

CIS 8040-Database Management

Assignment 1: Design and Implementation of a Relational Database

**Team 2: Gaurav Mishra, Kelly Duong, Manoj Potnuru, Md Kamruzzaman
Kamrul, Purvi Bharani**

Question 1: Select an application for which a database management system is needed. Describe the application and justify why it is an important application from a management perspective.

Ans:

Application Description:

The Bank Account Management System is a comprehensive application designed to facilitate a wide range of banking transactions for customers. This system enables customers to create and manage accounts, perform financial transactions like deposits and withdrawals, and access account reports. It can be accessed through various channels such as ATMs, telephone banking, online banking via computers, and mobile banking through smartphones.

Importance from a Management Perspective

- **Operational Efficiency:** Automates routine banking transactions, reducing the need for manual intervention and thereby increasing operational efficiency.
- **Customer Service Improvement:** Provides 24/7 banking facilities, enhancing customer convenience and satisfaction.
- **Financial Tracking and Reporting:** Helps in monitoring transactions and generating reports, which are vital for financial analysis and decision-making.
- **Security and Compliance:** Ensures adherence to financial regulations and enhances security measures to protect customer data and prevent fraud.
- **Market Competitiveness:** Offers a competitive edge in the banking industry by providing advanced and user-friendly banking solutions.

Question 2: State three important business rules that the database needs to be able to support. Explain how the database will support these business rules. An example is: employees must be designated as being either full-time or part-time employees. Note: Stating mapping ratios or min/max cardinalities is not a business rule.

Ans:

1) Transaction Limits with Managerial Approval

Rule: Transactions exceeding a certain amount require manager approval.

Database Support: Implement triggers to flag transactions above the threshold. These triggers can send automated alerts to managers for approval, ensuring compliance and oversight for large transactions.

2) Account Security Through Locking Mechanisms

Rule: Accounts with multiple failed login attempts are temporarily locked.

Database Support: The system tracks login attempts and utilizes a security protocol to lock accounts after a specified number of failed attempts. This information is logged and managed within the database to enhance security measures against unauthorized access.

3) Restricted Access to Account Information

Rule: Only authorized users can access and modify account information.

Database Support: Implement robust user authentication systems within the database, such as storing encrypted login credentials. The system could also incorporate multi-factor authentication processes, ensuring that only authorized personnel have access to sensitive account information.

4) Accurate and Timely Record-Keeping

Rule: All transactions must be recorded in real-time with time-stamped entries.

Database Support: The database system should be configured to automatically log every transaction with a timestamp. This ensures accurate historical data and aids in audits and compliance checks.

5) Compliance with Regulatory Standards

Rule: The system must comply with financial regulatory standards, including data privacy laws and anti-money laundering protocols.

Database Support: Ensure that the database management system is compliant with relevant laws and regulations. This includes data encryption, regular audits, and implementing protocols for anti-money laundering checks.

Question 3: Create a conceptual model for this application. Include proper names for entities, attributes, and relationships. Identify min/max cardinalities. You may use either the Chen or Crow's Feet representation. Use the notation presented in class. There should be 5-8 entities in the conceptual model.

Ans:

Conceptual Model Overflow:

Entities and Attributes:

- Customer
 - Attributes: CustomerID, Name, Address, Phone, Email
 - Relationships: The Customer is optional to have an account or more accounts and undergo Authentication.
- Account
 - Attributes: AccountID, Balance, DateOpened, TransactionLimit
 - Relationships: Owned by Customer, has Transactions, has Branch.
- Branch
 - Attributes: BranchID, Address, Phone.
 - Relationships: Services type at the branch, related to Transaction.
- Transaction
 - Attributes: TransactionID, Amount, Date, Type
 - Relationships: Occurs in an Account, and may require Manager Approval
- Manager
 - Attributes: ManagerID, Name
 - Relationships: Approves high-value Transactions
- Authentication
 - Attributes: CustomerID, AuthenticationMethod, LastLogin, FailedAttempts
 - Relationships: Validates Customer identity, linked to Customer

Relationships and Cardinalities:

- Customer-Account
 - A customer is optional to own an account or have multiple accounts (0, N).
 - But many Accounts are owned by one customer (1, N)
- Account-Branch
 - An account can have one branch specific for service (1,1).
 - Many branches may or do not have accounts (0, N).
- Branch-Manager
 - Branches must have one manager (1,1).
 - One manager working at one branch (1,1).
- Manager-Transaction

- A transaction may require zero or one manager's approval (0,1).
- A manager may not need to approve or can approve multiple transactions (0, N).
- Customer-Authentication
 - Each customer has one authentication record (1,1).
 - Authentication is specific to one customer (1,1).

Diagram Representation (Crow's Foot Notation):

Link: **Lucid.app**

https://lucid.app/lucidchart/a4af08ec-4321-4507-84c6-7654cbc209b1/edit?invitationId=inv_22e5fe0e-54de-4fc0-b0bc-801ef7a94bd5

