University of Windsor

Course Name

Advanced Database Topics (COMP-8157)

Document Type

Lab 4

Document Title

To learn Concurrency Control Concerns in Transactions

Professor

Dr. Shafaq Khan

Graduate Assistant Lead for Lab 4

Mr. Ali Farooghi

Student Name	Student ID
Manjinder Singh	110097177

Presenting Lost Update Problem (Concurrency Issue)

Table 1: Transactions Sequence with Time Event(As per SQL Code Attached)

Time Event 1	Transaction 1 (transaction1 - Lab 4 Problem of Concurrency - Manjinder Singh – 110097177.sql)	Transaction 2 (transaction2 - Lab 4 Problem of Concurrency - Manjinder Singh – 110097177)	Description
T1	Read current balance(b1) for account number 1 (Initial Balance: 1000)		Transaction 1 starts first and reads balance for account number 1
T2	Read Withdrawal Amount Value(100)		Read withdrawal value stored in the variable in Transaction 1.
T3	Wait for 10 seconds		Transaction 1 waits for 10 seconds.
T4		Read current balance(b2) for account number 1 (Initial Balance: 1000)	Transaction 2 starts after 1 second from Transaction 1 start Time and reads balance for account number 1 while Transaction 1 is in wait state.
T5		Read Withdrawal Amount Value(50)	Read withdrawal value stored in the variable in Transaction 2.
T6		Check if balance >= withdrawal amount (1000 >= 50)	Transaction 2 checks if the balance is sufficient to withdraw 50.
T7		Deduct amount 50 from balance (Balance: 1000 - 50 = 950)	Transaction 2 deducts 50 from the balance (resulting in a new balance of 950).
T8		Update BankAccount SET Balance = 950 WHERE AccountNumber = 1	Transaction 2 updates the balance in the database to 950.
T9		Print "Amount is Deducted based on the 2nd transaction instance in the database."	Transaction 2 prints the message "**Amount is Deducted based on the 2nd transaction instance in the database**."
T10	Check if balance >= withdrawal amount (1000 >= 100)		Transaction 1 checks if the balance is sufficient to withdraw 100 without checking the updated value of balance in the database.
T11	Deduct amount 100 from balance (Balance: 1000 - 100 = 900)		Transaction 1 deducts 100 from the balance variable of transaction 1(resulting in a new balance of 900).
T12	Update BankAccount SET Balance = 900 WHERE AccountNumber = 1		Transaction 1 updates the balance in the database to 900 which causes **Data Inconsistency.**

T13	Print "Amount is Deducted based on the 1st transaction instance in the database without considering the update already made by the second transaction."		Transaction 1 prints the message **"Amount is Deducted based on the 1st transaction instance in the database without considering the update already made by the second transaction."**
T14	Select * from BankAccount (Balance: 900)		Transaction 1 retrieves the data and displays the updated balance (900).

Table 2: SQL CODE(Also Attached SQL Files) – **Shows Concurrency Problem**

Transaction 1 (transaction1 - Lab 4 Problem of Concurrency - Manjinder Singh – 110097177.sql)	**Transaction 2** (transaction2 - Lab 4 Problem of Concurrency - Manjinder Singh – 110097177.sql)

`-- Manjinder Singh(110097177)`
`-- Answer 1`
`Create Database BankData;`
`Use BankData;`
`CREATE TABLE BankAccount ( AccountNumber`
`INT PRIMARY KEY, Balance DECIMAL(10, 2) );`
`Insert into BankAccount values(1, 1000);`
 	
`-- Answer 2`
`-- Transaction number - 1 Concurrency`
`Problem`
`BEGIN TRANSACTION;`
 	
`DECLARE @WithdrawAmountForTrans1`
`DECIMAL(10, 2); -- Variable for storing`
`withdrawal amount for transaction 1.`
`SET @WithdrawAmountForTrans1 = 100; --`
`Setting withdrawal amount for transaction`
`1.`
 	
`DECLARE @CurrentBalanceForTrans1`
`DECIMAL(10, 2); -- Variable for storing`
`current balance for account number 1.`
`SELECT @CurrentBalanceForTrans1 = Balance`
`FROM BankAccount`
`WHERE AccountNumber = 1; -- Storing value`
`of current balance from database to the`
`variable "@CurrentBalanceForTrans1".`
 	
