POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Samuele Forner 10726370**

Lara Longhi 11098675

Francesco Cavalieri 11020855

Group Number: **49**

Academic Year: 2024-2025

Contents

1 | Introduction

In this report we will present an analysis of two totally unrelated datasets:

- The first one contains data on food stores across the United States. Since the data points (which were available in csv format) presented a lot of objects belonging to clearly distinct classes (like Customers, Region, ... see more details later) and several different relationships with their own attributes, the most natural choice for the Database technology to use was Neo4j, in order to represent the different entities as nodes and correctly model the relationships between them.
  All the queries are somehow useful to improve the business in different ways, like finding the best spot for a new warehouse, analyzing the effectiveness of a promotion or even designing some new ones;

- The other one is about people's job experiences:
  Do they like their current occupation? Are they satisfied about their salary? Are they willing to change their career path?
  Also in this case we found the data in a csv format, where every different row of the file represented a different person. Moreover, there are little to no links between the data points.
  For this two reasons the perfect way to work with these data was to store them in a json-like format, and there is no better Database technology to work with this kind of data format, as far as we know, than MongoDB.

In the next pages we will provide a better insight about the datasets, as well as how we manipulated them in order to make them suitable for our purpose and which query we performed on them.

2 | Data Wrangling

The MongoDB dataset presented all the values of the fields of the documents in string format, thus preventing us from performing any kind of mathematical operation or comparison on them. In order to perform useful queries, we had to convert all the fields that contained numbers, from strings to integer values.
This goal has been achieved by running the following javascript script on the MongoDB shell:

```
db.Carrer.updateMany(
  {},
  [
    {
      $set: {
        // Convert any field of the document
        allFields: {
          $arrayToObject: {
            $map: {
              input: { $objectToArray: "$$ROOT" }, // Convert any document in key-values' arrays'
              as: "field",
              in: {
                k: "$$field.k", // Keep the key unchanged
                v: {
                  $cond: [
                    {
                      // Verify if the value is a numerical string
                      $and: [
                        { $eq: [{ $type: "$$field.v" }, "string"] },
                        { $regexMatch: { input: "$$field.v", regex: /^[0-9]+$/ } }
                      ]
                    },
                    { $toInt: "$$field.v" }, // Convert to number if the value is a numerical string
                    "$$field.v" // Keep the original values otherwise
                  ]
                }
              }
            }
          }
        }
      }
    },
    {
      $replaceRoot: { newRoot: "$allFields" } // Restore the documents to the original format
    }
  ]
)
```

3 | Datasets

3.1. FoodMart dataset

A dataset used to manage the classic Food Marts and their sales, located in different regions, but with a more flexible and scalable NoSql approach, in this case Neo4j.
Nodes are linked among them with relationships, here below is provided a comprehensive list of those relationships:

Node Labels	Relationships
[Customer]	[IN_REGION, PURCHASED_BY]
[Region]	[IN_REGION]
[ProductSubCategory]	[IN_FAMILY, IN_CATEGORY, IN_DEPARTMENT]
[ProductFamily]	[IN_FAMILY]
[ProductCategory]	[IN_CATEGORY]
[Department]	[IN_DEPARTMENT]
[Product]	[IN_CATEGORY, FROM_BRAND, LINE_ITEM]
[Brand]	[FROM_BRAND]
[City]	[IN_REGION, IN_CITY]
[Store]	[OF_TYPE, IN_CITY, IN_STORE]
[StoreType]	[OF_TYPE]
[Year]	[HAS_YEAR]
[Month]	[HAS_YEAR, HAS_DAY]
[Date]	[HAS_DAY, NEXT_DAY, STARTS_ON, ENDS_ON, ON_DATE]
[Promotion]	[STARTS_ON, ENDS_ON, APPLIED_PROMOTION]
[Sale]	[ON_DATE, IN_STORE, PURCHASED_BY, LINE_ITEM, APPLIED_PROMOTION]

An "unique" constraint is applied to each node in order to avoid to add duplicated elements:

```
CREATE CONSTRAINT FOR (r:Region) REQUIRE r.id IS UNIQUE;
CREATE CONSTRAINT FOR (r:Region) REQUIRE r.name IS UNIQUE;
CREATE CONSTRAINT FOR (c:Customer) REQUIRE c.id IS UNIQUE;
CREATE CONSTRAINT FOR (c:Customer) REQUIRE c.account_num IS UNIQUE;
CREATE CONSTRAINT FOR (pf:ProductFamily) REQUIRE pf.name IS UNIQUE;
CREATE CONSTRAINT FOR (d:Department) REQUIRE d.name IS UNIQUE;
CREATE CONSTRAINT FOR (pc:ProductCategory) REQUIRE pc.name IS UNIQUE;
CREATE CONSTRAINT FOR (ps:ProductSubCategory) REQUIRE ps.id IS UNIQUE;
CREATE CONSTRAINT FOR (b:Brand) REQUIRE b.name IS UNIQUE;
CREATE CONSTRAINT FOR (p:Product) REQUIRE p.id IS UNIQUE;
CREATE CONSTRAINT FOR (st:StoreType) REQUIRE st.name IS UNIQUE;
CREATE CONSTRAINT FOR (s:Store) REQUIRE s.id IS UNIQUE;
CREATE CONSTRAINT FOR (y:Year) REQUIRE y.year IS UNIQUE;
CREATE CONSTRAINT FOR (m:Month) REQUIRE m.id IS UNIQUE;
CREATE CONSTRAINT FOR (d:Date) REQUIRE d.id IS UNIQUE;
CREATE CONSTRAINT FOR (d:Date) REQUIRE d.date IS UNIQUE;
CREATE CONSTRAINT FOR (d:Date) REQUIRE d.day IS UNIQUE;
CREATE CONSTRAINT FOR (p:Promotion) REQUIRE p.id IS UNIQUE;
CREATE CONSTRAINT FOR (s:Sale) REQUIRE s.id IS UNIQUE;
```

