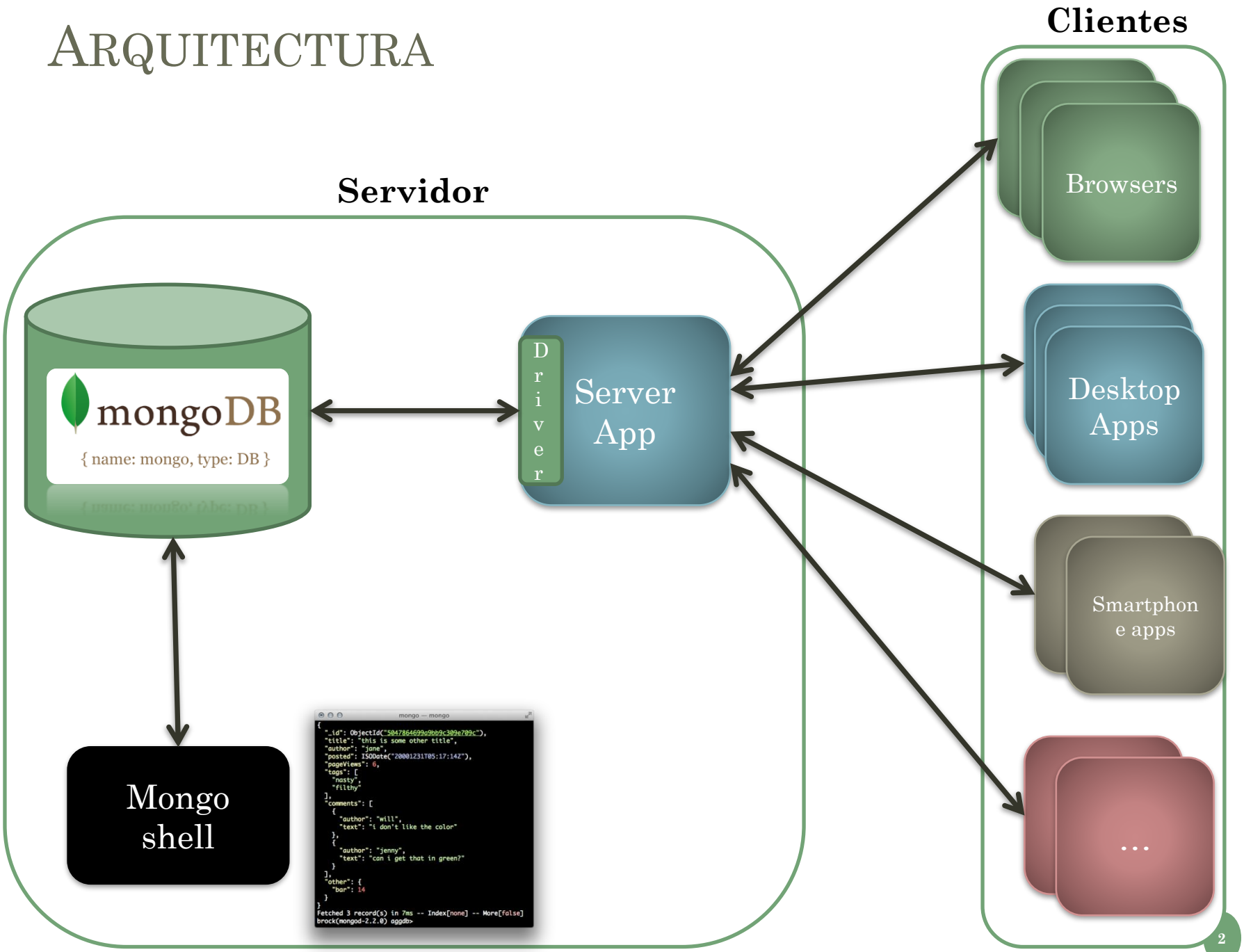
mongoDB Shell

Enrique Barra

Arquitectura



Mongo Shell

- Consola interactiva de Mongo
- Es JavaScript
  - Funciona la flecha arriba para obtener el último comando
  - Funciona el tab
  - Funcionaría:

```
for(var i=0;i<10;i++){
    print("hello mongo" + i);
}
```

- https://docs.mongodb.org/getting-started/shell/client/

- 2 cheat sheets:
  - https://blog.codecentric.de/files/2012/12/MongoDB-CheatSheet-v1_0.pdf
  - http://www.mongodbspain.com/wp-content/uploads/2014/03/MongoDBSpain-CheetSheet.pdf

CRUD

- CRUD: Create, Read, Update, Delete
  - Operaciones básicas en acceso a base de datos
  - Operaciones básicas en acceso a interfaces REST

- Operaciones de lectura:
  - Read
- Operaciones de escritura, modifican la información:
  - Create, Update, Delete

- En Mongo el CRUD es:
  - **C**reate -> Insert
  - **R**ead -> Find
  - **U**pdate -> Update
  - **D**elete -> Remove

Mongo Shell - comandos

- help
- use database
- db
- db.collection
- db.collection.findOne()
- db.collection.find()
  - $gt, $gte, $lt, $lte, $exists, $regex
  - $or, $and, $in, $all
- db.collection.insert()
- db.collection.update()
  - $set, $unset,
  - $push, $pop, $pull, $pullAll, $addToSet
  - Upsert, Multi update
- db.collection.remove()
- .count()
- db.collection.createIndex(), dropIndex(), getIndexes()

Primeros pasos

- Antes de nada usaremos mongoimport para llenar una bbdd
  - Ubuntu:=> mongoimport -d school -c students < students.json
  - Windows:=> .\mongoimport.exe -d school -c students --file 'path_to_students.json'
- Iniciar el shell:
  - > mongo
- Alternativamente podemos iniciarlo conectando a otro host, puerto o con autenticación:
  - > mongo --hostname http://example.com --port 27017 -u foo -p pass --authenticationDatabase arg
- comando para ver una pequeña ayuda:
  - > help
- Mostrar bases de datos que tiene la máquina
  - > show databases
  - > show dbs

