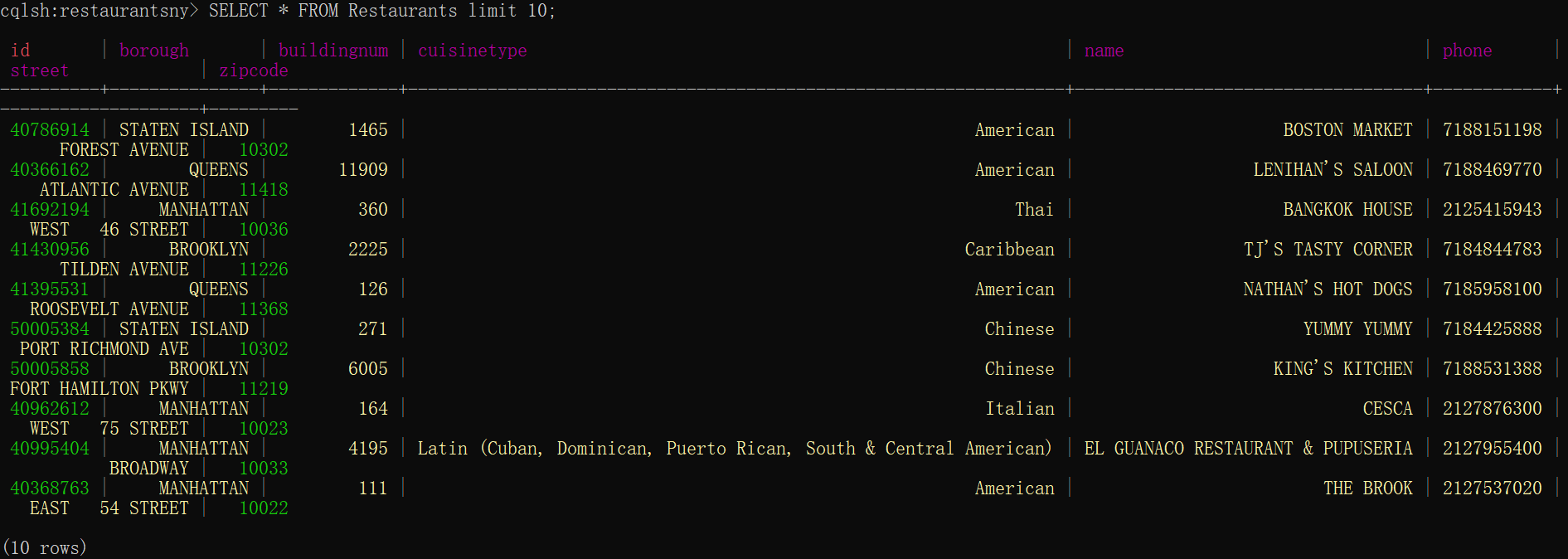
**Querying**

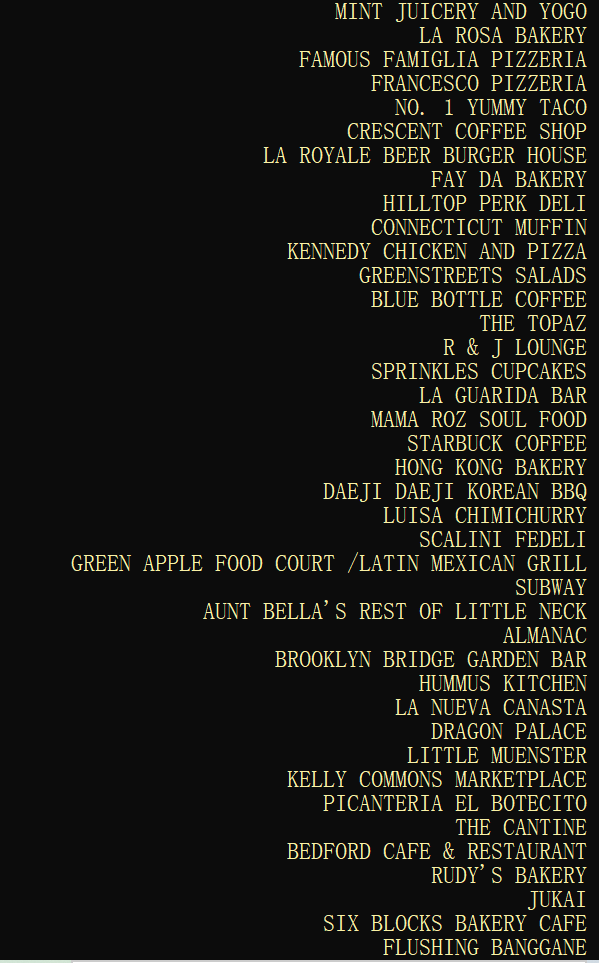
1.

