

# Inspections Restaurants Report - CCC2-02

Made by: Adrien DJEBAR, Emma FROMAGER, Abdelhak HACIB, Ridge LOWAO, Alex ROUSSEL

## 1. Introduction to the dataset

How to design schemas in NoSQL databases like Cassandra requires a different mindset compared to traditional SQL databases. In NoSQL, particularly in Cassandra, the schema design is heavily driven by the queries you need to perform. With that in mind, we have to understand our dataset thoroughly before we can design the schema, and also what kind of queries we have to perform.

Here's a peek of how our dataset is structured for a single row :

```
[  
 {  
   "idRestaurant": 40373938,  
   "restaurant": {  
     "name": "IHOP",  
     "borough": "BRONX",  
     "buildingnum": "5655",  
     "street": "BROADWAY",  
     "zipcode": "10463",  
     "phone": "7185494565",  
     "cuisineType": "American"  
   },  
   "inspectionDate": "2016-08-16",  
   "violationCode": "04L",  
   "violationDescription": "Evidence of mice or live mice present in facility's food and/or non-food areas.",  
   "criticalFlag": "Critical",  
   "score": 15,  
   "grade": ""  
 }  
 ]
```

Considering our dataset is of difficulty "2", we have to provide :

- 6 simple queries
- 2 complex queries
- 1 hard query

We have to design a schema that can answer these queries that we'll be running most frequently. Two approaches can be taken to design the schema:

- Multi-table design: Data is split into multiple tables, which can make simple queries more efficient. Two tables are created, one for *restaurants* which will hold all the relevant data about the restaurants, and one for *inspections* which will hold all the relevant data about the inspection of the said restaurants. Since in CQL there is no JOIN operation, if we would like to do queries that require data from both

tables, we would need materialized views or create a third table that would hold the data from both tables.

- Single table design: All data in one table, which can simplify complex and hard queries that need to access both *restaurant* and *inspection* data simultaneously. However, this approach can make simple queries more complex and less efficient. In this approach, the "restaurant" dictionary is denormalized so that we can access all the data in one table.

But what if we could have the best of both worlds ? We can use a multi-table design for simple queries, and a single table design for complex and hard queries. This way, we can have the best performance for all our queries. We can also use a materialized view to have the best of both worlds, but we'll have to see if it's necessary. NoSQL databases are more of a iterative process than SQL databases, so we'll have to see how our queries perform and adjust our schema accordingly.

## 2. Designing the schema

Let's take a closer look at how our dataset is structured. For each row, we can separate the data into two parts: the restaurant data and the inspection data. The inspection data hold the date of when the inspection happened, the violation code, a description, a critical flag, a score and a grade. While, the restaurant hold some information that aren't directly related to the inspection, like the name, the borough, the building number, the street, the zipcode, the phone number and the cuisine type. So it seems reasonable either way to split the data into two tables or to "flatten" the "restaurant" dictionary into it's elements.

```
[  
 {  
   "idRestaurant": 40373938,  
   "restaurant": {  
     "name": "IHOP",  
     "borough": "BRONX",  
     "buildingnum": "5655",  
     "street": "BROADWAY",  
     "zipcode": "10463",  
     "phone": "7185494565",  
     "cuisineType": "American"  
   },  
   "inspectionDate": "2016-08-16",  
   "violationCode": "04L",  
   "violationDescription": "Evidence of mice or live mice present in facility's food and/or non-food areas.",  
   "criticalFlag": "Critical",  
   "score": 15,  
   "grade": ""  
 }  
 ]
```

### 2.1 Mutli-table design

Here's how we can design the schema using a multi-table design:

```
CREATE TABLE IF NOT EXISTS restaurant (
    idRestaurant INT,
    name TEXT,
    borough TEXT,
    buildingnum TEXT,
    street TEXT,
    zipcode TEXT,
    phone TEXT,
    cuisineType TEXT,
    PRIMARY KEY (idRestaurant)
);

CREATE TABLE IF NOT EXISTS inspection (
    idRestaurant INT,
    inspectionDate DATE,
    violationCode TEXT,
    violationDescription TEXT,
    criticalFlag TEXT,
    score INT,
    grade TEXT,
    PRIMARY KEY (idRestaurant, inspectionDate)
);
```

Both tables have "idRestaurant" as the partition key, which is the unique identifier for each restaurant. This way, we keep a relationship between the two tables. The "inspection" table has "inspectionDate" as the clustering column, which will allow us to sort the data by date.

## 2.2 Single table design

Here's how we can design the schema using a single table design:

```
CREATE TABLE restaurant_inspection_combined (
    idRestaurant INT,
    inspectionDate DATE,
    name TEXT,
    borough TEXT,
    buildingnum TEXT,
    street TEXT,
    zipcode TEXT,
    phone TEXT,
    cuisineType TEXT,
    violationCode TEXT,
    violationDescription TEXT,
    criticalFlag TEXT,
    score INT,
    grade TEXT,
    PRIMARY KEY (idRestaurant, inspectionDate)
);
```

In this design, we have denormalized the data from both `restaurant` and `inspection` tables into a single table. Each row in this table would represent an inspection, including all relevant restaurant details. This approach allows us to run complex queries that need information from both entities without the need for joins.