The desired structure is represented in following figure:

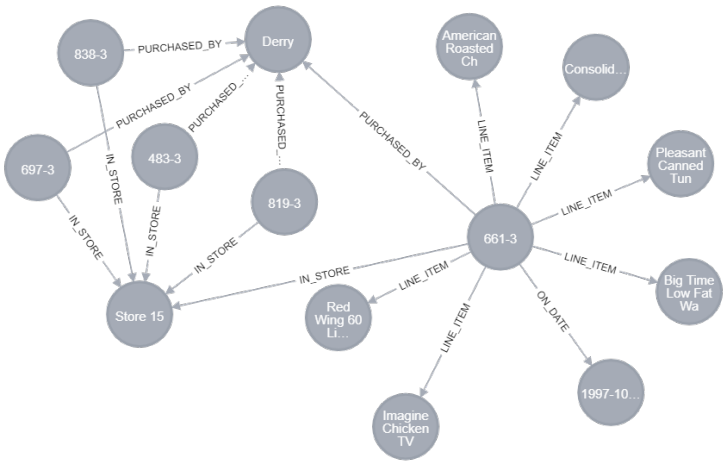


Figure 3.1: Example of nodes relationship

Table 3.1: Node Labels and Their Properties

Node Labels	Minimum Properties
[ProductFamily]	[name]
[ProductCategory]	[name]
[Department]	[name]
[Brand]	[name]
[Sale]	[id]
[StoreType]	[name]
[Year]	[year]
[ProductSubCategory]	[id, name]
[Month]	[id, month]
[City]	[name, country, state]
[Date]	[id, date, day, day_of_month, day_of_week]
[Promotion]	[id, name, media_type, cost, start_date, end_date]
[Region]	[id, sales_city, sales_state_province, sales_district, sales_region, sales_country]
[Product]	[id, name, SKU, SRP, gross_weight, net_weight, recyclable_package, low_fat, units_per_case, cases_per_pallet, shelf_width, shelf_height, shelf_depth]
[Store]	[id, name, city, postal_code, country, state, number, street_address, coffee_bar, video_store, salad_bar, prepared_food, florist]
[Customer]	[id, account_num, lname, fname, address1, city, state_province, postal_code, country, phone1, phone2, birthdate, marital_status, yearly_income, gender, total_children, num_children_at_home, education, date_accnt_opened, member_card, occupation, houseowner, num_cars_owned, fullname]

3.2. Career Changes dataset

A dataset with several details about different people's career, like their level of instruction, current occupation, previous experiences and other demographic factors.
Each person is represented by a single json document with the following structure (after the data wrangling process):

```
{
     // field : tipe
    _id: <ObjectId>,           //unique for every document
    Field of Study: String,
    Current Occupation: String,
    Age: Int,
    Gender: String,
    Years of Experience; Int,
    Education Level: String,  //highest level of education
    Industry Growth Rate: String,  //"High", "Medium", "Low"
    Job Satisfaction: Int,  //A rating from 1 to 10
    Work-Life Balance: Int,  //A rating from 1 to 10
    Job Opportunities: Int,  //number of available job
                                    //opportunities in the field
    Salary: Int,
    Job Security: Int,      //A rating from 1 to 10
    Career Change Interest: Int, //1 for yes, 0 for no
    Skills Gap: Int,  //A measure of how well the individual's
                     //current skills match their job requirements
    Family Influence: String, //degree of influence the family
                                 //has on their career choice
    Mentorship Available: Int, //1 for yes, 0 for no
    Certifications: Int,       //1 for yes, 0 for no
    Freelancing Experience: Int, //1 for yes, 0 for no
    Geographic Mobility: Int,    //1 for yes, 0 for no
    Professional Networks: Int, //how strong the individual's
                                   //professional network is (1-10)
    Career Change Events: Int,  //number of career changes
                                      //made in the past
    Technology Adoption: Int, //A measure of thecomfort level
                             //with adopting new technologies (1-10)
    Likely to Change Occupation: Int //1 for yes, 0 for no
}
```

4 | Queries

4.1. Neo4j Queries

Query.1 - Selling Hubs

Retrieve the cities with most sold units from their stores.
Useful to track where to deliver most of the items and detect the potential position of warehouses

```
MATCH (store:Store) - [:IN_CITY] - (citta:City)
WITH citta
MATCH (citta:City) - [ : IN_CITY] - (:Store) - [:IN_STORE] - (s:Sale) -[
    purchase:LINE_ITEM] - (:Product)
WITH citta, s , SUM(toInteger(purchase.quantity)) AS total_units
WITH citta , SUM(total_units) AS unit_per_city
RETURN citta,unit_per_city
ORDER BY unit_per_city DESC
```

City	Units per City
(:City {country: Mexico, name: Hidalgo, state: Zacatecas})	108,356
(:City {country: USA, name: Salem, state: OR})	80,762
(:City {country: USA, name: Tacoma, state: WA})	74,339
(:City {country: USA, name: Seattle, state: WA})	54,865
(:City {country: USA, name: Portland, state: OR})	54,521
(:City {country: USA, name: Los Angeles, state: CA})	52,630
(:City {country: USA, name: Spokane, state: WA})	52,477
(:City {country: USA, name: San Diego, state: CA})	52,413
...	...

