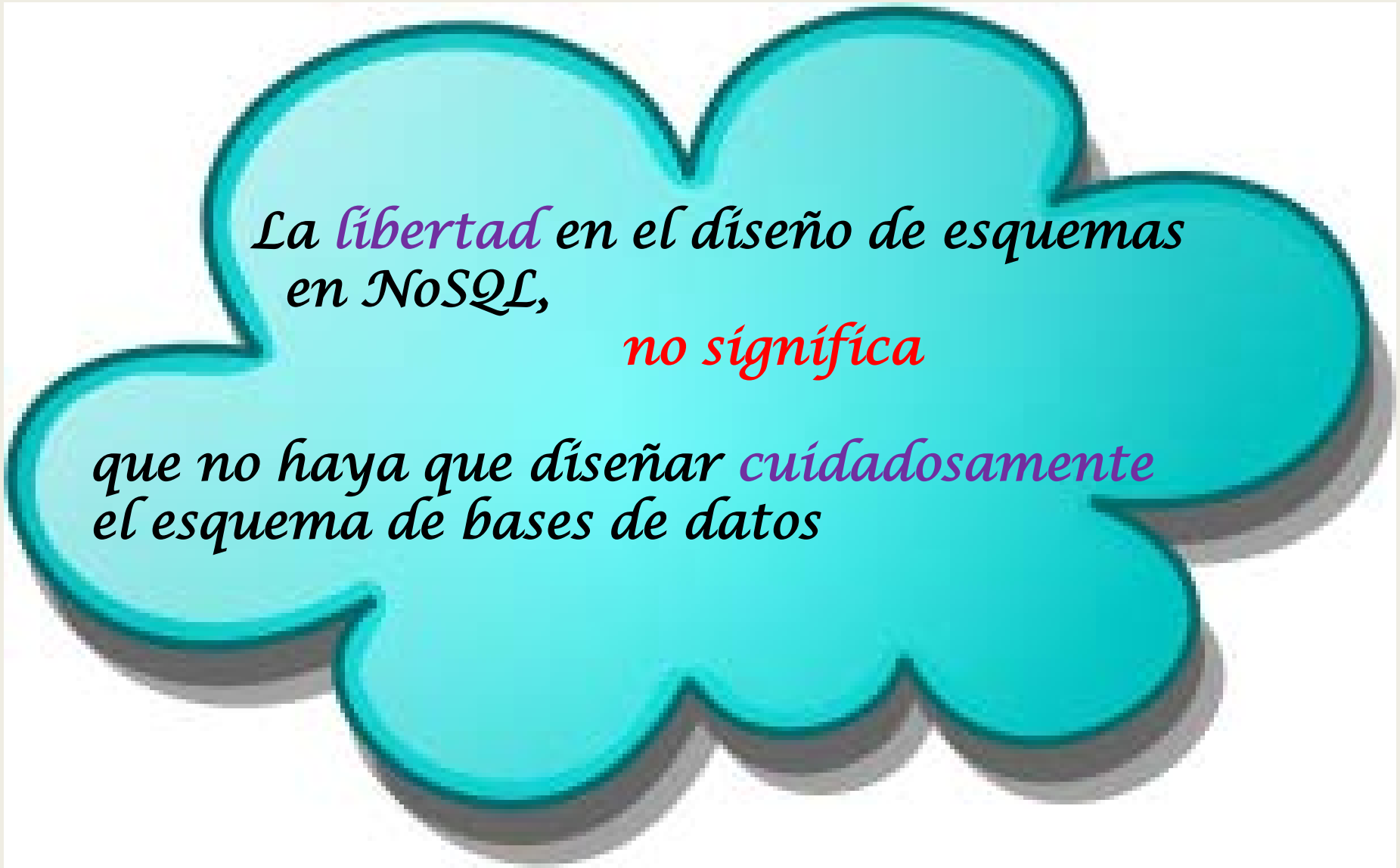
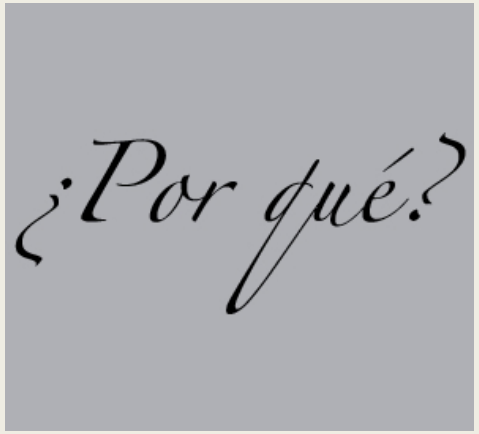
MODELADO EN MONGODB

NoSQL – Fdi -UCM

Índice

… dicho de otra forma





Total, si podemos reorganizar el esquema dinámicamente….

