SYNOPSIS

**1. Introduction**

The banking sector is the backbone of any economy. A reliable and efficient banking system is vital for managing customers, accounts, loans, transactions, and related operations. Our "Banking System" project is a comprehensive database management system (DBMS) built using Oracle SQL. It supports key banking functions such as account creation, deposits, withdrawals, fund transfers, loan management, EMI payments, and employee management. The system focuses on data consistency, integrity, and security while offering scalability and simplicity. By automating these processes, the system improves operational efficiency and minimizes manual intervention.

#### ****2. Objectives****

The primary objectives of the Banking System project are:

1. To maintain structured and secure banking data.
2. To store and manage customer information and contact details.
3. To handle multiple types of bank accounts per customer.
4. To allow secure deposit and withdrawal of money.
5. To enable inter-account fund transfers.
6. To manage customer loan requests and EMI payments.
7. To keep track of all transaction histories.
8. To link each bank branch to a parent bank and employees.
9. To implement and test PL/SQL procedures and functions.
10. To validate business rules such as balance checks before withdrawals.
11. To provide analytical functions like total loan amount by customer.
12. To reduce redundancy and improve data normalization.
13. To ensure proper foreign key relationships between entities.
14. To prepare an extendable and modular design for future improvements.
15. To allow easy data retrieval for decision-making and audits.

#### ****3. Problem Statement****

Banks deal with massive volumes of customer, transaction, and loan data. Managing this data manually or with fragmented systems often leads to inefficiency, data inconsistency, and delayed processing. A centralized, secure, and automated banking database is necessary to reduce these issues. Our project addresses this by building a full-fledged database system for banks that can automate routine operations, reduce human error, and maintain data integrity across branches, customers, and financial services.

#### ****4. Scope of the Project****

The current implementation supports essential banking operations like customer onboarding, bank account management, transaction tracking, and loan processing. In the future, this project can be expanded to include:

* Real-time web or mobile banking interfaces.
* Integration with credit score systems.
* Fraud detection mechanisms.
* Transactional analytics dashboards.
* Multi-level user authentication and access control.
* SMS/email alerts for transactions.
* Investment and insurance services modules.
* Support for international transactions.
* AI-driven customer support.
* Blockchain for secure record-keeping.

#### ****5. Database Design Methodology****

##### **a) Entity-Relationship Modeling**

We identified the core entities such as Bank, Branch, Customer, Bank\_Account, Transaction, Loan, Payment, Card, Employee, and Manage. Each of these entities holds critical data elements related to banking operations. Their relationships are defined with clear primary and foreign keys.

##### **b) Cardinality and Table Structure Analysis**

Each Bank can have multiple Branches; each Branch can serve multiple Customers; a Customer can own multiple Accounts, Loans, or Cards; each Account can have multiple Transactions. Cardinality was analyzed to avoid redundancy and ensure optimized table design.

##### **c) ER-to-Table Conversion**

Each entity from the ER model was converted into a table using SQL CREATE TABLE statements. Primary keys and foreign keys were carefully assigned to maintain referential integrity.

##### **d) Normalization Process**

The schema is normalized to at least the third normal form (3NF). Repeating groups are eliminated, each table has a primary key, and transitive dependencies are resolved. For example, account and branch information are kept in separate tables and linked via foreign keys.

##### **e) PL/SQL Implementation**

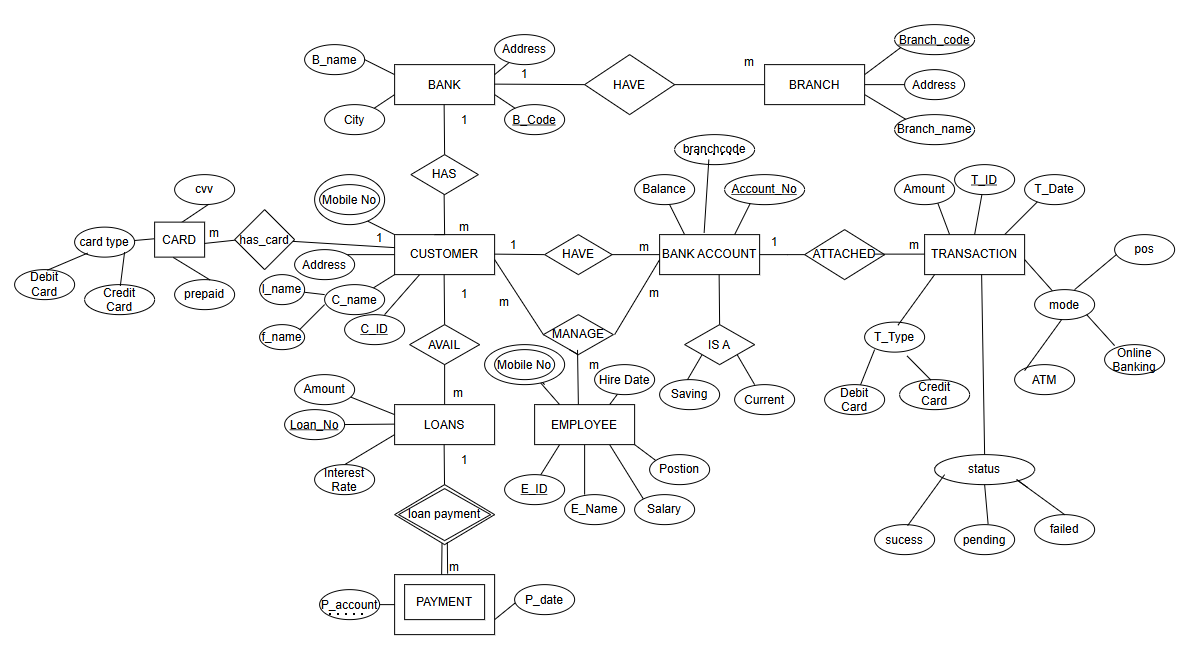
Advanced operations like deposits, withdrawals, transfers, and loan approvals are handled through PL/SQL stored procedures and functions. These scripts ensure that business rules are enforced at the database level (e.g., preventing overdrafts, logging transactions automatically).

#### ****6. Expected Outcome of the Project****

The final Banking System database will:

* Allow efficient data entry, querying, and updating.
* Automate transactional operations through PL/SQL.
* Maintain high data accuracy and reduce redundancy.
* Support all major banking workflows digitally.
* Be scalable for future enhancements or real-time integration.
* Improve performance through relational design and indexing.
* Provide a strong foundation for real-world banking application development.

