Big Data Technologies

Polytech Nice Sophia – 2024/2025

Solution of Lab 5

➢ First steps in MongoDB

1. In MongoDB, databases hold collections of documents. To select a database to use, in the mongo shell, run the command “db”. You can switch to a non-existent database with the following command “use myNewDB” where “myNewDB” is the name of the new database. Execute the command “use myNewDB”.

   use myNewDB

2. To create a collection associated to the current database, execute the command:

   db.createCollection('firstCollection');

3. Insert the first JSON document into the “firstCollection” collection created above.

   db.firstCollection.insertOne({name:'John', age: 26, skill:'MongoDB'});

   For each inserted record, the field “_id” is automatically added. The field “_id” is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.

4. The insertion can be verified by using the command shown below:

   db.firstCollection.find();

   You can also use the command “show collections” to show all the collections of the database.

5. Insert a second JSON document with the information: “_id=19”, “name=Peter”, “age=28”, “skill=HBase” and “company=TheBigData”. Is the second JSON document consistent with the first one?

   db.firstCollection.insertOne({_id: 19, name:'Peter', age: 28, skill:'HBase', company:'TheBigData'});

6. Is it possible to insert two times the same JSON document?

   Yes, it is but the “_id” must be different! MongDB is a NoSQL database!

7. Insert the following JSON documents into the “firstCollection” collection:

name	age	skill	company	country
Edward	31	Java, MongoDB, Python		USA
Mary	24	Python	TheBigData	Italy
Julia	25	Java, HBase, Python	SaveTheWorld	France

```
db.firstCollection.insertMany([
  { name: "Edward", age: 31, skill: "Java, MongoDB, Python", country: "USA"},
  { name: "Mary" , age: 24, skill: "Python", company: "TheBigData ", country: "Italy"},
  { name: "Julia", age: 25, skill: "Java, HBase, Python", company: "SaveTheWorld ", country: "France"}
]);
```

8. Modify the document whose “name” is John by replacing the name by “Jane”. You can use the command

   ```
   db.firstCollection.updateOne( { _id: ObjectId('???') }, { $set: { name: 'Jane' } } )
   ```

   where ??? is the value of the ObjectID assigned to the document.

9. To execute a MongoDB equivalent of a SQL like clause, MongoDB uses regex. Regex is a series of characters forming a pattern to match.

   What is the meaning of the query: db.firstCollection.find({skill:/.*Java.*/});

   Find all workers whose skill contains “Java”.

10. What is the meaning of the query: db.firstCollection.find({age:{$lt:26}});

    Find all workers whose age is strictly lower than 26.

11. Create a query that selects all the workers whose skill contains “Java” and age is strictly lower than 26. Hints: explanations to write a query are given on the page

    https://docs.mongodb.com/manual/reference/method/db.collection.find

    ```
    db.firstCollection.find({$and: [{skill:/.*Java.*/},{age:{$lt:26}}]});
    ```

12. Use the command “deleteOne” to remove the document with “_id” equal to 19.

    ```
    try {
        db.firstCollection.deleteOne({ "_id" : 19 } );
    } catch (e) {
        print(e);
    }
    ```

13. Delete all the documents of the collection.

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

14. Remove the “firstCollection” collection.

    ```
    db.firstCollection.drop( )
    ```

15. Quit the mongo shell.

    ```
    quit()
    ```

➢ First steps in Cassandra

1. A keyspace is an object that is used to hold column families, user defined types. Create a keyspace named “bigdata” with a simple strategy and a replication factor equal to 1.

```
CREATE KEYSPACE bigdata
WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};
```

2. Run the command “DESCRIBE keyspaces;” to check if your keyspace exists.

```
DESCRIBE keyspaces;
```

3. To use the created keyspace, you must use the “USE” command.

```
USE bigdata;
```

4. Create a table named “student” with the columns “student_id” (32-bit integer), “student_name” (text), “student_city” (text), “student_fees” (integer of arbitrary size) and “student_phone” (integer of arbitrary size). The primary key is “student_id”.

```
CREATE TABLE student(
  student_id int PRIMARY KEY,
  student_name text,
  student_city text,
  student_fees varint,
  student_phone varint
  );
```

5. We need to insert some data in student table. Insert the following students (see the table) with the command

```
INSERT INTO student (student_id,student_fees,student_name) VALUES(1,5000,'Alice');
```

student_id	student_fees	student_name
1	5000	Alice
2	3000	Bob
3	2000	Christopher

```
INSERT INTO student (student_id, student_fees, student_name)
VALUES(1,5000, 'Alice');
INSERT INTO student (student_id, student_fees, student_name)
VALUES(2,3000, 'Bob');
INSERT INTO student (student_id, student_fees, student_name)
VALUES(3, 2000, 'Christopher');
```

6. You can use the “SELECT” command to verify whether data is inserted or not.

   SELECT * FROM student;

7. What are the values of the missing columns?

   The columns with no data contain “null”.

