary	Benefits	Designation
010	75000	15000	Manager
105	65000	15000	Manager
152	80000	25000	Director
215	75000	12500	Manager
244	50000	12000	Clerk
300	45000	10000	Clerk
335	40000	10000	Clerk
400	32000	7500	Salesman
441	28000	7500	Salesman

Write the SQL commands for the following using above tables :

(*i*) To show firstname, lastname, address and city of all employees living in Pairs

(*ii*) To display the content of Employees table in descending order of Firstname.

(*iii*) To display the firstname, lastname and total salary of all managers from the tables Employes and EmpSalary, where total salary is calculated as Salary + Benefits.

(*iv*) To display the maximum salary among managers and clerks from the table EmpSalary.

Give the Output of following SQL commands :

(*i*) Select firstname, Salary from Employees, Empsalary where Designation = 'Salesman' and Employees.Empid = Empsalary.Empid ;

(*ii*) Select count(distinct designation) from EmpSalary ;

(*iii*) Select designation, sum(salary) from EmpSalary group by designation having count(*) >2 ;

(*iv*) Select sum(Benefits) from EmpSalary where Designation = 'Clerk' ;

Glossary

Aggregate Functions	Functions that return single values from groups of values. SUM, AVG, MAX, MIN, COUNT are SQL aggregate functions.
Base Table	A table that contains data not derived from that in any other table.
Join	A combined table of multiple tables resulting from a condition applied on common fields.
SQL	Structured Query Language A language that creates and operates on relational databases.
View	Table whose contents are derived from other table with the use of a query.

ASSIGNMENTS

SHORT ANSWER QUESTIONS

1. Consider the following tables STORE and SUPPLIERS and answer (*a*) and (*b*) parts of this question.

(Delhi 2010)

Table : STORE

ItemNo	Item	Scode	Qty	Rate	LastBuy
2005	Sharpener Classic	23	60	8	31-Jun-09
2003	Ball Pen 0.25	22	50	25	01-Feb-10
2002	Gel Pen Premium	21	150	12	24-Feb-10
2006	Gel Pen Classic	21	250	20	11-Mar-09
2001	Eraser Small	22	220	6	19-Jan-09
2004	Eraser Big	22	110	8	02-Dec-09
2009	Ball Pen 0.5	21	180	18	03-Nov-09

Table : SUPPLIERS

Scode	Sname
21	Premium Stationers
23	Soft Plastics
22	Tetra Supply

(*a*) Write SQL commands for the following statements :

(*i*) To display details of all the items in the Store table in ascending order of LastBuy.

(*ii*) To display ItemNo and Item name of those items from Store table whose Rate is more than 15 Rupees.

(*iii*) To display the details of those items whose Suppliers code (Scode) is 22 or Quantity in Store (Qty) is more than 110 from the table Store.

(*iv*) To display Minimum Rate of items for each Supplier individually as per Scode from the table Store.

(*b*) Give the output of the following SQL queries :

(*i*) SELECT COUNT (DISTINCT Scode) FROM Store ;

(*ii*) SELECT Rate*Qty FROM Store WHERE ItemNo = 2004 ;

(*iii*) SELECT Item, Sname FROM Store S, Suppliers P WHERE S.Scode = P.Scode AND Item No = 2006 ;

(*iv*) SELECT MAX (LastBuy) FROM Store ;

2. Consider the following tables Item and Customer. Write SQL commands for the statement (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*viii*) **(Outside Delhi 2008)**

Table : ITEM

i_ID	*ItemName*	*Manufacturer*	*Price*
PC01	Personal Computer	ABC	35000
LC05	Laptop	ABC	55000
PC03	Personal Computer	XYZ	32000
PC06	Personal Computer	COMP	37000
LC03	Laptop	PQR	57000

Table : CUSTOMER

C_ID	*CustomerName*	*City*	*I_ID*
01	N Roy	Delhi	LC03
06	H Singh	Mumbai	PC03
12	R Pandey	Delhi	PC06
15	C Sharma	Delhi	LC03
16	K Agarwal	Banglore	PC01

