Final Report - Corporativo

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

1. General project information

Information	Details
Partner Company	Startup de Pagamento LTDA
Project name	Dumbo's Project
Development Team	Gustavo Ferreira
Advising Professor	Guilherme Cestari
Start date	03/02/2025
Estimated End Date	11/04/2025

2. Project Objectives

The main objective of this project is to develop an optimized solution to correlate customer action event data from the API and Database of payments with their respective users **using graphs**. This will enable a more integrated and automated view of customer interactions, reducing data fragmentation and improving operational efficiency.

Additionally, an interactive dashboard will be developed for analyzing this data, serving two main audiences:

- **Support Team:** Will have a tool that speeds up problem diagnosis, avoiding manual queries across multiple databases and reducing SLA breaches.
- **Area Developers:** Will be able to publish data on product usage by their clients following the defined data standard, resulting in a unified database of event and customer data.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

3. Project Scope and Backlog

The project aims to structure a graph-oriented database in Neo4j to store and correlate user interaction events with their respective actions already existing in the database or obtained via API. The main objective is to improve service efficiency and generate strategic insights by providing an integrated view and facilitating complex queries.

3.1. Epics and Features

Below are the epics that were developed during the module, which were discussed and planned in collaboration with the partner company and the advising professor.

- **Epic 1: Definition of Database Architecture and Initial Modeling**
  - **Feature 1.1:** Comparative analysis between PostgreSQL and Neo4J based on project needs and data characteristics.
  - **Feature 1.2:** Creation of an Entity-Relationship (ER) diagram that clearly represents the entities and their relationships.
- **Epic 2: Technical Documentation and Environment Setup**
  - **Feature 2.1:** Detailed description of the attributes and entities present in the ER model, along with their respective relationships.
  - **Feature 2.2:** Development of a Dockerfile to provision and run Neo4J in an isolated environment, facilitating deployment.
- **Epic 3: Initial Development of the Data Structure**
  - **Feature 3.1:** Preparation of the first version of the SQL script for table creation and initial data insertion.
  - **Feature 3.2:** Analysis and implementation of indexes on frequently used fields to optimize query performance.
- **Epic 4: Testing and Performance Validation**
  - **Feature 4.1:** Generation of realistic mock data and execution of simulated tests to validate database queries and performance.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

- **Feature 4.2:** Ensuring that simple queries (e.g., retrieving customer data) return results in up to 1 second, and complex ones (e.g., retrieving payment events) in up to 5 seconds.

- **Epic 5: Query Optimization and Final Documentation**
  - **Feature 5.1:** Development of specific queries to address the system's main questions.
  - **Feature 5.2:** Creation of final documentation with best practices for the database structure and fields.

3.2. Out of scope

1. Development of the dashboard and frontend interface (to be addressed in another module).
2. Use of a relational database (PostgreSQL) as the primary data source.
3. Direct integration with other systems without using the intermediary API.

4. Module Roadmap and High-Level Schedule

As in Inteli's standard work model, it was agreed with the partner company that each sprint would last two weeks, ending with a presentation (sprint review) to validate the results achieved and align the next steps. Below is an overview of what was carried out during each sprint:

Sprint 1:

1. Database Analysis: Evaluation of PostgreSQL vs. Neo4J based on project needs and data requirements.
2. Entity-Relationship (ER) Diagram: Development of an ER diagram that clearly illustrates entities and their interrelationships.

Sprint 2:

1. Documentation of Attributes and Entities: Detailed description of each entity's attributes and their relationships.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

2. Dockerfile with Neo4J: Creation of a Dockerfile for provisioning and running Neo4J in an isolated environment, facilitating deployment.

Sprint 3:

1. First Version of the SQL Script: Development and delivery of the first version of the SQL script for table creation and initial data population.
2. Index Creation for Table Fields: Analysis and creation of indexes for frequently used fields to optimize queries.

Sprint 4:

1. Test Mock with Accurate Data: Generation of sample data and execution of simulated tests to validate queries and database performance.
    a. Simple queries (e.g., retrieving customer data) should take no longer than 1 second.
    b. Complex queries (e.g., retrieving all payment events) should take a maximum of 5 seconds.

Sprint 5:

1. Query Development to Answer Key Questions: Creation of specific queries to address the system's main inquiries.
2. Final Documentation: Preparation of a best practices guide for database fields and structure.

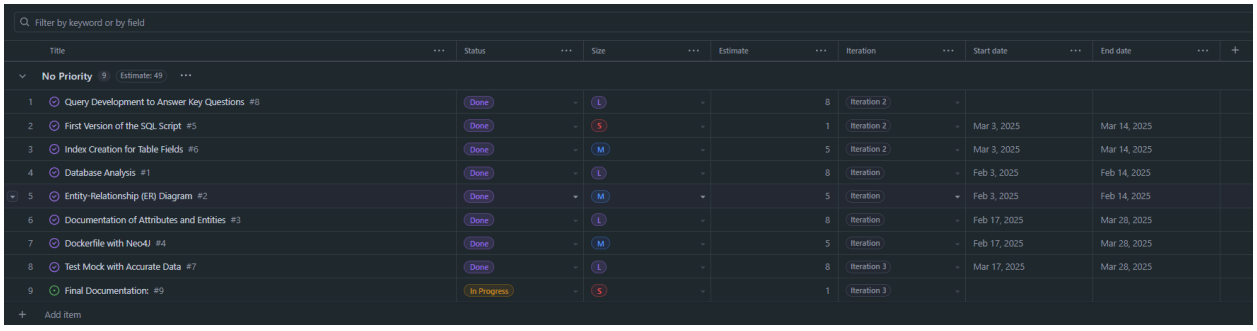

Figure 1: Project Kanban Board on GitHub

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

5. Project Constraints

Since this is an MVP (Minimum Viable Product) intended for validation before final implementation, the project must follow key constraints to ensure the solution's security, compliance, and efficiency, as well as data protection.

