Group Project - Neo4j

of Systems and Methods for Big and Unstructured Data Course (SMBUD)

held by

Brambilla Marco

Tocchetti Andrea

Group 78

Pisante Giuseppe 10696936

Raffaelli Martina 10709893

Academic year 2024/2025

POLITECNICO
MILANO 1863

Contents

1 Introduction

This project develops a comprehensive overview of an open source ecommerce dataset [2], through the use of the Neo4j technology, which is very well suited for this type of database.
In particular, in order to fully exploit such technology, our study was developed using a relational database management system (RDBMS), which facilitates the querying of data to support the Marketing and Supply Chain division with key functions like product recommendations, customer segmentation, sales analysis, and inventory forecasting. The system allows for tracking of customer purchase history, order details, product categories, payment methods, and discounts, helping businesses optimize their operations and enhance the customer experience.
This report provides an overview on the dataset, the data wrangling process, the assumptions made and a total of 20 Cypher queries that could provide interesting insights for the Marketing and Logistics departments.

2 Data Wrangling

Our main restructuring to the dataset was to add the entity history, with which we can track the information related to each user in time. This is useful to understand the evolution of the user's data and to make predictions based on the user's past behavior.
In order to add such entity to the dataser, the following query was executed:

```
LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (u:User {id: row.id}) // Match the user node
MERGE (h:History {user_id: row.id}) // Create a new History node for the user
ON CREATE SET
    h.city = row.city,
    h.state = row.state,
    h.postal_code = row.postal_code,
    h.orders_list = [],
    h.number_of_orders = 0
MERGE (u)-[:PERFORM_EVENT]->(h);
```

In addition, we ran the following Cypher query to evaluate the presence of null entries:

```
MATCH (o:DistributionCenter)
WHERE o.id IS NULL
RETURN count(o) AS missingIdCount;
```

Resulting in 0 missing IDs. This was repeated for all the relevant attributes of the entities, with the same result.
It is important to note that the following query would result in multiple null IDs, which would be expected as the attribute related to the package delivery could be null, indicating that the package hasn't been shipped, delivered or returned. In addition, the number of null related to the attribute "shipped_at" is lower than the number of null related to the attribute "delivered_at", which is expected as the package must be shipped first before being delivered.

```
MATCH (o:Order)
WHERE o.shipped_at IS NULL
RETURN count(o) AS missingIdCount;
```

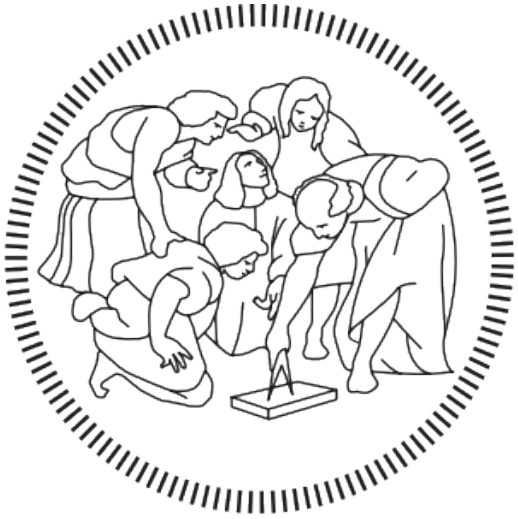
In conclusion, we did not find necessary to modify the dataset format, as it was already well structured and ready to be imported into the Neo4j database.

3 Assumptions

The project is based on the following assumptions:

- Each customer has a unique ID, name, surname, email address, and date of birth.
- Each product has a unique identifier, name, description, price, and category.
- Each product can be associated with multiple categories.
- Each customer can place multiple orders.
- Each order can contain multiple products.
- Each order has a unique ID, order date, total amount, and payment method.
- Each product can have multiple variants (e.g., size, color).
- Each product can have multiple reviews, where each review includes a rating and comments from a customer.
- Each payment is associated with a unique transaction ID, payment method, and amount.
- Each customer can leave a review for a product if and only if they have purchased it.

