Week 8 Querying Databases: The Non-relational Ways (1)

Assignment Solutions

Question 1

How can you find the total number of documents in a collection named "enrolls" in MongoDB? Select all the methods that work. Assume the collection is not sharded, and the MongoDB version is 3.4.

*A: **db.enrolls.count()**

*B: **db.enrolls.find().count()**

*C: **db.enrolls.aggregate([{$count: "count"}])**

*D: **db.enrolls.aggregate([{$group: {_id: null, count: {$sum: 1}}}])**

E: **db.enrolls.aggregate([{$group: {_id: null, count: {$count: 1}}}])**

Solution: ABCD

Explanation: For C, the $count stage was introduced in version 3.4.

For E, $count can be used as an aggregation accumulator, but it does not take any parameter– i.e., it should be used as `$count: {}`.

Question 2

Consider a MongoDB collection restaurants, with the following example document. Is it possible to find those restaurants that have received at least one score greater than 100, using the **find** method? If so, how?


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


*A: Yes. **db.restaurants.find({"grades.score": {$gt: 100}})**

B: No. One must unwind the **grades** array by using the **aggregate** method.

Solution: A

Explanation: For A, the find() method is able to look through an array, in this case, "grades.score", and a document is matched if at least one element in the array satisfies the condition.

For B, Aggregate() is more general and powerful than find(), but to deal with arrays in documents, we have to unwind first, and here we show that sometimes find() provides a good shortcut to look through arrays without explicit unwinding.

Question 3

Consider a MongoDB collection restaurants, with the following example document. How can you find those "inconsistent" restaurants that have received at least one grade of "A" as well as at least one grade of "C"?


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


```
A: db.restaurants.find( {$or: [ {"grades.grade": "A"},
{"grades.grade": "C"} ] } )
```

```
*B: db.restaurants.find( {$and: [ {"grades.grade": "A"},
{"grades.grade": "C"} ] } )
```

```
C: db.restaurants.find( {"grades.grade": ["A", "C"]} )
```

Solution: B

Explanation: For A, Or operator only satisfies at least one of the conditions, not necessarily both at the same time.

For B, The $all operator is equivalent to the $and operator (behavior since version 2.6). $and satisfies both conditions.

For C, This query matches documents whose "grades.grade" array is exactly ["A", "C"], that is, the restaurant has been graded exactly twice, and received "A" the first time, and "C" the second time.

Question 4

Consider a MongoDB collection restaurants, with the following example document. How can you sort the restaurants by the number of grade A ratings they have received, in descending order?


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


```
*A: db.restaurants.aggregate([ {$unwind: "$grades"}, {$match:
{"grades.grade": "A"}}, {$group: {_id: "$_id", count_gradeA: {$sum:
1}}}, {$sort: {count_gradeA: -1}} ])

B: db.restaurants.find({"grades.grade":
"A"}).count().sort("grades.grade": -1)
```

Solution: A

Explanation: For A, Feedback: Before matching for grade A ratings, we must unwind the grades array. After the $match stage, we have the documents each representing one instance of some restaurant receiving a grade A. So now if we group by restaurant IDs, and get the size of each group via $sum, we have essentially computed the number of times each restaurant has received grade A. Finally we sort the results in descending order.

For B, Here count() only counts the total number of restaurants that have received at least one grade A. And thus calling sort() on a single count value is illegal.

Question 5

Consider a MongoDB collection restaurants, with the following example document. How can you find all the "straight-A" restaurants, namely those that have received an "A" every time they are being graded. Sort the results by the total number of "A"s received, in descending order.


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


```
*A: db.restaurants.aggregate([ {$unwind: "$grades"}, {$group: { _id:
"$_id", count_grades: {$sum: 1}, count_A: {$sum: {$cond: [{$eq:
["$grades.grade", "A"]}, 1, 0]}} } }, {$project: { is_straight_A:
{$eq: ["$count_grades", "$count_A"]}, count_A: 1, } }, {$match:
{is_straight_A: true}}, {$sort: {"count_A": -1}}, ])
```

```
B: db.restaurants.aggregate([ {$unwind: "$grades"}, {$match:
{"grades.grade": "A"}}, {$group: { _id: "$_id", count_A: {$sum: 1},
} }, {$sort: {"count_A": -1}}, ])
```

Solution: A

Explanation: For A, After unwinding, we need to obtain both the total number of grades a restaurant has received (count_grades), and the total number of "A"s received (count_A). Also note that we cannot filter out all the non-A documents prior to grouping as we did in Question 3. Next we need to test if count_grades == count_A, which is the definition of "straight-A". This can be done in a $project stage. Then we can filter out all those that are not straight-A restaurants. Finally we sort the results by count_A in descending order as required.