2 (obvias) razones

1. Porque hay que hacer *consultas*!!!!!!!!

   (más nos vale conocer la estructura de la base de datos)

2. Porque el correcto diseño tiene un gran impacto en la eficiencia del sistema en producción

SQL Vs NoSQL: diferencia en modelización

Paradigma	Perspectiva de los datos
Bases de datos relacionales	Agnóstica con respecto a la aplicación/consultas concretas
Bases de datos NoSQL	Application Driven Schema

SQL Vs NoSQL: diferencia en modelización

Paradigma	Pregunta básica para modelar
Bases de datos relacionales	¿Cómo organizamos los datos? (dependencias, normalización)
Bases de datos NoSQL	: ¿Qué consultas vamos a realizar? (agrupar datos a los que se va a acceder simultáneamente,tener en cuenta la arquitectura….)

Errores comunes (I)

Los principios de diseño en SQL nos pueden llevar a errores de diseño en NoSQL

- Minimizar el número de escrituras:
  - *Las bases de datos NoSQL suelen estar optimizadas en escritura*
  - *Realizar más escrituras para reducir lecturas suele ser una buena opción*
  - *Las lecturas suelen ser más costosas y más difíciles de mejorar*

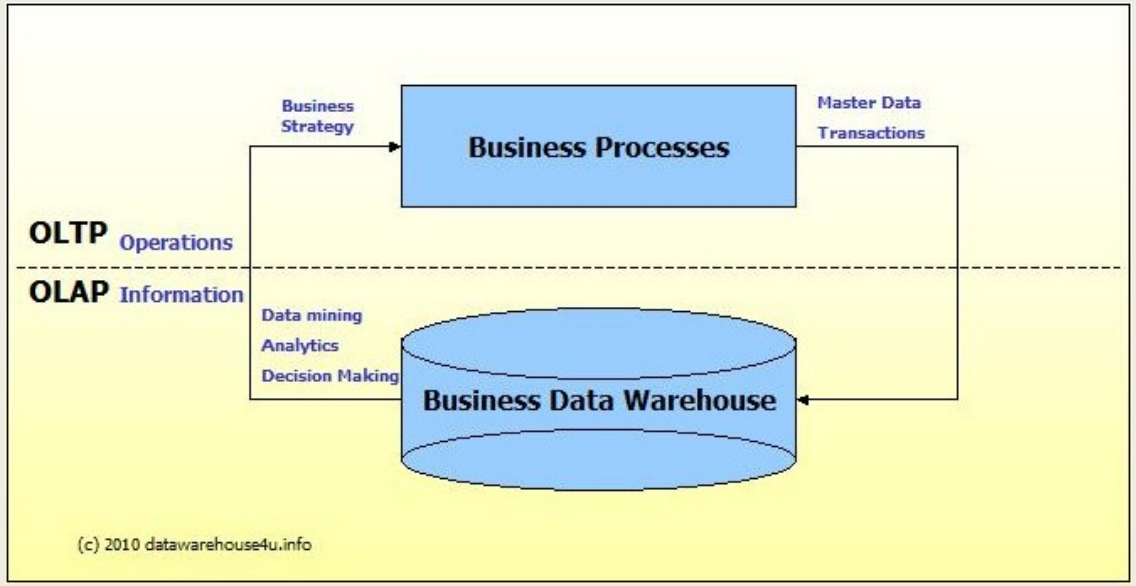
Errores comunes (II)

