1. a) Explain about different notations used in E-R Model.

The Entity-Relationship (E-R) model is a popular conceptual modeling technique used in database design. It

helps in visualizing and representing the structure and relationships between entities within a system. The E-R

model uses various notations to depict different aspects of the model. Here are some commonly used notations:

Entities: In the E-R model, an entity represents a real-world object or concept. It is depicted as a rectangle with

rounded corners, and the entity name is written inside the rectangle.

Attributes: Attributes define the properties or characteristics of an entity. They provide additional information

about the entity. Attributes are represented as ovals or ellipses connected to their respective entities. The attribute

name is written inside the oval.

Primary Key: A primary key is a unique identifier for an entity. It uniquely identifies each instance of the entity

and ensures data integrity. In the E-R model, a primary key is underlined within an attribute.

Relationships: Relationships represent the associations or connections between entities. They describe how

entities relate to each other. Relationships are depicted as diamonds or rhombuses, and they are connected to the

related entities using lines.

Cardinality and Participation: Cardinality indicates the number of instances of one entity that can be associated

with the instances of another entity in a relationship. It specifies the minimum and maximum number of

occurrences. Common notations for cardinality include "1" for one occurrence, "0..1" for zero or one occurrence,

"0.." for zero or more occurrences, and "1.." for one or more occurrences.

Relationship Attributes: Sometimes, a relationship can have attributes of its own. These attributes provide

additional information about the relationship itself. Relationship attributes are represented as ovals or ellipses connected to the relationship line.

Composite Attributes: Composite attributes are attributes that can be further divided into sub-attributes. They

are represented as nested ellipses within the attribute oval.

Derived Attributes: Derived attributes are attributes whose values can be derived or calculated based on other

attributes or entities. They are depicted using dashed ellipses or ovals.

Weak Entities: Weak entities depend on another entity for their existence. They are represented by double

rectangles, and their relationship with the identifying entity is denoted using a solid line with a diamond at the

weak entity end.

Inheritance: Inheritance allows entities to inherit attributes and relationships from a higher-level entity called a

superclass or parent entity. In the E-R model, inheritance is depicted using a solid line with a triangle at the

superclass end.

These are some of the common notations used in the E-R model. However, it's important to note that there are

variations and extensions to these notations based on specific modeling techniques and tools.

1.b) Design ER diagram for EMPLOYEE – DEPARTMENT database

To design an ER diagram for an EMPLOYEE - DEPARTMENT database, we need to identify the entities, their

attributes, and the relationships between them. Here's a simplified example:

Entities:

1. Employee:

EmployeeID (Primary Key)

FirstName

LastName

Position

Salary

2. Department:

DepartmentID (Primary Key)

DepartmentName

Location

Relationships:

1. Works_In (One-to-Many):

• An employee works in one department, but a department can have multiple employees.

• EmployeeID is the foreign key in the Department entity.

• DepartmentID is the foreign key in the Employee entity.

2. Manages (One-to-One):

• An employee manages one department, and a department is managed by one employee.

• EmployeeID is the foreign key in the Department entity.

```
+-----------------------+
| Employee |
+-----------------------+
| EmployeeID (PK) |
| FirstName |
| LastName |
| Position |
| Salary |
+-----------------------+
|
|
| (One-to-One)
|
+-----------------------+
| Manages |
+-----------------------+
| EmployeeID (FK, PK) |
+-----------------------+
|
|
| (One-to-Many)
|
+------------------------+
| Department |
+------------------------+
| DepartmentID (PK) |
| DepartmentName |
| Location |
| EmployeeID (FK) |
+------------------------+
```

In addition to the basic components of entities, attributes, and relationships, the Entity-Relationship (E-R) model

supports additional features that enhance its ability to represent complex data structures and constraints. Here are

some additional features of the E-R model:

Weak Entities: A weak entity is an entity that depends on another entity for its existence. It cannot be uniquely

identified by its attributes alone. A weak entity is identified by its relationship with a strong entity, known as the

identifying or owner entity. The weak entity's primary key includes the primary key of the owner entity as well as

its own attributes.

Aggregation: Aggregation allows you to treat a group of entities and relationships as a higher-level entity. It

represents a whole-part relationship between entities. Aggregation is denoted by a diamond-shaped symbol

connected to the participating entities. It helps in modeling complex relationships and hierarchical structures.

Multivalued Attributes: Multivalued attributes are attributes that can have multiple values for a single entity instance. For example, an employee entity can have multiple phone numbers. Multivalued attributes are

represented by double ovals or ellipses connected to the entity.

Subtypes and Supertypes (Generalization/Specialization): Subtypes and supertypes are used to model

inheritance relationships between entities. They allow you to define specialized entities based on a common

superclass entity. The superclass represents the common attributes and relationships, while the subtypes represent

the specific attributes and relationships unique to each subtype.

