DBMS Transaction Issues Discussion

Introduction:

In assignment 1&2, we designed a Hotel Management System (HMS) with Java and MySQL DBMS. However, we did not have considered synchronisation issue in the program. For example, if multiple clients want to use this system to book rooms simultaneously, the errors could happen. It is noticeable that multiple threads support is necessary for this system, and DBMS should also support this mechanism. In reality, DBMS provide this supporting named as "Transaction". In this report, the effects of lacking transaction support for the HMS would be discussed in detail, in terms of room booking and meal booing.

Issue 1: Room Booking

If there is no transaction support, when two guests are booking the same room for an overlapping time interval, the invalid data is possible to be insert into the "BookedRoom" table in my database. To be consistent with analysis data, I would adopt the answer of table design for coursework 1&2. Meanwhile, the data for analysis is "guests are booking room 803 and want to stay from '2020-5-12' to '2020-5-18'". The **table "RoomArrangement" is simplified as "R" table** in the following analysis, and the **processing data procedure of room ID and check-in & check-out date would be presented as "Process(data)"** in the transaction chart. This is because the room booking data should be checked at first in the program and then insert into table to maintain data validity. The first condition shows as below.

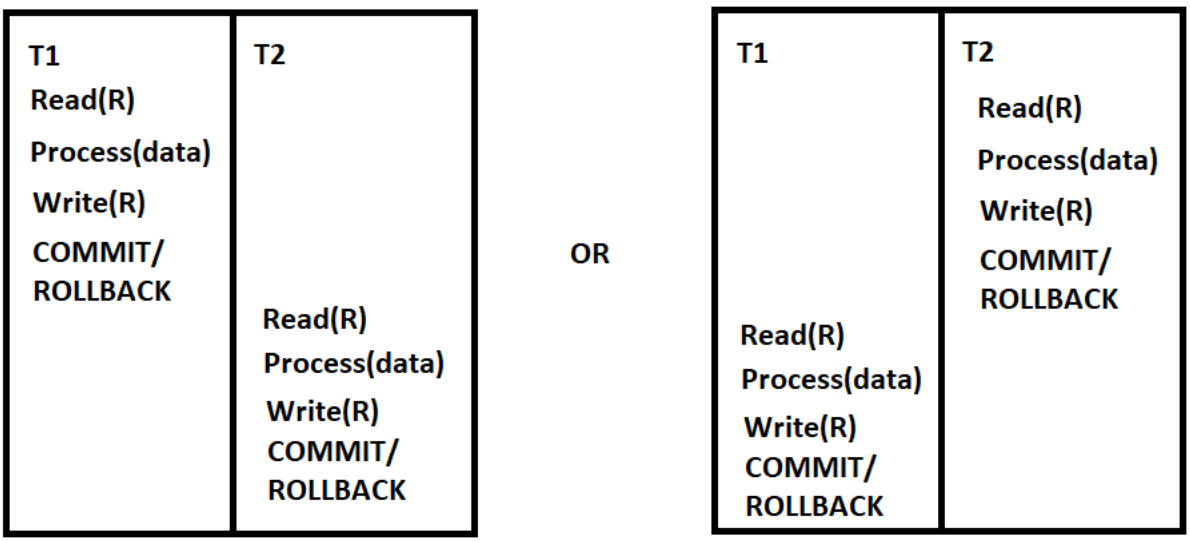

Figure 1: The First Situation of Room Booking Concurrency

In this case, we can notice that T1 and T2 are not conflict with each other, they can book or cancel the cancel room booking correctly. If T1 booked room first, then T2 must rollback, vice versa. If T1 cancel the room booking, then T2 is optional to book this room or cancel, vice versa. The "RoomArrangement" will be

no updates if both of them cancel the room booking, but the successfully booked room tuple should present as below table.

Table 1: The Tuple After "Serialized" Room Booking from Guests

BookingID	FloorLvl	RoomNo	RoomType	StartDate	EndDate	GuestEmail	PassportID
……	8	03	Large single bed	2020-05-12	2020-05-18	(From A/B)	…

If these two transactions distributed as below, the "Lost Update" issue could happen.

