



# TEMA 4: NO SQL

## Introducción

- **ERD** SQL → scale up (vertical) + CPU, RAM, ...  
mejorar mucho, puede ser caro
- **ERD NoSQL** → scale out (horizontal) ↑ n° DB  
+ máquinas, problemas de licencia
- **Resumen de "Incompatibilidad"**
  - en SQL es difícil mappear obj (clases) a tablas de bases de datos por tienen modelos diferentes
  - obj en herencia, encapsulación, listas anidadas, ...
- **ACID** → garantizar la validez en rango de fallos RELACIONALES
  - **Atomicity**: la transacción se ejec. o no se ejec.
  - **Consistency**: las transacciones mantienen la validez
  - **Isolation**: no imp. el orden en el q se ejec. las transac.
  - **Durability**: cuando la transac. se ha completado se guarda en memoria no volátil

## El Teorema de Brewer CAP

Hay q elegir dos entre:

- **Consistency**: los lectores reciben el valor más reciente o un error.
- **Availability**: los lectores reciben una respuesta q no sea un error
- **Partition Tolerant**: El Sist. funciona aunque se pierdan mensajes o haya retrasos



## BASE (el formulado por Brewer)

verdadero ACID de NoSQL

### Propiedades

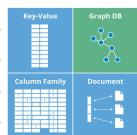
- **Basic Availability**: el Sist. esté disponible en rango de fallo
- **Soft State**: el estado puede cambiar con interacciones
- **Eventual Consistency**: después de un g de inputs, el Sist. sera consistente

## NoSQL

- No hay un modelo relacional, No usan SQL
- Pensado para Sist. distribuidos (clusters)
- **schemaless**: puedes añadir datos en rango q definir primero la estructura.
- **Open-Source**

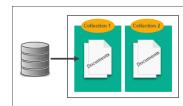
- El ritmo de desarrollo con bases de datos NoSQL puede ser mucho más rápido que con una base de datos SQL.
- La estructura de muchas formas diferentes de datos se gestiona y evoluciona más fácilmente con una base de datos NoSQL.
- La cantidad de datos en muchas aplicaciones no puede ser atendida de forma asequible por una base de datos SQL.
- La escala del tráfico y la necesidad de cero tiempo de inactividad no puede ser manejada por SQL.
- Nuevos paradigmas de aplicación pueden ser soportados más fácilmente.

- 4 tipos de  
BD NoSQL



### Document Databases

- Almacena datos semi estructurados (JSON, BSON o XML)
- Se pueden combinar los dos
- ELEM. específicos de pueden indexar para acelerar el acceso
- Casos de uso:
  - E-commerce platforms
  - Trading platforms
- Implementaciones
  - MongoDB
  - AWS DynamoDB



### Key-Value Stores → Pares de clave valor

- (hash table o diccionario)

[> Tabla 2 columnas de SQL]

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2/01/2015
K5	3,777,5623

## Wide-column store

- La info se org. en columnas
- nombre y formato de las columnas puede variar entre filas

### Casos de Uso

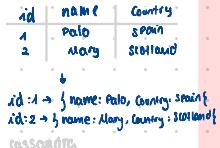
- Analytics

### Implementaciones

- Cassandra

- Google Bigtable

- Amazon DB



## Graph - Data Lakes

- se centra en la relación entre los elemos.  
"conexiones" cada uno es un NODO

### Casos de Uso:

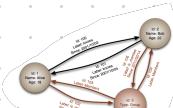
- Detec. de Fraude

- Redes Sociales

### Implementaciones

- Amazon Neptune

- Neo4j



## Polyglot Persistence

- Usar ≠ techs en función de las necesidades
- No usar la = para todo
- Basado en Polyglot Programming

## JSON

### Binary JavaScript Object Notation

• Versión codificada en binaria de JSON

### ofrece más tipos de datos

- date
- regex
- timestamp



## MongoDB

- Document DB, Gratis, con open-source Enterprise
- Modelo de Negocio → Data as a Service (DaaS)
- Ad-hoc queries (CQL)
- Índices
- Replicación
- Load Balancing
- Pipelines de Agregación
- Trasformaciones

### Conceptos

- document: una forma de organizar y almacenar datos como un conjunto de claves (campo - valor)
- Field (campo): identif. campo de un dato - tipo
- Value: dato relacionados con un id/campo

