Database Technologies Project – Phase 4

Overview

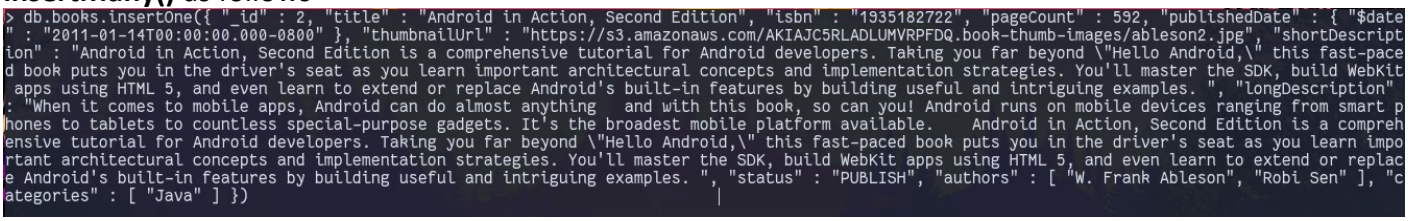
In the fourth phase of the project, we understand the usage of different kinds of NoSQL databases. The objectives of this phase are -

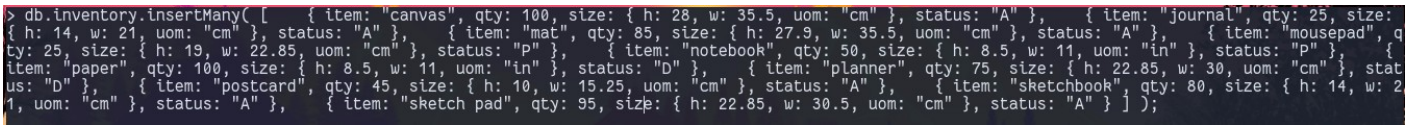
1. Demonstarate CRUD operations in a document store database (MongoDB)
2. Demonstarate CRUD operations in a key-value database (Redis)
3. Demonstrate creation of nodes and relations, and retrieval of related nodes in a graph database (Neo4j)

The entire project can be found at https://github.com/anihm136/DBMS_Project. Both MongoDB and Redis have been run on Docker containers, and Neo4j was run in Neo4j sandbox with the movies dataset preloaded

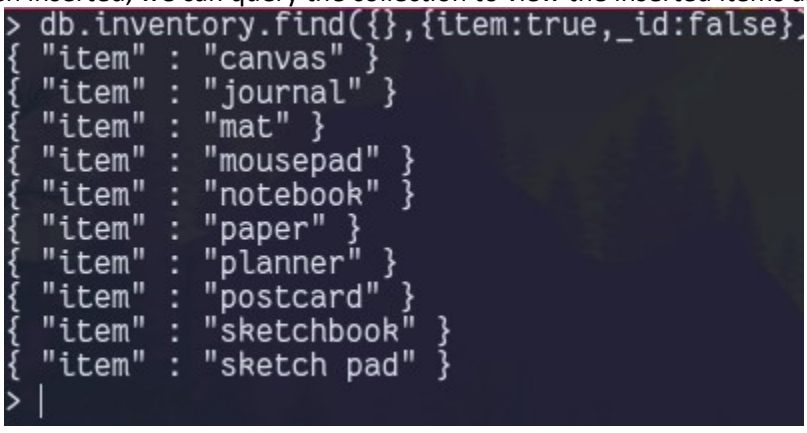
A. Document store database (MongoDB)

The dataset consists of a simple inventory of common stationery products. Each entry contains the name of the item and their dimensions, as well as a product status. Creation of the database and collections are automatically ahndled by MongoDB when a new document is created, therefore, the first operation is creation of a document. Documents may be created one at a time using **insertOne()**, or in bulk using **insertMany()** as follows -





Once the items have been inserted, we can query the collection to view the inserted items as follows -



Note that the **find** field is left empty to return all the documents, and only **item** is set to true to show only the names of the items

Updates can be performed on each document in the collection, either one at a time using **updateOne()** or in a batch using **updateMany()**, as follows -

```
> db.inventory.updateOne( {item:'paper'}, { $set:{'size.uom':'cm', status:'P'}, $currentDate:{lastModified:true} } )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

```
> db.inventory.updateMany(
... {'qty':{$lt:50}},
... {
... $set: {'size.uom':'in', status:'A'},
... }
... )
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

The **updateMany** command takes in a field to match, which can be set to a range using an operator as has been done above. The second field, using the **$set** operator, specifies the fields to update and the value to set them to. Note that, if multiple documents are matched in the condition for **updateOne,** only the first document is updated

