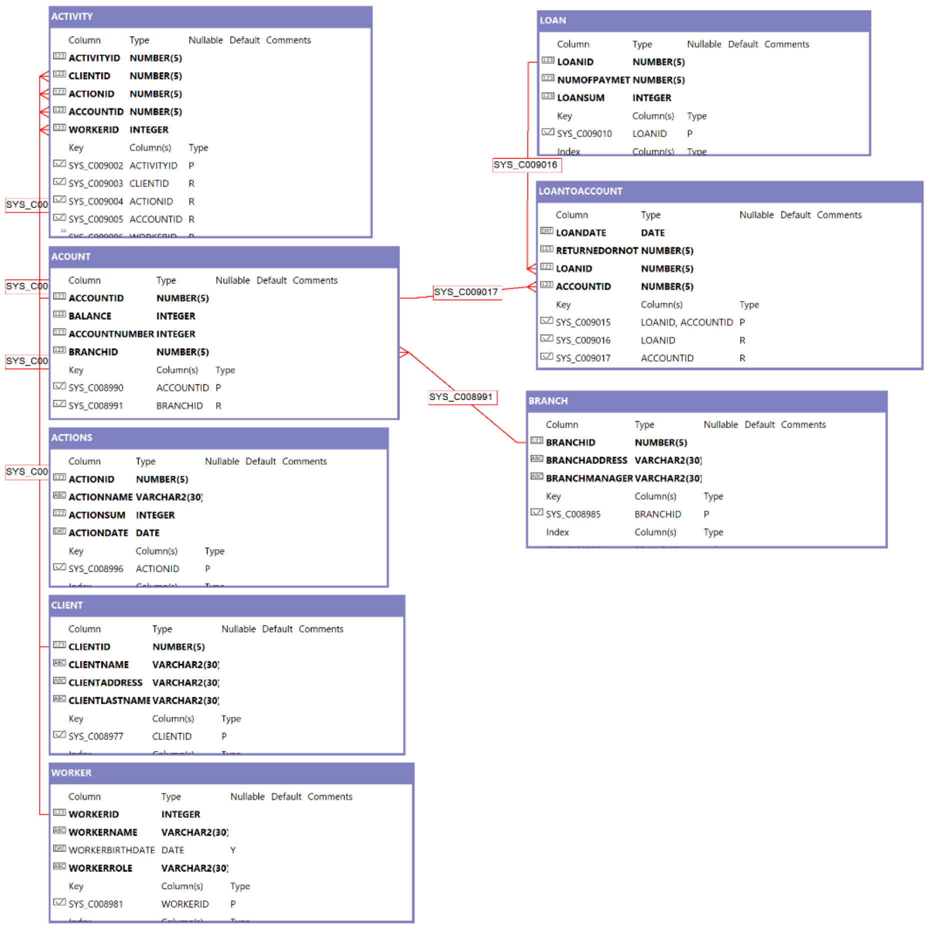
Step 4

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



ACTIVITY

Column	Type	Nullable	Default	Comments
ACTIVITYID	NUMBER(5)			
CLIENTID	NUMBER(5)			
ACTIONID	NUMBER(5)			
ACCOUNTID	NUMBER(5)			
WORKERID	INTEGER			

Key	Column(s)	Type
SYS_C009002	ACTIVITYID	P
SYS_C009003	CLIENTID	R
SYS_C009004	ACTIONID	R
SYS_C009005	ACCOUNTID	R
SYS_C009006	WORKERID	R

ACOUNT

Column	Type	Nullable	Default	Comments
ACCOUNTID	NUMBER(5)			
BALANCE	INTEGER			
ACCOUNTNUMBER	INTEGER			
BRANCHID	NUMBER(5)			

Key	Column(s)	Type
SYS_C008990	ACCOUNTID	P
SYS_C008991	BRANCHID	R

ACTIONS

Column	Type	Nullable	Default	Comments
ACTIONID	NUMBER(5)			
ACTIONNAME	VARCHAR2(30)			
ACTIONSUM	INTEGER			
ACTIONDATE	DATE			

Key	Column(s)	Type
SYS_C008996	ACTIONID	P

CLIENT

Column	Type	Nullable	Default	Comments
CLIENTID	NUMBER(5)			
CLIENTNAME	VARCHAR2(30)			
CLIENTADDRESS	VARCHAR2(30)			
CLIENTLASTNAME	VARCHAR2(30)			

Key	Column(s)	Type
SYS_C008977	CLIENTID	P

WORKER

Column	Type	Nullable	Default	Comments
WORKERID	INTEGER			
WORKERNAME	VARCHAR2(30)			
WORKERBIRTHDATE	DATE	Y		
WORKERROLE	VARCHAR2(30)			

Key	Column(s)	Type
SYS_C008981	WORKERID	P

LOAN

Column	Type	Nullable	Default	Comments
LOANID	NUMBER(5)			
NUMOFPAYMET	NUMBER(5)			
LOANSUM	INTEGER			

Key	Column(s)	Type
SYS_C009010	LOANID	P

Index	Column(s)	Type

LOANTOACCOUNT

Column	Type	Nullable	Default	Comments
LOANDATE	DATE			
RETURNEDORNOT	NUMBER(5)			
LOANID	NUMBER(5)			
ACCOUNTID	NUMBER(5)			

Key	Column(s)	Type
SYS_C009015	LOANID, ACCOUNTID	P
SYS_C009016	LOANID	R
SYS_C009017	ACCOUNTID	R

BRANCH

Column	Type	Nullable	Default	Comments
BRANCHID	NUMBER(5)			
BRANCHADDRESS	VARCHAR2(30)			
BRANCHMANAGER	VARCHAR2(30)			

Key	Column(s)	Type
SYS_C008985	BRANCHID	P

Index	Column(s)	Type

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.