Primeros pasos

- Cambiar a una base de datos (por defecto conecta a la BBDD "test")
- (Nota: También así se crea la nueva base de datos)
  - > use school
- Ver que bbdd estamos usando
  - > db
- Ver las colecciones de la bbdd
  - > show collections
  - > show tables
- Borrar una colección
  - > db.students.drop()

Find

Buscar – Read (la R del CRUD)

Find

- Buscar todos los documentos
  - > db.students.find()
- Buscar un documento cualquiera
  - > db.students.findOne()
- Buscar filtrando por campos
  - > db.students.find({name: "Gisela Levin"})
  - > db.students.findOne({name: "Gisela Levin"})
  - > db.students.find({age: 15})
- Imprimir formateado el documento
  - > db.students.find({name: "Gisela Levin"}).**pretty()**

Projection

- Los documentos de MongoDB pueden ser muy grandes
- Es interesante en ocasiones ver sólo algunos campos
- Por defecto indico que si que quiero el campo con un 1
- Mongo añade el _id siempre, si quiero quitarlo pongo un 0
- **Projection**: Devolver sólo algunos campos al buscar

- Ejemplo:
  - > db.students.find({age: 15}, {name: 1})
  - > db.students.find({name: "Gisela Levin"}, {nationality:1, _id: 0})

Cursors

- La operación “find” sobre una colección devuelve un cursor
- Cursor es **un puntero al conjunto de resultados** de una query. Los clientes pueden iterar con los cursores sobre los resultados. Por defecto tienen un timeout que los borra tras 10 minutos de inactividad
- Si el cursor no se guarda en una variable “var” en el shell se imprimen por defecto 20 documentos

- Métodos del cursor:
  - https://docs.mongodb.org/manual/reference/method/js-cursor/
- Métodos útiles: forEach (para recorrer los resultados), count, next, …
- Ejemplo de uso:
  - db.students.find().forEach(**function**(myDoc) { print("user: " + myDoc.name); });

- Hay métodos que se deben aplicar antes de ejecutar el cursor. Si ya se ha hecho la query no tienen efecto. Son skip, limit y sort.

Sort – Limit - Skip

- **Sort** para ordenar los resultados dependiendo de un campo o varios. Ascendente (1) o descendente (-1)
- **Limit** para limitar la cantidad de resultados
- **Skip** para saltarse los primeros n resultados (usado por ejemplo para paginación)

  - > db.students.find().sort({age: 1, name: -1}).limit(5).skip(5)

Query selectors

- Todos en: https://docs.mongodb.org/manual/reference/operator/query/

- Distintos tipos:
  - De comparación
  - Lógicos
  - De elemento
  - De evaluación
  - Geoespaciales
  - Binarios
  - Comentarios

Query selectors de comparación

- **$eq**: encuentra valores que sean iguales a uno dado
- **$ne**: encuentra valores que no sean iguales a uno dado (incluye los documentos que no contengan el campo dado).

- **$gt**: encuentra valores que sean mayores que uno dado
- **$gte**: encuentra valores que sean mayores o iguales que uno dado
- **$lt**: encuentra valores que sean menores que uno dado
- **$lte**: encuentra valores que sean menores o iguales que uno dado

- **$in**: encuentra valores que estén en un array
- **$nin**: encuentra valores que no estén en un array

Ejemplos

- db.students.find({age: {$eq: 24}})
  - Equivalente a: db.students.find({age: 24})
- db.students.find({age: {$ne: 24}})

- db.students.find({age: {$gt: 25}}).count()
- db.students.find({age: {$lte: 25}})

- db.students.find({nationality: {$in: ["spanish", "english"]}})
- db.students.find({nationality: {$nin: ["spanish", "english"]}})
- db.students.find({age: {$nin: [23, 26]}})

Algunas equivalencias con RDBMS

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({“name":“Gisela"})	where name = ‘Gisela'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}})	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}})	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}})	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}})	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}})	where likes != 50

Query selectors lógicos

- **$or**: une dos sentencias y encuentra los documentos que coincidan con cualquiera de las dos
- **$and**: une dos sentencias y encuentra los documentos que coincidan con ambas
- **$not**: invierte el efecto de una query y devuelve los documentos que no encajan con la expresión
- **$nor**: une dos sentencias y encuentra los documentos que no encajan con las dos sentencias a la vez

Ejemplos

- db.students.find({$and: [{age:{$gt: 25}},{nationality: “spanish”}]})
  - Equivalente a (and implícito):
  - db.students.find({age:{$gt: 25}, nationality: “spanish”})

- db.students.find({$or: [{age:{$gt: 25}},{age: {$lt: 15}}]})

- Se puede combinar $and y $or:
- db.inventory.find({
  $and : [
  { $or : [{ price : 0.99 }, { price : 1.99 }] },
  { $or : [{ sale : **true** }, { qty : { $lt : 20 } }] }
  ]
  })

Ejercicio clase

- **where likes>10 AND (by = ‘Enrique' OR title = 'MongoDB tut')**

Ejercicio clase

- **where likes>10 AND (by = ‘Enrique' OR title = 'MongoDB tut')**

```
db.mycol.find( {
  "likes": { $gt: 10 },
  $or: [
    { "by": "Enrique" },
    { "title": "MongoDB tut" }
  ]
}).pretty()
```

Ejercicio clase

- **where likes>10 AND (by = ‘Enrique' OR title = 'MongoDB tut')**

```
db.mycol.find( { $and: [
  {"likes": { $gt: 10 }},
  {$or: [
    { "by": "Enrique" },
    { "title": "MongoDB tut" }
  ]}
]
}).pretty()
```

Query selectors de elemento

- **$exists**: Encuentra los documentos que tengan determinado elemento

- **$type**: encuentra documentos que tengan un campo de un determinado tipo BSON https://docs.mongodb.org/manual/reference/operator/query/type/#op._S_type

- Ejemplos:
- > db.students.find({ age: { $gte: 25 }, email : { $exists: true } });

