**Step 4**

Running the table creation file of a banking service project in the existing database. Creating a DSD diagram from the new tables

****

Inferring the entities and relationships from this DSD diagram and building an ERD accordingly:

Sure! Let's go through the entities (tables) and their relationships in your database schema.

**Entities**

1.Client

- ClientId: Unique identifier for each client.

- ClientName: First name of the client.

- ClientAddress: Address of the client.

- ClientLastName: Last name of the client.

2. Worker

- WorkerId: Unique identifier for each worker.

- WorkerName: Name of the worker.

- WorkerBirthDate: Birth date of the worker.

- WorkerRole: Role or position of the worker in the organization.

3. Branch

- BranchId: Unique identifier for each branch.

- BranchAddress: Address of the branch.

- BranchManager: Manager of the branch.

4. Account

- AccountId: Unique identifier for each account.

- Balance: Balance amount in the account.

- AccountNumber: Number associated with the account.

- BranchId: The branch where the account is held (foreign key from Branch(

5. Actions

- ActionId: Unique identifier for each action.

- ActionName: Name of the action (e.g., deposit, withdrawal(

- ActionSum: Sum of money involved in the action.

- ActionDate: Date when the action took place.

6. Activity

- ActivityId: Unique identifier for each activity.

- ClientId: The client involved in the activity (foreign key from Client(

- ActionId: The action involved in the activity (foreign key from Actions(

- AccountId: The account involved in the activity (foreign key from Account(

- WorkerId: The worker involved in the activity (foreign key from Worker(

7. Loan

- LoanId: Unique identifier for each loan.

- NumOfPaymet: Number of payments for the loan.

- LoanSum: Sum of the loan amount.

8. LoanToAccount

- LoanDate: Date when the loan was issued.

- returnedOrNot: Indicator whether the loan is returned or not.

- LoanId: The loan involved (foreign key from Loan(

- AccountId: The account to which the loan is assigned (foreign key from Account(

**Relationships**

1. Client and Activity

- A client can have multiple activities. This relationship is established by `ClientId` in the `Activity` table.

2. Worker and Activity

- A worker can be involved in multiple activities. This relationship is established by `WorkerId` in the `Activity` table.

3. Branch and Account

- A branch can have multiple accounts. This relationship is established by `BranchId` in the `Account` table.

4. Account and Activity

- An account can have multiple activities. This relationship is established by `AccountId` in the `Activity` table.

5. Actions and Activity

- An action can be associated with multiple activities. This relationship is established by `ActionId` in the `Activity` table.

6. Loan and LoanToAccount

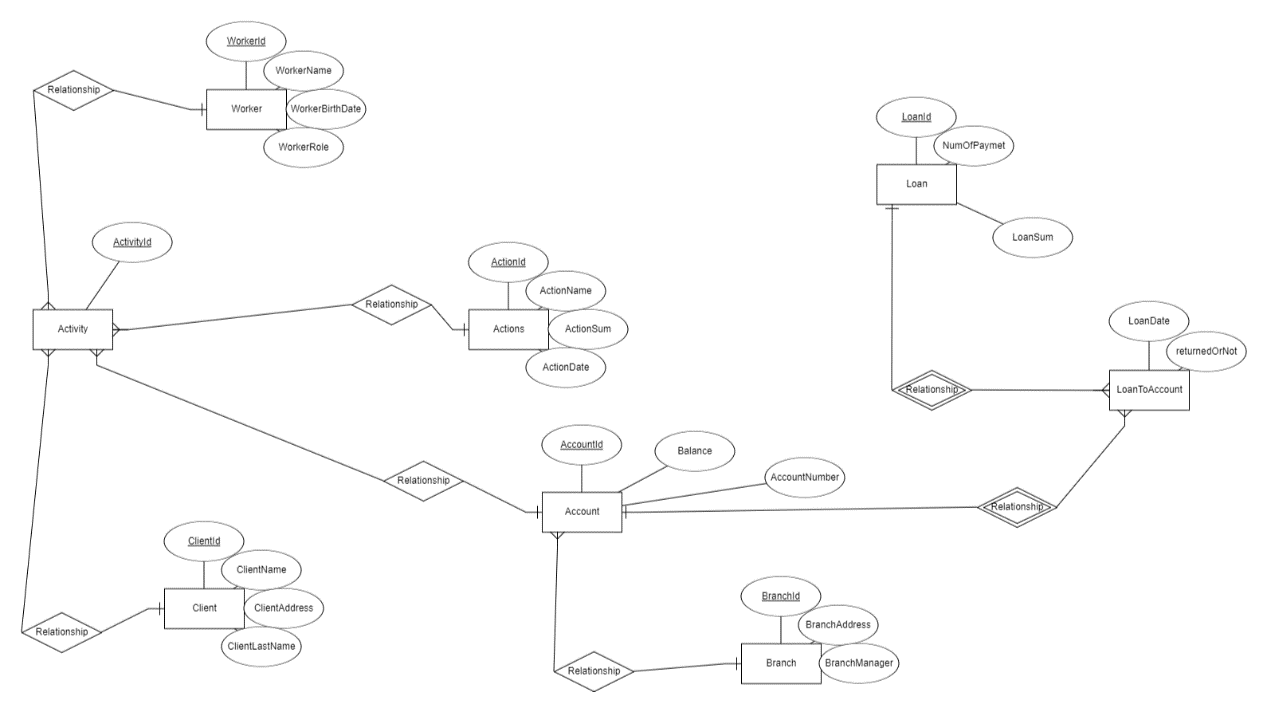
- A loan can be associated with multiple accounts. This relationship is established by `LoanId` in the `LoanToAccount` table.