(*i*) To display the details of those Customers whose City is Delhi

(*ii*) To display the details of Item whose Price is in the range of 35000 to 55000 (Both values included)

(*iii*) To display the CustomerName, City from table Customer, and ItemName and Price from table Item, with their corresponding matching I_ID

(*iv*) To increase the Price of all Items by 1000 in the table Item

(*v*) SELECT DISTINCT City FROM Customer ;

(*vi*) SELECT ItemName, MAX(Price), Count(*) FROM Item GROUP BY ItemName;

(*vii*) SELECT CustomerName, Manufacturer FROM Item, Customer
WHERE Item.Item_Id = Customer.Item_Id ;

(*viii*) SELECT ItemName, Price * 100 FROM Item WHERE Manufacturer = 'ABC' ;

3. Consider the following tables. Write SQL commands for the statements (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*viii*) **(Delhi 2007)**

TABLE : SENDER

SenderID	*SenderName*	*SenderAddress*	*SenderCity*
ND01	R Jain	2, ABC Appts	New Delhi
MU02	H Sinha	12, Newtown	Mumbai
MU15	S Jha	27/A, Park Street	Mumbai
ND50	T Prasad	122-K, SDA	New Delhi

TABLE : RECIPIENT

RecID	*SenderID*	*RecName*	*RecAddress*	*RecCity*
KO05	ND01	R Bajpayee	5, Central Avenue	Kolkata
ND08	MU02	S Mahajan	116, A Vihar	New Delhi
MU19	ND01	H Singh	2A, Andheri East	Mumbai
MU32	MU15	P K Swamy	B5, C S Terminus	Mumbai
ND48	ND50	S Tripathi	13, B1 D, Mayur Vihar	New Dehli

(*i*) To display the names of all Senders from Mumbai
(*ii*) To display the RecID, SenderName, SenderAddress, RecName, RecAddress for every Recipient
(*iii*) To display Recipient details in ascending order of RecName
(*iv*) To display number of Recipients from each city
(*v*) SELECT DISTINCT SenderCity FROM Sender ;
(*vi*) SELECT A. SenderName, B.RecName FROM Sender A, Recipient B
WHERE A. SenderID = B.SenderID AND B.RecCity = `Mumbai' ;
(*vii*) SELECT RecName, RecAddress FROM Recipient
WHERE RecCity NOT IN(`Mumbai', `Kolkata') ;
(*viii*) SELECT RecID, RecName FROM Recipient
WHERE SenderID = `MU02' OR SenderID = `ND50' ;

4. Consider the following tables WORKERS and PAYLEVEL and answer (A1) and (A2) parts of this question :
(Delhi 2011)

Table : WORKER

ECODE	NAME	DESIG	PLEVEL	DOJ	DOB
11	Radhe Shyam	Supervisor	P001	13-Sep-2004	23-Aug-1981
12	Chander Nath	Operator	P003	22-Feb-2010	12-Jul-1987
13	Fizza	Operator	P003	14-Jun-2009	14-Oct-1983
15	Ameen Ahmed	Mechanic	P002	21-Aug-2006	13-Mar-1984
18	Sanya	Clerk	P002	19-Dec-2005	09-Jun-1983

Table : PAYLEVEL

PLEVEL	PAY	ALLOWANCE
P001	26000	12000
P002	22000	10000
P003	12000	6000

(A1) Write SQL commands for the following statements :
(*i*) To display the details of all WORKERs in descending order of DOB.
(*ii*) To display NAME and DESIG of those WORKERs, whose PLEVEL is either P001 or P002.
(*iii*) To display the content of all the WORKERs table, whose DOB is in between '19-JAN-1984' and '18-JAN-1987'.
(*iv*) To add a new row with the following :
19, 'Daya Kishore', 'Operator', 'P003', '19-Jun-2008', '11-Jul-1984'

(A2) Give the output of the following SQL queries :
(*i*) SELECT COUNT (PLEVEL), PLEVEL FROM WORKER GROUP BY PLEVEL;
(*ii*) SELECT MAX(DOB), MIN(DOJ) FROM WORKER;
(*iii*) SELECT Name, Pay FROM WORKER W, PAYLEVEL P
HERE W.PLEVEL = P.PLEVEL AND W.ECODE < 13;
(*iv*) SELECT PLEVEL, PAY + ALLOWANCE FROM PAYLEVEL WHERE PLEVEL = 'P003';

