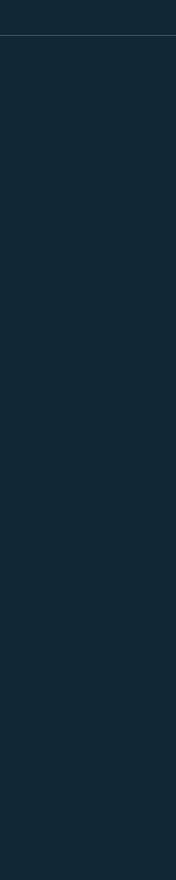
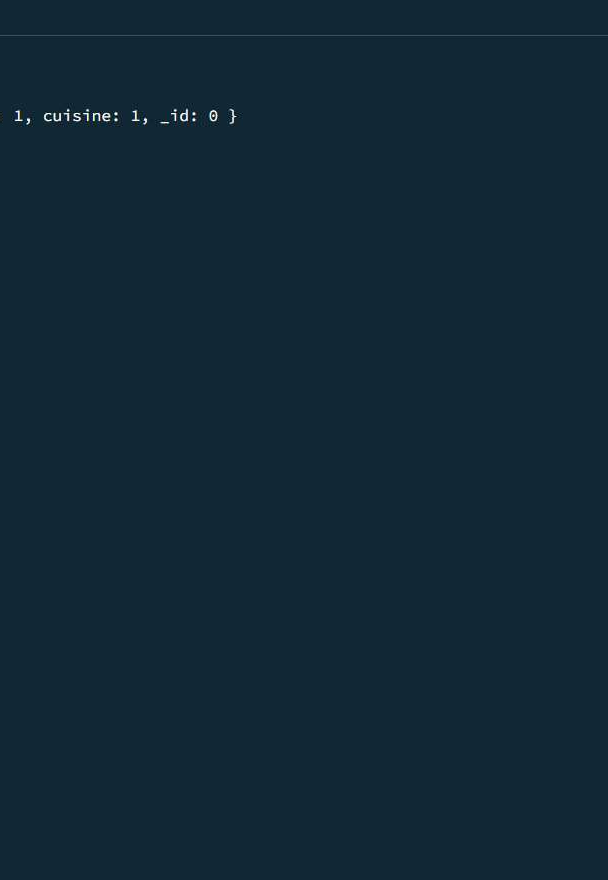
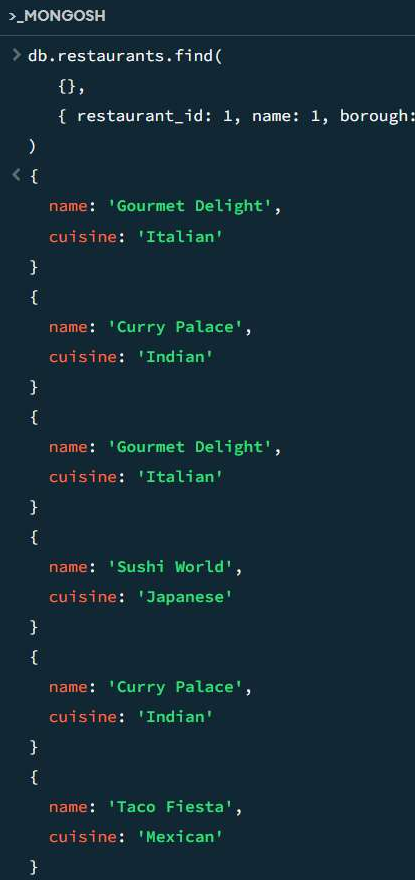


**Cycle Sheet - III**

**Write mongodb queries to::**

1 Display all the documents in the collection restaurants

db.restaurants.find().pretty()



1. Display the fields restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant.

db.restaurants.find(

{},

{ restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }

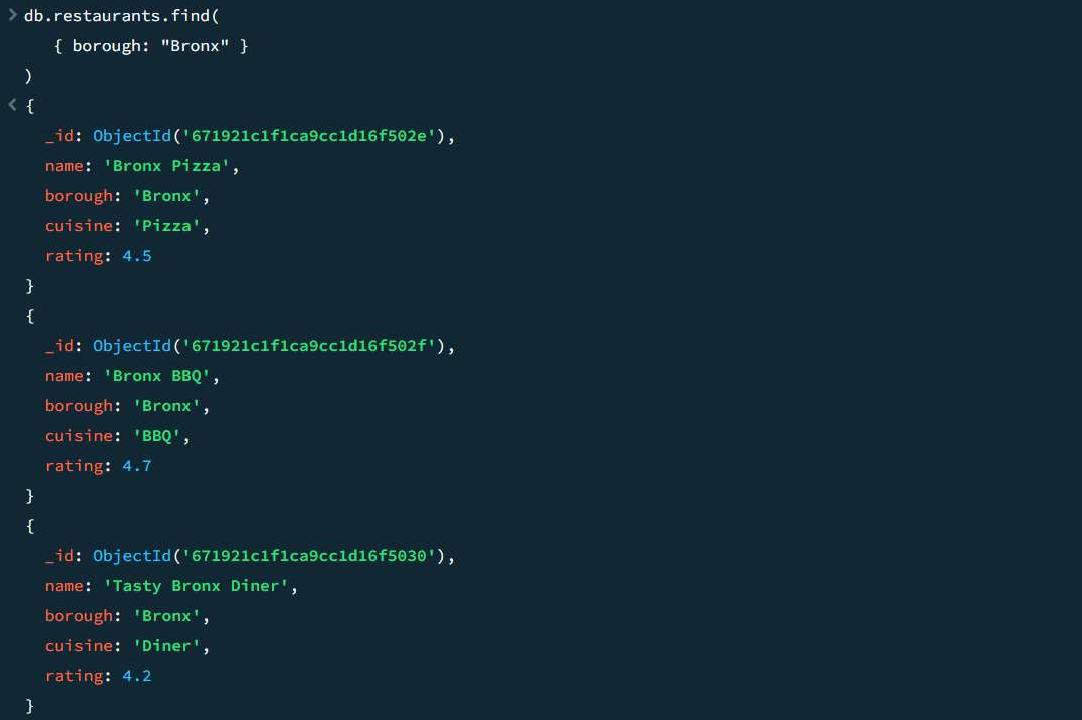
)

1. Display all the restaurant which is in the borough Bronx.

db.restaurants.find(

{ borough: "Bronx" }

)