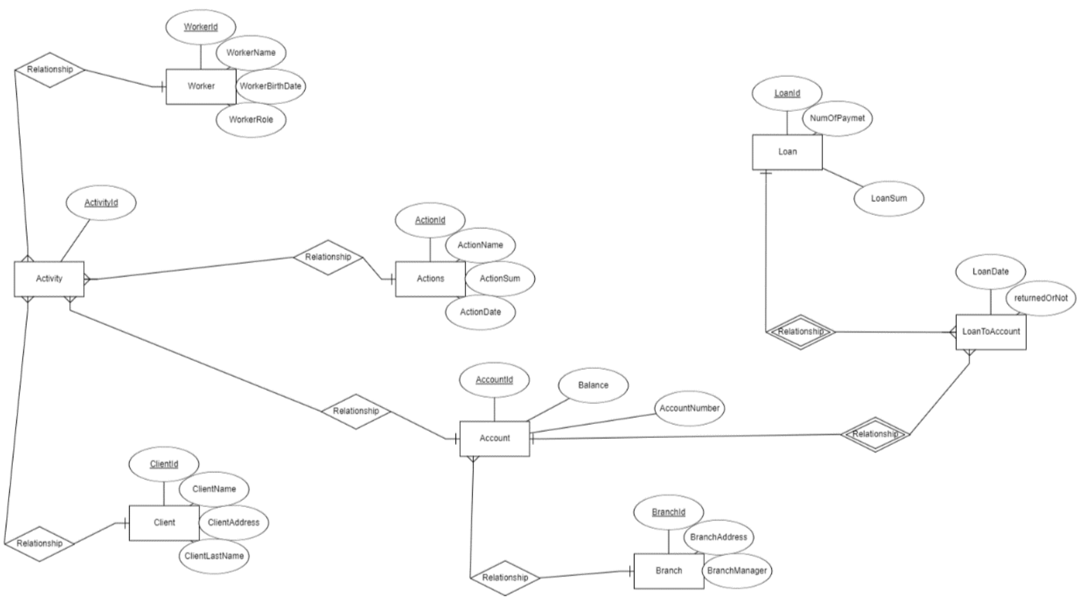
5 .Actions and Activity
- An action can be associated with multiple activities. This relationship is established by `ActionId` in the `Activity` table.

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

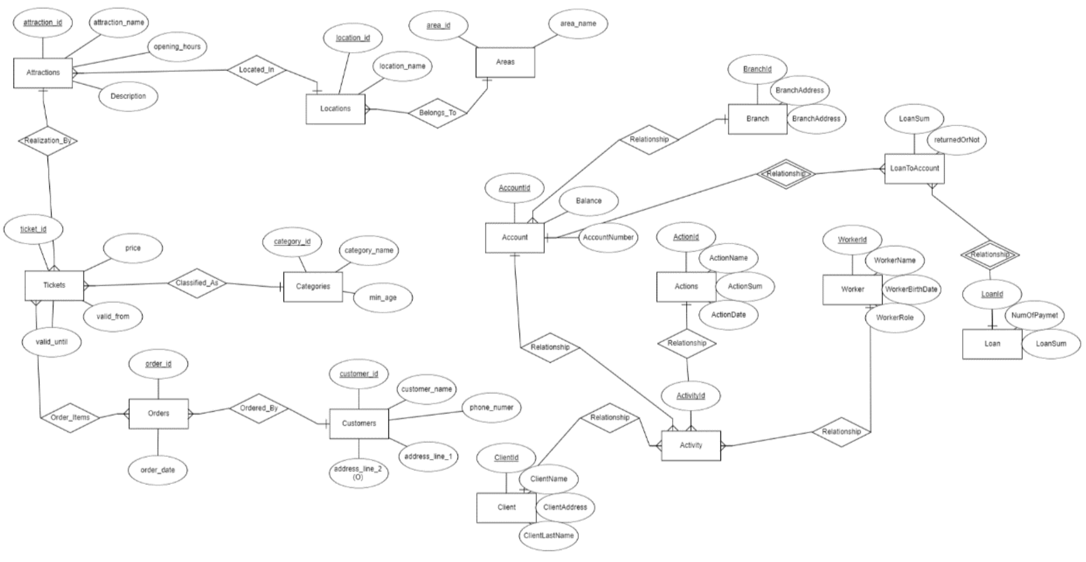
7 .Account and LoanToAccount
- An account can have multiple loans. This relationship is established by `AccountId` in the `LoanToAccount` table.

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.

