New Generation Data Models and DBMSs

Credit card fraud detection project

Negar Akhgar – 11242A

Professor Marco Mesiti

March - April 2024

1. Introduction

Payment card fraud is a recognized issue in today's world, leading to continually increasing financial losses. However, the objective of this project is not to identify fraudulent transactions. Instead, it aims to utilize a transaction data simulator code for the purpose of creating and populating a database. The simulator will produce three main tables: Customers, Terminals, and Transactions, each containing specific information:

- The Customers table will include customer details such as identifiers, coordinates indicating their position, spending frequency and amounts, and a list of available terminals.

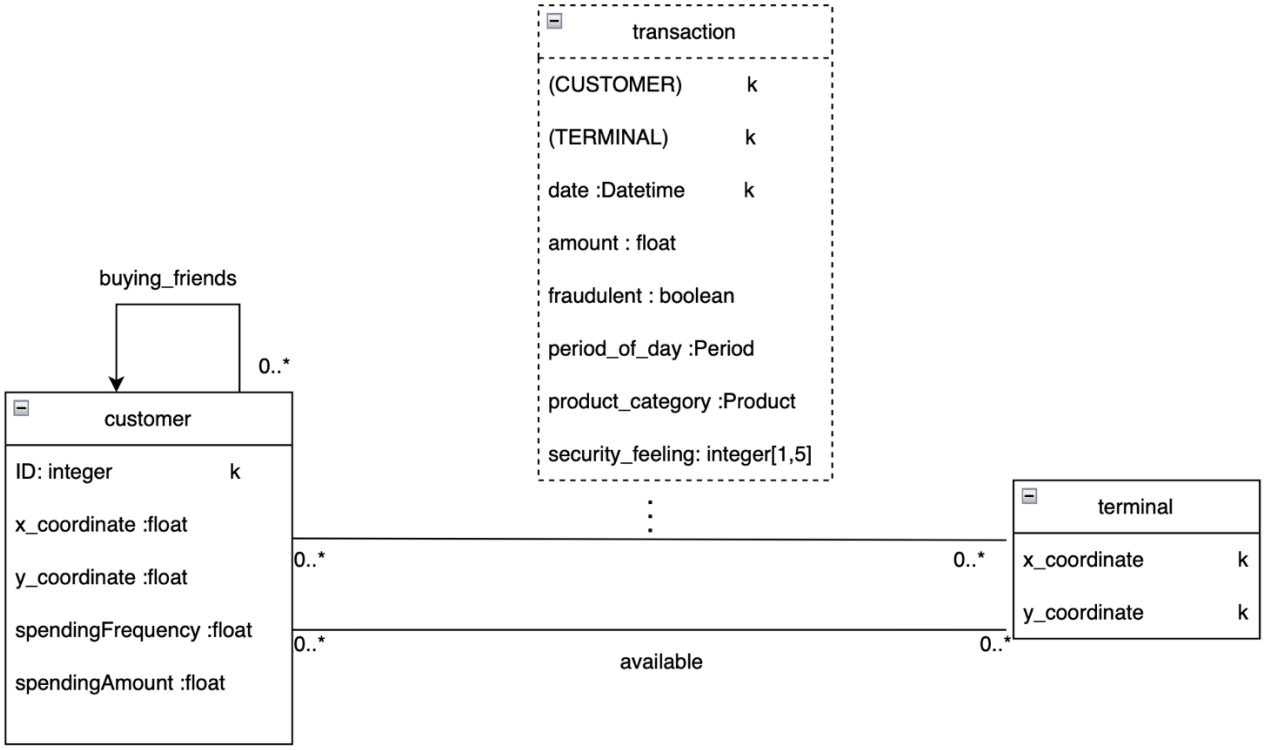
- The Terminals table will feature terminal identifiers and their respective coordinates.

- The Transactions table will describe transactions linked to a customer, conducted on a specific terminal. It will include customer and terminal identifiers, the execution time of the transaction, the transaction amount, and a boolean indicating whether the transaction is fraudulent or not.

> The code for generating this data can be accessed through the following link: [https://fraud-detection-handbook.github.io/fraud-detection-handbook].

2 .UML

The UML schema provides an accurate representation of how to model this domain. It involves designing the relationships between customers, terminals, and transactions using three separate entities for each. The association between customers and available terminals can be conceptualized as a relationship, where available terminals for a customer are those within a certain distance range. It's important to consider that a customer might not have any terminals nearby, resulting in no available terminals.Moving on to transactions, each transaction is initiated by a customer and conducted on a specific terminal. This relationship between a customer and a terminal encapsulates all transaction details, such as time and amount, as properties of the relationship. Additionally, it's worth noting that some customers may not have executed any transactions yet.