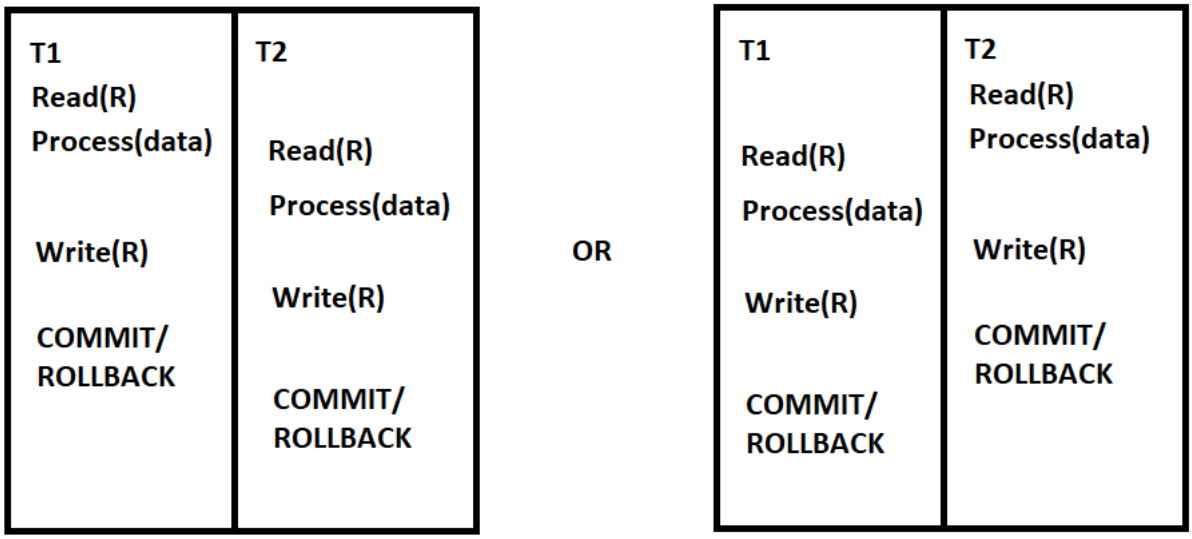

Figure 2: The Second Situation of Room Booking Concurrency

Due to the essential issue is the same, we only concentrate on the left transaction processing flow. If both T1 and T2 read data from "RoomArrangement" table at approximately the same time, the processing data would not include the data from another transaction. Therefore, both T1 and T2 transactions could insert the room booking records into the table and cause data invalid happen. If both of T1 and T2 choose commit at the end, the tuples in the table show as below.

Table 2: The Tuple After "Intersection" Room Booking from Guests

BookingID	FloorLvl	RoomNo	RoomType	StartDate	EndDate	GuestEmail	PassportID
……	8	03	Large single bed	2020-05-12	2020-05-18	(From A/B)	…
……	8	03	Large single bed	2020-05-12	2020-05-18	(From A/B)	…

This is invalid an operation, it could be defined as "Lost Update". The result could be valid if and only if

either T1 or T2 choose to rollback at the end, or both of T1 and T2 rollback at the end when lacking transaction support. If both T1 and T2 rollback, the table would not be updated. If either T1 or T2 rollback, the new tuple presented as **Table 1**.

If these two transactions distributed as below, the "Uncommitted Update" and "Inconsistent analysis" combined issue might happen.

