Universitat Politècnica de Catalunya

Facultad de Informática de Barcelona

BDMA JOINT PROJECT REPORT

SEMANTIC DATA MABNAGEMENT (SDM)

Spring 2024

Authors:
Arijit Samal
arijit.samal@estudiantat.upc.edu
Aayush Paudel
aayush.paudel@estudiantat.upc.edu

Supervisors:
Prof. OSCAR ROMERO

Contents

1 Introduction

This project leverages graph-based solutions to enhance data management and analytics within our Big Data Management (BDM) system. By representing our data as a graph, we can efficiently explore complex relationships, conduct advanced analytics, and derive insights that are difficult to achieve with traditional relational databases. This report outlines the methodology, implementation, and benefits of employing graph-based solutions, culminating in a proof of concept (PoC) to demonstrate the effectiveness of the process.

YOU CAN ACCESS THE PROJECT AT- GitHub Link

2 Purpose Statement

This project aims to utilize graph-based solutions to improve data exploration, analytics, and decision-making within our BDM system. By modeling our data as a graph, we can leverage advanced graph algorithms to uncover hidden patterns and relationships, ultimately enhancing our understanding and strategic planning for our business.

2.1 Benefits

- **Enhanced Relationship Exploration**: Graph databases allow us to model and query complex relationships between entities such as customers, products, and transactions more naturally and efficiently than relational databases.
- **Advanced Analytics**: Graph algorithms provide powerful algorithms for uncovering patterns, community structures, and key influencers within the data.
- **Improved Performance**: Graph databases are optimized for traversing relationships, making them ideal for queries that involve complex joins and deep hierarchies.
- **Flexibility**: A graph database schema can evolve easily, allowing us to add new types of relationships and nodes without significant rework.

3 Architecture

3.1 Lambda Graph Analytics Architecture

We employ a Lambda Architecture to efficiently handle large-scale data processing and analytics. In the Batch Layer, raw data from Parquet files in the formatted zone within the GCS (Google Cloud Storage) bucket is loaded and transformed using Apache Spark. This transformation process organizes the data into distinct entities (customers, products, transactions) and relationships (purchased, consumed, sold_to, located_at). The processed data is then stored in the Serving Layer using Neo4j, a graph database that facilitates the creation and management of a comprehensive graph data model.

For advanced analytics, the Analytics Layer leverages Neo4j's Graph Data Science (GDS) library to run complex graph algorithms such as shortest path, PageRank, and community detection. This architecture supports powerful graph-based insights and facilitates quick querying and analytics, ensuring both historical and near-real-time data processing capabilities. Additionally, detailed logging ensures effective monitoring and troubleshooting across the entire system.

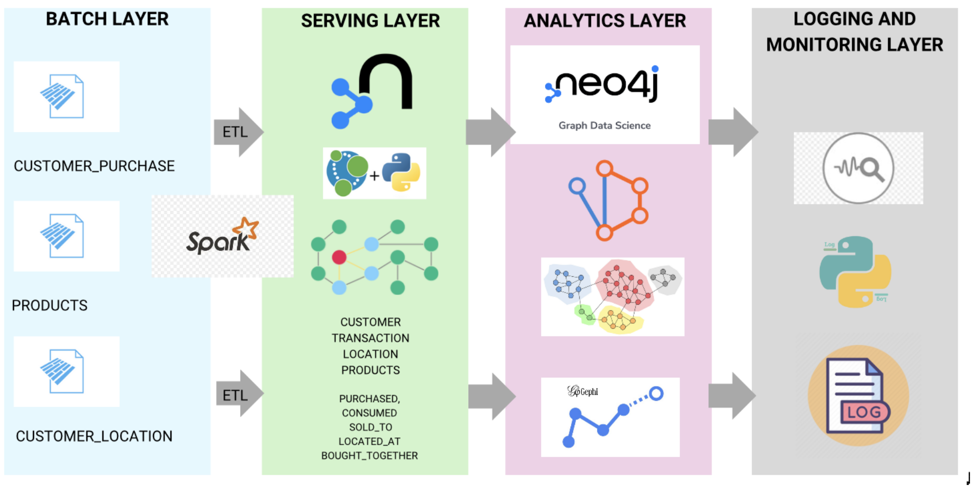


Figure 1: Lambda Graph Analytics Architecture

4 Graph Family

4.1 Chosen Graph Family: Property Graphs

4.1.1 Justification

- **Flexibility**: Property graphs allow nodes and relationships to have properties, capturing detailed information about customers, products, transactions, and locations. This is ideal for our BDM system where entities have rich, complex attributes.