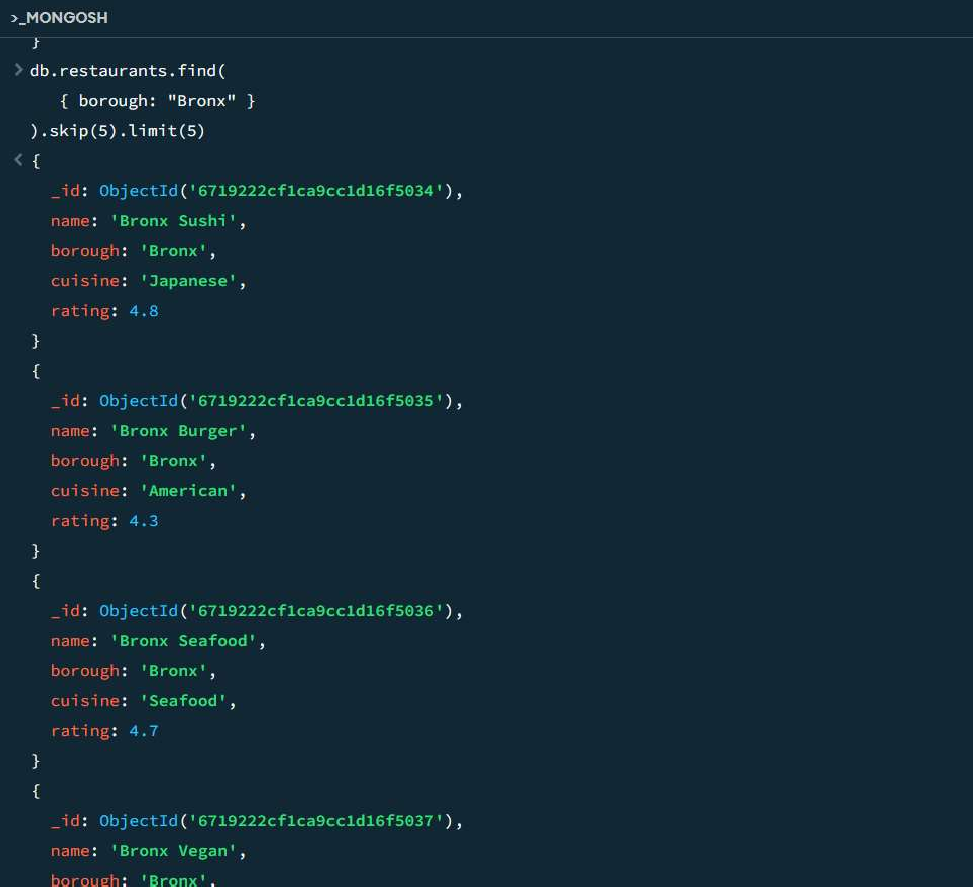


1. Display the next 5 restaurants after skipping first 5 which are in the borough Bronx.

db.restaurants.find(

{ borough: "Bronx" }

).skip(5).limit(5)

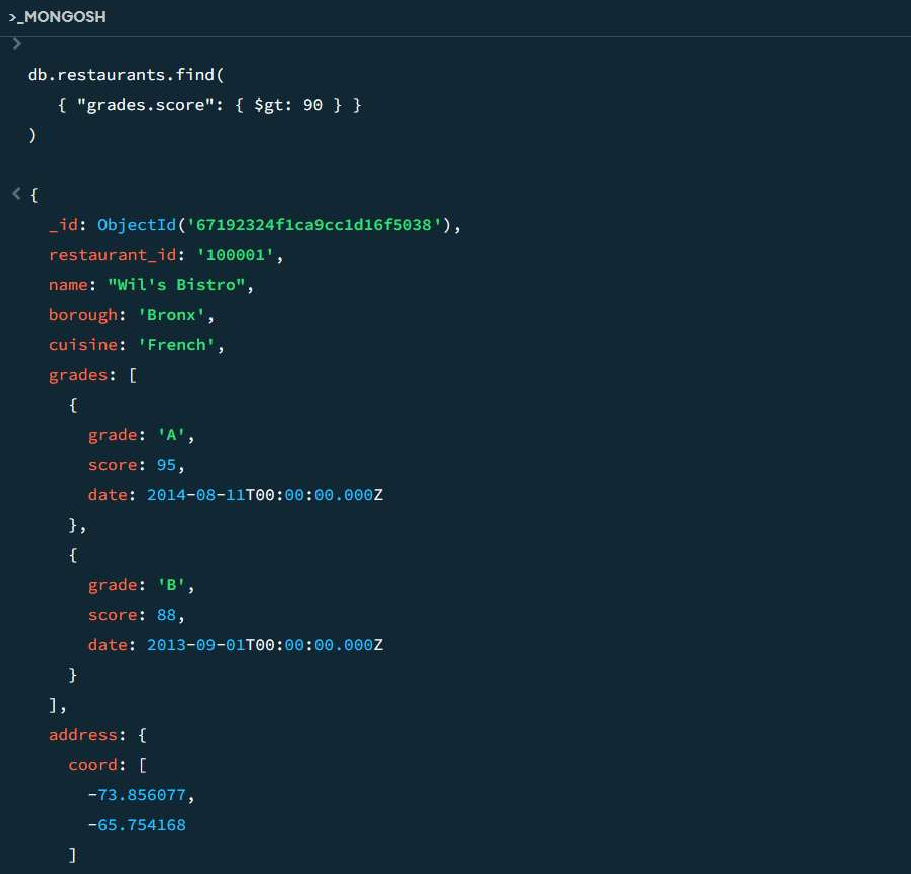


1. Find the restaurants who achieved a score more than 90.

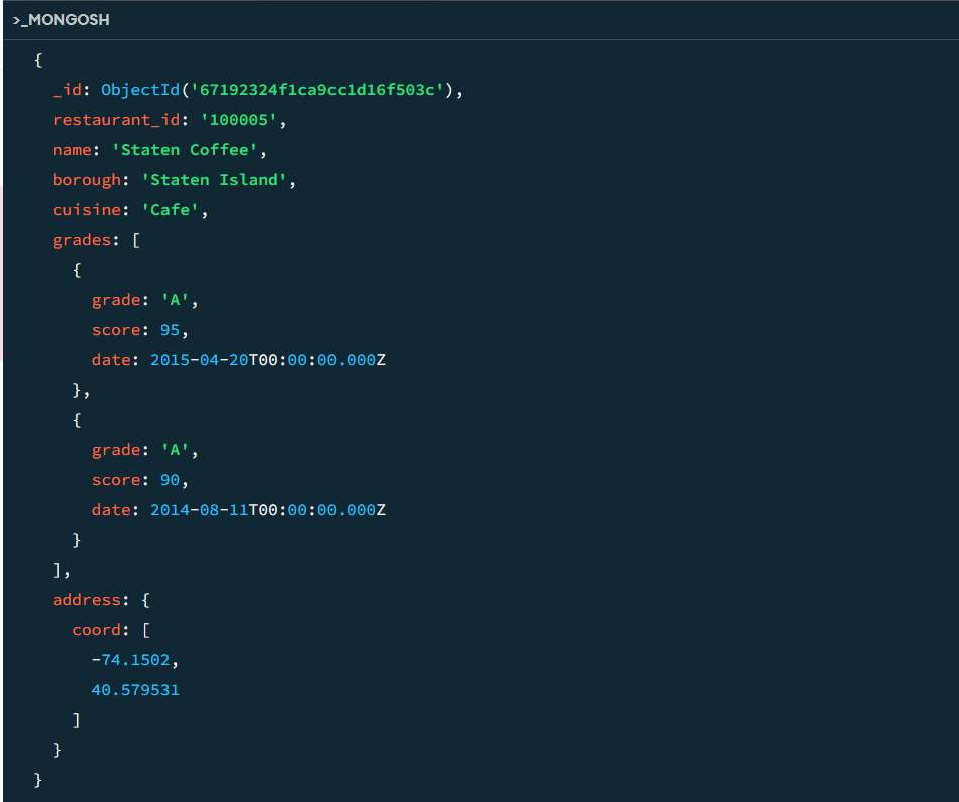
db.restaurants.find(

{ "grades.score": { $gt: 90 } }

)



