Explorer Bank

INFO 364 Group Project

By: Chris Combs, Juwan Jackson, Kevon Mahban, Chris Copeland, Emmanuel Afriyie

Business Scenario Introduction - The business scenario introduction is a general introduction or overview of the operational environment of the business activity, such as retailing, marketing, manufacturing, etc.

Explorer Banking serves a wide range of customers, from individuals to big businesses, offering everything from basic checking accounts to complex investment options. They've built a reputation for solid service and are always looking to stay ahead of the curve in the fast-moving world of finance. With technology changing how we manage money, the bank knows it’s vital to keep up to ensure they’re meeting everyone’s needs.

To accomplish this, Explorer Banking is rolling out a new database system. This upgrade isn’t just about keeping pace with tech trends, it’s about making things smoother and quicker for their customers and staying on top of the regulatory stuff without any friction. The new system will make it easier to handle data, speed up transactions, and pave the way for new services and improvements as the bank continues to grow and evolve in the digital era.

Database Requirement Analysis

Key Entities

1. Customer - This entity stores information related to a customer in the banking system. It includes the customer's ID, their first name, middle name, last name, street address, city, state, and their social security number. A customer may also have another customer on their family plan.

Cardinality constraints: A customer can own zero or many accounts. A customer can be advised by one and only one representative at a time.

2. Account - The account entity contains bank accounts relating to an individual customer. This entity has an Account ID to individually identify each entry, the foreign key Customer ID to tell which customer the account belongs to, the account type which can be checking or savings, the balance which is the amount of money in the bank account, the interest rate on the account, and the current status of the bank account. A customer may have no accounts or multiple accounts.

Cardinality constraints:
A single account can only be owned by one and only customer.
An account can have zero or many loans.
An account can be protected by one or many types of securities.
An account can manage zero or many transactions.
An account can use zero or many cards

3. Card - The card entity stores credit cards and debit cards relating to a specific account. It contains a supertype containing the unique identifier Card ID, the foreign key Account ID which specifies which account it belongs to, the card number, expiration date, and security code. The supertype is total specialization with disjoint, so a member of the card entity must be a member of a subtype, but only a member of one subtype. The credit subtype contains attributes for the credit limit, the available credit, the payment due date, and the statement balance. The debit subtype contains the pin number associated with the debit card.

Cardinality constraints:
A card can only be used one and only one account
A card can finalize 0 or many transactions

4. Transaction - This entity stores payments, transaction information. The foreign key Account ID specifies which account the transaction is associated with, including the Amounts and Transaction Dates of the account. The Card Id

specifying what card was used. And of course Withdrawals and Deposits regarding the transactions.

Cardinality constraints:
A Transaction is managed by one and only one account
A Transaction is finalized by one and only one card per transaction
A Transaction is tracked by one and only one mapping interaction
A single Transaction is organized by one and only one category.

5. Loan - The loan entity contains loans linked to a specific account, meaning an account could have zero loans or multiple loans linked to it. Also, a loan can only be linked to one account. The loan entity contains the primary key Loan ID, the foreign key Account ID, principal amount, interest rate, term, start date, end date, and loan type.

Cardinality constraints:
A loan is loaned by one and only one account

6. Loan Payment - The loan payment entity contains Loan Payment ID as the primary ID. LoanID as a foreign key as well as AccountID. The entity also contains the payment date and the amount paid. This stores data about which accounts have made a payment on a loan.

Cardinality constraints:
One account can pay many or one loan payments.
There can be one or many loan payments for a loan.

7. Credit Transaction- The credit transaction entity contains the primary key Credit Transaction ID. The entity also holds the date, amount, type, and statement balance of the credit account.

Cardinality constraints:
The credit subtype manages one or many credit transactions.
Credit transactions record many or one credit transaction to one transaction account.

8. Application- The application entity contains application Id as the primary key. It also contains loan ID and customer ID as foreign keys. The entity contains the status of the application as well.

Cardinality constraints:
A customer can have one or many applications.
There can be multiple applications but only one loan per application.

9. Representative- This entity stores information about the customer representative who is associated with aiding the customer. It includes the customer representative's ID, Branch ID, Name, E-mail, and Phone number. All for contact and association with the bank.

