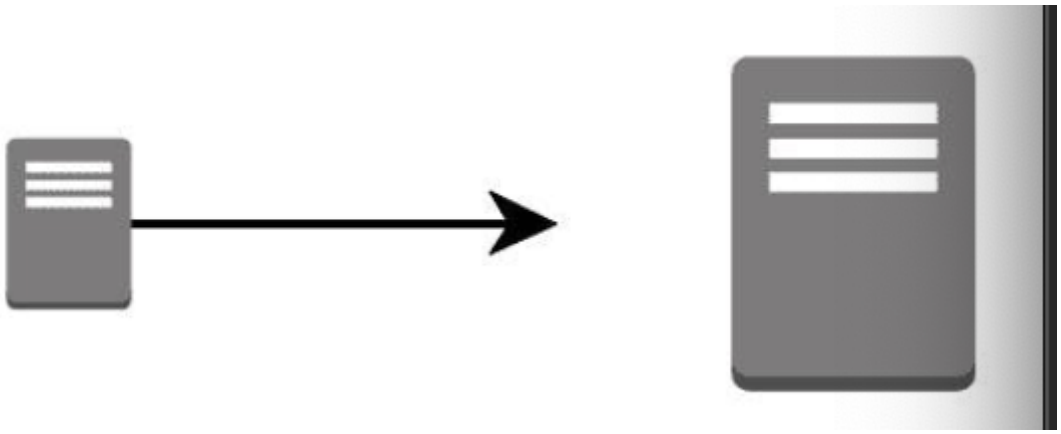
NoSQL

Introducción

BBDD SQL

- Las bases de datos relacionales aparecieron en los años 70
- La información se organiza en tablas y relaciones
- Scale up (vertical)
    - mejorar el equipo
    - puede ser muy car



BBDD NoSQL

- Las bases de datos no relacioes aparecieron en los años 60
- El término NoSQL
    - Not only SQL
    - Apareció por primera vez en 1998
- Soluciones específicas para la necesidades de grandes empresas
- 63% de las empresas gestionan datos superiores a los 50 PB
    - la mitad de la información es desestructurada
- Scale out (Horizontal)
    - añadir más máquinas
    - problemas de licencias

- Resolver el impedance mismatch
    - Conversión de clases y objetos a una tabla
    - Encapsulación
    - Herencia
    - Tipos de datos

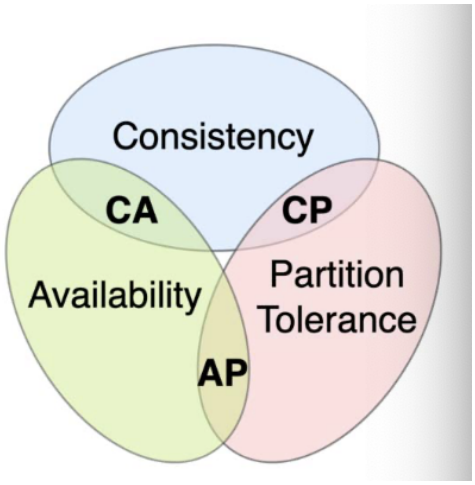
ACID

Garantizar la validez en caso de fallo

- Atomicity
  - la transacción se ejecuta completa o no se ejecuta
- Consistency
  - Las transacciones mantienen la validez
- Isolation
  - No importa el orden en el que se ejecutan las transacciones
- Durability
  - Cuando la transacción se ha completado se guarda en memoria no volátil

CAP

- El teorema de brewer
- Hay que elegir dos de entre
  - Consistency
    - Las lecturas reciben el valor más reciente o un error
  - Availability
    - Las peticiones reciben una respuesta que no sea un error
  - Partition tolerance
    - El sistema funciona aunque se pierdan paquetes o haya retardos



BASE

- También formulado por brewer
- Version ACID de NoSQL
- Propiedades
  - Basic Availability
    - el sistema está disponible en caso de fallo
  - Soft state
    - el estado puede cambiar sin interacciones
  - Eventual consistency
    - Después de un tiempo sin inputs, el sistema será consistente

