

Arquitectura de Software 75.73

NoSQL

NoSQL

1. Historia
2. BD Relacionales
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - Características
 - Aplicación

Historia



Pocos cambios a lo largo de los años

- **BD Relacionales**

20 años de reinado

La pregunta: ¿Cuál usar?



- **BD Orientadas a Objetos**

Principios de los '90



- **BD NoSQL**

Una opción adicional

NoSQL

1. Historia
2. **BD Relacionales**
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - Características
 - Aplicación

BD Relacionales – Beneficios

1. Almacenamiento

- Gran cantidad de información
- Flexibilidad para almacenar/buscar fragmentos de información



2. Modelo (*casi*) estándar

- Operación de forma casi estándar
- Salvo pequeñas diferencias (comerciales), casi todas las BD utilizan el mismo lenguaje (*SQL*)

BD Relacionales – Beneficios

3. Concurrency

- Usuarios accediendo en forma simultánea
 - Lectura y escritura
 - Mismos o diferentes fragmentos
- Difícil de lograr, incluso para buenos programadores
- Soporte efectivo en las BD relacionales, utilizando *Transactions*
 - Funcionaron muy bien por muchos años

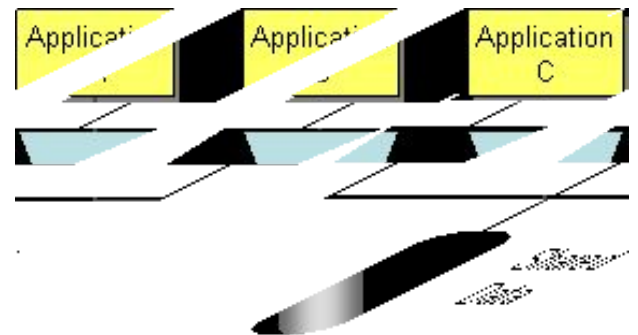
Transactions

- Controlan el acceso a datos
 - Usuarios *concurrentes*
- Ayudan al control de errores
 - Si algo falla, *rollback* de todo

BD Relacionales – Beneficios

4. Integración

- Se requiere que múltiples apps accedan a los mismos datos
 - Diferentes owners
- *Shared Database Integration*
 - Múltiples aplicaciones almacenan sus datos en una única BD
 - La BD, con su soporte de concurrencia, soporta varias aplicaciones como si fueran distintos usuarios dentro de una única aplicación



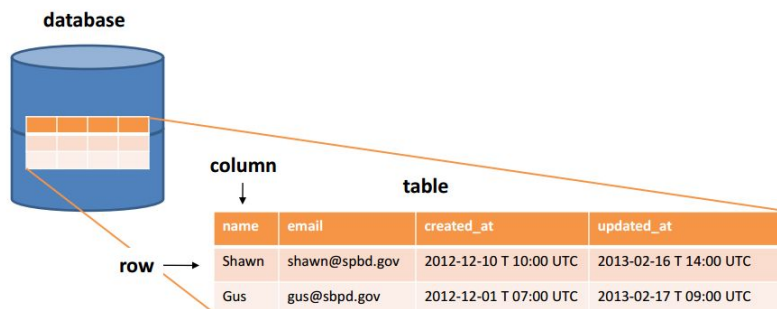
NoSQL

1. Historia
2. BD Relacionales
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - Características
 - Aplicación

BD Relacionales – Frustraciones

Formato (tablas y filas)

- Relaciones y tuplas (registros)
- Las relaciones son un set de tuplas
- Las tuplas son sets de parejas clave-valor
- Las operaciones de SQL reciben/devuelven relaciones, una elegante álgebra relacional

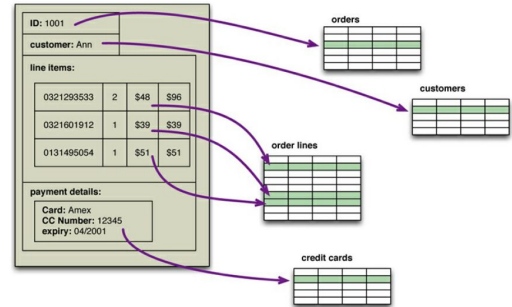


BD Relacionales – Frustraciones

1. Impedance Mismatch

Diferencia entre el modelo relacional y la estructura de datos en memoria

- Los tipos en las BD son simples, sin estructura
- No se aceptan listas, ni *nested records*
- En memoria, existen estructuras más interesantes
- Se requiere código de traducción



¿Cómo almacenar en disco con la misma estructura de memoria?

- Bases de datos (y lenguajes) orientados a objetos ☐ No prosperaron
- ORM Frameworks (mapeo objeto-relacional)
 - Problemas de performance
 - Problemas generales por ignorar que sucede por debajo

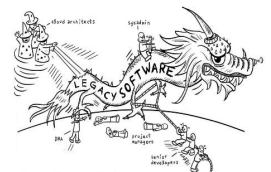
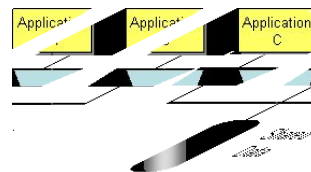


BD Relacionales – Frustraciones

2. Shared DB Integration

Conectar aplicaciones es positivo, pero también tiene sus inconvenientes

- Varias aplicaciones, estructuras más complejas
- Las aplicaciones deben coordinar cambios
- Distintas necesidades de estructura y performance
 - Ej. un índice puede ser crítico para una, un problema para otra
- No se puede delegar en las aplicaciones el soporte de integridad de datos
 - Las aplicaciones pueden tener distintos owners
 - Los motores de BD deben contar con todo el soporte



