Resumen Cap 2

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

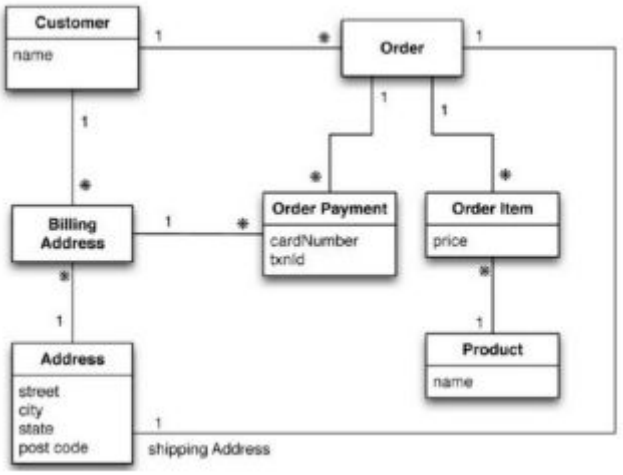
Se aleja del Modelo Relacional
Cada solución NoSQL tiene su propio modelo
Existen 4 categorías

1. Clave - Valor = Orientadas a la Agregación
2. Documentales = Orientadas a la Agregación
3. Orientadas a columnas = Orientadas a la Agregación
4. Grafos

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 conocidas sobre las tuplas.
- 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”
- 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.
- 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



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

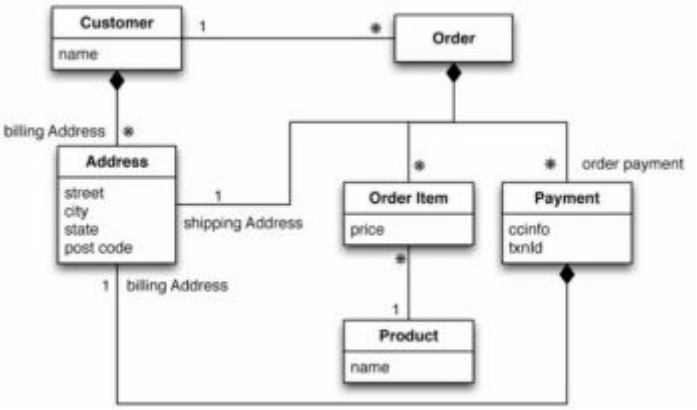
Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abeli079rft



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

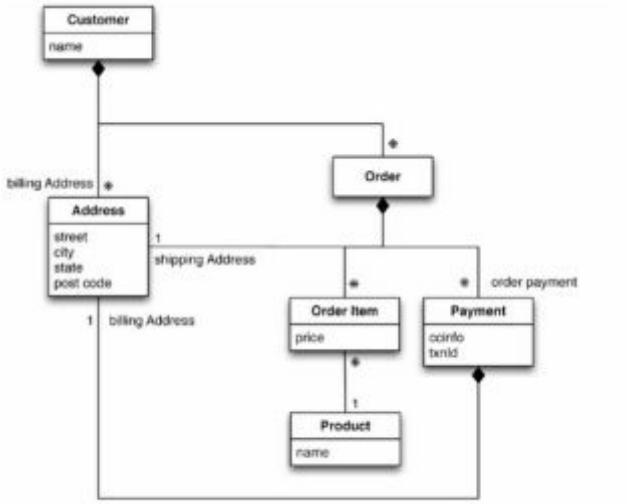


Figure 2.4. Embed all the objects for customer and the customer's orders

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

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 varıa dependiendo de cada situación

- 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 cómo los datos son usados dentro de la aplicación. (Esto a veces se sale de las manos de la modelación)
- RDBMS y de Grafos (Aggregate-ignorant) → (No conocen la agregación)
    - NO es el fin del mundo
- Es difícil definir los lımites de las agregaciones correctamente, especialmente si la misma data es usada en diferentes contextos
    - Por ejemplo: Agregación Orden o Producto?
- 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

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
- 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. (Lımite en espacio)
- Documentales:
    - Es capaz de distinguir la estructura dentro de las agregaciones.
    - Ciertos lımites en lo que se puede almacenar
        - Estructuras y tipos predefinidos
        - Flexibilidad en el acceso
- 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.

