Q1.

Step 1: Create Database and Table

First, create a database and a table for bank accounts.

```
CREATE DATABASE BankDB;
USE BankDB;

CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY AUTO_INCREMENT,
    AccountHolder VARCHAR(50),
    Balance DECIMAL(10, 2)
);
```

Step 2: Insert Sample Data

Insert some sample data to work with.

```
INSERT INTO Accounts (AccountHolder, Balance) VALUES ('Alice',
5000.00);
INSERT INTO Accounts (AccountHolder, Balance) VALUES ('Bob', 3000.00);
```

Step 3: Start a Transaction and Perform Operations

1. `START TRANSACTION` or `BEGIN`

Start a transaction. Any changes made after this command will be part of the transaction and will only be committed if you run `COMMIT`.

```
START TRANSACTION;
-- OR
BEGIN;
```

2. Perform an Operation (e.g., Update Balances)

Let's say we want to transfer 500 from Alice's account to Bob's account.

```
UPDATE Accounts SET Balance = Balance - 500 WHERE AccountHolder =
'Alice';
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountHolder =
'Bob';
```

3. `SET`

Use `SET` to modify session variables or other settings. For example, if you want to change the isolation level for this transaction:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

4. `COMMIT`

If everything is correct and you want to save the changes, use `COMMIT` to make the changes permanent.

```
COMMIT;
```

Optional: `ROLLBACK`

If you want to undo the transaction (e.g., if an error occurs or conditions aren't met), use `ROLLBACK` to revert changes.

```
ROLLBACK;
```

Verification

After the transaction is complete, you can check balances to confirm:

```
SELECT * FROM Accounts;
```

Q.2

Step 1: Create Database and User Table

```
CREATE DATABASE UserDB;
USE UserDB;

CREATE TABLE Users (
    UserID INT PRIMARY KEY AUTO_INCREMENT,
    Username VARCHAR(50),
    Email VARCHAR(50),
    Credit DECIMAL(10, 2)
);
```

Step 2: Insert Sample Data

Add some sample data to work with.

```
INSERT INTO Users (Username, Email, Credit) VALUES ('JohnDoe',
'john@example.com', 1000.00);
INSERT INTO Users (Username, Email, Credit) VALUES ('JaneSmith',
'jane@example.com', 1500.00);
```

Step 3: Perform Transactions and Operations

1. `START TRANSACTION` or `BEGIN`

To start a transaction, use `START TRANSACTION` or `BEGIN`. This will allow any changes to be part of a transaction and will only be applied permanently if you use `COMMIT`.

```
START TRANSACTION;
-- OR
BEGIN;
```

2. Perform an Operation (e.g., Update Credit)

For example, suppose we want to transfer 200 credits from JohnDoe to JaneSmith.

```
UPDATE Users SET Credit = Credit - 200 WHERE Username = 'JohnDoe';
UPDATE Users SET Credit = Credit + 200 WHERE Username = 'JaneSmith';
```

3. `SET`

You can use `SET` to configure transaction settings, such as isolation levels. For example, to set the transaction isolation level to `READ COMMITTED`:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

4. `COMMIT`

Once you're sure the changes are correct, use `COMMIT` to make them permanent.

```
COMMIT;
```

Optional: `ROLLBACK`

If an error occurs or you decide not to proceed with the transaction, use `ROLLBACK` to cancel the changes.

```
ROLLBACK;
```

Verify Changes

After committing, you can check the updated balances:

```
SELECT * FROM Users;
```

Q.3

Step 1: Create Database and Purchase Table

Create a database called `CustomerDB` and a table for storing customer purchase details.

```
CREATE DATABASE CustomerDB;
USE CustomerDB;

CREATE TABLE Purchases (
    PurchaseID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerName VARCHAR(50),
    ProductName VARCHAR(50),
    Quantity INT,
    Price DECIMAL(10, 2),
    Total DECIMAL(10, 2)
);
```

Step 2: Insert Sample Data

Add some sample data to work with.

```
INSERT INTO Purchases (CustomerName, ProductName, Quantity, Price,
Total) VALUES
('Alice', 'Laptop', 1, 1200.00, 1200.00),
('Bob', 'Phone', 2, 500.00, 1000.00);
```

Step 3: Perform Transactions and Operations

1. `START TRANSACTION` or `BEGIN`

To start a transaction, use `START TRANSACTION` or `BEGIN`. Any changes will be part of the transaction and will only be applied if you use `COMMIT`.

```
START TRANSACTION;
-- OR
BEGIN;
```

2. Perform an Operation (e.g., Update Purchase Details)

Suppose we want to update a purchase to adjust the quantity and total price. For example, Alice purchases one more laptop, so we update the quantity and total accordingly.

```
UPDATE Purchases SET Quantity = Quantity + 1, Total = Total + 1200.00
WHERE CustomerName = 'Alice';
```

3. `SET`

Use `SET` to adjust transaction settings if needed. For example, setting the isolation level to ensure the transaction is executed in `REPEATABLE READ` isolation mode:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

4. `COMMIT`

Once you're sure the updates are correct, use `COMMIT` to make the changes permanent.

```
COMMIT;
```

Optional: `ROLLBACK`

If an error occurs or you decide not to apply the changes, use `ROLLBACK` to revert the transaction.

```
ROLLBACK;
```

Verify Changes

After committing, you can check the updated purchase details:

```
SELECT * FROM Purchases;
```