Surgen los **servicios web** como forma de interconectar aplicaciones

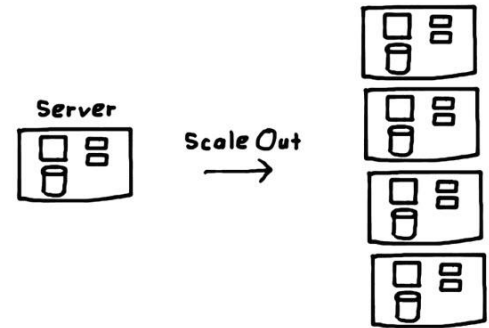
- Soportan estructuras de datos más complejas (XML, JSON, etc.)
- Desde fuera, no se asumen características relacionales
- Mayor flexibilidad de formato/DB en cada aplicación



BD Relacionales – Frustraciones

Escalabilidad de BD

- Cada vez más, los sistemas necesitan escalar, tanto en usuarios como en cantidad de datos.
 - Ej. Facebook, Google, Amazon, Twitter, etc.
- Esto requiere mayores recursos
- La escalabilidad vertical suele ser limitada
- Se busca poder escalar horizontalmente (*clusters*)
 - Equipos más económicos (*commodities*)
 - Mayor resistencia a fallos (sin único punto de falla)
- Las empresas promedio, ¿tienen los mismos requerimientos que las empresas mencionadas?



BD Relacionales – Frustraciones

3. Attack of the Clusters

Las BD relacionales no fueron diseñadas para trabajar en clusters

Diferentes servidores pueden ser utilizados para distintos sets de datos (*sharding*)



- Distribuye la carga
- El sharding debe ser controlado por la aplicación
- Se pierden los queries, integridad referencial, transacciones y controles de consistencia entre shards
- Costos de licenciamiento



NoSQL

1. Historia
2. BD Relacionales
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - Características
 - Aplicación

NoSQL

- La dificultad de usar BD relacionales en clusters impulsó a varias empresas a buscar alternativas de BD distribuidas (no relacionales)
-  (BigTable) y  (Dynamo), quizá las más influyentes
 - 2000's, motivadas en sus necesidades
 - Exitosas, creciendo, capacidad técnica y económica
- Meetup llamado #NoSQL (Jun 2009, San Francisco)
 - Un simple nombre terminó representando una tendencia

NoSQL

- No existe una definición de BD NoSQL
- Tampoco existe una autoridad que pueda crearla
- Sin embargo, las BD NoSQL comparten ciertas características

nosql



“Es mejor pensar NoSQL como un movimiento más que como una tecnología” (Fowler)

NoSQL

1. Historia
2. BD Relacionales
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - **Características**
 - Aplicación

NoSQL – Características

- **Generalmente son proyectos open-source**
 - Existen algunas BD comerciales
- **Basadas en las necesidades del siglo XXI**
 - Implementadas luego del año 2000
- **No usan lenguaje SQL (ni el modelo relacional)**
 - Algunas usan lenguaje de consultas (ej. Cassandra CQL)
 - Tiene sentido que sean similares a SQL, para simplificar su aprendizaje
 - Al momento, no habría ninguna que haya implementado algo que cumpla el estándar SQL más básico

NoSQL – Características

- **Diseñadas para funcionar con clusters**
 - Afecta a su modelo de datos y a su estrategia de consistencia
 - Las BD relacionales utilizan transacciones ACID para soportar consistencia, trayendo problemas con los clusters
 - Las BD NoSQL ofrecen una serie de opciones para soportar consistencia y distribución
- **Trabajan sin schema**
 - Aceptan agregar campos a los registros sin necesidad de predefinir cambios en la estructura
 - Útil para trabajar con datos no uniformes, o campos custom

ACID



Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.



The consistency property ensures that any transaction will bring the database from one valid state to another.



The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other.



The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

NoSQL

1. Historia
2. BD Relacionales
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - Características
 - Aplicación

NoSQL – Aplicación

- Las BD relacionales no desaparecerán (no serán únicas; será una de las opciones)
 - Seguirán siendo las más utilizadas, recomendadas para el uso general
 - Familiaridad, estabilidad, features, soportes, etc.
- Polyglot persistence: diferentes BD en diferentes circunstancias
 - Utilizar relacionales, NoSQL, o mix (según necesidades)
 - No dejarse llevar por tendencias (la más usada) o modas (lo trendy)
- Dos principales razones para usar NoSQL
 - Volumen de datos y reqs. de performance que demanden clustering
 - Simplificar (+ productividad) de desarrollo, evitando Impedance Mismatch

NoSQL

1. Historia
2. BD Relacionales
 - Beneficios
 - Frustraciones
3. NoSQL
 - Introducción
 - Características
 - Aplicación

Consultas?

Feedback

<https://goo.gl/forms/NvrORS12kuuBitpE3>

Christian Calónico
ccalonico@fi.uba.ar



Arquitectura de
Software 75.73



NoSQL

Christian Calónico
75.73 Arquitectura de Software



NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Modelos de Datos

- Un **modelo de dato** es la forma en que percibimos y manipulamos los datos.
- En una BD, el modelo de dato describe como se interactúa con los datos de la misma.
- Esto difiere del **modelo de almacenamiento**, que describe como la BD almacenan y manipula los datos internamente.



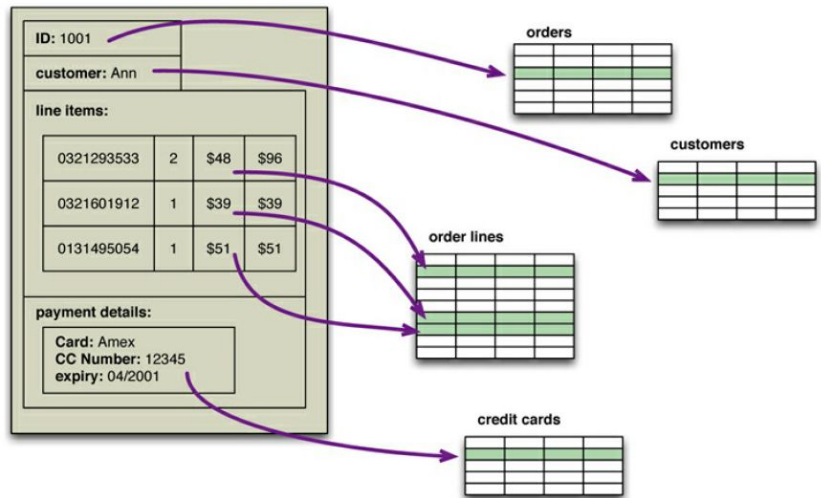
NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