BDs Orientadas a Columnas

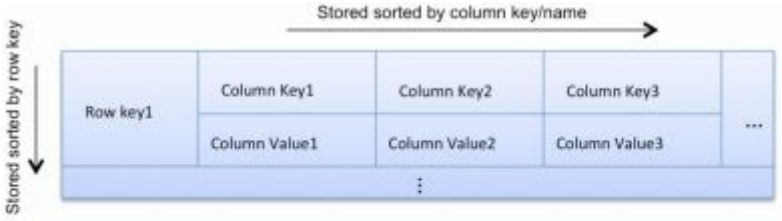
- RDBMS
    - Filas como unidad de almacenamiento
        - Ayuda a mejorar el rendimiento en las escrituras
    - Hay muchos escenarios donde las escrituras son esporádicas:
        - Se necesita leer algunas columnas de varias filas a la vez.
        - Grupos de columnas para todas las filas como unidad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

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

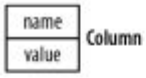
De-Moneda
USD
USD
USD
USD
USD
USD
USD
USD
EURO
EURO
EURO
EURO

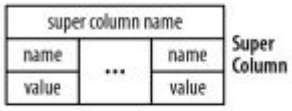
- Estructura de agregación a dos niveles

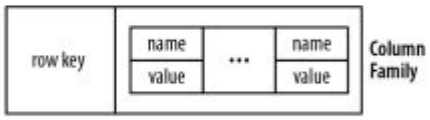


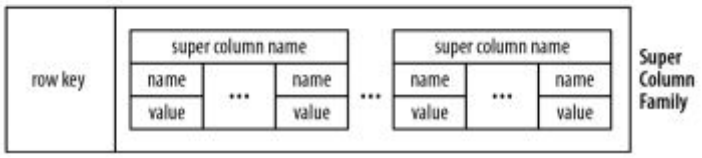
Column Family

Map<RowKey, **SortedMap**<ColumnKey, ColumnValue>>

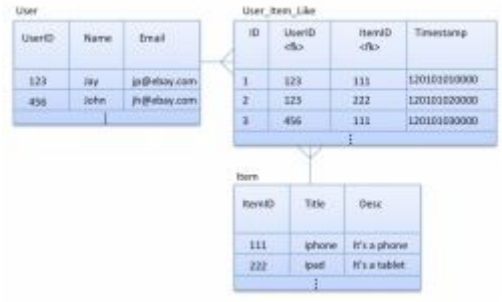






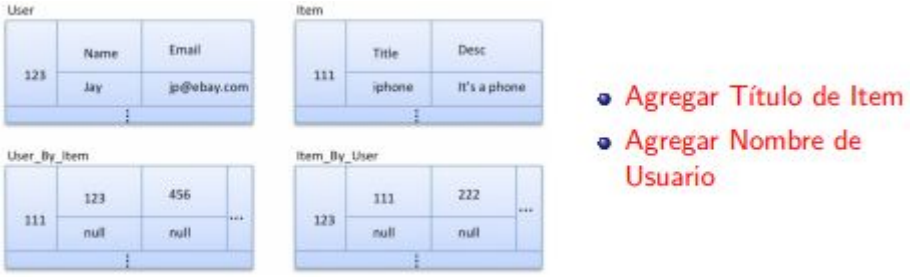


- Modelar de acuerdo a los patrones de búsqueda
  - Pensar en patrones de búsqueda por adelantado
    - Identificar los queries mas frecuentes
    - Detectar cuales pueden ser lentos
    - Asegurarse que el modelo satisface los queries más frecuentes y crıticos
  - Recordar que una “Familia de Columnas” es un SortedMap
    - Búsqueda, Ordenamiento, Agrupamiento, Filtrado, Agregaciones, etc.
- De-Normalizar y Duplicar por lecturas eficientes
  - No de normalizar 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 (Big Data)
- Likes: Relación entre usuarios e ítems