ER DIAGRAM



ER TO TABLE CONVERSION

1. BANK(B\_Code, City, B\_name, Address)
2. BRANCH(Branch\_code, B\_Code, Branch\_name, Address)
3. CUSTOMERX(C\_ID, Customerx, Address,Mobile\_No)
4. EMPLOYEE(E\_ID, E\_Name,Position, Salary ,Hire\_Date, Branch\_code ,Mobile\_No)
5. BANK\_ACCOUNT(Account\_No, Balance , Branch\_code ,C\_ID, Account\_Type)
6. CARD(Card\_No, Card\_Type, C\_ID)
7. LOANS(Loan\_ID, Amount , Interest\_Rate ,C\_ID, E\_ID)
8. PAYMENT(P\_Account, P\_Date, Loan\_No, Amount)
9. TRANSACTION(T\_ID, Account\_No, T\_Date, Amount, Modex,Status,Card\_No)

ENTITIES AND THEIR ATTRIBUTES &CONSTRAINTS

### ****1. BANK****

* **Primary Key**: B\_Code
* **Attributes**: City, B\_Name, Address

### ****2. BRANCH****

* **Primary Key**: Branch\_code
* **Foreign Key**: B\_Code → BANK(B\_Code)
* **Attributes**: Branch\_name, Address

### ****3. CUSTOMERX****

* **Primary Key**: C\_ID
* **Attributes**: Customerx (Name), Address, Mobile\_No

### ****4. EMPLOYEE****

* **Primary Key**: E\_ID
* **Foreign Key**: Branch\_code → BRANCH(Branch\_code)
* **Attributes**: E\_Name, Position, Salary, Hire\_Date, Mobile\_No

### ****5. BANK\_ACCOUNT****

* **Primary Key**: Account\_No
* **Foreign Keys**:
  + Branch\_code → BRANCH(Branch\_code)
  + C\_ID → CUSTOMERX(C\_ID)
* **Attributes**: Balance, Account\_Type

### ****6. LOANS****

* **Primary Key**: Loan\_No
* **Foreign Keys**:
  + C\_ID → CUSTOMERX(C\_ID)
  + E\_ID → EMPLOYEE(E\_ID) (optional)
* **Attributes**: Amount, Interest\_Rate

### ****7. PAYMENT****

* **Primary Key**: (Composite) P\_Account, P\_Date
* **Foreign Keys**:
  + P\_Account → BANK\_ACCOUNT(Account\_No)
  + Loan\_No → LOANS(Loan\_No)
* **Attributes**: Amount

### ****8. CARD****

* **Primary Key**: Card\_No
* **Foreign Key**: C\_ID → CUSTOMERX(C\_ID)
* **Attributes**: Card\_Type

### ****9. TRANSACTION****

* **Primary Key**: T\_ID
* **Foreign Keys**:
  + Account\_No → BANK\_ACCOUNT(Account\_No)
  + Card\_No → CARD(Card\_No) (optional)
* **Attributes**: T\_Date, Amount, Modex, Status

RELATIONSHIP AND CARDINALITY

**1. BANK – BRANCH**

* **Relationship:** *HAS / HAVE*
* **Cardinality:** One Bank → Many Branches (1:M)
* **Explanation:** A bank can have multiple branches located in different cities, but each branch belongs to exactly one bank.

**2. BANK – CUSTOMER**

* **Relationship:** *HAS*
* **Cardinality:** One Bank → Many Customers (1:M)
* **Explanation:** A bank services many customers, but each customer is associated with one bank in this design

**3. CUSTOMER – BANK ACCOUNT**

* **Relationship:** *HAVE*
* **Cardinality:** One Customer → Many Accounts (1:M)
* **Explanation:** A customer can hold multiple types of accounts (e.g., savings, current), but each account is owned by only one customer.

**4. BANK ACCOUNT – TRANSACTION**

* **Relationship:** *ATTACHED*
* **Cardinality:** One Bank Account → Many Transactions (1:M)
* **Explanation:** Each transaction is linked to one account, while a bank account may have many transactions over time.

**5. CUSTOMER – CARD**

* **Relationship:** *HAS\_CARD*
* **Cardinality:** One Customer → Many Cards (1:M)
* **Explanation:** A customer can own multiple cards (credit, debit, prepaid), but each card belongs to only one customer.

**6. CUSTOMER – LOANS**

* **Relationship:** *AVAIL*
* **Cardinality:** One Customer → Many Loans (1:M)
* **Explanation:** A customer can avail multiple loans, each recorded separately.

**7. LOANS – PAYMENT**

* **Relationship:** *LOAN PAYMENT*
* **Cardinality:** One Loan → Many Payments (1:M)
* **Explanation:** A loan is repaid in installments (EMIs), and each installment is recorded as a payment.

**8. CUSTOMER – EMPLOYEE**

* **Relationship:** *MANAGE*
* **Cardinality:** Many-to-Many (M:N)
* **Explanation:**  
  An employee can manage multiple customers (e.g., as a relationship manager), and a customer may be managed by multiple employees (e.g., for loans, services, account issues).

**9.BANK ACCOUNT – EMPLOYEE**

* **Relationship:** *MANAGE*
* **Cardinality:** Many-to-Many (M:N)
* **Explanation:**  
  Multiple employees (e.g., tellers, auditors, managers) may manage the same bank account at different times or roles, and one employee may manage multiple bank accounts.

NORMALISATION

* 1. **BANK**

**FDs:**  
 B\_Code → B\_name, City, Address

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| B\_Code | VARCHAR2(10) | Primary Key |
| B\_Name | VARCHAR2(100) | NOT NULL |
| City | VARCHAR2(100) | NOT NULL |
| Address | VARCHAR2(200) | NOT NULL |

**1NF**: No multivalued attributes  
**2NF**: Fully dependent on B\_Code (single attribute PK)  
**3NF**: No transitive dependency  
**BCNF**: B\_Code is only determinant

**2. BRANCH**

**FDs:**  
Branch\_code → Branch\_name, Address, B\_Code  
B\_Code → Bank Details (via FK)

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| Branch\_code | VARCHAR2(10) | Primary Key |
| Branch\_name | VARCHAR2(100) | NOT NULL |
| Address | VARCHAR2(200) | NOT NULL |
| B\_Code | VARCHAR2(10) | Foreign Key → BANK(B\_Code) |

**1NF**: Complies  
**2NF**: All attributes dependent on full PK  
**3NF**: No transitive dependencies  
**BCNF**: Satisfies all conditions

**3. CUSTOMERX**

**FDs:**  
C\_ID → Name, Address, Mobile\_No

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| C\_ID | NUMBER | Primary Key |
| Customerx | VARCHAR2(100) | NOT NULL |
| Address | VARCHAR2(200) | NOT NULL |
| Mobile\_No | VARCHAR2(15) | NOT NULL |

**1NF**: No repeating attributes  
**2NF**: All fields dependent on PK  
**3NF**: No non-key attribute depends on another non-key  
**BCNF**: C\_ID is the only determinant