Cardinality constraints:
Customer representatives aid one or many customers
Customer representatives is overviewed by one and only one Branch

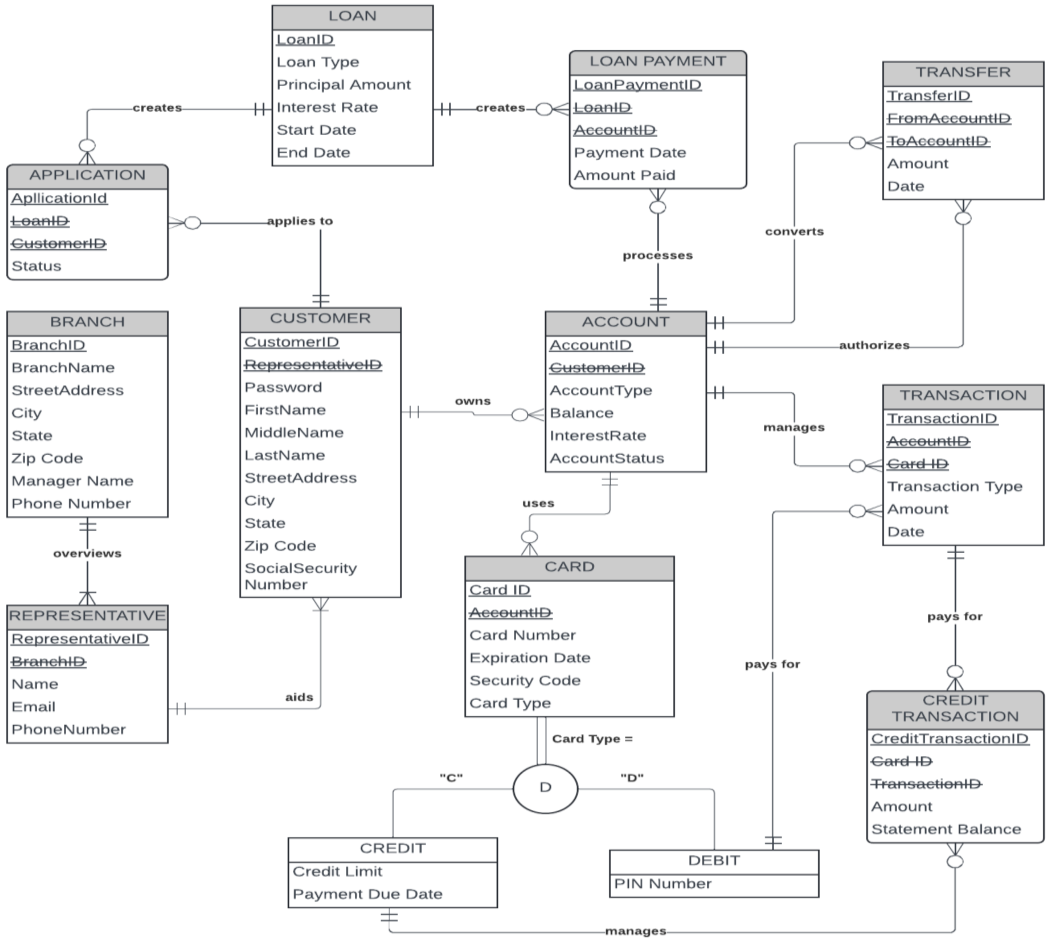
10. Branch- This entity stores information about the Branch that Customer Representatives are assigned to. It includes the Branch ID for identification of branches, Branch Name defining divisions of work, Location, Manager of representatives, and Phone Number for contact.

Cardinality constraints:
A Branch overviews one or many Customer Representatives

11. Transfer- This entity stores information about Account transfers to different accounts. It includes Transfer ID to identify each transfer, From Account ID which tracks which account the transfer is coming from. To Account ID which shows where the transfer is going. The amount that the account is transferring. As well as the date when the transfer occurred.

Cardinality Constraints:
Transfer is authorized by one account to many or one transfer.
There are one or many transfers that are sent to one account.

