# *CSCI 5408*

# *Data Management, Warehousing, And Analytics*

## *Lab 3: Transaction and Normalization*

**Prepared By**

Bhavisha Oza (B00935827)

## Summary

Transactions are a fundamental concept in database management systems that ensure data integrity and consistency. They group a set of database operations into a single logical unit, allowing them to succeed or fail together. Transactions have various states, including the active state where operations are being executed, the partially committed state where changes are applied but not yet finalized, and the committed state where changes are permanently saved [2]. In the hands-on session, I have tried doing transactions with commit, rollback and using savepoint.

## Steps followed:

- Designed a banking application database with the given list of tables.
- Created a database tables “customer”, “account_details”, “account_transfer_details”.
- Inserted few records in all the tables.
- Created a transaction environment for the asked details.
- Based on the transaction status, updated the balance query using stored procedure.

## Lab exercise:

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))
    a. **Database created for banking application**.
    **SQL Queries executed:**

    -- Create banking_application database
    CREATE SCHEMA banking_application;

    -- Switch to the banking_application database
    USE banking_application;

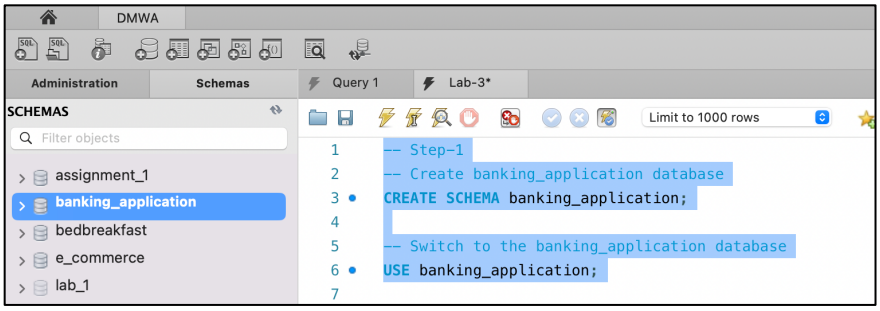


**Figure:1** Database and schema creation

    b. **Created Customer table**
    **SQL Queries executed:**

    -- Create customer details table

```
CREATE TABLE customer (
cust_id INT PRIMARY KEY,
cust_name VARCHAR (45) NOT NULL,
cust_age VARCHAR (45),
cust_DOB DATETIME,
mailing_address VARCHAR (45) NOT NULL,
permanent_address VARCHAR (255) NOT NULL,
primary_email VARCHAR (45) NOT NULL,
primary_phone_number VARCHAR (255) NOT NULL
);
```

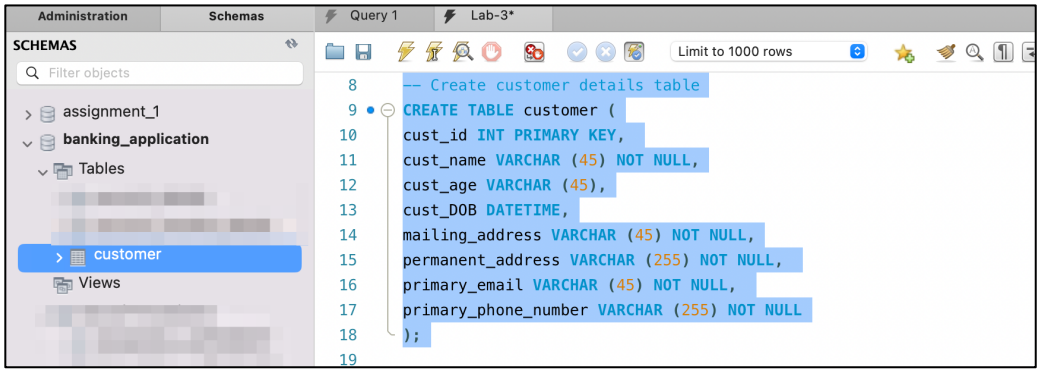


**Figure:2** Customer table creation

c. **Created account_details table**
**SQL Queries executed:**

```
-- Create account details table
CREATE TABLE account_details (
account_number INT PRIMARY KEY,
account_balance VARCHAR (255) NOT NULL,
cust_id INT,
FOREIGN KEY(cust_id) references customer(cust_id)
);
```

