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:

a) Show all of the fields of the restaurant named Billie's Bakery

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

b) 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}
)
```

c) 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});
```

d) 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
```

e) 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

2) 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.

a) 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**

b) 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**

c) 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/

3) Your are working on a table called users. It has a field called name, where everyone's name is stored as a varchar, in the format, "last,first" (for example, 'Smith,Sarah'). Write a query that selects all of the names from the user table in the format "first last" (for example, 'Sarah Smith'). You can assume that there is always one comma in all values of the name column.

```
SELECT name_parts[2] || ' ' || name_parts[1]
FROM (
   SELECT string_to_array('Versoza,Joe', ',') as name_parts
   FROM user
) as parsed_name;
```

4) 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);

car_ownership (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.

Use Common Table Expressions and sub-queries to answer the next couple of **challenging** questions (both use a similar pattern of creating a CTE for the aggregation, and then a subquery for finding and matching on a max value).

a) Output the car brand that was used in trips by the largest number of distinct customers. If there's a tie, it's ok to output all brands with the same number.

```
WITH temp AS
(SELECT car.carbrand, COUNT(DISTINCT trips.cusid) AS cuscount
FROM car
INNER JOIN trips
ON car.carid = trips.carid
GROUP BY car.carbrand)

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

b) Output the driverid and dname of the driver who earned the most money (sum of prices) during Jan 2017. Hint: use to_char(getontime, 'YYYY-MM'). If there's a tie, all drivers can be listed.

```
WITH temp  AS
(SELECT driverid, SUM(price) AS income
FROM trips
WHERE to_char(getontime, 'YYYY-MM') = '2017-01'
GROUP BY driverid)

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

5) 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 (<u>cid</u>, cname, cphone, ccity);
landlord (<u>lid</u>, lname, lphone, lcity);
residence (<u>rid</u>, rname, rstate, rcity, raddr, rtype, rarea, lid);
leases (<u>cid</u>, <u>rid</u>, <u>startdate</u>, enddate, price);
rating (<u>cid</u>, <u>rid</u>, <u>rtime</u>, 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:

a) List the pairs of landlords and residences by their corresponding ids where the landlord lives in Brooklyn and their rental(s) (residence) are in 'Queens' sorted by the residence id in reverse (highest id first), showing only the first 10 records. No group by is required (the landlord id can show up multiple times with different residence ids)

```
SELECT lid,rid
FROM landlord
INNER JOIN residence ON lcity='Brooklyn' AND rcity='Queens'
ORDER BY rid DESC
LIMIT 10;
```

b) Imagine that there's a new table that's created, called customer_rating. This table will have the *actual* customer name (column name) and their rating score (column score). It's ok if a customer's name appears more than once (since they may have multiple ratings). It's also ok if score is null. In a single query, populate this table from the existing tables; note that if a customer hasn't created any ratings, they should still be brought in to the new table, but with a null value for score.

```
INSERT INTO customer_rating
SELECT name, score
FROM customer
LEFT JOIN rating
ON customer.cid = rating.cid;
```

c) Output the count of residences per landlord; include both the count and the landlord's name in the results.

```
SELECT landlord.name, lr_counts.residence_count
FROM landlord
INNER JOIN
    (SELECT residence.lid, count(*) as residence_count
    FROM residence
    GROUP BY residence.lid
    HAVING count(*) > 1) as lr_counts
ON lr_counts.lid = landlord.lid;
```

d) Show all columns from customer where the customer(s) rented the most expensive residence in the dataset (if there's a tie, you can show the customers for both residences).

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

6) 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):

a) 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, C

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	C

b) 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	C

Solution –

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

empID	skill
1	C
1	Perl
1	Java
2	MySQL
2	C

c) 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