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

MongoDB
Write

Lecturer* : *Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for MongoDB Write Operations

- Insert
- Update,
- Delete

  - ***Reedings:***
    - *Have a look at Readings on the Home Page*

Write Operations

- There are three classes of write operations in MongoDB:
  - Insert that adds a new document to a collection,
  - Update that modifies an existing document, and
  - Remove that deletes an existing document from a collection.
  - The update and remove operations allow specifying criteria or conditions that identify documents (to be modified, or removed)
    - The syntax of the criteria is the same as in `find()` method
- All write operations are atomic on the level of a single document
- All examples on the following slides use MongDB methods in `mongo shell`

Insert Operation

- The following operations insert a new document into the collection `myclasses`:

```
var myclass = {code: "SWEN432",
               title: "Advanced DB",
               year: 2014,
               lastModified: "2014-02-28"}
```

```
db.myclasses.insert(myclass)
```

- The inserted document is going to have five fields: `_id`, `code`, `title`, `trimester`, and `year`
- MongoDB adds the field `_id` automatically and populates it with a unique `ObjectId`
  - Unless `_id` has been specified by the application

Insert Operation (continued)

- The operation returns a `WriteResult` object with the status of the operation
- A successful insert returns the number of documents inserted:

```
WriteResult({"nInserted": 1})
```

- An unsuccessful insert returns the error information
- If the collection did not exist before the insert operation, MongoDB would create the collection first and then insert the document
- An `insert` operation targets only one collection

Insert an Array of Documents

- An array of documents can be passed to the `insert()` method to insert each array element as a separate document

1. Create an array of documents:

```
var my_doc_array = [doc1, doc2,..., docn]
```

2. Insert documents:

```
db.myclasses.insert(my_doc_array);
```

3. The method returns a `BulkWriteResult` with the status of the operation

Modifying Documents

- MongoDB provides the `update()` method to update documents of a collection
- The method accepts as its parameters:
  - An update condition (criteria) document to select documents to update,
  - An update operation document to specify the modification to perform, and
  - An options document

```
db.collection.update(
{criteria}, {action}, {option})
```

- Update conditions (criteria) are specified using the same structure and syntax as the query conditions

Updating Specific Fields in a Document

- Mongo DB provides update operators to update values:
  - The `$set` operator is most frequently used, (if a field to be updated does not exist in the document, the `$set` operator creates it)
  - The `$push` operator inserts an elment in an array,
  - The `$currentDate` is useful

```
db.myclasses.update({'course.code': "SWEN432", year: 2014},
{$set: {'course.title': "Advanced Database Design and Implementation"},
$currentDate: {lastModified: true}})
```

- The update operation returns a `WriteResult` object containing the status of the operation

Multiple Document Update

- To update multiple document, the `multi` option has to be used:

```
db.myclasses.update(
{'course.code': {$regex: /^NWEN/}},
{$set: {year: 2015},
$currentDate: {lastModified: true}},
{multi: true}
)
```

- The update operation returns a `WriteResult` object

```
WriteResult({“nMatched”: 21,
“nUpserted: 0”, “nModified: 21”})
```

`upsert` Option

- By default, if no document matches the update query, the `update()` method does nothing
- However, by specifying the option `upsert: true`, the `update()` method:
  - Either updates matching document or documents, or
  - If no matching document exists, it inserts a new document using the update specification
- The new document is created by:
  - Using equality conditions in the update condition document,
  - Applying the update specification document, and
  - Generating a new `_id` field

upsert Example

```
db.myclasses.update(
{'course.code': "SWEN432", year: 2017},
{$set: {'course.title': "Cassandra",
lecturer: "Aaron"}},
{upsert: true}
)
```

```
WriteResult({
“nMatched”: 0,
“nUpserted”: 1,
“nModified”: 0,
“_id”: ObjectId(“53dbd684”)
})
```

*Updating Arrays of a Fixed Size (1)**

- The following procedure maintains arrays of fixed size after inserting new elements
- Assume Pavle is interested to keep (student, essay_score) pairs for only three best essays
- So, he modified the original SWEN432 document:

```
db.myclasses.update(
{code: "SWEN432"},
{$set: {essay: [
{name: "James", score: 83},
{name: "Lingshu", score: 80},
{name: "Matt", score: 79}
] } } )
```

*Updating Arrays of a Fixed Size (2)**

- After marking a new essay, Pavle updates the essay array using:
  - `$push` operator to insert a new element into the array,
  - `$each` modifier (needed in conjunction with `$sort` and `$slice`)
  - `$sort` modifier to order elements by descending scores, and
  - `$slice` modifier to keep the first three elements in an ordered array

```
db.myclasses.update(
{'course.code': "SWEN432", year: 2014},
{$push: {essay:
{$each:[{name: "Eileen", score: 82}],
$sort: {score: -1}, $slice: 3}
} } )
```

Renaming and Deleting a Field

- The `$rename` operator updates the name of a field

```
{$rename: { <field1>: <new_name1>, ...}}
```

```
db.myclasses.update(
{'course.code': "SWEN432"},
{$rename: {'students': 'enrolled'} } )
```

- The `$unset` operator deletes a particular field

```
{ $unset: { <field1>: "", ... } }
```

- The specified value in the `$unset` expression (i.e. `""`) does not impact the operation

```
db.myclasses.update(
{'course.code: "SWEN432"},
{$unset: {'coordinator': ""} } )
```

Delete Operation (1)

- In MongoDB, the `db.collection.remove()` method removes:
  - Either all documents from a collection, or
  - All documents that match a condition, or
  - Just a single document matching a condition
- The `remove()` method does not remove the indexes
- To remove all documents from a collection, it may be more efficient to use the `drop()` method, since it also removes indexes

Delete Operation (2)

- To delete all documents of a collection:

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

- To delete multiple documents that satisfy the remove criteria `{year: {$lt: 2014}}`:

```
db.myclasses.remove (
{year: {$lt: 2014}} )
```

- To delete only one document from the `myclasses` collection where the code field starts with SWEN:

```
db.myclasses.remove(
{code: {$regex: /^SWEN/}}, 1)
```

- To delete a single document sorted by some specified order, use the `findAndModify()` method

Comments on MongoDB Write and Update

- For all inserts and updates, MongoDB modifies each document in isolation
  - Clients never see documents in an intermediate state
- For multi-document operations, MongoDB does not provide any multi-document transactions or isolation

Summary

- The method `db.collection.insert()` writes a document into a collection
  - If there is no `_id` field specified in the document, MongoDB generates it
- The method `db.collection.update()` modifies an existing document, or may upsert a new one
- The `update()` method can also rename or delete a field
- The methods `db.collection.remove()` and `drop()` are used to delete `just` documents or both documents and indices of a collection, respectively