- **Neo4j Compatibility**: Neo4j is a property graph database offering robust support for graph analytics through its Graph Data Science (GDS) library. It provides a mature ecosystem, excellent performance, and comprehensive documentation.

- **Ease of Querying**: The Cypher query language, used with property graphs, is intuitive and powerful for expressing complex graph queries. It supports a wide range of operations.

5 Graph Design

5.1 Schema

The graph schema consists of the following components:
Nodes:

- **Customer**: Represents customers with properties such as `customer_id`, `customer_name`, and `email_id`.

- **Product**: Represents products with properties such as `product_id`, `unit_price`, `category`, and `avg_expiry_days`.

- **Transaction**: Represents transactions with properties such as `transaction_id`, `purchase_date`, `expected_expiry_date`, `expiry_date`, and `expected_price`.

- **Location**: Represents locations with properties such as `location_id`, `country_code`, `postal_code`, `place_name`, `latitude`, and `longitude`.

Relationships:

- `PURCHASED`: Connects `Customer` nodes to `Product` nodes, indicating that a customer purchased a product.
- `CONSUMED`: Connects `Product` nodes to `Transaction` nodes, indicating that a product was consumed in a transaction.
- `SOLD_TO`: Connects `Customer` nodes, indicating that one customer sold a product to another customer.
- `LOCATED_AT`: Connects `Customer` nodes to `Location` nodes, indicating the location of a customer.
- `BOUGHT_TOGETHER`: Connects `Product` nodes, indicating products that were frequently bought together.

5.2 Graph Schema Diagram

```
(Customer)-[:PURCHASED]->(Product)
(Customer)-[:SOLD_TO]->(Customer)
(Customer)-[:LOCATED_AT]->(Location)
(Product)-[:CONSUMED]->(Transaction)
(Product)-[:BOUGHT_TOGETHER]->(Product)
```

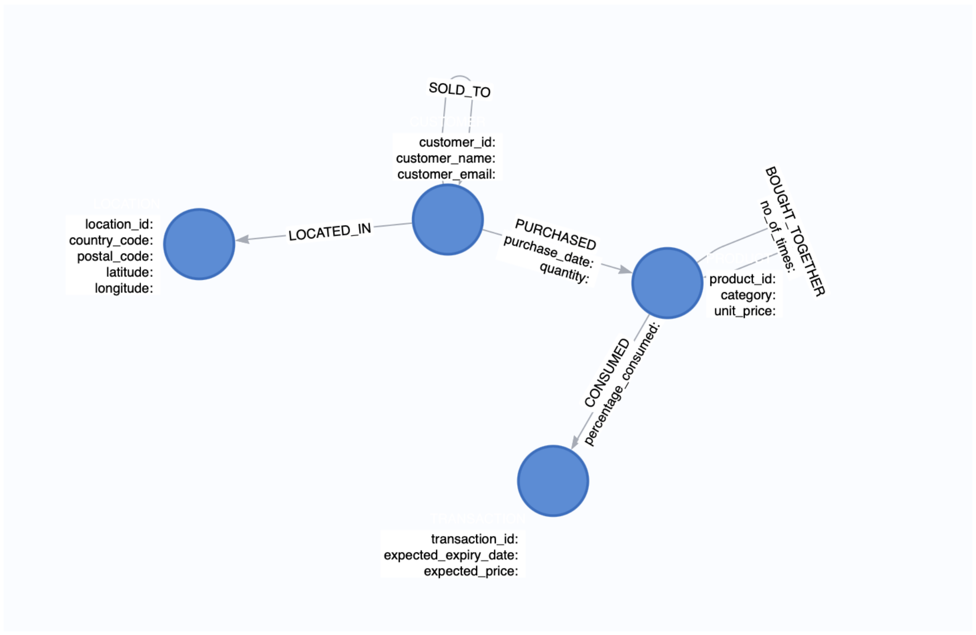


Figure 2: Graph Schema Diagram

ADD A GRAPH INSANCE IMAGE HERE ASKED IN THE DOCUMENT

5.3 Meta-Model Definition

- **Entities**: Customers, Products, Transactions, Locations
- **Relationships**: Purchased, Consumed, Sold_to, Located_at, Bought_together

- **Properties**: Each entity and relationship can capture relevant data points with multiple properties.

