STAGE 3

IBM AML SYNTHETIC TRANSACTION DATABASE

University of Miami
CSC423 - Database Systems (Spring 2025)

Pavel Stepanov, Sean McHale, Alexander Niejadlik

HIGH-LEVEL DATABASE DESCRIPTION:

The **IBM AML Synthetic Transaction Database** is a simulated financial transaction dataset designed for detecting and analyzing money laundering patterns. This database was created to study various anti-money laundering (AML) techniques and provides researchers and students with a comprehensive dataset that includes both legitimate and illicit financial transactions.

The database originates from **IBM** and was designed to simulate real-world money laundering scenarios across different banking institutions. It includes various laundering patterns such as FAN-IN, FAN-OUT, GATHER-SCATTER, SCATTER-GATHER, RANDOM, STACK, BIPARTITE, and CYCLE, allowing for thorough analysis of different money laundering techniques.

Source URL: https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml

The database contains 5,078,345 financial transactions across 515,080 bank accounts from 30,528 bank branches. It's organized into four main tables: BANK, LAUNDERING_PATTERN, BANK_ACCOUNT, and FINANCIAL_TRANSACTION, with three specialized views: SOURCE_CUSTOMER, TELLER, and AUDITOR, each designed for different user roles and access levels.

FINAL ER DIAGRAM:

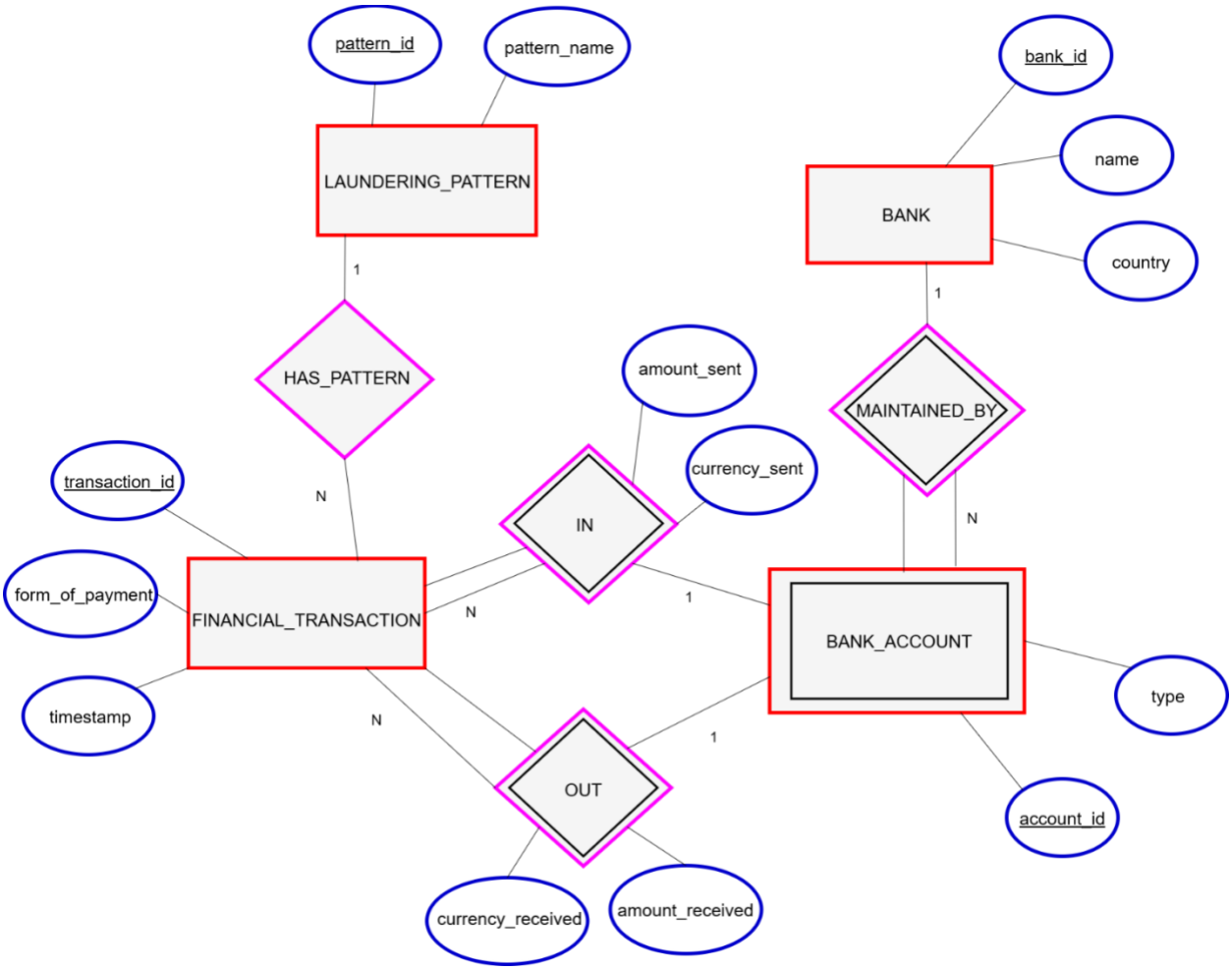


Figure 1

FINAL RELATIONAL SCHEMA:

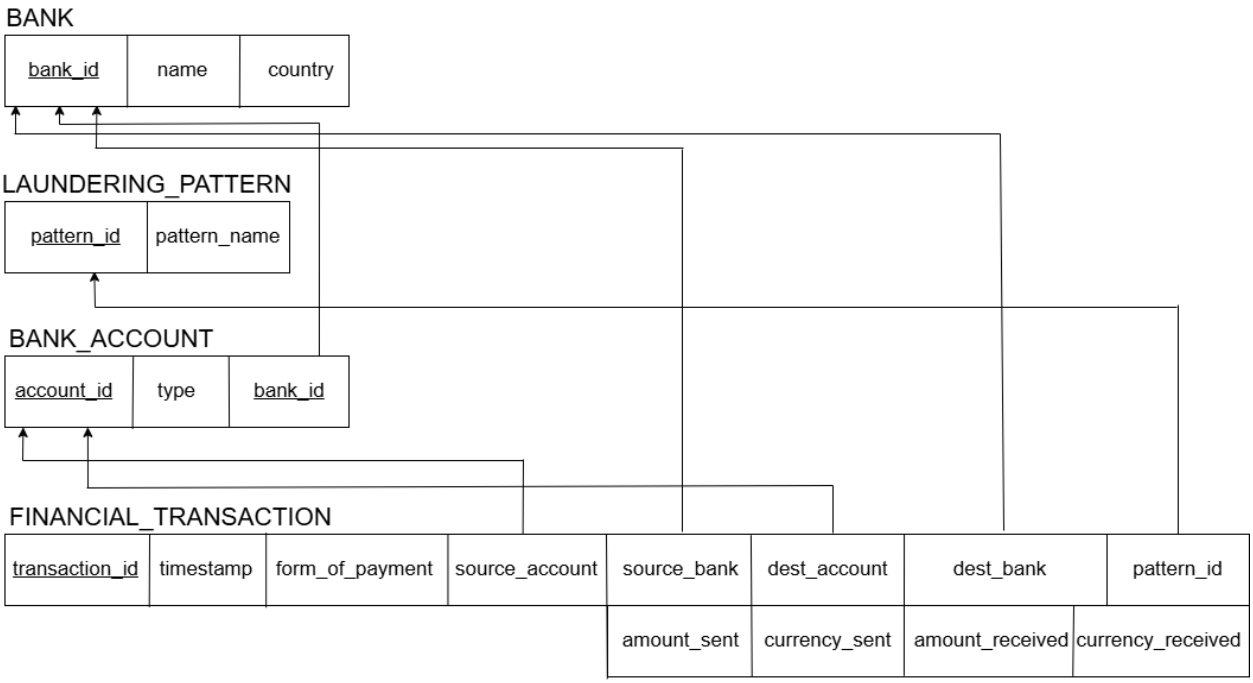


Figure 2

EXPLANATION:

BANK (bank_id, name, country)

- 30,528 tuples
- bank_id
  - numeric
  - the primary key and uniquely identifies each individual bank branch via a numerical code
- name
  - text
  - the name of each bank branch. This was originally generic identification numbers with tens of thousands of unique banks. We deemed it more natural to treat them as individual branches. Naming done for readability and accessibility. Done using random assignment from a small list of banks.
- country
  - text
  - the country of origin for each bank branch. Randomly assigned country to each bank branch using a small list of countries.
- bankCountry
  - Non-unique index on country

- bankName
    - unique index on name

LAUNDERING_PATTERN (pattern_id, pattern_name)