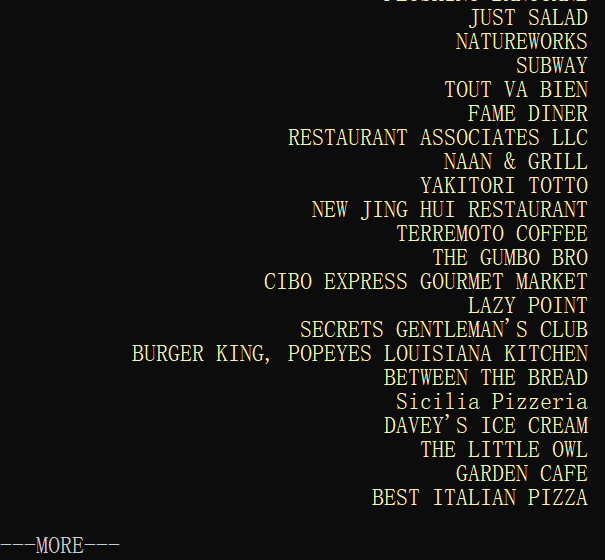
SELECT \* FROM Restaurants limit 10;



2.

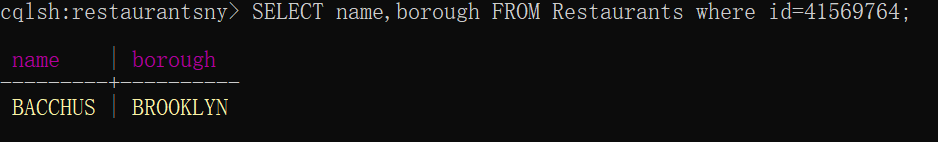
SELECT name FROM Restaurants;





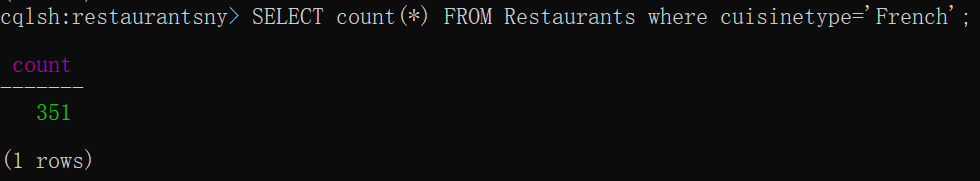
3.

SELECT name,borough FROM Restaurants where id=41569764;



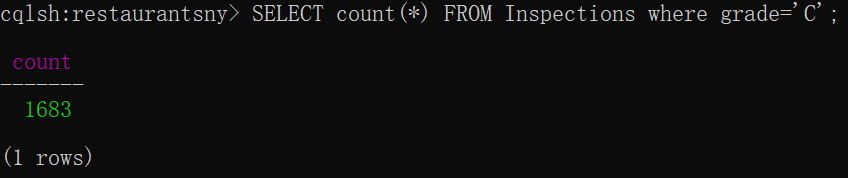
4.

SELECT count(\*) FROM Restaurants where cuisinetype='French';



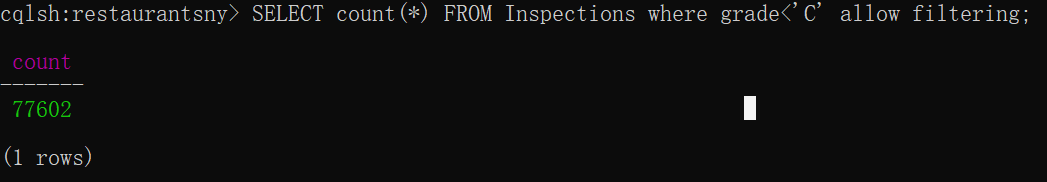
5.

