Bases de datos NoSQL: Modelado

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: https://citius.gal/team/jose-ramon-rios-viqueira

Curso 2023/2024

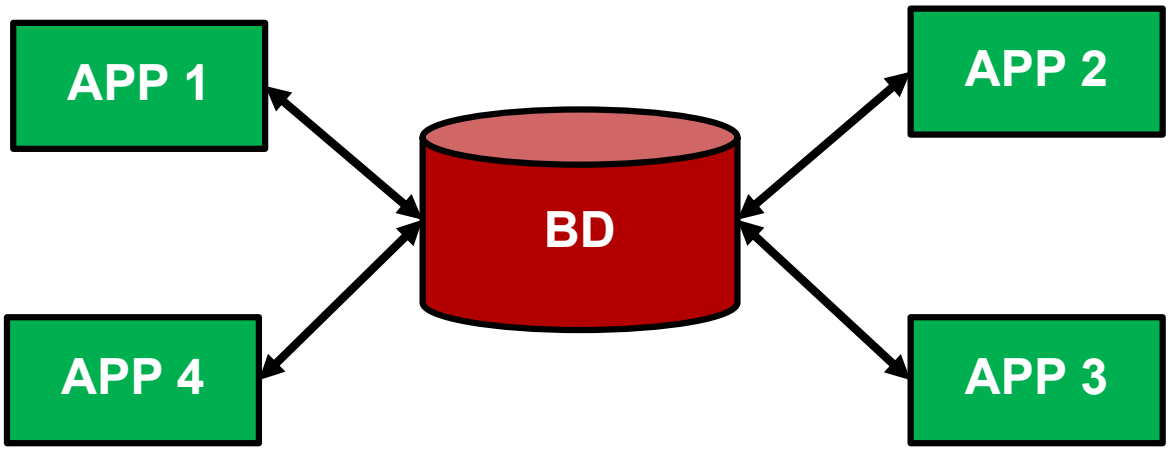
Guion

- Motivación: ¿Por qué NoSQL?
- Modelos de datos
- Bases de datos sin esquema
- Modelando para el acceso a datos

Motivación: ¿Por qué NoSQL?

- **Bases de datos Relacionales**
  - Dominantes durante muchos años (siguen siéndolo)
  - Filtrado eficiente de datos persistentes
  - Control en el acceso concurrente: Consistencia
  - Integración de aplicaciones
    - Base de datos compartida
    - Colaboración y coordinación a través de los datos persistentes
  - Estandarización
    - Modelo relacional
    - Lenguaje SQL

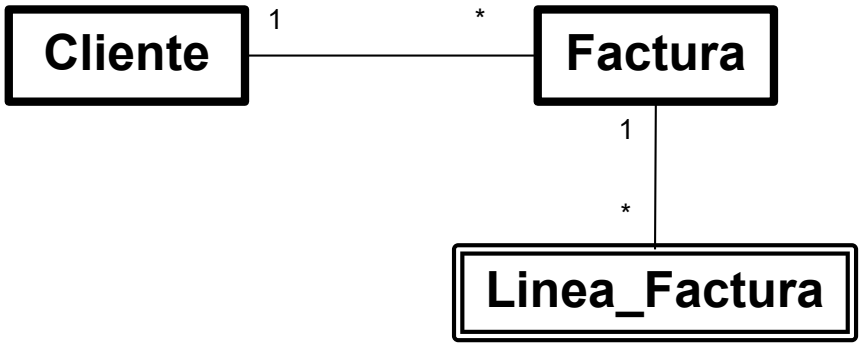
muy importante la estandarizacion, para no tener que recodificar aplicaciones al cambiar la BD (oracle, ibm, etc)

