University of El Oued
Faculty of Exact Sciences
Department of Computer Science

Lab Report 6: NoSQL Databases

Cassandra Setup & CQL Operations

Student: Mohammed Seddik Lifa

Specialization: Master II: AI & Data Science

Date: November 23, 2025

1 Objective

The primary objective of this lab is to install and configure a Cassandra database environment using Docker. The lab involves creating a Keyspace (‘resto$_N$Y‘), $definingaschemawithtablesforRestauran$

2 Implementation & Proof of Execution

2.1 A. Create Keyspace & Tables

Instruction: Create a keyspace named ‘resto$_N$Y‘$with$‘$SimpleStrategy$‘$replication.Then, create the‘Resta$

Code Pattern Executed:

```
CREATE KEYSPACE IF NOT EXISTS resto_NY WITH REPLICATION = ...;
CREATE TABLE Restaurant (id INT, Name VARCHAR... PRIMARY KEY (id));
CREATE TABLE Inspection (... PRIMARY KEY (idRestaurant, InspectionDate
    ));
```

Execution Proof: The keyspace and tables were created successfully. The screenshots below verify the schema creation and the addition of secondary indexes.

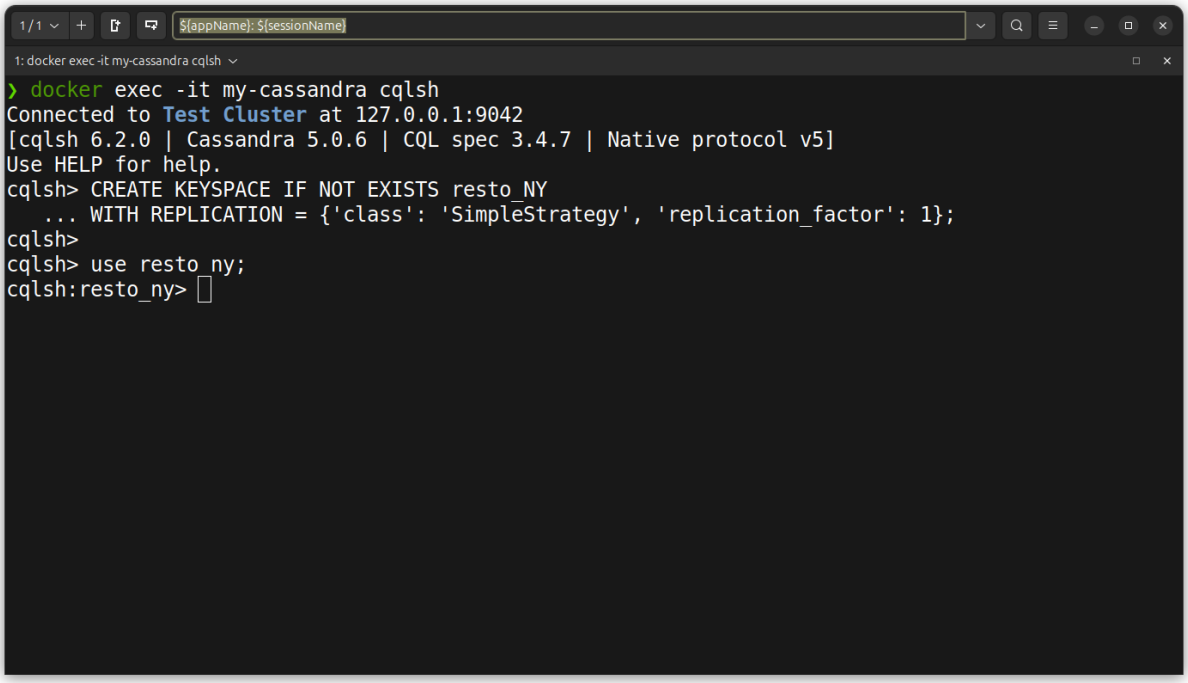

Figure 1: Creating the Keyspace

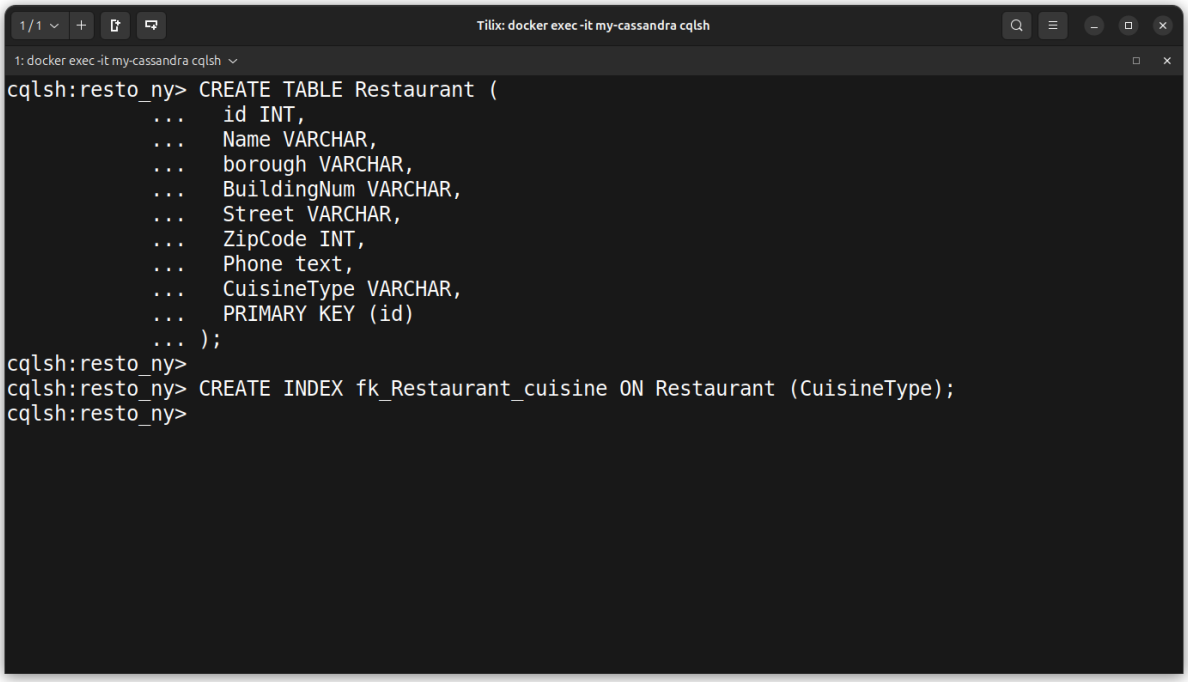

Figure 2: Creating Restaurant Table

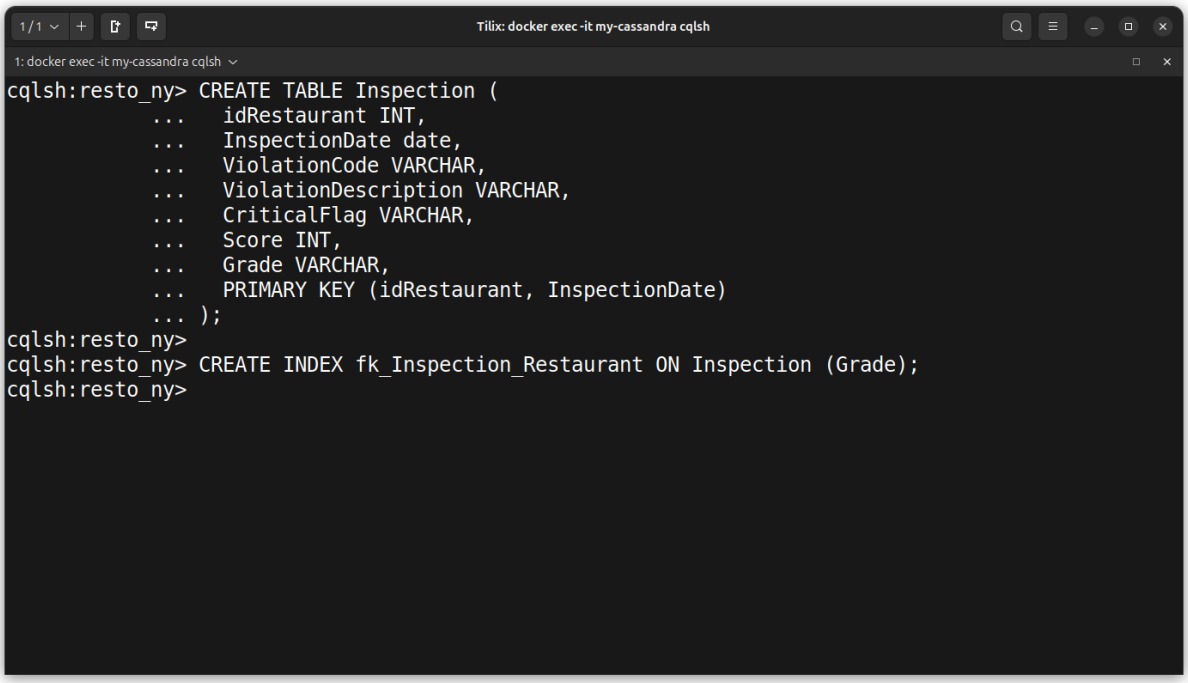