Query selectors de evaluación

- **$regex**: selecciona documentos cuyo valor encaja con una expresión regular
- **$mod**: hace una operación módulo sobre el valor de un campo y selecciona documento con un resultado determinado
- **$text**: busca texto. Es muy completo y complejo https://docs.mongodb.org/manual/reference/operator/query/text/#op._S_text
- **$where**: selecciona documentos que satisfacen una expresión JavaScript

Ejemplos

- db.students.find({ name: { $regex: /aimee.*/, $options: "i" } })
- db.students.find({ name: { $regex: /aimee.*/i } })
- db.students.find({ name: /aimee.*/i })

- db.students.find({age:{ $mod:[4,0]}})
  - Busca estudiantes cuya edad módulo 4 sea 0

- db.students.find({ $where: “this.age==35”})

```
db.students.find({ $where: function() {
                                    return (this.age == 35) }
                        })
```

Query selectors de array

- **$all**: encuentra arrays que contengan todos los elementos en la query
- **$elemMatch**: selecciona documentos si el elemento en el campo array encaja con **todos** los elementos especificados (elemmatch recibe varias queries y todas tienen que encajar para devolver el documento). Ver:
  - https://docs.mongodb.com/manual/reference/operator/query/elemMatch/
- **$size**: selecciona documentos con el campo array de un tamaño concreto

- Ejemplos:
- db.students.find({nationality: {$all: ["spanish", "english"]}})

- db.scores.find({ results: { $elemMatch: { $gte: 80, $lt: 85 } } })
  - Esta query busca solo documentos cuyo array "results" tenga al menos un elemento que cumpla que es mayor o igual que 80 y menor que 85

- db.students.find({ nationality: { $size: 1 } });

Notación punto

- Para realizar consultas en arrays y en subdocumentos
- Se basa en añadir un punto después del identificador del array o subdocumento para realizar consultas sobre un índice en concreto del array o sobre un campo concreto del subdocumento

- Ejemplos:
- > db.people.find({"tags.1":"enim"})
  - En el ejemplo buscamos todos los documentos que cumplan la condición de que el valor 1 del array sea *"enim"*. **Dos cosas importanes, los arrays empiezan con el índice 0 y es necesario que "tags.1" vaya entre comillas para no recibir un error en la Shell**

- > db.users.find({ "email.work": "peter@happyminds.es"}).pretty()

```
{
    "_id" : ObjectId("513b2c0c1042e447a6c00ed6"),
    "name" : "Peter",
    "email" : {
        "work" : "peter@happyminds.es",
        "personal" : "peterpersonal@happyminds.es"
    }
}
```

Ejercicio de clase

- Contar cuantos estudiantes tienen la primera del array de notas superior a 49

Ejercicio de clase

- Contar cuantos estudiantes tienen la primera del array de notas superior a 49

- db.students.find({"scores.0.score": {$gt: 49}}).count()

Query selectors geoespaciales

- GeoJSON geometría: https://docs.mongodb.org/manual/reference/geojson/#geospatial-indexes-store-geojson
- Indices 2D: https://docs.mongodb.org/manual/core/2d/
- Indices 2dsphere: https://docs.mongodb.org/manual/core/2dsphere/

- **$geoWithin**: Selecciona geometrías JSON.
- **$geoIntersects**: Selecciona geometrías que cortan con una geometría JSON
- **$near**: devuelve objetos geoespaciales próximos a un punto.
- **$nearSphere**: devuelve objetos geoespaciales próximos aun punto en una esfera.

Query selectors binarios

- **$bitsAllSet**: encuentra valores numéricos o binarios en los que todos los bits están a 1
- **$bitsAnySet**: encuentra valores numéricos o binarios en los que cualquier bits está a 1
- **$bitsAllClear**: encuentra valores numéricos o binarios en los que todos los bits están a 0
- **$bitsAnyClear**: encuentra valores numéricos o binarios en los que cualquier bits está a 0

- Ejemplos:
- db.collection.save({ _id: 1, a: 54, binaryValueofA: "00110110" })
- db.collection.find({ a: { $bitsAllSet: [1, 5] } })
  - This query uses the $bitsAllSet operator to test whether field a has bits set at position 1 and position 5, where the least significant bit is position 0.

Query selector comentarios

- **$comment**: Para poner un comentario a la query

- Ejemplo:
- db.records.find({ x: { $mod: [2, 0] }, $comment: "Find even values." })

Insert

Crear (la C del CRUD)

Insert

- > db.students.insert({name: "Enrique Barra", scores: [{type: "exam", score: 1.2}]})

- Si no se indica el _id, Mongo automáticamente lo asigna

- Se pueden insertar un solo documento o varios con una única orden, separados por comas

Insert

```
db.post.insert([
   {
       title: 'MongoDB Overview',
       description: 'MongoDB is no sql database',
       by: 'tutorials point',
       url: 'http://www.tutorialspoint.com',
       tags: ['mongodb', 'database', 'NoSQL'],
       likes: 100
   },

   {
       title: 'NoSQL Database',
       description: 'NoSQL database doesn't have tables',
    by: 'tutorials point',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 20,
    comments: [
      {
       user:'user1',
       message: 'My first comment',
       dateCreated: new Date(2013,11,10,2,35),
       like: 0
     }
   ]
  }
])
```

Update

Actualizar – Update (la U del CRUD)

UPDATE

- Reemplazando el **documento completo** (Cuidado, no añade, reemplaza todo)
  - > db.students.update({name: "Enrique Barra"},{name: "Henry Barra", scores:[{type: "exam", score: 1.2}]})

- Cambiando **un campo** ($set $unset)
  - > db.students.update({name: "Henry Barra"},{$set: {name: "Enri Barra"}})
  - > db.students.update({name: "Enri Barra"},{$unset: {scores: 1}})
- NOTA: insertar un campo es un update! Con $set!

