

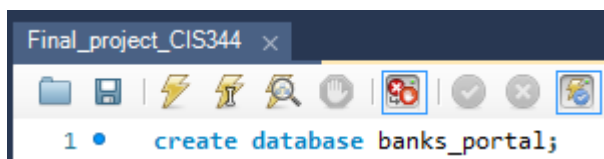
Dulce Zepeda

CIS344 – Final project

5/23/2023

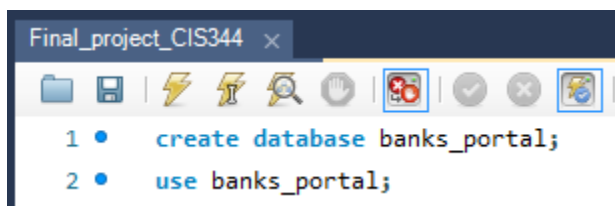
The purpose of this project is to complete the bank teller online platform which will allow them to conveniently maintain bank accounts and conduct transactions. The initial code provided is in Python and runs on a Python HTTP server. The primary objective is to establish a connection with a MySQL server and complete the database class to ensure seamless functionality of the platform.

- 1- The first task is that we have been asked to create a database named banks-portal.



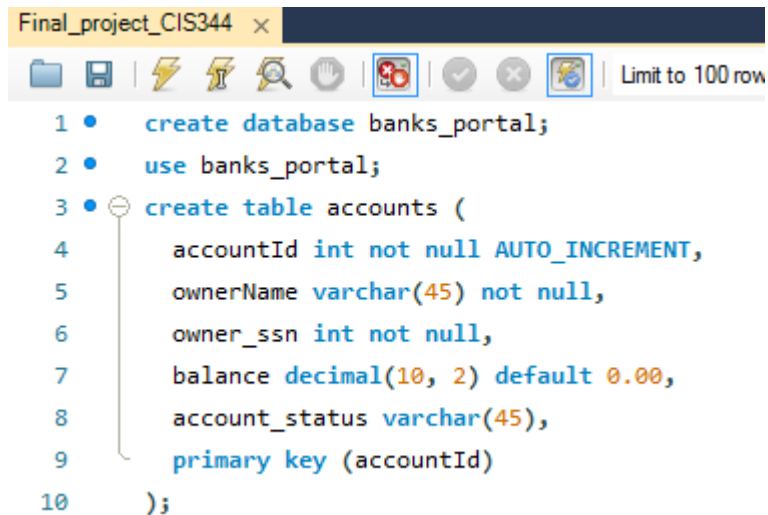
This command builds a brand-new database called "**banks\_portal**."

- 2- The second task is to use the previously created database



By changing the current database context to "**banks\_portal**", this query enables you to do additional SQL actions on that database.

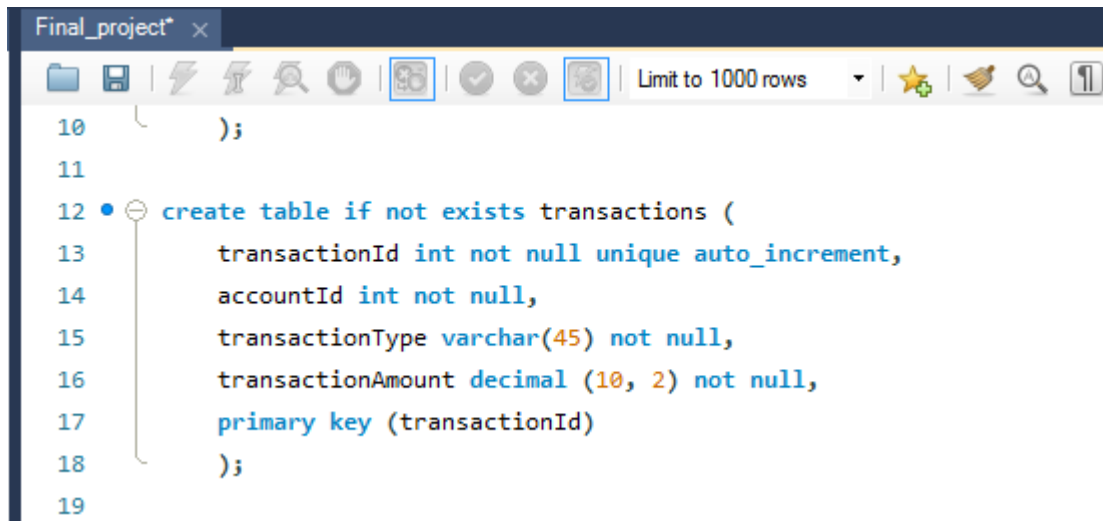
- 3- On the 3<sup>rd</sup> task, we were asked to create a table named accounts, with the following attributes and make **accountId** the primary key.



```
1 • create database banks_portal;
2 • use banks_portal;
3 • create table accounts (
4     accountId int not null AUTO_INCREMENT,
5     ownerName varchar(45) not null,
6     owner_ssn int not null,
7     balance decimal(10, 2) default 0.00,
8     account_status varchar(45),
9     primary key (accountId)
10 );
```