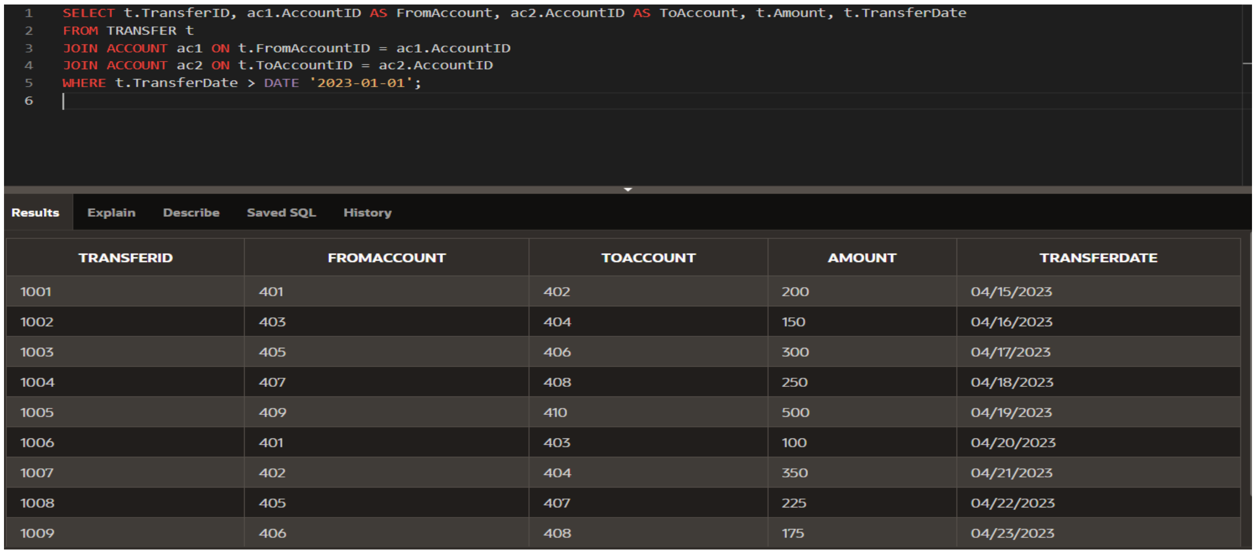
E-R Diagram



Demo of Data Queries

Query 1
Inter-account Transfers

This query helps monitor the transfers between accounts, indicating active account movements.

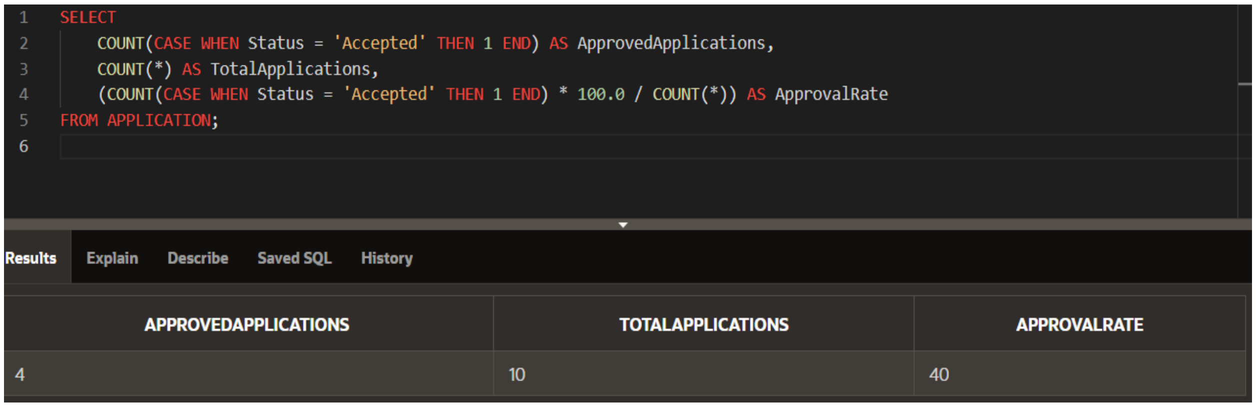


```sql
SELECT t.TransferID, ac1.AccountID AS FromAccount, ac2.AccountID AS ToAccount, t.Amount, t.TransferDate
FROM TRANSFER t
JOIN ACCOUNT ac1 ON t.FromAccountID = ac1.AccountID
JOIN ACCOUNT ac2 ON t.ToAccountID = ac2.AccountID
WHERE t.TransferDate > DATE '2023-01-01';
```

TRANSFERID	FROMACCOUNT	TOACCOUNT	AMOUNT	TRANSFERDATE
1001	401	402	200	04/15/2023
1002	403	404	150	04/16/2023
1003	405	406	300	04/17/2023
1004	407	408	250	04/18/2023
1005	409	410	500	04/19/2023
1006	401	403	100	04/20/2023
1007	402	404	350	04/21/2023
1008	405	407	225	04/22/2023
1009	406	408	175	04/23/2023

Query 2

Loan Approval Rate - Calculate the percentage of approved loan applications compared to the total number of applications received.

