The Virtual Bazaar

(Online Retail Store)

Project Deadline - 6

Group: 51, Krishna Somani(2021058), Nirmal Soni(2021074)

Project Scope:-

As the name suggests, it's a Virtual Bazaar, an online retail marketplace (like Amazon, Flipkart etc.) where a user can buy a product from a wide range of products, and the retailers can sell their products. Also, there will be a delivery person responsible for getting those products to the customer's doorstep.
And for the customer, all these facilities are available with just a click for any product and all that being home. The platform will have various genres of products with a lot of variety, like clothes, books, electronic gadgets, furniture, etc. A person can sign up as a user to buy the product, as a retailer, or as a delivery guy who can deliver the product to the customer's doorstep. Customers can browse different products from different categories, view and update their cart, choose any favorable mode of payment, and even update their delivery address. Furthermore, retailers can add, update (make changes to the details/particulars of their products at any time they want), and delete any listed products anytime they want. Also, the delivery person can choose to accept or reject any delivery request according to them.
Our database management system will help find details and other information about retailers, delivery persons, and customers systematically by maintaining and following all the required ACID(Atomicity, Consistency, Isolation, and Durability) properties.

User Guide:-

Welcome to the Virtual Bazaar; this is a brief user guide to the platform's features and working. The Virtual Bazaar is a platform where you can find your needs in just one click. And for that, first, you need to Signup and choose how you want to log in, and a user can log in as a Customer or Retailer or as a Delivery Person. Once you decide as a Customer, you must enter the essential details like Name, Age, address, and pNO. Etc. After feeling the elements, a user as a Customer can view products that are surfing through what's present there, which they can buy. The customer can then add the products as much as they would like to their cart in as much quantity as they want. The customer can then place the order by making the transaction. The customer can check the delivery details, such as when they should expect their order to get delivered or can contact the delivery person for any problem. A unique filter is also provided for the customers to sort the products based on the minimum price available after different discounts.

The other mode in which users can log in is the Retailer, a manner for sellers to sell their products on the Virtual Bazaar. So for this mode also, the user must first enter the essential credentials to let them sell their kinds of stuff like Name, age, address, phone number etc. Next, a Retailer gets two options. First is whether they can add products to the store, and the other one is whether they can see the statistics of their products, like how many customers bought that particular product or should the retailer increase or decrease the quantities for selling and discounts.

So if the Retailer chooses the first mode to add the products to the store, they need to enter the technical details such as Product ID, Category ID, Price of the product, product Name, how much discount they want to provide, and the product quantity. And for the second mode, the portal will offer all possible statistics with one click on that option.

The last option to log in with which a user can enter is a Delivery Person. For the same, the user needs to enter all the details showing them on the screen, and then that delivery person will be assigned to a particular delivery as soon as the customer makes the transaction.

Further, some special filters are present on our portal with whose click a user can see different statistics. For example, Suppose a user chooses the statistical Section of the Virtual Bazaar. In that case, the first option a user gets to see is to see the statistics of the average price of products and categories by their product name and category name. The second option they get to see is the Maximum number of products ordered by their classes, along with their category name and product name. The third option they get is to know the average cost, which is grouped by category and product names. The fourth option a user receives is the total revenue grouped by category and product. And last but not least, a user can see the statistics of the product quantity and products' names by their average prices.

Transactions

1) **-> Retailer updates the product details(like product quantity) and the customer reads product details (including product quantity).**

T1: Retailer sees the product quantity and updates for a product, product_id=10(for example). (Admin can also do this.)

read(A)
A:=0
write(A)
commit

```
START TRANSACTION;
SELECT Product_Quantity FROM Product WHERE Product_ID=10;
UPDATE Product SET Product_Quantity = 0 WHERE Product_ID = 10;
COMMIT;
```

T2: Customer reads(sees) the product details for the same product that the retailer is changing details of (for Product_ID=100, here) and adds that product to his order (cart) (decreasing its quantity).

read(A)
A:=A-1
read(B)
B=B+1
write(A)
write(B)
commit

```
START TRANSACTION;
SELECT * FROM Product WHERE Product_ID=10;
INSERT INTO Cart(Cart_Product_ID, Cart_Customer_ID, Cart_product_quantity)
VALUES (10,1,1)
COMMIT;
```

Constraint: For product quantity=0, the customer can’t add it to his cart.

