Bases de datos de documentos


```
{
  "data": [
    {
      "id": "X999_Y999",
      "from": {
        "name": "Tom Brady", "id": "X12"
      },
      "message": "Looking forward to 2010!",
      "actions": [
        {
          "name": "Comment",
          "link": "http://www.facebook.com/X999/posts/Y999"
        },
        {
          "name": "Like",
          "link": "http://www.facebook.com/X999/posts/Y999"
        }
      ],
      "type": "status",
      "created_time": "2010-08-02T21:27:44+0000",
      "updated_time": "2010-08-02T21:27:44+0000"
    }]
}
```


Modelado y Procesamiento de Grandes Volumenes de Datos

CPAP, FING, UdelaR – 2020

Parte de material obtenido del curso BDNR

Una base de datos de documentos
es una base **no relacional**
que almacena datos como
documentos estructurados,
típicamente
XML y **JSON**

A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers

—Object-oriented data community, mid-1990s

An Object database is like a closet which requires that you hang up your suit with tie, underwear, belt, socks and shoes all attached.

—David Ensor, same period

Agenda

- Algo de historia
- JavaScript Object Notation (JSON)
- JSON Databases: MongoDB como ejemplo
- Aspectos a tener en cuenta en el diseño
- Operaciones CRUD
- Aggregation Pipeline
- Shardening

Algo de historia

XML: la vedette de los 2000s

Un lenguaje de marcado para representar datos.

XML 1.0 W3C recommendation en Febrero 1998.

¿Para qué se pensó y para que se usa/usó?

Intercambio de datos entre aplicaciones

Interoperabilidad

Separar datos de presentación en la web 2.0

Agregar metadatos y esquema (dar estructura).

Estándares asociados a XML

XPath

Una sintaxis para recuperar partes del documento: filtros, comodines.

XQuery

Un lenguaje de consulta sobre XML. El SQL de XML!

XML Schema

Un documento XML especial que describe la estructura de otro documento XML

XSLT (eXtensible Stylesheet Language Transformations)

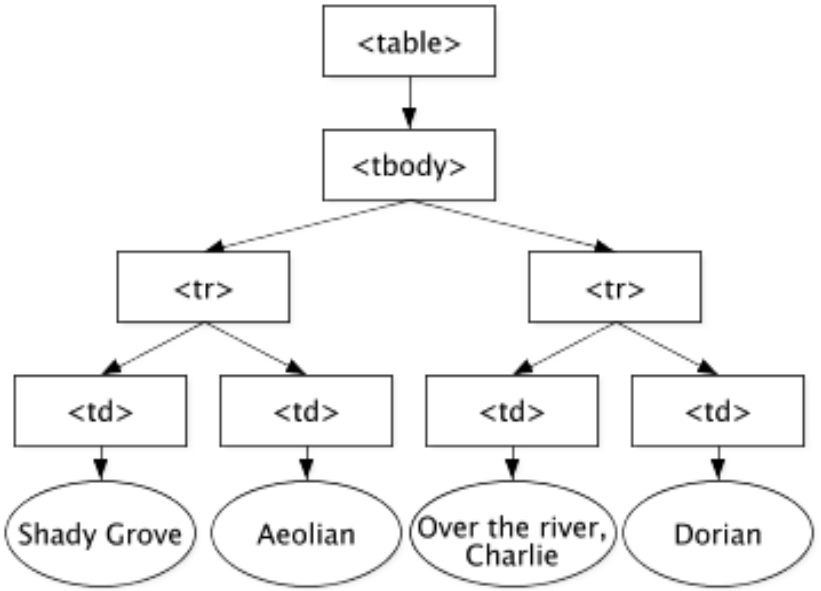
Un lenguaje de transformaciones. Permite generar otros formatos (ej: HTML)

DOM (Document Object Model)

Una API orientada a objetos para interactuar con documentos XML.

Estándares asociados a XML (DOM)

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Sobre el río, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