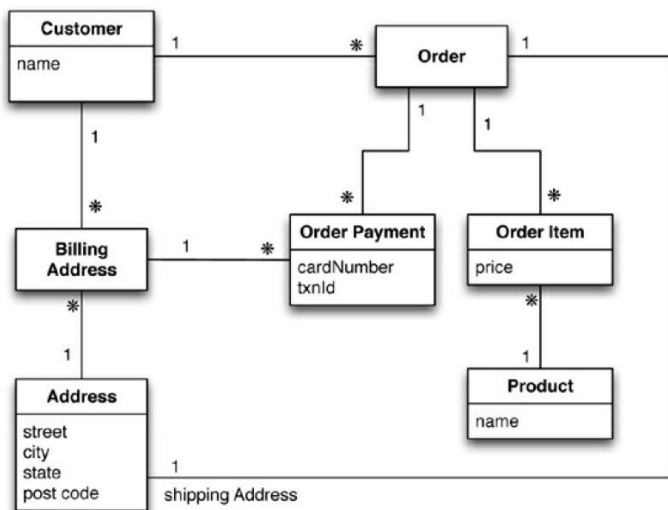
Modelo de Dato Relacional

El modelo de dato predominante en las últimas décadas fue el **relacional**

- Set de tablas
- Las tablas tienen filas, que representan entidades
- Las filas tienen columnas, cada una con un valor simple
- Una columna puede hacer referencia a otra fila, en la misma u otra tabla (relación)



Modelo de Dato Relacional



Customer	
Id	Name
1	Martin

Orders		
Id	CustomerId	ShippingAddressId
99	1	77

Product	
Id	Name
27	NoSQL Distilled

BillingAddress		
Id	CustomerId	AddressId
55	1	77

OrderItem			
Id	OrderId	ProductId	Price
100	99	27	32.45

Address	
Id	City
77	Chicago

OrderPayment				
Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

NoSQL

1. Modelos de Datos
 - Relacional
 - **Aggregates**
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Aggregates

- NoSQL abandona el modelo relacional
- Las BD NoSQL adoptan modelos distintos, clasificables en 4 categorías ampliamente utilizadas:
 - Key-Value
 - Document
 - Column-Family
 - Graph
- Las tres primeras comparten una característica: están **orientadas a la agregación**

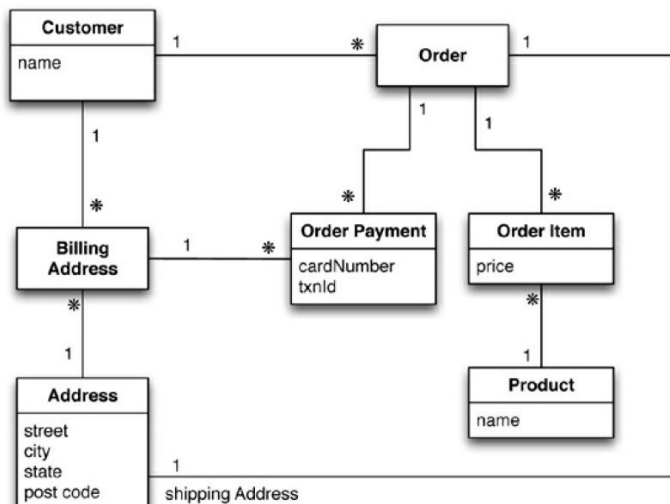


nosql

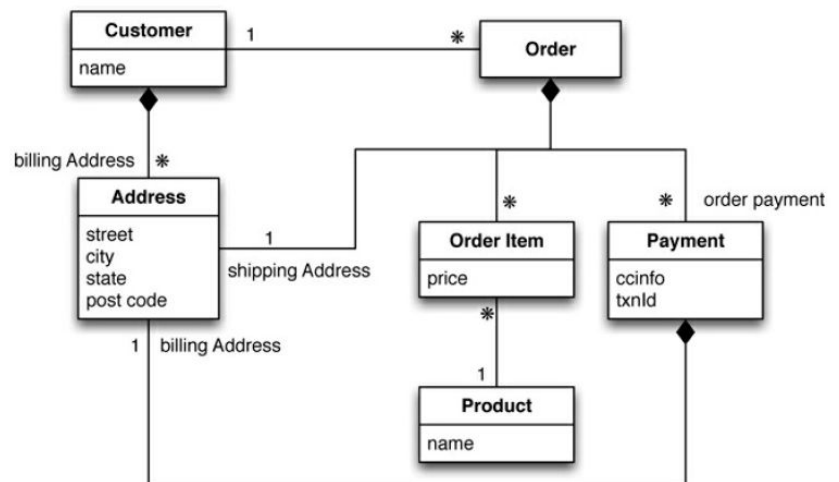
Aggregates

- La orientación a la agregación permite:
 - Romper la estructura limitada de las BD relacionales
 - Manipular la información *en unidades*, con estructura más *compleja*
 - *Anidar* tuplas
 - *Listas* de valores o tuplas
- Los aggregates serían una colección de objetos relacionados, que se desean tratar como *una unidad*
 - Comunicación con la BD en términos de aggregates
 - Manipular aggregates en forma atómica