### Replica Set

- ≈ 2 o más máquinas (clust o cloud), corriendo su MongoDB Cluster
- Grupo de 3 q almacenan los datos
- Si algo pasa a alguna máquina, datos: ✓

### Configuración

- opción A: clustér repro + script.sh
- opción B: MongoDB Compose

## Shell

```
cls: Limpiar la consola
show dbs: Mostrar las bases de datos
db: Devuelve el nombre de la base de datos activa
use <nombre_db>: Cambiar a la base de datos <nombre_db>
```

## 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
- options
  - db.collection<name>.drop(): Elimina la colección



# Tema 4: NoSQL

## MongoDB

### Cursor Method

- Métodos q se aplican a un cursor  
(por ejemplo, devuelve un índice)

Ej:  
cursor()  
limit()  
skip()  
batch()

### limit (number)

- Número de elementos que se devuelven
- 0: no hay límite
- Ej:  
db.zips.find().limit(10)

### skip (offset)

- Número de elementos al principio q ignora
- Útil junto con limit() para hacer paginado
- Ej:  
db.zips.find().skip(10)

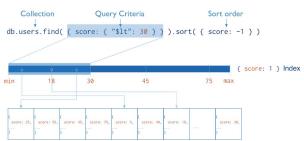
### Sort ( {field1: <value>, <field2: <value>}, ... )

- Ordenar los elementos
- Si no hay un campo único, el resultado puede ser inconsistente
- Maximo 32 fields.
- value:
  - 1 para orden ascendente
  - +1 para orden descendente
- Ej:  
db.zips.find().sort({ "pop": -1 }).limit(5)

## Índices A

### db.collection.createIndex()

- Crea un índice para facilitar las búsquedas
- Hay creado un índice por defecto para... -id



### Upsert

- db.<collection>.updateOne({query},{update}, {"upsert":true})
- Mezcla entre update e insert
  - Si 1º una coincidencia → se actualiza
  - Si 2º una coincidencia → se inserta

```
db.iot.updateOne({
  "sensor": r.sensor,
  "date": r.date,
  "valcount": { "$lt": 48 }
}, {
  "push": { "readings": { "v": r.value, "t": r.time } },
  "inc": { "valcount": 1, "total": r.value },
  { "upsert": true }
})

db.iot.updateOne({
  "sensor": "SENS0123987",
  "date": "2022-05-10",
  "valcount": { "$lt": 3 }
}, {
  "push": { "readings": { "v": 89, "t": ISODate() } },
  "inc": { "valcount": 1, "total": 89 },
  { "upsert": true }
})
```

Añadimos más comandos como \$exists, \$unwind, ...

### Modelado de datos

- Una forma de organizar los campos en un doc para mejorar el rendimiento de su aplicación y las características de persistencia
- Los datos se clasifican en la memoria en lo q se usan
- Los datos q se usan juntos deben almacenarse juntos

Ej: ¿Existen "room-type" existen?

- db.\_\_\_\_\_ aggregate ( { \$group : { \_id: "room-type" } } )
- db.\_\_\_\_\_ distinct ("room-type")

### Validación

#### Mongo DB permite usar JSON schema

- Se puede configurar al crear una colección
 

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

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

### BSON TYPES

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

### ObjectId

ObjectId("643e64d37bc71282283f4c4").getTimestamp()

## NODE.js

Driver oficial → MongoDB

Object modeling tool → Mongoose

various opciones para trabajar

### Conexión

1) Instalar driver: `npm install mongodb`

2) Connection String

Formato: `mongodb://[ user:pass@ ] host1[:port1] [, ... hostN[:portN]]`  
[ default authdb ] [ ?options ]

Ejemplo: `mongodb://127.0.0.1:27017`

```
const { MongoClient, ObjectId } = require("mongodb");
const url = "mongodb://127.0.0.1";
const client = new MongoClient(url);
const connectToDatabase = async () => {
  try {
    await client.connect();
  } catch (e) {
    console.error(e);
  }
}
const listDatabases = async (client) => {
  databasesList = await client.db().admin().listDatabases();
  console.log("databases:");
  databasesList.databases.forEach((db) => console.log(` - ${db.name}`));
}
const main = async () => {
  try {
    await connectToDatabase();
    await listDatabases(client);
  } catch (e) {
    console.error(e);
  } finally {
    client.close();
  }
}
main();
```