The collection after the above updates appears as follows -

```
> db.inventory.find({},{item:true,status:true})                                    Flameshot
{ "_id" : ObjectId("5fc62455c4638fe79c8e991a"), "item" : "mat", "status" : "A" }
{ "_id" : ObjectId("5fc62455c4638fe79c8e991b"), "item" : "mousepad", "status" : "A" }
{ "_id" : ObjectId("5fc62455c4638fe79c8e991c"), "item" : "notebook", "status" : "P" }
{ "_id" : ObjectId("5fc62455c4638fe79c8e991d"), "item" : "paper", "status" : "P" }
{ "_id" : ObjectId("5fc62455c4638fe79c8e991f"), "item" : "postcard", "status" : "A" }
{ "_id" : ObjectId("5fc62455c4638fe79c8e9920"), "item" : "sketchbook", "status" : "A" }
{ "_id" : ObjectId("5fc62455c4638fe79c8e9921"), "item" : "sketch pad", "status" : "A" }
```

Documents may be deleted using the **deleteOne()** and **deleteMany()** commands as follows -

```
> db.inventory.deleteOne({item:'journal'})
{ "acknowledged" : true, "deletedCount" : 1 }
```

```
> db.inventory.deleteMany({status:'D'})
{ "acknowledged" : true, "deletedCount" : 2 }
```

Similarly to the update command, multiple matched documents in the **deleteOne** command only result in the deletion of a single document (the first one matched)

B. Key-value database (Redis)

Redis is an in-memory key-value store database, which also supports data persistence through asynchronous writes to secondary storage. It has a simple CLI which supports CRUD operations. All simple CRUD operations are shown in the following image -

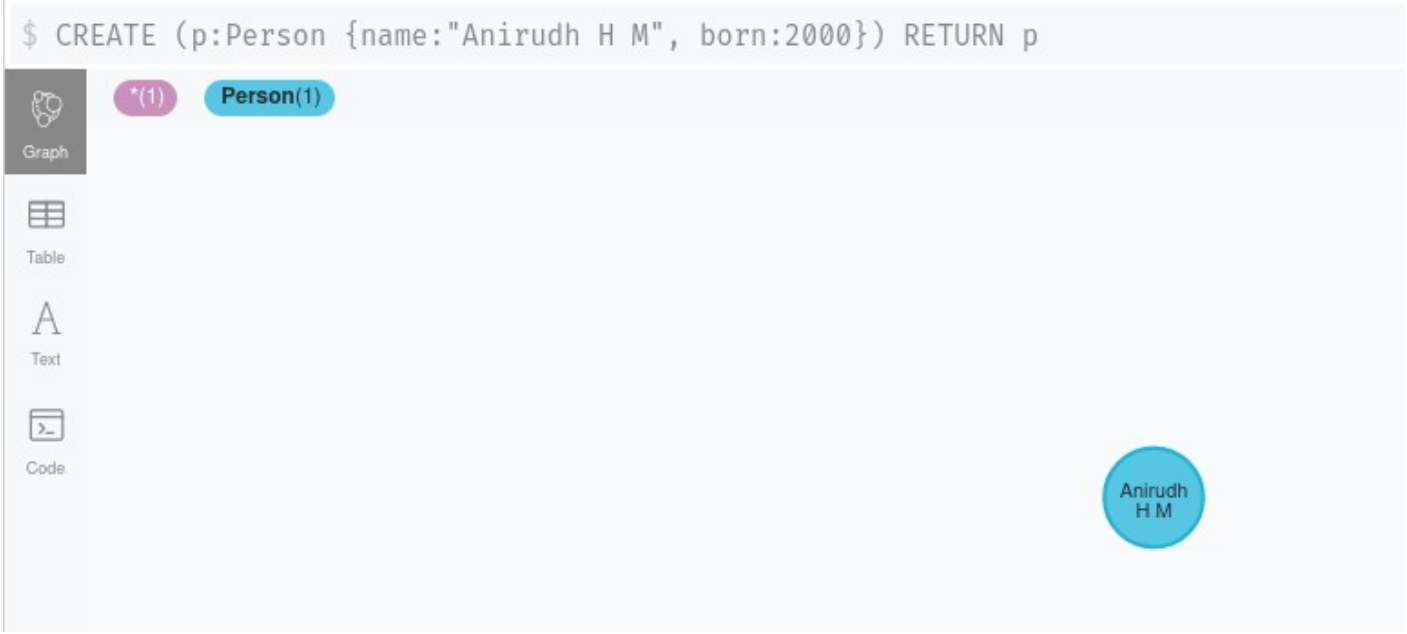
```
127.0.0.1:6379> set book1 "Mastering Emacs"
OK
127.0.0.1:6379> get book1
"Mastering Emacs"
127.0.0.1:6379> set book2 "Learning Vim the Hard Way"
OK
127.0.0.1:6379> get book2
"Learning Vim the Hard Way"
127.0.0.1:6379> set book1 "Mastering Emacs: Revisited"
OK
127.0.0.1:6379> get book1
"Mastering Emacs: Revisited"
127.0.0.1:6379> del book2
(integer) 1
127.0.0.1:6379> get book2
(nil)
127.0.0.1:6379>
```