The following fields are added to the “accounts” database by this query: **accountId**, **ownerName**, **owner\_ssn**, **balance**, and **account\_status**. Each account is guaranteed to be unique because the **accountId** column is designated as the main key. For each new row inserted, the AUTO\_INCREMENT attribute automatically creates a distinct value for **accountId**. The balance column is configured to default to 0.00, therefore if no value is entered during insertion, it will be set to that number. The **account\_status** column does not have a default value and, unless otherwise stated, accepts NULL entries by default.

- 4- On the 4<sup>th</sup> task, we are asked to create a table named “Transactions” if it does not exist, with the following attributes and with a primary key as “TransactionId”



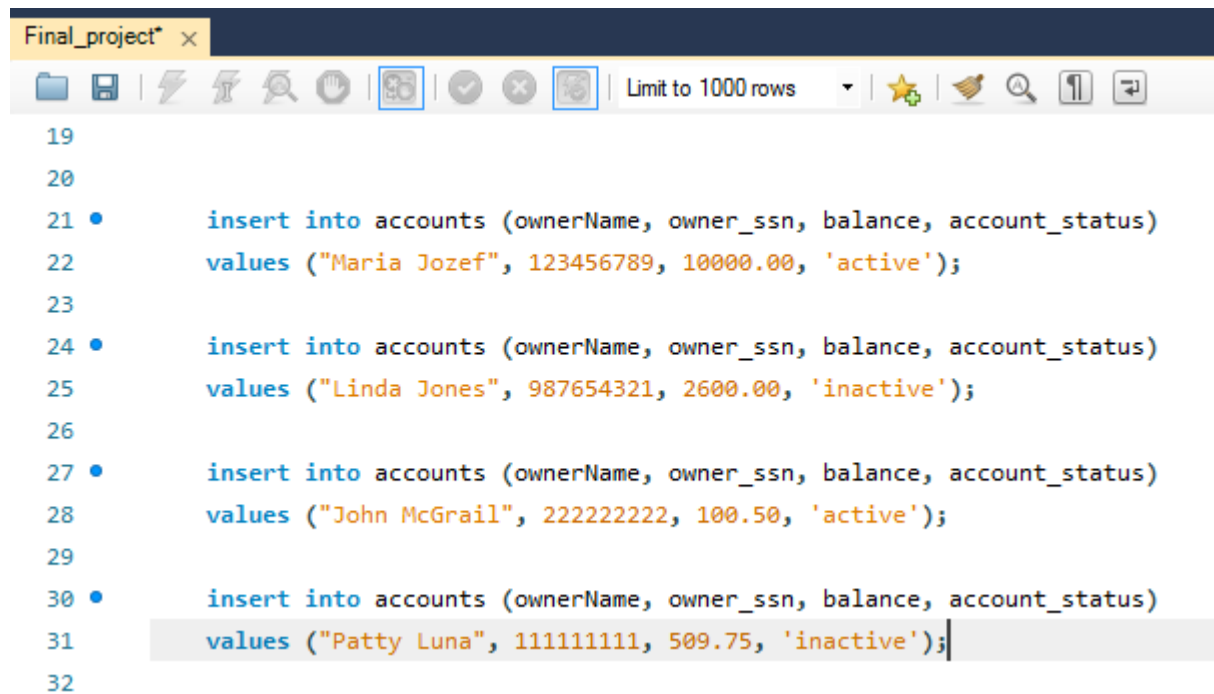
The screenshot shows a SQL IDE window titled "Final\_project". The toolbar includes icons for file operations, execution, and a "Limit to 1000 rows" dropdown. The SQL query is as follows:

```
10      );  
11  
12  ● ○ create table if not exists transactions (  
13      transactionId int not null unique auto_increment,  
14      accountId int not null,  
15      transactionType varchar(45) not null,  
16      transactionAmount decimal (10, 2) not null,  
17      primary key (transactionId)  
18  );  
19
```

The “Transactions” table is created by this query and contains the columns **transactionId**, **accountId**, **transactionType** and **transactionAmount**. As the primary key, the **transactionId** ensures that each transaction is distinct. Each time a new row is inserted, the AUTO\_INCREMENT attribute automatically creates a **transactionId** value that is distinct. The other columns, **accountId**, **transactionType**, and **transactionAmount**, are declared as NOT NULL, which requires that non-null values be present in each row for each of these fields.

