SIMPLE BANKING APPLICATION

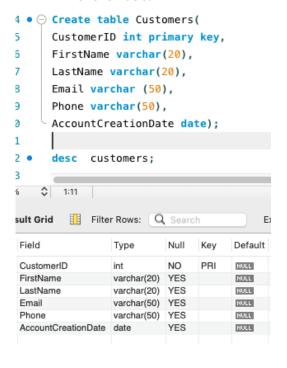
Efficient SQL Solutions for Simple Banking Applications

INTRODUCTION

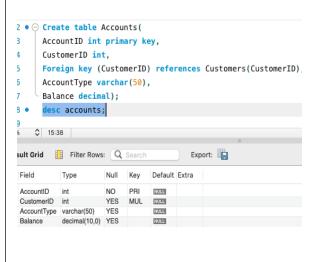
In the digital age, efficient financial management systems are essential. This case study explores a Simple Banking Application, highlighting the use of SQL for managing banking data. It covers the design and implementation of a basic banking system, focusing on core functionalities like account management, transactions, and balance tracking. We will examine the database schema, essential SQL queries, and practices for maintaining data integrity and security. This case study aims to demonstrate how SQL can effectively support the development of practical and reliable banking applications

DATA DEFINITION LANGUAGE(CREATE)

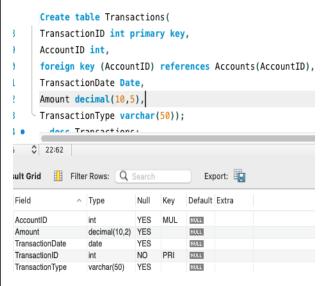
 Write SQL statements to create all the tables with the specified columns and foreign key references.



 Write SQL statements to create all the tables with the specified columns and foreign key references.

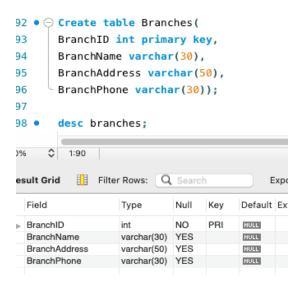


 Write SQL statements to create all the tables with the specified columns and foreign key references.



DATA DEFINITION LANGUAGE(CREATE)

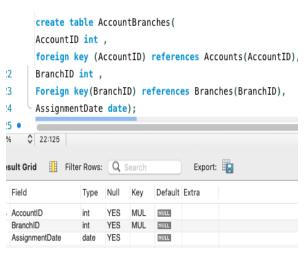
 Write SQL statements to create all the tables with the specified columns and foreign key references.



 Write SQL statements to create all the tables with the specified columns and foreign key references.

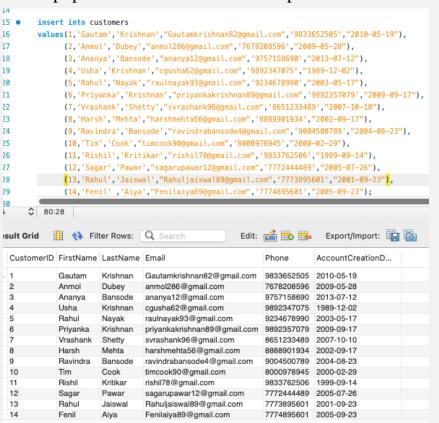
```
Create table Loans(
     LoanID int primary key,
     CustomerID int,
     Foreign key(CustomerID) references Customers(CustomerID),
     LoanAmount decimal(10,1),
     InterestRate Decimal(3,2),
     StartDate date,
     EndDate date):
   $ 12:153
            Filter Rows: Q Search
                                         Export:
Field
                                      Default Extra
LoanID
CustomerID
                           YES
LoanAmount
StartDate
                                      NULL
                                      NULL
EndDate
                           YES
```

 Write SQL statements to create all the tables with the specified columns and foreign key references.

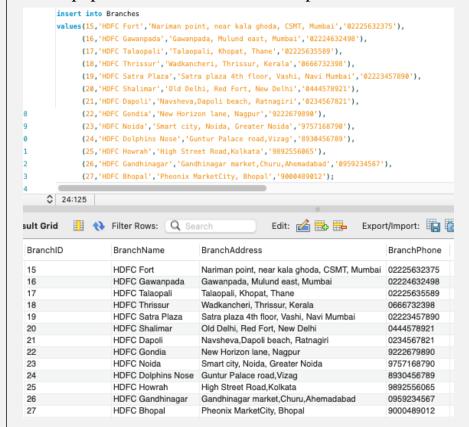


DATA MANUPULATION LANGUAGE (INSERT)

o Insert at least 10 records into each table to populate the database with sample data.

