What is Database ?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs (Application Programming Interface) for creating, accessing, managing, searching and replicating the data it holds.

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as foreign keys.

A Relational DataBase Management System (RDBMS) is software that:

- Enables you to implement a database with tables, columns and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Updates the indexes automatically.
- Interprets an SQL query and combines information from various tables.

RDBMS Terminology:

Before we proceed to explain MySQL database system, let's revise few definitions related to database.

- **Database:** A database is a collection of tables, with related data.
- **Table:** A table is a matrix with data. A table in a database looks like a simple spreadsheet.
- **Column:** One column (data element) contains data of one and the same kind, for example the column emp_name contains names of employees.
- **Row:** A row (tuple, entry or record) is a group of related data, for example the data of one employee.
- **Redundancy:** Storing data twice, redundantly to make the system faster.
- **Primary Key:** A primary key is unique. A key value cannot occur twice in one table. With a key, you can find at most one row.
- **Foreign Key:** A foreign key is the linking pin between two tables.
- **Compound Key:** A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
- **Index:** An index in a database resembles an index at the back of a book.
- **Referential Integrity:** Referential Integrity makes sure that a foreign key value always points to an existing row.

MySQL Database:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

MySQL uses many different data types broken into three categories: numeric, date and time, and string types.

Numeric Data Types:

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions:

- **INT** - A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **FLOAT(M,D)** - A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
- **DOUBLE(M,D)** - A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE.

- **DECIMAL(M,D) -** An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Date and Time Data Types:

The MySQL date and time datatypes are:

- **DATE -** A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME -** A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP -** A timestamp between midnight, January 1, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYY-MM-DD HH:MM:SS).
- **TIME** - Stores the time in HH:MM:SS format.
- **YEAR(M) -** Stores a year in 2-digit or 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4.

String Data Types:

Although numeric and date types are fun, most data you'll store will be in string format. This list describes the common string datatypes in MySQL.

- **CHAR(M) -** A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

- **VARCHAR(M) -** A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). You must define a length when creating a VARCHAR field.

SHOW DATABASES

You can use SHOW DATABASES to list all the existing databases in the server.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
.....................
```

Databases "mysql", "information_schema" and "performance_schema" are system databases used internally by MySQL. A "test" database is provided during installation for your testing.

Creating a Database

You can create a new database using SQL command.

Syntax: CREATE DATABASE *databaseName* ;

Ex: CREATE DATABASE COMPANY;

```
Query OK, 1 row affected (0.03 sec)
```

Setting the Default Database - USE

This command sets a particular database as the default (or current) database.

Syntax: USE *databaseName*
Ex: USE COMPANY
Database changed

Show all the tables in the current database.

mysql> SHOW TABLES;

```
+--------------------+
| Tables_in_company  |
+--------------------+
| person             |
.....................
```

Describe the fields (columns) of the "person" table

mysql> Desc Person;

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| ssn   | int(11)     | YES  |     | NULL    |       |
| name  | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.06 sec)
```

EXPERIMENT 1:-

Using any open source data modeling tool (MySQL/Rational Rose/ERwin), design and develop a ER diagram for the following Library database. Specify relevant Referential constraints.

BOOK_AUTHORS

BOOK_ID	AUTHOR_NAME

PUBLISHER

NAME	ADDRESS	PHONE

BOOK_COPIES

BOOK_ID	BRANCH_ID	NO_COPIES

BOOK_LOANS

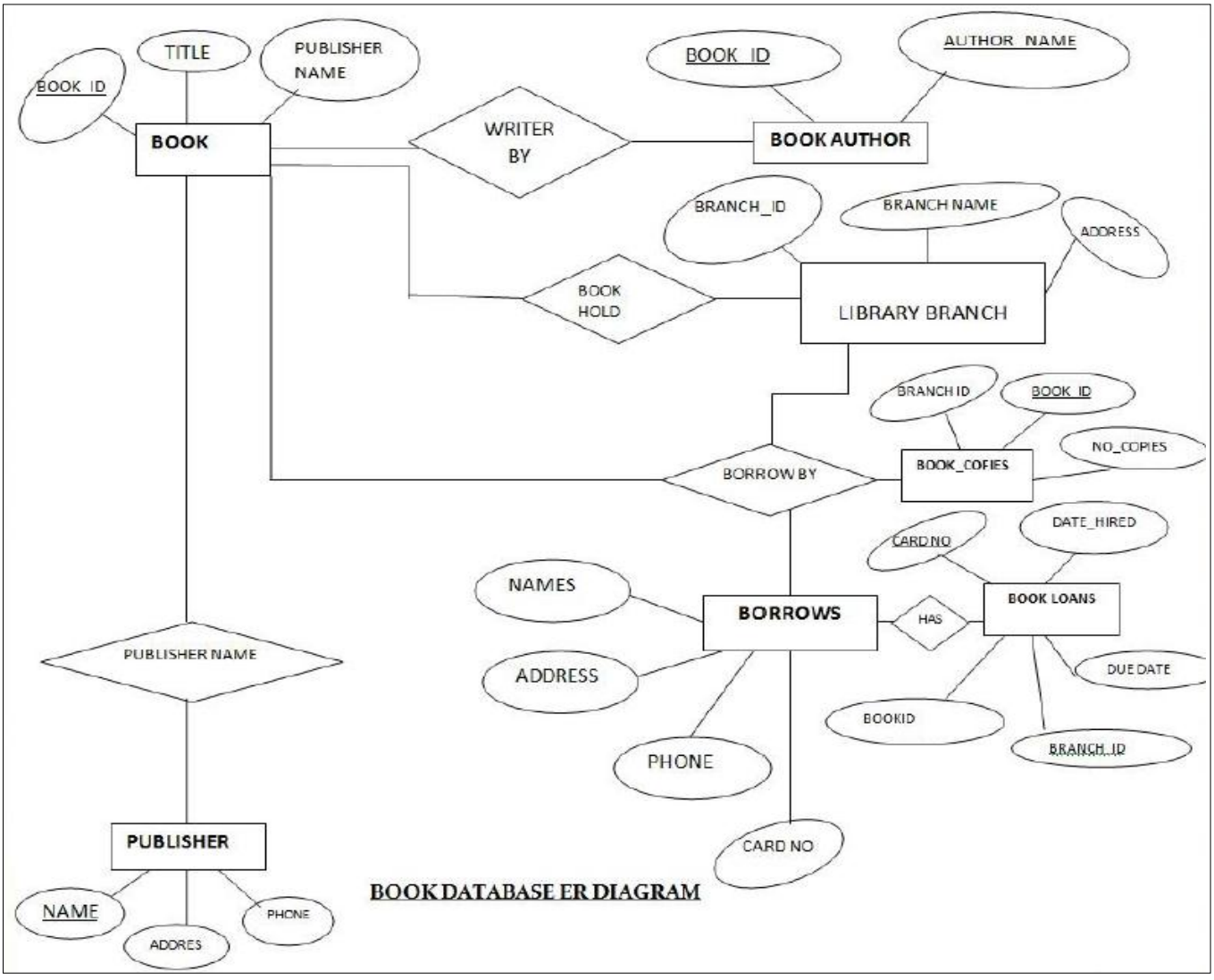
BOOK_ID	BRANCH_ID	CARD_NO	DATE_HIRED	DUE_DATE

LIBRARY_BRANCH

BRANCH_ID	BRANCH_NAME	ADDRESS

BORROWER

CARD_NO	NAME	ADDRESS	PHONE



EXPERIMENT 3:-

Create the following tables for a COMPANY database.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