A: Product quantity of product with ID=100 in inventory(store)   //for example

B: Product quantity of product with ID=100 in customer’s cart

Conflict Non-Serializable Schedule
Schedule 1:

T1	T2
	read(A)
read(A)	
A:=0	
write(A)	
commit	
	A:=A-1
	read(B)
	B=B+1
	write(A)
	write(B)
	commit

T1	**T2**
	Req. S(A)
Req S(A)	
	Acquies S{A) and reads A
Acquies S(A) and reads A	
Sets value of A to 0	
Write new A and release S(A)	
Commit its transaction.	
	Subtracts 1 from the value of A
	Req X(B) and is granted.
	Reads B and adds 1 to the value.
	Writes the new value of B
	Release X(A)
	Release X(B)
	Finally commit their transcation

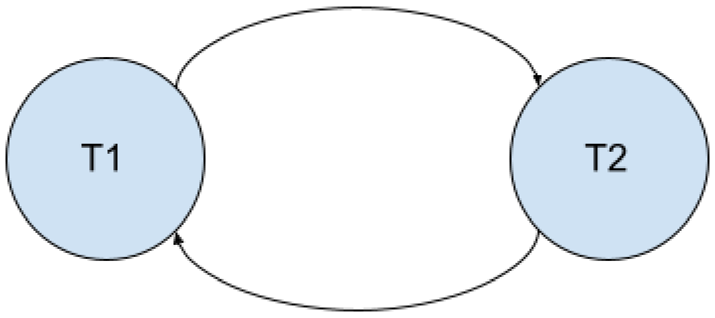
Stepwise functioning of T1 and T2:

- In this schedule, T1 and T2 both read the product quantity with read(A).
- Here, T1 represents the retailer changing the product quantity, setting it to 0.
- So, the retailer updates the product quantity to 0(before the customer has added the product to his), first, i.e. T1 makes A:=0 and write(A). As the product quantity is set to 0 it can't be decreased more, i.e. more of that product can't be purchased.
- But here, T2 is still working on the value read by T2 before (with read(A)).
- So T2 updates the quantity which was already set to 0 by A:=A-1 and write(A). Also, T2 meanwhile adds 1 to B (Product is added to customer's cart). So, there

is a conflict as T2 shouldn’t be able to decrease the value further (customer can’t buy a product whose quantity is set to 0 by retailer).

The given schedule is conflicting non- serializable because its precedence graph contains a cycle. A schedule is not conflicting serializable if its precedence graph has cycle In other words, a schedule is conflict serializable if and only if its precedence graph is acyclic.

But here if we draw the precedence graph, it contains a cycle.



So to ensure conflict serializability we can use the locking method. So the schedule now will look like:-

1. T1 acquires an exclusive lock on the order.
2. T1 cancels the order and then updates the product quantity information.
3. And then T1 releases the lock on the order.
4. Similarly T2 gets a ‘shared’ lock on the product page.
5. Now T2 reads the product page and hence gets the update product quantity information.
6. And then T2 also releases the lock on the product page.

So in the above process T1 is acquiring an exclusive lock which is preventing T2 from looking at the product page unti T1 transaction is performed. And then T2 is having a shared lock through which he is able to view the updated product quantities afet T1.

T1	**T2**

	Req. X(A) and acquires
	Reads A and sub 1 value of A
	Writes new value of A
	Releases X(A)
Req. X(A)	
Reads B and adds 1	
Writes new value of B	
Releases X(A)	
	Req. X(B) and has acquired
	Reads B and adds 1
	Writes new value of B
	Releases X(B)
	Then finally commit their transactions.

Conflict Serializable Schedule
Schedule 2:

S

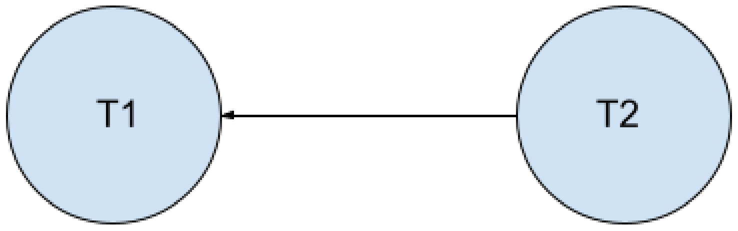
T1	T2
	read(A)
	A:=A-1
	write(A)
read(A)	
A:=0	
write(A)	
commit	
	read(B)

	B=B+1
	write(B)
	commit

Stepwise functioning of T1 and T2:

- Firstly T2 read(A)(customer reads the product details/quantity). And then T2 updates the quantity by A:=A-1 and write(A) (product quantity is decreased by 1 in inventory).
- T1 read the product quantity with read(A).
- Here, T1, A:=0, write(A) represents the retailer changing the product quantity, setting it to 0.
- Then, finally T2 adds 1 to B (Product is added to customer's cart) B=B+1, write(B).

