

Capítulo 2 - Modelos de Distribuidos de Datos / Teorema CAP

Jaime Veintimilla Reyes

November 24, 2020

- 1 Modelos de Distribución
- 2 Teorema CAP
- 3 Preguntas

Modelos de Distribución

- NoSQL DBs se ejecutan en un cluster de servidores.
 - + volúmen de datos
 - + complejidad en escalar verticalmente
 - NoSQL permite escalar horizontalmente.
 - La agregación es la unidad de distribución

Modelos de Distribución

- Dependiendo del modelo de distribución:
 - Manejar volúmenes de datos más grandes
 - Procesar mayor número de reads o writes
 - Mayor disponibilidad en caso de retardos en la red o particionamiento de red.
- Estos beneficios traen un costo (complejidad)
 - Se debe usar un cluster cuando los beneficios son evidentes.

Modelos de Distribución

- Replicación
 - Hace copias exactas de los datos en diferentes servidores
- Sharding
 - Diferente data en diferentes nodos
- Se puede usar cada una por separado o los dos a la vez

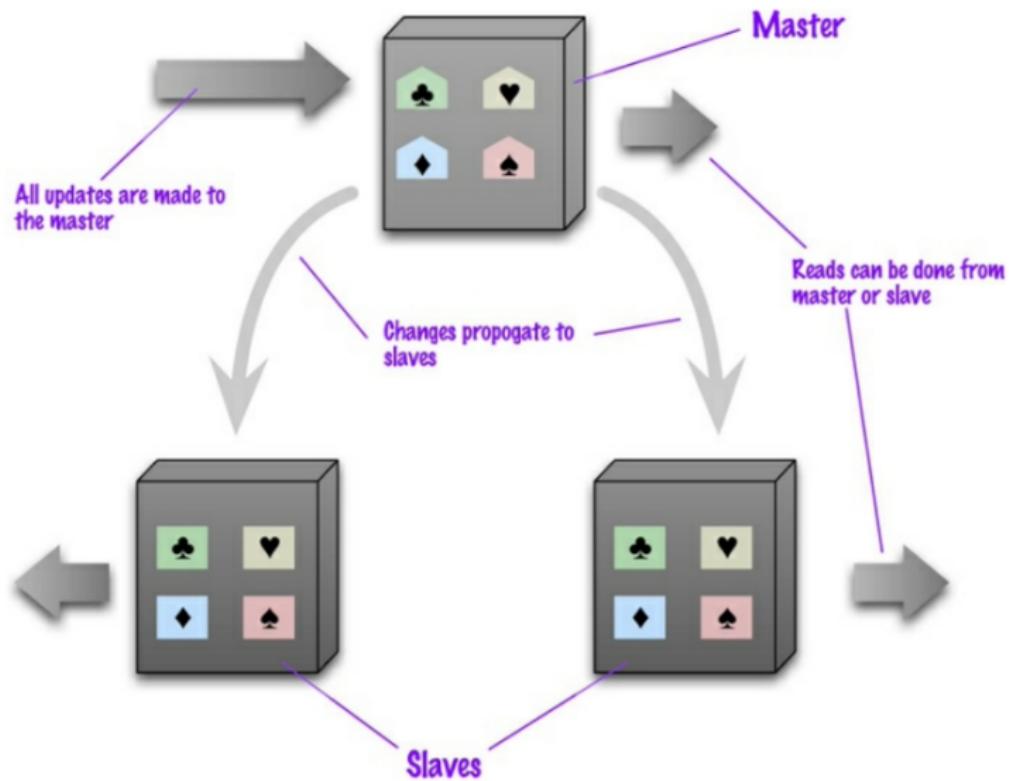
Modelos de Distribución

- Único Servidor
- Maestro-Esclavo (Replicación)
- Sharding
- Peer-to-Peer (Replicación)

- Sin distribución de datos
- Ejecutar la BD en una sola máquina que maneja todos las lecturas y escrituras
- Si bien las BDs NoSQL fueron diseñadas con la idea de ejecutarse en clusters también se lo puede hacer en un único servidor (BDs Grafos)

- Si la interacción con los datos es en su mayoría mediante el uso de agregaciones:
 - BDs Único Servidor Documentales o Clave-Valor
- No hay que dejarse sorprender por la palabra Big Data.
- Si se puede manejar los datos sin distribución, optar siempre por la opción más simple:
 - Único Servidor.

Maestro-Esclavo



- Más útil para datasets de lectura intensa y pocas escrituras
- Garantiza las lecturas (read resilience)
 - Si el master falla, los esclavos pueden seguir manejando peticiones de lectura (si la mayoría de accesos son de lectura)
- Si falla el master no hay escrituras
 - Hasta que el master se recupere Se designa un nuevo master

- Facilidad de reemplazar al master por un esclavo (incluso sin necesidad de escalar)
- Es como un único-servidor con un respaldo en caliente (hot backup).
 - Único servidor con mayor tolerancia a fallos
 - Para garantizar lecturas, se debe asegurar que los paths de lecturas y escrituras sean diferentes
- El master puede ser seleccionado manual o automáticamente (Leader Election).

- Replicación tiene ventajas pero tiene su inevitable lado negativo: inconsistencia
 - Diferentes clientes leyendo diferentes esclavos pueden ver datos diferentes
 - Algunos clientes no ven los últimos cambios del sistema
 - Si falla el master, todos los datos no sincronizados se pierden

Sharding

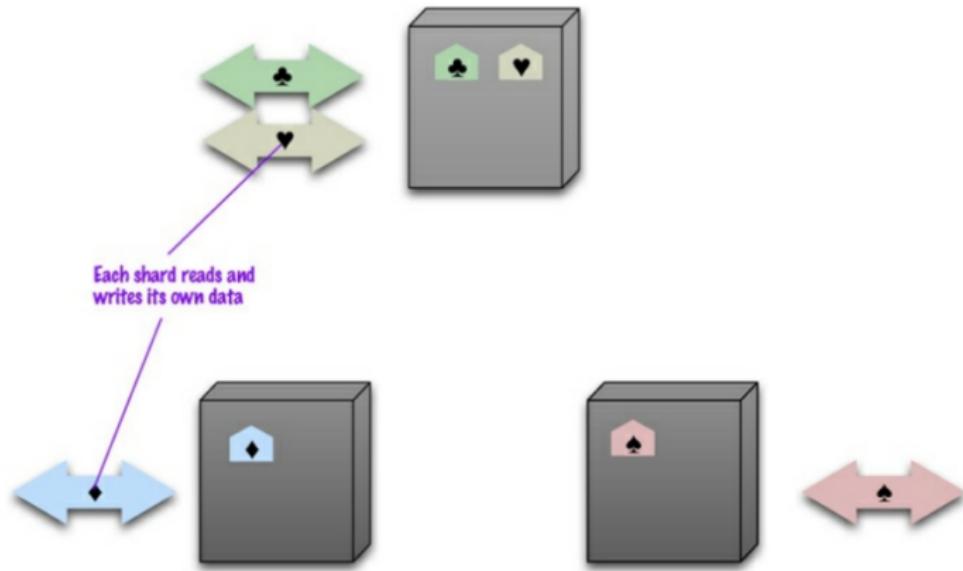


Figure 4.1. Sharding puts different data on separate nodes, each of which does its own reads and writes.