**4. EMPLOYEE**

**FDs:**  
E\_ID → E\_Name, Position, Salary, Hire\_Date, Branch\_code, Mobile\_No

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| E\_ID | NUMBER | Primary Key |
| E\_Name | VARCHAR2(100) | NOT NULL |
| Position | VARCHAR2(50) | NOT NULL |
| Salary | NUMBER | NOT NULL |
| Hire\_Date | DATE | NOT NULL |
| Branch\_code | VARCHAR2(10) | Foreign Key |
| Mobile\_No | VARCHAR2(15) | NOT NULL |

**1NF**: No multivalued fields  
**2NF**: All attributes fully depend on E\_ID  
**3NF**: No transitive dependency  
**BCNF**: E\_ID is the only determinant

**5. BANK\_ACCOUNT**

**FDs:**  
Account\_No → Balance, Branch\_code, C\_ID, Account\_Type

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| Account\_No | NUMBER | Primary Key |
| Balance | NUMBER | NOT NULL |
| Branch\_code | VARCHAR2(10) | Foreign Key |
| C\_ID | NUMBER | Foreign Key |
| Account\_Type | VARCHAR2(20) | CHECK (values IN ‘Saving’, ‘Current’) |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**1NF**: No repeating groups  
**2NF**: Single attribute PK  
**3NF**: No transitive dependency  
**BCNF**: Account\_No is a candidate key and only determinant

**6. LOANS**

**FDs:**  
Loan\_No → Amount, Interest\_Rate, C\_ID, E\_ID

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| Loan\_No | NUMBER | Primary Key |
| Amount | NUMBER | NOT NULL |
| Interest\_Rate | NUMBER(5,2) | NOT NULL |
| C\_ID | NUMBER | Foreign Key |
| E\_ID | NUMBER | Foreign Key |

**1NF**: Complies  
 **2NF**: Single attribute PK  
 **3NF**: No non-prime depends on another  
 **BCNF**: Loan\_No is the only determinant

**7. PAYMENT**

**FDs:**  
(P\_Account, P\_Date) → Loan\_No, Amount

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |  |  |
| P\_Account | NUMBER | Foreign Key |  |  |
| P\_Date | DATE | Composite PK |  |  |
| Loan\_No | NUMBER | Foreign Key |  |  |
| Amount | NUMBER | NOT NULL |  |  |

**1NF**: Complies  
**2NF**: Composite PK with full dependency  
**3NF**: No transitive dependency  
**BCNF**: All determinants are candidate keys

**8. CARD**

**FDs:**  
Card\_No → Card\_Type, C\_ID

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |  |  |  |
| Card\_No | VARCHAR2(20) | Primary Key |  |  |  |
| Card\_Type | VARCHAR2(20) | NOT NULL |  |  |  |
| C\_ID | NUMBER | Foreign Key |  |  |  |

**1NF**: No multivalued attributes  
**2NF**: Complies  
**3NF**: No transitive dependency  
**BCNF**: Card\_No is the only determinant

**9. TRANSACTION**

**FDs:**  
T\_ID → Account\_No, T\_Date, Amount, Modex, Status, Card\_No

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data-type** | **Constraint** |
| T\_ID | NUMBER | Primary Key |
| Account\_No | NUMBER | Foreign Key |
| T\_Date | DATE | NOT NULL |
| Amount | NUMBER | NOT NULL |
| Modex | VARCHAR2(20) | CHECK (ATM, Online, POS) |
| Status | VARCHAR2(10) | CHECK (Success, Pending, Failed) |
| Card\_No | VARCHAR2(20) | Foreign Key (Nullable) |

**1NF**: Complies  
 **2NF**: All fully dependent on T\_ID  
 **3NF**: No non-key transitive dependencies  
 **BCNF**: Satisfies BCNF

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

PL/SQL CODES

1. Create BANK Table

CREATE TABLE BANK (

B\_Code VARCHAR2(10) PRIMARY KEY,

City VARCHAR2(50) NOT NULL,

B\_Name VARCHAR2(100) NOT NULL,

Address VARCHAR2(255)

);

2. Create BRANCH Table

CREATE TABLE BRANCH (

Branch\_code VARCHAR2(10) PRIMARY KEY,

Branch\_name VARCHAR2(100) NOT NULL,

Address VARCHAR2(255),

B\_Code VARCHAR2(10) NOT NULL,

FOREIGN KEY (B\_Code) REFERENCES BANK(B\_Code)

);

3. Create CUSTOMERX Table

CREATE TABLE CUSTOMERX (

C\_ID VARCHAR2(10) PRIMARY KEY,

Customerx VARCHAR2(100) NOT NULL,

Address VARCHAR2(255),

Mobile\_No VARCHAR2(15) UNIQUE

);

4. Create EMPLOYEE Table

CREATE TABLE EMPLOYEE (

E\_ID VARCHAR2(10) PRIMARY KEY,

E\_Name VARCHAR2(100) NOT NULL,

Position VARCHAR2(50),

Salary NUMBER(10, 2),

Hire\_Date DATE,

Branch\_code VARCHAR2(10) NOT NULL,

Mobile\_No VARCHAR2(15) UNIQUE,

FOREIGN KEY (Branch\_code) REFERENCES BRANCH(Branch\_code)

);

5. Create BANK\_ACCOUNT Table

CREATE TABLE BANK\_ACCOUNT (

Account\_No VARCHAR2(20) PRIMARY KEY,

Balance NUMBER(15, 2) DEFAULT 0 CHECK (Balance >= 0),

Branch\_code VARCHAR2(10) NOT NULL,

C\_ID VARCHAR2(10) NOT NULL,

Account\_Type VARCHAR2(20) CHECK (Account\_Type IN ('Saving', 'Current', 'Fixed Deposit')),

FOREIGN KEY (Branch\_code) REFERENCES BRANCH(Branch\_code),

FOREIGN KEY (C\_ID) REFERENCES CUSTOMERX(C\_ID) -- Changed to reference CUSTOMERX

);

6. Create LOANS Table

CREATE TABLE LOANS (

Loan\_No VARCHAR2(10) PRIMARY KEY,

Amount NUMBER(15, 2) NOT NULL CHECK (Amount > 0),

Interest\_Rate NUMBER(5, 4) NOT NULL CHECK (Interest\_Rate > 0),

C\_ID VARCHAR2(10) NOT NULL,

E\_ID VARCHAR2(10), -- Assuming an employee might be assigned to a loan, but not strictly mandatory based on ER

FOREIGN KEY (C\_ID) REFERENCES CUSTOMERX(C\_ID), -- Changed to reference CUSTOMERX

FOREIGN KEY (E\_ID) REFERENCES EMPLOYEE(E\_ID)

);

7. Create PAYMENT Table

CREATE TABLE PAYMENT (

P\_Account VARCHAR2(20),

P\_Date DATE,

Loan\_No VARCHAR2(10) NOT NULL,

Amount NUMBER(15, 2) NOT NULL CHECK (Amount > 0),

PRIMARY KEY (P\_Account, P\_Date), -- Composite primary key

FOREIGN KEY (P\_Account) REFERENCES BANK\_ACCOUNT(Account\_No),

FOREIGN KEY (Loan\_No) REFERENCES LOANS(Loan\_No)