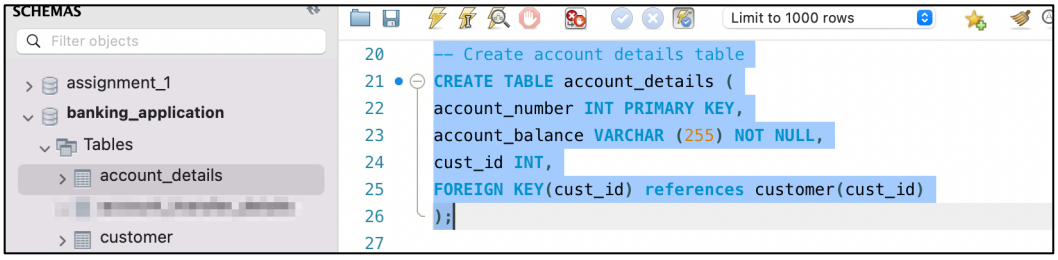


**Figure:3** account details table creation

d. **Created account_transfer_details table**
**SQL Queries executed:**

```
-- Create account transfer details table
CREATE TABLE account_transfer_details (
date_of_transfer DATETIME PRIMARY KEY,
```

recipient_name VARCHAR (255) NOT NULL,
status VARCHAR (255) NOT NULL,
account_balance VARCHAR (255) NOT NULL,
account_number INT,
FOREIGN KEY(account_number) references account_details(account_number),
cust_id INT,
FOREIGN KEY(cust_id) references customer(cust_id)
);

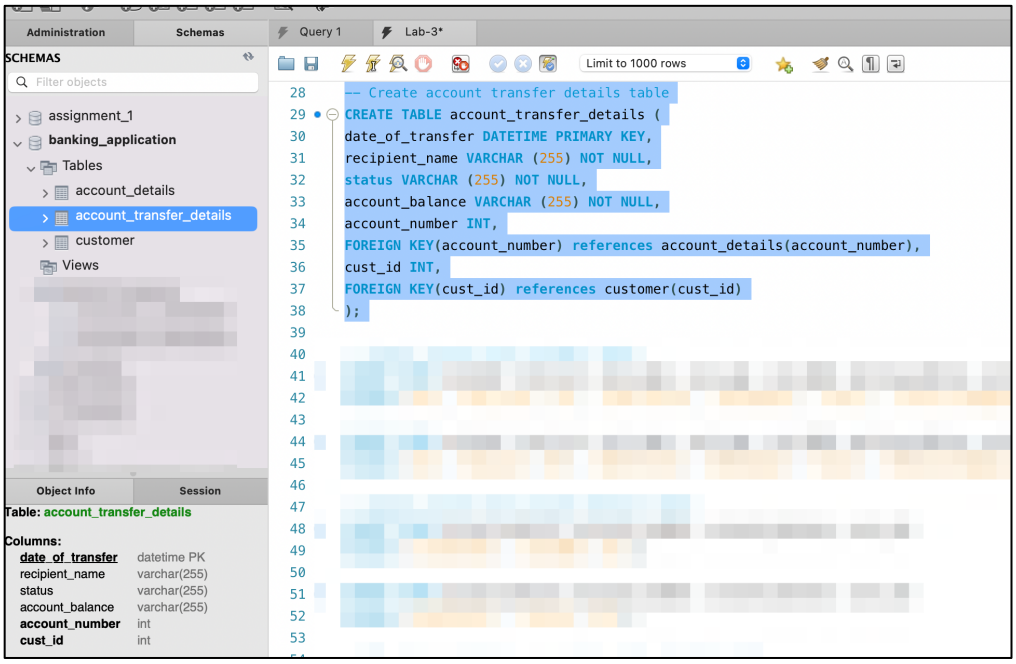


**Figure:4** account transfer details table creation

e. **Inserted few records manually into the tables:**
**SQL Queries executed:**

-- Insert record into customer table
INSERT INTO customer (cust_id, cust_name, cust_age, cust_DOB,
mailing_address, permanent_address, primary_email, primary_phone_number)
VALUES ('01', 'Bhavisha','25', '1997-04-01', 'ahmedabad', 'India',
'erbhavisha@gmail.com', '9876097710');