5. Write SQL statement to create EMPLOYEE relation which contain EmpNo, Name, Skill, PayRate.

6. Create a table with the undermentioned structures

Table : EMP

EmpNo	NUMBER(4)
DeptNo	NUMBER(2)
EmpName	CHAR(10)
Job	CHAR(10)
Manager	NUMBER(4)
Hiredate	DATE
Salary	NUMBER(7, 2)
Commission	NUMBER(7, 2))

Table : PROJECT

ProjId	NUMBER(4)
ProjDesig	CHAR(20)
ProjStartDT	DATE
ProjEndDT	DATE
BudgetAmount	NUMBER(7)
MaxNoStaff	NUMBER (2)

Table : DEPT

DeptNo	NUMBER(2)
DeptName	CHAR(12)
Location	CHAR(12)

7. Create a table called SALGRADE with the columns specified below :

LowSal	NUMBER(7,2)
HighSal	NUMBER(7,2)
Grade	NUMBER(2)

where *LowSal* is the lowest salary limit in the *Grade* and *HighSal* is the highest salary limit in the grade.

8. **Write SQL queries for** (*a*) to (*f*) and write the outputs for the SQL queries mentioned shown in (*g*1) to (*g*4) parts on the basis of tables **PRODUCTS** and **SUPPLIERS** **(Outside Delhi 2013)**

Table : PRODUCTS

Pin	*Pname*	*Qty*	*Price*	*Company*	*Supcode*
101	Digital camera 14X	120	12000	Renix	S01
102	Digital pad 11i	100	22000	Digi pop	S02
104	Pen drive 16 GB	500	1100	Storeking	S01
106	Led screen 32	70	28000	Dispexperts	S02
105	Car GPS system	60	12000	Moveon	S03

Table : SUPPLIERS

Supcode	*Sname*	*City*
S01	Get all inc	Kolkata
S03	Easy market corp	Delhi
S02	Digi busy group	Chennai

(*a*) To display the details of all the products in ascending order of product names (*i.e.*, Pname).

(*b*) To display product name and price of all those products, whose price is in the range of 10000 and 15000 (both values inclusive).

(*c*) To display the number of products, which are supplied by each supplier *i.e.*, the expected output should be :

S01 2
S02 2
S03 1

(*d*) To display the price, product name and quantity (*i.e.*, qty) of those products which have quantity more than 100.

(*e*) To display the names of those suppliers, who are either from DELHI or from CHENNAI.

(*f*) To display the name of the companies and the name of the products in descending order of company names.

(*g*) Obtain the outputs of the following SQL queries based on the data given in tables PRODUCTS and SUPPLIERS above.

(*g*1) SELECT DISTINCT SUPCODE FROM PRODUCTS ;

(*g*2) SELECT MAX (PRICE), MIN (PRICE) FROM PRODUCTS ;

(*g*3)
```
SELECT PRICE*QTY AMOUNT
FROM PROUDCTS WHERE PID = 104 ;
```

(*g*4)
```
SELECT PNAME, SNAME
FROM PRODUCTS P, SUPPLIERS S
WHERE P.SUPCODE = S.SUPCODE AND QTY > 100 ;
```

9. Consider the following tables CABHUB and CUSTOMER and answer (*a*) and (*b*) parts of this question : **(Delhi 2012)**

Table : CABHUB

Vcode	*VehicleName*	*Make*	*Color*	*Capacity*	*Charges*
100	Innova	Toyota	WHITE	7	15
102	SX4	Suzuki	BLUE	4	14
104	C Class	Mercedes	RED	4	35
105	A-Star	Suzuki	WHITE	3	14
108	Indigo	Tata	SILVER	3	12

Table : CUSTOMER

CCode	*CName*	*Vcode*
1	Hemant Sahu	101
2	Raj Lal	108
3	Feroza Shah	105
4	Ketan Dhal	104

(*a*) Write SQL commands for the following statements :

(*i*) To display the names of all the white colored vehicles.