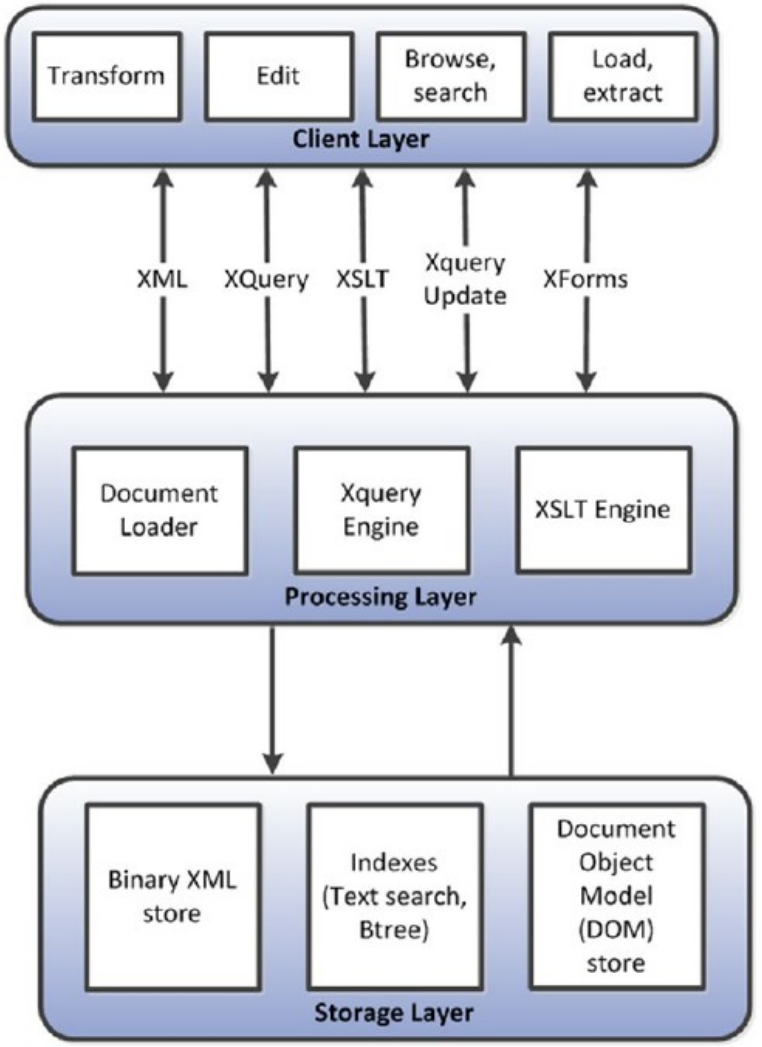


W3C: https://www.w3.org/2005/03/DOM3Core-es/introduccion.html

Y como muchas aplicaciones generaban XML ...

Surgen las BDs de documentos para gestionar colecciones de documentos XML.

Muchas soluciones comerciales.



Bases de datos nativas XML

BaseX, eXist, MarkLogic Server, Sedna, 28msec, MonetDB

- No se utilizan tablas, habitualmente se llaman containers o collections. Cada container puede contener una gran cantidad de XMLs (> 1MM) que tienen relación semántica entre ellos
- Xquery es el lenguaje de consultas más utilizado: eficiente, más dífcil de aprender que SQL
- Han caído en desuso en los últimos años, pero siguen vigentes

PostreSQL como base de datos híbrida

<state_id>187</state_id><comments>Mainly D2 drought conditions persisted through January and into February. D3 drought conditions were present across portions of Henry and eastern Dale counties.</comments>

```
SELECT xmlforest(state.name AS
state,comments)
FROM storm_info
JOIN state ON
state.id=storm_info.state_id
```

<state>ALABAMA</state><comments>Mainly D2 drought conditions persisted through January and into February. D3 drought conditions were present across portions of Henry and eastern Dale counties.</comments>

Caso de Uso

- Una planta industrial genera datos producto de la lectura de las temperatura de sus reactores químicos.

pero XML tiene algunos inconvenientes...

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup
                Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
            languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

"Some languages can be read by humans, but not by machines,

while others can be read by machines but not by humans.

XML solves this problem by being readable to neither."

Post by deleted, Reddit [1]

[1] https://www.reddit.com/r/ProgrammerHumor/comments/1mbgih/xml/

Mientras tanto, en el mundo de los desarrolladores de aplicaciones web ...

Javascript se impone como lenguaje de programación tanto del lado del cliente como del servidor (Node.js).

AJAX: mensajería asíncrona entre cliente y servidor (originalmente pensado sobre XML).

Aparece **Javascript Object Notation (JSON).**

JSON: The Fat-Free alternative to XML [1]

[1]http://www.json.org/xml.html

key

value

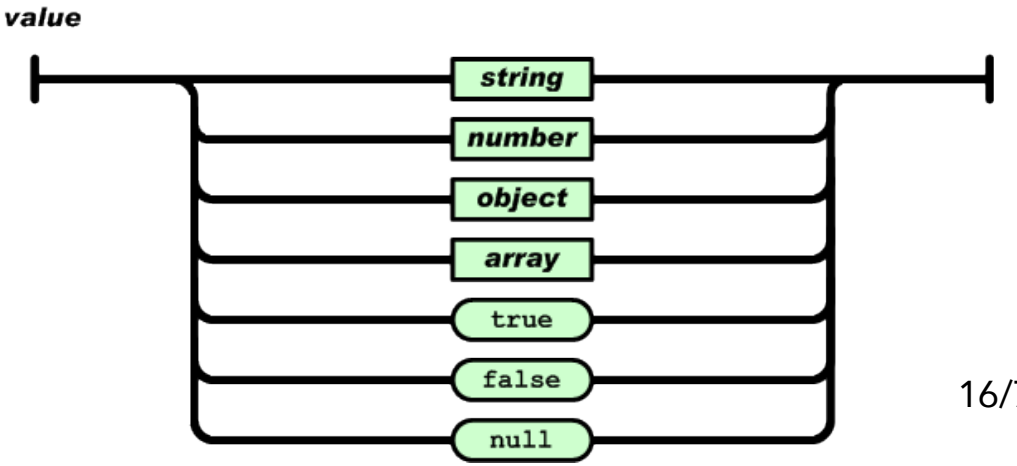
object

```
{ "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard Generalized Markup
                        Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language, used to
                     create markup languages such as DocBook.",
            "GlossSeeAlso": ["GML", "XML"]
          },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```

array

Las *key* son *strings*.

Los *values* pueden ser cualquier elemento de la notación.



XML vs JSON