## 3. Importing the data into Cassandra

### 3.1 Cleaning the JSON file

Before we can import the data into Cassandra, we have to clean the JSON file. The JSON file is not in a format that Cassandra can understand. However, our JSON objects are not in an array, and they are not separated by commas.

For this, we created a simple Python script to add the missing commas and to put the JSON objects into an array. Here's the script:

```
# Fix the JSON data not being in an array format
with open('InspectionsRestaurant.json', 'r') as read_obj, open('InspectionsRestaurantFixed.json', 'w') as write_obj:
    write_obj.write('[')

    # Read the file line by line
    for line in read_obj:
        # Add a comma to the end of the line and write
        write_obj.write(line.rstrip('\n') + ',\n')

    # Remove the last comma
    write_obj.seek(write_obj.tell() - 3)

    # Write the end of the file
    write_obj.write(']')
```

If we compare the line count of the original file and the cleaned file with "wc -l", we have the same number of lines.

|   |                  |
|---|------------------|
| Exercices/_dataset/Cassandra via  v3.11.5<br>17:42:24 > wc -l InspectionsRestaurant.json<br>442795 InspectionsRestaurant.json           | 13GiB/31GiB 38ms |
| Exercices/_dataset/Cassandra via  v3.11.5<br>17:42:41 > wc -l InspectionsRestaurantFixed.json<br>442795 InspectionsRestaurantFixed.json | 13GiB/31GiB 3s   |

### 3.2 Converting the JSON file to CSV

To efficiently convert the JSON file to CSV, we can use the library `pandas` in Python to create dataframes that enable to work with the data in a more structured way. We can then save the dataframes to CSV files. Here's the script:

```
import json
import os

import pandas as pd

# Create the data directory if it does not exist
os.makedirs("csv", exist_ok=True)
```

```
# Read the json file
with open("json/InspectionsRestaurantFixed.json") as f:
    data = json.load(f)

    restaurant_data = []
    inspection_data = []
    inspections_restaurants_data = []

    # Loop through the data once
    for item in data:
        # Create the restaurant data table
        restaurant = item["restaurant"]
        restaurant["idRestaurant"] = item["idRestaurant"]
        restaurant_data.append(restaurant)

        # Create the inspection data table
        inspection = {
            k: v if v != "" else "Not Yet Graded"
            for k, v in item.items()
            if k != "restaurant"
        }
        inspection_data.append(inspection)

        # Create the combined data table
        combined = {**restaurant, **inspection}
        inspections_restaurants_data.append(combined)

    # Create the dataframes
    restaurant_df = pd.DataFrame(restaurant_data)
    inspection_df = pd.DataFrame(inspection_data)
    inspections_restaurants_df = pd.DataFrame(inspections_restaurants_data)

    # Print the line count for each dataframe
    print(f"restaurant_df: {len(restaurant_df)} lines")
    print(f"inspection_df: {len(inspection_df)} lines")
    print(f"inspections_restaurants_df: {len(inspections_restaurants_df)} lines")

    # Save the dataframes to csv files
    restaurant_df.to_csv(
        "csv/restaurant.csv",
        sep=";",
        index=False,
        columns=[
            "idRestaurant",
            "name",
            "borough",
            "buildingnum",
            "street",
            "zipcode",
            "phone",
            "cuisineType",
        ],
    )
```

```

inspection_df.to_csv(
    "csv/inspection.csv",
    sep=";",
    index=False,
    columns=[
        "idRestaurant",
        "inspectionDate",
        "violationCode",
        "violationDescription",
        "criticalFlag",
        "score",
        "grade",
    ],
)
inspections_restaurants_df.to_csv(
    "csv/inspections_restaurants.csv",
    sep=";",
    index=False,
    columns=[
        "idRestaurant",
        "inspectionDate",
        "name",
        "borough",
        "buildingnum",
        "street",
        "zipcode",
        "phone",
        "cuisineType",
        "violationCode",
        "violationDescription",
        "criticalFlag",
        "score",
        "grade",
    ],
)

```

If we run the script, we can see that the line count for each dataframe is the same as the original JSON file.

```

🏠 Exercices\_dataset\Cassandra via 🐍 v3.11.5
📅 20:37:24 ➤ python -u "d:\OneDrive - De Vinci\Cours\ESILV\Année 4\Semestre 2\Core Computer Science\NoSQL\Exercices\_dataset\Cassandra\create_csv.py"
restaurant_df: 442795 lines
inspection_df: 442795 lines
restaurant_inspection_df: 442795 lines

```

We can then compress the CSV files into a tar.gz file to make it easier to copy to the Cassandra container.

```
tar -czvf inspections_restaurant.tar.gz *
```

### 3.3 Importing the CSV files into Cassandra

Now that we have the CSV files, we can import them into Cassandra. We can use a CQL script that creates the tables and imports the data into the tables. For it, we have to copy the CSV files and the CQL script to the Cassandra container. We can use the `docker cp` command to copy the files to the container.

```
docker cp /path/to/inspections_restaurant.tar.gz cassandra_nosql:/home
docker cp /path/to/create_import_table.cql cassandra_nosql:/home
```

We can then extract the tar.gz file and import the data into Cassandra.

```
tar xzvf inspections_restaurant.tar.gz -C ./inspections_restaurant
```

We can then use the `cqlsh` command to run the CQL script.

```
docker exec -it cassandra_nosql cqlsh -f /home/create_import_table.cql
```

And that's it! We have imported the data into Cassandra.