`WAITFOR DELAY '00:00:10' -- Adding delay`
`of 10 seconds while transaction 2 already`
`updates the value of balance.`
	`-- Manjinder Singh(110097177)`
	`-- Transaction number - 2`
	
	`-- Answer 3 With Transaction - 2`
	`Concurrency Problem`
	`use BankData;`
	`BEGIN TRANSACTION;`
	
	`DECLARE @WithdrawAmountForTrans2`
	`DECIMAL(10, 2); -- Variable for storing`
	`withdrawal amount for transaction 2.`
	`SET @WithdrawAmountForTrans2 = 50; --`
	`Setting withdrawal amount for transaction`
	`2.`
	
	`DECLARE @CurrentBalanceForTrans2`
	`DECIMAL(10, 2); -- Variable for storing`
	`current balance for account number 2.`
	`SELECT @CurrentBalanceForTrans2 = Balance`
	`FROM BankAccount`
	`WHERE AccountNumber = 1; -- Storing value`
	`of current balance from database to the`
	`variable "@CurrentBalanceForTrans2".`
	
	`-- Checking Balance if it is more than or`
	`equal to the withdrawal amount then only`
	`transacion  will take place.`
	`IF (@CurrentBalanceForTrans2 >=`
	`@WithdrawAmountForTrans2)`
	`BEGIN`
	`   -- Deducting the amount 50 and in the`
	`mean time transaction 1 is still in wait`
	`state. Amoutn updated from 1000 to 950.`

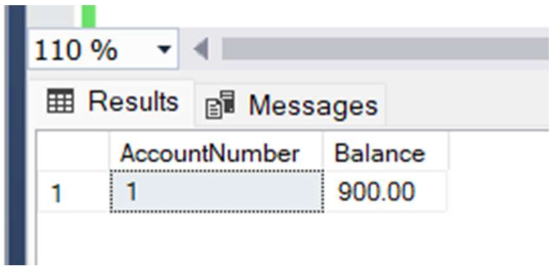
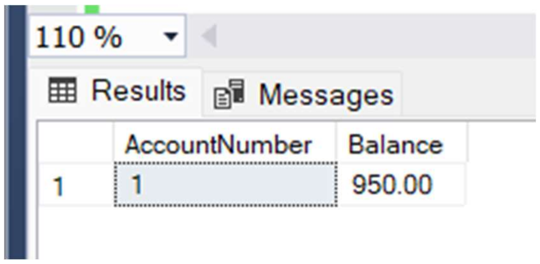
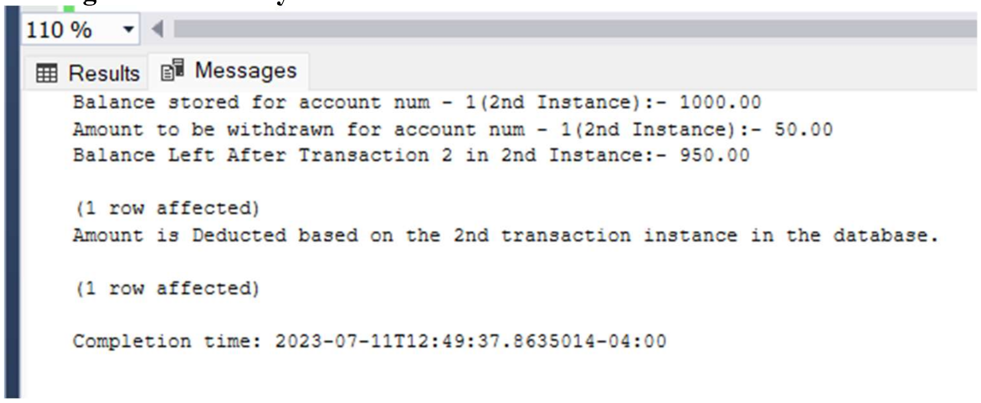
```sql
-- Checking Balance if it is more than or
equal to the withdrawal amount then only
transacion  will take place.
IF (@CurrentBalanceForTrans1 >=
@WithdrawAmountForTrans1)
BEGIN
  -- Deduct the amount based on the 1st
transaction instance without considering
the update made by the second transaction.
  -- Amount 100 is getting deducted from
1000 in transaction 1 even though
transaction 2 updated the value from 1000
to 950 but still it wont consider the
latest updated value.
    PRINT CONCAT('Balance stored for
account num - 1(1st Instance):- ',
@CurrentBalanceForTrans1);
       PRINT CONCAT('Amount to be
withdrawn for account num - 1(1st
Instance):- ',  @WithdrawAmountForTrans1);
       DECLARE @Balance_var DECIMAL(10,
2);
       set @Balance_var =
@CurrentBalanceForTrans1 -
@WithdrawAmountForTrans1
       PRINT CONCAT('Balance Left After
Transaction 1 in 1st Instance:-
',@Balance_var);

       UPDATE BankAccount
       SET Balance = @Balance_var
       WHERE AccountNumber = 1;
    PRINT 'Amount is Deducted based on the
1st transaction instance in the database
without considering the update already
made by the second transaction.'

       select * from BankAccount;
END

ELSE
BEGIN
PRINT 'Insufficient balance, rollback the
transaction number 2';
       ROLLBACK;
END

COMMIT TRANSACTION;
```

```sql
       PRINT CONCAT('Balance stored for
account num - 1(2nd Instance):- ',
@CurrentBalanceForTrans2);
       PRINT CONCAT('Amount to be
withdrawn for account num - 1(2nd
Instance):- ',  @WithdrawAmountForTrans2);
       DECLARE @Balance_var DECIMAL(10,
2);
       set @Balance_var =
@CurrentBalanceForTrans2 -
@WithdrawAmountForTrans2
       PRINT CONCAT('Balance Left After
Transaction 2 in 2nd Instance:-
',@Balance_var);

       UPDATE BankAccount
    SET Balance = @Balance_var
    WHERE AccountNumber = 1;
    PRINT 'Amount is Deducted based on the
2nd transaction instance in the
database.';

       select * from BankAccount;
END
ELSE
BEGIN
       PRINT 'Insufficient balance,
Rollback the transaction number 2';
       ROLLBACK;
END

COMMIT TRANSACTION;
```