```xml
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup
                Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
            languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

```json
{ "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard Generalized Markup
                        Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language, used to
                     create markup languages such as
                     DocBook.",
            "GlossSeeAlso": ["GML", "XML"]
           },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```

JSON es **menos verboso**: más liviano para el intercambio de datos.

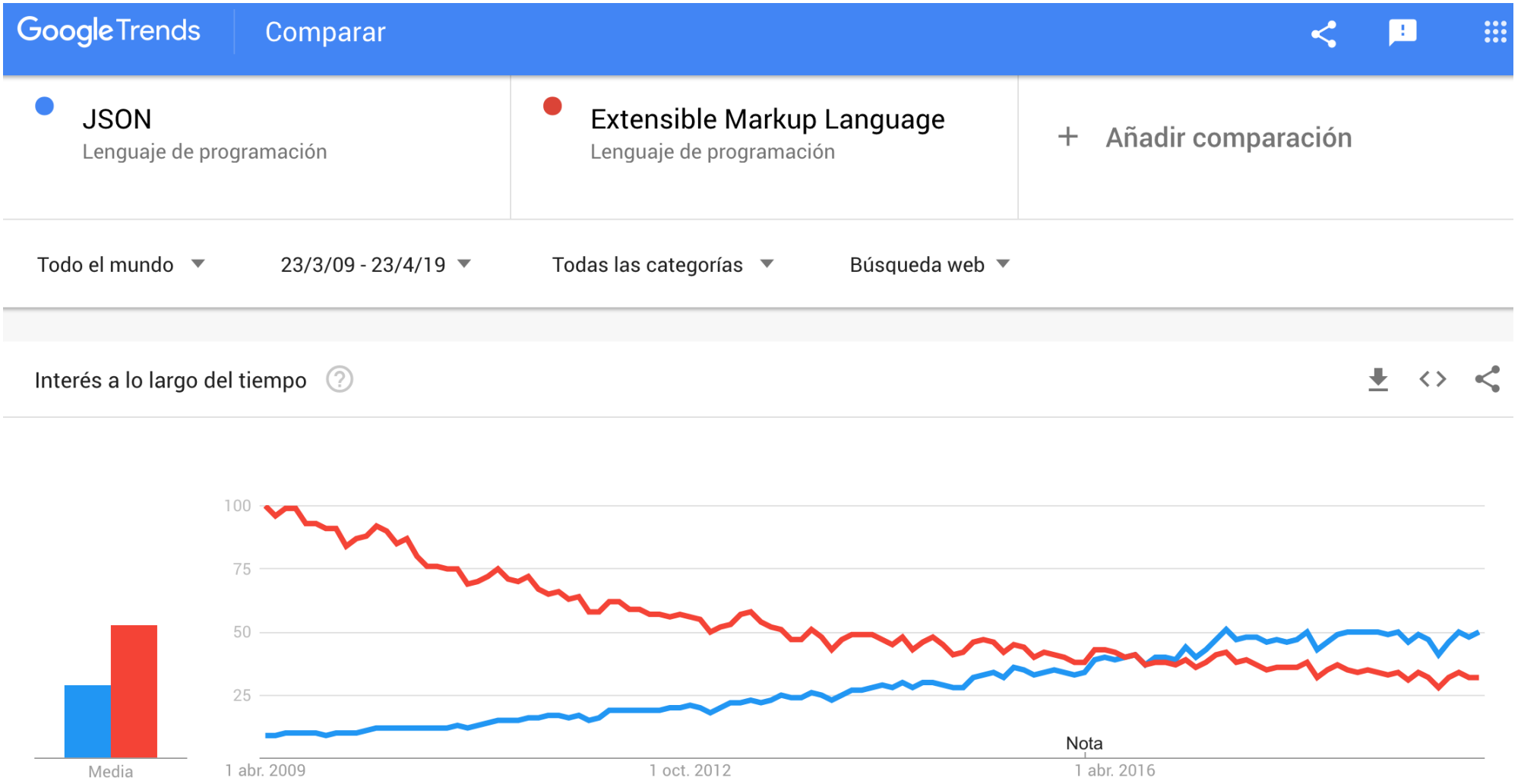
JSON representa **directamente** el objeto: más sencillo para el desarrollador

Si desarrollo OO y almaceno JSON **no tengo *impedance mismatch***.

Si desarrollo en Javascript, intercambio JSON y almaceno JSON tengo un ***full stack* Javascript**.

No tengo un esquema fijo, pero tengo **algo de estructura**.

XML vs JSON (2)


Google Trends
Comparar
JSON
Lenguaje de programación
Extensible Markup Language
Lenguaje de programación
+ Añadir comparación
Todo el mundo
23/3/09 - 23/4/19
Todas las categorías
Búsqueda web
Interés a lo largo del tiempo
100
75
50
25
Nota
Media
1 abr. 2009
1 oct. 2012
1 abr. 2016

JSON Databases: MongoDB como ejemplo

JSON Databases

- A diferencia de otras bases de datos No Relacionales, éstas no tienen un *manifesto*
- El único requisito: almacenar JSON

RDBMS

Tabla	Colección
Fila	Documento
Columna	Campo (*field*)
SQL Join	Documentos embebidos y referencias
SQL Group-by (agregación)	*Aggregation pipeline*

Los documentos dentro de una colección pueden tener campos diferentes (diferente esquema).

Se puede exigir la validación de documentos (a partir versión 3.2)

La ausencia de un esquema fijo (schema-less) no quiere decir que no exista una etapa de diseño y modelado

Siempre debemos partir de nuestro modelado conceptual

Schemaless = Schema Anarchy

Reducción de mismatch entre modelo de datos y código de aplicación

Evita **complejas interdepencias** entre equipos de ingenieros

Adaptación **dinámica a cambios** de la realidad