4 Dataset



POLITECNICO
MILANO 1863

Figure 1: E-R Diagram

4.1 Entities:

Starting from the considerations previously exposed regarding the implementation hypotheses, we have drawn an ER diagram (**Figure 1**) which includes 8 different entities and 8 many-to-many relationships described below in the logical model:

- **User**(user_id, Name, Email, Address, PhoneNumber, DateOfBirth)
- **Product**(product_id, ProductName, Description, Price, StockLevel, Category)
- **Order**(order_id, user_id, OrderDate, TotalAmount, PaymentMethod, ShippingAddress)
- **Order_Item**(order_id, product_id, Quantity, UnitPrice)

- **Payment**(payment_id, order_id, PaymentDate, PaymentAmount, PaymentMethod)
- **Review**(review_id, user_id, product_id, Rating, Comment, ReviewDate)
- **Category**(category_id, CategoryName)
- **Distribution_Center**(center_id, CenterName, Location)
- **History**(user_id, City, State, PostalCode, OrdersList, NumberOfOrders)

The **User** entity represents the individuals who make purchases on the platform, storing essential information such as their personal details and contact information.

The **Product** entity represents the items available for sale, including product details such as name, description, price, stock level, and category.

The **Order** entity captures information related to a user's order, such as the order date, total amount, payment method, and shipping address.

The **Order_Item** entity links products to specific orders, detailing the quantity of each product ordered and its unit price.

The **Payment** entity records the payment details for each order, including the payment method, amount paid, and the date of payment.

The **Review** entity stores customer feedback for purchased products, including ratings and comments, along with the review date.

The **Category** entity defines the product categories, allowing for classification of products into different types (e.g., electronics, clothing).

The **History** entity tracks the user's historical data, including the city, state, postal code, order list, and the number of orders placed.

Finally, the **Distribution_Center** entity represents the centers from which the products are shipped, capturing details such as center name and location.

4.2 Relationships:

User $(User) - [: PLACED] - > (Order)$
Relationship between a User and an Order, where a user can place multiple orders, and each order is linked to one user.

Order $(Order) - [: CONTAINS] - > (Order\_Item)$
Relationship between an Order and the Order_Item, where an order can contain multiple items, and each order item is linked to one order.

Product $(Product) - [: PART\_OF] - > (Order\_Item)$
Relationship between a Product and the Order_Item, where a product can appear in multiple order items, and each order item corresponds to a specific product.

User $(User) - [: WRITES] - > (Review)$
Relationship between a User and a Review, where a user can write multiple reviews, and each review is written by one user.

Product $(Product) - [: HAS\_REVIEW] - > (Review)$
Relationship between a Product and a Review, where a product can have multiple reviews, and each review is linked to one product.

Order $(Order) - [: PAID\_BY] - > (Payment)$
Relationship between an Order and a Payment, where an order can have one or more payments, and each payment corresponds to one order.

Order $(Order) - [: SHIPPED\_FROM] - > (Distribution\_Center)$
Relationship between an Order and a Distribution_Center, where an order is processed by a single distribution center, and each distribution center can process multiple orders.

Product $(Product) - [: BELONGS\_TO] - > (Category)$
Relationship between a Product and a Category, where a product can belong to one or more categories, and each category can contain multiple products.

User $(User) - [: PERFORM\_EVENT] - > (History)$
Relationship between a User and a History, where a user can have multiple historical records, and each historical record is linked to one user.

4.3 Constraints:

Based on the implementation assumptions and the entities defined above, we introduce several constraints to ensure the integrity and consistency of the data in the database. These constraints are applied to guarantee that each entity has a unique identifier and to prevent the creation of duplicate entries within the database.

Specifically, the following constraints are defined:

```
CREATE CONSTRAINT FOR (d:DistributionCenter) REQUIRE d.id IS UNIQUE
CREATE CONSTRAINT FOR (u:User) REQUIRE u.id IS UNIQUE;
CREATE CONSTRAINT FOR (p:Product) REQUIRE p.id IS UNIQUE;
CREATE CONSTRAINT FOR (i:InventoryItem) REQUIRE i.id IS UNIQUE;
CREATE CONSTRAINT FOR (o:Order) REQUIRE o.order_id IS UNIQUE;
CREATE CONSTRAINT FOR (oi:OrderItem) REQUIRE oi.id IS UNIQUE;
CREATE CONSTRAINT ON (h:History) ASSERT h.user_id IS UNIQUE;
```

These constraints enforce uniqueness on the primary identifiers of the key entities:

- The **DistributionCenter** entity's ID (`d.id`) must be unique.
- The **User** entity's ID (`u.id`) must be unique.
- The **Product** entity's ID (`p.id`) must be unique.
- The **InventoryItem** entity's ID (`i.id`) must be unique.
- The **Order** entity's ID (`o.order_id`) must be unique.
- The **OrderItem** entity's ID (`oi.id`) must be unique.
- The **History** entity's user ID (`h.user_id`) must be unique.

