08 CRUD Cursores

Iterate a Cursor in the mongo Shell

In the mongo shell, when you assign the cursor returned from the find() method to a variable using the var keyword, the cursor does not automatically iterate.

Práctica

```
> use gimnasio
> let clientes = db.clientes.find()

> while (clientes.hasNext()) { print(tojson(clientes.next()));} // Vacía la variable

> clientes = db.clientes.find()

> while (clientes.hasNext()) { printjson(clientes.next());}

> clientes = db.clientes.find()

> clientes.forEach(printjson)

> clientes = db.clientes.find()

> let arrayClientes = clientes.toArray()
> arrayClientes[2]
{
        "_id" : ObjectId("5e275f111d4ba0ea8964cde7"),
        "nombre" : "Carlos",
        "apellidos" : "Pérez Góngora"
}
```

Método count()

https://docs.mongodb.com/manual/reference/method/cursor.count/#cursor.count

Counts the number of documents referenced by a cursor. Append the count() method to a find() query to return the number of matching documents. The operation does not perform the query but instead counts the results that would be returned by the query.

Sintaxis

db.collection.find(<query>).count()

Práctica

> db.titulos.find({autor: /Mario/}).count()

Método limit()

https://docs.mongodb.com/manual/reference/method/cursor.limit/#cursor.limit

Use the limit() method on a cursor to specify the maximum number of documents the cursor will return.

Sintaxis

db.collection.find(<query>).limit(<number>)

Práctica

> db.titulos.find({}).limit(3)

Método skip()

https://docs.mongodb.com/manual/reference/method/cursor.skip/

Call the cursor.skip() method on a cursor to control where MongoDB begins returning results. This approach may be useful in implementing paginated results.

Sintaxis

db.collection.find(<query>).skip(<number>)

Práctica

> db.titulos.find({}).skip(2)

Método sort()

https://docs.mongodb.com/manual/reference/method/cursor.sort/

Specifies the order in which the query returns matching documents. You must apply sort() to the cursor before retrieving any documents from the database.

Sintaxis

db.collection.find(<query>).sort(<document>)

Práctica

> db.titulos.find({}).sort({autor: -1})

> db.titulos.find({}).sort({autor: -1, titulo: 1})

Método distinct()

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

Finds the distinct values for a specified field across a single collection or view and returns the results in an array.

Sintaxis

db.collection.distinct(field, query, options)

Options
{ collation: <document> }

Práctica

> db.clientes.distinct("apellidos")

```
[
        "Pérez",
        "Gómez",
        "López",
        "González",
        "Gutierrez",
        "Martínez",
        "Sánchez",
        "García",
        "García Pérez",
        "Gutiérrez López",
        "López Gómez",
        "Pérez Góngora",
        "Pérez \nGóngora",
        "Friedrich Hieronymus, \nbarón de Münchhausen",
        "Friedrich Hieronymus. \nbarón de Münchhausen"
]
```

Método collation()

https://docs.mongodb.com/manual/reference/method/cursor.collation/index.html

Specifies the collation for the cursor returned by the db.collection.find(). To use, append to the db.collection.find().

Sintaxis

db.collection.find().collation(<collation document>)

The cursor.collation() accepts the following collation document:

```
{
  locale: <string>,
  caseLevel: <boolean>,
  caseFirst: <string>,
  strength: <int>,
  numericOrdering: <boolean>,
  alternate: <string>,
  maxVariable: <string>,
  backwards: <boolean>
}
```

When specifying collation, the locale field is mandatory; all other collation fields are optional.

Field	Type	Description
locale	string	The ICU locale. To specify simple binary comparison, specify locale value of "simple".
strength	integer	Optional. The level of comparison to perform. Corresponds to ICU Comparison Levels. Possible values are: Value Description 1 Primary level of comparison. Collation performs comparisons of the base characters only, ignoring other differences such as diacritics and case. 2 Secondary level of comparison. Collation performs comparisons up to secondary differences, such as diacritics. That is, collation performs comparisons of base characters (primary differences) and diacritics (secondary differences). Differences between base characters takes precedence over secondary differences. 3 Tertiary level of comparison. Collation performs comparisons up to tertiary differences, such as case and letter variants. That is, collation performs comparisons of