The main constraints to be considered are:

1. **Restricted Data Access**

a) External users or those without a direct link to the company will not have access to internal information, ensuring data security and confidentiality.

2. **Dependency on Existing Infrastructure**

a) The solution must integrate with tools already used by the partner, such as Azure and AWS, without requiring drastic changes to the current infrastructure.

3. **Data Processing Restrictions**

a) Data handling and analysis must comply with strict security and regulatory rules established by the LGPD (General Data Protection Law), preventing unauthorized access or exposure of sensitive information such as: company name, CNPJ or CPF, list of performed transactions, and recipient companies.

b) The implementation must not compromise the performance of existing systems or databases.

4. **Preservation of Current Customer Service Structures**

a) The new system must not completely replace current service practices without a carefully planned and validated transition.

6. Development

6.1. Database Analysis

To store structured data in a graph format, two main database options emerge as viable alternatives: **PostgreSQL** (with graph extensions such as Apache AGE) and

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

Neo4j, a native graph database. The choice between these databases depends on factors such as maintenance, scalability, performance, and query efficiency.

1. **PostgreSQL:** As a relational database, PostgreSQL allows storing graph information using tables and foreign keys. However, modeling relationships in a graph format can become complex and less efficient for queries that require traversing multiple nodes.
2. **Neo4j:** Neo4j is a native graph database specifically designed to handle connected data. It uses a model based on **nodes** and **edges**, enabling optimized and more efficient queries for complex relationships. Its storage and indexing mechanism is built to deliver high performance when navigating highly interconnected data networks.to desempenho ao navegar por redes de informação altamente interligadas.

Although Neo4j is not currently used by the partner company, its use has been approved, and it will be built and deployed through AWS Kubernetes pods. Given that the project is based on analyzing relationships between entities and events, choosing Neo4j is more appropriate.

It offers better efficiency for complex relational queries, allowing for an intuitive and high-performance exploration of connections. Additionally, its Cypher query language simplifies insight extraction, making data analysis more effective and scalable.

6.2. Entity-Relationship Diagram (ERD)

An Entity-Relationship Diagram (ERD) is a visual representation that describes the logical structure of a database, showing how entities relate to each other. It is essential for understanding database modeling, allowing data to be structured in an organized and efficient way.

The presented diagram defines the data structure in a graph format, where nodes represent entities and edges represent relationships between them. Below are the main tables and their functionalities, along with the diagram figure:

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

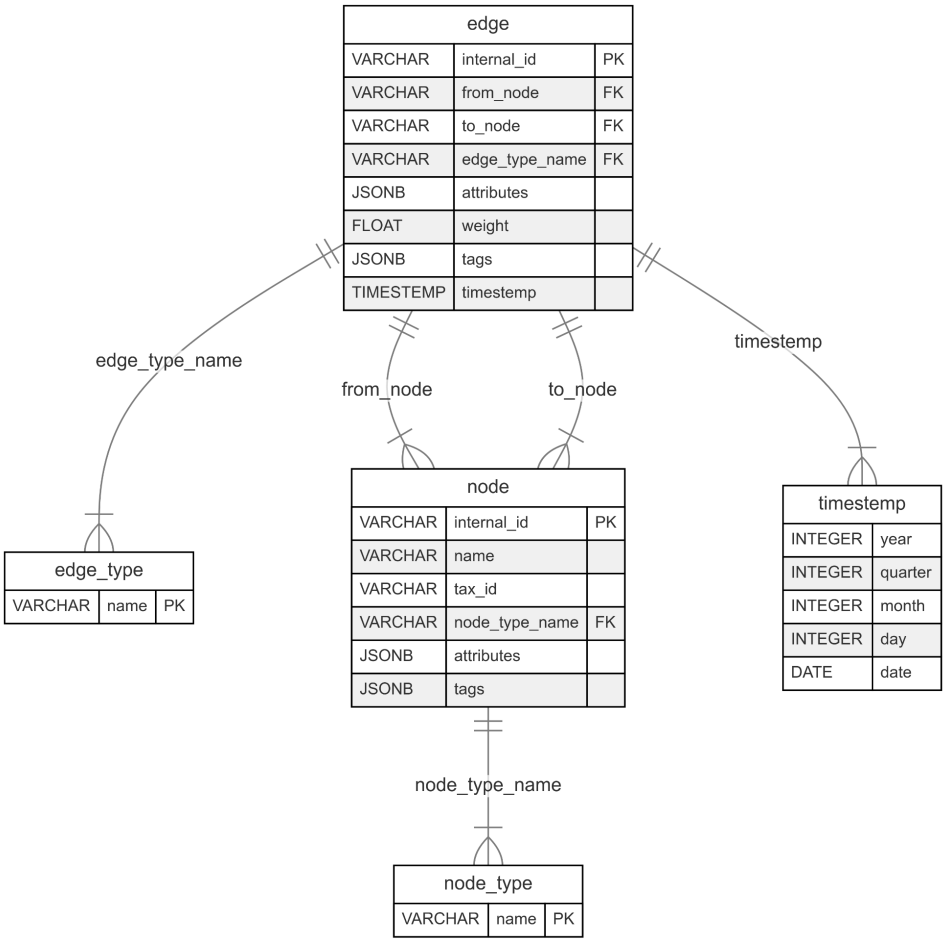

Figure 2: Entity-Relationship Diagram

1. **node (Nodes)**
    a. Represents the main entities in the graph, such as companies or clients.
    b. Includes attributes like `internal_id`, `name`, `tax_id` (tax identification), `node_type_name`, `attributes` (additional data in JSONB format), `tags` (categorization labels), and a timestamp to track creation/modification.
2. **edge (Edges)**
    a. Represents the connections between nodes, indicating relationships between different entities.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

   b. Includes the fields `from_node` and `to_node` to define the origin and destination of the relationship, `edge_type_name` to classify the type of connection, `attributes` (additional data), `weight` (relationship weight), and `tags` for segmentation.
3. **node_type (Node Types)**
   a. Defines the different types of nodes within the database.
   b. Contains a `name` field as a unique identifier.
4. **edge_type (Edge Types)**
   a. Defines the different types of possible relationships between nodes.
   b. Includes a `name` field for classification.

With this structure, the model facilitates the representation and querying of relationships between clients, events, and transactions, ensuring greater flexibility in analysis and decision-making.

6.3. Documentation of Attributes and Entities

The documentation of entities and their attributes in the generated diagram is an essential tool for system development. It facilitates the understanding of the data structure, ensuring a clear view of the entities, their attributes, and the relationships between them. Below, we present a detailed description of each table, specifying its properties, cardinalities, and association rules.

- **Entities:**

Name	**Nodo**
Description	Represents a node in the graph structure. Each node can have different types and attributes.
Type	Normal
Attributes	a. `internal_id` (VARCHAR) – Unique identifier of the node.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

	b. `name` (VARCHAR) – Name of the node. c. `tax_id` (VARCHAR) – CPF/CNPJ associated with the client node. d. `node_type_name` (VARCHAR) – Reference to the node type. e. `attributes` (JSONB) – Tags associated with the node, stored in JSON format. f. `tags` (JSONB) – Tags associadas ao nó, armazenadas em JSON.

Name	**Node_Type**
Description	Defines the different types of nodes that exist.
Type	Normal
Attributes	g. name (VARCHAR) – Node type name (Primary Key).

Name	**Edge**
Description	Represents a connection between two nodes, forming an edge in the graph.
Type	Normal
Attributes	h. `internal_id` (VARCHAR) – Unique identifier of the edge. i. `from_node` (VARCHAR) – Reference to the source node. j. `to_node` (VARCHAR) – Reference to the destination node. k. `edge_type_name` (VARCHAR) – Type of the edge. l. `attributes` (JSONB) – Additional attributes of the edge in JSON format.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

	m. `weight` (FLOAT) – Edge weight, which may indicate relevance or cost. n. `tags` (JSONB) – Tags associated with the edge, stored in JSON format. o. `timestamp` (TIMESTAMP) – Moment of creation or last update of the edge.

Name	**Edge_Type**
Description	Defines the different types of edges that exist.
Type	Normal
Attributes	p. name (VARCHAR) – Edge type name (Primary Key).

Name	**Timestamp**
Description	Represents detailed temporal information to reference specific dates and periods.
Type	Normal
Attributes	q. `year` (INTEGER) – Year. r. `quarter` (INTEGER) – Quarter. s. `month` (INTEGER) – Month. t. `day` (INTEGER) – Day. u. `date` (DATE) – Full date.

- **Relationships:**

Name	**Has**
Description	A node can have one node type.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

Type	Normal
Entities	`node` → `node_type`
Foreign Key	`node.node_type_name → node_type.name`
Cardinality	A node type can be associated with multiple nodes. A node has one node type.

Name	**Connected**
Description	An edge connects two nodes.
Type	Normal
Entities	`edge` → `node` (two times: `from_node` e `to_node`).
Foreign Key	`edge.from_node → node.internal_id` `edge.to_node → node.internal_id`
Cardinality	A node can be connected to multiple other nodes by different edges. An edge can only connect two nodes.

Name	**Has**
Description	An edge has an edge type.
Type	Normal
Entities	`edge` → `edge_type`
Foreign Key	`edge.edge_type_name → edge_type.name`
Cardinality	An edge type can be associated with multiple edges. An edge has one edge type.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

Name	Has
Description	The edge table has a time reference.
Type	Normal
Entities	`edge` → `timestamp`
Foreign Key	————
Cardinality	An edge has a reference to the timestamp data.