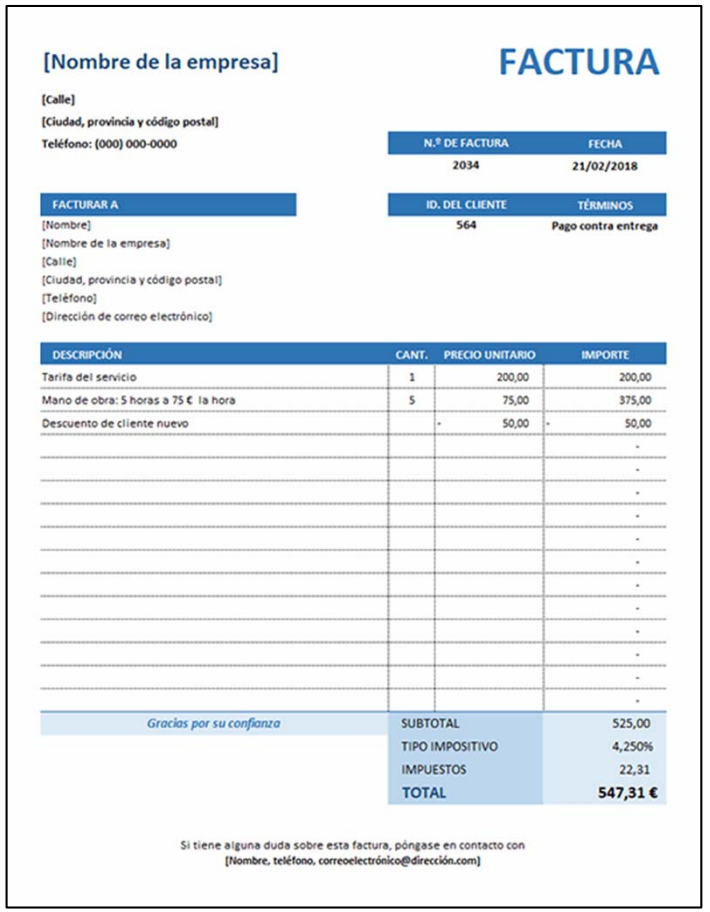


Motivación: ¿Por qué NoSQL?

Impedancia entre modelos Memoria/Disco

- Problema
  - Disco: tuplas simples (1ª Forma normal)
  - Memoria: Estructuras complejas (anidadas)
- Bases de datos orientadas a objetos
  - No llegaron a triunfar
- Mapeado objeto-relacional
  - No acaban de resolver el problema
  - Crean otros problemas
    - Eficiencia cuando tratamos de no usar el SGBDs





[Nombre de la empresa]

FACTURA

[Calle]
[Ciudad, provincia y código postal]
Teléfono: (000) 000-0000

N.º DE FACTURA	FECHA
2034	21/02/2018

FACTURAR A	ID. DEL CLIENTE	TÉRMINOS
[Nombre] [Nombre de la empresa] [Calle] [Ciudad, provincia y código postal] [Teléfono] [Dirección de correo electrónico]	564	Pago contra entrega

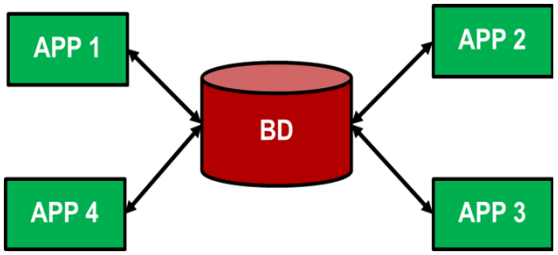
DESCRIPCIÓN	CANT.	PRECIO UNITARIO	IMPORTE
Tarifa del servicio	1	200,00	200,00
Mano de obra: 5 horas a 75 € la hora	5	75,00	375,00
Descuento de cliente nuevo		-50,00	-50,00

Gracias por su confianza	SUBTOTAL	525,00
	TIPO IMPOSITIVO	4,250%
	IMPUESTOS	22,31
	TOTAL	547,31 €

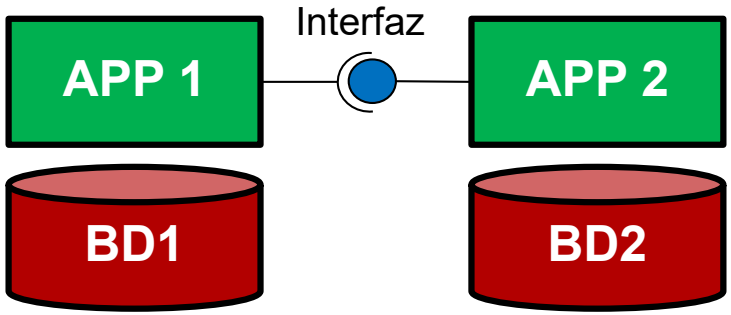
Si tiene alguna duda sobre esta factura, póngase en contacto con [Nombre, teléfono, correoelectrónico@dirección.com]

Motivación: ¿Por qué NoSQL?

Integración de aplicaciones a través de una BD centralizada



- Problemas
  - Base de datos compleja
  - Cambios en la BD necesitan coordinación entre aplicaciones
    - Ejemplo: Índice necesario para una crea problemas de eficiencia en otra
  - Son necesarios mecanismos de verificación de consistencia en la BD
- Alternativa: Cada aplicación tiene su BD
  - El equipo de cada app. conoce, mantiene y evoluciona su BD
  - Verificación de integridad en cada app.
  - Integración a través de **interfaces**
    - Servicios web
    - Uso de XML y JSON
  - Distintos modelos de BD en distintas aplicaciones
  - Se pueden mover responsabilidades de la BD a la app.
  - Incluso en estas configuraciones, las BD relacionales siguen siendo la opción dominante



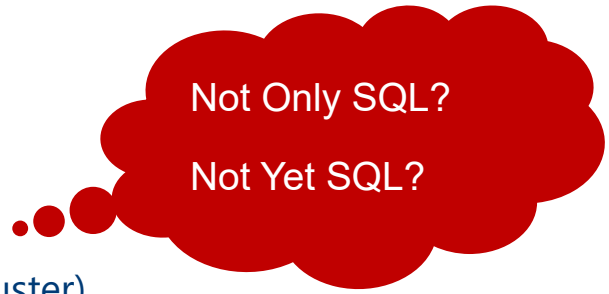
Motivación: ¿Por qué NoSQL?