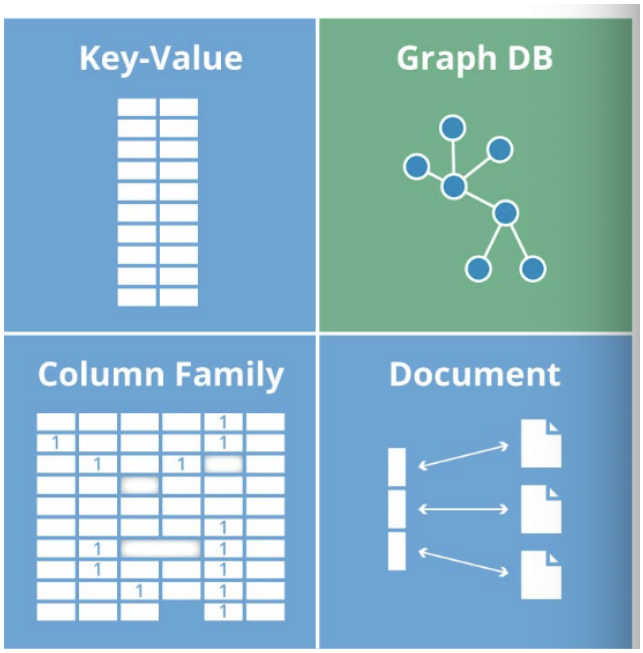
NoSQL

- Características NoSQL
  - No hay un modelo relacional
  - No usan SQL
  - Pensando para sistemas distribuidos (clusters)
  - Schemaless
    - puedes añadir datos sin tener que definir primero la estructura
  - Open Source
  - The pace of development with NoSQL db can be much faster than with a SQL database
  - The structure of many different forms of data is more easily handled and evolved with a NoSQL database
  - The amount of data in many applications cannot be served affordably by a sql database
  - The scale of traffic and need for zero downtime cannot be handled by sql
  - New application paradigms can be more easily supported

Tipos DB NoSQL

- Document
  - Almacenan datos semiestructurados
    - JSON
    - BSON
    - XML
  - Se pueden anidar los documentos
  - Elementos específicos se pueden indexar para acelerar el acceso
  - Casos de uso
    - Ecommerce platforms
    - Trading platforms
  - Implementaciones
    - MongoDB
    - AWS DynamoDB
- Key-value
  - Pareja clave - valor (hasta table o diccionario)
  - Equivalente a una tabla SQL con dos columnas
  - Casos de uso
    - Shopping carts
    - User preferences
    - User profiles
  - Implementaciones
    - AWS DynamoDB
    - Redis
- Wide-column
  - La información se organiza en columnas
  - El nombre y el formato de las columnas puede variar entre filas
  - Cada columna se almacena de forma separada en el disco
  - Casos de uso:
    - Analytics

    - Implementaciones
        - Cassandra
        - Google Bigtable
        - ScyllaDB
- Graph
    - Se centra en las relaciones entre los elementos cada elemento es un nodo
    - Las relaciones son las conexiones
    - Casos de uso
        - Fraud detection
        - Social networks
    - Implementaciones
        - Amazon neptune
        - Neo4j



Polyglot persistence:

- Usar diferentes tecnologías en función de las necesidades
- No usar una misma base de datos para todo
- Basado en polyglot programming

MongoDB

- Bases de datos NoSQL
- Document DB
- Gratis
- Modelo de negocio
    - Entrenamiento
    - Soporte
    - DB as a service (Atlas)
- Casi open source

.

MongoDB - características

- Ad-hoc queries (MQL)
- índices
- Replicación
- Load balancing
- Pipelines de agregación
- Transacciones

MongoDB - conceptos

- Document
  - a way to organize and store data as a set of field-value pairs
- Field
  - A unique identifier for a datapoint
- Value
  - data related to a given identifier
- Collection
  - An organized store of documents in MongoDB
  - Usually with common fields between documents
  - There can be many collections per database
  - There can be many documents pero collections
- Replica set
  - A few connected machines that store the same data
  - They ensure that if something happes to one of the machines the data will reamain intact
  - Comes from the word replicate - to copy something
- Instance
  - A single machine locally or in the cloud, running a certain software, in our case it is the mongodb database
- Cluster
  - group of severs that store your data

MongoDB - Mongo Shell

- cls
  - clear
  - limpiar la consola
- show dbs
  - Mostrar las bases de datos
- Db
  - devuelve el nombre de la base de datos activa
- use nombreDB
  - Cambiar a la base de datos nombreDB

MongoDB - Collections

- show collections

- show collections
    - Muestra las colecciones de la DB activa
        - db.createCollection(<name>, <options>)
            - Crea una nueva colección
            - Al insertar un elemento, se crea automáticamente si no existe
        - db.<collection>.drop()
            - elimina la colecćon

