## Admin matters

- next 2 weeks

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1 Sept</td>
<td>Tues</td>
<td>L13: MongoDB document database&lt;br&gt;Essay presentation: topic 7</td>
</tr>
<tr>
<td></td>
<td>2 Sept</td>
<td>Wed</td>
<td>L14: Neo4j graph database&lt;br&gt;Essay presentation: topic 8</td>
</tr>
<tr>
<td></td>
<td>4 Sept</td>
<td>Fri</td>
<td>Practical 6a: Tutorial on MongoDB</td>
</tr>
<tr>
<td>8</td>
<td>8 Sept</td>
<td>Tues</td>
<td>L15: Neo4j graph database</td>
</tr>
<tr>
<td></td>
<td>9 Sept</td>
<td>Wed</td>
<td>L16: Neo4j graph database</td>
</tr>
<tr>
<td>11</td>
<td>11 Sept</td>
<td>Fri</td>
<td>Practical 6b: MongoDB document database</td>
</tr>
</tbody>
</table>
Outline

MongoDB

1. Aggregates
2. Operations on arrays (reading for the student)
3. Cursors and iteration
4. mongo shell functions
5. Java - MongoDB applications
6. Importing and exporting documents (reading for the student)

Reference: MongoDB 2.6.4 manual (documentation)
Example collection for aggregates

Listing of the myCol2 collection without the _id

> db.myCol2.find({ }, { _id: 0, title: 1, likes: 1 } )

{ "title" : "NoSQL", "likes" : 400 }
{ "title" : "MongoDB", "likes" : 100 }
{ "title" : "MongoDB", "likes" : 150 }
{ "title" : "MongoDB", "likes" : 50 }
{ "title" : "Neo4j", "likes" : 50 }
{ "title" : "Neo4j", "likes" : 150 }
{ "title" : "Neo4j", "likes" : 250 }
### aggregate() method

**Syntax:**
```
db.COLLECTION_NAME.aggregate(AGGR_OPERATION)
```

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sum</td>
<td>compute the sum</td>
</tr>
<tr>
<td>$avg</td>
<td>compute the average</td>
</tr>
<tr>
<td>$min</td>
<td>find the minimum value</td>
</tr>
<tr>
<td>$max</td>
<td>find the maximum value</td>
</tr>
</tbody>
</table>
**Aggregation (2): example - sum**

**Query:** compute the total number of ‘likes’ for each title

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
</table>
| $sum    | db.myCol2.aggregate(  
|          |     [   
|          |       { $group:  
|          |           [ _id: "$title",  
|          |               num_likes: { $sum: "$likes" }  
|          |           ]  
|          |     ]  )  |

**Results:**

{ "_id" : "Neo4j", "num_likes" : 450 }  
{ "_id" : "MongoDB", "num_likes" : 300 }  
{ "_id" : "NoSQL", "num_likes" : 400 }
Aggregation (3): pipeline

db.COLLECTION_NAME.aggregate(AGGR_OPERATION)

- Aggregation pipeline syntax:

  
  ```javascript
  aggregate([  
    { $match: <document criterion> },
    { $group: <group spec> },
    { $group: <group spec> },
    ......................................
    { $match: <group criterion> }
    { $sort: <sort spec> },
    ])
  ```
Aggregation (4): pipeline example

Query: compute the total number of ‘likes’ for the title ‘Neo4j’

db.myCol2.aggregate(
    [
        { $match: { title: "Neo4j" } },
        { $group: {
            _id: "$title",
            num_likes: { $sum: "$likes" }
        } }
    ]
)

Results: { "_id": "Neo4j", "num_likes" : 450 }
Aggregation (5): pipeline example

Query: compute the total number of ‘likes’ for the documents and show results in ascending order of number of likes.

db.myCol2.aggregate(
    [        { $group: 
                { _id: "$title",
                  num_likes: { $sum: "$likes"} }
           ,
                 {$sort: {"num_likes": 1} }
    ]
)

Results: 
    { "_id" : "MongoDB", "num_likes" : 300 }
    { "_id" : "NoSQL", "num_likes" : 400 }
    { "_id" : "Neo4j", "num_likes" : 450 }
Aggregation (6): pipeline example

Query: compute the total number of ‘likes’ for the documents and show results in ascending order of number of likes, and only those with a total > 300

db.myCol2.aggregate(
   [
      { $match: {} },
      { $group: { _id: "$title", num_likes: { $sum: "$likes" } } },
      { $match: { num_likes: { $gt: 300} } }
   ]
)

Results:
   { "_id" : "Neo4j", "num_likes" : 450 }
   { "_id" : "NoSQL", "num_likes" : 400 }

*** the second $match implements HAVING ***
Google’s MapReduce programming model was designed for processing large data sets in a massively parallel manner.

The programming model is based on these simple concepts:

i. iteration over the input;
ii. computation of key/value pairs from each piece of input;
iii. grouping of all intermediate values by key;
iv. iteration over the resulting groups;
v. reduction of each group.

Map function: takes an input pair and produces a set of intermediate key/value pairs.

