Data Management

2024/2025

NoSQL Project

ability to compare different technologies

Group:
Masciotta Bruno 1989718
Campanella Edoardo 1984925

Submitted to:
Roberto Maria Delfino

SAPIENZA
UNIVERSITÀ DI ROMA

Introduction

- About Dataset
- Goal of the project

About Dataset

Description

This dataset is designed to help data scientists and machine learning enthusiasts develop fraud detection models. It contains 21 features capturing various aspects of financial transactions.

Use Cases

- Fraud detection model training
- Anomaly detection in financial transactions
- Risk scoring system for banks and fintech companies

Reference

https://www.kaggle.com/fraud-detection-transactions

Dataset Structure

```
Transaction_ID -- Unique identifier for each transaction (e.g., TXN_33553)
User_ID -- Unique identifier for the user (e.g., USER_1834)
Transaction_Amount -- Amount of money involved in the transaction (e.g., 39.79)
Transaction_Type -- Type of transaction (e.g., POS, Online, ATM)
Timestamp -- Date and time of the transaction (e.g., 2023-08-14 19:30:00)
Account_Balance -- User's account balance before the transaction (e.g., 93213.17)
Device_Type -- Geographical location where the transaction occurred (e.g., Sydney)
Merchant_Category -- Category of the merchant (e.g., Travel, Retail)
IP_Address_Flag -- Whether the IP address was flagged as suspicious (0 or 1)
Previous_Fraudulent_Activity -- Number of past fraudulent activities by the user (e.g., 0)
Daily_Transaction_Count -- Number of transactions made by the user that day (e.g., 7)
Avg_Transaction_Amount_7d -- User's average transaction amount in the past 7 days (e.g., 437.63)
Failed_Transaction_Count_7d -- Count of failed transactions in the past 7 days (e.g., 3)
Card_Type -- Type of payment card used (e.g., Amex, Visa)
Card_Age -- Age of the card in months (e.g., 65)
Transaction_Distance -- Distance between the user's usual location and transaction location(e.g.,83.17 km)
Authentication_Method -- How the user authenticated (e.g., Biometric, PIN)
Risk_Score -- Fraud risk score computed for the transaction (e.g., 0.8494)
Is_Weekend -- Whether the transaction occurred on a weekend (0 = Weekday, 1 = Weekend)
Fraud_Label -- Target variable (0 = Not Fraud, 1 = Fraud)
```

Goal of the project

Objective

Compare relational and NoSQL database systems to analyze their strengths and weaknesses in handling a dataset.

Key goals

- Select tools
  - Relational DBMS (MySQL)
  - NoSQL tool (Neo4j)
- Analyze the dataset
  - Perform analysis on both systems
  - Focus on efficiency and query performance
- Compare results
  - Pros and cons of relational DBMS
  - Pros and cons of NoSQL

MySQL approach

- Normalization and ER schema
- Data loading process
- Query analysis
- Performance metrics

Normalization and ER schema

What is normalization?

Normalization is a database design technique that:

- Reduces data duplication
- Improves query efficiency
- Ensures consistency in the database

Why is it important?

TransactionID	User_ID	Device_Type	Location	Merchant_Category
TX001	USER_1	Mobile	NYC	Grocery
TX002	USER_1	Mobile	NYC	Grocery
TX003	USER_1	Mobile	NYC	Grocery

