SOEN 363 - Phase-2

Team Members

- Anthony Chraim 40091014
- Ethan Benabou 40032543
- Justin Loh 40073776
- Mohamed Amine Kihal 40046046

Q3 - BigData with SQL

(a) - Dataset: Grocery Sales

You can find the data on Kaggle, at the following address:
https://www.kaggle.com/codemysteries/salesdb

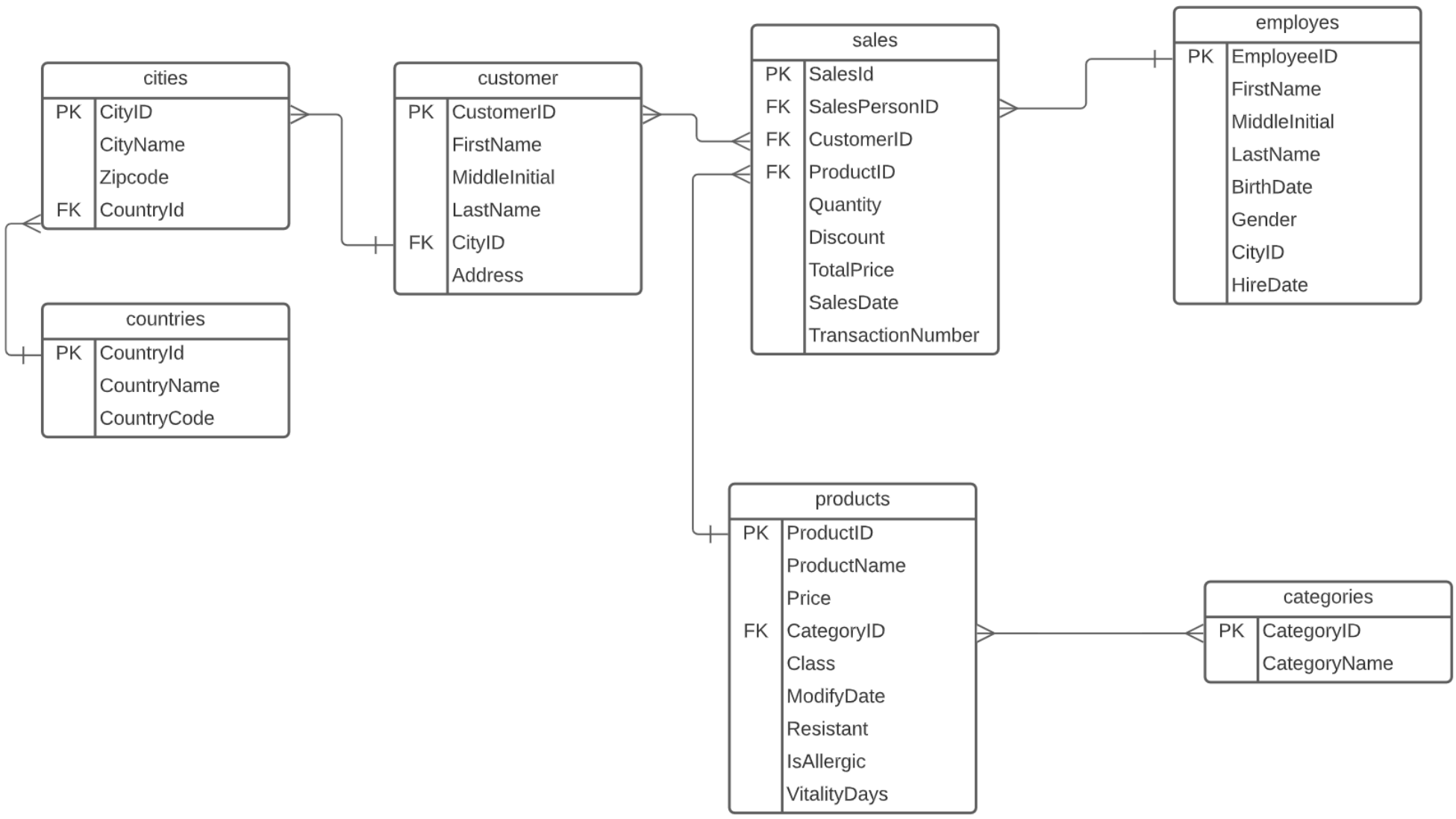
The data set that was chosen is a reasonably sized sales dataset for groceries sold in various cities throughout the entire United States of America. There are 7 files in total in the dataset:

- `categories.csv`
- `cities.csv`
- `countries.csv`
- `customers.csv`
- `employes.csv`
- `products.csv`
- `sales.csv`
- ***TOTAL SIZE:*** `753.22MB`

(b) - Creating the database: PostgreSQL

DATABASE: `PostgreSQL v13.2`

ENTITY RELATION DIAGRAM (ERD):



DATA DEFINITION (DDL):

```
-- Countries
CREATE TABLE Countries (
    CountryID INTEGER NOT NULL,
    CountryName VARCHAR(50),
    CountryCode VARCHAR(2),
    PRIMARY KEY (CountryID)
);

-- Cities
CREATE TABLE Cities (
    CityID INTEGER NOT NULL,
    CityName VARCHAR(50),
    Zipcode INTEGER,
    CountryID INTEGER NOT NULL,
    PRIMARY KEY (CityID),
    FOREIGN KEY (CountryID) REFERENCES countries(CountryID)
);

-- Customers
```

```
CREATE TABLE Customers (
    CustomerID INTEGER NOT NULL,
    FirstName VARCHAR(50),
    MiddleInitial VARCHAR(1),
    LastName VARCHAR(50),
    CityID INTEGER NOT NULL,
    Address VARCHAR(255),
    PRIMARY KEY (CustomerID),
    FOREIGN KEY (CityID) REFERENCES cities(CityID)
);

-- Categories
CREATE TABLE Categories (
    CategoryID INTEGER NOT NULL,
    CategoryName VARCHAR(50),
    PRIMARY KEY (CategoryID)
);

-- Products
CREATE TABLE Products (
    ProductID INTEGER NOT NULL,
    ProductName VARCHAR(255),
    Price REAL,
    CategoryID INTEGER NOT NULL,
    Class VARCHAR(50),
    ModifyDate TIMESTAMP,
    Resistant VARCHAR(20),
    IsAllergic VARCHAR(10),
    VitalityDays INTEGER,
    PRIMARY KEY (ProductID),
    FOREIGN KEY (CategoryID) REFERENCES categories(CategoryID)
);

-- Employees
CREATE TABLE Employes (
    EmployeeID INTEGER NOT NULL,
    FirstName VARCHAR(50),
    MiddleInitial VARCHAR(1),
    LastName VARCHAR(50),
    BirthDate TIMESTAMP,
    Gender VARCHAR(1),
    CityID INTEGER NOT NULL,
    HireDate TIMESTAMP,
    Address VARCHAR(255),
    PRIMARY KEY (EmployeeID),
    FOREIGN KEY (CityID) REFERENCES cities(CityID)
);

-- Sales
CREATE TABLE Sales (
    SalesID INTEGER NOT NULL,
    SalesPersonID INTEGER NOT NULL,
```

```
    CustomerID INTEGER NOT NULL,
    ProductID INTEGER NOT NULL,
    Quantity INTEGER,
    Discount REAL,
    TotalPrice REAL,
    SalesDate TIMESTAMP,
    TransactionNumber VARCHAR(20),
    PRIMARY KEY (SalesID),
    FOREIGN KEY (SalesPersonID) REFERENCES employes(EmployeeID),
    FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES products(ProductID)
);
```

(c) - Loading the data

Docker Image: https://hub.docker.com/repository/docker/obonobo/soen363-phase-2-postgres

For reproducability, a Docker image was created containing PostgreSQL v13.2. All the data has been preloaded into the Docker image. You can pull the image and examine the database by executing the following in your terminal (requires `docker` to be installed):

```
# Pull the latest docker image
docker pull obonobo/soen363-phase-2-postgres:latest

# Spin up an instance of postgres
docker run -d \
    -p 5432:5432 \
    -e PGDATA=/data \
    -e POSTGRES_DB=soen363 \
    -e POSTGRES_USER=admin \
    -e POSTGRES_PASSWORD=admin123 \
    obonobo/soen363-phase-2-postgres:latest
```

Alternatively, you can use the `docker-compose.yml` file packaged alongside this report. This will spin up 3 containers (PostgreSQL, PGAdmin, MongoDB): `docker-compose up -d`

The data required a few steps of processing before it was successfully loaded into Postgres. All the scripts used for preprocessing and data cleaning have been included alongside this report. Check out the `scripts/` dir for the source code.