EMPLOYEE TABLE

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

To create table of employee:

```
mysql> CREATE TABLE EMPLOYEE(
    ->  FNAME VARCHAR(10) NOT NULL,
    ->  MINIT CHAR(9),
    ->  LNAME VARCHAR(15) NOT NULL,
    ->  SSN VARCHAR(12),
    ->  BDATE DATE,
    ->  ADDRESS VARCHAR(10),
    ->  SEX CHAR(9),
    ->  SALARY INT,
    ->  SUPER_SSN VARCHAR(9),
    ->  DNO INT,
    ->  PRIMARY KEY(SSN),
    ->  FOREIGN KEY(SUPER_SSN)REFERENCES EMPLOYEE(SSN));
```

Query OK, 0 rows affected (0.19 sec)

INSERTING VALUES INTO EMPLOYEE TABLE:

```
mysql> INSERT INTO EMPLOYEE VALUES('Vinayak','V','BHAT','A10','1999-02-
23','Honnavar','m',27000,'A10',10);
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO EMPLOYEE VALUES('Vinod','s','Fernandis','A11','1992-01-12','H
onnavar','M',25000,'A11',11);
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO EMPLOYEE VALUES('Ravinand','G','Naik','A12','1991-06-20','Hon
 navar','M',26000,'A12',12);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO EMPLOYEE VALUES('Ram','D','Patil','A13','1991-12-27','Belgaum
','M',23000,'A13',13);
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO EMPLOYEE VALUES('HARESH','N','Devadiga','A14','1993-07-13','D
arwad','M',20000,'A14',14);
Query OK, 1 row affected (0.01 sec)
```

mysql> select * from employee;

```
FNAME     M LNAME     SSN   BDATE    ADDRESS    S   SALARY SUPER_SSN DNO
---------- - ---------- ---------- --------- ---------- - ---------
VINAYAK   V BHAT       A10 23-FEB-93 HONNAVAR M   27000    A10   10
VINOD     S FERNANDIS A11 12-JAN-92 HONNAVAR M   25000    A11   11
RAVINAND G NAIK       A12 20-JUN-91 HONNAVAR M   26000    A12   12
RAM       D PATIL     A13 27-DEC-91 BELGAM   M   23000    A13   13
HAREESH   N DEVADIGA  A14 13-JUL-91 DHARWAD  M   20000    A14   14
```

5 rows in set (0.00 sec)

DEPARTMENT TABLE

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

To create department table:

```
mysql> CREATE TABLE DEPARTMENT(
    ->  DNAME VARCHAR(12) NOT NULL,
    ->  DNUMBER INT NOT NULL,
    ->  MGR_SSN CHAR(9) NOT NULL,
    ->  MGR_STARY_DATE DATE,
    ->  PRIMARY KEY (DNUMBER),
    ->  UNIQUE (DNAME),
    ->  FOREIGN KEY(MGR_SSN)REFERENCES EMPLOYEE(SSN));
```

Query OK, 0 rows affected (0.09 sec)

Inserting Values Into Department Table:

```
mysql> INSERT INTO DEPARTMENT VALUES('Om',10,'A10','1997-02-12');
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO DEPARTMENT VALUES('Satyam',11,'A11','1999-12-13');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO DEPARTMENT VALUES('RELIANCE',12,'A12','2000-03-30');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO DEPARTMENT VALUES('infosis',13,'A13','1999-01-12');
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO DEPARTMENT VALUES('Research',14,'A14','1998-04-11');
Query OK, 1 row affected (0.03 sec)
```

mysql> select * from department;

```
DNAME         DNUMBER MGR_SSN    MGR_STARY
---------- --------- ---------- ---------
OM                 10 A10        12-FEB-07
SATYAM             11 A11        13-DEC-09
RELIANCE           12 A12        30-MAR-10
INFOSIS            13 A13        12-JAN-99
REALTECH           14 A14        11-APR-98
```

5 rows in set (0.00 sec)

DEPT_LOCATION TABLE

DEPT_LOCATIONS

Dnumber	Dlocation

To create dept_location Table:

```
mysql> CREATE TABLE DEPT_LOCATION(
    ->  DNUMBER INT,
    ->  DLOCATION VARCHAR(12) NOT NULL,
    ->  PRIMARY KEY(DNUMBER,DLOCATION),
    ->  FOREIGN KEY(DNUMBER) REFERENCES DEPARTMENT(DNUMBER));
```

Query OK, 0 rows affected (0.09 sec)

Inserting values into dept_location:

```
mysql> INSERT INTO DEPT_LOCATION VALUES(10,'Bangalore');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO DEPT_LOCATION VALUES(11,'dharwad');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO DEPT_LOCATION VALUES(12,'karwar');
Query OK, 1 row affected (0.16 sec)

mysql> INSERT INTO DEPT_LOCATION VALUES(13,'Hubli');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO DEPT_LOCATION VALUES(14,'Mangalore');
Query OK, 1 row affected (0.05 sec)
```

mysql> select * from dept_location;

```
DNUMBER DLOCATION
------- ----------
   10   BANGLORE
   11   DHARWAD
   12   KARWAR
   13   HUBLI
   14   MANGLORE
```

5 rows in set (0.00 sec)

PROJECT TABLE

PROJECT

Pname	Pnumber	Plocation	Dnum

To create project table:

```
mysql> CREATE TABLE PROJECT(
    ->  PNAME VARCHAR(12) NOT NULL,
    ->  PNUMBER INT,
    ->  PLOCATION VARCHAR(10),
    ->  DNUM INT,
    ->  PRIMARY KEY(PNUMBER),
    ->  UNIQUE(PNAME),
    ->  FOREIGN KEY(DNUM) REFERENCES DEPARTMENT(DNUMBER));
```

Query OK, 0 rows affected (0.17 sec)

Inserting values into project:

```
mysql> INSERT INTO PROJECT VALUES('FUTURE',100,'Bangalore',10);
Query OK, 1 row affected (0.34 sec)

mysql> INSERT INTO PROJECT VALUES('WIN_XP',110,'MYSORE',11);
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO PROJECT VALUES('rel_3d',120,'Darwad',12);
Query OK, 1 row affected (0.84 sec)

mysql> INSERT INTO PROJECT VALUES('i_pod',130,'mysore',13);
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO PROJECT VALUES('3d_AUD',140,'manglore',14);
Query OK, 1 row affected (0.05 sec)
```

mysql> select * from project;

```
PNAME         PNUMBER PLOCATION        DNUM
---------- --------- ---------- ---------
FUTURE S/W        100 BANGLORE           10
WIN_XP OS         110 MYSORE             11
REL 3D            120 DHARWAD            12
I-POD 2011        130 MYSORE             13
3D AUD S/W        140 MANGLORE           14
```