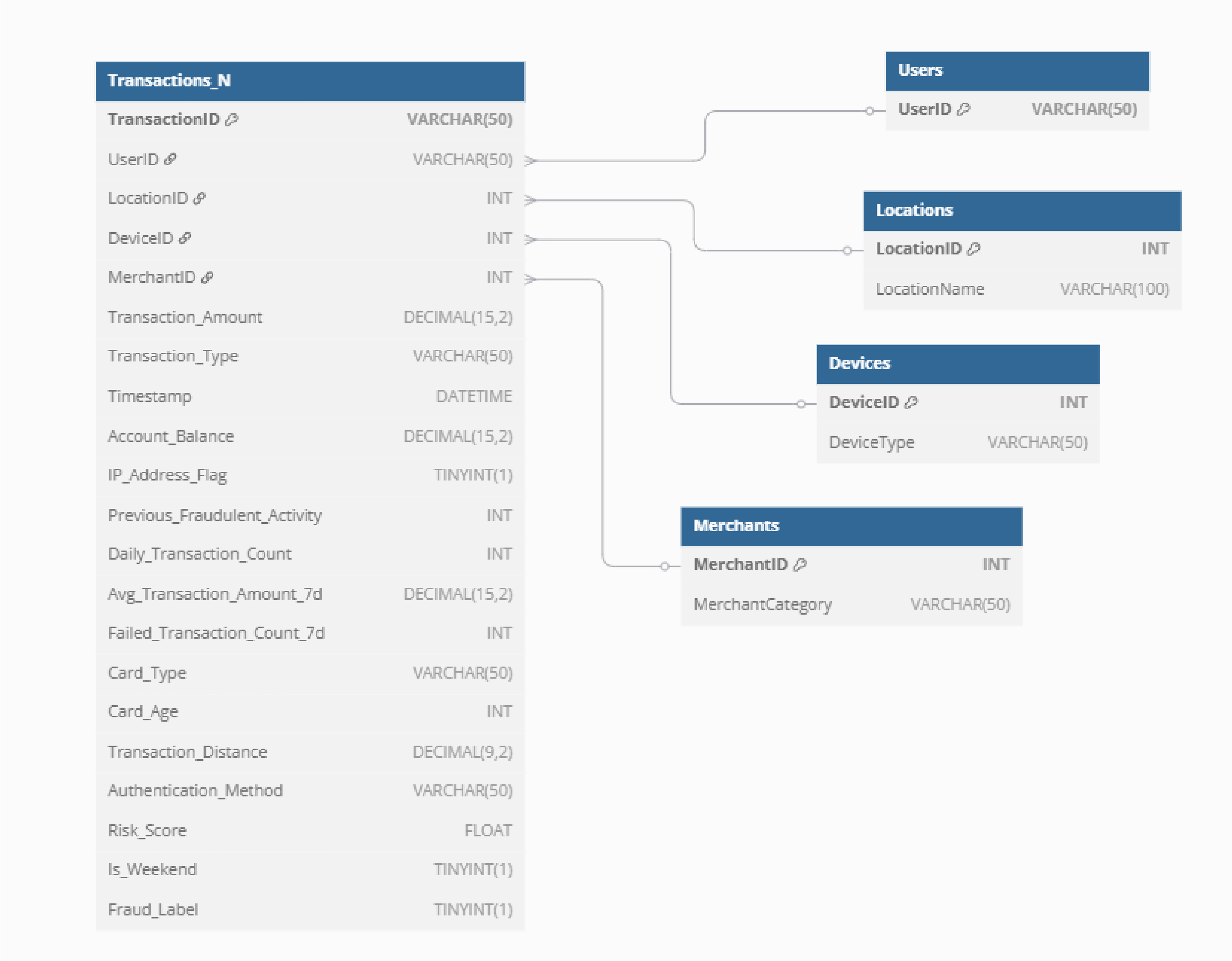
1.1 without normalization

Devices	Locations	Merchants
DeviceID: 1	LocationID: 1	MerchantID: 1
DeviceType: Mobile	LocationName: NYC	MerchantCategory: Grocery

1.2 with normalization

Without normalization repeating «mobile», «NYC» and «Grocery» for every transaction. Instead with normalization we move repeated values into separate tables, and this made it much cleaner, less duplication, faster queries and easier updates.