```
{
  "imdbID":"tt0944947",
  "Type":"series",
  "Title":"Game of Thrones",
  "Genre":"Adventure, Drama, Fantasy",
  "Actors":"Peter Dinklage, Lena Headey, Emilia Clarke, Kit Harington",
  "imdbRating":"9.5",
  "imdbVotes":"1,031,056"
}
{
  "imdbID":"tt1877832",
  "Type":"movie"
  "Title":"X-Men: Days of Future Past",
  "Year":"2014",
  "Genre":"Action, Adventure, Fantasy",
  "Actors":"Hugh Jackman, Michael Fassbender, Jennifer Lawrence, Peter
Dinklage",
  "imdbRating":"8.0",
  "imdbVotes":"514,203"
}
```

¿cómo representar objetos relacionados?

Estrategia 1: documentos embebidos

```
{
  "imdbID":"tt0944947",
  "Type":"series",
  "Title":"Game of Thrones",
  "Genre":"Adventure, Drama, Fantasy",
  "Actors":
       [
          { “imdbID”: “nm0227759” , “name”: "Peter Dinklage”},
          { “imdbID”: “nm0372176” , “name”: "Lena Headey”},
          { “imdbID”: “nm3229685” , “name”: "Kit Harington"}
       ]
  "imdbRating":"9.5",
  "imdbVotes":"1,031,056"
}
{
  "imdbID":"tt1877832",
  "Type":"movie"
  "Title":"X-Men: Days of Future Past",
  "Year":"2014",
  "Genre":"Action, Adventure, Fantasy",
  "Actors":
       [
          { “imdbID”: “nm0413168” , “name”: "Hugh Jackman”},
          { “imdbID”: “nm1055413”, “name”: "Michael Fassbender”},
          { “imdbID”: “nm2225369”, “name”: "Jennifer Lawrence”},
          { “imdbID”: “nm0227759” , “name”: "Peter Dinklage”}
       ]
  "imdbRating":"8.0",
  "imdbVotes":"514,203"
}
```

Evito los *joins*, pero tengo redundancia

¿Cómo busco todas las películas Peter Dinklage?

¿Qué pasa si tengo que modificar los datos de un actor?

Estrategia 2: documentos relacionados

```
{
  "imdbID":"tt0944947",
  "Type":"series",
  "Title":"Game of Thrones",
  "Genre":"Adventure, Drama, Fantasy",
  "Actors":
       ["nm0227759", "nm0372176", "nm3229685" ]
  "imdbRating":"9.5",
  "imdbVotes":"1,031,056"
}
{
  "imdbID":"tt1877832",
  "Type":"movie"
  "Title":"X-Men: Days of Future Past",
  "Year":"2014",
  "Genre":"Action, Adventure, Fantasy",
  "Actors":
       [ "nm0413168" ,"nm1055413","nm2225369","nm0227759"]
  "imdbRating":"8.0",
  "imdbVotes":"514,203"
}
...
{ "id":"nm0227759",
  "name":"Peter Dinklage",
  "description":"Actor, Battle of the Bastards"
}
```

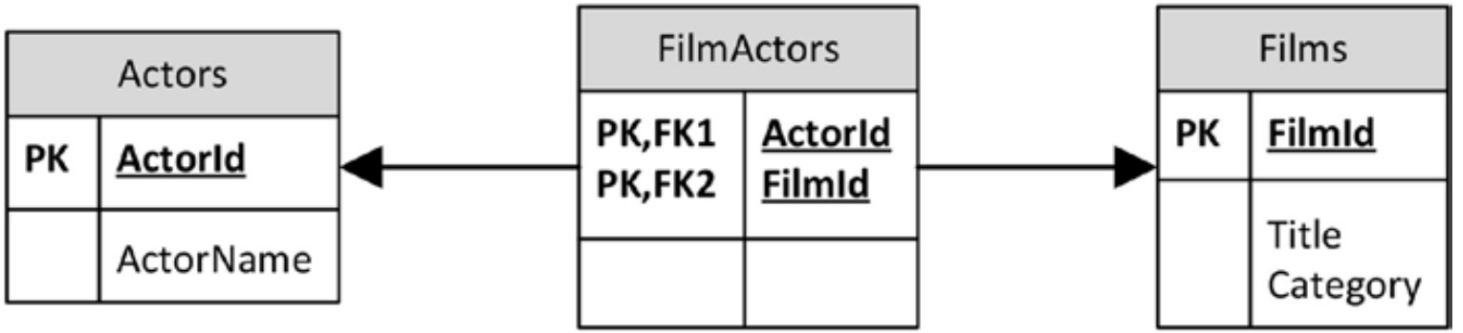
Evito redundancia ¿pero?

PRIMER ROUND

Relacional vs Documental

- Teóricamente, es posible diseñar un base documental como una relacional ***en tercera forma normal***
  - ***Siempre se busca eliminar redundancia***
- ***Secreto de las bases de datos relacionales***
  - ***Cuando precisamos mejorar la performance, solemos desnormalizar el esquema lógico***

Patrones de diseño – Documentos Embebidos




