SQL PROJECT

BANKING SYSTEM

By- Vedika Kirve

OVERVIEW

Objective:

Build a realistic banking system database to analyze customer behavior, account activity, loans, and branch performance using PostgreSQL.

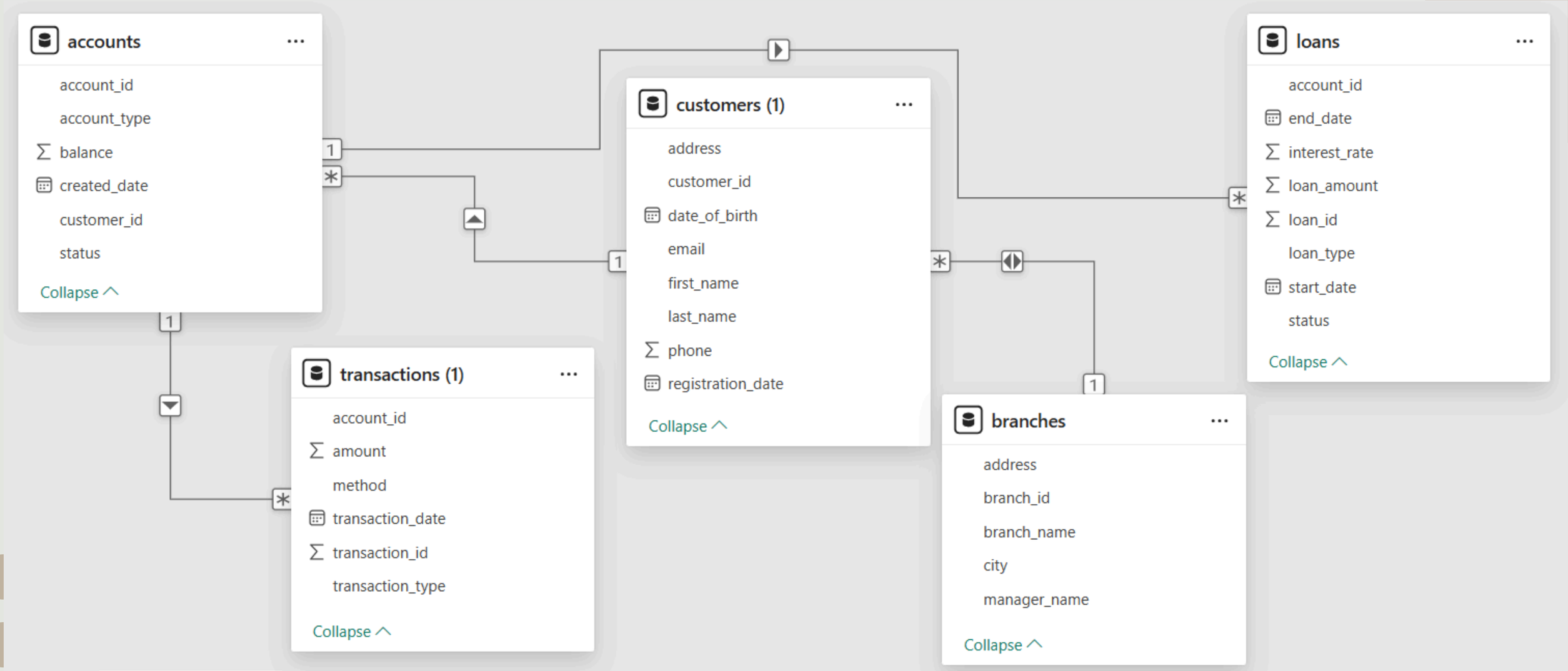
Dataset:

- Customers: 1,000 records
- Accounts: 1,500 records
- Transactions: 10,000 records
- Loans: 500 records
- Branches: 15 records

Advanced Features Implemented:

- Views: customer_summary, branch_summary
- Triggers: Auto-update account balance after transaction
- Stored Procedures: generate_monthly_statement(customer_id, month)
- Indexing for faster queries

DATABASE SCHEMA



BUSINESS QUESTIONS

Basic Queries

1. List all customers with their full name, email, and phone number.
2. Show all accounts with account type and balance.
3. Find all transactions above ₹50,000.
4. List all active loans with their account ID and loan amount.
5. Show all branches in the city “Mumbai”.

Intermediate Queries

1. Calculate total balance per customer.
2. Find customers with more than two accounts.
3. Show transactions per account in the last 30 days.
4. Identify customers who have both a loan and a savings account.
5. Find average loan amount per loan type.

