Sample Questions – Final:

1. The collection below shows the structure of a single restaurant from a collection of restaurant documents (appropriately called restaurant). Answer the following questions by creating a MongoDB query for them.

**Collection Name –** restaurants (To be used in MongoDB queries)

{

"address": {

"building": "300",

"coord": [-60.856000, 50.142007],

"street": "Northern Blvd",

"zipcode": "11462"

},

"borough": "Queens",

"cuisine": "Thai",

"grades": [

{"date": ISODate("2017-07-01T00:00:00Z"), "grade": "B", "score": 8},

{"date": ISODate("2018-07-01T00:00:00Z"), "grade": "C", "score": 7},

],

"name": "Basil Basil",

"restaurant\_id": "30075445"

},

QUESTIONS:

1. Show all of the fields of the restaurant named Billie’s Bakery

**db.restaurants.find({name: "Billie's Bakery"})**

1. Show the names and grades of restaurants that have at least one score between (inclusive) 6 and 8.

**db.restaurants.find(**

**{"grades.score": {$gte: 6 }, "grades.score": {$lte: 8}},**

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

**)**

**// or**

**db.restaurants.find(**

**{$and:**

**[{"grades.score": {$gte: 6 }}, {"grades.score": {$lte: 8}}]},**

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

**)**

1. Show all fields of all of the Bakeries (cuisine) that aren’t in Brooklyn (borough) ordered by name in reverse alphabetical order.

**db.restaurants.find(**

**{"borough": {$ne: "Brooklyn"}, "cuisine": "Bakery" }**

**).sort({"name": -1});**

1. Find all restaurants that have gotten an A at least once… and are either a bakery or a pizza place. Show their name, address, and order by borough name alphabetically.

**db.restaurants.find(**

**{$and:**

**[{"grades.grade": 'A'},**

**{$or: [{cuisine: 'Pizza'}, {cuisine: 'Bakery'}]}]**

**},**

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

**).sort({"borough": 1});  
// \* instead of using $or, you can use a $in operator and**

**// have its value be a list of potential values**

**// \* and is not explicitly needed here, but this shows nesting**

**// of operators**

1. Convert the collection into borough documents, each with a list of restaurant names, so the resulting collection of documents may look something like:

**{"\_id" : "Brooklyn", "rests" : ["Paul's Pizza", "Billie's Bakery", ... },**

**{"\_id" : "Queens", "rests" : [ ...]}  
...**

**db.restaurant.aggregate(**

**[{$group : {**

**\_id: "$borough",**

**rests: {$push: "$name"}**

**}}]**

**)**

**db.restaurants.find(**

**{$and:**

**[{"grades.grade": 'A'},**

**{$or: [{cuisine: 'Pizza'}, {cuisine: 'Bakery'}]}]**

**},**

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

**).sort({"borough": 1});  
// \* instead of using $or, you can use a $in operator and**

**// have its value be a list of potential values**

**// \* and is not explicitly needed here, but this shows nesting**

**// of operators**

**REFERENCE:** [https://www.w3resource.com/mongodb-exercises/ - PracticeOnline](https://www.w3resource.com/mongodb-exercises/" \l "PracticeOnline)

1. Consider the “students” and “advisors” table shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| student\_id | first\_name | last\_name | advisor\_id |
| 1 | Tanisha | Blake | 2 |
| 2 | Jess | Goldsmith | NULL |
| 3 | Tracy | Wu | 3 |
| 4 | Alvin | Grand | 1 |
| 5 | Felix | Zimmermann | 2 |

|  |  |  |
| --- | --- | --- |
| advisor\_id | first\_name | last\_name |
| 1 | James | Francis |
| 2 | Amy | Cheng |
| 3 | Lamar | Alexander |
| 4 | Anita | Woods |

For each of the following SQL queries written below, write their correct output.