The screenshot shows a terminal window with a dark background. At the top, there's a header bar with a yellow star icon, the text "Creation in progress...", and some other icons. Below the header, the terminal output is displayed in white text. It starts with a timestamp "13:04:17" and a message about creating a keyspace via "Cassandra via v3.11.5". It then shows the execution of a CQL script to create a table, noting that the table already exists. Following this, it shows the import of data from CSV files into three tables: "inspections\_restaurant", "inspection", and "restaurant\_inspections". Each import process is detailed with the number of rows processed, the rate, and the total time taken. Finally, the terminal shows a cleanup step where child processes are being terminated.

```
Creation in progress...
Exercises\_dataset\Cassandra via v3.11.5
13:04:17 > docker exec -it cassandra_nosql cqlsh -f /home/create_import_table.cql
/home/create_import_table.cql:49:AlreadyExists: Table 'inspections_restaurant.restaurant_inspections' already exists
Using 16 child processes

Starting copy of inspections_restaurant.restaurant with columns [idrestaurant, name, borough, buildingnum, street, zipcode, phone, cuisinetype].
Processed: 442795 rows; Rate: 48487 rows/s; Avg. rate: 97834 rows/s
442795 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 4.526 seconds (0 skipped).
Using 16 child processes

Starting copy of inspections_restaurant.inspection with columns [idrestaurant, inspectiondate, violationcode, violationdescription, criticalflag, score, grade].
Processed: 442795 rows; Rate: 58184 rows/s; Avg. rate: 97684 rows/s
442795 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 4.533 seconds (0 skipped).
Using 16 child processes

Starting copy of inspections_restaurant.restaurant_inspections with columns [idrestaurant, inspectiondate, name, borough, buildingnum, street, zipcode, phone, cuisinetype, violationcode, violationdescription, criticalflag, score, grade].
Processed: 442795 rows; Rate: 52383 rows/s; Avg. rate: 96141 rows/s
442795 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 4.606 seconds (0 skipped).

Exercises\_dataset\Cassandra via v3.11.5
13:04:42 X | 35GiB/63GiB 14s
```

The content of the `create_import_table.cql` file is as follows:

```
-- Description: This script creates the keyspace and the table for the restaurant
inspections data.
CREATE KEYSPACE IF NOT EXISTS inspections_restaurant
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
```

```
-- Create the table for the restaurant inspections data.  
USE inspections_restaurant;  
  
-- Create the table for the restaurant inspections data.  
CREATE TABLE IF NOT EXISTS restaurant (  
    idRestaurant INT,  
    name TEXT,  
    borough TEXT,  
    buildingnum TEXT,  
    street TEXT,  
    zipcode TEXT,  
    phone TEXT,  
    cuisineType TEXT,  
    PRIMARY KEY (idRestaurant)  
);  
  
CREATE TABLE IF NOT EXISTS inspection (  
    idRestaurant INT,  
    inspectionDate DATE,  
    violationCode TEXT,  
    violationDescription TEXT,  
    criticalFlag TEXT,  
    score INT,  
    grade TEXT,  
    PRIMARY KEY (idRestaurant, inspectionDate)  
);  
  
CREATE TABLE inspections_restaurants (  
    idRestaurant INT,  
    inspectionDate DATE,  
    name TEXT,  
    borough TEXT,  
    buildingnum TEXT,  
    street TEXT,  
    zipcode TEXT,  
    phone TEXT,  
    cuisineType TEXT,  
    violationCode TEXT,  
    violationDescription TEXT,  
    criticalFlag TEXT,  
    score INT,  
    grade TEXT,  
    PRIMARY KEY (idRestaurant, inspectionDate)  
);  
  
-- Import the data into the table.  
COPY restaurant (idRestaurant, name, borough, buildingnum, street, zipcode, phone,  
cuisineType)  
FROM '/home/inspections_restaurants/restaurant.csv' WITH HEADER=TRUE AND  
DELIMITER=';';  
  
COPY inspection (idRestaurant, inspectionDate, violationCode,  
violationDescription, criticalFlag, score, grade)
```

```
FROM '/home/inspections_restaurants/inspection.csv' WITH HEADER=TRUE AND  
DELIMITER=';';  
  
COPY inspections_restaurants (idRestaurant, inspectionDate, name, borough,  
buildingnum, street, zipcode, phone, cuisineType, violationCode,  
violationDescription, criticalFlag, score, grade)  
FROM '/home/inspections_restaurants/inspections_restaurants.csv' WITH HEADER=TRUE  
AND DELIMITER=';';
```