- **Necesidad del uso de clusters**
  - Incremento de la escala de algunas aplicaciones
    - Monitorización de la actividad sitios web
    - Grandes conjuntos de datos
    - Muchos usuarios
  - Necesidad de más recursos (Opciones)
    - Máquinas más potentes y caras (escalabilidad limitada)
    - Cluster de muchas máquinas pequeñas
      - Mayor Resiliencia y Fiabilidad
      - Menor Precio
  - BD relacionales no diseñadas para clusters
    - BD sobre sistema de archivos distribuido
      - Único punto de fallo
    - Varios SGBDs para almacenar distintas partes de los datos
      - La fragmentación debe controlarla la aplicación.
  - Aparecen alternativas: Big Table (Google), Dynamo (Amazon)
  - Esta si parece una amenaza para la hegemonía de las BD relacionales

Motivación: ¿Por qué NoSQL?

Aparición de las BD NoSQL

- Falta una definición clara del término NoSQL
  - Inicialmente: "Bases de datos no relacionales distribuidas y de código abierto"
    - Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB y MongoDB
- Características
  - No usan SQL (mucho debate sobre esto)
    - Lenguajes muy parecidos
    - ¿Si alguna acaba implementando SQL?
  - Funcionan sobre arquitecturas distribuidas (Cluster)
    - Impacto en el modelo de datos
    - Impacto en la solución de consistencia (problemas entre ACID y clusters)
      - Rango de opciones para distribución y consistencia
    - Excepción: Bases de datos de grafos
  - Necesidades de aplicaciones del siglo 21 (web, etc.)
  - Operan sin esquema
    - En BD relacionales acaban apareciendo campos como "customField6"



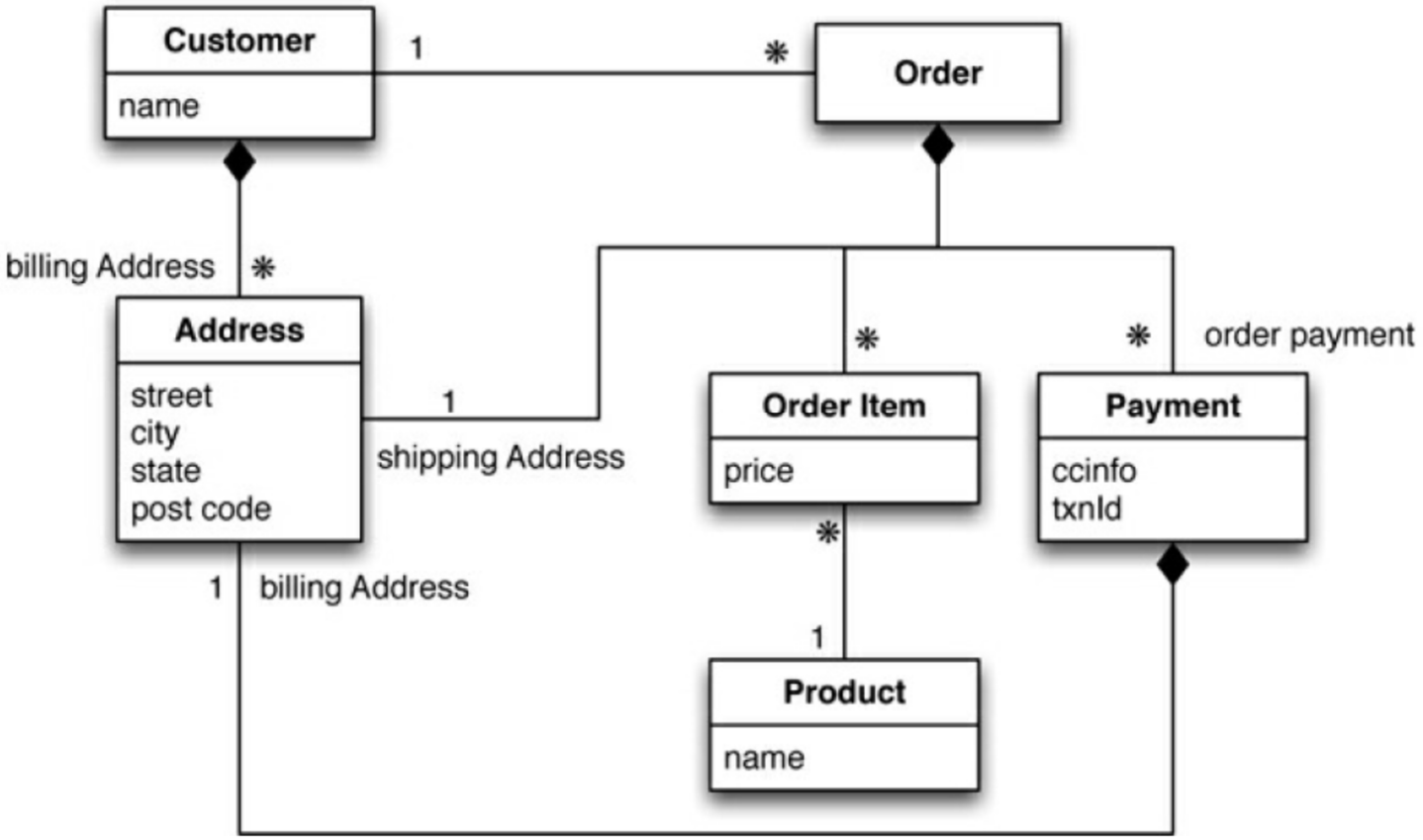
Motivación: ¿Por qué NoSQL?