Table 4.1: Neo4j query 1

Query.2 - Unforgettable day

Retrieve the most profitable day for each store

```
MATCH (store:Store) - [:IN_STORE] - (s:Sale) -[purchase:LINE_ITEM] - (:Product)
WITH store, s , SUM(toFloat(purchase.price - purchase.cost_price)/10000 * purchase.quantity) AS total_sale_profit
WITH store, s, round(total_sale_profit * 100) / 100 AS rounded_total_sale_profit
MATCH (s:Sale) - [:ON_DATE] - (day:Date)
WITH store , day , SUM(rounded_total_sale_profit) AS total_profit_day
WITH store , day , round(total_profit_day * 100) / 100 AS rounded_total_profit_day
ORDER BY  rounded_total_profit_day DESC
WITH store, collect({day: day, total: rounded_total_profit_day})[0] AS top_day_profit
RETURN store.name, top_day_profit.day.date AS day, top_day_profit.total AS max_total_per_day
ORDER BY store ASC
```

Store Name	Date	Max Total Per Day
Store 1	1998-12-20	3,336.88
Store 2	1998-12-23	196.65
Store 3	1998-11-07	3,164.02
Store 4	1998-01-02	2,845.41
Store 5	1998-12-14	160.41
Store 6	1998-11-05	2,775.15
Store 7	1998-11-05	3,859.08
Store 8	1998-12-03	4,254.03
Store 9	1998-06-03	1,404.08
Store 10	1998-08-29	2,880.64
Store 11	1998-12-23	3,285.34
Store 12	1998-09-16	4,383.52
Store 13	1997-07-27	9,282.36
Store 14	1998-12-29	181.26
.....	.....	.....

Table 4.2: Neo4j query 2

Query.3 - What's popular in Acapulco?

*Retrieve the 10 most popular couple of sold item in a sale at **Acapulco's** stores, useful in order of starting new promotions*

```
MATCH (store:Store{city:"Acapulco"}) - [:IN_STORE] - (s:Sale) , (p1:
    Product) - [:LINE_ITEM] - (s:Sale) - [:LINE_ITEM] - (p2:Product)
WITH p1, p2
WHERE id(p1) < id(p2)
WITH  p1, p2, COUNT(*) AS pair_count
RETURN p1, p2, pair_count
ORDER BY pair_count DESC
LIMIT 10
```

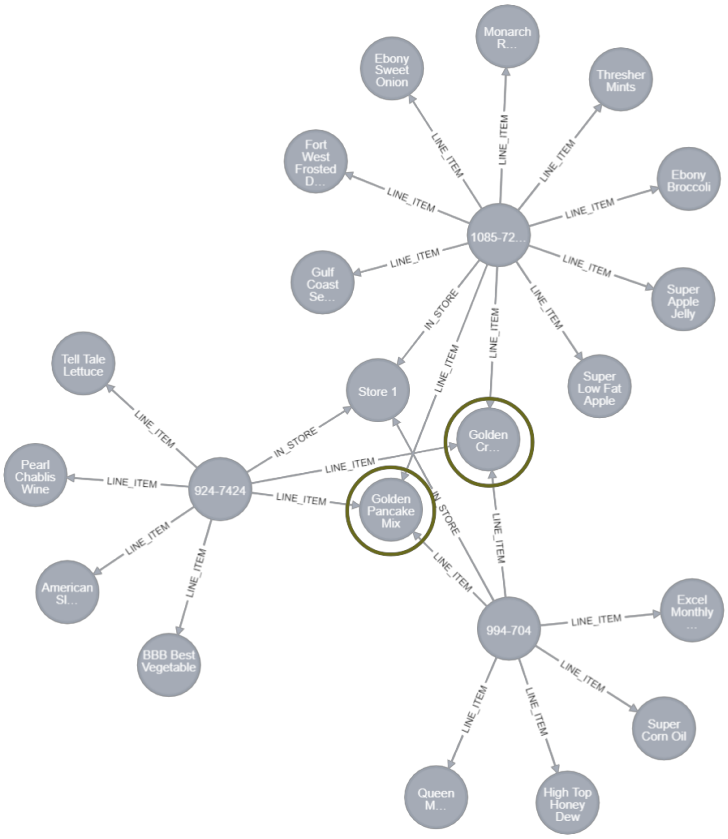


Figure 4.1: Example of sold items

Query.4 - Have you drank enough today?

*Trend of **Beers** sold over time at **USA's** stores, useful to spot trendings and eventually to manage the pricing*

```
MATCH ( store : Store { country : "USA" }) - [: IN_STORE ] - ( s : Sale
    ) - [sell:LINE_ITEM] -(beer:Product) - [:IN_CATEGORY] - (beer_cat:
    ProductSubCategory{name:"Beer"}),
(s:Sale) - [:ON_DATE] - (day:Date)
WITH beer , day ,SUM(sell.quantity) as sold_beers
WITH day , SUM(sold_beers) as total_sold_beers
RETURN day.date AS date , day.day_of_week , total_sold_beers
ORDER BY day.day ASC
```

Date	Day of Week	Total Sold Beers
1997-01-02	Thursday	2
1997-01-03	Friday	3
1997-01-05	Sunday	8
1997-01-06	Monday	3
1997-01-07	Tuesday	10
1997-01-09	Thursday	4
1997-01-10	Friday	2
1997-01-11	Saturday	12
1997-01-13	Monday	9
.....	.....	.....