(*ii*) To display name of vehicle, make and capacity of vehicles in ascending order of their seating capacity.

(*iii*) To display the highest charges at which a vehicle can be hired from CABHUB.

(*iv*) To display the customer name and the corresponding name of the vehicle hired by them.

(*b*) Give the output of the following SQL queries :

(*i*) SELECT COUNT (DISTINCT Make) FROM CABHUB ;

(*ii*) SELECT MAX(Charges), MIN(Charges) FROM CABHUB ;

(*iii*) SELECT COUNT(*), Make FROM CABHUB ;

(*iv*) SELECT Vehicle FROM CABHUB WHERE Capacity = 4 ;

10. Differentiate between SQL commands DROP TABLE and DROP VIEW. **(Outside Delhi 2000)**

11. Study the following tables DOCTOR and SALARY and write SQL commands for the questions (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*vi*) : **(Delhi 2006)**

Table : DOCTOR

ID	*NAME*	*DEPT*	*SEX*	*EXPERIENCE*
101	John	ENT	M	12
104	Smith	ORTHOPEDIC	M	5
107	George	CARDIOLOGY	M	10
114	Lara	SKIN	F	3
109	K George	MEDICINE	F	9
105	Johnson	ORTHOPEDIC	M	10
117	Lucy	ENT	F	3
111	Bill	MEDICINE	F	12
130	Morphy	ORTHOPEDIC	M	15

Table : SALARY

ID	*BASIC*	*ALLOWANCE*	*CONSULTATION*
101	12000	1000	300
104	23000	2300	500
107	32000	4000	500
114	12000	5200	100
109	42000	1700	200
105	18900	1690	300
130	21700	2600	300

(*i*) Display NAME of all doctors who are in "MEDICINE" having more than 10 years experience from the table DOCTOR.

(*ii*) Display the average salary of all doctors working in "ENT" department using the tables DOCTOR and SALARY. Salary = BASIC + ALLOWANCE.

(*iii*) Display the minimum ALLOWANCE of female doctors.

(*iv*) Display the highest consultation fee among all male doctors.

(*v*) SELECT count (*) from DOCTOR where SEX = "F"

(*vi*) SELECT NAME, DEPT, BASIC from DOCTOR, SALARY
WHERE DEPT = "ENT" AND DOCTOR.ID = SALARY.ID

12. (*a*) What are DDL and DML commands ?

(*b*) Study the following tables FLIGHTS and FARES and write SQL commands for the questions **(*i*)** to **(*iv*)** and give outputs for SQL queries **(*v*)** to **(*vi*).** **(Outside Delhi 2006)**

Table : FLIGHTS

FL_NO	STARTING	ENDING	NO_FLIGHTS	NO_STOPS
IC301	MUMBAI	DELHI	8	0
IC799	BANGALORE	DELHI	2	1
MC101	INDORE	MUMBAI	3	0
IC302	DELHI	MUMBAI	8	0
AM812	KANPUR	BANGALORE	3	1
IC899	MUMBAI	KOCHI	1	4
AM501	DELHI	TRIVANDRUM	1	5
MU499	MUMBAI	MADRAS	3	3
IC701	DELHI	AHMEDABAD	4	0

Table : FARES

FL_NO	AIRLINES	FARE	TAX%
IC701	Indian Airlines	6500	10
MU499	Sahara	9400	5
AM501	Jet Airways	13450	8
IC899	India Airlines	8300	4
IC302	Indian Airlines	4300	10
IC799	Indian Airlines	10500	10
MC101	Deccan Airlines	3500	4

(*i*) Display FL_NO and NO_FLIGHTS from "KANPUR" to "BANGALORE" from the table FLIGHTS.

(*ii*) Arrange the contents of the table FLIGHTS in the ascending order of FL_NO.

(*iii*) Display the FL_NO and fare to be paid for the flights from DELHI to MUMBAI using the tables FLIGHTS and FARES, where the fare to be paid = FARE + FARE*TAX%/100.

(*iv*) Display the minimum fare "Indian Airlines" is offering from the table FARES.

