# Crack SQL Basics – Notes with Real Questions

# I Practiced

*This guide includes everything I learned and practiced during my SQL journey.*

***Prepared by***

***Aliya Jabbar***

# INDEX

# CREATING DATABASE TABLES AND USING DATA TYPES

**Aim:**

- To create, show, use and drop a database
- To create a DDL to perform creation of table, altering of table and dropping a table.

## CREATE, SHOW, USE, AND DROP DATABASE

**Create the database**

Syntax

```
MYSQL>CREATE DATABASE <DATABASENAME>;
```

Eg:-

```
MYSQL>CREATE DATABASE SAS;
```

**Display a list of available databases.**

To list all of the databases you have stored

Syntax

```
MYSQL>SHOW DATABASES;
```

**Select your database.**

Once the database has been created, you will need to select it in order to begin editing it.

Syntax

```
MYSQL>USE <DATABASENAME>;
```

Eg:-

```
MYSQL>USE SAS;
```

**Display all the tables in a databases.**

Syntax

```
MYSQL>SHOW TABLES;
```

**Drop a Database**

Syntax

```
MYSQL>DROP DATABASE <DATABASENAME>;
```

Eg:-

```
MYSQL>DROP DATABASE SAS;
```

## DDL (DATA DEFINITION LANGUAGE) COMMANDS

- **CREATE** - is used to create objects in the database.
- **ALTER** - is used to alter the structure of database
- **DROP** - is used to delete the object from the database.

- **TRUNCATE** - is used to remove all the records from the table
- **RENAME** - is used to rename the objects.

**MySQL Data Types**

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

**Numeric Data Types:**

- **INT** - A normal-sized integer that can be signed or unsigned.
- **TINYINT** - A very small integer that can be signed or unsigned.
- **SMALLINT** - A small integer that can be signed or unsigned.
- **MEDIUMINT** - A medium-sized integer that can be signed or unsigned.
- **BIGINT** - A large integer that can be signed or unsigned.
- **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.
- **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. REAL is a synonym for 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 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.
- **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 Types:**

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.
- **BLOB or TEXT** - A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files
- **TINYBLOB or TINYTEXT** - A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** - A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** - A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.

## CREATE TABLE

**Syntax for creating a table:**

```
CREATE TABLE <TABLE NAME>
(
      <COLUMN-NAME1> <DATATYPE> (<SIZE>),
      <COLUMN-NAME2> <DATATYPE> (<SIZE>),
      ..........................................................................................
);
```

Q1. Write a query to create a table employee with empno, ename, designation, and salary.

```
MYSQL>CREATE TABLE EMP
      (
            EMPNO INT(4),
            ENAME VARCHAR (10),
            DESIGNATIN VARCHAR (10),
            SALARY DECIMAL (8, 2)
      );
```

**Syntax for describe the table:**

```
SQL>DESC <TABLE NAME>;
```

Q2. Write a query to display the column name and data type of the table employee.

```
MYSQL> DESC EMP;

      Name                          Null?        Type
 ------------------------ ----------- ------------
 EMPNO                                     INT(4)
 ENAME                                     VARCHAR(10)
 DESIGNATIN                                VARCHAR(10)
 SALARY                                    DECIMAL (8,2)
```

**Syntax For Create A table  from an Existing Table with All Fields**

```
MYSQL> CREATE TABLE <TRAGET TABLE NAME> SELECT * FROM < SOURCE
TABLE NAME>;
```

Q3. Write a query for create a from an existing table with all the fields

```
MYSQL> CREATE TABLE EMP1 AS SELECT * FROM EMP;
MYSQL> DESC EMP;

      Name                           Null?        Type
 ------------------------ ----------- ------------
 EMPNO                                  INT(4)
 ENAME                                  VARCHAR(10)
 DESIGNATIN                             VARCHAR(10)
 SALARY                                 DECIMAL (8,2)
```

**Syntax for Create a table from an Existing Table with Selected Fields**

```
MYSQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT <COLUMN
NAME1>,<COLUMN NAME2>,… FROM <SOURCE TABLE NAME>
```

Q4. Write a query for create a table from an existing table with selected fields

```
MYSQL> CREATE TABLE EMP2 AS SELECT EMPNO, ENAME FROM EMP;

 MYSQL> DESC EMP2
 Name                     Null?    Type
 ---------------------- -------- --------------
 -EMPNO                                 INT(4)

 ENAME                            VARCHAR (10)
```

**Syntax for create a new table from an existing table without any record:**

```
MYSQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT * FROM
     <SOURCE TABLE NAME> WHERE <FALSE CONDITION>;
```

Q5. Write a query for create a new table from an existing table without any record:

```
      MYSQL> CREATE TABLE EMP3 AS SELECT * FROM EMP WHERE
1>2;

      MYSQL> DESC EMP3;

            Name                           Null?        Type
       ------------------------ ----------- ------------
       EMPNO                                  INT (4)
       ENAME                                  VARCHAR(10)
       DESIGNATIN                             VARCHAR(10)
       SALARY                                 DECIMAL(8,2)
```

## ALTER & MODIFICATION ON TABLE

**Syntax for Alter & Modify on a Single Column:**

```
MYSQL > ALTER TABLE <TABLE NAME>
```

```
        MODIFY <COLUMN NAME> <DATATYPE>(SIZE);
```

Q6. Write a Query to Alter the column EMPNO INTEGER (4) TO EMPNO INTEGER (6).

```
MYSQL>ALTER TABLE EMP MODIFY EMPNO INT(6);
MYSQL> DESC EMP;
     Name                           Null?         Type
------------------------ ----------- ------------
EMPNO                                       INT(6)
ENAME                                       VARCHAR(10)
DESIGNATIN                                  VARCHAR(10)
SALARY                                      DECIMAL(8,2)
```

**Syntax for alter table with multiple column:**

```
MYSQL > ALTER TABLE <TABLE NAME>
          MODIFY <COLUMN NAME1> <DATATYPE> (SIZE),
          MODIFY <COLUMN NAME2> <DATATYPE> (SIZE),
          ………………………………………………………………………………;
```

Q7. Write a Query to Alter the table employee with multiple columns (EMPNO, ENAME.)

```
MYSQL>ALTER TABLE EMP MODIFY EMPNO INT (7), MODIFY ENAME VARCHAR (60);
MYSQL> DESC EMP;
     Name                           Null?         Type
------------------------ ----------- ------------
EMPNO                                       INT(7)
ENAME                                       VARCHAR (60)
DESIGNATIN                                  VARCHAR(10)
SALARY                                      DECIMAL(8,2)
```

**Syntax for add a new column:**

```
MYSQL> ALTER TABLE <TABLE NAME>

ADD <COLUMN NAME> <DATA TYPE> <SIZE>;
```

Q8. Write a query to add a new column in to employee

```
MYSQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR(6);
Table altered.
MYSQL> DESC EMP;

     Name                           Null?         Type
------------------------ ----------- ------------
EMPNO                                       INT(7)
ENAME                                       VARCHAR(12)
DESIGNATIN                                  VARCHAR(10)
SALARY                                      DECIMAL(8,2)
     QUALIFICATION                               VARCHAR(6)
```

**Syntax for add multiple columns:**

```
MYSQL> ALTER TABLE <TABLE NAME> ADD
     (
          <COLUMN NAME1> <DATA TYPE> <SIZE>,
          <COLUMN NAME2> <DATA TYPE> <SIZE>,
          ………………………………………………………………………………………
     );
```

Q9. Write a query to add multiple columns in to employee

```
MYSQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);
MYSQL> DESC EMP;

     Name                           Null?        Type
------------------------ ----------- ------------
EMPNO                                      INT(6)
ENAME                                      VARCHAR (10)
DESIGNATIN                                 VARCHAR(10)
SALARY                                     DECIMAL(8,2)
     QUALIFICATION                              VARCHAR(6)
     DOB                                        DATE
DOJ                                        DATE
```

**Syntax for remove a column:**

```
MYSQL> ALTER TABLE <TABLE NAME>

     DROP <COLUMN NAME>;
```

Q10. Write a query to drop a column from an existing table employee

```
MYSQL> ALTER TABLE EMP DROP DOJ;
MYSQL> DESC EMP;
     Name                           Null?        Type
------------------------ ----------- ------------
EMPNO                                      INT(6)
ENAME                                      VARCHAR (10)
DESIGNATIN                                 VARCHAR(10)
SALARY                                     DECIMAL(8,2)
     QUALIFICATION                              VARCHAR(6)
     DOB                                        DATE
```

**Syntax for remove multiple columns:**

```
MYSQL> ALTER TABLE <TABLE NAME>
          DROP <COLUMN-NAME1>,
          DROP <COLUMN-NAME2>,
          ……………………………;
```

Q11. Write a query to drop multiple columns from employee

```
MYSQL> ALTER TABLE EMP DROP DOB, DROP QUALIFICATION;
MYSQL> DESC EMP;
```

```
     Name                           Null?        Type
------------------------ ----------- ------------
EMPNO                                      INT(6)
ENAME                                      VARCHAR (10)
DESIGNATIN                                 VARCHAR(10)
SALARY                                     DECIMAL(8,2)
```

**Syntax for rename a column:**

```
MYSQL> ALTER TABLE <TABLE NAME>
     CHANGE <OLD-COLUMN-NAME> <NEW-COLUMN-NAME> <DATATYPE>;
```

Q12. Write a query to drop a column from an existing table employee

```
MYSQL> ALTER TABLE EMP CHANGE EMPNO ENO INT(10);
```

## DROP TABLE

**Syntax for deleting a Table:**

```
MYSQL>DROP TABLE <TABLE NAME>;
```

Q13. Write a query to delete the table emp2

```
SQL> DROP TABLE EMP2;
```

## RENAME TABLE

**Syntax for rename a Table:**

```
MYSQL> ALTER TABLE <TABLE-NAME> RENAME TO <NEW NAME>
```

Q14. Write a query to rename table emp to employee

```
SQL> ALTER TABLE EMP RENAME TO EMPLOYEE;
SQL> DESC EMPLOYEE;
     Name                           Null?        Type
------------------------ ----------- ------------
ENO                                        INT(6)
ENAME                                      VARCHAR(10)
DESIGNATIN                                 VARCHAR(10)
SALARY                                     DECIMAL(8,2)
```

## TRUNCATE TABLE

The truncate table statement

- Removes all rows from a table
- Release the storage space used by that table

**Syntax for complete a table:**

```
SQL> TRUNCATE TABLE <TABLENAME>;
```

Q15. Write a query to truncate an existing table employee

```
SQL> TRUNCATE TABLE EMPLOYEE;
```

# DATA MANIPULATION

**Aim**

To execute and verify the INSERT, UPDATE and DELETE commands

## DML (DATA MANIPULATION LANGUAGE)

- **INSERT** - is used to insert objects in the database.
- **UPDATE** - is used to update the records from the table
- **DELETE** - is used to delete the records form the table

### INSERT DATA IN TO TABLE

Syntax for Insert Records in to a table:

```
mysql > INSERT INTO <TABLE NAME> VALUES('VAL1','VAL2',...);
```

Q. Write a query to insert the records in to employee.

```
mysql > INSERT INTO EMP VALUES(101,'NAGARAJAN','LECTURER',15000);
mysql > SELECT * FROM EMP;
```

Syntax for Insert some Field of a Records in to a table:

```
mysql > INSERT INTO <TABLE NAME>(<COLUMN1>, <COLUMN2>, …) VALUES ('VAL1','VAL2',...);
```

Q. Write a query to insert the records in to employee.

```
mysql > INSERT INTO EMP(EMPNO,ENAME) VALUES(102,'RAHUL');
mysql > SELECT * FROM EMP;
```

### UPDATE DATA FROM TABLE

Syntax for Update Records from the table:

```
mysql > UPDATE <TABLE NAME>
        SET <COLUMNNANE>=<VALUE>
        WHERE <COLUMNNAME=<VALUE>;
```

Q. Write a query to update the records from employee.

```
mysql > UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;
mysql > SELECT * FROM EMP;
```

Syntax for update multiple Records from the table:

```
mysql > UPDATE <TABLE NAME>
     SET <COLUMNANE1> = <VALUE1>, <COLUMNANE2> = <VALUE2>,...
     WHERE <COLUMNNAME> = <VALUE>;
```

Q. Write a query to update multiple records from employee.

```
mysql>UPDATE EMP SET SALARY = 17000, DESIGNATIN='ASST. PROF'
WHERE EMPNO=101;
```

**DELETE DATA FROM TABLE**

Syntax for delete Records from the table:

```
mysql > DELETE FROM <TABLE NAME> WHERE <COLUMN NAME>=<VALUE>;
```

Q. Write a query to delete records from employee.

```
mysql >DELETE FROM EMP WHERE EMPNO=101;

mysql > SELECT * FROM EMP;
```

Syntax for delete All Records from the table:

```
mysql> DELETE FROM <TABLE NAME>

     WHERE <condition>;
```

Q. Write a query to delete all records from employee.

```
mysql >DELETE FROM EMP;

mysql > SELECT * FROM EMP;
```

# IMPLEMENTING THE CONSTRAINTS

**Aim**

To set relations among tables and verify different types of constraints.

MySQL constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint. Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

In MySQL, we have the following constraints:

- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **CHECK** - is used to limit the value range that can be placed in a column.

**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 for Column level constraints Using Primary key:**

```
CREATE TABLE <TABLE NAME>
(
      <COLUMN-NAME1> <DATATYPE> (<SIZE>) PRIMARY KEY,
      <COLUMN-NAME2> <DATATYPE> (<SIZE>),
      ........................................................................................
);
```

Q. Create a table BOOK with attributes (BOOKID, TITLE, AUTHOR, PUBLISHER, PRICE) and set BOOKID as PRIMARY KEY at column level.

```
CREATE TABLE BOOK
(
      BOOKID INT(7) PRIMARY KEY,
      TITLE VARCHAR(30),
      AUTHOR VARCHAR(30),
      PUBLISHER VARCHAR(30),
      PRICE DECIMAL(7,2)
);
```

**Table level constraints Using Primary key**

```
 CREATE TABLE <TABLE NAME>
 (
      <COLUMN-NAME1> <DATATYPE> (<SIZE>),
     <COLUMN-NAME2> <DATATYPE> (<SIZE>),
     ..........................................................................,
      PRIMARY KEY(<COLUMN-NAME>)
 );
```

**Q.** Create a table BOOK1 with attributes (BOOKID, TITLE, AUTHOR, PUBLISHER, PRICE) and set BOOKID as PRIMARY KEY at table level.

```
      CREATE TABLE BOOK1
      (
           BOOKID INT(7),
           TITLE VARCHAR(30),
           AUTHOR VARCHAR(30),
           PUBLISHER VARCHAR(30),
           PRICE DECIMAL(7,2),
           PRIMARY KEY(BOOKID)
      );
```

**Table level constraints Using Primary key with naming convention**

```
 CREATE TABLE <TABLE NAME>
 (
      <COLUMN-NAME1> <DATATYPE> (<SIZE>),
     <COLUMN-NAME2> <DATATYPE> (<SIZE>),
     ..........................................................................,
      CONSTRAINT <NAME-OF-PK-CONSTRAINT> PRIMARY KEY(<COLUMN-NAME>)
 );
```

**Q.** Create a table BOOK2 with attributes (BOOKID, TITLE, AUTHOR, PUBLISHER, PRICE) and set BOOKID as PRIMARY KEY with name BOOK_PK at table level.

```
      CREATE TABLE BOOK2
      (
           BOOKID INT(7),
           TITLE VARCHAR(30),
           AUTHOR VARCHAR(30),
           PUBLISHER VARCHAR(30),
           PRICE DECIMAL(7,2),
           CONSTRAINT BOOK_PK PRIMARY KEY(BOOKID)
      );
```

**Defining a PRIMARY KEY constraint on multiple columns**

```
 CREATE TABLE <TABLE NAME>
 (
      <COLUMN-NAME1> <DATATYPE> (<SIZE>),
     <COLUMN-NAME2> <DATATYPE> (<SIZE>),
     ......................................................................................,
       PRIMARY KEY( <COLUMN-NAME1>, <COLUMN-NAME2>, ...)
 );
```

**Q.** Create a table BOOK3 with attributes (BOOKID, TITLE, AUTHOR, PUBLISHER, PRICE) and set BOOKID and TITLE as PRIMARY KEY at table level.

```
        CREATE TABLE BOOK3
        (
                BOOKID INT(7),
                TITLE VARCHAR(30),
                AUTHOR VARCHAR(30),
                PUBLISHER VARCHAR(30),
                PRICE DECIMAL(7,2),
                PRIMARY KEY(BOOKID, TITLE)
        );
```

**Defining a PRIMARY KEY constraint on multiple columns with naming convention**

```
  CREATE TABLE <TABLE NAME>
  (
    <COLUMN-NAME1> <DATATYPE> (<SIZE>),
    <COLUMN-NAME2> <DATATYPE> (<SIZE>),
    ......................................................................................,
    CONSTRAINT <NAME-OF-PK-CONSTRAINT> PRIMARY KEY( <COLUMN-NAME1>, <COLUMN-NAME2>,
    ...)
  );
```

**Q.** Create a table BOOK4 with attributes (BOOKID, TITLE, AUTHOR, PUBLISHER, PRICE) and set BOOKID and TITLE as PRIMARY KEY with name BOOK_PK at table level.

```
        CREATE TABLE BOOK4
        (
                BOOKID INT(7),
                TITLE VARCHAR(30),
                AUTHOR VARCHAR(30),
                PUBLISHER VARCHAR(30),
                PRICE DECIMAL(7,2),
                CONSTRAINT BOOK_PK PRIMARY KEY(BOOKID, TITLE)
        );
```

**PRIMARY KEY Constraint on ALTER TABLE**

```
        MYSQL>ALTER TABLE <TABLE NAME> ADD PRIMARY KEY (<COLUMN NAMES>);
```

**Q.** Write a query to update EMPNO as primary key in the existing table EMP.

```
        MYSQL>ALTER TABLE EMP ADD PRIMARY KEY (EMPNO);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following syntax:

```
MYSQL>ALTER TABLE <TABLE NAME>

ADD CONSTRAINT <PK-NAME> PRIMARY KEY (<COLUMN NAME1, COLUMN NAME2, … >);
```

**Q.** Write a query to update EMPID and DEPTID as primary key with name PK_EMPLOYEE in the existing table EMPLOYEE.