Advanced Queries

1. Find the top 5 customers with the highest total balance.
2. Detect accounts with no activity for the last 6 months.
3. Calculate monthly revenue from interest on active loans.
4. Generate a summary of deposits vs withdrawals per branch.
5. Identify customers who frequently overdraw (negative balance).

Advanced Features

- Views: Create views for customer summaries, branch-wise revenue, or loan status reports.
- Triggers: Implement triggers to update account balances automatically after a transaction.
- Stored Procedures: Develop stored procedures to generate monthly statements or calculate interest.
- Indexing: Apply indexing on frequently queried columns like account_id, customer_id, and transaction_date for performance optimization.

Basic Queries

1. List all customers with their full name, email, and phone number

```
SELECT first_name || ' ' || last_name AS full_name, email, phone
FROM Customers;
```

	full_name text	email character varying (100)	phone character varying (15)
1	Navya Bhalla	nirvaan32@behl.net	+919600133890
2	Taimur Vaidya	zara07@hotmail.com	+916184959310
3	Ishaan Kashyap	navya83@hotmail.com	00305641395
4	Bhamini Bath	bosemishti@sule.biz	1669784801
5	Tarini Buch	ayesha01@vyas-thakur.com	+910391171822

2. Show all active accounts with their type and balance.

```sql
SELECT account_id, account_type, balance
FROM Accounts
WHERE status = 'Active';
```

	account_id [PK] integer	account_type character varying (20)	balance numeric (15,2)
1	2	Savings	140398.39
2	3	Loan	892287.39
3	4	Current	32750.90
4	5	Savings	505849.93
5	7	Current	589676.42

3. Find all transactions above ₹50,000.

```sql
SELECT transaction_id, account_id, transaction_type, amount
FROM Transactions
WHERE amount > 50000
ORDER BY amount DESC;
```

	transaction_id [PK] integer	account_id integer	transaction_type character varying (20)	amount numeric (15,2)
1	7911	546	Withdrawal	199971.45
2	2611	985	Deposit	199941.34
3	2751	1368	Withdrawal	199887.83
4	9196	1149	Deposit	199871.76
5	6702	233	Transfer	199850.43

4. List all active loans with their account ID and loan amount.

```
SELECT loan_id, account_id, loan_amount, loan_type, status
FROM Loans
WHERE status = 'Active';
```

	loan_id [PK] integer	account_id integer	loan_amount numeric (15,2)	loan_type character varying (50)	status character varying (20)
1	1	1144	4107177.17	Auto	Active
2	3	769	213373.96	Home	Active
3	6	1016	3515351.63	Auto	Active
4	11	1009	1221935.66	Auto	Active
5	12	400	3829194.57	Personal	Active

5. Show all branches in the city ‘Mumbai’.

```
SELECT branch_id, branch_name, address
FROM Branches
WHERE city = 'Mumbai';
```

	branch_id [PK] integer	branch_name character varying (100)	address character varying (255)

Intermediate Queries

1. Calculate the total balance per customer.

```
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS full_name,
       SUM(a.balance) AS total_balance
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
GROUP BY c.customer_id, full_name
ORDER BY total_balance DESC;
```

	customer_id [PK] integer	full_name text	total_balance numeric
1	537	Oorja Mallick	3870552.09
2	834	Eshani Zachariah	3342743.88
3	580	Yasmin Dhar	3234720.16
4	94	Shray Mandal	3193394.28
5	742	Advik Bir	3091378.49

2. Find customers with more than 2 accounts.

```
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS full_name,
       COUNT(a.account_id) AS num_accounts
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
GROUP BY c.customer_id, full_name
HAVING COUNT(a.account_id) > 2;
```

	customer_id [PK] integer	full_name text	num_accounts bigint
1	178	Vaibhav Hari	3
2	141	Aniruddh Badal	3
3	731	Hiran Chad	3
4	562	Siya Vasa	3
5	736	Siya Kala	3

3. Show transactions per account in the last 30 days.

```sql
SELECT account_id, COUNT(transaction_id) AS transaction_count,
       SUM(amount) AS total_amount
FROM Transactions
WHERE transaction_date >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY account_id
ORDER BY total_amount DESC;
```

	account_id integer	transaction_count bigint	total_amount numeric
1	584	4	383427.94
2	1464	2	333836.52
3	1246	3	330611.84
4	1294	2	323736.64
5	1300	3	316956.97