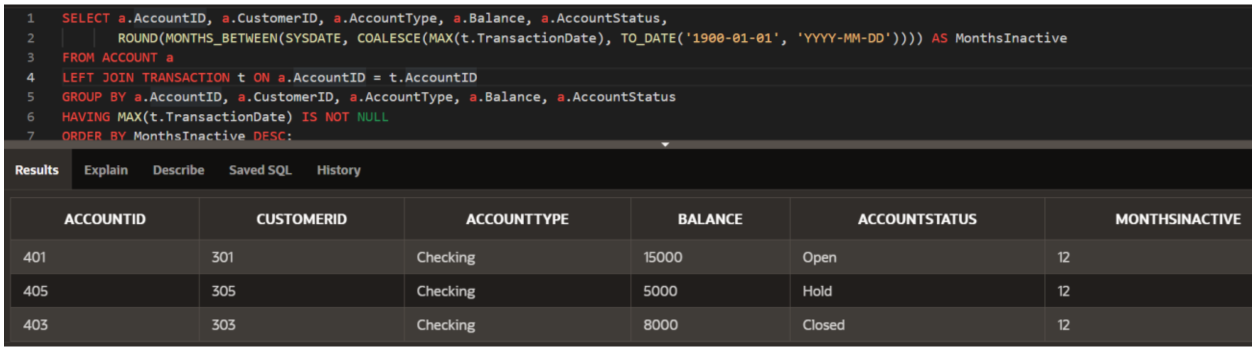


```sql
SELECT
    COUNT(CASE WHEN Status = 'Accepted' THEN 1 END) AS ApprovedApplications,
    COUNT(*) AS TotalApplications,
    (COUNT(CASE WHEN Status = 'Accepted' THEN 1 END) * 100.0 / COUNT(*)) AS ApprovalRate
FROM APPLICATION;
```

APPROVEDAPPLICATIONS	TOTALAPPLICATIONS	APPROVALRATE
4	10	40

Query 3

Inactive Accounts - Identify accounts that have been inactive for a certain length, sorting by months inactive. Can be very useful for the bank to find bank accounts that are more susceptible to fraudulent activity since they have been inactive for such a long period.



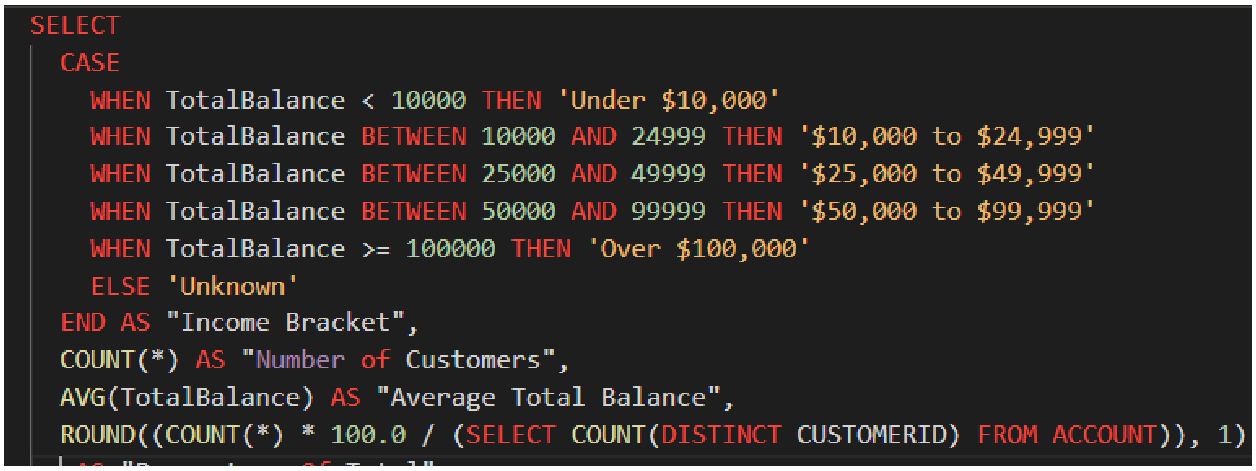
```sql
SELECT a.AccountID, a.CustomerID, a.AccountType, a.Balance, a.AccountStatus,
       ROUND(MONTHS_BETWEEN(SYSDATE, COALESCE(MAX(t.TransactionDate), TO_DATE('1900-01-01', 'YYYY-MM-DD')))) AS MonthsInactive
FROM ACCOUNT a
LEFT JOIN TRANSACTION t ON a.AccountID = t.AccountID
GROUP BY a.AccountID, a.CustomerID, a.AccountType, a.Balance, a.AccountStatus
HAVING MAX(t.TransactionDate) IS NOT NULL
ORDER BY MonthsInactive DESC;
```

