Day 8: Banking Fraud Detection System

Domain: Finance / Fraud Analytics

Difficulty Level: Advanced (Complex Relationships & Analytical Queries)

Goal: Identify and analyze suspicious banking transactions using SQL techniques involving pattern detection, window functions, and aggregations.

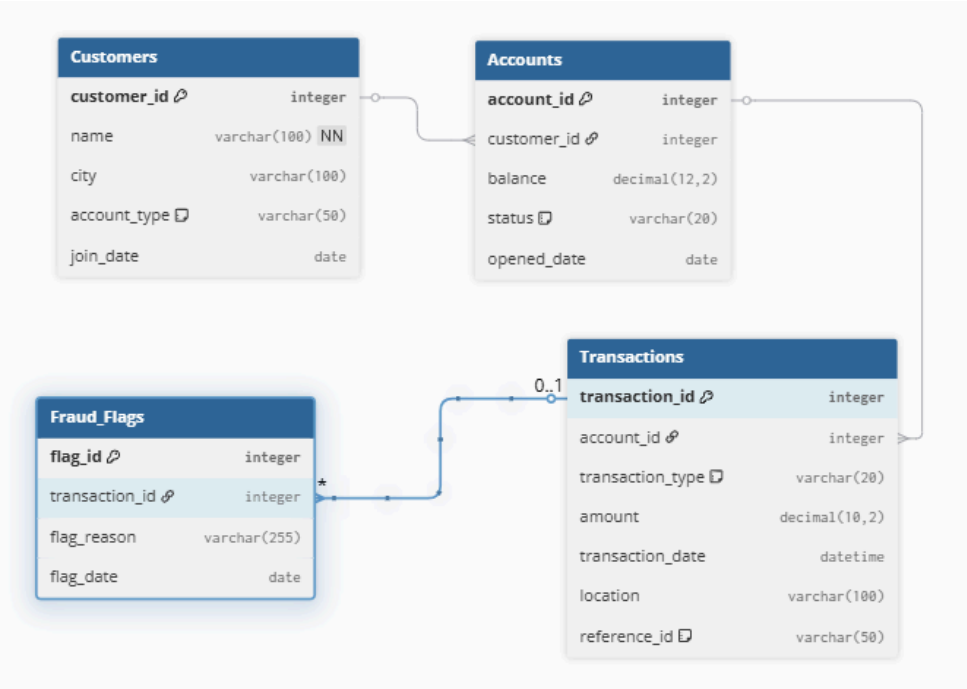
🏦 Scenario Description

A large private bank wants to **detect potentially fraudulent transactions** across its customers. Fraudulent behavior often shows patterns like:

- Frequent high-value transfers in a short time.
- Transactions in multiple cities within minutes.
- Large cash withdrawals just after large deposits.

Your task is to analyze the data to support the **Fraud Investigation Unit (FIU)** with insights using SQL queries.

🧩 Database Schema Overview



Tables

1 Customers

Column Name	Data Type	Constraints	Description
customer_id	INT	PRIMARY KEY	Unique ID for each customer

Column Name	Data Type	Constraints	Description
name	VARCHAR(100)	NOT NULL	Full name of customer
city	VARCHAR(100)		Home city
account_type	VARCHAR(50)		Type of account (Savings, Current, etc.)
join_date	DATE		Date of joining the bank

2️⃣ Accounts

Column Name	Data Type	Constraints	Description
account_id	INT	PRIMARY KEY	Unique account number
customer_id	INT	FOREIGN KEY → Customers(customer_id)	Account holder
balance	DECIMAL(12,2)		Current account balance
status	VARCHAR(20)		Active/Inactive
opened_date	DATE		Date when account was opened

3️⃣ Transactions

Column Name	Data Type	Constraints	Description
transaction_id	INT	PRIMARY KEY	Unique ID for each transaction
account_id	INT	FOREIGN KEY → Accounts(account_id)	Linked account
transaction_type	VARCHAR(20)		'Deposit', 'Withdrawal', or 'Transfer'
amount	DECIMAL(10,2)		Transaction amount
transaction_date	DATETIME		Timestamp of transaction
location	VARCHAR(100)		City where transaction occurred
reference_id	VARCHAR(50)		External reference (may repeat for linked transactions)

4️⃣ Fraud_Flags

Column Name	Data Type	Constraints	Description
flag_id	INT	PRIMARY KEY	Unique flag record
transaction_id	INT	FOREIGN KEY → Transactions(transaction_id)	Flagged transaction
flag_reason	VARCHAR(255)		Reason for flagging
flag_date	DATE		When it was flagged

🧠 ERD (Textual Representation)

```
Customers (1)——(M) Accounts (1)——(M) Transactions (1)——(M) Fraud_Flags
  | customer_id       | account_id       | transaction_id       | flag_id
```

Relationships:

- One customer can have multiple accounts.

- One account can have multiple transactions.
- A transaction can have zero or one fraud flag.