Figure 3: Creating Inspection Table

2.2 B. Data Import via Docker

Instruction: Copy the provided CSV files (‘restaurants.csv‘ and ‘restaurants$_inspections.csv$‘)$from the loca$
Execution Proof: The files were successfully copied to the container root.

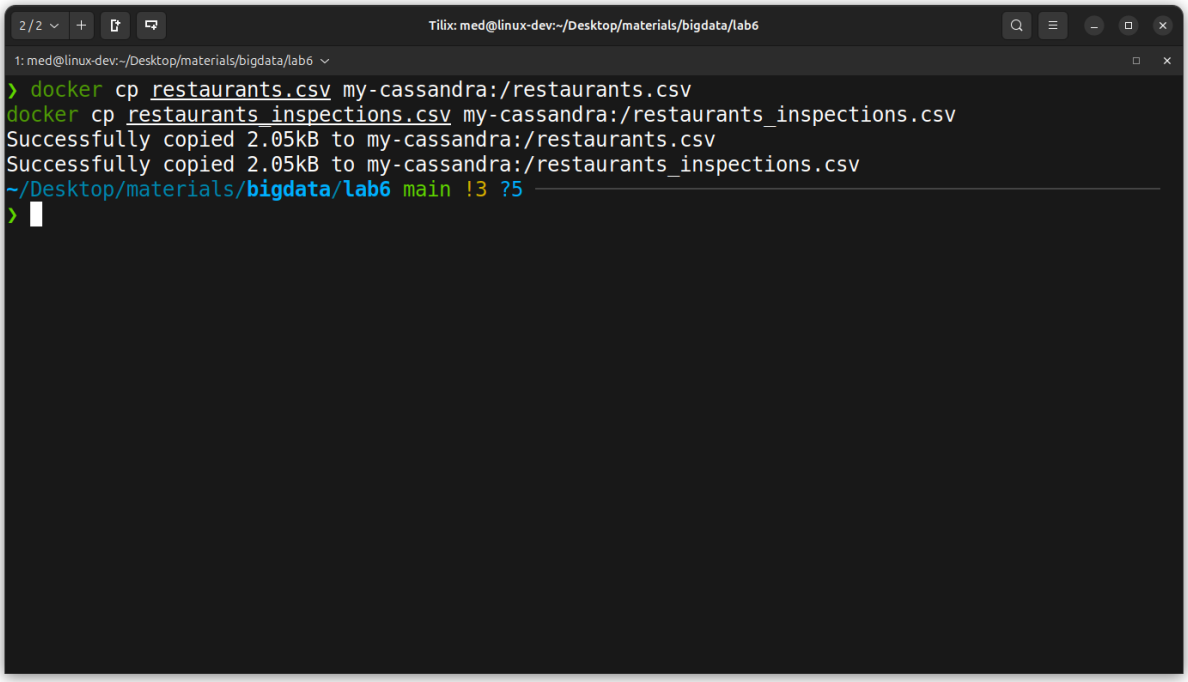

Figure 4: Copying CSV files to Docker container

2.3 C. CQL Queries & Results

The following section presents the execution of the 9 required queries.

1. List of all restaurants
Query: `SELECT * FROM Restaurant;`
Execution Proof: The system retrieves restaurant rows (displaying partial results).