Aggregates

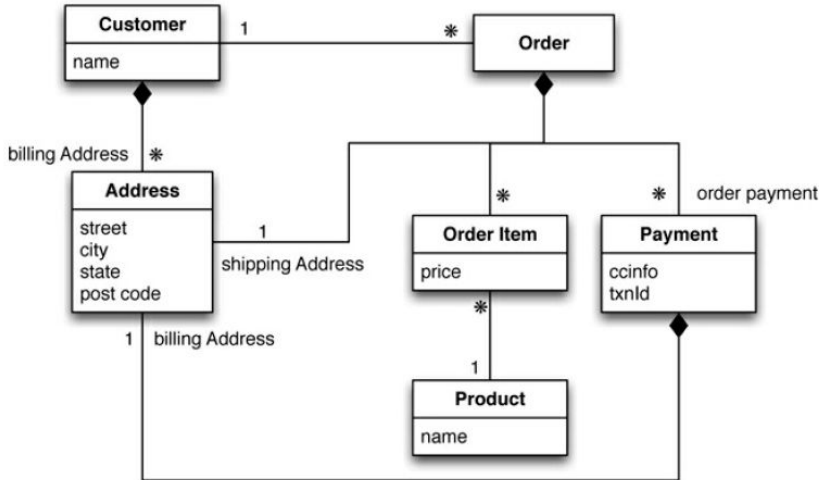


Modelo Relacional



Modelo Orientado a la Agregación

Aggregates



```
// in customers
{
  "id":1,
  "name":"Martin",
  "billingAddress":[{"city":"Chicago"}]
}

// in orders
{
  "id":99,
  "customerId":1,
  "orderItems":[
    {
      "productId":27,
      "price": 32.45,
      "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
  "orderPayment":[
    {
      "ccinfo":"1000-1000-1000-1000",
      "txnId":"abelif879rft",
      "billingAddress": {"city": "Chicago"}
    }
  ],
}
```

Aggregates

Key-Value & Document Data Models

- Aggregates, identificados (indexados) por un ID/clave
 - Schema-less; la BD no asume nada para optimizar
- En Key-Value, el aggregate es *opaco* para la BD
 - Gran blob (cualquier contenido, quizá de cierto tamaño máximo) al que no analiza
- En Document, la BD impone ciertas condiciones (tipos, estructuras, etc.)
 - Mayor flexibilidad al acceder (queries/índices sobre campos, info parcial, etc.)
 - En la práctica, muchas veces el límite entre Document y Key-Value es difuso

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

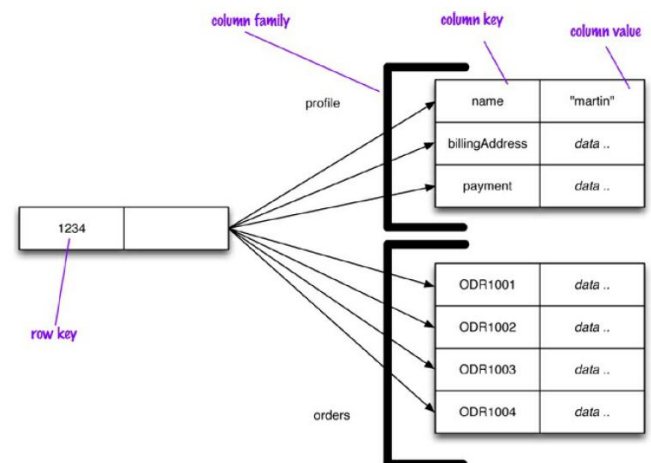
Aggregates

- Importante decidir cómo separar los datos en aggregates:
 - A veces, una agregación ayuda a una tarea (ej. ver una orden)
 - A veces, una agregación complica (ej. ventas anuales de gaseosa)
 - ¿Ordenes de un cliente juntas o dos aggregates distintos (con ID del cliente)?
 - Se ignoran las relaciones (No permite optimizar a la BD)
- En clustering, se minimiza la cantidad de nodos a acceder
 - Información relacionada, un aggregate, en un mismo nodo
- En BD relacionales, ACID garantiza atomicidad al modificar tablas
- Se dice que NoSQL no soporta ACID, sacrificando consistencia
 - En verdad, el aggregate es una *unidad atómica de actualización*
 - Si se modifican varios aggregates juntos, responsabilidad del programador

Aggregates

Column-Family Data Model

- Estructura de aggregates de dos niveles:
 - Primer clave identifica la fila (el aggregate de interés)
 - Además, se puede seleccionar la columna (dentro del aggregate)
 - Cada columna forma parte de una familia de columnas
 - Se asume que los datos de una familia de columnas se acceden usualmente en forma conjunta
- Estilo basado en BigTable de Google
- La BD utiliza el conocimiento para definir el almacenamiento de datos

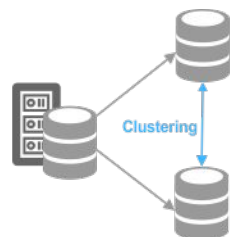


NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Modelos de Distribución

- El principal objetivo de NoSQL es clustering (escalabilidad horizontal)
 - Aggregate como unidad de almacenamiento
- El modelo de distribución definirá:
 - La habilidad para manejar mayores volúmenes de datos
 - La habilidad para manejar mayores tráficos de lectura y/o escritura
 - La habilidad para afrontar los enlentecimientos y caídas
- Clustering implica mayor complejidad
 - Utilizar sólo si realmente se requiere



Modelos de Distribución

- Estrategias para la distribución de datos:

Replicación	Mismo dato en múltiples nodos
Sharding	Distintos datos en distintos nodos

- Son estrategias ortogonales

- Pueden usar una, otra, o ambas

Single-Server

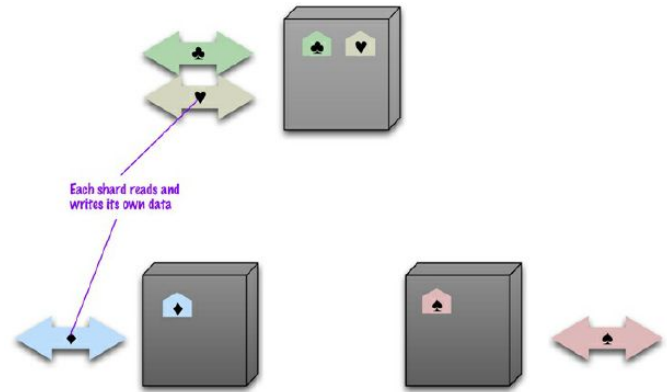
- Estrategia más simple (y recomendada): *no distribuir*
- BD en un único server, que ofrece lectura/escritura de los datos
- Sin la complejidad de otras opciones:
 - Simple de operar por administradores
 - Simple de comprender por *devs*
- En este caso, NoSQL sólo tendría sentido si el modelo de datos se ajusta más a las necesidades de la aplicación
 - Recordar que la principal fortaleza de NoSQL es el soporte de clustering

NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - **Sharding**
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Sharding

- Colocar **diferentes partes** de los datos en **diferentes servers**
 - Se busca reducir la carga de diferentes usuarios accediendo a diferentes partes de los datos
- Caso ideal
 - Diferentes usuarios hablando con diferentes servers
 - Cada usuario habla con 1 server, con rápida respuesta
 - Ej. 10 servidores, cada uno soporta 10% de la carga
- Lo ideal es difícil; acercarse requeriría:
 - ✓ Datos que se acceden juntos, en el mismo servidor
 - ✓ Datos dispuestos favoreciendo performance



Sharding

- Lo ideal es difícil; acercarse requeriría:
 - ✓ Datos que se acceden juntos, en el mismo servidor
 - ✓ Datos dispuestos favoreciendo performance

facebook

Múltiples factores a analizar. Ej.:

- Criterio que distribuya los datos de forma equitativa entre nodos
 - Ej. apellidos de A..D, E..F, R..Z
 - Quizá hasta de forma dinámica
- Aprovechar zonas geográficas para disminuir la latencia

Los aggregates ayudan en este aspecto

- Se diseñan combinando los datos que suelen accederse en forma conjunta
- Serían la unidad de distribución

- Mejora Performance (lectura/escritura)
- Reduce Robustez
(falta de un nodo implica indisponibilidad parcial de datos)

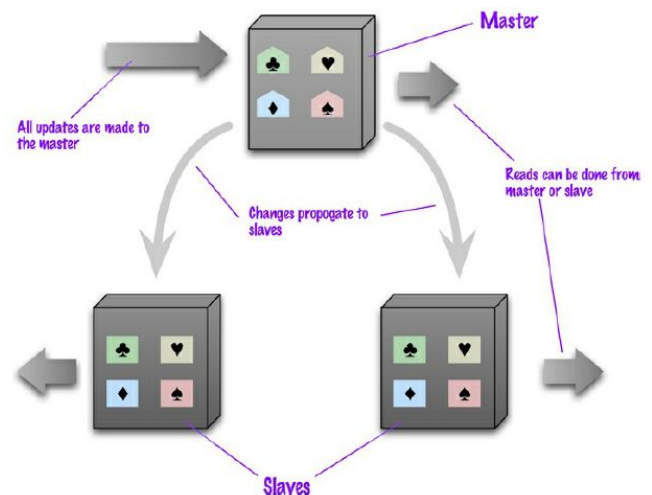
NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - **Master-Slave Replication**
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Master-Slave Replication

- Un nodo se designa como *master* o *primario*
 - Recibe todas las operaciones de update
- Los restantes nodos son *slaves* o *secundarios*
 - La información es **replicada** en los esclavos
 - El proceso de replicación se realiza a partir de las actualizaciones realizadas al master
- Las operaciones de lectura pueden realizarse tanto desde el master como de los distintos esclavos

Útil para escalar en escenarios de alto tráfico de lecturas (se distribuyen) y poco tráfico de escrituras (se aliviana por la disminución de lecturas)



Master-Slave Replication

- Mejora la resistencia a fallos
 - Si se cae un nodo (master o slave), los restantes pueden lecturas
 - Si se cae el master, caen las escrituras
 - El master es un único punto de falla
 - Se debe restaurar el master
 - La existencia de slaves simplifica la tarea; alguno puede ser designado como master
 - Designación automática del master puede reducir downtime entre fallas
- Potencial problema de consistencia
 - Ej. dos clientes leen datos distintos
 - Consistencia eventual

The Facebook logo, consisting of the word "facebook" in white lowercase letters on a blue rectangular background.

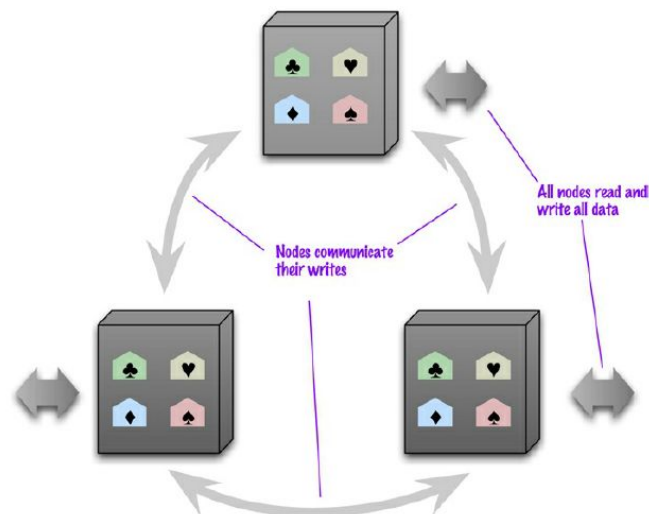
NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - **Peer-to-Peer Replication**
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Peer-to-Peer Replication