ER Schema



Transactions_N
TransactionID VARCHAR(50)
UserID VARCHAR(50)
LocationID INT
DeviceID INT
MerchantID INT
Transaction_Amount DECIMAL(15,2)
Transaction_Type VARCHAR(50)
Timestamp DATETIME
Account_Balance DECIMAL(15,2)
IP_Address_Flag TINYINT(1)
Previous_Fraudulent_Activity INT
Daily_Transaction_Count INT
Avg_Transaction_Amount_7d DECIMAL(15,2)
Failed_Transaction_Count_7d INT
Card_Type VARCHAR(50)
Card_Age INT
Transaction_Distance DECIMAL(9,2)
Authentication_Method VARCHAR(50)
Risk_Score FLOAT
Is_Weekend TINYINT(1)
Fraud_Label TINYINT(1)
Users
UserID VARCHAR(50)
Locations
LocationID INT
LocationName VARCHAR(100)
Devices
DeviceID INT
DeviceType VARCHAR(50)
Merchants
MerchantID INT
MerchantCategory VARCHAR(50)

Data loading process

Steps to load data

- Step 1: Load raw data
  - Use Python and Pandas to read the CSV file
  - Insert data into the Transactions table using SQL queries
- Step 2: Populate normalized tables
- Step 3: Link data
  - Populate *transactions_N* table by linking foreign keys

Code reference

- load_data.py
- insert_data.sql

Query analysis

Suspicious users and high-risk transactions

Analyze suspicious users and their high-risk transactions based on:

•**Unusual timing**: transactions occurring between 12 AM and 6 AM

•**Unusual distance**: transactions with distances significantly higher than the user's average.

Query & Output

https://colab/query-output

--OUTPUT

UserID	total_transactions	fraud_count	TransactionID	AnomalyType	Transaction_Distance	Timestamp
USER_7026	11	8	TXN_3867	Unusual Timing	1987.10	2023-09-01 05:22:00
USER_7026	11	8	TXN_31611	Unusual Timing	2180.86	2023-08-10 01:35:00
USER_7026	11	8	TXN_11941	Unusual Timing	3570.20	2023-05-15 00:16:00
USER_9983	10	7	TXN_34076	Highly Unusual Distance	4592.78	2023-12-20 07:15:00
USER_2268	11	7	TXN_14588	Unusual Timing	596.32	2023-12-17 00:19:00
USER_4936	11	7	TXN_24737	Unusual Timing	4719.07	2023-12-15 01:22:00

...

...

1.3 Output query

Query analysis

Key findings

1. **Suspicious users**
   - Users like `USER_7026` and `USER_9983` have high `fraud_count` values (8 and 7) and are involved in multiple flagged transactions.
   - These users are highly suspicious and should be prioritized for investigation.
2. **High-risk transaction patterns**
   - Unusual timing – late night transactions (12 AM – 6 AM) are strong indicators of fraud.
   - Unusual distance – geographic anomalies often involve large deviations from the user’s typical behavior

Performance metrics

Query performance optimization: before and after indexing

- Analyze the performance of a query to identify the top 10 users with the highest transaction counts.
- Compare query execution metrics before and after adding index.

1. Initial query performance
    - Execution time: 175 ms
    - Rows scanned: 50 000 rows
    - Temporary table usage: high
    - Cost: high cost due to full table scanning
2. Improved query performance
    - Execution time: reduced (69% improvement)
    - Rows scanned: still 50 000
    - Temporary table usage: eliminated
    - Cost: significantly reduced due to index usage