);

8. Create CARD Table

CREATE TABLE CARD (

Card\_No VARCHAR2(20) PRIMARY KEY,

Card\_Type VARCHAR2(20) CHECK (Card\_Type IN ('Debit Card', 'Credit Card', 'Prepaid')),

C\_ID VARCHAR2(10) NOT NULL,

FOREIGN KEY (C\_ID) REFERENCES CUSTOMERX(C\_ID) -- Changed to reference CUSTOMERX

);

9. Create TRANSACTION Table

CREATE TABLE TRANSACTION (

T\_ID VARCHAR2(20) PRIMARY KEY,

Account\_No VARCHAR2(20) NOT NULL,

T\_Date DATE NOT NULL,

Amount NUMBER(15, 2) NOT NULL CHECK (Amount >= 0),

Modex VARCHAR2(50),

Status VARCHAR2(20) CHECK (Status IN ('success', 'failed', 'pending')),

Card\_No VARCHAR2(20), -- Optional, as not all transactions use a card

FOREIGN KEY (Account\_No) REFERENCES BANK\_ACCOUNT(Account\_No),

FOREIGN KEY (Card\_No) REFERENCES CARD(Card\_No)

);

Functions and Procedures

1. Function to get Account Balance

CREATE OR REPLACE FUNCTION GET\_ACCOUNT\_BALANCE (

p\_account\_no IN VARCHAR2

)

RETURN NUMBER

IS

v\_balance NUMBER(15, 2);

BEGIN

SELECT Balance

INTO v\_balance

FROM BANK\_ACCOUNT

WHERE Account\_No = p\_account\_no;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account number ' || p\_account\_no || ' not found.');

RETURN NULL;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

RAISE;

END GET\_ACCOUNT\_BALANCE;

/

2. Procedure for Deposit

CREATE OR REPLACE PROCEDURE DEPOSIT\_TO\_ACCOUNT (

p\_account\_no IN VARCHAR2,

p\_amount IN NUMBER,

p\_mode IN VARCHAR2

)

IS

v\_current\_balance NUMBER(15, 2);

v\_t\_id VARCHAR2(20);

BEGIN

IF p\_amount <= 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Deposit amount must be positive.');

RETURN;

END IF;

SELECT Balance INTO v\_current\_balance

FROM BANK\_ACCOUNT

WHERE Account\_No = p\_account\_no

FOR UPDATE OF Balance;

UPDATE BANK\_ACCOUNT

SET Balance = Balance + p\_amount

WHERE Account\_No = p\_account\_no;

SELECT 'TRANS' || TO\_CHAR(SYSDATE, 'YYYYMMDDHH24MISS') || DBMS\_RANDOM.STRING('U', 5) INTO v\_t\_id FROM dual;

INSERT INTO TRANSACTION (T\_ID, Account\_No, T\_Date, Amount, Modex, Status)

VALUES (v\_t\_id, p\_account\_no, SYSDATE, p\_amount, p\_mode, 'success');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Deposit of ' || p\_amount || ' to account ' || p\_account\_no || ' successful.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account number ' || p\_account\_no || ' not found.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred during deposit: ' || SQLERRM);

ROLLBACK;

RAISE;

END DEPOSIT\_TO\_ACCOUNT;

/

3. Procedure for Withdrawal

CREATE OR REPLACE PROCEDURE WITHDRAW\_FROM\_ACCOUNT (

p\_account\_no IN VARCHAR2,

p\_amount IN NUMBER,

p\_mode IN VARCHAR2,

p\_card\_no IN VARCHAR2 DEFAULT NULL

)

IS

v\_current\_balance NUMBER(15, 2);

v\_t\_id VARCHAR2(20);

BEGIN

IF p\_amount <= 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Withdrawal amount must be positive.');

RETURN;

END IF;

SELECT Balance INTO v\_current\_balance

FROM BANK\_ACCOUNT

WHERE Account\_No = p\_account\_no

FOR UPDATE OF Balance;

IF v\_current\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_account\_no);

SELECT 'TRANS' || TO\_CHAR(SYSDATE, 'YYYYMMDDHH24MISS') || DBMS\_RANDOM.STRING('U', 5) INTO v\_t\_id FROM dual;

INSERT INTO TRANSACTION (T\_ID, Account\_No, T\_Date, Amount, Modex, Status, Card\_No)

VALUES (v\_t\_id, p\_account\_no, SYSDATE, p\_amount, p\_mode, 'failed', p\_card\_no);

COMMIT;

RETURN;

END IF;

UPDATE BANK\_ACCOUNT

SET Balance = Balance - p\_amount

WHERE Account\_No = p\_account\_no;

SELECT 'TRANS' || TO\_CHAR(SYSDATE, 'YYYYMMDDHH24MISS') || DBMS\_RANDOM.STRING('U', 5) INTO v\_t\_id FROM dual;

INSERT INTO TRANSACTION (T\_ID, Account\_No, T\_Date, Amount, Modex, Status, Card\_No)

VALUES (v\_t\_id, p\_account\_no, SYSDATE, p\_amount, p\_mode, 'success', p\_card\_no);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Withdrawal of ' || p\_amount || ' from account ' || p\_account\_no || ' successful.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account number ' || p\_account\_no || ' not found.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred during withdrawal: ' || SQLERRM);

ROLLBACK;

RAISE;

END WITHDRAW\_FROM\_ACCOUNT;

/

4. Procedure to Create a New Customer

CREATE OR REPLACE PROCEDURE CREATE\_NEW\_CUSTOMER (

p\_c\_id IN VARCHAR2,

p\_customerx\_name IN VARCHAR2,

p\_address IN VARCHAR2,

p\_mobile\_no IN VARCHAR2

)

IS

BEGIN

INSERT INTO CUSTOMERX (C\_ID, Customerx, Address, Mobile\_No)

VALUES (p\_c\_id, p\_customerx\_name, p\_address, p\_mobile\_no);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('New customer ' || p\_customerx\_name || ' with ID ' || p\_c\_id || ' created successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID or Mobile Number already exists.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred during customer creation: ' || SQLERRM);

ROLLBACK;

RAISE;

END CREATE\_NEW\_CUSTOMER;

/

5. Procedure to Create a New Bank Account

CREATE OR REPLACE PROCEDURE CREATE\_NEW\_ACCOUNT (

p\_account\_no IN VARCHAR2,

p\_branch\_code IN VARCHAR2,

p\_c\_id IN VARCHAR2,

p\_account\_type IN VARCHAR2

)

IS

v\_customer\_exists NUMBER;

v\_branch\_exists NUMBER;

BEGIN

SELECT COUNT(\*)

INTO v\_customer\_exists

FROM CUSTOMERX

WHERE C\_ID = p\_c\_id;

IF v\_customer\_exists = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_c\_id || ' does not exist.');