### 3.4 Automating the process

We can automate the process of importing the data into Cassandra by creating a shell script that does all the steps for us. First clone the [GitHub repository](#) and execute the `execute_pipeline.sh` script:

```
#!/bin/bash  
  
# Prompt the user for the Docker image name  
echo "Please enter the Docker image name for Cassandra:"  
read docker_name  
  
# Check if the Docker image name is empty  
if [ -z "$docker_name" ]; then  
    echo "No Docker image name provided. Exiting."  
    exit 1  
fi  
  
echo "Executing pipeline for Cassandra..."  
  
echo -e "[INFO] Correcting array comma in json file..."  
python fix_array_comma.py  
echo -e "[INFO] Done! ✓"  
  
echo -e "\n[INFO] Creating csv files from json for Cassandra readable files..."  
python create_csv.py  
echo -e "[INFO] Done! ✓"  
  
echo -e "\n[INFO] Compressing csv files to tar.gz for transfer in the docker..."  
(cd csv && tar -czvf ../inspections_restaurants.tar.gz *)  
  
echo -e "\n[INFO] Copying tar.gz file and script to docker..."  
docker cp inspections_restaurants.tar.gz $docker_name:/home  
docker cp create_import_table.cql $docker_name:/home  
  
echo -e "\n[INFO] Extracting tar.gz file in docker..."  
docker exec -it $docker_name mkdir /home/inspections_restaurants  
docker exec -it $docker_name tar -xzvf /home/inspections_restaurants.tar.gz -C  
/home/inspections_restaurants  
  
echo -e "\n[INFO] Creating Cassandra tables and import data..."  
docker exec -it $docker_name cqlsh -f /home/create_import_table.cql  
echo -e "[INFO] Done! ✓"
```

```
echo -e "\n[INFO] Executing pipeline... Done! ✓"
```

## 4. Running the queries

Now that we have imported the data into Cassandra, we can run the queries.

### 4.1 Simple queries

- List of Starbucks restaurant.

```
CREATE INDEX ON restaurant (name);
SELECT * FROM restaurant WHERE name='STARBUCKS';
```

|    | Query 1      |               |             |                 |           |             | Query 2                   |         |  |
|----|--------------|---------------|-------------|-----------------|-----------|-------------|---------------------------|---------|--|
|    | idrestaurant | borough       | buildingnum | cuisinetype     | name      | phone       | street                    | zipcode |  |
| 1  | 40737930     | MANHATTAN     | 55          | Café/Coffee/Tea | STARBUCKS | 2127507140  | EAST 53 STREET            | 10022   |  |
| 2  | 41600160     | MANHATTAN     | 11011109    | Café/Coffee/Tea | STARBUCKS | 2123823917  | AVENUE OF THE AMERICAS    | 10036   |  |
| 3  | 50036218     | MANHATTAN     | 1700        | Sandwiches      | STARBUCKS | 2126159700  | BROADWAY                  | 10019   |  |
| 4  | 40566621     | STATEN ISLAND | 2530        | Café/Coffee/Tea | STARBUCKS | 7189798608  | HYLAN BOULEVARD           | 10306   |  |
| 5  | 40587698     | MANHATTAN     | 540         | Café/Coffee/Tea | STARBUCKS | 2124964163  | COLUMBUS AVENUE           | 10024   |  |
| 6  | 40788886     | MANHATTAN     | 750         | Café/Coffee/Tea | STARBUCKS | 6462307208  | 6 AVENUE                  | 10010   |  |
| 7  | 41249345     | MANHATTAN     | 425         | Café/Coffee/Tea | STARBUCKS | 21264449462 | MADISON AVENUE            | 10017   |  |
| 8  | 50005906     | MANHATTAN     | 270         | Café/Coffee/Tea | STARBUCKS | 2122702624  | PARK AVE                  | 10017   |  |
| 9  | 40602866     | MANHATTAN     | 444         | Café/Coffee/Tea | STARBUCKS | 2127692296  | COLUMBUS AVENUE           | 10024   |  |
| 10 | 41642422     | MANHATTAN     | 491/2       | Café/Coffee/Tea | STARBUCKS | 2127771752  | 1 AVENUE                  | 10003   |  |
| 11 | 50000500     | MANHATTAN     | 625         | Café/Coffee/Tea | STARBUCKS | 2122739613  | 8TH AVE                   | 10018   |  |
| 12 | 50033039     | MANHATTAN     | 1           | Café/Coffee/Tea | STARBUCKS | 2127858409  | NEW YORK PLZ              | 10004   |  |
| 13 | 40587720     | MANHATTAN     | 1           | Café/Coffee/Tea | STARBUCKS | 2127603001  | PENN PLAZA                | 10119   |  |
| 14 | 50004817     | QUEENS        | TERMINAL 5  | Café/Coffee/Tea | STARBUCKS | 7186790536  | JFK INTERNATIONAL AIRPORT | 11430   |  |
| 15 | 41567009     | QUEENS        | 3101        | Café/Coffee/Tea | STARBUCKS | 7187771305  | BROADWAY                  | 11106   |  |
| 16 | 40532870     | MANHATTAN     | 124         | Café/Coffee/Tea | STARBUCKS | 2124622020  | 8 AVENUE                  | 10011   |  |
| 17 | 41014003     | MANHATTAN     | 1535        | Café/Coffee/Tea | STARBUCKS | 2127048937  | BROADWAY                  | 10036   |  |
| 18 | 41006744     | MANHATTAN     | 770         | Café/Coffee/Tea | STARBUCKS | 2128309001  | 8 AVENUE                  | 10036   |  |
| 19 | 40946403     | STATEN ISLAND | 468         | Café/Coffee/Tea | STARBUCKS | 7182735984  | FOREST AVENUE             | 10301   |  |
| 20 | 40545938     | MANHATTAN     | 1100        | Café/Coffee/Tea | STARBUCKS | 2123984588  | AVENUE OF THE AMERICAS    | 10036   |  |
| 21 | 50008135     | MANHATTAN     | 1542        | Café/Coffee/Tea | STARBUCKS | 2123692949  | 3RD AVE                   | 10028   |  |
| 22 | 40591262     | MANHATTAN     | 1841        | Café/Coffee/Tea | STARBUCKS | 2123070162  | BROADWAY                  | 10023   |  |
| 23 | 40748086     | MANHATTAN     | 72          | Café/Coffee/Tea | STARBUCKS | 6464862216  | GROVE STREET              | 10014   |  |