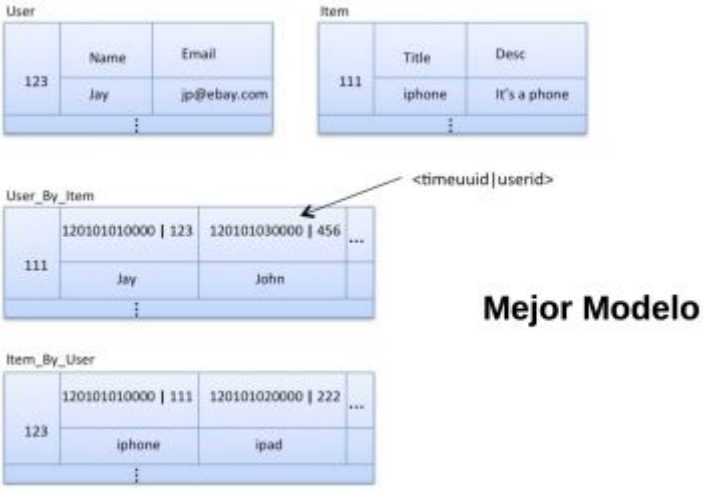


- Usuarios x UserID
- Items x ID
- Todos los items que le gustan a un usuario en particular
- Todos los usuarios a los que les gusta un ítem en particular
- Réplica del modelo relacional

- 

    - Usuarios x UserID
    - Items x ID
- Entidades Normalizadas con ındices personalizados


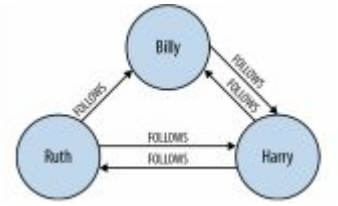
    - Usuarios x UserID
    - Items x ID
    - Todos los items que le gustan a un usuario en particular
    - Todos los usuarios a los que les gusta un ítem en particular
- Entidades Normalizadas con de-normalización en ındices personalizados
    - 

    - 

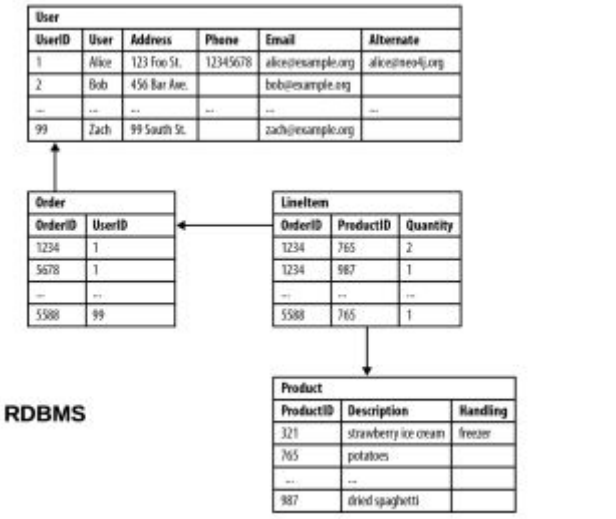
- **Resumen**

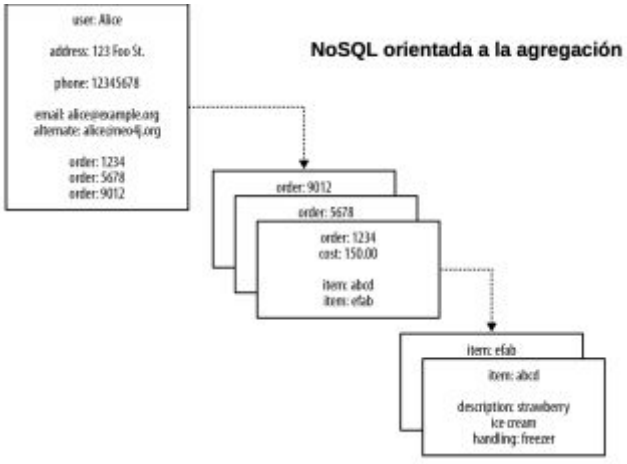
    - Un agregado es una colección de datos relacionados que queremos tratar como unidad de almacenamiento
    - BDs Clave-Valor, Documentales, Orientadas a Columnas son orientadas a la agregación
    - Agregados permiten la ejecución en clusters de servidores
    - Las BDs orientadas a agregación funcionan 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.