- Se ataca el problema del único punto de falla de Master-Slave Replication
 - No existe un master
 - Todas las réplicas tienen igual importancia
 - Todos los nodos aceptan lecturas/escrituras
- Continúan las inconsistencias transitorias de lecturas
- Aparecen conflictos write-write
 - Ej. dos usuarios, accediendo a nodos diferentes, modifican el mismo registro al mismo tiempo
 - Alternativa: coordinar nodos antes de escribir
 - Brinda seguridad, reduce performance
 - Alternativa: soportar eventual inconsistencia
 - Pero aprovechar máxima performance

facebook

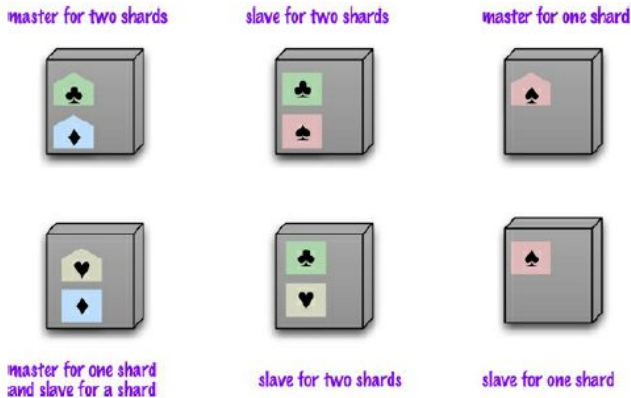


NoSQL

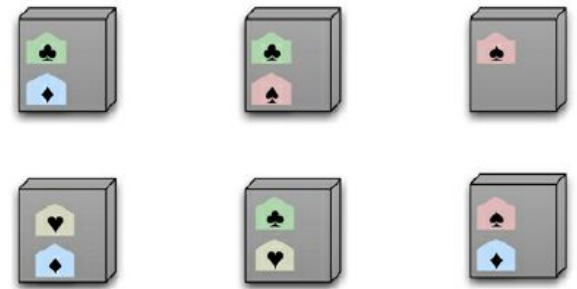
1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - **Sharding + Replication**
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Sharding + Replication

- Las estrategias de sharding y replicación pueden combinarse



Sharding + Master-Slave



Sharding + Peer-to-Peer

NoSQL

- Modelos de Datos
 - Relacional
 - Aggregates
- Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
- Consistencia**
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Consistencia

- Las BD relacionales y centralizadas ofrecen **consistencia fuerte**
- En BD NoSQL, orientadas a clustering, se piensa **distinto** a nivel consistencia.
 - “Teorema CAP”
 - “Consistencia Eventual”
- Se debe **pensar** que tipo de consistencia se requiere para la solución que se quiere desarrollar
 - ¿Puedo sacrificar consistencia para ganar otra cosa?
 - ¿Cuál sería el impacto?
 - ¿Con qué frecuencia se daría?

NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - **Update Consistency**
 - Read Consistency
 - Teorema CAP

Update Consistency

- Conflicto **write-write**

- Dos personas (A y B) actualizan el mismo dato al mismo tiempo

- Ejemplo 1 – Single server; updates en orden, uno tras otro (A→B)

- Se pierde el update de A (no suele ser problema usualmente)
- Salvo que el B lo hubiera hecho considerando un cierto estado original de la BD


- Ejemplo 2 – Si existe más de un nodo (ej. peer-to-peer replication)

- En cada nodo, los updates podrían darse en orden distinto (A→B o B→A)
- Podría resultar en valores finales distintos en cada nodo

- La replicación aumenta las chances de un conflicto write-write

Update Consistency

- Approachs al mantenimiento de la consistencia

- **Pesimista**: se intenta prevenir el conflicto
 - **Optimista**: deja ocurrir el conflicto, lo detecta, y corrige
- 

- Una táctica pesimista común es el *write-lock*

- Uno sólo puede acceder al lock

- Tácticas optimistas comunes serían:

- *Conditional update* – cada usuario chequea el valor justo antes de actualizar, para ver si cambio luego de su última lectura
- La utilizada por los VCS – almacenar ambos updates, indicar si se produce un conflicto, solicitar una solución al usuario (o hacerlo automáticamente si se puede)

NoSQL

1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Read Consistency

- Conflicto **read-write**
 - Un usuario (B) hace una lectura en medio de dos escrituras de (A)
- Ejemplo 1 – e-Commerce Web
 - El costo de envío depende del volumen de las cajas a enviar (productos vendidos)
 - (A) agrega un producto a la orden; posteriormente, (A) actualiza el costo de envío
 - En medio de las dos escrituras de (A), (B) lee el pedido (productos + costo de envío)
- **Logical consistency**
 - Diferentes elementos de datos tienen sentido en conjunto
 - Las BD relacionales utilizan *transactions* – (B) leería los dos cambios juntos, o ninguno
 - Las BD NoSQL utilizan *atomic updates* a nivel de aggregates
 - La orden, los ítems, y el costo de envío podrían ser parte del mismo aggregate
 - Si se usan distintos aggregates, existe una ventana de tiempo que daría inconsistencias (*inconsistency window*)

Read Consistency

- **Replication consistency**

- Un cierto dato tiene el mismo valor cuando es leído desde distintas réplicas
- La replicación exagera la *consistencia lógica*, extendiendo la *ventana de inconsistencia*. Ej.:
 - (A) modifica el dato X en el nodo 1, que se ha replicado en el nodo 2, pero aún no en el 3
 - En ese momento (B) y (C) leen el dato X (distinto!), uno desde nodo 2, el otro desde nodo 3

- **Consistencia eventual**

- Eventualmente, de no haber más updates, tanto (B) como (C) podrán leer el valor correcto

- Ejemplo 3 – Blog

- (A) y (B) leen el blog, pero (A) demora un par de minutos más en ver el último post
 - ¿Es un problema? Tener en cuenta que ganó load balancing, clustering, performance, etc.
- (A) escribe un nuevo post... pero luego de apretar Send, no lo ve (impacta un nodo, lee de otro)
 - Se requiere **Your-Writes Consistency**, en lo que se denomina **Session-Consistency** (ej. sticky session)