Recursive Relationships: Recursive relationships occur when an entity participates in a relationship with itself.

For example, in a company organizational structure, an employee may have a "Reports To" relationship with

another employee. Recursive relationships are represented by a line connecting an entity to itself, and a role or

attribute may be used to distinguish between different instances of the entity.

Constraints: Constraints are rules that define the integrity and behavior of the database. The E-R model allows

you to represent various constraints, such as uniqueness, cardinality, participation, and referential integrity.

Constraints ensure that the data stored in the database is accurate and consistent.

Derived Attributes: Derived attributes are attributes whose values are calculated or derived from other attributes

or entities. They are not stored in the database but can be computed when needed. Derived attributes are

represented using dashed ellipses or ovals.

These additional features of the E-R model provide flexibility and expressiveness in capturing complex data

structures, relationships, and constraints. They help in designing databases that accurately represent real-world

scenarios and support efficient data management and retrieval.

2.b) Design ER diagram for HOSPITAL database

To design an ER diagram for a HOSPITAL database, we need to identify the entities, their attributes, and the relationships between them. Here's a simplified example:

Entities:

1. Patient:

PatientID (Primary Key)

FirstName

LastName

Gender

DateOfBirth

ContactNumber

2. Doctor:

DoctorID (Primary Key)

FirstName

LastName

Specialization

ContactNumber

3.Nurse:

NurseID (Primary Key)

FirstName

LastName

Department

ContactNumber

4. Department:

DepartmentID (Primary Key)

DepartmentName

Location

5. Appointment:

AppointmentID (Primary Key)

PatientID (Foreign Key)

DoctorID (Foreign Key)

Date

Time

6. Prescription:

PrescriptionID (Primary Key)

PatientID (Foreign Key)

DoctorID (Foreign Key)

Medication

Dosage

Instructions

Relationships:

1. Admitted (Many-to-Many):

A patient can be admitted to multiple departments, and a department can have multiple patients admitted.

A junction table is used to represent this many-to-many relationship.

PatientID is the foreign key in the junction table.

DepartmentID is the foreign key in the junction table.

2. Assigned (One-to-Many):

A doctor is assigned to one department, but a department can have multiple doctors assigned to it.

DoctorID is the foreign key in the Department entity.

3. Attends (One-to-Many):

A patient attends multiple appointments, but an appointment is attended by one patient.

PatientID is the foreign key in the Appointment entity.

4. Attends (One-to-Many):

A doctor attends multiple appointments, but an appointment is attended by one doctor.

DoctorID is the foreign key in the Appointment entity.

Here's the ER diagram representation:

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

 | Patient |

 +-----------------------+
| PatientID (PK) |

 | FirstName |

 | LastName |

 | Gender |

 | DateOfBirth |

 | ContactNumber |

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

|

 |

 | (One-to-Many)

 |

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

 | Appointment |

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

 | AppointmentID (PK) |

 | PatientID (FK) |

 | DoctorID (FK) |

 | Date |

 | Time |

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

 |

 |

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

 | Prescription |

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

 | PrescriptionID (PK) |

 | PatientID (FK) |

 | DoctorID (FK) |

 | Medication |
 | Dosage |

 | Instructions |

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

 |

 |

 (Many-to-Many)

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

 | Admitted |

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

 | PatientID (FK) |

 | DepartmentID (FK) |

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

 |

 |

 |

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

 | Department |

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

 | DepartmentID (PK) |

 | DepartmentName |

 | Location |

 | DoctorID (FK) |

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

 |

 |

 | (One-to-Many)