- Minimizar los valores duplicados:
  - *La desnormalización y la redundancia son "ciudadanos de primera clase" en entornos NoSQL*
  - *El espacio en disco es un recurso barato (comparado con CPU, memoria, red..), y NoSQL está diseñado de esta manera*
  - *Cassandra no tiene Joins y en MongoDB son muy ineficientes, así que a veces no es siquiera una opción.*
  - *En Cassandra se opta más por la duplicación, en MongoDB dicen que es mejor "evitarla mientras sea posible".*

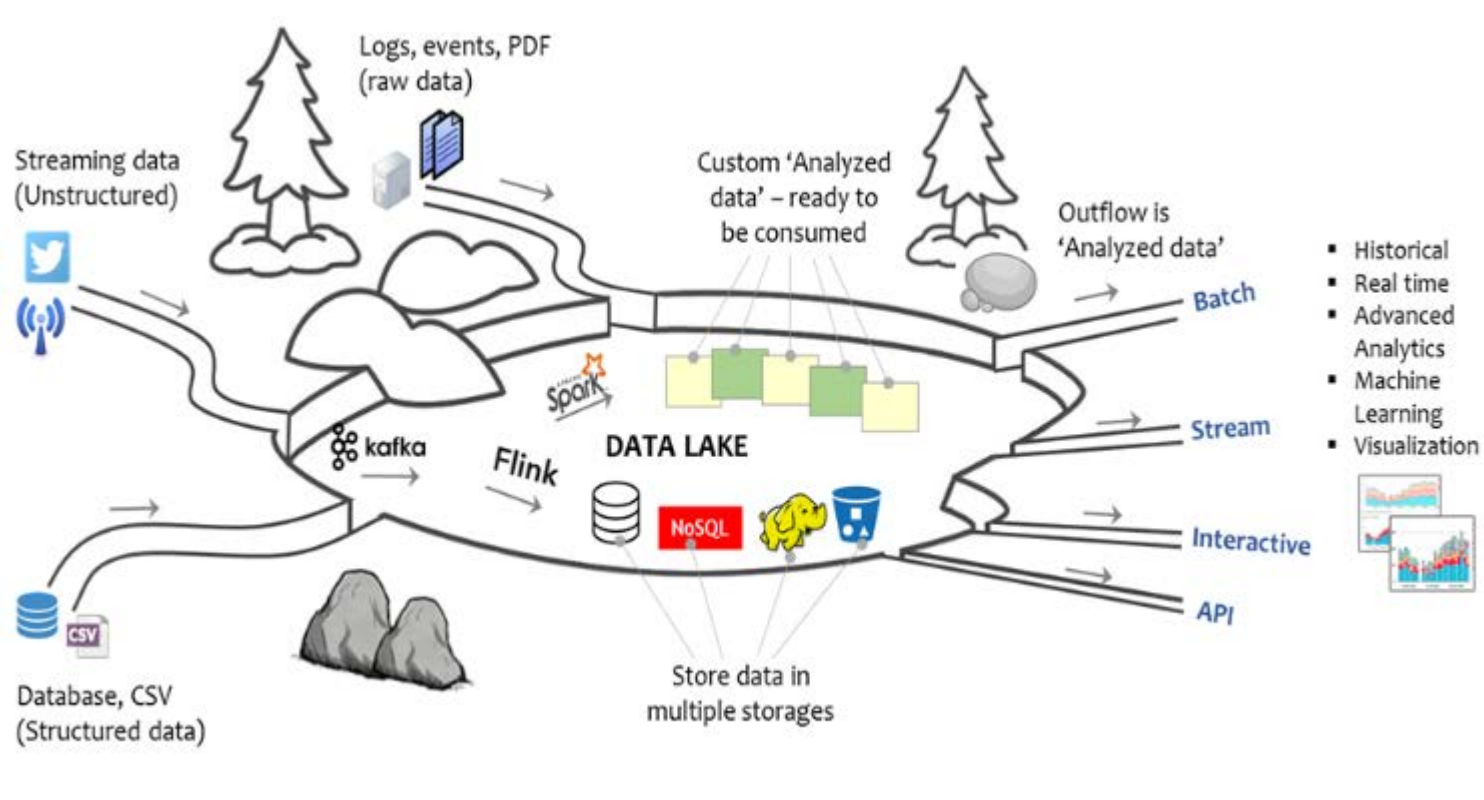
¿Qué perdemos con este nuevo enfoque?

