7/01/22

ETSI
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
uhu.es

NoSQL

Information Systems: Design & Development

What is NoSQL?

NoSQL: Not only SQL

Do not follow the relational data model

Data is not structured in tables

Follow different data structures

Simplicity in the models: key-value, graphs, etc.

NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

The main difference lies in the way the data is stored. For example, storing an invoice

In a DBMS we have information in different tables

We use a programming language to transform this data into real-life objects in order to manipulate them.

In NoSQL, we simply store the invoice as such

NoSQL databases have the following properties:

- They have higher scalability.
- They use distributed computing.
- They are cost effective.
- They support flexible schema.
- They can process both unstructured and semi-structured data.
- There are no complex relationships, such as the ones between tables in an RDBMS.

Videos:

https://www.youtube.com/watch?v=qUV2j3XBRHc

https://www.youtube.com/watch?v=0buKQHokLK8

Characteristics

(i) Flexible Data Model:

- NoSQL databases are highly flexible as they can store and combine any type of data, both structured and unstructured, unlike relational databases that can store data in a structured way only.

(ii) Evolving Data Model :

- NoSQL databases allow you to dynamically update the schema to evolve with changing requirements while ensuring that it would cause no interruption or downtime to your application.

(iii) Elastic Scalability:

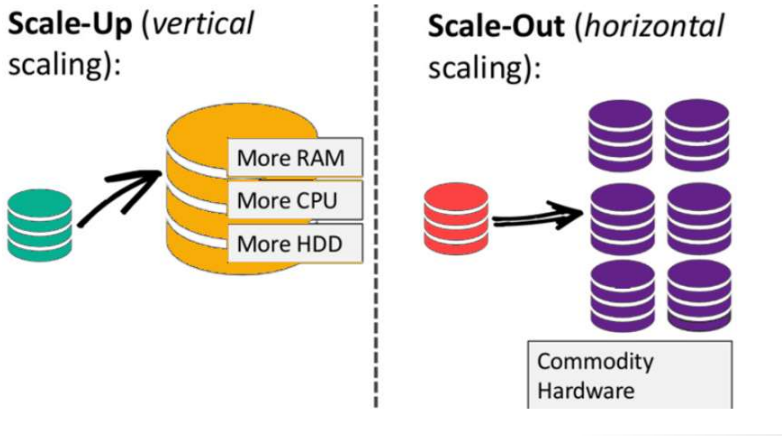
- NoSQL databases can scale to accommodate any type of data growth while maintaining low cost.

(iv) High Performance:

- NoSQL databases are built for great performance, measured in terms of both throughput (it is a measure of overall performance) and latency (it is the delay between request and actual response).

(v) Open-source:

- NoSQL databases don't require expensive licensing fees and can run on inexpensive hardware, rendering their deployment cost-effective.



Major disadvantages

(i) Lack of Standardization:

- There is no standard that defines rules and roles of NoSQL databases. The design and query languages of NoSQL databases vary widely between different NoSQL products – much more widely than they do among traditional SQL databases.

(ii) Backup of Database:

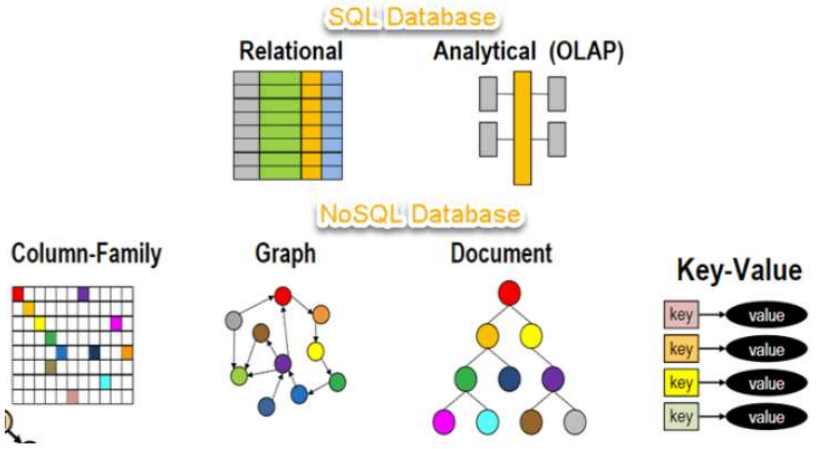
- Backups are a drawback in NoSQL databases. Though some NoSQL databases like MongoDB provide some tools for backup, these tools are not mature enough to ensure proper complete data backup solution.

