VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

MongoDB
Read

Lecturer : Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for MongoDB Read Operation

- CRUD Operations
- Read
  - `db.collection.find()` method
  - Query selection objects (documents)
  - Projections
  - Query Results Processing
- Cursor
  - ***Reedings:***
    - *Have a look at Readings on the Home Page*

CRUD Operations

- The acronym CRUD stands for:
  - **C**reate (insert),
  - **R**ead,
  - **U**pdate, and
  - **D**elete
- MongoDB CRUD operations target a single, specific collection
- All examples on the following slides use MongDB methods in `mongo shell`

A Document Example

```
{
_id: ObjectId("33667997ab01")
  course: {code: "SWEN432",
            title: "Advanced DB"},
  year: 2014,
  coordinator: "Pavle",
guest_lecturer: "Aaron",
  students: [{name:"Matt", surname: "Smith"},
  {name:"Jack", surname: "Brown"},...,
  {name:"Lingshu", surname: "Chang"}],
  no_of_st: 11
  prerequisites: ["SWEN304", "COMP261",
  "NWEN304"]
}
```

Read Operation (Queries)

- MongoDB provides a **`db.collection.find()`** method
  - It is accessible from the `mong shell`
  - MongoDB shell is a daemon process connected to a MongoDB server with a JavaScript interface
- The method accepts:
  - Query selection criteria (conditions),
  - Projection list, and
  - Modifiers (`sort`, `limit`, `skip`)

  and executes on the collection to be queried
  - There is also a **`db.collection.findOne()`** method that returns just one document
- The `find` method returns a cursor to the matching documents

`db.collection.find()`

- Parameters passed to the `find` method are JSON objects
- Example:

```
db.myclasses.find({coordinator: “Pavle”})
```

Here, the `{coordinator: “Pavle”}` document is a selection criteria based on the equality comparison operator

- If the selection object is empty, all documents of a collection are returned

```
db.myclasses.find({})
```

Operators of the Selection Object (1)

- General form:

```
<field_name>: {$<operator>: <value>}

<field_name>: {$<operator>: <value>, $<operator>:
<value>}
```

- Comparison operators supported (besides equality):
  - Non equality: `$ne`,
  - Numerical relations: `$gt`, `$gte`, `$lt`, `$lte`,
- Example:

```
{no_of_st: {$gt: 8}}
```

```
{no_of_st: {$gt: 8, $lt: 18}}
```

Operators of the Selection Object (2)

- (Non) Existence of a field: `$exists` with `false`, or `true`

```
{guest_lecturer: {$exists: true}}
```

- Logical junctions
  - AND junction: comparison expressions separated by a comma,
  - OR junction: special `$or` operator assigned to an array of expressions,
  - NOR junctions: special `$nor` operator assigned to an array of expressions,
  - A term can be negated using `$not` operator
- An OR selection object with an AND term

```
{$or: [{year: 2014},
  {coordinator: “Pavle”, no_of_st: {$gt: 8}}
]}
```

Regular Expressions

- Query selection criteria can be also based on regular expressions
- The `$regex` operator provides regular expression capabilities for pattern matching *strings* in queries
- Syntax:

```
{<field>: {$regex: /pattern/, $options: 'options'}}
```

- Some `<options>`:
  - `'i'` – for case insensitive matching,
  - `'s'` – allows any character to be replaced by `"."` (dot)
- Examples:

```
{prerequisites: {$regex: /^SWEN/}}
```

```
{prerequisites: {$regex: /.261/, $options: 's'}}
```

Selection in Embedded Documents

- Equality Match on Fields within an Embedded Document:

```
{'course.title': "Advanced DB"}
```

- Exact Match on the Whole Embedded Document:

```
{course:
{code: "SWEN432", title: "Advanced DB"}}
```

  - Field names of embedded documents have to be enclosed either between apostrophes or quotation marks
    - This does not apply to simple fields

Array Selection Objects

- To search for a single value inside an array, the array field name can be simply assigned to the desired value:

```
{prerequisites: “NWEN304”}
```

- Exact Match on an Array: `{<array>: <value>}`, requires that the whole `<array>` matches `<value>`

```
{prerequisites:
[“SWEN304”, “COMP261”,“NWEN304”]}
```

- Match a Specific Array Element by Value: `‘<array>.<index>’: <value>`

```
{‘prerequisites.0’: “SWEN304”}
```

Selection in an Array of Documents

- Match a Field of a Subdocument:

```
{'students.name': "Matt"}
```

- Match a Field Using the Array Index:

```
{'students.1.name': "Jack"}
```

- Single Element Satisfies Multiple Criteria:

```
{students: {$elemMatch:
{name:"Jack", surname: "Brown"}}}
```

