W9 Activity: Operators MongoDB

Start Assignment

- Due Sunday by 11:59pm
- Points 20
- Submitting a file upload

Submission

Submit one document that includes the following for each of the exercises:

- The **mongosh query command** you used
  - (a formatted code block would be neat, but however you need to format it is fine)
- A screenshot of your terminal window that shows **clearly** what the response from your DB connection was

Setup

Use the sample_mflix database for this activity: `use sample_mflix`

Exercise 1: Nested queries

- We can access nested *objects* like so:
  - `db.movies.find({ "awards.wins" : 1 } )`
  - Notice how we have to use the dot notation surrounded with quotes, `"awards.wins"`
- We access values in arrays like this:
  - `db.movies.find( { genres : "Drama" } )`

Exercise 2: Equal

The equality operator can either be called *explicitly* or *implicitly*.
We've actually already been using the equality operator implicitly.
Anytime we've used `db.collection.find( { field : value} )`, we've actually done the equivalent of:
`db.collection.find( { field : {$eq: value} } )`

- Use the explicit equality operator to retrieve one movie of type movie:
- `db.movies.findOne( { type : { $eq : 'movie' } } )`

Exercise 3: Not equal

We can rule out particular attributes by using the `$ne` operator:

- `db.movies.find( { type : { $ne : 'movie' } } )`

Exercise 4: Less than

- Use the following to find a movie under 120 minutes (2 hrs) using the less than operator, `$lt`
- `db.movies.find({ runtime : { $lt : 120 } } )`

Exercise 5: Greater than or equal to

This command will both use a nested document, and filter them by ratings that are greater than or equal to.

- `db.movies.find({ 'imdb.rating' : { $gte : 8 } } )`

Exercise 6: In (set)

We can check that a document meets either, or both, of a filter in a set using the `$in` operator.

- `db.movies.find({ genres : { $in : ['Comedy','Short'] } } )`
- Note that this is an *inclusive* check. We need to match at minimum, one in the set.
  This is the closest to the SQL `exists` statement in a WHERE clause.

Exercise 7: AND

We can combine conditions by using the logical `$and` operator and passing in an array of conditions.

- `db.movies.find( { $and : [{ type : { $eq : 'series' }}, { year : { $gt : 1980 }} ] } )`

Exercise 8: NOR

We can use this operator to avoid documents that meet multiple conditions.
In the following, a document can't be of *either* type movie, nor can it be more than 120 minutes long.

- `db.movies.find( { $nor : [ { type : 'movie' }, { runtime : { $gt : 120 }} ] } )`
- Notice how we have another operator nested within this one (`$gt` operator within the `$nor`)

Exercise 9: NOT

We can get the negated result set of any operator by using the `$not` operator.

- `db.movies.find( { rated : { $not : { $eq : 'TV-MA' } } } )`
  What's the difference between this and just using `$ne` (not equal)?
  This `$not` is for *logical disjunctions*, whereas `$ne` check if *an existing field* has a value not equal to the one defined.
  So for this query, the `$not` operator will return documents where either:
- the `rated` field is not equal to `'TV-MA'`
- or the `rated` field is not listed in the document

Exercise 10: Exists

We can check that a field is being used in a document by using the `$exists` operator. This is helpful because *we do not require* that documents have all the matching fields available.

- `db.movies.find( { tomatoes : { $exists : true } } )`
  Here, we're setting checking if the field exists by passing in **true**. If we were to use `{ $exists : false }` , it would return only documents where the field did not exist.

Note: this is **different than the SQL exists**. The SQL exists is closer to the *$in* operator.