Aparición de las BD NoSQL

- Nuevo abanico de opciones para el almacenamiento de datos, sin restringir la incorporación de nuevos tipos
  - Relacionales siguen siendo el tipo más utilizado (pero no el único)
- Uso como BD de aplicación y no como BD de integración
- Razones principales para considerar NoSQL
  - Tamaño y rendimiento que hace necesario el uso de un cluster
  - Productividad en el desarrollo de aplicaciones
    - Interacción con los datos más natural (cercano al entorno de aplicación)

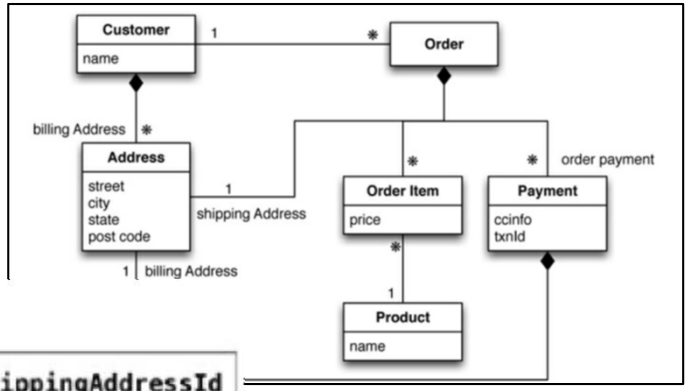
Agregados

l emodelo de datos aparece, entre otros motivos, por el de agragados



Modelos de datos

- **Agregados** agrega muchas estruturas
  - Modelo relacional no permite anidar estructuras



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

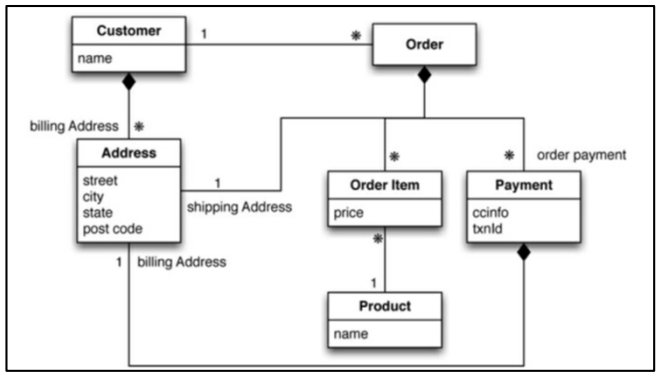
Id	City
77	Chicago

OrderPayment

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

Modelos de datos

- **Agregados**
  - NoSQL: Key-Value, Column-Family, Document, Graph




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

la gran diferencia es la utilizacion de agregados !!!!

JSON Document

Agregados

- NoSQL: Key-Value, Column-Family, Document, Graph
  - **Diseñar estructuras pensando en como se van a acceder**
    - Desnormalización para minimizar accesos
  - Consecuencias
    - La agregación puede beneficiar unas consultas y perjudicar otras
    - Trabajo con un cluster
      - Determinar datos que deben de ir juntos para minimizar accesos a varios nodos
    - **BD NoSQL en general no soportan ACID en transacciones que se expanden por varios agregados**

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

Modelos Key-Value y Document

carcateristicas de los modelos. Hay cuatro grandes tipos:

- Pares **(Clave, Agregado)**
  - Key-Value

HBASE es un poco mas que esto

cuando le pides una tira de bits, te devuelve la tabla de bits que corresponde

permiten busquedas por rango tambien en la clave

    - Agregado es opaco para la base de datos (BLOB)
    - No imponen restricciones sobre el contenido del agregado
    - Solo se puede consultar por clave
  - Document

ejemplo MongoDB

    - La base de datos entiende la estructura compleja del agregado
    - Definen las estructuras posibles para el agregado (Ejemplo: JSON)
    - Se puede

eprmite ver y consultar dentro del agregado

        - consultar sobre los campos del agregado
        - recuperar solo parte del agregado
        - etc.
  - Puede haber soluciones difíciles de clasificar por tener características de ambas

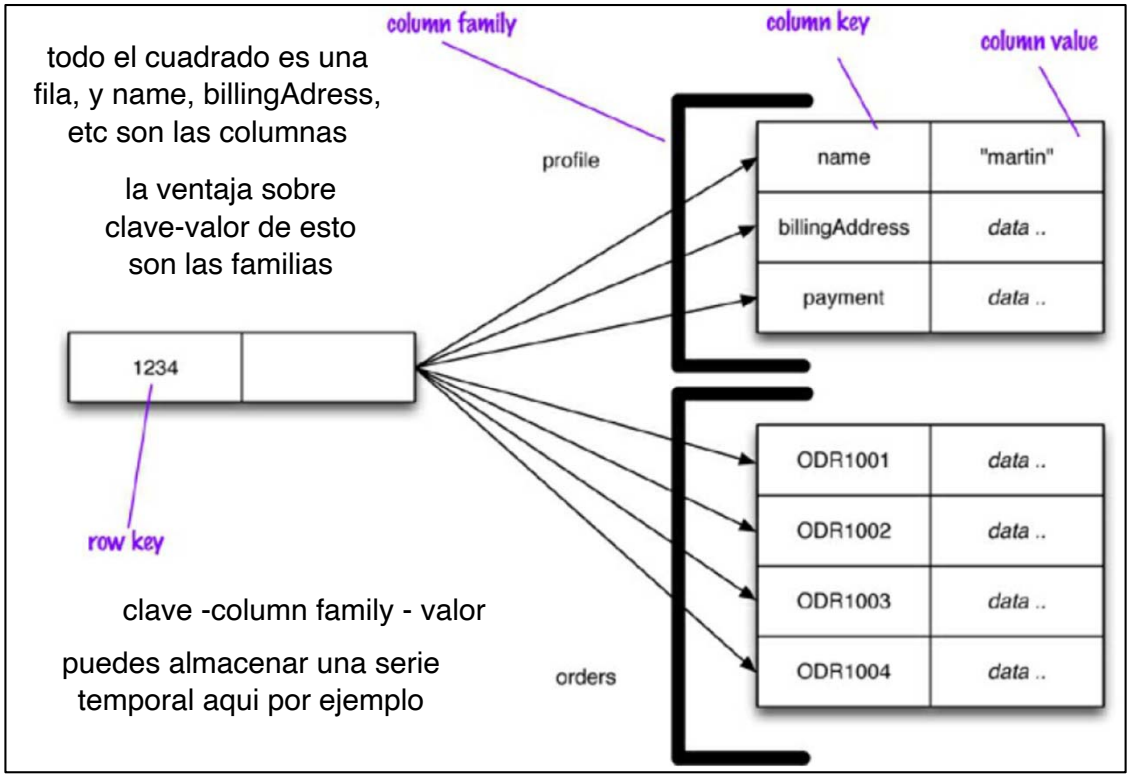
Ninguna de estas tiene esquema, la key-value no tiene ni estrutcura

Modelo Column-Family

al eovlucion de las clave valor añade un poco de estrtuctura, lo minimo

- Google Big Table — NO CONFUHDIR CON ALMACENAMIENTO COLUMNAR
- Estructura tabular con columnas dispersas y sin esquema (Column-Stores).
- **Mapeo en dos niveles** — NO es una tabla
- Influencia para HBase y Cassandra
- Column Families: Se almacenan juntas.

el esquema es el nombre de las familias, lo unico que se declara. En el ejemplo, profile y orders



¡Confusión!

Almacenamiento relacional por columnas

BD para almacenes de datos. Analítica on-line. Penalizan transacciones

EJEMPLO CASANDRA Y HBase.

para una clave, en vez de tener un valor, para cada familia tenemos pares clave valor

Modelos de datos

Modelo Column-Family

- Organización de los datos
  - Por filas
    - Cada fila es un agregado. Cada column family un bloque de datos dentro del agregado
  - Por columnas
    - Cada column family define un tipo de registro
- **Cassandra** usan una notacion un poco distinta
  - Una fila solo puede pertenecer a una column family
  - Una column family puede tener supercolumns (con columnas anidadas) Cassandra supercolumns == column family HBase
    - Concepto equivalente a las column families de **Big Table** o **HBase**
- Una column family puede tener columnas distintas en cada fila (sin esquema)
  - Un nuevo pedido supone insertar una nueva columna en la column family "orders" de la fila correspondiente al cliente
- **Cassandra**
  - Skinny row: pocas columnas. Mismas columnas en la mayoría de filas (ej: profile)
  - Wide row: Muchas columnas (miles). Filas con columnas muy variadas. Modelan una lista. Orden para las columnas. Recuperar todos los pedidos, o un rango de pedidos en un solo acceso.