Results Explain Describe Saved SQL History

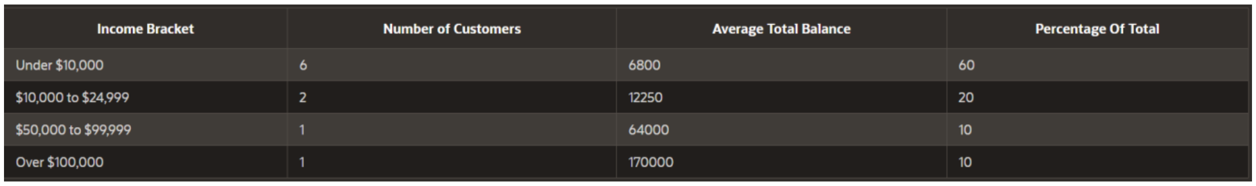
ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	ACCOUNTSTATUS	MONTHSINACTIVE
401	301	Checking	15000	Open	12
405	305	Checking	5000	Hold	12
403	303	Checking	8000	Closed	12

Query 4

Classify Customer Accounts - Puts customers into categories to get an idea of the customer's income levels. Can be very useful for the bank for potential target marketing and to see which markets they don't cater to the most.



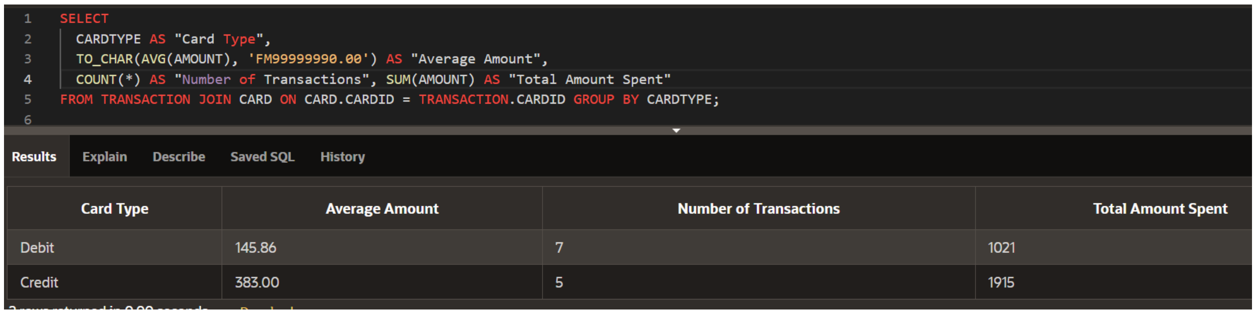
```sql
SELECT
  CASE
    WHEN TotalBalance < 10000 THEN 'Under $10,000'
    WHEN TotalBalance BETWEEN 10000 AND 24999 THEN '$10,000 to $24,999'
    WHEN TotalBalance BETWEEN 25000 AND 49999 THEN '$25,000 to $49,999'
    WHEN TotalBalance BETWEEN 50000 AND 99999 THEN '$50,000 to $99,999'
    WHEN TotalBalance >= 100000 THEN 'Over $100,000'
    ELSE 'Unknown'
  END AS "Income Bracket",
  COUNT(*) AS "Number of Customers",
  AVG(TotalBalance) AS "Average Total Balance",
  ROUND((COUNT(*) * 100.0 / (SELECT COUNT(DISTINCT CUSTOMERID) FROM ACCOUNT)), 1)
```



Income Bracket	Number of Customers	Average Total Balance	Percentage Of Total
Under $10,000	6	6800	60
$10,000 to $24,999	2	12250	20
$50,000 to $99,999	1	64000	10
Over $100,000	1	170000	10

Query 5

Money Spent per Card Type - Identify how certain card types are utilized, and which are used more frequently.