		base characters (primary differences), diacritics (secondary differences), and case and variants (tertiary differences). Differences between base characters takes precedence over secondary differences, which takes precedence over tertiary differences. This is the default level. 4 Quaternary Level. Limited for specific use case to consider punctuation when levels 1-3 ignore punctuation or for processing Japanese text. 5 Identical Level. Limited for specific use case of tie breaker.
caseLevel	boolean	Optional. Flag that determines whether to include case comparison at strength level 1 or 2. If true, include case comparison; i.e. ● When used with strength:1, collation compares base characters and case. ● When used with strength:2, collation compares base characters, diacritics (and possible other secondary differences) and

		case. If false, do not include case comparison at level 1 or 2. The default is false.
caseFirst	string	Optional. A field that determines sort order of case differences during tertiary level comparisons. Possible values are: Value Description “upper” Uppercase sorts before lowercase. “lower” Lowercase sorts before uppercase. “off” Default value. Similar to "lower" with slight differences. See http://userguide.icu-project.org/collation/customization for details of differences. numericOrdering boolean Optional. Flag that determines whether to compare numeric strings as numbers or as strings. If true, compare as numbers; i.e. "10" is greater than "2". If false, compare as strings; i.e. "10" is less than "2". Default is false.
alternate	string	Optional. Field that determines whether collation should consider

		whitespace and punctuation as base characters for purposes of comparison. Possible values are: Value Description "non-ignorable" Whitespace and punctuation are considered base characters. "shifted" Whitespace and punctuation are not considered base characters and are only distinguished at strength levels greater than 3. See ICU Collation: Comparison Levels for more information. Default is "non-ignorable".
maxVariable	string	Optional. Field that determines up to which characters are considered ignorable when alternate: "shifted". Has no effect if alternate: "non-ignorable" Possible values are: Value Description "punct" Both whitespaces and punctuation are “ignorable”, i.e. not considered base characters.

		"space" Whitespace are “ignorable”, i.e. not considered base characters. backwards boolean Optional. Flag that determines whether strings with diacritics sort from back of the string, such as with some French dictionary ordering. If true, compare from back to front. If false, compare from front to back. The default value is false.
normalization	boolean	Optional. Flag that determines whether to check if text require normalization and to perform normalization. Generally, majority of text does not require this normalization processing. If true, check if fully normalized and perform normalization to compare text. If false, does not check. The default value is false.

Práctica

```
> db.asociaciones.insert({apellidos: "López"})
WriteResult({ "nInserted" : 1 })
> db.asociaciones.insert({apellidos: "Lopez"})
WriteResult({ "nInserted" : 1 })
> db.asociaciones.insert({apellidos: "lópez"})
WriteResult({ "nInserted" : 1 })
> db.asociaciones.insert({apellidos: "lopez"})
WriteResult({ "nInserted" : 1 })
```

Opción locale

Establece un orden específico por defecto para cada idioma:

```
> db.asociaciones.find({},{apellidos: 1, _id:0}).sort({apellidos: 1})
{ "apellidos" : "Lopez" }
{ "apellidos" : "López" }
{ "apellidos" : "lopez" }
{ "apellidos" : "lópez" }

> db.asociaciones.find({},{apellidos: 1, _id:0}).collation({locale:"es", strength:
1}).sort({apellidos: 1})
{ "apellidos" : "López" }
{ "apellidos" : "Lopez" }
{ "apellidos" : "lópez" }
{ "apellidos" : "lopez" }
```

Opción strength: 1

Ignora diacríticos y case sensitive

```
> db.asociaciones.find({apellidos: "LóPeZ"}).collation({locale:"es", strength: 1})
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }
{ "_id" : ObjectId("5e31ac333fdb41f878dbe2c5"), "apellidos" : "Lopez" }
{ "_id" : ObjectId("5e31affd3fdb41f878dbe2c6"), "apellidos" : "lópez" }
{ "_id" : ObjectId("5e31b0023fdb41f878dbe2c7"), "apellidos" : "lopez" }
```

Opción strength: 2

Ignora solo case sensitive

```
> db.asociaciones.find({apellidos: "LóPeZ"}).collation({locale:"es", strength: 2})
```

{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }
{ "_id" : ObjectId("5e31affd3fdb41f878dbe2c6"), "apellidos" : "lópez" }

Opción strength: 3 <por defecto>
case sensitive y diacritics sensitive

> db.asociaciones.find({apellidos: "López"}).collation({locale:"es", strength: 3})
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }

Opción caseLevel

Para el nivel 1 de strength vuelve a tener en cuenta case sensitive (para que solo ignore diacríticos)