MongoDB - Read

- db.<collection_name>.findOne()
    - Devuelve un elemento de la colección
- db.<collection_name>.find()
    - Devuelve todos los elementos de la colección
    - si hay más de 20, devuelve 20 y un puntero para recorrerlos it
        - it
            - avanza los siguiente 20 elementos
- db.<collection_name>.find().pretty()
    - Devuelve los elementos para que sea más fácil de leerlos
- db.<collection_name>.find().count()
    - Devuelve el número de elementos
- db.<collection_name>.find(query)
    - Se puede añadir una query para filtrar la busqueda
        - eje
            - db.zips.find({"state: "AL"}).count

MongoDB - Insert

- db.<collection_name>.insertOne(<documents>)
    - Inserta el documento en la colección
- db.<colleciton_name>.insertMany([<documento1>, <documento2>])
    - Inserta múltiples documentos
- Al insertar un elemento, si no existe la colección o la base de datos se crean en ese momento
- Cada documento de una colección tiene que tener un _id único
- Si no lo indicamos nosotros al insetar se genera un ObjectId
    - Si intentamos insertar uno que ya existe salta una excepción
        - E11000 duplicate key error collection
        - Si estabamos insertando múltiples, no se insertan los que falten
            - podemos añadir la opcion
                - { "ordered" : false }
                    - para que se inserten

MongoDB - Update

- db.<collection_name>. updateOne(filter,update, options)
- db.<collection_name>. updateMany(filter,update, options)
    - Actualiza el valor de un campo

    - filter selection criteria
    - update modificaión a aplicar
- db.<collection_name>.replaceOne(filter,replacement, options)
    - Remplaza un documento por otro
    - replacement: nuevo documento
- Operadores para update
    - $set : Para cambiar un valor
    - $unset : para eliminar un campo
    - $inc : para incrementar un número
    - $push : para añadir un elemento a un array

MongoDB - Delete

- db.<colleciton_name>.deleteONe (filter, options)
- db.<colleciton_name>. deleteMany(filter, options)
    - elimina los documentos que cumplan el filtro
        - filter
            - criterio de selección

Comparadores

- $eq
    - =
    - por defecto
    - :
- $ne
    - not equal
    - !=
- $gt
    - greater than
    - >
- $lt
    - less than
    - <
- $gte
    - greater than or equal
    - >=
- $lte
    - less than or equal
    - <=

Operadores lógicos

- $and
    - por defecto
    - ,
- $not

- $nor
- $or

Ejercicios 1

- db.zips.find({pop : {$lt : 1000}}).count() -
- db.trips.find({"birth year": {$gt : 1998}}).count() - db.trips.find({"birth year": {$eq : 1998}}).count() -
- db.routes.find({stops : {$gte: 1}}).count() -
- db.inspections.find({result : "Out of Business"}).count()
  - db.inspections.find({sector : "Home Improvement Contractor - 100"}).count()
  - db.inspections.find({result : "Out of Business", sector : "Home Improvement Contractor - 100"}).count() -
- db.inspections.find({date : "Feb 20 2015"}).count()
- db.inspections.find({date : "Feb 21 2015"}).count()
- db.inspections.find({"$or" : [{date : "Feb 20 2015"}, {date : "Feb 21 2015"}]}).count()
- db.inspections.find({ $and : [{"$or" : [{date : "Feb 20 2015"}, {date : "Feb 21 2015"}]}, {sector : "Cigarette Retail Dealer - 127"}]}).count()
- db.inspections.find({ $and : [{"$or" : [{date : "Feb 20 2015"}, {date : "Feb 21 2015"}]}, {sector : { $ne : "Cigarette Retail Dealer - 127"}}]}).count() -
- db.inspections.find({"$or" : [{date : "Feb 20 2015"}, {date : "Feb 21 2015"}]}, {sector : { $ne : "Cigarette Retail Dealer - 127"}}).count()
  - MAL
- db.inspections.find({ "$or": [{ date: "Feb 20 2015" },{ date: "Feb 21 2015" }],sector: { $ne: "Cigarette Retail Dealer - 127" }}).count()
  - BIEN

Expressive $expr

- Allow us to use variables and conditional stetements
- {"$expr": { <expression>}}
- "$filename" => referencia el valor de ese field