 |

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

 | Doctor |
```

3.a)Explain logical database design for Employee-Department database.

Logical database design for an Employee-Department database involves translating the conceptual design, which

includes the Entity-Relationship (E-R) model, into a more structured and detailed representation of the database

schema. Here's an explanation of the logical design components for the Employee-Department database:

1.Entity Mapping: Each entity in the E-R model becomes a table in the logical design. For example, the

"Employee" entity becomes the "Employees" table, and the "Department" entity becomes the "Departments"

table.

2. Table Columns:Attributes of each entity become columns in the respective tables. For example, the "Employee"

entity's attributes like EmployeeID, FirstName, LastName, Position, and Salary become columns in the

"Employees" table.

3. Primary Keys: Identify the primary key for each table. The primary key uniquely identifies each row in the table

and helps maintain data integrity. For example, the "Employees" table can have EmployeeID as the primary key,

and the "Departments" table can have DepartmentID as the primary key.

4. Foreign Keys: Determine the relationships between entities and identify foreign keys. Foreign keys establish

relationships between tables by referencing the primary key of another table. For example, the "Employees" table

can have a foreign key DepartmentID referencing the primary key DepartmentID in the "Departments" table.

5.Relationship Mapping: Represent the relationships between entities in the E-R model using appropriate

techniques like one-to-many, many-to-many, or one-to-one mapping in the logical design. For example, a one-to-

many relationship between employees and departments can be implemented by having the foreign key

DepartmentID in the "Employees" table.

6. Constraints and Validation: Define constraints and validations to ensure data integrity and enforce business

rules. For example, you can enforce referential integrity by setting foreign key constraints that ensure the values

in the foreign key column match the values in the referenced primary key column.

7. Indexes:Identify columns that require indexing for efficient data retrieval. Indexes improve query performance

by allowing quick lookup based on indexed columns.

8. Normalization: Apply normalization techniques to eliminate redundancy and ensure data integrity.

Normalization involves breaking down larger tables into smaller, well-structured tables to minimize data

duplication and dependency anomalies.

The logical database design focuses on creating a clear and structured representation of the database schema,

defining tables, columns, primary and foreign keys, relationships, and constraints. It serves as a foundation for the

subsequent steps of physical database design, which involve choosing storage structures, defining data types, and

optimizing performance.

3b) Write step-by-step procedure for MySQL Software installation and Demonstrate it

on a Desktop.

step-by-step procedure for installing MySQL software on a desktop computer:

1. Download MySQL Installer: Visit the official MySQL website

(https://dev.mysql.com/downloads/installer/) and download the MySQL Installer appropriate for

your operating system (Windows, macOS, or Linux).

2. Launch the Installer: Once the installer is downloaded, run the executable file to launch the

MySQL Installer.

3. Choose Installation Type: In the MySQL Installer, select the installation type. For a typical

installation, choose "Developer Default" to install MySQL Server, connectors, and other

development tools. You can also select "Server Only" if you only need MySQL Server.

4. Select Products: In the product selection screen, choose the products you want to install. Typically,

you would select "MySQL Server" and other related products like "MySQL Workbench" for database management.

5. Version Selection: Choose the version of MySQL Server you wish to install. It is recommended to select the latest stable version.

6. Installation: Configure the installation options, including the installation directory, and click the "Execute" button to start the installation process.

7. Setup Type: Select the setup type. For a new installation, choose "Developer Default" or "Server Only" based on your selection in step 3.

8. Configuration: Configure the MySQL Server settings, including the server type (standalone or cluster), port number, authentication method, and root user password. You can also choose to install MySQL as a Windows service for automatic startup.

9. Installation Progress: The installer will proceed with the installation, displaying the progress and installing the selected products.

10. Complete Installation: Once the installation is complete, you will see a screen indicating the successful installation. You can choose to launch MySQL Workbench or other tools from this screen.

4.a) Create the employee table using following schema.

Employee(eid int,name varchar(10),salary float);

```
mysql> create table Employee(eid int,name varchar(10),salary float);
Query OK, 0 rows affected .
```

b) write an SQL query to add a new column gender.

```
mysql> alter table Employee add gender varchar(5);
mysql> desc employee;
+--------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| eid | int | YES | | NULL | |
| name | varchar(10) | YES | | NULL | |
| salary | float | YES | | NULL | |
| gender | varchar(5) | YES | | NULL | |
+--------+-------------+------+-----+---------+-------+
```

c)write an SQL query to change the name of the salary column as 'esal'

```
mysql> alter table employee change salary esal integer;

mysql> desc employee;

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

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

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

| eid | int | YES | | NULL | |

| name | varchar(10) | YES | | NULL | |

| esal | int | YES | | NULL | |

| gender | varchar(5) | YES | | NULL | |

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

d)write an SQL query to modify data type of the name column to varchar(30)

```
mysql> alter table employee modify name varchar(30);

mysql> desc employee;

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

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

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

| eid | int | YES | | NULL | |

| name | varchar(30) | YES | | NULL | |

| esal | int | YES | | NULL | |

| gender | varchar(5) | YES | | NULL | |

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

e)Write an SQL query to change the name of the table from employee to employee_details.

```
MySQL> alter table employee rename to employee_details;
```

f) Write an SQL query to remove all the rows from the employee table using DDL command.

```
MySQL> select *from Employee_details;

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

| eid | name | esal | gender |

+------+-------+-------+--------+
| 121 | vasu | 10000 | M |

| 122 | ramya | 50000 | F |

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

2 rows in set (0.00 sec)

MySQL> delete from employee_details;
```

Or

```
MySQL> truncate table employee_details;

Query OK, 2 rows affected (0.01 sec)

MySQL> select *from Employee_details;

Empty set
```

g.Write an SQL query to remove the employee_details table from the database.

