MongoDB

Universidad Autónoma de Coahuila

Ing. Urbano de J. Flores Zaragoza

¿Que es mongoDB?

- Humongous - enorme
- Base de datos no relacional
- Orientada a documentos
- No se requiere seguir un esquema
  - Los documentos de una misma colección pueden tener un esquema diferente
- Los documentos son objetos en formato JSON

Historia

- El desarrollo inició alrededor de 2007 por una empresa llamada 10gen que después cambió su nombre a MongoDB Inc
- El desarrollo inicial se enfocó en Platform As A Service, pero en 2009 sale como open source
- En 2010 sale la versión 1.4, primera versión productiva
- En 2014 sale la versión 2.4.9 la versión mas estable
- Actualmente la version 4.4.6 es la versión estable (Community)

Historia …

- Su uso se ha incrementado grandemente debido a su facilidad de uso y potencia
- Se usa ampliamente en análisis de información en tiempo real en servicios financieros, gubernamentales, negocios y mucho más
- Compañías como Google, facebook, Twitter, Bosch, Nokia son algunos usuarios de MongoDB

¿En dónde puedo usar mongoDB?

- En cualquier tipo de aplicación
- Es especialmente útil en entornos que requieran escalabilidad

Documento

- Un documento es similar a un registro en una BD tipo SQL
- Se define en formato JSON (Javascript Object Notation)

```
{
    nombre: “Luis”,
    apellidoPaterno: “Torres”,
    apellidoMaterno: “García”,
    edad: 19,
    materias: [ “M001”, “M002”, “M003”]
}
```

JSON

- Fácil de escribir y leer para los humanos
- Fácil de interpretar y generar para las computadoras
- Los objetos pueden ser anidados
- Se construye en:
  - Pares de nombre/valor
  - Lista de valores

BSON

- Binary JSON
- Serialización binaria de documentos JSON
- Permite referencias
- La estructura embebida de objetos reduce la necesidad de JOINS
- Metas:
  - Ligero
  - Eficiente (codificado/decodificado)
  - *Traversable*, (Traversing significa visitar o iterar los elementos de una estructura de datos), se logra agregando metadatos al JSON

MongoDB

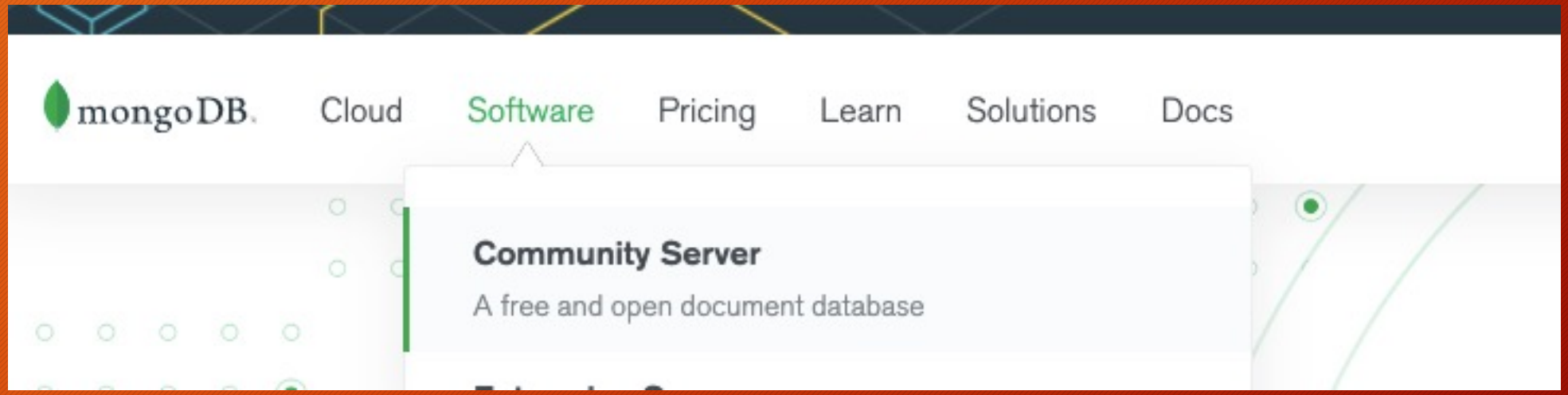
- Orientada a documentos
  - NoSQL, no se tiene el concepto de relacional
  - Flexible y adaptable
- Consultas eficientes
  - Soporta búsquedas por campos, rangos, expresiones regulares
- Indexado
  - Para incrementar el perfomance de las búsquedas
- Replicación
  - Pueden establecerse esquemas de alta disponibilidad
- Balanceo
  - Puede ejecutarse en varios servidores balancenado la carga y la duplicación de datos para seguir funcionando en caso de falla

Colecciones

- Son conjuntos de documentos
- Similar a las tablas en SQL

Instalación

- https://www.mongodb.com/



Comandos básicos

- Abrir la consola
  - show dbs -- muestra las bases de datos
  - use nombrebd -- se cambia a la bd, si no existe se va a crear cuando se cree la primera colección de documentos
  - db -- muestra en que base de datos estamos
  - db.createCollection(“Persona”) -- crea la colección, se crea la BD
  - db.dropDatabase()
  - show collections -- muestra las colecciones de la BD

Inserción

- MongoDB almacena documentos en colecciones (similar a las tablas de SQL)
- Si se agrega un documento a una colección no existente, mongoDB crea en forma automática la colección
- db.collecion.insertOne() -- agrega un documento a una colección

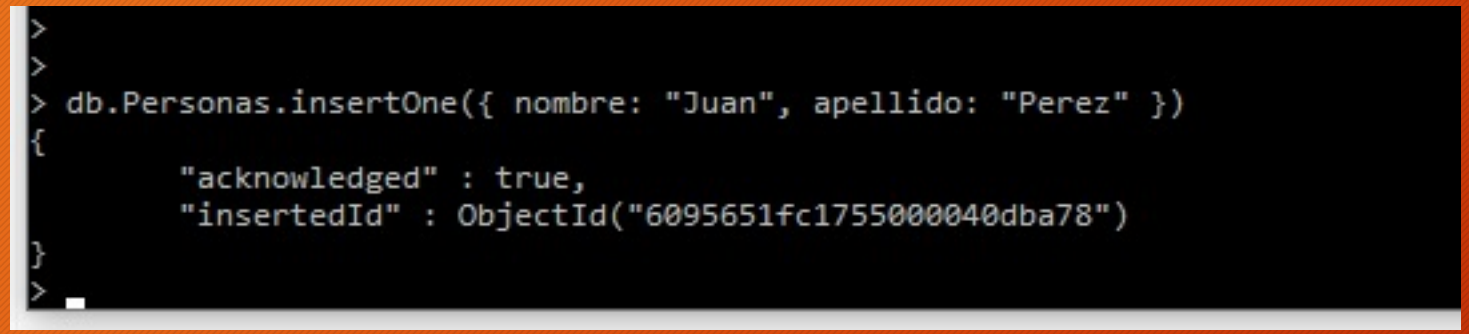
```
db.nombreColeccion.insertOne(
    {
        nombre: “juan”,
        edad: 30,
        estatus: “pendiente”
    }
)
```

```
>
>
> db.Personas.insertOne({ nombre: "Juan", apellido: "Perez" })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6095651fc1755000040dba78")
}
>
```