This UML schema incorporates additional data introduced by queries, such as the time period (morning, afternoon, evening, or night), product type, transaction security perception, and the relationship of purchasing habits among friends for customers who have made more than three transactions from the same terminal and share a similar average perception of security. This latter relationship, which concerns customers, is explicitly depicted in the schema.

Constraints and assumptions:

- The list available_terminals comprises terminals located within a specified radius around the customer.

- A transaction can be uniquely identified by its customer, terminal, and datetime.

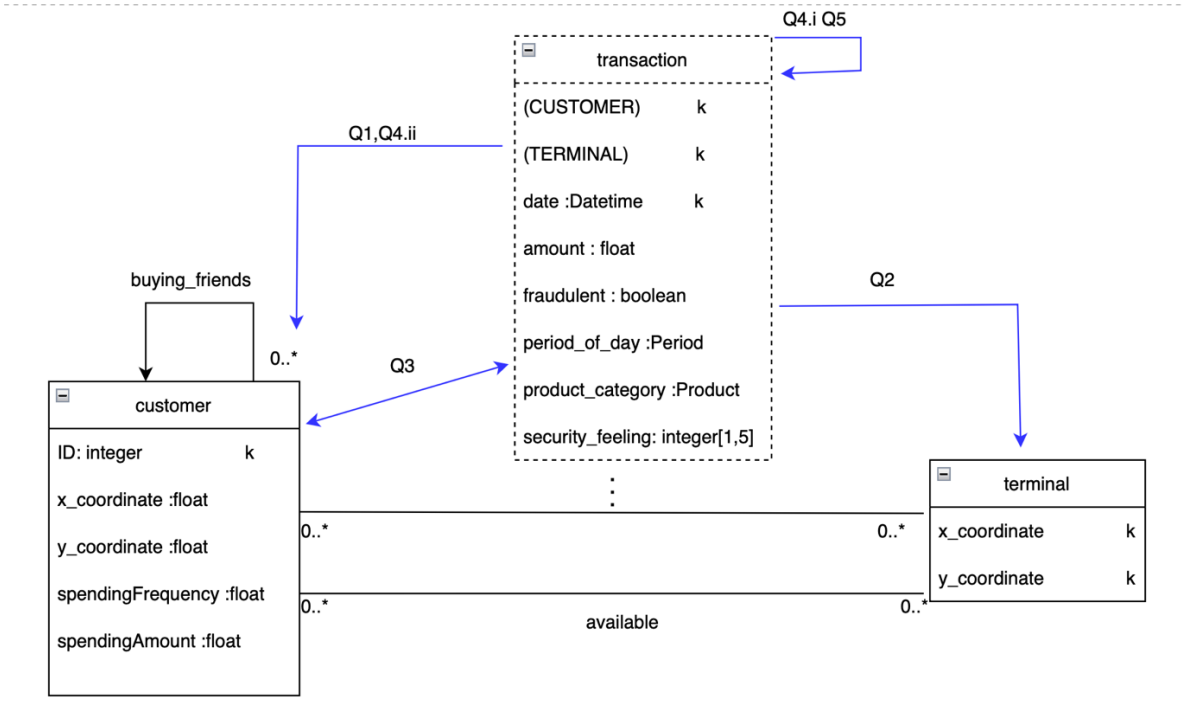
- The coordinates of the terminal need to be legitimate geospatial coordinates.

- Products are categorized into the following set: {high-tech, food, clothing, consumable, other}.

- One terminal can process only one transaction simultaneously for each customer.

3. Workload

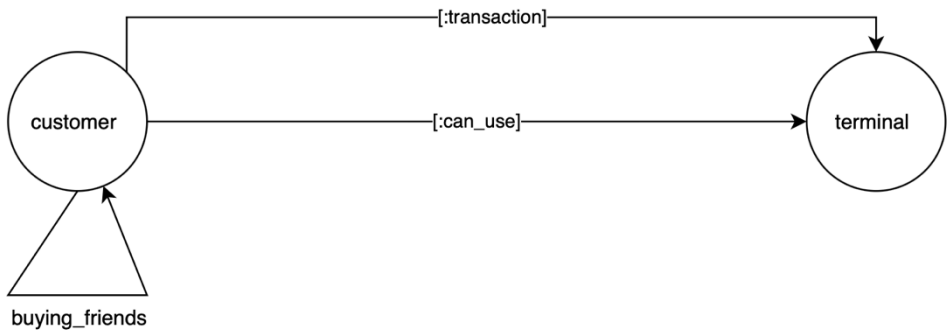
Based on the five queries needed for the project, here is the workload breakdown for each of them.