Data Message Chart 7 ms

286 rows

- The type of cooking present on Broadway street.

```
CREATE INDEX ON restaurant (street);
SELECT cuisinetype, name FROM restaurant WHERE street='BROADWAY';
```

|    | Query 1  | Query 2                           |
|----|--|-----------------------------------|
|    | cuisinetype  | name                              |
| 1  | Latin (Cuban, Dominican, Puerto Rican, South & Central American) | EL GUANACO RESTAURANT & PUPUSERIA |
| 2  | Chicken  | PLANET WINGS                      |
| 3  | Jewish/Kosher  | CAFE 11                           |
| 4  | American   | CROWN FRIED CHICKEN               |
| 5  | Pizza  | FAIRYTALE PIZZERIA                |
| 6  | Other  | RESTAURANT ASSOCIATES LLC         |
| 7  | Armenian   | GARDEN CAFE                       |
| 8  | Mexican  | CIPOTLE MEXICAN GRILL             |
| 9  | Spanish  | PARILLA LATINA                    |
| 10 | Seafood  | BROTHERS FISH MARKET              |
| 11 | Café/Coffee/Tea  | UPTOWN BOURBON                    |
| 12 | Spanish  | KALINA RESTAURANT                 |
| 13 | American   | WOOLWORTH TOWER KITCHEN           |
| 14 | Chinese  | FATIMA HALAL KITCHEN ASTORIA      |
| 15 | Sandwiches   | STARBUCKS                         |
| 16 | American   | MARCHA COCINA BAR                 |
| 17 | Café/Coffee/Tea  | IRVING FARM COFFEE ROASTERS       |
| 18 | Café/Coffee/Tea  | BLANK CAFE                        |
| 19 | Japanese   | KONDO JAPANESE RESTAURANT         |
| 20 | American   | COZY SOUP & BURGER                |
| 21 | American   | DIVINE BAR & GRILL                |
| 22 | Italian  | OBICA MOZZARELLA BAR              |
| 23 | Sandwiches/Salads/Mixed Buffet                                   | FRESH & CO                        |

Data   Message   Chart   13 ms

957 rows

- The grades of indian restaurants.

```
CREATE INDEX ON inspections_restaurants (cuisineType);
SELECT idrestaurant, name, cuisineType, grade, score FROM inspections_restaurants
WHERE cuisineType = 'Indian';
```

|    | Query 1      | Query 2                     |             |                |       |
|----|--------------|-----------------------------|-------------|----------------|-------|
|    | idrestaurant | name                        | cuisineType | grade          | score |
| 1  | 50044741     | INDIAN EXPRESS              | Indian      | Not Yet Graded | 0     |
| 2  | 50044741     | INDIAN EXPRESS              | Indian      | A              | 12    |
| 3  | 50049046     | NAAN & GRILL                | Indian      | Not Yet Graded | 38    |
| 4  | 50049046     | NAAN & GRILL                | Indian      | A              | 12    |
| 5  | 50017852     | CHOTE NAWAB                 | Indian      | A              | 9     |
| 6  | 50017852     | CHOTE NAWAB                 | Indian      | Not Yet Graded | 21    |
| 7  | 50017852     | CHOTE NAWAB                 | Indian      | A              | 5     |
| 8  | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | A              | 7     |
| 9  | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | A              | 5     |
| 10 | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | Not Yet Graded | 24    |
| 11 | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | A              | 12    |
| 12 | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | Not Yet Graded | 19    |
| 13 | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | B              | 15    |
| 14 | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | Not Yet Graded | 18    |
| 15 | 41487281     | GOLDEN TERRACE BANQUET HALL | Indian      | A              | 9     |
| 16 | 50050501     | BAWARCHI INDIAN CUISINE     | Indian      | A              | 13    |
| 17 | 40394066     | BOMBAY MASALA               | Indian      | Not Yet Graded | 24    |
| 18 | 40394066     | BOMBAY MASALA               | Indian      | A              | 13    |
| 19 | 40394066     | BOMBAY MASALA               | Indian      | Not Yet Graded | 0     |
| 20 | 40394066     | BOMBAY MASALA               | Indian      | A              | 5     |
| 21 | 40394066     | BOMBAY MASALA               | Indian      | Not Yet Graded | 19    |
| 22 | 40394066     | BOMBAY MASALA               | Indian      | A              | 10    |
| 23 | 40394066     | BOMBAY MASALA               | Indian      | A              | 9     |