Bases de datos de Grafos

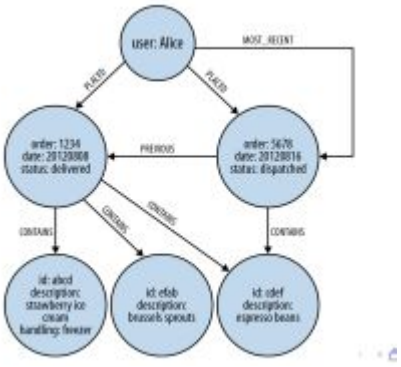
- Cómo manejar pequẽnos registros que tienen conexiones complejas entre śı?
- Modelo de datos de Grafos
    - Nodos y Relaciones Nodos tienen propiedades
    - Las relaciones son nombradas y directas y siempre tienen principio y fin
    - Las relaciones también pueden tener propiedades
- 

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



- 

- Recomendaciones
  - Quien compra helado de fresa también compra Cafe
- 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

- 

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.

- 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.
- 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.
- Así como permitir cambios, una BD sin esquema permite manejar fácilmente datos no-uniformes.
    - Datos donde cada registro tiene diferentes campos.
- Un esquema pone a todas las filas en una “camisa de fuerza”.
- Que 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
- Las BDs sin esquema pueden evitarnos muchos problemas que existen con las bases que tienen un esquema fijo
- Sin embargo,estas 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 cuanto escribimos un programa que accede a ciertos datos, este siempre depende implícitamente de un esquema.
- Un programa asume
    - Que ciertos nombres de campos se encuentran presentes y que estos hacen a 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 = Numero, a menos que se los programe para hacer eso.
- Así tengamos una BD sin esquema, siempre está presente el esquema implícito.
- **Esquema Implıcito** es un conjunto de suposiciones acerca de la estructura de los datos en el código que manipula esos datos.
- Esquema Implıcito presenta algunos problemas.
    - Para entender que datos se están 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.

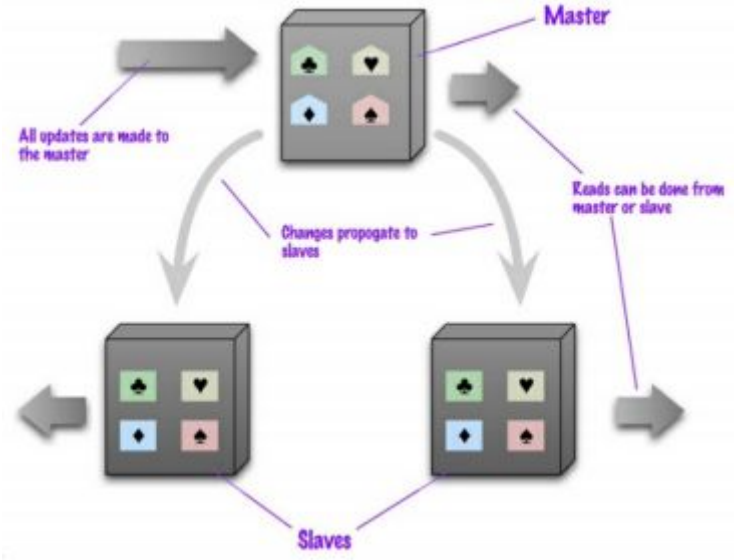
- Las BDs sin esquema mueven el esquema al código de la aplicación
    - Que pasa si diferentes aplicaciones hechas por diferentes personas acceden a la misma base de datos?
- Para reducir los problemas
    - Encapsular la interacción con la BD dentro de una sola aplicación que se integra con otras usando 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

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 leído como una unidad
- ¿Hay desventajas?
    - Que pasa si queremos saber cuánto 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
- 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 como 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
- Hay algunas vistas que son costosas de calcular
- Solucion: 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.
- Las VM son un aspecto central en la BDs orientas a la agregación, tal vez mas 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**
- 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