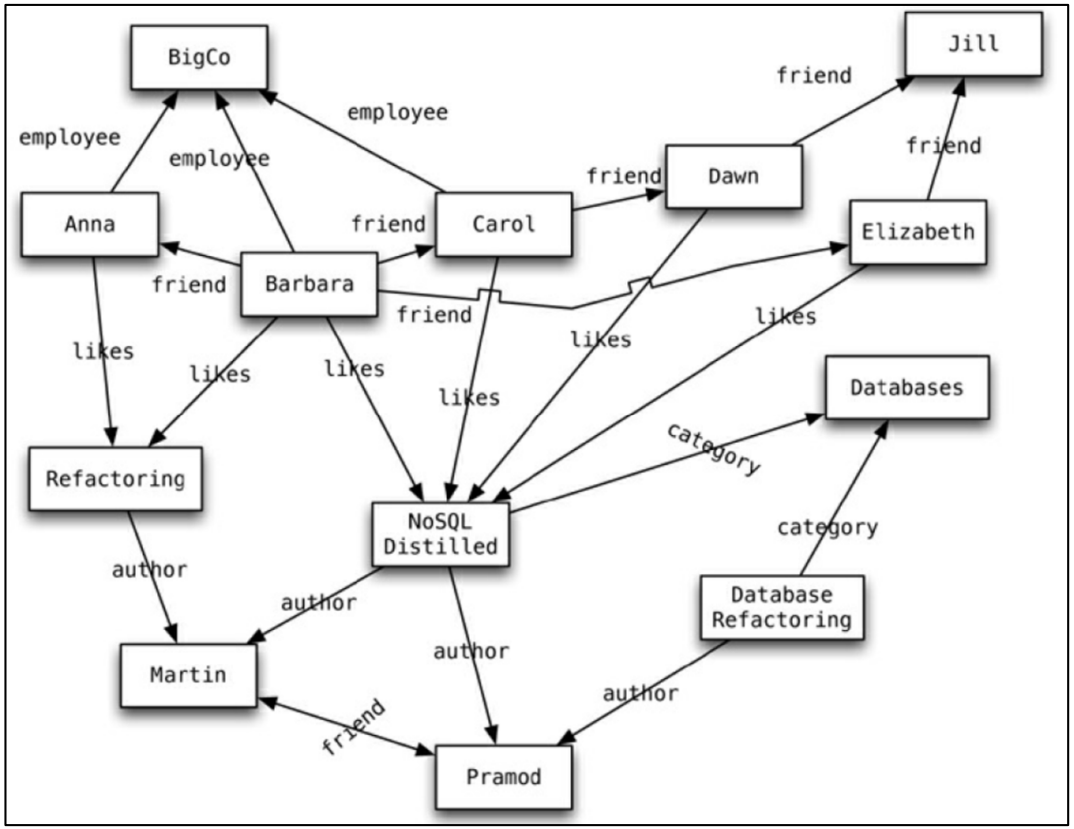
Modelos de datos

Relaciones

- Acceso a datos relacionados de distintas formas
  - **Ejemplo: Cliente - Pedido** relacion cliente-pedido
    - Obtener los pedidos de un cliente
    - Obtener los clientes de los pedidos de un producto concreto
  - La relación se almacena con una referencia a una clave. BD no conoce la existencia de la relación (es solo un dato más)
- Alternativa: Hacer visibles las relaciones para la base de datos
  - Dividir los datos y habilitar relaciones entre ellos
  - Modificar varios agregados en una sola transacción.
    - BD NoSQL no lo permiten normalmente
    - BD relacional proporciona ACID
  - BD relacional no muy eficiente si hay que realizar muchos joins para seguir muchas relaciones
    - Consultas difíciles de expresar
    - Bajo rendimiento

en estos casos de muchos join, muchas tablas en el from, hay que pensar si la relacional es la tecnologia que relamente hace falta —>

- **Bases de datos de Grafos**
  - Caso especial de NoSQL
    - No motivado por agregados y clusters
  - Motivación
    - Registros muy simples
    - Muchas relaciones entre los datos
    - Ejemplo
      - Libros de bases de datos escritos por alguien que le guste a alguno de mis amigos
  - Aplicaciones
    - Redes sociales, preferencias de productos, etc.
  - Modelo compuesto por Nodos y Arcos
    - Posiblemente con datos en nodos e incluso en arcos
    - Neo4J: Objetos java como propiedades de los nodos y arcos (sin esquema)
  - Navegación rápida de las relaciones. Inserción puede ser lenta.