7. Account and LoanToAccount

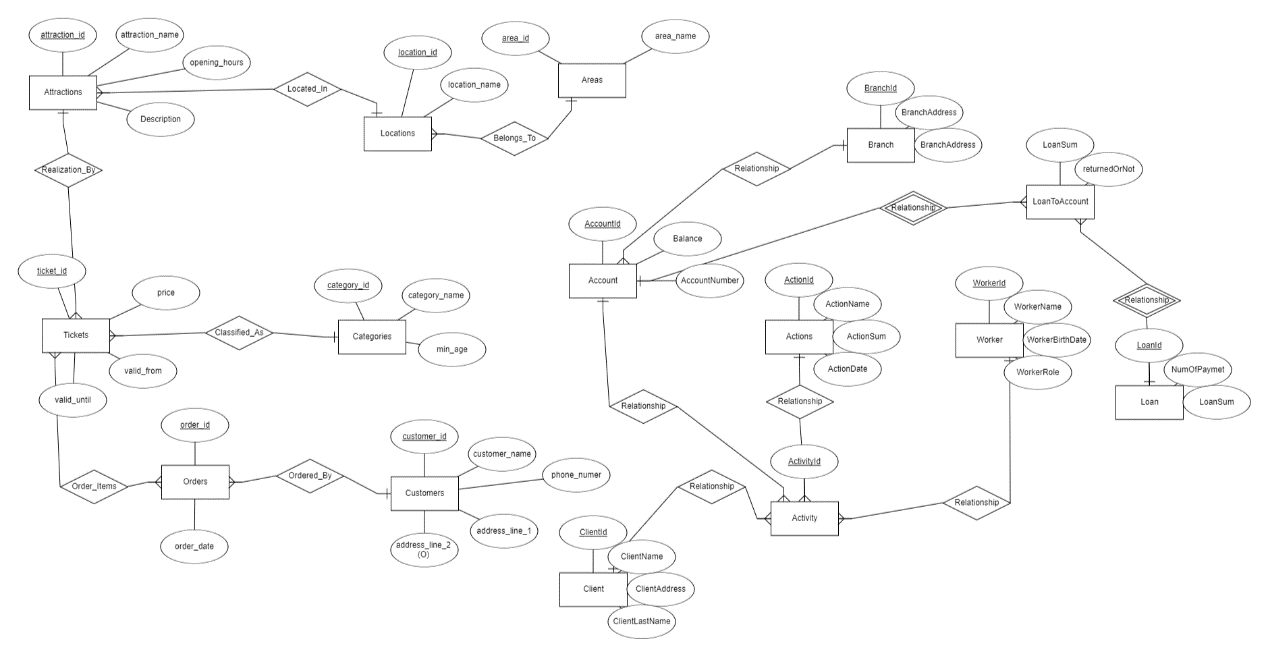
- An account can have multiple loans. This relationship is established by `AccountId` in the `LoanToAccount` table.

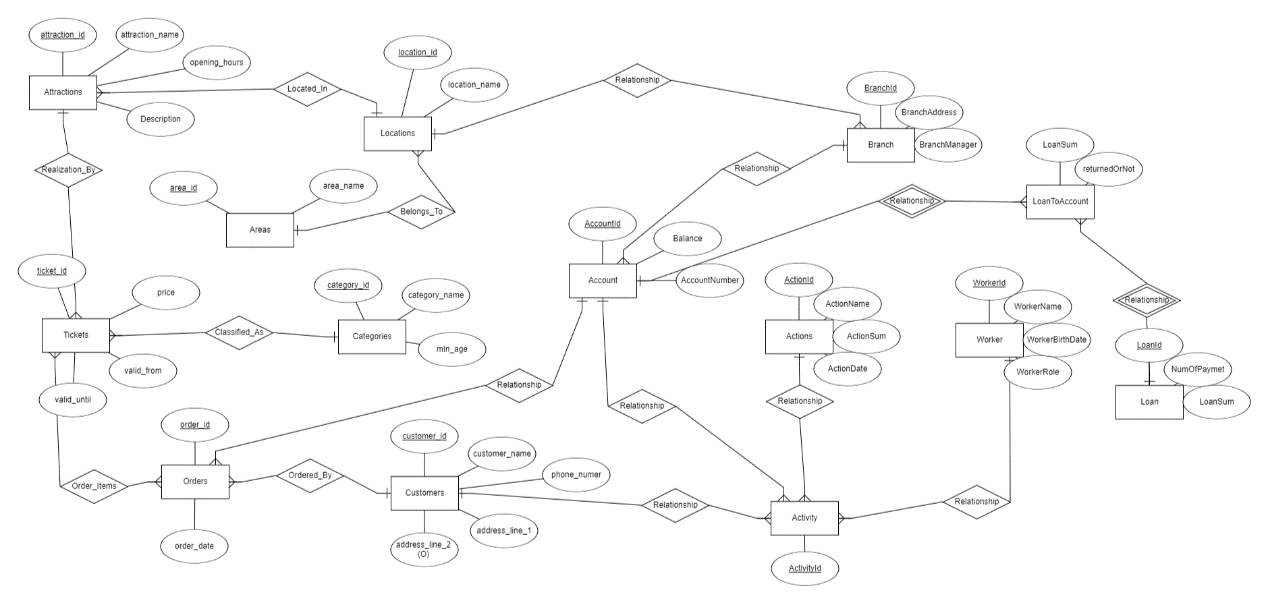
**A combination of account ID and loan ID can only appear once in the table - an account can only take one loan of the same type.**