```
{ "_id" : 97, "Title" : "BRIDE INTRIGUE",
              "Category" : "Action",
    "Actors" :
      [ { "actorId" : 65, "Name" : "ANGELA HUDSON" } ]
}

{ "_id": 115,"Title": "CAMPUS REMEMBER",
             "Category": "Action",
    "Actors" :
        [
          { "actorId": 8,"Name": "MATTHEW JOHANSSON" },
          { "actorId": 45,"Name": "REESE KILMER" },
          { "actorId": 168,"Name": "WILL WILSON" }
        ]
}

{ "_id" : 105, "Title" : "BULL SHAWSHANK",
               "Category" : "Action",
     "Actors" :
       [ { "actorId" : 2, "Name" : "NICK WAHLBERG" },
         { "actorId" : 23, "Name" : "SANDRA KILMER" } ]
}
```

Film Document

Actor document

Array of actors

Collection

Patrones de diseño – Documentos Embebidos

- Permite obtener documentos con una sola operación
  - *Data locally*
- No requiere hacer join por medio del código de la aplicación u otras operaciones del lenguaje de consulta (ej MongoDB: lookup)
- Evita la violación de *restricciones de integridad*
  - Ej: Cambiar el nombre de un actor o elimarlo
  - Queda a merced de la aplicación y sus desarrolladores ¿Volvemos a la década d de los 60 con esta problemática?

Patrones de diseño – Documentos Referenciados

- Diseño lógico “más” normalizado
- Las restricciones de integridad deben de ser mantenidas desde la aplicación
- Mayor flexibilidad en aplicaciones donde es poco lo que podemos anticipar en las tempranas etapas de diseño

Patrones de diseño – Algunos criterios

Documentos embebidos

- El patrón de consultas de la aplicación es conocido en la etapa de diseño
- Los datos son accedidos de una manera homogénea

Documentos referenciados

- La aplicación consulta los datos de diferentes formas
- No es posible anticipar el patrón de consultas de la aplicación en las etapas de diseño
- Alto número de relaciones 1:N

Patrones de diseño – Algunos criterios

Alto número de relaciones 1:N

- Pensemos en un blog, donde cada entrada puede tener cuentos o miles de comentarios de usuarios
  - Documentos grandes, se quiere mayor cantidad de RAM
  - Documentos que crecen a gran velocidad, deben ser mantenidos con mayor frecuencia (copiados a espacios más grandes)
  - Las bases documentales tienen límites para los tamaños de documentos (16MB en MongoDB)

Aspectos a tener en cuenta en el diseño

- No hay recetas como con las bases relacionales (formas normales)
- El diseño depende más del tipo de operaciones que voy a hacer que de la naturaleza de los datos
  - ¿qué quiero devolver?
  - ¿qué quiero actualizar?
  - ¿qué operaciones quiero priorizar?

Aspectos a tener en cuenta en el diseño (ii)

- Representación de las relaciones (embebidas vs referencias)
- Atomicidad de las operaciones (a nivel de documento)
- Tamaño de los documentos
- Tamaño y cantidad de documentos en las colecciones
- Operaciones a realizar
- Particionamiento (sharding)

¿Por qué y cuándo usar bases de datos de documentos?

SEGUNDO ROUND

Consultas en MongoDB

Operaciones CRUD

- Read

```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

```
SELECT name, address
FROM users
WHERE
      age > 18
LIMIT 5
```

Operaciones CRUD

- Al igual que las bases relacionales, MongoDB tiene un conjunto de operaciones DML CRUD (Create, Read, Update, Delete)

- La entidad atómica es el documento

Operaciones CRUD

- Create

INSERT INTO **USERS** (name, age, status)
VALUES (“sue”, 26, “pending”)

Operaciones CRUD

- Read

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

```
SELECT name, address
FROM users
WHERE
      age > 18
LIMIT 5
```

Operaciones CRUD

- Update

UPDATE users
SET status = “rejected”
WHERE
age > 18

Operaciones CRUD

- Delete

```
db.users.deleteMany(
   { status: "reject" }
)
```

collection

delete filter

```
DELETE FROM users
WHERE
   status = “rejected”
```

Aggregation Pipeline

- Las operaciones de agregación toman "registros" y retornan resultados procesados

- Agrupan valores de múltiples documentos, permitiendo aplicar operaciones sobre los datos agrupados, retornando un único resultado

- MongoDB provee de un Aggregation Framework

Aggregation Pipeline

- Consiste de varias etapas (stages)

- Cada etapa transforma el documento a medida que pasa a través del pipeline

- Las stages contienen expresiones. Las expresiones especifican la transformación que aplica al documento de entrada, no pudiendo hacer referencia a datos de otros documentos.

Aggregation Pipeline

- Cada stage puede aparecer varias veces en el mismo pipeline (a excepción de $out, $merge y $geoNear)

- Generalmente las expresiones son *stateless*

- Los acumuladores, usados en la stage de **$group**, mantienen su estado, en la medida que los documentos avanzan en el pipeline.

Aggregation Pipeline

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Stage 1: El stage $match filtra los documentos por el campo field, pasando al siguiente stage los documentos con status igual a “A”

Stage 2: El stage $group agrupa los documentos por el campo cust_id, para calcular la suma para cada grupo cust_id

Aggregation Pipeline

https://docs.mongodb.com/manual/_images/agg-pipeline.mp4

Caso de Uso

- Se reciben lecturas cada un minuto
- Cada lectura contiene:
  - Número de sensor
  - Temperatura
  - Timestamp
- ¿Cuál es la mejor estrategia de diseño?

