MongoDB

Extension to Databases

Professor: Stanislav Vakaruk

stanislav.vakaruk@u-tad.com

Document-oriented database

- Json-like data modeling .
  - Document : The data realated to an entity.

- More accessible data modeling for humans compared to relational databases.
  - Possibility of grouping/nesting data within the same document (Aggregate model)

- Efficient document access by keys.
  - Identifier
  - Indexed variables.

Document-oriented database

- Schemaless

```
{_id: <Object1>,
 name : “ Perico ”,
 contact : [{
            phone : [“123-456-789”, “987-654-321”],
            email: “ perico@correo.com ”
                        }],
 address : [{
            street : “ Main street ”,
            number : 2
                        }]
}
```

Indexed

Document-oriented database

- Schema and type validations can be optionally set with $jsonSchema .

```
{ $ jsonSchema : {
        required: [ "name", "major", " gpa ", "address" ],
        properties: {
                name: {
                        bsonType : "string",
                        description: "must be a string and is required"
                },
        address: {
                bsonType : "object",
                required: [ " zipcode " ],
                properties: {
                    "street": { bsonType : "string" },
                    " zipcode ": { bsonType : "string" }
}}}}}
```

Performance

- Set **indexes** on variables to speed up the filter.
- Avoid the ussage of $lookup operations (i.e. join).
- Do **nest information** whenever the nested information is not excessive.
- Use **reverse indexing** (Atlas Search) when performing **text queries** (partial search or search with regular expressions).

Performance

- Signs of a **design problem:**
  - Database with **too many collections**
  - **Lists** with too many elements
  - The **amount of information** contained in a single document is **very large.**
  - You have **too many indexes** .

INDEXES

- **createIndex()**: Creates an index on the specified field if it has not been created already
  - Types of indexes:
    - Ascending (1), descending (-1)
    - text
    - 2d and 2dsphere
- **dropIndex()**: Removes an index
- **getIndexes()**: Gets indexes of a collection.

```
db.students.createIndex (
        { name : 1 },
        { unique :true}
)
```

CRUD: Reading

- **find** returns a cursor to the list of documents that meet the search criteria. **findOne** returns a single document.
- Syntax:

db.collection. **find** (<criteria>, <projection>)

db.collection. **findOne** (<criteria>, <projection>)

Mongodb / pymongo

Mongodb

```
db.students.find (
        { "name.first " : "Perico "},
        { name: 1 }
).sort( { name: 1} )
```

In mongosh:

```
var myCursor = db.users.find( { type: 2 } );

while (myCursor.hasNext()) {
printjson(myCursor.next());}
```

CRUD: Writing

- SYNTAX:

  db.collection. **insertOne** (<document>)

  db.collection. **insertMany** (<array of documents>)

  – Returns the id of the inserted document or the list of ids

- Examples :

```
db.students.insertOne ({ name : "Perico"})

db.students.insertMany (
        [{ name : "Parrot"},
         { name : "Paquito"},
         { name : "Luisito"}]
)
```

CRUD: Update

- SYNTAX:

  db.collection. **updateOne** (<query>,<document>,<options>)

  db.collection. **updateMany** (<query>,<document>,<options>)

  db.collection. **replaceOne** (<query>,<document>,<options>)

  - Options:
    - Upsert : if it does not exist, it inserts it
  - Returns the result of the operation

- Examples :

```
db.students.updateOne ( { name : "Perico"},
                        {$set: { name : ”Pablito”, age : 21}}
                        )
```

CRUD: Update

- SYNTAX:

```
db.collection. findAndModify ( query: <document>,
                               update: <document>,
                               remove: <Boolean>
                               ... )
```

  – Modifies/deletes and returns a single document. By default it returns the unmodified document. To return the modified object use new: true.

- Example :

```
db.students.findAndMmodify ( query : { name : "Perico"},
                             update : { branch : " Health "},
                             new: true )
```

CRUD: Elimination

- SYNTAX:

  db.collection. **deleteOne** (<query> , ...)

  db.collection. **deleteMany** (<query> , ...)

