

# CSCI 5408

## DATA MANAGEMENT AND WAREHOUSING

### LAB ASSIGNMENT - 3

Banner ID: B00948977

Git Assignment Link :

[https://git.cs.dal.ca/sukumaran/csci5408\\_f23\\_b00948977\\_balaji\\_sukumaran](https://git.cs.dal.ca/sukumaran/csci5408_f23_b00948977_balaji_sukumaran)

# Table of contents

---

<b>Problem Statement 1:</b> Design a banking application database.....	<b>1</b>
<b>Problem Statement 2:</b> Create a transaction environment .....	<b>4</b>
<b>Problem Statement 3:</b> Based on this status, handle the update balance query.....	<b>5</b>



**Problem Statement 1:** Design a banking application database with the customer's details (minimal attributes - name, mailing address, permanent address, primary email, primary phone number), account details (minimal attributes – account number, account balance) and account transfer details (minimal attributes - account number, date of transfer, recipient name, status(varchar value waiting/accepted/declined)).

Created the banking database tables. To simulate transaction environment

Line #1: create table command for customer\_details

Line #2: create table command for account\_details

Line #3: create table command for transfer\_details

Tables created successfully.

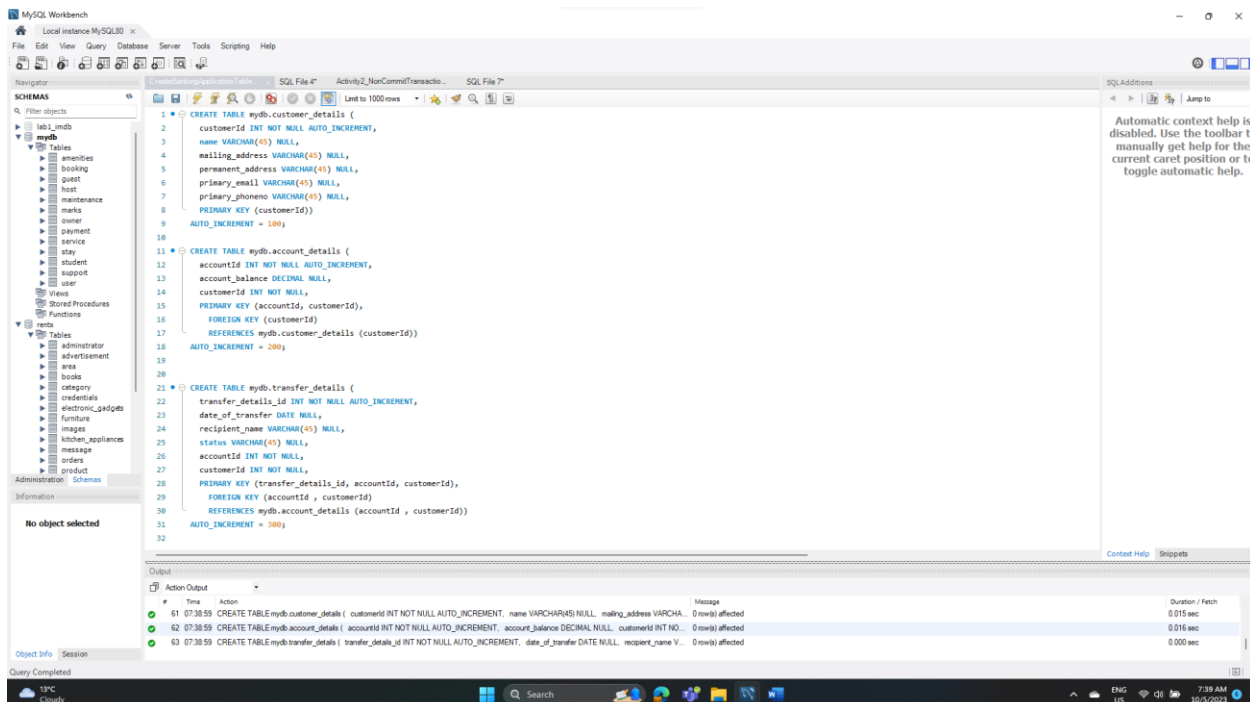


Figure 1: create statement for customer\_details, account\_details, transfer\_details