(d) - Reports

10 SQL queries were created to extract interesting pieces of information about grocery sales in the US.

1. How much do discounts effect sales?

This query reports the average quantity of items sold, ordered by the discount (if any) that has been applied to the item. As expected, items with discounts, on average, tend to have higher sales.

```
SELECT discount, AVG(quantity) FROM Sales
WHERE discount IN (
    SELECT DISTINCT discount
    FROM Sales
    ORDER BY discount DESC)
OR discount IS NULL
GROUP BY discount;
```

2. Which employee has sold the most seafood?

```
SELECT
    COUNT(*) AS numsales,
    firstname,
    lastname,
    birthdate,
    cityname,
    zipcode,
    countryname
FROM Sales s
    INNER JOIN products p ON s.productid = p.productid
    INNER JOIN categories c ON p.categoryid = c.categoryid
    INNER JOIN employes e ON s.salespersonid = e.employeeid
    INNER JOIN cities ci ON e.cityid = ci.cityid
    INNER JOIN countries co ON ci.countryid = co.countryid
WHERE categoryname = 'Seafood'
GROUP BY firstname, lastname, birthdate, cityname, zipcode, countryname
ORDER BY numsales DESC
LIMIT 1;
```

3. What are the top-selling item categories in Tucson, Arizona?

```
SELECT ca.categoryname, COUNT(ca.categoryid)
FROM customers cu
    INNER JOIN sales s  ON cu.customerid = s.customerid
    INNER JOIN cities c ON cu.cityid = c.cityid
    INNER JOIN products p ON s.productid = p.productid
    INNER JOIN categories ca ON p.categoryid = ca.categoryid
WHERE c.cityname = 'Tucson'
GROUP BY ca.categoryid
ORDER BY COUNT(ca.categoryid) DESC;
```

4. How many sales of chocalate does each city purchase?

```
SELECT COUNT(s.salesid), ci.cityname
FROM sales s
INNER JOIN products p ON s.productid = p.productid
INNER JOIN customers c ON s.customerid = c.customerid
INNER JOIN cities ci on c.cityid = ci.cityid
WHERE p.productname like '%Chocolate%'
GROUP BY ci.cityname
ORDER BY COUNT(s.salesid) DESC;
```

5. Which orders in our system are expired?

```
SELECT
    s.salesid,
    p.productname,
    p.modifydate,
    s.salesdate,
    (DATE_PART('day', s.salesdate- p.modifydate):: integer) as daydiff,
    p.vitalitydays,
CASE
    WHEN p.vitalitydays > (DATE_PART('day', s.salesdate- p.modifydate):: integer)
    THEN 'Good'
    ELSE 'Expired'
    END AS FOODSTATUS
FROM products p
INNER JOIN sales s ON s.productid = p.productid
WHERE s.salesdate IS NOT NULL
AND p.vitalitydays IS NOT NULL
AND p.vitalitydays IS NOT NULL
AND (DATE_PART('day', s.salesdate- p.modifydate):: integer) >= 0;
```

6. How many customers does each city have?

```
SELECT COUNT(c.cityid), c.cityid, ci.cityname
FROM customers c
INNER JOIN cities ci ON c.cityid = ci.cityid
GROUP BY c.cityid, ci.cityname
ORDER BY COUNT(c.customerid) DESC;
```

7. Who are the top 5 people who purchased the most wine?

```
SELECT COUNT(s.productid), c.firstname ,c.middleinitial, c.lastname
FROM sales s
INNER JOIN products p ON s.productid = p.productid
INNER JOIN customers c ON s.customerid = c.customerid
WHERE p.productname like '%Wine%'
GROUP BY s.productid,c.firstname,c.middleinitial, c.lastname
ORDER BY COUNT(s.productid) DESC
LIMIT 5;
```