```
MYSQL>ALTER TABLE EMPLOYEE
      ADD CONSTRAINT PK_EMPLOYEE PRIMARY KEY (EMPID, DEPTID);
```

**DROP a PRIMARY KEY Constraint**

```
MYSQL>ALTER TABLE <TABLE NAME>
      DROP PRIMARY KEY;
```

**Q.** Write a query to drop the primary key in the table BOOK4.

```
MYSQL>ALTER TABLE BOOK4
      DROP PRIMARY KEY;
```

**FOREIGN KEY Constraint**

A FOREIGN KEY in one table points to a PRIMARY KEY in another table. The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

**FOREIGN KEY Constraint on CREATE TABLE**

This session executed and verified based on the following parent table

PARENT TABLE

```
CREATE TABLE COURSE
(
      COURSEID INT(7) PRIMARY KEY,
      CNAME VARCHAR(30)
);
```

**Syntax for Column level constraints Using Foreign key:**

```
CREATE TABLE <TABLE NAME>
(
      <COLUMN-NAME1> <DATATYPE> (<SIZE>),
      <COLUMN-NAME2> <DATATYPE> (<SIZE>),
      ………………………………………………………………………………………,
      FOREIGN KEY(<FK-COLUMNS>) REFERENCES <PARENT-TABLE>(<PARENT-PK>)
);
```

Q. Write a query to create a table STUDENT with attributes(STUDID, SNAME, CID, TMARK) and set CID as foreign key which corresponds to COURSID in the table COURSE.

```
CREATE TABLE STUDENT
(
      STUDID INT(7) PRIMARY KEY,
      SNAME VARCHAR(30),
      CID INT(7),
      TMARK INT(4),
      FOREIGN KEY (CID) REFERENCES COURSE(COURSEID)
);
```

```
MYSQL>CREATE TABLE <TABLE NAME>
      (
          <COLUMN-NAME1> <DATATYPE> (<SIZE>),
          <COLUMN-NAME2> <DATATYPE> (<SIZE>),
          ..........................................................................................,
          CONSTRAINT   <FK-CONSTRAINT-NAME>   FOREIGN   KEY(<FK-COLUMNS>)
          REFERENCES <PARENT-TABLE>(<PARENT-PK>)
     );
```

Q. Write a query to create a table STUDENT1 with attributes(STUDID, SNAME, CID, TMARK) and set CID as foreign key with name STUD_FK which corresponds to COURSID in the table COURSE.

```
MYSQL>CREATE TABLE STUDENT1
     (
          STUDID INT(7) PRIMARY KEY,
          SNAME VARCHAR(30),
          CID INT(7),
          TMARK INT(4),
          CONSTRAINT STUD_FK FOREIGN KEY (CID) REFERENCES COURSE(COURSEID)
     );
```

**FOREIGN KEY Constraint on ALTER TABLE**

```
     MYSQL>ALTER TABLE <TABLE NAME>
          ADD FOREIGN KEY (<FK-COLUMNS>)
          REFERENCES <PARENT-TABLE>(<PARENT-PK>);
OR
     MYSQL>ALTER TABLE <TABLE NAME>
          ADD CONSTRAINT <FK-CONSTRAINT-NAME>
          FOREIGN KEY (<FK-COLUMNS>)
          REFERENCES <PARENT-TABLE>(<PARENT-PK>);
```

**Q.C**reate a table STUDENT2 with attributes(STUDID, SNAME, CID, TMARK) and the alter the table to set CID as foreign key which corresponds to COURSID in the table COURSE.

```
     MYSQL>CREATE TABLE STUDENT2
          (
               STUDID INT(7) PRIMARY KEY,
               SNAME VARCHAR(30),
               CID INT(7),
               TMARK INT(4)
          );

     MYSQL>ALTER TABLE STUDENT2
          ADD FOREIGN KEY (CID)
          REFERENCES COURSE(COURSEID);
```

**DROP a FOREIGN KEY Constraint**

```
     MYSQL>ALTER TABLE <TABLE NAME>
          DROP FOREIGN KEY <FK-CONSTRAINT-NAME>;
```

**Q.** Write a query to drop foreign key STUD_FK in the table STUDENT1.

```
     MYSQL>ALTER TABLE STUDENT1
```

```
      DROP FOREIGN KEY STUD_FK;
```

## 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.

The following query enforces the "P_ID" column and the "NAME" column to not accept NULL values:

```
MYSQL>CREATE TABLE PERSON
      (
          P_ID                                    INT NOT NULL,
          NAME                            VARCHAR(50) NOT NULL,
          ADDRESS                                  VARCHAR(50),
          CITY VARCHAR(50)
      );
```

## 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.

The following query creates a UNIQUE constraint on the "P_ID" column when the "PERSON1" table is created:

```
MYSQL>CREATE TABLE PERSON1
      (
          P_ID INT UNIQUE,
          NAME VARCHAR(50) NOT NULL,
          ADDRESS VARCHAR(50),
          CITY VARCHAR(50)
      );
```

**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 column it will allow 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.

**CHECK Constraint on CREATE TABLE**

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE PERSONS
(
ID INT NOT NULL,
LASTNAME VARCHAR(255) NOT NULL,
FIRSTNAME VARCHAR(255),
AGE INT,
CHECK (AGE>=18)
);
```

```
insert into persons values(101,'Raj','Akash',25);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
        CREATE TABLE Persons (

        ID int NOT NULL,

        LastName varchar(255) NOT NULL,

        FirstName varchar(255),

        Age int,

        City varchar(255),

        CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')

        );
```

### CHECK Constraint on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

```
        ALTER TABLE Persons

        ADD CHECK (Age>=18);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
        ALTER TABLE Persons

        ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
```

### DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

```
        ALTER TABLE Persons

        DROP CHECK CHK_PersonAge;
```

# RETRIEVING DATA USING BASIC SQL COMMANDS

## Aim

To retrieving data using basic SQL commands

## SELECT STATEMENT

SELECT command is used to select the object from the database.

**Selects all rows from the table**
Syntax:

```
MYSQL>SELECT * FROM <TABLENAME>;
```

Q. Write a query to display all the records from employee.

```
MYSQL>SELECT * FROM EMP;
```

**The retrieval of specific columns from a table:**
It retrieves the specified columns from the table
Syntax:

```
MYSQL>SELECT COLUMN1, …..,COLUMNN FROM <TABLENAME>;
```

Q. Write a query to display empno, ename of employee from employee.

```
MYSQL>SELECT EMPNO, ENAME FROM EMP;
```

**Elimination of duplicates from the select clause:**

It prevents retrieving the duplicated values .Distinct keyword is to be used.
Syntax:

```
MYSQL>SELECT DISTINCT COL1, COL2 FROM <TABLENAME>;
```

Q. Write a query to display distinct designation from employee.

```
MYSQL>SELECT DISTINCT DESIGNATATIN FROM EMP;
```

**Select command with where clause:**
To select specific rows from a table we include 'where' clause in the select command. It can appear only after the 'from' clause.
Syntax:

```
MYSQL>SELECT COLUMN_NAME1, …..,COLUMN_NAME-N
      FROM <TABLENAME>
      WHERE <CONDITION>;
```

Q. Write a query to display empno and ename of employees whose salary is > 40000.

```
MYSQL>SELECT EMPNO, ENAME FROM EMP WHERE SAL>40000;
```

**Select command with order by clause:**
Syntax:

```
MYSQL>SELECT <COLUMNNAME-1, ….., COLUMNNAME-N>
      FROM <TABLENAME>
      WHERE <CONDITION>
```

```
ORDER BY COLMNNAME1 <ASC|DESC>, COLUMNNAME2 <ASC|DESC>;
```

Q. Write a query to display empno and ename of employees in ascending order of ename.

```
MYSQL>SELECT EMPNO, ENAME FROM EMP ORDER BY ENAME ASC;
```

Q. Write a query to display empno and ename of employees in ascending order of empno.

```
MYSQL>SELECT EMPNO, ENAME FROM EMP ORDER BY EMPNO DESC;
```

Q. Write a query to display empno and ename of employees in ascending order of empno and descending order of ename.

```
MYSQL>SELECT EMPNO, ENAME FROM EMP ORDER BY EMPNO ASC, ENAME DESC;
```

**Select command to create a table:**
Syntax:

```
MYSQL>CREATE TABLE <TABLENAME> AS SELECT * FROM
<EXISTING_TABLENAME>;
```

Q. Write a query to create a table emp1 from table emp;

```
CREATE TABLE EMP1 AS SELECT * FROM EMP;
```

**Select command to insert records:**
Syntax:

```
MYSQL>INSERT INTO <TABLENAME>(SELECT COLUMNS FROM
<EXISTING_TABLENAME>);
```

Q. Write a query to insert data into table emp1 from table emp.

```
MYSQL>INSERT INTO EMP1 (SELECT * FROM EMP);
```

# AGGREGATE FUNCTIONS

**Aim**

To execute and verify aggregate functions.

## AGGREGATE FUNCTIONS

Aggregate functions return a single value, calculated from values in a column. Useful aggregate functions are:

- `AVG()` - Returns the average value
- `COUNT()` - Returns the number of rows
- `MAX()` - Returns the largest value

- `MIN()` - Returns the smallest value
- `SUM()` - Returns the sum

### The AVG() Function

The `AVG()` function returns the average value of a numeric column.
Syntax:

```
MYSQL>SELECT AVG(COLUMN_NAME) FROM TABLE_NAME
```

Q. Write a query to calculate average salary of all the employees.

```
MYSQL>SELECT AVG(SALARY) AS AVG_SAL FROM EMP;
```

### The COUNT() Function

The `COUNT()` function returns the number of rows that matches a specified criteria.

### COUNT(column_name)

The `COUNT(column_name)` function returns the number of values (NULL values will not be counted) of the specified column:
Syntax:

```
MYSQL>SELECT COUNT(COLUMN_NAME) FROM TABLE_NAME;
```

Q. Write a query to find total number of employees whose salary is >=15000.

```
MYSQL>SELECT COUNT(EMPNO) AS TOTAL_EMP FROM EMP WHERE SALARY>=15000;
```

### COUNT(*)

The `COUNT(*)` function returns the number of records in a table:
Syntax:

```
MYSQL>SELECT COUNT(*) FROM TABLE_NAME;
```

Q. Write a query to find total number of rows in emp table.

```
MYSQL>SELECT COUNT(*) AS NO_OF_ROWS FROM EMP;
```

### COUNT(DISTINCT column_name)

The `COUNT(DISTINCT column_name)` function returns the number of distinct values of the specified column:
Syntax:

```
MYSQL>SELECT COUNT(DISTINCT COLUMN_NAME) FROM TABLE_NAME;
```

Q. Write a query to find number of different designations of employees.

```
MYSQL>SELECT COUNT(DISTINCT DESIGNATION) FROM EMP;
```

### The **MAX()** Function

The MAX() function returns the largest value of the selected column.

Syntax:

```
MYSQL>SELECT MAX(COLUMN_NAME) FROM TABLE_NAME;
```

Q. Write a query to calculate maximum salary from all the employees.

```
MYSQL>SELECT MAX(SALARY) AS MAX_SAL FROM EMP;
```

### The **MIN()** Function

The MIN() function returns the smallest value of the selected column.

Syntax:

```
MYSQL>SELECT MIN(COLUMN_NAME) FROM TABLE_NAME;
```

Q. Write a query to calculate minimum salary from all the employees.

```
MYSQL>SELECT MIN(SALARY) AS MIN_SAL FROM EMP;
```

### The **SUM()** Function

The SUM() function returns the total sum of a numeric column.

Syntax:

```
MYSQL>SELECT SUM(COLUMN_NAME) FROM TABLE_NAME;
```

Q. Write a query to calculate total salary of all the employees

```
MYSQL>SELECT SUM(SALARY) AS TOTAL_SAL FROM EMP;
```

## GROUP BY AND HAVING CLAUSES

### Aim

To execute and verify GROUP BY and HAVING clauses.

### TABLE DESCRIPTION

This session is based on the following table description and data.

**TABLE NAME: EMPLOYEE**

| FIELD | TYPE |
|---|---|
| EMPID | INT(8) |
| ENAME | VARCHAR(25) |

| | |
|---|---|
| DEPTID | INT(5) |
| DESIGNATION | VARCHAR(25) |
| SALARY | DECIMAL(10,2) |
| DOJ | DATE |

| EMPID | ENAME | DEPTID | DESIGNATION | SALARY | DOJ |
|---|---|---|---|---|---|
| 10001 | VINOD | 301 | LECTURER | 56000.00 | 2000-03-01 |
| 10002 | RAHUL | 302 | LECTURER | 58000.00 | 2000-03-01 |
| 10003 | HARI | 303 | ASST PROF | 101000.00 | 1996-03-01 |
| 10004 | SAJIN | 302 | ASST PROF | 100800.00 | 1996-02-20 |
| 10005 | ANIL | 301 | ASSO PROF | 126000.00 | 1995-01-10 |
| 10006 | RAMESH | 302 | LECTURER | 54000.00 | 2001-07-10 |
| 10007 | BINU | 303 | LECTURER | 54500.00 | 2001-06-22 |

## The GROUP BY Statement

The MySQL GROUP BY clause is used in a  to collect data across multiple records and group the results by one or more columns. The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Syntax

```
SELECT EXPRESSION1,EXPRESSION2,...,EXPRESSION_N,
     AGGREGATE_FUNCTION(EXPRESSION)
FROM TABLES
[WHERE CONDITIONS]
GROUP BY EXPRESSION1, EXPRESSION2, ... EXPRESSION_M;
```

### Example - Using SUM function

**Q.** Write a query to display department ids and total salary in each departments.

```
     MYSQL>SELECT DEPTID, SUM(SALARY) AS TOTAL_SALARY
FROM EMPLOYEE
GROUP BY DEPTID;
```

### Example - Using COUNT function

**Q.** Write a query to display department ids and total number of employees in each departments.

```
     MYSQL>SELECT DEPTID, COUNT(*) AS NO_OF_EMPLOYEES

FROM EMPLOYEE

GROUP BY DEPTID;
```

### Example - Using MAX function

**Q.** Write a query to display department ids and maximum salary of employees in each departments.

```
      MYSQL>SELECT DEPTID, MAX(SALARY) AS MAX_SALARY
FROM EMPLOYEE
GROUP BY DEPTID;
```

### Example - Using MIN function

**Q.** Write a query to display department ids and minimum salary of employees in each departments.

```
      MYSQL>SELECT DEPTID, MIN(SALARY) AS MIN_SALARY
FROM EMPLOYEE
GROUP BY DEPTID;
```