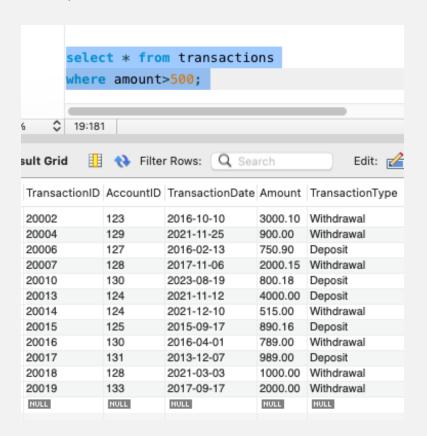


 Insert at least 10 records into each table to populate the database with sample data.

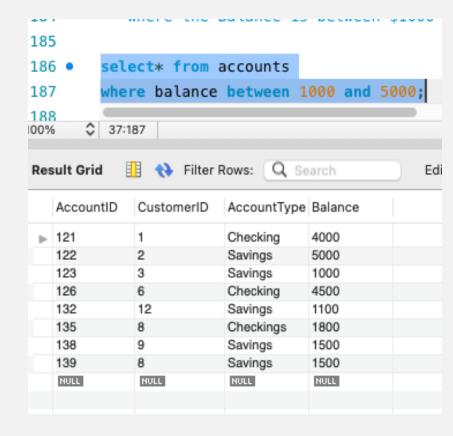


SELECT & WHERE CLAUSE

• Write a query to select all transactions from the Transactions table where the Amount is greater than \$500.

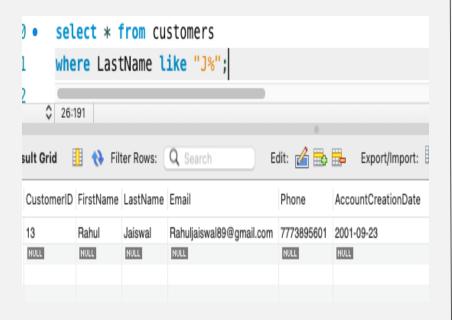


 Write a query to select all Accounts where the Balance is between \$1000 and \$5000 and the AccountType is 'Checking'

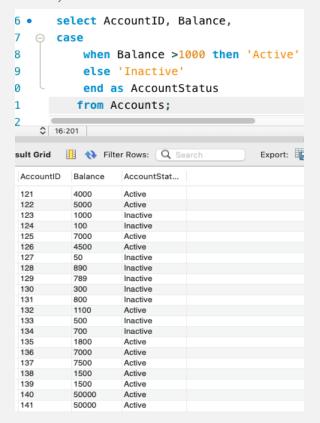


USING LIKE OPERATOR AND CASE STATEMENTS

 Write a query to select all Customers whose LastName starts with 'J'.

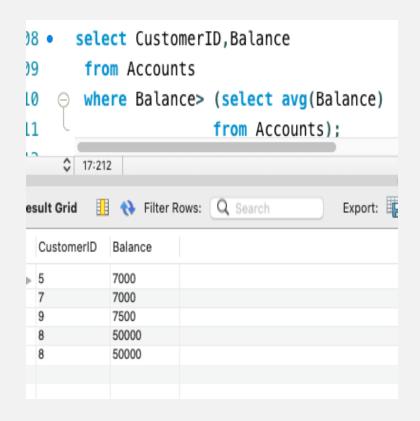


 Write a query to select AccountID and a new column AccountStatus from the Accounts table. If Balance is greater than \$1000, set AccountStatus to 'Active', otherwise 'Inactive'.



SUBQUERY AND GROUP BY

 Write a query to find all Customers who have a balance in their accounts greater than the average balance of all accounts. Use a subquery to find these CustomerIDs.

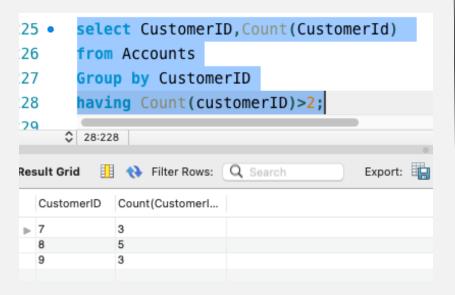


• Write a query to get the total Balance for each AccountType. Group the results by AccountType.

```
select* from Accounts;
220 •
221
        select AccountType,sum(Balance) Total_Balance
222 •
        from Accounts
223
        Group by AccountType;
224
225
      22:224
           Filter Rows: Q Search
                                             Export:
Result Grid
   AccountType Total_Balance
  Checking
             77039
             68990
   Savings
```

HAVING CLAUSE AND LIMIT

 Write a query to get the total number of accounts for each Customer, but only include customers who have more than 2 accounts. Use the HAVING clause.