5 rows in set (0.00 sec)

WORKS_ON TABLE

WORKS_ON

Essn	Pno	Hours

To create works_on table:

```
mysql> CREATE TABLE WORKS_ON(
    ->     ESSN VARCHAR(12),
    ->     PNO INT,
    ->     HOURS DECIMAL(10,2),
    ->     PRIMARY KEY(ESSN,PNO),
    ->     FOREIGN KEY(ESSN) REFERENCES EMPLOYEE(SSN),
    ->     FOREIGN KEY(PNO) REFERENCES PROJECT(PNUMBER));
```

Query OK, 0 rows affected (0.20 sec)

Inserting values into work_on:

```
mysql> INSERT INTO WORKS_ON VALUES('A10',100,'2.30');
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO WORKS_ON VALUES('A11',110,'5.30');
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO WORKS_ON VALUES('A12',120,'6.30');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO WORKS_ON VALUES('A13',130,'7.30');
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO WORKS_ON VALUES('A14',140,'2.30');
Query OK, 1 row affected (0.03 sec)
```

mysql> select * from works_on;

```
ESSN            PNO         HOURS
-----        -------     ---------
A10             100           6
A11             110           8
A12             120           6
A13             130           5
A14             140           8

5 rows in set (0.00 sec)
```

DEPENDENT TABLE

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

To create dependent table:

```
mysql> CREATE TABLE DEPENDENT(
    ->  ESSN VARCHAR(12),
    ->  DEPENDENT_NAME VARCHAR(12) NOT NULL,
    ->  SEX CHAR(9),
    ->  BDATE DATE,
    ->  RELATIONSHIP VARCHAR(12),
    ->  PRIMARY KEY(ESSN,DEPENDENT_NAME),
    ->  FOREIGN KEY(ESSN) REFERENCES EMPLOYEE(SSN));
```

Query OK, 0 rows affected (0.09 sec)

Inserting values into dependent table:

```
mysql> INSERT INTO DEPENDENT VALUES('A10','Vasant','m','1965-12-13','father');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO DEPENDENT VALUES('A11','Santosh','m','1965-02-14','father');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO DEPENDENT VALUES('A12','Maruthi','m','1989-02-14','Brother');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO DEPENDENT VALUES('A13','Pappu','m','2011-02-14','son');
Query OK, 1 row affected (0.07 sec)
```

mysql> select * from dependent;

```
ESSN       DEPENDENT_ S BDATE     RELATION
---------- ---------- - --------- --------
A10        VASANT     M 13-DEC-65 FATHER
A11        SANTOSH    M 14-FEB-65 FATHER
A12        MARUTI     M 14-FEB-89 BROTHER
A13        PAPPU      M 14-FEB-11 SON
```

4 rows in set (0.00 sec)

EXPERIMENT 4:-

Illustrate the use of constraints

- **NOT NULL**
- **PRIMARY KEY**
- **UNIQUE**
- **CHECK**
- **DEFAULT**
- **REFERENCES**

NOT NULL CONSTRAINT:-

The NOT NULL constraint enforces a column to NOT accept NULL values. The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

Syntax: column name data type (size) **not null**;

Ex: Fname varchar (20) **not null**;

PRIMARY KEY CONSTRAINT:-

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values. A primary key column cannot contain NULL values. Most tables should have a primary key, and each table can have only ONE primary key.

Syntax: column name data type (size) **primary key**;

Ex: Ssn char (9) **primary key**;

Or

Primary key (Ssn);

UNIQUE CONSTRAINT:-

The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it. Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Syntax: column name data type (size) **unique**;

Ex: dname varchar (20) **unique**;

Or

unique (dname);

CHECK CONSTRAINT:-

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

Syntax: column name data type **check** (value range);

Ex: dno int **CHECK** (dno>1 and dno<5);
Age int **CHECK** (Age>=18)

DEFAULT CONSTRAINT:-

The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

Syntax: column name data type **DEFAULT** 'Default value';

Ex: City varchar(255) **DEFAULT** 'Karwar';

*Note: The default value of City is set to Karwar.

REFERENCES CONSTRAINT:-

This constraint is used to link two tables together. This constraint is also known as **referential integrity constraint**. These constraints are specified using foreign key. A FOREIGN KEY (referencing key) is a field or collection of fields in one table that refers to the PRIMARY KEY in another table.

Ex: FOREIGN KEY (mgr_ssn) **REFERENCES** EMPLOYEE (ssn)

*Note that the "mgr_ssn" column in the "Department" table points to the "ssn" column in the "Employee" table.

*The "**ssn**" column in the "**employee**" table is the **PRIMARY KEY** in the "Employee" table.

*The "**mgr_ssn**" column in the "**Department**" table is a **FOREIGN KEY** in the "Department" table.

EXPERIMENT 5:-

DATA MANIPULATION: INSERTING VALUES INTO A TABLE.

The **INSERT** is used to add a single tuple to a relation. We must specify the relation name and a list of values for the tuple. The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.

EX: INSERT INTO EMPLOYEE **VALUES** ('Richard', 'Marini', '653298653', '1962-12-30', '98' ,Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);

A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command. This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.

EX: INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)

VALUES ('Richard', 'Marini', 4, '653298653');

EXPERIMENT 6:-

Illustrate the use of SELECT statement.

The **SELECT** statement is used to select data from a database. The result is stored in a result table, called the result-set.

SELECT statement can be depicted in two ways, as follows

Syntax 1: SELECT *column_name1* , *column_name2* FROM *table_name*;

Ex: Select Ssn, Name From employee;

Syntax 2: SELECT * FROM *table_name*;

Ex: Select * From employee;

Query: Retrieve FNAME,BDATE from employee table

```
mysql> Select FNAME, BDATE
    -> from employee;
```

FNAME	BDATE
VINAYAK	23-FEB-93
VINOD	12-JAN-92
RAVINAND	20-JUN-91
RAM	27-DEC-91
HAREESH	13-JUL-91

EXPERIMENT 7:-

Conditional retrieval - WHERE clause.

The **WHERE** clause is used to filter records obtained from the SELECT statement according to the condition applied. The WHERE clause is used to extract only those records that fulfill a specified criterion.

Syntax: SELECT *column_name,column_name*
FROM *table_name*
WHERE *column_name = operator value*;

Query: Retrieve the birth date and address of the employee whose name is SAYYAN SHAIKH.

```
mysql> SELECT BDATE, ADDRESS
    -> FROM EMPLOYEE
    -> WHERE FNAME=' SAYYAN' AND MINIT='N' AND LNAME='SHAIKH';
   BDATE       ADDRESS
---------  ----------
14-AUG-89   HALIYAL

1 rows in set (0.13 sec)
```

Query: Retrieve the name and address of all employees who works for the INFOSYS department.

```
mysql> SELECT FNAME, LNAME, ADDRESS
    ->  FROM EMPLOYEE, DEPARTMENT
    ->  WHERE DNAME='INFOSYS' AND DNO=DNUMBER;

   FNAME         LNAME       ADDRESS
--------   ----------  ----------
   VINOD      FERNANDIS    KARWAR

1 rows in set (0.13 sec)
```

EXPERIMENT 8:-

Query sorted - ORDER BY clause.