Table 4.3: Neo4j query 4

Query.5 - Best clients

For each year the top 3 customers who spent more.
Fidelity is the most valuable trait of our clients, so let's find who reward

```
MATCH (customer:Customer) - [:PURCHASED_BY] - (s:Sale) - [sell:LINE_ITEM
    ] - (p:Product)
WITH customer, s, SUM(toFloat(sell.price ) / 10000 * sell.quantity) AS
    money_spent
WITH customer, s, round(money_spent * 100) / 100 AS rounded_money_spent
MATCH (s) - [:ON_DATE] - (day:Date)
WITH customer, substring(day.date, 0, 4) AS year, SUM(toFloat(
    rounded_money_spent)) AS Y_spent
WITH customer,  year, round(Y_spent *100) /100 AS Y_spent_rounded
ORDER BY Y_spent_rounded DESC
WITH year, COLLECT({customer: customer.fullname, money_spent:
    Y_spent_rounded})[0..3] AS customers_per_year
RETURN year , customers_per_year
```

Year	Customers Per Year
1998	{money_spent: 6757.93, customer: Scott Littleford}, {money_spent: 6503.93, customer: Isaiah Heymsfield}, {money_spent: 6259.62, customer: Craig Blackwell}
1997	{money_spent: 3954.59, customer: Mary Francis Benigar}, {money_spent: 3882.0, customer: Wildon Cameron}, {money_spent: 3540.47, customer: Ida Rodriguez}

Query.6 - Top Stores

Find the top 5 stores that earned more

```
MATCH (s:Store)<-[:IN_STORE]-(sa:Sale)-[l:LINE_ITEM]->(:Product)
WITH s, sum(l.price*l.quantity)/10000 AS sales
ORDER BY sales DESC
RETURN s.name, sales LIMIT 5
```

Store Name	Sales
Store 13	565,242
Store 17	527,800
Store 15	387,822
Store 11	387,257
Store 24	375,108

Query.7 - Cutting Time

Find, for each store, which department made less money, that's usefull in order to design new promotion for that store

```
MATCH (st:Store)<-[:IN_STORE]-(sa:Sale)-[l:LINE_ITEM]->(pr:Product)-[:
    IN_CATEGORY]->(sc:ProductSubCategory)-[:IN_DEPARTMENT]->(dep:
    Department)
WITH st, dep, sum(l.price*l.quantity)/10000 AS sales
ORDER BY st.name ASC, sales ASC
WITH st, collect([dep, sales]) as dep_sales
RETURN st.name, dep_sales[0][0].name AS department, dep_sales[0][1] AS
    profit
```

Store Name	Department	Profit
Store 1	Carousel	558
Store 2	Carousel	37
Store 3	Carousel	967
Store 4	Carousel	538
Store 5	Canned Products	18
Store 6	Carousel	935
Store 7	Carousel	1198
Store 8	Carousel	761
Store 9	Carousel	122
Store 10	Carousel	567
Store 11	Carousel	1336
Store 12	Carousel	647
Store 13	Carousel	1422
Store 14	Checkout	41
Store 15	Carousel	1063
Store 16	Carousel	1127
Store 17	Carousel	1609
Store 18	Carousel	173
Store 19	Carousel	679
Store 20	Carousel	157
Store 21	Carousel	876
Store 22	Carousel	52
Store 23	Carousel	266
Store 24	Carousel	823

Table 4.4: Neo4j query 7

Query.8 - Promotions Effectiveness

Find the number of purchase made with and without using a promotion, to analyse their effectiveness:

```
MATCH (s:Sale)-[:APPLIED_PROMOTION]->(pr:Promotion)
WITH collect(s) AS cs, COUNT(DISTINCT s) AS with_promo
MATCH (ns:Sale)
WHERE NOT ns IN cs
WITH with_promo, COUNT(DISTINCT ns) AS no_promo
RETURN with_promo, no_promo
```

With promo	Without promo
15060	43248

Query.9 - Best Customers

For every member card type, find the customer who spent more money, in order to reward them in future context:

```
MATCH (c:Customer)<-[:PURCHASED_BY]-(s:Sale)-[l:LINE_ITEM]->(p:Product)
WITH c.fullname AS name, c.member_card AS card, sum(l.price*l.quantity)
   /10000 AS rank
ORDER BY card, rank DESC
WITH card, collect([rank, name]) AS ranking
RETURN card, ranking[0][1] AS winner, ranking[0][0] AS rank
```

Card	Winner	Rank
Bronze	Ida Rodriguez	8337
Gold	James Horvat	7702
Normal	Kristin MIller	6247
Silver	Wildon Cameron	7818

Query.10 - How Many Customers

Find how many customers each type of store has

```
MATCH (c:Customer)<-[:PURCHASED_BY]-(sa:Sale)-[:IN_STORE]->(s:Store)-[:
    OF_TYPE]->(st:StoreType)
WITH st.name AS name, COUNT(DISTINCT c) AS customers
ORDER BY customers
RETURN name, customers
```

Name	Customers
Small Grocery	977
Mid-Size Grocery	1064
Gourmet Supermarket	1535
Deluxe Supermarket	2283
Supermarket	4767

4.2. MongoDB Queries

Query.1 - Satisfaction

Compute the Satisfaction index (average of each Job Satisfaction) for each job, then rank them

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: "$Current Occupation",
      Satisfaction: {
        $avg: "$Job Satisfaction"
      }
    }
  },
  {
    $match: {
      _id: {
        $ne: null
      }
    }
  },
  {
    $project: {
      Occupation: "$_id",
      _id: 0,
      Satisfaction: 1
    }
  },
  {
    $sort: {
      Satisfaction: -1
    }
  }
])
```

Output