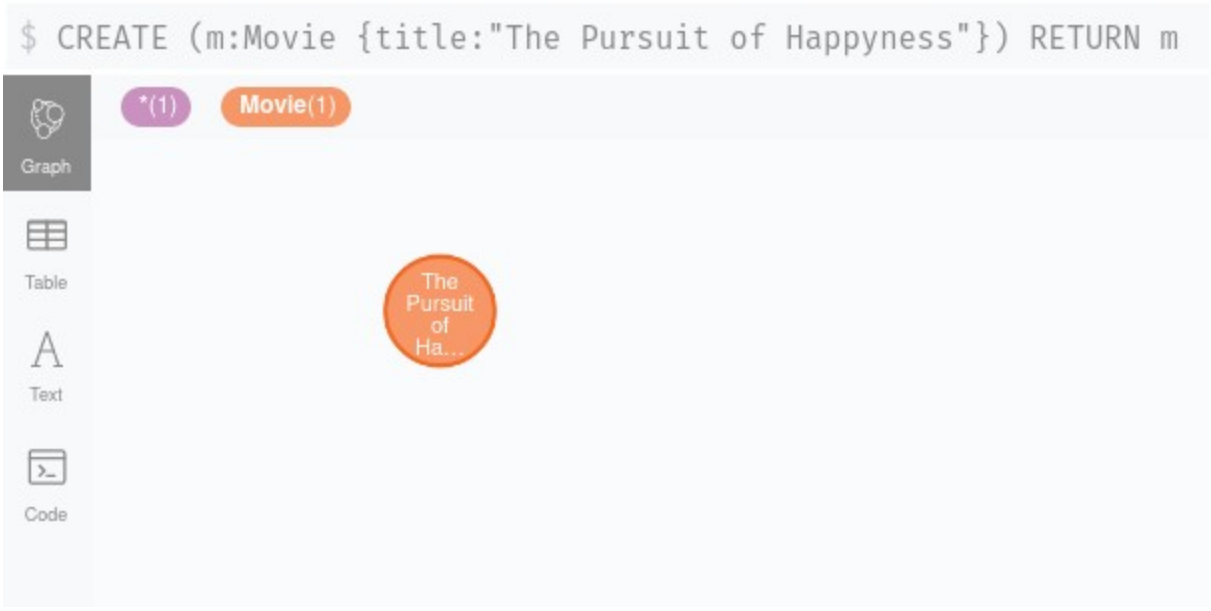
In addition, Redis supports many more advanced options, including a memory-only mode, multiple key-value stores, setting time limits for validity of keys and so on

C. Graph database (Neo4j)

Neo4j is a graph database that supports efficient storage of relations between entities (or nodes). The basic operations in a graph database such as Neo4j include creation and retrieval of nodes, creation and retrieval of relations and retrieval of a set of related nodes based on a matching pattern. Neo4j uses the cypher query language to perform operations on the database