SELECT count(\*) FROM Inspections where grade='C';



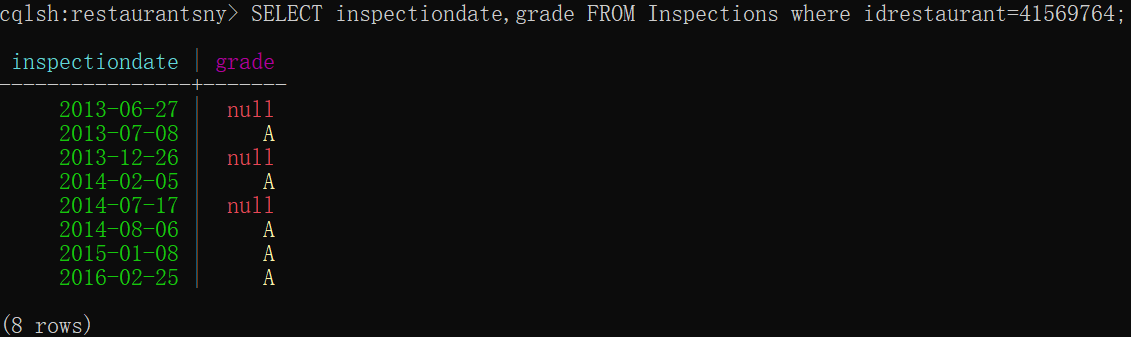
6.

SELECT count(\*) FROM Inspections where grade<'C' allow filtering;



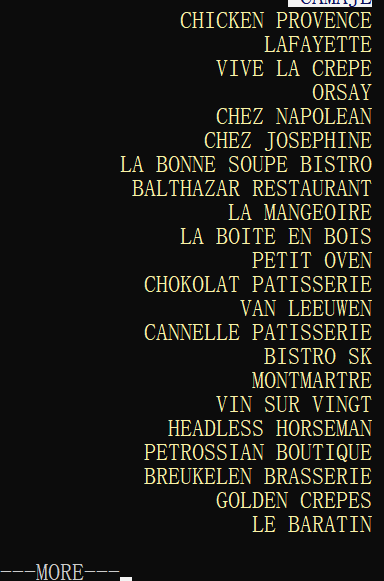
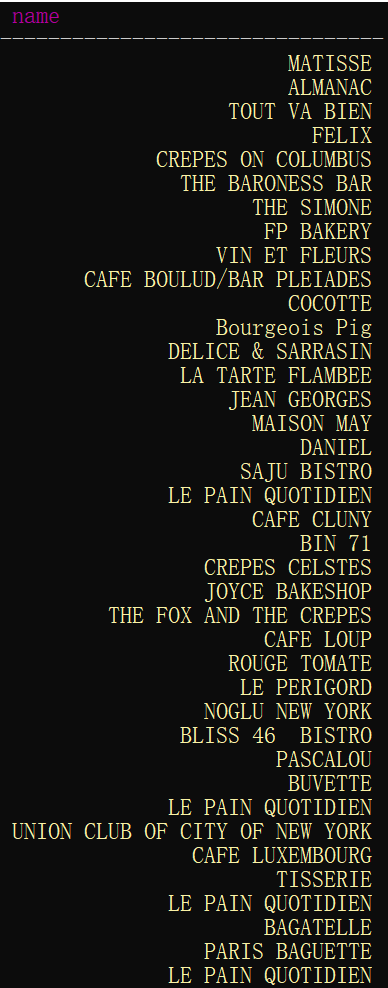
7.

SELECT inspectiondate,grade FROM Inspections where idrestaurant=41569764;