Sharding

- Normalmente una BD está ocupada porque diferentes usuarios están accediendo a diferentes partes de los datos
- Se puede soportar escalamiento horizontal poniendo diferentes partes de la data en diferente servidores (Sharding)
- En el caso ideal cada usuario se comunica con un solo servidor, obteniendo respuestas rápidas de ese servidor. (10 servidores, 10% c/u)

- Para lograr algo cercano al caso ideal
 - Los datos que son accedidos simultáneamente sean almacenados juntos en el mismo servidor
 - Agregaciones: unidad de distribución

Sharding

- Factores para mejorar el rendimiento
 - Si el acceso está basado en ubicación física, se puede poner los datos lo más cerca posible al lugar desde donde son accedidos.
 - Tratar de balancear la carga entre los servidores
 - Poner diferentes agregados juntos si se sabe que se van a leer en secuencia
- Sharding puede ser manual (lógica de la aplicación) o automática.

Sharding

- Permite escalar horizontalmente lectura y escrituras
 - Sharding no ayuda a mejorar la capacidad de recuperación ante fallos
 - Ante fallos solo los usuarios que acceden al shard sufrirán las consecuencias
- Pero no es bueno perder parte de los datos

Sharding

- Sharding es más facil de lograr con las agregaciones, pero no se debe tomar a la ligera
- Algunas bases son diseñadas para usar sharding, en este caso usar sharding desde el principio
- Otras bases no fueron diseñadas para eso, pero permiten pasar de único-servidor a sharding
- Saber identificar cuando se requiere usar sharding y usarlo mucho antes de que realmente se necesita

Peer-to-Peer

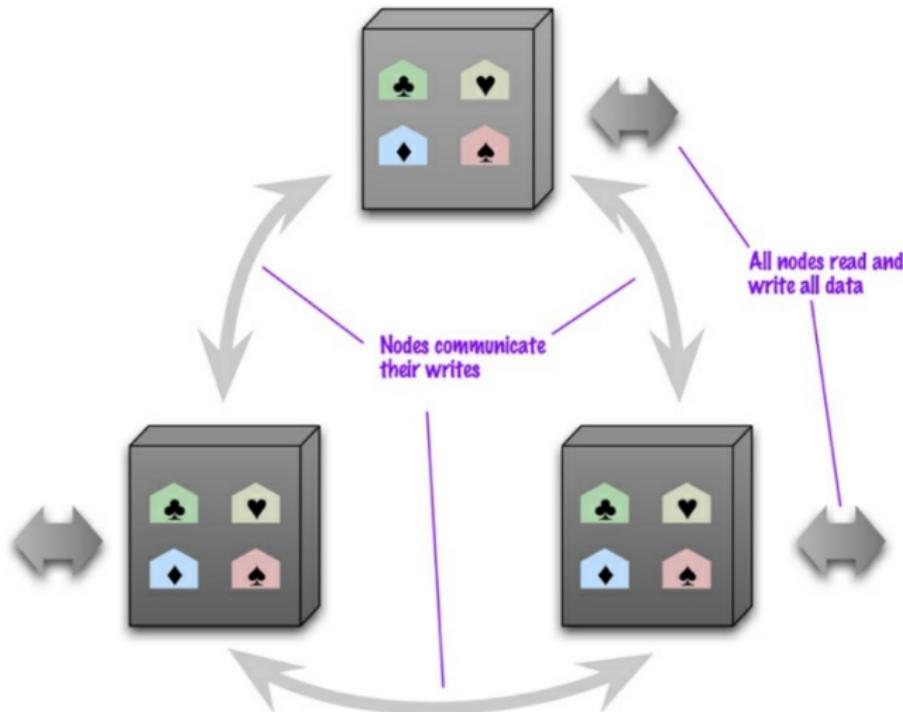


Figure 4.3. Peer-to-peer replication has all nodes applying reads and writes to all the data.

Combinar Sharding y Replicación

master for two shards



slave for two shards



master for one shard



master for one shard
and slave for a shard



slave for two shards



slave for one shard



Figure 4.4. Using master-slave replication together with sharding

Combinar Sharding y Replicación

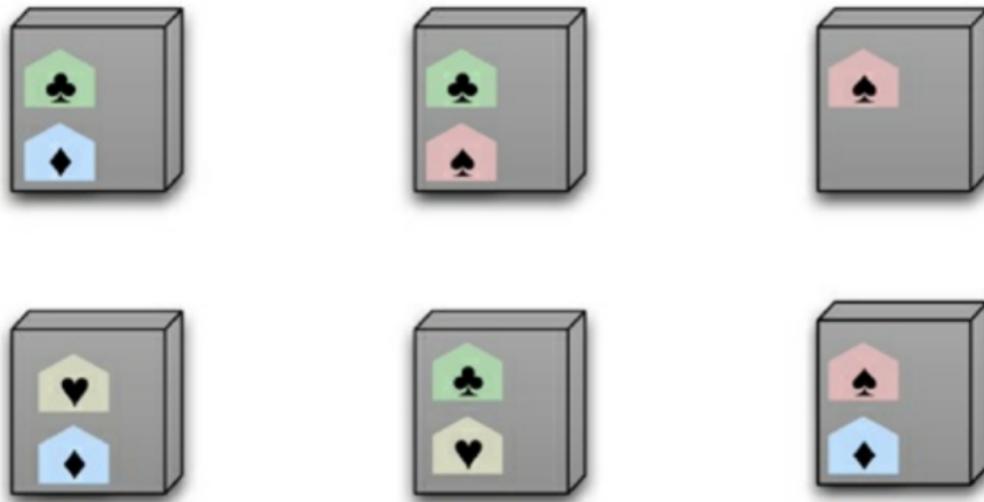


Figure 4.5. Using peer-to-peer replication together with sharding

- Todas las réplicas tienen el mismo peso, todas aceptan escrituras y la pérdida de una de ellas no afecta la lectura de los datos
- Complicación: Consistencia
 - Dos clientes tratando de escribir el mismo registro al mismo tiempo (conflicto de escritura)
 - Solución: Coordinar las escrituras.
 - No se necesita que todas se pongan de acuerdo pero si la mayoría