- The process begins by grouping transactions based on the customer, followed by computing the values, and ultimately linking each computed value back to its respective customer.
- Initially, the transactions are grouped according to their dates, then the values are calculated, and finally, each value is linked to its corresponding terminal.
- This query entails navigating from the given customer to their transactions, then tracing back to other customers, repeating this process until the desired level of co-customer association is achieved.
- All the necessary data is encompassed within the Transactions.

- The process begins by grouping and filtering the Transactions based on specific criteria, after which the corresponding customers are linked or associated with each other.
- All the necessary information is encompassed within the Transactions.

4. Logical model



Following the model established in section 3, the logical model comprises two main components: Terminal and Customer, interconnected by two relationships: Transaction and Can_use.

As outlined in section 2, the understanding of Can_use is consistent with the interpretation provided in the book: A Customer is associated with a Can_use relationship towards a Terminal if that Terminal falls within the specified range generated. However, the project's text offers a different interpretation: A Customer has a Can_use relationship when they conduct a Transaction at a Terminal.

Despite this distinction, in scenarios involving a significant number of Transactions, the difference is minimal because it's reasonable to assume that each Customer engages in at least one transaction at every Terminal available to them.

5. Dataset Generation

To create the three datasets needed (50, 100, 200 MB), we've developed a script called dataset_generator.py. It uses the code outlined in Chapter 3 of the book but with some important tweaks to enhance the convenience and efficiency of data generation. The functions generate_customer_profiles_table and generate_terminal_profiles_table remain unchanged from the book's description. As the names suggest, they produce DataFrames containing customer profiles and terminal profiles, respectively.

Dataset name	Nb_customet	Nb_terminals	Nb_days	Dataset size	Time to generate
small	1500	1500	426	69.07	2m 18.2s
Medium	2500	2500	426	117.05	3m 20.1s
big	5000	5000	426	238.06	7m 4.3s

6. Dataset Upload

6.1 Customer Upload

```
CALL apoc.periodic.iterate(
"LOAD CSV WITH HEADERS FROM 'file:///customers.csv' AS customers return customers
"WITH customers
CREATE (:Customer {
CUSTOMER_ID: customers.CUSTOMER_ID,
x_customer_id: customers.x_customer_id,
y_customer_id: customers.y_customer_id,
mean_spending_amount: customers.mean_amount,
std_spending_amount: customers.std_amount,
mean_transactions_per_day: customers.mean_nb_tx_per_day,
available_terminals: customers.available_terminals,
nb_terminals: customers.nb_terminals
})",
{batchSize:1000, iterateList:true});
```

6.2 Terminal Upload

```
CALL apoc.periodic.iterate(
"LOAD CSV WITH HEADERS FROM 'file:///terminals.csv' AS terminals return terminals",
"CREATE (:Terminal {
TERMINAL_ID: terminals.TERMINAL_ID,
x_terminal_id: terminals.x_terminal_id,
y_terminal_id: terminals.y_terminal_id
})",
{batchSize:1000, iterateList:true});
```

6.3 Transaction Upload

```
CALL apoc.periodic.iterate(
"LOAD CSV WITH HEADERS FROM 'file:///transactions.csv' AS transactions return
transactions",
"WITH transactions
MATCH (c:Customer {CUSTOMER_ID: transactions.CUSTOMER_ID})
MATCH (t:Terminal {TERMINAL_ID: transactions.TERMINAL_ID})
CREATE (c)-[:Transaction {
TRANSACTION_ID: transactions.TRANSACTION_ID,
transaction_datetime: datetime(transactions.TX_DATETIME),
transaction_amount: transactions.TX_AMOUNT,
fraud: transactions.TX_FRAUD}
]→(t) ",
{batchSize:1000, iterateList:true});
```

The imports utilize the LOAD CSV primitive to fetch the CSV files containing the information generated by the generator.

The queries fetch data from CSV files and create Customers, Terminals, and Transactions row by row, using batch operations due to the large dataset. When creating Transactions, existing Customers and Terminals are retrieved first, and relationships are established using the CREATE primitive instead of MERGE to avoid potential issues with non-existent entities. Importing Customers and Terminals must precede Transaction creation to ensure all necessary entities are present in the database.