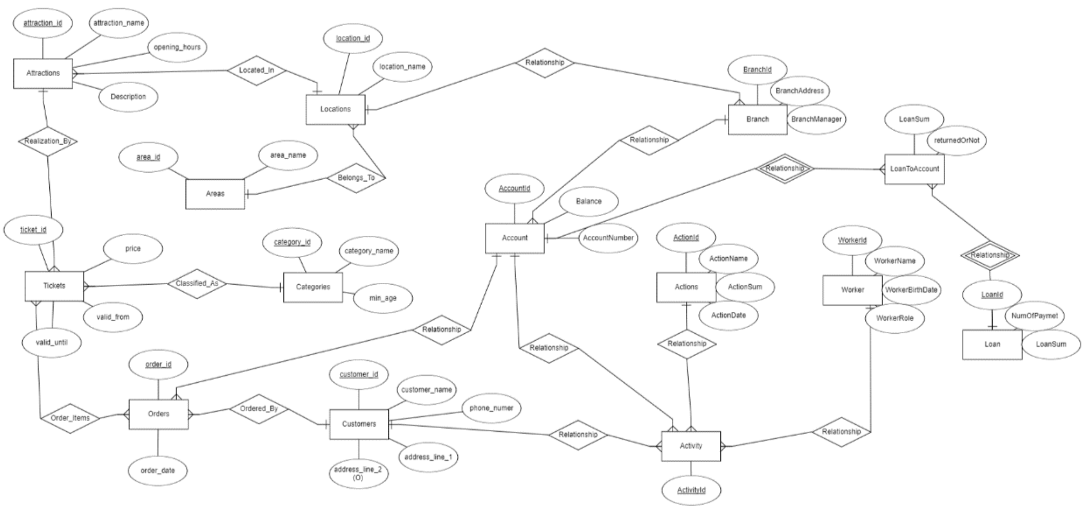
These entities and relationships create a structure where clients, workers, branches, accounts, actions, activities, and loans are interrelated, allowing comprehensive management and tracking of financial operations within the system.



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



Designing the new ERD by integrating the both projects.



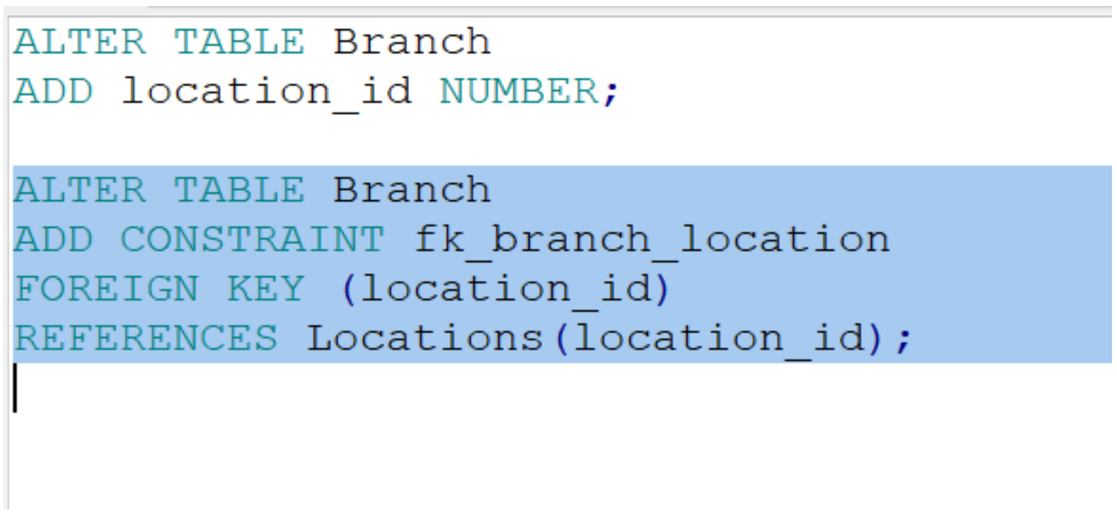
Updating the DB according to the new ERD.

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.

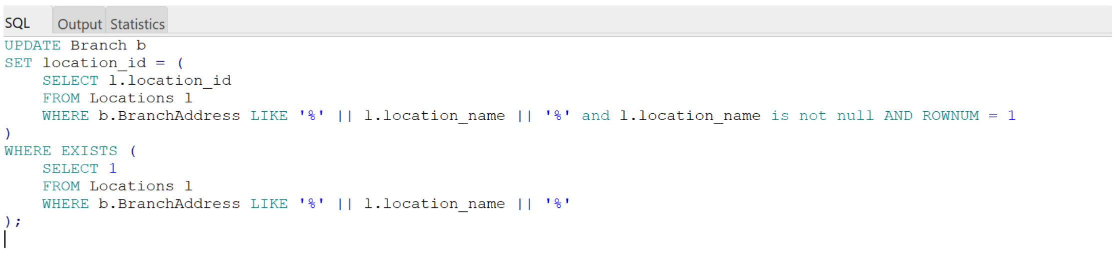


```sql
ALTER TABLE Branch
ADD location_id NUMBER;

ALTER TABLE Branch
ADD CONSTRAINT fk_branch_location
FOREIGN KEY (location_id)
REFERENCES Locations(location_id);
```