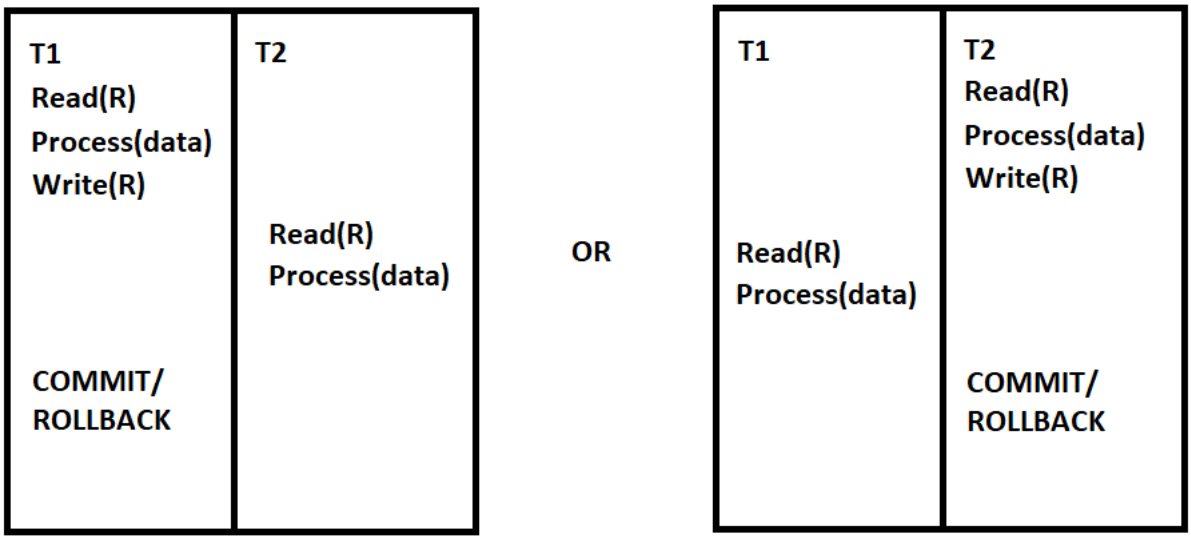

Figure 3: The Third Situation of Room Booking Concurrency

In this process, we can notice that in the left transaction process, if T1 book the room at first, then the analysis result of T2 would be "803 room is not available in this period". If T1 commit his or her order to the database, the data would be valid. However, if T1 rollback his or her room booking order, the analysis result from T2 would be invalid at that moment. That means T2 cannot book the 803 room in the specified time interval if T1 not finish his or her transaction. If T1 commit the room booking order, the tuple updates result should be presented as **Figure 1**.

Issue 2: Meal Booking

In the meal booking process, we need to check the workdays of chefs, this process would be done in the program. It seems that there is no other limitation of dishes booked by either one guest or multiple guests. However, if the hotel needs Administrator of this database to update the workdays of chefs or dishes of chefs, there might have problems. If the Administrator updates the dishes or workdays and a guest book the new inserted dishes or set the delivery time based on this updated table, then Administrator rollback the updates. The meal booking order become invalid because of the dirty read. The **operated tables would be displayed as "M&C" (Meal & Chef)** in transaction processing chart. The **updates of Administrator**

would be presented as "Update(M&C)" in the following chart, and the **processing produce would be shown as "Process(data)"** in this chart. The transaction chart of these process presents as below.

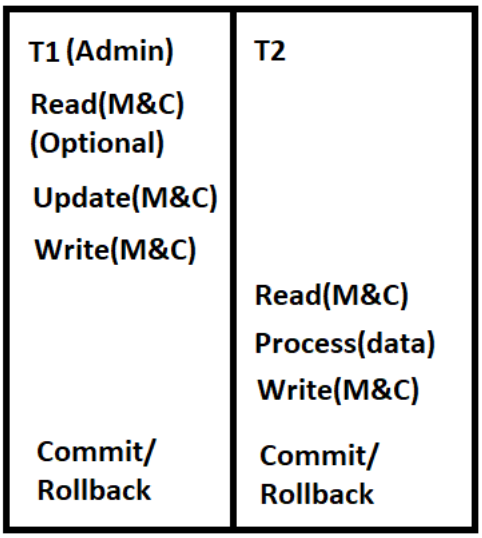

Figure 4: The Situation of Meal Booking Concurrency

We can notice that if a guest book the meal before or after the Administrator updating, this issue will not exist. If Administrator commits the final updates to the database or the final meal booking order has been rollback by guests, this issue will also not happen. However, if the Administrator rollback these updates and guests commit their meal booking order, that will cause dirty read on guests. The added tuples on the "MealReservations" would become invalid data. This tuple in the table should present as below.

Table 3: The Invalid Tuple Caused by "Uncommitted Update" of Meal and Chefs

ReservationID	FloorLvl	RoomNo	DeliveryTime	BookTime(optional)
(Uncommitted data)	…	…	…	…

Conclusion:

In this report, we have discussed effects of lacking transaction support on this HMS. The issues caused on room booking involves "Lost Update" issue and "Uncommitted Update & Inconsistent analysis" issue. It is noticeable that lacking transaction support in a concurrency needed system is fatal. We have multiple choices, the serializable is the most robust one. However, we do not have experienced the process of waiting all other users finish their transaction and then start the transaction of us. In reality, we can apply the "Schedule" and "Precedence Graph" analysis to confirm whether different transactions could operate normally at the same time. We can also adopt locking mechanism such as "Two-Phase Locking" to these tables, and prevent long time waiting in serializable condition. However, it also has some issue such as

deadlocks. For the DBMS of assignment 1&2, MySQL InnoDB engine provides transaction isolation mechanism, we can set each transaction isolation level as “read-uncommitted”, “read-committed”, “repeatable-read”, or “serializable”. I also adopt this mechanism to provide concurrency supporting in assignment 1&2. To sum up, we have numbers of approaches to solve concurrency issue in a program, but the most appropriate mechanism could only be decided base on the specific requirements.