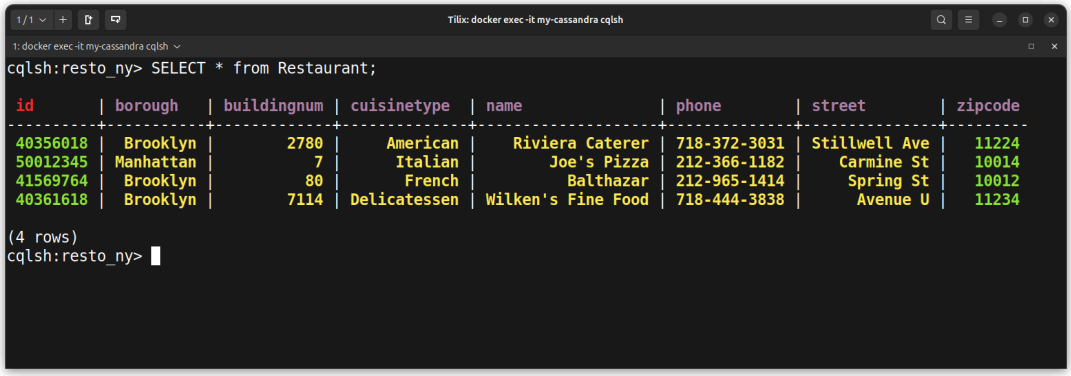

Figure 5: Listing all restaurants

2. List of restaurant names
Query: `SELECT Name FROM Restaurant;`
Execution Proof: Only the ‘Name‘ column is retrieved.

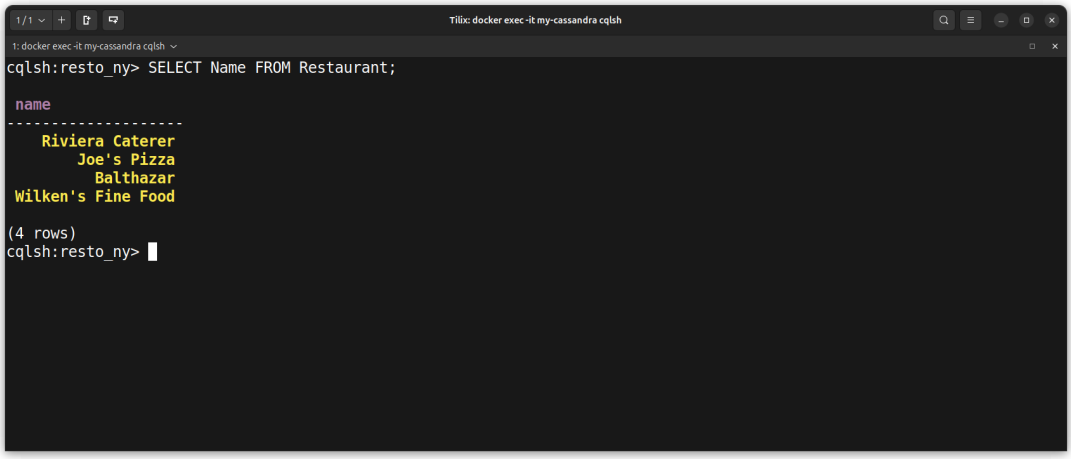

Figure 6: Listing restaurant names

3. Name and borough of restaurant ID 41569764
Query: `SELECT Name, borough FROM Restaurant WHERE id = 41569764;`
Execution Proof: The query uses the Primary Key ‘id‘ for a fast lookup.