8. Which employees are from the millenial generation?

```
SELECT *
FROM employes e
WHERE e.birthdate
    BETWEEN '1981-01-01 00:00:00'
    AND '1996-12-31 23:59:59';
```

9. Which employees are from the Gen X generation?

```
SELECT *
FROM employes e
WHERE e.birthdate
    BETWEEN '1965-01-01 00:00:00'
    AND '1980-12-31 23:59:59';
```

10. Who are the employees hired since 2016?

```
SELECT *
FROM employes e
WHERE e.hiredate
    BETWEEN '2016-01-01 00:00:00'
    AND NOW();
```

(e)

Sales table is the enormous table of all. Its quantities are searched frequently for market analysis. Creating the sales quantity index will speeds up the processing time noticeably.

```
CREATE INDEX sales_idx ON sales(quantity);
```

Q4 - NoSQL Databases

(a) - Dataset: Grocery Sales

We used the same dataset as for the SQL database

You can find the data on Kaggle, at the following address:
https://www.kaggle.com/codemysteries/salesdb

The data set that was chosen is a reasonably sized sales dataset for groceries sold in various cities throughout the entire United States of America. There are 7 files in total in the dataset:

- `categories.csv`
- `cities.csv`
- `countries.csv`
- `customers.csv`
- `employes.csv`
- `products.csv`
- `sales.csv`
- ***TOTAL SIZE:*** `753.22MB`

(b) - Modeling data as JSON documents

Using the same CSV files as with PostgreSQL, each row of data was modeled as a MongoDB document - which is a JSON data structure of key-value pairs.

For example, a sales record that looks like:

```
1,6,27039,381,7,NULL,0.00,2018-02-05 07:38:25.430,FQL4S94E4ME1EZFTG42G
```

becomes:

```
{
  "SalesID": 1,
  "SalesPersonID": 6,
  "CustomerID": 27039,
  "ProductID": 381,
  "Quantity": 7,
  "Discount": null,
  "TotalPrice": 0,
  "SalesDate": "2018-02-05 07:38:25.430",
  "TransactionNumber": "FQL4S94E4ME1EZFTG42G"
}
```

Frankly, the data is much more suited to a tabular format like CSV or SQL tables; in a real application scenario, a CSV dataset would more suitably be imported into a relational database. There is not much to gain from using an unstructured format like JSON on structured and tabular data.

(c) - Creating a NoSQL database

Here is one area where NoSQL is more convenient, but at a cost. MongoDB will let you create a database and database collections without defining a schema for your records. In other words, a schema-less MongoDB collection will allow the database operator to insert records of any shape, with no complaints. In this case, no schema was defined for the collections used in the assignment. Another difference in the MongoDB design is that records must be inserted into a collection in order for the collection

(d)

Docker Image: https://hub.docker.com/repository/docker/obonobo/soen363-phase-2-mongo

For reproducability, a Docker image was created containing MongoDB. All the data has been preloaded into the Docker image. You can pull the image and examine the database by executing the following in your terminal (requires `docker` to be installed):

```
# Pull the latest docker image
docker pull obonobo/soen363-phase-2-mongo:latest

# Spin up an instance of postgres
docker run -d \
    -p 5432:5432 \
    -e PGDATA=/data \
    -e POSTGRES_DB=soen363 \
    -e POSTGRES_USER=admin \
    -e POSTGRES_PASSWORD=admin123 \
    obonobo/soen363-phase-2-mongo:latest
```

Alternatively, you can use the `docker-compose.yml` file packaged alongside this report. This will spin up 3 containers (PostgreSQL, PGAdmin, MongoDB): `docker-compose up -d`

The data required a few steps of processing before it was successfully loaded into Postgres. All the scripts used for preprocessing and data cleaning have been included alongside this report. Check out the `scripts/` dir for the source code.