(*v*) SELECT FL_NO, NO_FLIGHTS, AIRLINES from FLIGHTS, FARES
WHERE STARTING = "DELHI" AND FLIGHTS.FL_NO = FARES.FL_NO.

(*vi*) SELECT count(distinct ENDING) from FLIGHTS.

13. *Consider the following tables EMPLOYEES and EMPSALARY. Write SQL commands for the statements (i) to (iv) and give outputs for SQL queries (v) to (viii).* **(Delhi 2005)**

Table : Employees

EMPID	FIRSTNAME	LASTNAME	ADDRESS	CITY
010	George	Smith	83 First Street	Howard
105	Mary	Jones	842 Vine Ave.	Losantiville

EMPID	FIRSTNAME	LASTNAME	ADDRESS	CITY
152	Sam	Tones	33 Elm St.	Paris
215	Sarah	Ackerman	440 U.S. 110	Upton
244	Manila	Sengupta	24 Friends Street	New Delhi
300	Robert	Samuel	9 Fifth Cross	Washington
335	Henry	Williams	12 Moore Street	Boston
400	Rachel	Lee	121 Harrison St.	New York
441	Peter	Thompson	11 Red Road	Paris

Table : Empsalary

EMPID	SALARY	BENEFITS	DESIGNATION
010	75000	15000	Manager
105	65000	15000	Manager
152	80000	25000	Director
215	75000	12500	Manager
244	50000	12000	Clerk
300	45000	10000	Clerk
335	40000	10000	Clerk
400	32000	7500	Salesman
441	28000	7500	Salesman

(*i*) To display Firstname, Lastname, Address and City of all employees living in Paris from the table Employees.

(*ii*) To display the content of Employees table in descending order of FIRSTNAME.

(*iii*) To display the Firstname, Lastname, and Total Salary of all Managers from the tables Employees and Empsalary, where Total Salary is calculated as Salary + Benefits.

(*iv*) To display the Maximum salary among Managers and Clerks from the table Empsalary.

(*v*) SELECT Firstname, Salary
FROM Employees, Empsalary
WHERE DESIGNATI ON = 'Salesman' AND
EMPLOYEES.EMPID = Empsalary.EMPID ;

(*vi*) SELECT COUNT(DISTINCT DESIGNATION) FROM Empsalary ;

(*vii*) SELECT DESIGNATION, SUM(SALARY)
FROM Empsalary
GROUP BY DESIGNATION HAVING COUNT (*) > 2 ;

(*viii*) SELECT SUM(BENEFITS)
FROM Employees
WHERE DESIGNATION = 'Clerk' ;

14. Consider the following tables GARMENT and FABRIC. Write SQL commands for the statements (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*viii*). **(Delhi 2009)**

Table : GARMENT

GCODE	Description	Price	FCODE	READYDATE
10023	PENCIL SKIRT	1150	F03	19-DEC-08
10001	FORMAL SHIRT	1250	F01	12-JAN-08
10012	INFORMAL SHIRT	1550	F02	06-JUN-08
10024	BABY TOP	750	F03	07-APR-07
10090	TULIP SKIRT	850	F02	31-MAR-07
10019	EVENING GOWN	850	F03	06-JUN-08
10009	INFORMAL PANT	1500	F02	20-OCT-08
10007	FORMAL PANT	1350	F01	09-MAR-08
10020	FROCK	850	F04	09-SEP-07
10089	SLACKS	750	F03	31-OCT-08

Table : FABRIC

FCODE	TYPE
F04	POLYSTER
F02	COTTON
F03	SILK
F01	TERELENE

(*i*) To display GCODE and DESCRIPTION of each GARMENT in descending order of GCODE.

(*ii*) To display the details of all the GARMENTs, which have READYDATE in between 08-DEC-07 and 16-JUN-08 (inclusive of both the dates).

(*iii*) To display the average PRICE of all the GARMENTs, which are made up of FABRIC with FCODE as F03.

(*iv*) To display FABRIC wise highest and lowest price of GARMENTs from GARMENT table. (Display FCODE of each GARMENT along with highest and lowest price).