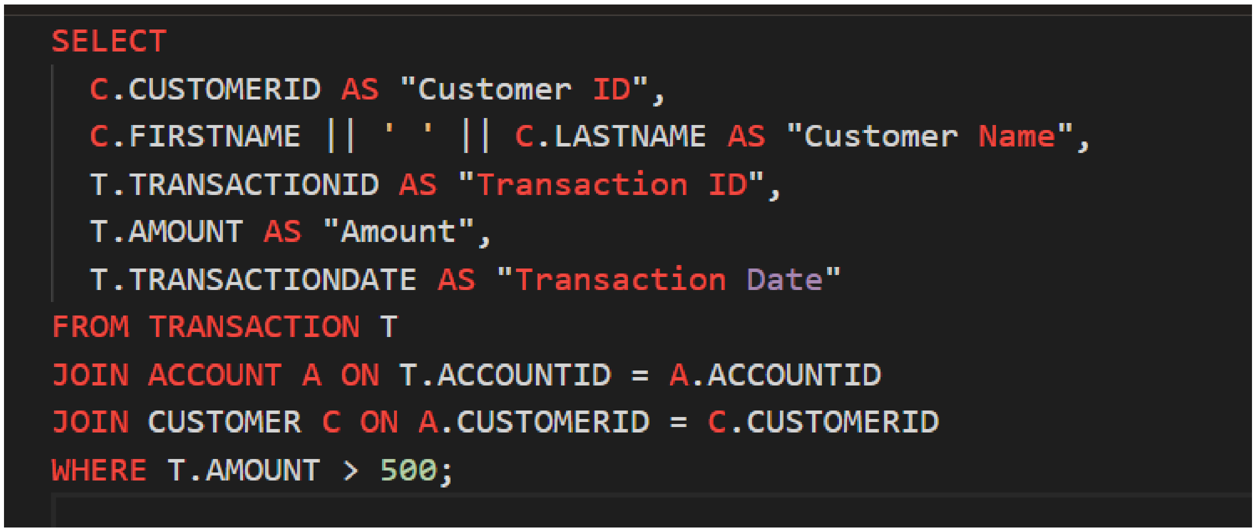


```sql
SELECT
  CARDTYPE AS "Card Type",
  TO_CHAR(AVG(AMOUNT), 'FM99999990.00') AS "Average Amount",
  COUNT(*) AS "Number of Transactions", SUM(AMOUNT) AS "Total Amount Spent"
FROM TRANSACTION JOIN CARD ON CARD.CARDID = TRANSACTION.CARDID GROUP BY CARDTYPE;
```

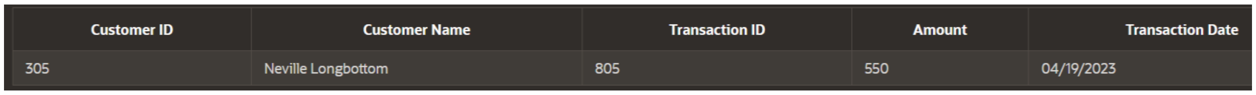
Card Type	Average Amount	Number of Transactions	Total Amount Spent
Debit	145.86	7	1021
Credit	383.00	5	1915

Query 6

Identify transactions over a certain amount of money, which can be used to identify fraudulent transactions.



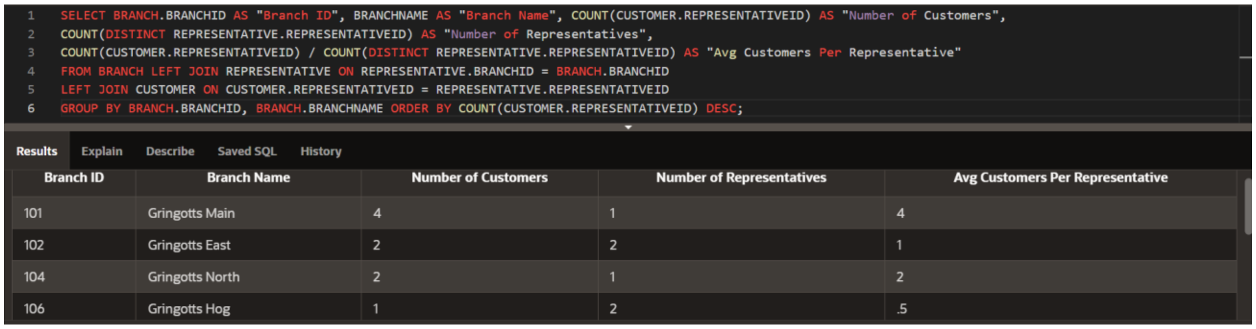
```sql
SELECT
  C.CUSTOMERID AS "Customer ID",
  C.FIRSTNAME || ' ' || C.LASTNAME AS "Customer Name",
  T.TRANSACTIONID AS "Transaction ID",
  T.AMOUNT AS "Amount",
  T.TRANSACTIONDATE AS "Transaction Date"
FROM TRANSACTION T
JOIN ACCOUNT A ON T.ACCOUNTID = A.ACCOUNTID
JOIN CUSTOMER C ON A.CUSTOMERID = C.CUSTOMERID
WHERE T.AMOUNT > 500;
```