"$elemMatch"

- Usado con arrays, los proyecta sólo si tienen un elemento que cumpla el criterio

Dot notation

- Usada para recorrer subdocumentos
- se pueden usar números para las posiciones de los arrays
- Tiene que estar entre comillas

Array operators

- $push
  - añadir un elemento a un array
    - crearlo si no existe
      - En sample_training.listingsAndReviews

        - En sample_training.listingsAndReviews
            - {"amenities": "Shampoo"}
                - Permite buscar en el array
            - {"amenities": ["Shampoo"]}
                - Busca exactamente ese array
                - Recordatorio, en los arrays el orden importa

"$all" : [ ]

- Para buscar arrays que contengan al menos esos elementos

$size : number

- Para especificar un tamaño exacto del array

Projection

- db.<collection>.find({<query>}, {<projection>})
- Permite obtener sólo los campos que queremos

"$regex"

- Para buscar patrones en strings usando regex

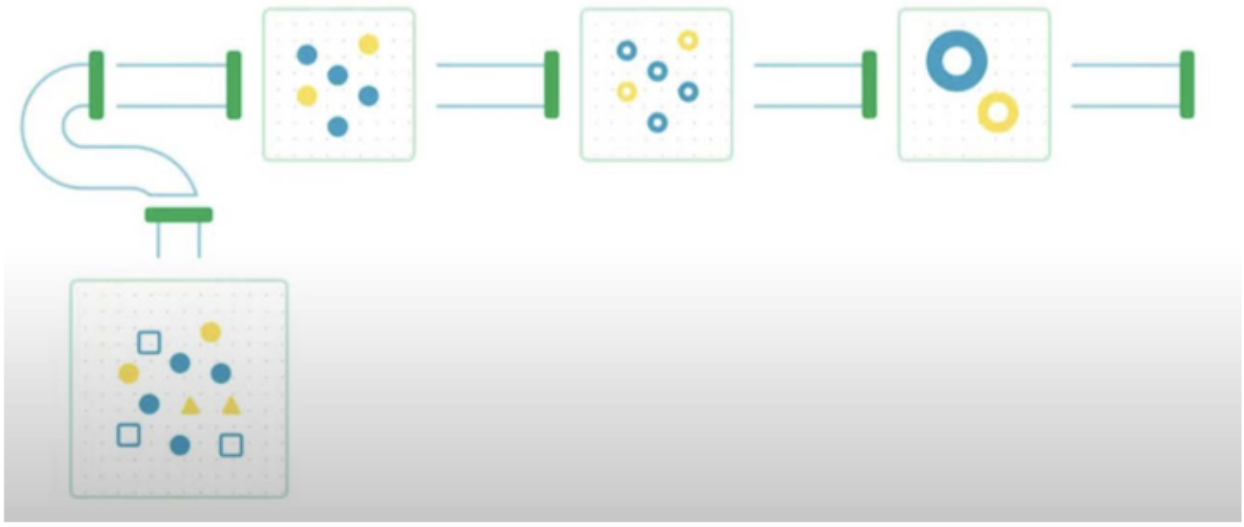
Ejercicios 2

- db.companies.find({$expr : {$gt : ["$number_of_employees" , "$founded_year"]}}).count()
- db.companies.find({$expr : {$eq : ["$permalink" , "$twitter_username"]}}).count()
- db.listingsAndReviews.find({"accommodates" : { $gt: 6}, "reviews" : {$size : 50}}, {"name": 1})
    - db.listingsAndReviews.find({"accommodates" : { $gt: 6}, "reviews" : {$size : 50}}, {"name": 1,"_id":0})
    - db.listingsAndReviews.find({"accommodates" : { $gt: 6}, "number_of_reviews" : 50}, {"name": 1})
    - db.listingsAndReviews.find({"accommodates" : { $gt: 6}, "number_of_reviews" : 50}, {"name": 1, "_id":0})
    - db.listingsAndReviews.findOne({"accommodates" : { $gte: 6}},{"name": 1})
    - db.listingsAndReviews.find({"reviews" : {$size : 50}}).count()
- db.listingsAndReviews.find({"property_type": "House", "amenities" : "Changing table"}).count()
    - db.listingsAndReviews.find({"amenities" : "Changing table"}).count()
- db.companies.find({"offices" : { $elemMatch : { city : "Seattle"}}}).count()
- db.companies.find({"offices.city" : "Seattle"}).count()
- db.companies.find({"funding_rounds" : { $size : 8}},{"name" : 1})
- db.trips.find({"start station location.coordinates.0" : {$lt : -74}}).count()
- db.inspections.find({"address.city" : "NEW YORK"}).count()
- db.listingsAndReviews.find({"amenities.0" : "Internet" }, {"name" : 1, "address" : 1})

MongoDB - Aggregation

- Pipeline para realizar operaciones sobre los datos

- Compuesto de varias etapas(stages)



db.<collection>.aggregate([{ <stage> }, ...])