(iii) Consistency:

- NoSQL puts a scalability and performance first but when it comes to a consistency of the data NoSQL doesn't take much consideration so it makes it little insecure as compared to the relational database e.g., in NoSQL databases if you enter same set of data again, it will take it without issuing any error whereas relational databases ensure that no duplicate rows get entry in databases.

NoSQL Database types

- **Key-value Pair Based :**
  - Redis, Tokyo, BerkeleyDB, JBoss cache, Velocity, Amazon Dynamo, Voldemort, Dynomite, SubRecord, . . .
- **Column-oriented Graph :**
  - Google BigTable, HBase, Cassandra, HyperTable, . . .
- **Document-oriented :**
  - CouchDB, MongoDB, Apache JackRabbit, ThruDB, . . .
- **Graphs based :**
  - Neo4j, VertexDB, Infogrid, Sones, Filament, Allegrograph, HyperGraphDB, . . .



MongoDB

- It is an open source NoSQL database
- The concept of **COLLECTION** is similar to that of **TABLE** in the relational model
- Tuples are **JSON** (JavaScript Object Notation) structures and are called **DOCUMENTS**
- MongoDB has several data management strategies that have positioned it where it is today:
- Your data division processes into different physical computers (clustering)
- Splitting very large documents into pieces that you store separately. When you retrieve the document, the driver automatically joins the document again

MongoDB

- The storage structure is very **flexible**
- Different documents in the same collection do not necessarily have to have the same fields or structure. Even documents with common fields don't necessarily have to have the same type of data.
- Comparison between SQL and MongoDB

http://docs.mongodb.org/manual/reference/sql-comparison/

CRUD operations

Create, Read, Update and Delete

https://www.youtube.com/watch?v=VELru-FCWDM

CRUD operations

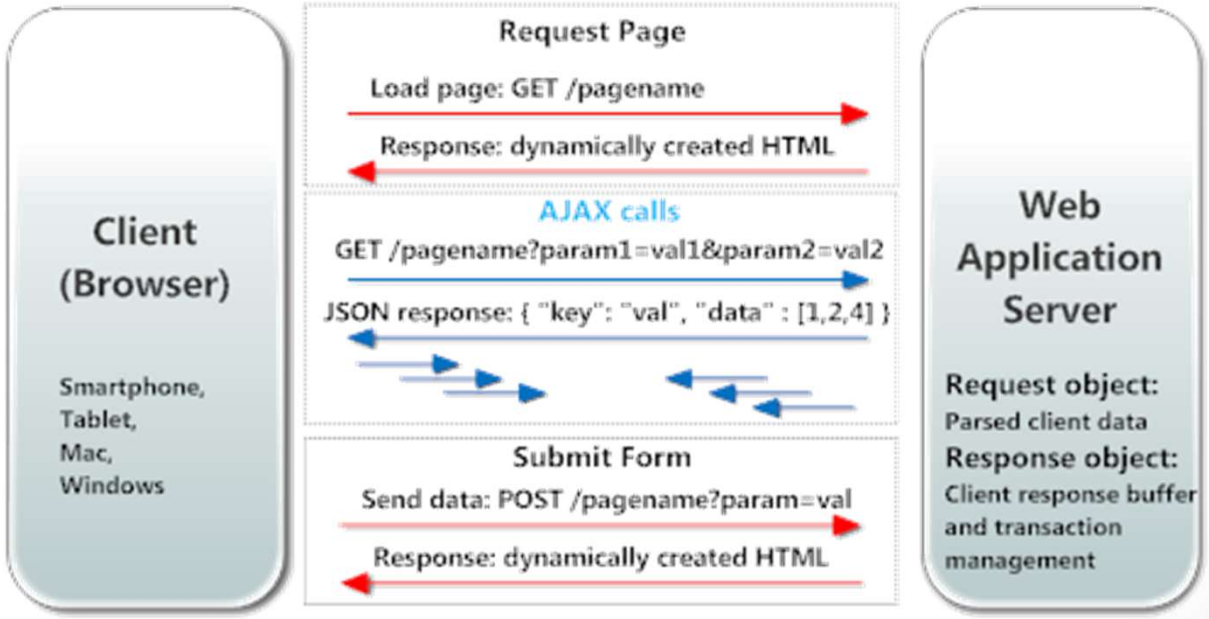
Create, Read, Update and Delete

- MongoDB stores data in a **JSON** format (http://www.json.org/json-es.html)
- Formally, documents in MongDB are BSON documents, that is, a binary representation of a JSON with some additional information
- Within a document, the value of a field can be of any type supported by BSON, including other documents, arrays, and document arrays
-

```
{
  name: "sue",                    <-- field: value
  age: 26,                        <-- field: value
  status: "A",                    <-- field: value
  groups: [ "news", "sports" ]    <-- field: value
}
```