# []



1. Find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.

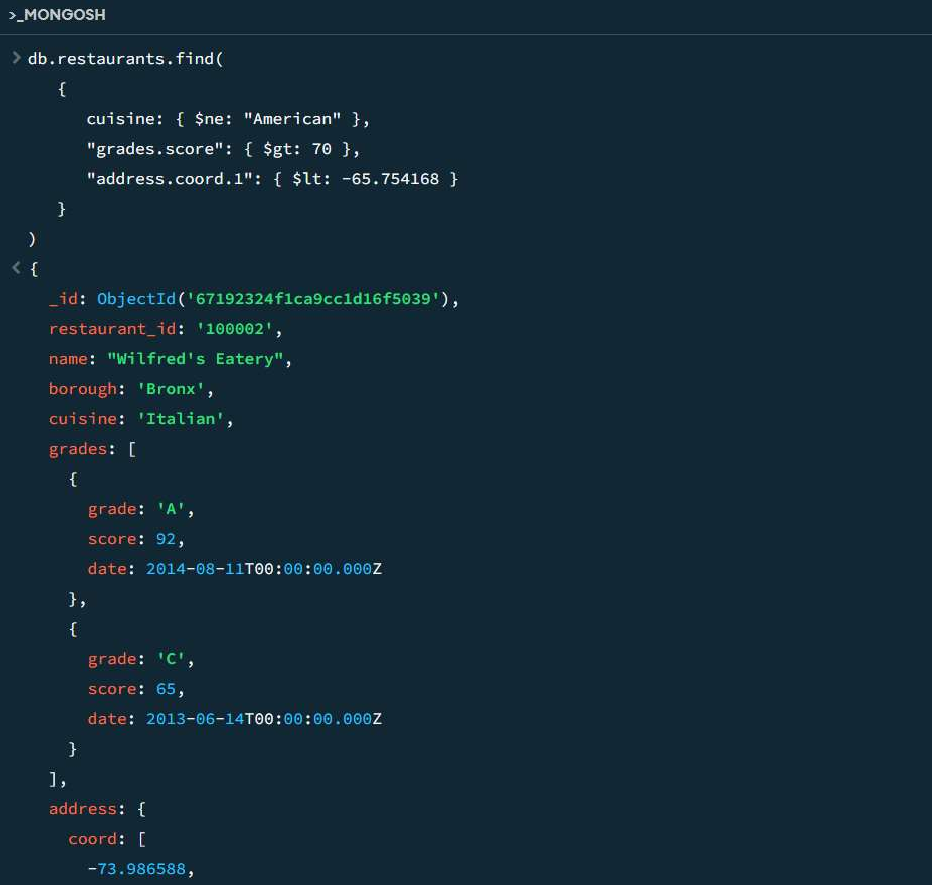
db.restaurants.find(

{

cuisine: { $ne: "American" }, "grades.score": { $gt: 70 }, "address.coord.1": { $lt: -65.754168 }

}

)



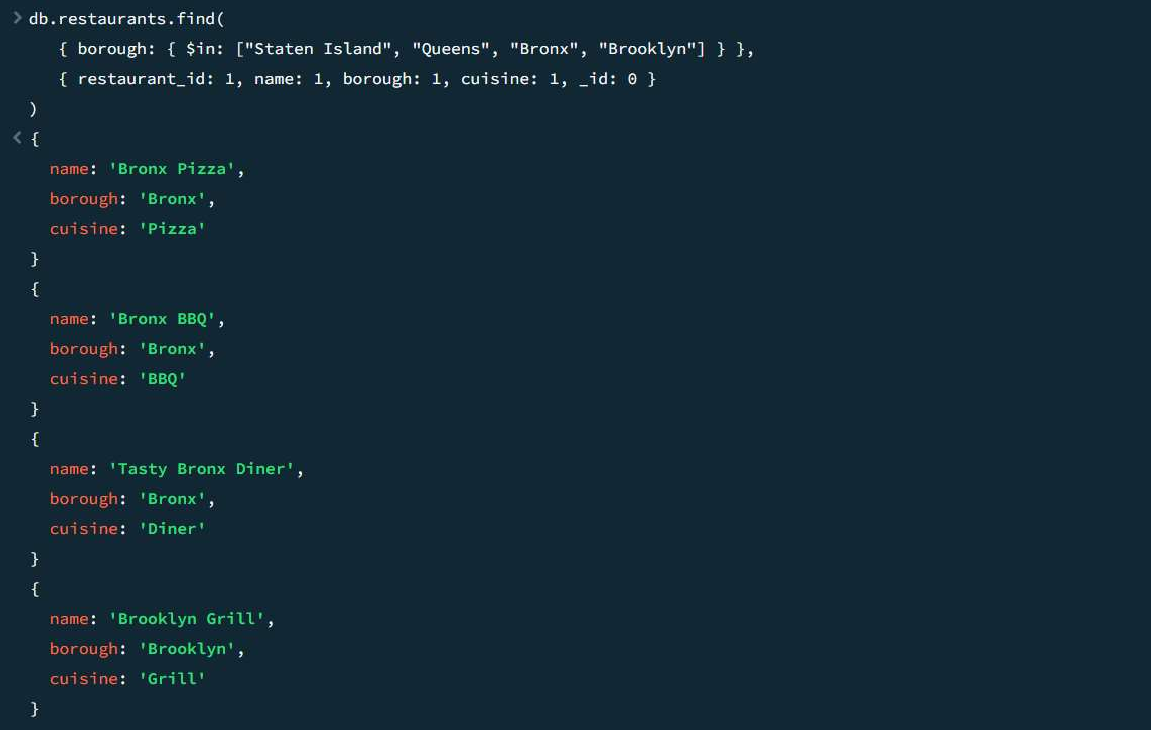
1. Find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.

db.restaurants.find(

{ name: { $regex: /^Wil/ } },

{ restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }

)



**]**

1. Find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronxor Brooklyn.

db.restaurants.find(

{ borough: { $in: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } },

{ restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0 }

)

1. Find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an isodate "2014-08-11T00:00:00Z" among many of survey dates.

**[]**

db.restaurants.find(

{

"grades.grade": "A", "grades.score": 11,

"grades.date": ISODate("2014-08-11T00:00:00Z")

},

{ restaurant\_id: 1, name: 1, grades: 1, \_id: 0 }

)

1. Arrange the name of the restaurants in descending along with all the columns. db.restaurants.find().sort({ name: -1 })



1. Find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan.

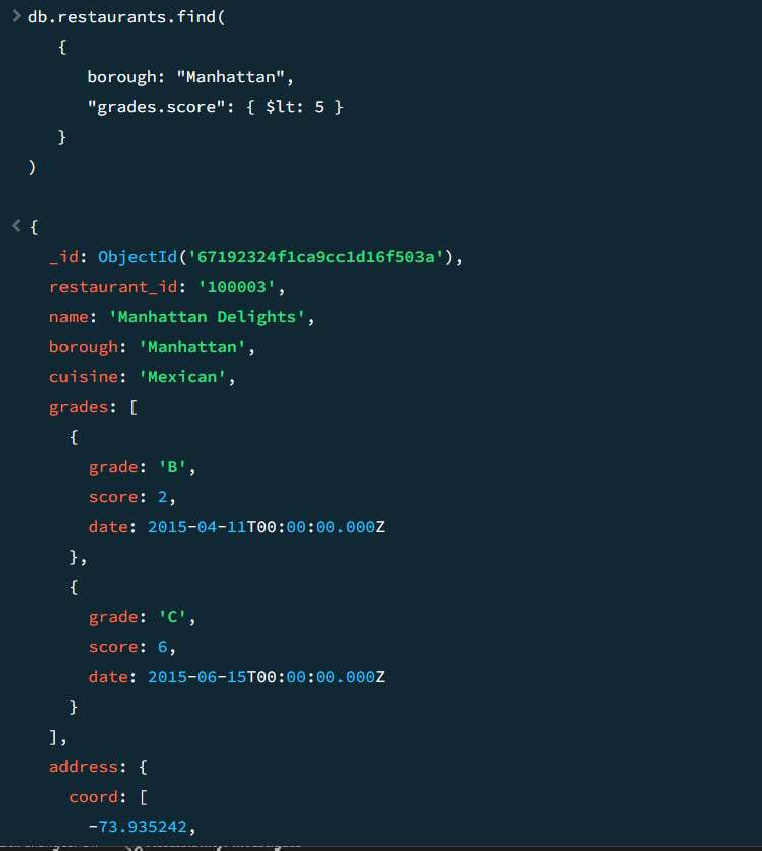
db.restaurants.find(

{

borough: "Manhattan", "grades.score": { $lt: 5 }

}

)