INSERT INTO customer (cust_id, cust_name, cust_age, cust_DOB,
mailing_address, permanent_address, primary_email, primary_phone_number)
VALUES ('02', 'Bhaumik','26', '1996-10-18', 'ahmedabad', 'India',
'bhaumik123@gmail.com', '9878620710');

-- Insert record into account transfer details table
INSERT INTO account_details (account_number, account_balance, cust_id)
VALUES ('123456789', '3231000','01');

-- Insert record into account transfer details table

INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
VALUES ('2023-06-12', 'Bhavisha','approved', '3231997', '123456789', '01');

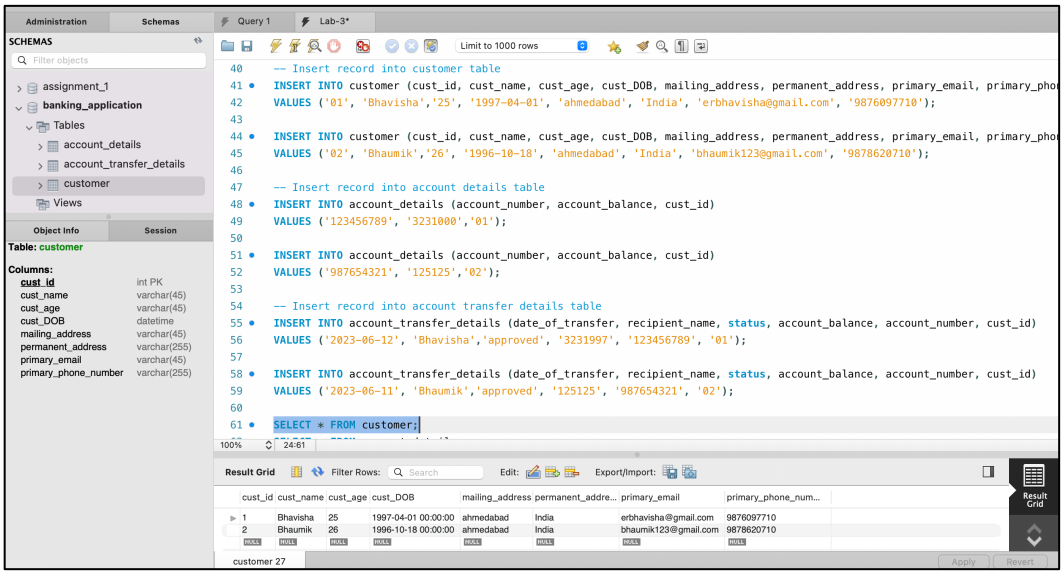


**Figure:5** Record insertion in customer table

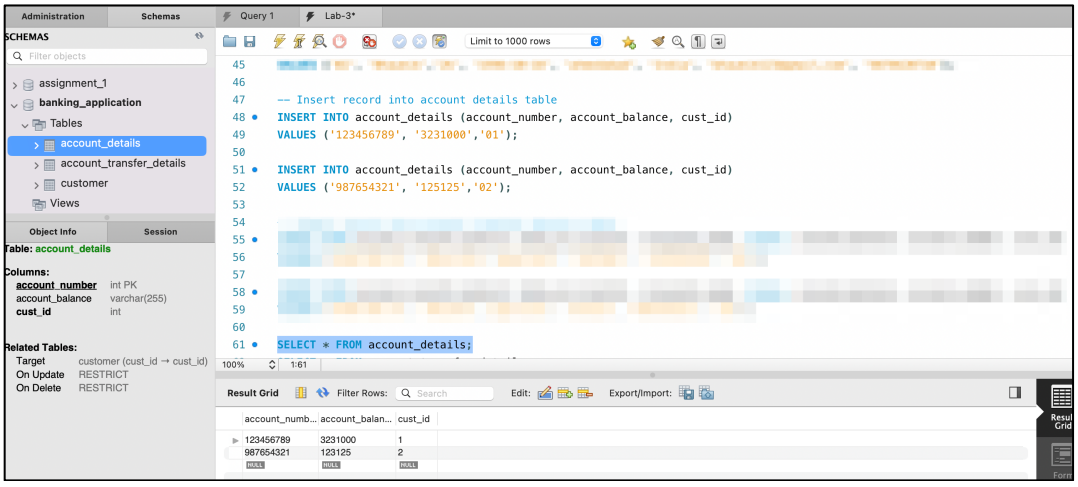