Performance metrics

Resource usage and space complexity

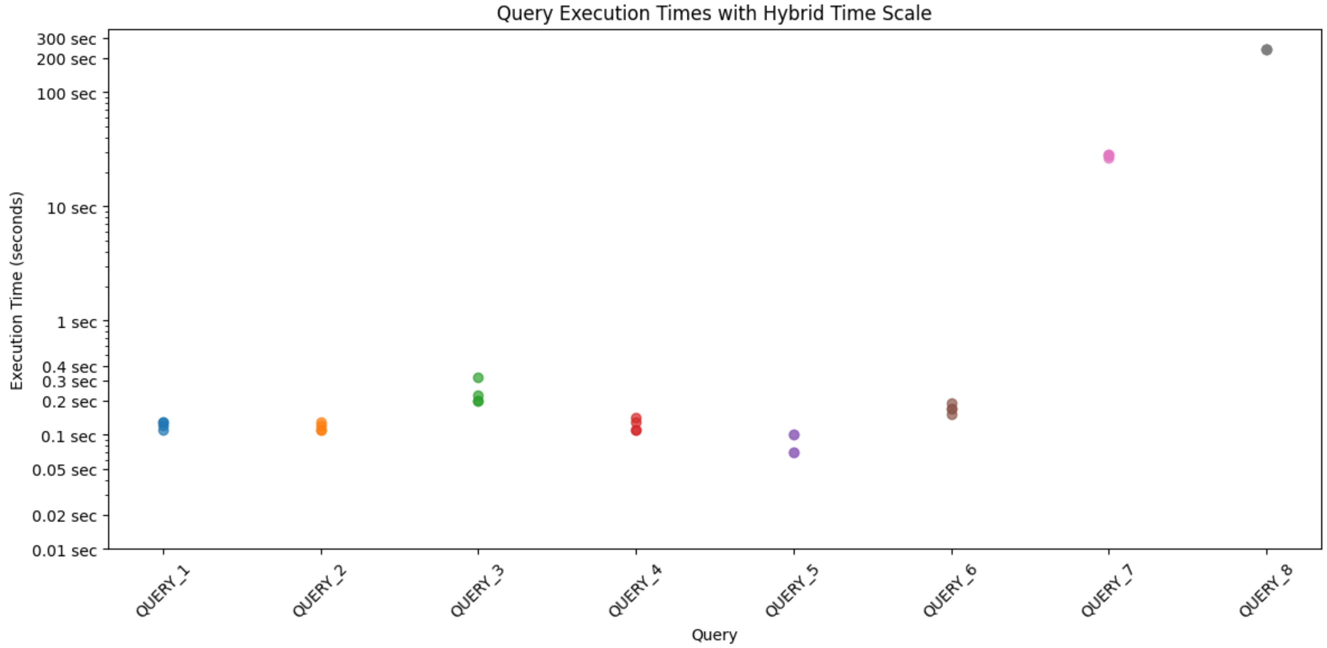
- Analyze resource usage and space complexity of the query and database tables.

```
--Space complexity analysis
SELECT
    table_name AS `Table`,
    ROUND((data_length + index_length) / 1024 / 1024, 2) AS `Size (MB)`
FROM
    information_schema.tables
WHERE
    table_schema = 'banking_db';

+------------------+-----------+
| Table            | Size (MB) |
+------------------+-----------+
| Devices          |      0.03 |
| Locations        |      0.03 |
| Merchants        |      0.03 |
| Transactions     |     12.52 |
| Users            |      0.36 |
| risk_transaction |      1.52 |
| transactions_N   |      7.52 |
+------------------+-----------+
```

Performance metrics
Time complexity

- Analyze time complexity of the query.



Critical aspects of self and multi-join operations

•**Self join** (query 7): likely involves joining a table with itself, which can be computationally expensive.

•**Multi-join** (query 8): involves multiple joins across several tables, which can lead to exponential growth in the number of rows processed.

