CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

ASSIGNMENT – 1

Problem2: Testcases and evidence of testing

Banner ID: B00948977

Git Assignment Link:
https://git.cs.dal.ca/sukumaran/csci5408_f23_b00948977_balaji_sukumaran/-/tree/main/Assignment1

Table of contents

Section 1: **Task A** Authentication (login and registration) testing

Step 1: Registration flow

Was able to open the application, following is the main menu.

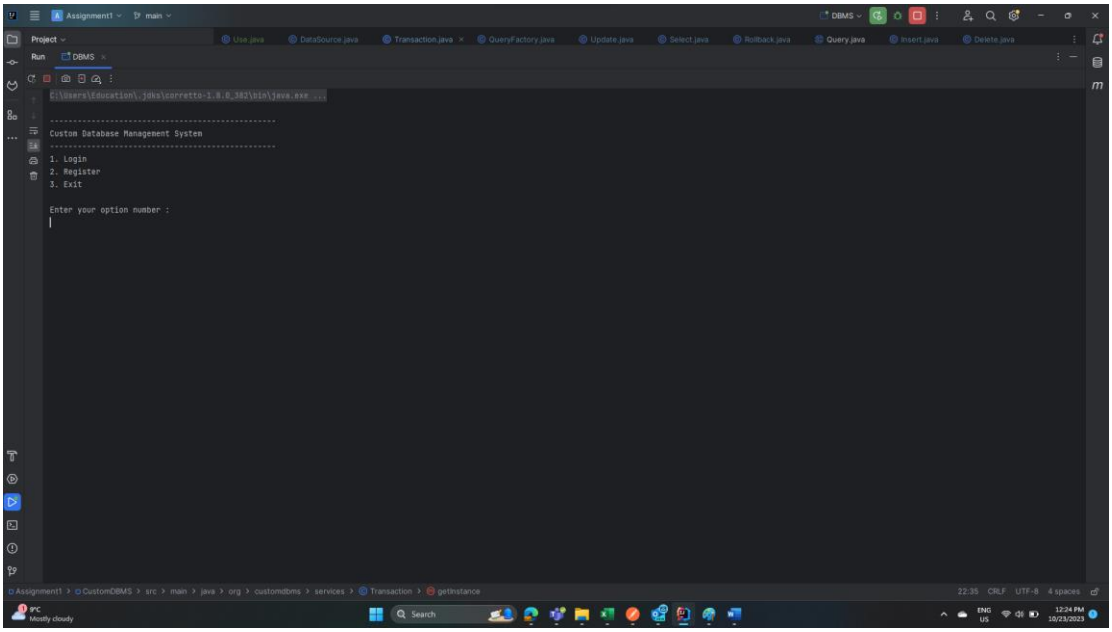


Figure 1: Custom DBMS main menu

Select option 2 to register but type an **invalid captcha**

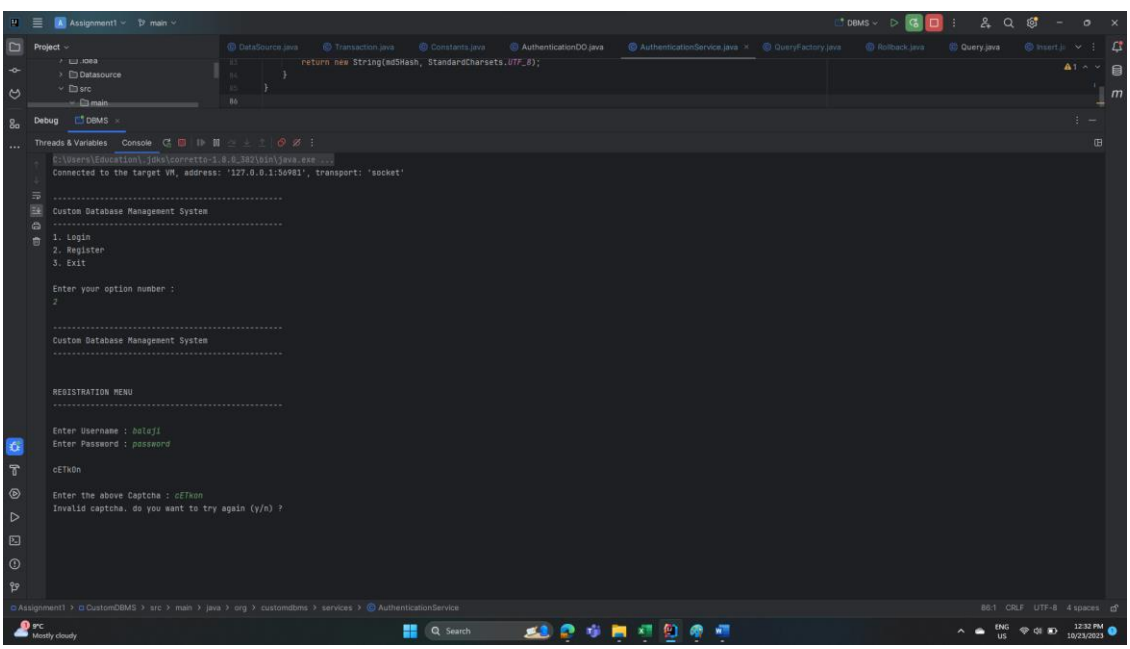


Figure 2: Registration menu invalid captcha

Retry again and give correct captcha. **User registered successfully.**

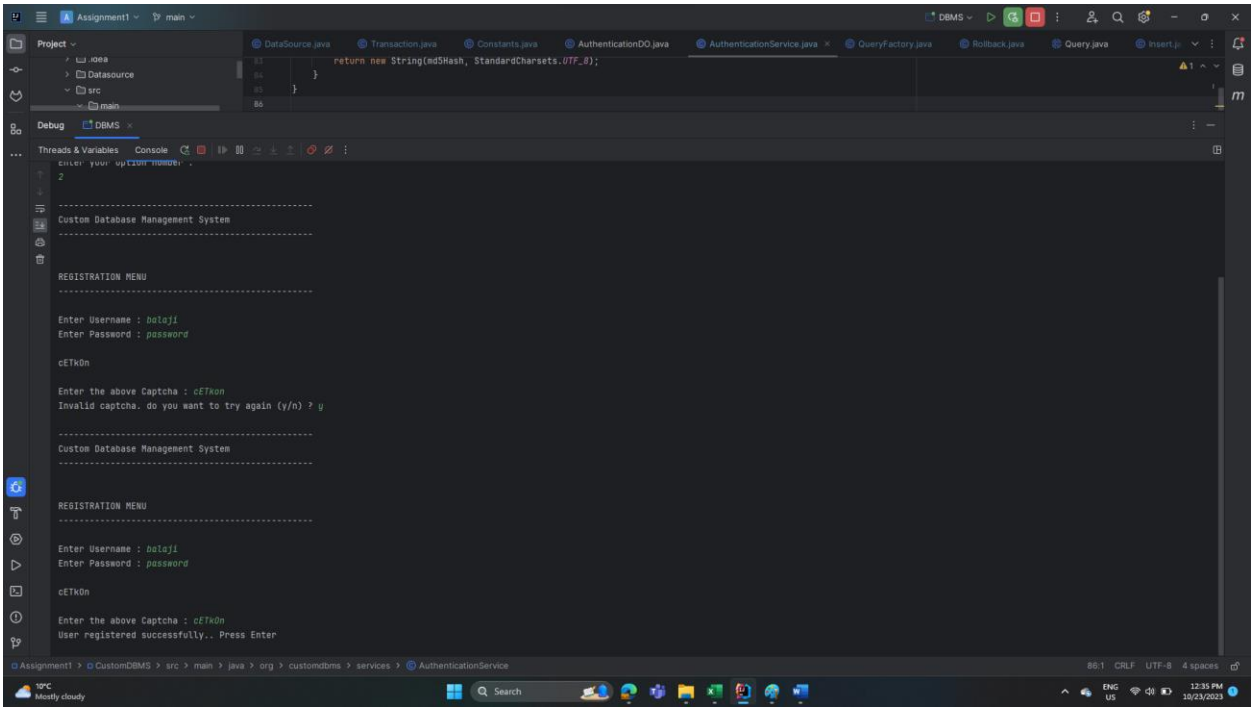


Figure 3: user registered successfully.

Credentials stored in the systems folder based on the application config and password is encrypted in the MD5 hashing algorithm.