1 Modelos de Distribución

2 Teorema CAP

3 Preguntas

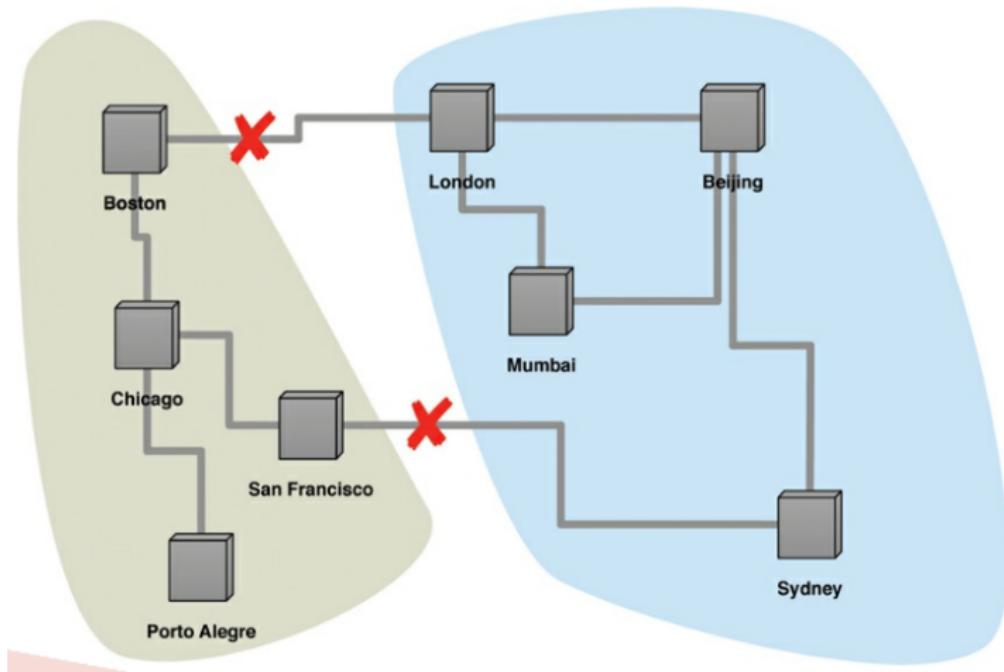
Teorema CAP

- Dr. Eric Brewer, 2000: Teorema CAP (1)
- Un sistema distribuido con datos compartidos puede tener a lo mucho dos de las siguientes propiedades:
 - Consistencia (Consistency)
 - Disponibilidad (Availability)
 - Tolerancia a Particionamientos de red (Partition tolerance).
- Gilbert and Lynch, 2002: Una definición más formal y pruebas (2)
 - (1) Harvest, Yield, and Scalable Tolerant Systems
 - (2) Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web

Teorema CAP

- **Consistencia:** Despues de una escritura exitosa, las lecturas posteriores siempre incluiran dicha escritura.
- **Disponibilidad:** Siempre se puede leer y escribir en el sistema. Toda petición (lectura/escritura) recibida por un nodo que se encuentra operativo debe proporcionar una respuesta.
- Tolerancia a **Particionamientos** de Red: La red podrá perder arbitrariamente varios mensajes enviados de un nodo a otro en presencia de particiones (Gilbert and Lynch).

Teorema CAP



Teorema CAP

- Particionamiento: No solo paquetes perdidos
 - Servidor caido (Una partición)
 - Todos los paquetes enviados hacia él se pierden.
 - La falla más sencilla de manejar (Hay seguridad que el servidor no da respuestas erroneas)

Teorema CAP

- CA Systems
 - Consistencia, Disponibilidad – No Particionamiento
 - Único Servidor: Sistema Monolítico (No red, no partición y CA garantizadas)
 - Clientes conectados a ese servidor?
- Sistemas Distribuidos que sea CA (Multi Nodo)
 - Los mensajes en la red nunca se pierden o retrasan
 - Los servidores nunca se caen
- Sistemas así **NO EXISTEN**

Teorema CAP

- Ante la presencia particiones de red (Errores), qué se sacrificará? Consistencia o Disponibilidad?
 - La posibilidad de particiones siempre está presente
- La decisión no es mutuamente exclusiva
 - El sistema no será completamente consistente ni completamente disponible
 - Combinación de las dos que se adapta a las necesidades.

Teorema CAP

- Consistencia sobre Disponibilidad
 - Garantiza la atomicidad de lecturas y escrituras rechazando algunas peticiones.
 - Bajar el sistema por completo
 - Rechazar escrituras (Two-Phase Commit)
 - Lecturas y escrituras en las particiones cuyo master esta en esa partición.

Teorema CAP

- Disponibilidad sobre Consistencia
 - Responderá a todas las peticiones
 - Lecturas desactualizadas
 - Aceptando conflictos de escritura → Mecanismos para resolver inconsistencias (vector clocks)
 - Hay varios sistemas donde es posible manejar resolución conflictos y en los cuales lecturas desactualizadas son aceptables.

Teorema CAP

- Disponibilidad o Consistencia **nunca** ambas
 - Sistema que dice ser CA : Servidores A, B y C
 - {A,B} {C}
 - Petición de escritura a C para actualizar un dato
 - 1. Aceptar la escritura sabiendo que ni A ni B sabrán de ella (hasta que la red vuelva a la normalidad)
 - 2. Rechazar la escritura, sabiendo que el cliente no podrá contactar A o B (hasta que la red vuelva a la normalidad)
 - Se debe escoger Disponibilidad (opción 1) o Consistencia (opción 2)

Teorema CAP

- En lugar de pensar cual de las dos propiedades nuestro sistema requiere (CA), pensar hasta donde puedo sacrificar cada una, antes que mi sistema empiece a funcionar mal.
- No importa lo que hagamos siempre habrán fallas en el sistema.
 - Dejar de responder peticiones (lectura/escritura)
 - Dar respuestas basadas en información incompleta

1 Modelos de Distribución

2 Teorema CAP

3 Preguntas

Preguntas

- ¿Alguna Pregunta?.