```
[{
  "Satisfaction": 5.558755464129597,
  "Occupation": "Psychologist"
},
{
  "Satisfaction": 5.556703762586116,
  "Occupation": "Biologist"
},
{
  "Satisfaction": 5.486328643482878,
  "Occupation": "Economist"
},
{
  "Satisfaction": 5.486003110419906,
  "Occupation": "Business Analyst"
},
{
  "Satisfaction": 5.483205501190161,
  "Occupation": "Lawyer"
},
{
  "Satisfaction": 5.473792394655704,
  "Occupation": "Software Developer"
},
{
  "Satisfaction": 5.4702396289616075,
  "Occupation": "Artist"
},
{
  "Satisfaction": 5.465002573340196,
  "Occupation": "Teacher"
},
{
  "Satisfaction": 5.46023166023166,
  "Occupation": "Doctor"
},
{
  "Satisfaction": 5.452835118892082,
  "Occupation": "Mechanical Engineer"
}]
```

Query.2 - sameDegreeOccupation

Find how many people are currently working in the same field they have a degree in:

```
SMBUD_project.Carrer.aggregate([
  {
    $match: {
      $or: [
        {$and: [
            { "Field of Study": "Psychology" },
            { "Current Occupation": "Psychologist" }
          ]
        },
        {$and: [
            { "Field of Study": "Medicine" },
            { "Current Occupation": "Doctor" }
          ]
        },
        {$and: [
            { "Field of Study": "Education" },
            { "Current Occupation": "Teacher" }
          ]
        },
        {$and: [
            { "Field of Study": "Arts" },
            { "Current Occupation": "Artist" }
          ]
        },
        {$and: [
            { "Field of Study": "Computer Science" },
            { "Current Occupation": "Software Developer" }
          ]
        },
        {$and: [
            { "Field of Study": "Business" },
            { "Current Occupation": "Business Analyst" }
          ]
        },
        {$and: [
            { "Field of Study": "Mechanical Engineering" },
            { "Current Occupation": "Mechanical Engineer" }
          ]
        },


```

```
        {$and: [
            { "Field of Study": "Biology" },
            { "Current Occupation": "Biologist" }
          ]
        },
        {$and: [
            { "Field of Study": "Law" },
            { "Current Occupation": "Lawyer" }
          ]
        },
        {$and: [
            { "Field of Study": "Economics" },
            { "Current Occupation": "Economist" }
          ]
        }
      ]
    }
  },
  {$group: {
      _id: {
        Field: "$Field of Study",
        Occupation: "$Current Occupation"
      },
      tot: { $sum: 1 }
    }
  },
  {$project: {
      Field: "$_id.Field",
      Occupation: "$_id.Occupation",
      tot: 1,
      _id: 0
    }
  }
]);
```

Output - Partial


```
[{
  "tot": 379,
  "Field": "Law",
  "Occupation": "Lawyer"
},
{
  "tot": 389,
  "Field": "Economics",
  "Occupation": "Economist"
},
{
  "tot": 395,
  "Field": "Psychology",
  "Occupation": "Psychologist"
},
{
  "tot": 415,
  "Field": "Computer Science",
  "Occupation": "Software Developer"
},
{
  "tot": 354,
  "Field": "Business",
  "Occupation": "Business Analyst"
},
{
  "tot": 352,
  "Field": "Arts",
  "Occupation": "Artist"
},
{
  "tot": 397,
  "Field": "Biology",
  "Occupation": "Biologist"
},
{
  "tot": 405,
  "Field": "Medicine",
  "Occupation": "Doctor"
},
...
}]
```

Query.3 - changeOccupation

For each job, find how many people would like to change carrer path and how many would not:

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: "$Current Occupation",
      change: {
        $sum: {
          $cond: [
            {$eq: [
                "$Likely to Change Occupation",1
              ]},
            1,0
          ]}},
      not_change: {
        $sum: {
          $cond: [
            {$eq: [
                "$Likely to Change Occupation",0
              ]},
            1,0
            ]}}
        }},
  {
    $project:
      {
        Occupation: "$_id",
        _id: 0,
        change: 1,
        not_change: 1
      }
  }
])
```

Output - Partial


```
[{
  "change": 2161,
  "not_change": 1606,
  "Occupation": "Economist"
},
{
  "change": 2275,
  "not_change": 1610,
  "Occupation": "Doctor"
},
{
  "change": 0,
  "not_change": 0,
  "Occupation": null
},
{
  "change": 2213,
  "not_change": 1676,
  "Occupation": "Psychologist"
},
{
  "change": 2212,
  "not_change": 1680,
  "Occupation": "Software Developer"
},
{
  "change": 2232,
  "not_change": 1654,
  "Occupation": "Teacher"
},
{
  "change": 2209,
  "not_change": 1618,
  "Occupation": "Mechanical Engineer"
},
{
  "change": 2206,
  "not_change": 1568,
  "Occupation": "Biologist"
},
...
]
```

Query.4 - Education vs. Salary Analysis

*Find which education level have a better **average** salary*

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: "$Education Level",
      averageSalary: {
        $avg: "$Salary"
      }
    }
  },
  {
    $match: {
      _id: {
        $ne: null
      }
    }
  },
  {
    $project: {
      educationLevel: "$_id",
      _id: 0,
      averageSalary: 1
    }
  },
  {
    $sort: {
      averageSalary: -1
    }
  }
])
```

Output