```
MySQL> drop table employee_details;
```

5.a) Create the Book table using following schema.

Book(bid int, title varchar(10),pages int);

```
MySQL> create table Book(bid int, title varchar(10),pages int);

Query OK, 0 rows affected (0.03 sec)
```

b) write an SQL query to add a new column price.

```
MySQL> alter table book add price varchar(10);
```

c)write an SQL query to change the name of the title column as ‘btitle’

```
MySQL> alter table book change title btitle varchar(20);

Query OK, 0 rows affected (0.02 sec)

MySQL> desc book;

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

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

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

| bid | int | YES | | NULL | |

| btitle | varchar(20) | YES | | NULL | |

| pages | int | YES | | NULL | |

| price | varchar(10) | YES | | NULL | |
+--------+-------------+------+-----+---------+-------+
```

d)write an SQL query to modify data type of the title column to varchar(30)

```
MySQL> alter table book modify btitle varchar(30);
```

e)Write an SQL query to change the name of the table from Book to Book_details.

```
MySQL> alter table book rename to book_details;
```

f) Write an SQL query to remove all the rows from the Book table using DDL command.

```
MySQL> truncate table book_details;
```

Or

```
MySQL> delete table book_details;
```

g.Write an SQL query to remove the Book table from the database

```
MySQL>drop table book_details;
```

6.a) Create the employee table using following schema. Make sure that eid column doesnot allow null values.

Employee(eid int,name varchar(0),salary float);

```
MySQL> create table Employee(eid int primary key,name varchar(10),salary float);
```

b) Create the department table using following schema. Make sure that budget should be more than 50000 for all

records.

department(did int,dname varchar(20),budget float);

```
MySQL> CREATE TABLE department (did INT,dname VARCHAR(20),budget FLOAT, CHECK(budget >

50000) );

Query OK, 0 rows affected
```

c) Create the student table using following schema. Make sure that sid column doesnot allow duplicate values.

student(sid int,name varchar(20),age int);

```
MySQL> CREATE TABLE student(sid INT PRIMARY KEY,name VARCHAR(20),age INT);
```

d) create sailors, boats, reserves tables using the following fields.

Note :sid is the primary key for sailors table

bid is the primary key for boats table

sid,bid is foreign keys in reserves table

Sailors(sid,sname,rating,age);

Boats(bid,bname,color);

Reserves(sid,bid,day);

```
mysql> CREATE TABLE sailors (
-> sid INT PRIMARY KEY,
-> sname VARCHAR(20),
-> rating INT,
-> age INT
-> );
Query OK, 0 rows affected (0.02 sec)
mysql> CREATE TABLE boats (
-> bid INT PRIMARY KEY,
-> bname VARCHAR(20),
-> color VARCHAR(20)
-> );
Query OK, 0 rows affected (0.02 sec)
mysql> CREATE TABLE reserves (
-> sid INT,
-> bid INT,
-> day DATE,
-> PRIMARY KEY (sid, bid),
-> FOREIGN KEY (sid) REFERENCES sailors(sid),
-> FOREIGN KEY (bid) REFERENCES boats(bid)
-> );
Query OK, 0 rows affected (0.04 sec)
```

7.a) Create the student table using following schema. Make sure that sid column doesnot allow null values.

student(sid int,name varchar(20),age int);

```
MySQL> CREATE TABLE student1(
-> sid INT NOT NULL,

-> name VARCHAR(20),

> age INT,

-> PRIMARY KEY (sid)

-> );
```

b) Create the voters table using following schema. Make sure that only records with more than 18 years age stored

in the table.

Voters(voter_id int, voter_name varchar(20),voter_age int);

```
MySQL> CREATE TABLE voters (

-> voter_id INT,

-> voter_name VARCHAR(20),

-> voter_age INT CHECK (voter_age > 18)

-> );
```

c) Create the employee table using following schema. Make sure that if the user does not provide a value for salary,

20000 need to be stored in the table.

Employee(eid int,name varchar(20),salary float);

```
MySQL> CREATE TABLE employee1(

-> eid INT,

-> name VARCHAR(20),

-> salary FLOAT DEFAULT 20000

-> );
```

8.a) Create the department table using following schema. Make sure that did column doesnot allow null values.

department(did int,dname varchar(20),budget float);

```
MySQL> CREATE TABLE department1(

-> did INT NOT NULL,

-> dname VARCHAR(20),

-> budget FLOAT,
-> PRIMARY KEY (did)
-> );
```

b) Create the SSC_Student table using following schema. Make sure that age of every SSC student more than or

equals to 15 years.

SSC_Student(sid int,name varchar(20),age int);

```
MySQL> CREATE TABLE SSC_Student (
-> sid INT,
-> name VARCHAR(20),
-> age INT CHECK (age >= 15)
-> );
Query OK, 0 rows affected
```

c) Create the employee table using following schema. Make sure that if the user does not provide a value for salary,

10000 need to be stored in the table.

Employee(eid int,name varchar(20),salary float);