Data   Message   Chart   19 ms

2,195 rows

- Show the score and grade with the violation description for each restaurant. It might be interesting to see if there is a correlation between the grade and the score.

```
SELECT name, violationdescription, score, grade FROM inspections_restaurants;
```

|    | name              | violationdescription                              | score | grade          |
|----|-------------------|---|-------|----------------|
| 1  | BOSTON MARKET     | Wiping cloths soiled or not stored in sanitiz...  | 12    | A              |
| 2  | BOSTON MARKET     | Food contact surface not properly washed, rin...  | 12    | A              |
| 3  | BOSTON MARKET     | Sanitized equipment or utensil, including in...   | 7     | A              |
| 4  | LENIHAN'S SALOON  | Evidence of mice or live mice present in faci...  | 30    | Not Yet Graded |
| 5  | LENIHAN'S SALOON  | Lighting inadequate; permanent lighting not p...  | 16    | B              |
| 6  | LENIHAN'S SALOON  | Plumbing not properly installed or maintained...  | 15    | Not Yet Graded |
| 7  | LENIHAN'S SALOON  | Filth flies or food/refuse/sewage-associated...   | 13    | A              |
| 8  | LENIHAN'S SALOON  | Evidence of mice or live mice present in faci...  | 31    | Not Yet Graded |
| 9  | LENIHAN'S SALOON  | Evidence of mice or live mice present in faci...  | 9     | A              |
| 10 | LENIHAN'S SALOON  | Non-food contact surface improperly construct...  | 13    | A              |
| 11 | LENIHAN'S SALOON  | Non-food contact surface improperly construct...  | 2     | A              |
| 12 | BANGKOK HOUSE     | Mechanical or natural ventilation system not...   | 54    | Not Yet Graded |
| 13 | BANGKOK HOUSE     | Non-food contact surface improperly construct...  | 11    | P              |
| 14 | BANGKOK HOUSE     | Cold food item held above 41°F (smoked fish...    | 15    | B              |
| 15 | BANGKOK HOUSE     | Food contact surface not properly washed, rin...  | 29    | Not Yet Graded |
| 16 | BANGKOK HOUSE     | Food contact surface not properly washed, rin...  | 9     | A              |
| 17 | BANGKOK HOUSE     | Cold food item held above 41°F (smoked fish...    | 31    | Not Yet Graded |
| 18 | BANGKOK HOUSE     | Hot food item not held at or above 140°F.         | 43    | C              |
| 19 | BANGKOK HOUSE     | Raw, cooked or prepared food is adulterated,...   | 15    | Not Yet Graded |
| 20 | BANGKOK HOUSE     | Live roaches present in facility's food and/o...  | 6     | A              |
| 21 | BANGKOK HOUSE     | Cold food item held above 41°F (smoked fish...    | 22    | Not Yet Graded |
| 22 | BANGKOK HOUSE     | Evidence of rats or live rats present in faci...  | 11    | A              |
| 23 | BANGKOK HOUSE     | Food contact surface not properly washed, rin...  | 13    | A              |
| 24 | TJ'S TASTY CORNER | Food Protection Certificate not held by super...  | 13    | A              |
| 25 | TJ'S TASTY CORNER | Evidence of mice or live mice present in faci...  | 18    | Not Yet Graded |
| 26 | TJ'S TASTY CORNER | Hot food item not held at or above 140°F.         | 17    | B              |
| 27 | TJ'S TASTY CORNER | Evidence of mice or live mice present in faci...  | 56    | Not Yet Graded |
| 28 | TJ'S TASTY CORNER | Facility not vermin proof. Harborage or condit... | 18    | B              |
| 29 | TJ'S TASTY CORNER | Hot food item not held at or above 140°F.         | 17    | Not Yet Graded |

Data Message Chart 2.963 s

150,882 rows

- Display the inspection dates for a specific restaurant named SPRING.

```
CREATE INDEX ON inspections_restaurants (name);
SELECT idRestaurant, name, inspectionDate FROM inspections_restaurants WHERE name = 'SPRING';
```

|    | Query 1      |        |                | Query 2 |
|----|--------------|--------|----------------|---------|
|    | idrestaurant | name   | inspectiondate |         |
| 1  | 41710788     | SPRING | 2013-01-30     |         |
| 2  | 41710788     | SPRING | 2013-09-26     |         |
| 3  | 41710788     | SPRING | 2014-01-24     |         |
| 4  | 41710788     | SPRING | 2014-06-07     |         |
| 5  | 41710788     | SPRING | 2014-06-27     |         |
| 6  | 41710788     | SPRING | 2014-08-21     |         |
| 7  | 41710788     | SPRING | 2014-12-17     |         |
| 8  | 41710788     | SPRING | 2015-01-07     |         |
| 9  | 41710788     | SPRING | 2015-05-11     |         |
| 10 | 41710788     | SPRING | 2015-05-28     |         |
| 11 | 41710788     | SPRING | 2015-11-05     |         |