### Example - Using AVG function

**Q.** Write a query to display department ids and average salary of employees in each departments.

```
      MYSQL>SELECT DEPTID, AVG(SALARY) AS AVG_SALARY
FROM EMPLOYEE
GROUP BY DEPTID;
```

## The HAVING Clause

The MySQL HAVING clause is used in combination with the to restrict the groups of returned rows to only those whose the condition is TRUE. The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax

```
SELECT EXPRESSION1, EXPRESSION2, ... EXPRESSION_N,
AGGREGATE_FUNCTION (EXPRESSION)
FROM TABLES
[WHERE CONDITIONS]
GROUP BY EXPRESSION1, EXPRESSION2, ... EXPRESSION_N
HAVING CONDITION;
```

### Example - Using SUM function

**Q.** Write a query to display department ids having total salary greater than 180000.

```
      MYSQL>SELECT DEPTID, SUM(SALARY) AS TOTAL_SALARY
FROM EMPLOYEE
GROUP BY DEPTID
           HAVING SUM(SALARY) >180000;
```

OR

```
     MYSQL>SELECT DEPTID, SUM(SALARY) AS TOTAL_SALARY
FROM EMPLOYEE
GROUP BY DEPTID
           HAVING TOTAL_SALARY >180000;
```

### Example - Using COUNT function

**Q.** Write a query to display department ids having total employees greater than two.

```
     MYSQL>SELECT DEPTID, COUNT(*) AS NO_OF_EMPLOYEES
FROM EMPLOYEE
GROUP BY DEPTID
           HAVING COUNT(*)>2;
```

OR

```
     MYSQL>SELECT DEPTID, COUNT(*) AS NO_OF_EMPLOYEES
FROM EMPLOYEE
GROUP BY DEPTID
           HAVING NO_OF_EMPLOYEES>2;
```

### Example - Using MAX function

**Q.** Write a query to display department ids and maximum salary of employee in each departments having more than two employees.

```
     MYSQL>SELECT DEPTID, MAX(SALARY)
FROM EMPLOYEE
GROUP BY DEPTID
           HAVING COUNT(DEPTID)>2;
```

### Example - Using MIN function

**Q.** Write a query to display department ids and minimum salary of employees in each departments having atmost two employees.

```
     MYSQL> SELECT DEPTID, MIN(SALARY)
FROM EMPLOYEE
GROUP BY DEPTID
           HAVING COUNT(DEPTID)<=2;
```

### Example - Using AVG function

**Q.** Write a query to display department ids having average salary greater than 75000.

```
     MYSQL> SELECT DEPTID, AVG(SALARY)
FROM EMPLOYEE
GROUP BY DEPTID
```

```
        HAVING AVG(SALARY)>75000;
```

## MySQL JOINS

**Aim**

To execute and verify different MySQL joint quires.

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

There are three types of MySQL joins:

- MySQL INNER JOIN (or sometimes called simple join)

- MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

This session is based on the following two tables "AVAILCOURSE" and "APPLICANT", having the following data.

**AVAILCOURSE**

| **COURSEID** | **CNAME** |
|---|---|
| 1 | BCA |
| 2 | BBA |
| 3 | BCom |
| 4 | BSc Maths |

**APPLICANT**

| **APPNO** | **NAME** | **CID** | **TMARK** |
|---|---|---|---|
| 101 | ANEESH | 2 | 900 |
| 102 | VARUN | 1 | 950 |
| 103 | HARI | 3 | 930 |
| 104 | VEENA | 5 | 880 |
| 105 | BINU | 6 | 938 |

### MySQL Inner JOIN (Simple Join)

The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

Syntax:

```
MYSQL>SELECT COLUMNS
      FROM TABLE1 INNER JOIN TABLE2
      ON TABLE1.COLUMN = TABLE2.COLUMN;
```

Q. Write a query to display course name, application number, applicant name and total mark of applicants by using natural join.

```
MYSQL>SELECT AVAILCOURSE.CNAME,APPLICANT.APPNO,APPLICANT.NAME,APPLICANT.TMARK
FROM AVAILCOURSE INNER JOIN APPLICANT
ON AVAILCOURSE.COURSEID= APPLICANT.CID;
```

**OR**

```
MYSQL>SELECT AVAILCOURSE.CNAME,APPLICANT.APPNO,APPLICANT.NAME,APPLICANT.TMARK
FROM AVAILCOURSE,APPLICANT
WHERE AVAILCOURSE.COURSEID= APPLICANT.CID;
```

## MySQL Left Outer Join

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

Syntax:

```
MYSQL>SELECT COLUMNS
      FROM TABLE1 LEFT [OUTER] JOIN TABLE2
      ON TABLE1.COLUMN = TABLE2.COLUMN;
```

Q. Write a query to display course name, application number, applicant name and total mark of applicants by using left outer join.

```
MYSQL>SELECT AVAILCOURSE.CNAME,APPLICANT.APPNO,APPLICANT.NAME,APPLICANT.TMARK
FROM AVAILCOURSE LEFT JOIN APPLICANT
ON AVAILCOURSE.COURSEID= APPLICANT.CID;
```

## MySQL Right Outer Join

The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.

Syntax:

```
MYSQL>SELECT COLUMNS
      FROM TABLE1 RIGHT [OUTER] JOIN TABLE2
      ON TABLE1.COLUMN = TABLE2.COLUMN;
```

R. Write a query to display course name, application number, applicant name and total mark of applicants by using right outer join.

```
MYSQL>SELECT AVAILCOURSE.CNAME,APPLICANT.APPNO,APPLICANT.NAME,APPLICANT.TMARK
FROM AVAILCOURSE RIGHT JOIN APPLICANT
ON AVAILCOURSE.COURSEID= APPLICANT.CID;
```

# SUBQURIES

**Aim**

To execute and verify MySQL subquries.

**SUBQUERY**

A subquery is a query in a query. It is also called an inner query or a nested query. A subquery can be used anywhere an expression is allowed. It is a query expression enclosed in parentheses. Subqueries can be used with SELECT, INSERT, UPDATE, or DELETE statements.

In this session, we will be using the following tables:

**CARS**

| ID | NAME | COST |
|---|---|---|
| 1 | AUDI | 52642 |
| 2 | MERCEDES | 57127 |
| 3 | SKODA | 9000 |
| 4 | VOLVO | 29000 |
| 5 | BENTLEY | 350000 |
| 6 | CITROEN | 21000 |
| 7 | HUMMER | 41400 |
| 8 | VOLKSWAGEN | 21600 |

**CUSTOMERS**

| CUSTOMERID | NAME |
|---|---|
| 1 | PAUL NOVAK |
| 2 | TERRY NEILS |
| 3 | JACK FONDA |
| 4 | TOM WILLIS |

**RESERVATIONS**

| ID | CUSTOMERID |
|---|---|
| 1 | 1 |

| | |
|---|---|
| 2 | 2 |
| 3 | 2 |
| 4 | 1 |
| 5 | 3 |

**Subquery with the INSERT statement**

Q. Write a query to insert data into the table CAR2 from the table CARS

```
MYSQL> CREATE TABLE CARS2
       (
             ID INT NOT NULL PRIMARY KEY,
             NAME VARCHAR(50) NOT NULL,
             COST INT NOT NULL
       );

MYSQL> INSERT INTO CARS2 AS SELECT * FROM CARS;

MYSQL> SELECT * FROM CARS2;
```

| ID | NAME | COST |
|---|---|---|
| 1 | AUDI | 52642 |
| 2 | MERCEDES | 57127 |
| 3 | SKODA | 9000 |
| 4 | VOLVO | 29000 |
| 5 | BENTLEY | 350000 |
| 6 | CITROEN | 21000 |
| 7 | HUMMER | 41400 |
| 8 | VOLKSWAGEN | 21600 |

**Scalar Subqueries**

A scalar subquery returns a single value.

```
MYSQL> SELECT NAME
       FROM CUSTOMERS
       WHERE CUSTOMERID=(SELECT CUSTOMERID FROM RESERVATIONS WHERE ID=5);
```

| NAME |
|---|
| JACK FONDA |

The query enclosed in parentheses is the subquery. It returns one single scalar value. The returned value is then used in the outer query. In this scalar subquery, we return the name of the customer from the Customers table, whose reservation has Id equal to 5 in the Reservations table.

**Table subqueries**

A table subquery returns a result table of zero or more rows

```
MYSQL> SELECT NAME

       FROM CUSTOMERS

       WHERE CUSTOMERID IN(SELECT DISTINCT CUSTOMERID FROM RESERVATIONS);
```

| **NAME** |
|---|
| PAUL NOVAK |
| TERRY NEILS |
| JACK FONDA |

The above query returns the names of the customers, who made some reservations. The inner query returns customer Ids from the Reservations table. We use the IN predicate to select those names of customers, who have their CustomerId returned from the inner select query.

```
MYSQL> SELECT DISTINCT NAME FROM CUSTOMERS JOIN RESERVATIONS ON
CUSTOMERS.CUSTOMERID=RESERVATIONS.CUSTOMERID;
```

| **NAME** |
|---|
| PAUL NOVAK |
| TERRY NEILS |
| JACK FONDA |

The previous subquery can be rewritten using MySQL join.

**Correlated subqueries**

A correlated subquery is a subquery that uses values from the outer query in its WHERE clause. The subquery is evaluated once for each row processed by the outer query.

```
MYSQL> SELECT NAME FROM CARS WHERE COST < (SELECT AVG(COST) FROM CARS);
```

| **NAME** |
|---|
| AUDI |
| MERCEDES |
| SKODA |
| VOLVO |

| |
|---|
| CITROEN |
| HUMMER |
| VOLKSWAGEN |

In the above correlated subquery, we return all cars that cost below the average price of all cars in the table.

**Subqueries with EXISTS, NOT EXISTS**

If a subquery returns any values, then the predicate EXISTS returns TRUE, and NOT EXISTS FALSE.

```
MYSQL> SELECT NAME

       FROM CUSTOMERS

       WHERE EXISTS (SELECT * FROM RESERVATIONS WHERE
       CUSTOMERS.CUSTOMERID=RESERVATIONS.CUSTOMERID);
```

| **NAME** |
|---|
| PAUL NOVAK |
| TERRY NEILS |
| JACK FONDA |

In the above SQL statement we select all customers' names, which have an entry in the Reservations table.

```
MYSQL> SELECT NAME

       FROM CUSTOMERS

       WHERE NOT EXISTS (SELECT * FROM RESERVATIONS WHERE
       CUSTOMERS.CUSTOMERID=RESERVATIONS.CUSTOMERID);
```

| **Name** |
|---|
| Tom Willis |

In this query, we return all customers that do not have an entry in the Reservations table. Both SQL queries are correlated queries.

# SET OPERATIONS IN SQL

**Aim**

To execute and verify set operations in MySQL.

**SET Operations**

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions. There are mainly 4 different types of SET operations.

- UNION
- UNION ALL
- INTERSECT
- EXCEPT

**1. Union**

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

**The First table**

| ID | NAME |
|---|---|
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

| ID | NAME |
|---|---|
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

Union SQL query will be:

```
    SELECT * FROM First
    UNION
    SELECT * FROM Second;
```

The result  set table will look like:

| ID | NAME |
|---|---|
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**2. Union All**

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
    SELECT column_names FROM table1
    UNION ALL
    SELECT column_names FROM table2;
```

Example:

Union All query will be like:

```
    SELECT * FROM First
    UNION ALL
    SELECT * FROM Second;
```

The resultset table will look like:

| ID | NAME |
|---|---|
| 1 | Jack |

| | |
|---|---|
| 2 | Harry |
| 3 | Jackson |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

### 3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
SELECT column_names FROM table1
INTERSECT
SELECT column_names FROM table2;
```

Example:

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like:

| ID | NAME |
|---|---|
| 3 | Jackson |

### 4. Except

- It combines the result of two SELECT statements. Except operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_names FROM table1
EXCEPT
SELECT column_names FROM table2;
```

Example

Minus query will be:

```
SELECT * FROM First
EXCEPT
SELECT * FROM Second;
```

The resultset table will look like:

| ID | NAME |
|---|---|
| 1 | Jack |
| 2 | Harry |

# PATTERN MATCHING AND RANGE SEARCHING

**Aim**

To execute and verify pattern matching and range searching operations in MySQL.

**PATTERN MATCHING**

SQL pattern matching allows you to search for patterns in data if you don't know the exact word or phrase you are seeking.

**The LIKE Operator**

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

| | |
|---|---|
| **%** | The percent sign represents zero, one, or multiple characters |
| _ | The underscore represents a single character |

The percent sign and the underscore can also be used in combinations.

LIKE Syntax

```
SELECT column1, column2, ...
```

```
        FROM table_name
        WHERE column  LIKE pattern;
```

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

| LIKE Operator | Description |
|---|---|
| SELECT * FROM Employee<br>WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| SELECT * FROM Employee<br>WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| SELECT * FROM Employee<br>WHERE CustomerName LIKE '%ab%' | Finds any values that have "ab" in any position |
| SELECT * FROM Employee<br>WHERE CustomerName LIKE '_r%' | Finds any values that have "r" in the second position |
| SELECT * FROM Employee<br>WHERE CustomerName LIKE 'a_ _%' | Finds any values that start with "a" and are at least 3 characters in length |
| SELECT * FROM Employee<br>WHERE ContactName LIKE 'a%b' | Finds any values that start with "a" and ends with "b" |

Example

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '_' operators –

| Sr.No. | Statement & Description |
|---|---|
| 1 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '200%'<br>Finds any values that start with 200. |
| 2 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '%200%'<br>Finds any values that have 200 in any position. |
| 3 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '_00%'<br>Finds any values that have 00 in the second and third positions. |
| 4 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '2_%_%'<br>Finds any values that start with 2 and are at least 3 characters in length. |
| 5 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '%2'<br>Finds any values that end with 2. |
| 6 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '_2%3'<br>Finds any values that have a 2 in the second position and end with a 3. |
| 7 | SELECT * FROM EMPLOYEE WHERE SALARY LIKE '2_ _ _3'<br>Finds any values in a five-digit number that start with 2 and end with 3. |

**RANGE SEARCHING**

In order to select data that is within a range of values, the BETWEEN operator is used.

**BETWEEN (BETWEEN --- AND)**

The BETWEEN operator allows you to easily test if an expression is within a range of values. The values can be text, date, or numbers. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement. The BETWEEN operator will return the records where expression is within the range of value1 and value2.

Syntax:

```
        SELECT column_name(s)
        FROM table_name
        WHERE column_name BETWEEN value1 AND value2;
```

Using BETWEEN with Numeric Values:

List all the Employee Fname, Lname who is having salary between 30000 and 45000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary BETWEEN 30000 AND 45000;
```

Using BETWEEN with Date Values:

Find all the Employee having Date of Birth Between 01-01-1985 and 30-12-1990.

```
SELECT Fname, Lname
FROM Employee
where DOB BETWEEN '1985-01-01' AND '1990-12-30';
```

Using NOT operator with BETWEEN

Find all the Employee name whose salary is not in the range of 30000 and 45000.

```
SELECT Fname, Lname
FROM Emplyoee
WHERE Salary NOT BETWEEN 30000 AND 45000;
```

**IN operator**

IN operator allows you to easily test if the expression matches any value in the list of values. It is used to remove the need of multiple OR condition in SELECT, INSERT, UPDATE or DELETE. You can also use NOT IN to exclude the rows in your list.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (list_of_values);
```

Example

Find the Fname, Lname of the Employees who have Salary equal to 30000, 40000 or 25000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary IN (30000, 40000, 25000);
```

Example

Find the Fname, Lname of the Employees who have Salary not equal to 30000, 40000 or 25000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary NOT IN (30000, 40000, 25000);
```

# VIEWS IN MYSQL

**Aim**

To execute and verify views in MySQL

**VIEWS**

In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)

Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name
from instructor
```

A **view** provides a mechanism to hide certain data from the view of certain users. Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a view.

Creating Views

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.