These entities and relationships create a structure where clients, workers, branches, accounts, actions, activities, and loans are interrelated, allowing comprehensive management and tracking of financial operations within the system.

****

Merging the Tickets Online Store and Banking Service ERDs to one ERD

****

Designing the new ERD by integrating the both projects.****

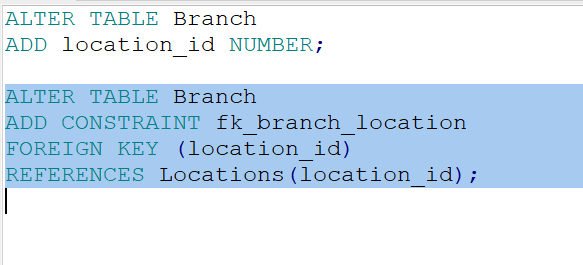
Updating the DB according to the new ERD.

**Conceptual Explanation of Decisions**

1. **Adding Location Information to Branches**
   * The Branch table was updated to include a reference to locations, allowing each branch to be associated with a specific location. This addition helps in managing and organizing branches based on their geographical locations.
2. **Updating Activity Relationships**
   * The activity table was modified to include a reference to the Customers table, reflecting that activities are now directly associated with customers rather than the old Client table. This change ensures that all customer-related activities are tracked accurately.
3. **Eliminating Redundancies**
   * The Client table was completely removed, and its relevant data was merged into the Customers table. This consolidation reduces redundancy and simplifies the database schema by having a single table to manage all client-related information.
4. **Linking Orders to Accounts**
   * The Orders table was updated to include a reference to accounts. This link ensures that orders are associated with the correct accounts, enhancing the tracking of financial transactions and order management.