Data Message Chart 2 ms 11 rows Export... ⚙️ 🔍

- Grades and scores: We want to see the relationship between grades and scores.

```
SELECT score, grade  
FROM inspections_restaurants;
```

|    | score | grade |
|----|-------|-------|
| 61 | 22    | Empty |
| 62 | 15    | B     |
| 63 | 17    | Empty |
| 64 | 13    | A     |
| 65 | 9     | A     |
| 66 | 12    | A     |
| 67 | 26    | Empty |
| 68 | 15    | B     |
| 69 | 25    | Empty |
| 70 | 10    | A     |
| 71 | 2     | A     |
| 72 | 9     | A     |
| 73 | 15    | Empty |
| 74 | 50    | Empty |
| 75 | 0     | C     |
| 76 | 9     | Empty |
| 77 | 20    | Empty |
| 78 | 11    | A     |

Data
Message
Chart
4.804 s
147,625 rows

We can see that a grade of A generally correlates with a low score, B is in between and C generally correlates with a high score (bad).

## 4.2 Complex queries

- Display the name and the grade of restaurants that have a `criticalFlag` set to `Critical`.

```
CREATE INDEX ON inspections_restaurants (criticalflag);
SELECT name, grade, inspectiondate FROM inspections_restaurants WHERE
criticalflag = 'Critical' GROUP BY idRestaurant;
```

|    | name                              | grade          | inspectiondate |
|----|-----------------------------------|----------------|----------------|
| 1  | BOSTON MARKET                     | A              | 2014-07-10     |
| 2  | LENIHAN'S SALOON                  | Not Yet Graded | 2013-06-11     |
| 3  | BANGKOK HOUSE                     | B              | 2013-06-10     |
| 4  | TJ'S TASTY CORNER                 | A              | 2013-10-16     |
| 5  | NATHAN'S HOT DOGS                 | A              | 2015-04-30     |
| 6  | YUMMY YUMMY                       | A              | 2014-04-21     |
| 7  | KING'S KITCHEN                    | A              | 2014-03-04     |
| 8  | CESCA                             | A              | 2013-03-13     |
| 9  | EL GUANACO RESTAURANT & PUPUSERIA | B              | 2013-07-29     |
| 10 | THE BROOK                         | Not Yet Graded | 2013-05-03     |
| 11 | LEO'S DELI & GRILL                | Not Yet Graded | 2015-04-16     |
| 12 | KAEDÉ JAPANESE RESTAURANT         | B              | 2014-11-26     |
| 13 | JOHNS PIZZA                       | A              | 2013-09-25     |
| 14 | RICE & BEANS LECHONERA            | A              | 2012-12-06     |
| 15 | DOUBLETREE GREENHOUSE 36          | Not Yet Graded | 2014-03-10     |
| 16 | VIDAELVA                          | A              | 2014-06-10     |
| 17 | TK VILLAGE BAKERY COMPANY         | A              | 2016-04-14     |
| 18 | BKB                               | Not Yet Graded | 2012-12-21     |
| 19 | SPRING                            | Not Yet Graded | 2013-09-26     |
| 20 | DUNKIN' DONUTS                    | A              | 2014-04-28     |
| 21 | PLANET WINGS                      | B              | 2016-01-14     |
| 22 | NEW CHINA RESTAURANT WANG INC     | A              | 2015-09-30     |
| 23 | JIN SUSHI & THAI                  | Not Yet Graded | 2015-05-06     |

Funnily enough, we can see restaurants with a `criticalFlag` set to `Critical` have a grade of `A` or `B`. This is interesting because we would expect a `C` grade for restaurants with a `criticalFlag` set to `Critical`. There's also a lot of restaurants that simply don't have a grade, so we can't really know if they are good or bad.

- Number of inspections for restaurants that have a 'C' grade

```
CREATE INDEX ON inspections_restaurants (grade);
SELECT idRestaurant, name, COUNT(inspectionDate) AS num_inspections FROM
inspections_restaurants WHERE grade = 'C' GROUP BY idRestaurant;
```

|    | Query 1      |                               | Query 2         |
|----|--------------|-------------------------------|-----------------|
|    | idrestaurant | name                          | num_inspections |
| 1  | 41692194     | BANGKOK HOUSE                 | 1               |
| 2  | 50019260     | LEO'S DELI & GRILL            | 1               |
| 3  | 50015473     | KAEDA JAPANESE RESTAURANT     | 1               |
| 4  | 41630020     | DOUBLETREE GREENHOUSE 36      | 1               |
| 5  | 50009349     | JIN SUSHI & THAI              | 1               |
| 6  | 50001970     | CROWN FRIED CHICKEN           | 1               |
| 7  | 41642313     | LA ROSA BAKERY                | 1               |
| 8  | 41049219     | KENNEDY CHICKEN AND PIZZA     | 1               |
| 9  | 40364576     | TOUT VA BIEN                  | 2               |
| 10 | 50015221     | CHINA DRAGON                  | 1               |
| 11 | 50045140     | AUNTIE GUAN'S KITCHEN 108     | 1               |
| 12 | 41696147     | BREUCKELN COLONY              | 1               |
| 13 | 41193088     | MOJITO'S                      | 1               |
| 14 | 41297769     | BROTHERS FISH MARKET          | 2               |
| 15 | 50042658     | FISH & SHRIMP RESTAURANT SPOT | 1               |
| 16 | 50004117     | G's RESTAURANT & BAR          | 1               |
| 17 | 50002789     | SING WAH RESTAURANT           | 1               |
| 18 | 50049610     | EMPANADAS MONUMENTAL          | 1               |
| 19 | 41708795     | TEA CUP CAFE                  | 1               |
| 20 | 40694203     | DUNKIN' DONUTS                | 1               |
| 21 | 40759586     | CAFE BUON GUSTO               | 1               |
| 22 | 50037680     | TILILA BAR & GRILL            | 1               |
| 23 | 50002707     | LIBERTY CATERING              | 1               |

