<u>Transactions</u>

Name- Udit

Roll No-2021213

Non Conflicting transaction:
select* from invoice;
<u>-- Non-Conflicting Transactions:</u>

-- 1)
Two users one is admin and other is customer:
Now the admin is updating the price of a product with product id = 10.
And user is purchasing a product id = 20
Both are accessing the product table but they are accessing different product
Start transaction;
Select * from product where product_id=10 //Read from product table
Update product set price=1000 where product_id=10 //write in product table
Commit;
Start transaction;
Select * from product where product_id=20 // read from product table
INSERT INTO ProductToPatient (Invoice_ID, Patient_ID, Product_ID) VALUES (200, 100, 20, 3); //write in ProductToPatient
Commit;

-- 2) Updating salary of an employee 6 but unfortunately we have done it wrongly by updating salary of employee 5 so I roll back and again update the salary of employee 6 and then commit

Case 1)
Start transaction;
select * from employee where Employee_id=5; //read from employee
update employee set salary=50000 where employee_id=5; // write in employee
select * from employee where Employee_id=5; //read from employee
Rollback; //changes got reversed
select * from employee where Employee_id=5;

select * from employee where Employee_id=6; //read from employee
update employee set salary=50000 where employee_id=6; //write in employee
select * from employee where Employee_id=6;
Commit; // corrected the changes
Case 2)
Start transaction;
select * from employee where Employee_id=5; //read from employee
update employee set salary=50000 where employee_id=5; // write in employee

select * from employee where Employee_id=5; //read from employee
Commit; //We have committed the changes but realized later that whatever we have done was wrong

Now we can do nothing except updating the salary of employee 5 and 6 again if we have backup of that.

-- 3)
Patient buys some medicine for 3 days but the pharmacist found that the medicine will expire tomorrow. At the last moment he decided to roll back the transaction from invoice and sales table but still reduce stock of that medicine as that medicine is of no use now as it is strictly prescribed for 3 consecutive days always.

Start transaction;
Select * from product where product_id=50; // Read from product table
INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (100, Amount, 'SBI', NOW() , 10); // Write in Invoice table
insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values invoice_id,100, 50 , 3) // Asking quantity of 3 doses for 3 days.(Write in producttopatient table)
rollback;
//but then he found the medicine would expire in 1 day.
Hence, rollback the transaction

– 4) There are 3 users who are viewing the same product 96 in the View product table.

Transaction A	Transaction B	Transaction C
Start transaction;		
Select * from product where product_id = 96 // read from product table	Start transaction;	
commit;	Select * from product where product_id = 96 // read from product table	Start transaction;
		Select * from product where product_id = 96 // read from product table commit;
	commit;	

In the above shown transactions, there is no conflict (as Read Read is only used) whenever these transactions start or end or even roll back for the same product as only read statements are involved .

<u>-- Conflicting transactions</u>

-- 1) If one employee updates the quantity of a product and then some other also update the quantity of product then there is a conflict

Start transaction; // B transaction starts
select * from product where product_id=31;INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (100, Amount, 'SBI', NOW() , 10);
insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values invoice_id,100, 31 , 10)   // Asking quantity of 10
Commit; // transaction B got committed

(Non-conflict serializable schedule)

Transaction A	Transaction B

**Start transaction; // A transaction starts
select * from product where product_id=31; // Read from product table
update product set stock=400 where product_id=31; //Write in product table**	

	**Start transaction; // B transaction starts
	select * from product where product_id=31; // <u>Problem of Dirty read</u> Read from product table
	
	INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (100, Amount, 'SBI', NOW() , 10);
	/Write in invoice table**

      Read product 31 information then Some system failure happens which cause failure of this transaction Rollback; // Transaction A ends	**insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values invoice_id,100, 31 , 10) // Asking quantity of 10 Commit; // transaction B got committed**

// But due to some failure or may be admin by mistakenly change the price of wrong product
// he needs to roll back his transaction A

But in between some users came and did some transactions on the wrong value.

To avoid such situations, we can use exclusive lock on transaction A, which will only get unlocked when transaction A is either committed or rolled back.