Caso de Uso

- Se reciben lecturas cada un minuto
- Cada lectura contiene:
  - Número de sensor
  - Temperatura
  - Timestamp
- ¿Cuál es la mejor estrategia de diseño?

Caso de Uso

```
{
   sensor_id: 12345,
   timestamp: ISODate("2019-01-31T10:00:00.000Z"),
   temperature: 40
}

{
   sensor_id: 12345,
   timestamp: ISODate("2019-01-31T10:01:00.000Z"),
   temperature: 40
}

{
   sensor_id: 12345,
   timestamp: ISODate("2019-01-31T10:02:00.000Z"),
   temperature: 41
}
```

Caso de Uso

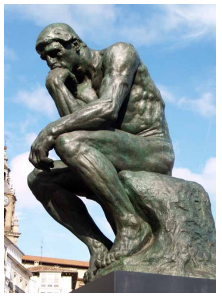
Bucket pattern

```
{
    sensor_id: 12345,
    start_date: ISODate("2019-01-31T10:00:00.000Z"),
    end_date: ISODate("2019-01-31T10:59:59.000Z"),
    measurements: [
       {
       timestamp: ISODate("2019-01-31T10:00:00.000Z"),
       temperature: 40
       },
       {
       timestamp: ISODate("2019-01-31T10:01:00.000Z"),
       temperature: 40
       },
       …
       {
       timestamp: ISODate("2019-01-31T10:42:00.000Z"),
       temperature: 42
       }
    ],
   transaction_count: 42,
   sum_temperature: 2413
}
```

Caso de Uso

Bucket pattern

- Con el valor calculado de **sum_temperature** es fácil obtener un bucket y calcular rápidamente el promedio de temperatura (sum_temperature / transaction_count)
- Cuando se trabaja con series temporales, puede ser importante conocer la temperatura promedio desde las 2:00 a las 3:00 pm en California, el 18 de Julio de 2020. Con bucketing y pre-aggregation no es complejo proveer de esta información.



- **Para reflexionar:** Cuando se aborda el diseño lógico de una base de datos documental ¿Qué diferencias observa en comparación con el modelo relacional?

Caso de Uso

Bucket pattern

Sample Use Case

One example of making time-series data valuable in the real world comes from an IoT implementation by Bosch. They are using MongoDB and time-series data in an automotive field data app. The app captures data from a variety of sensors throughout the vehicle allowing for improved diagnostics of the vehicle itself and component performance.

Other examples include major banks that have incorporated this pattern in financial applications to group transactions together.

https://www.mongodb.com/blog/post/building-with-patterns-the-bucket-pattern

Caso de Uso II

- Se tiene una web de venta de libros
- La lista de compradores , en la mayoría de los casos, puede almacenarse dentro del propio documento del libro
- En algunos casos, pueden existir libros con miles de compradores (best sellers)

Caso de Uso II – Outlier Pattern

```
{
    "_id": ObjectID("507f1f77bcf86cd799439011")
    "title": "A Genealogical Record of a Line of Alger",
    "author": "Ken W. Alger",
    …,
    "customers_purchased": ["user00", "user01", "user02"]

}
```

```
{
    "_id": ObjectID("507f191e810c19729de860ea"),
    "title": "Harry Potter, the Next Chapter",
    "author": "J.K. Rowling",
    …,
   "customers_purchased": ["user00", "user01", "user02", …, "user999"],
   "has_extras": "true"
}
```

Podemos fijar un límite de mil compradores dentro del array. Si aumenta, podemos crear un documento extra_customers, y consultarlo en caso de que has_extras = true