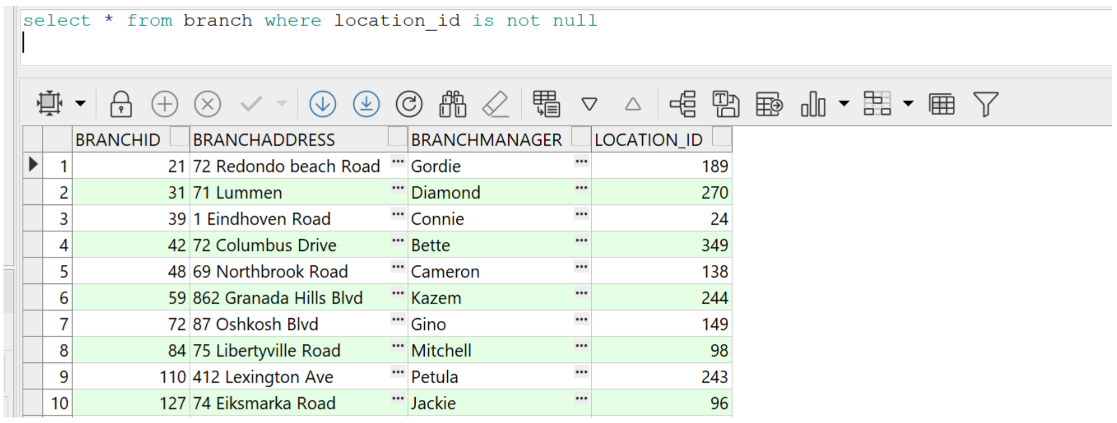
Setting location_id in Branch Table

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.



```sql
UPDATE Branch b
SET location_id = (
    SELECT l.location_id
    FROM Locations l
    WHERE b.BranchAddress LIKE '%' || l.location_name || '%' and l.location_name is not null AND ROWNUM = 1
)
WHERE EXISTS (
    SELECT 1
    FROM Locations l
    WHERE b.BranchAddress LIKE '%' || l.location_name || '%'
);
```

Verfiy the the data has been updated.



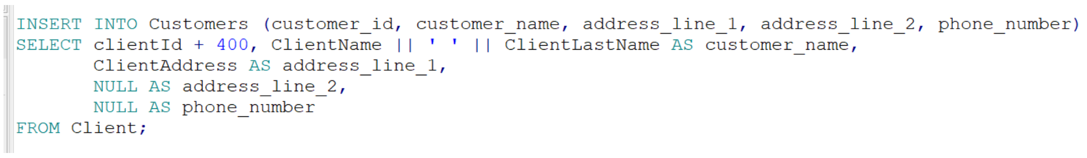
```sql
select * from branch where location_id is not null
```

	BRANCHID	BRANCHADDRESS	BRANCHMANAGER	LOCATION_ID
1	21	72 Redondo beach Road	Gordie	189
2	31	71 Lummen	Diamond	270
3	39	1 Eindhoven Road	Connie	24
4	42	72 Columbus Drive	Bette	349
5	48	69 Northbrook Road	Cameron	138
6	59	862 Granada Hills Blvd	Kazem	244
7	72	87 Oshkosh Blvd	Gino	149
8	84	75 Libertyville Road	Mitchell	98
9	110	412 Lexington Ave	Petula	243
10	127	74 Eiksmarka Road	Jackie	96

Data Migration from Client to Customers

Inserting Data into Customers

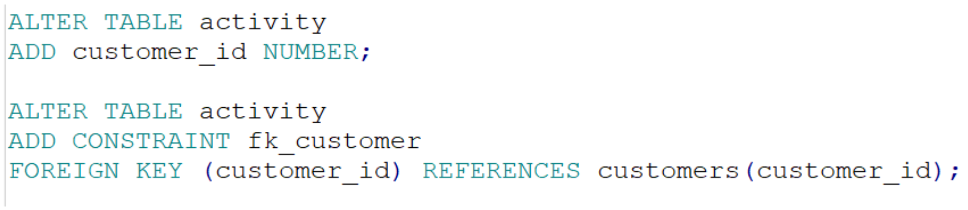
This insertion migrates data from the old Client table to the new Customers table. It creates new customer records by combining the first and last names of clients, and adjusting their IDs by adding 400 to ensure uniqueness. The client's address is also migrated, while the second address line and phone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



```
INSERT INTO Customers (customer_id, customer_name, address_line_1, address_line_2, phone_number)
SELECT clientId + 400, ClientName || ' ' || ClientLastName AS customer_name,
       ClientAddress AS address_line_1,
       NULL AS address_line_2,
       NULL AS phone_number
FROM Client;
```

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.