1. Find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

db.restaurants.find(

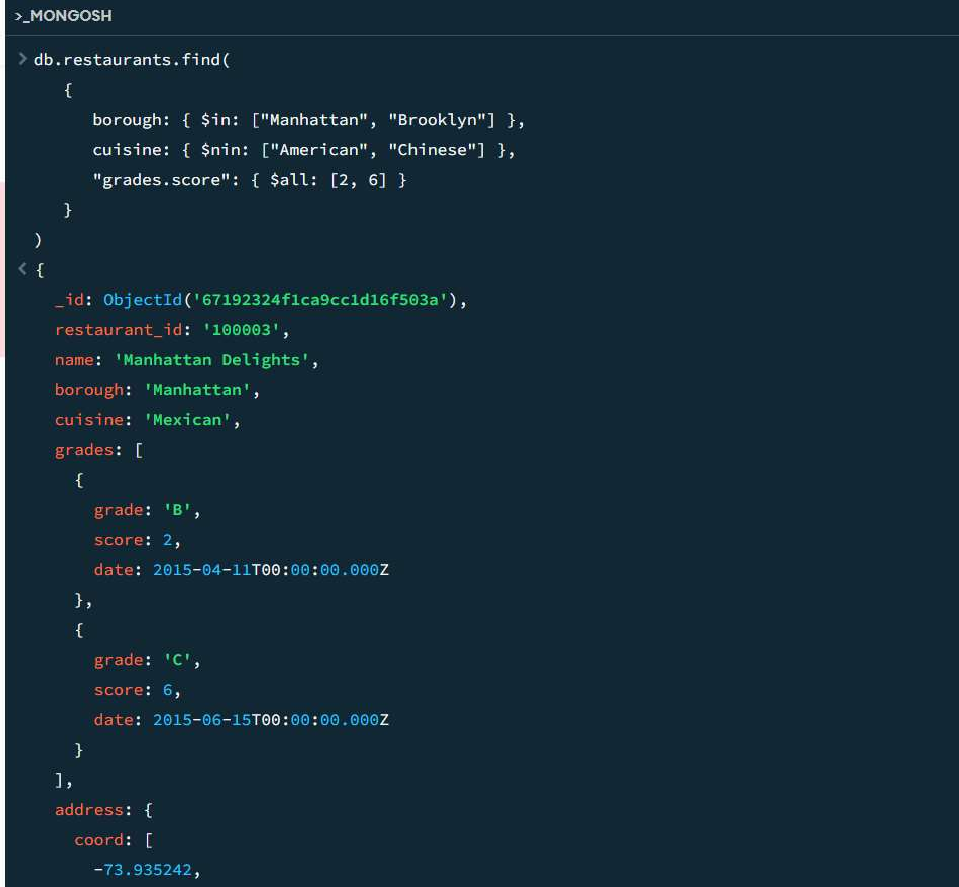
{

borough: { $in: ["Manhattan", "Brooklyn"] },

cuisine: { $nin: ["American", "Chinese"] },

"grades.score": { $all: [2, 6] }

}



)

1. Find the restaurants that have all grades with a score greater than 5 and are located in the borough of Manhattan or Brooklyn.

db.restaurants.find(

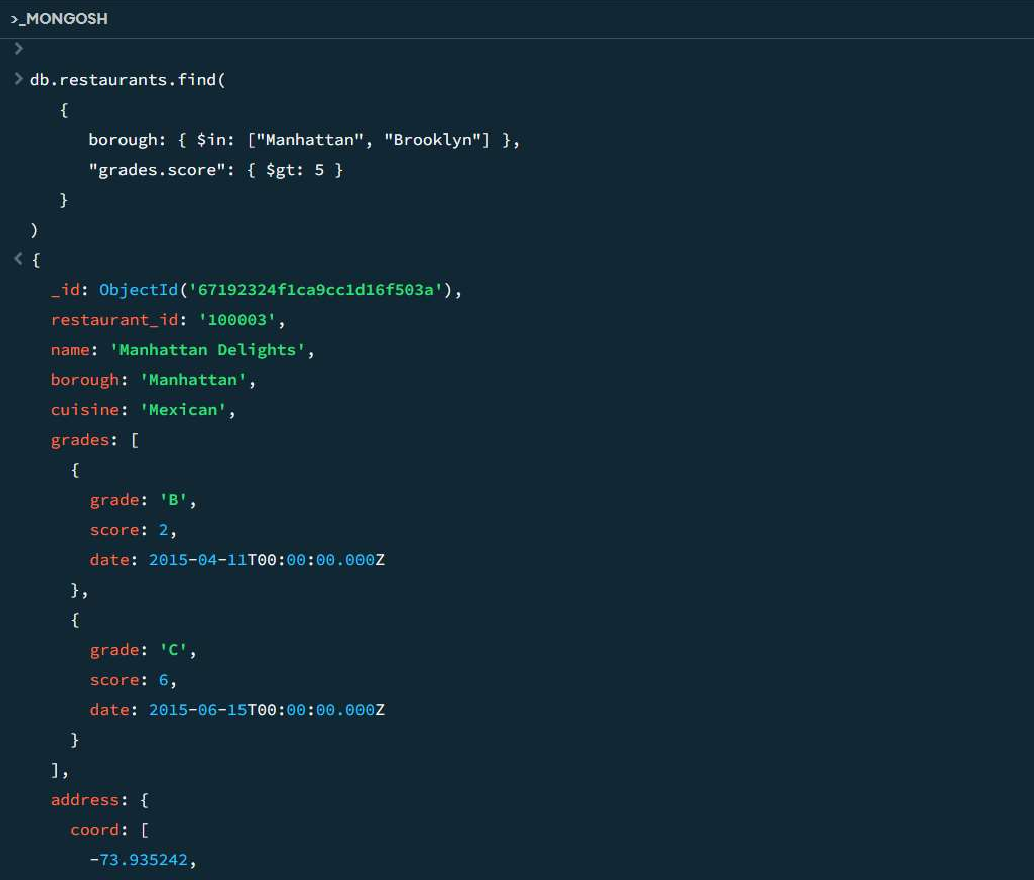
{

borough: { $in: ["Manhattan", "Brooklyn"] },

"grades.score": { $gt: 5 }

}

)



1. Find the average score for each restaurant.

db.restaurants.aggregate([

{

$unwind: "$grades"

},

{

$group: {

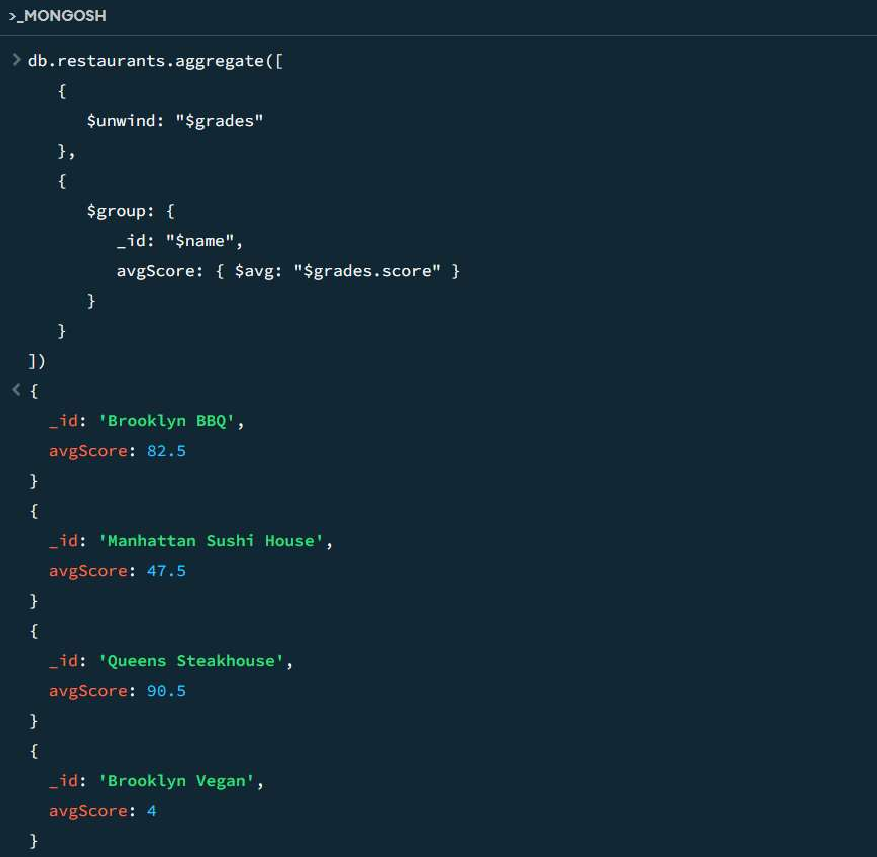
\_id: "$name",

avgScore: { $avg: "$grades.score" }

}

}

])