- Una o más etapas aplicadas en orden
- Stages:
  - $match
  - $project
  - $group

Importa el orden de los stages

$match

- {$match : {<query>} }
  - Devuelve todos los documentos que cumplan la query
  - Las queries son equivalentes a las de lectura
    - los find
  - db.listingsAndReviews.aggregate([{ "$match": { "amenities": "Wifi" } }])

$project

{$project: { <specification(s)> }}

- Devuelve todos los documentos que cumplan la query

- Las queries son equivalentes a las de lectura (los find)
- Más información
- Ej.:

```
db.listingsAndReviews.aggregate([
    {"$match": {"amenities": "Wifi"}},
    {"$project": {"price": 1, "address": 1}}])
```

$group

```
{$group:{
  _id: <expression>, // Group By Expression
  <field1>: { <accumulator1> : <expression1> }, ...
}}
```

- Agrupa todos los elementos que sean iguales
- <field>:
  - Operación a realizar
  - Ej.: $count, $max, $min, $sum…

$group

- Ej.:

```
db.listingsAndReviews.aggregate([
  { "$project": { "address": 1, "_id": 0 }},
  { "$group": { "_id": "$address.country" }}])

db.listingsAndReviews.aggregate([
    { "$project": { "address": 1, "_id": 0 }},
    { "$group": { "_id": "$address.country", "count": { "$sum":
1 } } }] )
```

Cursos Methods

- Métodos que se aplican a un cursor (por ejemplo lo que devuelve un find())
- Listado
  - Ej

        - count()
        - limit()
            - Número de elementos que se devuelven
                - 0 no hay limite
        - pretty()
        - skip()
            - Número de elementos al principio que ignoro
            - Útil junto con limit() para hacer paginado
        - sort()
            - Ordenar los elementos
            - Si no hay un campo único, el resultado puede ser inconsistente
            - Máximo 32 fields
            - value:
                - 1 para orden ascendente
                - -1 para orden descendente

MongoDB - Índices

- db.<collection>.createIndex()
    - Crear un indice para facilitar las busquedas
    - Hay creado un índice por defecto para _id

MongoDB - Upsert

- db.<collection>.updateOne({<query>}, {<update>}, {"upsert": true})
- Híbrido entre update e insert
    - Si hay coincidencia se actualiza
    - Si no hay una coincidencia se inserta
    - Usarlo sólo cuando se necesita

MongoDB - Modelado de datos

- A way to organize fields in a document to support your application performance and querying capabilities
- Data is stored in the way that it is used
- Data used together should be stored together

MongoDB - Validación

- MongoDB permite usar JSON Schema
- Se puede configurar al crear una colección
- También se puede configurar a posteriori
- Si no es válido el elemento
    - Devuelve un error
        - opción por defecto
    - Se puede configurar para que devuelva un warning

```
db.createCollection("contacts", { validator: {
    $jsonSchema: {
      type: "object",
      required: [ "phone", "name" ],
      properties: {
        phone: {type: "string"},
        name: {type: "string"}
}}})

db.contacts.insertOne({phone:"91123123", name:"Juan"})
db.contacts.insertOne({phone:91123123, name:"Sofía"})
```

BSON

- objectId
- ObjectId("56d5f7eb604eb380b0d8d8ce").getTimestamp()
  - ponerlo así

Ejercicios 3

- db.listingsAndReviews.aggregate([{ "$project": { "room_type": 1}},{ "$group": { "_id": "$room_type" }}])
  - db.listingsAndReviews.aggregate([{ "$group": { "_id": "$room_type" }}])
- db.companies.find({"founded_year": {$ne: null}}, {"name" : 1 , "founded_year":1}).sort({ "founded_year": 1 }).limit(5)
- db.trips.find({"birth year" : {$ne : ""}},{"birth year" : 1}).sort({"birth year": -1}).limit(1)