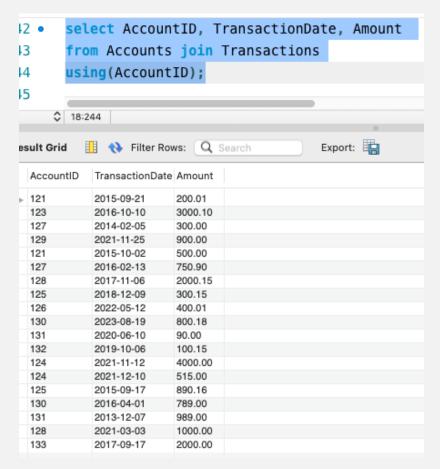


• Write a query to select the top 5 customers with the highest LoanAmount.

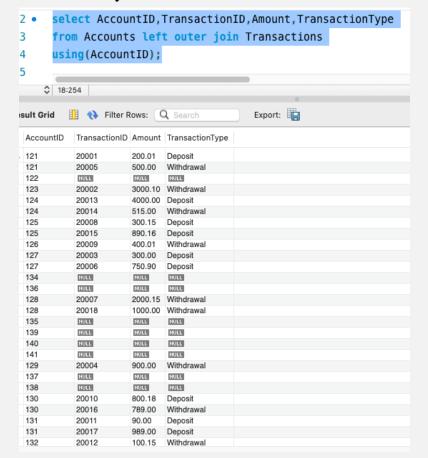
```
select customerID,LoanAmount
34 •
        from loans
35
        order by LoanAmount desc
36
        limit 5;
37
38
39
      C 1:238
              Filter Rows:
                            Q Search
Result Grid
  customerID LoanAmount
  7
           55000.0
  2
           50000.0
           34000.0
  12
           30000.0
           25000.0
```

JOINS

 Write a query to join Transactions with Accounts to get a list of all transactions with AccountID, TransactionDate, and Amount

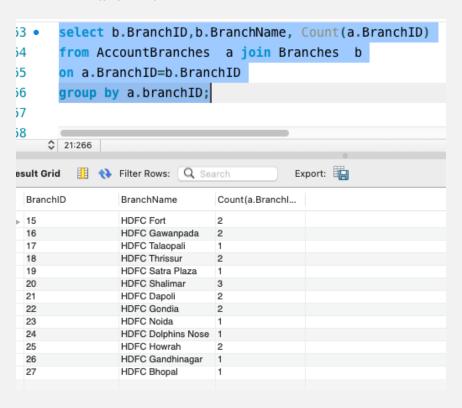


 Write a query to get a list of all Accounts and any associated Transactions. Include accounts that might not have any transactions.



SUBQUERY USING JOIN & JOIN WITH AGGREGATION

 Write a query to get the total number of accounts for each branch. Use an INNER JOIN between AccountBranches and Branches, and group by BranchID.

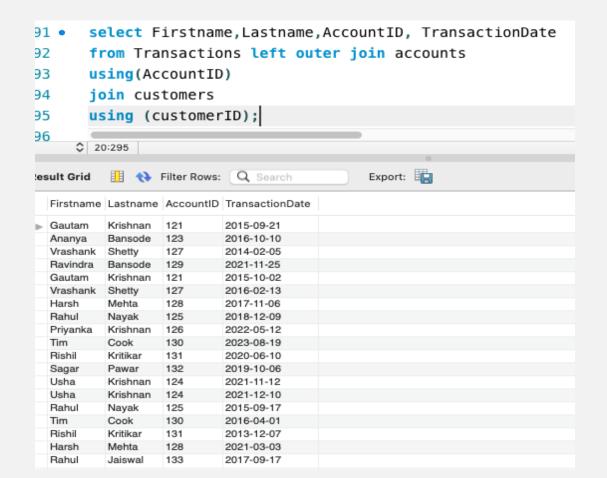


 Write a query to find all Branches that manage accounts with a total balance of more than \$100,000. Use a subquery in the WHERE clause to find these BranchIDs.

```
select branchID
·78 •
:79
      from Branches
    180
281
                     from AccountBranches join Accounts
182
                     using(AccountID)
                     group by BranchID
183
84
                     having sum(balance)>100000);
285
286
     22:283
        Filter Rows: Q Search
                                    Export:
Result Grid
  branchID
≥ 20
```

ADVANCED JOIN

 Write a query to list FirstName, LastName, AccountID, and TransactionDate for all transactions. Use INNER JOIN and LEFT JOIN as necessary to get all required details.



THANKYOU