6 Data Flow

6.1 Sources

- **Parquet Files**: Customer data, product data, transaction data, and location data are stored as Parquet files in the formatted zone of the GCS (Google Cloud Storage) bucket. These files are selected with Apache Spark for their efficient storage and processing capabilities. The data is read from the bucket and processed further to enable advanced analytics and data management tasks.

6.2 Data Transformation Process

1. **Load Data**: Load data from Parquet files into Spark DataFrames. This process involves reading the files from the GCS bucket.

2. **Transform Data**: Perform necessary transformations and cleanups, such as renaming columns, filtering rows, and joining tables to prepare the data for graph insertion.

3. **Write to Neo4j**: Use the Neo4j driver to write nodes and relationships to the graph database. This involves creating the nodes and relationships as per the defined schema.

6.3 Data Flow Diagram

The data flow consists of loading data, transforming data and finally, writing to neo4j which can be visualized in the architecture diagram in figure 1

7 Graph Analytics

7.1 Community Detection

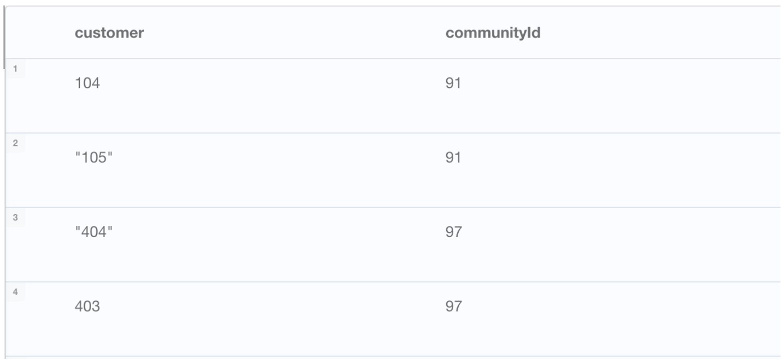
It is used to identify clusters of customers who are densely connected. This helps understand customer segments that interact closely and can be targeted for marketing campaigns.

Algorithm used: Louvain

Implementation Steps:

1. Create a graph projection for customers and their relationships.

2. Run the Louvain algorithm to detect communities and interpret the results.



	customer	communityId
1	104	91
2	"105"	91
3	"404"	97
4	403	97

Figure 3: Results from Louvain Algorithm

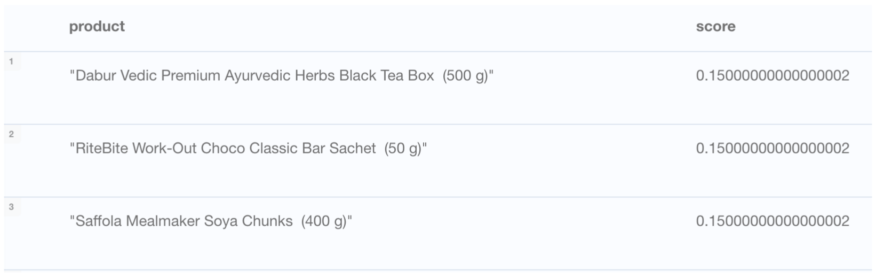
7.2 Centrality Analysis

This analysis is used to determine the most influential customers/products. This helps identify key players in the network who have significant influence. This information is crucial for strategic decisions such as targeted marketing, product promotions, and customer engagement initiatives, ensuring that efforts are focused on the most impactful areas.