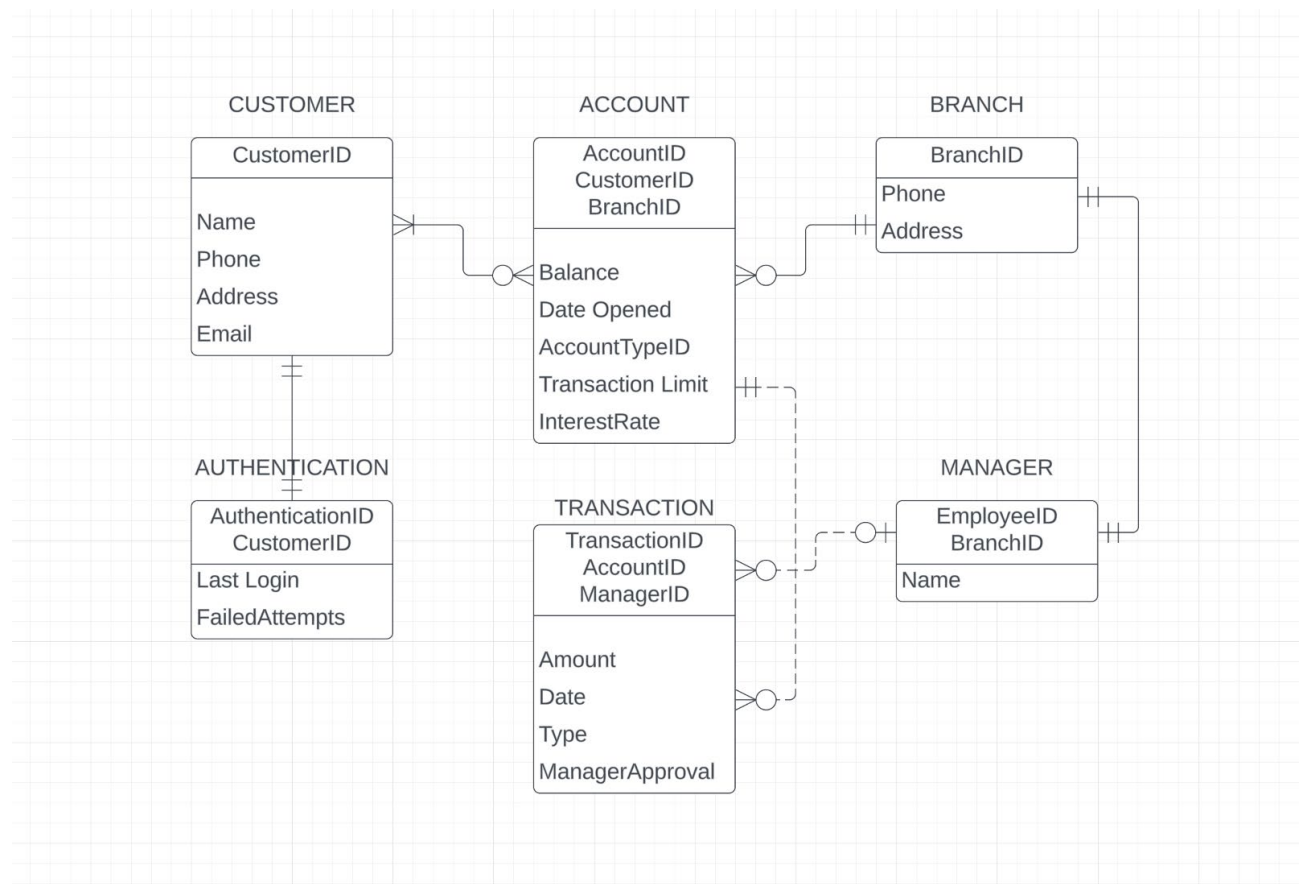
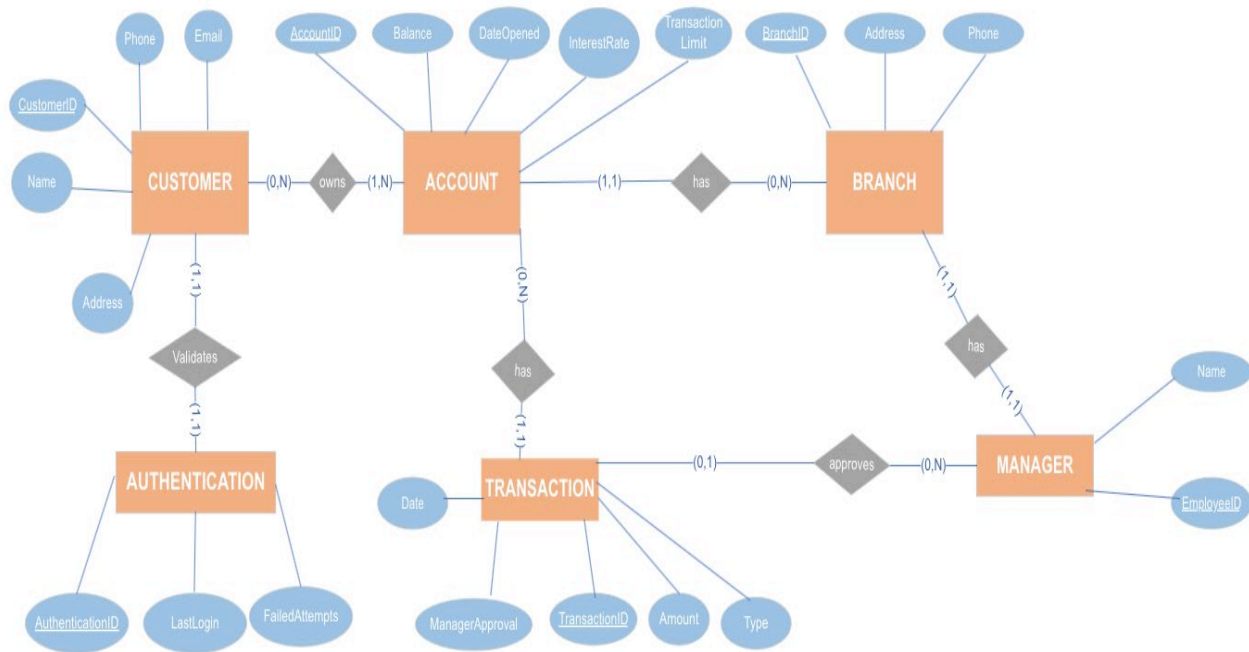


Diagram Representation (Chen Notation)

Link: [Chen's Notation](#)



Question 4: List 5 non-trivial queries that you would want to run against a populated database. For each query, justify why it would be important for the operations of a company.

Ans:

Query 1 - Identify Customers with Multiple Accounts:

Description: This query retrieves a list of customers who have multiple accounts. It joins the CUSTOMER and ACCOUNT tables, counting the number of accounts each customer has. Customers with more than one account are included in the result.

Justification: This query is useful for customer relationship management. Identifying customers with multiple accounts can help the company tailor its services, promotions, and communication to better serve these valuable customers. It also allows for a better understanding of customer behavior and preferences.

The screenshot displays the Oracle SQL Developer interface. On the left, the 'Connections' pane shows a connection to 'myOracle'. The 'Reports' pane is also visible. The main workspace shows a SQL query in the 'Query Builder' tab. The query is as follows:

```
SELECT
  CUSTOMER.CustomerID,
  CUSTOMER.Name,
  CUSTOMER.Address,
  CUSTOMER.Phone,
  CUSTOMER.Email,
  COUNT(ACCOUNT.AccountID) as NumberOfAccounts
FROM
  CUSTOMER
JOIN
  ACCOUNT ON CUSTOMER.CustomerID = ACCOUNT.CustomerID
GROUP BY
  CUSTOMER.CustomerID,
  CUSTOMER.Name,
  CUSTOMER.Address,
  CUSTOMER.Phone,
  CUSTOMER.Email
HAVING
  COUNT(ACCOUNT.AccountID) > 1;
```