(SQL FILES OF BOTH TRANSACTIONS ARE ATTACHED ON BRIGHTSPACE)

(CONTD. TO NEXT PAGE)

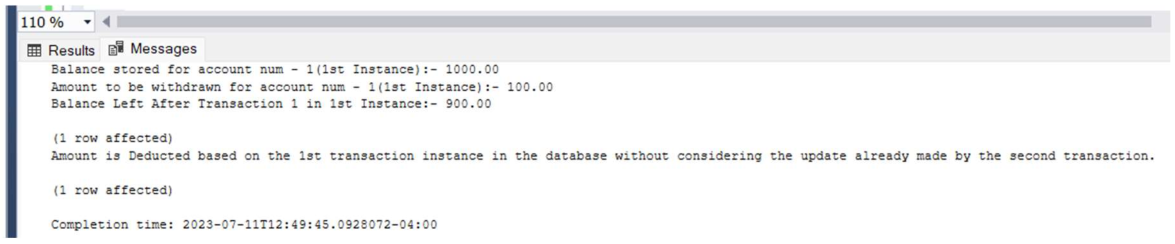
Table 3: Transactions information along with Details, Results, Messages.

<u>Transaction – 1</u>	**<u>Transaction - 2</u>**
Details of Transaction **(Runs Concurrently** (T2 runs after 1 second from T1 but parallel execution)**)**	
Account Number Considered - 1 Current Balance Read – 1000 Withdraw Amount – 100 Balance Left – 900(updates in database without checking new updated value by Transaction 2)	Account Number Considered - 1 Current Balance Read – 1000 Withdraw Amount – 50 Balance Left – 950(As per Transaction 2, account value is updated in database.)
Results on Console	
 110 % Results Messages AccountNumber Balance 1 1 900.00 **Figure 1:** output after Transaction 1 execution	 110 % Results Messages AccountNumber Balance 1 1 950.00 **Figure 2:** output after Transaction 2 execution
Messages Published	
Messages Published by Transaction 2: 	

110 %
Results Messages

```
Balance stored for account num - 1(2nd Instance):- 1000.00
Amount to be withdrawn for account num - 1(2nd Instance):- 50.00
Balance Left After Transaction 2 in 2nd Instance:- 950.00

(1 row affected)
Amount is Deducted based on the 2nd transaction instance in the database.

(1 row affected)

Completion time: 2023-07-11T12:49:37.8635014-04:00
```