```
[{
  "averageSalary": 115348.93935604877,
  "educationLevel": "Master's"
},
{
  "averageSalary": 115320.96336686077,
  "educationLevel": "Bachelor's"
},
{
  "averageSalary": 114665.00756878388,
  "educationLevel": "PhD"
},
{
  "averageSalary": 114579.17316287289,
  "educationLevel": "High School"
}]
```

Query.5 - Gender Distribution Analysis

This query calculates the percentage distribution of genders within each field of study, providing insights into gender representation across disciplines.

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: {
        fieldOfStudy: "$Field of Study",
        gender: "$Gender"
      },
      count: {$sum: 1}
    }
  },
  {
    $group: {
      _id: "$_id.fieldOfStudy",
      total: {$sum: "$count"},
      genders: {$push: {gender: "$_id.gender",count: "$count"}}
    }
  },
  {
    $match: {_id: {$ne: null}}
  },
  {
    $project: {
      fieldOfStudy: "$_id",
      _id: 0,
      genderPercentages: {
        $map: {
          input: "$genders",
          as: "gender",
          in: {
            gender: "$$gender.gender",
            percentage: {
              $multiply: [
                {$divide: ["$$gender.count","$total"]},
                100
              ]}
          }
        }
      }
    }
  }])
```

Output - Partial


```
[{
  "fieldOfStudy": "Biology",
  "genderPercentages": [
    {
      "gender": "Female",
      "percentage": 50.14027033919919
    },
    {
      "gender": "Male",
      "percentage": 49.85972966080082
    }
  ]
},
{
  "fieldOfStudy": "Medicine",
  "genderPercentages": [
    {
      "gender": "Female",
      "percentage": 50.797670296277545
    },
    {
      "gender": "Male",
      "percentage": 49.20232970372246
    }
  ]
},
{
  "fieldOfStudy": "Mechanical Engineering",
  "genderPercentages": [
    {
      "gender": "Female",
      "percentage": 49.40629839958699
    },
    {
      "gender": "Male",
      "percentage": 50.59370160041301
    }
  ]
},
...
]
```

Query.6 - Relation between experience and job opportunities

Analyse the relationship between years of experience and job opportunities, ranking them by highest average opportunity

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: {
        yearsOfExperience: "$Years of Experience"
      },
      avgJobOpportunities: {$avg: "$Job Opportunities"}
    }
  },
  {
    $project: {
      yearsOfExperience: "$_id.yearsOfExperience",
      avgJobOpportunities: 1,
      _id: 0
    }
  },
  {
    $sort: {avgJobOpportunities: -1}
  }
])
```

Output - Partial

```
[{
  "avgJobOpportunities": 52.05675954592363,
  "yearsOfExperience": 20
},
{
  "avgJobOpportunities": 51.99171842650104,
  "yearsOfExperience": 33
},
{
  "avgJobOpportunities": 51.48221757322176,
  "yearsOfExperience": 32
},
...
]
```

Query.7 - Total Certifications Analysis

Calculates the total number of certifications within each field of study and education level

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: {
        fieldOfStudy: "$Field of Study",
        educationLevel: "$Education Level"
      },
      totalCertifications: {$sum: "$Certifications"}
    }
  },
  {
    $project: {
      fieldOfStudy: "$_id.fieldOfStudy",
      educationLevel: "$_id.educationLevel",
      totalCertifications: 1,
      _id: 0
    }
  },
  {$sort: {totalCertifications: -1}}
])
```

Output - Partial

```
[{
  "totalCertifications": 318,
  "fieldOfStudy": "Arts",
  "educationLevel": "PhD"
},
{
  "totalCertifications": 314,
  "fieldOfStudy": "Medicine",
  "educationLevel": "High School"
},
{
  "totalCertifications": 310,
  "fieldOfStudy": "Biology",
  "educationLevel": "High School"
},
...
]
```

Query.8 - Skill gap Analysis

This query calculates the mean skill gap by the employee field of study, for each field of study, it also collects the cases where the skill gap is too high (greater or equal 5)

```
SMBUD_project.Carrer.aggregate([
  {
    $group: {
      _id: "$Field of Study",
      averageSkillGap: { $avg: "$Skills Gap" },
      documents: {
        $push: {
          docId : "$_id",
          skillsGap: "$Skills Gap",
          currentOccupation: "$Current Occupation",
          age: "$Age",
          salary: "$Salary"
        }
      }
    }
  },
  {
    $project: {
      fieldOfStudy: "$_id",
      _id: 0,
      averageSkillGap: 1,
      documents: {
        $filter: {
          input: "$documents",
          as: "doc",
          cond: { $gte: ["$$doc.skillsGap", 5] }
        }
      },
      howMany: { // same as before, just counting
        $size: {
          $filter: {
            input: "$documents",
            as: "doc",
            cond: { $gte: ["$$doc.skillsGap", 5] }
          }
        }
      }
    }
  }
])
```

Output - Partial


```
[{
  "averageSkillGap": 5.4410492157923205,
  "fieldOfStudy": "Psychology",
  "documents": [
    {
      "docId": {
        "$oid": "6750823f8a036fa9e4893c1d"
      },
      "skillsGap": 9,
      "currentOccupation": "Psychologist",
      "age": 34,
      "salary": 80448
    },
    ...
    ],
    "howMany": 2175
},
{
    "averageSkillGap": 5.537269569863746,
    "fieldOfStudy": "Economics",
    "documents": [
    {
      "docId": {
        "$oid": "6750823e8a036fa9e4893c09"
      },
      "skillsGap": 10,
      "currentOccupation": "Business Analyst",
      "age": 49,
      "salary": 116672
    },
    ...
    ],
    "howMany": 2294
},
...
]
```