Customer ID	Customer Name	Transaction ID	Amount	Transaction Date
305	Neville Longbottom	805	550	04/19/2023

Query 7

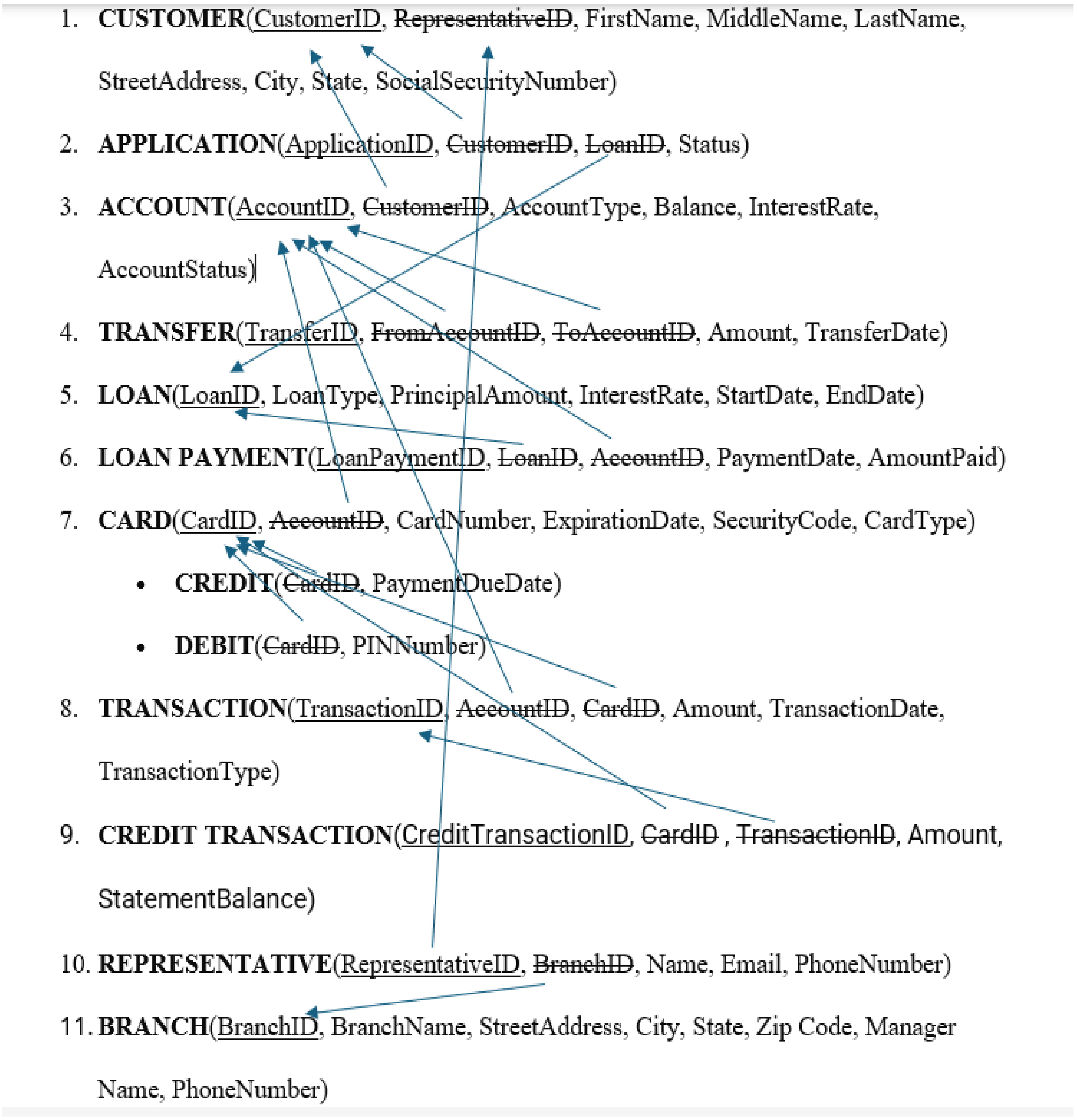
Identify the number of customers per representative, which might indicate which branches might need more representatives.



```sql
SELECT BRANCH.BRANCHID AS "Branch ID", BRANCHNAME AS "Branch Name", COUNT(CUSTOMER.REPRESENTATIVEID) AS "Number of Customers",
COUNT(DISTINCT REPRESENTATIVE.REPRESENTATIVEID) AS "Number of Representatives",
COUNT(CUSTOMER.REPRESENTATIVEID) / COUNT(DISTINCT REPRESENTATIVE.REPRESENTATIVEID) AS "Avg Customers Per Representative"
FROM BRANCH LEFT JOIN REPRESENTATIVE ON REPRESENTATIVE.BRANCHID = BRANCH.BRANCHID
LEFT JOIN CUSTOMER ON CUSTOMER.REPRESENTATIVEID = REPRESENTATIVE.REPRESENTATIVEID
GROUP BY BRANCH.BRANCHID, BRANCH.BRANCHNAME ORDER BY COUNT(CUSTOMER.REPRESENTATIVEID) DESC;
```