- Operadores update:
  - https://docs.mongodb.org/manual/reference/operator/update/

Operadores Update

- **$inc**: Incrementa el valor del campo en una cantidad
- **$mul**: multiplica el valor del campo por una cantidad
- **$rename**: renombra un campo
- **$set**: fija el valor de un campo en un documento
- **$unset**: borra el valor de un campo de un documento
- **$min**: sólo actualiza el campo si el valor especificado es menor que el que tiene el campo
- **$max**: sólo actualiza el campo si el valor especificado es mayor que el que tiene el campo
- **$currentDate**: pone el valor de un campo a una fecha actual (fecha o timestamp)

Ejemplos

- db.students.update({name: "Henry Barra"},{$set: {name: "Enri Barra"}})
- db.students.update({name: "Enri Barra"},{$unset: {scores: 1}})
- db.products.update({ sku: "abc123" }, { $inc: { qty: -2, "metrics.orders": 1 } })
- db.products.update({ _id: 1 }, { $mul: { price: 1.25 } })
- db.students.update({ _id: 1 }, { $rename: { 'nick': 'alias', 'cell': 'mobile' } })
- db.scores.update({ _id: 1 }, { $min: { lowScore: 150 } })

Otros operadores y opciones Update

- Añadiendo valores a un array ($push)
  - > db.students.update({_id: 0}, {$push: {scores: {type: "final", score: 9.8}}})

- Quitando valores de un array ($pull o $pullAll)
  - > db.students.update({_id: 0}, {$pull: {scores: {type: "exam"}}})

- Opcion "upsert" - **si no existe nada que actualizar se crea uno nuevo**

(upsert = update+insert)

- Es decir **si no hay match** en el find que hace el update (si devuelve 0 documentos) para actualizar que cree uno nuevo con el contenido propuesto (este comando va sin $set)
  - > db.students.update({name: "Red Jhon"}, {name: "Jhon Red"},{upsert: true})

- **Opcion "multi"** - por defecto se actualiza el primer match, con multi a true se actualizan todos (solo válido con $set)
  - > db.students.update({age: {$gte: 18}}, {$set: {adult: true}},{multi: true})

Remove

Borrar - Delete (la D del CRUD)

Remove

- Borrar todos
  - > db.students.remove()
- Borrar según un criterio (similar a buscar)
  - > db.students.remove({name: "Enri Barra"})

- Borrar una colección
  - > db.students.drop()

Indices

Indices

- Es una estructura de datos adicional que se utiliza para optimizar determinadas búsquedas
- Requiere su propio espacio en disco y contiene una copia de los datos de la tabla. Eso significa que un índice es una redundancia. Crear un índice no cambia los datos de la tabla
- El índice se tiene que actualizar cada vez que se inserta, actualiza o borran campos
- El índice tiene un funcionamiento similar al índice de un libro, guardando parejas de elementos: el elemento que se desea indexar y su posición en la base de datos
- No se puede indexar todos los campos

Tipos de Indices

- Campo simple:
  - Indices sobre un campo de tipo simple (por ejemplo un string o number)
  - Podrán ser índices primarios (realizados sobre la clave primaria de la colección) o secundarios (sobre cualquier otro campo que no sea clave primaria).
  - Ejemplo: db.records.createIndex({ userid: 1 })
- Campos compuestos (i.e. varios campos):
  - Indice sobre varios campos a la vez. Sirven para optimizar las queries realizadas sobre ambos campos a la vez.
  - Ejemplo: db.records.createIndex({ name: 1, age: -1 })
- Multiclave:
  - Incide sobre un campo que contiene como valor un array en lugar de un tipo simple.
  - Ejemplo: db.records.createIndex({ scores: 1 })

INDICES – OPCIONES Y SINTAXIS

- db.records.createIndex({ userid: 1 })
- db.accounts.dropIndex({ "tax-id": 1 })
- db.accounts.getIndexes()

- Asegurar que el campo sea único:
  - db.members.createIndex({ "user_id": 1 }, { unique: **true** })
- Que el índice sólo se cree para los documentos que tienen el campo
  - db.addresses.createIndex({ "xmpp_id": 1 }, { sparse: **true** })
- Que el índice expire después de varios segundos
  - db.eventlog.createIndex({ “time": 1 }, { expireAfterSeconds: 3600 })

- Mas info en: https://docs.mongodb.org/manual/core/indexes/

Recordemos el esquema de la relación M a N

ESTUDIANTES

```
{
    id: ObjectId("507f1f77bcf86cd799439011"),
    nombre: "Enrique Perez",
    notas: [ {}, {}],
    slug: 'nombre_corto',
    fecha_matriculacion: ISODate("2015-11-19T06:01:17.171Z"),
    fecha_nacimiento: ISODate("1982-09-19T06:01:17.171Z"),
    asignaturas_matriculadas: [345, 234, 898, 888, 341, 123],
    profesores: [1, 5, 7]
}
```

ASIGNATURAS

```
{
    id: ObjectId("507f1f77bcf86cd799435654"),
    nombre: "Bases de datos",
    slug: "BBDD",
    profesores: [1, 3],
    tipo: "troncal",
    curso: "segundo",
    comentarios: "Asignatura de master"
}
```

PROFESORES

```
{   id: 1,
    nombre: "Joaquin Solis",
    slug: 'joaquinsol',
    fecha_nacimiento: ISODate("1982-09-19T06:01:17.171Z")
}
```

Indices multiclave - multikey indexes

- Útiles en el caso de M a N con arrays de ids o en el caso de querer hacer búsquedas avanzadas
- https://docs.mongodb.org/manual/core/index-multikey/
- Cómo encontrar todos los profesores de un estudiante? -> fácil, los sacamos del array "profesores"
- Cómo encontrar todos los estudiantes de un profesor? -> usando un índice multiclave
- No hace falta especificar nada especial, **si el campo es un array el índice es multiclave**

```
db.students.createIndex({"profesores":  1})
```

```
//estudiantes que tiene el profesor de id 3:
db.students.find({"profesores": 3})
```

```
//estudiantes que tienen el profesor de id 0 y 3:
db.students.find({"profesores": { $all: [0, 3]}})
```

