**TOPICS: Advanced Transaction Management in MySQL Allowed Time:** 90 Minutes

**Instructions: Total Marks:** 30

1. Gossips are not allowed.
2. Teacher assistants are for your help, so be nice with them. Respect them as they are teaching you. Raise your hands if you have some problem and need help from TA. Avoid calling them by raising your voice and disturbing the environment of Lab.
3. TA may deduct your marks for any kind of ill-discipline or misconduct from your side.
4. Evaluation will be considered final and you cannot debate for the marks. So, focus on performing the tasks when the time is given to you.
5. Paste the query as well as result table screenshot as a result of each task

**Task 01: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ (25 Marks)**

**Part 1**: **Database and Table Setup**

Define and create the following tables in your database:

**1. Customers Table**

CustomerID (Primary Key, INT)

CustomerName (VARCHAR(100))

Email (VARCHAR(100))

**2. Products Table**

ProductID (Primary Key, INT)

ProductName (VARCHAR(100))

Stock (INT, must be non-negative)

Price (DECIMAL(10,2))

**3. Orders Table**

OrderID (Primary Key, INT, Auto Increment)

CustomerID (Foreign Key referencing Customers(CustomerID))

ProductID (Foreign Key referencing Products(ProductID))

Quantity (INT)

OrderDate (Default: Current Timestamp)

**Note:**

Ensure proper primary and foreign key constraints.

**Insert Sample Data**

Populate each table with at least 3 rows of sample data.

**Customers:**

Alice, Bob, Charlie with appropriate email addresses.

**Products:**

Laptop (Stock: 10, Price: 1000.00), Smartphone (Stock: 15, Price: 700.00), Headphones (Stock: 50, Price: 50.00).

**Orders:**

Leave this table empty initially. Orders will be added during the transaction tasks.

**Part 2**: **Transaction Management Tasks**

**Scenario 1**: Placing an Order

Simulate placing two orders:

1. Alice orders 5 Laptops.

2. Bob orders 12 Laptops

**Steps:**

1. Start a transaction.

2. Check stock levels and decrease stock if sufficient:

Decrease stock by the ordered quantity for valid orders.

Rollback the transaction if the stock is insufficient.

3. Insert the order details into the Orders table.

4. Use a savepoint after the stock update but before inserting the order.

5. Commit the transaction if all operations succeed; otherwise, rollback.

**Scenario 2**: Update an Order

Update Alice's order from 5 Laptops to 8 Laptops. Rollback if the total price exceeds 5000.

**Steps:**

1. Start a transaction.

2. Update the Orders table to reflect the new quantity.

3. Adjust the stock in the Products table accordingly:

Deduct or add stock based on the updated quantity.

4. Calculate the total price of the updated order.

5. Use a savepoint before updating the stock. Rollback to the savepoint if the price exceeds 5000.

6. Commit if the operations are valid; rollback otherwise.

**Scenario 3**: Cancel an Order

Cancel Alice's order for Laptops and restore the stock.

**Steps:**

1. Start a transaction.

2. Restore the product stock in the Products table.

3. Delete the order record from the Orders table.

4. Use a savepoint before deleting the order. Rollback to the savepoint if the stock update fails.

5. Commit if all operations are successful; rollback otherwise.

**Scenario 4**: Place Multiple Orders

Data for the Scenario:

Customer Charlie places the following orders:

1. 30 Headphones.

2. 10 Smartphones.

**Steps:**

1. Start a transaction.

2. Insert each order into the Orders table.

Adjust the stock in the Products table accordingly.

3. Use savepoints after each stock update to handle partial rollbacks if one order fails due to insufficient stock.

4. Commit the transaction only if all operations are valid.

**Part 3**: **Perform Queries on the Database**

Perform the following queries to validate and understand the state of your database:

1. Display complete details of all transactions, including:

* OrderID
* Customer Name
* Product Name
* Quantity Ordered
* Total Price
* Order Date

2. Show the updated stock levels of all products in the Products table.

3. Retrieve all orders placed by Charlie, displaying product details and total price for each order.

4. Display the total number of products ordered for each product, grouped by product name.

5. Verify that all transactions in scenarios above correctly updated or rolled back changes by comparing initial and final states of the tables.

**Task 02: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ (05 Marks)**

Track the values of **bal\_x** , **bal\_y and bal\_z** for these transactions at each time stamp.

**Initial values for variables are:**

**bal\_x = 1000**

**bal\_y = 500**

**bal\_z = 300**