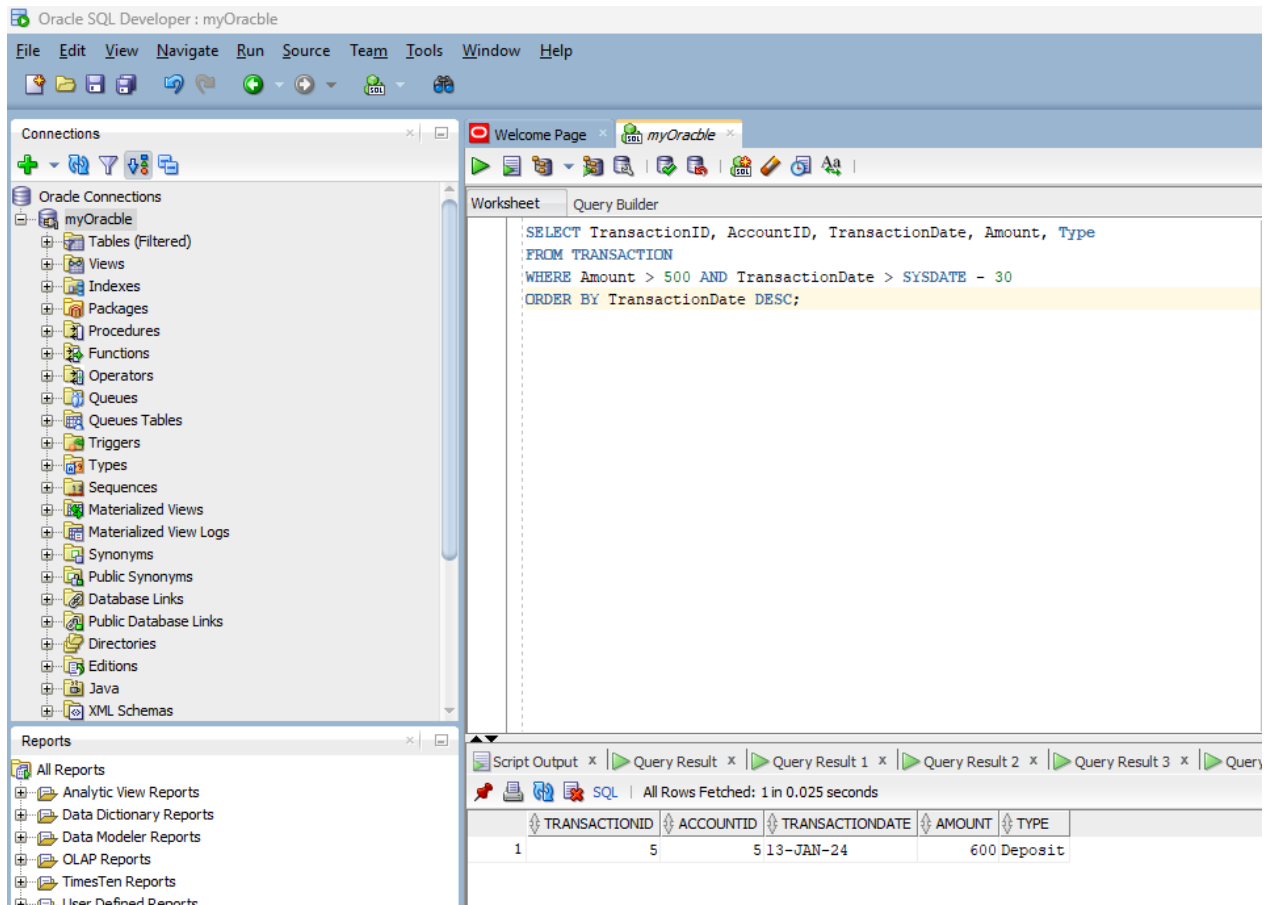
Below the query, the 'Script Output' pane shows the query results. The results are displayed in a table with the following columns: CUSTOMERID, NAME, ADDRESS, PHONE, EMAIL, and NUMBEROFACCOUNTS. The results are as follows:

CUSTOMERID	NAME	ADDRESS	PHONE	EMAIL	NUMBEROFACCOUNTS
1	John Doe	123 Main St	555-0101	john.doe@email.com	3
2	Alice Johnson	789 Oak St	555-0103	alice.johnson@email.com	2
3	Isabel Garcia	789 Pine St	555-0111	isabel.garcia@email.com	2
4	Jane Smith	456 Elm St	555-0102	jane.smith@email.com	3

Query 2 - Transaction above five hundred

Description: This query retrieves transactions with an amount greater than five hundred and which have occurred within the last 30 days. It selects TransactionID, AccountID, TransactionDate, Amount, and Type from the TRANSACTION table, filtering based on the specified criteria.

Justification: This query is important for monitoring high-value transactions. It helps the company identify significant financial activities within a recent time frame. Detecting large transactions can be crucial for fraud prevention and financial analysis.



The screenshot displays the Oracle SQL Developer interface. The 'Connections' pane on the left shows the 'myOracle' connection. The 'Query Builder' pane on the right contains the following SQL query:

```
SELECT TransactionID, AccountID, TransactionDate, Amount, Type
FROM TRANSACTION
WHERE Amount > 500 AND TransactionDate > SYSDATE - 30
ORDER BY TransactionDate DESC;
```

The 'Reports' pane at the bottom left lists various report types. The 'Script Output' pane at the bottom right shows the query results:

TRANSACTIONID	ACCOUNTID	TRANSACTIONDATE	AMOUNT	TYPE
1	5	5 13-JAN-24	600	Deposit

Query 3: List Inactive Accounts

Description: This query lists accounts that are considered inactive. It identifies accounts with a balance of zero and no transactions within the last 365 days. It includes AccountID, CustomerName, Balance, and the last transaction date.

Justification: Identifying inactive accounts is essential for managing account resources efficiently. It allows the company to identify accounts that may need attention, such as closure or reactivation. This query helps in maintaining a clean and updated account database.

The screenshot displays the Oracle SQL Developer interface. On the left, the 'Connections' pane shows a connection to 'myOracle'. The 'Reports' pane is also visible. The main window shows a SQL query in the 'Query Builder' tab. The query is as follows:

```
SELECT
  A.AccountID,
  C.Name AS CustomerName,
  A.Balance,
  MAX(T.TransactionDate) AS LastTransactionDate
FROM
  ACCOUNT A
JOIN
  CUSTOMER C ON A.CustomerID = C.CustomerID
LEFT JOIN
  TRANSACTION T ON A.AccountID = T.AccountID
GROUP BY
  A.AccountID, C.Name, A.Balance
HAVING
  A.Balance = 0
  AND MAX(T.TransactionDate) < TRUNC(SYSDATE) - 365;
```

Below the query, the 'Query Result' pane shows the results of the query. The results are displayed in a table with the following columns: ACCOUNTID, CUSTOMERNAME, BALANCE, and LASTTRANSACTIONDATE. The table contains one row of data:

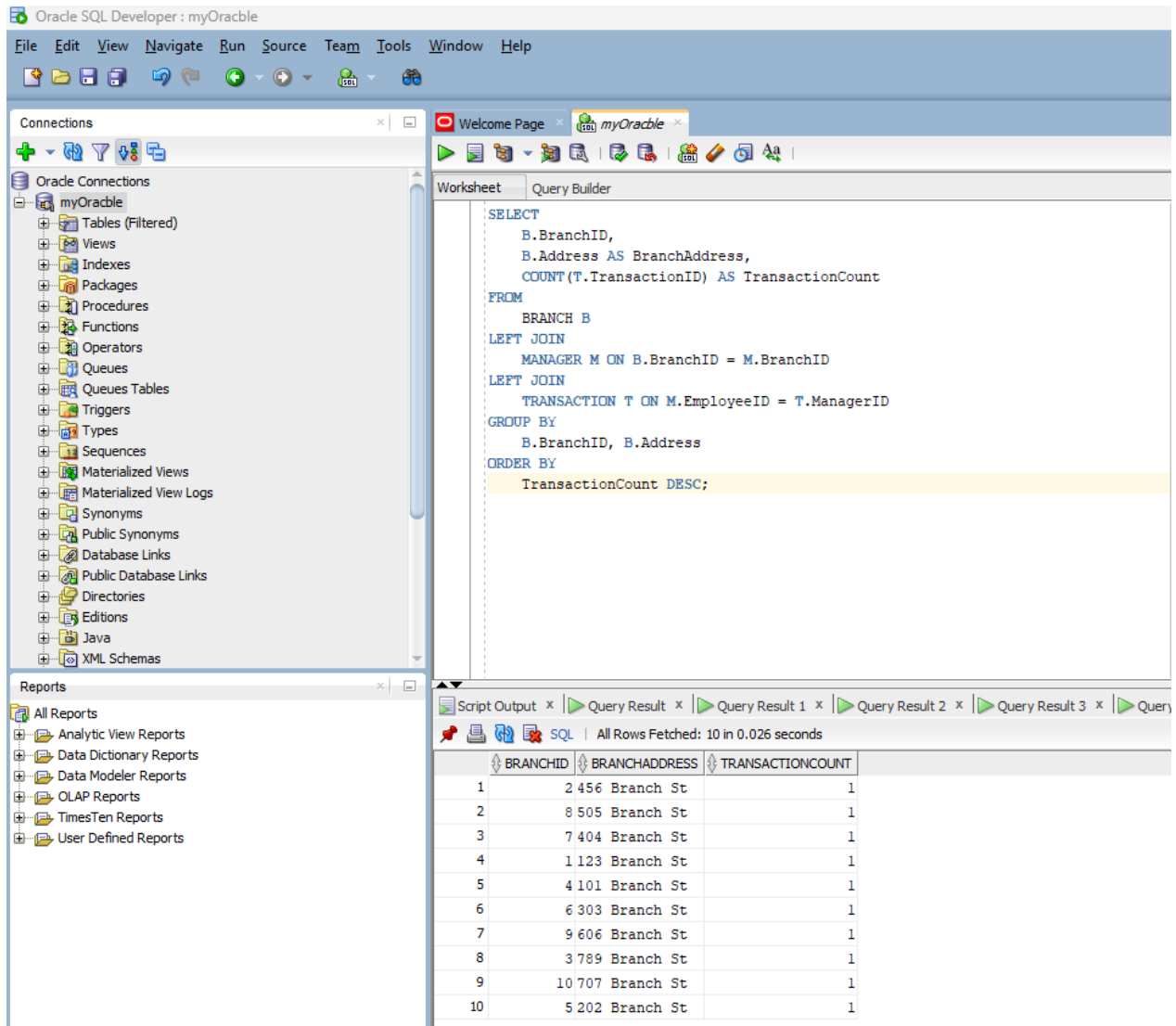
ACCOUNTID	CUSTOMERNAME	BALANCE	LASTTRANSACTIONDATE
1	1 John Doe	0	09-DEC-22

Justification: This ensures that transactions awaiting managerial approval are processed in a timely manner, maintaining operational efficiency and customer satisfaction. It also helps managers prioritize their tasks.

Query 4 - Determine Branch Activity Level

Description: This query determines the activity level of branches by counting the number of transactions associated with each branch. It retrieves BranchID, BranchAddress, and TransactionCount, ordering the results by transaction count in descending order.

Justification: Branch activity level is vital for assessing the performance of different branches. It helps the company allocate resources, identify high-performing branches, and improve customer service. This query provides insights into branch operations.



The screenshot displays the Oracle SQL Developer interface. The left pane shows the 'Connections' window with 'myOracle' selected. The main workspace shows a SQL query in the 'Query Builder' tab. The query is as follows:

```
SELECT
  B.BranchID,
  B.Address AS BranchAddress,
  COUNT(T.TransactionID) AS TransactionCount
FROM
  BRANCH B
LEFT JOIN
  MANAGER M ON B.BranchID = M.BranchID
LEFT JOIN
  TRANSACTION T ON M.EmployeeID = T.ManagerID
GROUP BY
  B.BranchID, B.Address
ORDER BY
  TransactionCount DESC;
```

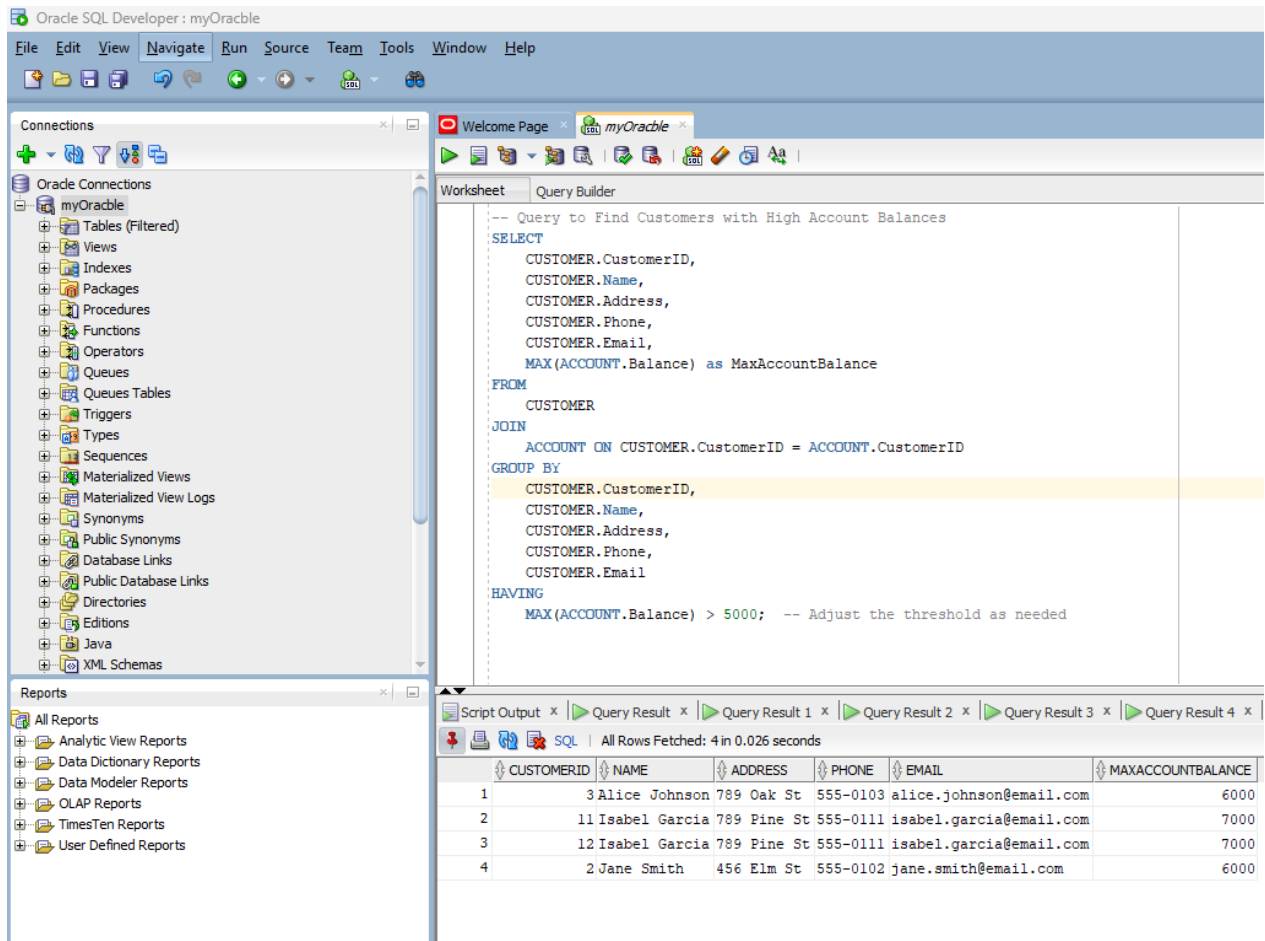
The bottom pane shows the 'Query Result' window with the following data:

	BRANCHID	BRANCHADDRESS	TRANSACTIONCOUNT
1	2 456	Branch St	1
2	8 505	Branch St	1
3	7 404	Branch St	1
4	1 123	Branch St	1
5	4 101	Branch St	1
6	6 303	Branch St	1
7	9 606	Branch St	1
8	3 789	Branch St	1
9	10 707	Branch St	1
10	5 202	Branch St	1

Query 5: Query to Find Customers with High Account Balances

Description: This query identifies customers with high account balances by joining the CUSTOMER and ACCOUNT tables, calculating their maximum account balance, and filtering for those exceeding a specified threshold, set at five thousand in the HAVING clause.

Justification: Valuable for tailoring services, marketing, and financial advice to high-balance customers, potentially increasing revenue. Aids in risk assessment and evaluating customer financial health. Threshold in HAVING clause adjustable for business goals and segmentation.



The screenshot displays the Oracle SQL Developer interface. The left pane shows the 'Connections' tree with 'myOracle' selected. The main workspace shows a SQL query in the 'Query Builder' tab. The query is as follows:

```
-- Query to Find Customers with High Account Balances
SELECT
  CUSTOMER.CustomerID,
  CUSTOMER.Name,
  CUSTOMER.Address,
  CUSTOMER.Phone,
  CUSTOMER.Email,
  MAX(ACCOUNT.Balance) as MaxAccountBalance
FROM
  CUSTOMER
JOIN
  ACCOUNT ON CUSTOMER.CustomerID = ACCOUNT.CustomerID
GROUP BY
  CUSTOMER.CustomerID,
  CUSTOMER.Name,
  CUSTOMER.Address,
  CUSTOMER.Phone,
  CUSTOMER.Email
HAVING
  MAX(ACCOUNT.Balance) > 5000; -- Adjust the threshold as needed
```

The bottom pane shows the 'Query Result' tab with the following data:

CUSTOMERID	NAME	ADDRESS	PHONE	EMAIL	MAXACCOUNTBALANCE
1	3 Alice Johnson	789 Oak St	555-0103	alice.johnson@email.com	6000
2	11 Isabel Garcia	789 Pine St	555-0111	isabel.garcia@email.com	7000
3	12 Isabel Garcia	789 Pine St	555-0111	isabel.garcia@email.com	7000
4	2 Jane Smith	456 Elm St	555-0102	jane.smith@email.com	6000

The queries can be found in the following link –

<https://colab.research.google.com/drive/1l-dwA82qNIfupYu32bWPmpYUHCasQtja?usp=sharing>

Question 5: For each entity, create a corresponding entity relation. Implement the entity relations in Oracle using your Oracle account provided for this course. Populate the relations with data. You will need to make up the data, which should be a reasonable reflection of the data that could occur in such an application. Show the populated relations. You can do this by using the "select * command. There should be at least 5-10 entries for each relation.

Ans:

In response to the assignment's instruction to "For each entity, create a corresponding entity relation," we have defined tables in the Oracle database to represent each entity within the database schema. Below is an overview of the tables (entity relations) created:

1. CUSTOMER Table:
 - Represents the CUSTOMER entity.
 - Contains columns such as CustomerID, Name, Address, Phone, and Email.
 - CustomerID is set as the primary key.
2. BRANCH Table:
 - Represents the BRANCH entity.
 - Contains columns like BranchID, Address, and Phone.
 - BranchID is set as the primary key.
3. ACCOUNT Table:
 - Represents the ACCOUNT entity.
 - Contains columns including AccountID, CustomerID, BranchID, Balance, DateOpened, InterestRate, and TransactionLimit.
 - CustomerID and BranchID are foreign keys that reference the CUSTOMER and BRANCH tables.
 - AccountID is set as the primary key.
4. MANAGER Table:
 - Represents the MANAGER entity.
 - Includes columns like EmployeeID, Name, and BranchID.
 - BranchID is a foreign key referencing the BRANCH table.
 - EmployeeID is set as the primary key.
5. TRANSACTION Table:
 - Represents the TRANSACTION entity.
 - Consists of columns such as TransactionID, AccountID, ManagerID, TransactionDate, Amount, and Type.
 - AccountID and ManagerID are foreign keys referring to the ACCOUNT and MANAGER tables, respectively.
 - TransactionID is set as the primary key.
6. AUTHENTICATION Table:
 - Represents the AUTHENTICATION entity.
 - Contains columns like AuthenticationID, CustomerID, LastLogin, and FailedAttempts.
 - CustomerID is a foreign key linking to the CUSTOMER table.

- AuthenticationID is set as the primary key.

Populating Data:

1) Creating & Populating Data in Customer Table

The screenshot displays the Oracle SQL Developer interface. On the left, the 'Connections' pane shows a connection to 'myOracle'. The 'Database Explorer' pane on the left lists various database objects, including Tables, Views, Indexes, Packages, Procedures, Functions, Operators, Queues, Queues Tables, Triggers, Types, Sequences, Materialized Views, Synonyms, Public Synonyms, Database Links, Public Database Links, Directories, Editions, Java, and XML Schemas. The 'Reports' pane at the bottom left shows a list of report types.

The main workspace is divided into two panes. The top pane, labeled 'Worksheet', contains SQL code for creating and populating the CUSTOMER table. The bottom pane, labeled 'Query Result', displays the output of the SQL query, showing 12 rows of data.