8. Create a query to get the name of the student whose id is 2.

   SELECT student_name FROM student WHERE student_id=2;

9. Update the fees of the student whose id is 3. The new value of the fees is 2500.

   UPDATE student SET student_fees=2500 WHERE student_id=3;

10. Delete the fees where student_id is 1.

    DELETE student_fees FROM student WHERE student_id=1;

11. Alter the table “student”: add a new column “student_email” (text).

    ALTER TABLE student
    ADD student_email text;

12. Drop the table “student”.

    DROP TABLE student;

13. Use the command “DESCRIBE tables;” to check that the table is removed.

    DESCRIBE tables;

14. Drop the keyspace “bigdata”.

    DROP keyspace bigdata;

➢ First steps in Redis

1. An introduction to the Redis commands is given in

   https://redis.io/topics/data-types-intro

   Create the key-value couple (“mykey”, “myvalue”)

   set "mykey" "myvalue" (or set mykey myvalue)

2. Get the value associated to the key “mykey”

   get mykey (or get “mykey”)

3. If you insert a new value with the same key, what is the result?

   The old value is replaced with the new value.

4. Insert the three following key-value couples in a single command: (k1, firstValue), (k2, secondValue) and (k3, thirdValue)

   mset k1 firstValue k2 secondValue k3 thirdValue

5. Get the two values associated to the keys k1 and k2 in a single command.

   mget k1 k2

6. List all the keys stored in the database.

   keys *

7. Delete the keys k1 and k3.

   del k1 k2

8. Store in the database the half-byte “1011” as a bitmap.

   setbit hbyte 0 1
   setbit hbyte 1 0
   setbit hbyte 2 1
   setbit hbyte 3 1

9. Quit the client.

   Quit

NB: you can use Redis from Spark. The insertion of complex key-value couple is easier, especially when the value is a bitmap (image, etc.).

➢ **First steps in Neo4j**

1. The node is surrounded with parentheses in Cypher, e.g. (node). If we later want to refer to the node, we can give it a variable like (p). You can assign labels to a node. Labels are kind of like tags and allow you to specify certain types of entities to look for or create. Create a node with variable “bigdata” and label “Topic”.

```
CREATE (bigdata:Topic)
RETURN bigdata;
```

2. You can assign one (or more) property to a node. A Property is a key-value pair. Create a node with variable “michael”, label “Person” and properties “name: 'Michael'” and “title: 'Expert'”.

```
CREATE (michael:Person { name: 'Michael', title: 'Expert' });
```

3. Create a note with variable “Jessica” and label “Person”.

```
CREATE (jessica:Person);
```

4. We can create a relationship between two nodes, Create the relationship with variable “rel” and type “LIKES” between “michael” and “bigdata”. A relationship type is similar to a node label.

```
CREATE (michael:Person { name: 'Michael', title: 'Expert' })-[rel:LIKES]->(bigdata:Topic);
```

5. Create the relationship with label “KNOWS” between “michael” and “jessica”.

```
CREATE (michael:Person { name: 'Michael', title: 'Expert' })-[:KNOWS]->(jessica:Person);
```

6. Create with a single command the two nodes “(jenn:Person {name: 'Jennifer'})” and (tbc:Company {name: 'TheBigCompany'}) with the relationship type “WORKS_FOR”. The sentence with the two nodes and the relationship is called a pattern.

```
CREATE (jenn:Person {name: 'Jennifer'})-[:WORKS_FOR]->(tbc:Company
{name: 'TheBigCompany'});
```

7. Find all the nodes. Hints: use the keywords “MATCH” and “RETURN”.

```
MATCH (n) RETURN n;
```

8. Find the labeled “Person” nodes in the graph.

```
MATCH (p:Person)
RETURN p;
```

9. Find Person nodes in the graph that have a title of 'Expert'.

```
MATCH (n:Person {title: 'Expert'})
RETURN n;
```

10. Find which Company Jennifer works for.

```
MATCH (:Person {name: 'Jennifer'})-[:WORKS_FOR]->(c:Company)
RETURN c;
```

11. Find the type of relationship between “michael” and “bigdata”.

```
MATCH (michael:Person { name: 'Michael', title: 'Expert' })-[r]->
(bigdata:Topic)
RETURN type(r);
```

12. Delete all nodes and relationships.

```
MATCH (n)
DETACH DELETE n;
```