al consulta suele ser entrar en un nodo y expandirse a buscar vecinos

los arcos son los enlaces

Bases de datos sin esquema

- Característica común a todas las BD NoSQL
  - No tener esquema
- No es necesaria la previa definición de un esquema para almacenar datos
  - **Clave-valor**: Cualquier valor se puede insertar para una clave
  - **Document**: No hay restricciones sobre el contenido de cada documento
  - **Column-family**: Cualquier dato se puede almacenar sobre un columna de una fila
  - **Grafos**: La propiedades de nodos y arcos son libres
- **Ventajas de trabajar sin esquema**
  - No tener que hacer asunciones a priori
  - Facilidad para incorporar cambios en los datos
  - Facilidad para trabajar con datos no uniformes
  - Problemas de tener esquema
    - Inflexible en la inserción de datos
      - Muchas columnas con valores nulos
      - Columnas con semántica poco definida
        - Ej: "columna3", "informacion", "comentarios", ...

segun van llegando los datos podemos insertar cualquier cosa

aparecen y desaparecen campos

campo de texto donde va cualquier cosa

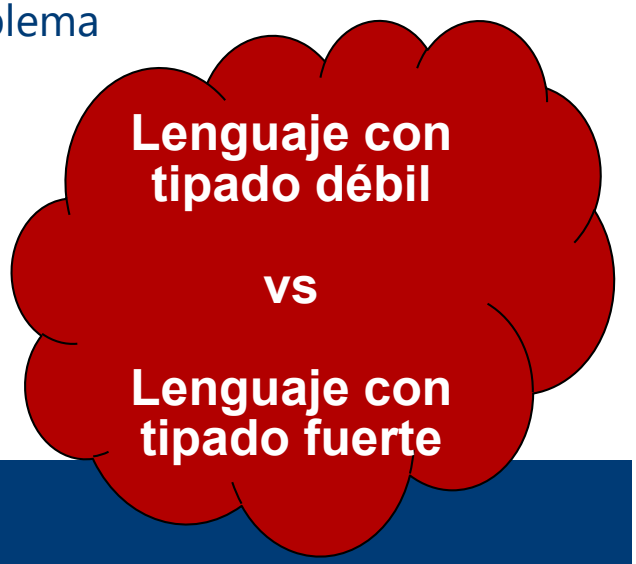
Bases de datos sin esquema

Problemas de trabajar sin esquema

- Las aplicaciones necesitan normalmente cierto formato y semántica en los datos.

hay que minimizar los literales para que el codigo se puede cambiar y mejorar de forma facil

  - Nombres de las columnas, tipos de datos, etc.
  - Casi cualquier programa que escribimos confía en la existencia de cierto esquema implícito en los datos
- Tener el esquema codificado en las aplicaciones puede dar problemas
  - Tener que analizar código para entender los datos
  - La BD no puede utilizar el esquema para mejorar la eficiencia
    - Column-family tiene cierto esquema, por eso puede ser más eficiente
  - No se pueden implementar restricciones de integridad en la base de datos
- Los esquemas tienen valor y su rechazo puede ser incluso alarmante
- Una BD sin esquema realmente desplaza el problema del esquema a la aplicación
  - Puede ser peor si las aplicaciones se realizan por personas distintas



Bases de datos sin esquema

- **¿Soluciones?**
  - Opción 1: Integrar todo el acceso a la base de datos en una única aplicación que proporciona servicios web para las demás
  - Opción 2: Delimitar diferentes partes de cada agregado para el acceso de aplicaciones diferentes
- Esquemas en bases de datos relacionales
  - Se critica su falta de flexibilidad
    - Pero se pueden modificar los esquemas
      - Añadir y quitar columnas por ejemplo
- Problemas al almacenar datos de formas nuevas en BD NoSQL.
  - Las aplicaciones tienen que funcionar con datos viejos y nuevos

es flexible, lo que no hace es inventar datos donde no hay

pera modificar, añades una columna y convives con dos columnas,
una con nulos hacia arribay otra con nulos hacia abajo

Bases de datos sin esquema

- **Cambios en el esquema**
  - Importancia en metodologías ágiles poder cambiar fácilmente de esquema
  - En NoSQL se pueden hacer cambios rápido, pero hay que tener cuidado con las migraciones de esquema
  - **Cambios de esquema en BD Relacionales**
    - Un cambio de esquema puede ser un proyecto en si mismo
    - Necesidad de crear scripts para la migración de datos de un esquema a otro
    - Proyectos nuevos
      - Almacenar los cambios con los scripts de migración de datos
      - Ejemplo: DBDeploy para manipular cambios en la BD
        - Guardar una tabla con todas la versiones del esquema
        - Herramientas para mantener el historial de versiones junto con las versiones de las aplicaciones
    - Proyectos legacy
      - Extraer el esquema de la BD
      - Proceder como con los proyectos nuevos
      - Mantener compatibilidad hacia atrás
        - Fase de transición en la que ambos esquemas funcionan
        - Código tipo Scaffolding (disparadores, vistas, etc.)