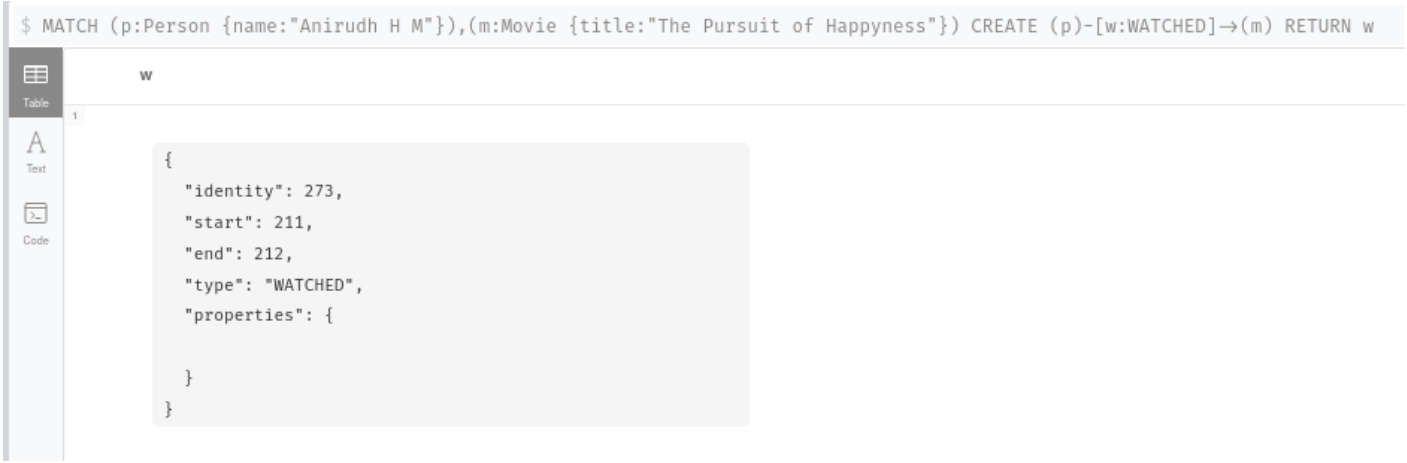
Creation of nodes is performed as follows -





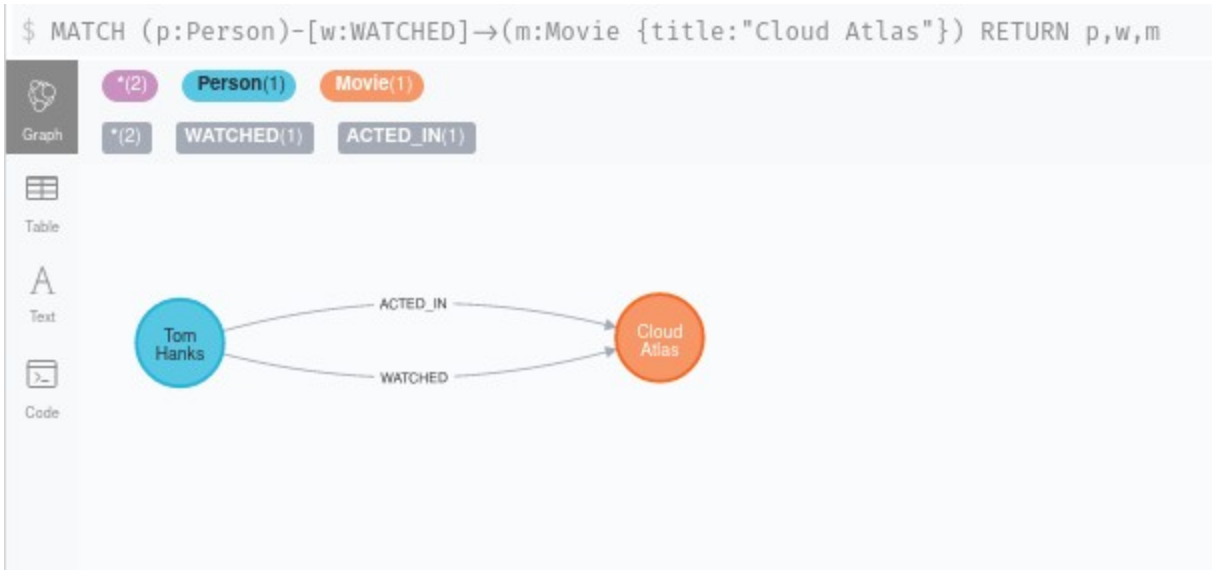
The basic components required for creating a node are the **label** (Person, Movie) and the **attributes** (name, born, title). The above cypher command creates a new node and returns the created node. In case the newly created node is expected to overwrite an existing node, and the desired behavior is to update the existing node, then the **MERGE** command can be used to create a new node if it does not exist, or update the existing one if it does.

New relations can be created between two nodes as follows -



The basic pattern for creating a new relation is to match nodes with specific patterns, and create relations with a given label between them. In the above example, we match a Person node with the given name and a Movie node with the given title, and create a new relationship with the label **WATCHED** between them, indicating that the person has watched the movie. This can be done with more general matching patterns as well, which match more than one node.

Querying based on relations is a unique feature of graph databases. The cypher query language allows us to specify a pattern of relations to match, and returns all nodes and relations which match the given pattern. This can be done as follows -



In the above example, we retrieve all the Persons who have watched the movie with the title “Cloud Atlas”. Incidentally, the Person “Tom Hanks” has also acted in the movie, which also shows up as a relation between the two nodes