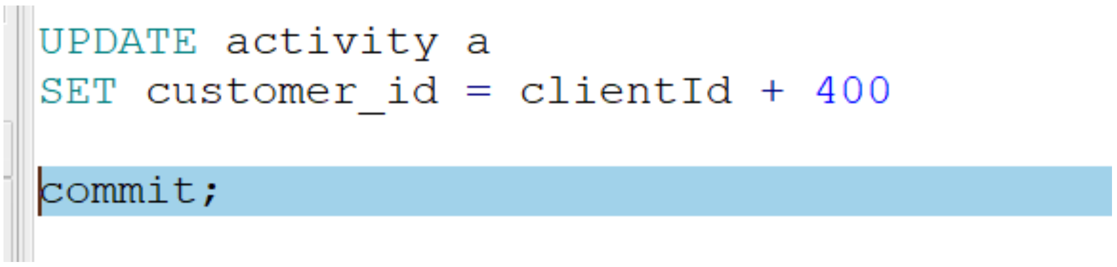
```
ALTER TABLE activity
ADD customer_id NUMBER;

ALTER TABLE activity
ADD CONSTRAINT fk_customer
FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
```

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.

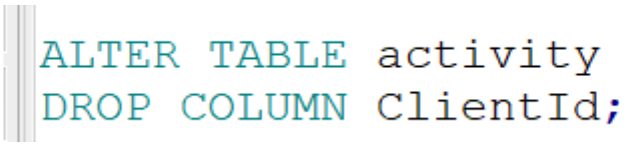


```
UPDATE activity a
SET customer_id = clientId + 400

commit;
```

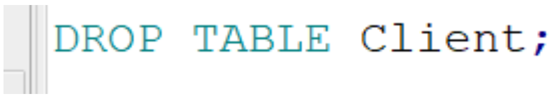
Removing Client Relationship from Activity

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



```
ALTER TABLE activity
DROP COLUMN ClientId;
```

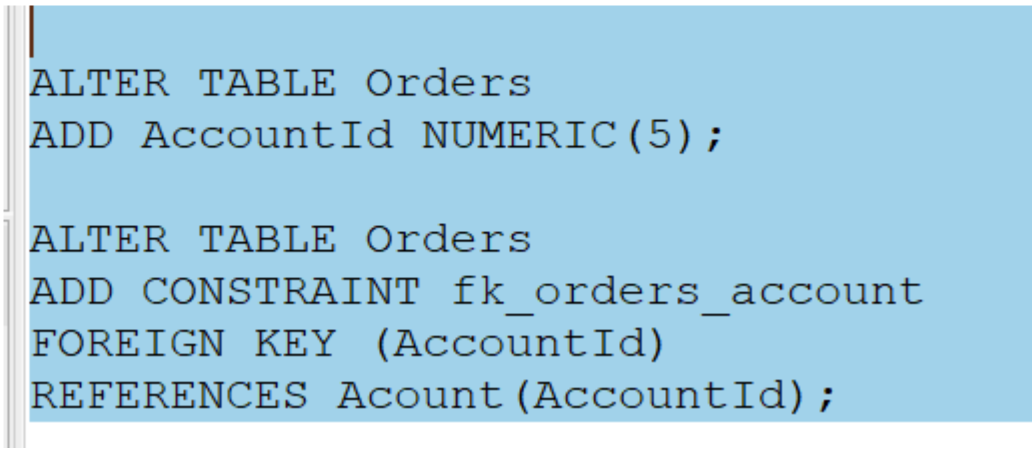
This deletes the Client table entirely, reflecting a decision to consolidate or replace the Client table with the Customers table.



```sql
DROP TABLE Client;
```

Modifying Orders to Include Account Relationship

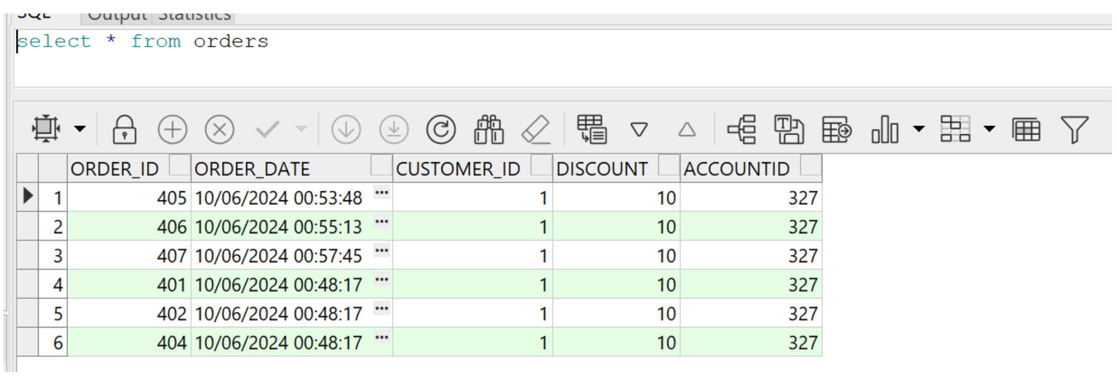
This adds an AccountId column to the Orders table to store references to accounts and sets AccountId as a foreign key in the Orders table, linking orders to specific accounts.



```sql
ALTER TABLE Orders
ADD AccountId NUMERIC(5);

ALTER TABLE Orders
ADD CONSTRAINT fk_orders_account
FOREIGN KEY (AccountId)
REFERENCES Acount(AccountId);
```

```
UPDATE Orders o
SET AccountId = (
    SELECT MIN(a.AccountId)
    FROM activity a
    WHERE o.customer_id = a.ClientId
)
WHERE EXISTS (
    SELECT 1
    FROM activity a
    WHERE o.customer_id = a.ClientId
);
```



```
select * from orders
```