The basic CREATE VIEW syntax is as follows −

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SELECT * FROM CUSTOMERS_VIEW;
```

The WITH CHECK OPTION in Views

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition. If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

Updating a View

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.

- The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
- The SELECT statement should not have the DISTINCT keyword.
- The view should have all NOT NULL values.
- The view should not be created using nested queries or complex queries.
- The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
    UPDATE CUSTOMERS_VIEW
SET AGE = 35
WHERE name = 'Ramesh';
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command. We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

Syntax:

```
INSERT INTO view_name(column1, column2 , column3,..)
VALUES(value1, value2, value3..);
```

Deleting a row from a View:

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

Syntax:

```
DELETE FROM view_name
WHERE condition;
```

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

# STRING FUNCTIONS IN MYSQL

## Aim

To execute and verify string function and date/time functions in MySQL.

### MySQL String Function

MySQL string functions manipulate the character string data effectively. The following list indicates the most commonly used MySQL string functions that allow you to manipulate character string data effectively.

- CONCAT(): Combines two or more strings together.
  ```
  SELECT CONCAT('Hello', ' ', 'World') AS Result;
  Output: 'Hello World'
  ```
- LENGTH(): Returns the length (number of characters) of a string.
  ```
  SELECT LENGTH('MySQL') AS Length;
  Output: 5
  ```
- UPPER(): Converts a string to uppercase.
  ```
  SELECT UPPER('mysql') AS Uppercase;
  Output: 'MYSQL'
  ```
- LOWER(): Converts a string to lowercase.
  ```
  SELECT LOWER('MySQL') AS Lowercase;
  Output: 'mysql'
  ```
- SUBSTRING(): Extracts a substring from a string.
  ```
  SELECT SUBSTRING('Hello, World!', 1, 5) AS Substr;
  Output: 'Hello'
  ```
- LEFT(): Returns a specified number of characters from the left side of a string.
  ```
  SELECT LEFT('MySQL', 3) AS LeftStr;
  Output: 'MyS'
  ```
- RIGHT(): Returns a specified number of characters from the right side of a string.
  ```
  SELECT RIGHT('MySQL', 3) AS RightStr;
  Output: 'SQL'
  ```
- TRIM(): Removes leading and trailing spaces or specified characters from a string.
  ```
  SELECT TRIM('  MySQL  ') AS Trimmed;
  Output: 'MySQL'
  ```
- REPLACE(): Replaces occurrences of a substring with another substring in a string.
  ```
  SELECT REPLACE('Hello, World!', 'World', 'MySQL') AS Replaced;
  Output: 'Hello, MySQL!'
  ```
- CONCAT_WS(): Concatenates strings with a specified separator.

```
SELECT CONCAT_WS(', ', 'SAS', 'Konni', 'Pathanamthitta) AS Address;
Output: 'SAS, Konni, Pathanamthitta’
```

- INSTR(): Finds the position of a substring within a string.
  ```
  SELECT INSTR('Hello, World!', 'World') AS Position;
  Output: 7
  ```
- REVERSE(): Reverses the characters in a string.
  ```
  SELECT REVERSE('MySQL') AS Reversed;
  Output: 'LQSyM'
  ```

EXAMPLE:

Consider the following table employees:

```
CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
email VARCHAR(100)
);
```

```
INSERT INTO employees (employee_id, first_name, last_name, email)
VALUES
(1, 'John', 'Doe', 'john.doe@example.com'),
(2, 'Jane', 'Smith', 'jane.smith@example.com'),
(3, 'Alice', 'Johnson', 'alice.johnson@example.com');
```

Using CONCAT() to Combine First Name and Last Name:

You can use CONCAT() to create a full name from the first_name and last_name columns:

```
SELECT employee_id, CONCAT(first_name, ' ', last_name) AS full_name
FROM employees;
```

Output:

```
+-------------+--------------+
| employee_id | full_name    |
+-------------+--------------+
| 1           | John Doe     |
| 2           | Jane Smith   |
| 3           | Alice Johnson|
+-------------+--------------+
```

Using SUBSTRING() to Extract Part of an Email:

You can use SUBSTRING() to extract the domain part of the email addresses:

```
SELECT employee_id, SUBSTRING(email, LOCATE('@', email) + 1) AS
email_domain
FROM employees;
```

Output:

```
+-------------+------------------+
| employee_id | email_domain     |
+-------------+------------------+
| 1           | example.com      |
| 2           | example.com      |
| 3           | example.com      |
+-------------+------------------+
```

Using REPLACE() to Update Data:
You can use REPLACE() to update email domains for a specific employee:

```
UPDATE employees
SET email = REPLACE(email, 'example.com', 'newdomain.com')
WHERE employee_id = 1;
```

Using CONCAT_WS() to Combine Multiple Columns:
You can use CONCAT_WS() to combine multiple columns with a specified separator:

```
SELECT employee_id, CONCAT_WS(', ', last_name, first_name) AS
last_name_first_name
FROM employees;
```

Output:

```
+-------------+----------------------+
| employee_id | last_name_first_name |
+-------------+----------------------+
| 1           | Doe, John            |
| 2           | Smith, Jane          |
| 3           | Johnson, Alice       |
+-------------+----------------------+
```

# OPERATORS IN MYSQL

## Aim

To execute and verify different operators in MySQL.

### MySQL Operators

In MySQL, operators are used to perform various operations on data, such as arithmetic operations, comparison operations, logical operations, and more. Here are some common operators in MySQL along with examples:

**Arithmetic Operators:**

- + (Addition): Adds two values together.
- - (Subtraction): Subtracts the right operand from the left operand.
- * (Multiplication): Multiplies two values.
- / (Division): Divides the left operand by the right operand.
- % (Modulus): Returns the remainder of the division of the left operand by the right operand.

Example:

```
SELECT 10 + 5;        -- Result: 15
SELECT 20 - 8;        -- Result: 12
SELECT 6 * 4;         -- Result: 24
SELECT 15 / 3;        -- Result: 5
SELECT 17 % 3;        -- Result: 2
```

**Comparison Operators:**

- = (Equal to): Tests if two values are equal.
- != or <> (Not equal to): Tests if two values are not equal.
- < (Less than): Tests if the left operand is less than the right operand.
- > (Greater than): Tests if the left operand is greater than the right operand.
- <= (Less than or equal to): Tests if the left operand is less than or equal to the right operand.
- >= (Greater than or equal to): Tests if the left operand is greater than or equal to the right operand.

Example:

```
SELECT 10 = 5;        -- Result: 0 (False)
SELECT 10 != 5;       -- Result: 1 (True)
SELECT 8 < 12;        -- Result: 1 (True)
SELECT 15 > 20;       -- Result: 0 (False)
SELECT 10 <= 10;      -- Result: 1 (True)
SELECT 30 >= 30;      -- Result: 1 (True)
```

**Logical Operators:**

- AND: Performs a logical AND operation.

- OR: Performs a logical OR operation.
- NOT: Negates a condition.

Example:

```
     SELECT (10 > 5) AND (8 < 12);   -- Result: 1 (True)
     SELECT (10 > 5) OR (8 > 12);    -- Result: 1 (True)
     SELECT NOT (10 = 5);            -- Result: 1 (True)
```

**Concatenation Operator:**

- CONCAT(): Concatenates two or more strings.

Example:

```
SELECT CONCAT('Hello', ' ', 'World');  -- Result: 'Hello World'
```

**Assignment Operator:**

- = (Assignment): Assigns a value to a variable or column.

Example:

```
     • SET var1= 10;        -- Assign 10 to variable var1
     • UPDATE table_name
       SET column1 = 20;    -- Assign 20 to column1 in a table
```

These are some of the common operators in MySQL. You can use them in SQL queries to perform various operations on data and manipulate the results according to your requirements.

**Examples with database table - 1**

Assuming we have a table named employees with the following structure:

```
SQL> CREATE TABLE employees (
id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
salary INT,
hire_date DATE
);
```

Sample data for employee table.

```
SQL> INSERT INTO employees (id, first_name, last_name, salary,
hire_date) VALUES
(1, 'John', 'Doe', 50000, '2020-01-15'),
(2, 'Jane', 'Smith', 60000, '2019-11-10'),
(3, 'Bob', 'Johnson', 55000, '2021-03-20'),
(4, 'Alice', 'Brown', 52000, '2018-05-05');
```

Arithmetic Operators:

1. Increment the salary of each employee by 15 %:

```
     UPDATE employees SET salary=salary*1.15;
```

2. Decrement ₹2000 from the salary of the employee whose id is 3.

```
     UPDATE employees SET salary=salary-2000 WHERE id=3;
```

Comparison Operators:

1. Find employees earning more than ₹55,000:

```
    SELECT * FROM employees WHERE salary > 55000;
```

2. Find employees hired in or after 2020:

```
    SELECT * FROM employees WHERE hire_date >= '2020-01-01';
```

Logical Operators:

1. Find employees with salaries between ₹50,000 and ₹60,000:

```
    SELECT * FROM employees WHERE salary >= 50000 AND salary <=
    60000;
```

2. Find employees hired before 2020 or with a salary greater than ₹60,000:

```
    SELECT * FROM employees WHERE hire_date < '2020-01-01' OR
    salary > 60000;
```

Concatenation Operator:

1. Combine the first and last names of employees:

```
    SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM
    employees;
```

Assignment Operator:

1. Update the salary of an employee with ID 3:

```
    UPDATE employees SET salary = 58000 WHERE id = 3;
```

**Examples with database tables – 2**

Assuming we have two tables: students and courses.

```
     SQL>CREATE TABLE students (
     student_id INT PRIMARY KEY,
     first_name VARCHAR(50),
     last_name VARCHAR(50),
     age INT,
     gender VARCHAR(10)
     );
     SQL>CREATE TABLE courses (
     course_id INT PRIMARY KEY,
     course_name VARCHAR(50),
     instructor VARCHAR(50),
     credits INT
     );
```

Sample data for students and courses tables

```
     SQL>INSERT   INTO   students   (student_id,   first_name,
     last_name, age, gender) VALUES
     (1, 'John', 'Doe', 20, 'Male'),
     (2, 'Jane', 'Smith', 22, 'Female'),
     (3, 'Bob', 'Johnson', 21, 'Male');
     SQL>INSERT   INTO   courses   (course_id,   course_name,
     instructor, credits) VALUES
     (101, 'Mathematics', 'Prof. Smith', 4),
     (102, 'History', 'Prof. Johnson', 3),
     (103, 'Biology', 'Prof. Brown', 4);
```

INSERT Statement with Operators:

Insert a new student into the students table with a calculated age:

```
     INSERT INTO students (student_id, first_name, last_name,
     age, gender)VALUES (4, 'Alice', 'Brown', YEAR(CURDATE()) -
     1998, 'Female');
```

UPDATE Statement with Operators:

Increase the age of all male students by 1:

```
UPDATE students
SET age = age + 1
WHERE gender = 'Male';
```

DELETE Statement with Operators:

Delete all students younger than 22:

```
DELETE FROM students
WHERE age < 22;
```

SELECT Statement with Operators:

Find students with a first name starting with 'J' and whose age is at least 21:

```
SELECT * FROM students
WHERE first_name LIKE 'J%' AND age >= 21;
```

Find courses with more than 3 credits or taught by 'Prof. Smith':

```
SELECT * FROM courses
WHERE credits > 3 OR instructor = 'Prof. Smith';
```

Combine first and last names of students:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM students;
```

# DATE AND TIME FUNCTIONS IN MYSQL

**Aim**

To execute and verify different operators in MySQL.

### Date and time functions in MySQL

MySQL provides a variety of date and time functions that allow you to manipulate and work with date and time values in your database. Here are some commonly used date and time functions in MySQL along with examples:

- NOW(): Returns the current date and time.
  ```
  SELECT NOW();
  ```
- CURDATE(): Returns the current date.
  ```
  SELECT CURDATE();
  ```
- CURTIME(): Returns the current time.
  ```
  SELECT CURTIME();
  ```
- DATE(): Extracts the date part from a datetime expression.
  ```
  SELECT DATE(NOW());
  ```
- TIME(): Extracts the time part from a datetime expression.
  ```
  SELECT TIME(NOW());
  ```
- DATE_FORMAT(): Formats a date or time value as specified.
  ```
  SELECT DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s');
  ```
- DAY(): Returns the day of the month for a given date.
  ```
  SELECT DAY('2023-09-04');
  ```
- MONTH(): Returns the month for a given date.
  ```
  SELECT MONTH('2023-09-04');
  ```
- YEAR(): Returns the year for a given date.
  ```
  SELECT YEAR('2023-09-04');
  ```
- HOUR(): Returns the hour for a given time.
  ```
  SELECT HOUR('14:30:45');
  ```
- MINUTE(): Returns the minute for a given time.
  ```
  SELECT MINUTE('14:30:45');
  ```
- SECOND(): Returns the second for a given time.
  ```
  SELECT SECOND('14:30:45');
  ```

### Examples with database table - 1

Suppose we have a table called orders with the following schema:

```
CREATE TABLE orders (
order_id INT PRIMARY KEY,
order_date DATE,
order_time TIME
);
```

Inserting Data:

```
INSERT INTO orders (order_id, order_date, order_time)
VALUES(1, '2023-09-04', '14:30:00'),
(2, '2023-09-05', '10:45:00'),
(3, '2023-09-06', '16:15:00');
```

Selecting Orders Placed Today:

```
SELECT *
FROM orders
WHERE order_date = CURDATE();
```

Selecting Orders Placed This Month:

```
SELECT *FROM orders
WHERE MONTH(order_date) = MONTH(CURDATE()) AND
YEAR(order_date) = YEAR(CURDATE());
```

Calculating Order Age in Days:

```
SELECT order_id, DATEDIFF(CURDATE(), order_date) AS
order_age_in_days
FROM orders;
```

Formatting Date and Time:

```
SELECT order_id, DATE_FORMAT(order_date, '%Y-%m-%d') AS
formatted_date, TIME(order_time) AS formattime
FROM orders;
```

Finding Orders Placed After a Certain Time:

```
SELECT *
FROM orders
WHERE order_time > '15:00:00';
```

Adding Days to Order Dates:

```
SELECT order_id, ADDDATE(order_date, 7) AS new_order_date
FROM orders;
+----------+----------------+
| order_id | new_order_date |
+----------+----------------+
|        1 | 2023-09-11     |
|        2 | 2023-09-12     |
|        3 | 2023-09-13     |
|        4 | 2023-09-16     |
+----------+----------------+
```

Subtracting Time from Order Times:

```
SELECT order_id, SUBTIME(order_time, '00:15:00') AS
new_order_time FROM orders;
```

Calculating Average Order Age in Days:

```
SELECT AVG(DATEDIFF(CURDATE(), order_date)) AS
avg_order_age_in_days
FROM orders;
```

Finding the Latest Order:

```
SELECT *FROM orders WHERE order_date = (SELECT
MAX(order_date) FROM orders);
```

**Examples with database table - 2**

Suppose we have a table called events with the following schema:

```
CREATE TABLE events (
event_id INT PRIMARY KEY,
event_name VARCHAR(255),
event_date DATE,
event_time TIME
);
```

Inserting Data:

```
INSERT  INTO  events  (event_id,  event_name,  event_date,
event_time) VALUES
(1, 'Meeting', '2023-09-04', '14:30:00'),
(2, 'Conference', '2023-09-05', '10:45:00'),
(3, 'Webinar', '2023-09-06', '16:15:00');
```

Updating Event Date:

Update the event date for a specific event:

```
UPDATE events
SET event_date = '2023-09-07'
WHERE event_id = 2;
```

Updating Event Time:

Update the event time for a specific event:

```
UPDATE events
SET event_time = '15:00:00'
WHERE event_id = 3;
```

Deleting Past Events:

Delete events that have already occurred (assuming today is '2023-09-05'):

```
DELETE FROM events
WHERE event_date < '2023-09-05';
```

Finding Events Scheduled for a Specific Date:

Find events scheduled for '2023-09-06':

```
SELECT *
FROM events
WHERE event_date = '2023-09-06';
```

## CUSTOMER-SALE DATABASE

### Aim

To create tables and perform queries in a customer-sale scenario

**Database Schema for a customer-sale scenario**

Customer(**cust id : integer**, cust_name: string)
Item(**item id: integer**, item_name: string, price: integer)
Sale(**bill no: integer**, bill_data: date, **cust_id: integer**, **item_id: integer**, qty_sold: integer)

For the above schema, perform the following.