Creación explícita de una colección

- db.createCollection("Personas")
- Podemos limitar el numero de documentos
  - db.createCollection("Personas", {capped: true, size: 100000, max: 5})
  - size: tamaño en bytes, max: número máximo de documentos
- La colecciones capped son de tamaño fijo y soportan operaciones masivas
- Una vez que se alcanza el límite de documentos, las inserciones subsequentes eliminant los documentos mas antiguos
- FIFO

Inserción



El ObjectId es un identificador único del documento, se genera uno por cada “registro” de la colección

Listado de documentos de una colección

- db.nombreColeccion.find() -- muestra los documentos de la colección

```
> db.Personas.find()
{ "_id" : ObjectId("609564f9c1755000040dba77"), "nombre" : "Juan", "apellido" : "Perez" }
```

Inserción múltiple

- db.nombreColeccion.insertMany([
  { … Documento JSON},
  { … Documento JSON }
  ,
  …

]

```
>
>
> db.Personas.insertMany([{ nombre: "Carlos", apellido: "Hdz" }, { nombre: "Luis", apellido: "Rodriguez"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("6095699fc1755000040dba7b"),
                ObjectId("6095699fc1755000040dba7c")
        ]
}
>
```

Inserción múltiple (obsoleto)

- db.nombreColeccion.insert([
  { … Documento JSON},
  { … Documento JSON }
  ,
  …

]

```
> db.Personas.insert([{ nombre: "Arturo", apellido: "Hdz" }, { nombre: "Maria", apellido: "Rodriguez"}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

Visualización de documentos

- db.nombreColeccion.find()
- db.nombreColeccion.find().pretty()

Ejercicio

- Leer el documento proporcionado por el instructor
- Identificar los documentos presentes en el modelo (puede utilizarse el enfoque relacional)
- Elegir algun documento del modelo y realizar varias inserciones utilizando los comandos vistos anteriormente

Búsquedas

- db.nombreColeccion.find({ criterio de busqueda })

```
> db.Personas.find({ nombre: "Maria" })
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
>
```

- db.nombreColeccion.find({ criterio de busqueda }).pretty()

--Para que sea mas legible el resultado

```
>
> db.Personas.find({ nombre: "Maria" }).pretty()
{
        "_id" : ObjectId("60956bbac1755000040dba7e"),
        "nombre" : "Maria",
        "apellido" : "Rodriguez"
}
>
```

Operadores de comparación ($eq)

- db.nombreColeccion.find({ propiedad : { $eq : valor } })

```
>
> db.Personas.find({ nombre: {$eq : "Maria"} }).pretty()
{
        "_id" : ObjectId("60956bbac1755000040dba7e"),
        "nombre" : "Maria",
        "apellido" : "Rodriguez"
}
>
```

Operadores de comparación ($gt)

- db.nombreColeccion.find({ propiedad : { $gt : valor } })

```
> db.Personas.find({ edad: {$gt : 20} }).pretty()
{
        "_id" : ObjectId("609575acc1755000040dba7f"),
        "nombre" : "Carlos",
        "apellido" : "Hdz",
        "edad" : 30
}
{
        "_id" : ObjectId("609575acc1755000040dba80"),
        "nombre" : "Luis",
        "apellido" : "Rodriguez",
        "edad" : 35
}
>
```

Operadores de comparación ($gte)

- db.nombreColeccion.find({ propiedad : { $gte : valor } })

```
> db.Personas.find({ edad: {$gte : 30} }).pretty()
{
        "_id" : ObjectId("609575acc1755000040dba7f"),
        "nombre" : "Carlos",
        "apellido" : "Hdz",
        "edad" : 30
}
{
        "_id" : ObjectId("609575acc1755000040dba80"),
        "nombre" : "Luis",
        "apellido" : "Rodriguez",
        "edad" : 35
}
```

Operadores de comparación

- $lt -- menor que
- $lte -- menor o igual
- $ne -- diferente

Operadores de comparación ($in)

- Similar al operador in de SQL

```
> db.Personas.find({ edad: {$in : [30, 35]} }).pretty()
{
        "_id" : ObjectId("609575acc1755000040dba7f"),
        "nombre" : "Carlos",
        "apellido" : "Hdz",
        "edad" : 30
}
{
        "_id" : ObjectId("609575acc1755000040dba80"),
        "nombre" : "Luis",
        "apellido" : "Rodriguez",
        "edad" : 35
}
>
```

Operadores de comparación ($in)

- Aplica también en propiedades de tipo arreglo

```
> db.Personas.find({ grados: {$in : ["licenciatura"]} }).pretty()
{
        "_id" : ObjectId("60958553c1755000040dba81"),
        "nombre" : "Carlos",
        "apellido" : "Hdz",
        "edad" : 30,
        "grados" : [
                "licenciatura",
                "maestria"
        ]
}
{
        "_id" : ObjectId("60958553c1755000040dba82"),
        "nombre" : "Luis",
        "apellido" : "Rodriguez",
        "edad" : 35,
        "grados" : [
                "licenciatura"
        ]
}
>
```

Operadores de comparación ($nin)

- Similar al operador not in de SQL

```
> db.Personas.find({ edad: {$nin : [30, 35]} }).pretty()
{
        "_id" : ObjectId("609564f9c1755000040dba77"),
        "nombre" : "Juan",
        "apellido" : "Perez"
}
{
        "_id" : ObjectId("6095651fc1755000040dba78"),
        "nombre" : "Juan",
        "apellido" : "Perez"
}
{
        "_id" : ObjectId("6095698fc1755000040dba79"),
        "nombre" : "Carlos",
        "apellido" : "Hdz"
}
```

Operadores lógicos ($or, $and, $not)

- Permite combinar varios criterios de busqueda