🧾 Sample Data

Customers

customer_id	name	city	account_type	join_date
1	John Doe	Mumbai	Savings	2019-02-10
2	Priya Singh	Delhi	Current	2020-04-15
3	Rohan Patel	Bengaluru	Savings	2021-01-09
4	Neha Sharma	Mumbai	Salary	2022-07-23

Accounts

account_id	customer_id	balance	status	opened_date
101	1	120000.00	Active	2019-02-10
102	2	80000.00	Active	2020-04-15
103	3	35000.00	Active	2021-01-09
104	4	45000.00	Inactive	2022-07-23

Transactions

transaction_id	account_id	transaction_type	amount	transaction_date	location	reference_id
1001	101	Deposit	50000.00	2024-12-01 09:15:00	Mumbai	REF001
1002	101	Withdrawal	49000.00	2024-12-01 09:25:00	Delhi	REF002
1003	102	Transfer	15000.00	2024-12-01 11:30:00	Delhi	REF003
1004	103	Deposit	8000.00	2024-12-02 08:00:00	Bengaluru	REF004
1005	103	Withdrawal	7500.00	2024-12-02 08:15:00	Chennai	REF005
1006	104	Deposit	25000.00	2024-12-03 09:45:00	Mumbai	REF006

Fraud_Flags

flag_id	transaction_id	flag_reason	flag_date
1	1002	High-value withdrawal from different city	2024-12-01
2	1005	Multiple cities in short duration	2024-12-02

🧩 SQL Questions

🟢 Easy:

List all transactions greater than ₹10,000 with customer details.

```
SELECT c.name, c.city, t.transaction_id, t.amount, t.transaction_type
FROM Transactions t
JOIN Accounts a ON t.account_id = a.account_id
JOIN Customers c ON a.customer_id = c.customer_id
WHERE t.amount > 10000;
```

Explanation:

Simple `JOIN` across three tables to get context and filter on transaction value.

🟡 Medium:

Find customers who made **transactions in multiple cities** on the same day.

```
SELECT c.customer_id, c.name, DATE(t.transaction_date) AS txn_day,
    COUNT(DISTINCT t.location) AS cities_visited
FROM Transactions t
JOIN Accounts a ON t.account_id = a.account_id
JOIN Customers c ON a.customer_id = c.customer_id
GROUP BY c.customer_id, txn_day
HAVING COUNT(DISTINCT t.location) > 1;
```

Explanation:

Grouping by customer and day; counting distinct locations reveals multi-city activity — a possible fraud indicator.

🔵 Hard:

Detect transactions where a **withdrawal occurred within 15 minutes** of a deposit in the same account.

```
SELECT t1.transaction_id AS deposit_id, t2.transaction_id AS withdrawal_id,
    t1.account_id, t1.amount AS deposit_amt, t2.amount AS withdraw_amt,
    TIMESTAMPDIFF(MINUTE, t1.transaction_date, t2.transaction_date) AS time_gap
FROM Transactions t1
JOIN Transactions t2
  ON t1.account_id = t2.account_id
 AND t1.transaction_type = 'Deposit'
 AND t2.transaction_type = 'Withdrawal'
 AND t2.transaction_date > t1.transaction_date
WHERE TIMESTAMPDIFF(MINUTE, t1.transaction_date, t2.transaction_date) <= 15;
```

Explanation:

Self-join on `Transactions` with a time-based condition — a classic fraud detection pattern.

🔴 Difficult:

Find **top 2 customers** with the **highest number of fraud flags**.

```
SELECT c.customer_id, c.name, COUNT(f.flag_id) AS total_flags
FROM Fraud_Flags f
JOIN Transactions t ON f.transaction_id = t.transaction_id
JOIN Accounts a ON t.account_id = a.account_id
JOIN Customers c ON a.customer_id = c.customer_id
GROUP BY c.customer_id
ORDER BY total_flags DESC
LIMIT 2;
```

Explanation:

Aggregates across joined tables with ordering and limit. Helps prioritize fraud investigations.

⚫ Expert:

Identify **suspicious transaction sequences**: same customer, same amount, within 10 minutes, across multiple locations.

```
SELECT c.name, t1.transaction_id, t2.transaction_id,
       t1.amount, t1.location AS loc1, t2.location AS loc2,
       TIMESTAMPDIFF(MINUTE, t1.transaction_date, t2.transaction_date) AS gap_min
FROM Transactions t1
JOIN Transactions t2
  ON t1.account_id = t2.account_id
 AND t1.transaction_id < t2.transaction_id
 AND t1.amount = t2.amount
 AND t1.location <> t2.location
 AND ABS(TIMESTAMPDIFF(MINUTE, t1.transaction_date, t2.transaction_date)) <= 10
JOIN Accounts a ON t1.account_id = a.account_id
JOIN Customers c ON a.customer_id = c.customer_id;
```

Explanation:

Detects mirrored or duplicate-amount transactions in close intervals — strong fraud signal.

💡 Tips & Optimization

- Create **indexes** on `transaction_date`, `account_id`, and `transaction_type` for faster temporal joins.
- Use **window functions** for next-day fraud modeling (not shown here).
- Periodically **archive old transactions** to improve performance.
- Normalize locations and ensure timezone consistency in multi-branch operations.