Pérdida de consistencia ¿es tan grave?

- En general, NoSQL se suele relacionar
  - + *con OLAP (On-line Analytical Processing)*
  - - *OLTP (On-line Transaction Processing)*



Pérdida de consistencia ¿es tan grave?



Pérdida de consistencia ¿es tan grave?

- En OLAP la consistencia o bien se tiene a priori o bien se adquiere durante un proceso ETL (o ELT)

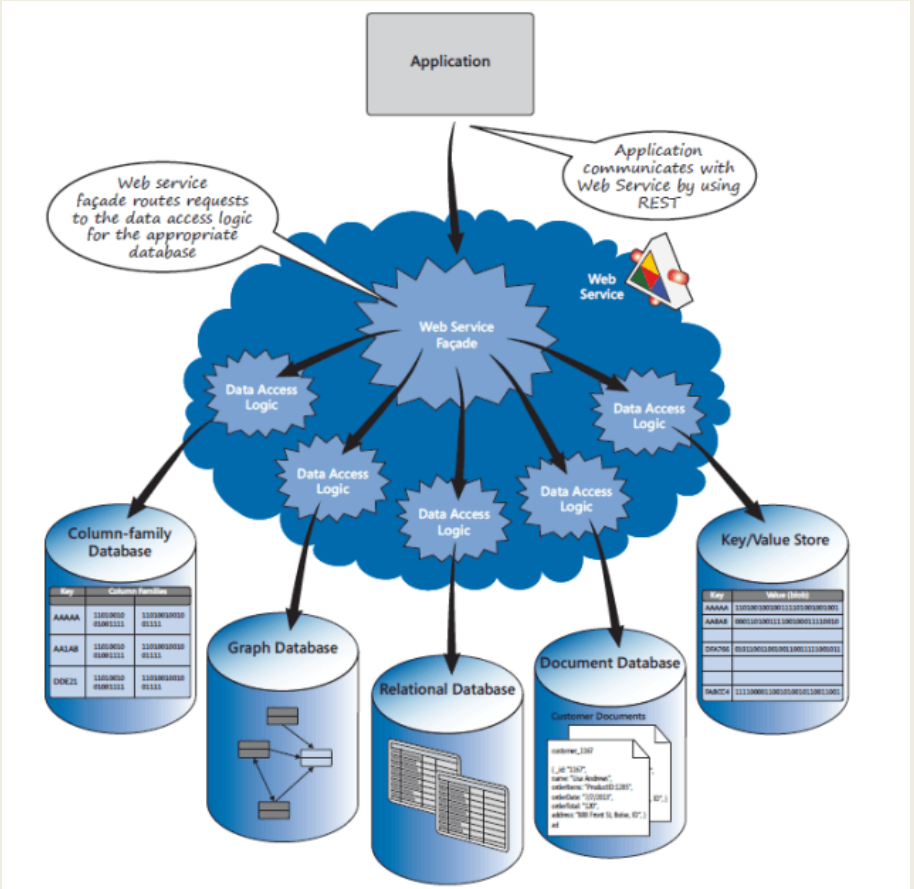
- Si la consistencia es realmente importante, tenerlo en cuenta
  - *Bien en el diseño*
  - *O mediante programación*
  - *….y plantearnos si de verdad estamos eligiendo la BBDD adecuada!*

Un pequeño “Disclaimer”

- Aunque vamos a ver principios generales, cada aplicación requerirá su propia solución

- Distintos problemas exigen herramientas diversas; es importante saber qué se planea hacer con los datos y estar preparado para disponer de varios sistemas complementarios

Persistencia políglota



http://www.jamesserra.com/archive/2015/07/what-is-polyglot-persistence/


mongoDB

MongoDB: factores iniciales (positivos)