Capítulo 2 - Modelos de Datos para Big Data

Jaime Veintimilla Reyes

November 10, 2020

- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas
- 9 Preguntas

Data Model

- El modelo con el cual la base de datos organiza su información
- Modelo de Datos Relacional
 - Conjunto de tablas
 - Filas que representan alguna entidad
 - Columnas: describen a la entidad (cada columna un solo valor).
 - cada columna puede apuntar a otra fila en la misma o en otra tabla, lo que representa una relación entre esas dos entidades

Modelos de Datos NoSQL

- Se aleja del Modelo Relacional
 - Cada solución NoSQL tiene su propio modelo
 - Existen 4 categorías
 - ① Clave - Valor
 - ② Documentales
 - ③ Orientadas a columnas
 - ④ Grafos
- } Orientadas a la Agregación

Agregados

- Modelo Relacional
 - Tuplas (Estructura de datos limitada)
 - No hay como anidar una tupla dentro de otra
 - Peor aún poner una lista de valores o tuplas dentro de otra
 - Esta simplicidad permite realizar todos los tipos de operaciones conocidos sobre las tuplas.

Agregados

- Normalmente se necesita trabajar con datos en unidades que tienen una estructura más compleja que un conjunto de tuplas.
- Se puede pensar en un registro complejo que permita anidar listas y otras estructuras dentro de él.
- DBs Clave-Valor, Documentales y Orientadas a Columnas usan esta clase de “registros complejos”

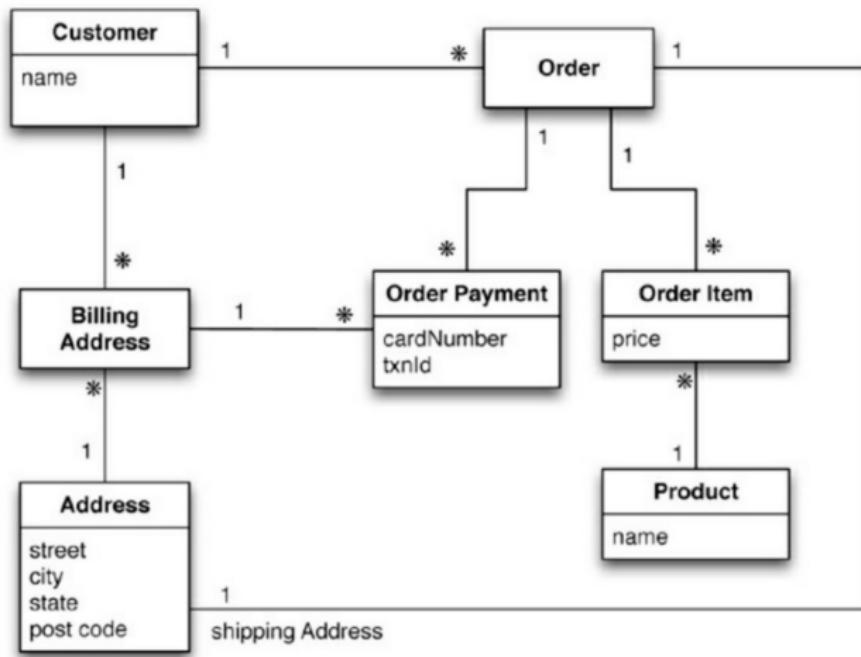
Agregados

- Un agregado es una colección de datos relacionados que nosotros queremos tratar como una unidad.
- **Unidad** para la manipulación de datos y para el manejo de la consistencia.

Agregados

- Actualizar agregados con operaciones atómicas y comunicarse con la base de datos en términos de agregados.
- Facilitan la ejecución en un cluster de computadores, ya que un agregado sería la unidad para replicación y “sharding” o fragmentación
- Facilitan el trabajo a los desarrolladores

Ejemplo: Relaciones y Agregados



Ejemplo: Relaciones y Agregados

Customer	
Id	Name
1	Martin

Orders		
Id	CustomerId	ShippingAddressId
99	1	77

Product	
Id	Name
27	NoSQL Distilled

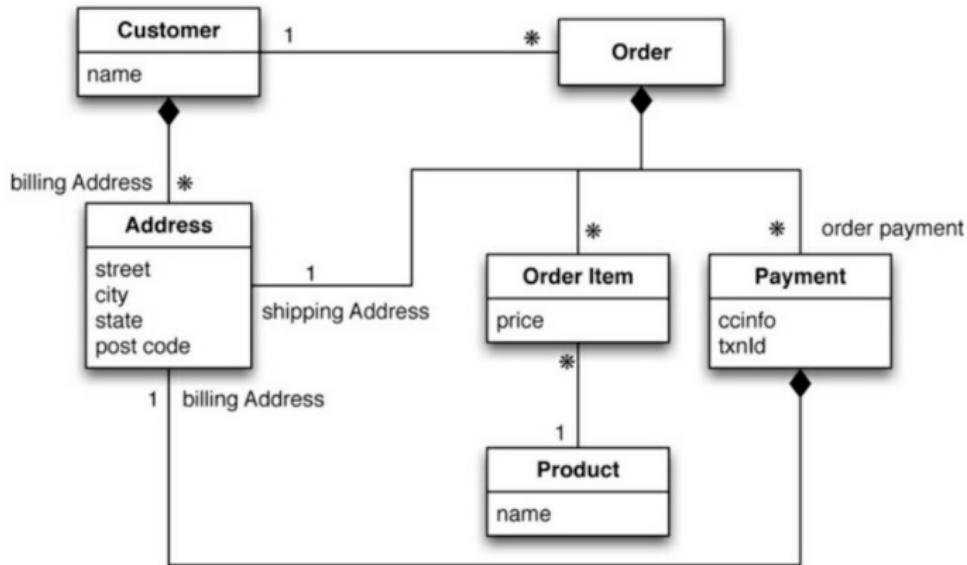
BillingAddress		
Id	CustomerId	AddressId
55	1	77

OrderItem			
Id	OrderId	ProductId	Price
100	99	27	32.45

Address	
Id	City
77	Chicago

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

Ejemplo: Relaciones y Agregados (Aggregate data model)