1. Find the highest score for each restaurant.

db.restaurants.aggregate([

{

$unwind: "$grades"

},

{

$group: {

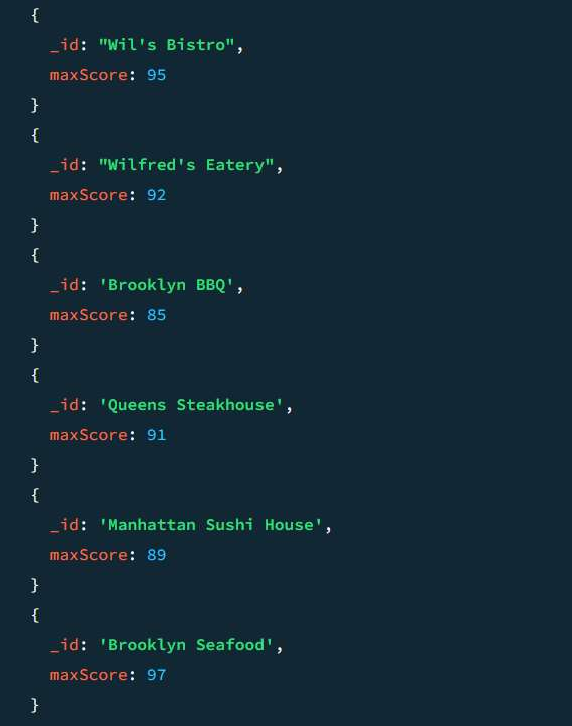
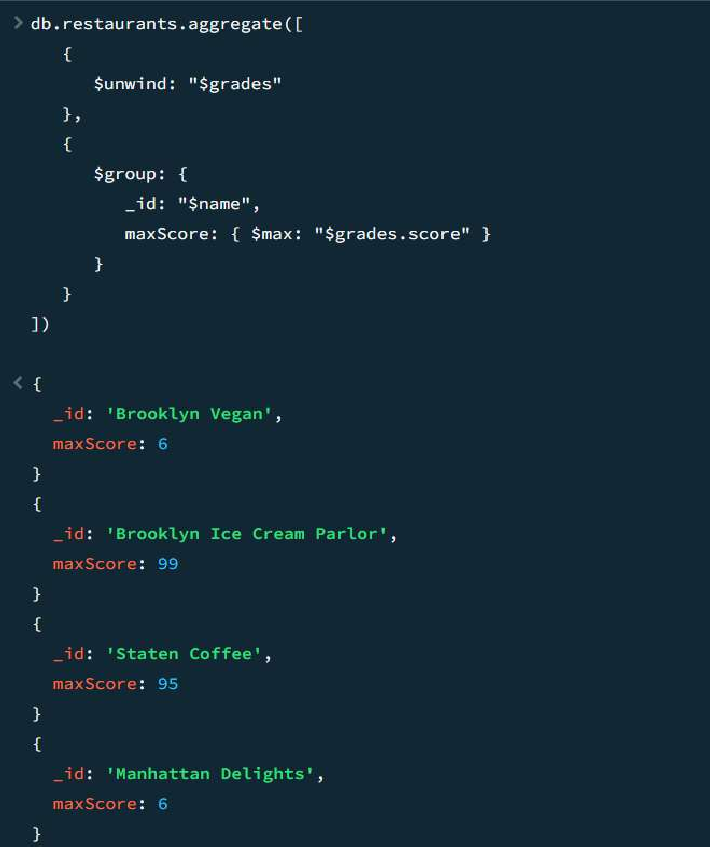
\_id: "$name",

maxScore: { $max: "$grades.score" }

}

}

])



**[**

1. Find the lowest score for each restaurant.

db.restaurants.aggregate([

{

$unwind: "$grades"

},

{

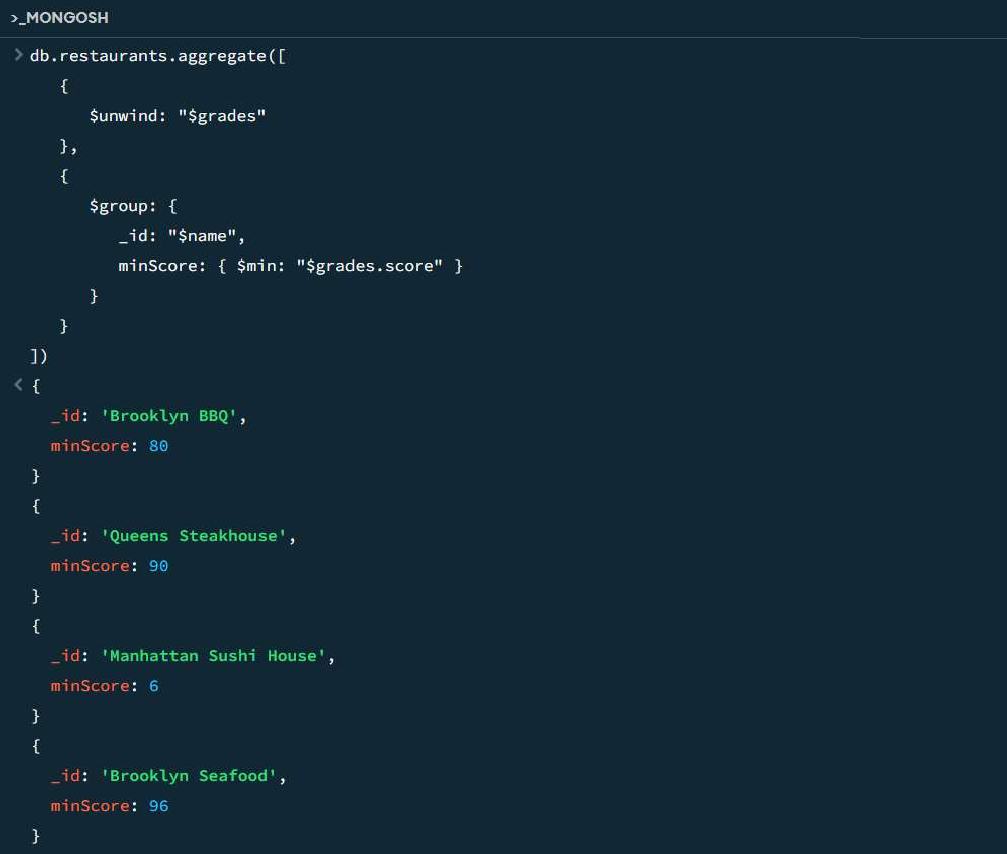
$group: {

\_id: "$name",

minScore: { $min: "$grades.score" }

}

}



# ]

])