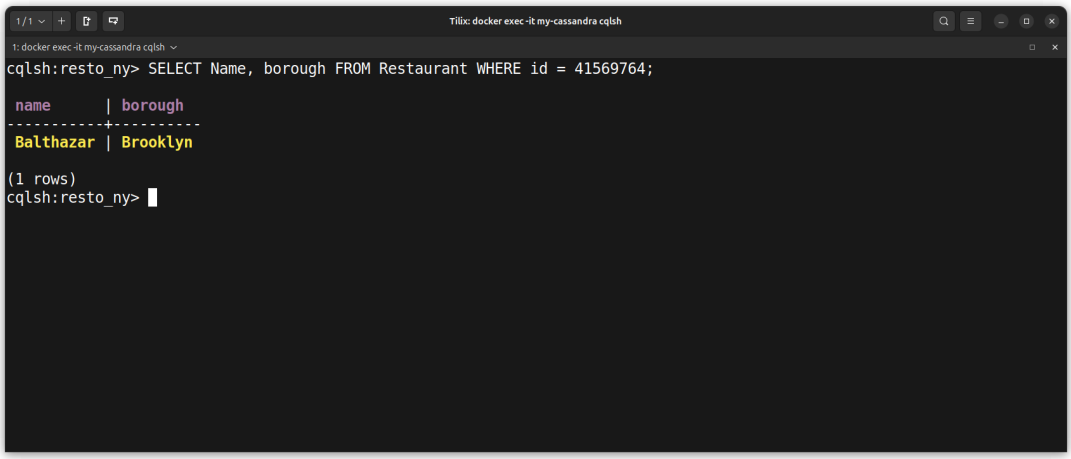

Figure 7: Specific restaurant details

4. Dates and grades of inspections for this restaurant
Query: `SELECT InspectionDate, Grade FROM Inspection WHERE idRestaurant = 41569764;`
Execution Proof: Retrieves inspection history associated with the restaurant ID.

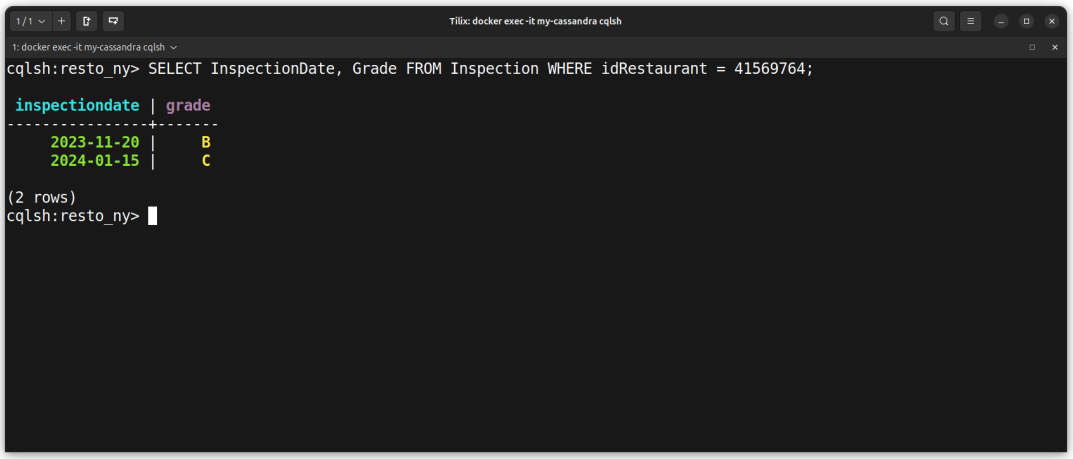

Figure 8: Inspection history

5. Restaurants with French cuisine
Query: `SELECT Name FROM Restaurant WHERE CuisineType = 'French';`
Execution Proof: This query works efficiently because we created a secondary index on 'CuisineType'.