> db.asociaciones.find({apellidos: "López"}).collation({locale:"es", strength: 1, caseLevel: true})
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }
{ "_id" : ObjectId("5e31ac333fdb41f878dbe2c5"), "apellidos" : "Lopez" }

Para el nivel 2 añade también case sensitive así que sería lo mismo que nivel 3

> db.asociaciones.find({apellidos: "López"}).collation({locale:"es", strength: 2, caseLevel: true})
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }

Opción caseFirst

> db.asociaciones.find({}).collation({locale:"es", caseFirst: "upper"}).sort({apellidos: 1})
{ "_id" : ObjectId("5e31ac333fdb41f878dbe2c5"), "apellidos" : "Lopez" }
{ "_id" : ObjectId("5e31b0023fdb41f878dbe2c7"), "apellidos" : "lopez" }
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }
{ "_id" : ObjectId("5e31affd3fdb41f878dbe2c6"), "apellidos" : "lópez" }

> db.asociaciones.find({}).collation({locale:"es", caseFirst: "lower"}).sort({apellidos: 1})
{ "_id" : ObjectId("5e31b0023fdb41f878dbe2c7"), "apellidos" : "lopez" }
{ "_id" : ObjectId("5e31ac333fdb41f878dbe2c5"), "apellidos" : "Lopez" }
{ "_id" : ObjectId("5e31affd3fdb41f878dbe2c6"), "apellidos" : "lópez" }
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }

en strength 1 exige caseLevel: true

> db.asociaciones.find({}).collation({locale:"es", strength: 1, caseLevel: true, caseFirst: "lower"}).sort({apellidos: 1})
{ "_id" : ObjectId("5e31affd3fdb41f878dbe2c6"), "apellidos" : "lópez" }
{ "_id" : ObjectId("5e31b0023fdb41f878dbe2c7"), "apellidos" : "lopez" }
{ "_id" : ObjectId("5e31ac2f3fdb41f878dbe2c4"), "apellidos" : "López" }

{ "_id" : ObjectId("5e31ac333fdb41f878dbe2c5"), "apellidos" : "Lopez" }

Opción numericOrdering

```
> db.asociaciones.insert({edad: "1"})
WriteResult({ "nInserted" : 1 })
> db.asociaciones.insert({edad: "10"})
WriteResult({ "nInserted" : 1 })
> db.asociaciones.insert({edad: "22"})
WriteResult({ "nInserted" : 1 })
> db.asociaciones.insert({edad: "2"})
WriteResult({ "nInserted" : 1 })
```

Ordena los datos numéricos de tipo string

```
> db.asociaciones.find({},{edad: 1, _id: 0}).sort({edad: 1})
{ "edad" : "1" }
{ "edad" : "10" }
{ "edad" : "2" }
{ "edad" : "22" }

> db.asociaciones.find({},{edad: 1, _id: 0}).collation({locale: "es", numericOrdering:
true}).sort({edad: 1})
{ "edad" : "1" }
{ "edad" : "2" }
{ "edad" : "10" }
{ "edad" : "22" }
```

Método createCollection()

```
db.createCollection(name, options)

db.createCollection( <name>,
  {
    capped: <boolean>,
    autoIndexId: <boolean>,
    size: <number>,
    max: <number>,
    storageEngine: <document>,
    validator: <document>,
    validationLevel: <string>,
    validationAction: <string>,
    indexOptionDefaults: <document>,
    viewOn: <string>,            // Added in MongoDB 3.4
```

```
    pipeline: <pipeline>,          // Added in MongoDB 3.4
    collation: <document>,         // Added in MongoDB 3.4
    writeConcern: <document>
  }
)
```

Ejemplo con collation

```
> db.createCollection("fooooooo",{collation: {locale:"es", strength: 1}})
{ "ok" : 1 }
> db.fooooooo.insert({nombre: "Piragüistas de Getafe"})
WriteResult({ "nInserted" : 1 })

> db.fooooooo.find({nombre: "Piragüistas de Getafe"})
{ "_id" : ObjectId("5e31bc083fdb41f878dbe2d0"), "nombre" : "Piragüistas de Getafe"
}
> db.fooooooo.find({nombre: "Piraguistas de Getafe"})
{ "_id" : ObjectId("5e31bc083fdb41f878dbe2d0"), "nombre" : "Piragüistas de Getafe"
}

db.getCollectionInfos( {name: "fooooo"} )
```