For B, This only counts the number of "A"s a restaurant has received, but without the knowledge of the total number of grades,we cannot determine whether a restaurant is straight-A.

Question 6

Consider a MongoDB collection restaurants, with the following example document. How can you find the average score for each of the cuisines, and sort them by the average score in ascending order? Here we define the average score of a cuisine such that every restaurant belonging to a certain cuisine is weighted equally, regardless of how many times it has been graded. That is, every restaurant contributes a single average score of its own to the cuisine's average score.


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


```
*A: db.restaurants.aggregate([ {$unwind: "$grades"}, {$group: { _id:
"$_id", avg_restaurant_score: {$avg: "$grades.score"}, cuisine:
{$first: "$cuisine"} } }, {$group: { _id: "$cuisine",
avg_cuisine_score: {$avg: "$avg_restaurant_score"}, } }, {$sort:
{avg_cuisine_score: 1}} ])
```

```
B: db.restaurants.aggregate([ {$unwind: "$grades"}, {$group: { _id:
"$cuisine", avg_score: {$avg: "$grades.score"} } }, {$sort:
{avg_score: 1}} ])
```

Solution: A

Explanation: For A, Since the average defined in the question weights each restaurant of certain cuisine equally, we need to first find the average score for each restaurant by using the first group stage. Note that since the second group stage takes the input only from the first group result, we have to retain the "cuisine" attribute, which can be obtained by using the $first operator to get the value from the first document in the group (note this works because restaurant ID functionally determines its cuisine). Now in the second group stage, we can compute the average score over all the restaurants belonging to a specific cuisine; effectively, we are taking the average of averages.

For B, This computes the average score over all the grades received for a given cuisine, that is, each restaurant of that cuisine is not weighted equally (those that have been graded more frequently will bear a larger weight), which violates the requirement.

Question 7

We just imported two collections, "students", with fields "_id", "sid", "major", and "enrolls", with fields "_id", "student_id", "course_id", "term", into the database "academic_world" in MongoDB.

1. How can you join "enrolls" into "students", which is the main collection that you want to work with?

2. Now say we have only 3 students in the "students" collection, their sID being: "alice1", "bob2", "cate3", and we know that "alice1" has taken 3 courses, "bob2" has taken 2 courses, and "cate3", as a new freshman, hasn't taken any courses yet.

How many documents in total would you get as the output of the join?

```
*A: db.students.aggregate([ {$lookup: { from: "enrolls", localField:
"sid", foreignField: "student_id", as: "enroll_records" } } ])

Output: 3 documents in total

B: db.students.aggregate([ {$lookup: { from: "enrolls", localField:
"student_id", foreignField: "sid", as: "enroll_records" } } ])

Output: 3 documents in total

C: db.students.aggregate([ {$lookup: { from: "enrolls", localField:
"student_id", foreignField: "sid", as: "enroll_records" } } ])

Output: 5 documents in total

D: db.students.aggregate([ {$lookup: { from: "enrolls", localField:
"sid", foreignField: "student_id", As: "enroll_records" } } ])

Output: 5 documents in total
```

Solution: A

Explanation: "localField" is a field from the collection on which aggregate() is called, while "foreignField" is a field from the "from" collection. The "$lookup" stage does not generate a new, "combined" document for every matching pair of documents as a SQL join would do; instead, it brings in all the matching documents as an array, whose name is specified in the "as" field. If there are no matching documents from the other collection, the array is simply empty, but the document itself is not eliminated from the result (hence "lookup" is regarded as an outer join). Therefore, "lookup" does not change the total number of documents in the output.