These constraints ensure that the database remains consistent, with no duplicate entries for the core entities.

4.4 Data Import:

```
  // Load Distribution Centers
LOAD CSV WITH HEADERS FROM "file:///distribution_centers.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (d:DistributionCenter {id: row.id})
ON CREATE SET d.name = row.name,
              d.latitude = toFloat(row.latitude),
              d.longitude = toFloat(row.longitude);

// Load Users
LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (u:User {id: row.id})
ON CREATE SET u.first_name = row.first_name,
              u.last_name = row.last_name,
              u.email = row.email,
              u.age = toInteger(row.age),
              u.gender = row.gender,
              u.state = row.state,
              u.city = row.city,
              u.country = row.country,
              u.latitude = toFloat(row.latitude),
              u.longitude = toFloat(row.longitude),
              u.traffic_source = row.traffic_source,
              u.created_at = row.created_at
ON MATCH SET u.count = coalesce(u.count, 0) + 1;

// Load Products
LOAD CSV WITH HEADERS FROM "file:///products.csv" AS row
```

```
WITH row WHERE row.id IS NOT NULL
MERGE (p:Product {id: row.id})
ON CREATE SET p.name = row.name,
              p.brand = row.brand,
              p.category = row.category,
              p.cost = toFloat(row.cost),
              p.retail_price = toFloat(row.retail_price),
              p.department = row.department,
              p.sku = row.sku,
              p.distribution_center_id = row.distribution_center_id
ON MATCH SET p.count = coalesce(p.count, 0) + 1;

// Load Inventory Items
LOAD CSV WITH HEADERS FROM "file:///inventory_items.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (i:InventoryItem {id: row.id})
ON CREATE SET i.product_id = row.product_id,
              i.created_at = row.created_at,
              i.sold_at = row.sold_at,
              i.cost = toFloat(row.cost),
              i.product_category = row.product_category,
              i.product_name = row.product_name,
              i.product_brand = row.product_brand,
              i.product_retail_price = toFloat(row.product_retail_price),
              i.product_department = row.product_department,
              i.product_sku = row.product_sku,
              i.product_distribution_center_id = row.product_distribution_center
ON MATCH SET i.count = coalesce(i.count, 0) + 1;

// Load Orders
LOAD CSV WITH HEADERS FROM "file:///orders.csv" AS row
WITH row WHERE row.order_id IS NOT NULL
MERGE (o:Order {order_id: row.order_id})
ON CREATE SET o.user_id = row.user_id,
              o.status = row.status,
              o.created_at = row.created_at,
              o.shipped_at = row.shipped_at,
              o.delivered_at = row.delivered_at,
              o.returned_at = row.returned_at,
              o.num_of_item = toInteger(row.num_of_item),
              o.gender = row.gender
```

```
ON MATCH SET o.count = coalesce(o.count, 0) + 1;

// Load Order Items
LOAD CSV WITH HEADERS FROM "file:///order_items.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (oi:OrderItem {id: row.id})
ON CREATE SET oi.order_id = row.order_id,
              oi.user_id = row.user_id,
              oi.product_id = row.product_id,
              oi.inventory_item_id = row.inventory_item_id,
              oi.status = row.status,
              oi.created_at = row.created_at,
              oi.shipped_at = row.shipped_at,
              oi.delivered_at = row.delivered_at,
              oi.returned_at = row.returned_at
ON MATCH SET oi.count = coalesce(oi.count, 0) + 1;
```

5 Cypher Queries

In order to provide the Marketing and Logistics departments with the necessary insights, we have developed a series of Cypher queries that address various aspects of the business operations.

5.1 Market Analysis for Marketing Department:

This section of the Cypher Queries aims at providing the Marketing department with insights on the share within the market by providing the number of orders in the Countries in which the Company operates. In particular, this is done by providing an overview on the total orders per country, the revenue per country, a demographic overview per country and an analysis of the most present brands within a specific category, which in this study was chosen as the category with the most orders for further relevance. In particular, a very useful instrument for the Marketing department is the segmentation of the user base, which allows to divide the users into different categories based on their frequency and spending habits.