Ejemplo: Relaciones y Agregado

```
// 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"}
    }
  ],
}
```

Ejemplo: Relaciones y Agregado

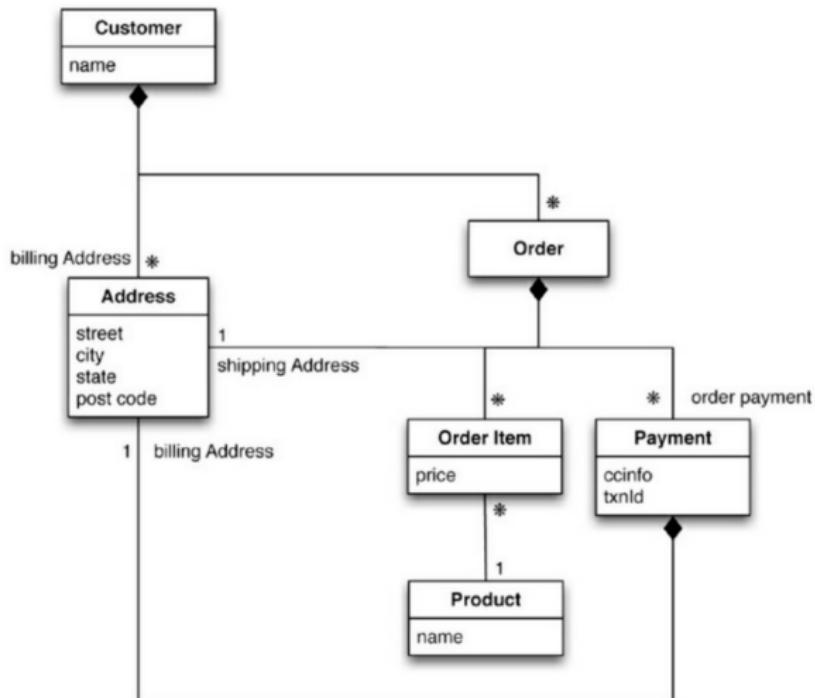


Figure 2.4. Embed all the objects for customer and the customer's orders

Ejemplo: Relaciones y Agregado

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

- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas
- 9 Preguntas

Consecuencias de la Agregación

- El modelo relacional captura los datos y sus relaciones muy bien, pero sin ninguna noción de una unidad de agregación.
- Las técnicas propuestas por las tecnologías NoSQL nos permiten crear agregaciones o estructuras compuestas. Pero....
 - Los que modelan no proporcionan ninguna información para distinguir una agregación de otra
 - Si la hay, esta varía dependiendo de cada situación

Consecuencias de la Agregación

- Cuando se trabaja con BDs orientadas a agregación, lo único claro es que la agregación es la unidad de interacción con la base de datos.
- Pero todo depende de como los datos son usados dentro de la aplicación. (Esto a veces se sale de las manos de la modelación)

Consecuencias de la Agregación

- RDBMS y de Grafos (Aggregate-ignorant) → (No conocen la agregación)
 - NO es el fin del mundo
- Es difícil definir los límites de las agregaciones correctamente, especialmente si la misma data es usada en diferentes contextos
 - Por ejemplo: Agregación Orden o Producto?

Consecuencias de la Agregación

- La agregación facilita la ejecución en clusters.
- Al definir agregaciones le decimos a la base de datos que bits van a ser manipulados juntos
- Y por lo tanto deberían residir en el mismo nodo.
- RDBMS → ACID (Tablas, Filas)
- NoSQL → BASE (Una agregación a la vez)
 - Varias Agregaciones de manera atómica

- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas
- 9 Preguntas

DBs Clave-Valor y Documentales

- Ambas son **fuertemente** orientadas a agregación
- Ambas bases consisten de varias agregaciones donde cada agregación tiene una clave o ID que es usada para recuperar los datos

DBs Clave-Valor y Documentales

- Diferencia
 - Clave-Valor:
 - La agregación es desconocida para la base de datos
 - Blob (Binary Large Objects) de bits sin ningún significado
 - Se puede almacenar lo que sea. (Límite en espacio)

DBs Clave-Valor y Documentales

- Documentales:
 - Es capáz de distinguir la estructura dentro de las agregaciones.
 - Ciertos límites en lo que se puede almacenar
 - Estructuras y tipos pre-definidos
 - Flexibilidad en el acceso

DBs Clave-Valor y Documentales

- Diferencia
 - Clave-Valor:
 - Permite buscar agregaciones por Clave o ID
 - Documentales:
 - Queries basados en la estructura interna de la agregación.
 - Puede ser la clave, pero por lo general es otro valor.

- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas
- 9 Preguntas

- RDBMS

- Filas como unidad de almacenamiento
 - Ayuda a mejorar el rendimiento en las escrituras
- Hay muchos escenarios donde las escrituras son esporádicas:
 - Se necesita leer algunas columnas de varias filas a la vez.
 - Grupos de columnas para todas las filas como unidad de almacenamiento.

BDs Orientadas a Columnas

De-Moneda	A-Moneda	Precio venta	Precio Compra	Banco	Fecha
USD	EURO	1.2300	1.2850	Austro	18/09/2014
USD	PLN	1.3400	1.3050	Austro	18/09/2014
USD	AZN	1.2700	1.5150	Austro	18/09/2014
USD	EGP	1.7600	1.5800	Austro	18/09/2014
USD	EEK	1.4000	1.4213	Austro	18/09/2014
USD	FJD	1.4425	1.4553	Machala	18/09/2014
USD	GIP	1.4681	1.4929	Machala	19/09/2014
USD	DKK	1.5177	1.4874	Machala	19/09/2014
EURO	USD	1.4571	1.4642	Machala	19/09/2014
EURO	FJD	1.4713	1.4749	Pichincha	19/09/2014
EURO	GIP	1.4785	1.4799	Pichincha	19/09/2014
EURO	EEK	1.4812	1.4766	Pichincha	19/09/2014