To accelerate the calculations, three indexes were established for the Customer and Terminal IDs, along with the datetime of the Transactions.

```
CREATE INDEX customer_id_index FOR (c:Customer) ON (c.CUSTOMER_ID);
CREATE INDEX terminal_id_index FOR (t:Terminal) ON (t.TERMINAL_ID);
CREATE INDEX transaction_datetime_index FOR (tr:Transaction) ON (tr.transaction_datetime);
```

7 Queries

7.1 Query 1

For each customer checks that the spending frequency and the spending amounts of the last month is under the usual spending frequency and the spending amounts for the same period.

```
MATCH   (c:Customer)-[t:Transaction]→(:Terminal)
WHERE   t.transaction_datetime.month = (date() - duration('P1M')).month
        AND t.transaction_datetime.year < date().year
WITH    c.CUSTOMER_ID as customer_id,
        count(t) as frequency,
        sum(toFloat(t.transaction_amount)) as spending_amounts
WITH    customer_id,
        avg(frequency) as avg_frequency,
        avg(spending_amounts) as avg_spending_amounts

MATCH   (c:Customer)-[t]→(:Terminal)
WHERE   t.transaction_datetime.month = (date() - duration('P1M')).month
        AND t.transaction_datetime.year = date().year
WITH    c.CUSTOMER_ID as customer_id,
        count(t) as transactions_count,
        sum(toFloat(t.transaction_amount)) as spending_amounts,
        avg_frequency,
        avg_spending_amounts
RETURN  customer_id,
        spending_amounts as current_amount,
        avg_spending_amounts as average_amount,
        transactions_count as current_frequency,
        avg_frequency as average_frequency
LIMIT   25
```

Explanation :

In this query after the assumption that the usual spending frequency is the spending fre- quency of the transactions executed in the same month in previous years ,This query evaluates customer transactions from the prior monthof the previous year , assessing spending levels and transaction frequencies. Initially, it sifts through transactions occurring in the previous month, generating metrics like frequency and spending amounts per customer. Subsequently, it calculates average frequency and spending for all customers. Finally, it retrieves transactions within the prior month's timeframe, derives metrics based on average values, and presents the results for additional scrutiny.

7.2 Query 2

“For each terminal identify the possible fraudulent transactions. The fraudulent transactionsare those whose import is higher than 20% of the maximal import of the transactions executed on the same terminal in the last month.”

```
MATCH   (t:Terminal)<-[tr:Transaction]-()
WHERE   t.transaction_datetime.month = (date() - duration('P1M')).month
        AND t.transaction_datetime.year = date().year
WITH    t,
        max(toFloat(tr.transaction_amount)) AS last_month_maximal,


MATCH   (t)<-[tr1:Transaction]-()
WHERE   tr1.transaction_datetime.month = date().month
        AND     toFloat(tr1.transaction_amount) > last_month_maximal+(last_month_maximal*0.2)
RETURN  t.TERMINAL_ID AS terminal,
        collect({id: tr1.TRANSACTION_ID, import: tr1.transaction_amount}) AS fraudulent_transactions,
        last_month_maximal
```

Explanation :

1. in this query we retrieve the transaction with the highest amount from the previous month for each terminal.

2. Transactions from the current month are filtered based on certain criteria.

3. We return any potentially fraudulent transactions, along with the corresponding terminal and the threshold transaction amount.

7.3 Query 3

“Given a user u, determine the “co-customer-relationships CC of degree k”. A user u’ is a co-customer of u if you can determine a chain “u1-t1-u2-t2-…tk-1-uk“ such that u1=u, uk=u’, and for each 1<=I,j<=k, ui <> uj, and t1,..tk-1 are the terminals on which a transaction has been executed. Therefore, CCk(u)={u’| a chain exists between u and u’ of degree k}. Please, note that depending on the adopted model, the computation of CCk(u) could be quite complicated. Consider therefore at least the computation of CC3(u) (i.e. the co-costumer relationships of degree 3).”

```
WITH    '1' as customer_ID
MATCH   (c: Customer {CUSTOMER_ID: customer_ID})-[:Transaction*10]-(c1: Customer)
WHERE   c<>c1
RETURN DISTINCT c1.CUSTOMER_ID
```

Explanation :

The query consists of three main phases:

1. The Customer ID is defined in the initial WITH clause.

2. Customers reachable within a path of 10 transactional relationships, starting from the selected Customer, are collected.