SQL Code:

```
-- Creating the CUSTOMER table
CREATE TABLE CUSTOMER (
  CustomerID NUMBER(38) PRIMARY KEY,
  Name VARCHAR2(255),
  Address VARCHAR2(255),
  Phone VARCHAR2(20),
  Email VARCHAR2(100)
);

-- Inserting data into CUSTOMER
INSERT INTO CUSTOMER VALUES (1, 'John Doe', '123 Main St', '555-0101', 'john.doe@email.com');
INSERT INTO CUSTOMER VALUES (2, 'Jane Smith', '456 Elm St', '555-0102', 'jane.smith@email.com');
INSERT INTO CUSTOMER VALUES (3, 'Alice Johnson', '789 Oak St', '555-0103', 'alice.johnson@email.com');
INSERT INTO CUSTOMER VALUES (4, 'Bob Brown', '101 Pine St', '555-0104', 'bob.brown@email.com');
INSERT INTO CUSTOMER VALUES (5, 'Carol White', '202 Maple St', '555-0105', 'carol.white@email.com');
INSERT INTO CUSTOMER VALUES (6, 'David Green', '303 Birch St', '555-0106', 'david.green@email.com');
INSERT INTO CUSTOMER VALUES (7, 'Eve Black', '404 Cedar St', '555-0107', 'eve.black@email.com');
INSERT INTO CUSTOMER VALUES (8, 'Frank Wright', '505 Cherry St', '555-0108', 'frank.wright@email.com');
INSERT INTO CUSTOMER VALUES (9, 'Grace Hall', '606 Apple St', '555-0109', 'grace.hall@email.com');
INSERT INTO CUSTOMER VALUES (10, 'Henry Ford', '707 Banana St', '555-0110', 'henry.ford@email.com');
INSERT INTO CUSTOMER VALUES (11, 'Isabel Garcia', '789 Pine St', '555-0111', 'isabel.garcia@email.com');
INSERT INTO CUSTOMER VALUES (12, 'Isabel Garcia', '789 Pine St', '555-0111', 'isabel.garcia@email.com');

SELECT * FROM CUSTOMER;
```

Query Result:

CUSTOMERID	NAME	ADDRESS	PHONE	EMAIL
1	John Doe	123 Main St	555-0101	john.doe@email.com
2	Jane Smith	456 Elm St	555-0102	jane.smith@email.com
3	Alice Johnson	789 Oak St	555-0103	alice.johnson@email.com
4	Bob Brown	101 Pine St	555-0104	bob.brown@email.com
5	Carol White	202 Maple St	555-0105	carol.white@email.com
6	David Green	303 Birch St	555-0106	david.green@email.com
7	Eve Black	404 Cedar St	555-0107	eve.black@email.com
8	Frank Wright	505 Cherry St	555-0108	frank.wright@email.com
9	Grace Hall	606 Apple St	555-0109	grace.hall@email.com
10	Henry Ford	707 Banana St	555-0110	henry.ford@email.com
11	Isabel Garcia	789 Pine St	555-0111	isabel.garcia@email.com
12	Isabel Garcia	789 Pine St	555-0111	isabel.garcia@email.com

2) Creating & Populating Data in Branch Table

Oracle SQL Developer : myOracle

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections

- myOracle
 - Tables (Filtered)
 - Views
 - Indexes
 - Packages
 - Procedures
 - Functions
 - Operators
 - Queues
 - Queues Tables
 - Triggers
 - Types
 - Sequences
 - Materialized Views
 - Materialized View Logs
 - Synonyms
 - Public Synonyms
 - Database Links
 - Public Database Links
 - Directories
 - Editions
 - Java
 - XML Schemas

Reports

- All Reports
- Analytic View Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

Welcome Page myOracle

Worksheet Query Builder

```
-- Creating the BRANCH table
CREATE TABLE BRANCH (
  BranchID NUMBER(38) PRIMARY KEY,
  Address VARCHAR2(255),
  Phone VARCHAR2(20)
);

-- Inserting data into BRANCH
INSERT INTO BRANCH VALUES (1, '123 Branch St', '555-0201');
INSERT INTO BRANCH VALUES (2, '456 Branch St', '555-0202');
INSERT INTO BRANCH VALUES (3, '789 Branch St', '555-0203');
INSERT INTO BRANCH VALUES (4, '101 Branch St', '555-0204');
INSERT INTO BRANCH VALUES (5, '202 Branch St', '555-0205');
INSERT INTO BRANCH VALUES (6, '303 Branch St', '555-0206');
INSERT INTO BRANCH VALUES (7, '404 Branch St', '555-0207');
INSERT INTO BRANCH VALUES (8, '505 Branch St', '555-0208');
INSERT INTO BRANCH VALUES (9, '606 Branch St', '555-0209');
INSERT INTO BRANCH VALUES (10, '707 Branch St', '555-0210');

SELECT * FROM BRANCH;
```

Script Output x Query Result x

SQL | All Rows Fetched: 10 in 0.024 seconds

BRANCHID	ADDRESS	PHONE
1	1 123 Branch St	555-0201
2	2 456 Branch St	555-0202
3	3 789 Branch St	555-0203
4	4 101 Branch St	555-0204
5	5 202 Branch St	555-0205
6	6 303 Branch St	555-0206
7	7 404 Branch St	555-0207
8	8 505 Branch St	555-0208
9	9 606 Branch St	555-0209
10	10 707 Branch St	555-0210

3) Creating & Populating Data in Account Table

Oracle SQL Developer: myOracle

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections

myOracle

- Tables (Filtered)
- Views
- Indexes
- Packages
- Procedures
- Functions
- Operators
- Queues
- Queues Tables
- Triggers
- Types
- Sequences
- Materialized Views
- Materialized View Logs
- Synonyms
- Public Synonyms
- Database Links
- Public Database Links
- Directories
- Editions
- Java
- XML Schemas

Reports

- All Reports
- Analytic View Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

Welcome Page myOracle

Worksheet Query Builder

```
AccountID NUMBER(38) PRIMARY KEY,
CustomerID NUMBER(38),
BranchID NUMBER(38),
Balance DECIMAL(15, 2),
DateOpened DATE,
InterestRate DECIMAL(5, 4),
TransactionLimit DECIMAL(15, 2),
CONSTRAINT fk_account_customer
    FOREIGN KEY (CustomerID)
        REFERENCES CUSTOMER(CustomerID),
CONSTRAINT fk_account_branch
    FOREIGN KEY (BranchID)
        REFERENCES BRANCH(BranchID)
);

-- Inserting data into ACCOUNT
INSERT INTO ACCOUNT VALUES (1, 1, 1, 0, SYSDATE, 0.02, 5000.00);
INSERT INTO ACCOUNT VALUES (2, 2, 2, 6000.00, SYSDATE, 0.025, 6000.00);
INSERT INTO ACCOUNT VALUES (3, 3, 3, 6000.00, SYSDATE, 0.03, 7000.00);
INSERT INTO ACCOUNT VALUES (4, 4, 4, 2500.00, SYSDATE, 0.035, 8000.00);
INSERT INTO ACCOUNT VALUES (5, 5, 5, 3000.00, SYSDATE, 0.04, 9000.00);
INSERT INTO ACCOUNT VALUES (6, 1, 6, 3500.00, SYSDATE, 0.045, 10000.00);
INSERT INTO ACCOUNT VALUES (7, 1, 7, 4000.00, SYSDATE, 0.05, 11000.00);
INSERT INTO ACCOUNT VALUES (8, 2, 8, 6000.00, SYSDATE, 0.055, 12000.00);
INSERT INTO ACCOUNT VALUES (9, 2, 9, 6000.00, SYSDATE, 0.06, 13000.00);
INSERT INTO ACCOUNT VALUES (10, 3, 10, 6000.00, SYSDATE, 0.065, 14000.00);
INSERT INTO ACCOUNT VALUES (10, 3, 10, 7000.00, SYSDATE, 0.065, 14000.00);
INSERT INTO ACCOUNT VALUES (10, 3, 10, 7000.00, SYSDATE, 0.065, 14000.00);
INSERT INTO ACCOUNT VALUES (10, 3, 10, 7000.00, SYSDATE, 0.065, 14000.00);
SELECT * FROM ACCOUNT;
```