1B, 100Bytes = 100GB x (3/6); 100MB/sec = 500sec

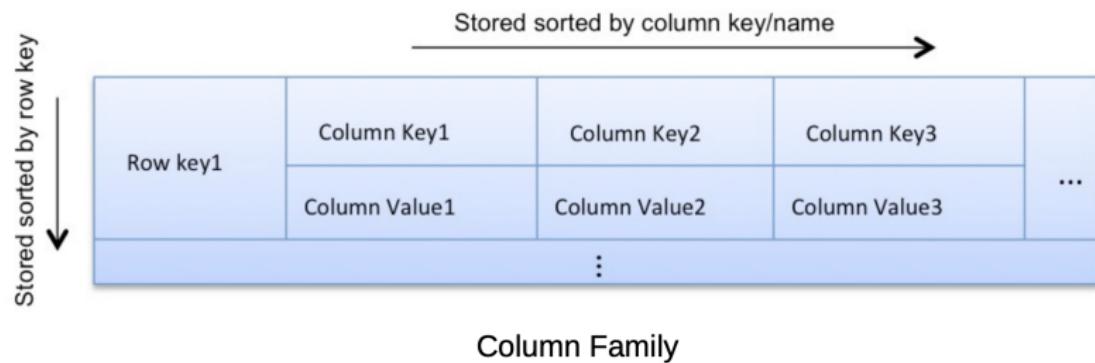
BDs Orientadas a Columnas

- Almacenamiento más eficiente debido a la compresión de datos
 - USDx8, EUROx4
- Funciones de Agregación
 - Max, Min, Sum, Avg

De-Moneda
USD
EURO
EURO
EURO
EURO

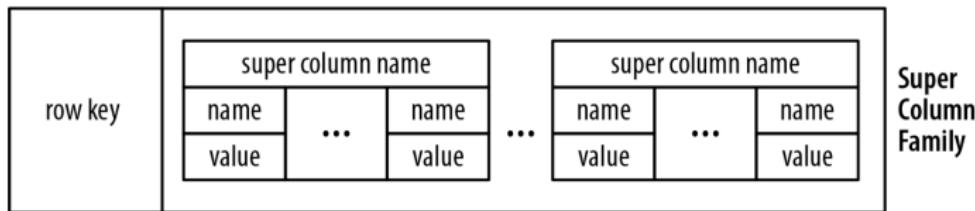
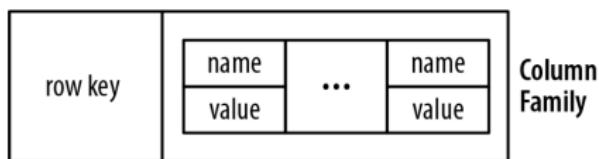
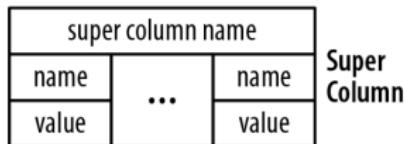
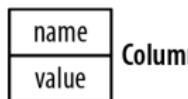
BDs Orientadas a Columnas

- Estructura de agregación a dos niveles



Map<RowKey, SortedMap<ColumnKey, ColumnValue>>

BDs Orientadas a Columnas



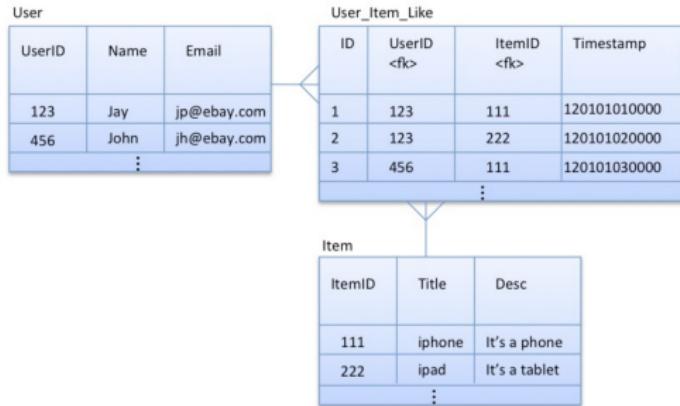
- Modelar de acuerdo a los patrones de búsqueda
 - Pensar en patrones de búsqueda por adelantado
 - Identificar los queries más frecuentes
 - Detectar cuales pueden ser lentos
 - Asegurarse que el modelo satisface los queries más frecuentes y críticos
 - Recordar que una “Familia de Columnas” es un SortedMap
 - Búsqueda, Ordenamiento, Agrupamiento, Filtrado, Agregaciones, etc.

BDs Orientadas a Columnas

- De-Normalizar y Duplicar por lecturas eficientes
 - No denormalizar si no se necesita, encontrar un balance
- Normalización
 - Pros: No hay duplicación de información, menos errores por modificación de datos, conceptualmente mas claro, etc...
 - Cons: Queries demasiado lentos si hay varios joins y demasiados datos (BigData)

BDs Orientadas a Columnas

- Likes: Relación entre usuarios e items



- Usuarios x UserID
- Items x ID
- Todos los items que le gustan a un usuario en particular
- Todos los usuarios a los que les gusta un item en particular

BDs Orientadas a Columnas

- Replica del modelo relacional

User		
	Name	Email
123	Jay	jp@ebay.com
	:	

Item		
	Title	Desc
111	iphone	It's a phone
	:	

User_Item_Like		
	UserID	ItemID
1	123	111
	:	

- Usuarios x UserID
- Items x ID

BDs Orientadas a Columnas

- Entidades Normalizadas con índices personalizados

User	Name	Email
123	Jay	jp@ebay.com
⋮	⋮	⋮

Item	Title	Desc
111	iphone	It's a phone
⋮	⋮	⋮

User_By_Item	123	456	⋮
111	null	null	⋮
⋮	⋮	⋮	⋮

Item_By_User	111	222	⋮
123	null	null	⋮
⋮	⋮	⋮	⋮

- Usuarios x UserID
- Items x ID
- Todos los items que le gustan a un usuario en particular
- Todos los usuarios a los que les gusta un item en particular

- Agregar Título de Item
- Agregar Nombre de Usuario

BDs Orientadas a Columnas

- Entidades Normalizadas con de-normalización en índices personalizados

User		
	Name	Email
123	Jay	jp@ebay.com
	⋮	

Item		
	Title	Desc
111	iphone	It's a phone
	⋮	

User_By_Item			
	123	456	...
111	null	null	⋮
	⋮		

Item_By_User			
	111	222	...
123	null	null	⋮
	⋮		

BDs Orientadas a Columnas

User		Item	
123	Name	Email	
	Jay	jp@ebay.com	
...		...	

User_By_Item			
111	120101010000 123	120101030000 456	...
	Jay	John	
...		...	