5.1.1 Total orders by Country

This queries the database to determine the total number of orders placed by users in each country, providing insights into the distribution of orders across different regions.

```
MATCH (u:User)-[:PLACES]->(o:Order)
RETURN u.country AS country, COUNT(o) AS total_orders
ORDER BY total_orders DESC;
```

5.1.2 Revenue per Country

This query calculates the total revenue generated by orders placed by users in each country, providing insights into the revenue distribution across different regions.

```
MATCH (u:User)-[:PLACES]->(o:Order)-[:CONTAINS]->(oi:OrderItem)
MATCH (oi)-[:REFERS_TO]->(p:Product)
RETURN u.country AS country, SUM(p.cost) AS total_revenue
ORDER BY total_revenue DESC;
```

Table 1: Total Orders by Country

Country	Total Orders
China	42,986
United States	28,099
Brasil	18,262
South Korea	6,620
France	5,968
United Kingdom	5,673
Germany	5,286
Spain	4,965
Japan	2,945
Australia	2,630
Belgium	1,441
Poland	325
Colombia	19
España	4
Austria	2
Deutschland	1

5.1.3 Demographic Overview

This query provides an overview of the user demographics, including the total number of users, the average age, and the country they belong to.

```
MATCH (u:User)
RETURN u.country AS country,
     COUNT(u) AS total_users,
     AVG(u.age) AS average_age
ORDER BY total_users DESC;
```

5.1.4 Categories with the most orders

This query identifies the top three product categories with the highest number of orders, providing insights into the most popular product categories among customers.

```
MATCH (oi:OrderItem)-[:REFERS_TO]->(p:Product)
RETURN p.product_category AS category, COUNT(oi) AS total_orders
ORDER BY total_orders DESC
LIMIT 3;
```

Table 2: Total Revenue by Country

Country	Total Revenue
China	1,800,865.85
United States	1,162,263.11
Brasil	747,519.90
South Korea	278,958.29
France	243,130.02
United Kingdom	242,263.07
Germany	217,875.61
Spain	210,278.08
Japan	124,807.02
Australia	106,009.74
Belgium	60,226.06
Poland	13,386.90
Colombia	572.51
España	91.87
Deutschland	65.70
Austria	41.51

5.1.5 Most present brands within the intimates category

This query identifies the top five brands with the highest number of sales within the intimates category, providing insights into the most popular brands in this product category.

```
MATCH (oi:OrderItem)-[:REFERS_TO]->(p:Product)
WHERE p.category = "Intimates"

RETURN p.brand AS brand,
      COUNT(oi) AS total_sales
ORDER BY total_sales DESC
LIMIT 5;
```

Table 3: Total Users and Average Age by Country

Country	Total Orders	Average Value
China	34,150	40.89
United States	22,522	41.21
Brasil	14,507	41.19
South Korea	5,316	41.25
France	4,700	41.57
United Kingdom	4,561	41.05
Germany	4,155	40.86
Spain	4,062	41.01
Japan	2,438	40.89
Australia	2,146	40.98
Belgium	1,185	39.54
Poland	235	42.43
Colombia	17	34.88
Deutschland	2	40.50
España	2	38.50
Austria	2	50.00

Table 4: Categories with the most orders

Category	Total Orders
Intimates	13,474
Jeans	12,698
Tops & Tees	11,925

Table 5: Most present brands within the intimates categor

Brand	Total Sales
Bali	405
Maidenform	383
Hanes	364
Laura	342
Vanity Fair	306

5.1.6 Most present brands within the intimates category per country

This query identifies the top-selling brand within the intimates category for each country, providing insights into the brand preferences of customers in different regions.

```
MATCH (u:User)-[:ORDERED]->(oi:OrderItem)-[:REFERS_TO]->(p:Product)
WHERE p.category = "Intimates"
WITH u.country AS country,
    p.brand AS brand,
    COUNT(oi) AS total_sales
ORDER BY country, total_sales DESC
WITH country, COLLECT({brand: brand, total_sales: total_sales}) AS brand_sal
RETURN country,
      brand_sales[0].brand AS top_brand,
      brand_sales[0].total_sales AS top_sales
ORDER BY country;
```

Table 6: Total Orders and Average Value by Country

Country	Total Orders	Average Value
China	34,150	40.89
United States	22,522	41.21
Brasil	14,507	41.19
South Korea	5,316	41.25
France	4,700	41.57
United Kingdom	4,561	41.05
Germany	4,155	40.86
Spain	4,062	41.01
Japan	2,438	40.89
Australia	2,146	40.98
Belgium	1,185	39.54
Poland	235	42.43
Colombia	17	34.88
Deutschland	2	40.50
España	2	38.50
Austria	2	50.00