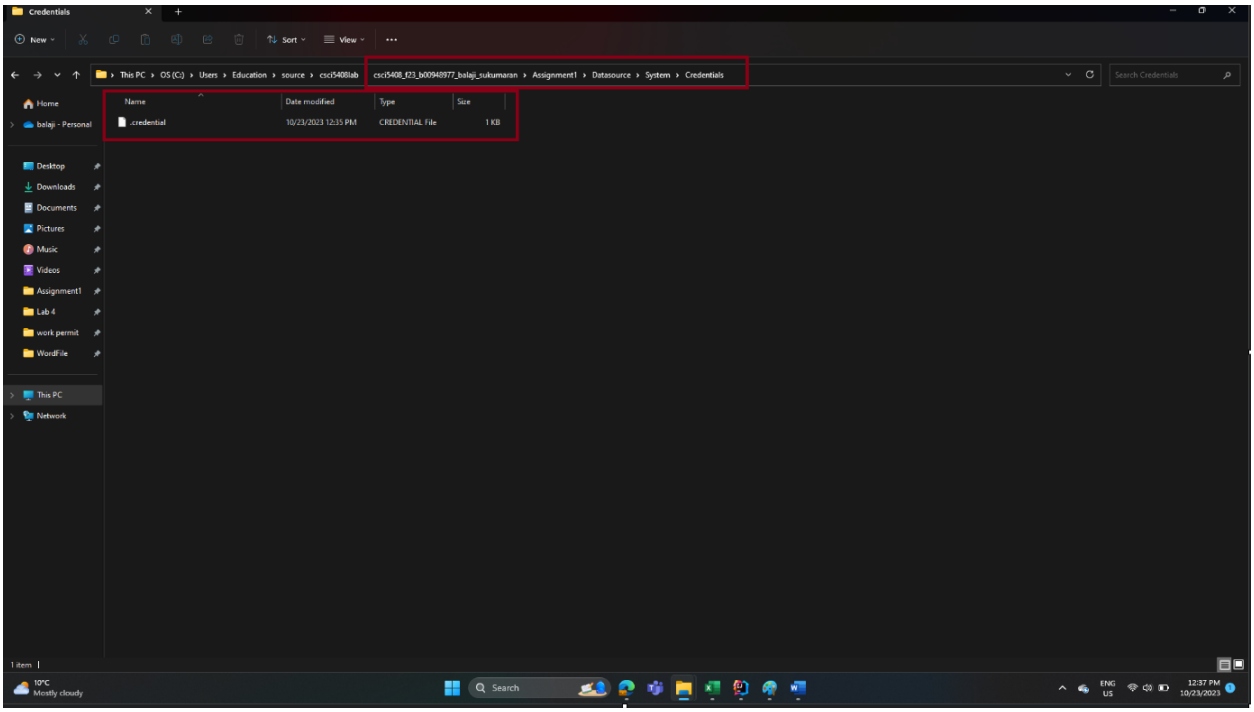


Figure 4: credential file folder structure

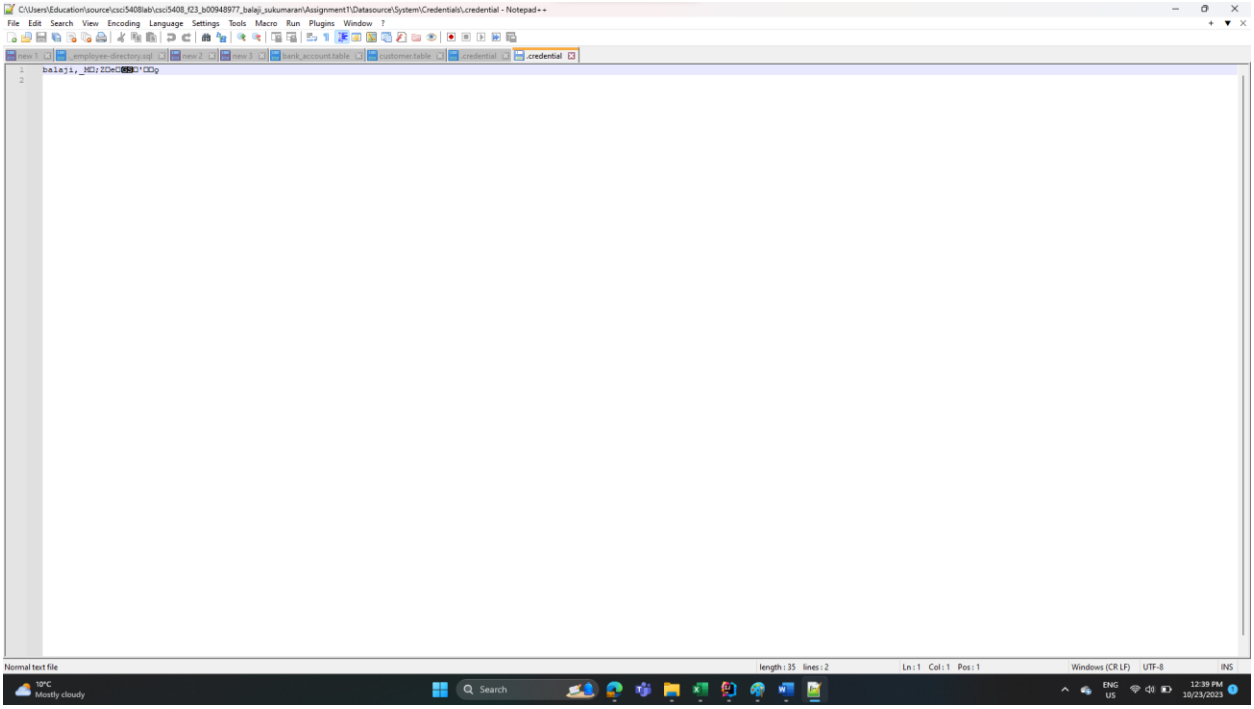


Figure 5: Encrypted password stored

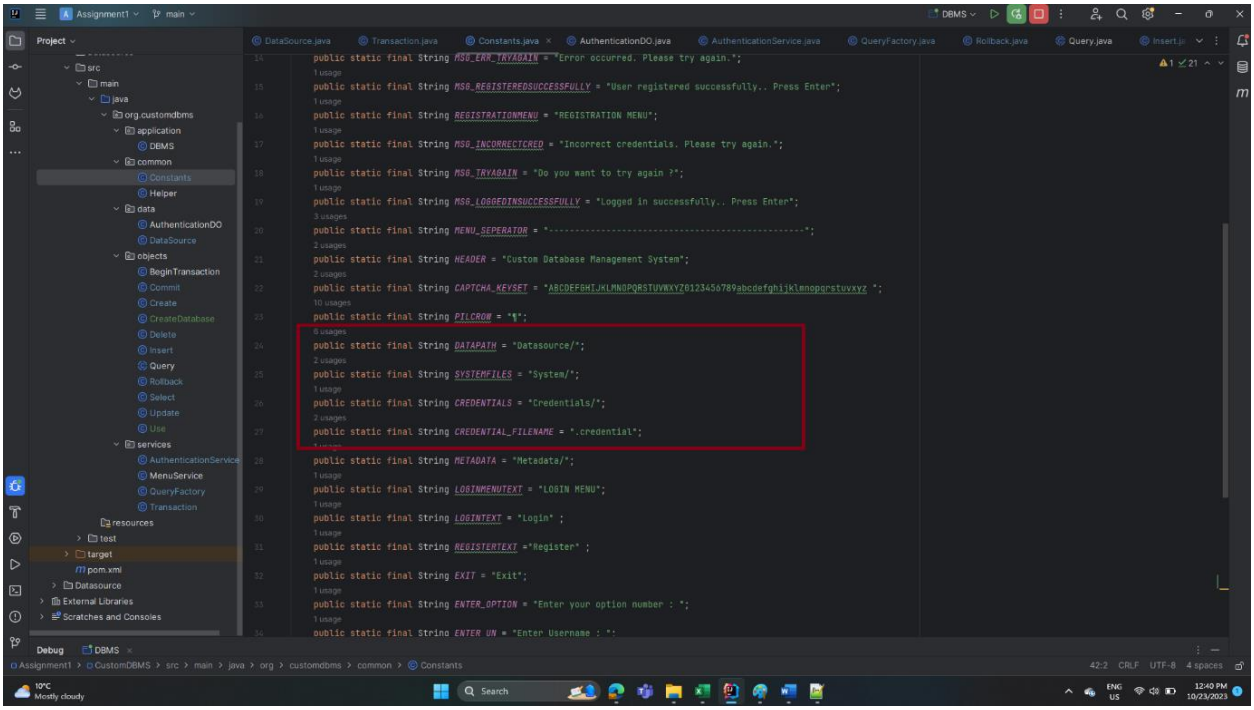


Figure 6: Credential folder configuration

Step 2: Login Flow,

Tried login using incorrect password and the system has showed the error.

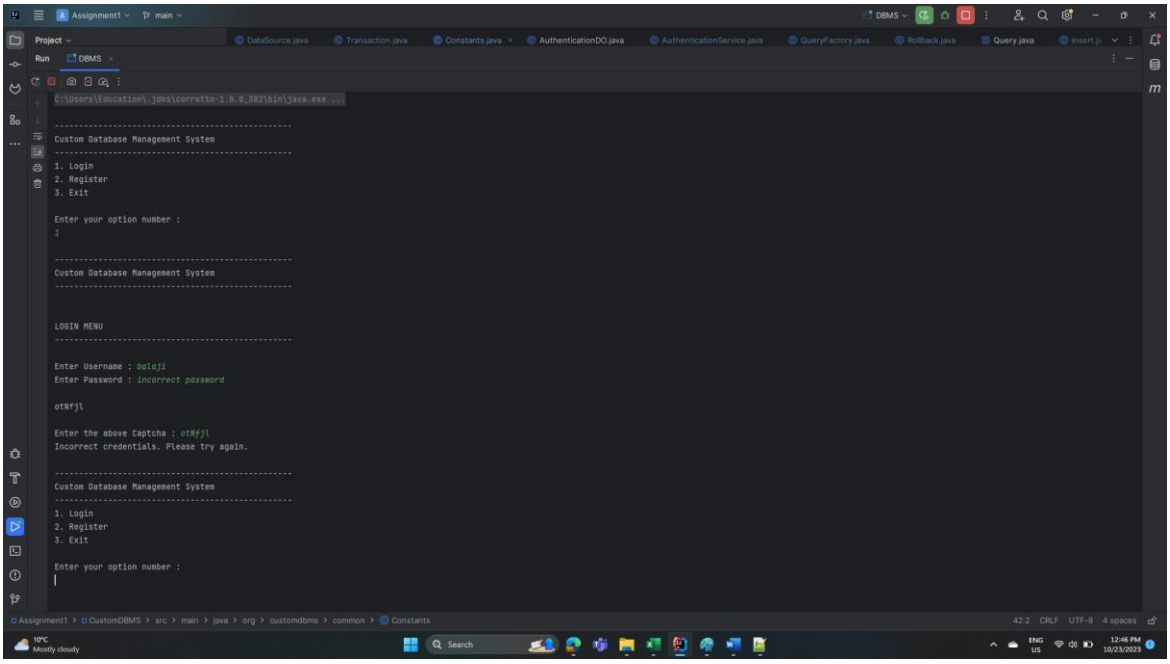


Figure 7: Login using incorrect password.

Tried login using correct password and the system has authenticated the user successfully and opened the query window.

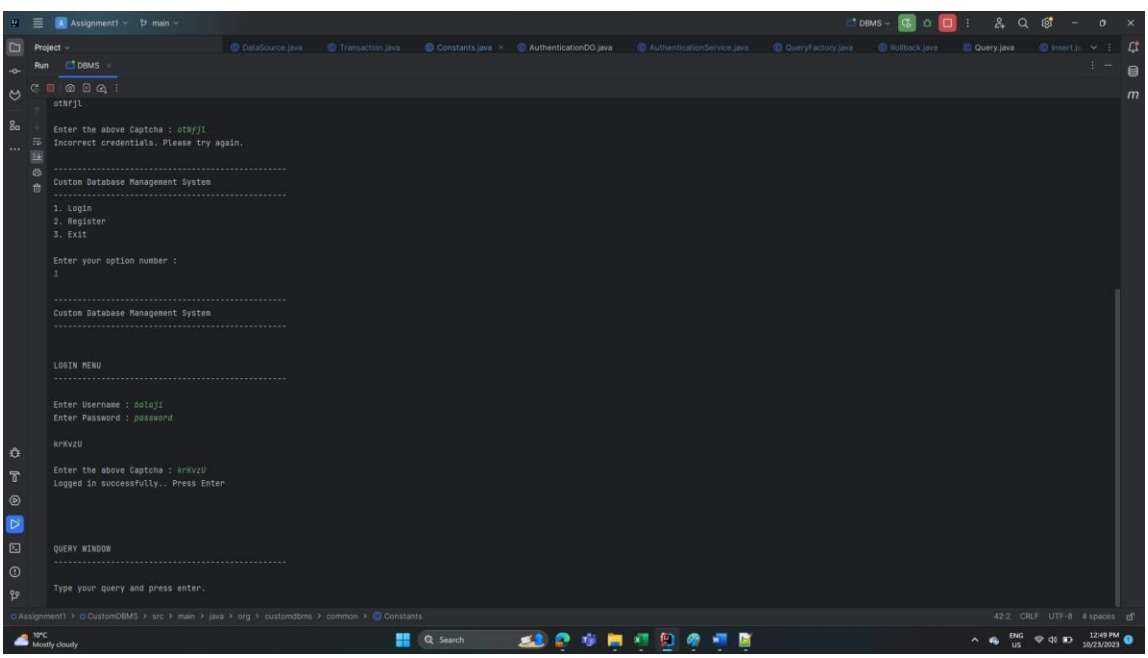


Figure 8: Login successful using user id and password

Internally the password is hashed in the following method using MD5 algorithm.

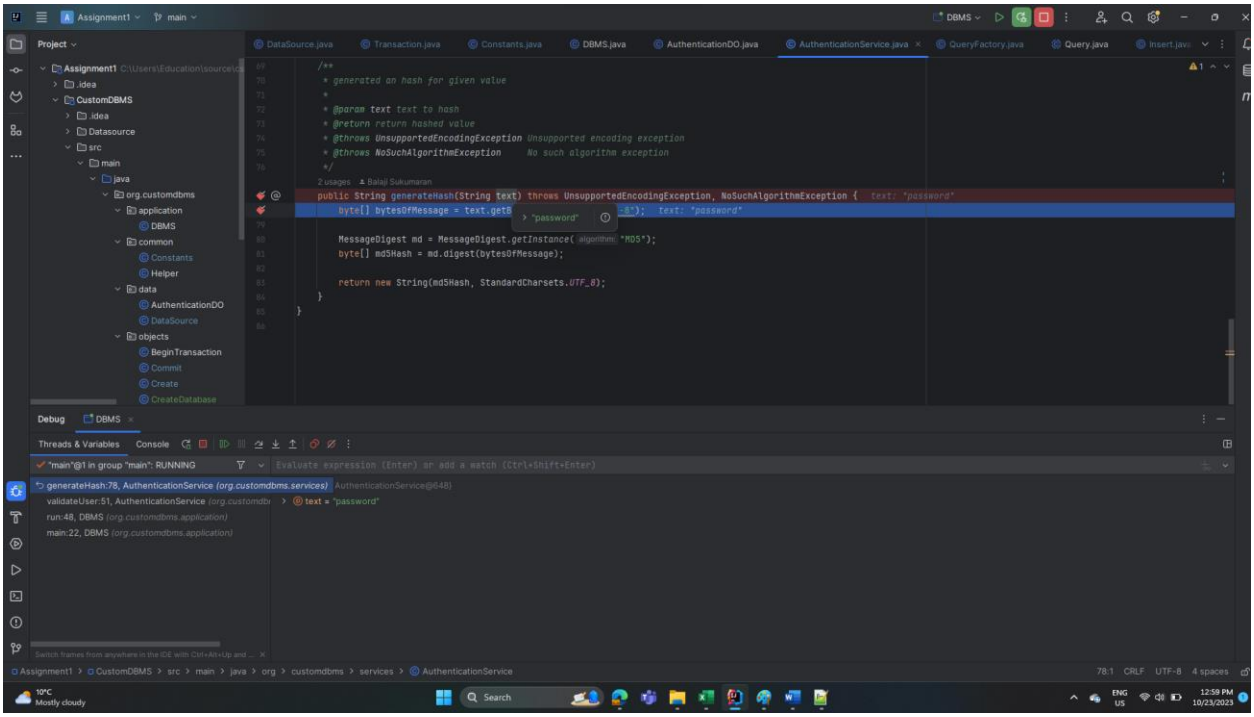


Figure 9: User password getting MD 5 hashed in this method

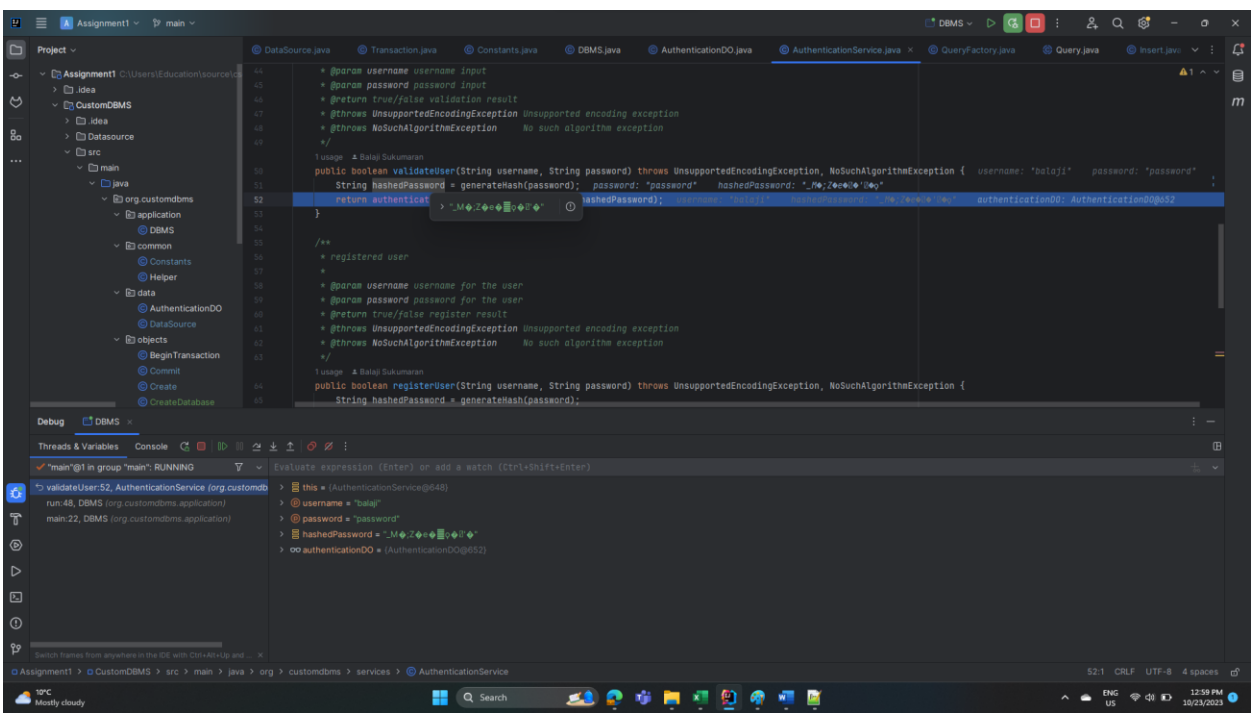


Figure 10: Hashed password

Section 2: **Novelty Task:** Data File

- the Data Files are stored in a Pilcrow ¶ separated file.