- Create the tables with the appropriate integrity constraints
- Insert around 5 records in each of the tables
- List all the bills for the current date with the customer names and item numbers
- List the total Bill details with the quantity sold, price of the item and the final amount
- List the details of the customer who have bought a product which has a price >200
- Give a count of how many products have been bought by each customer
- Give a list of products bought by a customer having cust_id as 5
- List the item details which are sold as of today

a) Create the tables with the appropriate integrity constraints

```
MYSQL> CREATE TABLE CUSTOMER
      (
            CUST_ID INTEGER(5) PRIMARY KEY,
            CUST_NAME VARCHAR(15)
      );


      +-----------+-------------+------+-----+---------+-------+
      | Field     | Type        | Null | Key | Default | Extra |
      +-----------+-------------+------+-----+---------+-------+
      | CUST_ID   | int         | NO   | PRI | NULL    |       |
      | CUST_NAME | varchar(15) | YES  |     | NULL    |       |
      +-----------+-------------+------+-----+---------+-------+
MYSQL> CREATE TABLE ITEM
      (
            ITEM_ID INTEGER(4) PRIMARY KEY,
            ITEM_NAME VARCHAR(15),
            PRICE DECIMAL(6,2)
      );
      +-----------+--------------+------+-----+---------+-------+
      | Field     | Type         | Null | Key | Default | Extra |
      +-----------+--------------+------+-----+---------+-------+
      | ITEM_ID   | int          | NO   | PRI | NULL    |       |
      | ITEM_NAME | varchar(15)  | YES  |     | NULL    |       |
      | PRICE     | decimal(6,2) | YES  |     | NULL    |       |
      +-----------+--------------+------+-----+---------+-------+
MYSQL>CREATE TABLE SALE
      (
            BILL_NO INTEGER(5) PRIMARY KEY,
            BILL_DATE DATE,
            CUST_ID INTEGER(5),
          ITEM_ID INTEGER(4),
          QTY_SOLD INTEGER(4),
          FOREIGN KEY(CUST_ID) REFERENCES CUSTOMER(CUST_ID),
          FOREIGN KEY(ITEM_ID) REFERENCES ITEM(ITEM_ID)
     );

     MYSQL> DESC SALE;

     +-----------+------+------+-----+---------+-------+
     | FIELD     | TYPE | NULL | KEY | DEFAULT | EXTRA |
     +-----------+------+------+-----+---------+-------+
     | BILL_NO   | INT  | NO   | PRI | NULL    |       |
     | BILL_DATE | DATE | YES  |     | NULL    |       |
     | CUST_ID   | INT  | YES  | MUL | NULL    |       |
     | ITEM_ID   | INT  | YES  | MUL | NULL    |       |
     | QTY_SOLD  | INT  | YES  |     | NULL    |       |
     +-----------+------+------+-----+---------+-------+
```

b) Insert around 5 records in each of the tables.

```
insert into customer3 values (1,'mariam'),(2,'haya'),(3,'jennifer'),
(4,'thor'),(5,'jimmy');
mysql> select * from customer3;
     +---------+-----------+
     | CUST_ID | CUST_NAME |
     +---------+-----------+
     |       1 | mariam    |
     |       2 | haya      |
     |       3 | jennifer  |
     |       4 | thor      |
     |       5 | jimmy     |
     +---------+-----------+
insert into item values
(20,'soap','60'),(21,'powder','200'),(22,'bulb','70'),(23,'bag','360
'),(24,'book','40');
     mysql> select  * from item;
     +---------+-----------+--------+
     | ITEM_ID | ITEM_NAME | PRICE  |
     +---------+-----------+--------+
     |      20 | soap      |  60.00 |
     |      21 | powder    | 200.00 |
     |      22 | bulb      |  70.00 |
     |      23 | bag       | 360.00 |
     |      24 | book      |  40.00 |
     +---------+-----------+--------+
```

```
insert into sale values (101,'2023-09-1',1,20,2),(102,'2022-3-
6',2,20,1),
(103,'2023-08-1',1,21,2),(104,'2023-2-3',3,23,3),(106,'2023-09-
1',5,20,2);
```

```
mysql> select * from sale;
+---------+------------+---------+---------+----------+
| BILL_NO | BILL_DATE  | CUST_ID | ITEM_ID | QTY_SOLD |
+---------+------------+---------+---------+----------+
|     101 | 2023-09-01 |       1 |      20 |        2 |
|     102 | 2022-03-06 |       2 |      20 |        1 |
|     103 | 2023-08-01 |       1 |      21 |        2 |
|     104 | 2023-02-03 |       3 |      23 |        3 |
|     106 | 2023-09-01 |       5 |      20 |        2 |
+---------+------------+---------+---------+----------+
```

c) List all the bills for the current date with the customer names and item numbers

```
MYSQL> SELECT C.CUST_NAME, I.ITEM_ID, S.BILL_NO
     FROM CUSTOMER C, ITEM I, SALE S
     WHERE  C.CUST_ID=S.CUST_ID  AND  S.ITEM_ID=I.ITEM_ID  AND
S.BILL_DATE=CURDATE();
     +-----------+---------+---------+
     | CUST_NAME | ITEM_ID | BILL_NO |
     +-----------+---------+---------+
     | mariam    |      20 |     101 |
     | jimmy     |      20 |     106 |
     +-----------+---------+---------+
```

d) List the total Bill details with the quantity sold, price of the item and the final amount

```
MYSQL> SELECT I.PRICE, S.QTY_SOLD,(I.PRICE*S.QTY_SOLD) TOTAL
     FROM ITEM I, SALE S
     WHERE I.ITEM_ID=S.ITEM_ID;
     +--------+----------+---------+
     | PRICE  | QTY_SOLD | TOTAL   |
     +--------+----------+---------+
     |  60.00 |        2 |  120.00 |
     |  60.00 |        1 |   60.00 |
     |  60.00 |        2 |  120.00 |
     | 200.00 |        2 |  400.00 |
     | 360.00 |        3 | 1080.00 |
     +--------+----------+---------+
```

e) List the details of the customer who have bought a product which has a price>200

```
MYSQL> SELECT C.CUST_ID, C.CUST_NAME
      FROM CUSTOMER C, SALE S, ITEM I
      WHERE I.PRICE>200 AND C.CUST_ID=S.CUST_ID AND I.ITEM_ID=S.ITEM_ID;
      +---------+-----------+
      | CUST_ID | CUST_NAME |
      +---------+-----------+
      |       3 | jennifer  |
      +---------+-----------+
      1 row in set (0.00 sec)
```

f) Give a count of how many products have been bought by each customer

```
MYSQL> SELECT CUST_ID, COUNT(ITEM_ID)
      FROM SALE
      GROUP BY CUST_ID;

      +---------+----------------+
      | CUST_ID | COUNT(ITEM_ID) |
      +---------+----------------+
      |       1 |              2 |
      |       2 |              1 |
      |       3 |              1 |
      |       5 |              1 |
      +---------+----------------+
```

g) Give a list of products bought by a customer having cust_id as 5

```
MYSQL> SELECT I.ITEM_NAME
      FROM ITEM I, SALE S
      WHERE S.CUST_ID=5 AND I.ITEM_ID=S.ITEM_ID;
      +-----------+
      | ITEM_NAME |
      +-----------+
      | soap      |
      +-----------+
```

h)  List the item details which are sold as of today

```
      MYSQL> SELECT I.ITEM_ID, I.ITEM_NAME
            FROM ITEM I, SALE S
            WHERE I.ITEM_ID=S.ITEM_ID AND S.BILL_DATE=CURDATE();
+---------+-----------+
| ITEM_ID | ITEM_NAME |
+---------+-----------+
|      21 | powder    |
+---------+-----------+
```

## EMPLOYEE-PAYMENT DATABASE

**Aim**

To create tables and perform queries in an Employee-Payment scenario.

### Database Schema for a Employee-Payment scenario

employee(**emp_id : integer**, emp_name: string)
department(**dept_id: integer**, dept_name:string)

paydetails(**emp_id : integer**, dept_id: integer, basic: integer, deductions: integer, additions: integer, DOJ: date)
payroll(**emp_id : integer**, pay_date: date)

For the above schema, perform the following—

- Create the tables with the appropriate integrity constraints.
- Insert around 10 records in each of the tables.
- List the employee details department wise.
- List all the employee names who joined after particular date.
- List the details of employees whose basic salary is between 10,000 and 20,000.
- Give a count of how many employees are working in each department.
- Give a names of the employees whose netsalary>10,000.
- List the details for an employee_id=5.

a) Create the tables with the appropriate integrity constraints

```
MYSQL>CREATE TABLE EMPLOYEE
      (
           EMP_ID NUMERIC(5) PRIMARY KEY,
           EMP_NAME VARCHAR(25)
      );
mysql> desc employee;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| EMP_ID   | decimal(5,0) | NO   | PRI | NULL    |       |
| EMP_NAME | varchar(25)  | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

```
MYSQL>CREATE TABLE DEPARTMENT1
      (
           DEPT_ID NUMERIC(5) PRIMARY KEY,
           DEPT_NAME VARCHAR(20)
      );
mysql> DESC DEPARTMENT1;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| DEPT_ID   | decimal(5,0) | NO   | PRI | NULL    |       |
| DEPT_NAME | varchar(20)  | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

```
MYSQL>CREATE TABLE PAYDETAILS
```

```
      (
           EMP_ID NUMERIC(5),
           DEPT_ID NUMERIC(5),
           BASIC DECIMAL(7,2),
           DEDUCTIONS NUMERIC(5,2),
           ADDITIONS NUMERIC(5,2),
           DOJ      DATE,foreign      key(emp_id)      REFERENCES
      EMPLOYEE(EMP_ID),foreign      key(dept_id)      REFERENCES
      DEPARTMENT1(DEPT_ID)
      );
mysql> desc paydetails;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| EMP_ID     | decimal(5,0) | YES  | MUL | NULL    |       |
| DEPT_ID    | decimal(5,0) | YES  | MUL | NULL    |       |
| BASIC      | decimal(7,2) | YES  |     | NULL    |       |
| DEDUCTIONS | decimal(5,2) | YES  |     | NULL    |       |
| ADDITIONS  | decimal(5,2) | YES  |     | NULL    |       |
| DOJ        | date         | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+

MYSQL>CREATE TABLE PAYROLL
      (
           EMP_ID NUMERIC(5)REFERENCES EMPLOYEE(EMP_ID),
           PAY_DATE DATE
      );
MYSQL> DESC PAYROLL;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| EMP_ID   | decimal(5,0) | YES  |     | NULL    |       |
| PAY_DATE | date         | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

b) Insert around 10 records in each of the tables

```
INSERT INTO EMPLOYEE VALUES
(11,'FIHA'),(12,'HIBA'),
(13,'HIRA'),(14,'LAILA'),
(15,'FIBA'),(16,'LUNA'),
(17,'GLORIA'),(18,'ALIYA'),
(19,'SNEHA'),(20,'SHILPA');
```

```
           MYSQL> SELECT * FROM EMPLOYEE;
                +--------+----------+
                | EMP_ID | EMP_NAME |
                +--------+----------+
                |     11 | FIHA     |
                |     12 | HIBA     |
                |     13 | HIRA     |
                |     14 | LAILA    |
                |     15 | FIBA     |
                |     16 | LUNA     |
                |     17 | GLORIA   |
                |     18 | ALIYA    |
                |     19 | SNEHA    |
                |     20 | SHILPA   |
                +--------+----------+

INSERT INTO DEPARTMENT1 VALUES(1001,'SALES'),
                (1002,'ACCOUNTING'),(1003,'HR'),(1004,'COMPUTER')
                ,(1005,'ADVERTISEMENT'),(1006,'SECURITY'),(1007,'RES
                EARCH'),(1008,'MANAGING');
MYSQL> SELECT *FROM DEPARTMENT1;
                +---------+---------------+
                | DEPT_ID | DEPT_NAME     |
                +---------+---------------+
                |    1001 | SALES         |
                |    1002 | ACCOUNTING    |
                |    1003 | HR            |
                |    1004 | COMPUTER      |
                |    1005 | ADVERTISEMENT |
                |    1006 | SECURITY      |
                |    1007 | RESEARCH      |
                |    1008 | MANAGING      |
                +---------+---------------+
```

```
insert into paydetails values
(11,1001,19700.7,23.8,435.9,'2020-9-5'),
(12,1001,13600.7,23.0,335.9,'2018-9-5'),
(13,1002,12700.7,23.0,435.9,'2019-8-5'),
(14,1003,8800.7,23.0,435.9,'2021-9-5'),
(15,1004,9900.7,23.0,135.9,'2020-9-5'),
          (16,1004,9800.7,23.0,135.9,'2020-3-5'),
          (17,1005,9100.7,43.0,135.9,'2020-9-5'),
          (18,1006,8900.7,43.0,235.9,'2021-9-5'),
          (19,1007,7900.7,93.0,435.9,'2021-8-5'),
          (20,1008,9900.7,83.0,135.9,'2020-6-2');

          mysql> select * from paydetails;
insert into  payroll values(11,'2022-03-12'),
          (12,'2021-04-9'),(13,'2021-05-4'),
          (14,'2021-04-5'),(15,'2021-03-9'),
          (16,'2021-05-12');
          +--------+---------+----------+------------+-----------+------------+
          | EMP_ID | DEPT_ID | BASIC    | DEDUCTIONS | ADDITIONS | DOJ        |
          +--------+---------+----------+------------+-----------+------------+
          |     11 |    1001 | 19700.70 |      23.80 |    435.90 | 2020-09-05 |
          |     12 |    1001 | 13600.70 |      23.00 |    335.90 | 2018-09-05 |
          |     13 |    1002 | 12700.70 |      23.00 |    435.90 | 2019-08-05 |
          |     14 |    1003 |  8800.70 |      23.00 |    435.90 | 2021-09-05 |
          |     15 |    1004 |  9900.70 |      23.00 |    135.90 | 2020-09-05 |
          |     16 |    1004 |  9800.70 |      23.00 |    135.90 | 2020-03-05 |
          |     17 |    1005 |  9100.70 |      43.00 |    135.90 | 2020-09-05 |
          |     18 |    1006 |  8900.70 |      43.00 |    235.90 | 2021-09-05 |
          |     19 |    1007 |  7900.70 |      93.00 |    435.90 | 2021-08-05 |
          |     20 |    1008 |  9900.70 |      83.00 |    135.90 | 2020-06-02 |
          +--------+---------+----------+------------+-----------+------------+


          select * from payroll;
          +--------+------------+
          | EMP_ID | PAY_DATE   |
          +--------+------------+
          |     11 | 2022-03-12 |
          |     12 | 2021-04-09 |
          |     13 | 2021-05-04 |
          |     14 | 2021-04-05 |
          |     15 | 2021-03-09 |
          |     16 | 2021-05-12 |
          +--------+------------+
```

c) List the employee details department wise

```
     MYSQL>SELECT EMP_ID,DEPT_ID
          FROM PAYDETAILS
          ORDER BY DEPT_ID ASC, EMP_ID ASC;
```

```
+--------+---------+
| EMP_ID | DEPT_ID |
+--------+---------+
|     11 |    1001 |
|     12 |    1001 |
|     13 |    1002 |
|     14 |    1003 |
|     15 |    1004 |
|     16 |    1004 |
+--------+---------+
```

d) List all the employee names who joined after particular date

```
MYSQL>SELECT E.EMP_NAME
FROM EMPLOYEE E,PAYDETAILS P
WHERE E.EMP_ID=P.EMP_ID AND P.DOJ>’2020-09-05’;
      +----------+
      | EMP_NAME |
      +----------+
      | LAILA    |
      | ALIYA    |
      | SNEHA    |
      +----------+
```

e) List the details of employees whose basic salary is between 10,000 and 20,000

```
MYSQL> SELECT E.EMP_ID, E.EMP_NAME, P.BASIC
      FROM EMPLOYEE E, PAYDETAILS P
      WHERE E.EMP_ID=P.EMP_ID AND Basic BETWEEN 10000 AND 20000;
      +--------+----------+----------+
      | EMP_ID | EMP_NAME | BASIC    |
      +--------+----------+----------+
      |     11 | Fiha     | 19700.70 |
      |     12 | hiba     | 13600.70 |
      |     13 | hira     | 12700.70 |
      +--------+----------+----------+
```

f) Give a count of how many employees are working in each department

```
MYSQL>SELECT COUNT(EMPID),DEPTID
      FROM PAYDETAILS
      GROUP BY DEPTID;
```

```
           +----------------+----------+
           | COUNT(EMP_ID)  | DEPT_ID  |
           +----------------+----------+
           |              2 |     1001 |
           |              1 |     1002 |
           |              1 |     1003 |
           |              2 |     1004 |
           +----------------+----------+
```

g) Give a names of the employees whose netsalary>10,000

```
      MYSQL> SELECT EMPNAME
           FROM EMPLOYEE
           WHERE EMPID
                IN(SELECT EMPID FROM PAYDETAILS WHERE BASIC-DEDUCTION>10000);
                +-----------+
                | EMP_NAME  |
                +-----------+
                | Fiha      |
                | hiba      |
                | hira      |
                +-----------+
```

h) List the  details for an employee_id=11