Script Output Query Result

SQL All Rows Fetched: 13 in 0.025 seconds

	ACCOUNTID	CUSTOMERID	BRANCHID	BALANCE	DATEOPENED	INTERESTRATE	TRANSACTIONLIMIT
1	1	1	1	0	13-JAN-24	0.02	5000
2	2	2	2	6000	13-JAN-24	0.025	6000
3	3	3	3	6000	13-JAN-24	0.03	7000
4	4	4	4	2500	13-JAN-24	0.035	8000
5	5	5	5	3000	13-JAN-24	0.04	9000
6	6	6	1	3500	13-JAN-24	0.045	10000
7	7	7	1	4000	13-JAN-24	0.05	11000
8	8	2	8	6000	13-JAN-24	0.055	12000
9	9	2	9	6000	13-JAN-24	0.06	13000
10	10	3	10	6000	13-JAN-24	0.065	14000
11	11	11	4	7000	13-JAN-24	0.065	14000
12	12	12	4	7000	13-JAN-24	0.065	14000
13	14	12	4	7000	13-JAN-24	0.065	14000

4) Creating & Populating Data in Manager Table

Oracle SQL Developer : myOracleble

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections

- myOracle
- Tables (Filtered)
- Views
- Indexes
- Packages
- Procedures
- Functions
- Operators
- Queues
- Queues Tables
- Triggers
- Types
- Sequences
- Materialized Views
- Materialized View Logs
- Synonyms
- Public Synonyms
- Database Links
- Public Database Links
- Directories
- Editions
- Java
- XML Schemas

Worksheet

Query Builder

```
-- Creating the MANAGER table with a foreign key to BRANCH
CREATE TABLE MANAGER (
  EmployeeID NUMBER(38) PRIMARY KEY,
  Name VARCHAR2(255),
  BranchID NUMBER(38),
  CONSTRAINT fk_manager_branch
    FOREIGN KEY (BranchID)
      REFERENCES BRANCH(BranchID)
);

-- Inserting data into ACCOUNT
-- Inserting data into MANAGER
INSERT INTO MANAGER VALUES (1, 'Alice Smith', 1);
INSERT INTO MANAGER VALUES (2, 'Bob Johnson', 2);
INSERT INTO MANAGER VALUES (3, 'Charlie Davis', 3);
INSERT INTO MANAGER VALUES (4, 'Diana Evans', 4);
INSERT INTO MANAGER VALUES (5, 'Edward Collins', 5);
INSERT INTO MANAGER VALUES (6, 'Fiona Brown', 6);
INSERT INTO MANAGER VALUES (7, 'George Anderson', 7);
INSERT INTO MANAGER VALUES (8, 'Hannah Martin', 8);
INSERT INTO MANAGER VALUES (9, 'Ian Clark', 9);
INSERT INTO MANAGER VALUES (10, 'Julia Scott', 10);
SELECT * FROM MANAGER;
```

Script Output x Query Result x

SQL | All Rows Fetched: 10 in 0.025 seconds

	EMPLOYEEID	NAME	BRANCHID
1	1	Alice Smith	1
2	2	Bob Johnson	2
3	3	Charlie Davis	3
4	4	Diana Evans	4
5	5	Edward Collins	5
6	6	Fiona Brown	6
7	7	George Anderson	7
8	8	Hannah Martin	8
9	9	Ian Clark	9
10	10	Julia Scott	10

Reports

- All Reports
- Analytic View Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

5) Creating & Populating Data in Transaction Table

Oracle SQL Developer : myOracle

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections

myOracle

Tables (Filtered)

Views

Indexes

Packages

Procedures

Functions

Operators

Queues

Queues Tables

Triggers

Types

Sequences

Materialized Views

Materialized View Logs

Synonyms

Public Synonyms

Database Links

Public Database Links

Directories

Editions

Java

XML Schemas

Reports

All Reports

Analytic View Reports

Data Dictionary Reports

Data Modeler Reports

OLAP Reports

TimesTen Reports

User Defined Reports

Worksheet

Query Builder

```
-- Creating the TRANSACTION table with foreign keys to ACCOUNT and MANAGER
CREATE TABLE TRANSACTION (
  TransactionID NUMBER(38) PRIMARY KEY,
  AccountID NUMBER(38),
  ManagerID NUMBER(38),
  TransactionDate DATE,
  Amount DECIMAL(15, 2),
  Type VARCHAR2(50),
  CONSTRAINT fk_transaction_account
    FOREIGN KEY (AccountID)
      REFERENCES ACCOUNT(AccountID),
  CONSTRAINT fk_transaction_manager
    FOREIGN KEY (ManagerID)
      REFERENCES MANAGER(EmployeeID)
);

-- Inserting data into TRANSACTION
INSERT INTO TRANSACTION VALUES (1, 1, 1, SYSDATE, 600.00, 'Deposit');
INSERT INTO TRANSACTION VALUES (2, 2, 2, SYSDATE, 300.00, 'Withdrawal');
INSERT INTO TRANSACTION VALUES (3, 3, 3, SYSDATE, 600.00, 'Deposit');
INSERT INTO TRANSACTION VALUES (4, 4, 4, SYSDATE, 500.00, 'Withdrawal');
INSERT INTO TRANSACTION VALUES (5, 5, 5, SYSDATE, 600.00, 'Deposit');
INSERT INTO TRANSACTION VALUES (6, 1, 6, SYSDATE, 100.00, 'Withdrawal');
INSERT INTO TRANSACTION VALUES (7, 2, 7, SYSDATE, 150.00, 'Deposit');
INSERT INTO TRANSACTION VALUES (8, 3, 8, SYSDATE, 200.00, 'Withdrawal');
INSERT INTO TRANSACTION VALUES (9, 4, 4, SYSDATE, 350.00, 'Deposit');
INSERT INTO TRANSACTION VALUES (10, 5, 5, SYSDATE, 400.00, 'Withdrawal');
SELECT * FROM TRANSACTION;
```