1. SELECT s.first\_name AS student\_name, a.first\_name AS advisor\_name FROM students AS s INNER JOIN advisors AS a ON s.advisor\_id = a.advisor\_id

|  |  |
| --- | --- |
| **student\_name** | **advisor\_name** |
| **Alvin** | **James** |
| **Tanisha** | **Amy** |
| **Felix** | **Amy** |
| **Tracy** | **Lamar** |

1. SELECT s.first\_name AS student\_name, a.first\_name AS advisor\_name FROM students AS s LEFT JOIN advisors AS a ON s.advisor\_id = a.advisor\_id

|  |  |
| --- | --- |
| **student\_name** | **advisor\_name** |
| **Alvin** | **James** |
| **Tanisha** | **Amy** |
| **Felix** | **Amy** |
| **Tracy** | **Lamar** |
| **Jess** | **NULL** |

1. SELECT s.first\_name AS student\_name, a.first\_name AS advisor\_name FROM students AS s RIGHT JOIN advisors AS a ON s.advisor\_id = a.advisor\_id

|  |  |
| --- | --- |
| **student\_name** | **advisor\_name** |
| **Alvin** | **James** |
| **Tanisha** | **Amy** |
| **Felix** | **Amy** |
| **Tracy** | **Lamar** |
| **NULL** | **Anita** |

**REFERENCE:** <https://www.deskbright.com/sql/sql-joins-interview-questions/>

1. Following is the database schema for a system similar to UBER. Here are the tables:

**Customer** (cusid, cusname, cusphone, cuscity);

**Driver** (driverid, dname, dphone, dcity);

**CarOwnership** (driverid, carid);

**Car** (carid, carbrand, carsize);

**Trips** (cusid, carid, driverid, getontime, getofftime, price, distance);

**Customers** are identified by a cusid, and we also store their name, phone number, and the city they live in. **Drivers** are identified by a driverid, and have a name, phone number, and city. One driver could own multiple cars, and a car could have multiple owners. **Cars** are identified by a carid, along with car brand and car size (e.g., compact, midsize, large). **Trips** are identified by cusid, carid, driverid, getontime, and we also store getofftime, price, and distance. The getontime is considered as the time when a trip takes place (which will be used in the queries). The getontime and getofftime attributes should store both time and date information.

Write SQL queries for the following questions. **NOTE:** You should use ‘Common Table Expression’ to write your SQL queries.

1. Output the cusid and cusname of the customer(s) who took the most expensive trip(s).

**Solution** -

1. Output the car brand that was used in trips by the largest number of distinct customers.

**Solution** - With temp AS

(SELECT Car.carbrand, COUNT(DISTINCT Trips.cusid) AS cuscount

FROM Car, Trips

WHERE Car.carid = Trips.carid

GROUP BY Car.carbrand)

SELECT temp.carbrand

FROM temp

WHERE temp.cuscount = (SELECT MAX(cuscount) FROM temp);

1. Output the driverid and dname of the driver who earned the most money (sum of prices) during Jan 2017.

**Solution** - With temp AS

(SELECT driverid, SUM(price) AS income

FROM Trips

WHERE YEAR(getontime) = '2017' and MONTH(getontime) = '1'

GROUP BY driverid)

SELECT Driver.driverid, dname

FROM Driver, temp

WHERE Driver.driverid = temp.driverid and income = (SELECT MAX(income) FROM temp);

1. In this problem, you have to write SQL queries for a database modeling the short-term leasing of houses or apartments, in a system somewhat similar to Airbnb. Here are the tables:

**Customer** (cid, cname, cphone, ccity);

**Landlord** (lid, lname, lphone, lcity);

**Residence** (rid, rname, rstate, rcity, raddr, rtype, rarea, lid);

**Leases** (cid, rid, startdate, enddate, price);

**Rating** (cid, rid, rtime, score);

**Customers** are identified by a cid, and we also store their name, phone number, and the city they live in. **Landlords** are identified by a lid, and have a name, phone number, and city. One landlord could own multiple houses/apartments, but a house could only have one owner. **Residences** are identified by rid, and have a rname, and rstate, rcity, and raddr to store the precise address of the house (e.g. rcity = ‘Brooklyn’, raddr = ‘3rd floor, 308 45st, 6 Avenue’), an rtype that stores the type of residence (e.g studio or 2BR-1BA), and rarea indicating the square footage. **Leases** are identified by cid, rid, lid, and startdate, and we also store enddate and price. The startdate and enddate attributes store both time and date information. When the lease ends, customers could give a rating to this house, where a **Rating** contains cid, hid, rtime, and a score. rtime is a timestamp which indicates the date and time when the customer made the rating; scores are ranging between 1 star (Terrible!) and 5 stars (Awesome!).