Query.9 - Gender distribution Analysis pt.2

For each pair of positions and education level, it calculates the distribution of men and women

```
SMBUD_project.Carrer.aggregate([
  // Stage 1 group occupation,gender,educationLevel
 {
    $group: {
      _id: {
        occupation: "$Current Occupation",
        gender: "$Gender",
        educationLevel: "$Education Level"
      },
      count: { $sum: 1 }
    }
  },
  // Stage 2: Explode and count female / males
  {
    $group: {
      _id: {
        occupation: "$_id.occupation",
        educationLevel: "$_id.educationLevel"
      },
      countsByGender: {
        $push: {
          gender: "$_id.gender",
          count: "$count"
        }
      }
    }
  },
  // Stage 3: Pivot countsByGender into separate fields for men and
     women
  {
    $project: {
      _id: 0, // Exclude default _id
      occupation: "$_id.occupation",
      educationLevel: "$_id.educationLevel",
      menCount: {
        $reduce: {
          input: "$countsByGender",
          initialValue: 0,
          in: {
            $cond: [
```

```
            { $eq: ["$$this.gender", "Male"] }, // Check if gender is
               Male
            { $add: ["$$value", "$$this.count"] },
            "$$value" // Keep the value as-is if not Male
          ]
        }
      }
    },
    womenCount: {
      $reduce: {
        input: "$countsByGender",
        initialValue: 0,
        in: {
          $cond: [
            { $eq: ["$$this.gender", "Female"] }, // Check if gender
               is Female
            { $add: ["$$value", "$$this.count"] },
            "$$value" // Keep the value as-is if not Female
          ]
        }
      }
    }
  },
   {$sort: { occupation: 1 }}
 ])
```

Output - Partial


```
[{
  "occupation": null,
  "educationLevel": null,
  "menCount": 0,
  "womenCount": 0
},
{
  "occupation": "Artist",
  "educationLevel": "PhD",
  "menCount": 476,
  "womenCount": 492
},
...
]
```

Query.10 - Carrer Salary trend

it calculates the salary trend for each position by number of experience

```
SMBUD_project.Carrer.aggregate([

  // Stage 1: Group by occupation and years of experience
  {
    $group: {
      _id: {
        occupation: "$Current Occupation",    // Group by occupation
        yearsOfExperience: "$Years of Experience" // Group by years of
           experience
      },
      avgSalary: { $avg: "$Salary" }
    }
  },
  // Stage 2: Sort by occupation and years of experience
  {
    $sort: {
      "_id.occupation": 1,  // Sort by occupation alphabetically
      "_id.yearsOfExperience": 1  // Sort by years of experience in
         ascending order
    }
  },
  // Stage 3: Group by occupation and create an array of average
     salaries (progression)
  {
    $group: {
      _id: "$_id.occupation", // Group by occupation
      progression: {
        $push: "$avgSalary"
      }
    }
  },
  // Stage 4: Project the final result to match the desired format
  {
    $project: {
      _id: 0, // Exclude the default _id field
      occupation: "$_id", // Include the occupation field
      progression: "$progression" // Include the progression array
    }
  }
])
```

Output - Partial

```
[{
  "occupation": "Doctor",
   "progression": [...]
   },
   {
       "occupation": "Lawyer",
       "progression": [...]
   },
{
  "occupation": "Economist",
  "progression": [
    111829.96551724138,
    111638.35643564357,
    119633.24509803922,
    115185.45192307692,
    113446.07142857143,
    123441.76923076923,
    110961.87912087912,
    115924.43564356436,
    ...
    ]
},
...
]
```

5 | Application

As extra work, we designed and developed an Android application. This application can be run both on physical Android devices and using the online simulator *Appetize.com*, ensuring greater flexibility of use.

Application Description

The application was developed to display queries and their respective results in a clear and interactive manner. The interface is simple and intuitive, allowing the user to enter a query and view the results directly on the app. Moreover, the application is compatible with various versions of Android, ensuring broad accessibility.

Main Features

- Executing queries via a clear and responsive user interface.
- Immediate display of query results.
- Compatibility with physical Android devices and the *https://appetize.io/* simulator.
- Responsive layout, optimized for different screen sizes.

Application Screenshots

The following figures show some screenshots of the application.