Inserted sample data for simulating transactions

Line #3 till #11: Insert statements for customer\_details

Line #13 till #20: Insert statements for account\_details

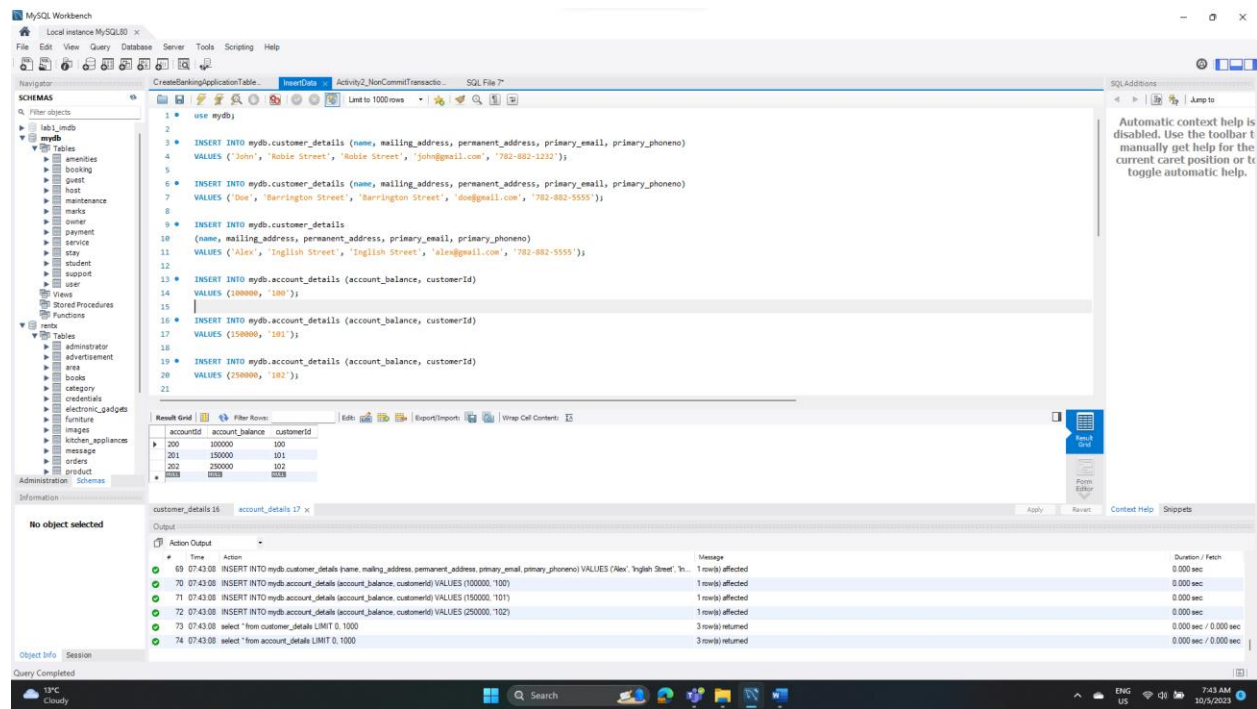


Figure 2: Insert Statement for customer\_details and account\_details

Records in customer\_details table

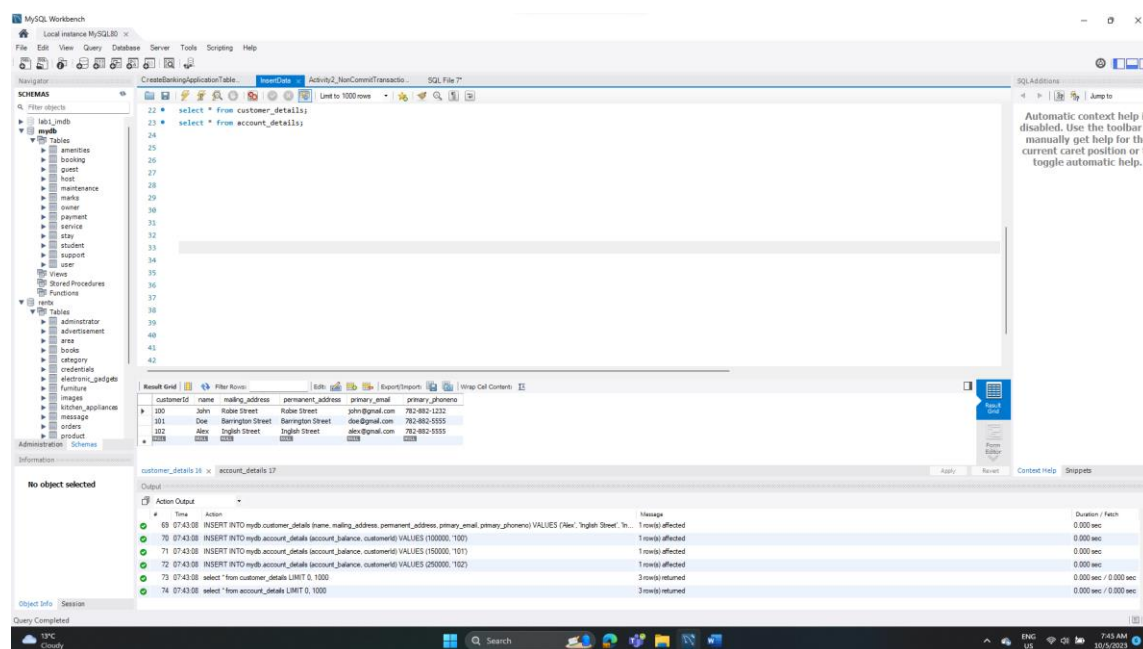


Figure 3: Contents of customer\_details table

## Records in account\_details table

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'mydb' database selected. The main editor window shows a SQL query: `select * from customer_details; select * from account_details;`. Below the query, the 'Result Grid' shows the contents of the 'account\_details' table. The table has three columns: 'accountid', 'account\_balance', and 'customerid'. The data rows are:

accountid	account_balance	customerid
200	100000	100
201	150000	101
202	250000	102

At the bottom, the 'Output' pane shows the execution log, including the following actions:

- 69 07:43:08 INSERT INTO mydb.customer\_details (name, mailing\_address, permanent\_address, primary\_email, primary\_phoneno) VALUES ('Alex', 'High Street', 'N...
- 70 07:43:08 INSERT INTO mydb.account\_details (account\_balance, customerid) VALUES (100000, '100')
- 71 07:43:08 INSERT INTO mydb.account\_details (account\_balance, customerid) VALUES (150000, '101')
- 72 07:43:08 INSERT INTO mydb.account\_details (account\_balance, customerid) VALUES (250000, '102')
- 73 07:43:08 select \* from customer\_details LIMIT 0, 1000
- 74 07:43:08 select \* from account\_details LIMIT 0, 1000

Figure 4: Contents of account\_details table

**Problem Statement 2:** Create a transaction environment where on every account debit, the record in the account details table is updated but not committed. Upon this update, insert a record in the account transfer details table with waiting state.

Line #4: Setting Autocommit to OFF, to keep the transaction in a limbo state when this query page is executed