Procesado de varios documentos

Users

```
{
    _id: 12
    name: “Enrique Barra”,
    bio: ”…”,
    location:”Madrid …”,
    email: “ebarra@dit.upm.es”,
    languages: [“Spanish”, “English”],
    …,
    trips: [
       {
           home_id: 56,
           start_date: DATETIME,
           end_date: DATETIME,
           reservation_date: DATETIME,
           …
       },  . . . ],
   likes: [
        {
           home_id: 56,
           start_date: DATETIME,
           end_date: DATETIME,
           reservation_date: DATETIME,
           …
       }, . . . ],
   reviews: [. . .]
}
```

Homes

```
{
    _id: 34
    user_id: 12
    name: “Casa Rural Kike”,
    description: ”…”,
    location:”Madrid …”,
    price: 100,
    rooms: 3,
    …,
    reviews: [
       {
           user_id: 333,
           message: TEXT,
           trip: 2
       },
        . . .
 ]
}
```

- Con el esquema de datos anterior (tipo Airbnb) con usuarios y casas:

- Nos han notificado desde hacienda que tenemos que aumentar el tipo de IVA al 21% (actualmente no cobrábamos IVA). Queremos procesar todas casas para aumentar el precio en 21%.

Aumentar el tipo de IVA

```
db.homes.find({    }).forEach(function(home) {
  var new_price = home.price*1.21;
  db.homes.update({_id: home._id}, {$set: {price: new_price}})
})
```

- //sería mejor haber puesto un campo iva y luego que la app haga la multiplicación

```
db.homes.find({}).forEach(function(home) {
  db.homes.update({_id: home._id}, {$set: {iva: 21}})
})
```

Aumentar el tipo de IVA

- db.homes.update({ }, { $mul: { price: 1.21 }}, {multi: true})
- //en este caso esta opción es mucho mucho más eficiente.
- //pero hay veces que no hay operador en el update y hay que hacerlo con foreach o similar, por ejemplo si nos piden añadir un texto legal en la descripción.
- db.homes.update({ }, {$set: {iva: 21}}, {multi: true})

```
{
    _id: 34
    user_id: 12
    name: "Casa Rural K",
    description: "…",
    location:"Madrid …",
    price: 100,
    rooms: 3,
    …,
    reviews: [
       {
           user_id: 333,
           message: TEXT,
             trip: 2
       },
          . . .
 ]
}
```

- Nos ha notificado por email el usuario premium “Enrique Barra” que ha reformado la casa que alquila y ahora tiene 2 habitaciones y además quiere añadir en su perfil francés como idioma. (si fuese por la app tendríamos el id, pero al ser por email tendremos que hacer dos queries).

CAMBIAR USUARIO Y CASA

var usr = db.users.find({ name: “Enrique Barra” }).toArray()[0]

//comprobamos que usr es solo 1, si devuelve un array deveríamos afinar la búsqueda por ejemplo con:

var usr = db.users.find({ name: “Enrique Barra”, email: “ebarra@dit.upm.es” }).toArray()[]

//toArray pasa del cursor a un array con los documentos que han resultado

db.users.update({_id: usr._id}, {$push: { languages: “French” }})

db.homes.update({ user_id: usr._id }, { $set: { rooms: 2}})

Transacciones (Nuevo en MongoDB 4.0)

Sesiones

- https://docs.mongodb.com/manual/reference/method/Mongo.startSession/#Mongo.startSession

- La funcionalidad de sesiones fue incluida en Mongo 3.6 para gestionar sesiones de los clientes y mejorar la interacción con ellos
- Tiene las opciones: causalConsistency, readConcern, readPreference, retryWrites, writeConcern

- Ejemplo:
- db.getMongo().startSession({ retryWrites: true, causalConsistency: true })

Transacciones

- https://docs.mongodb.com/manual/core/transactions/#transactions-and-the-mongo-shell

- Session.startTransaction()
- Session.commitTransaction()
- Session.abortTransaction()

- Ejemplo:

```
session = db.getMongo().startSession()
session.startTransaction()
    db.test.insert({today : new Date()})
    db.test.insert({some_value : "abc"})
session.commitTransaction()
```

Transacciones - consideraciones

- La atomicidad de documentos (previa a Mongo 4.0) es suficiente para más del 80% de los casos.
  - Que hayan metido transacciones no quiere decir que se utilicen cuando no hace falta
- Las transacciones se cancelan de manera automática tras 60 segundos (configurable)
- No recomiendan modificar más de 1000 documentos por transacción

CAMBIAR USUARIO Y CASA

var usr = db.users.find({ name: “Enrique Barra” }).toArray()[0]

//comprobamos que usr es solo 1, si devuelve un array deveríamos afinar la búsqueda por ejemplo con:

var usr = db.users.find({ name: “Enrique Barra”, email: “ebarra@dit.upm.es” }).toArray()[]

//toArray pasa del cursor a un array con los documentos que han resultado

db.users.update({_id: usr._id}, {$push: { languages: “French” }})

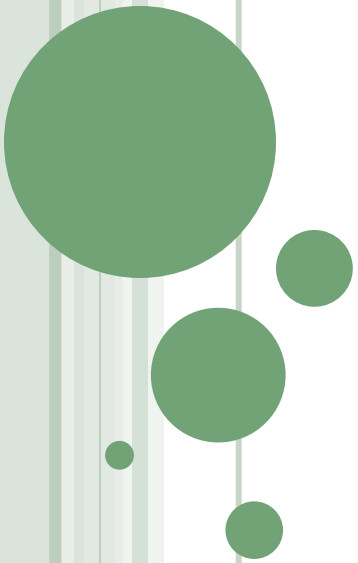
db.homes.update({ user_id: usr._id }, { $set: { rooms: 2}})