NoSQL

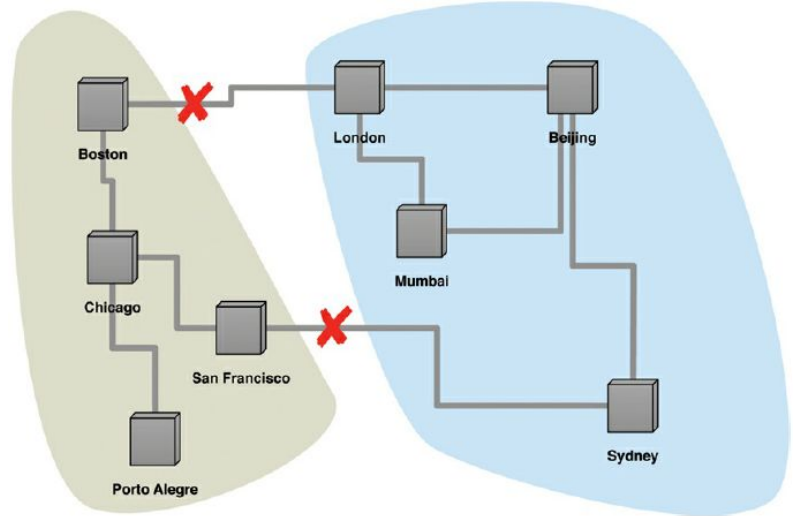
1. Modelos de Datos
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Teorema CAP

- La consistencia es buena, pero a veces hay que sacrificarla
 - Tradeoffs entre atributos, que dependen del problema a resolver

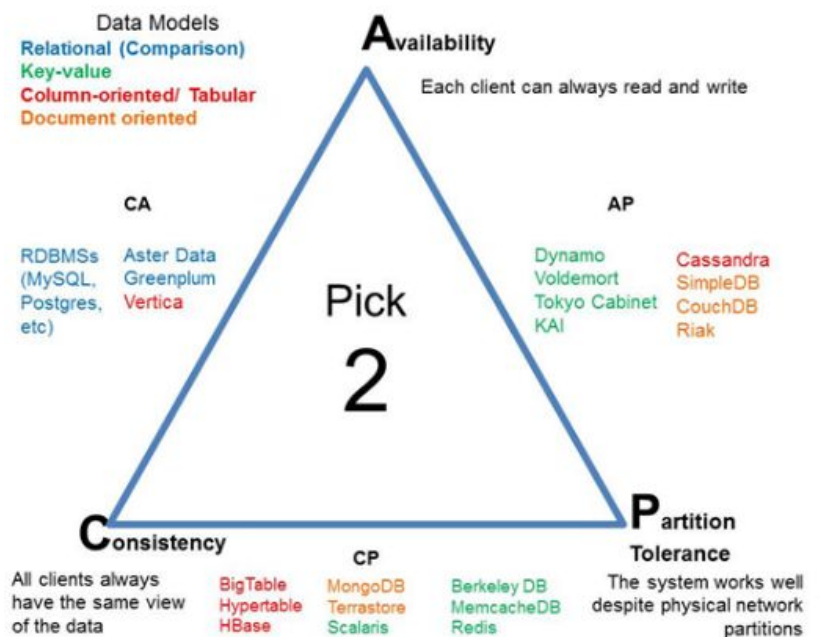
Teorema CAP [Eric Brewer, 2000]

- “If you get a network partition, you have to trade off availability of data versus consistency”



Teorema CAP

- El teorema dice que, de estas 3 propiedades, sólo se pueden elegir 2
 - **Consistency:** todos los nodos deben garantizar la misma información al mismo tiempo [...]
 - **Disponibilidad:** cuando se hace R/W en un nodo activo, este responde en tiempo acotado y sin errores
 - **Tolerancia al particionado:** ante una caída, el sistema sigue funcionando a pesar de que existan problemas de comunicación entre los nodos



Teorema CAP – Ejemplo

Capítulo 1: La idea

Una noche cuando tu esposa agradece que te acuerdes de vuestra fecha de aniversario de matrimonio, te viene a la cabeza una idea curiosa. Muchas personas tienen problemas para recordar las cosas, y sin embargo tu no tienes es problema, porque eres muy bueno recordando, entonces decides crear tu propia startup: Recuerdos Inc.

Escribes tu anuncio en el diario:

- *Recuerdos Inc: Nunca olvide, ¡sin necesidad de recordar! ¿Se ha sentido mal por olvidar demasiado? No se preocupe, simplemente llámenos, marque el 555-RECUERDE y díganos lo que necesita recordar después. ¿No recuerda la fecha de cumpleaños de su jefe? Llame al 555-RECUERDE y se la recordaremos. No olvide más, no necesita recordar nada, nosotros recordamos por usted. (Costo \$ 290, IVA incluido)*
- *Y contratas una línea telefónica, te compras unos cuadernos y empiezas a atender. Un diálogo en tu servicio sería más o menos así:*
- *Cliente: Hola, ¿podría guardar la fecha de cumpleaños de mi vecino?*
- *Tú: Claro, ¿cuando es?*
- *Cliente: el 4 de mayo*
- *Tú: (anotas la fecha en la página del cliente en un cuaderno) ¡Anotado! Llámenos cuando necesite saber el cumpleaños de su vecino*
- *Cliente: ¡Gracias!*
- *Tú: De nada, esta llamada tiene un costo de \$290 el minuto, gracias por llamar.*

Teorema CAP – Ejemplo

Capítulo 2: empiezas a escalar

Consigues algo de capital de riesgo, y empiezas a recibir cientos de llamadas al día. Y junto con esto empiezan los problemas, las llamadas se empiezan a encolar, la gente cuelga el teléfono después de esperar tanto rato. Y para colmo un día te enfermas y pierdes todo un día de atención. Así que decides escalar, le pides a tu mujer que te apoye y se incorpore al servicio. Compras una central telefónica, colocas un anexo para tu esposa y duplicas tu capacidad de atención (¡y duplicas tus ingresos!)



