DATABASE MANAGEMENT SYSTEMS
Final Project Deadline - Transactions

Gunar Sindhwani
Roll Number - 2020199
Kumar Ankit
Roll Number -2020214

Transaction - 1
→ (Transaction Structure)
Read Statement
Update Statement
Update Statement on a different Table using the update above
Commit Changes

Example of Such Transaction:
Below are 2 transactions.

```
Start Transaction
Select quantity from products where productid = 4;
Quantity: if quantity>5:continue ? abort
Quantity:= quantity - 5
Update products set quantity = Quantity where productid=4;
Insert into order values(productid, quantity)
commit
```

```
Start Transaction
Select quantity from products where productid = 4;
Quantity:= quantity + 5
```

```
Update products set quantity = Quantity where productid=4;
commit
```

Transaction - 2
→ (Transaction Structure)
Read Statement
Update Statement
Commit Changes

Example of Such Transaction:

```
Start Transaction
Select quantity from products where productid = 4;
Quantity:= quantity + 5
Update products set quantity = Quantity where productid=4;
commit
```

Non Conflict Serializable Transaction

Transaction 1 reads quantity from products with productid 4 and then writes to it. Transaction 2 reads from and writes to the same data item quantity. Transaction 2 writes to the same data item (quantity) that transaction 1 reads from and then writes to. If transaction 1 were to execute after transaction 2, it would overwrite the changes made by transaction 2, and vice versa. This violates the conflict serializability requirement, as the final state of the database would depend on the order in which the transactions are executed.

Transaction - 1 : Select quantity from products where productid = 4;
Transaction - 2 : Select quantity from products where productid = 4;
Transaction - 2 : Quantity:= quantity + 5
Transaction - 2 : Update products set quantity = Quantity where productid=1;
Transaction - 2 : commit
Transaction - 1 : Quantity: if quantity>5:continue ? abort
Transaction - 1 : Quantity:= quantity - 5
Transaction - 1 : Update products set quantity = Quantity where productid=4;
Transaction - 1 : Insert into order values(productid, quantity)
Transaction - 1 : commit

LOCK
Transaction - 1 : LOCK-S(quantity)
Transaction - 1 : read(quantity)
Transaction - 1 : unlock(quantity)
Transaction - 2 : LOCK-X(quantity)
Transaction - 2 : read(quantity)
Transaction - 2 : Quantity:= quantity + 5
Transaction - 2 : write(quantity)
Transaction - 2: unlock(quantity)
Transaction - 1 : LOCK-X(quantity)
Transaction - 1 : Quantity: if quantity>5:continue ? abort
Transaction -1 : Quantity:= quantity - 5
Transaction - 1 : Update products set quantity = Quantity where productid=4;
Transaction - 1 : unlock(quantity)
Transaction - 1: LOCK-X(order)
Transaction - 1 : Insert into order values(productid, quantity)
Transaction - 1 : unlock(order)

Explanation :

In this lock sequence, Transaction-1 initially acquires a shared (S) lock on the 'quantity' data item, which allows other transactions to read the data item but not modify it. Transaction-1 then reads the quantity value, after which it releases the shared lock.

Meanwhile, Transaction-2 acquires an exclusive (X) lock on the same 'quantity' data item, which prevents other transactions from either reading or modifying the data item. Transaction-2 then reads the quantity value, updates it by adding 5, and writes the new value back to the data item. After the write, it releases the exclusive lock.

After Transaction-2 releases the lock, Transaction-1 acquires an exclusive (X) lock on the 'quantity' data item to check if the quantity value is greater than 5. If it is, Transaction-1 subtracts 5 from the quantity value, updates the data item, and releases the lock. Finally, Transaction-1 acquires an exclusive (X) lock on the 'order' data item, inserts a new row, and releases the lock.

This lock sequence ensures that the transactions are executed in a conflict serializable manner, as the shared lock on the 'quantity' data item acquired by Transaction-1 allows Transaction-2 to acquire an exclusive lock only after Transaction-1 has released its shared lock and finished reading the data item. Similarly, Transaction-1 acquires an exclusive lock on the 'quantity' data item only after Transaction-2 has released its exclusive lock and finished updating the data item.

CONFLICT SERIALIZABLE TRANSACTIONS
Transaction - 1: Select quantity from products where productid = 4;
Transaction - 1 : Quantity: if quantity>5:continue ? abort
Transaction - 1: Quantity:= quantity - 5
Transaction - 1: Update products set quantity = Quantity where productid=4;
Transaction - 2 : Select quantity from products where productid = 4;
Transaction - 2 : Quantity:= quantity + 5
Transaction - 2 : Update products set quantity = Quantity where productid=4;
Transaction - 2 : commit
Transaction - 1: Insert into order values(productid, quantity)
Transaction - 1 : commit

Explanation :
This set of transactions is conflict serializable because there is no dependency between them.

In Transaction-1, we first read the quantity of productid 4, check if it's greater than 5, and if yes, then update the quantity by subtracting 5 and commit.

In Transaction-2, we also read the quantity of productid 4, but this time, we update it by adding 5 and commit.

Since there is no overlap in the items being read and updated between these two transactions, they can be executed in any order without causing a conflict. Therefore, these transactions are conflict serializable.

LOCK
Transaction - 1 : LOCK-X(quantity)
Transaction - 1 : read(quantity)
Transaction - 1 : Quantity: if quantity>5:continue ? abort
Transaction - 1 : Quantity:= quantity - 5
Transaction - 1: Update products set quantity = Quantity where productid=4;
Transaction - 1: write(quantity)
Transaction - 1: unlock(quantity)
Transaction - 2 : LOCK-X(quantity)
Transaction - 2 : Select quantity from products where productid = 4;
Transaction - 2 : Quantity:= quantity + 5
Transaction - 2 : Update products set quantity = Quantity where productid=4;
Transaction - 2: unlock(quantity)
Transaction - 1: LOCK-X(order)
Transaction - 1 : Insert into order values(productid, quantity)
Transaction - 1 : unlock(order)

Explanation :
The lock sequence above involves exclusive locks (LOCK-X) being acquired by both Transaction 1 and Transaction 2 on the "quantity" column of the "products" table.

Transaction 1 acquires the first LOCK-X, reads the value of "quantity", and checks if it is greater than 5. If it is, then it continues with the transaction, otherwise, it aborts. Transaction 1 then updates the value of "quantity" by subtracting 5, acquires a write lock, and releases the lock.

Transaction 2 acquires the second LOCK-X, selects the current value of "quantity" from the "products" table, adds 5 to it, updates the value, and then releases the lock.

Transaction 1 then acquires an exclusive lock (LOCK-X) on the "order" table, inserts a new row with the "productid" and updated "quantity" values, and then releases the lock.

This lock sequence ensures that only one transaction can update the value of "quantity" at a time to avoid conflicts and inconsistencies in the data. It also ensures that the "order" table is locked exclusively during the insertion of a new row to prevent concurrent transactions from inserting conflicting data.