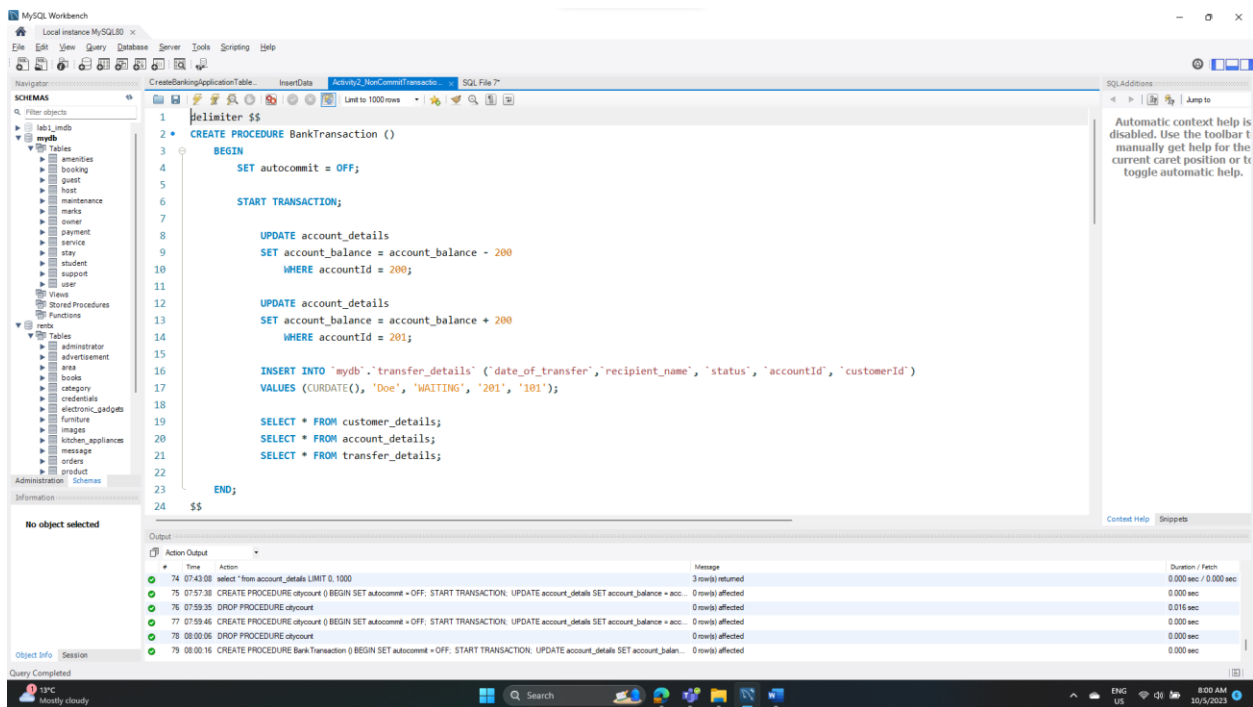
Line #6: Start of transaction

Line #8: Debited \$200 from John's account

Line #12: Transferred that \$200 to Doe's account

Line #16: INSERTED into transfer\_details table with a WAITING state indicating this state is not committed

Line #19,20,21: to see the current state of the customer\_details, account\_details and transfer\_details table.



**Problem Statement 3:** Assume that the transaction status gets verified by some X business logic (pure assumption here, no need to implement any logic), the transaction status is changed to either accepted or declined. Based on this status, handle the update balance query.

(Hint: Based on the transaction status, either commit or rollback the account details table SQL statements. Try exploring SAVEPOINT).

Scenario #1: If the backend logic changed it to ACCEPTED then the transaction should COMMIT and if it is REJECTED it should rollback everything that is done in this transaction

Line #23: checks for the state of the transfer\_details record. If it is ACCEPTED then commit, If REJECTED then ROLLBACK

The screenshot shows the MySQL Workbench interface. The main editor displays a SQL script for a stored procedure named 'BankTransaction'. The script is as follows:

```
1 use mydb;
2 CREATE PROCEDURE BankTransaction ()
3 BEGIN
4     SET autocommit = OFF;
5
6     START TRANSACTION;
7
8     UPDATE account_details
9     SET account_balance = account_balance - 200
10    WHERE accountId = 200;
11
12    UPDATE account_details
13    SET account_balance = account_balance + 200
14    WHERE accountId = 201;
15
16    INSERT INTO mydb.transfer_details (date_of_transfer, recipient_name, status, accountId, customerId)
17    VALUES (CURDATE(), 'Doe', 'REJECTED', '201', '101');
18
19    SELECT * FROM customer_details;
20    SELECT * FROM account_details;
21    SELECT * FROM transfer_details;
22
23    IF EXISTS (SELECT 1 FROM transfer_details WHERE status = 'ACCEPTED' AND accountId = '201')
24    THEN
25        COMMIT;
26    ELSE
27        ROLLBACK;
28    END IF;
29 END;
```

The left sidebar shows the 'SCHEMAS' tab with a tree view of the 'mydb' database, including tables like 'account\_details', 'customer\_details', and 'transfer\_details'. The bottom status bar shows the execution of the script, with a message indicating that no database was selected and the default DB should be used.



Scenario #2: If the backend logic changed it to ACCEPTED then the transaction should COMMIT and if it is REJECTED it should rollback up till a SAVEPOINT.

Here in case of in case of ACCEPTED it will commit if not It will rollback to JOHN\_ACC\_UPDATE savepoint. Which means everything from Line #12 rolled back.