{JSON}

- **JSON (JavaScript Object Notation)** is a format for data exchanges.
- One of the biggest advantages of using JSON is that it can be read by any programming language.
- Can be used for the exchange of information between different technologies



{JSON}

Name/Value Pair

To assign a name a value we must use the colon **':'** this separator is the equivalent of the equal symbol ('=') of any language

JSON values

The types of values that we can find ir

A number (integer or float)

A string (in single quotes)

A Boolean (true or false)

An array (in square brackets [])

An object (in curly braces {})

Null


```
{
    hey: "guy",
    anumber: 243,
  - anobject: {
        whoa: "nuts",
      - anarray: [
            1,
            2,
            "thr<h1>ee"
        ],
        more: "stuff"
    },
    awesome: true,
    bogus: false,
    meaning: null,
    japanese: "明日がある。",
    link: http://jsonview.com,
    notLink: "http://jsonview.com is great"
}
```

{JSON} tools

- **JsonViewer**

  http://jsonviewer.stack.hu/

  https://www.jsoneditoronline.org/

- **JsonGenerator**

  http://www.json-generator.com/

- **Netbeans**

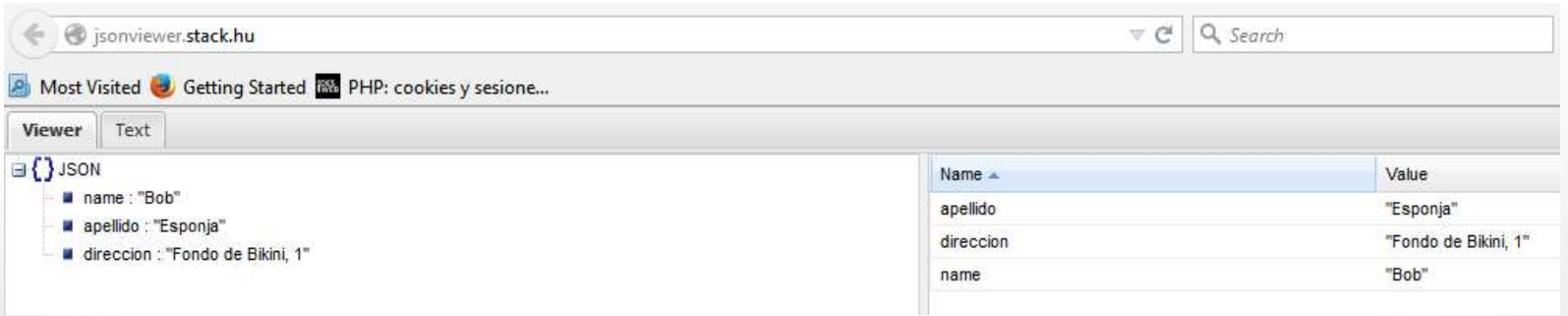
{JSON} Examples

```
{
   "name": "Bob"
   }
```

