

Question 1:

In a SQL Server database, you have been assigned the responsibility of managing a table named Orders for an e-commerce platform. The Orders table has the following structure:

- OrderID: An integer representing the unique identifier for each order.
- CustomerID: An integer representing the ID of the customer who placed the order.
- OrderAmount: A decimal representing the total amount of the order.
- OrderDate: A datetime representing the date and time when the order was placed.
- Status: A string representing the status of the order (e.g., "Pending", "Shipped", "Delivered").

You are required to perform the following tasks:

1. Create and Populate the Table: Create the Orders table and insert the following records:

2.Pessimistic Concurrency Control: Write a SQL query to implement pessimistic concurrency control by placing an exclusive lock on the order with OrderID 1 to update its Status to "Shipped". Ensure that no other transactions can access this record while it is being updated.

Input Ta:

OrderID	CustomerID	OrderAmount	OrderDate	Status
1	101	150.75	2023-09-20T10:00:00	Pending
2	102	300.5	2023-09-20T11:00:00	Shipped
3	103	75.25	2023-09-20T14:00:00	Delivered

Title for Question 1: Implementing Pessimistic Concurrency Control in MariaDB for Order Status Updates

Solution:

```
-- Implement pessimistic concurrency control to update the status of order
START TRANSACTION;

-- Update the Status to "Shipped" for the locked order
UPDATE Orders1
SET Status = 'Shipped'
WHERE OrderID = 1;

-- Commit the transaction to release the lock
COMMIT;

-- Query the updated table
SELECT * FROM Orders1;
```

TestCases:

S.No	Inputs	Outputs
------	--------	---------

1		OrderID CustomerID OrderAmount OrderDate Status ----- ----- ----- ----- ----- ----- 1 101 150.75 2023-09-25 10:30:00 Shipped 2 102 200.50 2023-09-25 11:15:00 Shipped 3 103 75.25 2023-09-25 12:45:00 Delivered
2		
3		
4		
5		
6		

White List:

Black List:

Question 2:

You are managing a SQL database for a library. You have a table named Books which stores information about each book in the library. The Books table has the following structure:

- BookID: An integer representing the unique identifier for each book.
- Title: A string representing the title of the book.
- Author: A string representing the author of the book.
- Quantity: An integer representing the number of copies of the book available in the library.
- Borrowed: An integer representing the number of copies of the book currently borrowed.

Your tasks are:

- 1. Create and Populate the Table:** Create the Books table and insert the following records:
- 2.Consistency:** Implement the Consistency property by writing a SQL transaction that borrows one copy of the book titled "Programming in C" and returns one copy of the book titled "Artificial Intelligence". Ensure that the Borrowed and Quantity fields maintain the consistency, i.e., Borrowed should always be less than or equal to Quantity.

Input Table:

BookID	Title	Author	Quantity	Borrowed
1	Programming in C	Brian Kernighan	5	2
2	Database Management	Heurleon Elberschultz	5	1
3	Artificial Intelligence	Stuart Russell	4	1

Title for Question 2: Maintaining Consistency in a Library Database: Borrowing and Returning Books

Solution:

```
-- Insert some records into the Books121 table
INSERT INTO Books121 (BookID, Title, Author, Quantity, Borrowed)
VALUES (1, 'Programming in C', 'Brian Kernighan', 5, 2),
```

```
(2, 'Database Management', 'Abraham Silberschatz', 3, 1),
(3, 'Artificial Intelligence', 'Stuart Russell', 4, 1);

-- Borrow one copy of "Programming in C"
UPDATE Books121
SET Borrowed = Borrowed + 1
WHERE Title = 'Programming in C' AND Borrowed < Quantity;

-- Return one copy of "Artificial Intelligence"
UPDATE Books121
SET Borrowed = Borrowed - 1
WHERE Title = 'Artificial Intelligence' AND Borrowed > 0;
```

TestCases:

S.No	Inputs	Outputs
1		BookID Title Author Quantity Borrowed ----- ----- ----- ----- ----- 1 Programming in C Brian Kernighan 5 3 2 Database Management Abraham Silberschatz 3 1 3 Artificial Intelligence Stuart Russell 4 0
2		
3		
4		
5		
6		

White List:

Black List:

Question 3:

You are managing a database for an e-commerce platform. The platform has a Products table with the following structure:

- ProductID: An integer representing the unique identifier for each product.
- ProductName: A string representing the name of the product.
- Price: A decimal representing the price of the product.
- Quantity: An integer representing the available quantity of the product.

Tasks:

1. Two different administrators are updating the price of the product at the same time, and a customer is trying to view the product.
2. Simulate MVCC to ensure that the customer can still view the product while it's being updated, and the administrators' updates are properly managed.