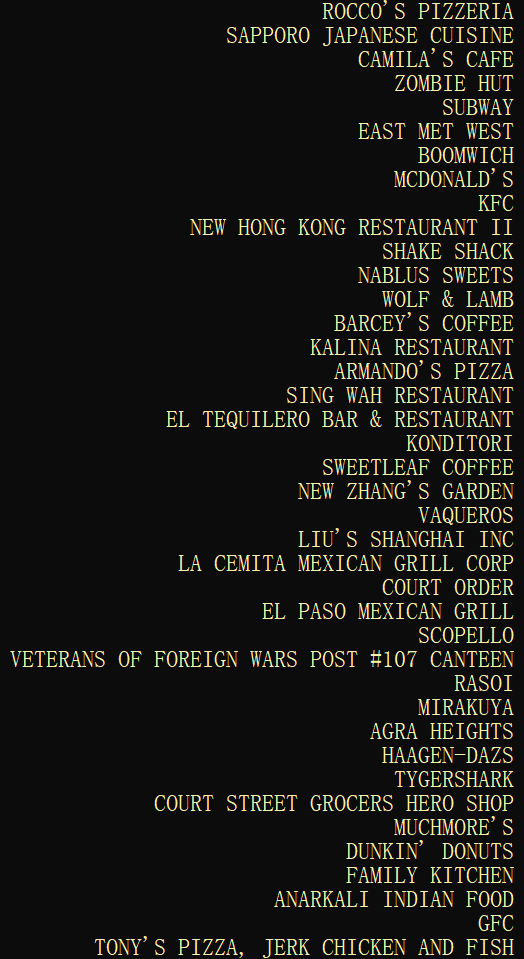
8.

SELECT name FROM Restaurants where cuisinetype='French';



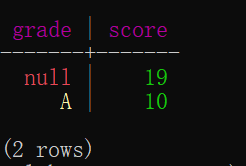
9.

SELECT name FROM Restaurants where borough='BROOKLYN' ALLOW FILTERING;



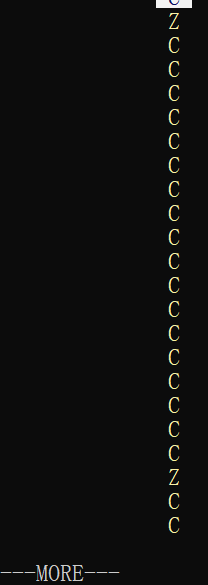
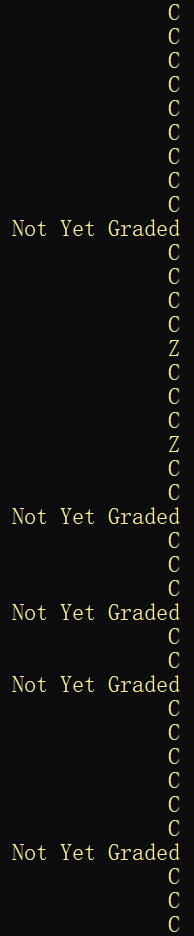
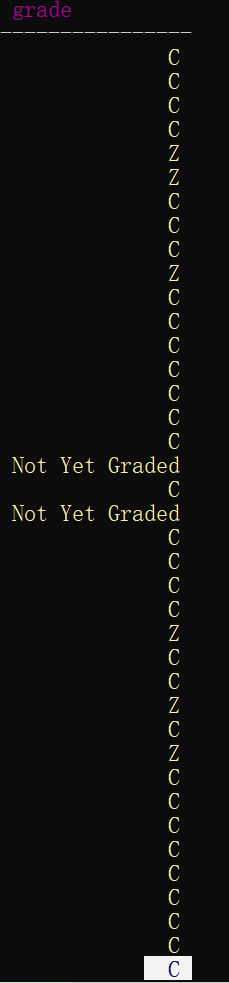
10.

SELECT grade,score FROM Inspections where idrestaurant=41569764 and score>=10 ALLOW FILTERING;



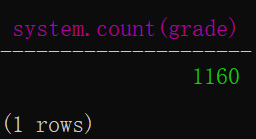
11.

SELECT grade FROM Inspections where score>30 and grade>' ' ALLOW FILTERING;



12.

SELECT count(grade) FROM Inspections where score>30 and grade>' ' ALLOW FILTERING;



13.

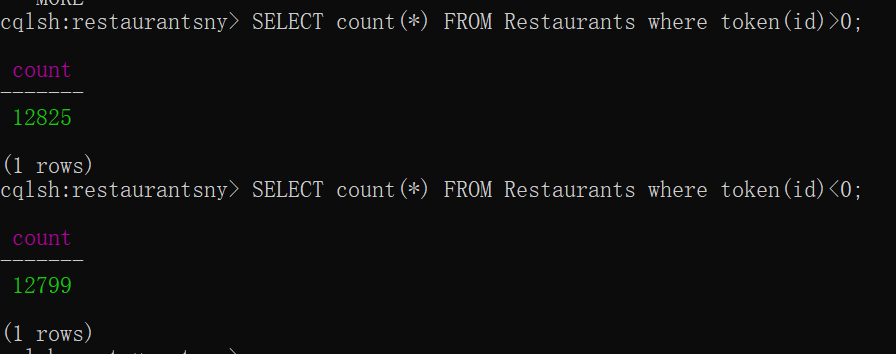
SELECT name FROM Restaurants where token(id)>0;



14.