It is an object;
keys are used

separated by commas

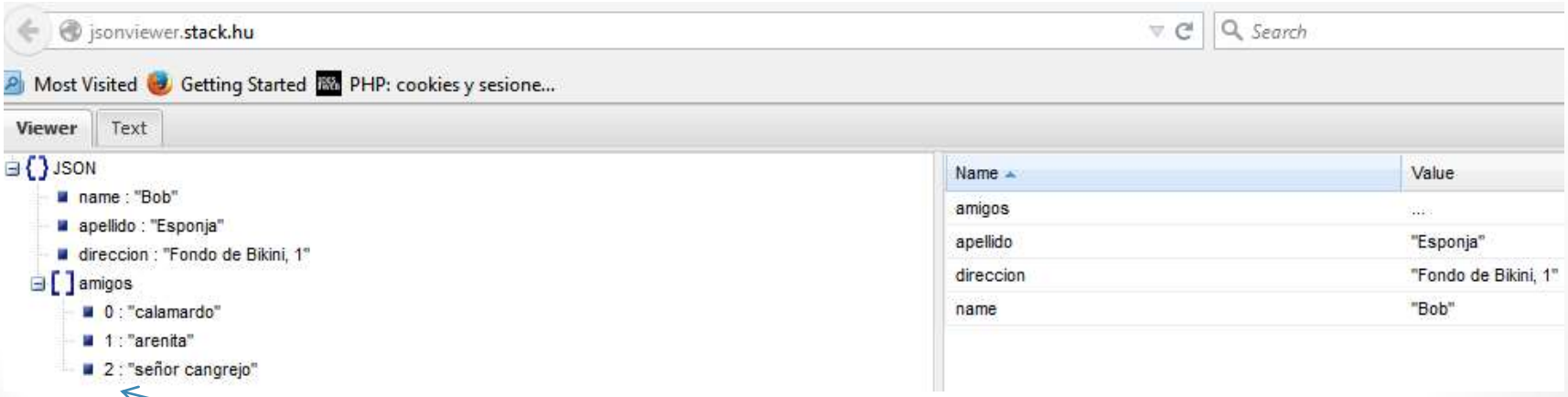
```
{
   "name": "Bob",
   "apellido": "Esponja",
   "direccion": "Fondo de Bikini, 1"
   }
```



{JSON} Examples

It is an array;
we use brakets

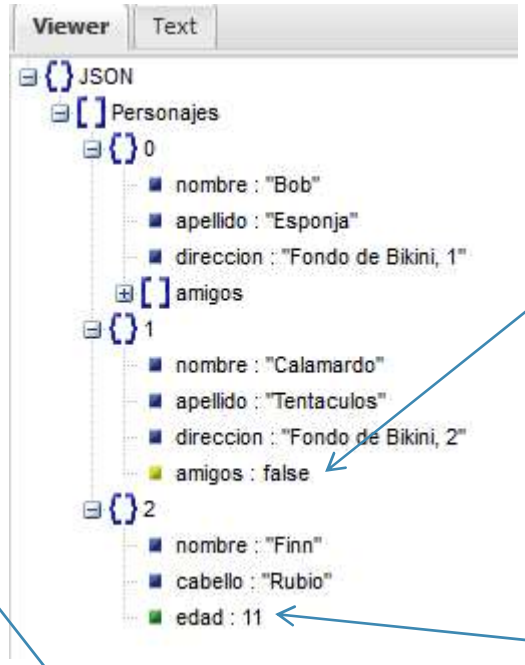
```
{
   "name": "Bob",
   "apellido": "Esponja",
   "direccion": "Fondo de Bikini, 1",
   "amigos": ["calamardo","arenita","señor cangrejo"]
   }
```



We will use this number when we want to reference it

{JSON} Examples

```
{
 "Personajes": [
  {
   "nombre": "Bob",
   "apellido": "Esponja",
   "direccion": "Fondo de Bikini, 1",
   "amigos": [
    "calamardo",
    "arenita",
    "señor cangrejo",
    "patricio"
   ]
  },
  {
   "nombre": "Calamardo",
   "apellido": "Tentaculos",
   "direccion": "Fondo de Bikini, 2",
   "amigos": false
  },
  {
   "nombre": "Finn",
   "cabello": "Rubio",
   "edad": 11
  }
 ]
}
```