RETURN;

END IF;

SELECT COUNT(\*)

INTO v\_branch\_exists

FROM BRANCH

WHERE Branch\_code = p\_branch\_code;

IF v\_branch\_exists = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Branch with code ' || p\_branch\_code || ' does not exist.');

RETURN;

END IF;

INSERT INTO BANK\_ACCOUNT (Account\_No, Balance, Branch\_code, C\_ID, Account\_Type)

VALUES (p\_account\_no, 0, p\_branch\_code, p\_c\_id, p\_account\_type);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('New account ' || p\_account\_no || ' created successfully for customer ' || p\_c\_id || ' at branch ' || p\_branch\_code);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account number ' || p\_account\_no || ' already exists.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred during account creation: ' || SQLERRM);

ROLLBACK;

RAISE;

END CREATE\_NEW\_ACCOUNT;

/

-- Trigger to update the account balance after a transaction is inserted

-- This trigger fires AFTER each row INSERTED into the TRANSACTION table

CREATE OR REPLACE TRIGGER TRG\_UPDATE\_ACCOUNT\_BALANCE

AFTER INSERT ON TRANSACTION

FOR EACH ROW

WHEN (NEW.Status = 'success')

DECLARE

v\_account\_type VARCHAR2(20);

BEGIN

SELECT Account\_Type

INTO v\_account\_type

FROM BANK\_ACCOUNT

WHERE Account\_No = :NEW.Account\_No;

IF :NEW.Amount > 0 THEN

UPDATE BANK\_ACCOUNT

SET Balance = Balance + :NEW.Amount

WHERE Account\_No = :NEW.Account\_No;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error in trigger: Account ' || :NEW.Account\_No || ' not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in trigger: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'Error updating account balance: ' || SQLERRM);

END TRG\_UPDATE\_ACCOUNT\_BALANCE;

/

-- Trigger to prevent withdrawal if account balance is insufficient

-- This trigger fires BEFORE each row INSERTED into the TRANSACTION table

CREATE OR REPLACE TRIGGER TRG\_PREVENT\_INSUFFICIENT\_WITHDRAWAL

BEFORE INSERT ON TRANSACTION

FOR EACH ROW

WHEN (NEW.Modex = 'Withdrawal')

DECLARE

v\_current\_balance BANK\_ACCOUNT.Balance%TYPE;

BEGIN

SELECT Balance

INTO v\_current\_balance

FROM BANK\_ACCOUNT

WHERE Account\_No = :NEW.Account\_No;

IF v\_current\_balance < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds for withdrawal from account ' || :NEW.Account\_No);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Account number ' || :NEW.Account\_No || ' not found for withdrawal.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20004, 'An unexpected error occurred during withdrawal check: ' || SQLERRM);

END TRG\_PREVENT\_INSUFFICIENT\_WITHDRAWAL;

/

-- PL/SQL block using a cursor to apply interest to savings accounts

DECLARE

CURSOR savings\_accounts\_cur IS

SELECT Account\_No, Balance

FROM BANK\_ACCOUNT

WHERE Account\_Type = 'Saving';

v\_account\_no BANK\_ACCOUNT.Account\_No%TYPE;

v\_balance BANK\_ACCOUNT.Balance%TYPE;

v\_interest\_rate NUMBER := 0.005;

v\_interest\_amount NUMBER(15, 2);

BEGIN

OPEN savings\_accounts\_cur;

LOOP

FETCH savings\_accounts\_cur INTO v\_account\_no, v\_balance;

EXIT WHEN savings\_accounts\_cur%NOTFOUND;

v\_interest\_amount := v\_balance \* v\_interest\_rate;

UPDATE BANK\_ACCOUNT

SET Balance = Balance + v\_interest\_amount

WHERE Account\_No = v\_account\_no;

DBMS\_OUTPUT.PUT\_LINE('Applied interest of ' || v\_interest\_amount || ' to account ' || v\_account\_no);

END LOOP;

CLOSE savings\_accounts\_cur;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest application process completed.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred during interest application: ' || SQLERRM);

ROLLBACK;

RAISE;

END;

/

-- PL/SQL block using a cursor to generate an account statement

DECLARE

p\_account\_no BANK\_ACCOUNT.Account\_No%TYPE := 'ACC1001';

p\_start\_date DATE := TO\_DATE('2025-05-01', 'YYYY-MM-DD');

p\_end\_date DATE := TO\_DATE('2025-05-31', 'YYYY-MM-DD');

CURSOR account\_statement\_cur IS

SELECT

T\_ID,

T\_Date,

Amount,

Modex,

Status,

Card\_No

FROM

TRANSACTION

WHERE

Account\_No = p\_account\_no

AND T\_Date BETWEEN p\_start\_date AND p\_end\_date

ORDER BY

T\_Date, T\_ID;

v\_t\_id TRANSACTION.T\_ID%TYPE;

v\_t\_date TRANSACTION.T\_Date%TYPE;

v\_amount TRANSACTION.Amount%TYPE;

v\_mode TRANSACTION.Modex%TYPE;

v\_status TRANSACTION.Status%TYPE;

v\_card\_no TRANSACTION.Card\_No%TYPE;

v\_current\_balance BANK\_ACCOUNT.Balance%TYPE;

BEGIN

DBMS\_OUTPUT.ENABLE(NULL);

DBMS\_OUTPUT.PUT\_LINE('--- Account Statement for Account: ' || p\_account\_no || ' ---');

DBMS\_OUTPUT.PUT\_LINE('Period: ' || TO\_CHAR(p\_start\_date, 'YYYY-MM-DD') || ' to ' || TO\_CHAR(p\_end\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE(RPAD('T\_ID', 10) || ' | ' || RPAD('Date', 10) || ' | ' || RPAD('Amount', 10) || ' | ' || RPAD('Mode', 20) || ' | ' || RPAD('Status', 10) || ' | ' || RPAD('Card\_No', 15));

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------------');

OPEN account\_statement\_cur;

LOOP

FETCH account\_statement\_cur INTO v\_t\_id, v\_t\_date, v\_amount, v\_mode, v\_status, v\_card\_no;

EXIT WHEN account\_statement\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(

RPAD(v\_t\_id, 10) || ' | ' ||

TO\_CHAR(v\_t\_date, 'YYYY-MM-DD') || ' | ' ||

RPAD(TO\_CHAR(v\_amount, 'FM9999999.00'), 10) || ' | ' ||

RPAD(v\_mode, 20) || ' | ' ||

RPAD(v\_status, 10) || ' | ' ||

RPAD(NVL(v\_card\_no, 'N/A'), 15)

);

END LOOP;

CLOSE account\_statement\_cur;

BEGIN

SELECT Balance

INTO v\_current\_balance

FROM BANK\_ACCOUNT

WHERE Account\_No = p\_account\_no;

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('Current Balance as of ' || TO\_CHAR(SYSDATE, 'YYYY-MM-DD') || ': ' || TO\_CHAR(v\_current\_balance, 'FM99999999.00'));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account number ' || p\_account\_no || ' not found for final balance.');

END;

DBMS\_OUTPUT.PUT\_LINE('--- End of Statement ---');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account number ' || p\_account\_no || ' not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred during statement generation: ' || SQLERRM);

IF account\_statement\_cur%ISOPEN THEN

CLOSE account\_statement\_cur;

END IF;

RAISE;

END;

/

This view is intended to be queried by filtering on C\_ID to show a specific customer's data.