(*v*) SELECT SUM (PRICE) FROM GARMENT WHERE FCODE = 'F01' ;

(*vi*) SELECT DESCRIPTION, TYPE FROM GARMENT, FABRIC WHERE GARMENT.FCODE = FABRIC.FCODE AND GARMENT.PRICE >= 1260 ;

(*vii*) SELECT MAX (FCODE) FROM FABRIC ;

(*viii*) SELECT COUNT (DISTINCT PRICE) FROM GARMENT ;

15. Consider the following tables DRESS and Material. Write SQL commands for the statements (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*viii*). **(Outside Delhi 2009)**

Table : DRESS

DCODE	DESCRIPTION	PRICE	MCODE	LAUNCHDATE
10001	FORMAL SHIRT	1250	M001	12-JAN-08
10020	FROCK	750	M004	09-SEP-07

DCODE	DESCRIPTION	PRICE	MCODE	LAUNCHDATE
10012	INFORMAL SHIRT	1450	M002	06-JUN-08
10019	EVENING GOWN	850	M003	06-JUN-08
10090	TULIP SKIRT	850	M002	31-MAR-07
10023	PENCIL SKIRT	1250	M003	19-DEC-08
10089	SLACKS	850	M003	20-OCT-08
10007	FORMAL PANT	1450	M001	09-MAR-08
10009	INFORMAL PANT	1400	M002	20-OCT-08
10024	BABY TOP	650	M003	07-APR-07

Table : MATERIAL

MCODE	TYPE
M001	TERELENE
M002	COTTON
M004	POLYESTER
M003	SILK

(*i*) To display DCODE and DESCRIPTION of each dress in ascending order of DCODE.

(*ii*) To display the details of all the dresses which have LAUNCHDATE in between 05-DEC-07 and 20-JUN-08 (inclusive of both the dates).

(*iii*) To display the average PRICE of all the dresses which are made up of material with MCODE as M003.

(*iv*) To display materialwise highest and lowest price of dresses from DRESS table. (Display MCODE of each dress along with highest and lowest price)

(*v*) SELECT SUM (PRICE) FROM DRESS WHERE
MCODE = 'M001' ;

(*vi*) SELECT DESCRIPTION, TYPE FROM DRESS,
MATERIAL WHERE DRESS.MCODE =
MATERIAL.MCODE AND DRESS.PRICE >= 1250 ;

(*vii*) SELECT MAX(MCODE) FROM MATERIAL ;

(*viii*) SELECT COUNT(DISTINCT PRICE) FROM DRESS ;

16. Consider the following tables Stationery and Consumer. Write SQL commands for the statement (*i*) to (*iv*) and give outputs for SQL queries (*v*) to (*viii*) **(Delhi 2008C)**

Table : STATIONERY

S_ID	StationeryName	Company	Price
DP01	Dot Pen	ABC	10
PL02	Pencil	XYZ	6
ER05	Eraser	XYZ	7
PL01	Pencil	CAM	5
GP02	Gel Pen	ABC	15

Table : CONSUMER

C_ID	ConsumerName	Address	S_ID
01	Good Learner	Delhi	PL01
06	Write Well	Mumbai	GP02
12	Topper	Delhi	DP01
15	Write & Draw	Delhi	PL02
16	Motivation	Banglore	PL01

(*i*) To display the details of those Consumers whose Address is Delhi

(*ii*) To display the details of Stationery whose Price is in the range of 8 to 15 (Both values included)

(*iii*) To display the ConsumerName, Address from Table Consumer, and Company and Price from table Stationery, with their corresponding matching S_ID

(*iv*) To increase the Price of all Stationery by 2

(*v*) SELECT DISTINCT Address FROM Consumer ;

(*vi*) SELECT Company, MAX(Price), Min(Price), Count(*)
FROM Stationery GROUP BY Company ;

(*vii*) SELECT Consumer.ConsumerName, Stationery.StationeryName,
Stationery.Price FROM Stationery, Consumer
WHERE Consumer.S_Id = Stationery.S_Id

(*viii*) SELECT StationeryName, Price * 3
FROM Stationery