Consistencia

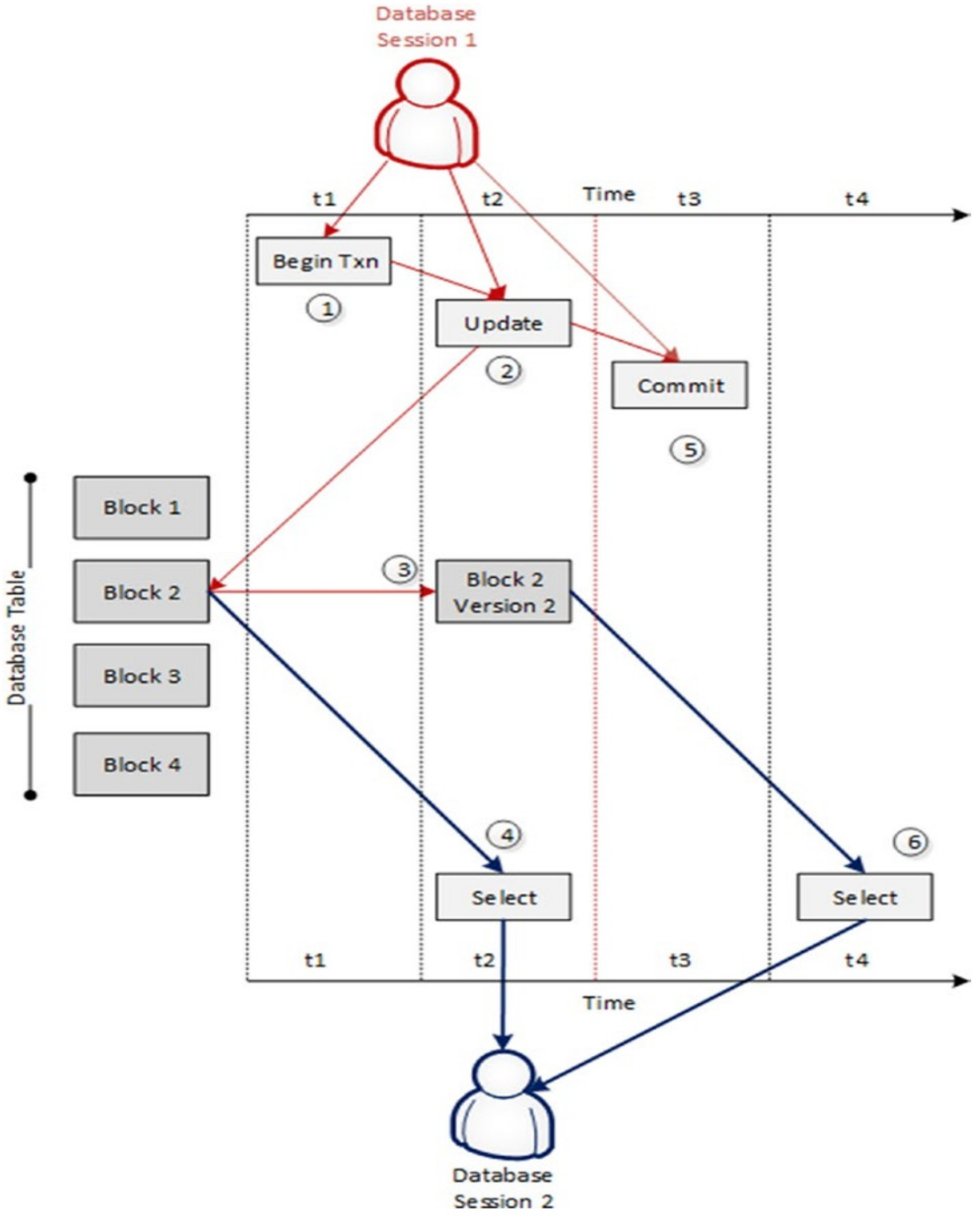
- MongoDB implementa *locks*
  - *Cambia la granularidad en versiones, a partir de 3.0 a nivel de documento*
- Consistencia estricta en modo single node
- Consistencia eventual en *replica sets*

Consistencia

- Repasando ACID
  - **A**tomic: Transacción indivisible. Se aplica todo o nada.
  - **C**onsistent: La base de datos se mantiene en un estado consistente
  - **I**solated: Cuando se ejecutan varias transacciones en parelelo, una transacción no se ve afectada por las otras
  - **D**urable: Una vez que la transacción termina, los cambios se deben persistir en la base de datos, sin importar si hay fallas de software o hardware.

Consistencia

- La forma más simple (y usada) de implementar ACID es por medio de *locks* (bloqueos)
- Una forma de implementar ACID es por medio de MVCC: multi-version concurrency control
- En este modelo, copias de los datos se mantienen con timestamps o números de secuencia, lo que permite tener diferentes instancias de la base de datos visibles para las diferentes transacciones en ejecución


Database
Session 1
t1
t2
Time
t3
t4
Begin Txn
1
Update
2
Commit
5
Block 1
Block 2
Block 3
Block 4
Database Table
3
Block 2
Version 2
4
Select
6
Select
t1
t2
t3
t4
Time
Database
Session 2

Consistencia

- Algunos niveles de consistencia
  - **Estricto:** Una lectura siempre retorna el valor más reciente
  - **Casual:** Las lecturas pueden no retornar los últimos valores, pero nunca retorarán valores fuera de secuencia

    Ej: Transacción 1 escribe valores A, B y C. La Transaccion 2 nunca verá el valor C si antes no vio el valor B.
  - **Monótona:** Una transacción nunca verá datos "revertidos" por otra transacción.

Consistencia

- Una forma de reducir los niveles de consistencia es limitar su alcance a una sola operación u objeto
- En una base de datos relacional es posible mantener la consistencia entre varias operaciones.
  - Por ejemplo: es posible eliminar un registro en una tabla e insertar un registro en otra tabla como una operación atómica.
- Prácticamente ninguna base de datos no relacional implementa este nivel de consistencia

Consistencia

- Niveles de consistencia
  - **Lee tus propias escrituras:** se garantiza que la transacción, al menos, tenga visibilidad sobre sus propias operaciones.
  - **Eventual:** El sistema puede ser inconsistente en cualquier momento, pero todas las operaciones individuales se aplicarán de forma consistente. Si todas las actualizaciones se detienen, la base de datos eventualmente alcanzará un estado consistente
  - **Débil:** No se garantiza que la base de datos nunca se vuelva inconsistente. Por ejemplo, si un servidor falla, se puede perder una actualización.