4. Identify customers who have both a loan and a savings account.

```sql
SELECT DISTINCT c.customer_id, c.first_name || ' ' || c.last_name AS full_name
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
JOIN Loans l ON a.account_id = l.account_id
WHERE a.account_type = 'Savings';
```

	customer_id [PK] integer	full_name text

5. Find the average loan amount per loan type.

```
SELECT loan_type, ROUND(AVG(loan_amount), 2) AS avg_loan_amount
FROM Loans
GROUP BY loan_type;
```

	loan_type character varying (50)	avg_loan_amount numeric
1	Auto	2406975.44
2	Personal	2629339.71
3	Home	2630555.55

Advanced Queries

1. Find the top 5 customers with the highest total balance.

```
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS full_name,
       SUM(a.balance) AS total_balance
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
GROUP BY c.customer_id, full_name
ORDER BY total_balance DESC
LIMIT 5;
```

	customer_id [PK] integer	full_name text	total_balance numeric
1	537	Oorja Mallick	3870552.09
2	834	Eshani Zachariah	3342743.88
3	580	Yasmin Dhar	3234720.16
4	94	Shray Mandal	3193394.28
5	742	Advik Bir	3091378.49

2. Detect accounts with no activity in the last 6 months.

```sql
SELECT a.account_id, c.first_name || ' ' || c.last_name AS full_name
FROM Accounts a
JOIN Customers c ON a.customer_id = c.customer_id
WHERE a.account_id NOT IN (
    SELECT DISTINCT account_id
    FROM Transactions
    WHERE transaction_date >= CURRENT_DATE - INTERVAL '6 months'
);
```

	account_id [PK] integer	full_name text
1	9	Mishti Rama
2	12	Eshani Dara
3	13	Aarna Behl
4	17	Mishti Rama
5	22	Shalv Tank

3. Calculate monthly revenue from interest on active loans (assuming simple interest for demonstration).

```sql
SELECT DATE_TRUNC('month', start_date) AS month,
       ROUND(SUM(loan_amount * interest_rate / 100 / 12),2) AS estimated_monthly_interest
FROM Loans
WHERE status = 'Active'
GROUP BY month
ORDER BY month;
```

	month timestamp with time zone	estimated_monthly_interest numeric
1	2020-09-01 00:00:00+05:30	45778.26
2	2020-10-01 00:00:00+05:30	109345.68
3	2020-11-01 00:00:00+05:30	71795.39
4	2020-12-01 00:00:00+05:30	238912.99
5	2021-01-01 00:00:00+05:30	129751.18

4. Generate a summary of deposits vs withdrawals per branch.

```sql
SELECT b.branch_name,
       SUM(CASE WHEN t.transaction_type = 'Deposit' THEN t.amount ELSE 0 END) AS total_deposits,
       SUM(CASE WHEN t.transaction_type = 'Withdrawal' THEN t.amount ELSE 0 END) AS total_withdrawals
FROM Branches b
JOIN Accounts a ON a.customer_id IN (SELECT customer_id FROM Customers) -- simplified mapping
JOIN Transactions t ON t.account_id = a.account_id
GROUP BY b.branch_name
ORDER BY total_deposits DESC;
```

	branch_name character varying (100)	total_deposits numeric	total_withdrawals numeric
1	Surendranagar Dudhrej Branch	676174051.34	671095288.84
2	Thrissur Branch	338087025.67	335547644.42
3	Bettiah Branch	338087025.67	335547644.42
4	South Dumdum Branch	338087025.67	335547644.42
5	Guntakal Branch	338087025.67	335547644.42

5. Identify customers who frequently overdraw (negative balance).

```sql
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS full_name, COUNT(*) AS overdraft_count
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
WHERE a.balance < 0
GROUP BY c.customer_id, full_name
HAVING COUNT(*) >= 1
ORDER BY overdraft_count DESC;
```

	customer_id [PK] integer	full_name text	overdraft_count bigint

Advanced Features

Customer Summary View: total balance, number of accounts, active loans per customer.

```
CREATE OR REPLACE VIEW customer_summary AS
SELECT c.customer_id,
       c.first_name || ' ' || c.last_name AS full_name,
       COUNT(a.account_id) AS total_accounts,
       SUM(a.balance) AS total_balance,
       COUNT(l.loan_id) FILTER (WHERE l.status='Active') AS active_loans
FROM Customers c
LEFT JOIN Accounts a ON c.customer_id = a.customer_id
LEFT JOIN Loans l ON a.account_id = l.account_id
GROUP BY c.customer_id, full_name;
```

```
SELECT * FROM customer_summary ORDER BY total_balance DESC;
```

	customer_id integer	full_name text	total_accounts bigint	total_balance numeric	active_loans bigint
1	266	Kismat Dutta	0	[null]	0
2	420	Jhanvi Bora	0	[null]	0
3	525	Tara Bhasin	0	[null]	0
4	327	Bhavin Barad	0	[null]	0
5	409	Samar Kaur	0	[null]	0