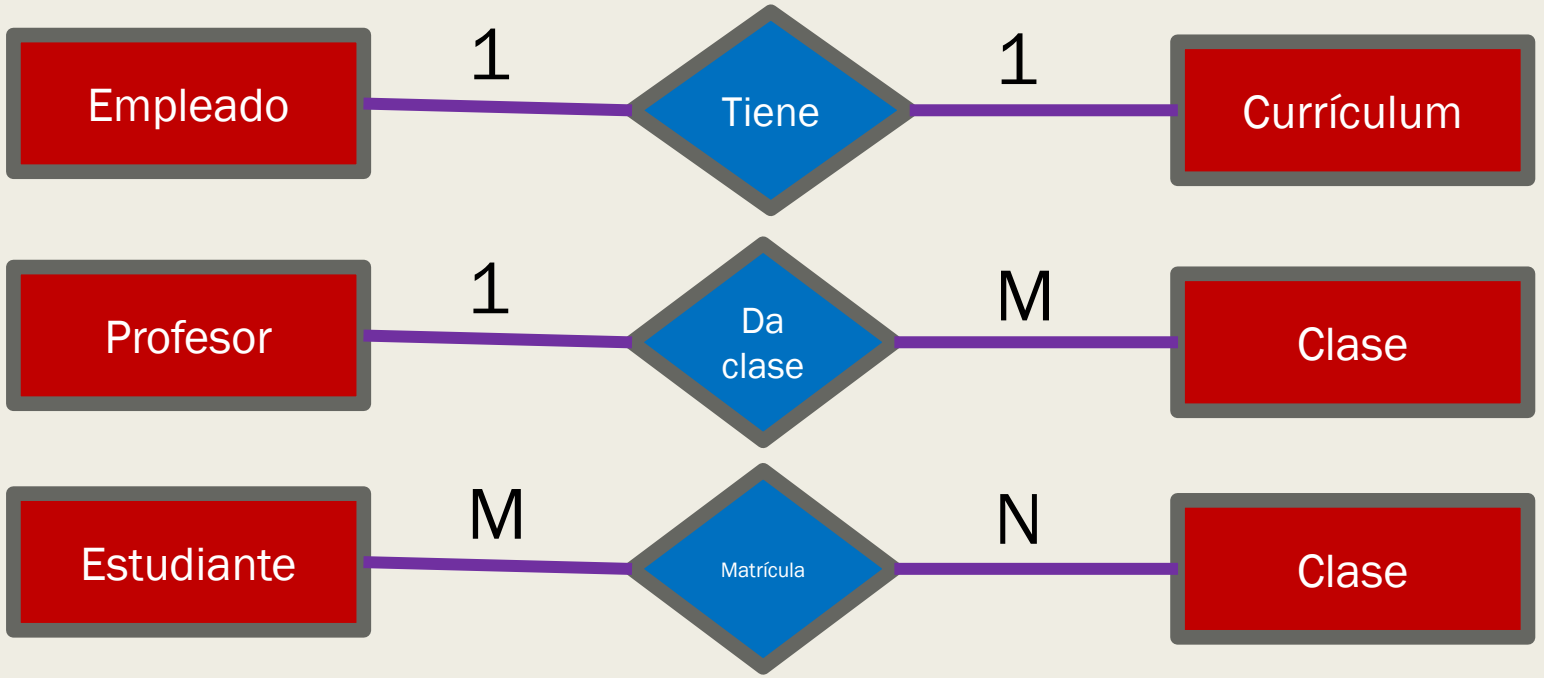
- Importante (y muy desconcertante): no partimos de una metodología bien establecida
- Al diseñar hay que tener en cuenta que disponemos de un entorno "más rico" que en SQL:
  - *Tablas embebidas*
  - *Documentos embebidos con nivel de profundidad arbitrario (árboles)*
  - *Claves que pueden faltar (similar a los nulos pero con diferencias)*
  - *Posibilidad de tener documentos en la misma colección con esquemas distintos*

MongoDB: factores iniciales (negativos)

- Como regla general debemos olvidar los joins
- Tampoco tenemos foreign keys
- Ni transacciones aunque sí operaciones atómicas

Modelado de cardinalidades

- Vamos a ver como modelar los distintos tipos de relaciones típicos en SQL dentro de MongoDB