Data type Boolean

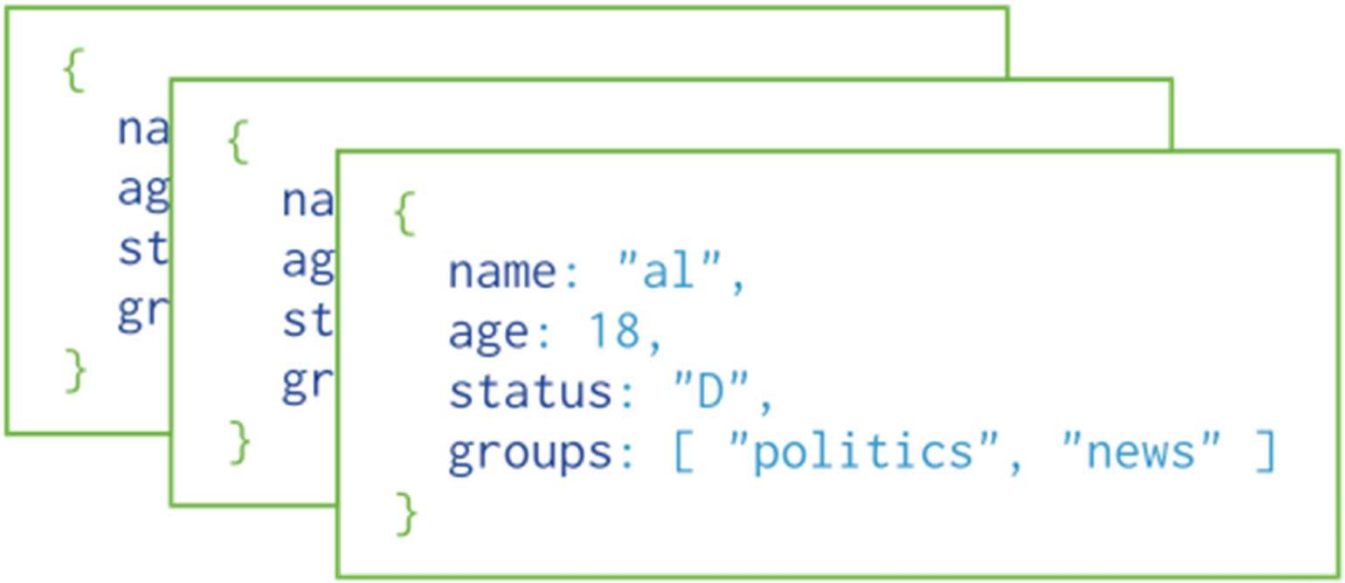
Data type integer

We create an array of characters

Not everyone has the same fields

CRUD Create, Read, Update and Delete

- Documents are stored in **collections**.
- A collection is a set of related **documents** that have indexes in common
- Can be considered as the concept of **table** in the relational model
-



CRUD operations – insert (deprecated)

- **db.collection.insert()** command adds new documents to a collection
- Operations are like *Javascript* functions

```
db.users.insert (
   {
      name: "sue",
      age: 26,
      status: "A"
   }
)
```

collection

field: value

field: value

field: value

document

```
INSERT INTO users
            ( name, age, status )
VALUES      ( "sue", 26, "A" )
```

table

columns

values/row

CRUD operations – insert (deprecated)

```
db.autores.insert ( {
            nombre:   'Andrés',
            apellido:   'Rodríguez',
            secciones:
              ['Cocina Fácil' , 'Postres']  } );
```

```
var autorDelPost = {
            nombre: 'Andrés',
            apellido: 'Rodríguez',
            secciones:
              ['Cocina Fácil' , 'Postres']  };

db.autores.insert (autorDelPost);
```

```
db.autores.insert ( {
            nombre: 'Amalia',
            apellido: 'González',
            secciones: ['Entrantes' , 'Cocina Fácil' , 'Peques' ],
            administrador:  true  } );

db.autores.insert ( {
            nombre: 'Alberto',
            apellido: 'Ruiz',
            secciones:  'Postres',
            genero:   "M"  } );
```

CRUD operations – insertOne InsertMany

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

```
db.products.insertOne( { _id: 10, item: "box", qty: 20 } )
```

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

```
db.products.insertMany( [
   { item: "card", qty: 15 },
   { item: "envelope", qty: 20 },
   { item: "stamps" , qty: 30 }
] );
```

CRUD operations - insert

- when a new document is inserted MongDB adds a **"_id"** field and assigns it a unique value (**object id**)
- the **_id** field is used as the default index
- you can **manually** specify the value of the _id field when you insert a record but you **must ensure** that this value is unique or duplicate primary key fails

```
db.people.insert({ _id: "Pedro", name: "Pedro", age: 25 });

db.people.insert({ _id: "Pedro", name: "Pedro", age: 25 });
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 11000,
                "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key error index:
                        test.people.$_id_  dup key: { : \"Pedro\" }"
        }
```

Operaciones CRUD - update

- **db.collection.update()** updates documents within a collection
- You can accept conditions to select the documents to update and options (such as multi to update multiple documents)

```
db.users.update(                          <-- collection
   { age: { $gt: 18 } },                  <-- update criteria
   { $set: { status: "A" } },             <-- update action
   { multi: true }                        <-- update option
)
```

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

CRUD operations - update

```
db.coleccion.update (
   filter,
   change,
   {
      upsert: booleano,
      multi:  booleano
   }
);
```

- **filter**: the search condition of the document(s) to be updated
- **change:** the changes to be made are specified. There are 2 types of updates:
- Modify an entire document for another
- Modify only the specified fields.