Messages Published by Transaction 1:



```
Balance stored for account num - 1(1st Instance):- 1000.00
Amount to be withdrawn for account num - 1(1st Instance):- 100.00
Balance Left After Transaction 1 in 1st Instance:- 900.00

(1 row affected)
Amount is Deducted based on the 1st transaction instance in the database without considering the update already made by the second transaction.

(1 row affected)

Completion time: 2023-07-11T12:49:45.0928072-04:00
```

Presenting Solution of Lost Update Problem (Resolved Concurrency Issue of Transaction 1 and 2)

Transaction 1 and 2 are replicated to transaction 3 and 4 respectively but with addition of the "TRANSACTION ISOLATION LEVEL REPEATABLE READ".

Table 4: Transactions Sequence with Time Event(As per SQL Code Attached)

Time Event 1	**Transaction 3** (transaction3 - Lab 4 Solution to Concurrency Problem - Manjinder Singh - 110097177)	**Transaction 4** (transaction4 - Lab 4 Solution to Concurrency Problem - Manjinder Singh – 110097177)	**Description**
T1	Read current balance(b1) for account number 1 (Initial Balance: 1000)		Transaction 3 starts first and reads balance for account number 1
T2	Read Withdrawal Amount Value(100)		Read withdrawal value stored in the variable in Transaction 3.
T3	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ		Under the Repeatable Read isolation level, a transaction ensures that all queries within the transaction see a consistent snapshot of the data as of the start of the transaction. This means that any changes made by other concurrent transactions after the start of the current transaction will not be visible to it. Transaction 3 will be prioritized over other transactions.
T4	Wait for 10 seconds		Transaction 3 waits for 10 seconds.

T5		SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	Under the Repeatable Read isolation level, a transaction ensures that all queries within the transaction see a consistent snapshot of the data as of the start of the transaction. Also Transaction 3 already set this level before transaction 4 so transaction 3 update operation will be prioritized but transaction 4 will be able to read and will fail to perform update operation.
T6		Read current balance(b2) for account number 1 (Initial Balance: 1000)	Transaction 4 starts after 1 second from Transaction 3 start Time and reads balance for account number 1 while Transaction 3 is in wait state.
T7		Read Withdrawal Amount Value(50)	Read withdrawal value stored in the variable in Transaction 4.
T8		Check if balance >= withdrawal amount (1000 >= 50)	Transaction 4 checks if the balance is sufficient to withdraw 50.
T9		Deduct amount 50 from balance.	Transaction 4 will be able to deduct 50 from the balance variable as it is an arithmetic operation on a variable.
T10		Update BankAccount SET Balance = 950 WHERE AccountNumber = 1 (This operation fails to update Balance: 1000 - 50 = 950)	Transaction 4 wont be able to update the balance in the database to 950 due to isolation level of repeatable read.
T11		Print "Balance updated." (This operation wont be able to print due to isolation level.)"	Transaction 4 wont be able to print the message "Balance Updated."as isolation level is set in transaction 3 which is supposed to be prioritized over transaction 4.
T12	Check if balance >= withdrawal amount (1000 >= 100)		Transaction 3 checks if the balance is sufficient to withdraw 100.
T13	Deduct amount 100 from balance (Balance: 1000 - 100 = 900)		Transaction 3 deducts 100 from the balance variable of transaction 1(resulting in a new balance of 900).
T14	Update BankAccount SET Balance = 900 WHERE AccountNumber = 1		Transaction 3 updates the balance in the database to 900 and **Data Consistency** is maintained.
T15	Print "Amount is Deducted based on the 1st transaction instance in the database without considering the update already made by the second transaction."		Transaction 3 prints the message "**Amount is Deducted based on the 1st transaction instance in the database without considering the update already made by the second transaction.**"
T16	Select * from BankAccount (Balance: 900)		Transaction 3 retrieves the data and displays the updated balance (900).

