Conflicting Transactions and User Guide

Anirudh S. Kumar
Roll Number - 2021517
IIIT - Delhi
anirudh21517@iiitd.ac.in

Aakarsh Jain
Roll Number - 2021507
IIIT - Delhi
aakarsh21507@iiitd.ac.in

April 24, 2023

1 Selling a Product from a Retailer

Consider the transaction T with two steps S_1 and S_2 as follows

- S_1: Reading the remaining product quantity from the retailer
- S_2: Changing the value of the remaining product quantity after order delivery.

1.1 Conflict Serializable Schedule

Now consider two such transactions, T_1 and T_2, in the following schedule.

1. $S_1^{T_1}$ - T_1 reads from the database
2. $S_2^{T_1}$ - T_1 writes to the database
3. $S_1^{T_2}$ - T_2 reads from the database
4. $S_2^{T_2}$ - T_2 writes to the database

We can see that in this schedule, changing the ordering between the transactions does not affect the final value in the database. Therefore, this is an example of a conflict serializable schedule.

1.2 Non-Conflict-Serializable Schedule

Now consider the same two transactions, T_1 and T_2, in the following schedule.

1. $S_1^{T_1}$ - T_1 reads from the database
2. $S_1^{T_2}$ - T_2 reads from the database
3. $S_2^{T_2}$ - T_2 writes to the database
4. $S_2^{T_1}$ - T_1 writes to the database

This is an example of a non-conflict serializable schedule. Consider the scenario where the initial quantity in the database was 20

1. $S_1^{T_1}$ - T_1 reads from the database $\rightarrow$ T_1 **reads 20**
2. $S_1^{T_2}$ - T_2 reads from the database $\rightarrow$ T_2 **reads 20**
3. $S_2^{T_2}$ - T_2 writes to the database $\rightarrow$ T_2 **buys 5 and updates the value to 15 in the database**
4. $S_2^{T_1}$ - T_1 writes to the database $\rightarrow$ T_1 **buys 2 and updates the value to 18 in the database**

If we were to reverse the order of operations, we would see that the final value in the database would be 15 instead. Therefore, this is a non-conflict serializable schedule.

2 Buying a deleted product

Consider two such transactions T_1 and T_2 each of them with two steps as follows :-

- T_1
  - S_1 - Reading the remaining product quantity of product P from the retailer
  - S_2 - Changing the value of the remaining product quantity P after order delivery.
- T_2
  - S_1 - Fetching the details of the product P
  - S_2 - Removing it from the inventory list.

2.1 Conflict Serializable Schedule

Now consider two such transactions, T_1 and T_2, in the following schedule.

1. $S_1^{T_1}$ - T_1 checks for availability of product P in the databse
2. $S_2^{T_1}$ - T_1 order the product P if available
3. $S_1^{T_2}$ - T_2 fetches the product P from the database
4. $S_2^{T_2}$ - T_2 deletes the product P from the database.

If we reverse the ordering of execution of T_1 and T_2, then T_1 will abort before it orders the product, so data integrity remains.

2.2 Non-Conflict-Serializable Schedule

Now consider the same two transactions, T_1 and T_2, in the following schedule.

1. $S_1^{T_1}$ - T_1 checks for availability of product P in the databse
2. $S_1^{T_2}$ - T_2 fetches the product P from the database
3. $S_2^{T_2}$ - T_2 deletes the product P from the database.
4. $S_2^{T_1}$ - T_1 order the product P if available

This is a case of a non-conflict-serializable schedule. In the first step, T_1 thinks that the product exists in the database and is available for ordering, but then T_2 fetches and deletes the product, causing the second step of T_1 to cause inconsistencies.

3 User Guide

- Login/Signup - Click Signup and enter the relevant details. After signup, click the Login button to log in to the site.
- Booking Product - First Login/Signup. After that Go to Products → Pick a Category → Pick a Product → Add to cart → Cart → Checkout → Add Details → Place Order.
- Booking Lab Tests - First Login/Signup. Afterwards, Go to Labs → Pick a test → Buy Now → Add Details → Place Order.