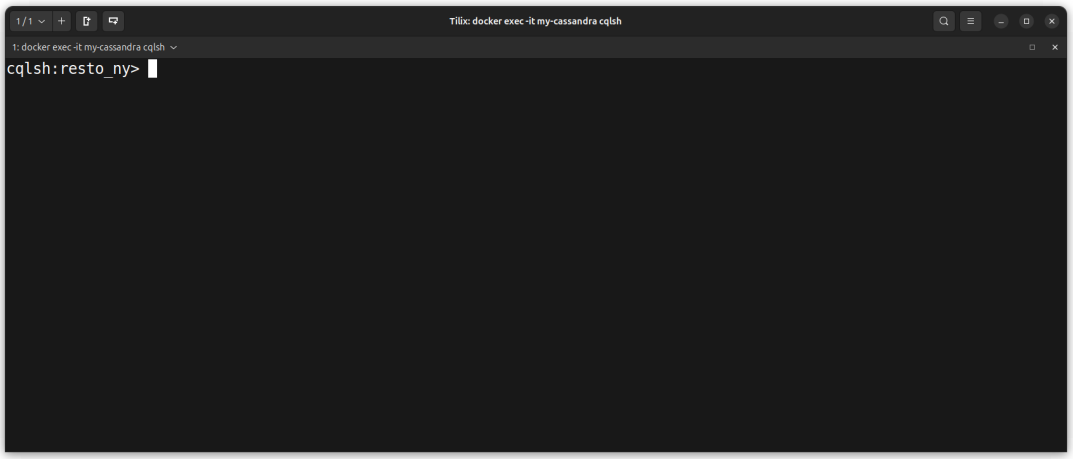

Figure 9: French Cuisine Restaurants

6. Restaurants in BROOKLYN (Allow Filtering)
Query: `SELECT Name FROM Restaurant WHERE borough = 'BROOKLYN' ALLOW FILTERING;`
Execution Proof: Since 'borough' is not indexed, 'ALLOW FILTERING' is required.

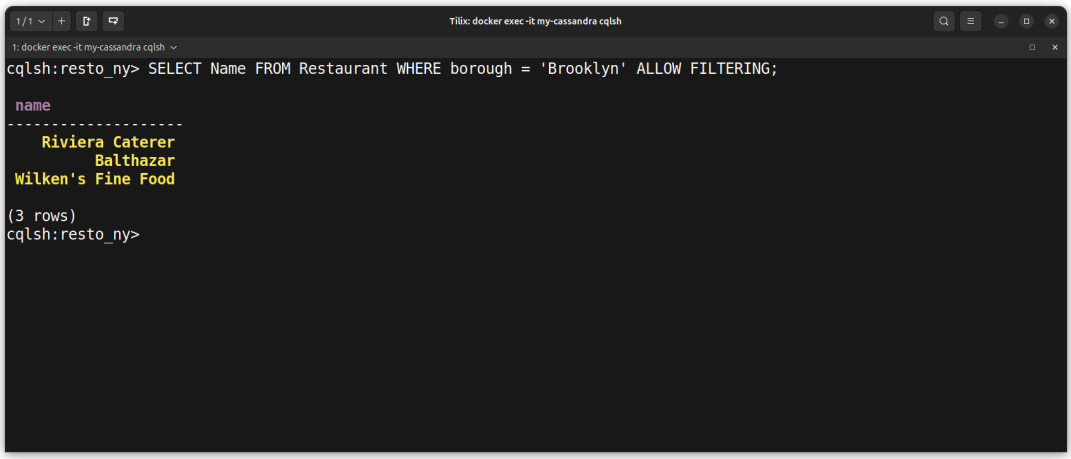

Figure 10: Brooklyn Restaurants

7. Grades for restaurant 41569764 with score 10
Query: `SELECT Grade, Score FROM Inspection WHERE idRestaurant = 41569764 AND Score >= 10 ALLOW FILTERING;`
Execution Proof: Filters inspections by both ID and score.

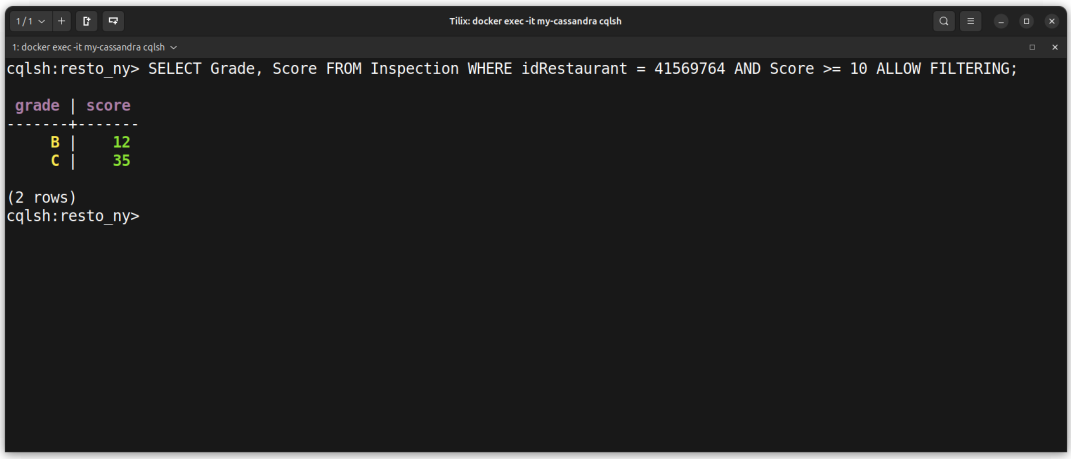

Figure 11: High scoring inspections

8. Grades where score ¿ 30
Query: `SELECT Grade FROM Inspection WHERE Score > 30 ALLOW FILTERING;`
Execution Proof: Scans the table for high-risk inspections.