```
MySQL> CREATE TABLE employee3 (
-> eid INT,
-> name VARCHAR(20),
-> salary FLOAT DEFAULT 10000
-> );
```

9.a) Create the employee table using following schema and data.

Employee(eid int,name varchar(20),salary float);

eid	name	salary
101	Mr.X	120000
102	Mr.Y	150000
103	Mr.Z	90000
104	Mr.P	80000

```
MySQL> CREATE TABLE employee (
-> eid INT,
-> name VARCHAR(20),
-> salary FLOAT
-> );
MySQL> INSERT INTO employee (eid, name, salary)
-> VALUES (101, 'Mr.X', 120000),
-> (102, 'Mr.Y', 150000),
-> (103, 'Mr.Z', 90000),
-> (104, 'Mr.P', 80000);
Query OK, 4 rows affected .
```

b) write an SQL query to display the names and salary of all employees in the organization

```
mysql> SELECT name, salary
-> FROM employee;
+------+--------+
| name | salary |
+------+--------+
| Mr.X | 120000 |
| Mr.Y | 150000 |
| Mr.Z | 90000 |
| Mr.P | 80000 |
+------+--------+
```

c) write an SQL query to change the salary of Mr.Z from 90000 to 95000.

```
MySQL> UPDATE employee
-> SET salary = 95000
> WHERE name = 'Mr.Z';
```

d) write an SQL query to display employee names whose salary is more than 100000

```
MySQL> SELECT name
-> FROM employee
-> WHERE salary > 100000;
```

e) write an SQL query to remove the record of Mr.P from the table.

```
MySQL> DELETE FROM employee

-> WHERE name = 'Mr.P';
```

f) write an SQL query to remove all the records from the table using DML command.

```
MySQL> DELETE FROM employee;
```

10.a) Create the student table using following schema. Make sure that sid column doesnot allow null values.

student(sid int,name varchar(20),age int);

sid	name	age
101	Mr.X	15
102	Mr.Y	14
103	Mr.Z	19
104	Mr.P	20

```
MySQL> CREATE TABLE student5(

-> sid INT NOT NULL,

-> name VARCHAR(20),

-> age INT

-> );

Query OK, 0 rows affected (0.03 sec)

MySQL> INSERT INTO student5 (sid, name, age)

-> VALUES (101, 'Mr.X', 15),

-> (102, 'Mr.Y', 14),

-> (103, 'Mr.Z', 19),

-> (104, 'Mr.P', 20);

Query OK, 4 rows affected
```

b) write an SQL query to display the names and age of all students in the college.

```
mysql> SELECT name, age

> FROM student5;

+------+------+
| name | age |
+------+------+
| Mr.X | 15 |
| Mr.Y | 14 |
| Mr.Z | 19 |
| Mr.P | 20 |
+------+------+
```

c) write an SQL query to change the age of Mr.Z from 19 to 20.

```
MySQL> UPDATE student5
-> SET age = 20
-> WHERE name = 'Mr.Z';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

d) write an SQL query to display student names whose age is more than 18

```
MySQL> SELECT name
-> FROM student5
-> WHERE age > 18;
+------+
| name |
+------+
| Mr.Z |
| Mr.P |
+------+
2 rows in set
```

e) write an SQL query to remove the record of Mr.P from the table.

```
MySQL> DELETE FROM student5 WHERE name = 'Mr.P';
```

f) write an SQL query to remove all the records from the table using DML command.

```
mysql> DELETE FROM student5;
```

Second Internal

1 a) create sailors, boats, reserves tables using the following fields.

Note: sid is the primary key for sailors table

bid is the primary key for boats table

sid,bid is foreign keys in reserves table

Sailors(sid,sname,rating,age);

Boats(bid,bname,color);

Reserves(sid,bid,day);

```
CREATE TABLE sailors (
   ->    sid integer not null,
   ->    sname varchar(32),
   ->    rating integer,
   ->    age real,
   ->    CONSTRAINT PK_sailors PRIMARY KEY (sid)
   -> );
Query OK, 0 rows affected (0.03 sec)
CREATE TABLE boats (
   ->    bid integer not null,
   ->    bname varchar(25),
   ->    color varchar(21),
   ->    CONSTRAINT PK_boats PRIMARY KEY (bid)
   -> );
Query OK, 0 rows affected (0.02 sec)
CREATE TABLE reserves (
   ->    sid integer not null,
   ->    bid integer not null,
   ->    day datetime not null,
   ->    CONSTRAINT PK_reserves PRIMARY KEY (sid, bid, day),
   ->    FOREIGN KEY (sid) REFERENCES sailors(sid),
   ->    FOREIGN KEY (bid) REFERENCES boats(bid)
   -> );


mysql> INSERT INTO sailors (sid, sname, rating, age) VALUES (22, 'Dustin', 7, 45.0), (23, 'John', 8, 35.0),
(24, 'Mike', 9, 25.0);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0


mysql> INSERT INTO boats (bid, bname, color) VALUES (101, 'Boat1', 'red'), (102, 'Boat2', 'green'), (103,
'Boat3', 'blue');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0


mysql> INSERT INTO reserves (sid, bid, day) VALUES (22, 101, '1998-10-10'), (23, 102, '1998-10-11'), (24,
103, '1998-10-12');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

b) Write a SQL Query to find name of the sailors who have reserved both red or green

boat