Trigger: Auto-update account balance after transaction

```
CREATE OR REPLACE FUNCTION update_account_balance()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.transaction_type = 'Deposit' THEN
        UPDATE Accounts
        SET balance = balance + NEW.amount
        WHERE account_id = NEW.account_id;
    ELSIF NEW.transaction_type = 'Withdrawal' THEN
        UPDATE Accounts
        SET balance = balance - NEW.amount
        WHERE account_id = NEW.account_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_balance
AFTER INSERT ON Transactions
FOR EACH ROW
EXECUTE FUNCTION update_account_balance();
```

Stored Procedure: Generate monthly statement for a customer

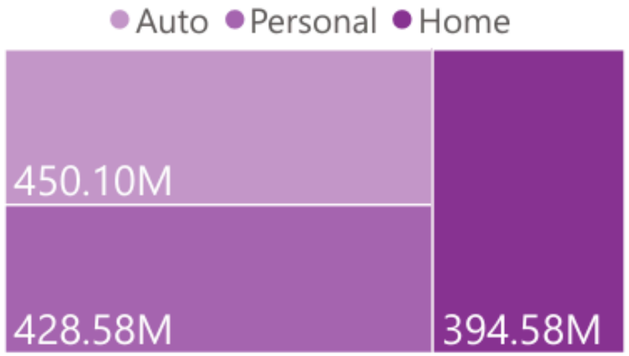
```
CREATE OR REPLACE FUNCTION generate_monthly_statement(p_customer_id INT, p_month DATE)
RETURNS TABLE(
    transaction_id INT,
    account_id INT,
    transaction_type VARCHAR,
    amount NUMERIC,
    transaction_date TIMESTAMP
) AS $$
BEGIN
    RETURN QUERY
    SELECT t.transaction_id, t.account_id, t.transaction_type, t.amount, t.transaction_date
    FROM Transactions t
    JOIN Accounts a ON t.account_id = a.account_id
    WHERE a.customer_id = p_customer_id
      AND DATE_TRUNC('month', t.transaction_date) = DATE_TRUNC('month', p_month)
    ORDER BY t.transaction_date;
END;
$$ LANGUAGE plpgsql;
```

Indexing for Faster Queries

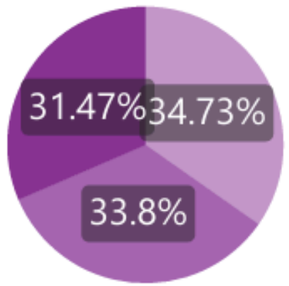
```sql
CREATE INDEX idx_account_customer ON Accounts(customer_id);
CREATE INDEX idx_transaction_account ON Transactions(account_id);
CREATE INDEX idx_loan_account ON Loans(account_id);
```

