Databases for Data Science

Lecture 10 · 2022-10-07

The plan

Lectures		
Today	10/07	MongoDB: Aggregation, Exercises
Monday	10/10	MongoDB wrap-up; discuss other NoSQL DBs
Wednesday	10/12	JSON in PostGreSQL; topics
After break Monday	10/24	Final Exam

Summary: MongoDB CRUD

verb	SQL	Mongo	
Create	`INSERT INTO`	`.insert_one()`	`.insert_many()`
Read	`SELECT`	`.find_one()`	`.find()`
Update	`UPDATE`	`.update_one()`	`.update_many()`
Delete	`DELETE`	`.remove_one()`	`.remove_many()`

The Mongo functions accept JS objects that specify the intended behavior:

- Filters, e.g. `{ 'address.country_code': 'US' }`
- Projections, e.g. `{ 'name': 1 }`
- Operations, e.g. `{ '$push': {'amenities': {'$each': ['VR', 'e-bikes'] }}}`

What about joins?

Well, it's trickier - we're non-relational.

Any ideas?

Map-Reduce

Two stages.

- Map: take each record and produce a new key-value pair
- Reduce: group by keys, aggregate values

```sql
SELECT
  major, avg(gpa)
FROM
  student
GROUP BY
  major;
```

Aggregation pipeline

MongoDB has a `mapReduce` function, but it's deprecated.

Instead, we use *aggregation pipelines*.

- These are more general...
- ...but more complicated!

Aggregation pipeline

Idea: carry out a series of **stages**.

```
coll = db.listingsAndReviews

coll.aggregate( [
  {
    '$group':
    {
      # Use _id to specify what we GROUP BY
      '_id': '$bedrooms',

      # Add other columns derived from each group
      'how_many': {'$sum': 1},
      'avg_bath': {'$avg': '$bathrooms'}}
  }
])
```

Equivalent SQL:

```
SELECT
  bedrooms AS _id,
  COUNT(*) AS how_many,
  AVG(bathrooms) AS avg_bath
FROM
  listingsAndReviews
GROUP BY
  bedrooms;
```

Aggregation framework syntax

Let's break this down.

- `aggregate` is a function.
- It takes a list of *stages*, so we call `aggregate([...])`.
- Each stage is an *object*. It has one key that specifies the type of stage.
    - e.g., `{ '$group': {...}}`
- That key maps to a value which parametrizes the stage.
- Inside this specification, we write a `$` whenever we're referring to one of the *original* fields.
    - and also when using an operator.

```
coll.aggregate( [
  {
    '$group':
    {
      '_id': '$bedrooms',
      'how_many': {'$sum': 1},
      'avg_bath': {'$avg': '$bathrooms'}}
  }
])
```

Aggregation pipeline

There are *many* different kinds of stage.

- see mongodb.com/docs/manual/reference/operator/aggregation-pipeline/

For today, we're most interested in:

- `$group`: performs grouping (aka reducing, aka `SELECT ... GROUP BY`)
- `$match`: performs filtering (aka `WHERE`)
- `$project`: performs mapping (aka `SELECT`)

Examples

Get the total number of rooms at each property.

```
coll.aggregate([
    {
        '$project': {
            # Use "1" to indicate that the given field should be included
            'name': 1,

            # Construct a new field
            'total_rooms': {'$add': ['$bedrooms','$bathrooms'] }
        }
    }
])
```

- What is the equivalent SQL?

Examples

```
coll.aggregate([
    {
      '$group': {
        '_id': '$address.country_code',
        'bed': {'$avg': '$bedrooms'},
        'bath': {'$avg': '$bathrooms'},
      }
    },
    {
        '$project': {
            '_id': 0,
            'country': '_id',
            'rooms': {'$add': ['$bed','$bath'] }
        }
    }
])
```

- What does each stage do?
- What is the equivalent SQL?

[More exercises]