6.4. Neo4j setup with Docker

During Sprint 2, the configuration of the **docker-compose.yml** and **Dockerfile** files was completed, which are responsible for managing and launching a Docker container with all the necessary configurations to run both the user interface (UI) application and the Neo4j backend across different types of machines and operating systems.

6.5. First Version of the SQL Script

For the first version of the data script, a **".cql"** file was created containing all the Cypher queries for data creation. This file is also used to automatically load the data when starting the Docker container, ensuring that all example data is inserted into the database without the need for manual input.

It is worth mentioning that the data was generated based on the previously mentioned use cases and follows the structure defined in the Entity-Relationship Diagram presented in **section 15.2**. The file can be found in the **"apps/scripts"** folder.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

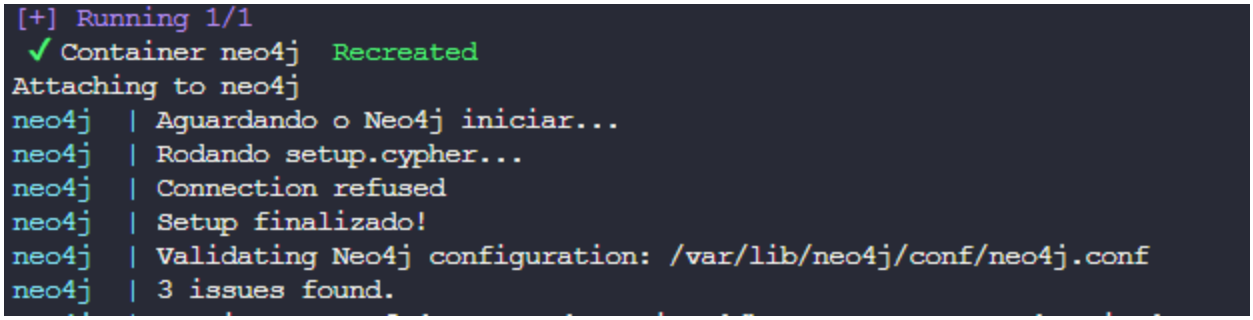

Figure 3: Docker Compose startup logs

6.6. Index Creation for Table Fields

Along with the initial phase of creating sample data, a fundamental aspect to consider is the performance of query execution, especially given the volume of data.

To mitigate potential performance issues in the future without the need to scale the database infrastructure horizontally, it is possible to adopt strategies such as creating indexes on specific fields.

This approach offers benefits such as:

1. **Improved query speed**, reducing response time for frequent searches.
2. **Optimized resource usage**, preventing unnecessary database overload.
3. **Greater scalability**, allowing the database to handle larger volumes of information without compromising performance.

However, it is also important to consider the downsides of this approach:

1. **Additional storage costs**: Creating indexes can consume more disk space, especially when applied to many fields or fields with high cardinality.
2. **Impact on write operations**: Inserting, updating, or deleting data may become slower, as the database needs to keep the indexes updated.
3. **Maintenance complexity**: Choosing the right indexes must be done carefully, as creating inadequate indexes can negatively impact performance, and removing or modifying indexes also requires attention.

6.6.1. Indexes in Neo4j

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

In **Neo4j**, creating indexes is essential for improving query performance, especially in large graphs. Indexes help speed up the search for nodes and relationships based on specific property values.

There are two main types of indexes in Neo4j:

- **Property Indexes**: Allow fast lookup of node and relationship properties. They are used to quickly locate a node or relationship based on the value of a specific property, such as a customer ID or a name.

- **Full-Text Indexes**: Enable large-scale text searches on node or relationship properties. They are useful when performing text-based queries, such as searching for keywords in descriptions or titles.

Creating indexes in Neo4j is straightforward using **Cypher**. For example:

CREATE INDEX ON :Company(tax_id);

This statement creates an index on the **tax_id** property of nodes of type **Company**.

Creating indexes in **Neo4j** provides several benefits, such as improving query performance by reducing the time needed to retrieve data. However, it is essential to use them in moderation, as excessive indexing can negatively impact storage and the response time of write operations. To maximize the benefits of indexes, it is recommended to analyze the most frequent queries and create indexes strategically based on them.

An essential tool for this analysis is the **EXPLAIN** command in Cypher. It allows you to view the query execution plan without actually executing it, helping to understand how the database will process the search. To use it, simply add **EXPLAIN** before the desired Cypher query.

The output will display the execution plan, highlighting operations such as **NodeIndexSeek** (efficient index-based search), **NodeByLabelScan** (search without

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

an index), and other processing steps. With this analysis, it is possible to identify performance bottlenecks and determine whether creating an index would improve query efficiency.