**Figure:6** Record insertion in account_details table

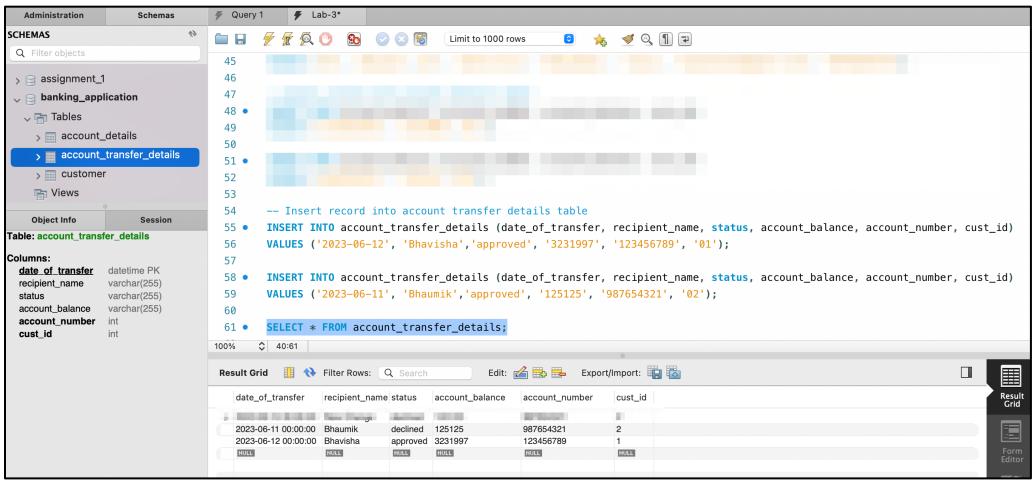


**Figure:7** Record insertion in account_transfer_details table

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.

   **SQL Queries executed:**

   -- Step-2
   SET autocommit = 0;
   -- Create a transaction environment
   START TRANSACTION;

   UPDATE account_details
   SET account_balance = account_balance - 1000
   WHERE account_number = 987654321;

   SELECT * FROM account_details;

   INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status, account_balance, account_number, cust_id)
   VALUES ('2023-06-10', 'New Change', 'waiting', '124125', '987654321', '02');

SELECT * FROM account_transfer_details;