Mejor Modelo

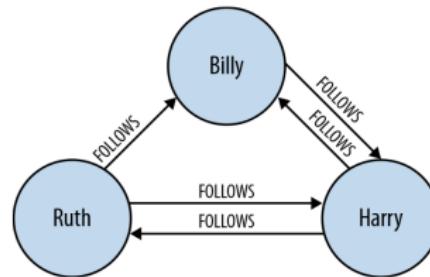
Item_By_User			
123	120101010000 111	120101020000 222	...
	iphone	ipad	
...		...	

- Un agregado es una colección de datos relacionados que queremos tratar como unidad de almacenamiento
- BDs Clave-Valor, Documentales, Orientadas a Columnas son orientadas a la agregación
- Agregados permiten la ejecución en clusters de servidores
- Las BDs orientadas a agregación funcionan muy bien cuando las interacciones con los datos son hechas con la misma agregación.
- Aggregate-Ignorant BDs son mejores cuando las interacciones con la base usan datos que están organizados de múltiples formas.

- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas
- 9 Preguntas

Bases de datos de Grafos

- Cómo manejar pequeños registros que tienen conexiones complejas entre sí?
- Modelo de datos de Grafos
 - Nodos y Relaciones Nodos tienen propiedades
 - Las relaciones son nombradas y directas y siempre tienen principio y fin
 - Las relaciones tambien pueden tener propiedades



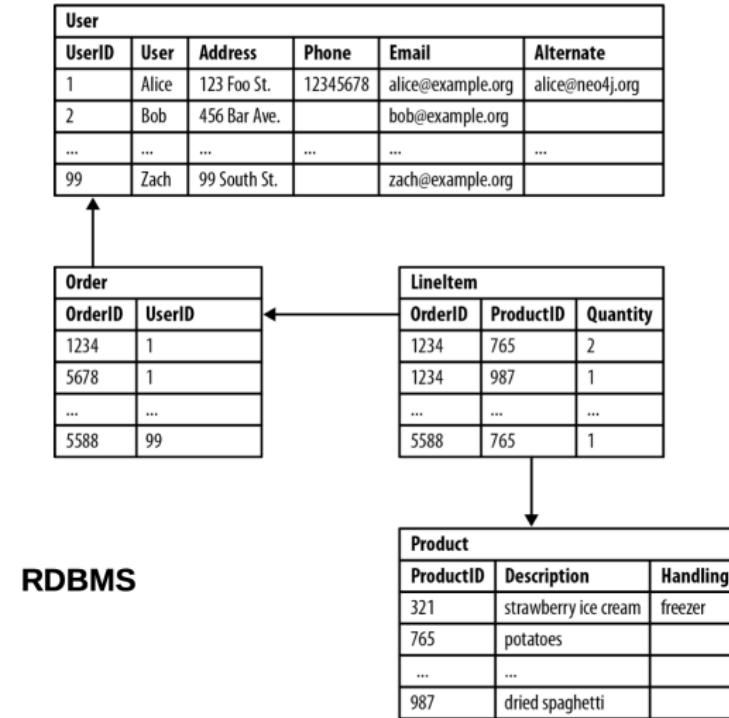
Bases de datos de Grafos

- Base de datos cuya estructura de almacenamiento sigue el modelo de datos de grafos.
- Sirven para capturar datos que consisten de relaciones bastante complejas:
 - Redes sociales
 - Preferencias de productos
 - Control de Tráfico
 - Detección de fraude
 - Etc...

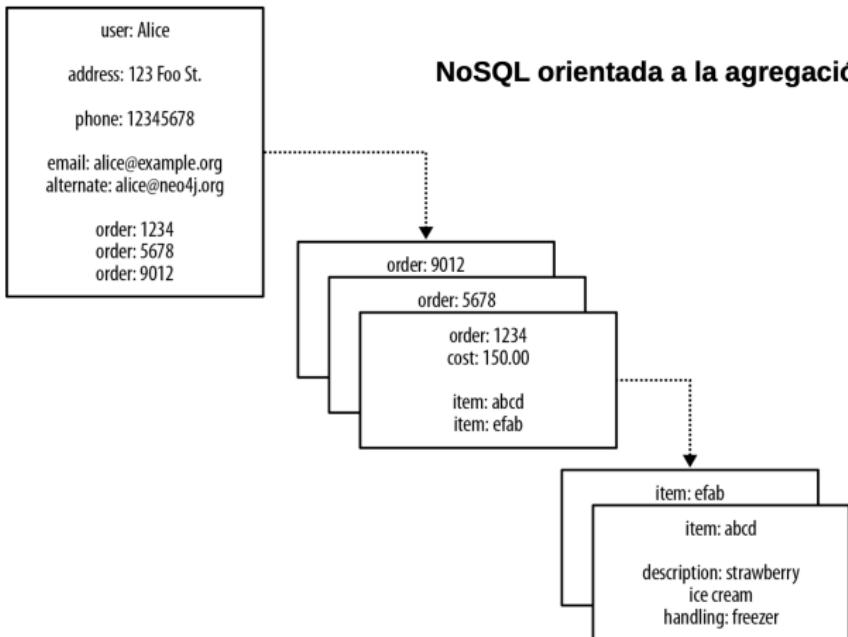
Bases de datos de Grafos

- Ejemplos:
 - Deme todos los restaurantes que han sido recomendados por mis amigos
 - Cúal es el camino más corto para ir de Cuenca a Machala?
 - Cuáles son los productos que a John y Ana les gustan.
- Por qué son útiles
 - RDBMS y NoSQL orientas a agragación no permiten capturar relaciones complejas en sus modelos de datos.

Bases de datos de Grafos

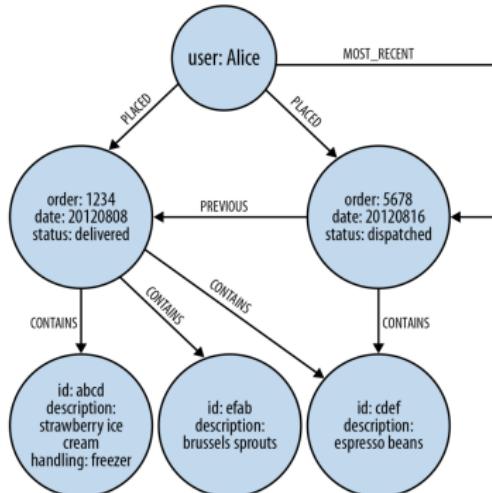


Bases de datos de Grafos



Bases de datos de Grafos

- Recomendaciones
 - Quien compra helado de fresa también compra Café
- Enriquecer nuestros datos
 - Unirlo a grafos de otros dominios (Geo y Social)
- Todos los sabores de helado que le gusta a la gente que vive cerca de Juan y a las que les gusta el espresso pero no les gusta las coles de bruselas



- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas
- 9 Preguntas

Bases de datos sin esquema

- RDBMS

- Se debe definir un esquema de antemano
- Una estructura definida que nos diga que tablas y columnas existen y que tipo de datos va a tener cada columna.