Table 5: SQL CODE(Also Attached SQL Files) – **Solution to Concurrency Problem**

Transaction 3
(transaction3 - Lab 4 Solution to Concurrency Problem - Manjinder Singh – 110097177.sql)

```sql
-- Manjinder Singh(110097177)
Create Database BankData;
Use BankData;
CREATE TABLE BankAccount ( AccountNumber
INT PRIMARY KEY, Balance DECIMAL(10, 2) );
Insert into BankAccount values(1, 1000);

-- Transaction number - 3 Solution to
Concurrency Problem
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE
READ -- Isolation Level set to Repeatable
Read.

DECLARE @WithdrawAmountForTrans3
DECIMAL(10, 2); -- Variable for storing
withdrawal amount for transaction 3.
SET @WithdrawAmountForTrans3 = 100; --
Setting withdrawal amount for transaction
3.

DECLARE @CurrentBalanceForTrans3
DECIMAL(10, 2); -- Variable for storing
current balance for account number 1.
SELECT @CurrentBalanceForTrans3 = Balance
FROM BankAccount
WHERE AccountNumber = 1; -- Storing value
of current balance from database to the
variable "@CurrentBalanceForTrans1".

WAITFOR DELAY '00:00:10' -- Adding delay
of 10 seconds while transaction 4 already
trying to update the value of balance due
to isolation level.

-- Checking Balance if it is more than or
equal to the withdrawal amount then only
transacion  will take place.
IF (@CurrentBalanceForTrans3 >=
@WithdrawAmountForTrans3)
BEGIN
  -- Deduct the amount based on this
transaction instance without considering
any other transaction for the same accoutn
number beacuse this transacton has set
lock on priorty.
  -- Amount 100 is getting deducted from
1000 in transaction 3 and also transaction
4 won't be able to update due to
REPEATABLE READ Mode of Isolataion Level.
```

Transaction 4
(transaction4 - Lab 4 Solution to Concurrency Problem - Manjinder Singh – 110097177.sql)

```sql

-- Manjinder Singh(110097177)
-- Transaction number - 4 Solution to
Concurrency Problem
-- update BankAccount set Balance=1000;
-- This transaction wont be executed for
update operation due to ISOLATION LEVEL
REPEATABLE READ Mode set on Transaction 3
use BankData;
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE
READ -- Isolation Level set to Repeatable
Read.

DECLARE @WithdrawAmountForTrans4
DECIMAL(10, 2); -- Variable for storing
withdrawal amount for transaction 4.
SET @WithdrawAmountForTrans4 = 50; --
Setting withdrawal amount for transaction
4.

DECLARE @CurrentBalanceForTrans4
DECIMAL(10, 2); -- Variable for storing
current balance for account number 1.
SELECT @CurrentBalanceForTrans4 = Balance
FROM BankAccount
WHERE AccountNumber = 1; -- Storing value
of current balance from database to the
variable "@CurrentBalanceForTrans4".


-- Checking Balance if it is more than or
equal to the withdrawal amount then only
transacion  will take place.
IF (@CurrentBalanceForTrans4 >=
@WithdrawAmountForTrans4)
BEGIN
   -- Trying to Deduct the amount 50 and
in the mean time transaction 3 is still in
wait state. Amount wont be updated from
1000 to 950 due to high isolation level
set for Transaction number 3.
       PRINT CONCAT('Balance stored for
account num - 1(4th Instance):- ',
@CurrentBalanceForTrans4);
       PRINT CONCAT('Amount to be
withdrawn for account num - 1(4th
Instance):- ',  @WithdrawAmountForTrans4);
       DECLARE @Balance_var DECIMAL(10,
2);
```