Neo4j approach

- Data loading process
- Query analysis
- Performance metrics

Data Loading in Neo4j

- Place the CSV file in the /import folder

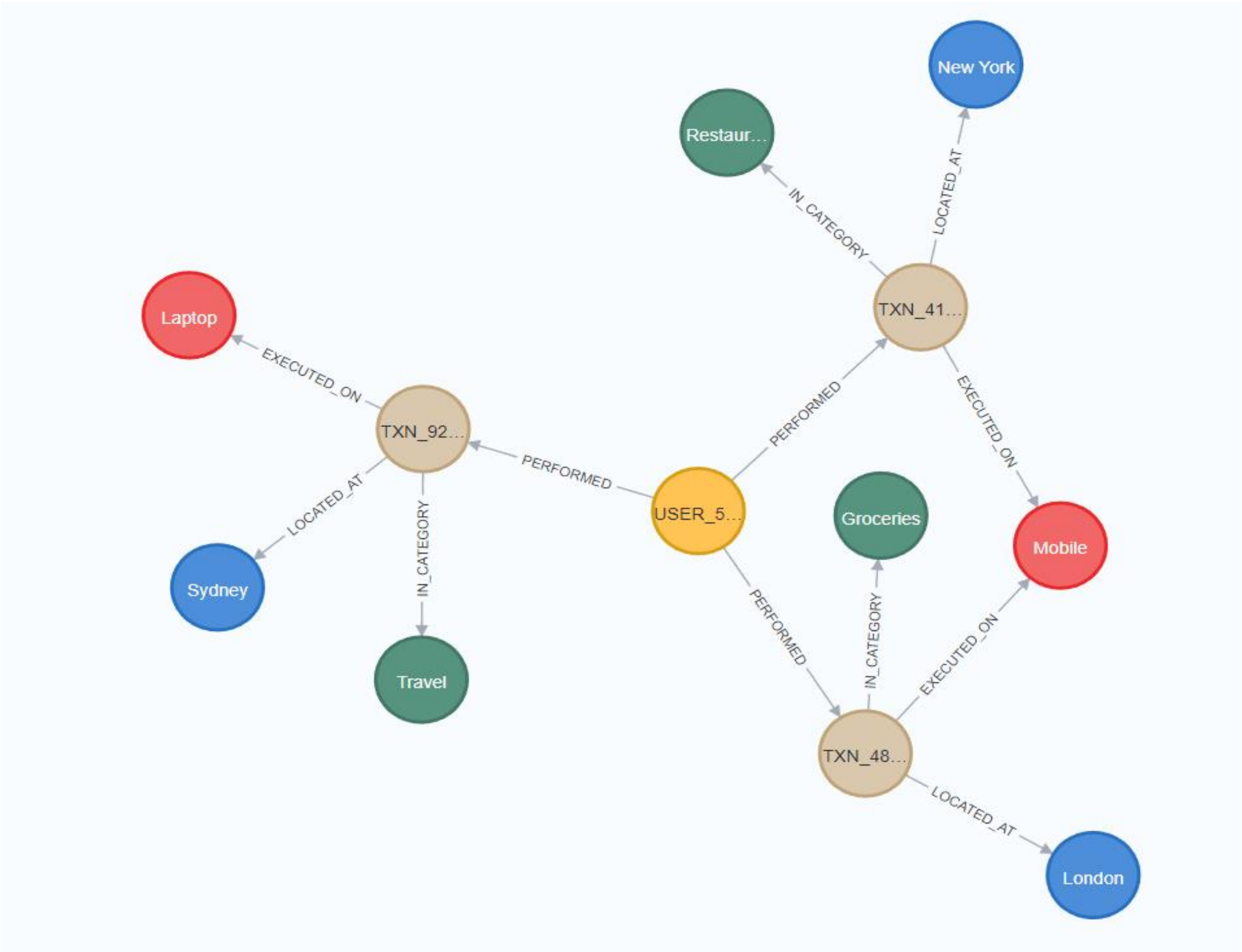
- Use this command to read data:

```
LOAD CSV WITH HEADERS FROM 'file:///synthetic_fraud_dataset.csv' AS row
Return row
```

- Create nodes and relationships:

```
MERGE (u:User {user_id: row.`User_ID`})
MATCH (u:User {user_id: row.`User_ID`})
MATCH (t:Transaction {transaction_id: row.`Transaction_ID`})
MERGE (u)-[:PERFORMED]->(t)
```

Result Graph



Query analysis

- Identifies the **top 10 users** with the **most transactions**, showing their fraud transactions and fraud rate.
- Compute the average of Fraud_Rate considering all dataset.
- Compare result of the query with the average.