```
mysql> SELECT s.sname
   -> FROM sailors s
   -> JOIN reserves r ON r.sid=s.sid
   -> JOIN boats b ON r.bid=b.bid AND b.color='red'
   -> WHERE r.sid IN(SELECT s.sid FROM sailors s JOIN reserves r ON r.sid=s.sid JOIN boats b ON
r.bid=b.bid WHERE b.color='green');
Empty set (0.00 sec)
```

c) Write a SQL Query to find the name of the sailors who have reserved boat bid 103

```
mysql> SELECT S.sname
    -> FROM Sailors S
    -> WHERE EXISTS (SELECT * FROM Reserves R WHERE R.bid=103 AND R.sid=S.sid);
+-------+
| sname |
+-------+
| Mike  |
+-------+
1 row in set (0.01 sec)
```

d) Write a SQL Query to find the name of the sailors who have not reserved boat bid 104

```
mysql> SELECT s.sname
    -> FROM sailors s
    -> WHERE NOT EXISTS (
    ->     SELECT *
    ->     FROM reserves r
    ->     WHERE r.sid = s.sid AND r.bid = 104
    -> );
+--------+
| sname  |
+--------+
| Dustin |
| John   |
| Mike   |
+--------+
3 rows in set (0.00 sec)
```

e) Write a SQL Query to find all SIDs of sailors who have a rating of 10 or reserved boat 104

```
mysql> SELECT DISTINCT r.sid
    -> FROM reserves r
    -> JOIN sailors s ON r.sid = s.sid
    -> WHERE s.rating = 10 OR r.bid = 104;
Empty set (0.01 sec)
```

2 a) create sailors, boats, reserves tables using the following fields.

Note: sid is the primary key for sailors table

bid is the primary key for boats table

sid,bid is foreign keys in reserves table

```
Sailors(sid,sname,rating,age);
Boats(bid,bname,color);
Reserves(sid,bid,day);

CREATE TABLE sailors (
    sid INT PRIMARY KEY,
    sname VARCHAR(255),
    rating INT,
    age INT
);

CREATE TABLE boats (
    bid INT PRIMARY KEY,
    bname VARCHAR(255),
    color VARCHAR(255)
);

CREATE TABLE reserves (
    sid INT,
    bid INT,
    day DATE,
    PRIMARY KEY (sid, bid),
    FOREIGN KEY (sid) REFERENCES sailors(sid),
    FOREIGN KEY (bid) REFERENCES boats(bid)
);
```

b) Write a SQL Query to display average age of all sailors

```
mysql> SELECT AVG(age) AS avg_age
    -> FROM sailors;
+---------+
| avg_age |
+---------+
|      35 |
+---------+
1 row in set (0.01 sec)
```

c) Write a SQL Query to display number of sailors in the sailors table

```
mysql> SELECT COUNT(*) AS num_sailors
    -> FROM sailors;
+-------------+
| num_sailors |
+-------------+
|           3 |
+-------------+
1 row in set (0.02 sec)
```

d) Write a SQL Query to display name of the sailor who is older than all sailors

```
mysql> SELECT sname
    -> FROM sailors
    -> WHERE age = (
    ->     SELECT MAX(age)
    ->     FROM sailors
    -> );
+--------+
| sname  |
+--------+
| Dustin |
+--------+
1 row in set (0.00 sec)
```

e) Write a SQL Query to find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.

```
mysql> SELECT rating, AVG(age) AS avg_age
    -> FROM sailors
    -> WHERE age >= 18
    -> GROUP BY rating
    -> HAVING COUNT(*) >= 2;
Empty set (0.00 sec)
```

3. a) create sailors, boats, reserves tables using the following fields.

Note: sid is the primary key for sailors table bid is the primary key for boats table

sid,bid is foreign keys in reserves table

Sailors(sid,sname,rating,age);

Boats(bid,bname,color);

Reserves(sid,bid,day);

Same as 1(a)

b) Write a SQL query to create a view with the name young_sailors (age <30)

Young_sailors(sid,sname,age)

```
mysql> CREATE VIEW young_sailors AS
   -> SELECT sid, sname, age
   -> FROM sailors
   -> WHERE age < 30;
Query OK, 0 rows affected (0.01 sec)
```

c) Write a SQL query to create a view with the name old_sailors (age >30)

old_sailors(sid,sname,age)

```
mysql> CREATE VIEW old_sailors AS
   -> SELECT sid, sname, age
   -> FROM sailors
   -> WHERE age > 30;
Query OK, 0 rows affected (0.01 sec)
```

d) Write a SQL query to delete young_sailors view.