Algorithm used: PageRank

Implementation Steps:

1. Create a graph projection for customers and their relationships.
2. Run the PageRank algorithm to calculate centrality scores and interpret the top influential nodes.



	product	score
1	"Dabur Vedic Premium Ayurvedic Herbs Black Tea Box (500 g)"	0.15000000000000002
2	"RiteBite Work-Out Choco Classic Bar Sachet (50 g)"	0.15000000000000002
3	"Saffola Mealmaker Soya Chunks (400 g)"	0.15000000000000002

Figure 4: Results from Page Rank

7.3 Pathfinding

Pathfinding aims to determine the shortest path between two customers within the network. This analysis is crucial for understanding the most efficient route of interactions or connections between any two customers in the network. By identifying the shortest interaction path, businesses can gain insights into direct and indirect customer relationships, optimize communication strategies, and enhance network navigation and efficiency.

Algorithm: Shortest Path

Implementation Steps:

1. Create a graph projection including all relevant nodes and relationships.
2. Run the shortest path algorithm between two specified nodes and interpret the resulting

path.

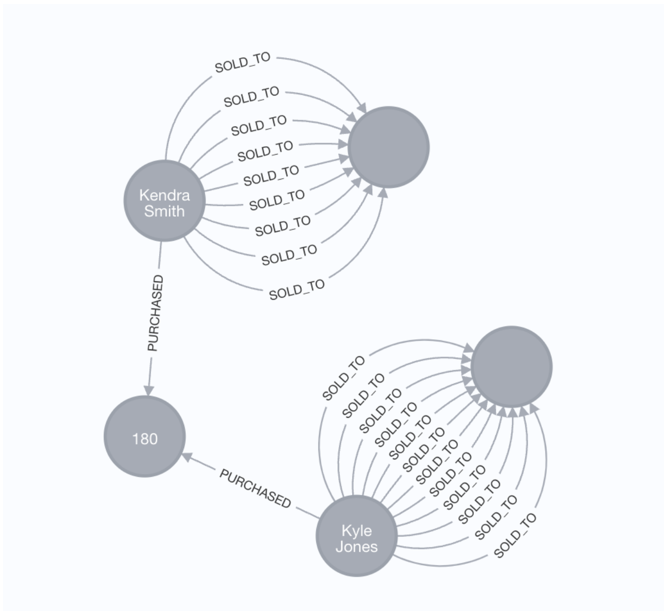


Figure 5: Results from Shortest Path Algorithm

Interpretation: As shown in Figure 5, the shortest path between Customer '232' (Kyle Jones) and Customer '233' involves a series of transactions centered around the product "Happilo Premium Natural Whole W320 Cashews (Kaju) (200 g)." Kyle Jones purchased this product on March 5, 2024, and sold it to Customer '233' on April 19, 2024. Before this, Kyle Jones bought the same product from Kendra Smith (Customer '231') on March 2, 2024. This path highlights the interconnectedness of these customers through their purchases and sales of the same product, illustrating a clear transactional relationship.

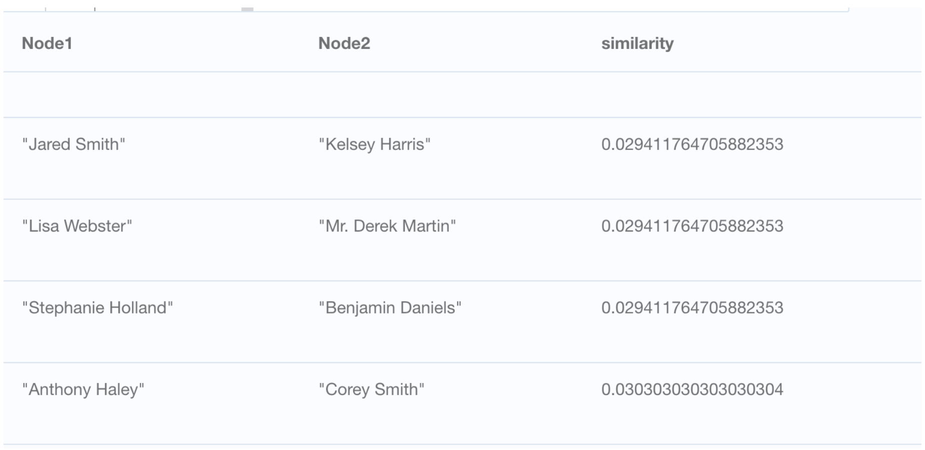
7.4 Link Prediction

The purpose of link prediction is to forecast potential new connections between nodes in the network, such as between customers or products. This analysis helps identify which pairs of nodes are likely to form connections in the future, providing insights into evolving relationships and emerging trends. This algorithm calculates the likelihood of a connection forming between two nodes based on their shared neighbors, giving higher weight to less common neighbors. By predicting future links, businesses can proactively engage with customers or products likely to become influential, fostering new opportunities for growth and interaction within the network.