Query:

```
MATCH (u:User)-[:PERFORMED]→(t:Transaction)
WITH u.user_id AS UserID,
     COUNT(t) AS TotalTx,
     SUM(t.fraud_label) AS FraudTx
RETURN UserID, TotalTx, FraudTx, toFloat(FraudTx)/TotalTx*100 AS FraudRate
ORDER BY TotalTx DESC
LIMIT 10;
```

Output:

UserID	TotalTx	FraudTx	FraudRate
"USER_9998"	16	4	25.0
"USER_6599"	16	3	18.75
"USER_3925"	16	3	18.75
"USER_1027"	15	3	20.0
"USER_5014"	15	4	26.666666666666668
"USER_3415"	15	5	33.33333333333333
"USER_6229"	14	3	21.428571428571427
"USER_6237"	14	3	21.428571428571427
"USER_6700"	14	3	21.428571428571427
"USER_2620"	14	4	28.57142857142857

Query Analysis: Results

- Most users in the top 10 have a fraud rate **significantly below the average (32.10%)**, with rates ranging from 18.75% to 33.33%.
- The user with the **highest fraud rate** (USER_3415) has a fraud rate of 33.33%, which is slightly higher than the average.

Conclusions:

- High number of transactions **NOT Implies** high number of frauds
- Frauds **doesn’t depend only** on number of transactions, but for example also on device, merchant category or distance.

Query Analysis

Combination of 2 queries:

1. Compute the transaction and frauds for each merchant category
2. Compute the transaction and frauds for each card_age category: New, Medium, Old

Query:

https://colab/query_analysis

Output (Partial):

MerchantCategory	CardCategory	TotalTransactions	FraudulentTransactions	FraudRatio
"Clothing"	"Medium"	1051	359	34.15794481446242
"Restaurants"	"Medium"	962	326	33.88773388773389
"Travel"	"New"	985	332	33.70558375634518
"Electronics"	"New"	956	319	33.36820083682008

Query Analysis: Results

Conclusion:

- **Highest risk**: “Medium” cards (25–48 months) in Clothing (34.2 %) and Restaurants (33.9%).

- **“Old” cards (4+ years):** slightly lower rates (31.5 – 32.7 %) despite high volumes.

- **High-volume categories**: Restaurants “Old” (7 990 TNX) & Travel “Old” (8 041 TNX) at ~32 % fraud.

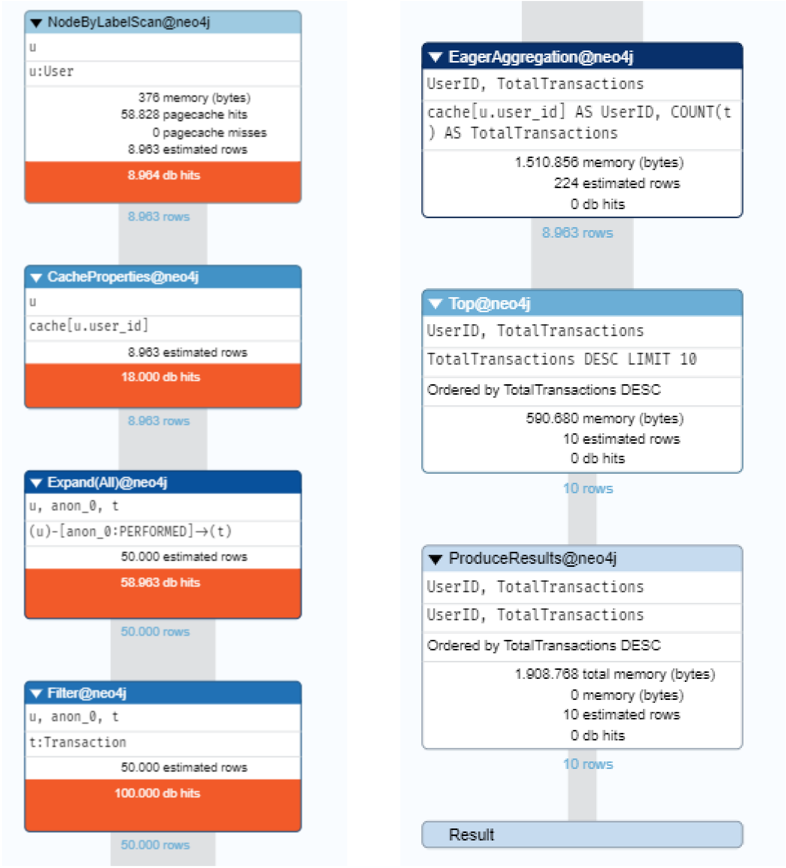
- **Volume vs. risk disconnect**: High transaction volume (like Restaurants “Old”) does not always imply higher fraud rates than lower-volume segments.