The **ORDER BY** keyword is used to sort the result-set by one or more columns either in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

Syntax:

SELECT *column_name*, *column_name* **FROM** *table_name* **ORDER BY** *column_name* **ASC|DESC**, *column_name* **ASC|DESC**;

Query: Retrieve a list of employee's fname & lname order by fname in Ascending order.

```
mysql> SELECT FNAME, LNAME
    -> FROM EMPLOYEE
    -> ORDER BY FNAME ASC;
```

```
   FNAME       LNAME
---------- ----------
  HAREESH     DEVADIGA
  RAM         PATIL
  RAVINAND    NAIK
  VINAYAK     BHAT
  VINOD       FERNANDIS

5 rows in set (0.00 sec)
```

Query: Retrieve a list of employee's fname & lname order by fname in descending order

```
mysql> SELECT FNAME, LNAME
    -> FROM EMPLOYEE
    -> ORDER BY FNAME DESC;
```

```
  FNAME       LNAME
---------- ----------
VINOD       FERNANDIS
VINAYAK     BHAT
RAVINAND    NAIK
RAM         PATIL
HAREESH     DEVADIGA

5 rows in set (0.00 sec)
```

EXPERIMENT 9:-

Grouping the result of query - GROUP BY clause and HAVING clause.

The **GROUP BY** statement is used in conjunction with the aggregate functions to group the result-set by one or more columns. Aggregate functions often need an added GROUP BY statement.

Syntax: **SELECT** column_name, **aggregate_function**(column_name)
FROM table_name **WHERE** column_name operator value **GROUP BY** column_name;

Query: Retrieve each department no, the no. of employee in department and their average salary.

```
mysql> SELECT DNO, COUNT(*), AVG(SALARY)
    -> FROM EMPLOYEE
    -> GROUP BY DNO;
```

```
  DNO     COUNT(*)  AVG(SALARY)
------- -------- -----------
  10        1       27000.00
  11        1       25000.00
  12        1       26000.00
  13        1       23000.00
  14        1       20000.00
```

5 rows in set (0.00 sec)

HAVING clause:

HAVING is similar to WHERE, but it can operate on the GROUP BY aggregate functions; whereas WHERE operates only on columns.

Example:

```
mysql> Select ssn , count(*), avg(salary)
    -> From employee
    -> Group by dno
    -> Having count(*)>=2;
+------+-----------+-----------------+
| ssn | count(*) | avg(salary)   |
+------+-----------+-----------------+
|1237|     2    | 55000.0000  |
+------+-----------+-----------------+

1 row in set (0.00 sec)
```

Query: For each department that has more than one employee, retrieve the department no. and the no. of its employees who are making more than $23000.

```
mysql>  SELECT DNUMBER,COUNT(*)
    ->  FROM DEPARTMENT, EMPLOYEE
    ->  WHERE DNUMBER=DNO AND SALARY>23000 AND
    ->  DNO IN(SELECT DNO
    ->  FROM EMPLOYEE
    ->  GROUP BY DNO  HAVING COUNT(*)>0);
```

```
  DNUMBER    COUNT(*)
--------   ---------
  10          1
  11          1
  12          1
```

3 rows in set (0.00 sec)

EXPERIMENT 10:-

Aggregate functions in SQL (Count, Sum, Max, Min, Avg).

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:-

AVG() - Returns the average value

COUNT() - Returns the number of rows

FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - Returns the sum.

Query: Find the sum of salary, minimum, maximum, average of salary from employee table

```
mysql > SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
    -> FROM EMPLOYEE;
```

```
SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)
----------- ----------- ----------- -----------
 121000        27000       20000      24200.0000
1 row in set (0.00 sec)
```

Query: Find the total no of employees in the company

```
mysql > SELECT COUNT(*)
    -> FROM EMPLOYEE;
```

```
 COUNT(*)
---------
    5
1 row in set (0.00 sec)
```

***Note:** The function **COUNT(*)** returns the rows selected; COUNT(*columnName*) counts only the non-NULL values of the given column.

Query: Count the no. of distinct salary from employee table.

```
mysql> SELECT count(distinct salary)
    -> FROM EMPLOYEE;
```

```
COUNT(DISTINCTSALARY)
---------------------
          5
1 row in set (0.00 sec)
```

***Note:** A column may have duplicate values, we could use keyword **DISTINCT** to select only distinct values. We can also apply DISTINCT to several columns to select distinct combinations of these columns.

EXPERIMENT 11:-

SQL operators (And, Or, Like, Between, In).

LIKE operator allows comparison conditions on only part of a character string. This can be used for string pattern matching. Partial strings are specified using two reserved characters: % replaces an arbitrary number of zero or more characters, and the underscore (_) replaces a single character. Another comparison operator, which can be used for convenience, is **BETWEEN.**

The standard **Arithmetic operators** for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric values.

The wildcard '_'matches any single character; '%' matches any number of characters (including zero). For example

- 'abc%' matches strings beginning with 'abc';
- '%xyz' matches strings ending with 'xyz';
- '%aaa%' matches strings containing 'aaa';
- '___' matches strings containing exactly three characters; and
- 'a_b%' matches strings beginning with 'a', followed by any single character, followed by 'b', followed by zero or more characters.

Query: Retrieve all employees whose address is in KARWAR.

mysql> **SELECT** fname, lname **From** employee **WHERE** address **LIKE** '%KARWAR%';

Query: Retrieve all employees in department 5 whose salary $15000 and $100000

mysql> **SELECT** * **FROM** employee

- > **WHERE** (salary **BETWEEN** 15000 **AND** 100000) **AND** dno=5;

Query: Retrieve the Names of the employees whose salary is greater than the salary of all the employees in the department 3.

mysql> **SELECT * FROM** employee

-> **WHERE** salary>**ALL** (**Select** salary **From** employee **Where** dno=3);

EXPERIMENT 12:-

Query multiple tables using JOIN operation.

The concept of a **joined table** (or **joined relation**) was incorporated into SQL to permit users to specify a table resulting from a join operation in the **FROM clause** of a query. This construct may be easier to comprehend than mixing together all the select and join conditions in the WHERE clause.

Query: Retrieves the name and address of every employee who works for the 'Research' department using join command.

```
mysql> Select Fname, Lname, Address
    > From (EMPLOYEE join DEPARTMENT on Dno=Dnumber)
    > Where Dname='Research';
```

NATURAL JOIN on two tables R and S, no join condition is specified; an implicit EQUIJOIN condition for each pair of columns with the same name from R and S is created. Each such pair of columns is included only once in the resulting relation.

Query: Retrieves the name and address of every employee who works for the 'Research' department using natural join command.

```
mysql> Select Fname, Lname, Address
    > From (EMPLOYEE natural join DEPARTMENT)
    > Where Dname='Research';
```

MULTIWAY JOIN: This allows the specification of the join of three or more tables as a single joined table.