- Hacer esto ultimo con una transacción para que ejecute todo o nada

Transacción: Cambiar usuario y casa

```
var usr = db.users.find({ name: “Enrique Barra” }).toArray()[0]
//comprobamos que usr es solo 1, si devuelve un array deveríamos afinar la búsqueda por ejemplo con:
var usr = db.users.find({ name: “Enrique Barra”, email: “ebarra@dit.upm.es” }).toArray()[0]


session = db.getMongo().startSession()
session.startTransaction()
    db.users.update({_id: usr._id}, {$push: { languages: “French” }})
    db.homes.update( { user_id: usr._id }, { $set: { rooms: 2}})
session.commitTransaction()
```



Schema Validation

Dos opciones extra a "validator"

- Opción **validationLevel**, que determina cuán estrictamente MongoDB aplica las reglas de validación a los documentos existentes durante una actualización
  - "off" : no se aplica la validación
  - "estricto": es el valor por defecto. La validación se aplica a todas las inserciones y actualizaciones
  - "moderado": la validación se aplica a todos los documentos válidos existentes. Los documentos no válidos son ignorados
- La opción **validationAction**, que determina si MongoDB debe lanzar error y rechazar los documentos que violan las reglas de validación o advertir sobre las violaciones en el log pero permitir los documentos inválidos
  - "error": es el valor por defecto. El documento debe pasar la validación para ser escrito
  - "warn": se escribe un documento que no pasa la validación pero se registra un mensaje de advertencia

- Al crear la colección:

```
db.createCollection("students", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "address" ],
         properties: {
            name: {
               bsonType: "string",
               description: "must be a string and is required"
            },
```

- Sobre una colección existente:

```
db.runCommand( { collMod: "contacts",
   validator: { $jsonSchema: {
      bsonType: "object",
      required: [ "phone" ],
      properties: {
         phone: {
            bsonType: "string",
            description: "must be a string and is required"
         },
```

AutoEvaluación

Y como hacer ejemplos

MONGO SHELL - EJEMPLOS

- En primer lugar se puede utilizar la colección mystudents.json e importarla con mongoimport como vimos y ejecutar cualquier query de las vistas

- Otra opción:
- Llenado de la bbdd, la podemos llenar con:

```
for(var i=0;i<1000;i++){
    var names = ["examen", "ensayo", "quiz"];
    for(var j=0;j<3;j++){
        db.scores.insert({"student": i, "type": names[j], "score":Math.round(Math.random()*100)});
    }
}
```

- Y podremos hacer queries sobre ella:
  - db.scores.find({“student”: 19, type:”ensayo”},{“score”:true, “_id”:false})
  - db.scores.find({type: “ensayo”, score: {$gt: 95, $lte:98}},{“student”:true, “_id”:false})
  - db.scores.find({$or:[{"score":{$lt:50}},{"score":{$gt:90}}]})

Test

- Which of the following documents matches this query?
  - db.users.find({ friends : { $all : ["Joe" , "Bob"] }, favorites : { $in : ["running" , "pickles"] } })

A. { name : "William" , friends : ["Bob" , "Fred"] , favorites : ["hamburgers", "running"] }
B. { name : "Stephen" , friends : ["Joe" , "Pete"] , favorites : ["pickles", "swimming"] }
C. { name : "Cliff" , friends : ["Pete" , "Joe" , "Tom" , "Bob"] , favorites : ["pickles", "cycling"] }
D. { name : "Harry" , friends : ["Joe" , "Bob"] , favorites : ["hot dogs", "swimming"] }

Test

- Which of the following documents matches this query?
  - db.users.find({ friends : { $all : ["Joe" , "Bob"] }, favorites : { $in : ["running" , "pickles"] } })

A. { name : "William" , friends : ["Bob" , "Fred"] , favorites : ["hamburgers", "running"] }
B. { name : "Stephen" , friends : ["Joe" , "Pete"] , favorites : ["pickles", "swimming"] }
C. { name : "Cliff" , friends : ["Pete" , "Joe" , "Tom" , "Bob"] , favorites : ["pickles", "cycling"] }
D. { name : "Harry" , friends : ["Joe" , "Bob"] , favorites : ["hot dogs", "swimming"] }

- **Which of the following MongoDB query is equivalent to the following SQL query:**

  UPDATE users SET status = "C" WHERE age > 25

- **A** db.users.update({ age: { $gt: 25 } }, { status: "C" })
- **B** db.users.update({ age: { $gt: 25 } }, { $set: { status: "C" } })
- **C** db.users.update({ age: { $gt: 25 } }, { $set: { status: "C" } }, { multi: true })
- **D** db.users.update({ age: { $gt: 25 } }, { status: "C" }, { multi: true })

- **Q 1 - Which of the following MongoDB query is equivalent to the following SQL query:**

  UPDATE users SET status = "C" WHERE age > 25

- **A** db.users.update({ age: { $gt: 25 } }, { status: "C" })
- **B** db.users.update({ age: { $gt: 25 } }, { $set: { status: "C" } })
- C db.users.update({ age: { $gt: 25 } }, { $set: { status: "C" } }, { multi: true })
- **D** db.users.update({ age: { $gt: 25 } }, { status: "C" }, { multi: true })

Notación punto

- Suppose a simple e-commerce product catalog called *catalog* with documents that look like this:

```
{ product : "Super Duper-o-phonic",
  price: 10000,
  reviews: [
    {
      user: "fred",
      comment: "Great!",
      rating: 5
    },
    {
      user: "tom",
      comment: "I agree with Fred, somewhat!",
      rating: 4
    }
  ]
}
```

  - Write a query that finds all products that cost more than 10,000 and that have a rating of 5 or better.

Notación punto

- Suppose a simple e-commerce product catalog called *catalog* with documents that look like this:

```
{ product : "Super Duper-o-phonic",
  price: 10000,
  reviews: [
    {
      user: "fred",
      comment: "Great!",
      rating: 5
    },
    {
      user: "tom",
      comment: "I agree with Fred, somewhat!",
      rating: 4
    }
  ]
}
```

  - Write a query that finds all products that cost more than 10,000 and that have a rating of 5 or better.