5.1.7 Top traffic source per Country

This query identifies the top traffic source for users in each country, providing insights into the most effective marketing channels for customer acquisition.

```
MATCH (u:User)
WHERE u.traffic_source IS NOT NULL
WITH u.country AS country, u.traffic_source AS traffic_source, COUNT(*) AS a
WITH country, traffic_source, amount
ORDER BY country, amount DESC
WITH country, COLLECT({traffic_source: traffic_source, amount: amount}) AS t
RETURN country, traffic_data[0].traffic_source AS top_traffic_source, traffi
ORDER BY country;
```

Table 7: Search Data by Country

Country	Platform	Count
Australia	Search	1518
Austria	Facebook	1
Belgium	Search	856
Brasil	Search	10147
China	Search	23876
Colombia	Search	9
Deutschland	Search	1
España	Search	2
France	Search	3310
Germany	Search	2921
Japan	Search	1746
Poland	Search	177
South Korea	Search	3701
Spain	Search	2845
United Kingdom	Search	3197
United States	Search	15768

5.1.8 Segmentation of User base

This query segments users based on their frequency and spending habits into four categories: High Frequency, High Spending; High Frequency, Low Spending; Low Frequency, High Spending; Low Frequency, Low Spending.

```
MATCH (u:User)-[:PLACES]->(o:Order)-[:CONTAINS]->(oi:OrderItem)-[:REFERS_TO]
WITH u, o, SUM(p.cost) AS total_price, COUNT(oi) AS order_count
WITH u, total_price, order_count,
    CASE
        WHEN total_price > 30 AND order_count > 3 THEN 'High Frequency, High
        WHEN total_price <= 30 AND order_count > 3 THEN 'High Frequency, Low
        WHEN total_price > 30 AND order_count <= 3 THEN 'Low Frequency, High
        ELSE 'Low Frequency, Low Spending'
    END AS segment
RETURN segment, COUNT(u) AS user_count
ORDER BY user_count DESC;
```

Table 8: Customer Segments by Frequency and Spending

Segment	Count
Low Frequency, Low Spending	67152
Low Frequency, High Spending	51702
High Frequency, High Spending	6340
High Frequency, Low Spending	32

5.1.9 Best and Worst Selling Product Categories by Season

This query analyzes the purchasing patterns of product categories by season. For each season (Winter, Spring, Summer, and Fall), it identifies the most frequently purchased category (best-selling) and the least frequently purchased category (worst-selling). The results are based on the frequency of orders containing products from each category, with the categories being ranked by the total number of purchases in each season.

```
MATCH (u:User)-[:PLACES]->(o:Order)-[:CONTAINS]->(oi:OrderItem)-[:REFERS_TO]
WITH p.category AS product_category,
  CASE
      WHEN toInteger(substring(o.created_at, 5, 2)) IN [12, 1, 2] THEN 'Wint
      WHEN toInteger(substring(o.created_at, 5, 2)) IN [3, 4, 5] THEN 'Sprin
      WHEN toInteger(substring(o.created_at, 5, 2)) IN [6, 7, 8] THEN 'Summe
      WHEN toInteger(substring(o.created_at, 5, 2)) IN [9, 10, 11] THEN 'Fal
  END AS season,
  COUNT(*) AS frequency
```

```
WITH season, product_category, frequency
ORDER BY season, frequency DESC
WITH season, collect({category: product_category, freq: frequency}) AS categ
RETURN season, categories[0] AS top_category, categories[-1] AS worst_catego
```

Table 9: Customer Segments by Frequency and Spending

Season	TopCategory	WorstCategory
"Fall"	"category": "Intimates","freq": 3854	"category": "Clothing Sets","freq": 63
"Spring"	"category": "Intimates", "freq": 2543	"category": "Clothing Sets", "freq": 36
"Summer"	"category": "Intimates", "freq": 3078	"category": "Clothing Sets", "freq": 52
"Winter"	"category": "Intimates", "freq": 3999	"category": "Clothing Sets", "freq": 62

