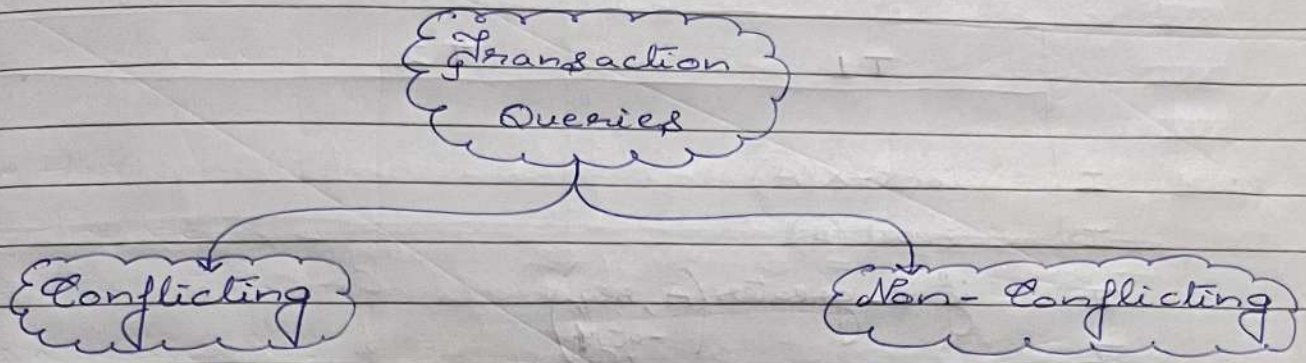
Project Deadline - 6




A. Conflicting Transaction Queries

Query 1:

Two Customers trying to add the same product in their shopping carts at the same time.

** What can be the Issue?

If, 2 customers are trying to add the same product, which is limited in stock, at the same time may cause an issue, where both of them are able to add the product in their cart even though the quantity of the product may not be sufficient with respect to their requirements. This will lead to an error in the database & hence, making it a conflicting transaction query.

Sol$^{n}$ :→ Serialization can be used in this scenario (Assuming this scenario is quite common) & hence, is the most effective approach.

| | T1 | T2 |
|---|---|---|
| I | lock-X(A) | |
| II | read(A) | |
| III | if is-greater(A, req. -qty1) | |
| IV | | |
| V | A := A - reqqty1 | |
| VI | write(A) | |
| VII | unlock(A) | |
| VIII | commit | |
| IX | | lock-X(A) |
| X | | read(A) |
| XI | | if is-greater(A, reqqty2) |
| XII | | A := A - reqqty2 |
| XIII | | write(A) |
| XIV | | unlock(A) |
| XV | | commit |

Notations :⇒ 'A' denotes the stock / quantity of the Product which the 2 customers are trying to add.

'reqqty1' & 'reqqty2' denote the required product quantity of both the customers, respectively.

'is-greater' is a function which returns whether the 1st parameter is greater than or equal to the 2nd parameter.

Query 2:

The seller of the product trying to update the price of the item while at the same time, a customer tries to buy it (checkout cart).

** What can be the issue?

If a seller of the product tries to update the price of the item & a customer tries to checkout the cart with the same item at the same time, an issue may arise where the price is not updated and may lead to inconsistency in the database & hence, making it a conflicting transaction query.

Soln :→

| | T1 | T2 |
|---|---|---|
| I | | lock-S(P) |
| II | lock-X(P) | read(P) |
| III | read(P) | old_price := P |
| IV | P := New_Price | unlock(P) |
| V | write(P) | |
| VI | unlock(P) | |
| VII | commit | |
| VIII | | lock-S(P) |
| IX | | read(P) |
| X | | temp := P |
| XI | | unlock(P) |
| XII | | |

| | |
|---|---|
| XIII | read (C) |
| XIV | C := C + (temp - old-price) * qty |
| XV | write (C) |
| XVI | commit |
| XVII | |
| XVIII | |
| XIX | |

#Here 'qty' => Quantity of Product (whose price is updated) in customer's cart.

Steps :->

* Transaction (T2) denotes the one referring to the customer's cart & the first step is to use a shared lock on the Product whose original price is stored in 'old-price'.

* In the next few steps, after unlocking the shared resource, Transaction (T1) pertaining to the seller puts an exclusive lock on the product whose price needs to be updated.

* Once the exclusive lock is released from the product, Transaction (T2) applies the shared lock to the product & stores the updated price of the product in 'temp'.

* Once the shared lock is released, Transaction (T2) reads the cart Total of the customer & updates the total by writing it on the stored resource, & finally commits those changes.

—x—x—x—x—x—x—x—x—

B. Non-Conflicting Transaction Queries

Query 1:

Increasing the stock/quantity of a product whose quantity has fallen below a certain threshold.

**Why is the query not/non-conflicting?

Since, increasing the stock/quantity of a product whose current quantity is below a certain threshold won't affect/cause an issue to any other transaction query, hence it is categorised as 'Non-conflicting'.

Implementation in MySql :⇒

```
use hawerkstore;

start Transaction;
Update product p
set p.qty=150
where p.qty<10 and p.id in
(select m.product_id from map_sellers m
where p.id = m.product_id and m.seller_id in
(select seller.id from seller
 where m.seller_id = seller.id)
);
Commit;
```

Query 2:

Updating the Phone No. of a customer with a given name & phone no.

** Why is the query non-conflicting?

Since, the transaction query updates the personal information of a customer & does not hinder with any other transaction, hence, it is 'Non-Conflicting'.

Implementation in MySql :⇒

```
use hawarkstore;

Start Transaction;
Update customer_phone
set phone = '8542687456'
where phone = '6177222794' and id in
(select id from customers where
first_name = 'Aleda' and last_name = 'Le Grys');

Select customers.id, phone from customer_phone
join customers
on customers.id = customer_phone.id
where first_name = 'Aleda' and last_name = 'Le Grys';
Commit;
```

Query 3:

Adding / Hiring a New Delivery Man to a given warehouse.

** Why is the query non-conflicting?

Since, adding a new delivery man entry to a database for a given warehouse won't hinder any other transaction, hence, it is 'Non-Conflicting'.

Implementation in MySql :⇒

```
use haurwkstore;

Start Transaction;

Insert into delivery_man (warehouse_id, first_name
, last_name, phone)
values (2, 'Honey', 'KaKKar', 9182730465);

Commit;
```

x——x——x——x——x——x——x——x——x—

Query 4:

The Admin managing the Online Retail Store deleting a Category from the Category Table:

** Why is it non-conflicting?

This query might look like a conflicting one on the first glance but deleting a given category from the category table won't hinder other transactions. Moreover, no inconsis-tencies will be arised in the database since customers having products of this category in their ~~carts~~ pending orders will still get the product, but ~~the~~ customers won't be able to add new products from this category to their carts once the action is performed. Hence, it is a 'Non-Conflicting' Transaction.

Implementation in MySql :⇒

```
use hourwkstore;

Start Transaction;

Delete from category
where name = 'stationary';

Commit;
```