Financial Institution Analytics Project

Credit Card Transaction Analysis & Customer Behavior Insights

Executive Summary

This comprehensive analytics project demonstrates advanced SQL skills through real-world financial data analysis. The project showcases expertise in aggregate functions, window functions, time-series analysis, and strategic business intelligence for a mid-sized financial institution.

Project Overview

As a Senior Data Analyst at Bank, you are tasked with analyzing credit card transaction data to provide actionable insights to the executive team (CEO, COO, CFO). This analysis will drive strategic decisions for the upcoming fiscal year and demonstrate the value of data-driven decision making.

Business Context & Objectives

Bank needs to understand:

- **Customer Spending Patterns**: How do customers use their credit cards across different channels?
- **Geographic Performance**: Which states/regions drive the most transaction volume?
- **Channel Optimization**: How effective are digital vs. traditional banking channels?
- **Branch Performance**: Which physical locations perform best?
- **Risk Management**: What transaction patterns indicate potential fraud or high-risk behavior?
- **Seasonal Trends**: How do spending patterns vary throughout the year?
- **Strategic Planning**: What insights can guide future business decisions?

Database Schema

```
-- Core Tables
transactions (
    transaction_id SERIAL PRIMARY KEY,
    customer_id INTEGER,
    branch_id INTEGER,
    transaction_amount DECIMAL(12,2),
    transaction_date TIMESTAMP,
    channel VARCHAR(50),
    card_type VARCHAR(30)
);

customers (
    customer_id SERIAL PRIMARY KEY,
    state VARCHAR(2),
    credit_score INTEGER,
    account_type VARCHAR(30),
    registration_date DATE
);

branches (
    branch_id SERIAL PRIMARY KEY,
    branch_name VARCHAR(100),
    state VARCHAR(2),
    region VARCHAR(50),
    branch_type VARCHAR(30)
);
```

Analysis Framework

Phase 1: Foundational Analytics

1. Transaction Volume Analysis

```
-- Total credit card transactions processed
SELECT COUNT(*) AS total_transactions,
       SUM(transaction_amount)::DECIMAL(15,2) AS total_volume,
       AVG(transaction_amount)::DECIMAL(10,2) AS avg_transaction_amount
FROM transactions;
```

2. Geographic Performance Analysis

```
-- Transaction volume by state
SELECT c.state,
       COUNT(*) AS transaction_count,
       SUM(t.transaction_amount)::DECIMAL(15,2) AS total_amount,
       AVG(t.transaction_amount)::DECIMAL(10,2) AS avg_amount
FROM transactions t
JOIN customers c ON t.customer_id = c.customer_id
GROUP BY c.state
ORDER BY total_amount DESC;
```

3. Branch Performance Ranking

```sql
-- Top 5 performing branches (excluding digital channels)
SELECT t.branch_id,
       b.branch_name,
       b.state,
       COUNT(*) AS transaction_count,
       SUM(t.transaction_amount)::DECIMAL(15,2) AS total_volume,
       AVG(t.transaction_amount)::DECIMAL(10,2) AS avg_transaction_amount
FROM transactions t
JOIN branches b ON t.branch_id = b.branch_id
WHERE t.channel NOT IN ('online', 'mobile_app', 'digital_wallet')
GROUP BY t.branch_id, b.branch_name, b.state
ORDER BY total_volume DESC
LIMIT 5;
```

4. Channel & Card Type Analysis

```sql
-- Comprehensive analysis using GROUPING SETS
SELECT channel,
       card_type,
       COUNT(*) AS transaction_count,
       SUM(transaction_amount)::DECIMAL(15,2) AS total_amount,
       AVG(transaction_amount)::DECIMAL(10,2) AS avg_amount
FROM transactions
GROUP BY GROUPING SETS (
    (channel),
    (card_type),
    (channel, card_type)
)
ORDER BY channel, card_type;
```