Figure 4: Query with index

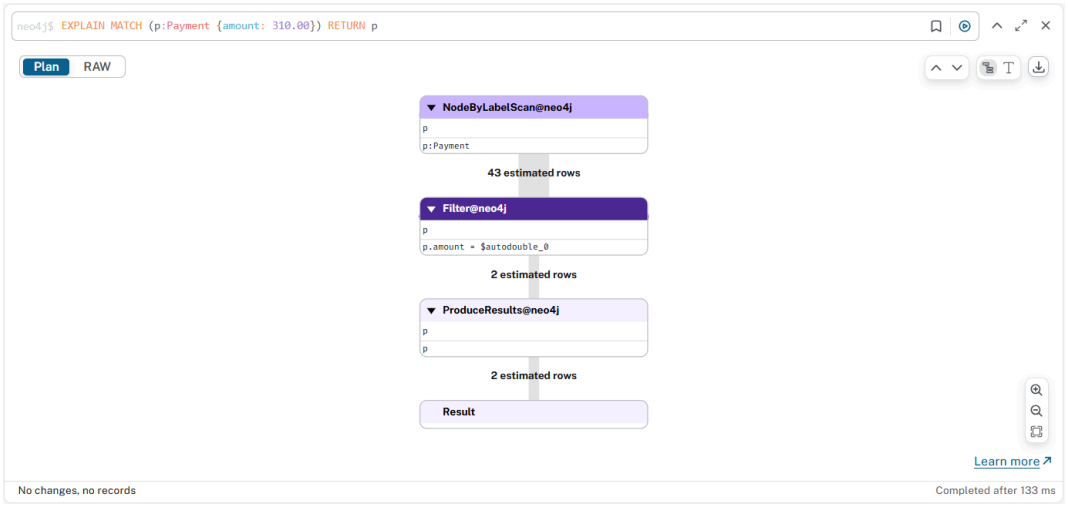

Figure 5: Query without index

In the case of Neo4j, using indexes can be a powerful strategy to optimize queries, but it is crucial to implement them strategically to avoid adverse effects on overall system performance.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

6.6.2. Test Mock with Accurate Data

Sprint 4 aims to conduct tests based on the defined data structure and, consequently, assess the quality of the data generated through queries written in the Cypher language. Below, you can see the results obtained after executing the queries for each formulated question. It is worth noting that all the queries used can be found in the project's repository on GitHub.

1. **A list of all active products belonging to all clients.**

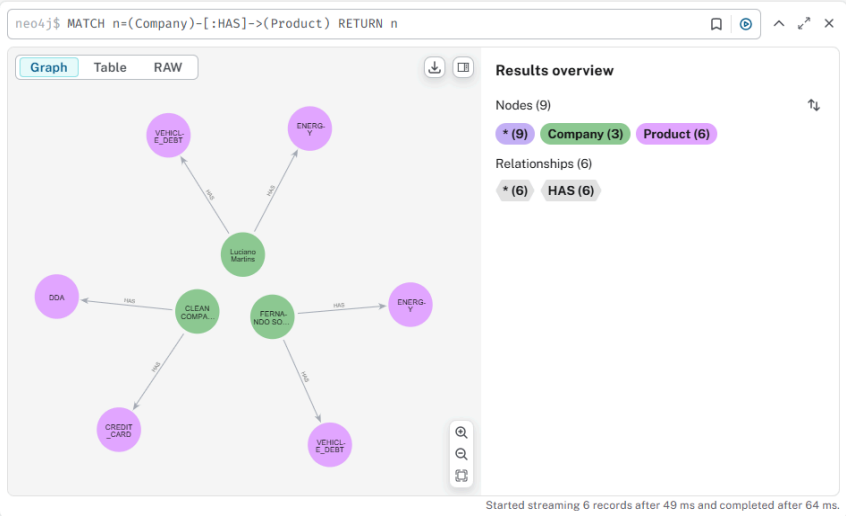

Figure 6: Result of executing the first query.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

2. A list of all deactivated products belonging to all clients.

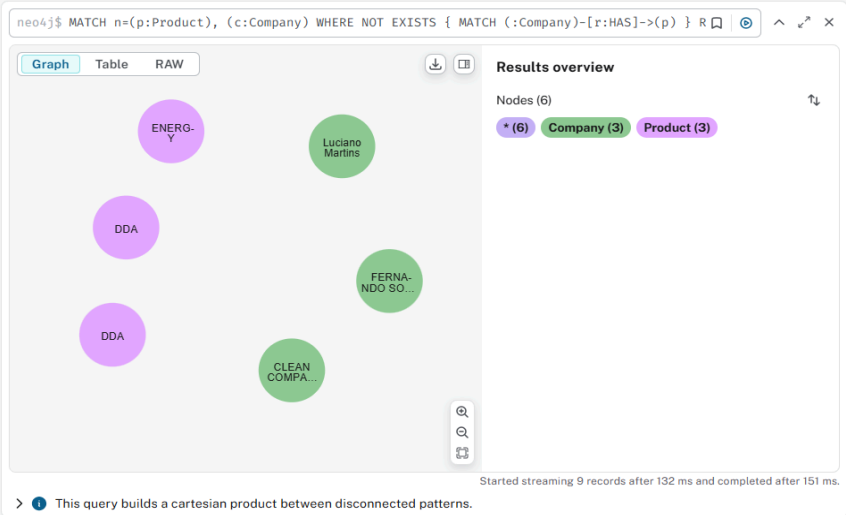

Figure 7: Result of executing the second query.