SELECT count(\*) FROM Restaurants where token(id)>0;

SELECT count(\*) FROM Restaurants where token(id)<0;



Rows are evenly distributed in different Partition, hash function plays a good position.

**Additional queries**

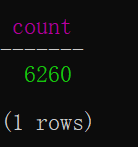
1.

CREATE INDEX ix\_Restaurant\_borough ON Restaurants (Borough);

SELECT Name FROM Restaurants WHERE borough='BROOKLYN' ;

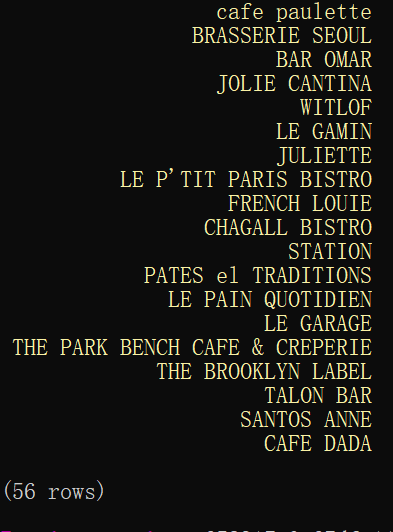


SELECT count(\*) FROM Restaurants WHERE borough='BROOKLYN' ;



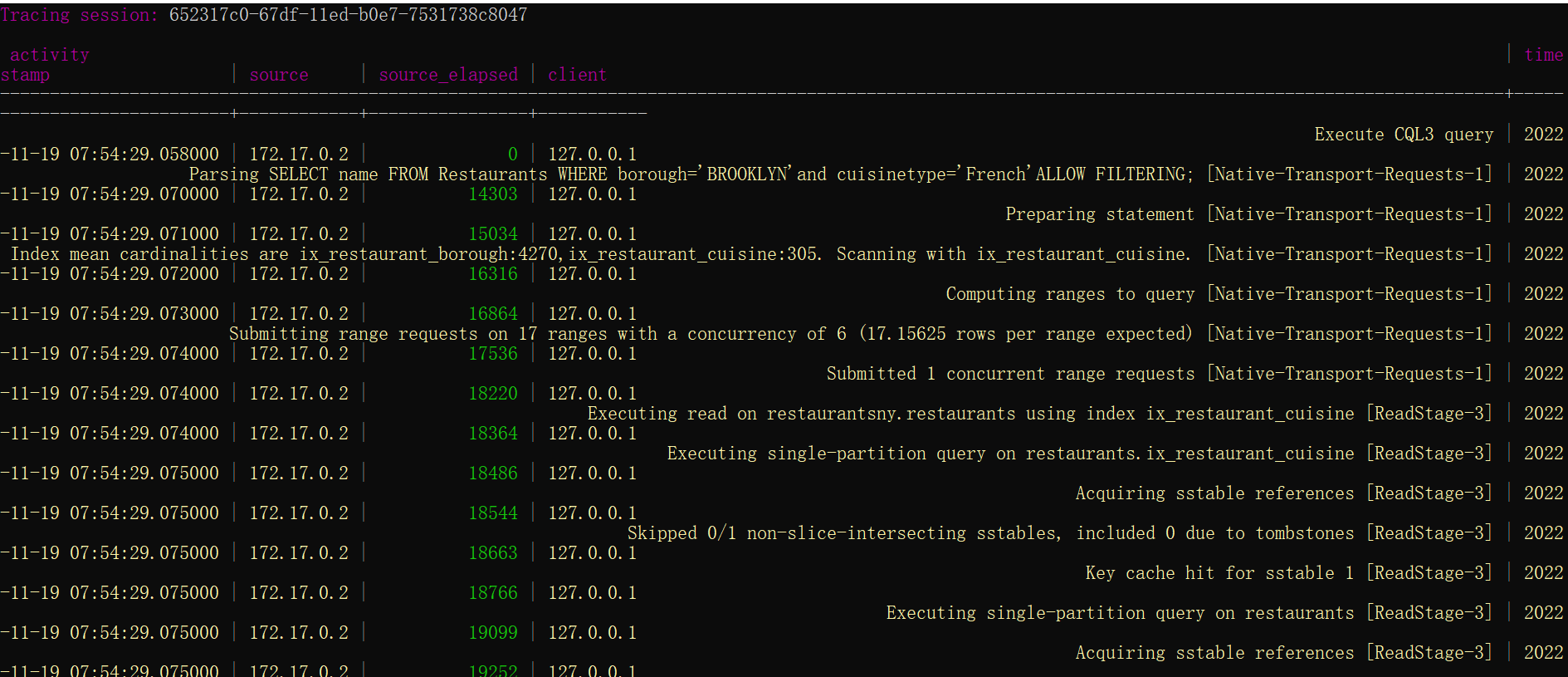
2.