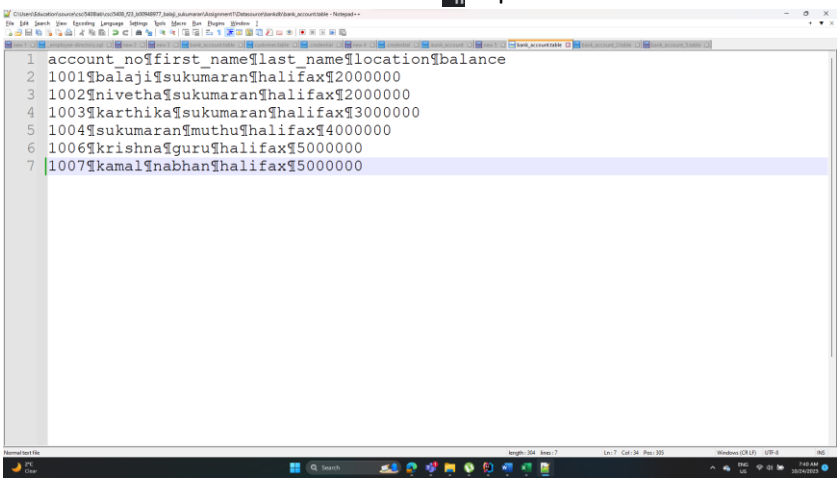


Figure 11: Pilcrow separated file

- Table meta-data will be stored in .(table_name) extension. For example table: bank_account meta data file will be stored as .bank_account

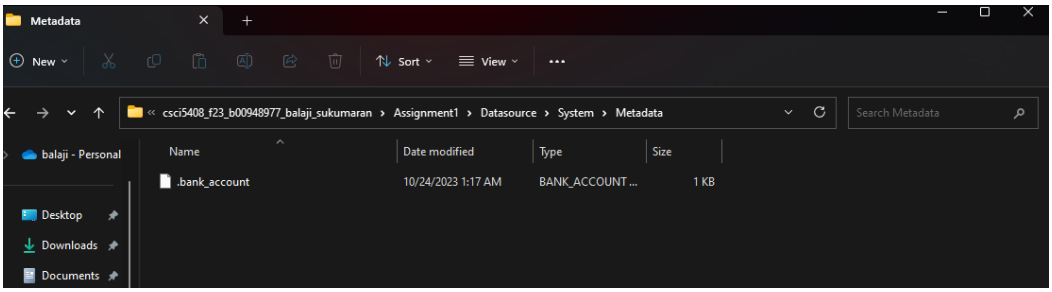


Figure 12: meta data file stored as .<tablename>

- Table file will be stored in .table extension. For example table: bank_account table file will be stored as bank_account.table.

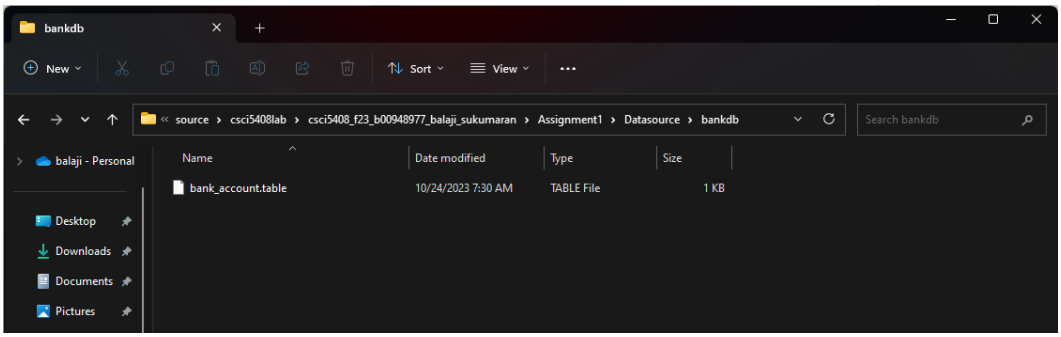


Figure 13: table file stored in.table extension

Section 3: **Task B** Implementation of Queries (DDL & DML) testing.

Step 1: Create database.

Entered “create database bankDB;” in the query window it executed successfully.

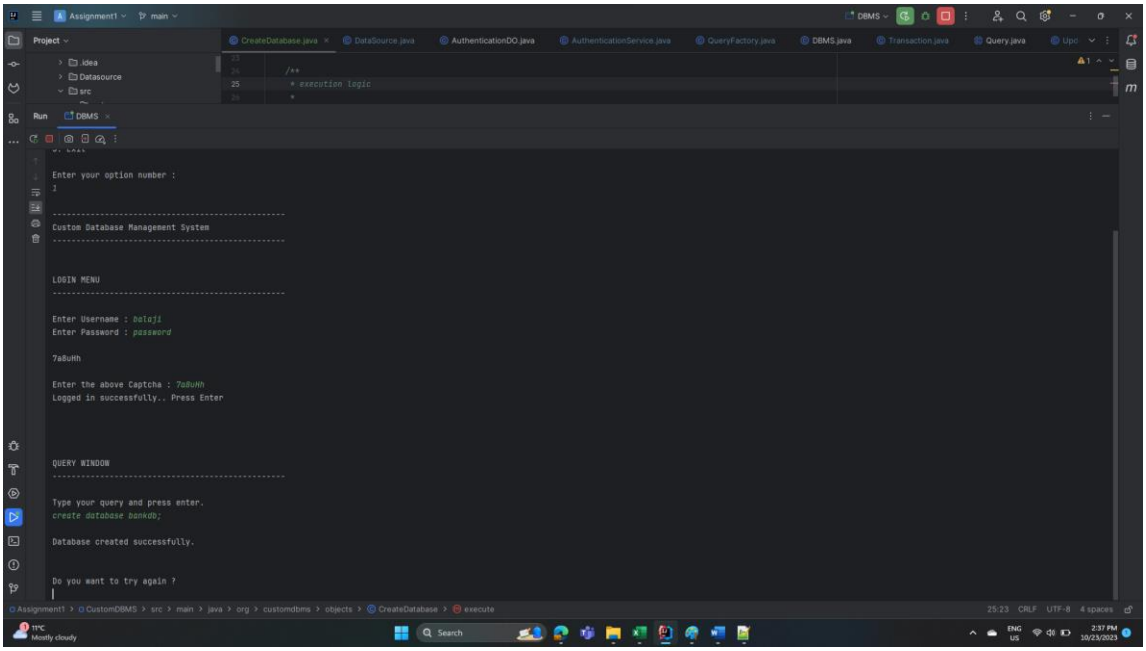


Figure 14: database created successfully

Internally it created a folder in the file system.

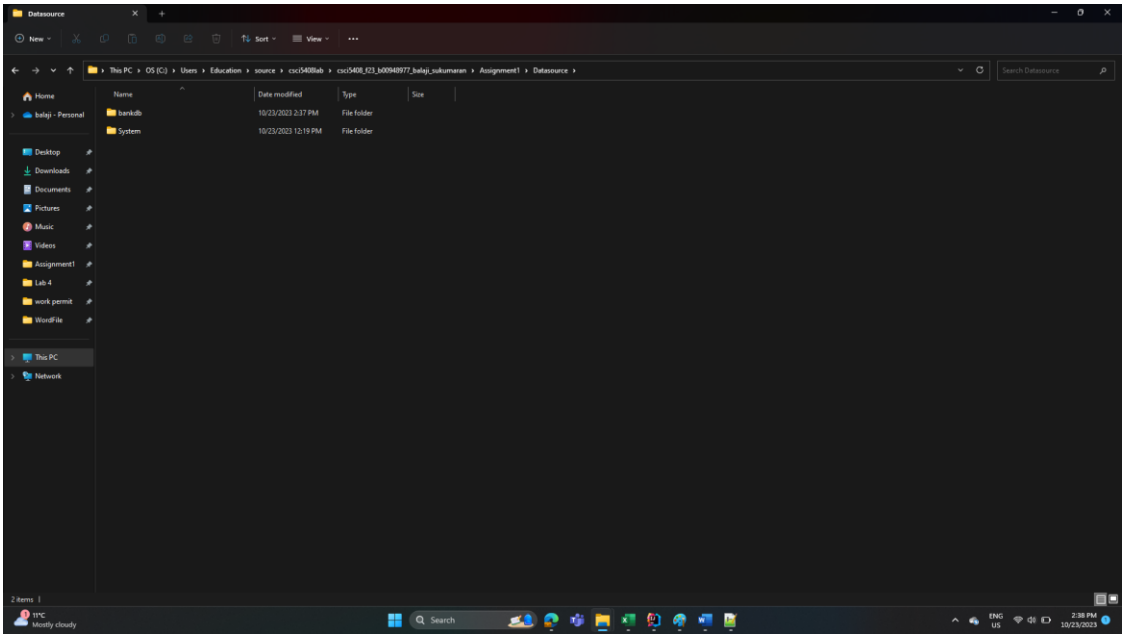


Figure 15: Folder created in the file system

Step 2: Tried to use the created database.

Execute “use bankdb”

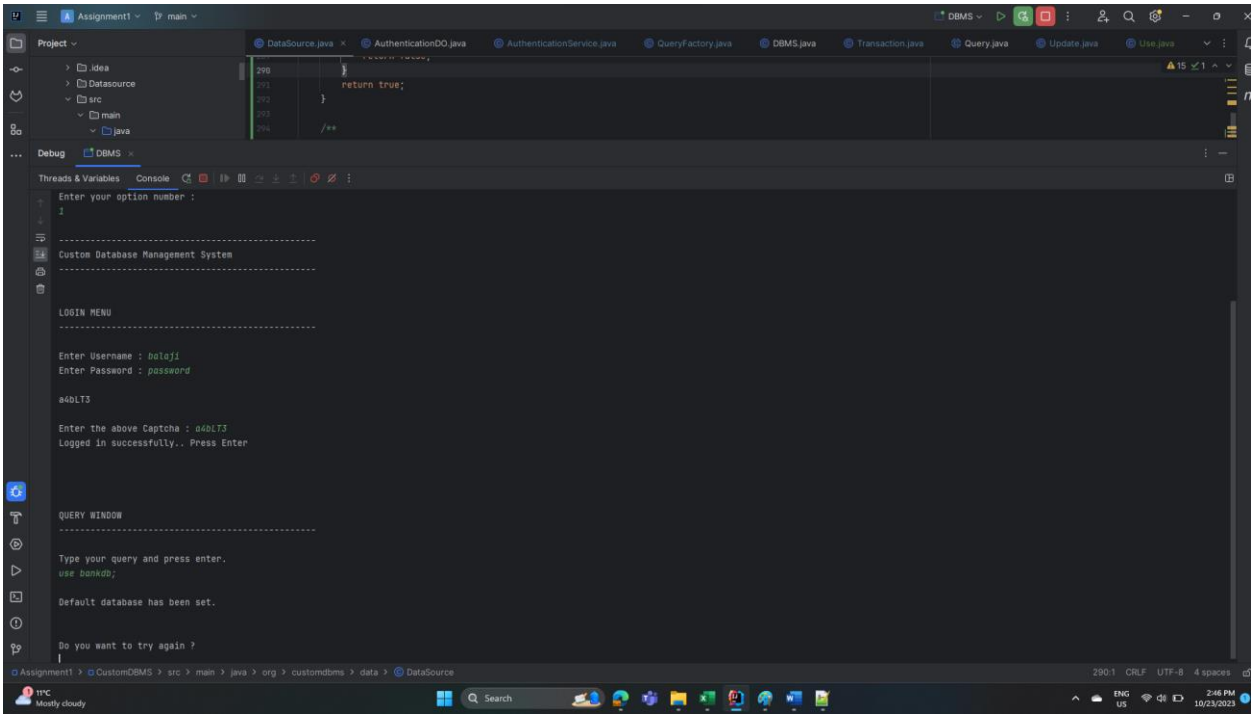


Figure 16: Default database chosen.

Internally, It marks the application where all the tables exists

```
/**
 * set the database name
 *
 * @param dbName db name
 * @return set status
 */
3 usages  new *
public boolean setDBName(String dbName) {   dbName: "bankdb"
    try {
        File file = new File( pathname: Constants.DATAPATH + dbName + "/");

        if (!file.exists())
            file.mkdirs();

        databaseName = dbName;   dbName: "bankdb"
        tablePath = tablePath + databaseName + "/";   tablePath: "Datasource/bankdb/"   databaseName: "bankdb"
    } catch (Exception e) {
        return false;
    }
    return true;
}
```

Figure 17: table access path has been set.

Step 3: Create a Table

Created a table using the “create table bank_account(account_no int primary key, first_name varchar(50), last_name varchar(50), location varchar(10), balance int)”. query executed successfully.