3. A list of all active products of a client, filtered by CNPJ/CPF.

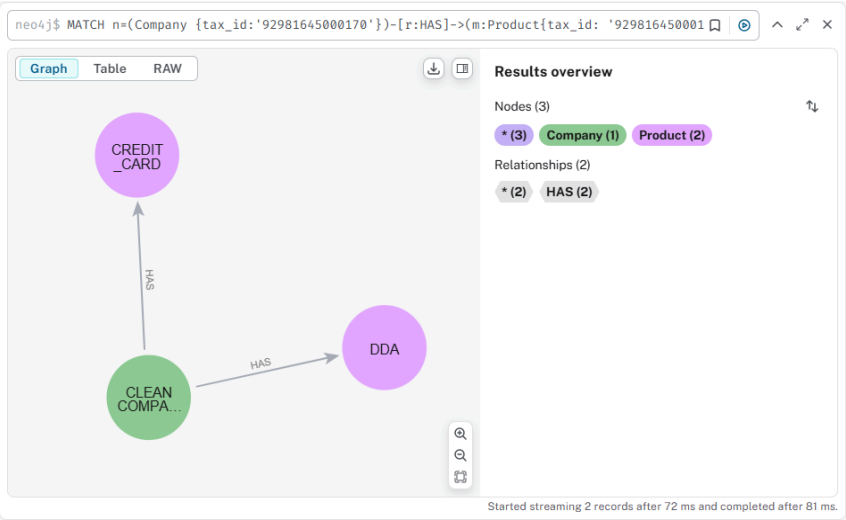

Figure 8: Result of executing the third query.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

4. A list of all deactivated products of a client, filtered by CNPJ/CPF.

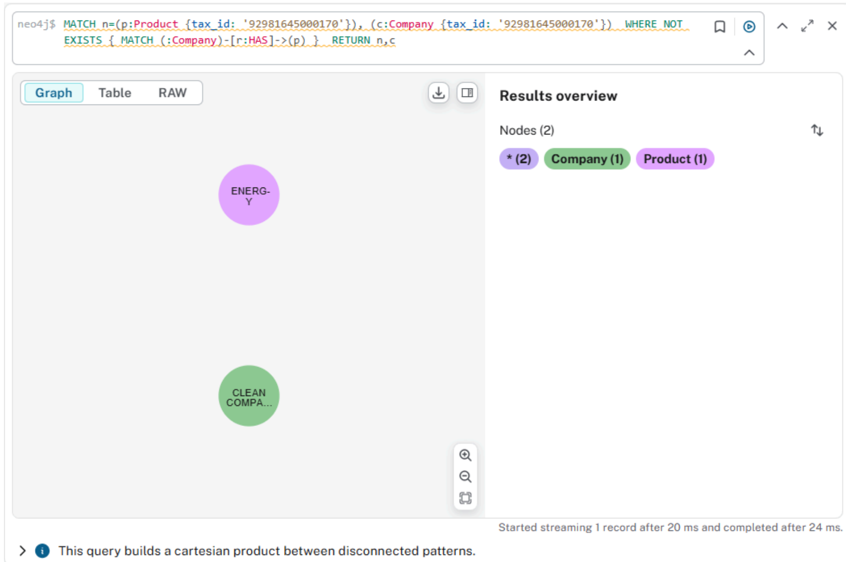

Figure 9: Result of executing the fourthquery.

5. A list of all payments, ordered in descending order.

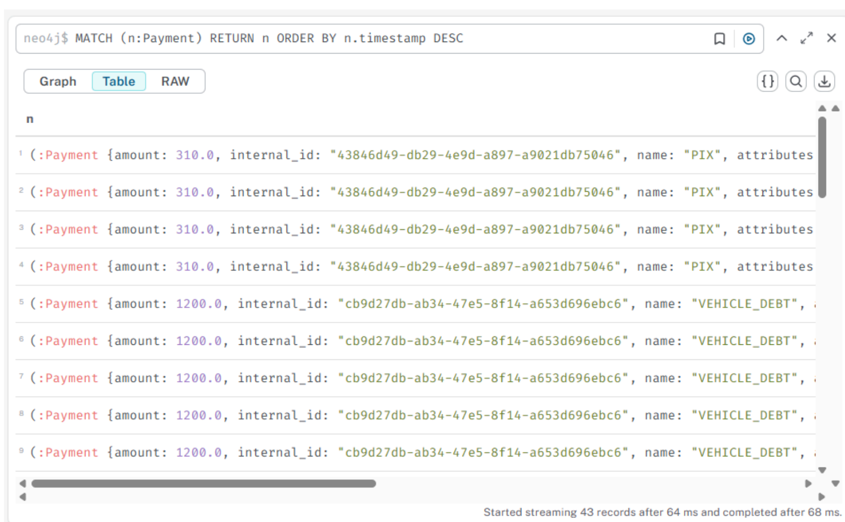

Figure 10: Result of executing the fifth query.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