- 9 tuples
- pattern_id
    - integer
    - the primary key and uniquely identifies each form of laundering with a numerical code
    - pattern_name is a bit annoying and inefficient to filter since they're strings and integer assignment is easy in this case
- pattern_name
    - text
    - the name of the type of laundering pattern occuring. None is the default value, meaning no illegal activity occurred. Used the 8 forms of laundering described in the original dataset: FAN-IN, FAN-OUT, GATHER-SCATTER, SCATTER-GATHER, RANDOM, STACK, BIPARTITE, and CYCLE
- No indexes
    - Table very small, so no need for indexing

BANK_ACCOUNT (account_id, type, bank_id)

- 515,080 tuples
- account_id
    - text
    - the primary key and uniquely identifies each individual account
- type
    - text
    - the kind of account it is: individual, corporate, or government
    - Uses random assignment with 3 possible choices
- bank_id
    - numeric
    - foreign key which tells us which bank each account belongs to
    - An account cannot exist without a bank
- accountBID
    - non-unique index on bank_id
- accountType
    - non-unique index on type

FINANCIAL_TRANSACTION (transaction_id, timestamp, form_of_payment, amount_sent, currency_sent, amount_received, currency_received, source_account, source_bank, dest_account, dest_bank, pattern_id)

- 5,078,345 tuples
- transaction_id
    - auto-incrementing integer
    - the primary key and uniquely identifies each individual transaction
    - Reasoning for this: multi-attribute keys are less reliable (especially for normalization), timestamp doesn't include seconds so overlap is possible if it's used, and Professor recommended using a transaction_id in meeting with student
- timestamp
    - DATETIME (YYYY-MM-DD HH:MM:SS)
    - Tells us when the transaction happened
- Form_of_payment
        - Text
        - How was the transaction made? For example, cheque or credit card?
- Amount_sent
    - Decimal
    - How much money did the source send
- Currency_sent
    - Text
    - What currency did the source send the money in
- Amount_received
    - Decimal
    - How much money did the destination receive
- Currency_received
    - Text
    - What currency did the source receive the money in
- Source_account
    - Text
    - Foreign key that gives us the origin of the transaction via the BANK_ACCOUNT table
- Source_bank
    - Text
    - Foreign key that tells us the bank which the source_account belongs to via BANK
- Dest_account
    - Text
    - Foreign key that gives us the destination of the transaction via the BANK_ACCOUNT table
- Dest_bank
    - Text
    - Foreign key that tells us the bank which the dest_account belongs to via BANK

- Pattern_id
    - Integer
    - Foreign key from laundering pattern. Identifies each transaction with a form of laundering (extremely important)
- transSA
    - non-unique index on source_account
- transDA
    - non-unique index on dest_account
- transSB
    - non-unique index on source_bank
- transDB
    - non-unique index on dest_bank
- transPATID
    - non-unique index on pattern_id
- transTS
    - non-unique index on timestamp
- transSource
    - non-unique index on source_account and source_bank
- transDest
    - non-unique index on dest_account and dest_bank
- transAmounts
    - non-unique index on amount_sent and amount_received

VIEW DESCRIPTION

1. SOURCE_CUSTOMER

This view is for a bank account holder who wishes to send money to another account or themselves. We created this one as a security measure and for personalization. We don't want a person sending money to see data related to illicit activity, private bank information, or information about the dest_account beyond the ID they're sending money to. We also use aliasing to make the query more personal since all the data is about themselves. For example, "T.source_account AS your_account" works since we intend the source_account to belong exclusively to whoever is using the view (assuming they properly filter in the WHERE clause). Finally, two new attributes are created. The first tells the customer whether their transaction was a deposit or a transfer (readability) and the second tells the customer whether their transaction was successfully processed (important notification for a customer).

2. TELLER

This view is for a bank teller at the bank. As an employee, they should be able to see all details about a transaction except the money laundering data. We deemed that information beyond the scope of the job of a teller. Any criminal activity would be investigated and handled by a higher authority figure. A teller should concern themselves solely with the transactions themselves. Therefore, they can see all the information about the source and destination accounts including all details about their banks. For the sake of readability, we decided to inform the teller whether the transaction was a deposit or transfer. We also included whether there was a currency exchange as part of the transaction and the total amount of processing fees.

3. AUDITOR

This view is what someone whose investigating money laundering patterns among banks sees. Assume this is someone who works for some international institutions or government agency. They can see everything a TELLER sees plus the laundering data. We add an attribute which tells us easily whether a transaction was "LEGAL" or "ILLICIT" since the pattern_id doesn't make it obvious, and the pattern_names are annoying to filter.