CREATE OR REPLACE VIEW CUSTOMER\_FULL\_DETAILS\_V AS

SELECT

c.C\_ID,

c.Customerx AS Customer\_Name,

c.Address AS Customer\_Address,

c.Mobile\_No AS Customer\_Mobile,

ba.Account\_No,

ba.Balance AS Account\_Balance,

ba.Account\_Type,

b.Branch\_code,

b.Branch\_name,

b.Address AS Branch\_Address,

bk.B\_Name AS Bank\_Name,

bk.City AS Bank\_City,

t.T\_ID AS Transaction\_ID,

t.T\_Date AS Transaction\_Date,

t.Amount AS Transaction\_Amount,

t.Modex AS Transaction\_Mode,

t.Status AS Transaction\_Status,

t.Card\_No AS Transaction\_Card\_No,

card.Card\_Type AS Transaction\_Card\_Type

FROM

CUSTOMERX c

JOIN

BANK\_ACCOUNT ba ON c.C\_ID = ba.C\_ID

JOIN

BRANCH b ON ba.Branch\_code = b.Branch\_code

JOIN

BANK bk ON b.B\_Code = bk.B\_Code

LEFT JOIN

TRANSACTION t ON ba.Account\_No = t.Account\_No

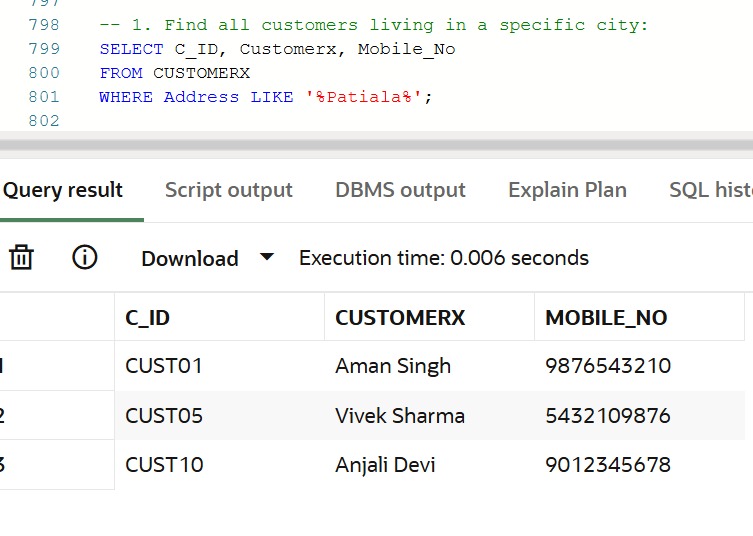
LEFT JOIN

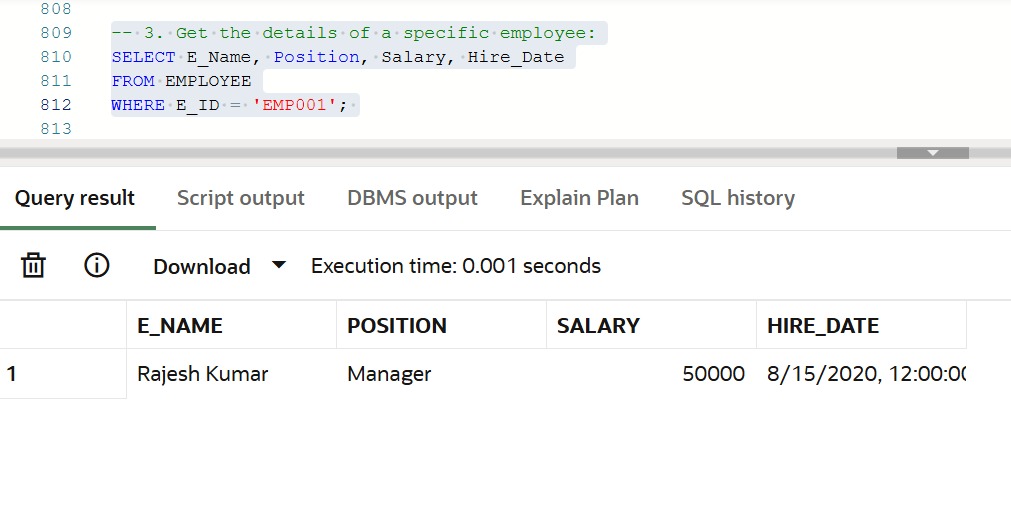
CARD card ON t.Card\_No = card.Card\_No;

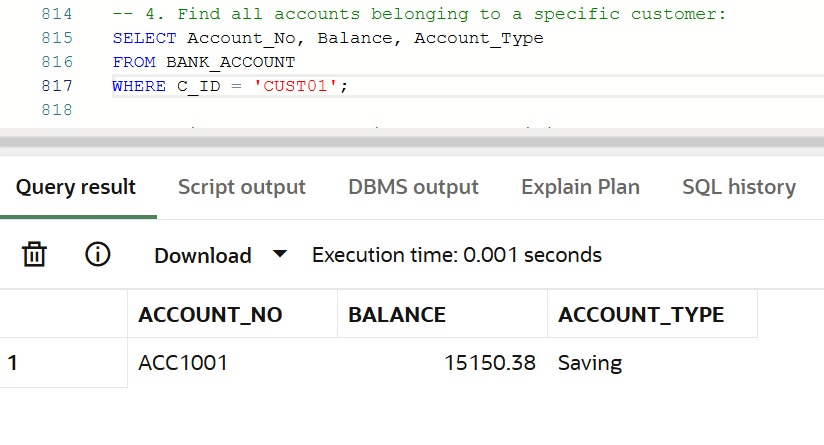
SELECT \* FROM CUSTOMER\_FULL\_DETAILS\_V WHERE C\_ID = 'CUST01';

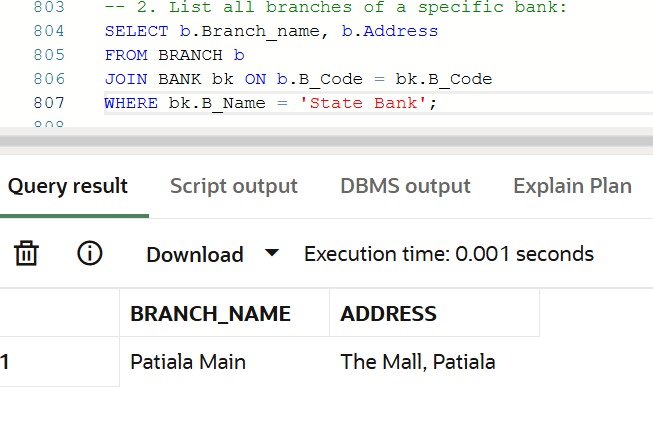
 

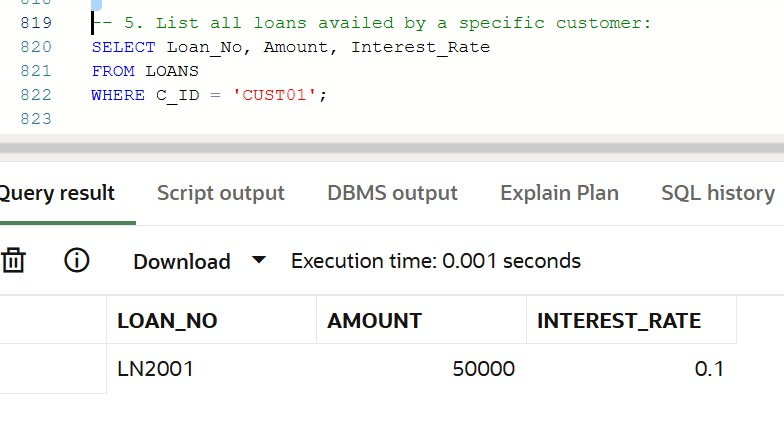
SOME SAMPLE QUERIES WITH OUTPUT

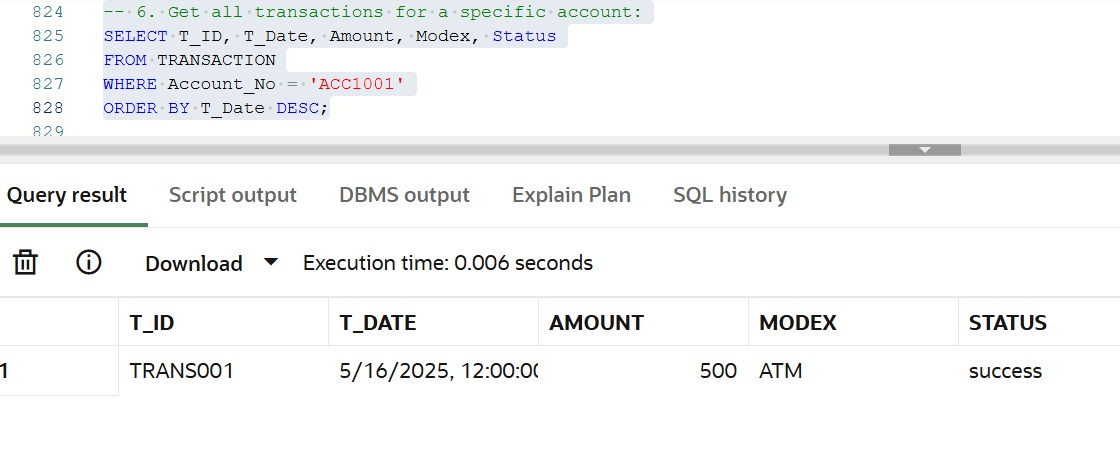


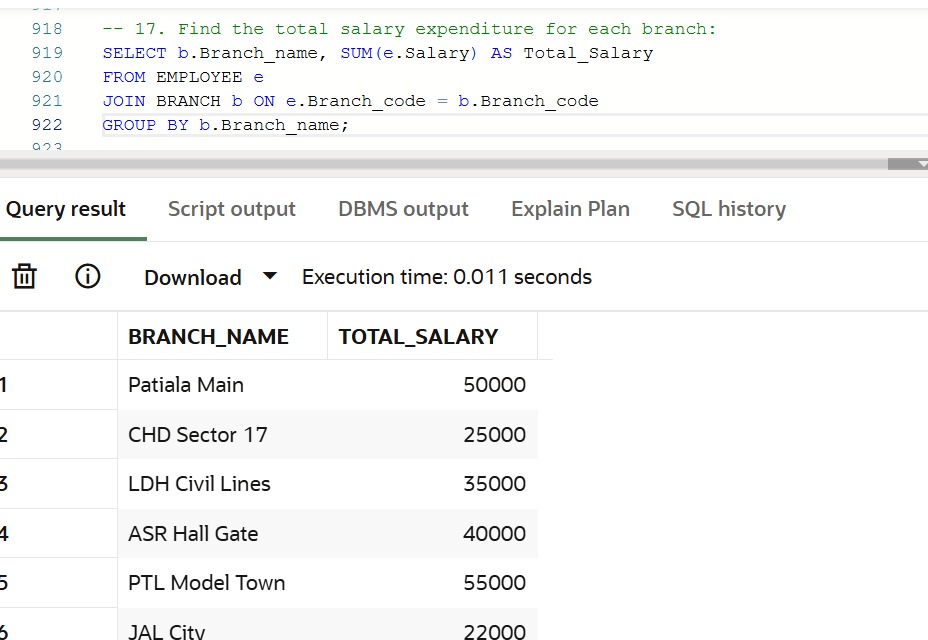


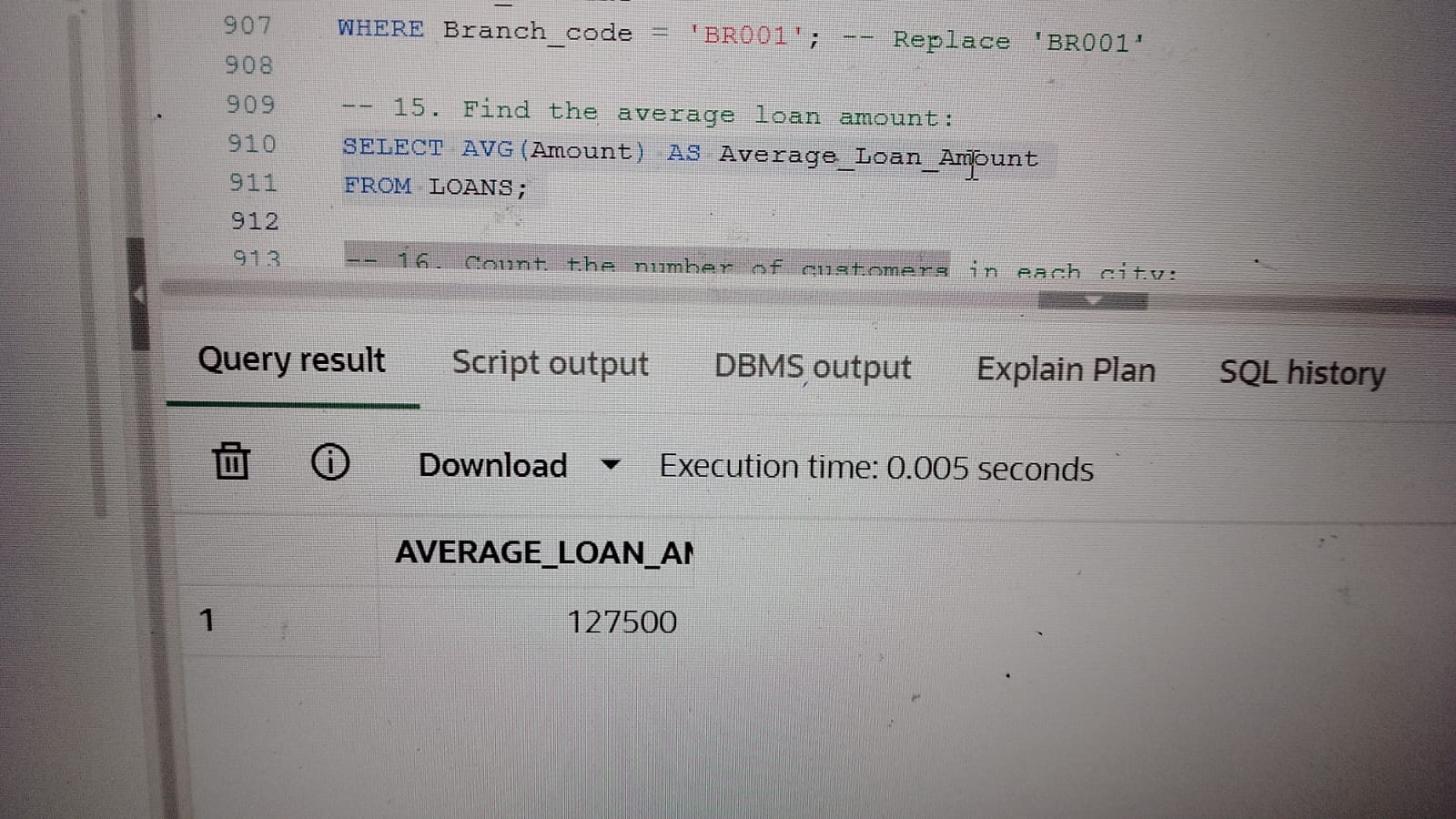


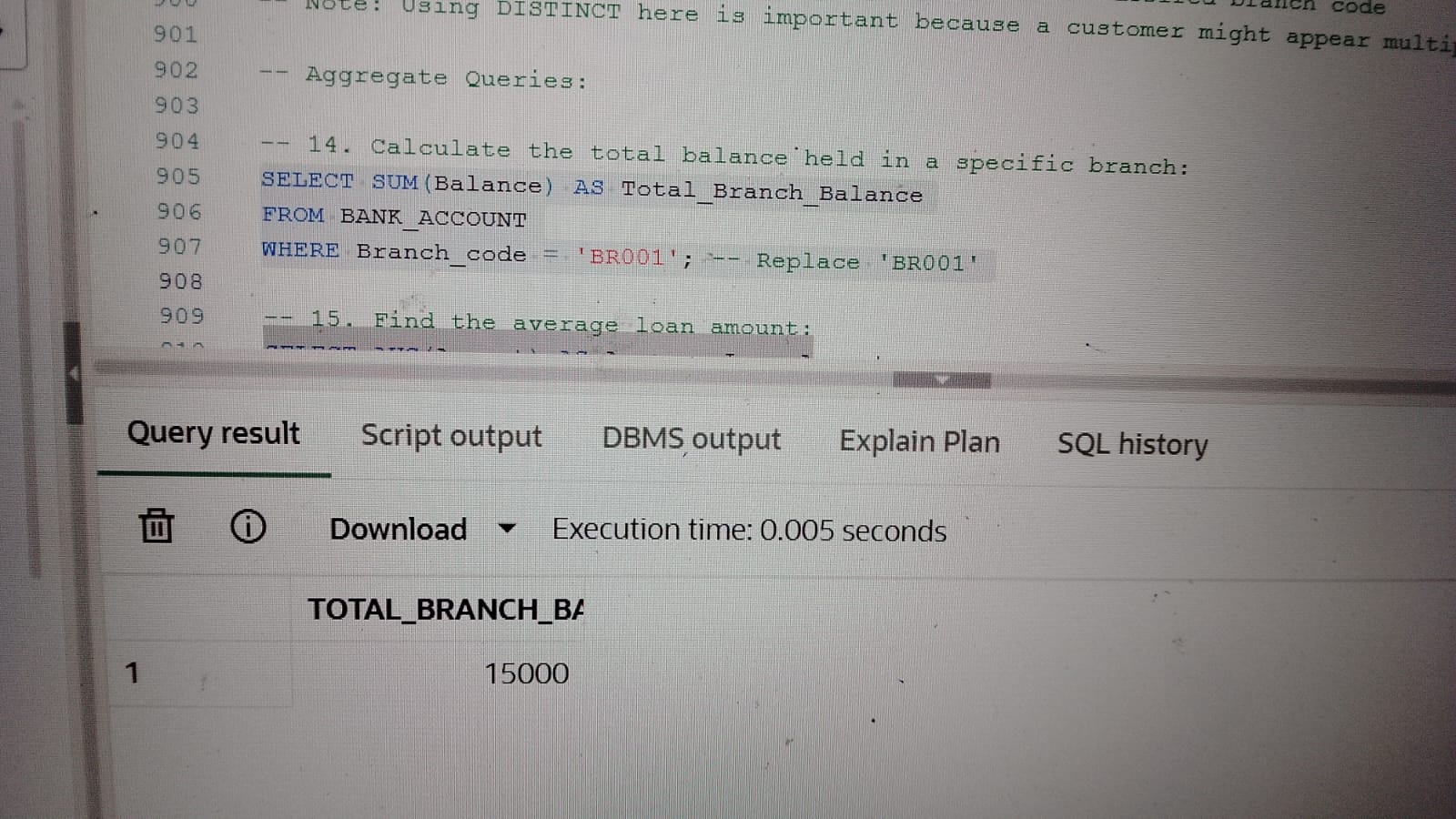












REFERENCES

* Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education.
* Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of Database Systems* (7th ed.). Pearson Education.
* Date, C. J. (2003). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.
* Oracle. (n.d.). *PL/SQL Language Reference*. Oracle Documentation.  
  <https://docs.oracle.com/en/database/oracle/oracle-database/>
* W3Schools. (n.d.). *SQL Tutorial*.  
  <https://www.w3schools.com/sql/>
* GeeksforGeeks. (n.d.). *SQL Tutorials and Examples*.  
  <https://www.geeksforgeeks.org/sql-tutorial/>
* Oracle Live SQL. (n.d.). *Oracle Cloud Live SQL Environment*.  
  <https://livesql.oracle.com/>