The result we obtain may seem strange since in all the seasons the top and worst categories are the same, so we checked the correctness of the result with the following query, which computes the frequency of every category in each season:

```
MATCH (u:User)-[:PLACES]->(o:Order)-[:CONTAINS]->(oi:OrderItem)-[:REFERS_TO]->(p
WITH p.category AS product_category,
     CASE
         WHEN toInteger(substring(o.created_at, 5, 2)) IN [12, 1, 2] THEN 'Winte
         WHEN toInteger(substring(o.created_at, 5, 2)) IN [3, 4, 5] THEN 'Spring
         WHEN toInteger(substring(o.created_at, 5, 2)) IN [6, 7, 8] THEN 'Summer
         WHEN toInteger(substring(o.created_at, 5, 2)) IN [9, 10, 11] THEN 'Fall
     END AS season,
     COUNT(*) AS frequency
RETURN product_category, season, frequency
ORDER BY frequency DESC;
```

5.1.10 Age-based Segmentation

This query segments users based on their age into the following groups: 18-24, 25-34, 35-44, 45-54, 55-64, and 65+. It then calculates the total number of users, total amount spent, and average amount spent per user for each age group and returns the results in ascending order of age group. Analysing the results we can see that the values of the average amount spent per user are quite similar across all age groups, so we should rely on the total amount spent to identify the most valuable segments, which is going to be the 65+ group.

```
MATCH (u:User)-[:PLACES]->(o:Order)-[:CONTAINS]->(oi:OrderItem)-[:REFERS_TO]->
WITH u,
    CASE
        WHEN u.age >= 18 AND u.age <= 24 THEN '18-24'
        WHEN u.age >= 25 AND u.age <= 34 THEN '25-34'
        WHEN u.age >= 35 AND u.age <= 44 THEN '35-44'
        WHEN u.age >= 45 AND u.age <= 54 THEN '45-54'
        WHEN u.age >= 55 AND u.age <= 64 THEN '55-64'
        ELSE '65+'
    END AS age_group,
    p.cost AS product_price
WITH age_group, COUNT(DISTINCT u) AS user_count, SUM(product_price) AS total_s
WITH age_group, user_count, total_spent,
    CASE
        WHEN user_count > 0 THEN total_spent / user_count
        ELSE 0
    END AS avg_spent_per_user
RETURN age_group, user_count, total_spent, avg_spent_per_user
ORDER BY age_group;
```

Table 10: Customer Segmentation

AgeGroup	UserCount	TotalSpent	AvgSpentPerUser
"18-24"	9505	616204.0050523246	64.82945871144919
"45-54"	13507	876656.9810595925	64.903900278344
"35-44"	13654	879153.417123852	64.38797547413593
"25-34"	13486	882255.4374447308	65.42009768980652
"55-64"	13630	896658.8652415214	65.78568343664867
"65+"	16262	1057426.51384831	65.02438284640942

5.1.11 Top Product Categories Purchased by Gender in the Selected Segment

This query identifies the top 5 product categories with the highest purchase frequency for each gender (male and female) for the users belonging to the 65+ age group. This analysis can provide insights into the preferences and behaviors of the older demographic, helping tailor marketing strategies to better target this segment.

```
MATCH (u:User)-[:PLACES]->(o:Order)-[:CONTAINS]->(oi:OrderItem)-[:REFERS_TO]->
```

```
WHERE u.age >= 65
WITH u.gender AS gender, p.category AS product_category, COUNT(*) AS frequency
WITH gender, product_category, frequency
ORDER BY frequency DESC
WITH gender, COLLECT({category: product_category, frequency: frequency}) AS ca
WITH gender, categories[0..5] AS top_categories
UNWIND top_categories AS top_category
RETURN gender, top_category.category AS product_category, top_category.frequen
ORDER BY gender, frequency DESC;
```

Table 11: Top 5 Product Categories per Gender

Gender	ProductCategory	Frequency
"F"	"Intimates"	1349
"F"	"Swim"	544
"F"	"Dresses"	539
"F"	"Fashion Hoodies & Sweatshirts"	536
"F"	"Maternity"	524
"M"	"Underwear"	786
"M"	"Tops & Tees"	773
"M"	"Jeans"	771
"M"	"Pants"	726
"M"	"Fashion Hoodies & Sweatshirts"	692

5.1.12 Top 10 Users by Orders Number and Total Spending

This query identifies the top 10 users who placed the highest number of orders. It also calculates the total amount spent by each user by summing the costs of the products in their orders. The results are sorted first by the number of orders in descending order, and in case of ties, by total spending in descending order. For each user, the query returns their user ID, email, number of orders, and total spending, offering a comprehensive overview of the platform's most valuable customers. The idea behind it could be to increase fidelization by offering them special discounts or promotions.

```
MATCH (u:User)-[:PLACES]->(o:Order)
WITH u, COUNT(o) AS num_orders
MATCH (u)-[:PLACES]->(:Order)-[:CONTAINS]->(oi:OrderItem)-[:REFERS_TO]->(p:Pro
WITH u.id AS user_id,
```

```
    u.email AS email,
    num_orders,
    SUM(p.cost) AS total_spent
ORDER BY num_orders DESC, total_spent DESC
LIMIT 10
RETURN user_id, email, num_orders, total_spent;
```