```
      MYSQL> SELECT * FROM EMPLOYEE WHERE EMP_ID=11;
+--------+-----------+

| EMP_ID | EMP_NAME  |

+--------+-----------+

|     11 | Fiha      |

+--------+-----------+
```

## STUDENT-LIBRARY DATABASE

**Aim**

To create tables and perform queries in a Student Library scenario

## Database Schema for a Student Library scenario

student(**stud_no : integer**, stud_name: string)

membership(**mem_no: integer**, stud_no: integer)

book(**book_no: integer**, book_name:string, author: string)

iss_rec(**iss_no:integer**, iss_date: date, **mem_no: integer, book_no: integer**)

For the above schema, perform the following-

- Create the tables with the appropriate integrity constraints
- Insert around 5 records in each of the tables
- List all the student names with their membership numbers
- List all the issues for the current date with student and Book names
- List the details of students who borrowed book whose author is ‘Balaguruswamy’
- Give a list of books taken by student with stud_no as 5

a) Create the tables with the appropriate integrity constraints

```
MYSQL>CREATE TABLE STUDENT
     (
           STUD_NO INTEGER(5) PRIMARY KEY,
           STUD_NAME VARCHAR(15)
     );
mysql> desc student;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stud_no   | int         | NO   | PRI | NULL    |       |
| stud_name | varchar(15) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+

MYSQL>CREATE TABLE MEMBERSHIP
     (
           MEM_NO INTEGER(5) PRIMARY KEY,
           STUD_NO INTEGER(5),
           FOREIGN KEY(STUD_NO) REFERENCES STUDENT(STUD_NO)
     );
mysql> desc membership;


+---------+------+------+-----+---------+-------+
| Field   | Type | Null | Key | Default | Extra |
+---------+------+------+-----+---------+-------+
| mem_no  | int  | NO   | PRI | NULL    |       |
| stud_no | int  | YES  | MUL | NULL    |       |
+---------+------+------+-----+---------+-------+

MYSQL>CREATE TABLE BOOK
      (
            BOOK_NO INTEGER(5) PRIMARY KEY,
            BOOK_NAME VARCHAR(20),
            AUTHOR VARCHAR(2)
      );
MYSQL> DESC BOOK;
+-----------+-------------+------+-----+---------+-------+
| FIELD     | TYPE        | NULL | KEY | DEFAULT | EXTRA |
+-----------+-------------+------+-----+---------+-------+
| BOOK_NO   | INT         | NO   | PRI | NULL    |       |
| BOOK_NAME | VARCHAR(20) | YES  |     | NULL    |       |
| AUTHOR    | VARCHAR(40) | YES  |     | NULL    |       |


MYSQL>CREATE TABLE ISS_REC
      (
            ISS_NO INTEGER PRIMARY KEY,
            ISS_DATE DATE,
            MEM_NO INTEGER(5),
            BOOK_NO INTEGER(5),
            FOREIGN KEY(MEM_NO) REFERENCES MEMBERSHIP(MEM_NO),
            FOREIGN KEY(BOOK_NO) REFERENCES BOOK(BOOK_NO)
      );
```

```
mysql> desc iss_rec;
+----------+------+------+-----+---------+-------+
| Field    | Type | Null | Key | Default | Extra |
+----------+------+------+-----+---------+-------+
| ISS_NO   | int  | NO   | PRI | NULL    |       |
| ISS_DATE | date | YES  |     | NULL    |       |
| MEM_NO   | int  | YES  | MUL | NULL    |       |
| BOOK_NO  | int  | YES  | MUL | NULL    |       |
+----------+------+------+-----+---------+-------+
```

b) Insert around 5 records in each of the tables.

```
MYSQL> INSERT INTO  STUDENT VALUES(1,'MANASA'),(2,'KALYANI'),
-> (3,'LAKSHYA'),(4,NANDHINI),(5,'JANANI');
```

```
MYSQL>INSERT INTO MEMBERSHIP
VALUES(101,1),(102,2),(103,3),(104,4),(105,5);
```

```
MYSQL> INSERT INTO BOOK VALUES(1001,'WINGS OF FIRE','APJ '),
(1002,'THE LION','ROLI'),(1003,'SHOW BUSINESS','SHASHI
THAROOR'),(1004,'GENERAL SCIENCE','LUCENT'),(1005,'STAR
WAR','MARK BRAKE'),(1006,'OOP USING C++','BALAGURUSWAMY');
```

```
SELECT * FROM BOOK;
```

```
+---------+-----------------+----------------+
| BOOK_NO | BOOK_NAME       | AUTHOR         |
+---------+-----------------+----------------+
|    1001 | WINGS OF FIRE   | APJ            |
|    1002 | THE LION        | ROLI           |
|    1003 | SHOW BUSINESS   | SHASHI THAROOR |
|    1004 | GENERAL SCIENCE | LUCENT         |
|    1005 | STAR WAR        | MARK BRAKE     |
|    1006 | OOP USING C++   | BALAGURUSWAMY  |
+---------+-----------------+----------------+
```

```
Mysql> INSERT INTO ISS_REC VALUES  (31,'2022-05-10',101,1001),
(32,'2022-08-4',102,1002),
(33,'2023-02->-4',102,1003),
(34,'2022-01-14',105,1003),
(35,'2021-3-24',105,10->05),
(36,'2021-3-24',105,1005);
```

c) List all the student names with their membership numbers

```
MYSQL> SELECT S.STUD_NAME, M.MEM_NO
      FROM STUDENT S, MEMBERSHIP M
      WHERE M.STUD_NO=S.STUD_NO;
      +--------+-----------+
      | mem_no | stud_name |
      +--------+-----------+
      |    101 | manasa    |
      |    102 | kalyani   |
      |    103 | lakshya   |
      |    104 | nandhini  |
      |    105 | janani    |
      +--------+-----------+
```

d) List all the issues for the current date with student and Book names

```
MYSQL> SELECT I.ISS_NO, S.STUD_NAME, B.BOOK_NAME
      FROM ISS_REC I, MEMBERSHIP M, STUDENT S, BOOK B
      WHERE I.MEM_NO=M.MEM_NO AND M.STUD_NO=S.STUD_NO AND
      I.ISS_DATE=CURDATE();

      +-----------+---------------+--------+
      | stud_name | book_name     | iss_no |
      +-----------+---------------+--------+
      | kalyani   | show business |     33 |
      +-----------+---------------+--------+
```

e) List the details of students who borrowed book whose author is ‘Balaguruswamy’

```
MYSQL> SELECT * FROM STUDENT
WHERE STUD_NO
IN(SELECT STUD_NO FROM MEMBERSHIP WHERE MEM_NO
      IN(SELECT MEM_NO FROM ISS_REC WHERE BOOK_NO
            IN(SELECT BOOK_NO FROM BOOK WHERE AUTHOR=’BALAGURUSWAMY’)
      )
);
```

f) Give a list of books taken by student with stud_no as 5

```
MYSQL> SELECT BOOK_NAME
      FROM BOOK
      WHERE BOOK_NO
      IN (SELECT BOOK_NO FROM ISS_REC WHERE MEM_NO
            IN(SELECT MEM_NO FROM MEMBERSHIP WHERE STUD_NO
                  IN(SELECT STUD_NO FROM STUDENT WHERE STUD_NO=5)
            )
```

```
          );
+---------------+
| BOOK_NAME     |
+---------------+
| show business |
| star war      |
+---------------+
```

## MOVIE-RELEASE DATABASE

**Aim**

**Database Schema for** a **movie-release scenario**
To create tables and perform queries in Movie-Release scenario
table 1: movie(mov_no number primary key,mov_title varchar(25) not null, unique
mov_type varchar(20) comedy, action, horror,mov_star varchar(50) not null
mov_price number not null)
table 2: release(rel_no number primary key,mov_no number foreign key,rel_date date not null)

For the above schema, perform the following-
1. Create the above two tables and populate with suitable records.
2. Calculate the average price for each type that has a maximum price of 60,000.
3. Display the details of movie number and movie type released on a particular date.
4. Display the movie name and stars released after 01-04-2015 and price less than 50,000.

1. Create the above two tables and populate with suitable records.

```
MYSQL> CREATE TABLE MOVIE
(
MOV_NO NUMERIC PRIMARY KEY,
MOV_TITLE VARCHAR(25) NOT NULL
     UNIQUE,MOV_TYPE VARCHAR(20) CHECK(MOV_TYPE
IN('COMEDY','ACTION','HORROR')),
MOV_STAR VARCHAR(20) NOT NULL,
MOV_PRICE NUMERIC NOT NULL
);

     MYSQL> DESC MOVIE;
+-----------+---------------+------+-----+---------+-------+
| Field     | Type          | Null | Key | Default | Extra |
+-----------+---------------+------+-----+---------+-------+
| mov_no    | decimal(10,0) | NO   | PRI | NULL    |       |
| mov_title | varchar(25)   | NO   | UNI | NULL    |       |
| mov_type  | varchar(20)   | YES  |     | NULL    |       |
| mov_star  | varchar(20)   | NO   |     | NULL    |       |
| mov_price | decimal(10,0) | NO   |     | NULL    |       |
+-----------+---------------+------+-----+---------+-------+
MYSQL> INSERT INTO MOVIE VALUES
(1,'MOVIE1','HORROR','STAR1',30000) ,
(3,'RAJAMANIKAM','COMEDY','MAMOOTY',450000),
(4,'GOD FATHER','COMEDY','MUKESH',390000);

SELECT * FROM MOVIE;



+--------+-------------+----------+----------+-----------+
| MOV_NO | MOV_TITLE   | MOV_TYPE | MOV_STAR | MOV_PRICE |
+--------+-------------+----------+----------+-----------+
|      1 | MOVIE1      | HORROR   | STAR1    |     30000 |
|      3 | RAJAMANIKAM | COMEDY   | MAMOOTY  |    450000 |
|      4 | GOD FATHER  | COMEDY   | MUKESH   |    390000 |
+--------+-------------+----------+----------+-----------+
MYSQL> CREATE TABLE MOVIERELEASE
(
RELNO NUMERIC PRIMARY KEY,
MOV_NO NUMERIC,REL_DATE DATE NOT NULL,
FOREIGN KEY(MOV_NO) REFERENCES MOVIE(MOV_NO)
);

mysql> DESC MOVIERELEASE;
+----------+---------------+------+-----+---------+-------+
| FIELD    | TYPE          | NULL | KEY | DEFAULT | EXTRA |
+----------+---------------+------+-----+---------+-------+
| RELNO    | DECIMAL(10,0) | NO   | PRI | NULL    |       |
| MOV_NO   | DECIMAL(10,0) | YES  | MUL | NULL    |       |
| REL_DATE | DATE          | NO   |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+

MYSQL> INSERT INTO MOVIERELEASE VALUES
(101,1,'2015-01-23'),(102,4,'2015-04-01'),
(103,3,'2016-03-10');

MYSQL> SELECT* FROM MOVIERELEASE;
+-------+--------+------------+
| RELNO | MOV_NO | REL_DATE   |
+-------+--------+------------+
|   101 |      1 | 2015-01-23 |
|   102 |      4 | 2015-04-01 |
|   103 |      3 | 2016-03-10 |
+-------+--------+------------+
```

2. Calculate the average price for each type that has a maximum price of 60,000.

```
MYSQL> SELECT MOV_TYPE ,AVG(MOV_PRICE) AS AVG_PRICE
FROM MOVIE WHERE MOV_PRICE<=60000 GROUP BY MOV_TYPE
+----------+------------+
          | MOV_TYPE | AVG_PRICE  |
          +----------+------------+
          | HORROR   | 30000.0000 |
          +----------+------------+
```

3. Display the details of movie number and movie type released on a particular date.

```
        MYSQL> SELECT M.MOV_NO ,M.MOV_TYPE FROM MOVIE M JOIN
        MOVIERELEASE R ON M.MOV_NO=R.MOV_NO WHERE R.REL_DATE='2015-01-
        23';
        +--------+----------+
        | mov_no | mov_type |
        +--------+----------+
        |      1 | horror   |
        +--------+----------+
```

4. Display the movie name and stars released after 01-04-2015 and price less than 50,000.

```
        MYSQL> SELECT  M.MOV_TITLE  ,M.MOV_STAR  FROM  MOVIE  M  JOIN
        MOVIERELEASE R ON M.MOV_NO=R.MOV_NO  WHERE M.MOV_PRICE
        <50000 AND R.REL_DATE > '2015-01-04';
        +-----------+----------+
        | MOV_TITLE | MOV_STAR |
        +-----------+----------+
        | MOVIE1    | STAR1    |
        +-----------+----------+
```

## BUS RESERVATION SYSTEM DATABASE

**Aim**

To create tables and perform queries in a bus reservation system application databases
:
BUS (ROUTENO, SOURCE, DESTINATION)
PASSENGER (PID, PNAME, DOB, GENDER)
BOOK_TICKET (PID, ROUTENO, JOURNEY_DATE, SEAT_NO)

For the above schema, perform the following-

- List the details of passengers who have travelled more than three times on the same route
- Remove the passenger records whose date of birth month is April
- Change the jouney_date of passenger 101
- List the details of passengers who have reserved tickets on 06-05-2017

```
MYSQL> CREATE TABLE BUS(
           ROUTENO NUMERIC(10)PRIMARY KEY,
           SOURCE VARCHAR(20),DESTINATION VARCHAR(20));

     SELECT * FROM BUS;
           +---------+---------+----------------+
           | ROUTENO | SOURCE  | DESTINATION    |
           +---------+---------+----------------+
           |      11 | KOLLAM  | PATHANAMTHITTA |
           |      12 | KONNI   | KOODAL         |
           |      13 | CHENNAI | MADURAI        |
           |      14 | KOCHI   | ANGAMALY       |
           |      15 | VAGAMON | MUNNAR         |
           |      16 | GAVI    | PATHANAMTHITTA |
           +---------+---------+----------------+
CREATE TABLE PASSENGER(
           PID NUMERIC PRIMARY KEY,
           PNAME VARCHAR(10),
           DOB DATE NOT NULL,GENDER VARCHAR(20)
           );
```

```
MYSQL> SELECT * FROM PASSENGER;
           +-----+----------+------------+--------+
           | PID | PNAME    | DOB        | GENDER |
           +-----+----------+------------+--------+
           | 101 | KANCHANA | 2022-09-08 | FEMALE |
           | 102 | JACK     | 2000-08-19 | MALE   |
           | 103 | MADHU    | 2001-09-05 | FEMALE |
           | 104 | JOHN     | 2022-09-05 | MALE   |
           | 105 | HAIRA    | 2004-04-09 | FEMALE |
           +-----+----------+------------+--------+
```

```
     CREATE TABLE BOOK_TICKET (PID NUMERIC ,ROUTENO NUMERIC,
     JOURNEY_DATE DATE NOT NULL,
     SEAT_NO NUMERIC,FOREIGN KEY(PID)
     REFERENCES PASSENGER(PID),
FOREIGN KEY (ROUTENO) REFERENCES BUS(ROUTENO));

     mysql> desc book_ticket;
     +--------------+---------------+------+-----+---------+-------+
     | Field        | Type          | Null | Key | Default | Extra |
     +--------------+---------------+------+-----+---------+-------+
     | pid          | decimal(10,0) | YES  | MUL | NULL    |       |
     | routeno      | decimal(10,0) | YES  | MUL | NULL    |       |
     | journey_date | date          | NO   |     | NULL    |       |
     | seat_no      | decimal(10,0) | YES  |     | NULL    |       |
     +--------------+---------------+------+-----+---------+-------+
```

```
INSERT INTO BUS VALUES(11,'KOLLAM','PATHANAMTHITTA'),
           (12,'KONNI','KOODAL'),(13,'CHENNAI','MADURAI'),
      (14,'KOCHI','ANGAMALY'),(15,'VAGAMON','MUNNAR'),(16,'GAVI
      ','PATHANAMTHITTA');

MYSQL> INSERT INTO PASSENGER VALUEs  (101,'KANCHANA','2022-09-
8','FEMALE'),
           (102,'JACK','2000-8-19','MALE'),
           (103,'MADHU','2001-09-5','FEMALE'),
           (104,'JOHN','2022-09-5','MALE'), (105,'HAIRA','2004-
           04-9','FEMALE');



mysql>INSERT    INTO    BOOK_TICKET    VALUES(101,11,'2023-7-
09',0001),(102,12,'2017-05-06',0002),
(103,13,'2016-05-7',0003),(104,14,'2017-05-06',0004),
(105,15,'2016-09-16',0005),
(106,12,'2022-7-09',0001),
(107,12,'2023-7-09',0001);

SELECT * FROM BOOK_TICKET;



+------+---------+--------------+---------+
| Pid  | Routeno | Journey_Date | Seat_No |
+------+---------+--------------+---------+
|  101 |      11 | 2023-07-09   |       1 |
|  102 |      12 | 2017-05-06   |       2 |
|  103 |      13 | 2016-05-07   |       3 |
|  104 |      14 | 2017-05-06   |       4 |
|  105 |      15 | 2016-09-16   |       5 |
+------+---------+--------------+---------+
```

1) List the details of passengers who have travelled more than three times on the same route

```
SELECT P.PID,P.PNAME,P.GENDER  FROM PASSENGER P JOIN
(SELECT PID,ROUTENO FROM BOOK_TICKET GROUP BY PID,
ROUTENO HAVING COUNT(*)>3)
BT ON P.PID=BT.PID;
```

2) Remove the passenger records whose date of birth month is april

```
DELETE FROM BOOKTICKET
WHERE P_ID IN (SELECT P_ID FROM PASSENGER WHERE MONTH(DOB) = 12);
```

**After deleting**

```
mysql>    select    b.pid,p.pname,b.journey_date    from    passenger
p,book_ticket b
-> where p.pid=b.pid;
+------+----------+--------------+
| pid  | pname    | journey_date |
+------+----------+--------------+
|  101 | kanchana | 2017-03-09   |
|  102 | jack     | 2017-05-06   |
|  103 | madhu    | 2016-05-07   |
|  104 | john     | 2017-05-06   |
|  105 | haira    | 2016-09-16   |
+------+----------+--------------+
```

3)Change the journey_date of passenger 101.