Banking System Dashboard

Loan Type Distribution by Loan Amount



Account Types



Total Transactions

10K

Total Loans

500

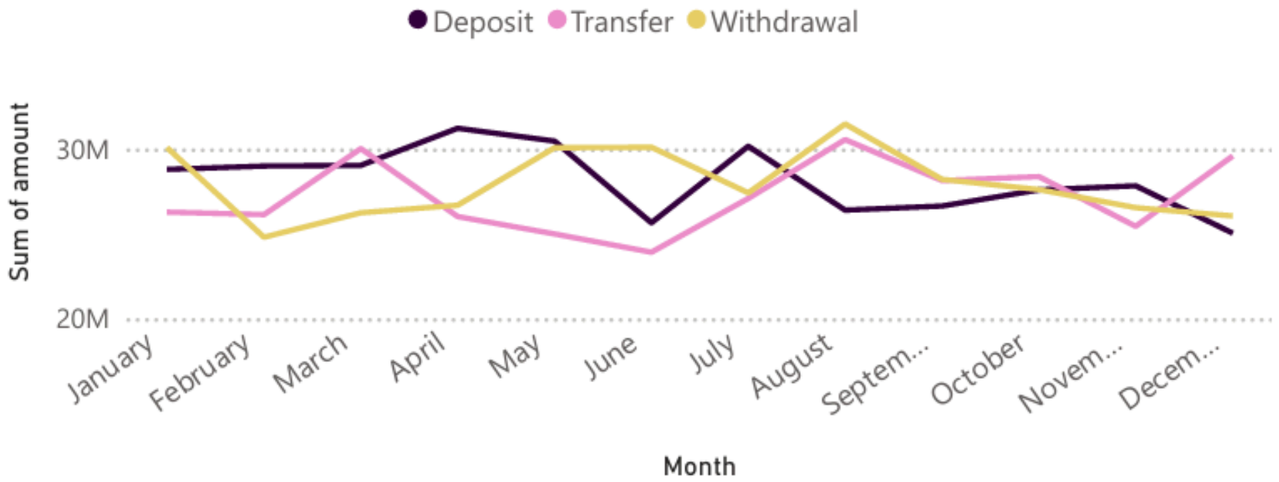
Total Customers

1000

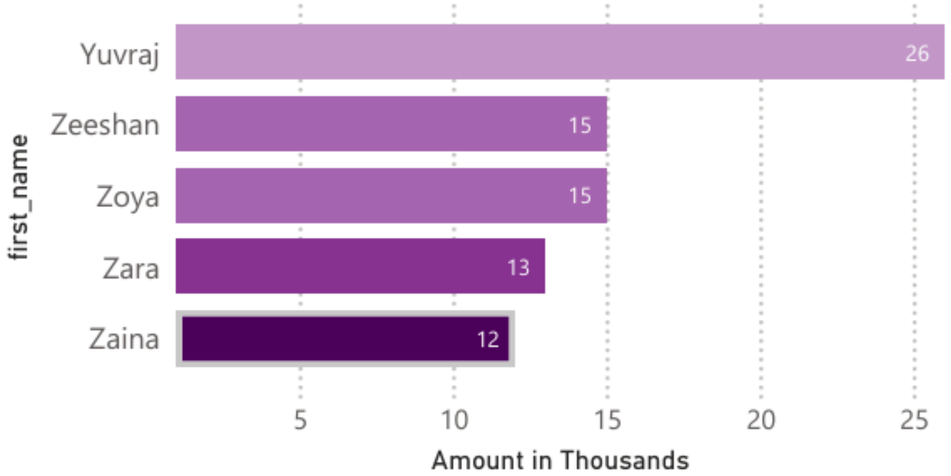
Total Accounts

1500

Transaction per Month with Types

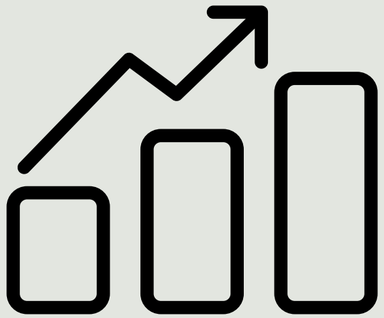


Top 5 Customer by Deposits



BUSINESS INSIGHTS

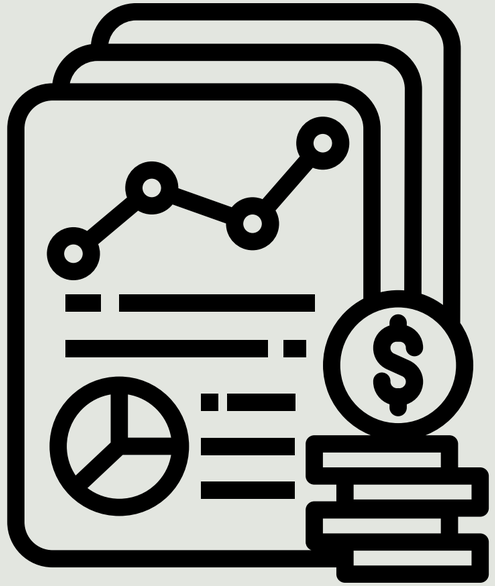
- **Balanced Product Mix** → Savings, Current, and Loan accounts are nearly equal, indicating strong product diversification.
- **Healthy Loan Portfolio** → Loans are evenly spread across auto, personal, and home sectors, reducing credit risk concentration.
- **Steady Deposits but Seasonal Withdrawals** → Deposits are stable, withdrawals/transfers peak in certain months → bank should prepare liquidity buffers.
- **High-Value Customer Dependency** → A small group of customers hold large deposits → poses risk if they move funds elsewhere.
- **Cross-Selling Opportunity** → With customers already holding multiple accounts, targeting loans/insurance/wealth products is viable.



CONCLUSION

The bank shows:

- Strong **customer engagement** (10K transactions).
- **Diverse loan** and **account base**, reducing risk.
- Dependence on **few high-value customers** → requires loyalty programs.
- Seasonal transaction patterns → liquidity planning is key.



Thank You

For your attention