- **upsert:** (optional, false by default). if true and the filter does not find any results a new document is inserted with the "change"
- **multi:** (optional, false by default). If the filter returns more than one result and is specified as true, the change is made to all results. Otherwise, it will only occur in the first (the one with the lowest Id)

CRUD operations

- Example

```
db.autores.insert ( {
   nombre:  'Ricardo',
   apellido:   'Sanc'
});
```

```
db.autores.update (
   { nombre: 'Ricardo' },
   {
      nombre: 'Ricardo',
      apellido: 'Sánchez',
      secciones: ['Peques','Postres'],
      administrador:  true
   }
);
```

- in this case the entire document has been modified
- The overwrite operation of a document does not modify its unique identifier _id

CRUD operations – update

Operators:

To modify specific fields we must use modification operators:

$inc - increments a field of numeric type

$rename - rename document fields

$set - allows you to specify the fields to be modified

$unset - deletes fields from the document

More about all operators in:
http://docs.mongodb.org/manual/reference/operator/update/#id1

CRUD operations

- Examples:

```
{
  _id: 1,
  item: "TBD",
  stock: 0,
  info: { publisher: "1111", pages: 430 },
  tags: [ "technology", "computer" ],
  ratings: [ { by: "ijk", rating: 4 }, { by: "lmn", rating: 5 }
],
  reorder: false
}
```

```
db.books.update(
  { _id: 1 },
  {
    $inc: { stock: 5 },
    $set: {
      item: "ABC123",
      "info.publisher": "2222",
      tags: [ "software" ],
      "ratings.1": { by: "xyz", rating: 3 }
    }
  }
)
```

```
{
  "_id" : 1,
  "item" : "ABC123",
  "stock" : 5,
  "info" : { "publisher" : "2222", "pages" : 430 },
  "tags" : [ "software" ],
  "ratings" : [ { "by" : "ijk", "rating" : 4 }, { "by" : "xyz", "rating" : 3 } ],
  "reorder" : false
}
```

CRUD operations

Modification Operators (arrays)

$pop - deletes the first or last value of an array

$pull - removes values from an array that comply with the filter

$pullAll - deletes the specified values from an array

$push - adds an element to an array

$addToSet - add elements to an array only if they do not already exist (ensures that no duplicate elements are added)

$each - used in conjunction with $addToSet or $push to specify that multiple elements are to be added to the array

CRUD operations

- Examples:

```
{ _id: 1, scores: [ 8, 9, 10 ] }
```

```
db.students.update( { _id: 1 }, { $pop: { scores: -1 } } )
```

```
{ _id: 1, scores: [ 9, 10 ] }
```

```
{ _id: 1, scores: [ 0, 2, 5, 5, 1, 0 ] }
```

```
db.survey.update( { _id: 1 }, { $pullAll: { scores: [ 0, 5 ] } } )
```

```
{ "_id" : 1, "scores" : [ 2, 1 ] }
```

```
db.students.update (
  { _id: 1 },
  { $push: { scores: 89 } }
)
```

```
db.students.update(
  { name: "joe" },
  { $push: { scores: { $each: [ 90, 92, 85 ] } } }
)
```

CRUD operations

- Examples:

```
{ "_id": 1,
  "alias": [ "The American Cincinnatus", "The American Fabius" ],
  "mobile": "555-555-5555",
  "nmae": { "first" : "george", "last" : "washington" }
}
```

```
db.students.update( { _id: 1 }, { $rename: { "nmae": "name" } } )
```

```
db.students.update( { _id: 1 }, { $rename: { "name.first": "name.fname" } } )
```

```
{
  "_id" : 1,
  "alias" : [ "The American Cincinnatus", "The American Fabius" ],
  "mobile" : "555-555-5555",
  "name" : { "fname" : "george", "last" : "washington" }
}
```

CRUD operations

- For more information about update() command, see:
- http://docs.mongodb.org/manual/reference/method/db.collection.update/#db.collection.update

CRUD operations – delete (deprecated)

- **db.collection.remove()** deletes documents from a collection
- Accepts filters to select documents to delete

```
db.users.remove(          <-- collection
   { status: "D" }        <-- remove criteria
)
```

```
DELETE FROM users         <-- table
WHERE  status = 'D'       <-- delete criteria
```