```
UPDATE BOOK_TICKET SET JOURNEY_DATE='2017-03-09' WHERE PID='101';
SELECT * FROM BOOK_TICKET;
+------+---------+--------------+---------+
| PID  | ROUTENO | JOURNEY_DATE | SEAT_NO |
+------+---------+--------------+---------+
|  101 |      11 | 2017-03-09   |       1 |
|  102 |      11 | 2017-05-06   |       2 |
|  103 |      13 | 2016-05-07   |       3 |
|  104 |      14 | 2017-05-06   |       4 |
|  105 |      15 | 2016-09-16   |       5 |
+------+---------+--------------+---------+
```

4)List the details of passengers who have reserved tickets on 06-05-2017.

```
SELECT   P.PID,P.PNAME   FROM   PASSENGER   P   JOIN   BOOK_TICKET   BT   ON
P.PID=BT.PID WHERE BT.JOURNEY_DATE='2017-05-06';
+-----+-------+
| PID | PNAME |
+-----+-------+
| 102 | JACK  |
| 104 | JOHN  |
+-----+
```

# CUSTOMER-SALES DATABASE

**Aim**
To create below tables and perform queries in a  Customer and sales

. TABLE 1: CUSTOMER

| COLUMN NAME | DATA TYPE | CONSTRAINTS |
|---|---|---|
| C_NO | Varchar(20) | Primary Key |
| C_NAME | Varchar(20) | Not Null |
| ADDRESS | Varchar(20) | |
| CITY | Varchar(20) | Chennai, Mumbai, Ernakulam |
| BALANCE | Number | |

TABLE 2: SALES

| COLUMN NAME | DATA TYPE | CONSTRAINTS |
|---|---|---|
| ORDER_NO | Varchar(6) | Primary key |
| ORDER_DATE | Date | |
| C_NO | Number | Foreign Key |

Write SQL queries for the following

- Create the above tables and populate with suitable records
- Find the names of all customers having first letter ‘k’ without repetition
- Find the name of customers with ORDER_DATE ‘10-01-2021’.
- Find the order number of all customers whose balance is greater than 5000.

**a)** Create the above tables and populate with suitable records

```
mysql> CREATE TABLE CUSTOMER
           (
           C_NO VARCHAR(6) PRIMARY KEY,
           CNAME VARCHAR(20) NOT NULL,
           ADDRESS VARCHAR(20),
           CITY VARCHAR(20) CHECK (CITY
                IN('CHENNAI','MUMBAI','ERNAKULAM')),
           BALANCE NUMERIC(10)
           );


mysql> DESC CUSTOMER;


           +---------+---------------+------+-----+---------+-------
           +
           | FIELD   | TYPE          | NULL | KEY | DEFAULT | EXTRA
           |
           +---------+---------------+------+-----+---------+-------
           +
           | C_NO    | VARCHAR(6)    | NO   | PRI | NULL    |
           |
           | CNAME   | VARCHAR(20)   | NO   |     | NULL    |
           |
           | ADDRESS | VARCHAR(20)   | YES  |     | NULL    |
           |
           | CITY    | VARCHAR(20)   | YES  |     | NULL    |
           |
           | BALANCE | DECIMAL(10,0) | YES  |     | NULL    |
           |
           +---------+---------------+------+-----+---------+-------
           +


mysql> CREATE TABLE SALE1
(
ORDER_NO VARCHAR(6) PRIMARY KEY,
ORDER_DATE DATE,
C_NO VARCHAR(6),
FOREIGN KEY(C_NO)
REFERENCES CUSTOMER6(C_NO)
);

mysql> DESC SALE1;
+------------+------------+------+-----+---------+-------+
| Field      | Type       | Null | Key | Default | Extra |
+------------+------------+------+-----+---------+-------+
| Order_No   | Varchar(6) | No   | Pri | Null    |       |
| Order_Date | Date       | Yes  |     | Null    |       |
| C_No       | Varchar(6) | Yes  | Mul | Null    |       |
+------------+------------+------+-----+---------+-------+



INSERT INTO CUSTOMER VALUES
(101,'KAIRA','KAIRAVILLA','CHENNAI',5000),
(102,'JACKIE','JACKIEVILLA','CHENNAI',8000),
(103,'MANASA','MANASABHAVANAM','ERNAKULAM',5000),
(104,'SARA','SARA ROSY VILLA','CHENNAI',7000),
(105,'MANU','SREE NILAYAM','ERNAKULAM',4000);

SELECT * FROM CUSTOMER;


+------+--------+-----------------+-----------+---------+
| C_NO | CNAME  | ADDRESS         | CITY      | BALANCE |
+------+--------+-----------------+-----------+---------+
| 101  | KAIRA  | KAIRA VILLA     | CHENNAI   |    5000 |
| 102  | JACKIE | JACKIEVILLA     | CHENNAI   |    8000 |
| 103  | MANASA | MANASA BHAVANAM | ERNAKULAM |    5000 |
| 104  | SARA   | SARA ROSY VILLA | CHENNAI   |    7000 |
| 105  | MANU   | SREE NILAYAM    | ERNAKULAM |    4000 |
+------+--------+-----------------+-----------+---------+

INSERT INTO SALE VALUES
('O1','2020-8-09',101),
('O2','2020-09-10',102),
('O3','2020-08-18',103),
('O4','2021-01-28',104),
('O5','2022-09-23',105);
mysql>SELECT * FROM SALE;
+----------+------------+------+
| order_no | order_date | c_no |
+----------+------------+------+
| o1       | 2020-08-09 | 101  |
| o2       | 2020-09-10 | 102  |
| o3       | 2020-08-18 | 103  |
| o4       | 2021-01-28 | 104  |
| o5       | 2022-09-23 | 105  |
+----------+------------+------+
```

**b)** Find the names of all customers having first letter ‘k’ without repetition

```
mysql>Select cname from customer where cname like 'k%';
+-------+
| CNAME |
+-------+
| KAIRA |
+-------+
```

**c)** Find the name of customers with ORDER_DATE ‘10-01-2021’.

```
mysql> SELECT C.CNAME FROM CUSTOMER C JOIN SALE S ON C.C_NO=S.C_NO
WHERE S.ORDER_DATE='2021-01-10';

+-------+
| CNAME |
+-------+
| SARA   |
+--------+
```

**d)** Find the order number of all customers whose balance is greater than 5000.

```
mysql>  SELECT S.ORDER_NO FROM SALE S JOIN CUSTOMER C ON S.C_NO=C.C_NO
WHERE BALANCE >5000;
+----------+
| ORDER_NO |
+----------+
| O2       |
| O4       |
+----------+
```

## MEDICINE -CUSTOMER DATABASE

**Aim**

To create tables and perform queries in an Medicine Customer scenario.

**Database Schema for a Medicine Customer scenario**

bus (**Med_Id: integer**, med_name: string, price: integer, exp_date : date, company : string)

customer (**c_id : integer**, c_name : string, Med_id : integer, qty_: integer, amount : integer)

For the above schema, perform the following-

- Create the above tables and populate with suitable set of records.
- Retrieve the details of customers with name of medicine which have minimum amount.
- Obtain the details of all medicines whose amount exceeds 500.
- Update the amount by 10% reduction, if amount of purchase greater than 100.

1)Create the above tables and populate with suitable set of records.

```
CREATE TABLE MEDICINE(
MEDID NUMERIC PRIMARY KEY,
MEDNAME VARCHAR (20) NOT NULL, PRICE NUMERIC,
EXPDATE DATE, COMPANY VARCHAR (20)
CHECK(COMPANY IN('CANDILA','CIPLA','BIOCON'))
);

mysql> Desc Medicine;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| Medid   | Decimal(10,0) | No   | Pri | Null    |       |
| Medname | Varchar(20)   | No   |     | Null    |       |
| Price   | Decimal(10,0) | Yes  |     | Null    |       |
| Expdate | Date          | Yes  |     | Null    |       |
| Company | Varchar(20)   | Yes  |     | Null    |       |
+---------+---------------+------+-----+---------+-------+

INSERT INTO MEDICINE VALUES           (101,'MEDICINE1',50,'2022-09-
11','CANDILA'),
(102,'MEDICINE2',65,'2023-12-14','CIPLA'),
(103,'MEDICINE3',1,50,'2022-05-03','BIOCON'),
      (104,'MEDICINE4',40,'2023-06-21','CANDILA'),
      (105,'MEDICINE5',50,'2023-12-06','BIOCON');
SELECT * FROM MEDICINE;

+-------+-----------+-------+------------+---------+
| MEDID | MEDNAME   | PRICE | EXPDATE    | COMPANY |
+-------+-----------+-------+------------+---------+
|   101 | MEDICINE1 |    50 | 2022-09-11 | CANDILA |
|   102 | MEDICINE2 |    65 | 2023-12-14 | CIPLA   |
|   103 | MEDICINE3 |   150 | 2022-05-03 | BIOCON  |
|   104 | MEDICINE4 |    40 | 2023-06-21 | CANDILA |
|   105 | MEDICINE5 |    50 | 2023-12-06 | BIOCON  |
+-------+-----------+-------+------------+---------+

CREATE TABLE CUSTOMER1
(
CNO NUMERIC PRIMARY KEY,
CNAME VARCHAR(20), MEDID NUMERIC,
QTY NUMERIC ,AMOUNT NUMERIC,
FOREIGN KEY(MEDID) REFERENCES MEDICINE(MEDID)
);
MYSQL> DESC CUSTOMER1;
+--------+---------------+------+-----+---------+-------+
| FIELD  | TYPE          | NULL | KEY | DEFAULT | EXTRA |
+--------+---------------+------+-----+---------+-------+
| CNO    | DECIMAL(10,0) | NO   | PRI | NULL    |       |
| CNAME  | VARCHAR(20)   | YES  |     | NULL    |       |
| MEDID  | DECIMAL(10,0) | YES  | MUL | NULL    |       |
| QTY    | DECIMAL(10,0) | YES  |     | NULL    |       |
| AMOUNT | DECIMAL(10,0) | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
INSERT INTO CUSTOMER1 VALUES
(11,'SANDRIA',101,5,200),  (12,'LULU',102,2,100),
(13,'DORA',103,6,250), (14,'SAM',104,2,300),
(15,'PANCHAMI',105,4,400);
SELECT * FROM CUSTOMER1;
+-----+----------+-------+------+--------+
| CNO | CNAME    | MEDID | QTY  | AMOUNT |
+-----+----------+-------+------+--------+
|  11 | SANDRIA  |   101 |    5 |    200 |
|  12 | LULU     |   102 |    2 |    100 |
|  13 | DORA     |   103 |    6 |    250 |
|  14 | SAM      |   104 |    2 |    300 |
|  15 | PANCHAMI |   105 |    4 |    400 |
+-----+----------+-------+------+--------+
```

2) Retrieve the details of customers with name of medicine which have minimum amount.

```
MYSQL> SELECT M.MEDID,M.MEDNAME,M.PRICE,M.EXPDATE FROM MEDICINE M JOIN
CUSTOMER1 C ON M.MEDID = C.MEDID WHERE C.AMOUNT>100;
+-------+-----------+-------+------------+
| MEDID | MEDNAME   | PRICE | EXPDATE    |
+-------+-----------+-------+------------+
|   101 | MEDICINE1 |    50 | 2022-09-11 |
|   103 | MEDICINE3 |   150 | 2022-05-03 |
|   104 | MEDICINE4 |    40 | 2023-06-21 |
|   105 | MEDICINE5 |    50 | 2023-12-06 |
+-------+-----------+-------+------------+
```

3) Obtain the details of all medicines whose amount exceeds 500.

```
MYSQL> SELECT C.CNO,C.CNAME,M.MEDNAME,C.AMOUNT AS MINIMUMAMOUNT
FROM  CUSTOMER1  C  JOIN  MEDICINE  M  ON  C.MEDID=M.MEDID  WHERE
C.AMOUNT= (SELECT MIN(AMOUNT) FROM CUSTOMER1);
+-----+-------+-----------+---------------+
| CNO | CNAME | MEDNAME   | MINIMUMAMOUNT |
+-----+-------+-----------+---------------+
|  12 | LULU  | MEDICINE2 |           100 |
+-----+-------+-----------+---------------+
```

4) Update the amount by 10% reduction, if amount of purchase greater than 100.

```
MYSQL> UPDATE CUSTOMER1 SET AMOUNT =AMOUNT*0.9 WHERE AMOUNT>100;
```

```
TABLE AFTER UPDATION
MYSQL> SELECT * FROM CUSTOMER1;

+-----+----------+-------+------+--------+
| CNO | CNAME    | MEDID | QTY  | AMOUNT |
+-----+----------+-------+------+--------+
|  11 | SANDRIA  |   101 |    5 |    180 |
|  12 | LULU     |   102 |    2 |    100 |
|  13 | DORA     |   103 |    6 |    225 |
|  14 | SAM      |   104 |    2 |    270 |
|  15 | PANCHAMI |   105 |    4 |    360 |
+-----+----------+-------+------+--------+
```

## EMPLOYEE -PROJECT DATABASE

**Aim**

To create below tables and perform queries in a  employee project

**TABLE 1: EMPLOYEE**

| COLUMN NAME | DATA TYPE | CONSTRAINTS |
|---|---|---|

| | | |
|---|---|---|
| E_NUM | Number | Primary key |
| NAME | Varchar(30) | Not Null |
| DESIGNATION | Varchar(30) | Not Null |
| CITY | Varchar(15) | Kolkata, Chennai, Mumbai |
| SALARY | Number | Not Null |

**TABLE 2: PROJECT**

| COLUMN NAME | DATA TYPE | CONSTRAINTS |
|---|---|---|
| P_ID | Number | Primary key |
| E_NUM | Number | Foreign key |

**Write SQL queries for the following**

- Create the above tables and populate with suitable records.
- List the employees whose designation is SYSTEM MANAGER or PROGRAMMER.
- List the name of employees with employee number and project ID.
- Retrieve the E-NUM of employees who have not assigned a project.

a) Create the above tables and populate with suitable records.