Bases de datos sin esquema

- NoSQL (el almacenamiento es más casual)
 - Clave-Valor: permite almacenar cualquier tipo de datos
 - Documentales: No hay restricciones en la estructura de los documentos
 - Columnas: permiten almacenar cualquier tipo de datos en cualquier columna que se desee.
 - Grafos: permite agregar libremente nodos y aristas y propiedades a los nodos y las aristas.

Bases de datos sin esquema

- Con un esquema se tiene que definir por adelantado lo que se necesita almacenar.
- Sin un esquema que nos ate, se puede almacenar fácilmente lo que sea que se necesite.
- Esto permite cambiar fácilmente nuestra base de datos a medida que se conoce más el proyecto.
- Se puede fácilmente agregar nuevas cosas a medida que se las va descubriendo
- Si no se necesita algo, simplemente se deja de almacenarlo.

Bases de datos sin esquema

- Así como permitir cambios, una BD sin esquema permite manejar fácilmente datos no-uniformes.
 - Datos donde cada registro tiene diferente campos.
- Un esquema pone a todas las filas en una “camisa de fuerza”.
- **Qué pasa si hay diferentes tipos de datos en cada fila?**
 - Se termina con columnas en valor null
 - O columnas sin sentido (Columna_4)
- NoSQL evita esto permitiendo que cada registro tenga lo que necesita.
- Ni más ni menos

Bases de datos sin esquema

- Las BDs sin esquema pueden evitarnos muchos problemas que existen con las bases que tienen un esquema fijo
- Sin embargo éstas traen algunos problemas consigo.
 - Los programas necesitan saber que la dirección de facturación es llamada **direccionDeFacturacion** y no **direccionParaFacturacion**
 - Y que la cantidad es un entero **5** y no **cinco**
- El hecho es que cuando escribimos un programa que accede a ciertos datos, éste siempre depende implícitamente de un esquema.

Bases de datos sin esquema

- Un programa asume
 - Que ciertos nombres de campos se encuentran presentes y que estos hacen referencia a datos con cierto significado
 - Algo acerca del tipo de datos almacenado en ese campo.
- Los programas no son humanos
 - No deducir que Nro = Número, a menos que se los programe para hacer eso.
- Así tengamos una BD sin esquema, siempre está presente esquema implícito.
- **Esquema Implícito** es un conjunto de suposiciones acerca de la estructura de los datos en el código que manipula esos datos.

Bases de datos sin esquema

- Esquema Implícito presenta algunos problemas.
 - Para entender que datos se estan manejando hay que escarbar dentro del código.
 - Si el código es bien estructurado se puede definir fácilmente el esquema, pero nadie nos garantiza eso
 - La BD no puede usar el esquema para recuperar o almacenar los datos más eficientemente.
 - La BD no puede validar los datos para evitar inconsistencias.

Bases de datos sin esquema

- Las BDs sin esquema mueven el esquema al código de la aplicación
 - Qué pasa si diferentes aplicaciones hechas por diferentes personas acceden a la misma base de datos?
- Para reducir los problemas
 - Encapsular la interacción con la BD dentro de una sola aplicación que se integra con otras usando servicios web.
 - Delimitar diferentes partes de un agregado para las diferentes aplicaciones
 - Diferentes secciones de un documento
 - Diferentes “Familia de columnas”
- Usar una base de datos sin esquema no es la panacea
- No uniformidad en los datos es una buena razón para usar una BD sin esquema

- 1 Data Model
- 2 Agregación
- 3 Consecuencias de la Agregación
- 4 DBs Clave-Valor y Documentales
- 5 BDs Orientadas a Columnas
- 6 Bases de datos de Grafos
- 7 Bases de datos sin esquema
- 8 Vistas materializadas**
- 9 Preguntas

Vistas materializadas

- Ventajas de los modelos orientados a agregación
 - Si se quiere acceder a **órdenes**, es conveniente tener todos los datos de una orden en un agregado que será almacenado y leido como una unidad
- Hay desventajas?
 - Qué pasa si queremos saber cuanto se ha vendido un producto en las últimas semanas?
 - La agregación juega en contra. Hay que obligatoriamente leer cada orden para responder a esa pregunta

Vistas materializadas

- RDBMS tienen la ventaja que permiten acceder a los datos de diferente manera
- Además proporcionan un mecanismo que permite observar los datos de manera distinta a la que están almacenados: **Vistas**
- Una vista es cómo una tabla pero es calculada a partir de las tablas base
- Cuando se accede a la vista la BD calcula los datos a mostrarse en la vista

Vistas materializadas

- Hay algunas vistas que son costosas de calcular
- Solución: **Vistas Materializadas (VM)**
 - Son vistas precalculadas y son almacenadas en caché en disco.
 - Son efectivas para datos que son de lectura-intensa pero que pueden estar algo desactualizados.
- NoSQL no tiene vistas materializadas pero pueden tener queries precalculados en una caché en disco.

Vistas materializadas

- Las VM son un aspecto central en la BDs orientadas a la agregación, tal vez más que las RDBMS
 - Las aplicaciones tienen que ejecutar queries que no se adaptan al modelo de agregación
 - Es usual crear vistas materializadas usando **Map-Reduce**

Vistas materializadas

- Dos estrategias para crear vistas materializadas
 - Actualizar la VM al mismo tiempo que se actualiza los datos base.
 - Ejecutar procesos en batch que actualizan la VM en intervalos regulares
- Se puede calcular externamente, leyendo los datos calculando la vista y grabando de nuevo en la base de datos
- Es importante conocer el modelo de negocios para saber cuan desactualizada puede estar una VM

1 Data Model

2 Agregación

3 Consecuencias de la Agregación

4 DBs Clave-Valor y Documentales

5 BDs Orientadas a Columnas

6 Bases de datos de Grafos

7 Bases de datos sin esquema

8 Vistas materializadas

9 Preguntas

Preguntas

- ¿Alguna Pregunta?.