### Insert

`insertOne (document)`

`insertMany (documents - array)`

list DB



# TEMA 4: NO SQL

## NODE.js

### Índices

Acelera las búsquedas (-id por defecto)

Conviene usarlos en queries → 2 elementos en orden

```
const index = { _id: 1 };
```

```
client.db.collection().createIndex(index);
```

```
const MongoClient = require('mongodb');
const url = "mongodb://127.0.0.1:27017";
const client = new MongoClient(url);
const connectToDatabase = async () => {
  try {
    await client.connect();
  } catch (e) {
    console.error(e);
  }
}
const main = async () => {
  try {
    await connectToDatabase();
    const result = await movies.createIndex({ title: 1 });
    console.log(`Index created: ${result}`);
  } catch (err) {
    console.error(`Error finding documents: ${err}`);
  } finally {
    await client.close();
  }
}
main();
```

```
const MongoClient = require('mongodb');
const url = "mongodb://127.0.0.1:27017";
const client = new MongoClient(url);
const connectToDatabase = async () => {
  try {
    await client.connect();
  } catch (e) {
    console.error(e);
  }
};
const dbname = "sample_mflix";
const collection_name = "movies";
const movies = client.db(dbname).collection(collection_name);
const main = async () => {
  try {
    await connectToDatabase();
    const query = { title: /Mystique/ };
    const sort = { year: 1 };
    const projection = { _id: 0, title: 1, year: 1 };
    let result = await movies.find(query).sort(sort).project(projection);
    await result.forEach(doc => console.log(doc));
  } catch (err) {
    console.error(`Error finding documents: ${err}`);
  } finally {
    await client.close();
  }
}
main();
```

se aplica como los índices facilitan

### Transacciones

- + permite ejecutar una serie de op. en conjunto
- + se aplican todas o ninguna (Atomicidad)
- + cumplen ACID
- + Necesitamos una sesión común a todas las operaciones

```
const session = client.startSession();
const transactionResults = await session.withTransaction(async () => {
  // Las operaciones
});
```

### Pasos para realizar la transacción (ES6)

- ① Retirar la cantidad Cuenta Origen
- ② Añadir cantidad en la Cuenta destino
- ③ Añadir info a la Colección transfers
- ④ Añadir info de la transferencia a la Cuenta Origen
- ⑤ Añadir info de la transferencia a la Cuenta destino

```
const MongoClient = require('mongodb');
const url = "mongodb://127.0.0.1:27017";
const client = new MongoClient(url);
const dbname = "bank";
const accounts = client.db(dbname).collection("accounts");
const transfers = client.db(dbname).collection("transfers");
let account_id_sender = "MOB574189308";
let account_id_receiver = "MOB43652528";
let transfer_id = "TR21872187";
let transaction_amount = 100;
const session = client.startSession();
const main = async () => {
  try {
    const transactionResults = await session.withTransaction(async () => {
      // Continua en la siguiente página
    });
    if (transactionResults) {
      console.log("Transaction completed successfully.");
    } else {
      console.log("Transaction failed.");
    }
  } catch (err) {
    console.error(`Transaction aborted: ${err}`);
  }
  process.exit(1)
} finally {
  await session.endSession();
  await client.close();
}
main();
```

### Más:

- collections: reglas de ordenación de strings
  - GridFS: Almacenar archivos > 16MB
  - Cifrado
  - Compound Operations:
    - findOneAndDelete()
    - findOneAndUpdate()
    - findOneAndReplace()
- ...

### Transacción

```
const transactionResults = await session.withTransaction(async () => {
  const senderUpdate = await accounts.updateOne(
    { account_id: account_id_sender },
    { $inc: { balance: -transaction_amount } },
    { session }
  );
  const receiverUpdate = await accounts.updateOne(
    { account_id: account_id_receiver },
    { $inc: { balance: transaction_amount } },
    { session }
  );
  const transfer = {
    transfer_id: transfer_id,
    amount: transaction_amount,
    from_account: account_id_sender,
    to_account: account_id_receiver,
  };
  const insertTransferResults = await transfers.insertOne(transfer, { session });
  const updateSenderTransferResults = await accounts.updateOne(
    { account_id: account_id_sender },
    { $push: { transfers_complete: transfer.transfer_id } },
    { session }
  );
  const updateReceiverTransferResults = await accounts.updateOne(
    { account_id: account_id_receiver },
    { $push: { transfers_complete: transfer.transfer_id } },
    { session }
  );
});
```