```
> db.Personas.find({ $or: [{ edad: {$gt : 20, $lt: 35}}, {nombre: "Juan" }] })
{ "_id" : ObjectId("609564f9c1755000040dba77"), "nombre" : "Juan", "apellido" : "Perez" }
{ "_id" : ObjectId("6095651fc1755000040dba78"), "nombre" : "Juan", "apellido" : "Perez" }
{ "_id" : ObjectId("609575acc1755000040dba7f"), "nombre" : "Carlos", "apellido" : "Hdz", "edad" : 30 }
{ "_id" : ObjectId("60958553c1755000040dba81"), "nombre" : "Carlos", "apellido" : "Hdz", "edad" : 30, "grados'
nciatura", "maestria" ] }
```

- Existen también los operadores $and y $not

Expresiones regulares

- /expresion/ -- documentos que contiene la expresion en la propiedad, similar al like de SQL

```
> db.Personas.find({nombre: /uro/})
{ "_id" : ObjectId("60956bbac1755000040dba7d"), "nombre" : "Arturo", "apellido" : "Hdz" }
>
```

- /^expresion/ -- inician con la expresión
- /expresion$/ -- finalizan con la expresión
- Alternativa

```
> db.Personas.find({nombre: {$regex:  "o$"}})
{ "_id" : ObjectId("60956bbac1755000040dba7d"), "nombre" : "Arturo", "apellido" : "Hdz" }
>
```

Ejercicio

- Realizar diversas consultas con el modelo de Salón de Fiestas

Ordenamiento

- db.nombreColeccion.find().sort({propiedad : valor})

  valor: 1 ascendente, -1 descendente

```
> db.Personas.find().sort({nombre: 1})
{ "_id" : ObjectId("60956bbac1755000040dba7d"), "nombre" : "Arturo", "apellido" : "Hdz" }
{ "_id" : ObjectId("6095698fc1755000040dba79"), "nombre" : "Carlos", "apellido" : "Hdz" }
{ "_id" : ObjectId("6095699fc1755000040dba7b"), "nombre" : "Carlos", "apellido" : "Hdz" }
{ "_id" : ObjectId("609575acc1755000040dba7f"), "nombre" : "Carlos", "apellido" : "Hdz", "edad"
{ "_id" : ObjectId("60958553c1755000040dba81"), "nombre" : "Carlos", "apellido" : "Hdz", "edad"
nciatura", "maestria" ] }
{ "_id" : ObjectId("609564f9c1755000040dba77"), "nombre" : "Juan", "apellido" : "Perez" }
{ "_id" : ObjectId("6095651fc1755000040dba78"), "nombre" : "Juan", "apellido" : "Perez" }
{ "_id" : ObjectId("6095698fc1755000040dba7a"), "nombre" : "Luis", "apellido" : "Rodriguez" }
{ "_id" : ObjectId("6095699fc1755000040dba7c"), "nombre" : "Luis", "apellido" : "Rodriguez" }
{ "_id" : ObjectId("609575acc1755000040dba80"), "nombre" : "Luis", "apellido" : "Rodriguez", "e
{ "_id" : ObjectId("60958553c1755000040dba82"), "nombre" : "Luis", "apellido" : "Rodriguez", "e
licenciatura" ] }
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
```

Proyecciones

- db.nombreColeccion.find({}, {propiedad: valor, propiedad: valor, …}) --valor: 1 o 0

```
> db.Personas.find({},{nombre:1})
{ "_id" : ObjectId("609564f9c1755000040dba77"), "nombre" : "Juan" }
{ "_id" : ObjectId("6095651fc1755000040dba78"), "nombre" : "Juan" }
{ "_id" : ObjectId("6095698fc1755000040dba79"), "nombre" : "Carlos" }
{ "_id" : ObjectId("6095698fc1755000040dba7a"), "nombre" : "Luis" }
{ "_id" : ObjectId("6095699fc1755000040dba7b"), "nombre" : "Carlos" }
{ "_id" : ObjectId("6095699fc1755000040dba7c"), "nombre" : "Luis" }
```

- Ocultar el objectId

```
> db.Personas.find({},{nombre:1, _id: 0})
{ "nombre" : "Juan" }
{ "nombre" : "Juan" }
{ "nombre" : "Carlos" }
{ "nombre" : "Luis" }
{ "nombre" : "Carlos" }
```

Proyecciones (map)

- db.nombreColeccion.find().map(u =>u.propiedad)

```
> db.Personas.find().map(u=>u.nombre)
[
        "Juan",
        "Juan",
        "Carlos",
        "Luis",
```

```
> db.Personas.find().map(u=> "Mi nombre es " + u.nombre)
[
        "Mi nombre es Juan",
        "Mi nombre es Juan",
        "Mi nombre es Carlos",
```

Proyecciones (foreach)

- db.nombreColeccion.find().forEach(u =>print(u.propiedad))

```
> db.Personas.find().forEach(u=> print("Mi nombre es " + u.nombre))
Mi nombre es Juan
Mi nombre es Juan
Mi nombre es Carlos
Mi nombre es Luis
```

Limitar los resultados

- db.nombreColeccion.find().limit(n) -- n es el numero de documentos

```
> db.Personas.find().limit(3)
{ "_id" : ObjectId("609564f9c1755000040dba77"), "nombre" : "Juan", "apellido" : "Perez" }
{ "_id" : ObjectId("6095651fc1755000040dba78"), "nombre" : "Juan", "apellido" : "Perez" }
{ "_id" : ObjectId("6095698fc1755000040dba79"), "nombre" : "Carlos", "apellido" : "Hdz" }
>
```

- db.nombreCollecion.find().skip(n) -- ignora los primeros n documentos

```
> db.Personas.find().skip(3)
{ "_id" : ObjectId("6095698fc1755000040dba7a"), "nombre" : "Luis", "apellido" : "Rodriguez" }
{ "_id" : ObjectId("6095699fc1755000040dba7b"), "nombre" : "Carlos", "apellido" : "Hdz" }
{ "_id" : ObjectId("6095699fc1755000040dba7c"), "nombre" : "Luis", "apellido" : "Rodriguez" }
{ "_id" : ObjectId("60956bbac1755000040dba7d"), "nombre" : "Arturo", "apellido" : "Hdz" }
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
```

Convertir a Array

- db.nombreColeccion.find().toArray()

```
> db.Personas.find().toArray()
[
        {
                "_id" : ObjectId("609564f9c1755000040dba77"),
                "nombre" : "Juan",
                "apellido" : "Perez"
        },
        {
                "_id" : ObjectId("6095651fc1755000040dba78"),
                "nombre" : "Juan",
                "apellido" : "Perez"
        },
```

```
> db.Personas.find().toArray()[2]
{
        "_id" : ObjectId("6095698fc1755000040dba79"),
        "nombre" : "Carlos",
        "apellido" : "Hdz"
}
>
```

Convertir a Array

- Podemos usar los métodos aplicables a un Array, por ejemplo filter