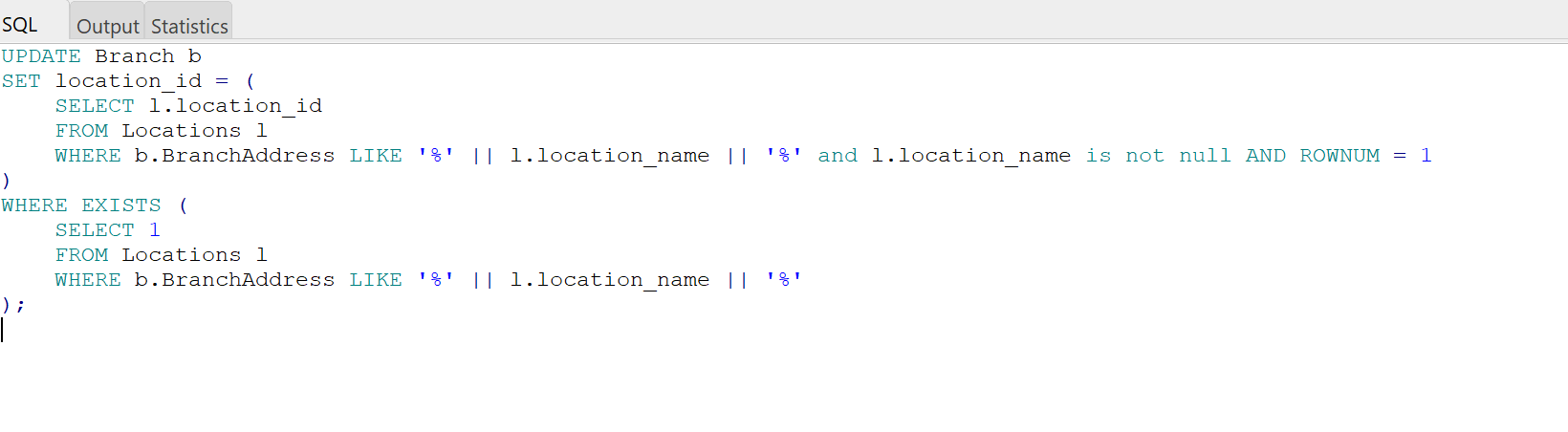
**Adding Foreign Key to Branch**

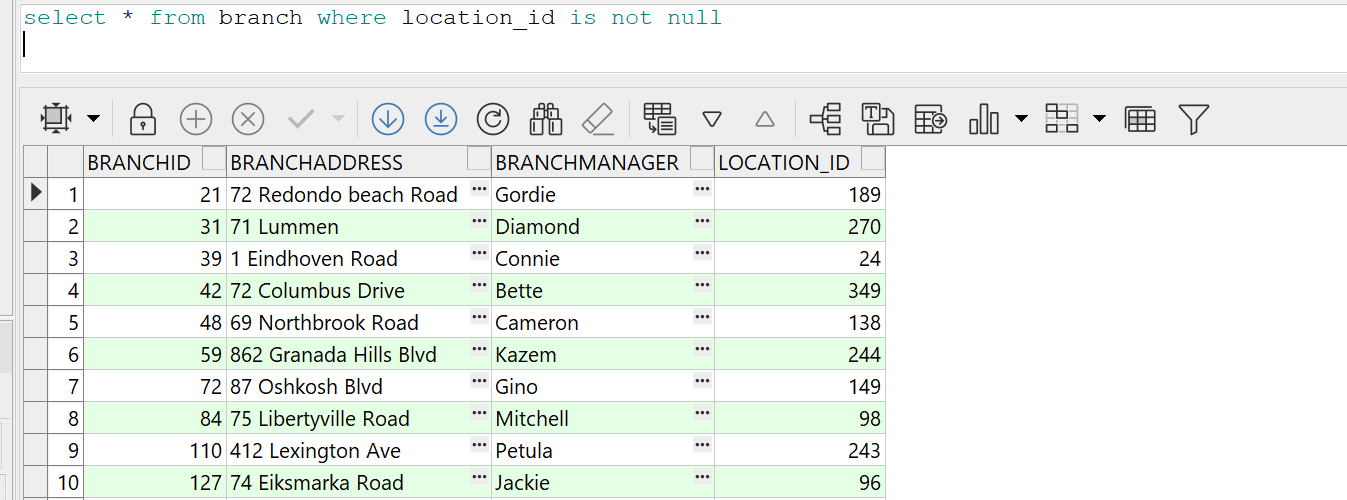
This adds a new column location\_id to the Branch table to store references to locations and sets location\_id as a foreign key, linking the Branch table to the Locations table. This change allows branches to be associated with specific locations.



**Setting location\_id in Branch Table**

This update assigns location\_id to branches based on a match between BranchAddress and location\_name from the Locations table. It uses a subquery to find a matching location name within the branch address and assigns the corresponding location\_id.

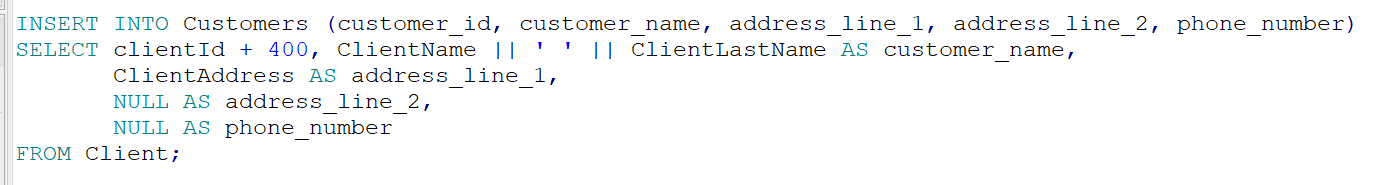
**Verfiy the the data has been updated.**



### Data Migration from Client to Customers

**Inserting Data into Customers**

This insertion migrates data from the old Client table to the new Customers table. It creates new customer records by combining the first and last names of clients, and adjusting their IDs by adding 400 to ensure uniqueness. The client's address is also migrated, while the second address line and phone number are set to NULL.



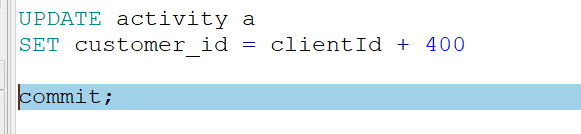
**Modifying the Activity Table to Reflect Customer Relationship.**

This adds a new customer\_id column to the activity table to store references to customers and sets customer\_id as a foreign key in the activity table, linking activities to specific customers.

### Updating Activity Table

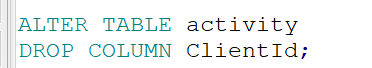
**Updating customer\_id in Activity**

This update adjusts the activity table by setting the customer\_id to the old ClientId incremented by 400. This ensures that activities are now linked to the new Customers entries.



**Removing Client Relationship from Activity**

This removes the ClientId column from the activity table, eliminating the previous relationship with the Client table.