Teorema CAP – Ejemplo

Capítulo 3: tu servicio es malo

- *Juan: Hola*
- *Tú: Buenas tardes señor, gracias por llamar a Recuerdo Inc. ¿en que podemos servirle?*
- *Juan: ¿Puede decirme cuando es mi viaje a Puerto Montt?*
- *Tú: Un segundo por favor (buscas en tu cuaderno, ¡y no hay ningún vuelo anotado en la página de Juan!) Señor, creo que hay un error, usted no nos ha dicho nada sobre un viaje a Puerto Montt.*
- *Juan: ¡Qué! ¡Si yo los llamé ayer! (y cuelga)*

Conversas con tu esposa y te das cuenta que ella atendió a Juan ayer, pero el vuelo está anotado en el cuaderno de ella, y no en el tuyo. **¡Tu sistema distribuido no es consistente!** Un cliente puede ser atendido por tu esposa, y cuando llame de nuevo puede ser redirigido a otra persona, y Recuerdos Inc. ¡no tendrá una respuesta consistente para el cliente!

Teorema CAP – Ejemplo

Capítulo 4: arreglas el problema de consistencia

Conversas con tu esposa y le dices:

- *Siempre que recibamos una llamada de un cliente antes de completar la llamada le avisaremos al otro.*
- *En ese momento ambos anotaremos el recordatorio.*
- *De este modo ambos tendremos copia de los recuerdos y no tendremos que ir a leer el cuaderno del otro.*

Esto implica que cuando se tenga que actualizar el cuaderno no podremos atender a nadie. Eso no es un gran problema porque la mayor parte de las veces los requerimiento son de consultas que implican buscar algo en los cuadernos, más que escribir. Además lo importante es que no podemos dar respuestas incorrectas, así que este es un costo que debemos asumir.

Pero tu mujer te hace notar que no has considerado el caso cuando uno de los dos no se presenta a trabajar. En ese momento no es posible atender una conversación de actualización, porque no está disponible el otro para copiar a su cuaderno. Estamos ante un problema de **disponibilidad**, si el otro no está, ¡no es posible completar la llamada!

Teorema CAP – Ejemplo

Capítulo 5: una solución astuta

¿Cómo logras que las anotaciones en los cuadernos queden consistentes y disponibles, ante la ausencia de uno de los dos? Para resolver esto se te ocurre una idea brillante, y se la dices a tu pareja:

- Cuando una persona llame para ingresar un recuerdo, es decir, cuando se requiera escribir algo en los cuadernos, lo que haremos es que si está la otra persona disponible entonces le avisaremos en ese momento y ambos actualizarán sus apuntes.
- Si la otra persona no está disponible, entonces le enviaremos un email con la actualización.
- Al otro día, cuando la persona que estuvo ausente regrese lo primero que debe hacer es leer sus emails y actualizar su cuaderno antes de poder empezar a atender el teléfono.

Genial! Ahora Recuerdos Inc. tiene un servicio **consistente** y de **alta disponibilidad**.

Teorema CAP – Ejemplo

Capítulo 6: tu mujer está enojada contigo

Pero un día discutes con tu mujer, y estás tan molesto que decides salir y presentarte a trabajar. ¿Qué tal si ella está tan enojada contigo que decide no actualizarte de nada y no manda los emails acordados?.

*Tu genial idea fracasa, porque a pesar de que es una idea astuta que garantiza consistencia y alta disponibilidad no es **tolerante a una partición**. Puedes decidir ser tolerante a fallos tomando la decisión de no atender llamados hasta que te hayas reconciliado con tu mujer, pero si haces eso entonces **¡sacrificas la disponibilidad** durante todo ese tiempo!.*

Teorema CAP – Ejemplo

Conclusión:

Así que el teorema CAP nos dice que cuando diseñas un sistema no puedes tener las tres características: Consistencia, Disponibilidad o Tolerancia a Particiones. Sólo puedes elegir dos:

- *Consistencia: tus clientes, una vez que han actualizado información siempre tendrán la información más actualizada cuando llamen nuevamente. No importa que tan rápido vuelvan a llamar.*
- *Disponibilidad: Recuerdos Inc siempre estará disponible para llamadas mientras cualquiera de ustedes (tú o tu esposa) se presente a trabajar.*
- *Tolerancia a Partición: Recuerdos Inc operará siempre, aunque se haya perdido la comunicación entre tú y tu esposa*

Teorema CAP – Ejemplo

Consistencia Eventual:

¿Qué pasaría si contrataras a un ayudante que se encargue de actualizar los cuadernos cada vez que hay un cambio?. Este personaje puede actualizar los cuadernos en “background”, mientras no se ocupan, su labor es mantener consistente las anotaciones, claro que con un cierto retraso. La base no se encuentra consistente de inmediato, pero con esto no sacrificas la disponibilidad, a cambio de un cierto grado de inconsistencia temporal.

Así es como funcionan muchas bases de datos NoSQL, por ejemplo. Lo importante es que el ayudante haga su trabajo con cierta celeridad, si realiza su labor digamos que cada 5 minutos, entonces reduces las probabilidades de que te encuentres con inconsistencias.

Por lo tanto debemos aceptar que no podemos controlar todas las situaciones y hay veces que nuestras arquitecturas van a fallar. Podemos diseñarlas para maximizar estas tres propiedades, pero eventualmente algo deberemos sacrificar. La pregunta es qué. ¿La disponibilidad por la consistencia, o la consistencia por la disponibilidad? Ese es el desafío, y buscar medidas de mitigación es nuestra labor como arquitectos. Pero no olviden, que habrá ocasiones en que simplemente tendremos que devolver el dinero, o pedir disculpas, eso es seguro, tenemos un teorema que lo demuestra. :)

NoSQL

1. Modelos de Datos
 - Definición
 - Relacional
 - Aggregates
2. Modelos de Distribución
 - Sharding
 - Master-Slave Replication
 - Peer-to-Peer Replication
 - Sharding + Replication
3. Consistencia
 - Update Consistency
 - Read Consistency
 - Teorema CAP

Consultas?

Feedback

<https://goo.gl/forms/NvrORS12kuuBitpE3>

