Logo, company name

Description automatically generated

**School of Computer Science**

**Masters in Applied Computing (M.A.C)**

**Subject Code: COMP8157**

**Subject Name: Advanced Database Topics**

**Professor Dr Shafaq Khan**

**Lab 4**

**by**

**Hamza Baig (110089314)**

Table of Contents

[Database: 3](#_Toc129892013)

[bank\_account database: 3](#_Toc129892014)

[Insert initial account information: 3](#_Toc129892015)

[Methods for database: 4](#_Toc129892016)

[Deposit: 4](#_Toc129892017)

[Code: 4](#_Toc129892018)

[Withdrawal: 6](#_Toc129892019)

[Code: 6](#_Toc129892020)

[In case of error (withdrawing money you don’t have): 7](#_Toc129892021)

[View Statement: 7](#_Toc129892022)

[Code: 7](#_Toc129892023)

# Database:

## bank\_account database:

-- Create the bank\_accounts table

CREATE TABLE bank\_accounts (

    account\_number INT PRIMARY KEY,

    account\_holder\_name VARCHAR(50),

    balance DECIMAL(10, 2)

);

Graphical user interface, text, application, email

Description automatically generated

## Insert initial account information:

-- Set the initial account balance to $200

INSERT INTO bank\_accounts (account\_number, account\_holder\_name, balance)

VALUES (110089314, 'Hamza Baig', 200);

Graphical user interface, text, application, email

Description automatically generated

-- Show bank accounts

SELECT \* FROM bank\_accounts;

Table

Description automatically generated

# Methods for database:

I have used isolation level repeatable read, as it would block the second transaction and let the first one finish first, in this way there will be no conflict, or lost updates, as one transaction completes first.

## Deposit:

### Code:

-- Deposit money

DECLARE @deposit AS INT=200;

DECLARE @balance AS DECIMAL(10, 2);

SET @balance=isnull((SELECT balance FROM bank\_accounts WHERE account\_number = 110089314),0);

-- Transaction 2

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

UPDATE bank\_accounts SET balance = @balance+@deposit WHERE account\_number = 110089314;

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

From the above images, it is evident that the transaction 1 gives the old balance and meanwhile, transaction 2 is blocked, so it executes and gives the updated values.

## Withdrawal:

### Code:

-- Withdraw money

DECLARE @withdraw AS INT=800;

DECLARE @current\_balance AS DECIMAL(10, 2);

SET @current\_balance=isnull((SELECT balance FROM bank\_accounts WHERE account\_number = 110089314),0);

-- Transaction 2

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

IF (@current\_balance-@withdraw>0)

    UPDATE bank\_accounts SET balance = @current\_balance-@withdraw WHERE account\_number = 110089314;

ELSE

    PRINT 'You do not have enough money in your account!';

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

The above example also demonstrates how isolation level repeatable read is blocking the second transaction and completing the first transaction first.

### In case of error (withdrawing money you don’t have):

Graphical user interface, text, application, email

Description automatically generated

## View Statement:

### Code:

-- Display balance

SELECT balance FROM bank\_accounts WHERE account\_number = 110089314;

-- Transaction 1

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION

SELECT balance FROM bank\_accounts WHERE account\_number = 110089314

-- Do Some work

WAITFOR DELAY '00:00:10'

SELECT balance FROM bank\_accounts WHERE account\_number = 110089314

COMMIT TRANSACTION

Graphical user interface, text, application, email

Description automatically generated