3. The unique IDs of the gathered customers are then returned.

7.4 Query 4

“Extend the logical model that you have stored in the NOSQL database by introducing the
following information (pay attention that this operation should be done once the NOSQL database has been already loaded with the data extracted from the datasets):
i. Each transaction should be extended with:
1. The period of the day {morning, afternoon, evening, night} in which the transactionhas been executed.
2. The kind of products that have been bought through the transaction {high-tech,food, clothing, consumable, other}
3. The feeling of security expressed by the user. This is an integer value between 1 and 5 expressed by the user when conclude the transaction.
The values can be chosen randomly.
ii. Customers that make more than three transactions from the same terminal expressing a similar average feeling of security should be connected as “buying_friends”. Therefore also this kind of relationship should be explicitly stored in the NOSQL database and can be queried. Note, two average

feelings of security are considered similar when their difference is lower than 1."

```
MATCH ()-[t:Transaction]-()
WITH t, round(rand() * 4)+1 AS rand1, round(rand() * 4)+1 AS rand2
SET t.period = CASE
        WHEN t.transaction_datetime.hour ⩾ 6 AND t.transaction_datetime.hour ⩽ 12 THEN 'morning'
        WHEN t.transaction_datetime.hour ⩾ 13 AND t.transaction_datetime.hour ⩽18 THEN 'afternoon'
        WHEN t.transaction_datetime.hour ⩾ 19 AND t.transaction_datetime.hour ⩽ 22 THEN 'evening'
        ELSE 'night'
        END
SET t.product_kind = CASE
        WHEN rand1 = 1 THEN 'high-tech'
        WHEN rand1 = 2 THEN 'food'
        WHEN rand1 = 3 THEN 'clothing'
        WHEN rand1 = 4 THEN 'consumable'
        ELSE 'other'
        END
SET t.security_feeling = rand2;
```

Explanation :

In this query for every transaction, two random values are chosen from a uniform distribution between 1 and 4. These values are then used in CASE clauses to assign properties like "product_kind" and "security_feeling" to the transaction. Additionally, the "period" property is set based on the hour information from the "transaction_datetime" property.

7.4.2 part 2

```
MATCH   (c1:Customer)-[t1:Transaction]→(t:Terminal)←[t2:Transaction]-(c2:Customer)
WITH    c1,
        c2,
        count (DISTINCT t1) as nb_1,
        count (DISTINCT t2) as nb_2,
        avg(t1.security_feeling) as avgSecurity1,
        avg(t2.security_feeling) as avgSecurity2
WHERE           c1<>c2
        AND     nb_1 > 3
        AND     nb_1 > 3
        AND     abs(avgSecurity1 - avgSecurity2) <  1

CREATE  (c1)-[:Buying_friends]→(c2),
        (c1)←[:Buying_friends]-(c2)
```

Explanation :

In this query couples of customers who have conducted transactions at the same terminal are identified. The number of transactions and the average security feeling values are calculated. Customers who have made more than three transactions and have similar average security feelings (within 1 unit) are linked together with a "Buying_friends" relationship.

7.5 Query 5

“For each period of the day identifies the number of transactions that occurred in that period, and the average number of fraudulent transactions.”

```
MATCH ()-[tr:Transaction]-()
WITH tr.period as period,
count(tr) as nb_tr,
COUNT(CASE WHEN toFloat(tr.fraud) <> 0 THEN 1 END) as nb_frauds
RETURN period,
nb_tr as transactions_number,
nb_frauds as frauds,
(toFloat(nb_frauds)/toFloat(nb_tr)*100) as fraud_percent
```

Explanation:

In this query transactions are collected. The number of transactions and fraudulent transactions are tallied for each period (morning, afternoon, evening, night). For each period, the counts of both fraudulent and total transactions are presented alongside the percentage of fraudulent transactions.

8 Performance

In the following table are shown the times taken to generate the databases:

Dataset name	*Time to generate*

small	2m 18.2 s
medium	3m 20.1 s
big	7m 4.3 s

In the following table are shown the times taken to load the datasets in the database:

Dataset name	*Upload time*
small	10.77 m
medium	20.77 m
big	3.50 h

In the following table are shown the times of execution of the different queries in the different
databases:

query	*small*	*Medium*	*large*
a	*2.71 s*	7.3 s	18.31 s
b	*1.64 s*	3.78 s	7.07 s
c	5m 0.69 s	4m 59.25 s	2m 23.7 s
d.i	5,67 s	16.02 s	26.59 s
d.ii	1544,31 s	3799,50 s	15m 29.95 s
e	1.34 s	1,62 s	4.7 s

The performance is improved by the three new indexes implemented on the Customer ID, Terminal ID, and Transaction Datetime. Nearly all queries depend on database access starting from either a Customer or Terminal ID, and a significant number of them retrieve Transactions based on their datetime.

To enhance queries seeking statistics or specific relationships, we can utilize a computed pattern. This approach involves precomputing certain values and ensuring they stay updated, thereby avoiding complex and time-consuming operations during query execution. However, this strategy entails a trade-off between execution time, insertion time, and introduces redundancy, which may not always be beneficial.