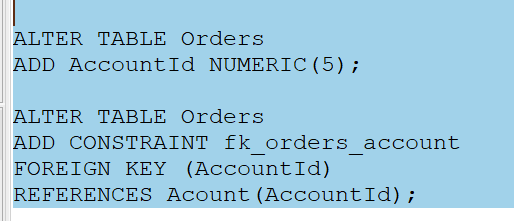


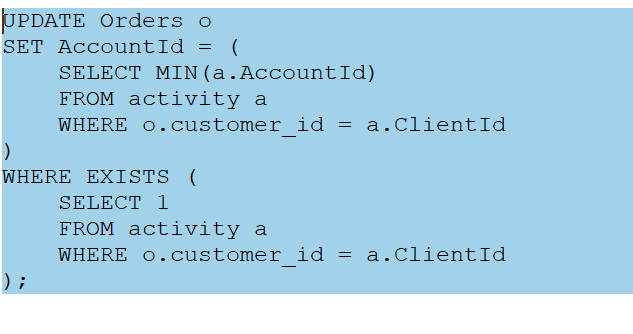
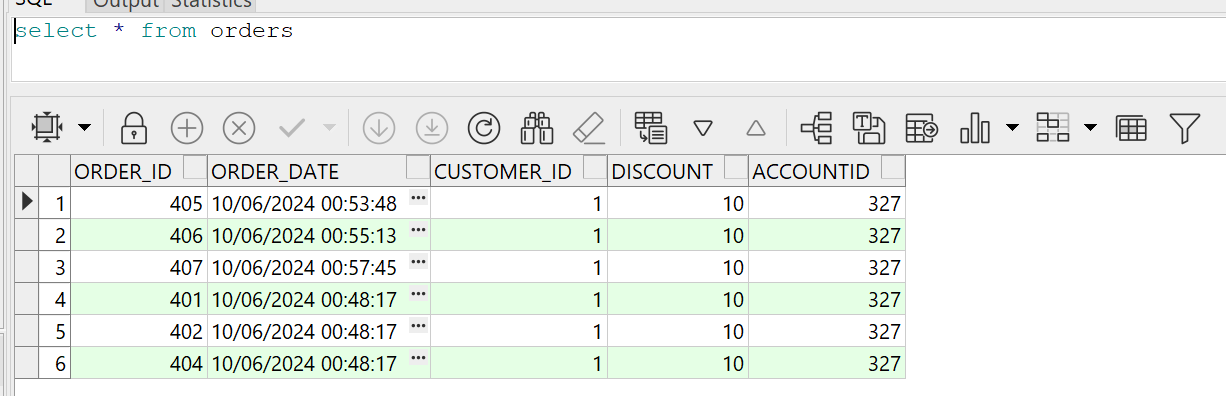
This deletes the Client table entirely, reflecting a decision to consolidate or replace the Client table with the Customers table.



**Modifying Orders to Include Account Relationship**

This adds an AccountId column to the Orders table to store references to accounts and sets AccountId as a foreign key in the Orders table, linking orders to specific accounts.