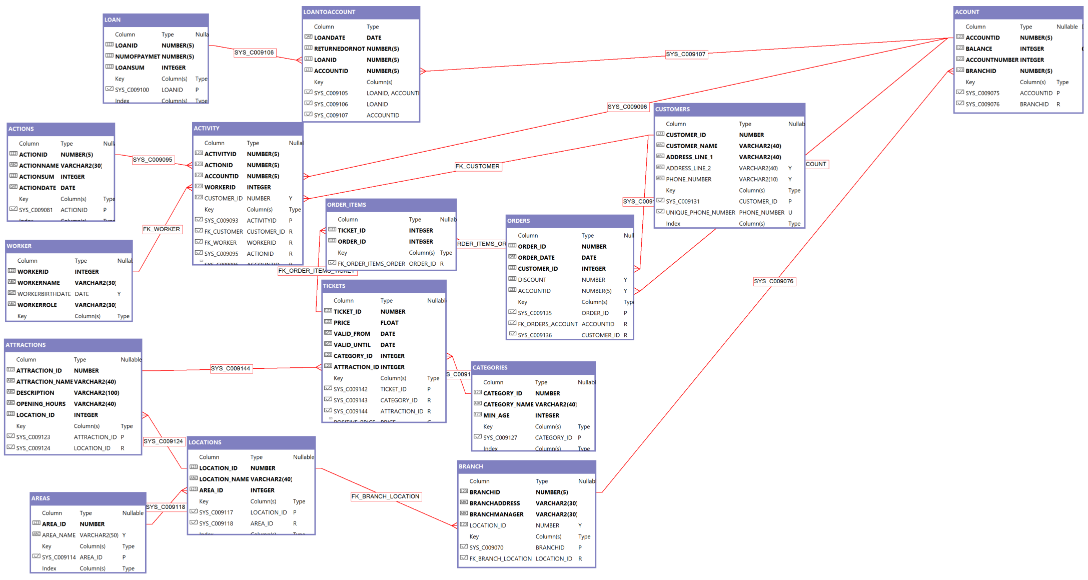
	ORDER_ID	ORDER_DATE	CUSTOMER_ID	DISCOUNT	ACCOUNTID
1	405	10/06/2024 00:53:48	1	10	327
2	406	10/06/2024 00:55:13	1	10	327
3	407	10/06/2024 00:57:45	1	10	327
4	401	10/06/2024 00:48:17	1	10	327
5	402	10/06/2024 00:48:17	1	10	327
6	404	10/06/2024 00:48:17	1	10	327

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:

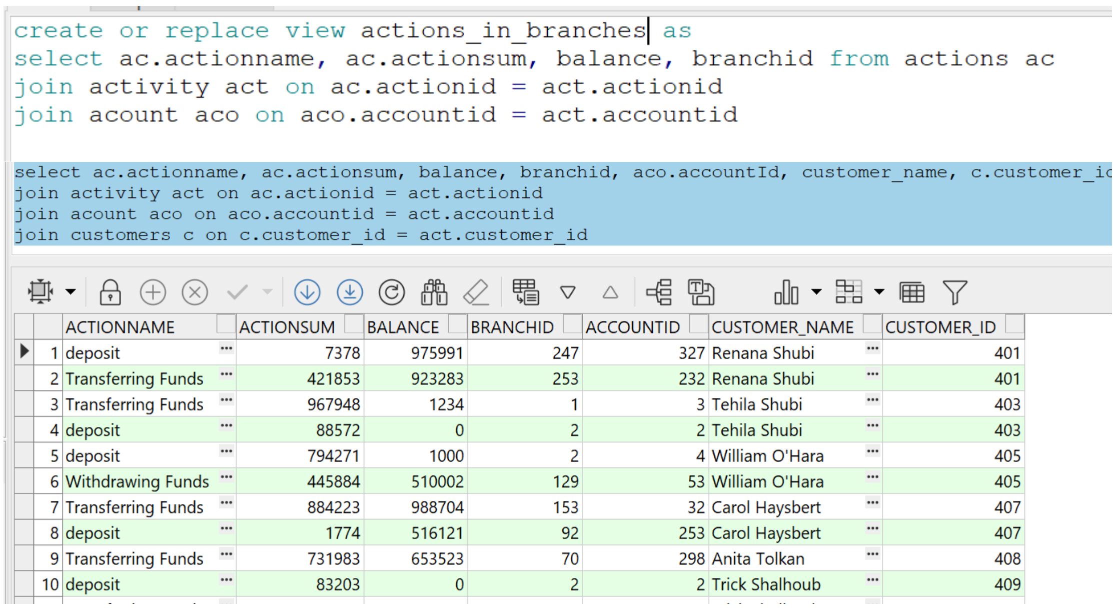
- Action Details: actionname and actionsum from the actions table.
- Account Balance: balance from the acount table.
- Branch ID: branchid from the acount table.
- Account ID: accountId from the acount table.
- Customer Details: customer_name and customer_id from the customers table.