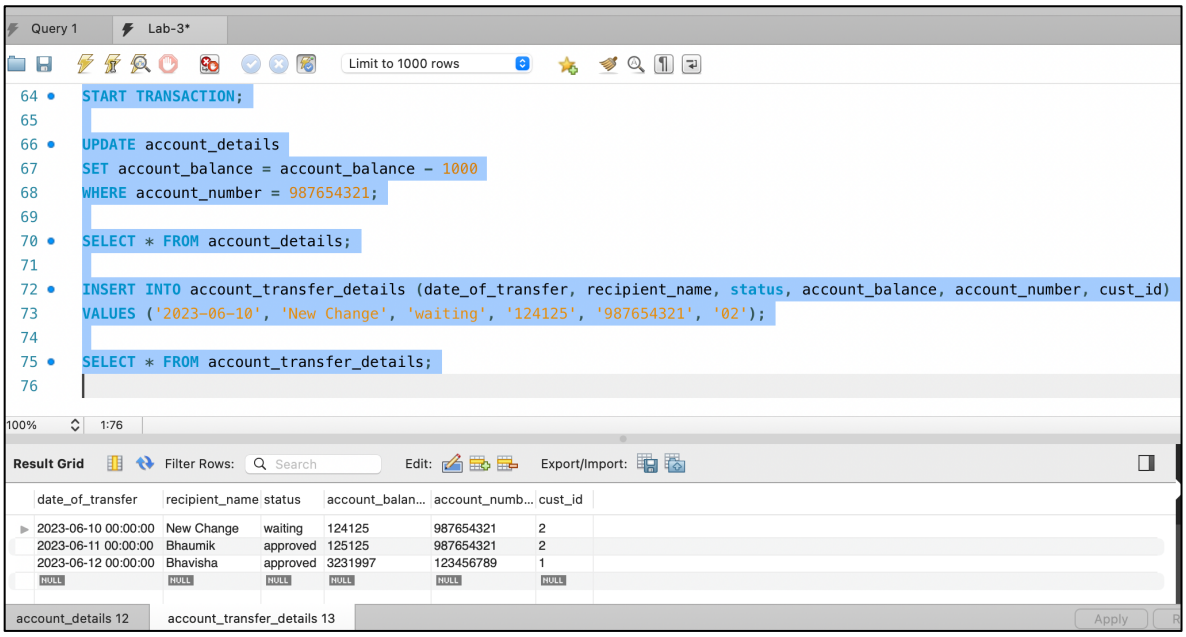


**Figure:8** account_transfer_details table after transaction completion

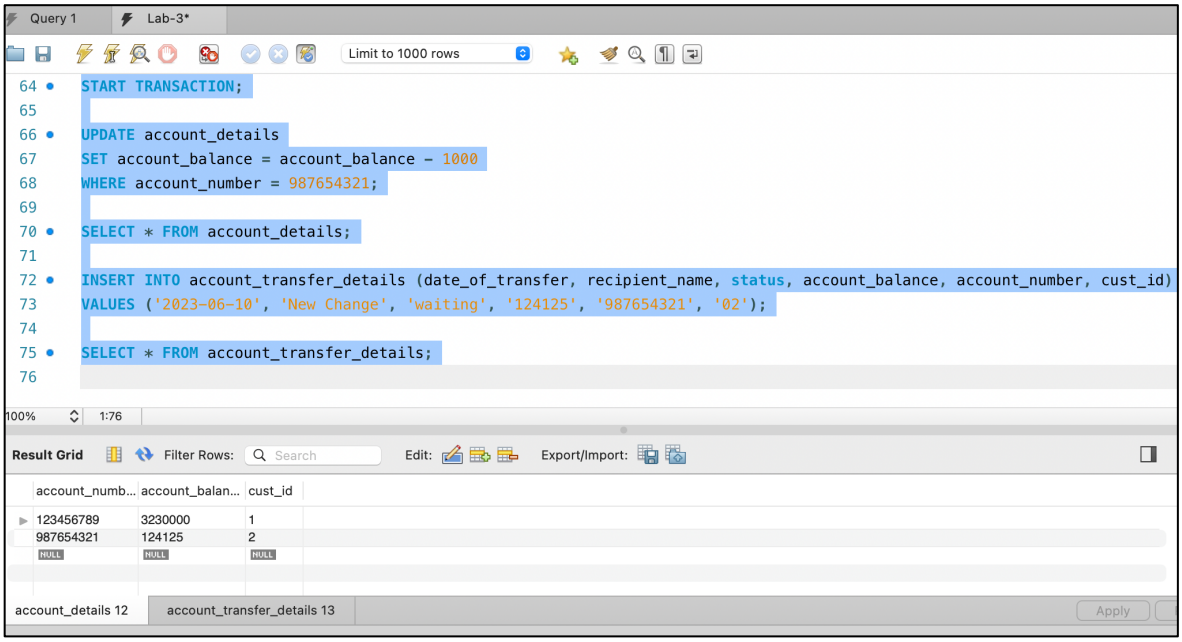


**Figure:9** account _details table after transaction completion

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)

Based on the given requirement, considered 2 scenarios that
- Transaction status is approved.
- Transaction status is approved.

To check the condition if the status is approved or rejected, created a stored procedure for both scenarios [3]
- transaction_verification1: for 'approved' transaction status.
    - This stored procedure performs an update on the account balance, inserts a record into the transfer details table, verifies the transaction status, and either commits or rolls back the changes based on the status. Finally, it retrieves and displays the transfer details records.
- transaction_verification2: for 'declined' transaction status.
    - the stored procedure performs a transaction with a **savepoint**, updates the status in the account_transfer_details table, verifies the status, and either commits or rolls back the changes based on the status. It then returns the updated rows from the table.