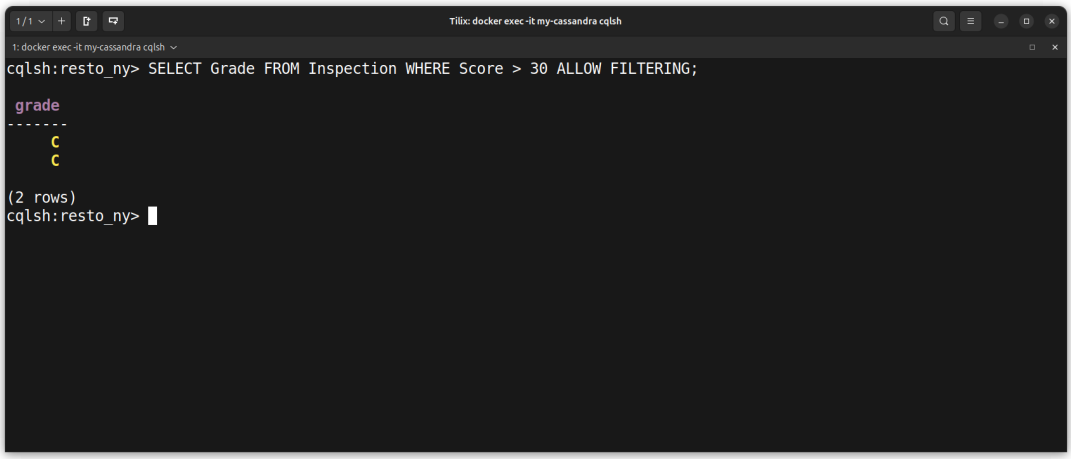

Figure 12: Inspections with Score ¿ 30

9. Count of rows from previous query
Query: `SELECT COUNT(*) FROM Inspection WHERE Score > 30 ALLOW FILTERING;`
Execution Proof: Returns the number of matching inspections.

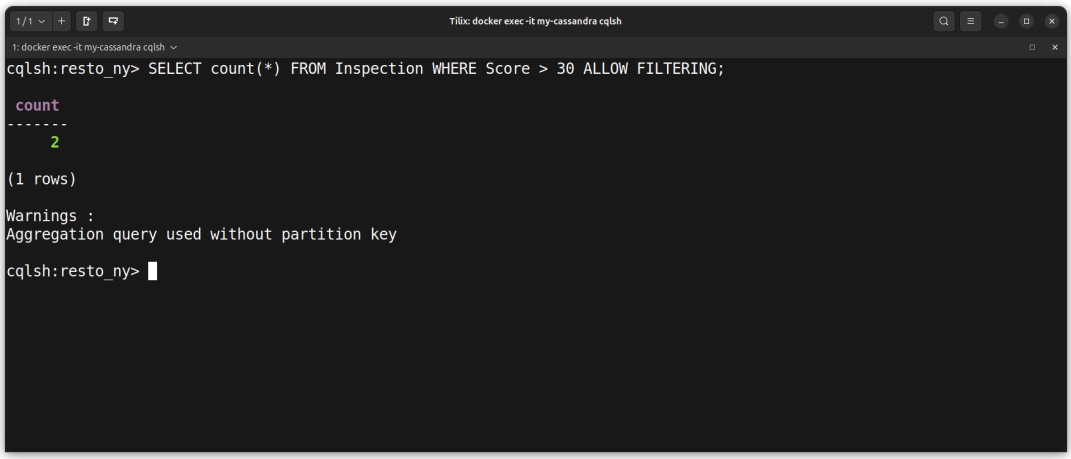

Figure 13: Count result