The given schedule is conflicting serializable because its precedence graph is acyclic. A schedule is not conflicting serializable if its precedence graph has cycle In other words, a schedule is conflict serializable if and only if its precedence graph is acyclic.



Also above schedule can be converted to serializable using swapping. In swapping we swap non-conflicting steps in the schedule.

Swapping Non-Conflicting Steps we get:

S

T1	T2
	read(A)
	A:=A-1
	write(A)
	read(B)
	B=B+1
	write(B)
	commit
read(A)	
A:=0	
write(A)	
commit	

2) **-> Customer cancelling the order:- A customer has cancelled the order but at the same time another customer is reading that product page but did not get the updated quantity information after cancelling of the order by first customer.**

So there are two transactions happening in the above scenario and that's:-

T1:- The first transaction is the customer cancels an alread placed order.

START TRANSACTION;
DELETE from final_order where order_id={ord_id};
DELETE from delivery where delivery_order_id={ord_id};
UPDATE Product SET Product_Quantity =Product_Quantity+1 WHERE Product_ID = 10;
COMMIT;

T2:- The second transaction is another user is viewing details of the same product and adding it to his cart (including product quantity).

```
START TRANSACTION;
SELECT * FROM Product WHERE Product_ID=10;                    <for example>
INSERT INTO Cart(Cart_Product_ID, Cart_Customer_ID, Cart_product_quantity)
VALUES (10,1,1)
COMMIT;
```

A: A is a quantity of the product which was placed.
B: B is a quantity of the product in cart of second user.

Conflict Non-Serializable Schedule
Schedule 1:

S

T1	T2
	read(A)
read(A)	
A:=A+1	
write(A)	
commit	
	A:=A - 1
	write(A)
	read(B)
	B:=B+1
	write(B)
	commit

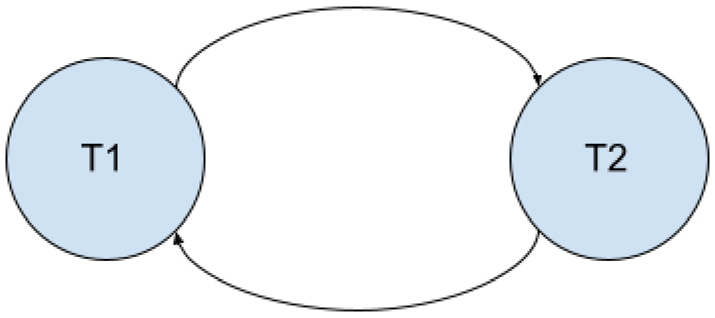
S is a schedule containing transactions T1 and T2 for quantities of the product first placed by the customer and then cancelled and then T2 views the product list.

- In this schedule, T2 read the product quantity with read(A).
- Now, T1 read(A)

- Customer 1 cancels the order so A gets increased by 1. T1, A:=A+1 , write(A).
- Now, here T2 which already read(A) buys a product,T2 updates the quantity which was already decreased by A:=A-1 and write(A). Here, if customer 2 knew that product quantity was going to be increased by 1 again he might have ordered 1 more product.
- Also, T2 meanwhile adds 1 to B (Product is added to customer's cart).

The given schedule is conflicting non- serializable because its precedence graph contains a cycle. A schedule is not conflicting serializable if its precedence graph has cycle In other words, a schedule is conflict serializable if and only if its precedence graph is acyclic.

But here if we draw the precedence graph, it contains a cycle.



Now to make the schedule conflict serializable, we need to ensure that the transactions are executed in a way that avoids any conflict. In this scenario, the conflict arises because T2 reads the product page before T1 has updated the quantity information.