Query: For every project located in "Ankola", list the project number, the controlling department number and the department manager last name, address and birth date.

```
mysql> Select Pnumber, Dnum, Lname, Address, Bdate
    > From ((PROJECT join DEPARTMENT on Dnum=Dnumber)
    >                    join EMPLOYEE on Mgr_ssn=Ssn)
    > Where Plocation='Ankola';
```

Query: Retrieve name, address, dname of employees whose dnumber between 10 and 13 in department.

```
mysql> SELECT FNAME,LNAME,DNAME,ADDRESS
    -> FROM EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER
    -> WHERE DNUMBER BETWEEN 10 AND 13;
```

EXPERIMENT 13:-

Write nested and complex queries using multiple tables.

Some queries require that existing values in the database be fetched and then used in a comparison condition. Such queries can be conveniently formulated by using **nested queries**, which are complete select-from-where blocks within the WHERE clause of another query. That other query is called the **outer query**.

The comparison operator **IN**, which compares a value *v* with a set (or multiset) of values *V* and evaluates to **TRUE** if *v* is one of the elements in *V*.

In addition to the IN operator, a number of other comparison operators can be used to compare a single value *v* (typically an attribute name) to a set or multiset *v* (typically a nested query). The = ANY (or = SOME) operator returns TRUE if the value *v* is equal to *some value* in the set *V* and is hence equivalent to IN. The two keywords ANY and SOME have the same effect. Other operators that can be combined with ANY (or SOME) include >, >=, <, <=, and <>. The keyword ALL can also be combined with each of these operators.

The **EXISTS** function in SQL is used to check whether the result of a correlated nested query is *empty* (contains no tuples) or not. The result of EXISTS is a Boolean value **TRUE** if the nested query result contains at least one tuple, or **FALSE** if the nested query result contains no tuples.

Query: Retrieve the project numbers of projects that have an employee with last name 'Smith'.

```
mysql> Select distinct Pnumber
       From PROJECT
       Where Pnumber IN ( Select Pno
                          From WORKS_ON, EMPLOYEE
                          Where Essn=Ssn AND Lname='Smith' );
```

Query: Retrieve the names of employees whose salary is greater than the salary of all the employees in department 5.

```
mysql>Select Lname, Fname
      From  EMPLOYEE
      Where Salary > ALL (Select Salary
                          From EMPLOYEE
                          Where Dno=5 );
```

Query: Retrieve the names of employees who have no dependents.

```
mysql>select Fname, Lname
      from EMPLOYEE
      where NOT EXISTS ( select *
                         from DEPENDENT
                         where Ssn=Essn );
```

Query: List the names of managers who have at least one dependent.

```
mysql>select Fname, Lname
      from EMPLOYEE
      where EXISTS ( select *
                     from DEPENDENT
                     where Ssn=Essn )
                      and
                EXISTS ( select *
                         from DEPARTMENT
                         where Ssn=Mgr_ssn );
```

EXPERIMENT 14:-

Perform UPDATE, ALTER, DELETE, DROP operations on tables.

UPDATE:-

The **UPDATE** statement is used to update existing records in a table. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated.

Query: Change the first name an employee to 'Sayyan' whose ssn is '162CS1601'.

```
mysql> Update employee
       Set fname='Sayyan'
       Where ssn='162CS1601';
```

Query: Change the location and controlling department number of project number 10 to 'Karwar' and 5.

```
mysql> Update project
       Set plocation='Karwar' and dnum=5
       Where pnumber=10;
```

Query: Raise salary 10% of all employees who work in the 'research' department.

```
mysql>Update employee
      Set salary=1.1*salary
      Where dno in (Select dnumber
                    From department
                    Where dname='research');
```

ALTER:-

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table. The definition of the base table can be changed by using the alter command. For base tables, the possible alter table actions includes adding a column, changing a column data type, renaming a table name etc...

Add: add keyword used to add new column to the existing table.

Syntax: **Alter table** tablename **Add** columnname datatype;

Ex: **alter table** EMPLOYEE **add** job varchar(30);

Delete: Drop keyword used to delete column to the existing table.

Syntax: **Alter table** tablename **Drop** columnname;

Ex: **alter table** EMPLOYEE **Drop** job ;

Modify: Modify keyword used to modify the existing table datatype.

Syntax: **Alter table** tablename **Modify** columnname datatype;

Ex: **alter table** EMPLOYEE **modify** address varchar(80);

Rename: Rename keyword used to rename the table name.

Syntax: **Alter table** tablename **Rename** to newtablename;

Ex: **alter table** EMPLOYEE **rename** to EMP;

DELETE:-

The **DELETE** statement is used to **delete rows** in a table. It includes a WHERE clause, similar to that used in an SQL query, to select the rows to be deleted. Rows are explicitly deleted from only one table at a time. A missing WHERE clause specifies that all rows in the relation are to be deleted.

Syntax: mysql> **Delete**
From tablename
Where condition;

Query: Delete employee information who's last name is 'Shaikh'.

mysql>**Delete**
From EMPLOYEE
Where lname='Shaikh';

Query: Delete employee information whose ssn is '162CS1601'.

mysql>**Delete**
From EMPLOYEE
Where ssn='162CS1601';

Query: Delete all employees' information who works in 'research' department.

```
mysql>Delete
        From EMPLOYEE
        Where dno in (Select dnumber
                      From department
                      Where dname='research');
```

Query: Delete all employee information from company.

```
mysql>Delete
        From EMPLOYEE;
```

DROP:

Indexes, tables, and databases can easily be deleted / removed with the **DROP** statement. The drop command can be used to drop a schema, view, table etc. That means it **deletes the table** from the database itself.

There are two drop Option.

- Cascade
- Restrict

If the restrict option is chosen, a table is dropped only if it is not referenced in any table.With the cascade option, all such referenced tables are dropped automatically from the schema along with the table itself.

Syntax: Drop Table tablename (option);

Ex: drop table EMPLOYEE;

EXPERIMENT 15:-

Illustrate the use of CREATE VIEW command and manipulating.

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Syntax:

```
CREATE VIEW view_name
AS SELECT column_name(s)
FROM table_name
WHERE condition;
```

A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view. If we do not need a view any more, we can use the **DROP VIEW** command to dispose of it.

Ex: Create view EMP
As select fname, lname
From employee;

Ex: Create view works_on1
As select fname, lname, pname, hours
From employee, project, works_on
Where ssn=essn and pno=pnumber;

Ex: Drop view works_on1;

EXPERIMENT 16:-

Use COMMIT and ROLL BACK commands.

ROLLBACK Command:

The **ROLLBACK** command is the transactional command used to undo transactions that have not already been saved to the database. The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Syntax: ROLLBACK;

COMMIT Command:

This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone.

Syntax: COMMIT;

* Note: Before using commit or rollback command one have to disable the so-called autocommit mode, which is set by default and commit every single SQL statement.