Algorithm: Node Similarity (Adamic-Adar)

Implementation Steps:

1. Create a graph projection for customers and products with their relationships.
2. Run the node similarity algorithm using the Adamic-Adar metric and interpret the top predicted links.



Node1	Node2	similarity
"Jared Smith"	"Kelsey Harris"	0.029411764705882353
"Lisa Webster"	"Mr. Derek Martin"	0.029411764705882353
"Stephanie Holland"	"Benjamin Daniels"	0.029411764705882353
"Anthony Haley"	"Corey Smith"	0.030303030303030304

Figure 6: Results from Node Similarity

7.5 Proximity-Based Customer Grouping

This involves clustering customers based on their geographical locations, which aids in regional marketing campaigns and understanding local customer clusters.

Implementation Steps:

1. Create a point representation for each customer's location.
2. Calculate the distance between each pair of customers.
3. Group customers by proximity and extract the nearest neighbors.
4. Interpret and log the results, showing the nearest customers for each customer.

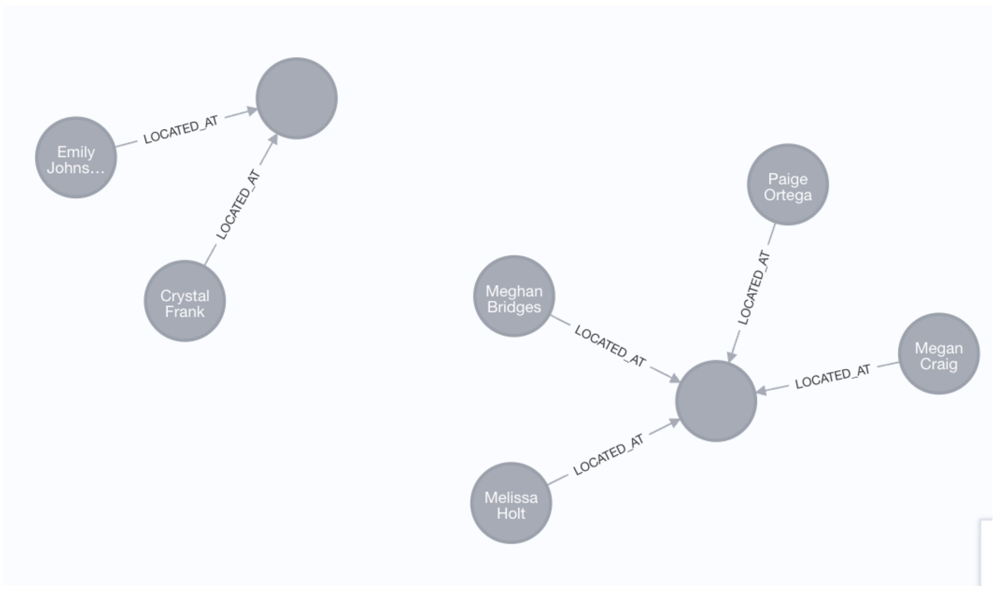


Figure 7: Proximity Based Customer Grouping

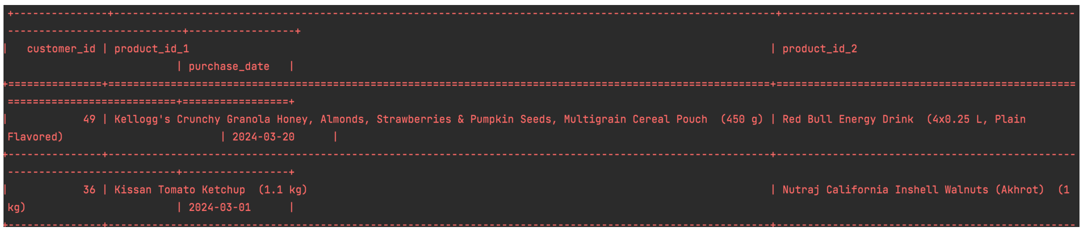
Explanation: The 'group_nearest_customers' function groups customers based on their geographical proximity. It calculates the distance between each pair of customers using their latitude and longitude and identifies the nearest neighbors. This information is logged to help understand local customer clusters. It can be used for targeted regional marketing, allowing businesses to better cater to the preferences and behaviors of geographically close customer groups.