Performance Metrics

- The operator **PROFILE** allow to analyze the metrics of a query such as:
  1. Memory
  2. Db Hits
  3. Page cache Hits/misses
  4. Estimated Rows

Query:

```
PROFILE
MATCH (u:User)-[:PERFORMED]->(t:Transaction)
WITH u.user_id AS UserID, COUNT(t) AS TotalTransactions
RETURN UserID, TotalTransactions
ORDER BY TotalTransactions DESC
LIMIT 10;
```



Performance Metrics: Possible Improvement!

- To improve the metrics, we use the concept of Indexes and Constraints:

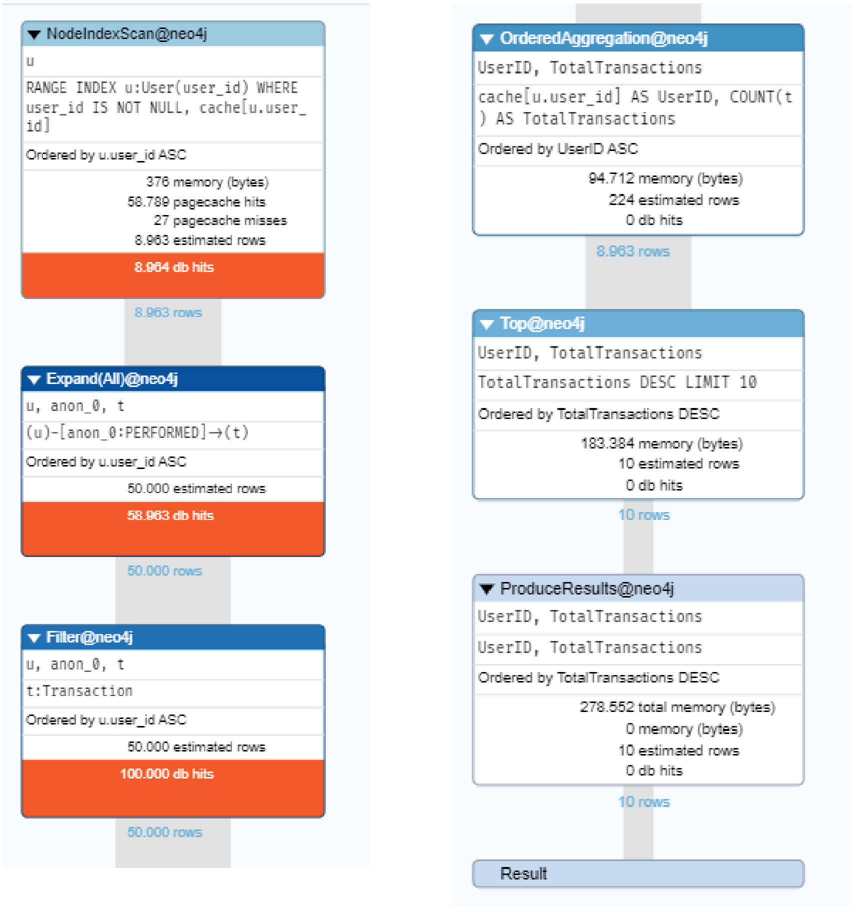
1. Transaction:

```
CREATE INDEX FOR (t:Transaction)
ON (t.transaction_id);
```

