Practical Assessment 3: MySQL Fundamentals

Task 1: Database Creation and Table Creation

1.Create a new MySQL database with a unique name.
2.Create a table named "users" with the following columns:
3.ID (integer, auto-increment, primary key)
4.Name (varchar)
5.Email (varchar)
6.Age (integer)
7.Insert at least five rows of sample data into the "users" table.

```
CREATE TABLE users (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(50),
    Email VARCHAR(100),
    Age INT
);

INSERT INTO users (Name, Email, Age) VALUES
('Raj', 'raj@example.com', 25),
('Abhi', 'abhi@example.com', 28),
('Aman', 'aman@example.com', 22),
('Manju', 'manju@example.com', 30),
('Vijay', 'vijay@example.com', 27);
```

Task 2: Data Retrieval

1.Write a query to retrieve all the records from the "users" table.

```
SELECT * FROM `users`
```

2.Write a query to retrieve users with an age greater than or equal to 30.

```
SELECT * FROM `users` WHERE Age>30
```

3.Write a query to retrieve users with a specific email domain (e.g., "@example.com").

```
SELECT * FROM users WHERE Email LIKE 'raj@example.com';
```

4.Write a query to retrieve the names of users in alphabetical order.

```
SELECT Name FROM users ORDER BY Name ASC;
```

Task 3: Data Manipulation

1.Write a query to update the age of a specific user.

```
UPDATE `users` SET `Age`='32' WHERE Name="manju";
```

2.Write a query to delete a specific user from the "users" table.

```
DELETE FROM `users` WHERE name= "vijay";
```

3.Write a query to insert a new user into the "users" table.

```
INSERT into users(Name,Email,Age)VALUES('vijay','vijay@gmail.com', 34);
```

Task 4: Data Aggregation and Sorting

1.Write a query to calculate the average age of all users.

```
SELECT AVG(Age) AS Average_Age FROM users;
```

2.Write a query to count the number of users in the "users" table.

```
SELECT COUNT(ID) FROM users;
```

3.Write a query to retrieve the oldest user from the "users" table.

```
SELECT * FROM users WHERE Age = (SELECT MAX(Age) FROM users);
```

Task 5: Joining Tables

1.Create a new table named "orders" with the following columns:
2.OrderID (integer, auto-increment, primary key)
3.UserID (integer, foreign key referencing the "users" table)
4.OrderDate (date)
5.Insert at least five rows of sample data into the "orders" table.
6.Write a query to retrieve the order details along with the corresponding user information (name and email)

```
CREATE TABLE orders ( orderid INT AUTO_INCREMENT PRIMARY KEY, userid INT, order
date DATE, FOREIGN KEY (userid) REFERENCES users(ID) );
```

```
SELECT * FROM users u, orders o WHERE u.ID=o.userid;
```

Task 6: Data Backup and Restore

1.Export the database schema and data into a SQL dump file.
2.Drop the database and recreate it.
3.Restore the database using the SQL dump file.

Task 7: Data Security and Privileges

1.Create a new MySQL user with limited privileges to access only the "users" table.
2.Test the new user's access by performing select, insert, update, and delete operations on the "users" table.

Task 8: Data Constraints and Validation

1.Alter the "users" table to add a constraint that ensures the age column cannot have a negative value.

```
ALTER TABLE users ADD CONSTRAINT chec  CHECK (age >= 0);
```

2.Attempt to insert a row with a negative age value and verify that the constraint prevents the insertion.

Task 9: Data Filtering and Conditional Retrieval

1.Write a query to retrieve users whose names start with the letter "J".

```
SELECT * FROM `users` WHERE Name LIKE 'J%';
```

2.Write a query to retrieve users who are older than 40 and have a Gmail email address.

```
SELECT * FROM users where Age>40 AND Email IN (SELECT Email FROM users);
```

Task 10: Database Setup

1.Create two tables: "users" and "posts".
2.The "users" table should have the following columns:
3.ID (integer, auto-increment, primary key)
4.Name (varchar)
5.Email (varchar)
6. The "posts" table should have the following columns:
7.ID (integer, auto-increment, primary key)
8.Title (varchar)
9.Content (text)
10.User ID (integer, foreign key referencing the "users" table)

```
CREATE TABLE posts(ID int AUTO_INCREMENT PRIMARY KEY,Title varchar(50), Content text, userid int, FOREIGN KEY(userid) REFERENCES users(ID));
```

Task 11: Data Insertion

1. Insert at least three users into the "users" table.
2. Insert at least five posts into the "posts" table, assigning each post to a specific user using the User ID column.

Task 12: Data Retrieval

1. Write a query to retrieve all posts along with the corresponding user information (name and email).

```
SELECT Name, Email,Title, Content FROM users u,posts p where u.ID=p.userid;
```

2. Write a query to retrieve all posts for a specific user, given their user ID.
```
SELECT Name, Email,Title, Content FROM users u,posts p where u.ID=p.userid and u.ID=1;
```

3. Write a query to retrieve the user(s) who have the most posts.

```
SELECT u.ID, u.Name, COUNT(p.ID) AS post_count FROM users u JOIN posts p ON u.ID = p.userid GROUP BY u.ID, u.Name HAVING COUNT(p.ID) = ( SELECT MAX(post_count) FROM ( SELECT COUNT(*) AS post_count FROM posts GROUP BY userid ) AS counts );
```

Task 13: Data Modification

1. Write a query to update the name of a specific user
```
UPDATE `users` SET Name="rajesh" WHERE name="raj";
```

2. Write a query to delete all posts for a specific user.

```
DELETE FROM posts WHERE userid=3;
```

Task 14: Data Validation and Integrity

1. Add a foreign key constraint to the "posts" table that references the "users" table, ensuring referential integrity.
```
ALTER TABLE posts ADD CONSTRAINT FOREIGN KEY(sid) REFERENCES users(id);
```
2. Attempt to insert a post with a non-existing user ID and verify that the constraint prevents the insertion.

Task 15: Eager Loading

Write a query to retrieve all users along with their associated posts in a single query.

```
SELECT u.ID, u.Name, u.Email, p.ID AS PostID, p.Title, p.Content FROM users u LEFT JOIN posts p ON u.ID = p.userid ORDER BY u.ID, p.ID;
```