DOCUMENTED SQL CODE AND OUTPUT:

1

List deposit history for accounts 8000EBD30, 8016BBF90, and 800128AC0

Description:

Shows all deposit transactions for these accounts ordered by most recent first

Output:

timestamp, account, account type, bank, amount, currency, payment method, recipient, status

```
SELECT
    -- information that a source_customer has access to
    timestamp,
    your_account,
    your_account_type,
    your_bank,
    type_of_transaction,
    amount_sent,
    currency_sent,
    form_of_payment,
    sent_to,
    transaction_status

FROM
    -- Using SOURCE_CUSTOMER view
    SOURCE_CUSTOMER

WHERE
    -- For a single customer who wants to see their deposit records
    (your_account = '8000EBD30' or your_account = '8016BBF90' or your_account = '800128AC0') and type_of_transaction = 'Deposit'

ORDER BY
    -- Recent transactions first
    timestamp DESC;
```

	timestamp	your_accour	your_accour	your_ba	type_of_	amount_se	currency_	form_of_paym	sent_to	transaction_status
1	2022-09-09 00:02:00	8016BBF90	Government	002991	Deposit	94.94	US Dollar	ACH	8016BBF90	Sent Successfully
2	2022-09-03 00:15:00	8016BBF90	Government	002991	Deposit	36.04	US Dollar	ACH	8016BBF90	Sent Successfully
3	2022-09-01 16:11:00	800128AC0	Government	001	Deposit	166461.98	US Dollar	Reinvestment	800128AC0	Sent Successfully
4	2022-09-01 00:22:00	800128AC0	Government	001	Deposit	22.9	US Dollar	Reinvestment	800128AC0	Sent Successfully
5	2022-09-01 00:20:00	8000EBD30	Individual	010	Deposit	3697.34	US Dollar	Reinvestment	8000EBD30	Sent Successfully

2

Bank with highest rate of illicit activity

Description:

Identifies the bank with the highest percentage of suspicious transactions

Output:

bank name, illicit rate, number of illicit transactions, total transactions

```sql
SELECT
    B.name AS bank_name,
    AggregatedRates.illicit_rate,
    AggregatedRates.illicit_transactions,
    AggregatedRates.total_transactions
FROM (
    SELECT
        bank_id,
        COUNT(*) AS total_transactions,
        SUM(is_illicit) AS illicit_transactions,
        -- Calculate rate as percentage
        CAST(SUM(is_illicit) AS REAL) * 100.0 / COUNT(*) AS illicit_rate
    FROM (
        -- list all banks
        SELECT
            source_bank AS bank_id,
            CASE
                WHEN pattern_id != 10 THEN 1 -- Illicit
                ELSE 0
            END AS is_illicit
        FROM FINANCIAL_TRANSACTION

        UNION ALL -- Combine source and destination transactions

        SELECT
            dest_bank AS bank_id,
            CASE
                WHEN pattern_id != 10 THEN 1 -- Illicit
                ELSE 0
            END AS is_illicit
        FROM FINANCIAL_TRANSACTION
    ) AS BankParticipationData

    GROUP BY bank_id

    HAVING COUNT(*) > 0
) AS AggregatedRates
JOIN BANK B ON AggregatedRates.bank_id = B.bank_id
ORDER BY AggregatedRates.illicit_rate DESC
LIMIT 1;
```

bank_name	illicit_rate	illicit_transactions	total_transactions
HSBC 14980	100	2	2

3

Individual vs corporate accounts in laundering

Description:

Compares involvement of individual and corporate accounts in laundering activities

Output:

account type, total accounts, implicated accounts, implication rate percentage

```sql
WITH IllicitTransactionAccounts AS (
    -- Get a DISTINCT list of all account IDs involved in illicit transactions
    SELECT DISTINCT source_account AS account_id
    FROM FINANCIAL_TRANSACTION
    WHERE pattern_id != 10 AND source_account IS NOT NULL

    UNION -- removes duplicates

    SELECT DISTINCT dest_account AS account_id
    FROM FINANCIAL_TRANSACTION
    WHERE pattern_id != 10 AND dest_account IS NOT NULL

), AccountTypeStats AS (
    -- Calculate total accounts and implicated accounts for relevant types
    SELECT
        BA.type,
        COUNT(DISTINCT BA.account_id) AS total_accounts,
        COUNT(DISTINCT I.account_id) AS implicated_accounts
    FROM
        BANK_ACCOUNT BA
    LEFT JOIN
        IllicitTransactionAccounts I ON BA.account_id = I.account_id
    WHERE
        BA.type IN ('Individual', 'Corporate')
    GROUP BY
        BA.type
)
-- Final SELECT to calculate and compare the implication rates
SELECT
    type,
    total_accounts,
    implicated_accounts,
    CASE
        WHEN total_accounts > 0 THEN
            CAST(implicated_accounts AS REAL) * 100.0 / total_accounts
        ELSE
            0.0
    END AS implication_rate_percent
FROM
    AccountTypeStats
ORDER BY
    implication_rate_percent DESC;
```