```
mysql> DROP VIEW IF EXISTS young_sailors;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> select *from young_sailors;
ERROR 1146 (42S02): Table 'bsr.young_sailors' doesn't exist
```

e) Write a SQL query to delete old_sailors view.

```
mysql> select *from old_sailors;
+-----+--------+------+
| sid | sname  | age  |
+-----+--------+------+
|  22 | Dustin |   45 |
|  23 | John   |   35 |
+-----+--------+------+
2 rows in set (0.01 sec)

mysql> DROP VIEW IF EXISTS old_sailors;
Query OK, 0 rows affected (0.01 sec)

mysql> select *from old_sailors;
ERROR 1146 (42S02): Table 'bsr.old_sailors' doesn't exist
```

4. a) create sailors, boats, reserves tables using the following fields.

Note: sid is the primary key for sailors table

bid is the primary key for boats table

sid,bid is foreign keys in reserves table

Sailors(sid,sname,rating,age);

Boats(bid,bname,color);

Reserves(sid,bid,day);

Same as 1(a)

b) What are triggers? Explain with detailed syntax for creating triggers?

In SQL, a trigger is a special type of stored procedure that is automatically executed in response to certain events or changes in the database. These events can include INSERT, UPDATE, and DELETE operations on a table. Triggers are used to enforce business rules, perform complex calculations, and maintain data integrity.

example of the syntax for creating a trigger:

```
CREATE TRIGGER trigger_name

AFTER INSERT ON table_name

FOR EACH ROW

BEGIN

   -- trigger code goes here

END;
```

c) Demonstrate creating trigger with an example

```
mysql> delimiter //

mysql> create trigger tr1 before insert on sailors

   -> for each row

   -> begin

   -> if new.age>50 then set new.age=50;

   -> else

   -> set new.age=40;

   -> end if;

   -> end;

   -> //

Query OK, 0 rows affected (0.02 sec)

mysql> insert into sailors values(25,'joe',5,55);

   -> //

Query OK, 1 row affected (0.01 sec)


mysql> select *from sailors;
```

```
    -> //
+-----+--------+--------+------+
| sid | sname  | rating | age  |
+-----+--------+--------+------+
|  22 | Dustin |      7 |   45 |
|  23 | John   |      8 |   35 |
|  24 | Mike   |      9 |   25 |
|  25 | joe    |      5 |   50 |
+-----+--------+--------+------+
4 rows in set (0.00 sec)
```

5. a) create sailors, boats, reserves tables using the following fields.

   Note: sid is the primary key for sailors table

   bid is the primary key for boats table

   sid,bid is foreign keys in reserves table

Sailors(sid,sname,rating,age);

Boats(bid,bname,color);

Reserves(sid,bid,day);

Same as 1(a)

b) Write a procedure to retrieve name and age of sailors whose age is more than 30

```
mysql> CREATE PROCEDURE pr(age int)
    -> BEGIN
    ->     SELECT name, age
    ->     FROM sailors
    ->     WHERE age > 30;
    -> END;
    -> //
Query OK, 0 rows affected (0.04 sec)
```

c) Write a procedure to display names of the boat with red color.

```
mysql> CREATE PROCEDURE p2(color int)
   -> BEGIN
   ->     SELECT name
   ->     FROM boat
   ->     WHERE color = 'red';
   -> END;
   -> //
Query OK, 0 rows affected (0.01 sec)
```

d) Write a procedure to display names of the sailors who reserved green color boat.