```
> db.Personas.find().toArray().filter(p=>p.edad=30)
[
        {
                "_id" : ObjectId("609564f9c1755000040dba77"),
                "nombre" : "Juan",
                "apellido" : "Perez",
                "edad" : 30
        },
        {
                "_id" : ObjectId("6095651fc1755000040dba78"),
                "nombre" : "Juan",
                "apellido" : "Perez",
                "edad" : 30
```

Convertir a Array

```
var allProductsArray = db.products.find().toArray();

if (allProductsArray.length > 0) { printjson (allProductsArray[0]); }
```

Método count() y size()

- db.nombreColeccion.find().count() -- no le afecta limit ni skip

```
> db.Personas.find().count()
12
>
```

- db.nombreColeccion.find().size() -- le afecta limit y skip

```
> db.Personas.find().limit(2).count()
12
> db.Personas.find().limit(2).size()
2
>
```

$elemMatch

- encuentra documentos que cumplen al menos un criterio de busqueda de un arreglo

- Suponga la colección scores:
  { _id: 1, results: [82, 85, 88] }
  { _id: 2, results: [75, 88, 89] }

```
db.scores.find(
   { results: { $elemMatch: { $gte: 80, $lt: 85 } } }
)
```

```
{ "_id" : 1, "results" : [ 82, 85, 88 ] }
```

Modelado de datos

- Una decisión clave en el diseño de modelos de datos en MongoDB es la manera como se estructura un documento y como se representan las relaciones entre datos
- Existen dos formas de representar las relaciones:
  - Documentos referenciados
  - Documentos embebidos

Modelado de datos

- Los documentos referenciados
  - Las referencias almacenan información que permite identificar el documento al cual se hace referencia (id)
  - Esto se usa en modelos normalizados
  - Pueden representar relaciones muchos a muchos muy complejas
  - Modelan conjuntos de datos jerárquicos muy grandes
  - Las referencias proporcionan mas flexibilidad que los documentos embebidos
  - Requieren mas queries para obtener los documentos referenciados

Modelado de datos

- Los documentos embebidos
  - Modelo desnormalizado
  - Permite el almacenamiento de documentos relacionados en el mismo “registro”
  - Las aplicaciones requieren menos consultas a la BD
  - Proporcionan mejor perfomance para las consultas porque pueden extraer todos los datos en una misma consulta

Relaciones 1 a 1

- Documentos embebidos

```
estudiante
{
        nombre: “Juan”,
        apellido: “Perez”,
        domicilio: {
                calle: “Hidalgo”
                numero: 333
                colonia: “Centro”
        }
}
```

Relaciones 1 a muchos

- Documentos embebidos

```
estudiante
{
        nombre: “Juan”,
        apellido: “Perez”,
        materias: [
                {nombre: “matematicas”, creditos: 6},
                {nombre: “fisica”, creditos: 6},
                {nombre: “programación”, creditos: 6},
        ]
}
```

Relaciones muchos a muchos

- Referencias

```
>
> db.Materias.find()
{ "_id" : 1, "nombre" : "matematicas" }
{ "_id" : 2, "nombre" : "fisica" }
{ "_id" : 3, "nombre" : "programacion" }
>
```

estudiante

{

nombre: “Juan”,

apellido: “Perez”,

materias: [1,2,3]

}

Ejercicio

- Indentifique en el modelo de Salón de Fiestas alguna relación 1 a muchos y muchos a muchos
- Realice el llenado de los documentos

Referencias

- Para muchos casos es suficiente el modelo desnormalizado usando documentos embebidos
- En algunos casos es necesario almacenar documenatos relacionados en colecciones diferentes
  - Referencias manuales
  - Referencias DBRefs

Referencias manuales

- Guardamos el _id del documento de otra colección en otro documento de otra colección
- Se debe ejecutar un segundo query para obtener el documento relacionado
- Este esquema puede ser suficiente en muchos caso

DBRefs

- Son referencias de un documento a otro por medio del campo _id, el nombre de la coleccion y opcionalmente el nombre de la base de datos
- Por medio de esos datos, DBRefs permite que los documentos localizados en multiples colecciones sean mas fáciles de enlazar en una sola colección
- Para resolver DBRefs, deben realizarse querys adicionales para obtener los documentos referenciados
- Se recomienda el uso de referencias manuales
- No todos los drivers soportan DBRefs

Ejemplo de DBRefs

```
{
  "_id" : ObjectId("5126bbf64aed4daf9e2ab771"),
  // .. application fields
  "creator" : {
                "$ref" : "creators",
                "$id" : ObjectId("5126bc054aed4daf9e2ab772"),
                "$db" : "users"
              }
}
```

Eliminar documentos

- db.nombreColeccion.remove({expresion}) -- elimina todos los documentos que coincidan con la expresión

```
> db.Personas.remove({edad: {$eq: 30}})
WriteResult({ "nRemoved" : 2 })
>
```

- db.nombreColeccion.remove({expresion}, true -- Elimina la primera ocurrencia

```
>
> db.Personas.remove({apellido: "Hdz"}, true)
WriteResult({ "nRemoved" : 1 })
>
```

Eliminar una colección

- db.nombreColeccion.drop()

Actualizar documentos

- db.nombreColeccion.updateOne({filter}, {$set: {$update}})
  -- solo actualiza un documento

```
> db.Personas.updateOne({
...     nombre: "Luis"
... },
... {
...     $set: {
...         nombre: "Gerardo"
...     }
... })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Es recomendable usar el ObjectId

Actualizar documentos

- db.nombreColeccion.updateMany({filter}, {$set: {$update}})
  -- actualiza varios documento

```
>
> db.Personas.updateMany({nombre: "Luis" }, {$set: {nombre: "Alejandro"} })
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
>
```

Actualizar documentos

- db.nombreColeccion.update() -- similar a updateOne()

```
>
> db.Personas.update({nombre: "Luis" }, {$set: {nombre: "Alejandro"}},{upsert: true} )
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("6099fc8b8092171a2d3281a4")
})
>
```

Si se agrega upsert: true, si no hay match en la condición se agrega un documento nuevo a la colección.