	type	total_accounts	implicated_accounts	implication_rate_percent
1	Corporate	171844	124995	72.73748283326738
2	Individual	171919	124847	72.6196639114932

4

Top 3 countries with most launderers

Description:

Lists three countries with the highest number of unique laundering accounts

Output:

country, number of unique launderer accounts (ordered alphabetically)

```sql
WITH LaunderingAccounts AS (
    SELECT DISTINCT source_account AS account_id
    FROM FINANCIAL_TRANSACTION
    WHERE pattern_id != 10 AND source_account IS NOT NULL

    UNION

    SELECT DISTINCT dest_account AS account_id
    FROM FINANCIAL_TRANSACTION
    WHERE pattern_id != 10 AND dest_account IS NOT NULL

), CountryLaundererCounts AS (
    SELECT
        B.country,
        COUNT(DISTINCT BA.account_id) AS unique_launderer_accounts
    FROM
        LaunderingAccounts L
    JOIN
        BANK_ACCOUNT BA ON L.account_id = BA.account_id
    JOIN
        BANK B ON BA.bank_id = B.bank_id
    WHERE
        B.country IS NOT NULL
    GROUP BY
        B.country

), Top3CountriesByCount AS (
    SELECT
        country,
        unique_launderer_accounts
    FROM
        CountryLaundererCounts
    ORDER BY
        unique_launderer_accounts DESC
    LIMIT 3
)

SELECT
    country,
    unique_launderer_accounts
FROM
    Top3CountriesByCount
ORDER BY
    country ASC;
```

country	unique_launderer _accounts
China	50540
Germany	49209
Russia	53473

5	Day and time with most laundering activity **Description:** Finds the single timestamp with highest number of illicit transactions **Output:** timestamp, number of laundering instances

```sql
--5. On what day and at what time did the most laundering occur?
SELECT
    timestamp,
    COUNT(*) AS launderingInstanceCounter
FROM
    FINANCIAL_TRANSACTION
WHERE
    pattern_id IS NOT NULL
    AND timestamp IS NOT NULL
GROUP BY
    timestamp
ORDER BY
    launderingInstanceCounter DESC
LIMIT 1;
```

timestamp	launderingInstanceCounter
2022-09-01 00:04:00	11193

6 Laundering patterns by frequency

Description:

Lists all laundering patterns ordered by how often they occur

Output:

pattern name, frequency count (most to least common)

```sql
SELECT
    L.pattern_name,
    COUNT(T.transaction_id) AS rate
FROM
    FINANCIAL_TRANSACTION T
INNER JOIN
    LAUNDERING_PATTERN L ON T.pattern_id = L.pattern_id
GROUP BY
    L.pattern_name
ORDER BY
    rate DESC;
```

pattern_name	rate
None	3554931
FAN-OUT	191173
RANDOM	191141
STACK	190729
CYCLE	190712
SCATTER-GATHER	190086
BIPARTITE	190086
GATHER-SCATTER	189893
FAN-IN	189594

7	Most common payment form used by launderers **Description:** Identifies the payment method most frequently used in illicit transactions **Output:** form of payment, frequency count

```sql
--7. What is the most common form of payment that launderers use?
SELECT
    form_of_payment,
    COUNT(*) AS rate
FROM
    FINANCIAL_TRANSACTION
WHERE
    pattern_id IS NOT NULL
    AND form_of_payment IS NOT NULL
GROUP BY
    form_of_payment
ORDER BY
    rate DESC
LIMIT 1;
```

form_of_payment	rate
Cheque	1864331

8

Total money sent between Sept 3-8, 2022

Description:

Sums all money sent during this period grouped by currency

Output:

currency, total amount

```
SELECT
    currency_sent,
    SUM(amount_sent) AS totalCurrencyValue
FROM
    FINANCIAL_TRANSACTION
WHERE
    DATE(timestamp) >= '2022-09-03'
    AND DATE(timestamp) <= '2022-09-08'
    AND amount_sent IS NOT NULL
    AND currency_sent IS NOT NULL
GROUP BY
    currency_sent
ORDER BY
    currency_sent;
```

	currency_sent	totalCurrencyValue
1	Australian Dollar	13180515651.86
2	Bitcoin	951495.28
3	Brazil Real	207714939108.55002
4	Canadian Dollar	38419726051.42
5	Euro	74940874336.37
6	Mexican Peso	176272498475
7	Ruble	1745105489116.81
8	Rupee	2710319875979.76
9	Saudi Riyal	27454124869.18
10	Shekel	40111650707.97
11	Swiss Franc	21408140914.01
12	UK Pound	12723474771.54
13	US Dollar	177337161860.08
14	Yen	2830159198734.31
15	Yuan	100300280890.98

9

Currencies used in illicit activity

Description:

Lists currencies by how often they're used in suspicious transactionsfirst

Output:

currency, frequency count (most to least used)

```sql
WITH ImplicatedCurrency AS (
    SELECT currency_sent AS currency FROM FINANCIAL_TRANSACTION
    WHERE pattern_id IS NOT NULL AND currency_sent IS NOT NULL
    UNION ALL
    SELECT currency_received AS currency FROM FINANCIAL_TRANSACTION
    WHERE pattern_id IS NOT NULL AND currency_received IS NOT NULL
)
SELECT
    currency,
    COUNT(*) AS rate
FROM
    ImplicatedCurrency
GROUP BY
    currency
ORDER BY
    rate DESC;
```

	currency	rate
1	US Dollar	3774513
2	Euro	2340314
3	Swiss Franc	472744
4	Yuan	420303
5	Shekel	387172
6	Rupee	382267
7	UK Pound	361993
8	Ruble	312539
9	Yen	311528
10	Bitcoin	294217
11	Canadian Dollar	281399
12	Australian Dollar	275280
13	Mexican Peso	221189
14	Saudi Riyal	178985
15	Brazil Real	142247

10 Accounts acting as middlemen

Description:

Finds accounts that both send and receive suspicious money

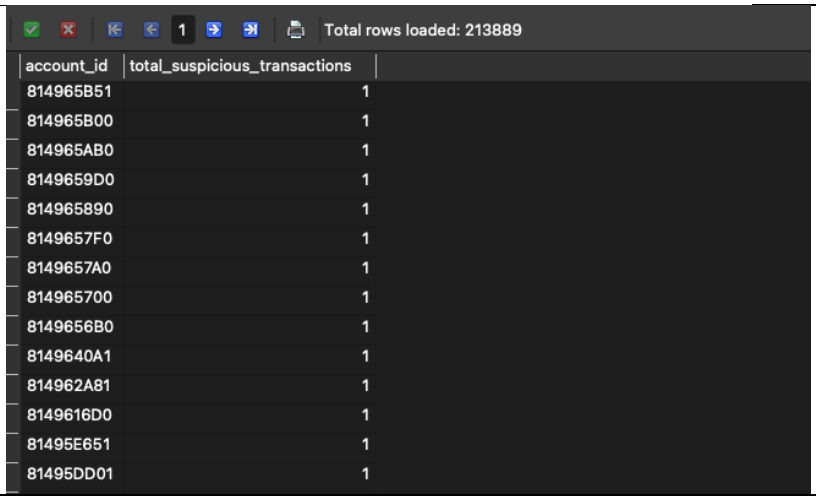
Output:

account ID, total suspicious transactions (ordered by frequency)

```sql
SELECT
    account_id,
    COUNT(*) as total_suspicious_transactions
FROM (
    SELECT source_account as account_id
    FROM FINANCIAL_TRANSACTION
    WHERE pattern_id != 10  -- Not normal transaction

    INTERSECT

    SELECT dest_account as account_id
    FROM FINANCIAL_TRANSACTION
    WHERE pattern_id != 10  -- Not normal transaction
)
GROUP BY account_id
ORDER BY total_suspicious_transactions DESC;
```



Total rows loaded: 213889

account_id	total_suspicious_transactions
814965B51	1
814965B00	1
814965AB0	1
8149659D0	1
814965890	1
8149657F0	1
8149657A0	1
814965700	1
8149656B0	1
8149640A1	1
814962A81	1
8149616D0	1
81495E651	1
81495DD01	1