1. Find the count of restaurants in each borough.

db.restaurants.aggregate([

{

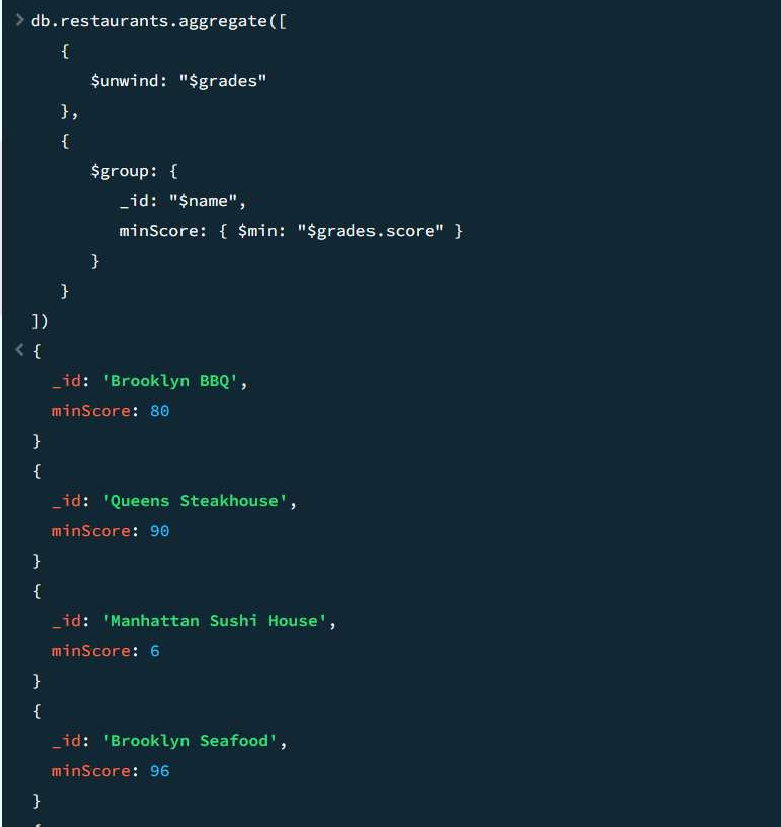
$group: {

\_id: "$borough", count: { $sum: 1 }

}

}

])



# []

1. Find the count of restaurants that received a grade of 'A' for each borough.

db.restaurants.aggregate([

{

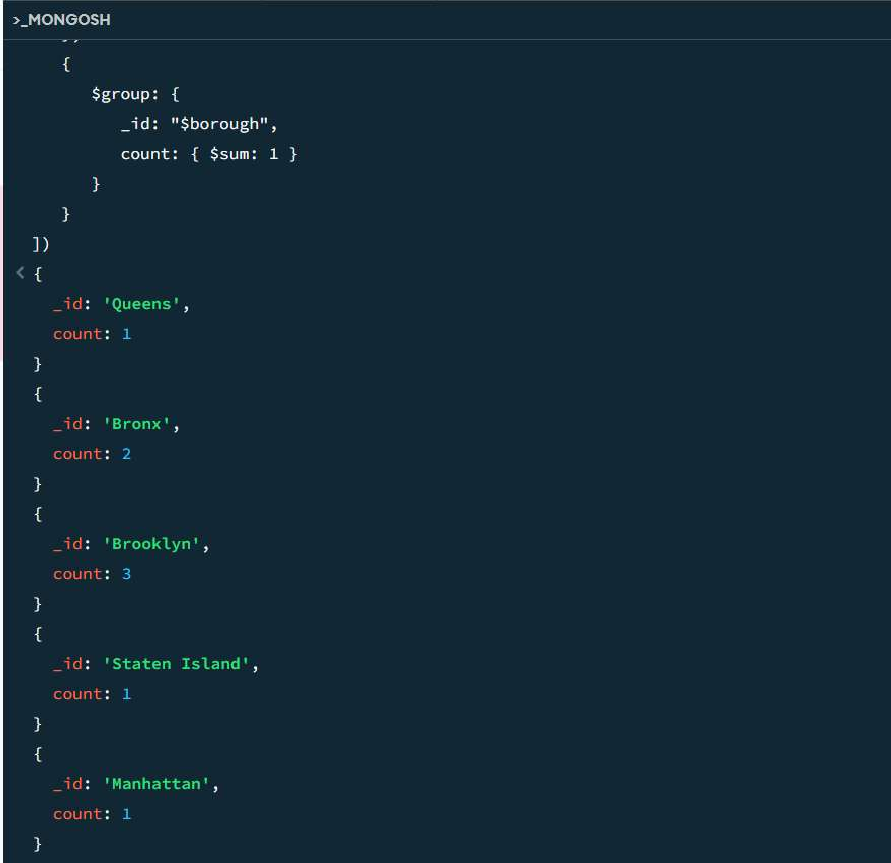
$match: { "grades.grade": "A" }

},

{

$group: {

\_id: "$borough", count: { $sum: 1 }



}

}

])

1. Find the lowest score for each borough.

db.restaurants.aggregate([

{

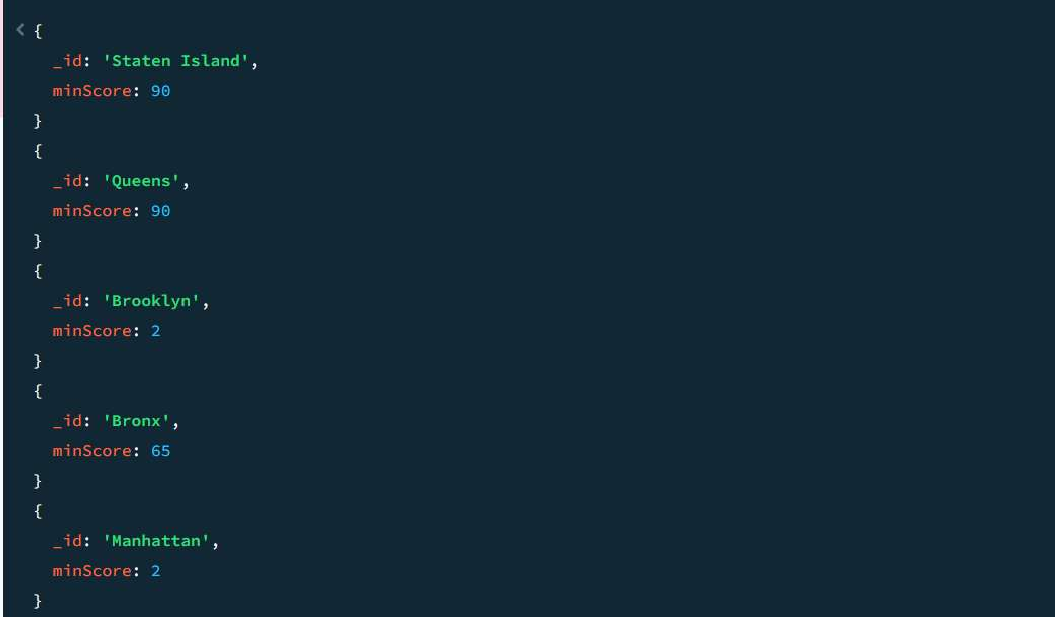
$unwind: "$grades"

},

{

$group: {

\_id: "$borough",



minScore: { $min: "$grades.score" }

}

}

])

1. Find the name and address of the restaurants that received a grade of 'A' on a specific date.

db.restaurants.find(

{

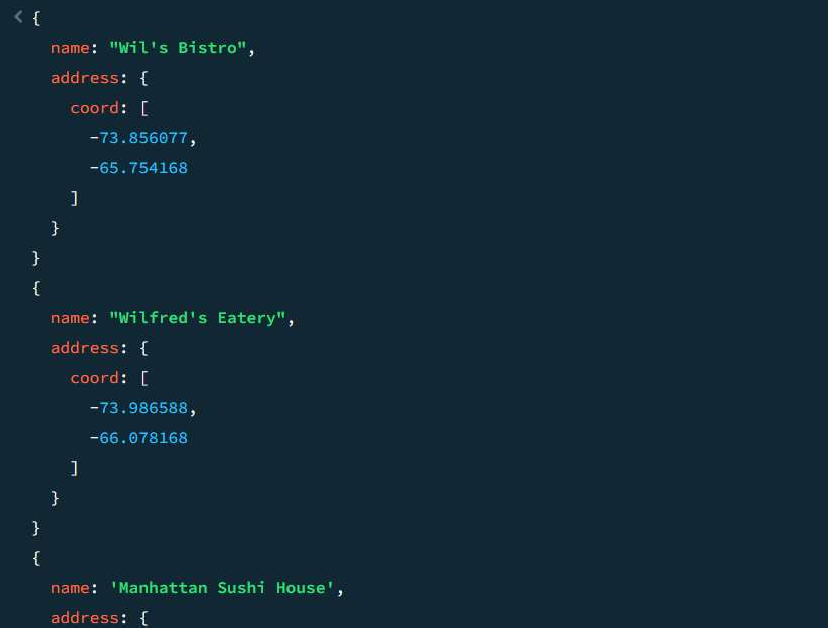
"grades.grade": "A",

"grades.date": ISODate("2014-08-11T00:00:00Z")