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.



```
create or replace view actions_in_branches as
select ac.actionname, ac.actionsum, balance, branchid from actions ac
join activity act on ac.actionid = act.actionid
join acount aco on aco.accountid = act.accountid

select ac.actionname, ac.actionsum, balance, branchid, aco.accountId, customer_name, c.customer_i
join activity act on ac.actionid = act.actionid
join acount aco on aco.accountid = act.accountid
join customers c on c.customer_id = act.customer_id
```

	ACTIONNAME	ACTIONSUM	BALANCE	BRANCHID	ACCOUNTID	CUSTOMER_NAME	CUSTOMER_ID
1	deposit	7378	975991	247	327	Renana Shubi	401
2	Transferring Funds	421853	923283	253	232	Renana Shubi	401
3	Transferring Funds	967948	1234	1	3	Tehila Shubi	403
4	deposit	88572	0	2	2	Tehila Shubi	403
5	deposit	794271	1000	2	4	William O'Hara	405
6	Withdrawing Funds	445884	510002	129	53	William O'Hara	405
7	Transferring Funds	884223	988704	153	32	Carol Haysbert	407
8	deposit	1774	516121	92	253	Carol Haysbert	407
9	Transferring Funds	731983	653523	70	298	Anita Tolkan	408
10	deposit	83203	0	2	2	Trick Shalhoub	409

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:

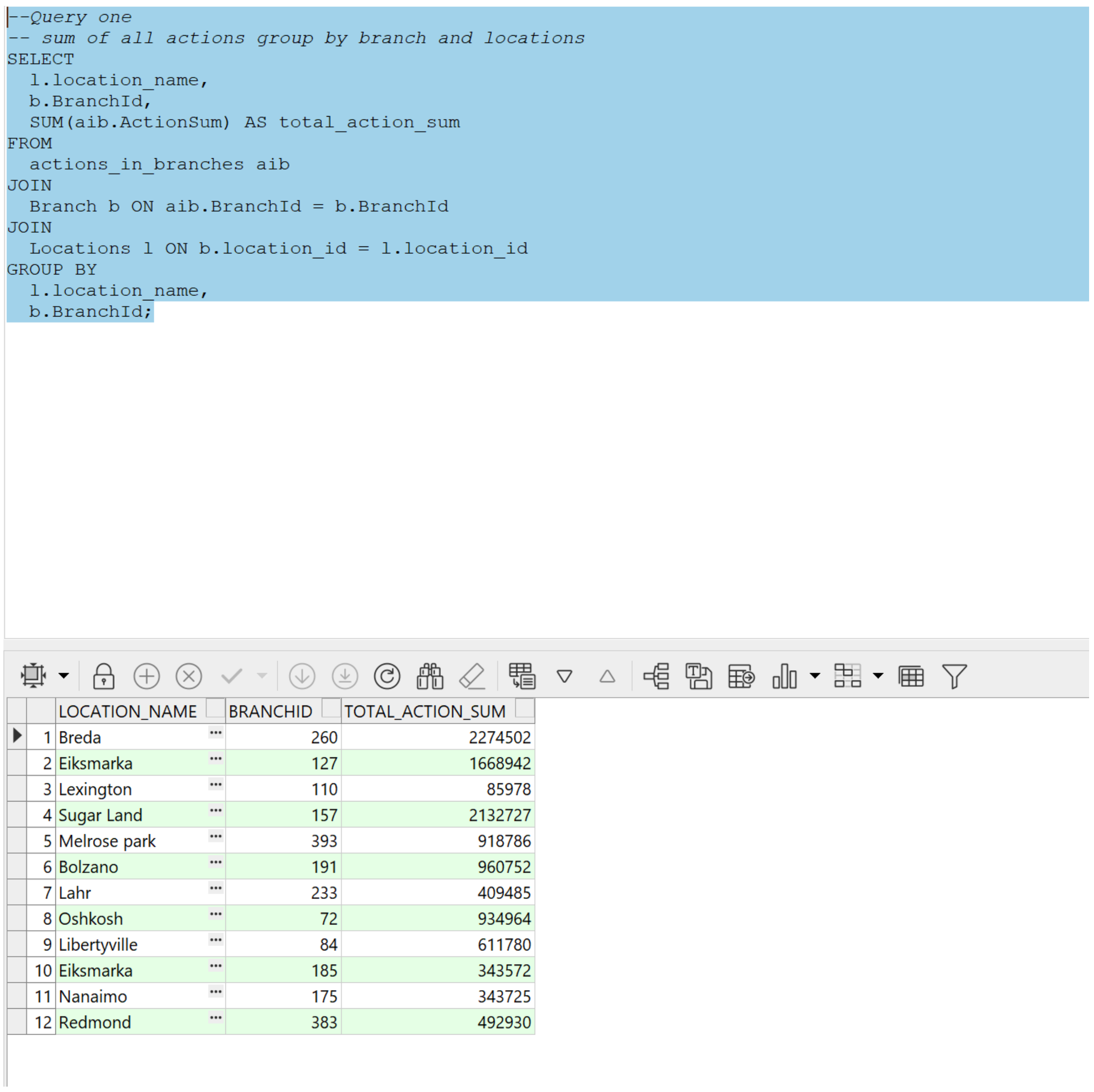
- **Joining actions_in_branches and Branch**: To associate each action with its branch.
- **Joining Branch and Locations**: To associate each branch with its location.

