Bases de Datos

Clase 15: NoSQL, teorema CAP, JSON

Hasta ahora

- Bases de datos relacionales
- SQL

NoSQL

Término común para denominar bases de datos con:

- Menos restricciones que el modelo relacional
- Menos esquema
- Más distribución

NoSQL: ¿Por qué?

Sistemas de bases de datos relacionarles no están pensadas para un entorno altamente distribuido.

- WWW, google, instagram, etc

Teorema CAP

Plantea que para una base de datos distribuida es imposible mantener simultáneamente estas tres características:

- Consistency
- Availability
- Partition Tolerance

Teorema CAP

Plantea que para una base de datos distribuida es imposible mantener simultáneamente estas tres características:

- Consistency (todos los usuarios ven lo mismo siempre)
- Availability (toda consulta siempre recibe una respuesta, aunque sea error)
- Partition Tolerance (el sistema opera bien pese a estar físicamente dividido)

Teorema CAP

Osea si queremos que el sistema opere bien de forma distribuida, hay que abandonar al menos una entre:

- Consistency (todos los usuarios ven lo mismo siempre)
- Availability (toda consulta siempre recibe una respuesta, aunque sea error)

Teorema CAP

Osea si queremos que el sistema opere bien de forma distribuida, hay que abandonar al menos una entre:

- Consistency (todos los usuarios ven lo mismo siempre)
- Availability (toda consulta siempre recibe una respuesta, aunque sea error)

¡Pero los DBMS relacionales están diseñados para eso!

NoSQL

Término común para denominar bases de datos con:

- Menos restricciones que el modelo relacional
- Menos esquema
- Más distribución

La falta de restricciones permite botar consistencia

Ojo: NoSQL para muchos hoy es

Cualquier cosa que no sea una BD relacional, incluso si prioriza Avaliablity y Consistency

Ojo: NoSQL para muchos hoy es

Cualquier cosa que no sea una BD relacional, incluso si prioriza Avaliablity y Consistency

- BD key-value
- BD grafos
- BD documentos

BD Key - Value

- Son grandes tablas de hash persistentes
- Esta categoría es difusa, pues muchas de las aplicaciones de otros tipos de BD usan key - value y hashing hasta cierto punto

BD de Grafos

Especializadas para guardar relaciones

- En general, almacenan sus datos como property graphs
- Algunos ejemplos son Neo4J, Virtuoso, Jena, Blazegraph

BD Orientadas a Documentos

Especializadas en documentos

- CouchDB, MongoDB (estas y otras BD almacenan sus datos en documentos JSON)
- JSON no es el único estándar de documentos (por ejemplo, existe también XML)

JSON

Su nombre viene de JavaScript Object Notation

Estándar de intercambio de datos semiestructurados / datos en la Web

- JSON se acopla muy bien a los lenguajes de programación

JSON

Ejemplo


```
{
  "statuses": [
    {
      "id": 72545937362390630 4,
      "text": "@visitlondon: Have you been to any of these
               quirky London museums? https://t.co/tnrar8UttZ",
      "retweeted_status": {
        "metadata": {
          "result_type": "recent",
          "iso_language_code": "en"
        },
        "retweet_count": 239,
        "retweeted": false
      }
    }
  ]
}
```

JSON

La base son los pares key - value

```
{
  “nombre": “Matías”
}
```

Valores pueden ser:

- Números
- Strings (entre comillas)
- Valores booleanos
- Arreglos (por definir)
- Objetos (por definir)
- `null`

JSON

Sintaxis

Los objetos se escriben entre `{}` y contienen una cantidad arbitraria de pares key - value

```
{
  “nombre": “Matías”, “apellido": “Jünemann”
}
```

JSON

Sintaxis

Los arreglos se escriben entre `[]` y contienen valores

```
{
  “profesores”: [
    {“nombre”: “Juan”, “apellido": “Reutter”},
    {“nombre”: “Cristian”, “apellido": “Riveros”},
    {“nombre”: “Marcelo”, “apellido": “Arenas”}
  ]
}
```

JSON vs SQL

SQL:

- Esquema de datos
- Lenguajes de consulta independientes del código

JSON:

- Más flexible, no hay que respetar necesariamente un esquema
- Más tipos de datos (como arreglos)
- Human - Readable

JSON

Lenguaje de consultas

Hay intentos de lenguajes de consulta para objetos JSON que usen su estructura de árbol:

- Por ejemplo, JSONPath

Importante: JSON está ahí para los programadores que NO buscan separar datos del código

JSON

Lenguaje de consultas

¿Por qué necesitamos esquemas para JSON?

- JSON Schema: propuesta toma fuerza el 2013 - 2014
- Harta investigación en el DataLab UC

JSON Schema

```
{
  “first_name”: “Alexis”,
  “last_name”: “Sánchez”,
  “age”: 28,
  “club”: {
    “name”: “Arsenal FC”,
    “founded”: 1886,
  }
  “first_club”: “Cobreloa”,
  “va_al_mundial”: false
}
```

JSON Schema

```
{
    "type": "object",
    "properties": {
        "first_name": { "type": "string" },
        "last_name": { "type": "string" },
        "age": { "type": "integer" },
        "club": {
            "type": "object",
            "properties": {
                "name": { "type": "string" },
                "founded": { "type": "integer" }
            },
            "required": ["name"]
        }
        “first_club": { "type": "string" },
        “va_al_mundial”: { "type": “boolean" },
    },
    "required": ["first_name", "last_name", "age", "club"]
}
```

BD de documentos

Especializadas en documentos: almacenan muchos documentos JSON

- Si quiero libros: un documento JSON por libro
- Si quiero personas: un documento JSON por persona

Notar que esto es altamente jerárquico

BD de documentos

Qué hacen bien:

- Si quiero un libro o persona en particular
- Cruce de información **simple**

Muy útiles a la hora de desplegar información en la web

BD de documentos

Pueden verse como un cache de una BD relacional
¿Por qué?