Disable autocommit by setting it to false (0)

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> create table book(book_id  varchar(10),
    -> title varchar(10),
    -> pub_name varchar(10),
    -> primary key(book_id));

Query OK, 0 rows affected (0.20 sec)
```

Insert the values to table book

```
mysql> insert into book values('101','dbms','ebpb');
        Query OK, 1 row affected (0.00 sec)

mysql> insert into book values('102','co','sapna');
        Query OK, 1 row affected (0.00 sec)

mysql> insert into book values('103','cn','ebpb');
        Query OK, 1 row affected (0.00 sec)
```

mysql> select * from book;

```
        +----------+-------+----------------+
        | book_id| title  | pub_name |
        +----------+-------+----------------+
        | 101      | dbms |     ebpb      |
        | 102      |  co    |     sapna      |
        | 103      |   cn    |     ebpb      |
        +---------+---------+----------------+
       3 rows in set (0.00 sec)
```

Set a commit point

mysql> commit;

Query OK, 0 rows affected (0.05 sec)

Update/ change the book table values

mysql> update book set pub_name='arya' where book_id=103;

```
        Query OK, 1 row affected (0.00 sec)
        Rows matched: 1  Changed: 1  Warnings: 0
```

mysql> select * from book;

```
        +---------+---------+----------------+
        | book_id | title |  pub_name|
        +---------+---------+----------------+
        | 101     | dbms |      ebpb     |
        | 102     | co      |      sapna   |
        | 103     | cn      |       arya    |
        +---------+---------+----------------+
        3 rows in set (0.00 sec)
```

Execute Rollback (Undone the changes)

mysql> rollback;

Query OK, 0 rows affected (0.05 sec

As a result, Update operation would not impact the table and SELECT statement would produce the following result:

```
mysql> select * from book;
+---------+-------+----------+
| book_id | title | pub_name |
+---------+-------+----------+
| 101     | dbms  | ebpb     |
| 102     | co    | sapna    |
| 103     | cn    | ebpb     |
+---------+-------+----------+
3 rows in set (0.00 sec)
```

EXPERIMENT 17:-

Use SAVEPOINT commands.

SAVEPOINT Command:

This command is used to temporarily save a transaction so that one can rollback to that point whenever necessary.

or

A **SAVEPOINT** is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

This command serves only in the creation of a **SAVEPOINT** among transactional statements. The **ROLLBACK** command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as follows:

ROLLBACK TO SAVEPOINT_NAME;

RELEASE SAVEPOINT Command:

The **RELEASE SAVEPOINT** command is used to remove a SAVEPOINT that you have created.

Syntax:

RELEASE SAVEPOINT SAVEPOINT_NAME;

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the SAVEPOINT.

mysql> set autocommit=0;

Query OK, 0 rows affected (0.00 sec)

mysql> select *from book;

```
+-----------+-------+----------------+
| book_id   | title |    pub_name    |
+-----------+-------+----------------+
|    101    | dbm   |      ebpb      |
|    102    | co    |       sapna    |
|    103    | cn    |       ebpb     |
+-----------+-------+----------------+
```

3 rows in set (0.02 sec)

mysql> update book set title='Cprog' where book_id=101;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from book;

```
+---------+-------+----------+
| book_id | title | pub_name |
+---------+-------+----------+
| 101     | Cprog | ebpb     |
| 102     | co    | sapna    |
| 103     | cn    | ebpb     |
+---------+-------+----------+
```

3 rows in set (0.00 sec)

mysql> savepoint one;

Query OK, 0 rows affected (0.00 sec)

mysql> insert into book values('104','Web',’sapna’);

Query OK, 1 row affected (0.00 sec)

mysql> select * from book;

```
+---------+-------+----------+
| book_id | title | pub_name |
+---------+-------+----------+
| 101     | Cprog | ebpb     |
| 102     | co    | sapna    |
| 103     | cn    | ebpb     |
| 104     | Web   | sapna    |
+---------+-------+----------+
```

4 rows in set (0.00 sec)

mysql> savepoint two;

Query OK, 0 rows affected (0.00 sec)

mysql> update book set pub_name='prakash

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select *from book;

```
+---------+-------+----------+
| book_id | title | pub_name |
+---------+-------+----------+
| 101     | Cprog | ebpb     |
| 102     | co    | sapna    |
| 103     | cn    | ebpb     |
| 104     | Web   | prakash  |
+---------+-------+----------+
```

4 rows in set (0.00 sec)

mysql> rollback to savepoint two;

Query OK, 0 rows affected (0.00 sec)

mysql> select *from book;