Actualizar documentos

- db.nombreColeccion.updateOne($unset) -- elimina una propiedad del documento

```
> db.Personas.updateOne({nombre: "Gerardo"},{$unset : {nombre: ""}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

aplica igual para updateMany

Actualizar documentos

$inc -- incrementa el valor de una propiedad

```
>
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$inc: {edad: 2} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

$rename -- renombra una propiedad

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$rename: {"edad": "anios"} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

- $mul -- multiplica el valor de una propiedad

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$mul: {anios: 2} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Actualizar documentos

$max -- actualiza solo si el valor es mayor al actual

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$max: {anios: 80} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

$min -- actualiza solo si el valor es menor al actual

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$min: {anios: 30} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Actualizar documentos

$push -- agrega un valor a una propiedad tipo arreglo

```
>
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$push: {grados: "licenciatura"} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

$addToSet -- agrega un valor a una propiedad tipo arreglo solo si no existe

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$addToSet: {grados: "licenciatura"} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
>
```

$each – agrega varios valores a un arreglo

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$push: {grados: {$each: ["primaria", "secundaria"]}} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }) },{$push: {grados: {$each: ["primaria", "secundaria"]}} })
>
```

Actualizar documentos

$sort -- agrega un valor a una propiedad tipo arreglo y la ordena

```
>
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$push: {grados: {$each: ["kinder"], $sort:1}} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
> _
```

Si queremos solo ordenar enviamos el arreglo vacío

```
>
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$push: {grados: {$each: [], $sort:-1}} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Actualizar documentos

$position -- agrega un valor a una propiedad tipo arreglo en una posición determinada

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$push: {grados: {$each: ["ninguno"], $position:0}} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

$pull -- elimina un valor de una propiedad tipo arreglo (todas las ocurrencias)

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$pull: {grados: "licenciatura" }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

$pullAll -- elimina varios valores de una propiedad tipo arreglo (todas las ocurrencias)

```
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$pullAll: {grados: ["licenciatura","ninguno"] }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Actualizar documentos

$pop -- elimina el primer o último elemento de un arreglo
1: último, -1: primero

```
>
> db.Personas.updateOne({_id:  ObjectId("609575acc1755000040dba80") },{$pop: {grados: 1 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Uso de operadores lógicos

$and

```
> db.Personas.find({$and: [{nombre: "Alejandro"},{anios: {$gt:20}}]})
{ "_id" : ObjectId("609575acc1755000040dba80"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "anios" : 30, "
ia" ] }
{ "_id" : ObjectId("60958553c1755000040dba82"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "grados" : [ "l
```

$or

```
> db.Personas.find({$or: [{nombre: "Alejandro"},{edad: {$gt:20}}]})
{ "_id" : ObjectId("6095699fc1755000040dba7c"), "nombre" : "Alejandro", "apellido" : "Rodriguez" }
{ "_id" : ObjectId("609575acc1755000040dba80"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "anios" : 30, "grados
ia" ] }
{ "_id" : ObjectId("60958553c1755000040dba82"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "grados" : [ "licenci
```

Uso de operadores lógicos

$not -- funciona con expresiones regulares

```
> db.Personas.find({nombre: {$not: /Alejandro/}})
{ "_id" : ObjectId("6095698fc1755000040dba7a"), "apellido" : "Rodriguez" }
{ "_id" : ObjectId("60956bbac1755000040dba7d"), "nombre" : "Arturo", "apellido" : "Hdz" }
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
>
```

$nor -- lista los que no cumplen las condiciones especificadas

```
> db.Personas.find({$nor: [{nombre: "Alejandro"},{edad: {$gt:20}}]})
{ "_id" : ObjectId("6095698fc1755000040dba7a"), "apellido" : "Rodriguez" }
{ "_id" : ObjectId("60956bbac1755000040dba7d"), "nombre" : "Arturo", "apellido" : "Hdz" }
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
>
```

Uso de operadores lógicos

$exists -- lista los documentos que contienen el campo
true: contiene, false: no lo contiene

```
> db.Personas.find({anios: {$exists: true}})
{ "_id" : ObjectId("609575acc1755000040dba80"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "anios" : 30, "g
ia" ] }
{ "_id" : ObjectId("60958553c1755000040dba82"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "grados" : [ "li
>
```

$mod -- evalua el residuo

```
> db.Personas.find({anios: {$mod: [2,0]}})
{ "_id" : ObjectId("609575acc1755000040dba80"), "nombre" : "Alejandro", "apellido" : "Rodriguez", "anios" : 30, "grados" :
ia" ] }
>
```

Documentos embebidos

```
db.Alumnos.insertMany(
[
   {
      nombre: "Luis",
      domicilio: {          calle: "Hidalgo",          numero: 123,          colonia: "Centro"        }
   },
   {
      nombre: ”Juan",
      domicilio: {          calle: "Hidalgo",          numero: 333,          colonia: "Centro"        }
   },
   {
      nombre: ”Maria",
      domicilio: {          calle: "Carranza",          numero: 111,          colonia: "Republica"        }
   }
])
```

Documentos embebidos

Obtener todos los que viven en la calle Hidalgo

```
> db.Alumnos.find({"domicilio.calle": "Hidalgo"})
{ "_id" : ObjectId("609c53a7c24e1b88273e8087"), "nombre" : "Luis", "domicilio" : { "calle" : "Hidalgo", "nu
"colonia" : "Centro" } }
{ "_id" : ObjectId("609c53a7c24e1b88273e8088"), "nombre" : "Juan", "domicilio" : { "calle" : "Hidalgo", "nu
"colonia" : "Centro" } }
>
```

```
> db.Alumnos.find().sort({"domicilio.colonia":-1})
{ "_id" : ObjectId("609c53a7c24e1b88273e8089"), "nombre" : "Ma
, "colonia" : "Republica" } }
{ "_id" : ObjectId("609c53a7c24e1b88273e8087"), "nombre" : "Lu
"colonia" : "Centro" } }
{ "_id" : ObjectId("609c53a7c24e1b88273e8088"), "nombre" : "Ju
"colonia" : "Centro" } }
```

Ejercicio

- Realice diversas consultas sobre el modelo de Salon de Fiestas

Operador $where

$where -- permite pasar una expresion de JS

```
> db.Personas.find({$where: function(){ return (this.nombre == "Maria")}})
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
>
```

es preferible usar $expr por cuestiones de perfomance

```
>
> db.Personas.find({$expr: {$eq: ["$nombre","Maria"]}})
{ "_id" : ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
>
>
```

```
> db.Budget.find( { $expr: { $gt: [ "$spent" , "$budget" ] } } )
{ "_id" : 1, "category" : "food", "budget" : 400, "spent" : 450 }
{ "_id" : 2, "category" : "drinks", "budget" : 100, "spent" : 150 }
{ "_id" : 5, "category" : "travel", "budget" : 200, "spent" : 650 }
>
```

Eliminar documentos

db.nombreColeccion.deleteOne({expresion})

```
> db.Personas.deleteOne({nombre:"Alejandro"})
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

db.nombreColeccion.deleteMany({expresion})