Modelos de Distribución

- NoSQL DBs se ejecutan en un cluster de servidores.
    - + volumen de datos
    - + complejidad en escalar verticalmente

    - NoSQL permite escalar horizontalmente.
    - La agregación es la unidad de distribución
- Dependiendo del modelo de distribución:
    - Manejar volúmenes de datos mas grandes
    - Procesar mayor numero 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.
- 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)
- **Unico Servidor**
    - Sin distribución de datos
    - Ejecutar la BD en una sola máquina que maneja todos las lecturas y escrituras
    - Si bien las BDs NoSQL fueron dise˜nadas 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 mas simple:
        - Único Servidor.
- **Maestro-Esclavo (Replicación)**
    - 

    - Mas 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 máster
    - 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 ultimos cambios del sistema
        - Si falla el master, todos los datos no sincronizados se pierden
- **Sharding**

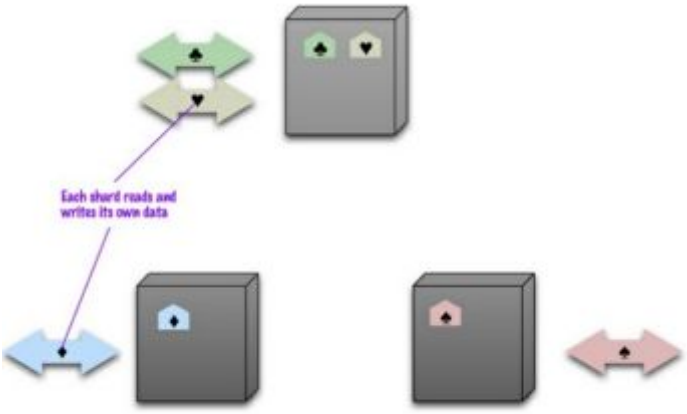


Figure 4.1. Sharding puts different data on separate nodes, each of which does its own reads and writes.

    - Normalmente una BD esta 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
    - Factores para mejorar el rendimiento
        - Si el acceso esta basado en ubicación física, se puede poner los datos lo mas 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.
- 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 es mas fácil de lograr con las agregaciones, pero no se debe tomar a la ligera
- Algunas bases son dise˜nadas para usar sharding, en este caso usar sharding desde el principio
- Otras bases no fueron dise˜nadas 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 (Replicacion)**

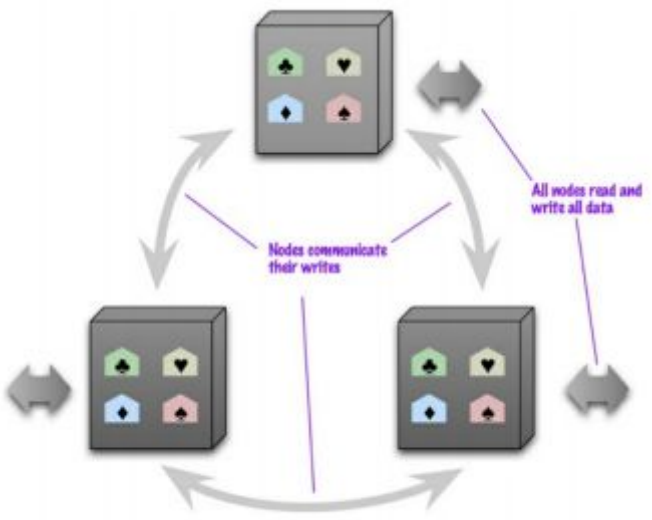


Figure 4.3. Peer-to-peer replication has all nodes applying reads and writes to all the data.