Above schedule is non conflict serializable as we can see the cycle:
Transaction A—-> Transaction B —--> Transaction A
Corrected Transaction:

Start transaction; // A transaction starts
select * from product where product_id=31;
update product set stock=400 where product_id=31;
Rollback; // Transaction A ends

Start transaction; // B transaction starts
select * from product where product_id=31;
INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (100, Amount, 'SBI', NOW() , 10);
insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values invoice_id,100, 31 , 10) // Asking quantity of 10
Commit; // transaction B got committed

(conflict serializable schedule)

Transaction A	Transaction B

Exclusive lock transaction A
Start transaction; // A transaction starts
select * from product where product_id=31; // read from product
update product set stock=400 where product_id=31; // update product table
Rollback; // Transaction A ends
Release lock A
	Transaction B comes in the queue, waits until transaction A releases the lock.
	
	Exclusive lock transaction B
	Start transaction; // B transaction starts
	select * from product where product_id=31; // <u>Problem of Dirty read</u>
	// read from product
	
	INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (100, Amount, 'SBI', NOW() , 10); //write in invoice
	insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values invoice_id,100, 31 , 10) // Asking quantity of 10
	Commit; // transaction B got committed
	
	Release lock from transaction B

Above schedule is conflict serializable as we can see there is no cycle:
Transaction A—-> Transaction B
–2)
A user purchased a product but the quantity asked was less than the stock value, hence the remaining transaction would be aborted and products rollback into our inventory, but at the same time another user came and asked for the same product as User 1 has selected some products, user 2 unable to buy those product at that time but after rollback of transaction user 1, user 2 can take those products.

Initial stock of product 3 => 5
User A demands for product 3 of quantity => 10
-> this was unavailable, hence the transaction got aborted
User B demands for product 3 of quantity => 2
-> this was unavailable initially as user A had held that product but after his transaction got aborted those products got rolled back into the database and user B was able to take that product.

(Non-conflict serializable schedule)

Transaction A	Transaction B

Start transaction; // User A
 	
INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (101, Amount, 'HDFC', NOW() , 10);
Write in invoice;
insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values (invoice_id, 100, 3, 2) // Asking quantity of 2
Write in producttopatient (sales table)	

	Start transaction; // User B
	INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (101, Amount, 'HDFC', NOW() , 10);
	Write in invoice;
	insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values (invoice_id, 100, 3, 2) // Asking quantity of 2
	Not Available as user A had held those product
	Rollback; //transaction B ends
	Write in producttopatient (sales table)

Transaction A recontinue **Not Available as user A had held those product** **Rollback;**	

Above schedule is non conflict serializable as we can see the cycle:
Transaction A—-> Transaction B —--> Transaction A

Corrected Transaction:

But we can clearly see that transaction B was very much possible, and this was write write conflict that is a big problem for our database.

To avoid this, whenever we write(insert, update and delete) we can apply an exclusive lock to that transaction.

(conflict serializable schedule)

Transaction A	**Transaction B**

Apply exclusive lock
Start transaction; // User A
 	
INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (101, Amount, 'HDFC', NOW() , 10);
Write in invoice;
 	
insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values (invoice_id, 100, 3, 2) // Asking quantity of 2
 	
commit;
Release exclusive lock
	Transaction B comes in queue; // User B
	Waits until lock from A got released
	
	Start transaction B;
	Apply exclusive lock ;
	INSERT INTO Invoice (Patient_ID, Amount, Mode_of_Payment , time_of_payment, Branch_ID) VALUES (101, Amount, 'HDFC', NOW() , 10);
	Write in invoice;
	
	insert into producttopatient (invoice_id,Patient_Id,product_Id,Quantity) values (invoice_id, 100, 3, 2) // Asking quantity of 2
	Write in producttopatient (sales table)
	
	Not Available as user A had held those product
	Rollback; //transaction B ends

	Release exclusive lock;

Above schedule is conflict serializable as we can see there is no cycle:
Transaction A—-> Transaction B