Project Deadline-6

Transactions (Conflicting & Non-Conflicting)

Transactions

A transaction represents a logical unit of work that is performed against a database. It is a sequence of operations (such as queries or updates). Transactions must follow the “ACID” rule i.e Atomicity, Consistency, Integrity, Durable.

Conflicting Transactions

There are 3 anomalies that can arise upon interleaving actions of different transactions (say, T1 and T2):

1. Write-Read (WR) Conflict: T2 reads a data object previously written by T1
2. Read-Write (RW) Conflict: T2 writes a data object previously read by T1
3. Write-Write (WW) Conflict: T2 writes a data object previously written by T1

Demonstrating (WR) conflict :

Overview : To demonstrate a Write-Read (WR) conflict in your database setup, we'll simulate two transactions (T1 and T2) involving the Product table. The conflict will occur when T2 reads a data object that was previously written by T1.

T1	T2
R(A)	
W(A)	
	R(A)
Commit	Commit

Explanation : The value of A written by T1 is read by T2 before T1 has completed all its changes !
Here A represents data records in Products table where productID = “P1” i.e product with productID = “P1”

Transaction T1 (Write Operation):

```sql
START TRANSACTION;

-- Assume T1 updates product information
UPDATE Product
SET price = 25.99
WHERE productID = 'P1';  -- Example productID being updated

-- Commit the transaction
COMMIT;
```

Transaction T2 (Read Operation):

```sql
START TRANSACTION;

-- Assume T2 reads the product information after T1's update
SELECT *
FROM Product
WHERE productID = 'P1';  -- Example productID being read

-- Commit the transaction
COMMIT;
```

Demonstrating (RW) conflict :

Overview : To demonstrate this conflict, Transaction T1 reads a productID, and then Transaction T2 writes to the productID that was previously read by T1. This conflict can lead to T2 overwriting changes that T1 did not anticipate.

T1	T2
R(A)	
	R(A)
	W(A)
	Commit
R(A)	
Abort	

Explanation : Unrepeatable reads are also known as RW conflicts.
This conflicts arise when transaction T2 writes a data object A that has been read by another transaction T1,while T1 is still in progress.Here A represents data records in Products table where productID = “P1” i.e product with productID = “P1”

Transaction T1 (Read Operation):

```
START TRANSACTION;

-- T1 reads a product's information
SELECT *
FROM Product
WHERE productID = 'P1';  -- Example productID being read

-- Commit the transaction
COMMIT;
```

Transaction T2 (Write Operation):

```
START TRANSACTION;

-- Assume T2 updates the same product's information
UPDATE Product
SET price = 29.99
WHERE productID = 'P1';  -- Example productID being updated

-- Commit the transaction
COMMIT;
```

Demonstrating (WW) conflict :

Transaction T1 and Transaction T2 attempt to write to the same data object concurrently. This conflict can lead to one transaction's changes being overwritten by the other.

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)

Overview : Overwriting Uncommitted Data: WW Conflicts.WW conflicts arise when transaction T2 writes a data object A that has been written by another transaction T1, while T1 is still in progress. This results is loss of data object being written.

Explanation : WW conflicts arise when transaction T2 writes a data object A that has been written by another transaction T1, while T1 is still in progress

Transaction T1 (Write Operation):

```
START TRANSACTION;

-- Assume T1 updates a product's information
UPDATE Product
SET price = 25.99
WHERE productID = 'P1';  -- Example productID being updated

-- Commit the transaction
COMMIT;
```

Transaction T2 (Write Operation):

```
START TRANSACTION;

-- Assume T2 also updates the same product's information
UPDATE Product
SET price = 27.99
WHERE productID = 'P1';  -- Example productID being updated

-- Commit the transaction
COMMIT;
```

Non-Conflicting Transactions

Customer Purchase

Transaction T1: Customer Adds Items to Cart and Places Order

```
START TRANSACTION;

-- Step 1: Customer adds items to the cart (assuming cartID and productID
are known)
INSERT INTO CartItem (cartItemID, cartID, price, quantity, dateAdded)
```

```
VALUES ('C1', 'CustCart', 20.99, 2, CURDATE());  -- Example cartItemID and
values

-- Step 2: Customer places an order
INSERT INTO `Order` (orderID, orderDate, delStatus, custName, addressID,
cartID)
VALUES ('O1', CURDATE(), 'Pending', 'John Doe', 'A1', 'CustCart');  --
Example orderID and values

-- Commit the transaction
COMMIT;
```

Transaction T2: Deduct Item Quantities from Inventory and Update Customer's Purchased Items

```
START TRANSACTION;

-- Step 1: Deduct item quantities from inventory (assuming productID and
purchased quantity are known)
UPDATE Product
SET availableUnits = availableUnits - 2  -- Assuming 2 units were purchased
WHERE productID = 'P1';  -- Example productID being updated

-- Step 2: Update customer's purchased items (assuming cartID and productID
are known)
UPDATE CartItem
SET datePurchased = CURDATE()
WHERE cartID = 'CustCart' AND cartItemID = 'C1';  -- Example cartID and
cartItemID being updated

-- Commit the transaction
COMMIT;
```

Expected Outcome:

- Transaction T1 will add cart items and create an order without affecting the inventory or customer's purchased items.
- Transaction T2 will deduct the purchased item quantities from inventory and update the customer's purchased items in the cart without conflicting with Transaction T1.

Customer Personal Information update & Customer liking a product

Overview : Updating two different data items for customers i.e setting new email and password, and liking a product.

Transaction T1: Customer Updates Personal Information

```
START TRANSACTION;

-- Update customer's email and password
UPDATE Customer
SET email = 'newemail@example.com', password = 'newpassword'
WHERE custName = 'John Doe';  -- Example custName being updated

-- Commit the transaction
COMMIT;
```

Transaction T2: Customer Places a Rating/Like on a Product

```
START TRANSACTION;

-- Insert a new record in Likes table to indicate customer's like/rating on
a product
INSERT INTO Likes (custName, productID)
VALUES ('John Doe', 'P1');  -- Example custName and productID

-- Commit the transaction
COMMIT;
```

Explanation : By executing Transaction T1 and Transaction T2 concurrently or in any order, they will independently perform their respective operations without affecting each other's outcomes. Each transaction operates on distinct tables and data elements within the database, demonstrating the concept of non-conflicting transactions in a multi-user database environment.

Selective Product retrieval and related price hike

Overview :

1. We retrieve the product that are famous i.e have more than 5 likes
2. Price Hike the products by 2x which have more than 5 likes

Transaction T1: Get Products with More Than 5 Likes

```
START TRANSACTION;

-- Retrieve products with more than 5 likes
SELECT p.productID, p.title, p.category, p.price, p.availableUnits,
p.maxUnitsCap, p.portfolioID
```

```
FROM Product p
INNER JOIN (
    SELECT productID, COUNT(*) AS numLikes
    FROM Likes
    GROUP BY productID
    HAVING numLikes > 5
) l ON p.productID = l.productID;

-- Commit the transaction
COMMIT;
```

Transaction T9: Increase Prices of Products with More Than 5 Likes

```
START TRANSACTION;

-- Update prices of products with more than 5 likes
UPDATE Product
SET price = price * 2
WHERE productID IN (
    SELECT l.productID
    FROM Likes l
    GROUP BY l.productID
    HAVING COUNT(*) > 5
);

-- Commit the transaction
COMMIT;
```

Explanation :

Although, the Transactions may seem conflicting for some interleaved schedule. But as

1. **Transaction 1** : Only perform **Read(A)** on the data items.
2. **Transaction 2 :** Perform **Read(A)** on the data items same as transaction 1 but that is done to find consequent data records **B** which reside in the product table and make changes to their prices.
3. These write changes made i.e **Write(B)** does not affect the result of **Read(A).**

Where,

- **A :** are the data records in the likes table with (count(likes) > 5) corresponding to the productID of products.
- B : are the data records in the product table corresponding to the productID retrieved in **A.**

Multiple Designers adding items to portfolio

Overview :

Transaction T10: Designer1 Adds a Product to Portfolio

```
START TRANSACTION;

-- Insert a new product into magnam's portfolio
INSERT INTO Product (productID, title, content, category, price,
availableUnits, maxUnitsCap, portfolioID)
VALUES ('P7', 'Designer1 Product 1', 'Description of Designer1 Product 1',
'Fashion', 49.99, 100, 200, 'Xfepz');  -- Example product details for
Designer1

-- Commit the transaction
COMMIT;
```

Transaction T11: Designer2 Adds a Product to Portfolio

```
START TRANSACTION;

-- Insert a new product into soluta's portfolio
INSERT INTO Product (productID, title, content, category, price,
availableUnits, maxUnitsCap, portfolioID)
VALUES ('P8', 'Designer2 Product 1', 'Description of Designer2 Product 1',
'Home Decor', 39.99, 50, 100, 'aNPmG');  -- Example product details for
Designer2

-- Commit the transaction
COMMIT;
```

Explanation : The following properties make these two non-conflicting for all interleaved schedules. Where designer 1 : magna and designer2 : soluta

Isolation by Portfolio:

- Transaction T10 inserts a product into Designer1's portfolio (Port1), while Transaction T11 inserts a product into Designer2's portfolio (Port2).
- They operate on different portfolios, ensuring no overlap or conflict in portfolio data.

Independent Product Insertions:

- T10 and T11 insert different products (Product records) with unique details and portfolioID values.

- The products added by Designer1 and Designer2 are distinct, preventing conflicts in the Product table.

No Shared Data Modifications:

- Each transaction modifies separate database records (products and portfolios) without affecting shared data used by the other transaction.

Commitment of Changes:

- T10 and T11 commit their changes independently, confirming the successful addition of products for Designer1 and Designer2 without interference.