```
> db.Personas.deleteMany({nombre:"Alejandro"})
{ "acknowledged" : true, "deletedCount" : 4 }
>
>
```

Ejercicio

- Realice eliminaciones sobre el modelo de Salón de Fiestas

Índices en mongoDB

- Permite búsquedas muy rápidas sobre campos definidos
- Existe un índice por default sobre la propiedad _id
- Existen índices simples y compuestos

Índices en mongoDB

- Consultar los índices
  - db.nombreColeccion.getIndexes()
- Crear un índice
  - db.collection.createIndex({”name”:1}) -- 1 ascendente -1 descendente
- eliminar un índice
  - db.collection.dropIndex(“name”)
- índice único
  - db.collection.createIndex({“name”:1},{unique:true})

Índices en mongoDB

- índice background
  - db.collection.createIndex({“name”:1},{brackground:true})
- índice compuesto
  - db.collection.createIndex({carrera:1, matricula:1 }) -- afecta el orden en la consulta
- Buscar sobre un índice
  - db.collection.find().hint({carrera:1, matricula:1})
- No usar índices
  - db.Personas.find().hint({$natural:1})

Índices case insensitive

- db.colleccion.createIndex({nombre:1}, {
  collation: {
  locale: “en”,
  strength: 1 // 1 Ignora acentos, 2 no ignora acentos
  }
})

en : Inglés
es: Español

db.Personas.find({apellido: "perez"}).collation({locale:"es", strength:1})

Índices de texto

- Permite buscar información en diferentes campos de texto

```
db.Personas.createIndex({
    nombre: "text",
    apellido: "text"
})
```

Buscar:

```
db.Personas.find({    $text: {$search: "juan",
                              $caseSensitive: true    } // por default es false
        }
)
```

Tipos de datos

- BSON: formato de intercambio de datos (Binary JSON)
- ObjectId : Identificador único, se crea por la función ObjectId()
- Se puede crear su propio ObjectId
  - x=ObjectId("123456789012345678901234") // Debe ser 24 caracteres
- Se puede obtener la fecha y hora en que se creó
  - x.getTimestamp()
- Se puede obtener solo el valor
  - x.valueOf()

Tipos de datos

- Date() : devuelve una fecha como cadena
- new Date() : regresa una fecha como objeto ISODate
- new Date("2021-01-02") : crea la fecha específica

Tipos de datos

- NumberInt : Entero corto
- NumberLong : Entero largo
- typeof(15)
- typeof(NumberInt(15))
- typeof(NumberInt(15)+1)

- NumberDecimal(3232.99999999999999)
- NumberDecimal("3232.99999999999999")

Esquema

- Es un objeto JSON que permite definir la forma y contenido de los documentos que se almacenan en una colección

Por ejemplo:

```
{  "properties": {
    "_id": { "bsonType": "objectId" },
    "name": { "bsonType": "string" }
  }
}
```

rechaza el siguiente update

```
collection.updateOne( { "_id": BSON.ObjectId("5ae782e48f25b9dc5c51c4d0") },
  { "$set": { "name": 42 } }
)
```

Crear un esquema

```
> db.createCollection("Alumno",
... {
...     validator: {
...         $jsonSchema: {
...             bsonType: "object",
...             required: ["matricula", "nombre"],
...             properties:{
...                 matricula: {
...                     bsonType: "number",
...                     description: "Matrícula del alumno"
...                 },
...                 nombre: {
...                     bsonType: "string",
...                     description: "Nombre del alumno"
...                 }
...             }
...         }
...     }
... }
... )
{ "ok" : 1 }
>
```

Crear un esquema

```
> db.Alumno.insertOne({
...     matricula: "123",
...     nombre: "Juan"
... })
WriteError({
        "index" : 0,
        "code" : 121,
        "errmsg" : "Document failed validation",
        "op" : {
                "_id" : ObjectId("60aac04d2ca88f4a18a193c4"),
```

Esquema

- En una relación uno a muchos se puede referenciar el id de la colección

```
> db.createCollection("Alumno",
... {
...     validator: {
...         $jsonSchema: {
...             bsonType: "object",
...             required: ["matricula", "nombre", "idcarrera"],
...             properties:{
...                 matricula: {
...                     bsonType: "number",
...                     description: "Matrícula del alumno"
...                 },
...                 nombre: {
...                     bsonType: "string",
...                     description: "Nombre del alumno"
...                 },
...                  idcarrera: {
...                     bsonType: "number",
...                     description: "Id de la carrera del alumno"
...                 }
...             }
...         }
...     }
... )
{ "ok" : 1 }
```

Esquema

- En una relación muchos a muchos se puede establecer un arreglo de identificadores

```
> db.createCollection("Alumno",
... {
...     validator: {
...         $jsonSchema: {
...             bsonType: "object",
...             required: ["matricula", "nombre", "idcarrera"],
...             properties:{
                    matricula: {
                        bsonType: "number",
                        description: "Matrícula del alumno"
                    },
                    nombre: {
                        bsonType: "string",
                        description: "Nombre del alumno"
                    },
                     idcarrera: {
                        bsonType: "number",
                        description: "Id de la carrera del alumno"
                    },
                    materias: {
                        bsonType: "array",
                        description: "Ids de las materias que cursa el alumno"
                    }
...             }
...         }
...     }
... }
... )
{ "ok" : 1 }
```

```
> db.Alumno.insertOne({
...     matricula: 123, nombre: "Juan Perez", idcarrera: 11,
...     materias: [ ObjectId("60abefbdc34eafa0edcda953"),
...     ObjectId("60abefc3c34eafa0edcda954") ]
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("60abf031c34eafa0edcda956")
}
```

Esquema - validaciones

- Los esquemas pueden validar algunos aspectos de los datos

```
> db.Alumno.insertOne({
...     matricula: 13323,
...     nombre: "Juan 333",
...     idcarrera: 11,
...     edad: 201
... })
WriteError({
        "index" : 0,
        "code" : 121,
        "errmsg" : "Document failed validation",
        "op" : {
                "_id" : ObjectId("60abf38cc34eafa0
```

```
> db.createCollection("Alumno",
... {
...     validator: {
...         $jsonSchema: {
...             bsonType: "object",
...             required: ["matricula", "nombre", "idcarrera"],
...             properties:{
...                 matricula: { bsonType: "number",
...                     description: "Matrícula del alumno" },
...                 nombre: { bsonType: "string",
...                     description: "Nombre del alumno" },
...                  idcarrera: { bsonType: "number",
...                     description: "Id de la carrera del alumno"  },
...                 edad: {
...                     bsonType: "number",
...                     description: "Edad del alumno",
...                     minimum: 0,
...                     maximum: 200
...                 }
...             }
...         }
...     }
... }
... )
{ "ok" : 1 }
```

Esquema - validaciones

- Los esquemas pueden validar algunos aspectos de los datos