Reduce function: accepts an intermediate key & a set of values for that key. It merges these values together to form a possibly smaller set of values.
Aggregation (8): map-reduce

```javascript
db.myCol2.mapReduce(
    function()
    {  emit (this.title, this.likes);  },

    function(key, values)
    {  return Array.sum(values)  },

    {
      query:  {title: "Neo4j" },
      out:    "likes_totals"
    }
)
```

- **map function**
- **reduce function**
- **query** empty query `{ }selects all results
- **results collection**
Aggregation (9): map-reduce

Output in shell:
{
    "result" : "likes_totals",
    "timeMillis" : 607,
    "counts" : {
        "input" : 3,
        "emit" : 3,
        "reduce" : 1,
        "output" : 1
    },
    "ok" : 1
}

Output data:
> show collections
blogs
likes_totals
myCol
myCol2
myCollection
mycol
system.indexes

> db.likes_totals.find()
{ "_id" : "Neo4j", "value" : 450 }
Single purpose aggregation operations:

> db.myCol2.count()
7
>
> db.myCol2.distinct( "title")
[ "NoSQL", "MongoDB", "Neo4j" ]
>
Array operations

- Reading for the student
- pages 86, 87 of MongoDB manual
## Cursors & iteration

**Cursors:** A cursor is a pointer to current db ‘record’

**Example:** *iterate over the Cursor with a Loop*

<table>
<thead>
<tr>
<th>Action</th>
<th>JavaScript statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <em>query</em> <code>testCollection</code> &amp; assign resulting cursor object to <code>var c:</code></td>
<td><code>var c = db.testCollection.find()</code></td>
</tr>
</tbody>
</table>
| 2. **Print** the full result set using a *while* loop | `while ( c.hasNext() )
printjson( c.next() )` |
| 3. **Manipulate** a cursor object as an *array*, e.g. print doc at array index 4 | `printjson( c[4] )` |

- `hasNext()` function returns true if the cursor has documents.
- `next()` method returns the next document.
- `printjson()` method renders the document in a JSON-like format.
1. A JavaScript function which creates new data e.g. for use in testing

```javascript
function insertData(dbName, colName, num) {
    var col = db.getSiblingDB(dbName).getCollection(colName);
    for (i = 0; i < num; i++) {
        col.insert({ x: i });
    }
    print(col.count());
}
```

2. Can store the function in file `.mongorc.js` for future use.

3. e.g. of function call: `insertData("mydb", "testCollection", 20)`
Creating & testing a Java-MongoDB application

1. Start the `mongod` server process
2. start the `mongo` client (to check if Java app is correctly creating and inserting data)

screenshots of `mongod` and `mongo`:
package testMongoDBpackage;

import java.io.*;
// available from mongo-2.10.1.jar available on COS326 website
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodbDBObject;
import com.mongodb.MongoClient;
import java.util.List;
import java.util.Set;
import static java.util.concurrent.TimeUnit.SECONDS;
public class MongoDBJDBC {

    public static void main( String args[] ) {

        try {
            // connect to mongodb server & perform db. operations
        }
        catch(Exception e) {
            System.err.println( e.getClass().getName() + 
                ": " + e.getMessage() );
        }
    }
}
// connect to mongodb server
MongoClient mongoClient =
    new MongoClient( "localhost" , 27017 );

// Now connect to a database
DB db = mongoClient.getDB( "mydb" );
System.out.println("Connect to database successfully");

//get collection names and print them
Set<String> colls = db.getCollectionNames();
for (String s : colls) {
    System.out.println(s);
}
//create a new collection or get existing collection
DBCollection coll = db.getCollection("testCollection");

//create a json object with an imbedded document
BasicDBObject doc =
    new BasicDBObject("name", "MongoDB")
    .append("type", "database")
    .append("count", 1)
    .append("info",
        new BasicDBObject("x", 203).append("y", 102));

//and insert into db collection
coll.insert(doc);

//get the doc from the db & print contents
DBObject myDoc = coll.findOne();
System.out.println(myDoc);
// insert multiple documents of the form {"key": value}
for (int key = 1; key <= 10; key++) {
    coll.insert(new BasicDBObject("key", key));
}

// count the documents in a collection.
System.out.println(coll.getCount() + " documents counted");

// get and print all docs using a cursor
DBCursor cursor = coll.find();
try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next());
    }
} finally {
    cursor.close();
}
//get and print one doc using a cursor
BasicDBObject query = new BasicDBObject("key", 7);

cursor = coll.find(query);

try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next());
    }
    System.out.println(cursor.next());
} finally {
    cursor.close();
}
//use a relational op. query and find to get and print docs
//the range is 5 < key <= 10:
query = new BasicDBObject("key",
    new BasicDBObject("$gt", 5)
    .append("$lte", 10));
cursor = coll.find(query);

try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next());
    }
} finally {
    cursor.close();
}
//to drop a collection

coll.drop();
Importing and exporting data (documents)

Mongo tools for import & export of data (documents)

- MS Windows:  `mongoimport.exe`
- `mongoexport.exe`

- Use these from the cmd (command) shell
  
  > `mongoimport collectionspec infilespec`
  > `mongoexport collectionspec outfilespec`

Note: it is also possible to import / export csv format
Importing data (documents)

- Import json data into a collection

   mongoimport  --db mydb
                 -- collection myCol
                 -- file  myfile.json  \n              \n   (all on one line)

   will read json objects from file myfile.json and insert them into existing
collection myCol in database mydb

OR:

   mongoimport  --db mydb
                 -- collection myCol
                 -- jsonArray
                 -- file  myfile.json  \n              \n   (all on one line)

   will read an array of  json objects from file myfile.json and insert the
individual objects into existing collection myCol in database mydb
Exporting data (documents)

- Export json data to a file in json format

```
mongoexport --db mydb
    -- collection myCol
    -- out  myfile.json
```

*(all on one line)*

will read json objects from existing collection myCol in database mydb and write them to file myfile.json

OR:

```
mongoexport --db mydb
    -- collection myCol
    -- jsonArray
    -- file  myfile.json
```

*(all on one line)*

will read json objects from existing collection myCol in database mydb and write them as an array in file myfile.json