So to ensure conflict serializability we can use the locking method. So the schedule now will look like:-

7. T1 acquires an exclusive lock on the order.
8. T1 cancels the order and then updates the product quantity information.
9. And then T1 releases the lock on the order.
10. Similarly T2 gets a 'shared' lock on the product page.
11. Now T2 reads the product page and hence gets the update product quantity information.

12. And then T2 also releases the lock on the product page.

So in the above process T1 is acquiring an exclusive lock which is preventing T2 from looking at the product page unti T1 transaction is performed. And then T2 is having a shared lock through which he is able to view the updated product quantities afet T1.

T1	**T2**
	Req. X(A) and acquires
	Reads A and sub 1 value of A
	Writes new value of A
	Releases X(A)
Req. X(A)	
Reads B and adds 1	
Writes new value of B	
Releases X(A)	
	Req. X(B) and has acquired
	Reads B and adds 1
	Writes new value of B
	Releases X(B)
	Then finally commit their transactions.

Conflict Serializable Schedule
Schedule 2:

S

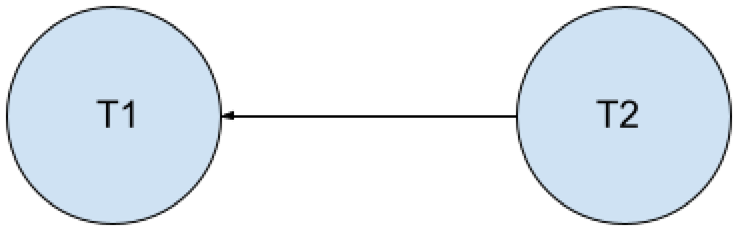
T1	T2
	read(A)
	A:=A - 1

	write(A)
read(A)	
A:=A+1	
write(A)	
commit	
	read(B)
	B:=B+1
	write(B)
	commit

Stepwise functioning of T1 and T2:

- Firstly T2 read(A)(customer reads the product details/quantity). And then T2 updates the quantity by A:=A-1 and write(A) (product quantity is decreased by 1 in inventory).
- T1 read the product quantity with read(A).
- Customer 1 cancels the order so A gets increased by 1. T1, A:=A+1 , write(A). Product quantity increases by 1.
- Then, finally T2 adds 1 to B (Product is added to customer's cart) B=B+1, write(B).

The given schedule is conflicting serializable because its precedence graph is acyclic. A schedule is not conflicting serializable if its precedence graph has cycle In other words, a schedule is conflict serializable if and only if its precedence graph is acyclic.



Also above schedule can be converted to serializable using swapping. In swapping we swap non-conflicting steps in the schedule.

Swapping Non-Conflicting Steps:

S

T1	T2
	read(A)
	A:=A - 1
	write(A)
	read(B)
	B:=B+1
	write(B)
	commit
read(A)	
A:=A+1	
write(A)	
commit	

3) **-> Reordering a new sale:- Two customers attempt to purchase the last available item of a product at the same time.**

T1:- The first transaction is the first customer places an order for a product with quantity =1

START TRANSACTION;

```
INSERT INTO Final_Order(Order_Customer_id,Order_product_id, Total_Charges,
Ordered_date) VALUES (1,7,10.5,'2023-03-26')
COMMIT;
```

T2:- The second transaction is the first customer places an order for a product with quantity =1

```
START TRANSACTION;
INSERT INTO Final_Order(Order_Customer_id,Order_product_id, Total_Charges,
Ordered_date) VALUES (2,7,10.5,'2023-03-26')
COMMIT;
```

A: Quantity of Product Available in Stock Currently

S

T1	T2
read(A)	
A=A-1	
write(A)	
commit	
	read(A)
	A=A-1
	write(A)
	commit

S is a schedule containing transactions T1 and T2 for product purchases by two customers.

T1 and T2 represent two transactions where two customers first see the quantity of product available, which is 1 for both and then want to buy a particular product. Here, if A represents the quantity of product available in the stock currently, then for both, the purchases of product A should decrease by 1 (it can also be taken a different value. For example, if Customer 1 buys five and Customer 2 buys two, but the meaning of transaction remains same).

But here, there can be a conflict between two transactions when Customer 2 sees there is a product available, they tries to buy it, but meanwhile, it is already been bought by Customer 1 and the product quantity is now 0. So, in that case, Customer 2 shouldn't be able to buy that product. In other words, if T2 reads (A) before T1 write(A), there can be a conflict.