Table 12: User Orders and Spending

User ID	Email	Num Orders	Total Spent
15746	matthewwashington@example.net	4	753.38
80671	timothysmith@example.com	4	724.57
94183	matthewmaynard@example.org	4	689.41
78682	danielhayden@example.net	4	687.55
22656	michaelkim@example.net	4	660.35
75693	andreagray@example.com	4	623.43
9461	juanmiller@example.org	4	614.99
5836	nathandavis@example.com	4	608.57
93588	johnjohnson@example.net	4	599.86
62477	jessicasloan@example.net	4	591.42

5.1.13 Users without Orders

This query retrieves the user IDs and email addresses of all users who have not placed any orders. The idea behind it is that the marketing department could use the user's email to encourage them to make a purchase by sending them targeted promotions or discounts.

```
MATCH (u:User)
WHERE NOT EXISTS {
    MATCH (u)-[:PLACES]->(:Order)
}
RETURN u.id AS user_id, u.email AS email
ORDER BY u.id
LIMIT 10;
```

5.2 Logistic Analysis for Logistics Department:

This section of the Cypher Queries aims at providing the Logistics department with insights on the real-time tracking of the orders and all the possible

Table 13: User IDs and Emails

User ID	Email
10	henrydennis@example.org
1001	michaelanderson@example.org
10016	aprilstafford@example.org
10020	jasmineperkins@example.com
10027	charlessheppard@example.org
10033	sarahward@example.org
10068	maureenbryant@example.org
10073	markwoods@example.net
10081	sandymeyer@example.org
10082	ronniemills@example.com

information related to tuser in order to make the delivery as smooth as possible. In particular, this is done by evaluating the closest distribution center for a specific user, the status of the order, the update of the history of the orders of user and to check if the user has some order pending, in which case the items can be sent together.

5.2.1 Closest Distribution Center

This query calculates the distance between a specific user and all distribution centers in the database, and returns the name of the closest distribution center along with the distance in meters.

```
MATCH (u:User)-[:PERFORM_EVENT]->(h:History)
WHERE u.id = "100"
MATCH (d:DistributionCenter)
WHERE d.latitude IS NOT NULL AND d.longitude IS NOT NULL
WITH u, d,
    point({latitude: u.latitude, longitude: u.longitude}) AS user_location,
    point({latitude: d.latitude, longitude: d.longitude}) AS distribution_cent
WITH u, d, user_location, distribution_center_location,
    point.distance(user_location, distribution_center_location) AS distance
ORDER BY distance
LIMIT 1
RETURN d.name, distance;
```

Table 14: Data for Los Angeles CA

Location	Value
Los Angeles CA	10539304.46257035

5.2.2 Order History

In order to provide our logistic department with the most updated information, we have to provide them with the history of the orders of a specific user. In particular, we have to fill the history node with the latest information. In order to do that, we first add the new order to order, and then we add the new order to the history of the user.

```
MATCH (u:User)
WHERE u.id="100"
WITH u
MATCH (o:Order)
WITH u, MAX(o.order_id) AS max_order_id
CREATE (newOrder:Order {
    order_id: toString(200000),
    status: "Processing",
    gender: u.gender,
    created_at: date(),
    returned_at: null,
    shipped_at: null,
    delivered_at: null,
    num_of_item: 3
})
MERGE (u)-[:PLACES]->(newOrder);
```

```
MATCH (u:User)-[:PERFORM_EVENT]->(h:History)
WHERE u.id = "100"
MATCH (o:Order)
WHERE o.order_id ="300000"
SET h.orders_list = coalesce(h.orders_list, []) + "300000",
    h.number_of_orders = h.number_of_orders +1
MERGE (u)-[:USER_HAS_ORDER]->(o)
```

5.2.3 Number of orders pending

This query retrieves the number of orders that are currently pending for a specific user. This information can be used by the logistics department

to prioritize the processing and delivery of orders for users with multiple pending orders.

```
MATCH (u:User)-[:PLACES]->(o:Order)
WITH u, count(o) AS total_orders
WHERE total_orders = 10 AND u.id = "100"
RETURN u.id AS user_id, total_orders
```