SELECT name FROM Restaurants WHERE borough='BROOKLYN' and cuisinetype='French' ALLOW FILTERING;



3.

Both index on borough and cuisinetype are used.



4.

No, Cassandra does not support joins (NoSQL heritage).

2.2

**First access pattern**

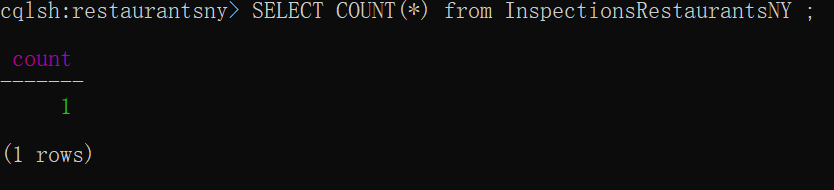
1.

CREATE TYPE restaurants( Name VARCHAR, borough VARCHAR, BuildingNum VARCHAR, Street VARCHAR, ZipCode INT, Phone text, CuisineType VARCHAR);

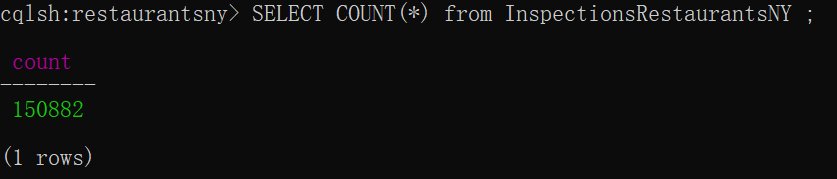
CREATE TABLE InspectionsRestaurantsNY(idRestaurant INT,restaurant frozen<restaurants>,InspectionDate date, ViolationCode VARCHAR, ViolationDescription VARCHAR, CriticalFlag VARCHAR, Score INT, GRADE VARCHAR,primary key(idRestaurant, InspectionDate));

2.

INSERT INTO InspectionsRestaurantsNY JSON '{"idRestaurant":1111, "restaurant": {"name": "IHOP", "borough": "BRONX", "buildingnum": "5655", "street": "BROADWAY", "zipcode":"10463", "phone":"7185494565", "cuisineType": "American"}, "inspectionDate": "2016-08-16", "violationCode": "04L", "violationDescription": "testtest", "criticalFlag": "Critical", "score":15, "grade": ""}';