Write SQL queries for the following questions:

1. List the pairs of landlord and residence where landlord lives in Brooklyn and residence is in ‘Queens’.

**Solution** - SELECT lid,rid FROM Landlord NATURAL JOIN Residence WHERE lcity='Brooklyn' AND rcity='Queens';

1. Output the residence type that is owned by the largest number of distinct landlords.

**Solution** - SELECT rtype, COUNT (DISTINCT (lid)) AS count FROM Residence GROUP BY rtype HAVING count=max(count);

1. Output the customer(s) who took the most expensive residence(s).

**Solution** - SELECT c.\* FROM Customer c INNER JOIN Leases l ON c.cid=l.cid WHERE l.price = (Select max(price) from Leases);

1. For the next set of questions, you will be asked to convert the given tables to the right normalization form (for example 1NF, 2NF and 3NF):
2. The table given below is not in 1NF form. First specify the reason for that and then convert it to 1NF form by taking appropriate steps:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| empID | name | job | deptID | skills |
| 1 | Bob | Programmer | 1 | C, Perl, Java |
| 2 | Alice | DBA | 2 | MySQL, PostgreSQL |

**Solution** – Not in 1NF because skills have multiple values.

|  |  |  |  |
| --- | --- | --- | --- |
| empID | name | job | deptID |
| 1 | Bob | Programmer | 1 |
| 2 | Alice | DBA | 2 |

|  |  |
| --- | --- |
| empID | skill |
| 1 | C |
| 1 | Perl |
| 1 | Java |
| 2 | MySQL |
| 2 | PostgreSQL |

1. Convert the table given below to its right 2NF.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| empID | name | job | deptID | skill |
| 1 | Bob | Programmer | 1 | C |
| 1 | Bob | Programmer | 1 | Perl |
| 1 | Bob | Programmer | 1 | Java |
| 2 | Alice | DBA | 2 | MySQL |
| 2 | Alice | DBA | 2 | PostgreSQL |

**Solution** –

|  |  |  |  |
| --- | --- | --- | --- |
| empID | name | job | deptID |
| 1 | Bob | Programmer | 1 |
| 2 | Alice | DBA | 2 |

|  |  |
| --- | --- |
| empID | skill |
| 1 | C |
| 1 | Perl |
| 1 | Java |
| 2 | MySQL |
| 2 | PostgreSQl |

1. The table below is not in its 3NF. Specify the reason for that and convert it to 3NF.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| empID | name | job | deptID | dept |
| 1 | Bob | Programmer | 1 | Engineering |
| 2 | Alice | DBA | 2 | Databases |
| 3 | Kim | Programmer | 1 | Engineering |

**Solution** – empID determines name, job, deptID and dept. But, deptID also determines dept – a transitive dependency. Hence, the solution is to separate dept table.

|  |  |  |  |
| --- | --- | --- | --- |
| empID | name | job | deptID |
| 1 | Bob | Programmer | 1 |
| 2 | Alice | DBA | 2 |
| 3 | Kim | Programmer | 1 |

|  |  |
| --- | --- |
| deptID | dept |
| 1 | Engineering |
| 2 | Databases |

**REFERENCE** - <https://www.cc.gatech.edu/~simpkins/teaching/gatech/cs2340/slides/db-normalization.pdf>

1. For the SQL Query Plan shown below, answer the following questions?

**Gather** (cost=1000.00..5739.49 rows=124 width=78)

Workers Planned: 2

-> **Parallel Seq Scan** on web\_user (cost=0.00..4727.09 rows=52 width=78)

Filter: ((first)::text = 'tegan'::text)

(4 rows)

1. Write the SQL query in order to generate the SQL query plan shown above.

**Solution** – EXPLAIN SELECT \* FROM web\_user WHERE first='tegan';

1. Explain in one sentence, what does **‘Gather’**, **‘Seq Scan’** and **‘Parallel Seq Scan**’ tell you about how postgres will find the data.

**Solution**:

* **Seq Scan** scans the whole table
* **Gather** shows how many workers are available for parallelization
* **Parallel Seq Scan** runs multiple queries in parallel in order to execute the original query.

**REFERENCE** - <https://cs.nyu.edu/courses/fall18/CSCI-UA.0480-007/_site/slides/db/indexes.html?print>