```
mysql> create table employee2(
e_num numeric primary key,
name varchar(30) not null,
designation varchar(20) not null,
                city varchar(15) check (city
           in('kolkata','chennai','mumbai')),
salary numeric not null);

desc employee2;
```

```
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| e_num       | decimal(10,0) | NO   | PRI | NULL    |       |
| name        | varchar(30)   | NO   |     | NULL    |       |
| designation | varchar(20)   | NO   |     | NULL    |       |
| city        | varchar(15)   | YES  |     | NULL    |       |
| salary      | decimal(10,0) | NO   |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
mysql>              insert       into       employee2       values
      (1,'mehak','HR','kolkata',40000),(2,'Nayana','system
      manager','kolkata',30000),(3,'soumaya','analyst',
'chennai',30000),(4,'leena','hr','mumbai',50000),
(5,'azeem','System analyst','mumbai',40000);

mysql> select * from employee2;
+-------+---------+----------------+---------+--------+
| e_num | name    | designation    | city    | salary |
+-------+---------+----------------+---------+--------+
|     1 | mehak   | HR             | kolkata |  40000 |
|     2 | Nayana  | system manager | kolkata |  30000 |
|     3 | soumaya | programmer     | chennai |  30000 |
|     4 | leena   | hr             | mumbai  |  50000 |
|     5 | azeem   | System analyst | mumbai  |  40000 |
|     7 | Hima r  | system manager | kolkata |  40000 |
+-------+---------+----------------+---------+--------+
mysql> create table project(pid numeric primary key,
e_num numeric ,foreign key(e_num)
references   employee2(e_num));
mysql> desc project;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| pid   | decimal(10,0) | NO   | PRI | NULL    |       |
| e_num | decimal(10,0) | YES  | MUL | NULL    |       |
+-------+---------------+------+-----+---------+-------+

insert into project values
(11,1),(12,2),(13,3),(14,4),(15,5);

mysql> select * from project;
```

```
+-----+-------+
| pid | e_num |
+-----+-------+
|  11 |     1 |
|  12 |     2 |
|  13 |     3 |
|  14 |     4 |
|  15 |     5 |
+-----+-------+
```

b)List the employees whose designation is SYSTEM MANAGER or PROGRAMMER.

```
mysql> select * from employee2 where designation  in('system
manager','analyst');
+-------+---------+----------------+---------+--------+
| e_num | name    | designation    | city    | salary |
+-------+---------+----------------+---------+--------+
|     2 | Nayana  | system manager | kolkata |  30000 |
|     3 | soumaya | analyst        | chennai |  30000 |
+-------+---------+----------------+---------+--------+
```

c) List the name of employees with employee number and project ID.

```
mysql> select e.name,e.e_num,p.pid from employee2 e left join
project p on e.e_num =p.e_num;
+---------+-------+------+
| name    | e_num | pid  |
+---------+-------+------+
| mehak   |     1 |   11 |
| Nayana  |     2 |   12 |
| soumaya |     3 |   13 |
| leena   |     4 |   14 |
| azeem   |     5 |   15 |
+---------+-------+------+
```

d)Retrieve the E-NUM of employees who have not assigned a project.

```
mysql> select e_num from employee2 where e_num not in (select
e_num from project );

+-------+

| e_num |

+-------|      7 |
```

## CUSTOMER-INVOICE DATABASE

**Aim**

To create below tables and perform queries in  customer invoice

**TABLE 1: CUSTOMER**

| COLUMN NAME | DATA TYPE | CONSTRAINTS |
|---|---|---|
| C_ID | Number | Primary key |
| C_NAME | Varchar(15) | Not null |
| AREA | Varchar(15) | Aluva, Ernakulam, Kothamangalam |
| PHONE_NO | Number | |

**TABLE 2: INVOICE**

| COLUMN NAME | DATA TYPE | CONSTRAINTS |
|---|---|---|
| INV_NO | Number | Primary key |
| C_ID | Number | Foreign key |
| INV_DATE | Date | Not null |

**Write SQL queries for the following**

- Create the above two tables and populate with suitable records.
- Find the names of all customers who have been issued an invoice.
- Find the INV_DATE for the customer 'MATHEW'.
- Change the INV_DATE of customer 'John' to 28-01-2021

a) Create the above two tables and populate with suitable records.

```
create table customer4

(

c_id numeric primary key,

c_name varchar(15) not null,

area varchar(15) check (area
      in('aluva','ernakulam','kothamangalam')),

phone_no numeric(30)

);

mysql> DESC CUSTOMER4;

+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| c_id     | decimal(10,0) | NO   | PRI | NULL    |       |
| c_name   | varchar(15)   | NO   |     | NULL    |       |
| area     | varchar(15)   | YES  |     | NULL    |       |
| phone_no | decimal(10,0) | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
insert into customer4 values
(1,'manju','aluva',9456677898),
(2,'kinjal','ernakulam',9876543209),
(3,'kanaka','kothamangalam',9874655366),
(4,'haya','ernakulam','9887676556');

mysql> select * from customer4;
+------+--------+---------------+------------+
| c_id | c_name | area          | phone_no   |
+------+--------+---------------+------------+
|    1 | manju  | aluva         | 9456677898 |
|    2 | kinjal | ernakulam     | 9876543209 |
|    3 | kanaka | kothamangalam | 9874655366 |
|    4 | haya   | ernakulam     | 9887676556 |
|    5 | mathew | ernakulam     | 9887676556 |
+------+--------+---------------+------------+



create table invoice
(
inv_no numeric primary key,
c_id numeric,

inv_date date not null,

foreign key(c_id) references customer4(c_id)

);

mysql> desc invoice ;

+----------+--------------+-----+-----+--------+-------+

| Field    | Type         | Null | Key | Default | Extra |

+----------+--------------+-----+-----+--------+-------+

| inv_no   | decimal(10,0) | NO   | PRI | NULL    |       |

| c_id     | decimal(10,0) | YES  | MUL | NULL    |       |

| inv_date | date         | NO   |     | NULL    |       |

+----------+--------------+-----+-----+--------+-------+

mysql>    INSERT    INTO    INVOICE    VALUES(11,1,'2021-07-
2'),(12,2,'2022-08-3'),

(13,3,'2022-07-6'),(14,4,'2022-05-3');

MYSQL> SELECT * FROM INVOICE;

+--------+------+------------+

| INV_NO | C_ID | INV_DATE   |

+--------+------+------------+

|     11 |    1 | 2021-07-02 |

|     12 |    2 | 2022-08-03 |

|     13 |    3 | 2022-07-06 |

|     14 |    4 | 2022-05-03 |

|     15 |    5 | 2021-01-02 |

+--------+------+------------+
```

b) Find the names of all customers who have been issued an invoice.

```
mysql>   SELECT C.C_NAME FROM CUSTOMER4 C ,
INVOICE I WHERE I.C_ID=C.C_ID;
+--------+
| C_NAME |
+---------+
| MANJU  |
| KINJAL |
| KANAKA |
| HAYA   |
+---------+
```

c) Find the INV_DATE for the customer ‘MATHEW’.

```
mysql>SELECT INV_DATE FROM INVOICE WHERE
C_ID=(SELECT C_ID FROM CUSTOMER4 WHERE C_NAME='MATHEW');
+------------+
| INV_DATE   |
+------------+
| 2021-01-02 |
+------------
```

d ) Change the INV_DATE of customer ‘John’ to 28-01-2021

```
Mysql>  Update  Invoice  Set  Inv_Date='2021-01-29'  Where
C_Id=(Select C_Id From Customer4 Where C_Name='Manju');
```

**After Updation**

```
Mysql> Select * From Invoice;
+--------+------+------------+
| Inv_No | C_Id | Inv_Date   |
+--------+------+------------+
|     11 |    1 | 2021-01-29 |
|     12 |    2 | 2022-08-03 |
|     13 |    3 | 2022-07-06 |
|     14 |    4 | 2022-05-03 |
|     15 |    5 | 2021-01-02 |
```

## EMPLY-DEPT DATABASE

**Aim**

To create tables and perform queries in an Emply Dept scenario.

**Database Schema for a Emply Dept scenario**

dept (**deptcode : integer**, deptname : string)

emply (**Empcode : varchar**, empname : string, address: string, age : string, deptcode : varchar)

For the above schema, perform the following-

a) Display records from EMPLY table for employees whose age is between 25 and 45.

b)Retrieve the Deptcode and total no of employees in each department.

c)Retrieve Empcode, empname, address, deptcode for all employees in
"account" and "stock" departments.
d)Display average, maximum and minimum age of employees.
e)Delete all records belonging to research department in the EMPLY table

```
mysql> CREATE TABLE EMPLY1(
EMPCODE VARCHAR(20) PRIMARY KEY,
EMPNAME VARCHAR(50) ,
ADDRESS VARCHAR(100),
AGE NUMERIC,
DEPT_CODE VARCHAR(30),
FOREIGN KEY (DEPT_CODE) REFERENCES
DEPT1(DEPTCODE));

DESC EMPLY1;
+-----------+--------------+------+-----+---------+-------+
| FIELD     | TYPE         | NULL | KEY | DEFAULT | EXTRA |
+-----------+--------------+------+-----+---------+-------+
| EMPCODE   | VARCHAR(5)   | NO   | PRI | NULL    |       |
| EMPNAME   | VARCHAR(50)  | YES  |     | NULL    |       |
| ADDRESS   | VARCHAR(100) | YES  |     | NULL    |       |
| AGE       | INT          | YES  |     | NULL    |       |
| DEPT_CODE | VARCHAR(5)   | YES  | MUL | NULL    |       |
+-----------+--------------+------+-----+---------+-------+

MYSQL> INSERT INTO EMPLY1 VALUES
('E101','ANJALY','ANJALY NIVAS',25,'D301'),
('E102','BOBBY','ALAPUZHA',25,'D305'),
('E103','ARAVIND','CHENNAI',31,'D305'),
('E104','LAKSHMI','MANNAR',55,'D707'),
('E105','DAISY','CHAITHRAM ANGAMALY',35,'D707'),
('E106','ESHA','MUMBAI',23,'D707'),
('E107','GEORGY','PALA',45,'D909'),
('E108','PRAKASH','VENNIKULAM',36,'D110'),
('E109','MADHAVAN','MYNAKUM ,KOTTAYAM',46,'D202'),
('E110','ANUGRAHA','APRNA ANGAMALY',47,'D301'),
('E111','DEVA','TRICHY',38,'D301'),
('E112','SAJU','DHANYA,ERNAKULAM',27,'D202'),
('E113','PRIYESH','PRIYA NIVAS,KOTTAYAM',26,'D302');


MYSQL> SELECT * FROM EMPLY1;
+---------+----------+----------------------+------+-----------+
| EMPCODE | EMPNAME  | ADDRESS              | AGE  | DEPT_CODE |
+---------+----------+----------------------+------+-----------+
| E101    | ANJALY   | ANJALY NIVAS         |   25 | D301      |
| E102    | BOBBY    | ALAPUZHA             |   25 | D305      |
| E103    | ARAVIND  | CHENNAI              |   31 | D305      |
| E104    | LAKSHMI  | MANNAR               |   55 | D707      |
| E105    | DAISY    | CHAITHRAM ANGAMALY   |   35 | D707       |
| E106    | ESHA     | MUMBAI               |   23 | D707       |
| E107    | GEORGY   | PALA                 |   45 | D909       |
| E108    | PRAKASH  | VENNIKULAM           |   36 | D110       |
| E109    | MADHAVAN | MYNAKUM ,KOTTAYAM    |   46 | D202       |
| E110    | ANUGRAHA | APRNA ANGAMALY       |   47 | D301       |
| E111    | DEVA     | TRICHY               |   38 | D301       |
| E112    | SAJU     | DHANYA,ERNAKULAM     |   27 | D202       |
| E113    | PRIYESH  | PRIYA NIVAS,KOTTAYAM |   26 | D302       |
+---------+----------+----------------------+------+------------+
```

```
mysql> CREATE TABLE DEPT1(
DEPTCODE VARCHAR(20) PRIMARY KEY,
DEPTNAME VARCHAR(30));

MYSQL> DESC DEPT1;

+----------+-------------+------+-----+---------+-------+

| FIELD    | TYPE        | NULL | KEY | DEFAULT | EXTRA |

+----------+-------------+------+-----+---------+-------+

| DEPTCODE | VARCHAR(20) | NO   | PRI | NULL    |       |

| DEPTNAME | VARCHAR(30) | YES  |     | NULL    |       |

+----------+-------------+------+-----+---------+-------+


INSERT INTO DEPT1 VALUES('D301','SALES'),

('D302','ACCOUNT'),('D707','RESEARCH'),

('D909','ADVERTISING'),('D202','STOCK'),

('D110','COMPUTER'),('D305','MARKETING');

MYSQL> SELECT * FROM DEPT1;

+----------+-------------+

| DEPTCODE | DEPTNAME    |

+----------+-------------+

| D110     | COMPUTER    |

| D202     | STOCK       |

| D301     | SALES       |

| D302     | ACCOUNT     |
| D305     | MARKETING   |
| D707     | RESEARCH    |
| D909     | ADVERTISING |
+----------+-------------+
```

a) Display records from EMPLY table for employees whose age is between 25 and 45.

```
SELECT * FROM EMPLY1 WHERE AGE BETWEEN 25 AND 45;
+---------+---------+----------------------+------+-----------+
| empcode | empname | address              | age  | dept_code |
+---------+---------+----------------------+------+-----------+
| e101    | anjaly  | anjaly nivas         |   25 | d301      |
| e102    | bobby   | alapuzha             |   25 | d305      |
| e103    | aravind | chennai              |   31 | d305      |
| e105    | daisy   | chaithram angamaly   |   35 | d707      |
| e107    | Georgy  | pala                 |   45 | d909      |
| e108    | prakash | vennikulam           |   36 | d110      |
| e111    | deva    | trichy               |   38 | d301      |
| e112    | saju    | dhanya,Ernakulam     |   27 | d202      |
| e113    | priyesh | priya nivas,kottayam |   26 | d302      |
+---------+---------+----------------------+------+-----------+
```

b)Retrieve the Deptcode and total no of employees in each department.

```
mysql> select dept_code ,count(empcode) as totalemployess from
emply1 group by dept_code;
```

```
+-----------+----------------+
| dept_code | totalemployess |
+-----------+----------------+
| d110      |              1 |
| d202      |              2 |
| d301      |              3 |
| d302      |              1 |
| d305      |              2 |
| d707      |              3 |
| d909      |              1 |
+-----------+---------------+
```

c)Retrieve Empcode, empname, address, deptcode for all employees in "account" and "stock" departments.

```
SELECT Empcode, Empname, Address, Dept_Code
FROM EMPLY1
WHERE Dept_Code IN ('D302', 'D202');
```

```
+---------+----------+---------------------+-----------+
| Empcode | Empname  | Address             | Dept_Code |
+---------+----------+---------------------+-----------+
| e109    | Madhavan | mynakum ,kottayam   | d202      |
| e112    | saju     | dhanya,Ernakulam    | d202      |
| e113    | priyesh  | priya nivas,kottayam | d302      |
+---------+----------+---------------------+-----------+
```

d)Display average, maximum and minimum age of employees.

```
mysql> SELECT AVG(Age) AS AverageAge, MAX(Age) AS MaxAge,
MIN(Age) AS MinAge
FROM EMPLY1;
+------------+--------+--------+
| AverageAge | MaxAge | MinAge |
+------------+--------+--------+
|    35.3077 |     55 |     23 |
+------------+--------+--------+
```

e)Delete all records belonging to research department in the EMPLY table

```
MYSQL> DELETE FROM EMPLY1 WHERE DEPT_CODE IN(SELECT DEPTCODE FROM
DEPT1 WHERE DEPTNAME='RESEARCH');
```

**After delete**

```
MYSQL> SELECT * FROM EMPLY1;

+---------+----------+---------------------+------+-----------+
| EMPCODE | EMPNAME  | ADDRESS              | AGE  | DEPT_CODE |
+---------+----------+----------------------+------+-----------+
| E101    | ANJALY   | ANJALY NIVAS         |   25 | D301      |
| E102    | BOBBY    | ALAPUZHA             |   25 | D305      |
| E103    | ARAVIND  | CHENNAI              |   31 | D305      |
| E107    | GEORGY   | PALA                 |   45 | D909      |
| E108    | PRAKASH  | VENNIKULAM           |   36 | D110      |
| E109    | MADHAVAN | MYNAKUM ,KOTTAYAM    |   46 | D202      |
| E110    | ANUGRAHA | APRNA ANGAMALY       |   47 | D301      |
| E111    | DEVA     | TRICHY               |   38 | D301      |
| E112    | SAJU     | DHANYA,ERNAKULAM     |   27 | D202      |
| E113    | PRIYESH  | PRIYA NIVAS,KOTTAYAM |   26 | D302      |
+---------+----------+----------------------+------+-----------+
```

## Final Notes

Thank you for reading through this guide. These notes are the result of continuous learning, practice, and effort. Whether you're just starting or revising for interviews or exams, I hope this guide helps you build a strong foundation in SQL.

### *Final Suggestion:*

"*Practice consistently, understand the logic behind queries, and don't be afraid to make mistakes—every error is a step closer to mastery. Share your success with others and keep learning!*"

*If you find this helpful follow for more:*

*www.linkedin.com/in/aliya-jabbar*