```
> db.Alumno.insertOne({
...     matricula: 13323,
...     nombre: "Juan 333",
...     edad: 20,
...     sexo: "M"
... })
WriteError({
        "index" : 0,
        "code" : 121,
        "errmsg" : "Document failed validation",
```

```
> db.createCollection("Alumno",
... {
...     validator: {
...         $jsonSchema: {
...             bsonType: "object",
...             required: ["matricula", "nombre"],
...             properties:{
...                 matricula: { bsonType: "number",
...                     description: "Matrícula del alumno" },
...                 nombre: { bsonType: "string",
...                     description: "Nombre del alumno" },
...                 edad: {
...                     bsonType: "number", description: "Edad del alumno",
...                     minimum: 0,maximum: 200
...                 },
...                 sexo: {
...                     enum: ["Masculino", "Femenino"],
...                     description: "Sexo del alumno"
...                 }
...             }
...         }
...     }
... }
... )
{ "ok" : 1 }
```

Ejercicio

- Elija alguna colección del modelo de Salón de Fiestas y elabore su esquema correspondiente

Vistas

- Ofrece vistas de solo lectura sobre colecciones existentes u otras vistas:
  - Permite excluir información privada o confidencial
  - Una vista puede agregar campos calculados
  - Una vista puede hacer JOIN de datos de dos colecciones

Vistas

- Ofrece consultas restringidas

```
>
> db.createView("vistaMasculinos",
...     "Alumno",
...     [{$match: {sexo: "Masculino" }} ])
{ "ok" : 1 }
>
```

```
>
> db.vistaMasculinos.find()
{ "_id" : ObjectId("60abf49dc34eafa0edcda95a"), "matricula" : 13323,
lino" }
>
>
```

- no se puede insertar ni actualizar a través de una vista

Ejercicio

- Elabore algunas vistas sobre el modelo Salón de Fiestas

Agregaciones

- Permite transformar y combinar documentos de una colección
- Básicamente se construye un pipeline que procesan los documentos a través de diversos bloques:
  - filtros
  - proyecciones
  - grupos
  - ordenamientos
  - limites
  - saltos

Agregaciones

- Por ejemplo: obtener los 5 autores con mayor productividad, asumiendo que los articulos están en una colección:
  1. Proyectar los autores de cada artículo
  2. Agrupar los autores por nombre, contando las ocurrencias
  3. Ordenar por el conteo de ocurrencias en forma descendente
  4. Limitar el resultado a 5

Agregaciones

- {"$project" : {"author" : 1}}
- {"$group" : {"_id" : "$author", "count" : {"$sum" : 1}}
- {"$sort" : {"count" : -1}}
- {"$limit" : 5}

```
>
>     db.articles.aggregate({"$project" : {"author" : 1}},
...       {"$group" : {"_id" : "$author", "count" : {"$sum" : 1}}},
...       {"$sort" : {"count" : -1}},
...      {"$limit" : 5})
{ "_id" : "Smith", "count" : 7 }
{ "_id" : "Sanchez", "count" : 5 }
{ "_id" : "Suarez", "count" : 4 }
{ "_id" : "Rdz", "count" : 1 }
{ "_id" : "Lopez", "count" : 1 }
>
```

Agregaciones $match

- Filtra documentos en base a una expression. Se recomienda que se ponga lo mas pronto possible.

```
>
>   db.articles.aggregate({"$match": {"author": "Sanchez"}})
{ "_id" : ObjectId("60abff95c34eafa0edcda960"), "author" : "Sanchez", "title" : "mat1" }
{ "_id" : ObjectId("60abff95c34eafa0edcda961"), "author" : "Sanchez", "title" : "mat2" }
{ "_id" : ObjectId("60abff95c34eafa0edcda962"), "author" : "Sanchez", "title" : "mat3" }
{ "_id" : ObjectId("60abff95c34eafa0edcda963"), "author" : "Sanchez", "title" : "mat4" }
{ "_id" : ObjectId("60abff95c34eafa0edcda964"), "author" : "Sanchez", "title" : "mat5" }
>
```

- Se puede combinar con otros operadores incluso con sí mismo

```
db.Personas.aggregate({"$match": {"apellido": "Rodriguez"}}, {"$match": {"nombre": "Maria"}})
ObjectId("60956bbac1755000040dba7e"), "nombre" : "Maria", "apellido" : "Rodriguez" }
```

Agregaciones $project

- Extrae campos de documentos, renombra campos y realiza operaciones

```
> db.articles.aggregate({"$project" : {"author" : 1, "_id" : 0}})
{ "author" : "Suarez" }
{ "author" : "Suarez" }
{ "author" : "Suarez" }
{ "author" : "Suarez" }
{ "author" : "Sanchez" }
```

- Renombra

```
> db.articles.aggregate({"$project" : {"authorId" : "$_id", "_id": 0}})
{ "authorId" : ObjectId("60abff95c34eafa0edcda95c") }
{ "authorId" : ObjectId("60abff95c34eafa0edcda95d") }
{ "authorId" : ObjectId("60abff95c34eafa0edcda95e") }
{ "authorId" : ObjectId("60abff95c34eafa0edcda95f") }
{ "authorId" : ObjectId("60abff95c34eafa0edcda960") }
```

Agregaciones $project

- No hace uso de indices cuando se renombra un campo

```
> db.articles.aggregate({"$project" : {"autor" : "$author"}},
...                   {"$sort" : {"autor" : 1}})
{ "_id" : ObjectId("60abff95c34eafa0edcda96f"), "autor" : "Diaz" }
{ "_id" : ObjectId("60abff95c34eafa0edcda96c"), "autor" : "Gomez" }
{ "_id" : ObjectId("60abff95c34eafa0edcda96d"), "autor" : "Lopez" }
{ "_id" : ObjectId("60abff95c34eafa0edcda96e"), "autor" : "Rdz" }
{ "_id" : ObjectId("60abff95c34eafa0edcda960"), "autor" : "Sanchez" }
```

Agregaciones matemáticas

- $add

```
> db.employees.aggregate({
...     "$project" : {
...         "totalPay" : {
...             "$add" : ["$salary", "$bonus"]
...         }
...     }
... })
{ "_id" : ObjectId("60ad3df0c34eafa0edcda970"), "totalPay" : 1100 }
```

- $substract