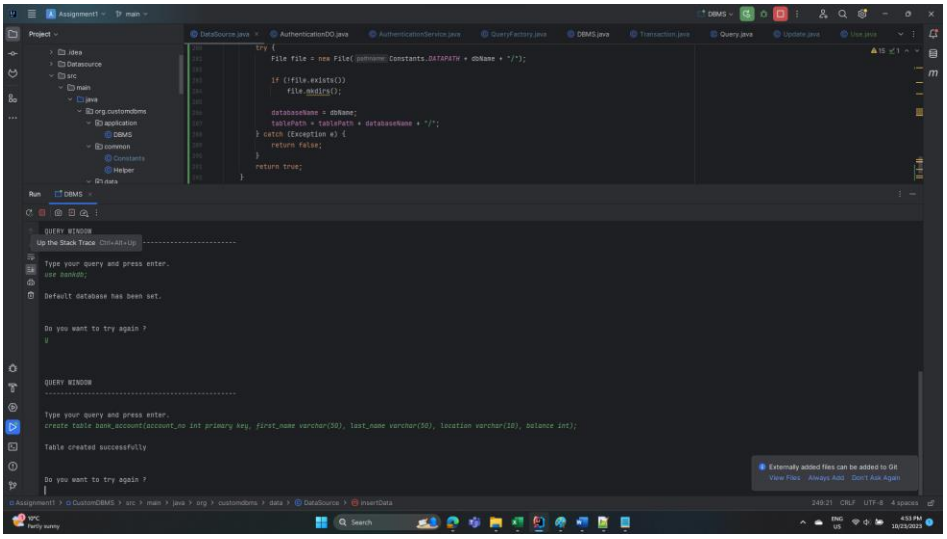


Figure 18: create table query

In the file system it created two files .bank_account contains the meta data about the table and the actual table is contained in bank_account.table

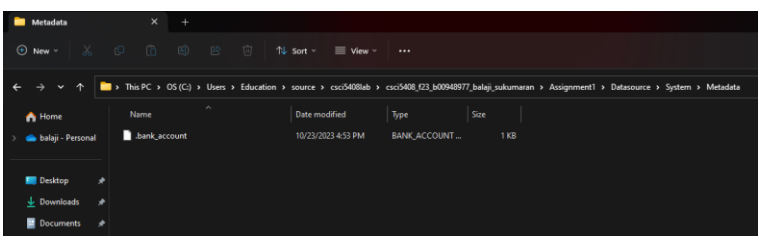


Figure 19: new file created for table metadata

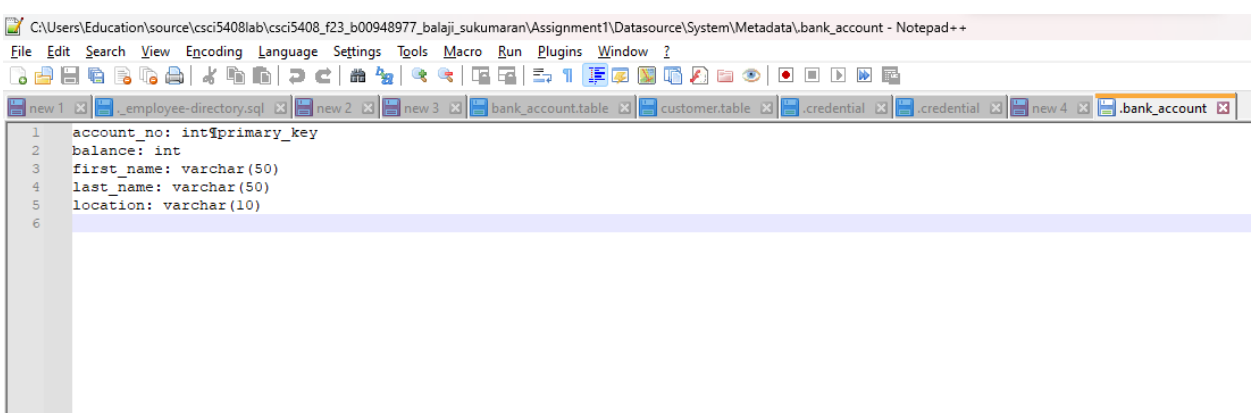


Figure 20: meta-data file contents

Table file is created under the database folder bankdb.

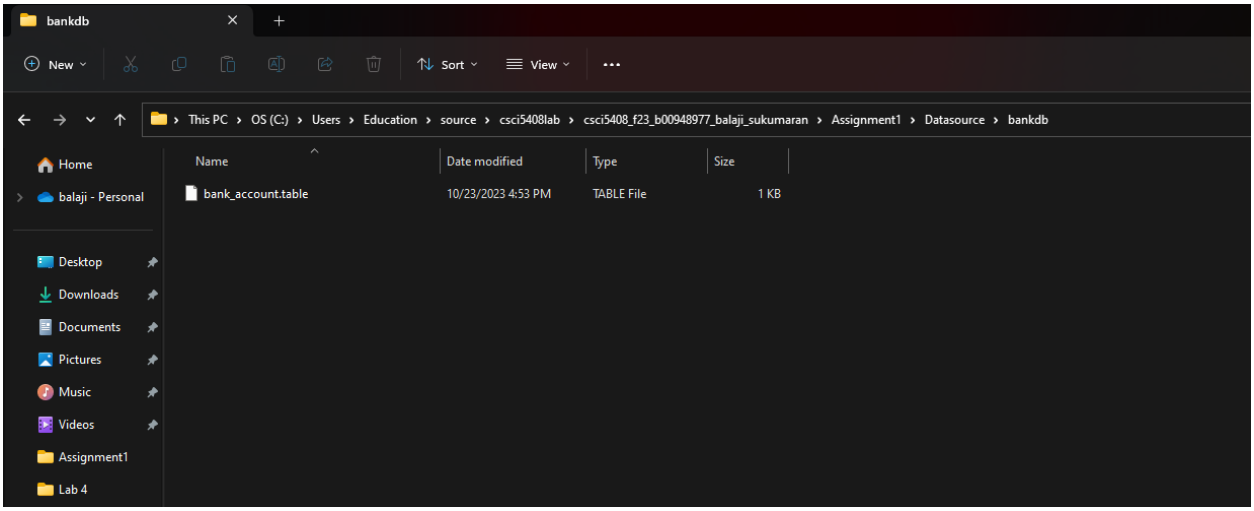


Figure 21: table file created.

Following is the table file contents.

It is a Pilcrow ¶ separated file.

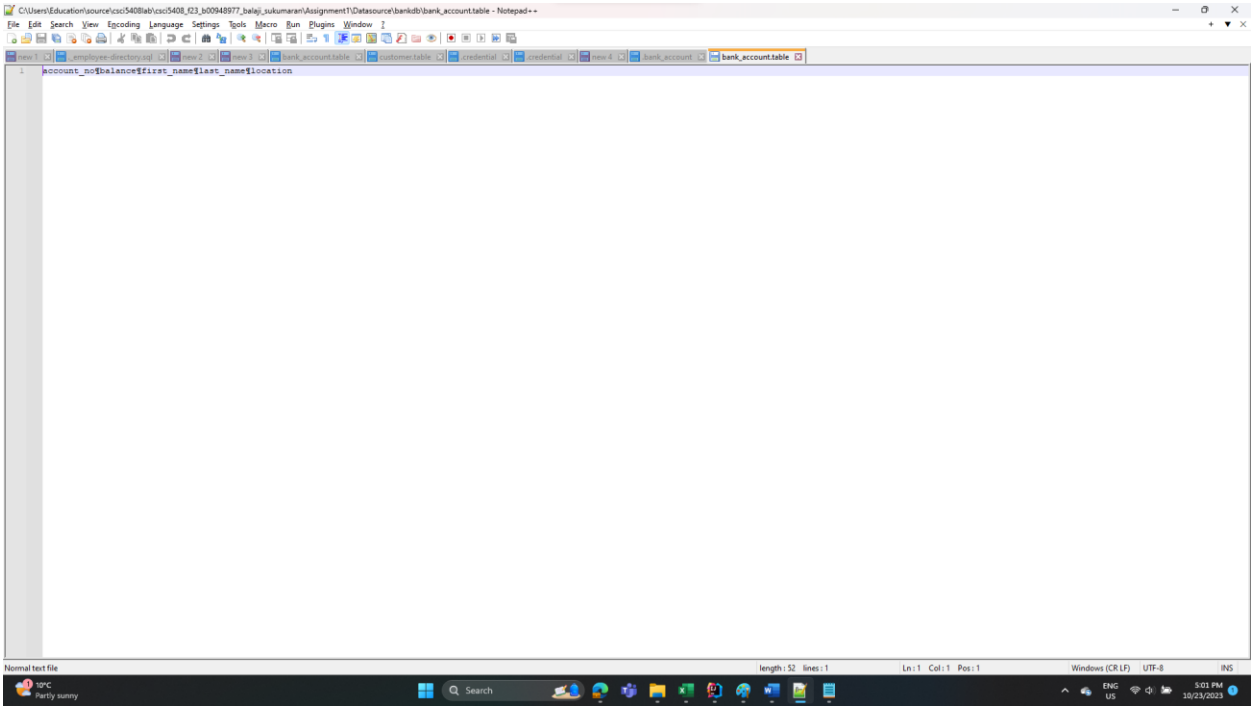


Figure 22: Pilcrow separated file.

Step 4: Insert into a table.

Tried to insert one row into the bank_account table. Record inserted successfully.

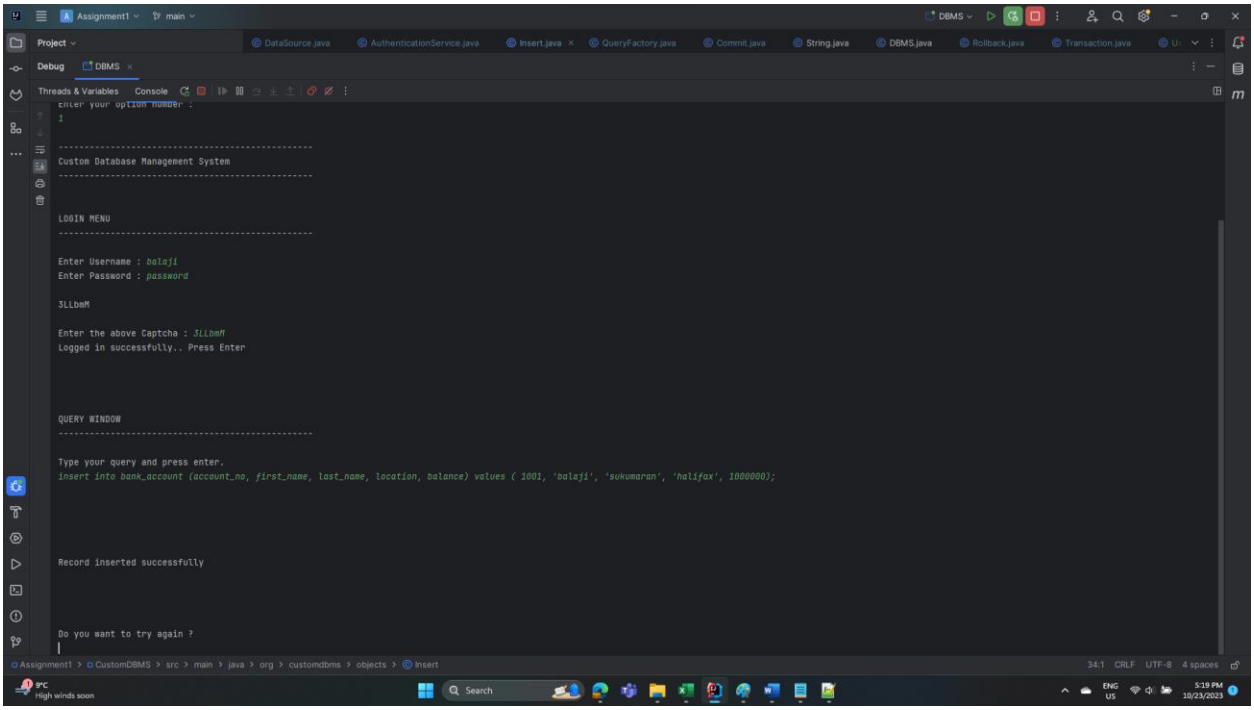


Figure 23: Record inserted successfully.

The updated table file looks like this.

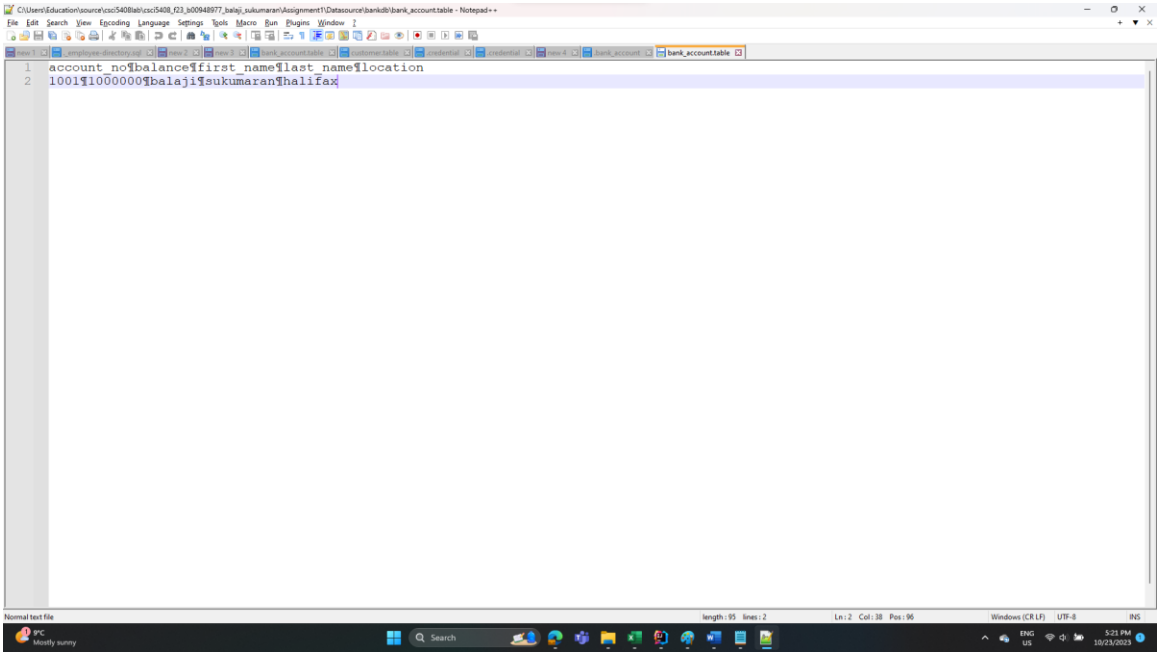


Figure 24: Bank_account table file after single record insertion.

Tried to insert multiple records. One after another single record. All the records are inserted successfully.