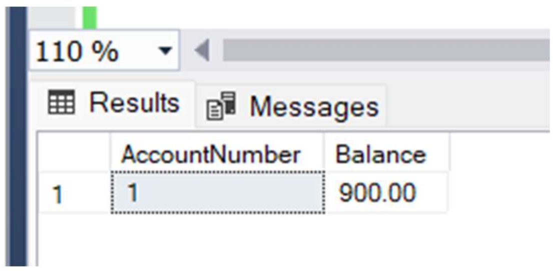
```
    PRINT CONCAT('Balance stored for
account num - 1(3rd Instance):- ',
@CurrentBalanceForTrans3);
       PRINT CONCAT('Amount to be
withdrawn for account num - 1(3rd
Instance):- ',  @WithdrawAmountForTrans3);
       DECLARE @Balance_var DECIMAL(10,
2);
       set @Balance_var =
@CurrentBalanceForTrans3 -
@WithdrawAmountForTrans3
       PRINT CONCAT('Balance Left After
Transaction 3 in 3rd Instance:-
',@Balance_var);

       UPDATE BankAccount
       SET Balance = @Balance_var
       WHERE AccountNumber = 1;
    PRINT 'Amount is Deducted based on the
3rd transaction instance in the database
while Transaction 4 is locked to perform
update operation.'

       select * from BankAccount;
END

ELSE
BEGIN
PRINT 'Insufficient balance, rollback the
transaction number 3.';
       ROLLBACK;
END

COMMIT TRANSACTION;
```

```
       set @Balance_var =
@CurrentBalanceForTrans4 -
@WithdrawAmountForTrans4
       PRINT CONCAT('Balance Left After
Transaction 4 in 4th Instance:-
',@Balance_var);

       UPDATE BankAccount
    SET Balance = @Balance_var
    WHERE AccountNumber = 1;
    PRINT 'Balance updated.'; -- Amount is
supposed to be deducted based on the 4th
transaction instance in the database but
it wont be due to isolation level.

       select * from BankAccount;
END
ELSE
BEGIN
       PRINT 'Insufficient balance,
Rollback the transaction number 4';
       ROLLBACK;
END

COMMIT TRANSACTION;
```

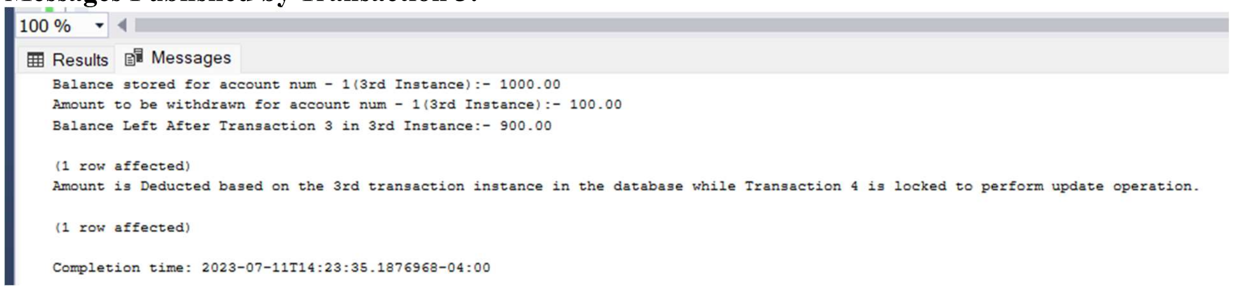
(SQL FILES OF BOTH TRANSACTIONS ARE ATTACHED ON BRIGHTSPACE)

(CONTD. TO NEXT PAGE)

Table 6: Transactions information along with Details, Results, Messages.

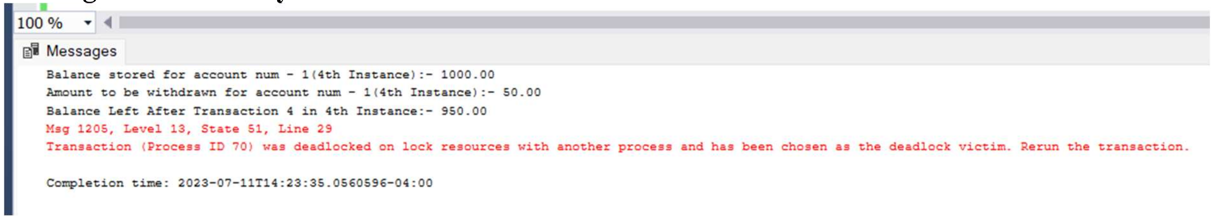
Transaction 3 (transaction3 - Lab 4 Solution to Concurrency Problem - Manjinder Singh – 110097177.sql)	**Transaction 4** (transaction4 - Lab 4 Solution to Concurrency Problem - Manjinder Singh – 110097177.sql)
Details of Transaction **(Runs Concurrently** (T4 runs after 1 second from T3 but parallel execution)**)**	
Account Number Considered - 1 Current Balance Read – 1000 Withdraw Amount – 100 Balance Left – 900(updates in database without checking new updated value by Transaction 2)	Account Number Considered - 1 Current Balance Read – 1000 Withdraw Amount – 50 Balance Left – NA(As per Transaction 4 wont be able to update due to highest level of isolation.)
Results on Console	
 110 % · Results · Messages AccountNumber · Balance 1 · 1 · 900.00 **Figure 3**: output after Transaction 3 execution	NA(because update operation will fail due to Isolation level set on transaction 3 which will be prioritized over transaction 4)
Messages Published	