6. Group of clients who made a total value of X in the month of X.

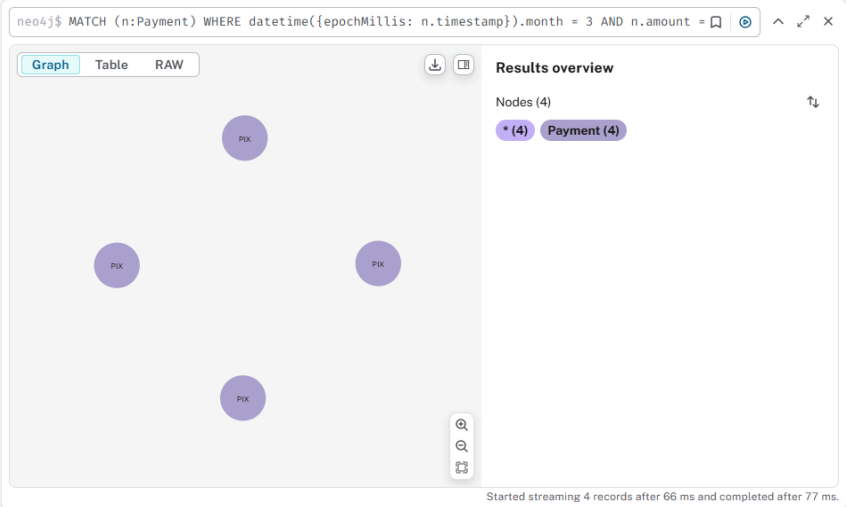

Figure 11: Result of executing the sixth query.

7. Filter payments by payment ID.

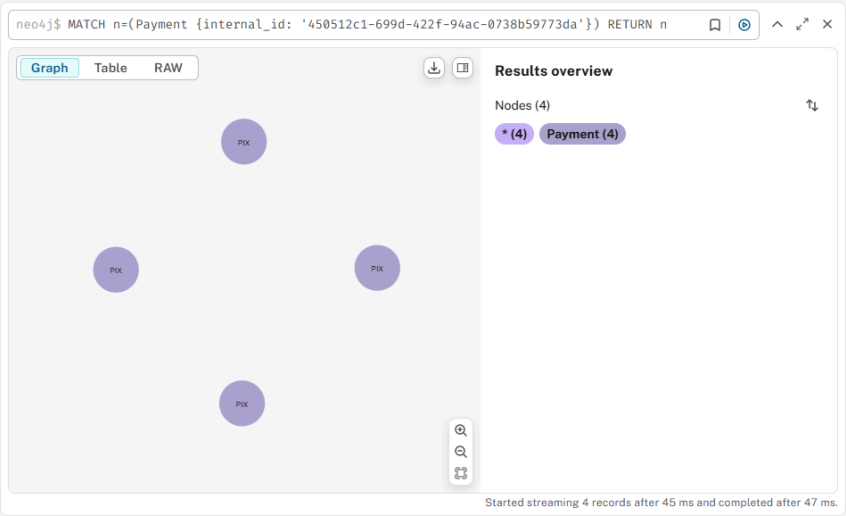

Figure 12: Result of executing the seventh query.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

8. Filter all payments based on CNPJ/CPF.

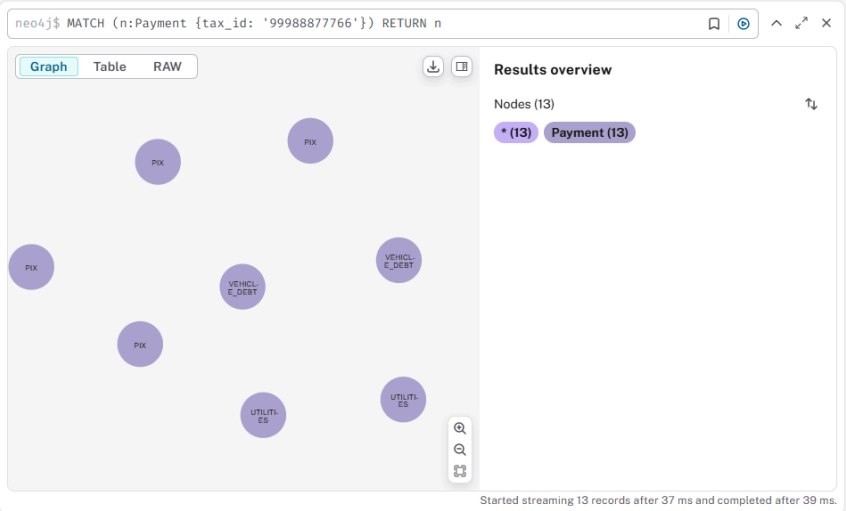

Figure 13: Result of executing the eighth query.

9. Filter payments by product type and CNPJ/CPF.

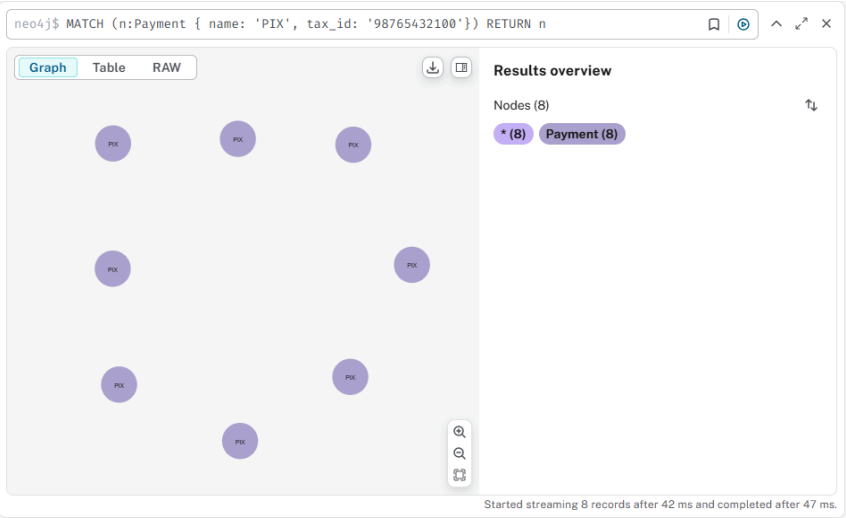

Figure 14: Result of executing the ninth query.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