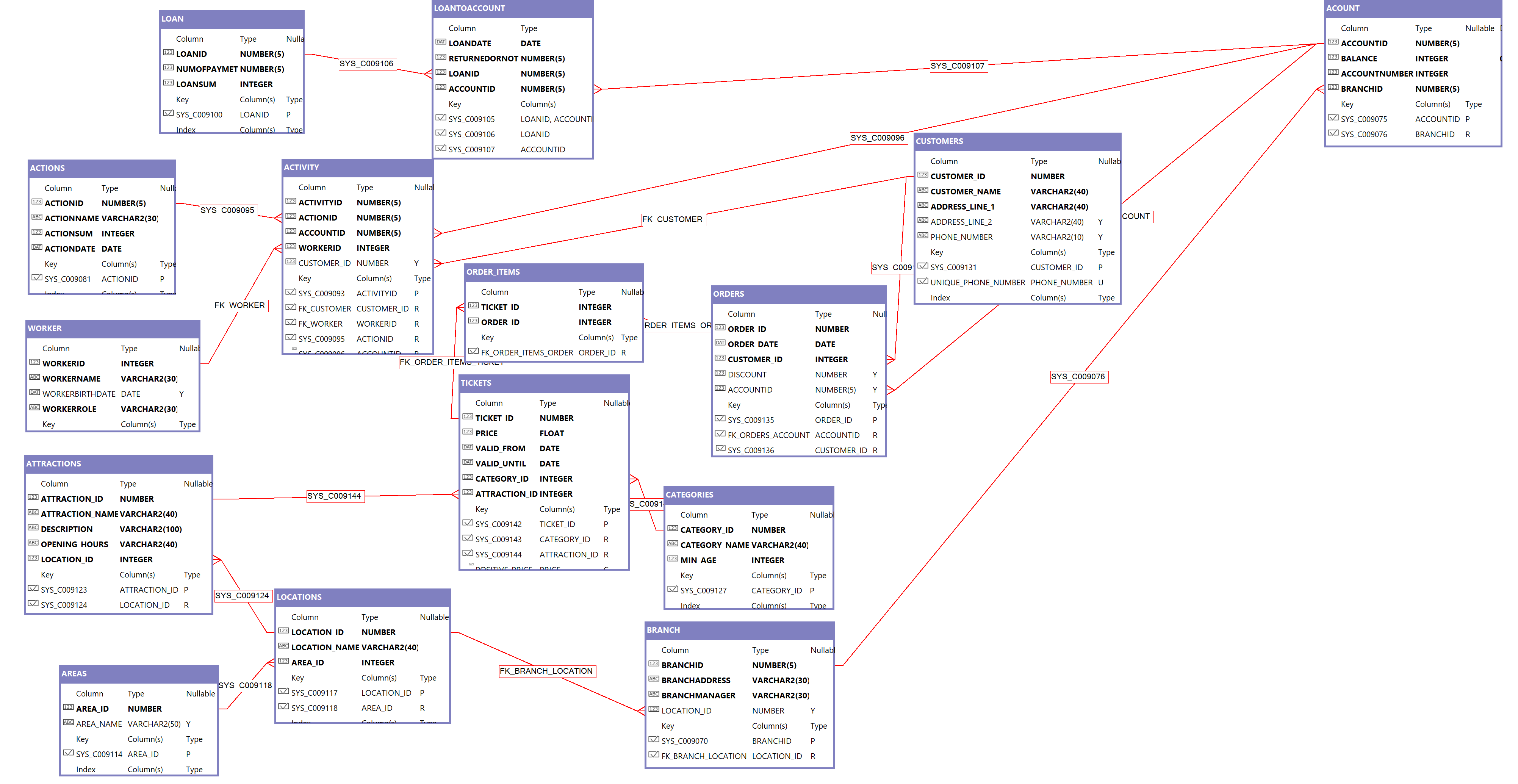
 

**Before creating the new DSD I have noticed that some of the relationships have been removed.** Added foreign key constraints to enforce relationships between tables, ensuring data integrity and consistency across the database.

The following foreign key constraints:

* Ensure that workerId in the activity table references valid entries in the worker table.
* Ensure that order\_id in the order\_items table references valid entries in the orders table.
* **Ensure** that ticket\_id in the order\_items table references valid entries in the tickets table.

The final DSD:



### Creating a View: actions\_in\_branches

The view actions\_in\_branches consolidates data from several tables to provide a comprehensive view of actions occurring in various branches, including details about the action, the account, and the customer involved.

**Components of the View:**

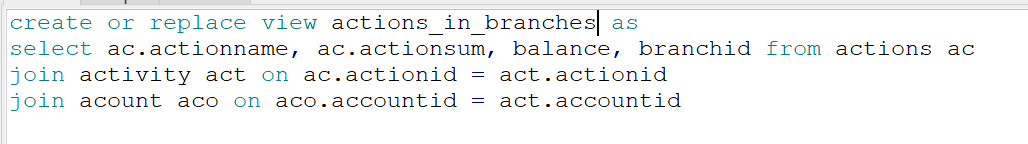
* **Action Details**: actionname and actionsum from the actions table.
* **Account Balance**: balance from the acount table.
* **Branch ID**: branchid from the acount table.
* **Account ID**: accountId from the acount table.
* **Customer Details**: customer\_name and customer\_id from the customers table.

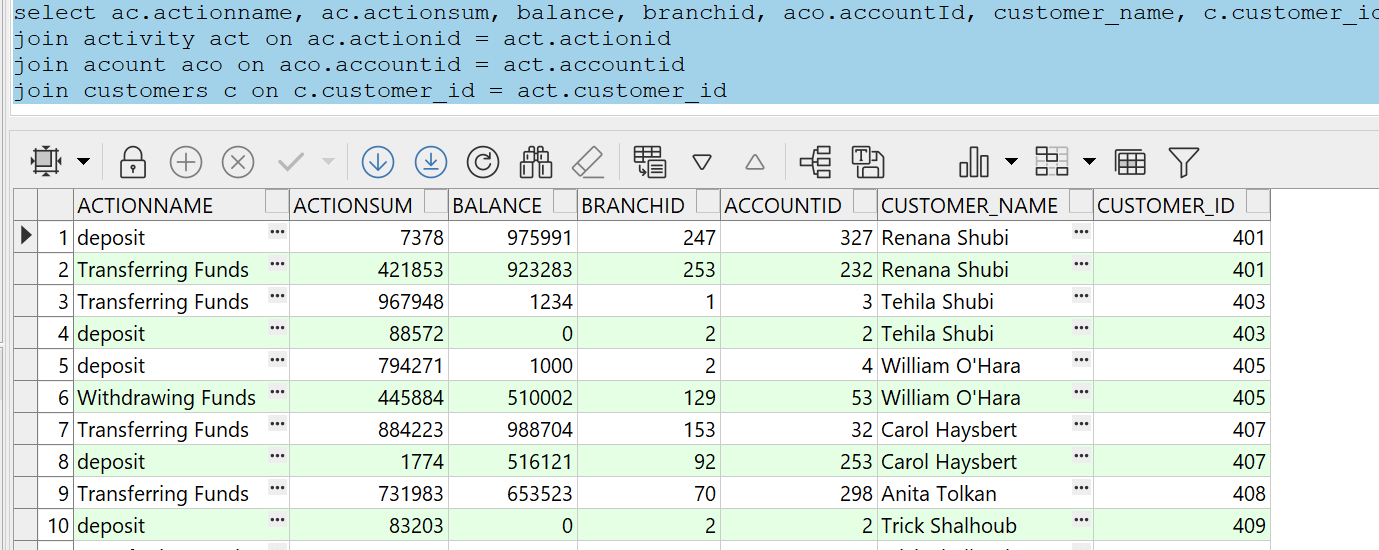
**Joins Used:**

* **Joining actions and activity**: To link each action to an activity.
* **Joining activity and acount**: To link each activity to an account.
* **Joining activity and customers**: To link each activity to a customer.