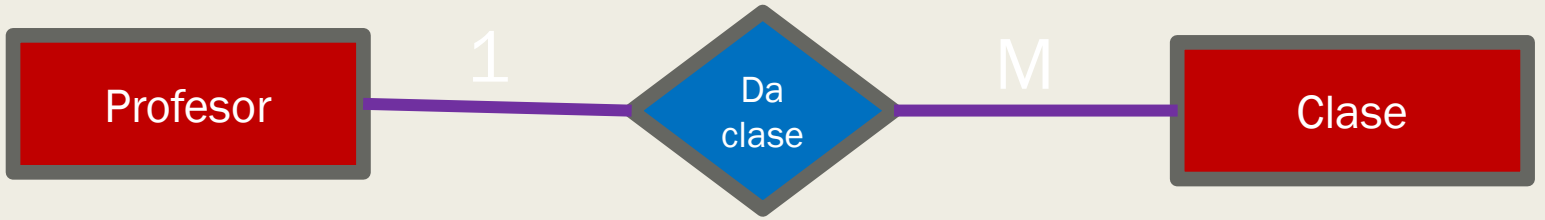
- Se puede generar “claves externas”:
  - *En empleado tener un link al cv, y en el cv el identificador único*
  - *En el cv un id del empleado*
- O se puede embeber
  - *Empleado en currículo (por ejemplo bajo clave “datos personales”)*
  - *CV en empleado*

¿Cómo se decide?

- Frecuencia de acceso
  - *Puede ser que por ejemplo el CV se consulte muy de tarde en tarde*
  - *→ manternerlos separados*
- Tamaño
  - *CV ocupe mucha memoria y se accede de cuando en cuando, o ocupan juntos >16Mb*
  - *→ separados*
- Crecimiento
  - *Supongamos que el CV puede crecer*
  - *Eso signfica en MongoDB moverlo dentro de la colección, puede que queramos ahorrar tiempo moviendo la menor cantidad de datos*
  - *→ separados*

- Tipo de consulta
  - *Casi siempre interesa consultar todos los datos*
  - *→ Unirlos*

- Atomicidad
  - *No queremos correr riesgo de inconsistencias*
  - *→ Unidos*



Depende del tamaño de M

1. Si M es pequeño se puede embeber como un array clase en profesor. Si clase tiene muchos datos, se mantiene por separado y se guardan solo sus ids
2. Si M es mediano y los datos del profesor no son demasiado grandes, se puede embeber el profesor en la clase (*)
3. Si M es muy grande, o los datos del profesor ocupan demasiado se pueden mantener como dos colecciones, con clase conteniendo el _id del profesor

(*) A tener en cuenta cuando embebemos documentos

- Es importante pensar en si una de las entidades puede existir sin la otra y en qué orden se insertarán
  - *En el caso de profesores y clases, puede no ser una buena idea embeber el profesor en la clase si metemos a los profesores al principio y las clases más tarde (además del problema de la redundancia)*

Modelado de relaciones 1 a M

¿Cómo modelarías este caso?

Posibilidades

1. Realmente es un caso de “pocos” a “pocos”
   - *Dejar separadas y añadir a uno un array con los _ids del otro*
   - *¿Cuál elegir?*
2. Realmente “Muchos a pocos”
   - *Separadas*
   - *Crearemos el array en la parte muchos, con ids de la parte pocos.*
3. Realmente “muchos a muchos”
   - *Separadas*
   - *Tenemos que crear una tercera que las relaciones (como en SQL)*

Modelado de relaciones M a N

¿Cómo modelarías este caso?

Documentos embebidos y velocidad

- Mejora de eficiencia en lectura
  - *Discos duros: alta latencia (para ellgar al principio) y alto ancho de banda (una vez llegado allí el resto va más veloz)*

- Modificaciones “one round trip”:
  - *Se puede modificar sobre la marcha evitando la latencia*

Árboles: problema

- Problema clásico, difícil de resolver en SQL
- Ejemplo: los productos que se ordenan por categoría y subcategorías:
  - *Informática -> Impresoras -> láser -> HP -> consumibles -> tóner HP400*
- Cuando se vende un producto nos interesa conocer todas sus categorías

¿Cómo modelarlo?

Árboles: solución

Productos

{

nombre: “HP tóner M400”,

_Id: 35,

Precio: 124,

categorias: [3,6,7,11,13]

}

{

nombre: “consumibles”,

_Id: 3,

categorias: [6,7,11,13]

}

{

nombre: “HP”,

_Id: 6,

categorias: [7,11,13]

}