- db.catalog.find({ "price" : { "$gt" : 10000 } , "reviews.rating" : { "$gte" : 5 } });

- What does the following command return:

```
db.posts.find( { 'tags.0': "tutorial" } )
```

- **A** - All the posts whose tags array contains tutorial
- **B** - All the posts which contains only one tag element in the tag array
- **C** - All the posts having the first element of the tags array as tutorial
- **D** - All the posts which contains 0 or more tags named tutorial

- What does the following command return:

```
db.posts.find( { 'tags.0': "tutorial" } )
```

- **A** - All the posts whose tags array contains tutorial
- **B** - All the posts which contains only one tag element in the tag array
- **C** - All the posts having the first element of the tags array as tutorial
- **D** - All the posts which contains 0 or more tags named tutorial

- **Which of the following commands create an unique index on author field of the posts collection?**

- **A** - db.posts.createIndex({"author":1 }, {"unique": true});
- **B** - db.posts.createIndex({"author": unique });
- **C** - db.posts.createIndex({"author": {"$unique":1} });
- **D** - db.posts.createIndexUnique({"author":1 });

- **Which of the following commands create an unique index on author field of the posts collection?**

- **A** - db.posts.createIndex({"author":1 }, {"unique": true});
- **B** - db.posts.createIndex({"author": unique });
- **C** - db.posts.createIndex({"author": {"$unique":1} });
- **D** - db.posts.createIndexUnique({"author":1 });

- **Consider that our posts collection contains an array field called tags that contains tags that the user enters.**

{ _id: 1, tags: ["tutorial", "fun", "learning"], post_text: "This is my first post", //other elements of document }

- **Which of the following commands will find all the posts that have been tagged as tutorial?**

- **A** - db.posts.find({ tags : "tutorial" });
- **B** - db.posts.find({ tags : ["tutorial"] });
- **C** - db.posts.find({ $array : {tags: "tutorial"} });
- **D** - db.posts.findInArray({ tags : "tutorial" });

- **Q 3 - Consider that our posts collection contains an array field called tags that contains tags that the user enters.**

{ _id: 1, tags: ["tutorial", "fun", "learning"], post_text: "This is my first post", //other elements of document }

- **Which of the following commands will find all the posts that have been tagged as tutorial?**

- **A** - db.posts.find({ tags : "tutorial" });
- **B** - db.posts.find({ tags : ["tutorial"] });
- **C** - db.posts.find({ $array : {tags: "tutorial"} });
- **D** - db.posts.findInArray({ tags : "tutorial" });

- Explanation: Searching an array is no different than searching a normal field. Hence the first option.

- **Q 1 - In a collection that contains 100 post documents, what does the following command do?**

  db.posts.find().skip(5).limit(5)

- **A** - Skip and limit nullify each other. Hence returning the first five documents.
- **B** - Skips the first five documents and returns the sixth document five times
- **C** - Skips the first five documents and returns the next five
- **D** - Limits the first five documents and then return them in reverse order

- **Q 1 - In a collection that contains 100 post documents, what does the following command do?**

  db.posts.find().skip(5).limit(5)

- **A** - Skip and limit nullify each other. Hence returning the first five documents.
- **B** - Skips the first five documents and returns the sixth document five times
- **C** - Skips the first five documents and returns the next five
- **D** - Limits the first five documents and then return them in reverse order

- **Q 3 - What does the following MongoDB command return?**

  db.posts.find({ likes : { $gt : 100 }, likes : { $lt : 200 } });

- **A** - Posts with likes greater than 100 but less than 200
- **B** - Posts with likes greater than or equal to 100 but less than or equal to 200
- **C** - Posts with likes less than 200
- **D** - Will return syntax error

- **Q 3 - What does the following MongoDB command return?**

  db.posts.find({ likes : { $gt : 100 }, likes : { $lt : 200 } });

- **A** - Posts with likes greater than 100 but less than 200
- **B** - Posts with likes greater than or equal to 100 but less than or equal to 200
- **C** - Posts with likes less than 200
- **D** - Will return syntax error

- Explicación:
- When the mongo shell interprets this query, it will override the first condition $gt and consider only the $lt one. To apply both the less than and greater than condition, you will have to use the $and operator.

SQL TO MONGODB

Crear tabla/colección

Crea una tabla users que tiene

id,

user_id (string de 30),

age (number),

status (char de 1)

Crear tabla/colección

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    user_id VARCHAR(30),
    age TINYINT,
    status CHAR(1),
    PRIMARY KEY (id)
)
```

Crear tabla/colección

```
db.createCollection("users")
```

- Realmente no hace falta crearla de modo explícito, al meter el primer documento se crea

```
db.users.insert( {
  user_id: "abc123",
  age: 55,
  status: "A"
} )
```

Editar tabla/colección

Añadir a la tabla users un campo join_date que es una fecha

Quitar de la tabla users un campo join_date que es una fecha

Editar tabla/colección

```
ALTER TABLE users ADD join_date DATETIME
```

```
ALTER TABLE users DROP COLUMN join_date
```

Editar tabla/colección

```
ALTER TABLE users ADD join_date DATETIME
```

```
db.users.update(
    { },
    { $set: { join_date: new Date() } },
    { multi: true }
)
```

```
ALTER TABLE users DROP COLUMN join_date
```

```
db.users.update(
    { },
    { $unset: { join_date: "" } },
    { multi: true }
)
```

Crear índice

Crear un índice en la tabla users en el campo user_id

Crear un índice multicolumna en la tabla users en el campo user_id ascendente y age descendente

Nota: Un índice multi-columna sirve para acelerar las búsquedas basadas en las mismas columnas en el mismo orden.

Crear índice

```
CREATE INDEX idx_user_id_asc ON users(user_id)
```

```
CREATE INDEX   idx_user_id_asc_age_desc ON users(user_id, age DESC)
```

Crear índice

```
CREATE INDEX idx_user_id_asc ON users(user_id)
```

```
db.users.createIndex( { user_id: 1 } )
```

```
CREATE INDEX   idx_user_id_asc_age_desc ON users(user_id, age DESC)
```

```
db.users.createIndex( { user_id: 1, age: -1 } )
```

Insertar

Insertar en la tabla usuarios los valores user_id: "bcd001", age: 45 y status: "A"

Insertar