```
mysql> CREATE PROCEDURE p3(color int)
   -> BEGIN
   ->     SELECT sailors.name
   ->     FROM sailors
   ->     JOIN reserves ON sailors.sid = reserves.sid
   ->     JOIN boat ON reserves.bid = boat.bid
   ->     WHERE boat.color = 'green';
   -> END;
   -> //
Query OK, 0 rows affected (0.02 sec)
```

6. What is the need of normalization? Explain 1st, 2nd and 3rd normal forms with suitable examples?

Normalization is an important process in database design that helps in improving the efficiency, consistency, and accuracy of the database. It makes it easier to manage and maintain the data and ensures that the database is adaptable to changing business needs1.

The need for normalization arises due to several reasons. Some of them are:

1. It eliminates redundant data.
2. It reduces chances of data error.
3. The normalization is important because it allows the database to take up less disk space.

4. It also helps in increasing performance.

5. It improves the data integrity and consistency2.

Normalization is divided into several normal forms such as 1st normal form (1NF), 2nd normal form (2NF), 3rd normal form (3NF), and Boyce-Codd normal form (BCNF). Each normal form has its own set of rules that must be followed to ensure that the database is normalized properly.

First Normal Form (1NF) - A relation is said to be in 1NF if it contains no repeating groups or arrays3.

Second Normal Form (2NF) - A relation is said to be in 2NF if it is in 1NF and every non-key attribute is fully dependent on the primary key3.

Third Normal Form (3NF) - A relation is said to be in 3NF if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key3.

For example, consider a table named “Student” with columns “Student ID”, “Student Name”, “Course ID”, “Course Name”, “Course Instructor”. This table violates the first normal form because it contains repeating groups of data such as “Course ID”, “Course Name”, and “Course Instructor”. To normalize this table, we can create two tables named “Student” and “Course” with columns as shown below:

7. What is normalization? Explain BCNF, 4th and 5th normal forms with suitable examples?

Normalization is an important process in database design that helps in improving the efficiency, consistency, and accuracy of the database. It makes it easier to manage and maintain the data and ensures that the database is adaptable to changing business needs1.

Normalization is divided into several normal forms such as 1st normal form (1NF), 2nd normal form (2NF), 3rd normal form (3NF), Boyce-Codd normal form (BCNF), 4th normal form (4NF), and 5th normal form (5NF). Each normal form has its own set of rules that must be followed to ensure that the database is normalized properly.

Boyce-Codd Normal Form (BCNF) - A relation is said to be in BCNF if it is in 3NF and for every functional dependency X → Y, X must be a superkey2.

Fourth Normal Form (4NF) - A relation is said to be in 4NF if it is in BCNF and has no non-trivial multi-valued dependencies34.

Fifth Normal Form (5NF) - A relation is said to be in 5NF if it is in 4NF and cannot be decomposed into any number of smaller tables without loss of data35.

For example, consider a table named “Employee” with columns “Employee ID”, “Employee Name”, “Department ID”, “Department Name”, “Project ID”, “Project Name”. This table violates the BCNF because there are functional dependencies such as “Department ID” → “Department Name” and “Project ID” → “Project Name” which do not satisfy the superkey condition. To normalize this table, we can create three tables named “Employee”, “Department”, and “Project” with columns as shown below:

Table 1: Employee

Employee ID	Employee Name	Department ID	Project ID
1	John	101	201
2	Jane	102	202

Table 2: Department

Department ID	Department Name
101	Sales
102	Marketing

Table 3: Project

Project ID	Project Name
201	Project A
202	Project B

Here, we have eliminated the functional dependencies by creating three separate tables3

8) a) create sailors, boats, reserves tables using the following fields.

Note: sid is the primary key for sailors table

bid is the primary key for boats table

sid,bid is foreign keys in reserves table

Sailors(sid,sname,rating,age);

Boats(bid,bname,color);

Reserves(sid,bid,day);

Same as 1(a)

b) What are cursors? Write syntax for:

i) Declaring a cursor ii) Opening a cursor iii) Fetching a cursor iv) closing a cursor

Cursor: Cursors are used when the SQL Select statement is expected to return more than one row. Cursors are supported inside procedures and functions. Cursors must be declared and its definition contains the query. The cursor must be defined in the DECLARE section of the program. A cursor must be opened before processing and close after processing.

Syntax to declare the cursor:

```
DECLARE <cursor_name> CURSOR FOR <select_statement>
```

Multiple cursors can be declared in the procedures and functions but each cursor must have a unique name. And in defining the cursor the select_statement cannot have INTO clause.

Syntax to open the cursor:

```
OPEN <cursor_name>
```

By this statement we can open the previously declared cursor.

Syntax to store data in the cursor :

```
FETCH <cursor_name> INTO <var1>,<var2>…….
```

The above statement is used to fetch the next row if a row exists by using the defined open cursor.

Syntax to close the cursor :

```
CLOSE <cursor_name>
```

By this statement we can close the previously opened cursor. If it is not closed explicitly then a cursor is closed at the end of compound statement in which that was declared

```
mysql> create procedure kcurl1(s_id int)
  -> begin
  -> declare x_no int;
  -> declare x_name varchar(50);
  -> declare x_age int;
  -> declare c5 cursor for select sid,sname,age from sailors where sid=s_id;
  -> open c5;
  -> fetch c5 into x_no,x_name,x_age;
  -> select x_no,x_name,x_age from sailors where sid=s_id;
  -> close c5;
  -> end;
  -> //
Query OK, 0 rows affected (0.01 sec)
```