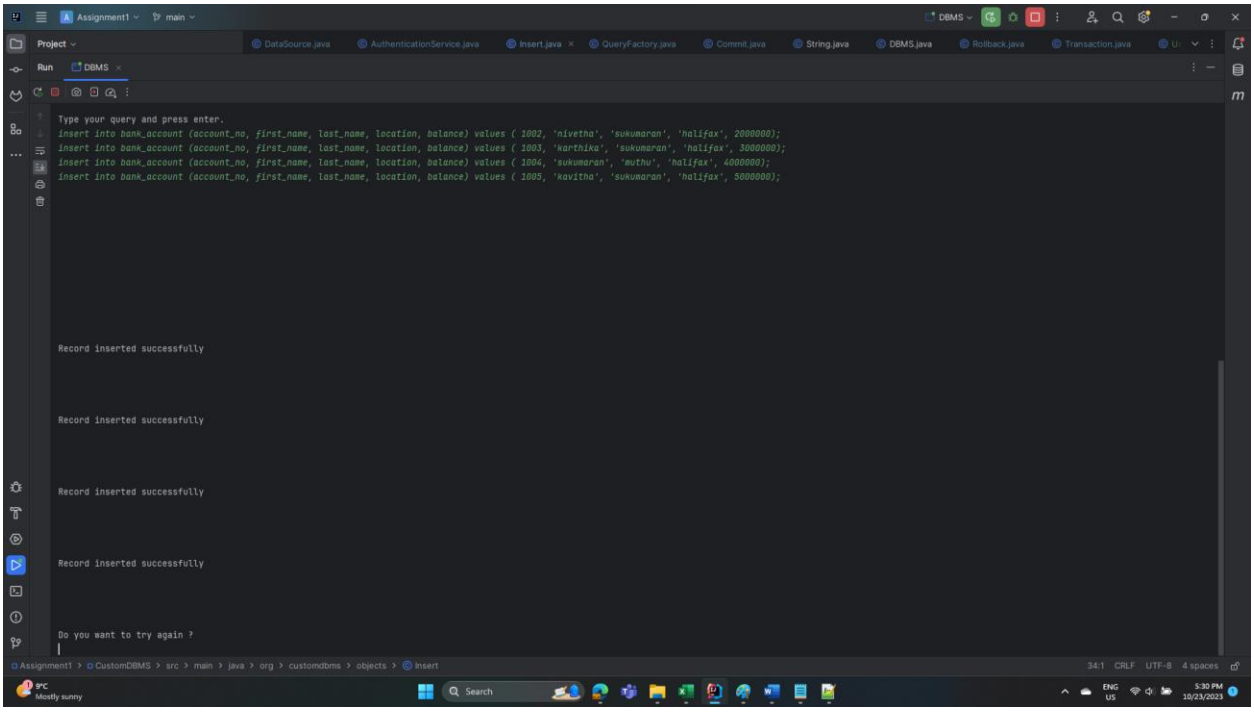


Figure 25: Inserting multiple records successfully.

The updated table file all the records looks like this.

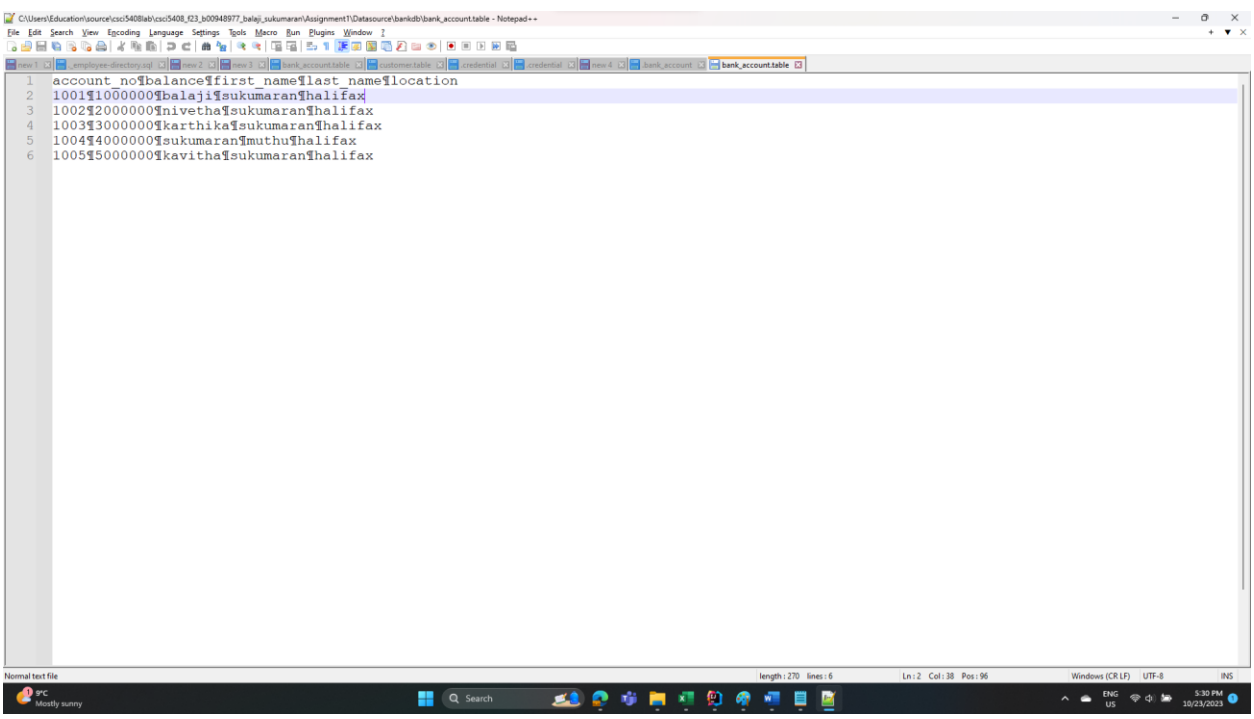


Figure 26: bank_account.table file with 5 records

Step 5: Select from the bank_account table.

Executed “Select * from bank_account” query

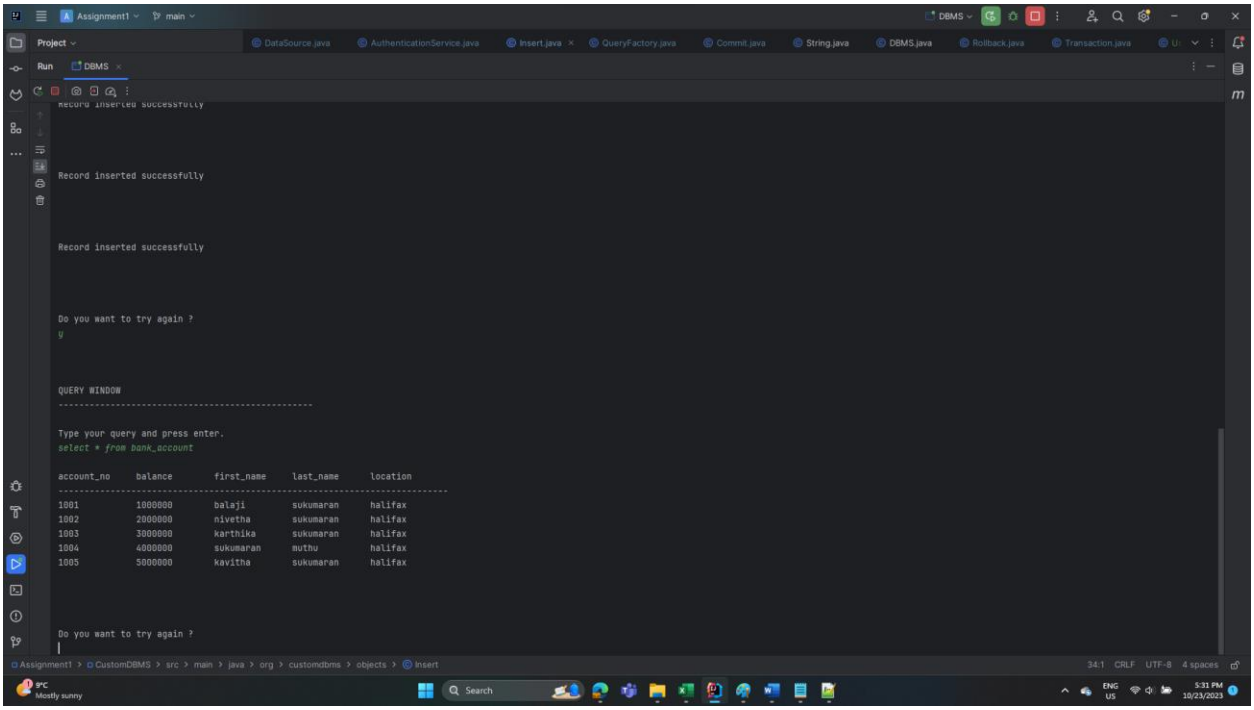


Figure 27: select all from bank_account

Selected few columns and applied a where clause. The records were fetched correctly

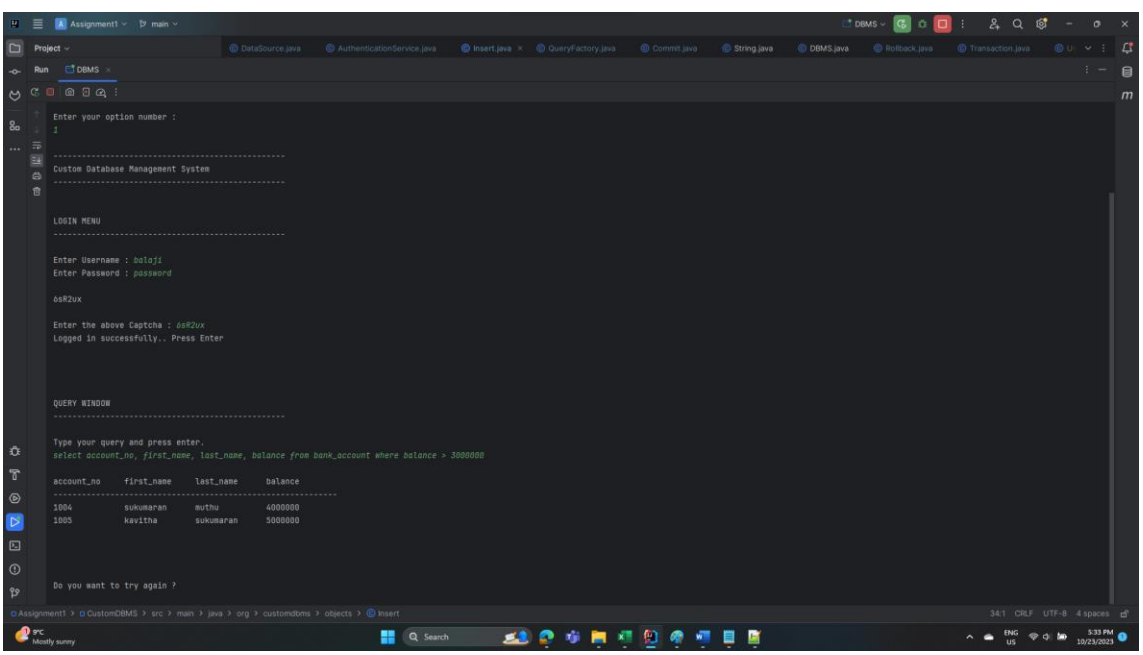


Figure 28: select and where clause

Step 6: update the bank_account table.

Execute the “update bank_account set location='toronto' where account_no = 1001”. Record updated successfully.

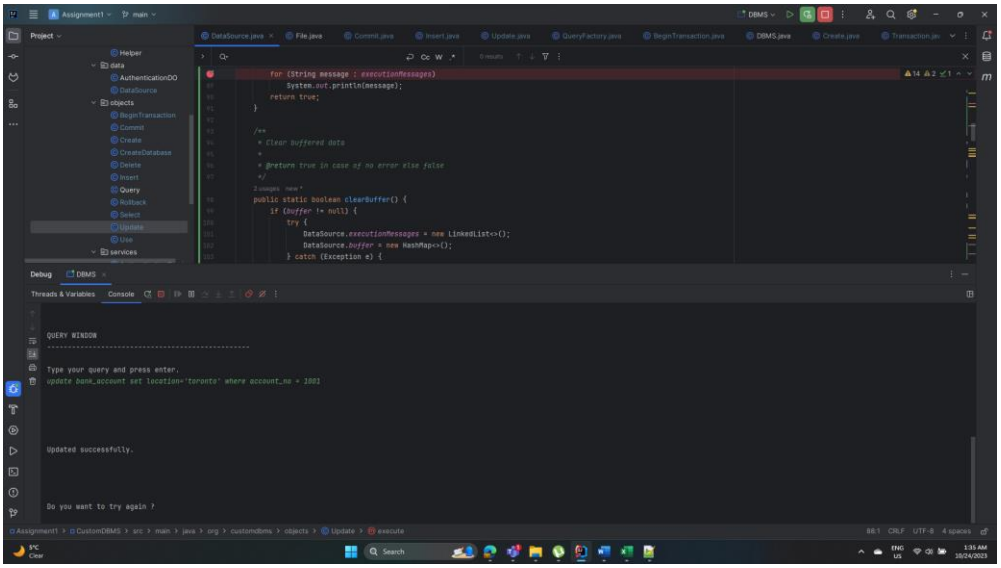


Figure 29: Record updated successfully.

The file in the file system got updated according

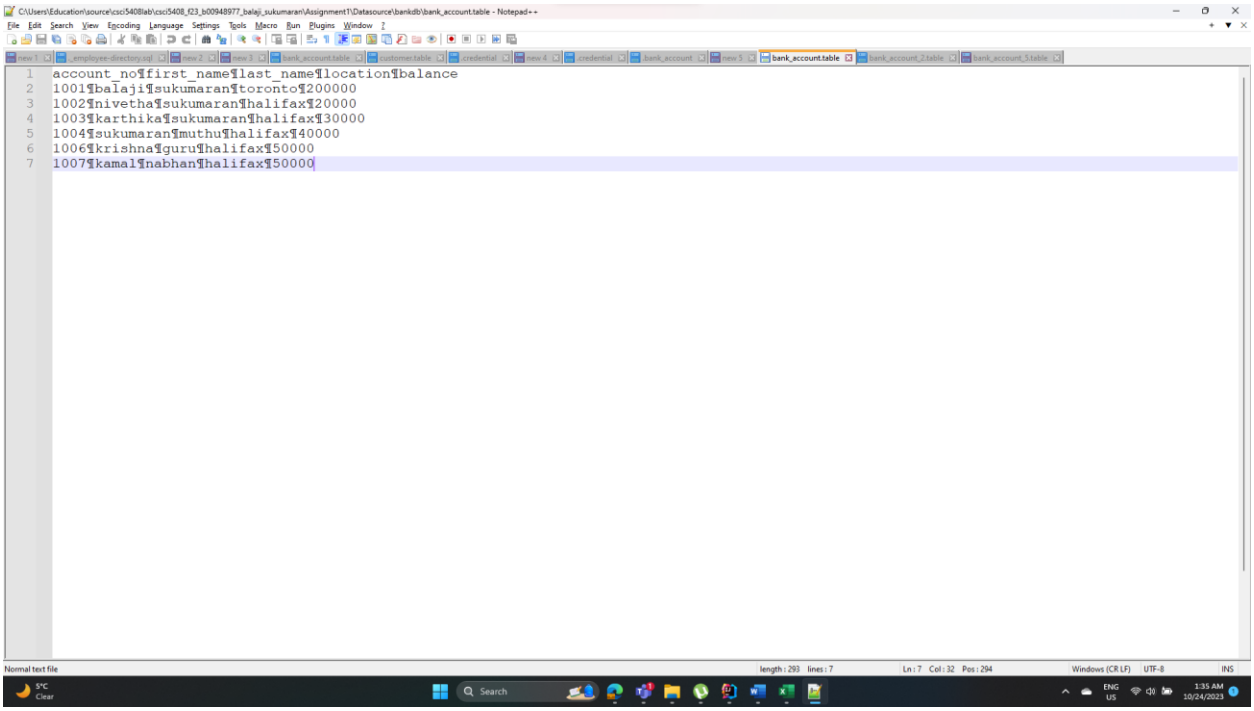


Figure 30: bank_account table file after update

Step 7: Delete records from bank_account table.