2. User:

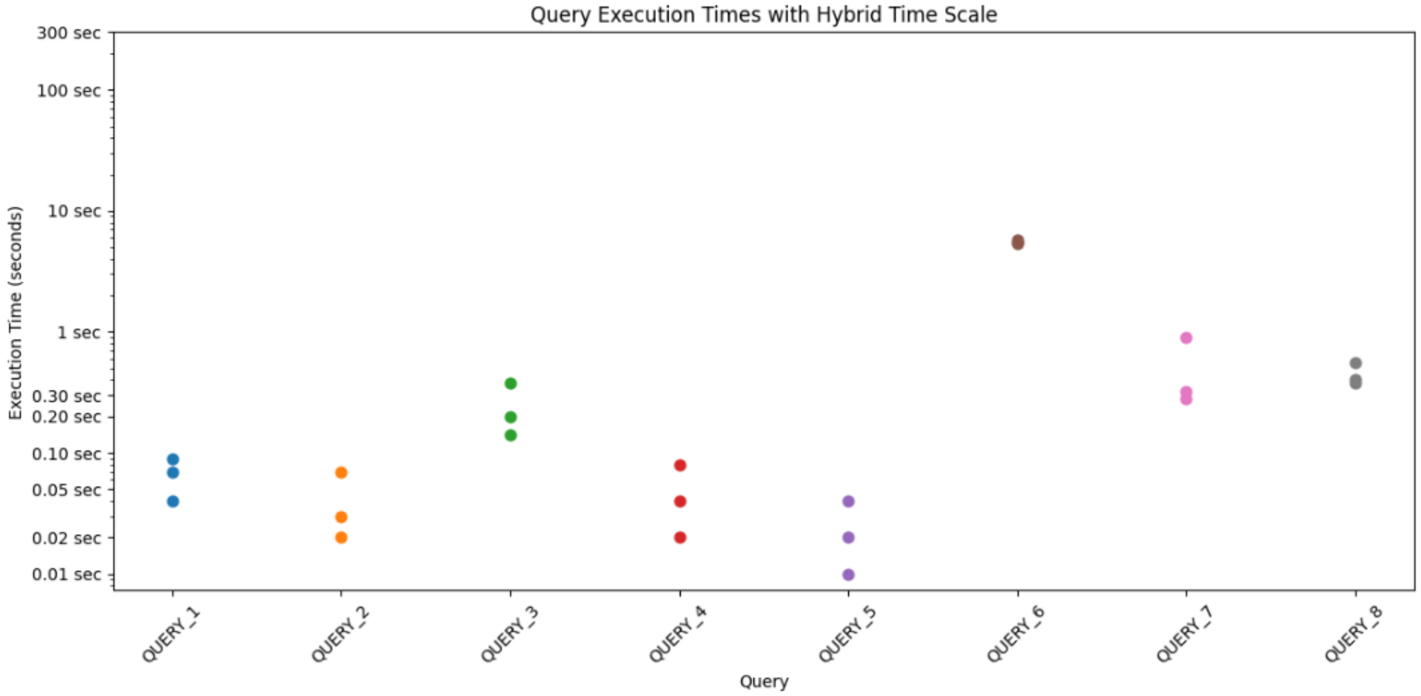
```
CREATE CONSTRAINT user_id_unique FOR (u:User)
REQUIRE u.user_id IS UNIQUE;
```

- Improvement about memory during the phase of aggregation, top and produceresults.



Performance Metrics: Time Complexity

- The following graph show **Time Complexity** of the queries:



- **Critical Aspect**: Traverse of the whole graph during a query composed by complex JOIN in the case of a big dataset.

Conclusion

- Summary of key points
- References

Summary of key points

MySQL (Relational Database):

•Best suited for structured data with predictable relationships.

•Multi – join can be computationally expensive.

Use MySQL when:

•Data is structured

•Queries are simple

•Transactions are independent

Neo4j (Graph Database):

•Optimized for analyzing interconnected data.

•Suitable for real-time analysis of highly connected data.

Use Neo4j when:

•Real-time analysis is critical

•Patterns are complex

•Schema flexibility is important

References

GitHub repo

https://github.com/masciotta02/NoSQL_Project

Other resources

Tutor presentation (Neo4j)

THANK YOU FOR YOUR ATTENTION!