```
INSERT INTO users(user_id, age, status) VALUES ("bcd001",45,"A")
```

Insertar

```
INSERT INTO users(user_id,  age, status) VALUES ("bcd001",45,"A")
```

```
db.users.insert(
   { user_id: "bcd001", age: 45, status: "A" }
)
```

SELECT

Seleccionar todas las entradas de la tabla

Seleccionar el id, user_id y status de la tabla users

Buscar los usuarios con status “A”

Mostrar user_id y status de los usuarios donde el status es “A”

Buscar los usuarios cuyo status no es “A”

Select

```
SELECT * FROM users
```

```
SELECT id, user_id, status FROM users
```

```
SELECT * FROM users WHERE status = "A"
```

```
SELECT user_id, status FROM users WHERE status = "A"
```

```
SELECT * FROM users WHERE status != "A"
```

SELECT

```
SELECT * FROM users
db.users.find()

SELECT id, user_id, status FROM users
db.users.find(
    { },
    { user_id: 1, status: 1 }
)

SELECT * FROM users WHERE status = "A"
db.users.find(
    { status: "A" }
)

SELECT user_id, status FROM users WHERE status = "A"
db.users.find(
    { status: "A" },
    { user_id: 1, status: 1, _id: 0 }
)

SELECT * FROM users WHERE status != "A“
db.users.find(
    { status: { $ne: "A" } }
)
```

Select II

Buscar los usuarios cuyo status es “A” y la edad 50

Buscar los usuarios cuyo status es “A” o la edad 50

Buscar los usuarios cuyo edad es mayor de 25

Buscar los usuarios cuyo de edad entre 25 y 50

SELECT II

```
SELECT * FROM users WHERE status = "A" AND age = 50
```

```
SELECT * FROM users WHERE status = "A" OR age = 50
```

```
SELECT * FROM users WHERE age > 25
```

```
SELECT * FROM users WHERE age > 25 AND   age <= 50
```

SELECT II

```
SELECT * FROM users WHERE status = "A" AND age = 50
db.users.find(
    { status: "A",
      age: 50 }
)

SELECT * FROM users WHERE status = "A" OR age = 50
db.users.find(
    { $or: [ { status: "A" } ,
             { age: 50 } ] }
)

SELECT * FROM users WHERE age > 25
db.users.find(
    { age: { $gt: 25 } }
)

SELECT * FROM users WHERE age > 25 AND   age <= 50
db.users.find(
   { age: { $gt: 25, $lte: 50 } }
)
```

SELECT III

Seleccionar usuarios con el user_id que contenga “bc”

Seleccionar usuarios con el user_id que empiece por “bc”

Buscar usuarios con status “A” y ordenar la salida por user_id ascendente

Buscar usuarios con status “A” y ordenar la salida por user_id descendente

SELECT III

```
SELECT * FROM users WHERE user_id like "%bc%"
```

```
SELECT * FROM users WHERE user_id like "bc%"
```

```
SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC
```

```
SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC
```

SELECT III

```
SELECT * FROM users WHERE user_id like "%bc%"
db.users.find( { user_id: /bc/ } )
-o-
db.users.find( { user_id: { $regex: /bc/ } } )


SELECT * FROM users WHERE user_id like "bc%"
db.users.find( { user_id: /^bc/ } )
-o-
db.users.find( { user_id: { $regex: /^bc/ } } )


SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC
db.users.find( { status: "A" } ).sort( { user_id: 1 } )


SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC
db.users.find( { status: "A" } ).sort( { user_id: -1 } )
```

Select IV

Contar los usuarios

Contar los usuarios que tienen campo user_id

Contar los usuarios con más de 30 años

Contar los distintos estados que hay

Mostar un usuario

Mostrar 5 usuarios saltándome los 10 primeros

Select IV

```
SELECT COUNT(*) FROM users
```

```
SELECT COUNT(user_id) FROM users
```

```
SELECT COUNT(*) FROM users WHERE age > 30
```

```
SELECT DISTINCT(status) FROM users
```

```
SELECT * FROM users LIMIT 1
```

```
SELECT * FROM users LIMIT 5 SKIP 10
```

SELECT IV

```
SELECT COUNT(*) FROM users
db.users.count()              -O-       db.users.find().count()


SELECT COUNT(user_id) FROM users
db.users.count( { user_id: { $exists: true }})
db.users.find( { user_id: { $exists: true }}).count()


SELECT COUNT(*) FROM users WHERE age > 30
db.users.count( { age: { $gt: 30 }}) -o- db.users.find( { age: { $gt: 30 }}).count()


SELECT DISTINCT(status) FROM users
db.users.distinct("status")


SELECT * FROM users LIMIT 1
db.users.findOne()        -o-       db.users.find().limit(1)


SELECT * FROM users LIMIT 5 SKIP 10
db.users.find().limit(5).skip(10)
```

UPDATE

Actualizar los usuarios con edad mayor de 25, ponerles status “C”

Actualizar los usuarios con status “A”, incrementarles la edad en 3

UPDATE

```
UPDATE users SET status = "C" WHERE age > 25
```

```
UPDATE users SET age = age + 3 WHERE status = "A"
```

Update

```
UPDATE users SET status = "C" WHERE age > 25
```

```
db.users.update(
   { age: { $gt: 25 } },
   { $set: { status: "C" } },
   { multi: true }
)
```

```
UPDATE users SET age = age + 3 WHERE status = "A“
```

```
db.users.update(
   { status: "A" } ,
   { $inc: { age: 3 } },
   { multi: true }
)
```

Delete

Borrar los usuarios con status “D”

Borrar todos los usuarios

Delete

```
DELETE FROM users WHERE status = "D"
```

```
DELETE FROM users
```

Delete

```
DELETE FROM users WHERE status = "D"
```

```
db.users.remove( { status: "D" } )
```

```
DELETE FROM users
```

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

Query translator

- http://www.querymongo.com/