5. Digital Transformation Metrics

```sql
-- Digital vs Traditional channel analysis
SELECT
    SUM(CASE WHEN channel IN ('online', 'mobile_app', 'digital_wallet')
        THEN transaction_amount ELSE 0 END)::DECIMAL(15,2) AS digital_volume,
    SUM(CASE WHEN channel NOT IN ('online', 'mobile_app', 'digital_wallet')
        THEN transaction_amount ELSE 0 END)::DECIMAL(15,2) AS traditional_volume,
    COUNT(CASE WHEN channel IN ('online', 'mobile_app', 'digital_wallet')
        THEN 1 END) AS digital_transactions,
    COUNT(CASE WHEN channel NOT IN ('online', 'mobile_app', 'digital_wallet')
        THEN 1 END) AS traditional_transactions,
    ROUND(
        COUNT(CASE WHEN channel IN ('online', 'mobile_app', 'digital_wallet') THEN 1 END) * 100.0 / COUNT(*), 2
    ) AS digital_adoption_rate
FROM transactions;
```

Phase 2: Advanced Time-Series Analysis

6. Daily Transaction Trends

```
-- Daily transaction volume for 2024
WITH daily_transactions AS (
    SELECT
        transaction_date::date AS transaction_date,
        COUNT(*) AS daily_count,
        SUM(transaction_amount) AS daily_amount
    FROM transactions
    WHERE transaction_date::date BETWEEN '2024-01-01' AND '2024-12-31'
    GROUP BY transaction_date::date
)
SELECT transaction_date,
       daily_count,
       daily_amount::DECIMAL(15,2) AS daily_transaction_amount
FROM daily_transactions
ORDER BY transaction_date;
```

7. Rolling 30-Day Performance Analysis

```
-- Rolling averages for trend analysis
WITH daily_transactions AS (
    SELECT
        transaction_date::date AS transaction_date,
        SUM(transaction_amount) AS daily_amount
    FROM transactions
    WHERE transaction_date::date BETWEEN '2024-01-01' AND '2024-12-31'
    GROUP BY transaction_date::date
),
rolling_metrics AS (
    SELECT transaction_date,
           daily_amount,
           AVG(daily_amount) OVER (
               ORDER BY transaction_date
               ROWS BETWEEN 29 PRECEDING AND CURRENT ROW ) AS rolling_30_day_avg,
           MIN(daily_amount) OVER (
               ORDER BY transaction_date
               ROWS BETWEEN 29 PRECEDING AND CURRENT ROW ) AS rolling_30_day_min,
           MAX(daily_amount) OVER (
               ORDER BY transaction_date
               ROWS BETWEEN 29 PRECEDING AND CURRENT ROW) AS rolling_30_day_max
FROM daily_transactions
)
SELECT transaction_date,
       daily_amount::DECIMAL(15,2) AS daily_amount,
       rolling_30_day_avg::DECIMAL(15,2) AS rolling_average,
       rolling_30_day_min::DECIMAL(15,2) AS rolling_minimum,
       rolling_30_day_max::DECIMAL(15,2) AS rolling_maximum

FROM rolling_metrics
ORDER BY transaction_date;
```

8. Performance Decile Analysis

```
-- Performance benchmarking using deciles
WITH daily_transactions AS (
    SELECT
        transaction_date::date AS transaction_date,
        SUM(transaction_amount) AS daily_amount
    FROM transactions
    WHERE transaction_date::date BETWEEN '2024-01-01' AND '2024-12-31'
    GROUP BY transaction_date::date
),
rolling_avg AS (
    SELECT transaction_date,
           daily_amount,
           AVG(daily_amount) OVER (
               ORDER BY transaction_date
               ROWS BETWEEN 29 PRECEDING AND CURRENT ROW
           ) AS rolling_30_day_avg
    FROM daily_transactions
)
SELECT transaction_date,
       daily_amount::DECIMAL(15,2) AS daily_amount,
       rolling_30_day_avg::DECIMAL(15,2)  AS rolling_average,
       NTILE(10) OVER (ORDER BY rolling_30_day_avg DESC) AS performance_decile,
       CASE
           WHEN NTILE(10) OVER (ORDER BY rolling_30_day_avg DESC) <= 2 THEN 'Top 20%'
           WHEN NTILE(10) OVER (ORDER BY rolling_30_day_avg DESC) <= 5 THEN 'Above Average'
           WHEN NTILE(10) OVER (ORDER BY rolling_30_day_avg DESC) <= 8 THEN 'Below Average'
           ELSE 'Bottom 20%'
       END AS performance_category
FROM rolling_avg
WHERE rolling_30_day_avg IS NOT NULL
```

```
ORDER BY performance_decile, transaction_date;
```

