Transactions in speedX

Tanishq Dass (2022533), Rohan Devgon (2022414), Mehraz Abedin Raz (2022293)

April 18, 2024

4 Non-Conflicting Transactions:

4 transactions accessing or manipulating different tables allowing them to proceed simultaneously without the need for coordination or synchronization.

```
186   -- T1
187   begin;
188
189   UPDATE store_inventory
190   SET quantity = 20
191   WHERE store_ID = 1 AND product_ID = 1;
192   commit;
193
194   -- T2
195   begin;
196
197   select * from discount where discount_ID = 4;
198   UPDATE discount
199   SET discount_percent = 20
200   WHERE discount_ID = 4;
201   commit;
202
203   -- T3
204   begin;
205
206   INSERT INTO customer (customer_ID, customer_name, customer_password, contact_number, customer_address, customer_email_id, customer_pincode)
207   VALUES (11, 'Tanishq Dass', 'passwwwrd', '9876543231', '126 Main St', 'tanishq@example.com', '110032');
208   commit;
209
210   -- T4
211   begin;
212   select *
213   from rating
214   where rating = 4;
215   commit;
```

3 Conflicting Transactions:

```
220
221     -- T1
222 ●   begin;
223 ●   select * from customer where customer_ID = 30;
224 ●   update customer set customer_pass = "newpass" where customer_ID = 30;
225 ●   commit;
226
227     -- T2
228 ●   begin;
229 ●   update customer set contact_number = "9999900000" where customer_ID = 30;
230 ●   commit;
231
232     -- T3
233 ●   begin;
234 ●   INSERT INTO customer (customer_ID, customer_name, customer_password,
235         contact_number, customer_address, customer_email_id, customer_pincode)
236     VALUES
237     (30, 'Tanishk Singh', 'password123', '9876543266',
238     '167 Main St', 'tanishk4@example.com', '110001');
239 ●   commit;
240
```

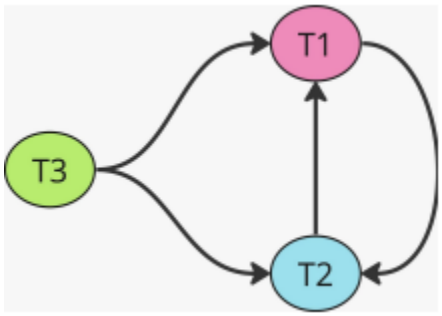
Non-Conflict Serialisable Schedule:

STEP	T1	T2	T3
1.	START		
2.		START	
3.			START
4.			write(Customer)
5.			COMMIT
6.	read(Customer)		
7.		write(Customer)	
8.		COMMIT	
9.	write(COMMIT)		
10.	COMMIT		

Operation Table:

STEP	TRANSACTIONS	OPERATIONS	DATA VALUE
1.	T1	START TRANSCTION;	-
2.	T2	START TRANSACTION;	-
3.	T3	START TRANSACTION;	-
4.	T3	`INSERT INTO customer (customer_ID, customer_name, customer_password, contact_number, customer_address, customer_email_id, customer_pincode) VALUES (30, 'Tanishk Singh', 'password123', '9876543266', '167 Main St', 'tanishk4@example.com', '110001');`	All
5.	T3	COMMIT;	-
6.	T1	`select * from customer where customer_ID = 30;`	All
7.	T2	`update customer set contact_number = "9999900000" where customer_ID = 30;`	contact_number
8.	T2	COMMIT;	-
9.	T1	`update customer set customer_pass = "newpass" where customer_ID = 30;`	customer_pass
10.	T1	COMMIT;	-

Precedence Graph:



As we can observe, G(V', E') where V' = {T1,T2} there exists a cycle in the subgraph. Thus, the schedule is non-conflict serialisable.

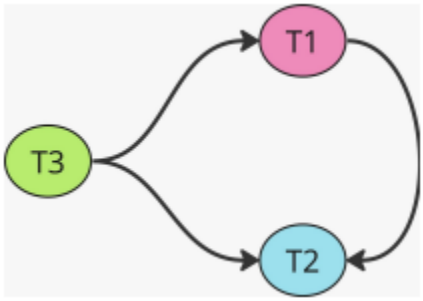
Conflict Serialisable Schedule:

STEPS	T1	T2	T3
1.	START		
2.		START	
3.			START
4.			write(Customer)
5.			COMMIT
6.	read(Customer)		
7.	write(Customer)		
8.	COMMIT		
9.		write(Customer)	
10.		COMMIT	

Operation Table:

STEPS	TRANSACTIONS	OPERATIONS	DATA VALUE
1.	T1	START TRANSACTION;	-
2.	T2	START TRANSACTION	-
3.	T3	START TRANSACTION	-
4.	T3	`INSERT INTO customer (customer_ID, customer_name, customer_password, contact_number, customer_address, customer_email_id, customer_pincode) VALUES (30, 'Tanishk Singh', 'password123', '9876543266', '167 Main St', 'tanishk4@example.com', '110001');`	All
5.	T3	COMMIT;	-
6.	T1	`select * from customer where customer_ID = 30;`	All
7.	T1	`update customer set customer_pass = "newpass" where customer_ID = 30;`	customer_pass
8.	T1	COMMIT;	-
9.	T2	`update customer set contact_number = "9999900000" where customer_ID = 30;`	contact_number
10.	T2	COMMIT;	-

Precedence graph:



As observed, the graph is acyclic. Thus, the schedule is serialisable.