Data    Message    Chart    26 ms

1,684 rows

Export...

We can see something interesting. Indeed, the number of inspections for "bad restaurants" (because they have a C mark) is generally equal to 1. This is interesting because we would expect a "bad restaurant" to have more inspections than a "good restaurant". This is a good example of how the data can be misleading.

## 4.3 Hard query

- Creation of the state function

```
CREATE OR REPLACE FUNCTION avgState(state tuple<int, int>, val int)
CALLED ON NULL INPUT RETURNS tuple<int, int> LANGUAGE java AS '
if (val != null) {
    state.setInt(0, state.getInt(0) + val);
    state.setInt(1, state.getInt(1) + 1);
}
return state;';
```

|    | idrestaurant | avg_score        |
|----|--------------|------------------|
| 1  | 40786914     | 10,3333333333333 |
| 2  | 40366162     | 16,125           |
| 3  | 41692194     | 21,5833333333333 |
| 4  | 41430956     | 24,5833333333333 |
| 5  | 41395531     | 6                |
| 6  | 50005384     | 11               |
| 7  | 50005858     | 10               |
| 8  | 40962612     | 16               |
| 9  | 40995404     | 16,625           |
| 10 | 40368763     | 14,5             |
| 11 | 50019260     | 16               |
| 12 | 50015473     | 17,6666666666667 |
| 13 | 41704055     | 16,875           |
| 14 | 40860476     | 16,7142857142857 |
| 15 | 50044741     | 6                |
| 16 | 41630020     | 14,25            |
| 17 | 41569081     | 10,75            |
| 18 | 50048575     | 13               |
| 19 | 41569184     | 17               |
| 20 | 41710788     | 13,4545454545455 |
| 21 | 41443240     | 6,5              |
| 22 | 41517701     | 18               |
| 23 | 50015747     | 13,3333333333333 |

Data    Message    Chart    9,393 s

26 001 rows

- Creation of the final function to calculate the average

```
CREATE OR REPLACE FUNCTION avgFinal(state tuple<int, int>)
CALLED ON NULL INPUT RETURNS double
LANGUAGE java AS '
int sum = state.getInt(0);
int count = state.getInt(1);
return (double) sum / count;
';
```

|    | idrestaurant | avg_score |
|----|--------------|-----------|
| 1  | 40786914     | 10        |
| 2  | 40366162     | 16        |
| 3  | 41692194     | 21        |
| 4  | 41430956     | 24        |
| 5  | 41395531     | 6         |
| 6  | 50005384     | 11        |
| 7  | 50005858     | 10        |
| 8  | 40962612     | 16        |
| 9  | 40995404     | 16        |
| 10 | 40368763     | 14        |
| 11 | 50019260     | 16        |
| 12 | 50015473     | 17        |
| 13 | 41704055     | 16        |
| 14 | 40860476     | 16        |
| 15 | 50044741     | 6         |
| 16 | 41630020     | 14        |
| 17 | 41569081     | 10        |
| 18 | 50048575     | 13        |
| 19 | 41569184     | 17        |
| 20 | 41710788     | 13        |
| 21 | 41443240     | 6         |
| 22 | 41517701     | 18        |
| 23 | 50015747     | 13        |

Data    Message    Chart    726 ms

26 001 rows

- Creation of the UDA function

```
CREATE OR REPLACE AGGREGATE average ( int )
SFUNC avgState STYPE tuple<int,int>
FINALFUNC avgFinal INITCOND (0,0);
```

- We display the average score of each restaurant using our newly created UDA function "average"

```
SELECT idrestaurant, average(score) AS total_score
FROM restaurant_inspections
GROUP BY idrestaurant;
```

- We do the same but this time using the included aggregate function "AVG"

```
SELECT idrestaurant, AVG(score) AS avg_score
FROM restaurant_inspections
GROUP BY idrestaurant;
```

At first, we can see that using our UDA function "average", we achieve better results leading us to believe that UDA functions would be the go too if we need precision. But that precision comes with a price. If we compare the time it takes for each query to run and return results, a huge gap in compute time is visible between our UDA function and the onboard aggregate function.

## 5. Conclusion

In conclusion, we have successfully imported the data into Cassandra and ran the queries. We have designed the schema using a multi-table design and a single table design and automated the process of importing the data into Cassandra. We have run simple, complex and hard queries to practice and better understand CQL as a whole.

Thanks for reading! 🚀