### Purpose of the View:

* To provide a comprehensive and easy-to-query dataset that includes all relevant details about actions, accounts, branches, and customers.





### Query 1: Sum of All Actions Grouped by Branch and Location

This query calculates the total sum of actions for each branch and location.

**Components of the Query:**

* **Location Name**: location\_name from the Locations table.
* **Branch ID**: BranchId from the Branch table.
* **Total Action Sum**: The sum of ActionSum for each combination of branch and location.

**Joins Used:**

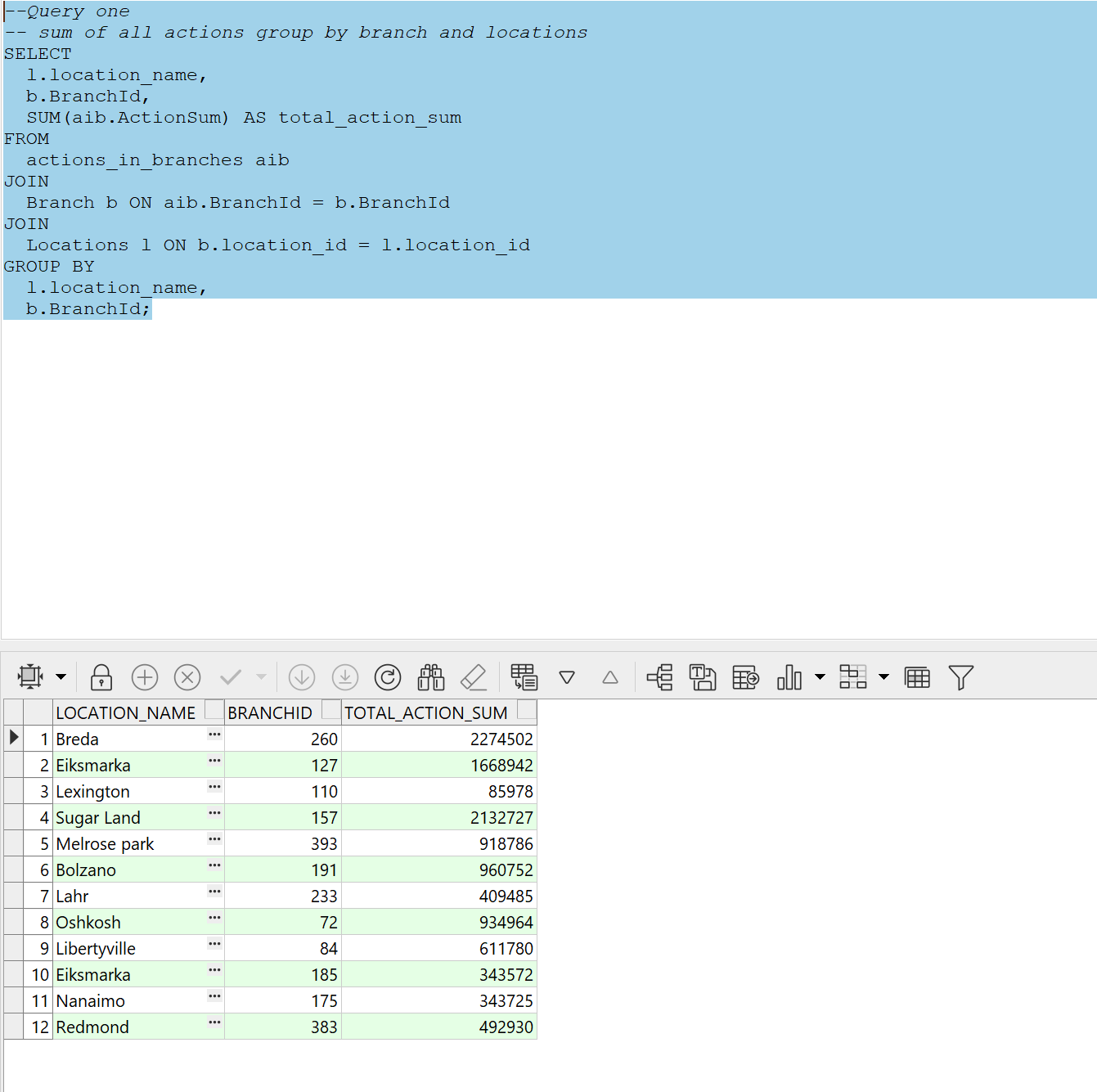
* **Joining actions\_in\_branches and Branch**: To associate each action with its branch.
* **Joining Branch and Locations**: To associate each branch with its location.

**Grouping**:

* By location\_name and BranchId, to ensure that the sum is calculated for each unique combination of branch and location.