5.2.4 Management of multiple orders

This query retrieves the order IDs of all orders that are currently pending for a specific user. This information can be used by the logistics department to group multiple orders together for delivery, reducing the number of shipments and improving efficiency.

```
MATCH (u:User)-[:PLACES]->(o:Order)
WHERE u.id="100" AND o.status = 'Processing'
RETURN o.order_id, o.status AS order_status, o.created_at;
```

Table 15: Data for Processing Task

ID	Status	Date
300000	Processing	2024-12-11

5.2.5 Set shipping date or delivered date

This query sets the shipping date or delivered date for a specific order. This information can be used to track the progress of the order and provide customers with real-time updates on the status of their delivery.

```
MATCH (o:Order )
WHERE o.order_id = "300000"
RETURN o.shipped_at;
```

This can be implemented for the delivery date by switching the attribute **shipped_at** with **delivered_at**.

5.2.6 Order Status

This query retrieves the status of the most recent order placed by a specific user. This information can be used by the logistics department to track the progress of the order and provide real-time updates to the user

```
MATCH (u:User)-[:PLACES]->(o:Order)
WHERE u.id = "100"
WITH o ORDER BY o.created_at DESC LIMIT 1
RETURN o.order_id, o.status AS order_status, o.created_at;
```

Table 16: Data for Processing Task

ID	Status	Timestamp
561	Processing	2023-12-15 17:28:00+00:00

5.2.7 Cross-Sell Opportunity

This query identifies the top 10 pairs of products that are frequently purchased together in the same order. This information can be used by the logistics department to optimize the packaging and delivery process by grouping frequently purchased items together.

```
MATCH (oi1:OrderItem)-[:REFERS_TO]->(p1:Product),
    (oi2:OrderItem)-[:REFERS_TO]->(p2:Product)
WHERE oi1.order_id = oi2.order_id AND p1.id < p2.id
WITH p1.id AS product1_id, p2.id AS product2_id, oi1.order_id AS order_id
WITH product1_id, product2_id, COUNT(DISTINCT order_id) AS frequency
RETURN product1_id, product2_id, frequency
ORDER BY frequency DESC
LIMIT 10;
```

!!!! TODO: ADD TABLE WITH RESULTS

5.2.8 Average Shipping Time by Country

This query computes the average shipping time for each country. In particular it checks if the order has been delivered and created in the same month, and then calculates the difference in days between the two dates.

```
MATCH (u:User)-[:PLACES]->(o:Order)
WHERE o.delivered_at IS NOT NULL AND o.created_at IS NOT NULL
  AND toInteger(substring(o.delivered_at, 5, 2)) = toInteger(substring(o.creat
WITH u.country AS country, AVG(toInteger(substring(o.delivered_at, 8, 2)) - to
    avg_shipping_time
RETURN country, avg_shipping_time
ORDER BY avg_shipping_time DESC;
```

Table 17: Data for Processing Task

Country	**AvgShippingTime**
"Austria"	6.0
"Colombia"	4.857142857142857
"Japan"	3.9200000000000004
"France"	3.845439650464229
"Spain"	3.8244325767690226
"Brasil"	3.8161648177496104
"Germany"	3.8119062697910056
"Belgium"	3.805104408352669
"United States"	3.797328420082921
"China"	3.7945538906214358
"South Korea"	3.794032723772863
"United Kingdom"	3.7925840092699894
"Australia"	3.779141104294479
"Poland"	3.670103092783506
"España"	2.0

6 Conclusion

In conclusion, we have successfully implemented a comprehensive set of Cypher queries to analyze the e-commerce dataset and provide valuable insights for different departments within the organization. The queries cover a wide range of use cases, by leveraging the power of Neo4j's graph database and Cypher query language.
This project was a very useful instrument to apply hands-on knowledge of Cypher queries and graph databases, and it allowed us to explore the potential of Neo4j in real-world scenarios.

7 References & Sources

[1] Course Slides

[2] Mustafa Keser, *Looker E-commerce BigQuery Dataset*, https://www.kaggle.com/datasets/mustafakeser4/looker-ecommerce-bigquery-dataset?select=order_items.csv%20reference/

[3] https://neo4j.com/docs/cypher-manual/current/

[4] https://neo4j.com/developer/cypher/