- More information:

http://docs.mongodb.org/manual/reference/method/db.collection.remove/#db.collection.remove

CRUD operations – deleteOne, deleteMany

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

`db.collection.deleteOne()` deletes the first document that matches the filter. Use a field that is part of a unique index such as `_id` for precise deletions.

```
db.orders.deleteOne( { "_id" : ObjectId("563237a41a4d68582c2509da") } );
```

```
db.orders.deleteOne( { "expiryts" : { $lt: ISODate("2015-11-01T12:40:15Z") } } )
```

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

```
db.orders.deleteMany( { "client" : "Crude Traders Inc." } );
```

CRUD operations - queries

- **db.collection.find()** is used for queries
- Accepts selection and projection criteria and returns a cursor with the retrieved documents

```
db.users.find(                          <-- collection
    { age: { $gt: 18 } },               <-- query criteria
    { name: 1, address: 1 }             <-- projection
).limit(5)                              <-- cursor modifier
```

```
SELECT _id, name, address   <-- projection
FROM   users                <-- table
WHERE  age > 18             <-- select criteria
LIMIT  5                    <-- cursor modifier
```

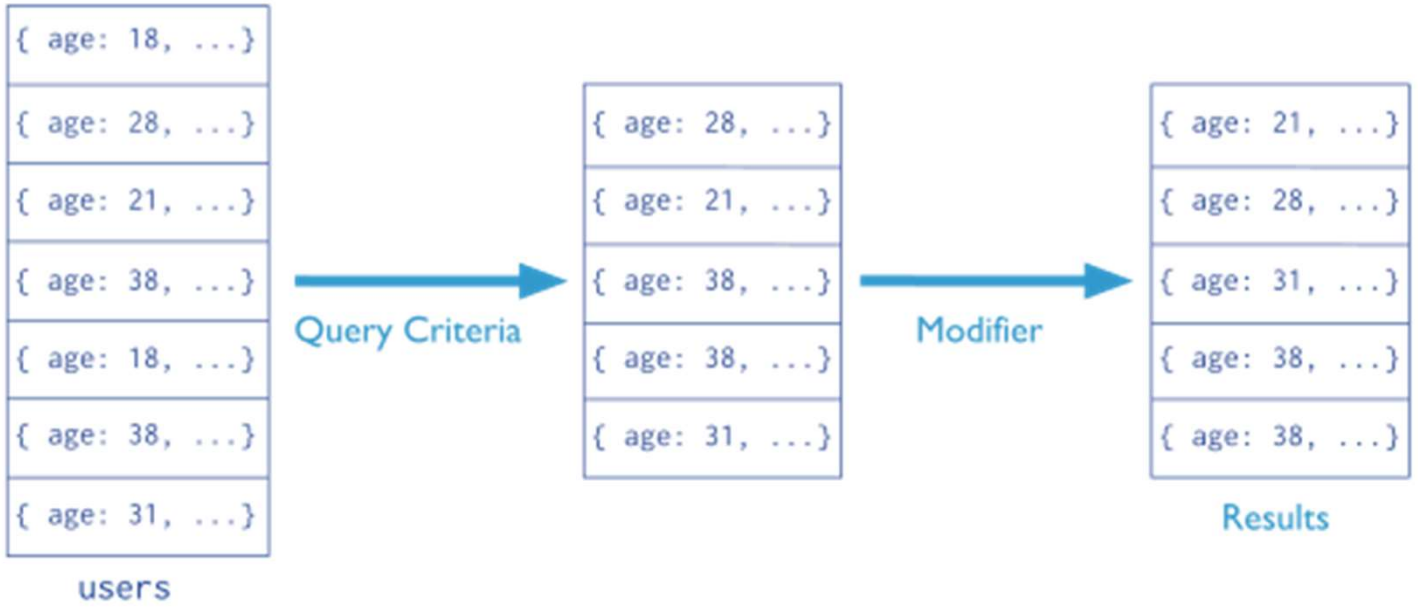
CRUD operations - queries

Collection

Query Criteria

Modifier

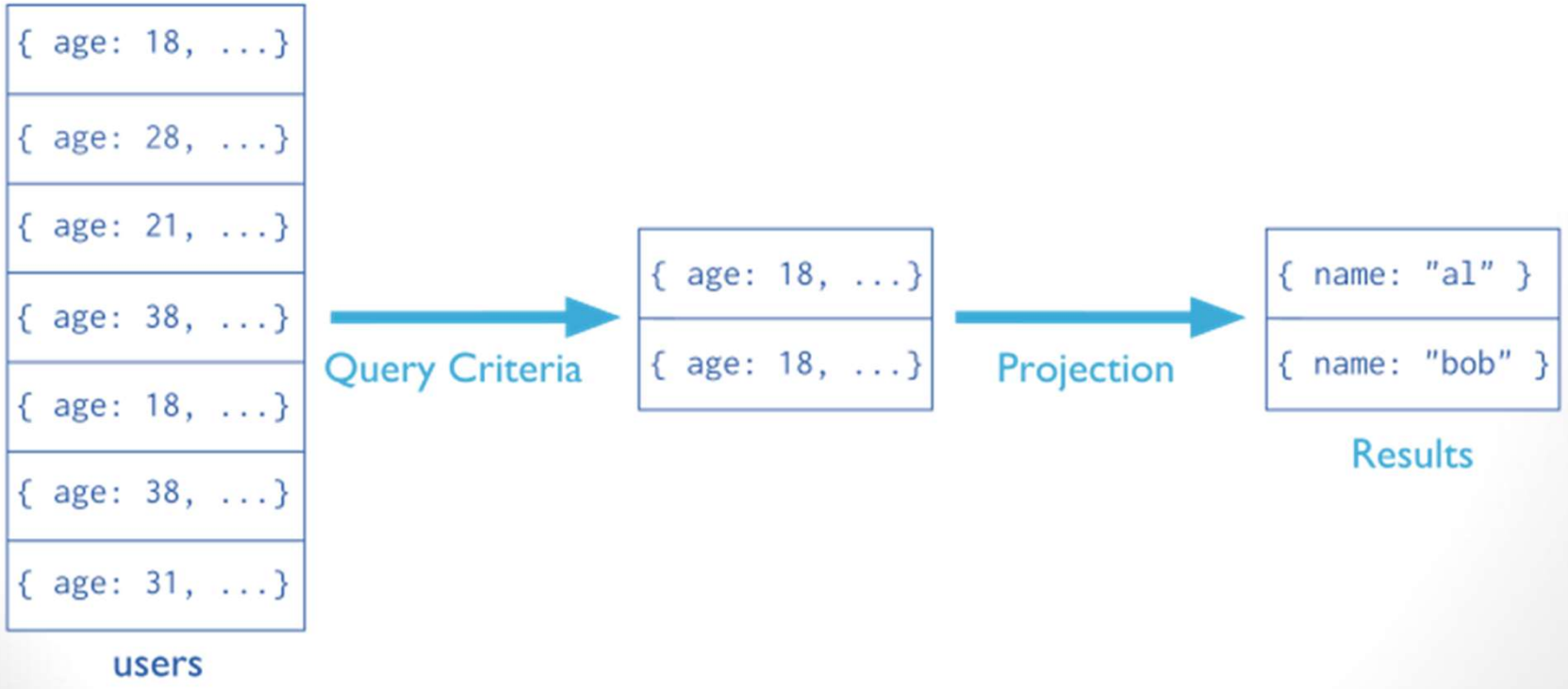
```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```



CRUD operations - queries

Collection Query Criteria Projection

```
db.users.find( { age: 18 }, { name: 1, _id: 0 } )
```



CRUD operations - queries

- More examples:

```
db.records.find( { "user_id": { $lt: 42 } }, { "history": 0 } )
```

Displays all fields except "history"

```
db.records.find( { "user_id": { $lt: 42 } }, { "name": 1, "email": 1 } )
```

Displays the "name" and "email" fields and includes the "_id" field

```
db.records.find( { "user_id": { $lt: 42} }, { "_id": 0, "name": 1 , "email": 1 } )
```

Same as the previous one but not including the "_id" field

CRUD operations - queries

- Selection and projection operators:

http://docs.mongodb.org/manual/reference/operator/query/

- Also, it can be used:
- sort(): http://docs.mongodb.org/manual/reference/method/cursor.sort/#cursor.sort
- limit(): http://docs.mongodb.org/manual/reference/method/cursor.limit/#cursor.limit
- skip(): http://docs.mongodb.org/manual/reference/method/cursor.limit/#cursor.limit

- More information and examples:
- https://www.guru99.com/mongodb-tutorials.html