10. Filter payments with status 'TEMPORIZED'.

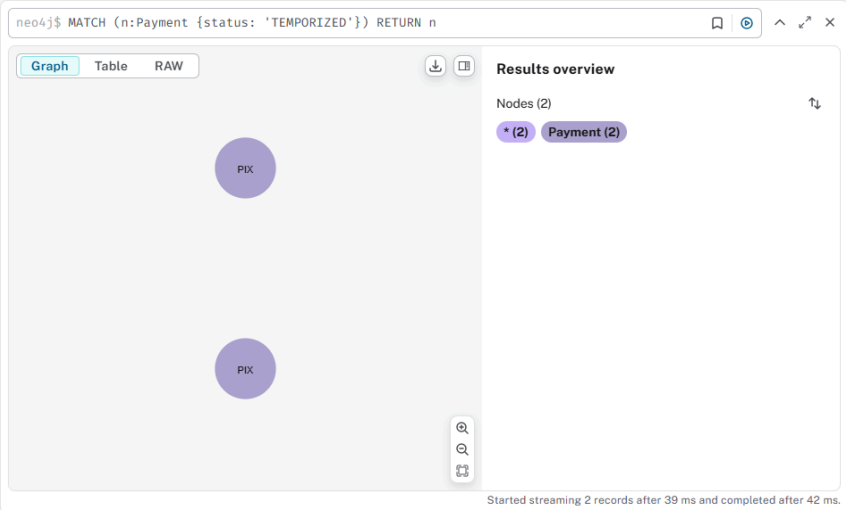

Figure 15: Result of executing the tenth query.

11. Filter payments with status 'FAILED'.

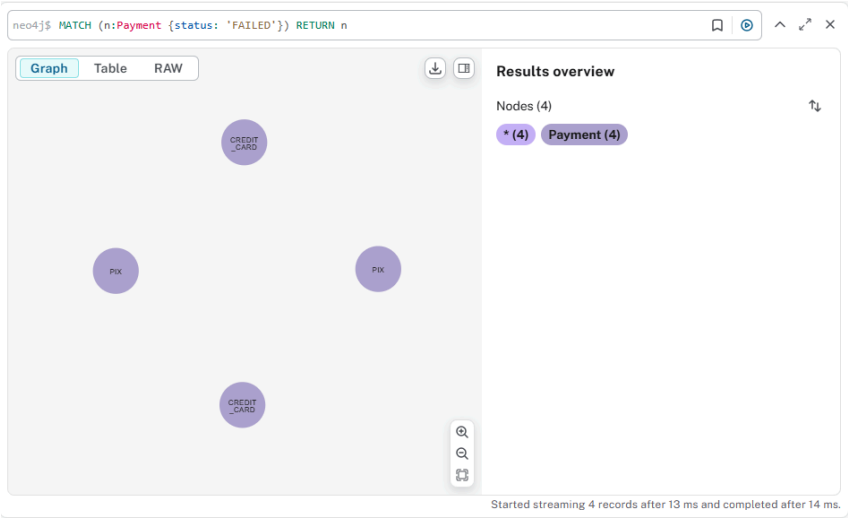

Figure 16: Result of executing the eleventh query.

6.6.3. User access control management.

In Neo4j, it is also possible to manage user access through the creation, updating, and deletion of users, as well as the administration of roles assigned

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

to each user. Neo4j offers the flexibility to force users to change their password after the first login or allow them to keep the password unchanged. Additionally, it is possible to update the roles assigned to users to provide different levels of access as needed.

Example of a query to create a user:

CREATE USER username SET PASSWORD 'password' CHANGE PASSWORD AFTER FIRST LOGIN

Explanation: This command creates a user named username and sets a password. The CHANGE PASSWORD AFTER FIRST LOGIN option forces the user to change the password on the first login.

Example of a query to update a user's roles:

GRANT ROLE reader TO username

Explanation: This command assigns the reader role to the user username, which can be useful for defining specific read-only permissions.

Existing types of roles in Neo4j:

- **admin**: A user with full permissions to administer the database, including creating and deleting users, as well as managing roles.
- **reader**: A user with read-only permissions, meaning they can query the data but cannot make changes.
- **publisher**: A user with read and write permissions, but without the ability to manage users or roles.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br

7. Conclusion

Through this work, it was possible to understand that data collected by systems are essential tools for analyzing user behavior and generating valuable insights for the continuous improvement of products and services. In this context, the use of a graph-oriented database proves to be especially effective, as it facilitates the visualization and exploration of connections between data, expanding the possibilities for analyzing both current records and those to be stored in the future.

Inteli - Instituto de Tecnologia e Liderança
Avenida Professor Almeida Prado, 520, Butantã, São Paulo, SP.
CEP: 05508-070
www.inteli.edu.br