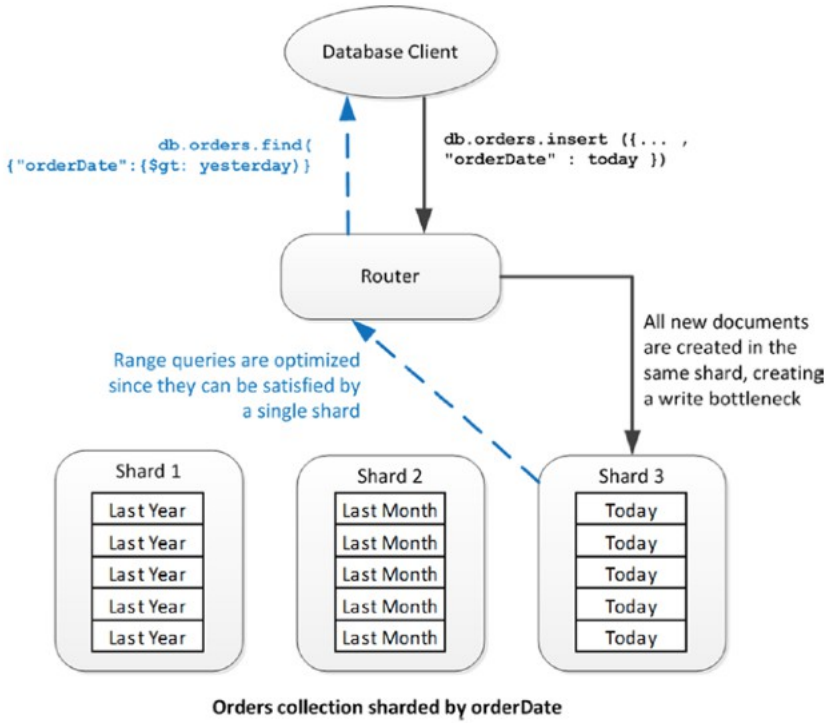
Consistencia

- En la práctica, los sistemas no relacionales implementan consistencia estricta o eventual
- Los sistemas relacionales ofrecen consistencia ACID (estricta)
- MongoDB:
  - En instalaciones de **un solo nodo**, por omisión, ofrece consistencia **estricta**
  - En instalaciones de varios nodos, implementa consistencia eventual

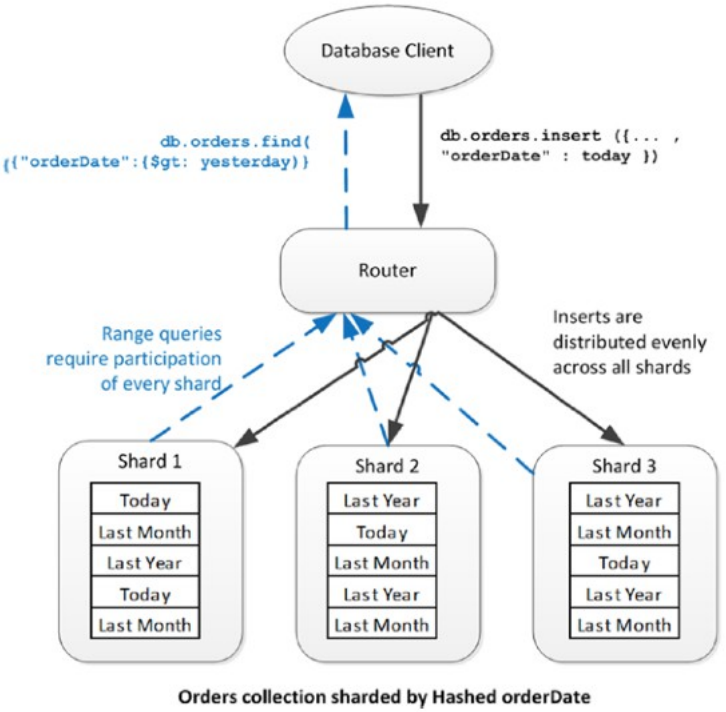
Consistencia

- MongoDB:
  - La granularidad de los locks ha evolucionado en diferentes versiones:
    - Primeras versiones: a nivel de base de datos
    - 2.0: colección
    - 3.0: documento
    - 4.0: multi-documento
    - 4.2: Transacciones distribuidas, soporte multi-documento en múltiples nodos
  - No subestimar la granuralidad a nivel de documento
    - Documentos embebidos

Sharding: ejemplos



Orders collection sharded by orderDate



Orders collection sharded by Hashed orderDate

Sharding

- MongoDB soporta al menos dos tipos de sharding:
  - Por rangos de clave (key-based):
    - Más eficiente la lectura de rangos
  - Por hash sobre la clave (hash-based)
    - Mejora la distribución en los inserts

Resumiendo

- Las bases de datos de documentos se inspiran en las ideas de las bases OO
- El éxito de las bases documentales JSON se basa fundamentalmente en su alto nivel de adopción por los desarrolladores.
  - Brindan más estructura que las key-value stores.
  - Son muy útiles cuando no existe un esquema fijo (eg: etapas tempranas de desarrollo, esquemas variados)

Material adicional (i)

- Web Seminars de MongoDB
  - Están los links en EVA del curso
- Documentación en el sitio de MongoDB
  - https://docs.mongodb.com/

Cursos online en el sitio de MongoDB

- https://university.mongodb.com/courses/catalog

Material adicional (ii)

- Diseño de base de datos documentales
  - Documentación en el sitio de MongoDB
    - https://docs.mongodb.com/manual/data-modeling/
  - Consejos de diseño
    - https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1
    - https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-2
    - https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-3