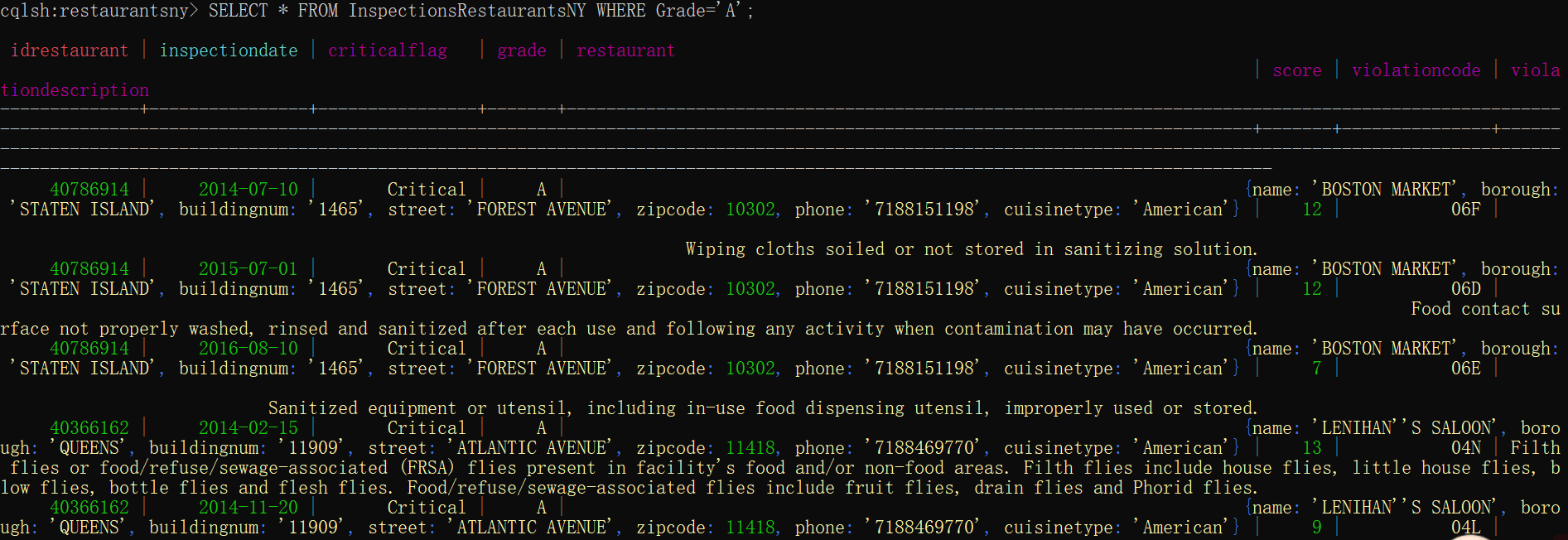


3.



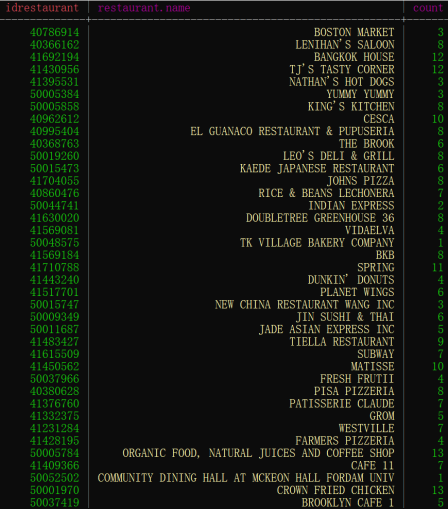
CREATE INDEX ix\_Inspections\_Restaurant ON InspectionsRestaurantsNY(Grade);

SELECT \* FROM InspectionsRestaurantsNY WHERE Grade='A' GROUP BY idRestaurant;



5.

SELECT idRestaurant,restaurant.name,COUNT(\*) FROM InspectionsRestaurantsNY GROUP BY idRestaurant;



**Second access pattern**

1.

No, Frozen point means point type is frozen, we can't partially provide point value, we have to provide the full value of point.

2.

CREATE TABLE InspectionsRestaurantsMap(id INT, Name VARCHAR, borough VARCHAR, BuildingNum VARCHAR, Street VARCHAR, ZipCode INT, Phone text, CuisineType VARCHAR, Inspections map<VARCHAR, frozen<Inspection>>, primary key(id));

4.

INSERT INTO InspectionsRestaurantsMap JSON '{"id":50041578,"name": "IHOP", "borough": "BRONX", "buildingnum": "5655", "street": "BROADWAY", "zipcode":"10463", "phone":"7185494565", "cuisineType": "American", "Inspections":{"BRONX":{"inspectionDate": "2016-08-16", "violationCode": "04L", "violationDescription": "Evidence of mice or live mice present or non-food areas.", "criticalFlag": "Critical", "score":15, "grade": "A"}}}';

5.

SELECT \* FROM InspectionsRestaurantsMap WHERE inspections contains KEY 'BRONX'ALLOW FILTERING;