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.



```sql
--Query one
-- sum of all actions group by branch and locations
SELECT
  l.location_name,
  b.BranchId,
  SUM(aib.ActionSum) AS total_action_sum
FROM
  actions_in_branches aib
JOIN
  Branch b ON aib.BranchId = b.BranchId
JOIN
  Locations l ON b.location_id = l.location_id
GROUP BY
  l.location_name,
  b.BranchId;
```

	LOCATION_NAME	BRANCHID	TOTAL_ACTION_SUM
1	Breda	260	2274502
2	Eiksmarka	127	1668942
3	Lexington	110	85978
4	Sugar Land	157	2132727
5	Melrose park	393	918786
6	Bolzano	191	960752
7	Lahr	233	409485
8	Oshkosh	72	934964
9	Libertyville	84	611780
10	Eiksmarka	185	343572
11	Nanaimo	175	343725
12	Redmond	383	492930

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:

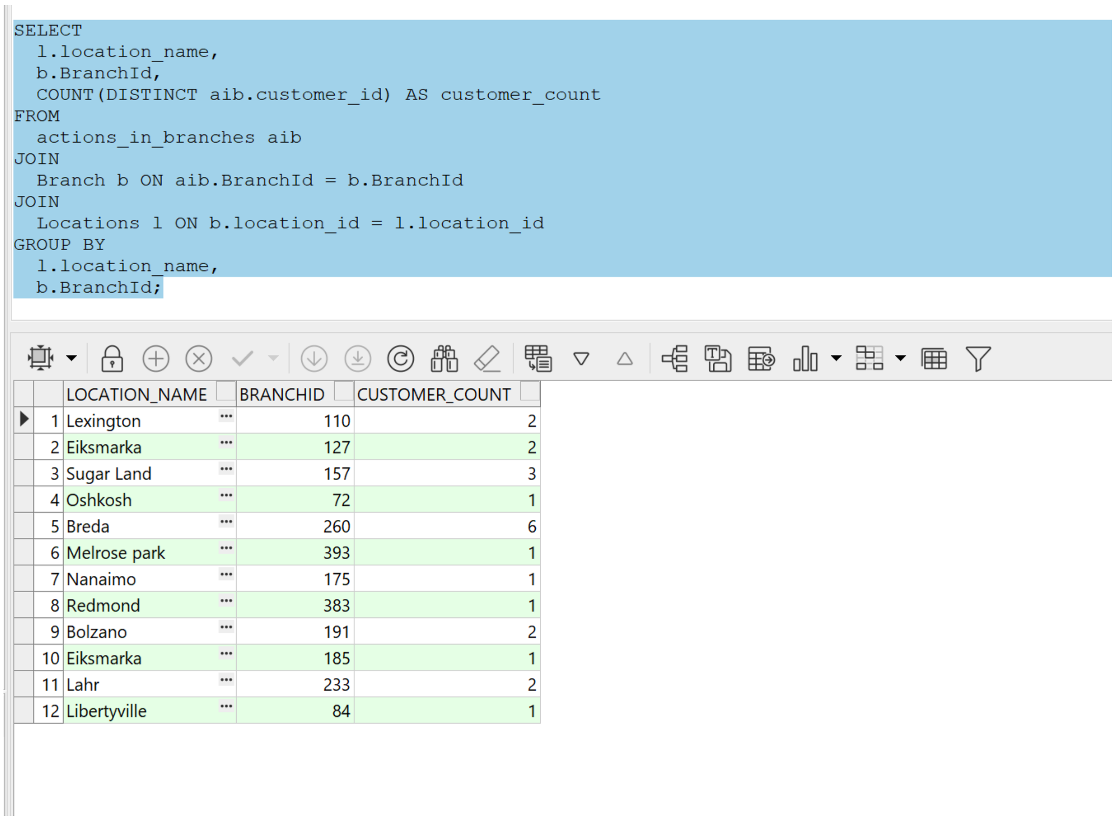
- **Joining actions_in_branches and Branch**: To associate each action with its branch.
- **Joining Branch and Locations**: To associate each branch with its location.

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.



```
SELECT
  l.location_name,
  b.BranchId,
  COUNT(DISTINCT aib.customer_id) AS customer_count
FROM
  actions_in_branches aib
JOIN
  Branch b ON aib.BranchId = b.BranchId
JOIN
  Locations l ON b.location_id = l.location_id
GROUP BY
  l.location_name,
  b.BranchId;
```

	LOCATION_NAME	BRANCHID	CUSTOMER_COUNT
1	Lexington	110	2
2	Eiksmarka	127	2
3	Sugar Land	157	3
4	Oshkosh	72	1
5	Breda	260	6
6	Melrose park	393	1
7	Nanaimo	175	1
8	Redmond	383	1
9	Bolzano	191	2
10	Eiksmarka	185	1
11	Lahr	233	2
12	Libertyville	84	1

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 ☺