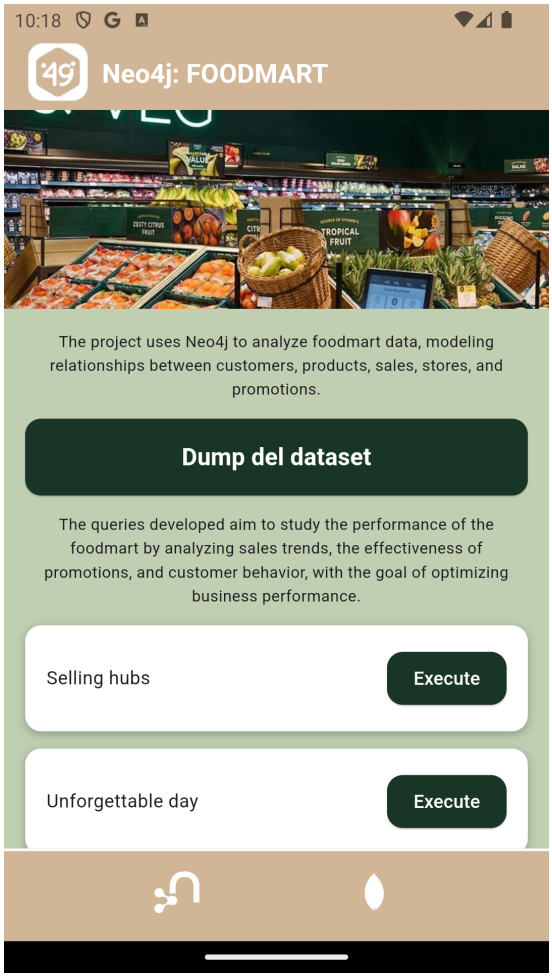
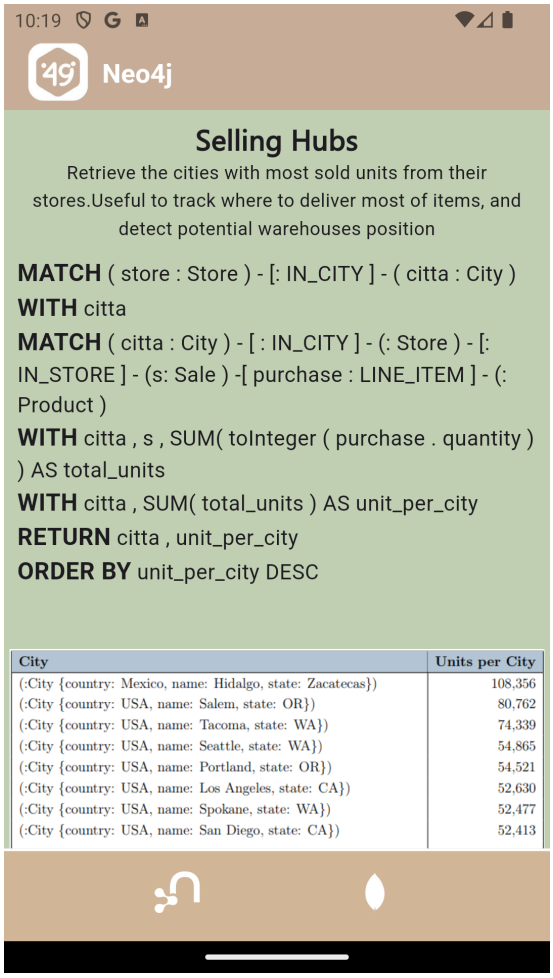


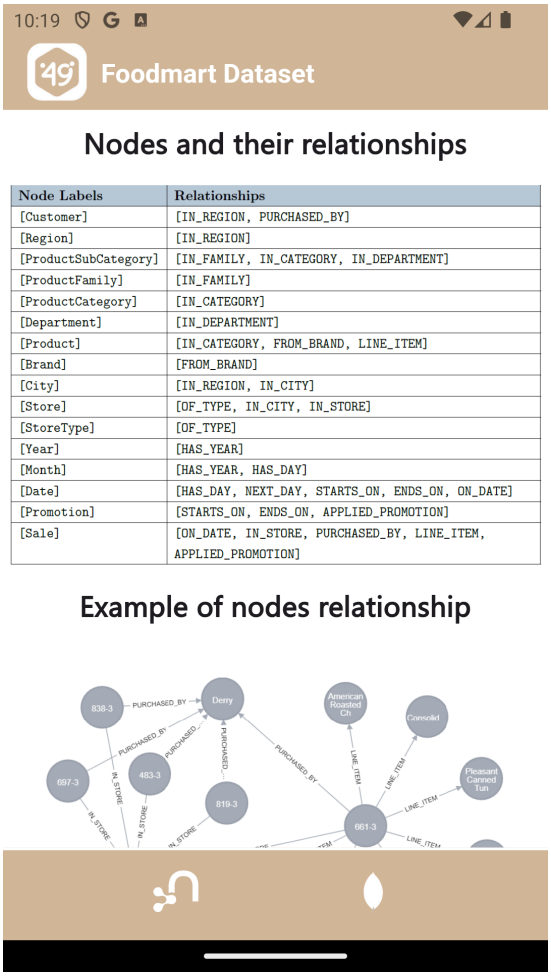
Figure 5.1: Neo4j app section

Figure 5.2: MongoDB app section



City	Units per City
(:City {country: Mexico, name: Hidalgo, state: Zacatecas})	108,356
(:City {country: USA, name: Salem, state: OR})	80,762
(:City {country: USA, name: Tacoma, state: WA})	74,339
(:City {country: USA, name: Seattle, state: WA})	54,865
(:City {country: USA, name: Portland, state: OR})	54,521
(:City {country: USA, name: Los Angeles, state: CA})	52,630
(:City {country: USA, name: Spokane, state: WA})	52,477
(:City {country: USA, name: San Diego, state: CA})	52,413

Figure 5.3: Example view of a query



Node Labels	Relationships
[Customer]	[IN_REGION, PURCHASED_BY]
[Region]	[IN_REGION]
[ProductSubCategory]	[IN_FAMILY, IN_CATEGORY, IN_DEPARTMENT]
[ProductFamily]	[IN_FAMILY]
[ProductCategory]	[IN_CATEGORY]
[Department]	[IN_DEPARTMENT]
[Product]	[IN_CATEGORY, FROM_BRAND, LINE_ITEM]
[Brand]	[FROM_BRAND]
[City]	[IN_REGION, IN_CITY]
[Store]	[OF_TYPE, IN_CITY, IN_STORE]
[StoreType]	[OF_TYPE]
[Year]	[HAS_YEAR]
[Month]	[HAS_YEAR, HAS_DAY]
[Date]	[HAS_DAY, NEXT_DAY, STARTS_ON, ENDS_ON, ON_DATE]
[Promotion]	[STARTS_ON, ENDS_ON, APPLIED_PROMOTION]
[Sale]	[ON_DATE, IN_STORE, PURCHASED_BY, LINE_ITEM, APPLIED_PROMOTION]

Figure 5.4: Example of a dataset dump view

List of Figures

List of Tables