DBMS Project Deadline-6

Yash Singh 2022589
Shrey Yadav 2022483
Tushar Agrawal 2022545

April 20, 2024

Non-Conflicting Transactions

Transaction Pair 1: Update Different Property Information

This scenario involves two transactions where each user updates different properties independently, ensuring there is no conflict between the operations.

Transaction-1 (T1) on Property P1	**Transaction-2 (T2) on Property P2**
Read(P1 Description)	Read(P2 Price)
Update P1 Description from "Old Desc 1" to "New Desc 1"	Update P2 Price from 200000 to 210000
Write(Updated P1 Description)	Write(Updated P2 Price)
Commit	Commit

Table 1: Transactions updating different properties in parallel without any conflict.

Transaction-1 (T1) on Property P1	**Transaction-2 (T2) on Property P2**
Read(P1 Description)	
-	Read(P2 Price)
Update P1 Description from "Old Desc 1" to "New Desc 1"	-
-	Update P2 Price from 200000 to 210000
Write(Updated P1 Description)	-
-	Write(Updated P2 Price)
Commit	-
-	Commit

Table 2: Interleaved execution of transactions updating different properties.

Transaction Pair 2: Update and Remove Property from Sale

This scenario involves two transactions where T1 updates a property's description and T2 removes a different property from sale, demonstrating the non-conflicting nature of concurrent updates to different records.

Transaction-1 (T1) on Property P1	**Transaction-2 (T2) on Property P3**
Read(P1 Description)	Read(P3 AvailabilityStatus)
Update P1 Description from ”Old Desc” to ”New Desc”	Update P3 AvailabilityStatus from True to False
Write(Updated P1 Description)	Write(Updated P3 AvailabilityStatus)
Commit	Commit

Table 3: Transactions handling updates on description and sale status on different properties.

Transaction-1 (T1) on Property P1	**Transaction-2 (T2) on Property P3**
Read(P1 Description)	-
Update P1 Description from ”Old Desc” to ”New Desc”	–
Write(Updated P1 Description)	-
Commit	-
-	Update P3 AvailabilityStatus from True to False
-	Write(Updated P3 AvailabilityStatus)
-	Commit

Table 4: Interleaved execution showing non-conflicting updates to property description and availability status.

Transaction Pair 3:Two Users Deleting Their Properties

This scenario involves two users simultaneously deleting their properties from the system without conflict.

User 1 Transaction: Delete Property	**User 2 Transaction: Delete Property**
Delete Property P1 Instance from Database	Delete Property P2 Instance from Database
Commit	Commit

Table 5: Transaction steps for two users deleting their properties without conflict.

User 1 Transaction: Delete Property	**User 2 Transaction: Delete Property**
Delete Property P1 Instance from Database	-
Commit	-
-	Delete Property P2 Instance from Database
-	Commit

Table 6: Interleaved execution for two users deleting their properties without conflict.

Transaction Pair 4:Admins Updating Service Prices

This scenario involves two administrators simultaneously updating the prices of services S1 and S2 without conflict.

Admin 1 Transaction: Update Service Price	Admin 2 Transaction: Update Service Price
Update Price of Service S1	Update Price of Service S2
Commit	Commit

Table 7: Transaction steps for admins updating service prices without conflict.

Admin 1 Transaction: Update Service Price	Admin 2 Transaction: Update Service Price
Update Price of Service S1	-
-	Update Price of Service S2
Commit	-
-	Commit

Table 8: Interleaved execution for admins updating service prices without conflict.

Double Booking Property (Conflicting Transactions)

This scenario involves two users simultaneously trying to book the same property for the same period, potentially leading to a conflict if the property is not available for multiple bookings simultaneously.

User 1 Transaction: Book Property	User 2 Transaction: Book Property
Read Property Availability	Read Property Availability
Read Property Details	Read Property Details
Write Update Booking Details for User 1	Write Update Booking Details for User 2
Commit	Commit

Table 9: Conflicting transaction steps for double booking a property.

Detailed Transaction Steps with Locking Mechanisms

Concurrent Property Booking Transactions

This example illustrates how two transactions can concurrently access and update property booking details with explicit locking to avoid conflicts.

Transaction-1 (T1)	**Transaction-2 (T2)**
Lock Read(Property Availability)	-
Read Property Availability	-
Unlock Read(Property Availability)	-
Lock Write(Property)	-
Write Lock Property for Booking	-
Unlock Write(Property)	-
-	Lock Read(Property Availability)
-	Read Property Availability
-	Unlock Read(Property Availability)
Lock Read(Property Details)	-
Read Property Details	-
Unlock Read(Property Details)	-
Lock Write(Booking Details)	-
Write Update Booking Details for User 1	-
Unlock Write(Booking Details)	-
Commit	-
-	Lock Write(Property)
-	Write Lock Property for Booking
-	Unlock Write(Property)
-	Lock Read(Property Details)
-	Read Property Details
-	Unlock Read(Property Details)
-	Lock Write(Booking Details)
-	Write Update Booking Details for User 2
-	Unlock Write(Booking Details)
-	Commit

Table 10: Interleaved execution with locking for two concurrent booking transactions.

Conflicting Transactions: Double Booking Service

Handling Concurrent Booking for the Same Service at the Same Time Slot

This example demonstrates how to handle conflicts when two transactions attempt to book the same service for the same time slot using locks to ensure data consistency and prevent double bookings.

Transaction-1 (T1)	**Transaction-2 (T2)**
Lock Read(Service Availability)	-
Read Service Availability (Available)	-
Unlock Read(Service Availability)	-
-	Lock Read(Service Availability)
-	Read Service Availability (Available)
-	Unlock Read(Service Availability)
Lock Write(Booking)	-
Check Service Availability Again	-
Attempt to Book Service	-
Unlock Write(Booking)	-
Commit	-
-	Wait for T1 to Commit
-	Read Service Availability (Not Available)
-	Decision to Rollback
-	Rollback

Table 11: Synchronized handling of transactions to prevent double booking. T1 successfully books the service, while T2 is blocked until T1 commits and realizes the service is no longer available, leading to a rollback.