},

{ name: 1, "address": 1, \_id: 0 }

)



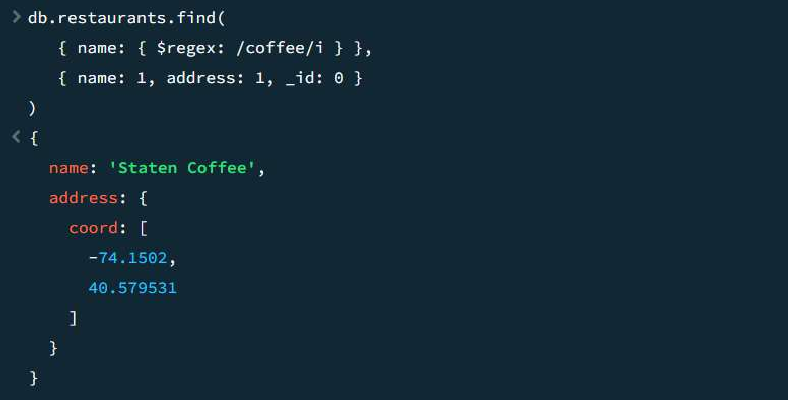
1. Find the name and address of the restaurants that have the word 'coffee' in their name.

db.restaurants.find(

{ name: { $regex: /coffee/i } },

{ name: 1, address: 1, \_id: 0 }

)



# []

1. Find the cuisine type that is most likely to receive a "C" grade.

db.restaurants.aggregate([

{

$match: { "grades.grade": "C" }

},

{

$group: {

\_id: "$cuisine", count: { $sum: 1 }

}

},

{

$sort: { count: -1 }

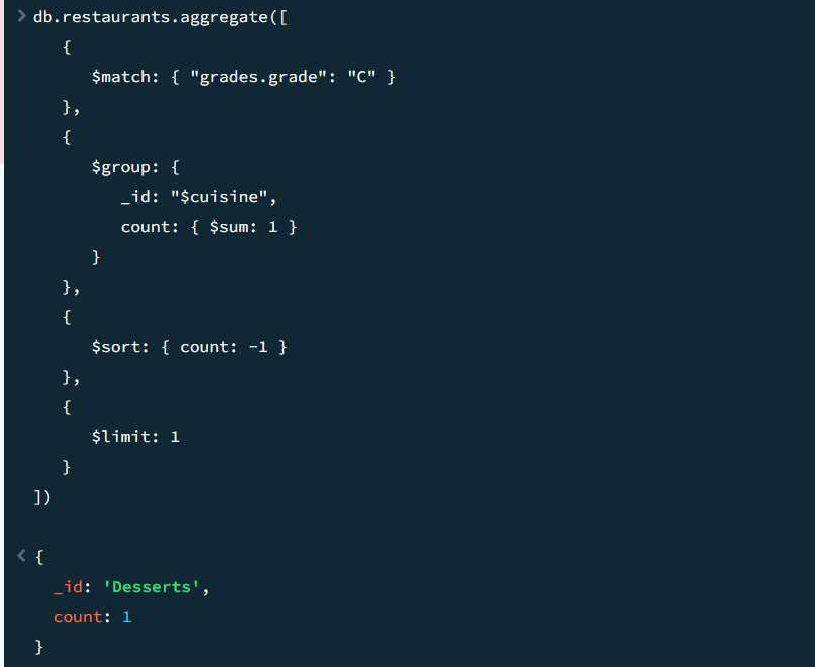
},

{

$limit: 1

}

])



1. Find the restaurants that achieved the highest total score.

db.restaurants.aggregate([

{

$unwind: "$grades"

},

{

$group: {

\_id: "$name",

totalScore: { $sum: "$grades.score" }

}

},

{

$sort: { totalScore: -1 }

},



{

$limit: 1

}

])

1. Find the top 5 restaurants with the highest average score for each cuisine type, along with their average scores.

db.restaurants.aggregate([

{

$unwind: "$grades"

},

{

$group: {

\_id: { cuisine: "$cuisine", name: "$name" },

# [

avgScore: { $avg: "$grades.score" }

}

},

{

$sort: { avgScore: -1 }

},

{

$group: {

\_id: "$\_id.cuisine",

topRestaurants: { $push: { name: "$\_id.name", avgScore: "$avgScore" } }

}

},

{

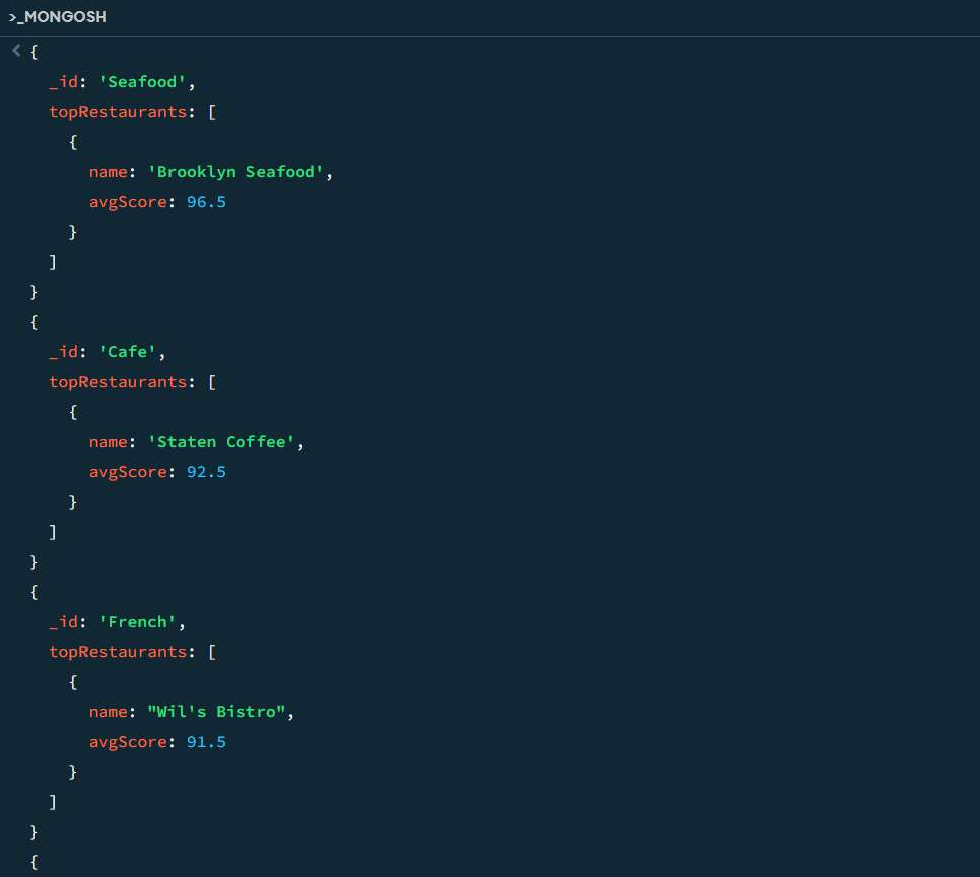
$project: {

topRestaurants: { $slice: ["$topRestaurants", 5] }

}

}

])