**Scenario-1:** For the **approved** transaction status, create a stored procedure to perform the update balance [3], see the **Figure:10** for the same.

**SQL Queries executed:**

```
-- Step-3
-- Step-3.1
-- Scenario-1: For the approved transaction status
-- Create a stored procedure to perform the update balance

DELIMITER //
CREATE PROCEDURE transaction_verification1()
BEGIN
    SET autocommit = 0;

    START TRANSACTION;

    -- Create a savepoint before transaction
    SAVEPOINT before_transaction_update;

     UPDATE account_details
    SET account_balance = account_balance - 1000
    WHERE account_number = 987654321;

    SELECT * FROM account_details;

    INSERT INTO account_transfer_details (date_of_transfer, recipient_name, status,
    account_balance, account_number, cust_id)
    VALUES ('2023-06-10', 'New Change', 'waiting', '124125', '987654321', '02');
```

```
-- Scenario-1: Update status as approved
        UPDATE account_transfer_details
        SET status = 'approved'
        WHERE account_number = 987654321;

        SET @transaction_status = (SELECT status FROM account_transfer_details);

        -- Consider a condition, that if transaction status is aprroved, do commit. Else
        rollback to the SAVEPOINT created
         IF (@transaction_status = 'approved' )
         THEN
                COMMIT;

        ELSEIF @transaction_status = 'declined'
    THEN
                ROLLBACK TO SAVEPOINT before_transaction_update;
        END IF;

    SELECT * FROM account_transfer_details;

END //

DELIMITER ;

-- Call the defined stored procedure
CALL transaction_verification1();
```

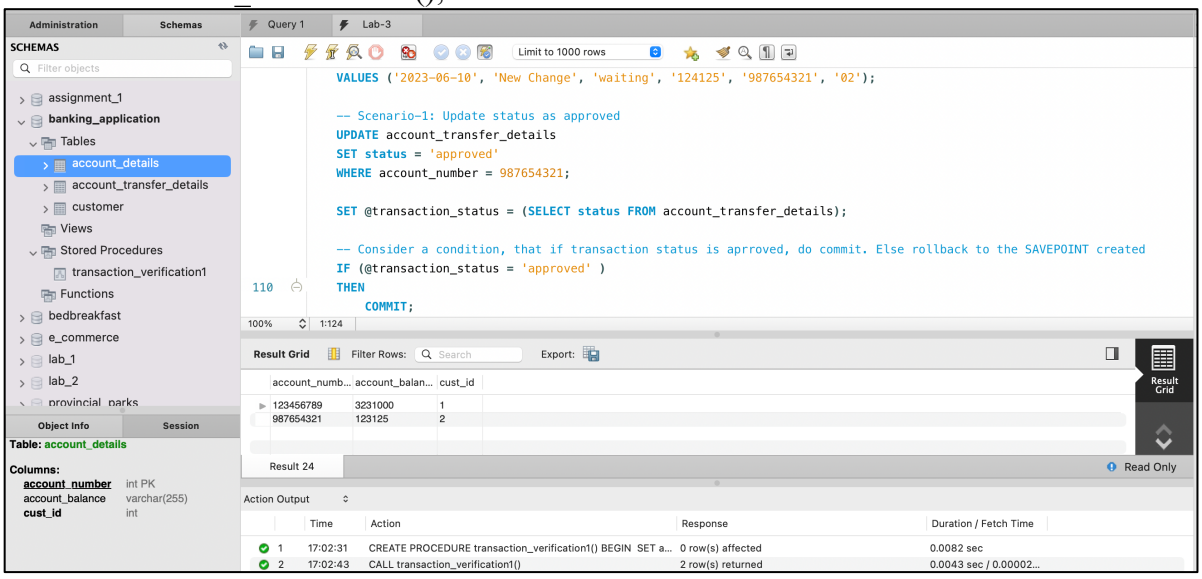


**Figure:10** updated account _details table after executing stored procedure for Approved status

**Scenario-2:** For the **declined** transaction status, create a stored procedure to perform the update balance [3], see the **Figure:11** for the same.