Executed “delete from bank_account where account_no=1001”. Record executed successfully.

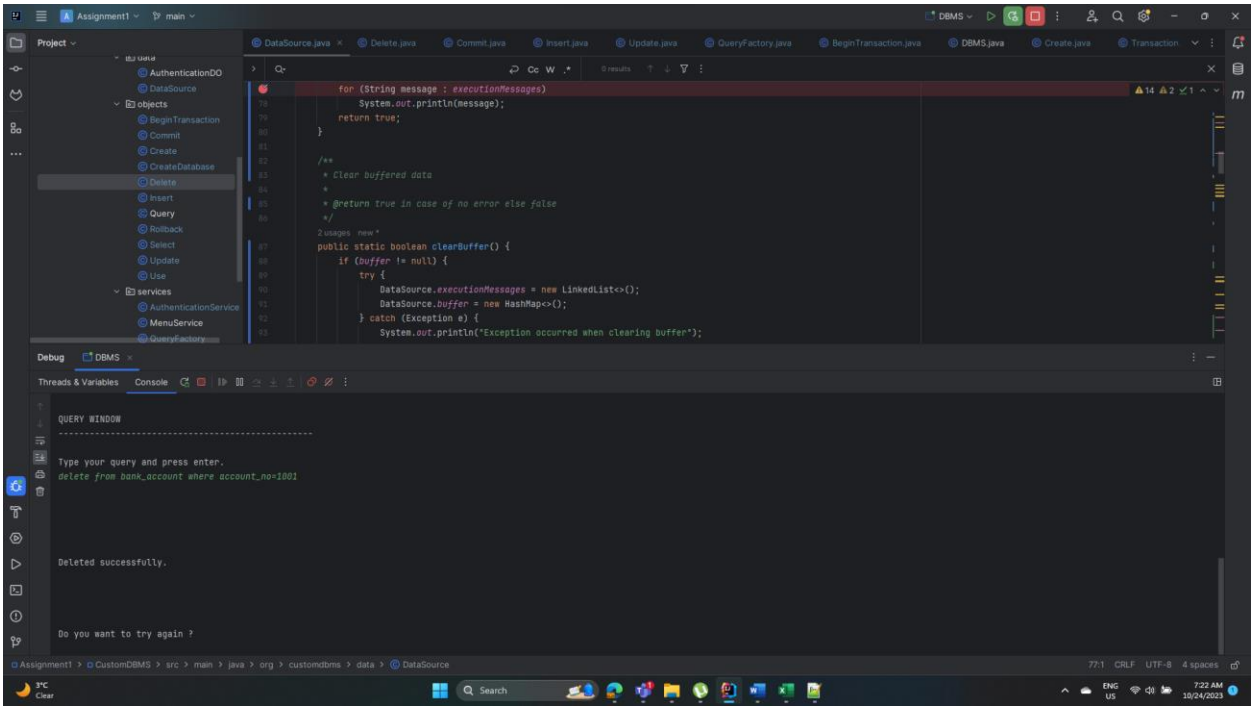


Figure 31: Record deleted successfully.

The table file in the filesystem has been updated to the following (record: 1001 is deleted)

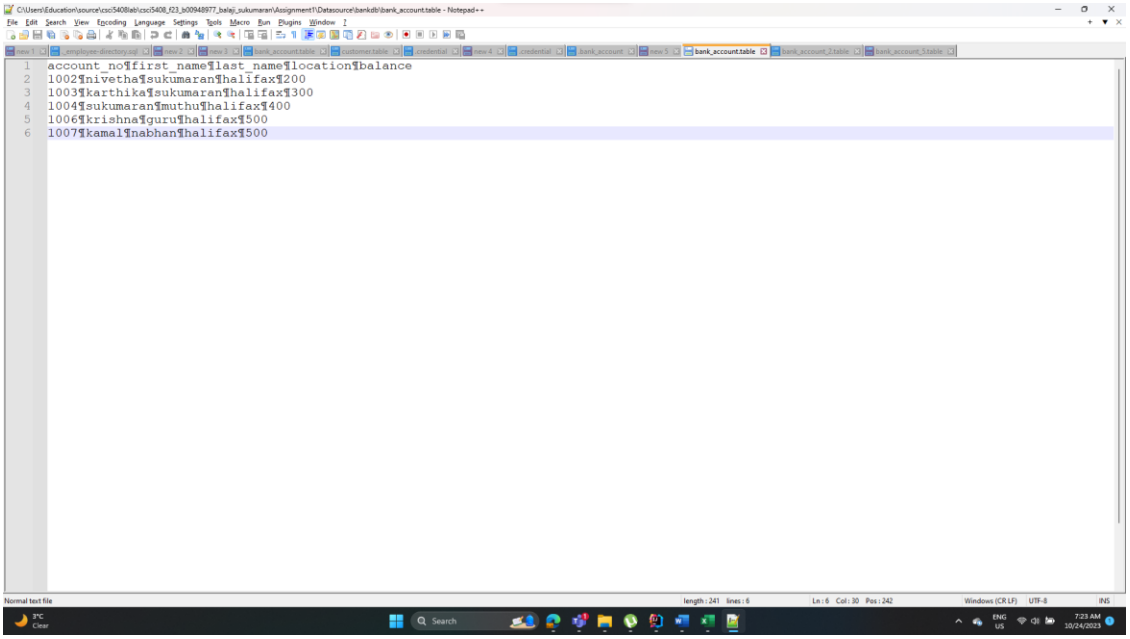


Figure 32: Record deleted from the table file.

Section 4: **Task C**, Implementation of transaction testing

Table setup

Bank_account: bank_account {account_no, first_name, last_name, location, balance}

Bank_transfer details: bank_transfer_details {transaction_id, sender_no, receiver_no, transfer_amt}

We have 5 bank accounts in **bank_account** table and once a transfer has been made it should be logged in the **bank transfer details table**

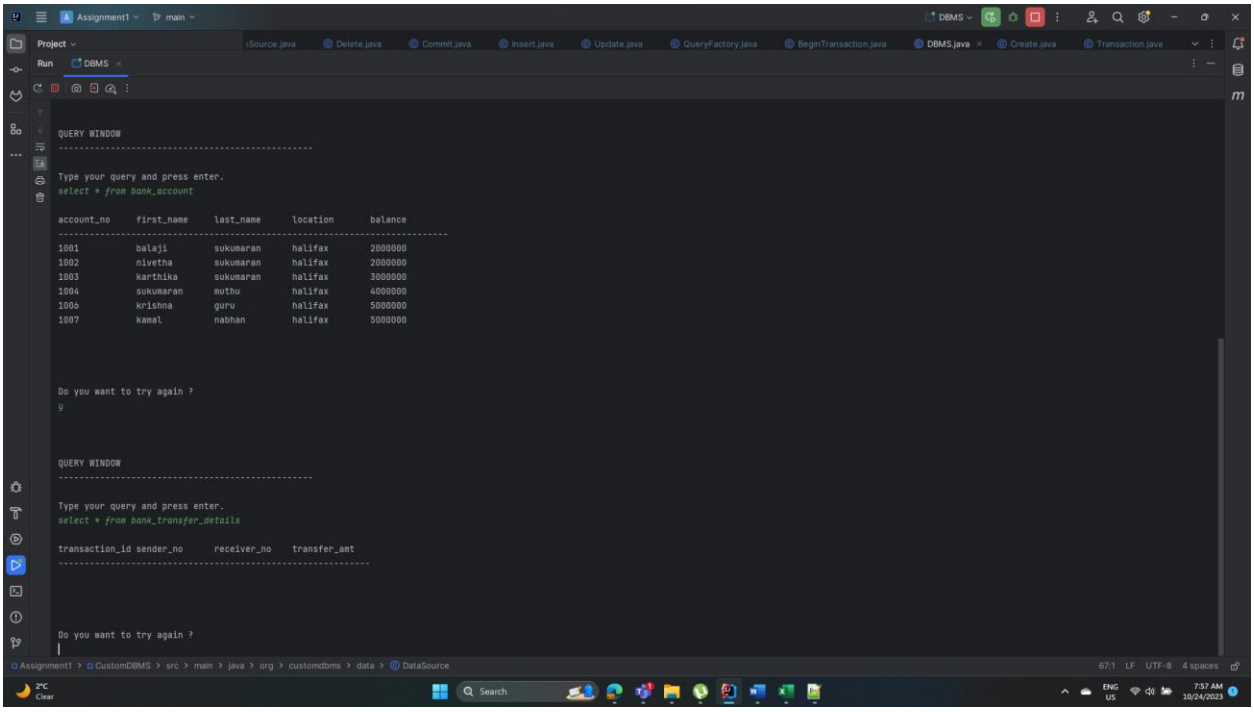


Figure 33: bank_account and bank_transfer_details tables

Commit Flow and its intermediate state of the buffer.

Trying to execute the following query to simulate a banking transaction.

1. Account no 1001 (Balaji) is sending $1000 to Account no(Nivetha) 1002.
2. This transaction should be logged in the **bank_transfer_details** table
3. Commit the transaction and persist the changes

Query:

start transaction;

update bank_account set balance=balance-1000 where account_no=1001

update bank_account set balance=balance+1000 where account_no=1002

insert into bank_transfer_details (transaction_id, sender_no, receiver_no, transfer_amt) values (100001, 1001,1002,1000);

commit;

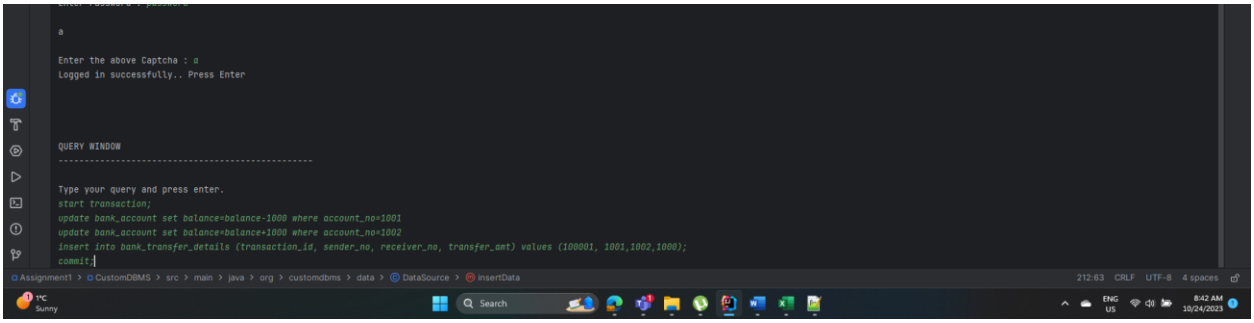


Figure 34: executing the transaction query.

When the flow executes till the first update (Money deducted from Balaji's account). Before transferring the amount to Nivetha's account. The buffer contains the following.

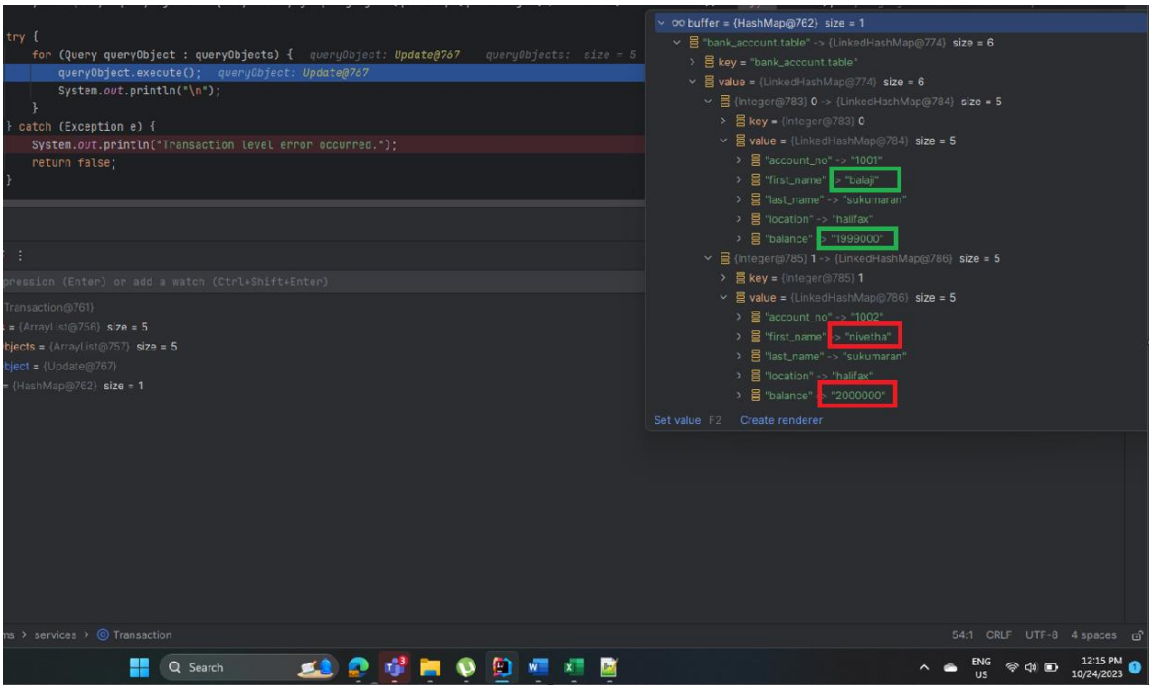


Figure 35: Buffer after the first update.