- Combination of Elements Satisfies Multiple Criteria:

```
{'students.name':"Jack",
'students.surname': "Chang"}
```

About Projections

- Queries return all fields in all matching documents by default
- Projections are defined in the second argument of the `find()` method (the first is the query selector)
- Projections may either specify:
  - A list of fields to return (designated by `{<field_name>: 1}`), or
  - A list of fields to exclude (designated by `{<field_name>: 0}`)

  in the result,
  - Only the exclusion of `_id` field can be mixed with fields to return
- Examples:

```
db.myclasses.find({no_of_st: {$gt: 8}},
                  {'course.title': 1, _id: 0})
```

```
db.myclasses.find({no_of_st: {$gt: 8}},
                 {coordinator: 0, no_of_st: 0})
```

Projection for Array Fields (1)

- Projection operators for fields that contain arrays :
- `$slice`,
- `$`, and
- `$elemMatch`
- The following selection object returns just the first two elements in the `prerequisites` array:

```
{prerequisites: {$slice: 2}}
```

- `$` projects the first element in an array that matches the query condition:

```
db.myclasses.find({‘students.name’:
“Matt”}, { “students.$": 1 })
```

Projection for Array Fields (2)

- To specify criteria on multiple fields of documents inside that array, the `$elemMatch` query operator is used
- The following query will return any embedded documents inside a `students` array that have a `name` equal to `“Jack”` and a `surname` equal to `“Brown”`:

```
db.myclasses.find({students: {$elemMatch:
                         {name:“Jack”,
                          surname: “Brown”}}},
                         { "grades.$": 1 })
```

- The `$elemMatch` allows projecting based:
  - On a condition not in the query, or
  - On multiple fields in the array’s embedded documents

Query Result Processing

- Query results can be further:
  - Arranged using `sort` operation,
  - Restricted by the number `n` of first documents to return by `limit` operation, or
  - Restricted by the number `n` of first documents not to return by `skip` operation
- Examples:

```
db.myclasses.find({}).sort(
{no_of_st: -1}).limit(1)
```

  - `find({})` returns all documents in `myclasses` collection,
  - `sort({no_of_st}: -1)` sorts documents in descending order
  - `limit(1)` returns the document with the greatest `no_of_st`

```
db.myclasses.find({}).sort(
{no_of_st: 1}).skip(2)
```

Cursor

- In the `mongo shell`, the `find()` method queries a collection and returns a `cursor` to the resulting documents
- In the mong shell, the `cursor` is ***automatically*** iterated up to 20 times to print the first 20 matching documents
- The `cursor` can also be assigned to a variable using the `var` keyword

```
var mycursor =
db.myclasses.find({no_of_st: {$gt:8}});
```

- To print up to 20 first documents:

```
mycursor
```

Querying Referenced Documents

- Consider referencing documents in Data Modeling
  - Query: Retrieve course title, coordinator’s name, and guest lecturer’s name of all SWEN classes
- The query script is given on the next slide
  - The `find()` method assigns **references** to SWEN classes to the variable `curs`
  - In each iteration over the variable `curs:`
    - The current `class` document is assigned to the variable `my_class`
    - The coordinator’s (guest lecturer’s) **document** is assigned to the variable `c_lec (c_lec)` using `findOne()` method and lecturer’s `(`guest lecturer’s`)` `_id` field in the current class document,
    - The variable `ret` is assigned `title` of the class, the coordinator’s `name,` and the guest lecturer’s `name`
    - The content of the variable `ret` is printed using `print(tojson())`
- **Note:** the method `findOne()` returns a document, not a cursor

Querying Referenced Documents

```
> var curs = db.class_ref.find({
  _id: {$regex: /^SWEN/}})
> while (curs.hasNext()) {
  my_class = curs.next()
  c_lec = db.lecturer.findOne({
  _id: my_class.coordinator})
  g_lec = db.lecturer.findOne({
  _id: my_class.guest_lecturer})
  ret = {title : my_class.title,
  coordinator : c_lec.name, guest_lecturer :
  g_lec.name}
  print(tojson(ret)) }
```

Summary (1)

- The read operation is defined from within the mongo shell
- It uses `db.collection.find()` method
- The method accepts: selection criteria, projection list, and modifiers as its arguments
- Selection ctiteria:
  - Comparison,
  - Existence,
  - Logical junctions (and, or,…),
  - Regular expressions,
  - Array selection objects

Summary (2)

- Projection list contains either a:
  - List of fields to return, or
  - List of fields not to return,
  - The only exception is `_id` field that may be marked as not to return among the list of fields to return
- The read operation returns a cursor to matching documents
  - In mongo shell, up to 20 first documents are displayed on the standard output
- Cursor can be used to write handy scripts