Further, in this example:

Non - Serializable conflict

S

T1	T2
read(A)	read(A)
A:=A-1	
write(A)	
commit	A:=A-1
	write(A)
	commit

If both the users enter the products section at the same time and both see product quantity as 1. i.e. T1 and T2 read(A) at the same time.

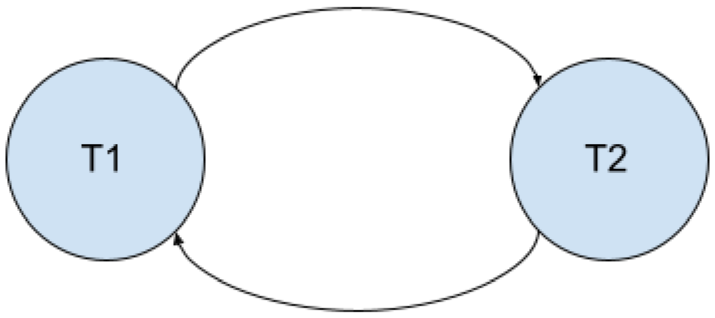
But then, if Customer 1 adds the product to his cart, which changes the product quantity to 0. I.e. T1 changes "A" to "A-1" (which means A=0 as A was 1 initially), and T1 write(A).

So now, if Customer 2 (which reads A=1), tries to add the product to his cart, which also executes "A = A-1" operation (i.e. T2 changes A to A-1 and T2 writes (A)). This will cause a conflict, because A has already changed to 0 from 1.

So now, to check if it is conflict non-serializable or serializable, we can draw a precedence graph. A non-serial schedule is conflict serializable if it can convert into a serial schedule by swapping its non-conflicting operations.

A schedule is not conflict serializable if its precedence graph has cycle In other words a schedule is conflict serializable if and only if its precedence graph is acyclic.

But here if we draw the precedence graph, its cyclic hence, its non conflicting serializable.



Serializable Conflict

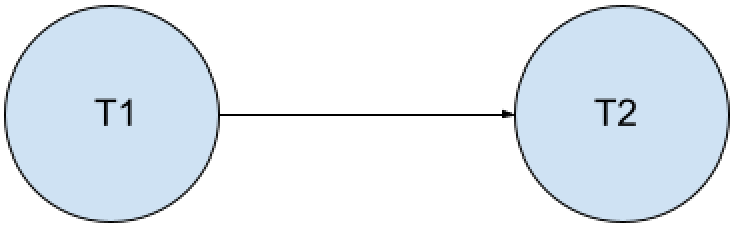
S

T1	T2
read(A)	
A:=A-1	
write(A)	
commit	
	read(A)
	A:=A-1
	write(A)

	commit

A schedule is not conflict serializable if its precedence graph has cycle In other words a schedule is conflict serializable if and only if its precedence graph is acyclic.

As precedence graph here is acyclic its, conflicting serializable.



4) **-> Customer is updating his billing address after placing the order and the delivery person has the order or information to deliver it to his old address since he couldn't get the updated address once the order was placed.**

T1:- the address being updated by the customer after placing the order.

START TRANSACTION;
UPDATE customer set district= "Okhla" WHERE Customer_ID=102;

COMMIT;

T2:- The delivery person reads(sees) the address he has after customer placed the order(which is still the initial address of customer) and delivers the product.

START TRANSACTION;
SELECT district FROM Customer WHERE Customer_ID=102;

COMMIT;

A:- Customer updating the address after placing the order.

B:- Delivery Status of product: delivered

Conflict Non-Serializable Schedule
Schedule 1:

S

T1	T2
read(A)	
	read(A)
write(A)	
	write(B)

We are unable to swap instructions in the above schedule to obtain the serial schedules and hence it is a conflict non-serializable schedule.

Conflict Non Serializable Schedule
Schedule 2:

S

T1	T2
read(A)	
write(A)	
	read(A)
	write(B)

So the above schedule is a conflict serializable since it is mainatinaing the consistency of the database. So in the above case T1 is updating the address while T2 is reading the same address after T1 writes it out. And hence it is a conflict serializable.

The given schedule is conflicting serializable because its precedence graph is acyclic. A schedule is not conflicting serializable if its precedence graph has cycle In other words, a schedule is conflict serializable if and only if its precedence graph is acyclic.