7.6 Product Co-Purchase Analysis

The purpose of product co-purchase analysis is to identify pairs of products that are frequently bought together by customers. This analysis helps understand customer purchase patterns and enables businesses to develop effective product bundling strategies.

Implementation Steps:

1. Perform a self-join on the transaction data to create pairs of products bought by the same customer in the same transaction.
2. Count the occurrences of each product pair.
3. Select the top pairs of products most frequently bought together.
4. Join with the original data to get additional details such as customer ID and purchase date.
5. Create and log the relationships in the graph database.



```
+--------------+----------------------------------------------------------------------------------------------+-----------------------------------------------
--------------------------+----------------+
|   customer_id | product_id_1                                                                                 | product_id_2
                          | purchase_date   |
+==============+==============================================================================================+===============================================
==========================+================+
|           49 | Kellogg's Crunchy Granola Honey, Almonds, Strawberries & Pumpkin Seeds, Multigrain Cereal Pouch  (450 g) | Red Bull Energy Drink  (4x0.25 L, Plain
Flavored)                          | 2024-03-20      |
+--------------+----------------------------------------------------------------------------------------------+-----------------------------------------------
--------------------------+----------------+
|           36 | Kissan Tomato Ketchup  (1.1 kg)                                                              | Nutraj California Inshell Walnuts (Akhrot)  (1
kg)                       | 2024-03-01      |
+--------------+----------------------------------------------------------------------------------------------+-----------------------------------------------
```

Figure 8: Results from Co-Purchase Analysis

Explanation: The 'create_bought_together_relationships' function identifies pairs of products frequently purchased together by customers. This function generates product pairs bought in the same transaction by performing a self-join on the transaction data. It then calculates the frequency of each product pair, selecting the top pairs with the highest co-purchase occurrences. These relationships are logged in the graph database, allowing businesses to visualize and analyze co-purchase patterns. Insights gained from this analysis can inform product bundling and cross-selling strategies, ultimately enhancing customer experience and driving sales.

8 Proof of Concept (PoC)

8.1 Tools

- **Graph Database**: Neo4j, chosen for its robust support for property graphs and graph analytics.
- **Data Processing**: Apache Spark, chosen for its ability to handle large datasets and perform complex transformations efficiently.
- **Programming Language**: Python, chosen for its extensive library support and ease of integration with Neo4j and Spark.
- **Graph Analytics**: Neo4j Graph Data Science (GDS) Library, chosen for its comprehensive suite of graph algorithms for graph-based analytics for our business.

8.2 Implementation

1. **Setup**: Install and configure Neo4j and Apache Spark. Ensure both tools are connected and ready for data processing.
2. **Data Loading**: Load data from Parquet files into Spark DataFrames.
3. **Data Transformation**: Transform the data to fit the schema requirements for Neo4j.
4. **Graph Population**: Use the Neo4j driver to populate the database with nodes and relationships.
5. **Graph Analytics**: Perform various graph analytics such as community detection, centrality analysis, pathfinding, link prediction etc.
6. **Validation**: Validate the results by interpreting the insights and ensuring they align with business goals.

8.3 Validation and Interpretation

The validation process involves interpreting the results of the graph analytics to ensure they provide meaningful insights. For example:

- **Community Detection**: Verify that detected communities make sense regarding customer behavior.
- **Centrality Analysis**: Ensure that identified influential nodes align with known key players in the business.
- **Pathfinding**: Confirm that the shortest paths identified are logical and useful for understanding customer interactions.
- **Link Prediction**: Validate the predicted links by comparing them with historical data and known trends.

9 Conclusion

This report outlines the methodology and implementation of property graph-based data management and analytics solutions within our BDM system. By leveraging the Neo4j graph database and its Graph Data Science (GDS) library, we can perform advanced graph analytics such as community detection, centrality analysis, pathfinding, and link prediction. The proof of concept demonstrates the end-to-end process, showcasing the benefits and insights derived from graph-based analytics.