5- The 5<sup>th</sup> task is to populate the table accounts with the following values.

("Maria Jozef", 123456789, 10000.00, "active"), ("Linda Jones", 987654321, 2600.00, "inactive"), ("John McGrail", 222222222, 100.50, "active"), ("Patty Luna", 111111111, 509.75, "inactive")



```
19
20
21 •      insert into accounts (ownerName, owner_ssn, balance, account_status)
22      values ("Maria Jozef", 123456789, 10000.00, 'active');
23
24 •      insert into accounts (ownerName, owner_ssn, balance, account_status)
25      values ("Linda Jones", 987654321, 2600.00, 'inactive');
26
27 •      insert into accounts (ownerName, owner_ssn, balance, account_status)
28      values ("John McGrail", 222222222, 100.50, 'active');
29
30 •      insert into accounts (ownerName, owner_ssn, balance, account_status)
31      values ("Patty Luna", 111111111, 509.75, 'inactive');
32
```

The “accounts” table is expanded with new rows using the INSERT INTO statement in this query. The **ownerName**, **owner\_ssn**, **balance**, and **account\_status** values are provided in the order of the appropriate table columns for each row specified within the VALUES clause.

- 6- The 6<sup>th</sup> task is to populate the table transactions with the provided values.

```
33 •      insert into Transactions (accountId, transactionType, transactionAmount)
34      values (1, 'deposit', 650.98);
35
36 •      insert into Transactions (accountId, transactionType, transactionAmount)
37      values (3, 'withdraw', 899.87);
38
39 •      insert into Transactions (accountId, transactionType, transactionAmount)
40      values (3, 'deposit', 350.00);
41
```

This query inserts rows of data into the “Transactions” table using the INSERT INTO statement. The accountId, transactionType, and transactionAmount values for each row provided in the VALUES clause in the order that they appear in the appropriate table columns.

- 7- The 7<sup>th</sup> task is to Create a stored Procedure named accountTransactions(accountID)

```
42      delimiter //
43 •      create procedure accountTransactions (in accountId int)
44      begin
45          select * from Transactions where accountId = accountId;
46      end //
47      delimiter ;
48
```

The create procedure statement creates a store procedure named accountTransactions which accepts an accountId parameter. A select statement is used within the method to retrieve all transactions from the transactions table that match the supplied accountId.

8- We are asked to create a stored Procedure named deposit(accountID, amount)

```
49  delimiter //
50 • create procedure deposit(in accountId int, in amount decimal(10,2))
51  begin
52      start Transaction;
53      insert into Transactions (accountId, transactionType, transactionAmount)
54      values (accountId, 'deposit', amount);
55      update accounts set balance = balance + amount where accountId = accountId;
56      commit;
57  end //
58  delimiter ;
59
```

Here we are creating a deposit store procedure with the CREATE PROCEDURE statement with the parameters accountId and amount. The procedure initiates a transaction and then adds a new row in the transaction database with the specified accountId, transaction type as 'deposit', and amount. It then adds the deposited money to the balance of the relevant account in the account table. The commit statement is then used to commit the modifications and end the transaction.

- 9- We are asked to create a Create a stored Procedure named withdraw(accountID, amount)

```
60     delimiter //
61 •   create procedure withdraw(in accountId int, in amount decimal(10,2))
62     begin
63     start Transaction;
64     insert into Transactions (accountId, transactionType, transactionAmount)
65     values (accountId, 'withdraw', amount);
66     update accounts set balance = balance - amount where accountId = accountId;
67     commit;
68     end //
69     delimiter ;
```

Here we are creating a withdraw stored procedure with the CREATE PROCEDURE statement which takes as parameters accountId and amount. It initiates a transaction and adds a new row in the transactions database with the provided accountId, transaction type as 'withdraw' and the specified amount, similar to the deposit operation. The related account's balance in the accounts table is then updated subtracting the withdrawn amount. The commit statement is then used to commit the modifications and end the transaction.