9. Seasonal Pattern Analysis

```
-- Monthly growth trends
SELECT TO_CHAR(transaction_date, 'YYYY-MM') AS month,
       SUM(transaction_amount)::DECIMAL(15,2) AS monthly_total,
       LAG(SUM(transaction_amount)) OVER (
           ORDER BY TO_CHAR(transaction_date, 'YYYY-MM')
       ) AS previous_month_total,
       ROUND(
           ((SUM(transaction_amount) - LAG(SUM(transaction_amount)) OVER (
               ORDER BY TO_CHAR(transaction_date, 'YYYY-MM')
           )) / LAG(SUM(transaction_amount)) OVER (
               ORDER BY TO_CHAR(transaction_date, 'YYYY-MM')
           ) * 100), 2
       ) AS month_over_month_growth
FROM transactions
WHERE transaction_date >= '2024-01-01'
GROUP BY TO_CHAR(transaction_date, 'YYYY-MM')
ORDER BY month;
```

10. Day-of-Week Analysis

```sql
-- Spending patterns by day of week
SELECT TO_CHAR(transaction_date, 'Day') AS day_of_week,
       EXTRACT(DOW FROM transaction_date) AS day_number,
       COUNT(*) AS transaction_count,
       SUM(transaction_amount)::DECIMAL(15,2) AS total_amount,
       AVG(transaction_amount)::DECIMAL(10,2) AS avg_amount,
       RANK() OVER (ORDER BY SUM(transaction_amount) DESC) AS volume_rank,
       RANK() OVER (ORDER BY AVG(transaction_amount) DESC) AS avg_amount_rank
FROM transactions
WHERE transaction_date >= '2024-01-01'
GROUP BY TO_CHAR(transaction_date, 'Day'), EXTRACT(DOW FROM transaction_date)
ORDER BY day_number;
```

Phase 3: Risk Analysis & Customer Intelligence

11. Customer Segmentation Analysis

```sql
-- Customer value segmentation
WITH customer_metrics AS (
    SELECT customer_id,
           COUNT(*) AS transaction_frequency,
           SUM(transaction_amount) AS total_spending,
           AVG(transaction_amount) AS avg_transaction_amount,
           STDDEV(transaction_amount) AS spending_volatility,
           MAX(transaction_amount) AS max_transaction,
           MIN(transaction_amount) AS min_transaction
    FROM transactions
    WHERE transaction_date >= '2024-01-01'
    GROUP BY customer_id
)
SELECT
    CASE
        WHEN avg_transaction_amount >= 1000 THEN 'Premium'
        WHEN avg_transaction_amount >= 500 THEN 'High Value'
        WHEN avg_transaction_amount >= 100 THEN 'Medium Value'
        ELSE 'Standard'
    END AS customer_segment,
    COUNT(*) AS customer_count,
    AVG(total_spending)::DECIMAL(12,2) AS avg_total_spending,
    AVG(transaction_frequency)::DECIMAL(8,2) AS avg_frequency,
    AVG(avg_transaction_amount)::DECIMAL(10,2) AS segment_avg_amount
FROM customer_metrics
GROUP BY
    CASE
        WHEN avg_transaction_amount >= 1000 THEN 'Premium'
        WHEN avg_transaction_amount >= 500 THEN 'High Value'
        WHEN avg_transaction_amount >= 100 THEN 'Medium Value'
        ELSE 'Standard'
        END
ORDER BY segment_avg_amount DESC;
```