Script Output x Query Result x

SQL | All Rows Fetched: 10 in 0.025 seconds

TRANSACTIONID	ACCOUNTID	MANAGERID	TRANSACTIONDATE	AMOUNT	TYPE
1	1	1	1 09-DEC-22	600	Deposit
2	2	2	2 09-DEC-22	300	Withdrawal
3	3	3	3 09-DEC-22	600	Deposit
4	4	4	4 13-JAN-24	500	Withdrawal
5	5	5	5 13-JAN-24	600	Deposit
6	6	1	6 09-DEC-22	100	Withdrawal
7	7	2	7 09-DEC-22	150	Deposit
8	8	3	8 09-DEC-22	200	Withdrawal
9	9	4	9 13-JAN-24	350	Deposit
10	10	5	10 13-JAN-24	400	Withdrawal

6) Creating & Populating Data in Authentication Table

The screenshot displays the Oracle SQL Developer interface with the 'myOracle' database selected. The left-hand 'Connections' pane shows the database structure, including Tables, Views, Indexes, Packages, Procedures, Functions, Operators, Queues, Queues Tables, Triggers, Types, Sequences, Materialized Views, Materialized View Logs, Synonyms, Public Synonyms, Database Links, Public Database Links, Directories, Editions, Java, and XML Schemas. The main workspace shows a SQL script for creating and populating the AUTHENTICATION table.

```
-- Creating the AUTHENTICATION table with a foreign key to CUSTOMER
CREATE TABLE AUTHENTICATION (
  AuthenticationID NUMBER(38) PRIMARY KEY,
  CustomerID NUMBER(38),
  LastLogin DATE,
  FailedAttempts NUMBER(38),
  CONSTRAINT fk_authentication_customer
    FOREIGN KEY (CustomerID)
      REFERENCES CUSTOMER (CustomerID)
);

-- Inserting data into AUTHENTICATION
INSERT INTO AUTHENTICATION VALUES (1, 1, SYSDATE, 3);
INSERT INTO AUTHENTICATION VALUES (2, 2, SYSDATE, 3);
INSERT INTO AUTHENTICATION VALUES (3, 3, SYSDATE, 0);
INSERT INTO AUTHENTICATION VALUES (4, 4, SYSDATE, 2);
INSERT INTO AUTHENTICATION VALUES (5, 5, SYSDATE, 0);
INSERT INTO AUTHENTICATION VALUES (6, 6, SYSDATE, 3);
INSERT INTO AUTHENTICATION VALUES (7, 7, SYSDATE, 3);
INSERT INTO AUTHENTICATION VALUES (8, 8, SYSDATE, 1);
INSERT INTO AUTHENTICATION VALUES (9, 9, SYSDATE, 0);
INSERT INTO AUTHENTICATION VALUES (10, 10, SYSDATE, 4);
INSERT INTO AUTHENTICATION VALUES (11, 11, SYSDATE, 3);
INSERT INTO AUTHENTICATION VALUES (12, 12, SYSDATE, 3);
INSERT INTO AUTHENTICATION VALUES (13, 13, SYSDATE, 3);
SELECT * FROM AUTHENTICATION;
```

The 'Script Output' pane at the bottom shows the results of the SQL execution, displaying 13 rows of data from the AUTHENTICATION table. The columns are AUTHENTICATIONID, CUSTOMERID, LASTLOGIN, and FAILEDATTEMPTS.

AUTHENTICATIONID	CUSTOMERID	LASTLOGIN	FAILEDATTEMPTS
1	1	1 13-JAN-24	3
2	2	2 13-JAN-24	3
3	3	3 13-JAN-24	0
4	4	4 13-JAN-24	2
5	5	5 13-JAN-24	0
6	6	1 13-JAN-24	3
7	7	2 13-JAN-24	3
8	8	3 13-JAN-24	1
9	9	4 13-JAN-24	0
10	10	5 13-JAN-24	4
11	11	11 13-JAN-24	3
12	12	12 13-JAN-24	3
13	13	12 13-JAN-24	3

Question 6: Describe how you would expand your conceptual model if you were to implement it for a real-world application. Identify what additional constructs (entities, relationships, attributes) you would add and the purpose of each. Based on the expanded model, what additional queries would you be able to answer?

Ans:

We can add additional features like Loan, Investment, ATM for a banking application as entities, explain the purpose, and discuss additional queries.

Additional Constructs for Expansion:

- **Loan**
 - **Attributes:** LoanID, Amount, InterestRate, StartDate, EndDate, CustomerID, LoanTypeID
 - **Purpose:** To manage customer loans, track loan amounts, interest rates, and repayment schedules.
 - **Additional Queries:**
 - Find customers with outstanding loans above a certain amount.
 - Generate reports on loan repayments due within the next month.
- **Investment**
 - **Attributes:** InvestmentID, Type, Amount, Date, CustomerID, RiskLevel
 - **Purpose:** To track customer investments, types of investments (stocks, bonds, etc.), and associated risks.
 - **Additional Queries:**
 - Identify high-risk investments made by customers.
 - Calculate the total investment value per customer.
- **ATM**
 - **Attributes:** ATMid, Location, Status (Active/Out of Service), CashAvailable
 - **Purpose:** To manage ATM locations and statuses, including cash availability.
 - **Additional Queries:**
 - Locate ATMs that are running low on cash.
 - Find ATMs in a specific geographic area.
- **Loan Type**
 - **Attributes:** LoanTypeID, Description, TypicalTerm, TypicalInterestRate
 - **Purpose:** To categorize distinct types of loans (e.g., personal, mortgage, auto) and their typical terms.
 - **Additional Queries:**
 - Determine the most popular loan type based on the number of loans issued.
- **Account-Loan Relationship**
 - **Attributes:** AccountID, LoanID

-
- **Purpose:** To link customer accounts to their loans for easy tracking of loan payments and account deductions.
 - **Additional Queries:**
 - List all loans linked to a specific account.
 - **Customer-Investment Relationship**
 - **Attributes:** CustomerID, InvestmentID
 - **Purpose:** To associate customers with their investments.
 - **Additional Queries:**
 - Summarize total investment values by customer.
 - **ATM-Transaction**
 - **Attributes:** ATMID, TransactionID
 - **Purpose:** To track which transactions are made at which ATMs.
 - **Additional Queries:**
 - Analyze the frequency of transactions at each ATM.

Impact of Expansion on Queries

The expansion of the conceptual model allows for a more comprehensive analysis of banking operations. For instance, with the addition of loans and investments, the bank can better understand its financial exposure and customer investment behavior. The integration of ATM data aids in operational management and customer service improvements.

These additional entities and relationships provide a deeper insight into the banking ecosystem, allowing for more sophisticated queries that can inform decision-making, risk management, customer relationship management, and strategic planning. The ability to analyze diverse aspects of banking operations holistically is crucial for a modern, dynamic financial institution.