Branch ID	Branch Name	Number of Customers	Number of Representatives	Avg Customers Per Representative
101	Gringotts Main	4	1	4
102	Gringotts East	2	2	1
104	Gringotts North	2	1	2
106	Gringotts Hog	1	2	.5

3NF Relations:



1. **CUSTOMER**(CustomerID, ~~RepresentativeID~~, FirstName, MiddleName, LastName, StreetAddress, City, State, SocialSecurityNumber)
2. **APPLICATION**(ApplicationID, ~~CustomerID~~, ~~LoanID~~, Status)
3. **ACCOUNT**(AccountID, ~~CustomerID~~, AccountType, Balance, InterestRate, AccountStatus)
4. **TRANSFER**(TransferID, ~~FromAccountID~~, ~~ToAccountID~~, Amount, TransferDate)
5. **LOAN**(LoanID, LoanType, PrincipalAmount, InterestRate, StartDate, EndDate)
6. **LOAN PAYMENT**(LoanPaymentID, ~~LoanID~~, ~~AccountID~~, PaymentDate, AmountPaid)
7. **CARD**(CardID, ~~AccountID~~, CardNumber, ExpirationDate, SecurityCode, CardType)
    - **CREDIT**(~~CardID~~, PaymentDueDate)
    - **DEBIT**(~~CardID~~, PINNumber)
8. **TRANSACTION**(TransactionID, ~~AccountID~~, ~~CardID~~, Amount, TransactionDate, TransactionType)
9. **CREDIT TRANSACTION**(CreditTransactionID, ~~CardID~~ , ~~TransactionID~~, Amount, StatementBalance)
10. **REPRESENTATIVE**(RepresentativeID, ~~BranchID~~, Name, Email, PhoneNumber)
11. **BRANCH**(BranchID, BranchName, StreetAddress, City, State, Zip Code, Manager Name, PhoneNumber)

Another example of the 3nf relationships (arrows):

CUSTOMER(CustomerID) -----< APPLICATION(CustomerID)

CUSTOMER(CustomerID) -----< ACCOUNT(CustomerID)

ACCOUNT(AccountID) -----< TRANSFER(FromAccountID)

ACCOUNT(AccountID) -----< TRANSFER(ToAccountID)

CUSTOMER(CustomerID) -----< LOAN(CustomerID)

LOAN(LoanID) -----< APPLICATION(LoanID)

LOAN(LoanID) -----< LOAN PAYMENT(LoanID)

ACCOUNT(AccountID) -----< CARD(AccountID)

CARD(CardID) -----< CREDIT(CardID)

CARD(CardID) -----< DEBIT(CardID)

ACCOUNT(AccountID) -----< TRANSACTION(AccountID)

CARD(CardID) -----< TRANSACTION(CardID)

TRANSACTION(TransactionID) -----< CREDIT TRANSACTION(TransactionID)

In this representation:

- < denotes a one-to-many relationship.
- Each line represents a foreign key relationship between two tables.
- The arrow direction indicates the direction of the relationship, with the foreign key pointing to the primary key it references.