- Example :

  db.students.deleteOne ({ branch : " Health "}}

CRUD: Search (filter) operators

- Comparison
  - $gt : greater than
  - $gte : greater than or equal to
  - $in : in the list
  - $lt : less than
  - $lte : less than or equal to
  - $ne : different from
  - $nin : not in the list

```
db.students.find ( { age :{ $gt :18}} )
db.students.find ( { branch : {$in: [" Health ", "Science"]}} )
```

CRUD: Search (filter) operators

- Logical
  - $and
  - $nor
  - $not
  - $or

```
db.students.find ( { $and : [{ subject : "Math"},
                             { subject : " Science "}]} )
```

- Element
  - $exists : the field exists
  - $type : the field is of a certain type

```
db.students.find ( { class : {$ exists : true }} )
```

CRUD: Search (filter) operators

- Assessment
  - $mod : performs the field module and checks if it matches the one being searched for
  - $regex : search with regular expressions
    - Requires indexing. In Atlas use *full-text search*
  - $text : search for text in text type indexes
    - Requires indexing. In Atlas use *full-text search*
  - $where : search with JavaScript expressions

```
db.students.find ( {$text :{$search :" Perico "}} )
```

CRUD: Search (filter) operators

- Geospatial
  - $geoIntersects : documents that intersect a specified geometry (2dsphere)
  - $geoWithin : Documents that are within a specified geometry (2d and 2dsphere)
  - ordered list of documents near a point within the range $minDistance and $maxDistance (2d and 2dsphere)
    - Geodesic distance: $nearSphere
    - Cartesian distance: $near

```
db.students.find ({ loc : {
    $geoIntersects : {
        $geometry : {
            type : " Polygon ",
            coordinates : [
              [ [ 0, 0 ], [ 3, 6 ], [ 6, 1 ], [ 0, 0 ] ]
            ]
        }}}
```

GeoJSON object

longitude latitude

CRUD: Search (filter) operators

- Arrays
  - Normal query: documents with an array where at least one element matches the query.
  - $elemMatch : documents with an array where at least one element matches a multiple query.
  - $all : all elements of the list are in the array.
  - $size : array size matches the specified one.

```
db.students.find ( { grades :
                    { $ elemMatch :
                              {$ gt : 8, $ lte : 10}}
                    } )
```

CRUD: Reading , projection in arrays

- Limitation on arrays
  - $: Limits the result of a query on documents with arrays , returning only the first match in the array that satisfies the query.
  - $elemMatch : Limits the result of a query on documents with arrays by returning only the first match in the array based on a condition specified in the projection. Can return a document with an empty array.
  - $slice : Limits the result of a query to documents with arrays returning the documents within the specified indexes.

```
db.students.find ({ subject :
                    { $all :[" Biology ", " Maths "]}} ,
                  { " subject .$ " : 1 })
```

CRUD: Modification Operators

- Operators on fields:
  - $currentDate : Enter the current date
  - $inc : Increments the current value by the specified value
  - $max : Update if the field is greater than the specified
  - $min : Update if the field is less than the specified value
  - $mul : Multiplies the field by the specified value
  - $rename : Rename the field
  - $setOnInsert : Initializes or changes the value of a field if a new document is inserted.
  - $set : Initializes or changes the value of a field
  - $unset : Removes the field from the document.

- Example :

```
db.students.updateOne ({ name : "Perico"},
                       { $setOnInsert : { freshmen : true}},
                       { upsert : true })
```

CRUD: Modification Operators

- Array operators :
  - $: The operator affects the first match in a list field.
  - $[] : The operator affects all elements of a list field.
  - $addToSet : Adds an item to the list field if it does not already exist.
  - $pop : Removes the first or last item from a list field.
  - $pullAll : Removes all items from a list field that are in a specified list.
  - $pull : Removes all items from a list field that meet a condition
  - $push : Adds an element to a list field

- Example :

```
db.students.update ({ name : "Perico", subjects : " Maths "},
                    { $set:
                              {“ subjects .$”: " Applied Maths "}}
                    )
```

CRUD: Modification Operators

- Array Modifiers :
  - $each : Adds each of the elements of a list to a list field. Used with $addToSet and $push
  - $position: Inserts each of the elements of a list ($push + $each) into a specific position in a list field.
  - $slice : Limits the size of a list field during element insertion ($push + $each)
  - $sort : Sorts the elements of a list field during element insertion ($push + $each)
- Example :

```
db.students.update ({ name : "Perico"},
                                { $push : { sport_preferences : {
    $each : [{ name : "Football", priority : 2}
                { name : "Basket", priority : 1 }],
    $sort : { priority : 1 }}}})
```