```
+---------+-------+----------+
| book_id | title | pub_name |
+---------+-------+----------+
| 101     | Cprog | ebpb     |
| 102     | co    | sapna    |
| 103     | cn    | ebpb     |
| 104     | Web   | sapna    |
+---------+-------+----------+
```

4 rows in set (0.00 sec)

mysql>commit;

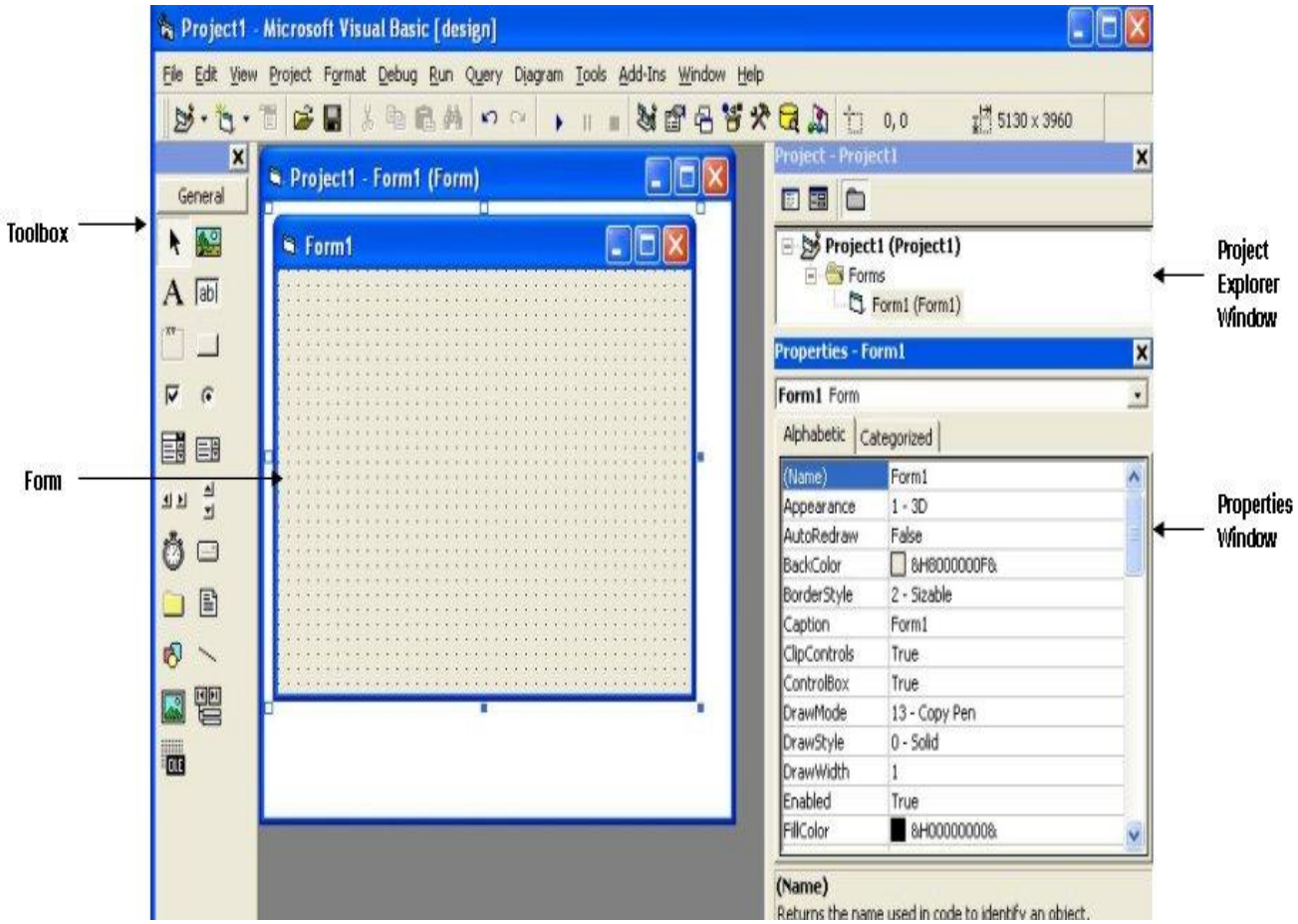
Query OK, 0 rows affected (0.00 sec)

GUI (GRAPHICAL USER INTERFACE)

What is Visual Basic?

VISUAL BASIC is a high level programming language which evolved from the earlier DOS version called BASIC. BASIC means Beginners' All-purpose Symbolic Instruction Code. It is a relatively easy programming language to learn. VISUAL BASIC is a VISUAL and **Event-driven Programming Language**. Some of the events are load, click, double click, drag and drop, pressing the keys and more. The code looks a lot like English Language. people prefer to use Microsoft Visual Basic today, as it is a well developed programming language and supporting resources are available everywhere.

In VB, programming is done in a graphical environment. In VB, you just need to drag and drop any graphical object anywhere on the form, and you can change its color any time using the properties window.



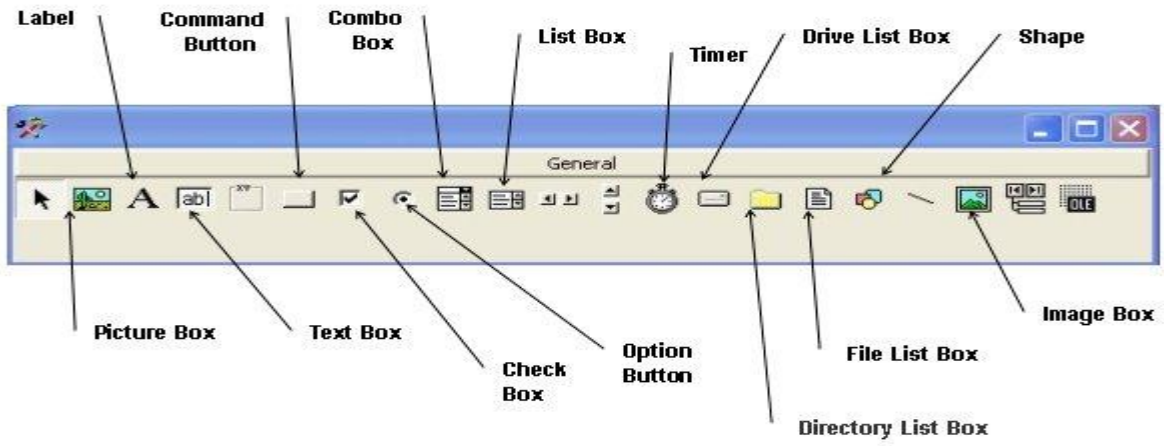
Every object in Visual Basic 6, such as a form, a command button, a text box and more has a set of properties that describe them. Setting the objects' properties is the first step in building a Visual Basic application, i.e. designing the interface.

The next step in building a Visual Basic application is to write code to response to the events. Events usually comprise actions triggered by the user, such as clicking the mouse buttons, pressing a key on the keyboard, dragging an object and more.

Event	Description
Click	The user clicks the mouse button on an object.
DblClick	The user double-clicks the mouse button on an object.
DragDrop	The user drags an object to another location.
GotFocus	An object receives focus.
KeyDown	The user presses a keyboard key.
KeyPress	The user presses and releases a keyboard key .
KeyUp	The user releases a keyboard key while an object has focus.
LostFocus	An object loses focus.
MouseDown	The user presses any mouse button.
MouseUp	The user releases any mouse button.

Common controls

Below figure is the VB6 toolbox that shows the basic controls.



The Text Box: Text box is the standard control for accepting input from the user as well as to display the output.

The Label: Label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs.

The Command Button: Command button is one of the most important controls as it is used to execute commands. It displays an illusion that the button is pressed when the user click on it.

The List Box: List Box is to present a list of items where the user can click and select the items from the list.

The Combo Box: Combo Box is also to present a list of items where the user can click and select the items from the list.

The Check Box: Check Box control lets the user selects or unselects an option. When the Check Box is checked, its value is set to 1 and when it is unchecked, the value is set to 0.

The Option Box: Option Box control also lets the user selects one of the choices. However, two or more Option Boxes must work together because as one of the Option Boxes is selected, the other Option Boxes will be unselected

The Image Box: Image Box is another control that handles images and pictures.

Declaring Variables:

In Visual Basic, it is a good practice to declare the variables before using them by assigning names and data types. They are normally declared in the general section of the codes' windows using the **Dim** statement.

Dim VariableName As DataType
Dim yourName As String
Dim firstnum As Integer

Visual Basic Data Types:

Visual Basic classifies the information mentioned above into two major data types, they are the **numeric data types** and the **non-numeric data types**.

1. Numeric Data Types: Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and more. Examples of numeric data types are examination marks, height, weight, the number of students in a class, share values, price of goods, monthly bills, fees and others.

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

2. Non-numeric Data Types: Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

Operators in Visual Basic

To compute inputs from users and to generate results, we need to use various mathematical operators. In Visual Basic, except for + and -, the symbols for the operators are different from normal mathematical operators.

Operator	Mathematical function	Example
^	Exponential	2^4=16
*	Multiplication	4*3=12, (5*6))2=60
/	Division	12/4=3
Mod	Modulus(return the remainder from an integer division)	15 Mod 4=3 255 mod 10=5
\	Integer Division(discards the decimal places)	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

Logical Operators

In addition to conditional operators, there are a few logical operators which offer added power to the VB programs.

Operator	Meaning
And	Both sides must be true
or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates truth

Conditional Operators

To control the VB program flow, we can use various conditional operators. Basically, they resemble mathematical operators.

Operator	Meaning
=	Equal to
>	More than
<	Less Than
>=	More than and equal
<=	Less than and equal
<>	Not Equal to

Control Structures

If...Then

The If...Then selection structure performs an indicated action only when the condition is True; otherwise the action is skipped.

Syntax:

```
If <condition> Then
   statement
End If
```

e.g.:
```
If average>75 Then
    Grade= "A"
    End If
```

If...Then...Else

The If...Then...Else selection structure allows the programmer to specify that a different action is to be performed when the condition is True than when the condition is False.

Syntax:

```
If <condition > Then
statements
Else
statements
End If
```

e.g.:
```
If average>50 Then
Grade = "Pass"
Else
Grade = "Fail"
End If
```

Nested If...Then...Else

Nested If...Then...Else selection structures test for multiple cases by placing If...Then...Else selection structures inside If...Then...Else structures.

Syntax:

```
If < condition 1 > Then
statements
ElseIf < condition 2 > Then
statements
ElseIf < condition 3 > Then
statements
Else
Statements
End If
```

Select...Case

Select...Case structure is an alternative to If...Then...ElseIf for selectively executing a single block of statements from among multiple blocks of statements. Select...case is more convenient to use than the If...Else...End If.

Syntax:

```
Select Case Index
Case 0
Statements
Case 1
Statements
Case else
Statements
End Select
```

Do While

The **Do While...Loop** is used to execute statements until a certain condition is met. The following Do Loop counts from 1 to 100.

```
Dim number As Integer
number = 1
Do While number <= 100
number = number + 1
Loop
```

While... Wend

A **While...Wend** statement behaves like the **Do While...Loop** statement. The following **While...Wend** counts from 1 to 100

```
Dim number As Integer
number = 1
```

```
While number <=100
number = number + 1
Wend
```

Do...Loop While

The **Do...Loop While** statement first executes the statements and then tests the condition after each execution. The following program block illustrates the structure:

```
Dim number As Long
number = 0
Do
number = number + 1
Loop While number < 501
```

The program executes the statements between Do and Loop While structure in any case. Then it determines whether the counter is less than 501. If so, the program again executes the statements between Do and Loop While else exits the Loop.

Do Until...Loop

Unlike the **Do While...Loop** and **While...Wend** repetition structures, the **Do Until... Loop** structure tests a condition for falsity. Statements in the body of a **Do Until...Loop** are executed repeatedly as long as the loop-continuation test evaluates to False.

```
Dim number As Long
number=0
Do Until number > 1000
number = number + 1
Print number
Loop
```

Numbers between 1 to 1000 will be displayed.

For...Next Loop

The **For...Next** Loop is another way to make loops in Visual Basic. **For...Next** repetition structure handles all the details of counter-controlled repetition. The following loop counts the numbers from 1 to 100.

```
Dim x As Integer
For x = 1 To 100
Print x
Next
```

Design an application which performs the following operations on the data base the form using ADO 1.ADD 2.UPDATE 3.DELETE

Source code:

```
Private Sub cmdadd_Click()
Data1.Recordset.AddNew
cmdadd.Visible = True
cmdupdate.Visible = True
End Sub

Private Sub cmddelete_Click()
If Data1.Recordset.RecordCount > 0 Then
If MsgBox("Do you want to delete? press Yes or No", vbYesNo + vbInformation) =
vbYes Then
Data1.Recordset.Delete
Data1.Refresh
End If
End If
End Sub

Private Sub cmdupdate_Click()
Data1.Recordset.Update
cmdadd.Visible = True
cmdupdate.Visible = True
End Sub

Private Sub cmdexit_Click()
End
End Sub
```

PROCEDURE:-

Step 1: Right click on the toolbar components select the Microsoft ADO data control 6.0 (OLEDB) & Data Grid Controller and apply ok.

Step 2: Drag the ADODC tool and Data Grid on the form.

Step 3: Right click on ADODC and Go to Properties, a dialog box will appear Click on build option and select Microsoft access version 7.0 MDB.

Step 4: Give the Database path to source and click Test Connection button. If connection is successful then apply changes click OK.

Step 5: Now In resource tab choose the Respective table and click OK

Step 6: Design a proper front end using labels, buttons and text box. Set name and captions.

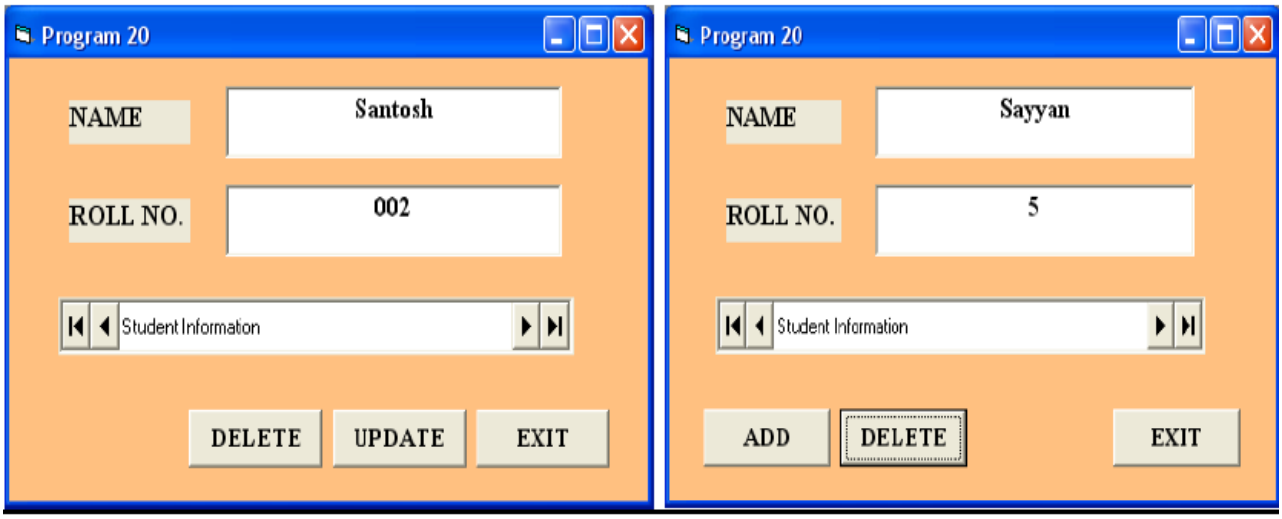
Step 7: For each text box, data grid controller**,** set Record source and Record Field from property window**.**

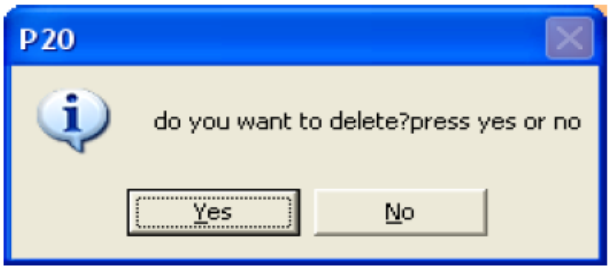
Step 8: Design 4 command buttons for add, update, delete, and exit in front end and set name Property for each.

Step 11: Then write the code for each button option.

Step 12: To run the Program press F5 or Run option from menu bar

OUTPUT:-


Program 20
NAME
Santosh
ROLL NO.
002
Student Information
DELETE
UPDATE
EXIT
Program 20
NAME
Sayyan
ROLL NO.
5
Student Information
ADD
DELETE
EXIT


P20
do you want to delete?press yes or no
Yes
No