```
> db.employees.aggregate({
...     "$project" : {
...         "totalPay" : {
...             "$subtract" : [{"$add" : ["$salary", "$bonus"]}, 100]
...         }
...     }
... })
{ "_id" : ObjectId("60ad3df0c34eafa0edcda970"), "totalPay" : 1000 }
```

Agregaciones matemáticas

- $multiply
- $divide
- $mod
- $ceil
- $floor

Agregaciones de fechas

- $year
- $month
- $week
- $dayOfMonth
- $dayOfWeek
- $dayOfYear
- $hour
- $minute
- $second

```
> db.employees.aggregate({
...     "$project" : {
...         "hiredIn" : {"$month" : "$hiredate"}
...     }
... })
{ "_id" : ObjectId("60ad4323c34eafa0edcda971"), "hiredIn" : 5 }
```

Agregaciones de strings

- $substr
- $concat
- $toLower
- $toUpper

```
> db.employees.aggregate({
...     "$project" : {
...         "inicial" : {
...             "$substr" : ["$name", 0,1]
...         }
...     }
... })
{ "_id" : ObjectId("60ad4323c34eafa0edcda971"), "inicial" : "J" }
>
```

- $dateToString

```
> db.employees.aggregate({
...     "$project" : {
...         "hiredIn" : {$dateToString: {format:"%d/%m/%Y", date: "$hiredate"} }
...     }
... })
{ "_id" : ObjectId("60ad4323c34eafa0edcda971"), "hiredIn" : "25/05/2021" }
>
>
```

Expresiones Lógicas

- $cmp -- retorna 0 si son iguales
- $strcasecmp -- comparaciòn case insensitive
- $eq $ne $gt $gte $lt $lte
- $and : [expr1, expr2, ..]
- $or
- $not
- $cond : [boolExpr, trueExpr, falseExpr]
- $ifNull : [expr, replExpr]

Expresiones Lógicas

Suppose a professor wanted to generate grades using a somewhat complex calculation: the students are graded 10% on attendance, 30% on quizzes, and 60% on tests (unless the student is a teacher's pet, in which case the grade is set to 100). We could express these rules as follows:

```
> db.students.aggregate(
... {
...     "$project" : {
...         "grade" : {
...             "$cond" : [
...                 "$teachersPet",
...                 100, // if
...                 {    // else
...                     "$add" : [
...                         {"$multiply" : [.1, "$attendanceAvg"]},
...                         {"$multiply" : [.3, "$quizzAvg"]},
...                         {"$multiply" : [.6, "$testAvg"]}
...                     ]
...                 }
...             ]
...         }
...     }
... })
```

Ejercicio

- Diseñe algunos reportes básicos del modelo Salón de Fiestas y elabore las proyecciones necesarias para generar dichos reportes

Agrupado

- Se permite la agrupación en ciertas propiedades

```
> db.articles.aggregate({"$group" : {_id: "$author"} })
{ "_id" : "Rdz" }
{ "_id" : "Suarez" }
{ "_id" : "Smith" }
```

Siempre hay que pasar el _id para que pueda agrupar por la propiedad determinada

Agrupado

- Operadores
  - $sum
  - $avg
- De frontera
  - $max
  - $min
  - $first
  - $last

```
> db.articles.aggregate({
...     "$group" : {
...         _id : "$author",
...         "numero" : {"$sum" : 1}
...     }
... })
{ "_id" : "Rdz", "numero" : 1 }
{ "_id" : "Lopez", "numero" : 1 }
{ "_id" : "Sanchez", "numero" : 5 }
```

```
> db.scores.aggregate(
... {
...     "$group" : {
...         "_id" : "$grade",
...         "lowestScore" : {"$min" : "$score"},
...         "highestScore" : {"$max" : "$score"}
```

Crear colecciones a partir de una consulta

- Se utiliza $out

```
> db.articles.aggregate([{$match: {author: "Suarez"}}, {$out: "SuarezArticles"}])
> db.SuarezArticles.find()
{ "_id" : ObjectId("60abff95c34eafa0edcda95c"), "author" : "Suarez", "title" : "prog1" }
{ "_id" : ObjectId("60abff95c34eafa0edcda95d"), "author" : "Suarez", "title" : "prog2" }
{ "_id" : ObjectId("60abff95c34eafa0edcda95e"), "author" : "Suarez", "title" : "prog3" }
{ "_id" : ObjectId("60abff95c34eafa0edcda95f"), "author" : "Suarez", "title" : "prog4" }
>
```

Joins

- Une dos colecciones mediante una propiedad específica

```
{
    $lookup: {
        from: <coleccion a unir>,
        localField: <campo de coleccion de entrada>,
        foreignField: <campo de colección "from">,
        as: <campo de salida>
    }
}
```

```
> db.Ciudad.aggregate([
...     {
...         $lookup: {
...             from: "Estado",
...             localField: "idEstado",
...             foreignField: "_id",
...             as: "Ciudad_estado"
...         }
...     }
... ])
{ "_id" : 1, "nombre" : "Saltillo", "idEstado" : 2, "Ciudad_estado" : [ { "_id" : 2, "nombre" : "Coahuila" } ] }
{ "_id" : 2, "nombre" : "Monclova", "idEstado" : 2, "Ciudad_estado" : [ { "_id" : 2, "nombre" : "Coahuila" } ] }
>
```

Joins

```
>
> db.Ciudad.aggregate([
...     {
...         $lookup: {
...             from: "Estado",
...             localField: "idEstado",
...             foreignField: "_id",
...             as: "Ciudad_estado"
...         }
...     },
...     {
...         $project: {
...             nombreCiudad: "$nombre",
...             nombreEstado: "$Ciudad_estado.nombre"
...         }
...     }
... ])
{ "_id" : 1, "nombreCiudad" : "Saltillo", "nombreEstado" : [ "Coahuila" ] }
{ "_id" : 2, "nombreCiudad" : "Monclova", "nombreEstado" : [ "Coahuila" ] }
>
>
```

Ejercicio

- Elabore algún reporte que requiera JOIN de dos colecciones

MongoDB Compass

Programación de mongoDB con Node.js

Programación de mongoDB con PHP

Seguridad

- Existen dos bases de datos:
  - admin y local
- Existen roles predefinidos
  - read
  - readWrite
  - dbAdmin
  - userAdmin

```
> use admin
switched to db admin
> db.createUser(
...     {
...       user: "appAdmin",
...       pwd: "456",
...       roles:
...         [
...           { role: "readWrite", db: "Personas" }
...         ]
...     }
... )
Successfully added user: {
        "user" : "appAdmin",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "Personas"
                }
        ]
}
```

Seguridad

```
C:\Users\urbanoflores>mongo -u root -p --authenticationDatabase admin --host 127.0.0.1 --port 27017
MongoDB shell version v4.4.6
Enter password:
```