(e)

1. How much do discounts effect sales?

```
db.sales.aggregate([
  {
    $group: {
      _id: "$Discount",
      avgAmount: { $avg: "$Quantity" },
    },
  },
]);
```

2. What were the sales like for the product with ID 47 in January 2018?

```
// A mongo query to return info for a product purchased in a certain
month, its quantity, and how much discount was applied to it.
// Useful for tracking requirements for loyalty rewards in the future
db.sales.find({
  $and: [
    { ProductID: 47 },
    { Quantity: { $gt: 20 } },
    { Discount: { $gte: 0.2 } },
    { SalesDate: { $regex: "^2018-01" } },
  ],
});
```

3. Who are all the male employees that have been hired after 2000 and are originally from baltimore?

```
var output = [];

db.cities
  .find({ CityName: "Baltimore" })
  .forEach((document) => {
    output.push(document.CountryID);
  });

db.employes.find({
  $and: [
    { HireDate: { $gte: "2000-01-13 00:00:00.000" } },
    { Gender: "M" },
    { CityID: { $in: output } },
  ],
});
```

4. What are the total number of items sold across the entire database?

```
db.sales.aggregate([
  {
    $group: {
      _id: null,
      totalAmount: { $sum: "$Quantity" },
      count: { $sum: 1 },
    },
  },
]);
```

5. How many items are that people are allergic to?

```
db.products.countDocuments({
  IsAllergic: true,
});
```

6. What is the average price of tea?

```
db.products.aggregate([
  {
    $match: {
      ProductName: { $regex: ".*[Tt][Ee][aA].*" },
    },
  },
  {
    $group: {
      _id: null,
      average: { $avg: "$Price" },
    },
  },
]);
```

7. Which employees did we hire after 2016?

```
db.employes.find({ HireDate: { $gte: "2016-01-01 00:00:00.000" } });
```

8. What is the total price of all items sold across the entire database?

```
db.sales.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "ProductID",
      foreignField: "ProductID",
      as: "products",
    },
  },
  {
    $project: {
      product: { $first: "$products" },
    },
  },
  {
    $project: {
      total: { $multiply: ["$Quantity", "$product.Price"] },
    },
  },
  {
    $group: {
      _id: null,
      sum: { $sum: "$total" },
    },
  },
]);
```

9. How many people were hired after 2016?

```
var pastas = db.products.find({ ProductName: { $regex: "%Pasta%" } });
db.sales.find({ ProductID: { $in: pastas } }).count();
```

10. How many women were hired since 2016?

```
var females = db.employes.find({ Gender: "F" });
db.females
  .find(
    { EmployeeID: { $in: females } },
    { HireDate: { $gte: "2016-01-01 00:00:00.000" } }
  )
  .count();
```

(f) - Investigate the balance between the consistency and the availability in your NoSQL system

The CAP theorem states that a distributed system cannot be simultaneously consistent, available and have partition tolerance. Consistency happens when all the clients either receive the most recent read, or an error. Availability happens when the client always gets a response when it requests for data (without necessarily having to be consistent). Partition tolerance is when the cluster continues to work no matter the number of messages that are dropped by the network. In the case of mongoDB, the system resolves partitions and maintains consistency, but it compromises on availability. The client only has one primary node that receives the all of the writes. Every other node (secondary nodes) will just copy it and apply the changes to their own datasets. When the primary node becomes unavailable, a secondary node will take its place and all the other secondary nodes will copy it. Once every secondary node has copied it, the cluster will become available again. During the time that the secondary nodes are copying the new primary node, the system will be unavailable (few seconds).

(g) - Indexing in MongoDB

When you are dealing with Big Data, any ineffeciencies in the execution of your queries will be very costly. That is why it's very important that your queries are optimized, and that you choose a good indexing strategy. MongoDB supports the creation of indexes on your document collections. This enables the efficient execution of queries against the database. If you neglect to add indexes on important fields in your collection, then MongoDB will have to perform a costly *collection scan*, in which it sequentially checks every document in the collection to find what you are looking for.

In general, indexing in MongoDB works similarly to indexing in a relational database; a B-tree data structure is used, and you get an index by default on the `_id` field (which is the MongoDB reserved primary key for all collections), but you can also create custom indexes using Mongo's JavaScript based query language.

So let's examine how an index is created in MongoDB. Here is an example of how an index would be created on the `Quantity` field of the `sales` collection:

```
use soen363;  // Switch our demo database

// Create a descending index on the `Quantity` field
db.sales.createIndex({ Quantity: -1 });
```

Note the use of `-1` makes the index a *descending* index, but this is only important for a compound index.