Now, when the flow goes executes the second update (Money transferred to Nivetha). Buffer will be in following state.

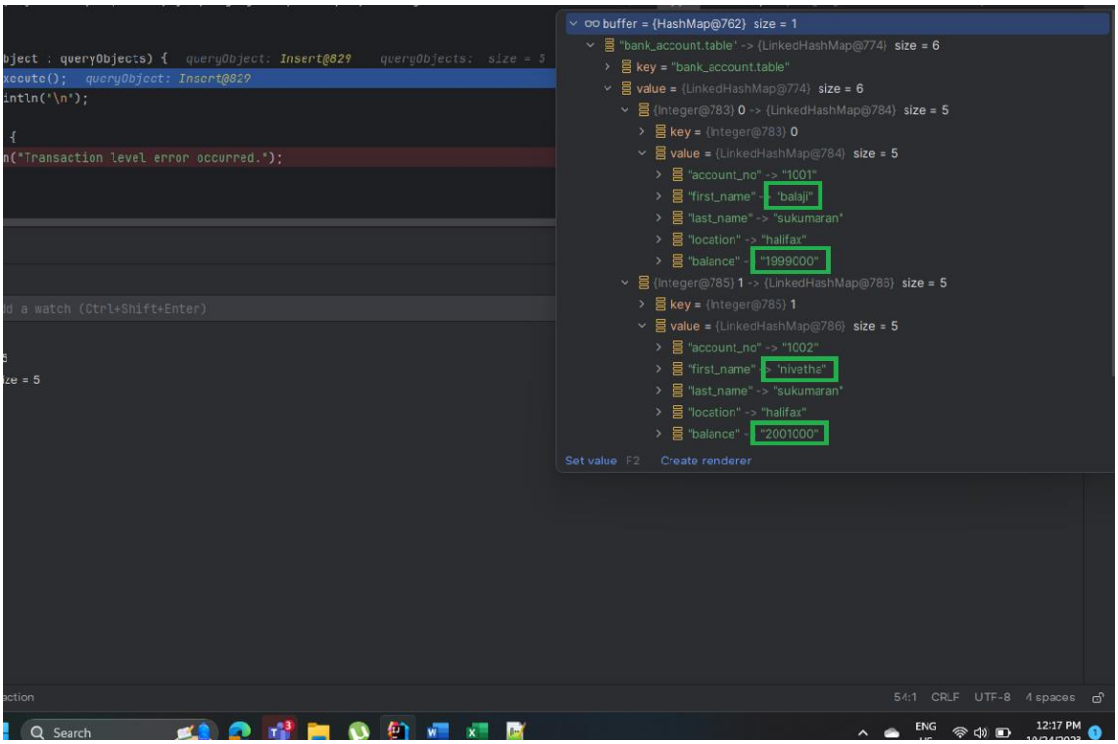


Figure 36: Buffer after the second update

Now, after logging this transaction, it the buffer will have two table states, bank_transfer_details and bank_account

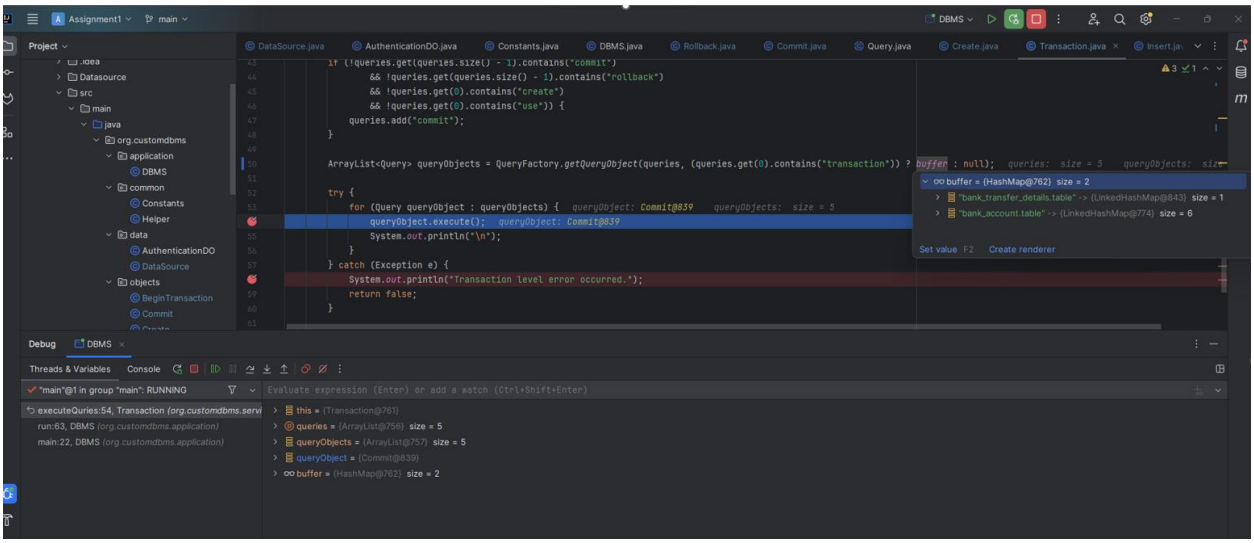


Figure 37: buffer with two table changes

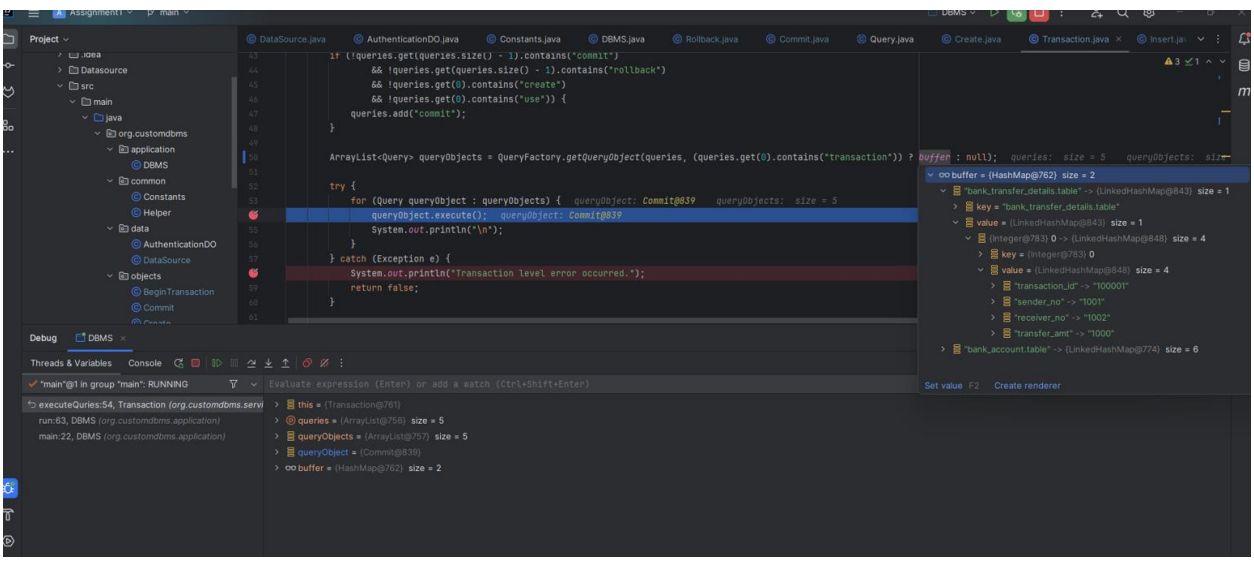


Figure 38: bank_transfer_details in buffer contains the transfer detail

Since, the transaction is not committed yet. The table files are not updated yet.

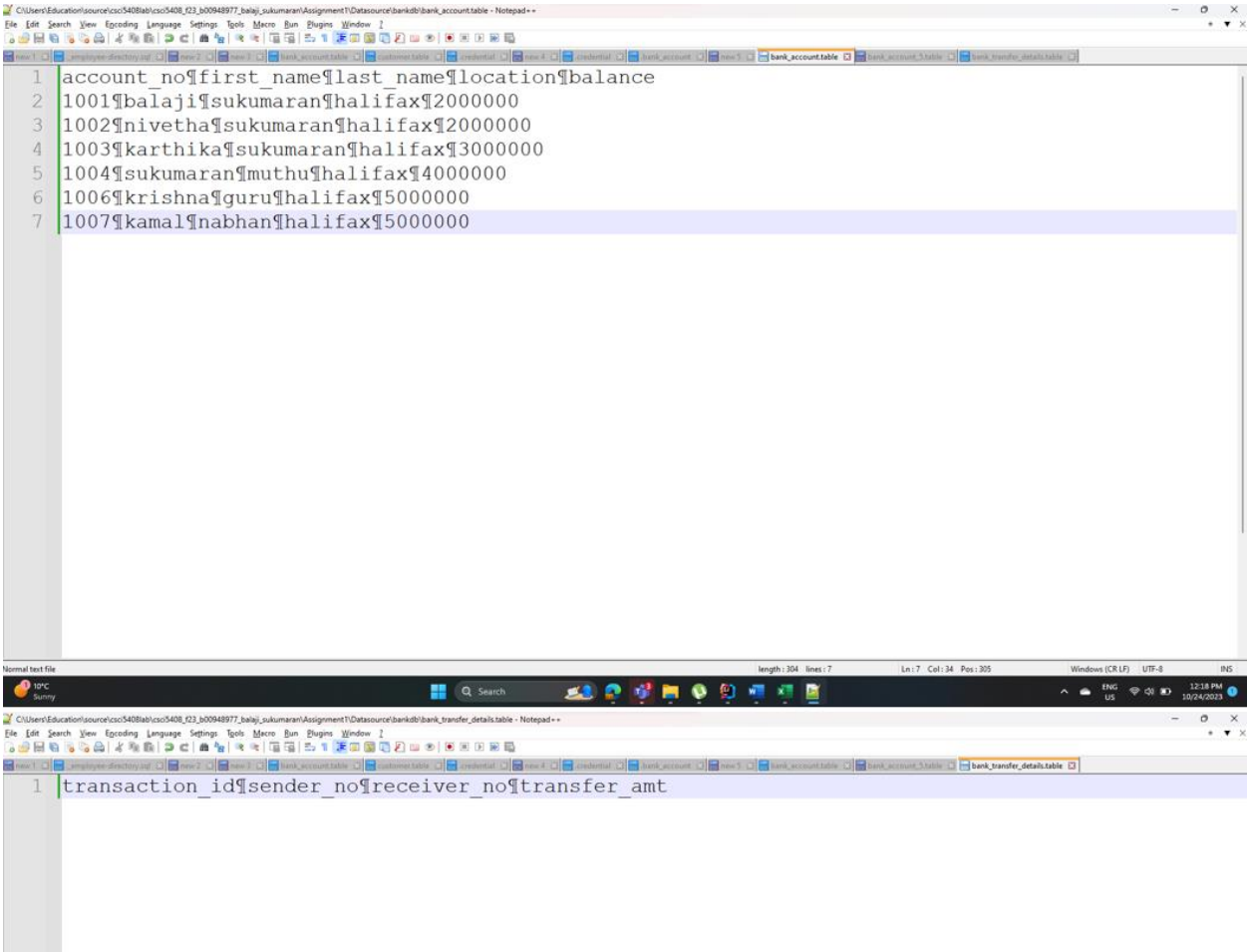
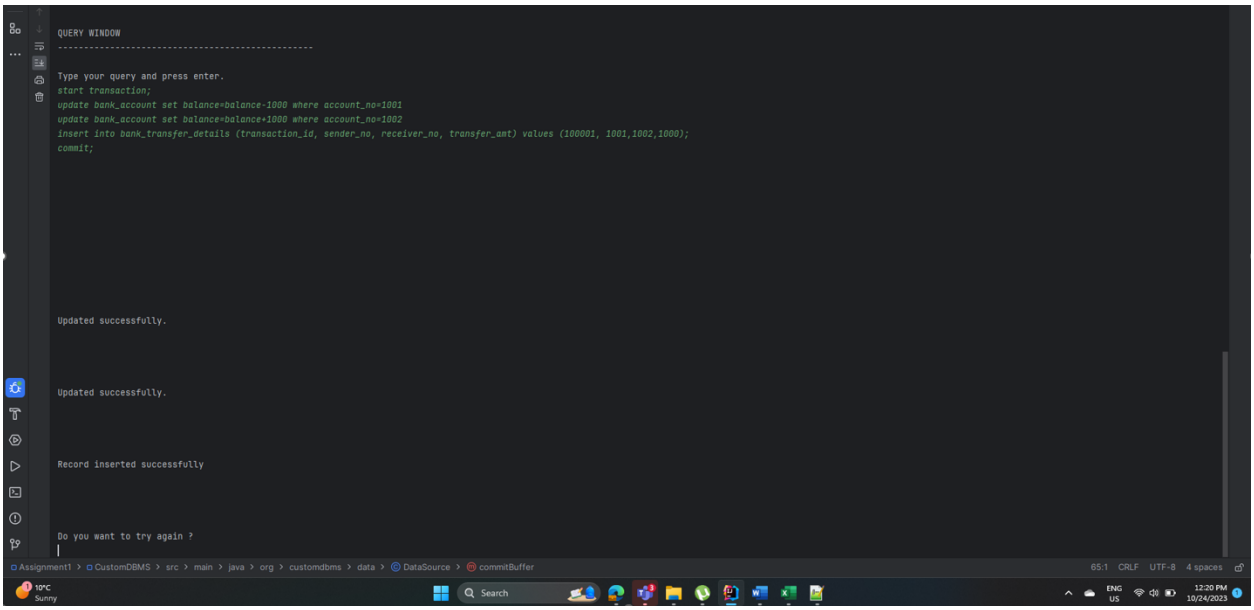


Figure 39: table file before committing.