12. Risk Pattern Detection

```
-- High-frequency transaction analysis for risk assessment
SELECT customer_id,
       COUNT(*) AS transaction_frequency,
       SUM(transaction_amount)::DECIMAL(15,2) AS total_amount,
       AVG(transaction_amount)::DECIMAL(10,2) AS avg_amount,
       MAX(transaction_amount) AS max_transaction,
       MIN(transaction_amount) AS min_transaction,
       STDDEV(transaction_amount)::DECIMAL(10,2) AS amount_volatility,
       COUNT(DISTINCT channel) AS channel_diversity,
       CASE
           WHEN COUNT(*) > 50 AND STDDEV(transaction_amount) > 1000 THEN 'High Risk'
           WHEN COUNT(*) > 30 AND STDDEV(transaction_amount) > 500 THEN 'Medium Risk'
           ELSE 'Low Risk'
       END AS risk_category
FROM transactions
WHERE transaction_date >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY customer_id
HAVING COUNT(*) > 15
ORDER BY transaction_frequency DESC, amount_volatility DESC;
```

Key Performance Indicators (KPIs)

Metric	**Description**	**Business Value**
Total Transaction Volume	Sum of all transaction amounts	Revenue tracking
Digital Adoption Rate	% of transactions through digital channels	Digital transformation progress
Average Transaction Value	Mean transaction amount by segment	Customer value assessment
Rolling 30-Day Average	Smoothed daily performance trend	Operational stability
Performance Deciles	Daily performance ranking	Benchmarking and goal setting
Customer Lifetime Value	Total spending per customer	Customer retention strategy
Channel Effectiveness	Revenue per channel	Resource allocation
Risk Score	Fraud/unusual pattern indicator	Risk management

Business Insights & Recommendations

1. Channel Optimization

- **Digital channels** show higher average transaction values
- **Recommendation**: Invest in mobile app improvements and online user experience

2. Geographic Expansion

- **High-performing states** indicate market opportunities
- **Recommendation**: Focus marketing efforts on underperforming regions

3. Customer Segmentation

- **Premium customers** drive disproportionate revenue
- **Recommendation**: Develop targeted retention programs for high-value segments

4. Risk Management

- **Transaction patterns** can predict fraudulent behavior

- **Recommendation**: Implement real-time monitoring for high-risk profiles

5. Seasonal Planning

- **Monthly trends** reveal predictable spending patterns
- **Recommendation**: Adjust staffing and marketing campaigns based on seasonal data

Technical Implementation Notes

- **Data Types**: Use `DECIMAL(15,2)` for financial calculations to ensure precision
- **Performance**: Implement proper indexing on `transaction_date`, `customer_id`, and `channel`
- **Null Handling**: Include `COALESCE()` functions for robust data processing
- **Window Functions**: Leverage `PARTITION BY` for efficient grouped calculations
- **Time Zones**: Ensure consistent date handling across all analyses

Project Deliverables

1. **Executive Dashboard**: High-level KPIs and trends
2. **Detailed Analytics Report**: Comprehensive findings with visualizations
3. **Risk Assessment Summary**: Customer risk profiles and recommendations
4. **Strategic Recommendations**: Data-driven business insights
5. **SQL Code Repository**: Well-documented, reusable queries
6. **Performance Benchmarks**: Baseline metrics for future comparison

This project demonstrates proficiency in:

- Advanced SQL analytics
- Time-series analysis
- Customer segmentation

- Risk assessment
- Business intelligence
- Strategic thinking
- Data-driven decision making

This analysis showcases the ability to transform raw transaction data into actionable business intelligence, demonstrating both technical SQL expertise and strategic business acumen essential for senior data analyst roles.