### Purpose of the Query:

* To provide insights into the financial activity (total action sums) at each branch within different locations.



### Query 2: Count of Unique Customers Grouped by Branch and Location

This query counts the number of unique customers involved in actions for each branch and location.

**Components of the Query:**

* **Location Name**: location\_name from the Locations table.
* **Branch ID**: BranchId from the Branch table.
* **Customer Count**: The count of distinct customer\_id values for each combination of branch and location.

**Joins Used:**

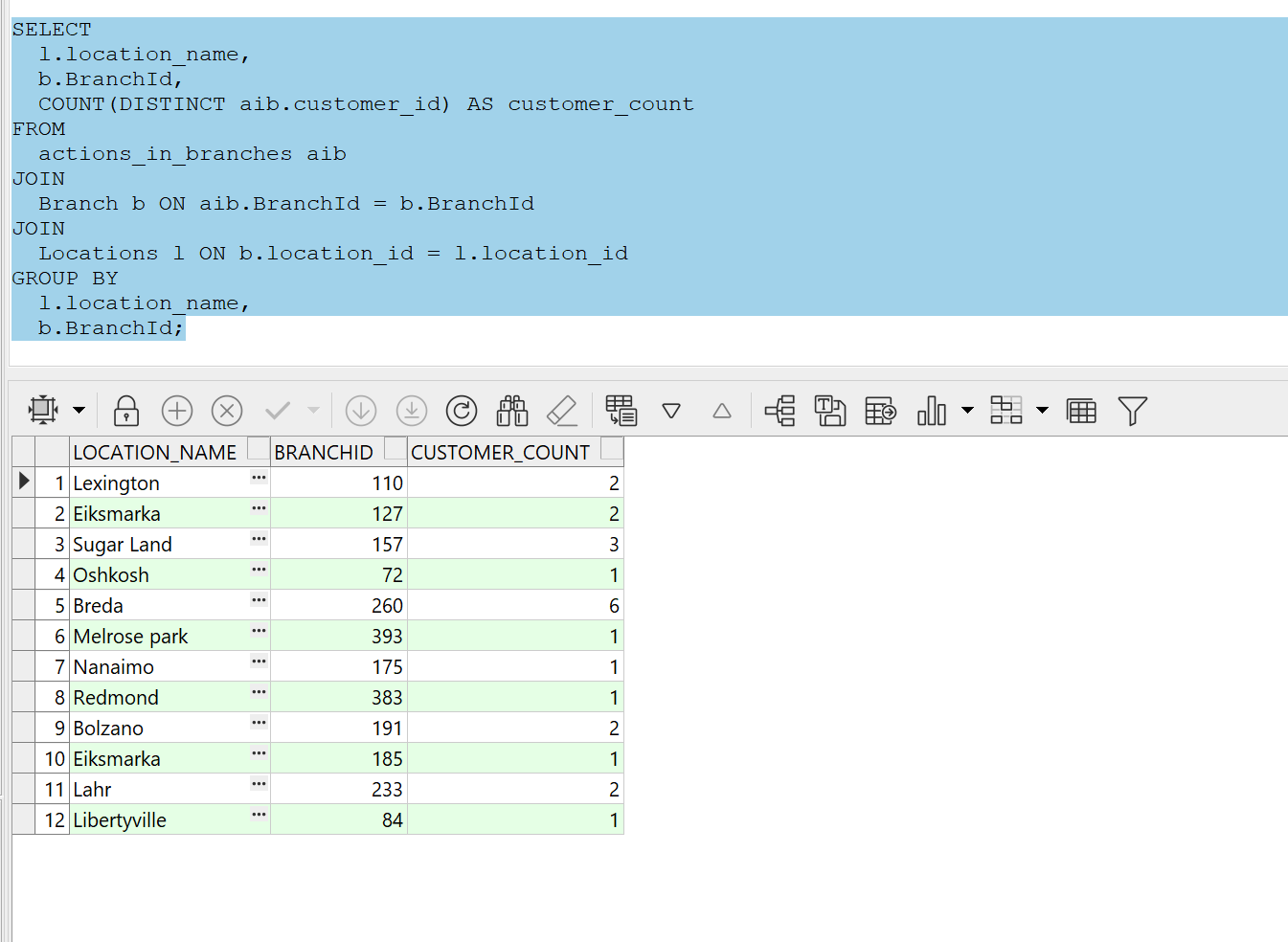
* **Joining actions\_in\_branches and Branch**: To associate each action with its branch.
* **Joining Branch and Locations**: To associate each branch with its location.

**Grouping**:

* By location\_name and BranchId, to ensure that the count is calculated for each unique combination of branch and location.

### Purpose of the Query:

* To understand customer engagement at each branch within different locations by counting the number of unique customers involved in actions.



### Overall Purpose:

The view and the two queries together provide valuable insights into both the financial activities and customer engagement across branches and locations, helping in making informed decisions about branch performance and customer outreach strategies.

**Important note!!!**

Because I received permission to carry out the project alone, I was told by Shulamit that I could only complete half of the requirements - in this case I created one view and two queries.

**Have a fun reviewing ☺**