- **Combinar Sharding y Replicación**

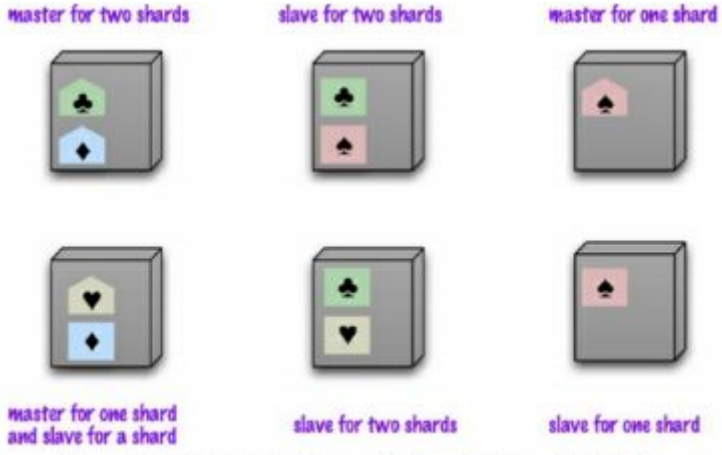


Figure 4.4. Using master-slave replication together with sharding

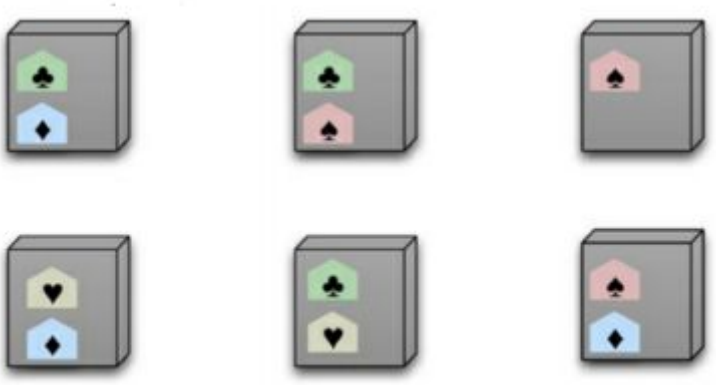
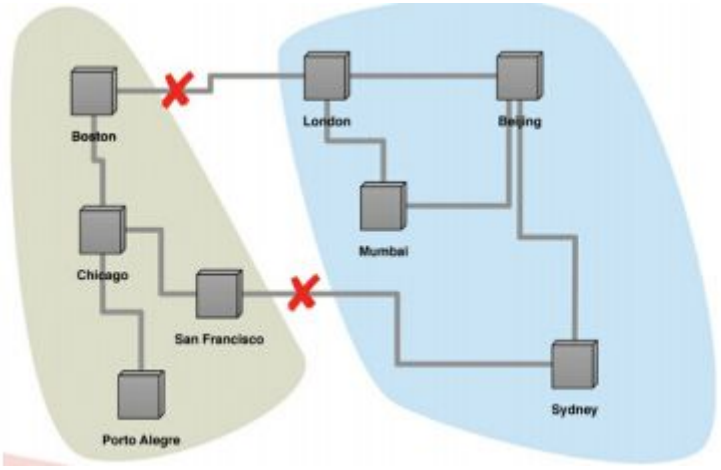


Figure 4.5. Using peer-to-peer replication together with sharding

-
- **Peer to peer**
    - Todas la 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

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 mas formal y pruebas (2)
    - (1) Harvest, Yield, and Scalable Tolerant Systems
    - (2) Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web
- **Consistencia**: Después de una escritura exitosa, las lecturas posteriores siempre incluyen 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)

- 

- Particionamiento: No solo paquetes perdidos
    - Servidor caido (Una partición)
        - Todos los paquetes enviados hacia el se pierden.
        - La falla mas sencilla de manejar (Hay seguridad que el servidor no da respuestas erróneas)
- CA Systems
    - Consistencia, Disponibilidad – No Particionamiento
        - Único Servidor: Sistema Monolıtico (No red, no particion 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 ası NO EXISTEN
    - Ante la presencia particiones de red (Errores), que se sacrificara? Consistencia o Disponibilidad?
        - La posibilidad de particiones siempre esta 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.
- 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.
- 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.

- 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 (opcion 1) o Consistencia (opción 2)
    - 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 habrá fallas en el sistema.
        - Dejar de responder peticiones (lectura/escritura)
        - Dar respuestas basadas en información incompleta