1. Find the top 5 restaurants in each borough with the highest number of "A" grades.

db.restaurants.aggregate([

{

$match: { "grades.grade": "A" }

},

{

$group: {

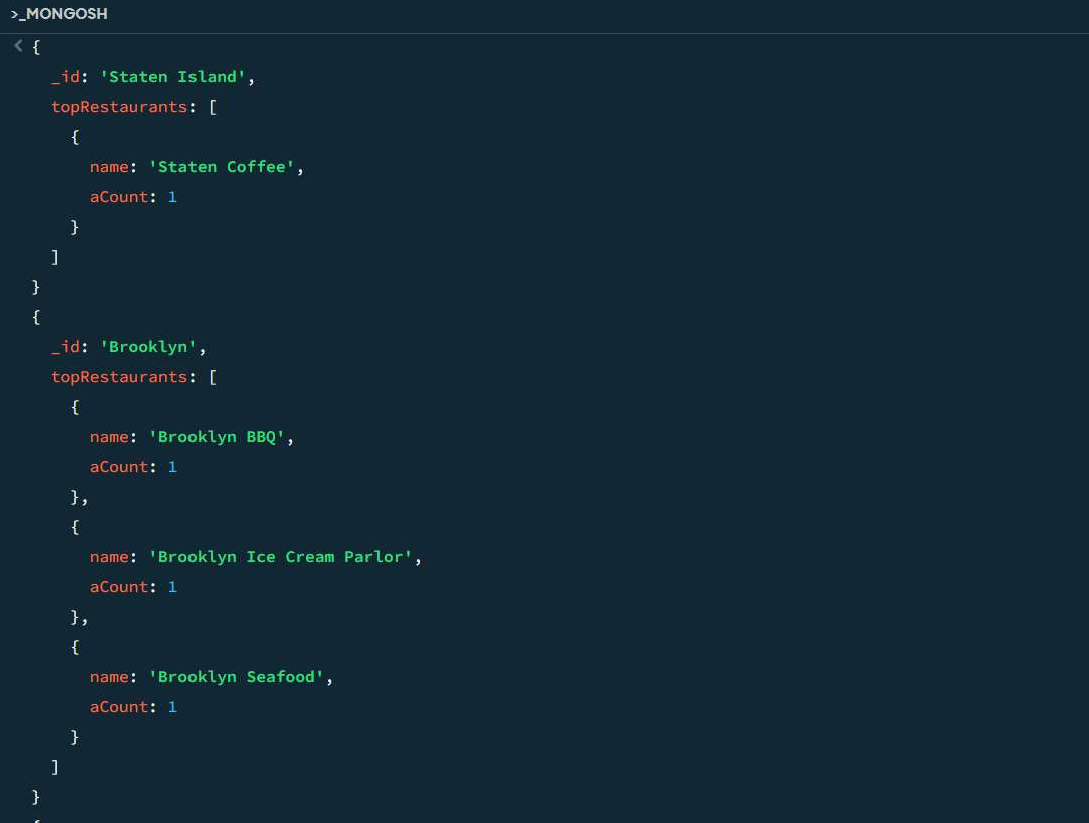
\_id: { borough: "$borough", name: "$name" },

aCount: { $sum: 1 }

}

},

{



# []

$sort: { aCount: -1 }

},

{

$group: {

\_id: "$\_id.borough",

topRestaurants: { $push: { name: "$\_id.name", aCount: "$aCount" } }

}

},

{

$project: {

topRestaurants: { $slice: ["$topRestaurants", 5] }

}

}

])

II. 1. Write a Java program to insert one document in restaurants collection.

import com.mongodb.client.MongoClient; import com.mongodb.client.MongoClients; import com.mongodb.client.MongoCollection; import com.mongodb.client.MongoDatabase; import org.bson.Document;

public class InsertRestaurant {

public static void main(String[] args) {

// MongoDB Connection String

String uri = "mongodb://localhost:27017"; // Replace with your MongoDB URI if needed try (MongoClientmongoClient = MongoClients.create(uri)) {

// Connect to the "DB\_regno" database

MongoDatabase database = mongoClient.getDatabase("DB\_regno");

// Get the "restaurants" collection

MongoCollection<Document> collection = database.getCollection("restaurants");

// Create a new restaurant document

Document newRestaurant = new Document("restaurant\_id", "40361577")

.append("name", "Sunset Diner")

.append("borough", "Brooklyn")

.append("cuisine", "American")

.append("grades", new Document("date", "2024-10-22T00:00:00Z")

.append("grade", "A")

.append("score", 10))

.append("address", new Document("street", "Sunset Blvd")

.append("zipcode", "11215")

.append("building", "345")

.append("coord", new double[]{40.675945, -73.974485}));

// Insert the document into the collection collection.insertOne(newRestaurant);

System.out.println("Document inserted successfully!");

} catch (Exception e) { e.printStackTrace();

}

}

}

2. Write a Java/PHP program to display all the restaurant which is in the borough Bronx. import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients; import com.mongodb.client.MongoCollection; import com.mongodb.client.MongoCursor; import com.mongodb.client.MongoDatabase; import org.bson.Document;

import static com.mongodb.client.model.Filters.eq;

public class FindBronxRestaurants { public static void main(String[] args) {

// MongoDB Connection String

String uri = "mongodb://localhost:27017"; // Adjust if needed try (MongoClientmongoClient = MongoClients.create(uri)) {

// Connect to the "DB\_regno" database

MongoDatabase database = mongoClient.getDatabase("DB\_regno");

// Get the "restaurants" collection

MongoCollection<Document> collection = database.getCollection("restaurants");

// Query to find all restaurants in the borough "Bronx" MongoCursor<Document> cursor = collection.find(eq("borough", "Bronx")).iterator();

// Display the results System.out.println("Restaurants in Bronx:");

while (cursor.hasNext()) {

Document restaurant = cursor.next(); System.out.println("Restaurant Name: " + restaurant.getString("name")); System.out.println("Cuisine: " + restaurant.getString("cuisine")); System.out.println("Address: " + restaurant.get("address")); System.out.println(" ");

}

} catch (Exception e) { e.printStackTrace();

}

}

}

III 1. Write a Java program to display name, email address and address of foreign students only. import java.sql.Connection;

import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.ResultSet;

import java.sql.SQLException;

public class ForeignStudents {

// Database credentials (adjust as per your Oracle DB configuration) static final String DB\_URL = "jdbc:oracle:thin:@localhost:1521:xe"; static final String USER = "your\_username";

static final String PASS = "your\_password";

public static void main(String[] args) { Connection conn = null;

PreparedStatementpstmt = null; ResultSetrs = null;

try {

// Step 1: Register Oracle JDBC driver Class.forName("oracle.jdbc.driver.OracleDriver");

// Step 2: Open a connection

conn = DriverManager.getConnection(DB\_URL, USER, PASS);

// Step 3: Create SQL query to retrieve foreign students' details

String sql = "SELECT NAME, EMAIL, ADDRESS FROM STUDENT WHERE IS\_FOREIGN = 'Y'";

// Step 4: Prepare and execute the SQL query pstmt = conn.prepareStatement(sql);

rs = pstmt.executeQuery();

// Step 5: Process the result set System.out.println("Foreign Students Information:"); System.out.println(" ");

while (rs.next()) {

String name = rs.getString("NAME"); String email = rs.getString("EMAIL"); String address = rs.getString("ADDRESS");

// Display student details System.out.println("Name: " + name);

System.out.println("Email: " + email); System.out.println("Address: " + address); System.out.println(" ");

}

} catch (SQLException | ClassNotFoundException e) { e.printStackTrace();

} finally {

// Step 6: Clean up resources try {

if (rs != null) rs.close();

if (pstmt != null) pstmt.close(); if (conn != null) conn.close();

} catch (SQLException e) { e.printStackTrace();

}

}

}

}