**SQL Queries executed:**

```
-- Step-3.2
-- Scenario-2: For the declined transaction status
-- Create a stored procedure to perform the update balance

DELIMITER //
CREATE PROCEDURE transaction_verification2()
BEGIN
	SET autocommit = 0;

	START TRANSACTION;

	-- Create a savepoint before transaction
	SAVEPOINT before_transaction_update;

   -- Scenario-2: Update status as declined
	UPDATE account_transfer_details
	SET status = 'declined'
	WHERE account_number = 987654321;

	SET @transaction_status = (SELECT status FROM account_transfer_details);

	-- Consider a condition, that if transaction status is aprroved, do commit. Else
	rollback to the SAVEPOINT created
	IF (@transaction_status = 'approved' )
	THEN
		COMMIT;

	ELSEIF @transaction_status = 'declined'
	THEN
		ROLLBACK TO SAVEPOINT before_transaction_update;
	END IF;

	SELECT * FROM account_transfer_details;

END //

DELIMITER ;

-- Call the defined stored procedure
CALL transaction_verification2();
```

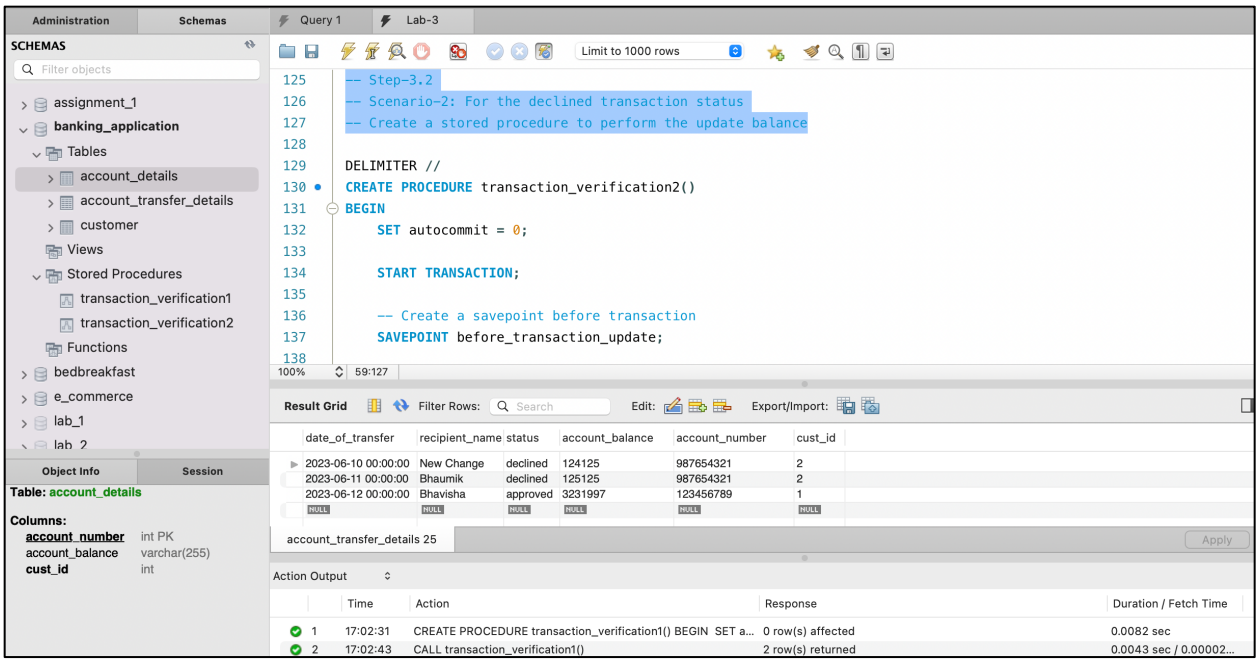


**Figure:11** updated account _transfer_details table after executing stored procedure for Declined status

## References:


[1] "MySQL Community Downloads," *MySQL* [Online]. Available: https://dev.mysql.com/downloads/workbench/ [Accessed: May 10, 2023].

[2] "Lab-3," *Brightspace Dalhousie University* [Online]. Available: https://dal.brightspace.com/d2l/le/content/271677/viewContent/3634711/View [Accessed: June 5, 2023].

[3] "MySQL CREATE PROCEDURE," *MySQLTUTORIAL* [Online]. Available: https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx [Accessed: June 7, 2023].