Once the commit query, gets executed successfully. The control flow will try to persist the buffered changes once successful the table files get updated with buffered data.



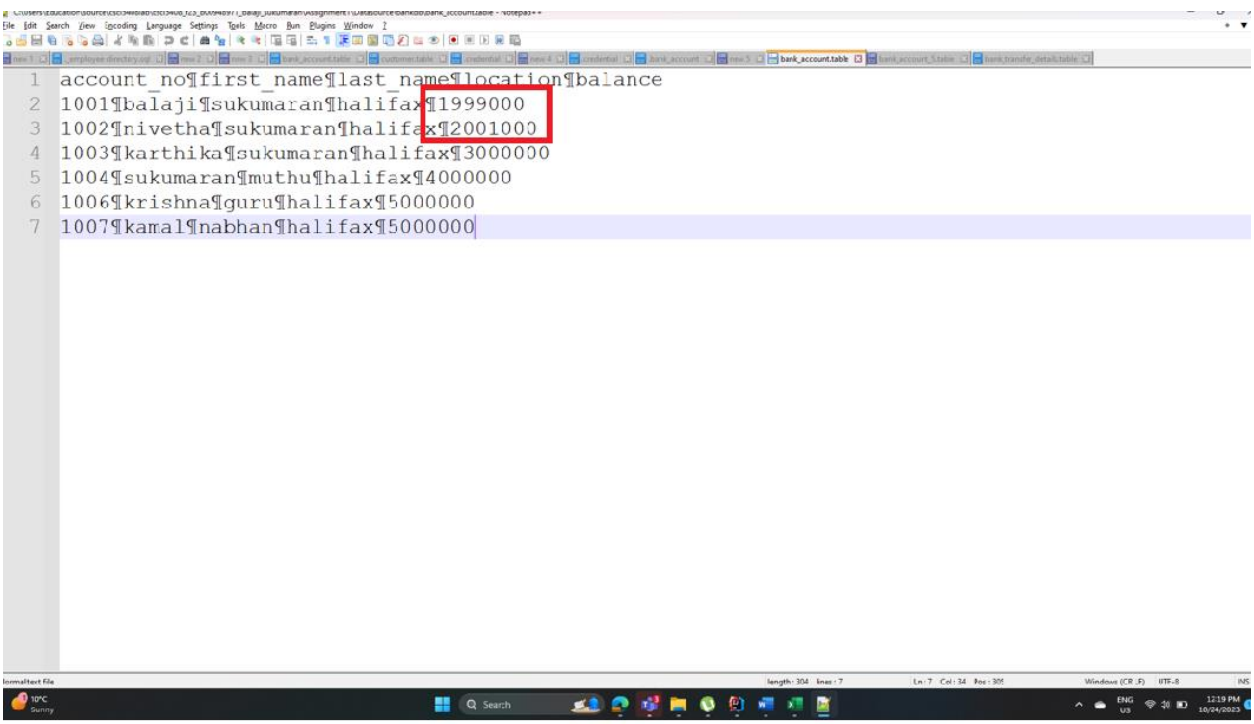


Figure 40: Final state of the table file

Rollback flow and its intermediate state of the buffer.

Executing the same query but with a rollback condition.

```
QUERY WINDOW
-----------------------------------------------

Type your query and press enter.
start transaction;
update bank_account set balance=balance-1000 where account_no=1001
update bank_account set balance=balance+1000 where account_no=1002
insert into bank_transfer_details (transaction_id, sender_no, receiver_no, transfer_amt) values (100001, 1001,1002,1000);
rollback;
```

Figure 41: Transaction with rollback

After executing the first update, money deducted from Balaji's account. Changes are stored in the buffer.

```
queryObjects:  size = 5

buffer = {HashMap@749} size = 1
  "bank_account.table" -> {LinkedHashMap@774} size = 6
    key = "bank_account.table"
    value = {LinkedHashMap@774} size = 6
      {Integer@785} 0 -> {LinkedHashMap@786} size = 5
        key = {Integer@785} 0
        value = {LinkedHashMap@786} size = 5
          "account_no" -> "1001"
          "first_name" -> "balaji"
          "last_name" -> "sukumaran"
          "location" -> "halifax"
          "balance" -> "1998000"
      {Integer@787} 1 -> {LinkedHashMap@788} size = 5
        key = {Integer@787} 1
        value = {LinkedHashMap@788} size = 5
          "account_no" -> "1002"
          "first_name" -> "nivetha"
          "last_name" -> "sukumaran"
          "location" -> "halifax"
          "balance" -> "2001000"
Set value  F2   Create renderer
```

Figure 42: Buffer after first update

After executing the second update, Money credited to Nivetha's account. Changes are logged in buffer accordingly

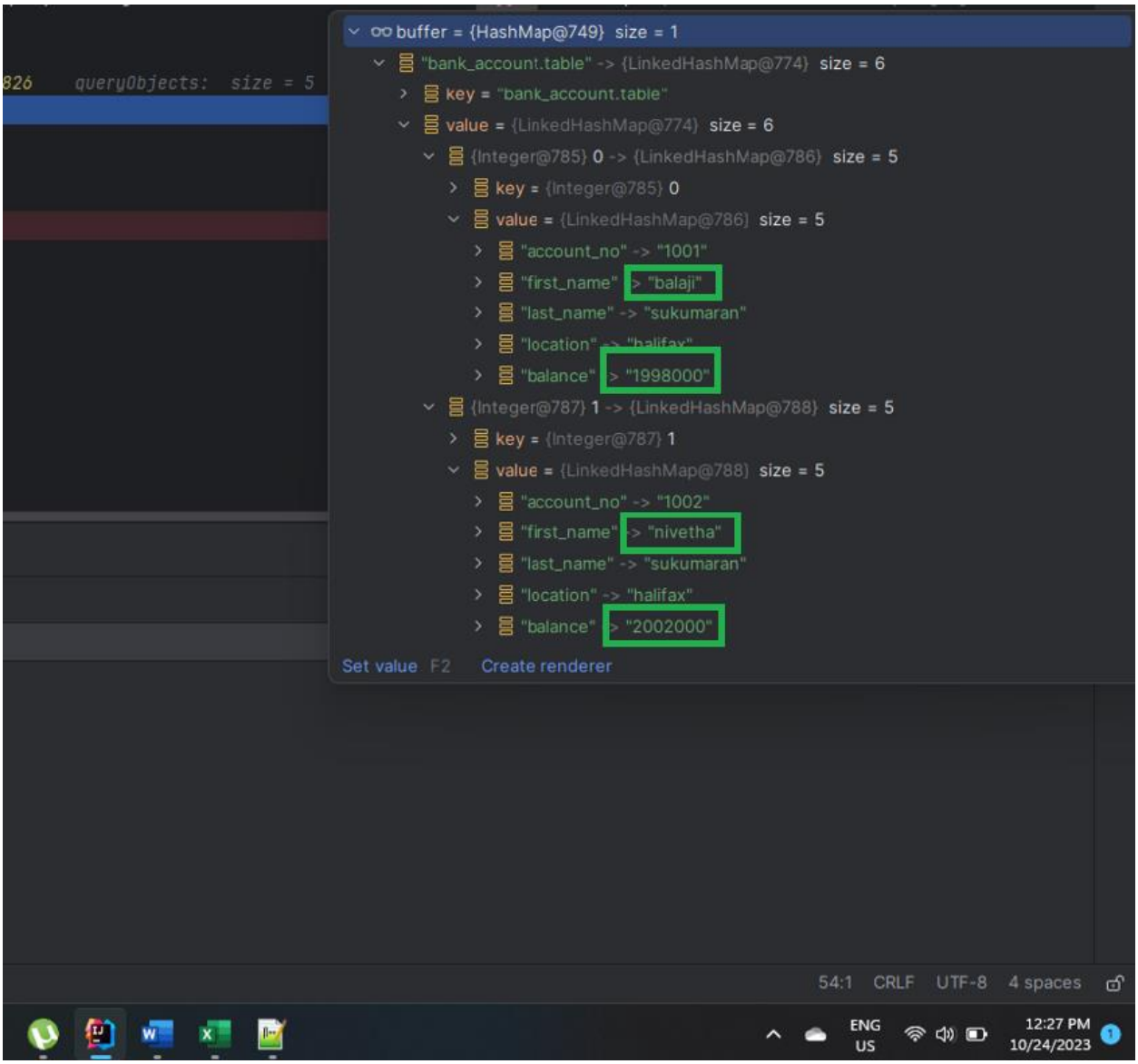


Figure 43: Buffer after second update

The flow will log the money transfer in the bank_account_transfer table and the buffer will hold two tables now and the new buffered bank_account_transfer table has the log.

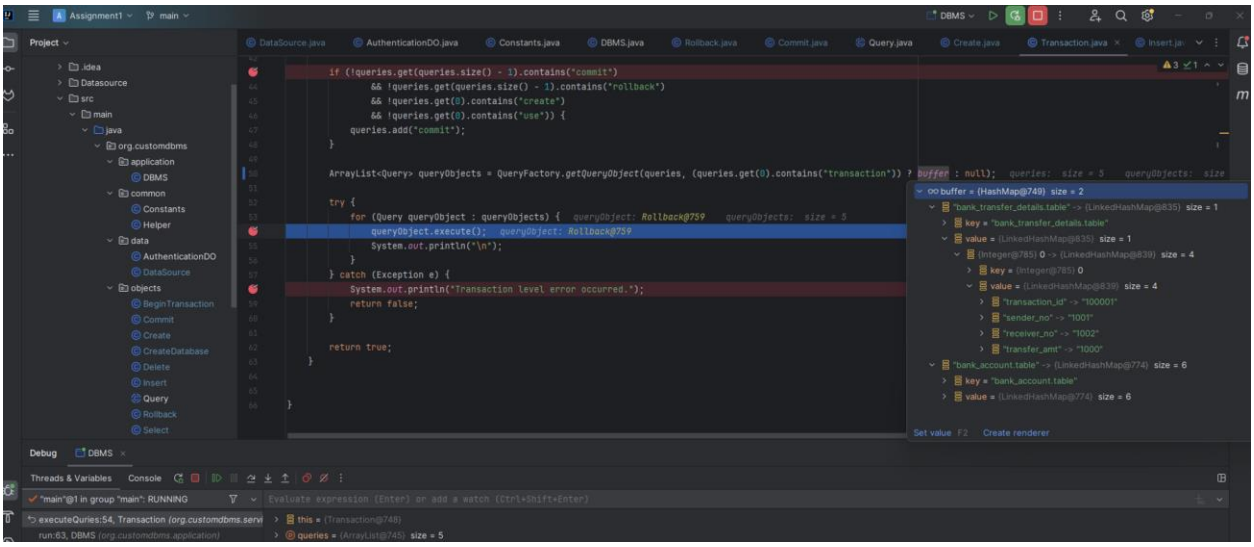


Figure 44: Buffer after logging the transaction.

Now, since the given query has rollback. The changes won't be reflected in the table file. The buffered records get cleared.

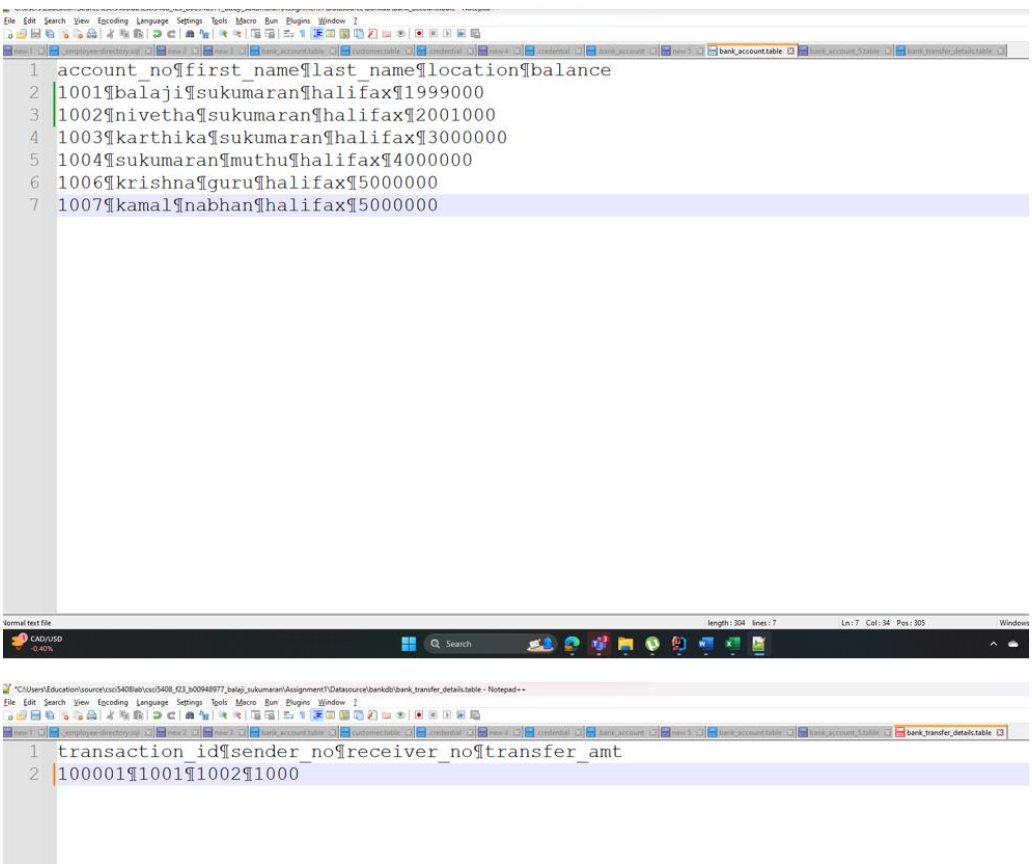


Figure 45: Final table file with old data