Messages Published by Transaction 3:



```
Balance stored for account num - 1(3rd Instance):- 1000.00
Amount to be withdrawn for account num - 1(3rd Instance):- 100.00
Balance Left After Transaction 3 in 3rd Instance:- 900.00

(1 row affected)
Amount is Deducted based on the 3rd transaction instance in the database while Transaction 4 is locked to perform update operation.

(1 row affected)

Completion time: 2023-07-11T14:23:35.1876968-04:00
```

Messages Published by Transaction 4:



```
Balance stored for account num - 1(4th Instance):- 1000.00
Amount to be withdrawn for account num - 1(4th Instance):- 50.00
Balance Left After Transaction 4 in 4th Instance:- 950.00
Msg 1205, Level 13, State 51, Line 29
Transaction (Process ID 70) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Completion time: 2023-07-11T14:23:35.0560596-04:00
```

Answer 3 In Detail:

CONCURRENCY PROBLEM EXPLANATION WRT SQL CODE:

The **lost update problem** arises in the above scenario (because both transactions T1 and T2 operate concurrently without proper synchronization).

The problem of lost update is resolved in transactions T3 and T4.

Before explaining this, let's discuss about "`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`":

When we set the isolation level to Repeatable Read, it guarantees that all the queries performed within your transaction will see a consistent snapshot of the data as it existed at the beginning of the transaction. This means that any changes made by other transactions after your transaction started will not be visible to it.

Here are the key points to understand about the Repeatable Read isolation level:

- ➔ Data read within your transaction remains unchanged, even if it is modified by other concurrent transactions. This ensures that the data you access remains consistent throughout your transaction.
- ➔ Concurrent transactions are not allowed to modify or delete data that has been read by your transaction until it is committed or rolled back. This prevents any conflicting modifications that could lead to data inconsistencies.
- ➔ If you execute the same query multiple times within your transaction, you will get the same results each time. This consistency is maintained regardless of any changes made by other transactions.

By setting the Repeatable Read isolation level, you can rely on a stable and predictable snapshot of the data throughout your transaction. This helps maintain data integrity and ensures that your operations are based on a consistent view of the data.

It's important to note that using Repeatable Read may have implications on concurrency and performance. It can result in longer-held locks, potentially limiting the concurrency of other transactions and leading to increased resource contention. Therefore, it's crucial to choose the appropriate isolation level based on the specific requirements of your application to balance data consistency and concurrency.

In our considered example, the **lost update problem** can be observed when Transaction 1 deducts the amount based on the initial balance (1000) instead of considering the update made by Transaction 2 (which changed the balance to 950 from 1000 after update). As a result, the final balance was not consistent due to the **lost update. (More details are added in the above tables 1,2 and 3)**

The issue occurs when Transaction 1 updates the balance, unaware of any changes made by Transaction 2. Consequently, the update made by the Transaction 2 is lost, and the final balance did not reflect the expected result (which was required to be 850 instead of 900 after operations of transaction 1 and 2).

To mitigate the lost update problem, **proper synchronization mechanisms or isolation levels** should be implemented. These mechanisms ensure that concurrent transactions consider each other's updates and prevent lost updates from occurring, ensuring the integrity of the data.

On the Other hand, when we execute the transaction 3 and then after a second, when we execute transaction 4, it was observed that after wait time of Transaction 3, the update made by the transaction 3 reflects in the database which changed value from 1000 to 900. Also, Transaction 4 which was executing parallelly was able to read the balance value but unable to update the value in Transaction 4 for the balance column. The main reason behind that is the isolation level of type repeatable read which was active on transaction 3 which is set to be prioritized over other transactions as execution of this first executed transaction (number 3) was taken place before transaction 4. So transaction 3 was able to perform updates whereas transaction 4 displays the error, "`Transaction (Process ID 70) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.`"

So in this way, data concurrency problem can be handled for the specific type - LOST UPDATE.