Input Table:

ProductID	ProductName	Price	Quantity
1	Laptop	1000	10

Title for Question 3: Simulating Multi-Version Concurrency Control (MVCC) in an E-commerce Database

Solution:

```
-- Start the first transaction (Admin 1)
BEGIN;

-- Update the Price of Laptop by Admin 1
UPDATE Product0
SET Price = 1100
WHERE ProductID = 1;

-- Commit Transaction 1 (Admin 1)
COMMIT;

-- Start the second transaction (Admin 2)
BEGIN;

-- Update the Price of Laptop by Admin 2
UPDATE Product0
SET Price = 1200
WHERE ProductID = 1;

-- Commit Transaction 2 (Admin 2)
COMMIT;

SELECT * FROM Product0 ;
```

TestCases:

S.No	Inputs	Outputs
1		ProductID ProductName Price Quantity ----- ----- ---- ----- 1 Laptop 1200 10
2		
3		
4		
5		
6		

White List:

Black List:

Question 4:

In a SQL Server database, you are tasked with managing a table named Inventory that stores information about various items in a store's inventory. The table has the following structure:

ItemID: An integer representing the unique identifier for each item.

ItemName: A string representing the name of the item.

Category: A string representing the category to which the item belongs.

StockQuantity: An integer representing the quantity of the item available in stock.

Your tasks:

- 1.Create and Populate the Table: Create the Inventory table and insert the following records into it:
- 2.Lock Granularity Query: Write SQL queries to demonstrate the concept of lock granularity by placing a row-level lock on the item "Laptop", and a page-level lock on items in the "Appliances" category.

Input Table :

ItemID	ItemName	Category	StockQuantity
1	Laptop	Electronics	20
2	Coffee Maker	Appliances	15
3	Desk Chair	Furniture	10

Title for Question 4: Managing Locks on Inventory Items

Solution:

```
-- Query to place a row-level lock on the item "Laptop"
START TRANSACTION;
    SELECT * FROM Inventory WHERE ItemName = 'Laptop' FOR UPDATE;
COMMIT; -- Commit the transaction to release the lock

-- Query to place a page-level lock on items in the "Appliances" category
START TRANSACTION;
    SELECT * FROM Inventory WHERE Category = 'Appliances' FOR UPDATE;
COMMIT; -- Commit the transaction to release the lock
```

TestCases:

S.No	Inputs	Outputs
1		ItemID ItemName Category StockQuantity ----- ----- ----- ----- 1 Laptop Electronics 20 ItemID ItemName Category StockQuantity ----- ----- ----- 2 Coffee Maker Appliances 15
2		
3		

S.No	Inputs	Outputs
4		
5		
6		

White List:

Black List:

Question 5:

Suppose you are working with a database for an online store. The database contains a table named Products with the following structure:

- ProductID: An integer representing the unique identifier for each product.
- ProductName: A string representing the name of the product.
- Quantity: An integer representing the available quantity of the product.

Your tasks are:

- 1.Create and Populate the Table: Create the Products table and insert the following records:
- 2.Serializability: Simulate two transactions:Ensure serializability by using locks such that at any time, only one transaction can access the Quantity of Laptop.

Input Table:

ProductID	ProductName	Quantity
1	Laptop	10
2	Mouse	20
3	Keyboard	15

Title for Question 5: Increase the Quantity of Laptop by 5 in Transaction

Solution:

```
-- Start the first transaction (Transaction 1)
START TRANSACTION;

-- Increase the Quantity of Laptop by 5 in Transaction 1
UPDATE Product123456
SET Quantity = Quantity + 5
WHERE ProductName = 'Laptop';

-- Commit Transaction 1
COMMIT;
```

```
-- Start a new transaction for the second update (Transaction 2)
START TRANSACTION;

-- Decrease the Quantity of Laptop by 2 in Transaction 2
UPDATE Product123456
SET Quantity = Quantity - 2
WHERE ProductName = 'Laptop';

-- Commit Transaction 2
COMMIT;
```

TestCases:

S.No	Inputs	Outputs
1		ProductID ProductName Quantity ----- ----- ----- 1 Laptop 13 2 Mouse 20 3 Keyboard 15
2		
3		
4		
5		
6		

White List:

Black List:

Question 6:

You are managing a SQL database for an e-commerce platform. You have a table named Orders which stores information about each order placed on the platform. The Orders table has the following structure:

- OrderID: An integer representing the unique identifier for each order.
- CustomerID: An integer representing the unique identifier for each customer.
- ProductID: An integer representing the unique identifier for each product.
- Quantity: An integer representing the quantity of the product ordered.
- Status: A string representing the status of the order (e.g., Pending, Shipped, Delivered).

Your tasks are:

1. Create and Populate the Table:

- Create the Orders table and insert the following records:

2. Durability:

- Implement the Durability property by writing a SQL transaction that updates the status of OrderID 1 to 'Shipped'. Once the update is committed, it should be permanently stored in the database,

and the system should be able to recover the updated status even after a failure.

Input Table:????

OrderID	CustomerID	OrderAmount	OrderDate	Status
1	101	100.75	2019-09-26 01:30:00	Pending
2	102	200.5	2019-09-26 18:00:00	Shipped
3	103	75.25	2019-09-26 04:45:00	Delivered

Title for Question 6: Updating Order Status with Durability in Orders321 Table

Solution:

```
-- Update the status of OrderID 1 to 'Shipped'
UPDATE Orders321
SET Status = 'Shipped'
WHERE OrderID = 1;

SELECT * FROM Orders321 ;
```

TestCases:

S.No	Inputs	Outputs
1		OrderID CustomerID ProductID Quantity Status ----- ----- ----- ----- ----- 1 101 501 2 Shipped 2 102 502 1 Shipped 3 103 503 3 Pending
2		
3		
4		
5		
6		

White List:

Black List:
