SVKM's NMIMS
School of Technology Management & Engineering, Shirpur
A.Y. 2023 - 24
Course: Database Management Systems

Project Report

Program	B. Tech CE	
Semester	4th	
Name of the Project:	Bank Management System	
Details of Project Members		
Batch	Roll No.	Name
A1	B223	Lakhan Agrawal
A1	B218	Kaushal Prajapat
A1	B228	Ayush Tiwari
A1	B201	Krushna Patil
Date of Submission: 14/03/2024		

Contribution of each project Members:

Roll No.	Name:	Contribution
B223	Lakhan Agrawal	COMPLETE
B201	Krushna Patil	COMPLETE
B228	Ayush Tiwari	COMPLETE
B218	Kaushal Prajapat	COMPLETE

Project Report

Selected Topic:-

BANK MANAGEMENT SYSTEM

by

Lakhan Agrawal, Roll number: B223

Ayush Tiwari, Roll number: B228

Krushna Patil, Roll number: B201

Kaushal Prajapat, Roll number: B218

Course: DBMS

AY: 2023-24

Table of Contents

I. Storyline

Bank Database Management System The purpose of this project is to create a database management system for a bank. Oracle 11g has been used as the underlying database. The DDL and DML statements have been written using Oracle PL/SQL developer. This project intends to provide a simplistic approach towards designing a database for a banking system. In any banking database system the most important relationship is that of the customer to the account. In general there are several categories and types of accounts a customer can have. For this project I have taken into consideration the following types of accounts: -

- Checking
- Saving

As a part of this project the essential details of the customer are to be captured like name, address, date of birth, phone number, email, etc. A customer id is assigned to every customer who wants to be a part of the banking system. This customer id acts as a key to the other details related to the customer such as account information, loan information and credit card information. The accounts table carries important information about the account such as the balance in the account and the type of the account. Each of the entries of the account carry a customer id associated with the account to form a correct relation. Going with the industry standard I have chosen to include the details of any credit card or loan that has been issued to the customer as a part of the banking relationship. As with every customer most of the accounts are opened at a branch so it is logical to have a relationship between a customer and a branch. Every branch can have one or many customers associated with it. This makes it easier for the administrator or a bank employee to look up and track the branch where an account was first opened. This project encompasses a complete view of how a banking system database would look like. All the tables mentioned above intend to capture and retain important information related to the account and the customer. SQL queries can then be used to query the details of customers, customer phone numbers, accounts, loans or credit cards.

II. Components of Database Design

- **Table:** Customer
- **Attributes: (**PK) Customer_Id , First_Name , Last_Name , Birth_Date , Street_Number , Street_Name , Unit , City , State , Zip_Code , Email_id , Social_Security_Number , (F) , Branch_Code , Credit_Card , Loan

- **Table:** Branch
- **Attributes: (**PK) Branch_code , Head_office , Branch_details , City_names

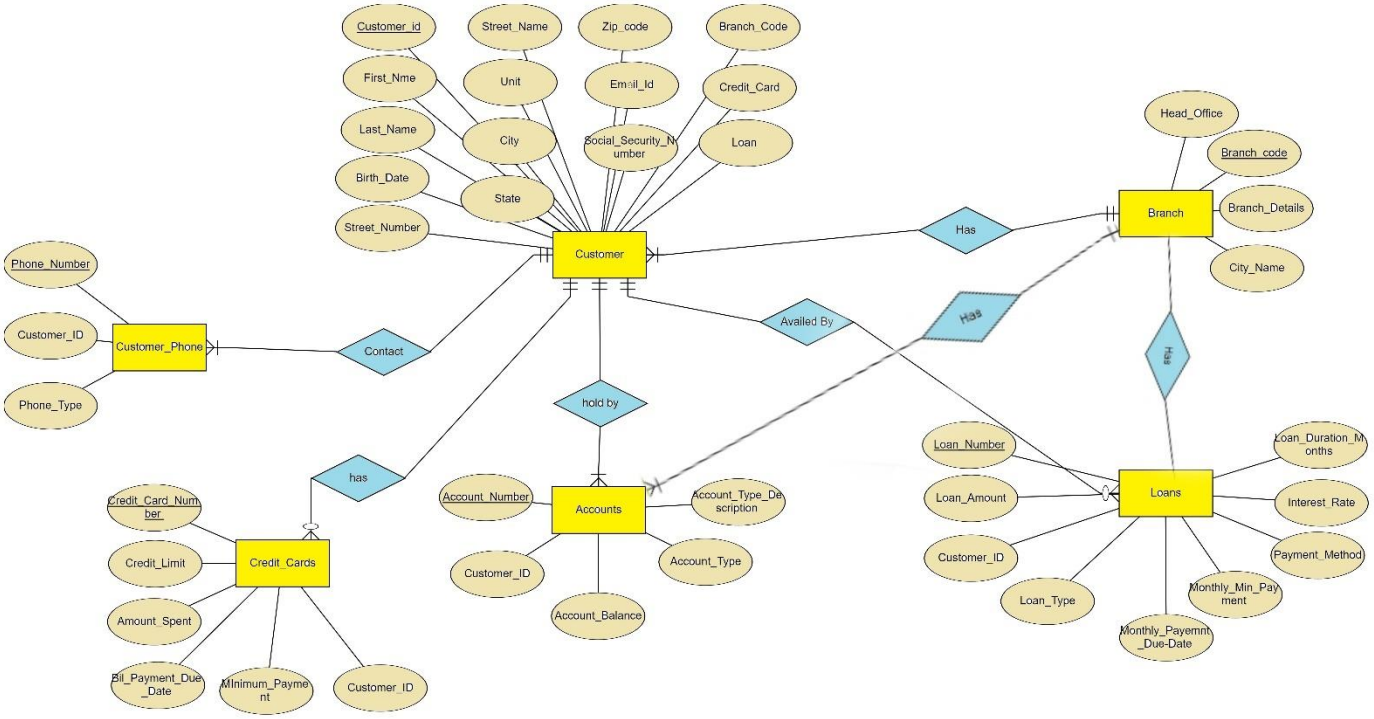
- **Table:** Customer_Phone
- **Attributes: (**PK) Phone_Number , (FK) Customer_id , Phone_type

- **Table:** Accounts
- **Attributes: (**PK) account_Number , (FK) Customer_id , Account_Balance , Account_Type , Account_Type_Descripition

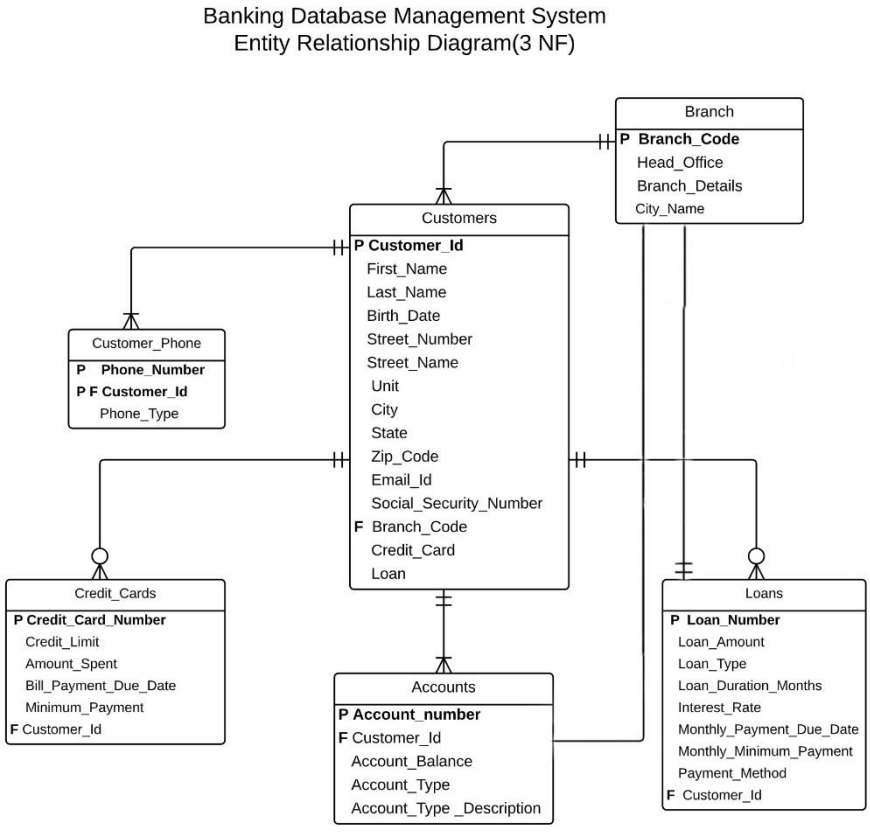
- **Table:** Loans
- **Attributes: (**PK) Loan_Number , Loan_Amount , Loan_Type , Loan_Duration_Months , Intrest_Rate , Monthly_Payment_due_date , Monthly_Minimum_Payment , Payment_Method , (FK) Customer_Id

- **Table:** Credit_Cards
- **Attributes: (**PK) Credit Card Number , Credit_Limit , Amount_spent , Bill_Payment_due_date , Minimum_Payment , (FK) Customer_id

III. Entity Relationship Diagram



IV. Relational Model



V. Normalization

1NF :-

Now, let's check each table:

1. **Branch Table**:
- Each attribute appears to hold atomic values.
- All columns have unique names.
- No repeating groups.
- **Conclusion**: The Branch table is in 1NF.

2. **Customers Table**:
- Most attributes hold atomic values, but address-related attributes like street_number and street_name could potentially be broken down further.
- All columns have unique names.
- No repeating groups.
- **Conclusion**: The Customers table is in 1NF, but further normalization might be needed for address details.

3. **Customer_Phone Table**:
- Each attribute holds atomic values.
- All columns have unique names.
- No repeating groups.
- **Conclusion**: The Customer_Phone table is in 1NF.

4. **Accounts Table**:
- Each attribute holds atomic values.
- All columns have unique names.
- No repeating groups.
- **Conclusion**: The Accounts table is in 1NF.

5. **Credit_Cards Table**:
- Each attribute holds atomic values.
- All columns have unique names.
- No repeating groups.
- **Conclusion**: The Credit_Cards table is in 1NF.

6. **Loans Table**:
- Each attribute holds atomic values.
- All columns have unique names.
- No repeating groups.
- **Conclusion**: The Loans table is in 1NF.

All tables appear to meet the requirements of the First Normal Form (1NF)

2NF:-

To check if the tables satisfy the Second Normal Form (2NF), we need to ensure that they meet the following criteria:

They must already be in 1NF.

All non-key attributes (columns) must be fully functionally dependent on the primary key.
Let's evaluate each table against these criteria:

1. Branch Table:
   - Already in 1NF.
   - All attributes are functionally dependent on the primary key (branch_code).
   - Conclusion: The Branch table is in 2NF.

2. Customers Table:
   - Already in 1NF.
   - Non-key attributes seem to depend on the whole primary key (customer_id), except for the branch_code.
   - Conclusion: The Customers table is in 2NF.

3. Customer_Phone Table:
   - Already in 1NF.
   - All attributes are functionally dependent on the primary key (phone_number, customer_id).
   - Conclusion: The Customer_Phone table is in 2NF.

4. Accounts Table:
   - Already in 1NF.
   - All attributes are functionally dependent on the primary key (account_number).
   - Conclusion: The Accounts table is in 2NF.

5. Credit_Cards Table:
   - Already in 1NF.
   - All attributes are functionally dependent on the primary key (credit_card_number).
   - Conclusion: The Credit_Cards table is in 2NF.

6. Loans Table:
    - Already in 1NF.
    - All attributes are functionally dependent on the primary key (loan_number).
    - Conclusion: The Loans table is in 2NF.

In summary, all tables appear to meet the requirements of the Second Normal Form (2NF) as they are already in 1NF, and all non-key attributes are fully functionally dependent on the primary key.

3NF:

To check if the tables satisfy the Third Normal Form (3NF), we need to ensure that they meet the following criteria:

They must already be in 2NF.
No transitive dependencies should exist; that is, non-key attributes should not depend on other non-key attributes.
Let's evaluate each table against these criteria:

1. **Branch Table**:
    - Already in 2NF.
    - All attributes seem directly related to the primary key (branch_code).
    - **Conclusion**: The Branch table is in 3NF.

2. **Customers Table**:
    - Already in 2NF.
    - Non-key attributes appear to depend only on the primary key (customer_id).
    - **Conclusion**: The Customers table is in 3NF.

3. **Customer_Phone Table**:
    - Already in 2NF.
    - All attributes are directly related to the primary key (phone_number, customer_id).
    - **Conclusion**: The Customer_Phone table is in 3NF.

4. **Accounts Table**:
    - Already in 2NF.
    - All attributes are directly related to the primary key (account_number).
    - **Conclusion**: The Accounts table is in 3NF.

5. **Credit_Cards Table**:
   - Already in 2NF.
   - All attributes are directly related to the primary key (credit_card_number).
   - **Conclusion**: The Credit_Cards table is in 3NF.

6. **Loans Table**:
   - Already in 2NF.
   - All attributes are directly related to the primary key (loan_number).
   - **Conclusion**: The Loans table is in 3NF.

In summary, all tables appear to meet the requirements of the Third Normal Form (3NF) as they are already in 2NF, and there are no transitive dependencies present within the tables.

BCNF:

To check if the tables satisfy the Boyce-Codd Normal Form (BCNF), we need to ensure that they meet the following criteria:

They must already be in 3NF.
Every determinant must be a candidate key.
Let's evaluate each table against these criteria:

1. **Branch Table**:
   - Already in 3NF.
   - The determinant (branch_code) is a candidate key.
   - **Conclusion**: The Branch table is in BCNF.

2. **Customers Table**:
   - Already in 3NF.
   - The determinant (customer_id) is a candidate key.
   - **Conclusion**: The Customers table is in BCNF.

3. **Customer_Phone Table**:
   - Already in 3NF.
   - The determinant (phone_number, customer_id) is a candidate key.
   - **Conclusion**: The Customer_Phone table is in BCNF.

4. **Accounts Table**:
    - Already in 3NF.
    - The determinant (account_number) is a candidate key.
    - **Conclusion**: The Accounts table is in BCNF.

5. **Credit_Cards Table**:
    - Already in 3NF.
    - The determinant (credit_card_number) is a candidate key.
    - **Conclusion**: The Credit_Cards table is in BCNF.

6. **Loans Table**:
    - Already in 3NF.
    - The determinant (loan_number) is a candidate key.
    - **Conclusion**: The Loans table is in BCNF.

In summary, all tables appear to meet the requirements of the Boyce-Codd Normal Form (BCNF) as they are already in 3NF, and every determinant is a candidate key.
Top of Form

VI. SQL Queries

1. Different types of accounts that have been opened up by customers at the bank. Also get the maximum balance with respect to each of the account types and the customer details with respect to those accounts.

```
SQL>
SQL> SELECT cc.customer_id, cc.first_name, cc.last_name, acc.account_balance,
  2  acc.account_number,acc.account_type
  3  FROM accounts acc, customers cc
  4  WHERE account_balance IN (
  5                            SELECT MAX(a.account_balance) as max_balance
  6                            FROM accounts a
  7                            GROUP BY a.account_type)
  8            AND cc.customer_id = acc.customer_id;

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME                                          ACCOUNT_BALANCE
-------------------------------------------------- ---------------
ACCOUNT_NUMBER ACCOUNT_TY
-------------- ----------
         9 Ravi
Yadav                                                      180000
       1000009 Savings

         6 Swati
Mishra                                                    150000.5
       1000006 Checking
```

2. Query to find out the customers that have more than $5000 with the bank. Order the result set by total_account_balance in descending order.

```
SQL> SELECT c.first_name,c.customer_id, SUM(account_balance) AS total_account_balance,c.state
  2  FROM customers c
  3      JOIN accounts a ON
  4      c.customer_id = a.customer_id
  5  GROUP BY c.first_name,c.customer_id, c.state
  6  HAVING SUM(account_balance) > 5000
  7  ORDER BY total_account_balance DESC;

FIRST_NAME                                         CUSTOMER_ID
-------------------------------------------------- -----------
TOTAL_ACCOUNT_BALANCE STAT
--------------------- ----
Ravi                                                         9
              180000 RJ

Swati                                                        6
            150000.5 TS

Rajesh                                                       3
           120000.75 TN


FIRST_NAME                                         CUSTOMER_ID
-------------------------------------------------- -----------
TOTAL_ACCOUNT_BALANCE STAT
--------------------- ----
Pooja                                                        8
            95000.25 MH

Arjun                                                        5
              80000 KA

Priya                                                        2
             75000.5 DL


FIRST_NAME                                         CUSTOMER_ID
-------------------------------------------------- -----------
TOTAL_ACCOUNT_BALANCE STAT
--------------------- ----
```

3. Query that returns the account_balances that are greater than the average account_balance for all the accounts.Return the customer_id, account_number and account_balance for each customer.

```
SQL>
SQL> SELECT c.customer_id,account_number, account_balance
  2  FROM accounts a
  3      JOIN customers c ON
  4      a.customer_id= c.customer_id
  5  WHERE account_balance >(
  6                        SELECT AVG(account_balance)
  7                        FROM accounts);

CUSTOMER_ID ACCOUNT_NUMBER ACCOUNT_BALANCE
----------- -------------- ---------------
          3        1000003       120000.75
          6        1000006        150000.5
          8        1000008        95000.25
          9        1000009          180000
```

4. This query calculates the average loan amount for each loan type.

```
SQL> SELECT
  2      loan_type,
  3      AVG(loan_amount) AS avg_loan_amount
  4  FROM
  5      loans
  6  GROUP BY
  7      loan_type;

LOAN_TYPE                   AVG_LOAN_AMOUNT
--------------------------- --------------
Personal Loan                    12333.3333
Home Loan                             30000
Education Loan                   14333.3333
Car Loan                              29000
```

5. This query lists customers along with their credit card details who have the maximum credit card debt.

```
SQL> SELECT
  2      c.customer_id,
  3      c.first_name,
  4      c.last_name,
  5      cc.credit_card_number,
  6      cc.amount_spent
  7  FROM
  8      customers c
  9  JOIN
 10      credit_cards cc ON c.customer_id = cc.customer_id
 11  WHERE
 12      cc.amount_spent = (SELECT MAX(amount_spent) FROM credit_cards);

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME                                          CREDIT_CARD_NUMBER
-------------------------------------------------- ------------------
AMOUNT_SPENT
------------
         2 Priya
Singh                                                 2.2223E+15
     2500.75
```

6 This query calculates the total credit card debt for each branch.

```
SQL> SELECT
  2      b.branch_code,
  3      b.head_office,
  4      SUM(cc.amount_spent) AS total_credit_card_debt
  5  FROM
  6      branch b
  7  LEFT JOIN
  8      customers c ON b.branch_code = c.branch_code
  9  LEFT JOIN
 10      credit_cards cc ON c.customer_id = cc.customer_id
 11  GROUP BY
 12      b.branch_code, b.head_office;

BRANCH_COD HEAD_OFFICE                    TOTAL_CREDIT_CARD_DEBT
---------- ------------------------------ ----------------------
B001       Mumbai Central                                 1200.5
B002       Delhi Central                                 2500.75
B003       Chennai Central                                800.25
B004       Kolkata Central                                1600.5
B005       Bangalore Central                             1000.75
B006       Hyderabad Central                             1800.25
B007       Ahmedabad Central                              1400.5
B008       Pune Central                                  1920.75
B009       Jaipur Central                                1280.25
B010       Lucknow Central                                1120.5

10 rows selected.
```

7. Show All Savings Accounts:

```
SQL> SELECT * FROM accounts WHERE account_type = 'Savings';

ACCOUNT_NUMBER CUSTOMER_ID ACCOUNT_BALANCE ACCOUNT_TY
-------------- ----------- --------------- ----------
ACCOUNT_TYPE_DESCRIPTION
--------------------------------------------------
       1000001           1           50000 Savings
Regular Savings Account

       1000003           3       120000.75 Savings
Premium Savings Account

       1000005           5           80000 Savings
Joint Savings Account


ACCOUNT_NUMBER CUSTOMER_ID ACCOUNT_BALANCE ACCOUNT_TY
-------------- ----------- --------------- ----------
ACCOUNT_TYPE_DESCRIPTION
--------------------------------------------------
       1000007           7        60000.75 Savings
Regular Savings Account

       1000009           9          180000 Savings
Premium Savings Account
```

8. Retrieve Account Balances for a Specific Branch:

```
SQL> SELECT a.account_number, a.account_balance
  2  FROM accounts a
  3  JOIN customers c ON a.customer_id = c.customer_id
  4  WHERE c.branch_code = 'B001';

ACCOUNT_NUMBER ACCOUNT_BALANCE
-------------- ---------------
       1000001           50000
```

9. Show the Average Loan Amount for Each Loan Type:

```
SQL> SELECT loan_type, AVG(loan_amount) AS avg_loan_amount
  2  FROM loans
  3  GROUP BY loan_type;

LOAN_TYPE                    AVG_LOAN_AMOUNT
---------------------------- ---------------
Personal Loan                     12333.3333
Home Loan                              30000
Education Loan                    14333.3333
Car Loan                               29000
```

10.shows the names of customers

```
SQL>  select first_name,last_name from customers;

FIRST_NAME
--------------------------------------------------
LAST_NAME
--------------------------------------------------
Amit
Verma

Priya
Singh

Rajesh
Kumar


FIRST_NAME
--------------------------------------------------
LAST_NAME
--------------------------------------------------
Neha
Sharma

Arjun
Reddy

Swati
Mishra


FIRST_NAME
--------------------------------------------------
LAST_NAME
--------------------------------------------------
Anuj
Patel

Pooja
Shukla
```

11.Calculate the total balance of all accounts for each branch:

```
SQL> SELECT b.branch_code, b.city_name, SUM(a.account_balance) AS total_balance
  2  FROM branch b
  3  JOIN customers c ON b.branch_code = c.branch_code
  4  JOIN accounts a ON c.customer_id = a.customer_id
  5  GROUP BY b.branch_code, b.city_name;

BRANCH_COD CITY_NAME                    TOTAL_BALANCE
---------- ------------------------------ -------------
B001       Mumbai                              50000
B002       Delhi                             75000.5
B003       Chennai                          120000.75
B004       Kolkata                           25000.25
B005       Bangalore                            80000
B006       Hyderabad                         150000.5
B007       Ahmedabad                         60000.75
B008       Pune                              95000.25
B009       Jaipur                              180000
B010       Lucknow                           30000.5

10 rows selected.
```

12.Calculate the total outstanding balance for all loans:

```
SQL> SELECT SUM(loan_amount) AS total_outstanding_balance
  2  FROM loans;

TOTAL_OUTSTANDING_BALANCE
-------------------------
                   198000
```

13.List customers who have both a credit card and a loan with amounts exceeding $10,000:

```
SQL> SELECT c.customer_id, c.first_name, c.last_name
  2  FROM customers c
  3  JOIN credit_cards cc ON c.customer_id = cc.customer_id
  4  JOIN loans l ON c.customer_id = l.customer_id
  5  WHERE cc.credit_card_number IS NOT NULL AND l.loan_amount > 10000;

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          1 Amit
Verma

          2 Priya
Singh

          4 Neha
Sharma


CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          5 Arjun
Reddy

          6 Swati
Mishra

          7 Anuj
Patel
```

14. List customers who have accounts with balances greater than $5000:

```
SQL> SELECT c.customer_id, c.first_name, c.last_name
  2  FROM customers c
  3  JOIN accounts a ON c.customer_id = a.customer_id
  4  WHERE a.account_balance > 5000;

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          1 Amit
Verma

          2 Priya
Singh

          3 Rajesh
Kumar


CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          4 Neha
Sharma

          5 Arjun
Reddy
```

15. List customers who have a loan with an interest rate greater than 10%:

```
SQL> SELECT c.customer_id, c.first_name, c.last_name
  2  FROM customers c
  3  JOIN loans l ON c.customer_id = l.customer_id
  4  WHERE l.interest_rate > 10;

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          1 Amit
Verma

          5 Arjun
Reddy

          9 Ravi
Yadav
```

16. Calculate the average loan amount per loan type:

```
SQL> SELECT loan_type, AVG(loan_amount) AS avg_loan_amount
  2  FROM loans
  3  GROUP BY loan_type;

LOAN_TYPE                     AVG_LOAN_AMOUNT
----------------------------- ---------------
Personal Loan                      12333.3333
Home Loan                               30000
Education Loan                     14333.3333
Car Loan                                29000
```

17. Query to Print Acc. No. and loans

```
SQL> SELECT a.account_number, l.loan_number
  2  FROM accounts a
  3  JOIN loans l ON a.customer_id = l.customer_id;

ACCOUNT_NUMBER LOAN_NUMBER
-------------- -----------
       1000001   100000001
       1000002   100000002
       1000003   100000003
       1000004   100000004
       1000005   100000005
       1000006   100000006
       1000007   100000007
       1000008   100000008
       1000009   100000009
       1000010   100000010

10 rows selected.
```

18.query for list of customers who have savings acc

```
SQL>    SELECT c.customer_id, c.first_name, c.last_name
  2  FROM customers c
  3  JOIN accounts a ON c.customer_id = a.customer_id
  4  WHERE a.account_type = 'Savings';

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          1 Amit
Verma

          3 Rajesh
Kumar

          5 Arjun
Reddy


CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          7 Anuj
Patel

          9 Ravi
Yadav
```

19.query for list of customers who have checking acc

```
SQL> SELECT c.customer_id, c.first_name, c.last_name
  2  FROM customers c
  3  JOIN accounts a ON c.customer_id = a.customer_id
  4  WHERE a.account_type = 'Checking';

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          2 Priya
Singh

          4 Neha
Sharma

          6 Swati
Mishra


CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
          8 Pooja
Shukla

         10 Anita
Srivastava
```

20.query for contact details of customers

```
SQL> SELECT customer_id, first_name, last_name, street_number, street_name, unit, city, state, zip_code, email_id
  2  FROM customers;

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
STREET_NUMBER                  STREET_NAME                    UNIT
------------------------------ ------------------------------ ----------
CITY                           STAT   ZIP_CODE
------------------------------ ---- ----------
EMAIL_ID
----------------------------------------
         1 Amit
Verma
123                            Main Street                    Apt 301

CUSTOMER_ID FIRST_NAME
----------- --------------------------------------------------
LAST_NAME
--------------------------------------------------
STREET_NUMBER                  STREET_NAME                    UNIT
------------------------------ ------------------------------ ----------
CITY                           STAT   ZIP_CODE
------------------------------ ---- ----------
EMAIL_ID
----------------------------------------
Mumbai                         MH       400001
amit.verma@email.com
```

VII. Self -Learning beyond classroom

Self-learning a project involving a financial_services management system entails understanding the database schema, practicing SQL querying to extract customer details, account information, and transaction histories. It involves mastering SQL constructs like joins, subqueries, and aggregates, ensuring data integrity through normalization principles, and considering broader aspects such as regulatory compliance and security measures. Through hands-on experimentation and practical exploration, learners gain proficiency in database management and financial data analysis, equipping them with valuable skills for real-world applications.

VIII. Learning from the Project

Learning from the above project involves a comprehensive exploration of database management in the context of financial services. By studying the database schema and practicing SQL queries, learners can grasp fundamental concepts such as data modeling, normalization, and relational database design. Through hands-on experimentation with complex queries, learners gain practical insights into data manipulation, retrieval, and analysis, honing their problem-solving skills. Moreover, delving into the nuances of financial data management fosters an understanding of regulatory compliance, security measures, and ethical considerations in the finance industry. Overall, learning from this project provides a solid foundation in database management and equips learners with essential skills applicable across various domains.

IX. Project Demonstration

Project Demonstration: Bank Management System

Introduction: The Bank Management System project aims to illustrate the practical application of database management concepts in a financial services environment. The demonstration showcases the functionalities of the system, including customer management, account handling, loan processing, and branch operations.

1. Environment Setup:

Initiate the SQL*Plus environment or any other SQL client tool.

Connect to the database instance hosting the Bank Management System database.

2. Database Schema Overview:

Present an overview of the database schema, including tables such as customers, accounts, loans, credit_cards, and branch.

Explain the relationships between entities, primary and foreign key constraints, and data types.

3. Customer Management:

Retrieve and display customer details such as customer ID, name, contact information, and address.

Demonstrate the process of adding new customers to the database using SQL INSERT statements.

Showcase querying capabilities to search for customers based on specific criteria such as city, state, or email ID.

4. Account Handling:

Illustrate account management functionalities by displaying account numbers, types, and balances for each customer.

Execute SQL queries to create new accounts, update account balances, and delete inactive accounts.

Showcase transactions such as deposits, withdrawals, and transfers between accounts.

5. Loan Processing:

Demonstrate the loan processing workflow by retrieving loan details including loan number, amount, type, and duration.

Use SQL queries to approve, reject, or modify loan applications based on predefined criteria.

Showcase calculations of monthly payment amounts and interest rates for different types of loans.

6. Credit Card Services:

Display credit card information including card number, credit limit, and amount spent for each customer.

Execute SQL queries to issue new credit cards, adjust credit limits, and track credit card transactions.

Illustrate the process of generating monthly statements and calculating minimum payment amounts.

7. Branch Operations:

Present branch details such as branch code, name, head office, and city.

Showcase querying capabilities to retrieve information about branch locations, customer footfall, and branch performance metrics.

Execute SQL queries to identify branches with the highest number of accounts or outstanding loans.

8. Advanced Queries and Analytics:

Demonstrate the use of aggregate functions, subqueries, and joins to derive actionable insights from the database.

Present visualizations or reports generated from SQL queries to illustrate key findings and performance metrics.

Conclusion: The project demonstration concludes by highlighting the versatility and power of SQL for managing financial data effectively. Learners gain practical experience in database management, SQL querying, and financial services operations, equipping them with valuable skills for real-world applications in the finance industry.

X. Challenges Faced

1. **Data Integrity:** Ensuring the accuracy and consistency of data across multiple tables posed challenges, particularly when managing transactions and updating account balances.
2. **Complex Queries:** Crafting complex SQL queries to retrieve specific information, such as analyzing loan repayment trends or identifying high-value customers, required advanced knowledge of SQL constructs and data manipulation techniques.
3. **Performance Optimization:** Optimizing query performance, especially when dealing with large datasets, was crucial to maintain system responsiveness and efficiency.
4. **Security Concerns:** Addressing security concerns related to sensitive financial data, including access control, encryption, and vulnerability management, required stringent measures to safeguard against unauthorized access and data breaches.
5. **Regulatory Compliance:** Ensuring compliance with regulatory requirements, such as data privacy laws and financial regulations, necessitated meticulous attention to detail and adherence to industry standards.
6. **User Interface Design:** Designing an intuitive and user-friendly interface for interacting with the database system posed challenges in terms of usability, accessibility, and user experience.
7. **Error Handling:** Implementing robust error handling mechanisms to deal with exceptions, invalid inputs, and system failures was essential to maintain data integrity and system reliability.
8. **Scalability:** Planning for scalability and future growth of the database system required careful consideration of factors such as data volume, performance requirements, and resource allocation.
9. **Training and Education:** Providing adequate training and education to users and stakeholders on database usage, SQL querying, and best practices in financial data management proved essential for effective system adoption and utilization.

- **Integration with External Systems:** Integrating the bank management system with external systems, such as third-party applications or legacy systems,

presented challenges in terms of data synchronization, compatibility, and interoperability.

By addressing these challenges effectively, the Bank Management System project demonstration enables participants to gain valuable insights and practical experience in overcoming real-world obstacles in database management and financial services operations.

XI. Conclusion

In conclusion, the Bank Management System project demonstration provides a comprehensive overview of database management in the context of financial services. Through hands-on exploration of SQL querying, database schema navigation, and practical application of financial transactions, participants gain invaluable insights into the complexities of managing banking operations effectively. The demonstration underscores the importance of efficient data management, adherence to regulatory compliance, and leveraging analytical tools to derive actionable insights from financial data. By equipping participants with practical skills and knowledge, the demonstration empowers them to address real-world challenges in the finance industry and make informed decisions to drive organizational success.