BD de documentos

Si quiero cruzar información:

- Documentos de alumnos
- Documentos de ramos

Muy fácil:

- Todos los alumnos que toman un ramo ¿Cómo lo hago?
- Los ramos con más alumnos ¿Cómo lo hago?

BD de documentos

Si quiero cruzar información:

- Documentos de alumnos
- Documentos de ramos

No tan fácil:

- Los ramos con más alumnos en ingeniería ¿Cómo lo hago?
- Si el join es complejo, la estructura jerárquica es un impedimento

Teorema CAP

Plantea que para una base de datos distribuida es imposible mantener simultáneamente las tres características:

- Consistency
- Availability
- Partition Tolerance

BD Documentos vs Teorema CAP

- Distintas aplicaciones en una misma base de datos acceden a distintos documentos al mismo tiempo
- En general diseñadas para montar varias instancias que (en teoría) tienen la misma información
- Propagan updates en forma descoordinada

Proveen "**Consistencia Eventual**"

Consistencia Eventual

La consistencia eventual puede generar problemas

Si dos aplicaciones intentan acceder al mismo documento en MongoDB, estas pueden ser versiones diferentes del documento

Map Reduce

- Algoritmo eficiente para computación paralela
- Paradigma de programación - mezcla de datos con controlador
- Sacrificios para acelerar computación

Computación Paralela

- Datos tan grandes que no caben en un computador
- Repartidos en muchos servidores
- Cada servidor no conoce lo que tiene el resto
- No podemos dar el lujo de comunicar todos los datos

Map Reduce

- Map: recibe datos y genera pares key - values
- Reduce: recibe pares con el mismo key y los agrega

Map Reduce

Ejemplo: ver palabra más utilizado en un texto T

Map Reduce

Ejemplo

Para contar palabras de un texto

Map Reduce

Ejemplo

Map:

- Recibe un pedazo de texto
- Por cada palabra, emite el par `(palabra, número de ocurrencias)`

Reduce:

- Cada reduce recibe todos los pares asociados a la misma palabra
- Junta todos estos pares y suma las ocurrencias

Map Reduce

Arquitectura

Mappers:

- Nodos encargados de hacer Map
- Reciben parte del documento y lo envían a los reducers

Reducers:

- Nodos encargados de hacer Reduce
- Reciben los Map y los agregan
- El output es la unión de cada Reducer

Map Reduce

No es un descubrimiento nuevo, pero recientemente se ha visto calzar perfectamente con las necesidades de las grandes BD

Es la arquitectura más importante en sistemas que reciben grandes bases de datos

- Hadoop: La implementación de Google de Map - Reduce, presente en muchos sistemas con computación distribuida

Map Reduce

Ejemplo: Join

¿Cómo hago un join entre **R(A,B)** y **S(B,C)** con Map Reduce?

- Modelo: un archivo con el nombre de la tabla y sus tuplas
- En cada nodo llegan tuplas de R y tuplas de S
- La salida final debe tener el join $R \bowtie_{R.B=S.B} S$

Map Reduce

Ejemplo: Join

- Map: uso el atributo de join como key
    - Input (a,b) en **R**: genero par b:(a,R)
    - Input (b,c) en **S**: genero tupla b:(c,S)

Map Reduce

Ejemplo: Join

- Map: uso el atributo de join como key
  - Input (a,b) en **R**: genero par b:(a,R)
  - Input (b,c) en **S**: genero tupla b:(c,S)

- Reduce: Recibe pares con el mismo atributo B, que vienen de **R** o de **S**. Hago el producto cruz de todos los atributos A que tengo de R y todos los atributos C de S.

Map Reduce

Ejemplo: Join

- Map: uso el atributo de join como key
  - Input (a,b) en **R**: genero par b:(a,R)
  - Input (b,c) en **S**: genero tupla b:(c,S)

- Reduce: Recibe pares con el mismo atributo B, que vienen de **R** o de **S**. Hago el producto cruz de todos los atributos A que tengo de R y todos los atributos C de S.

Map Reduce

Ejemplo: Join

- Map: uso el atributo de join como key
  - Input (a,b) en **R**: genero par b:(a,R)
  - Input (b,c) en **S**: genero tupla b:(c,S)

- Reduce: Recibe pares con el mismo atributo B, que vienen de **R** o de **S**. Hago el producto cruz de todos los atributos A que tengo de R y todos los atributos C de S.

- Funciona por que

$$R \bowtie_{R.B=S.B} S = \bigcup_{b \in \pi_B(R)} b \times \left(\pi_A(\sigma_{R.B=b}(R)) \times \pi_C(\sigma_{S.B=b}(S))\right)$$