Bases de datos sin esquema

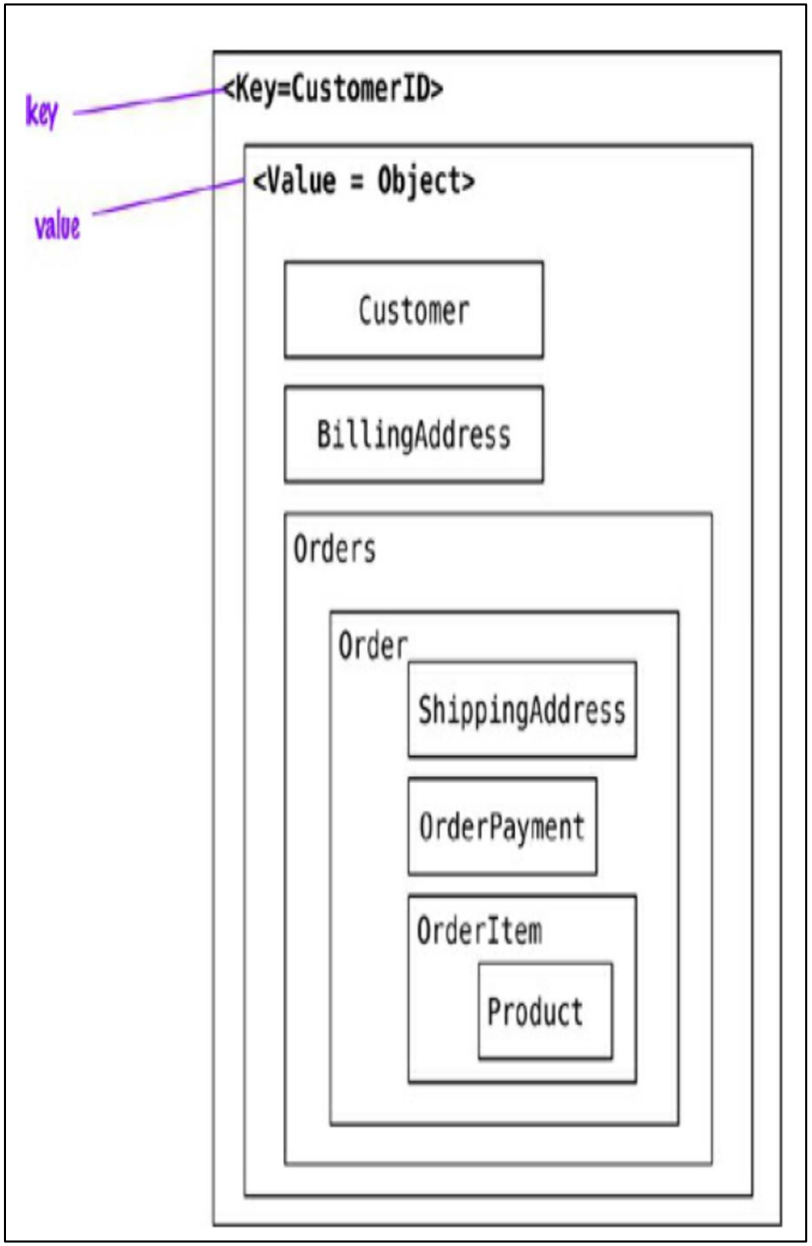
- **Cambios en el esquema**
  - **Cambios de esquema en BD NoSQL**
    - Se intenta no tener que realizar estos cambios de esquema
    - Engañoso pensar que las BD NoSQL no tienen esquema
      - El esquema está en la aplicación
        - Debe parsear los datos obtenidos de la base de datos
      - Si no cambiamos la aplicación el error de esquema que daría la BD lo da la aplicación
        - Debemos cambiar el código que lee y el que escribe
    - Migración incremental
      - Migrar todos los datos al nuevo esquema puede ser muy costoso
      - Alternativa
        - Leer de ambos esquemas (período de transición)
        - Guardar en el más reciente
      - Algunos datos no llegan a migrarse nunca
    - Migración en BD de grafos
      - Alternativa a migrar todos los tipos de arco en la base de datos
        - Definir nuevos arcos con el nuevo esquema

el esquema esta en la aplicacion !

transicionar a partir de la propia aplicacion,
lee de ambos y siempre guarda en el nuevo

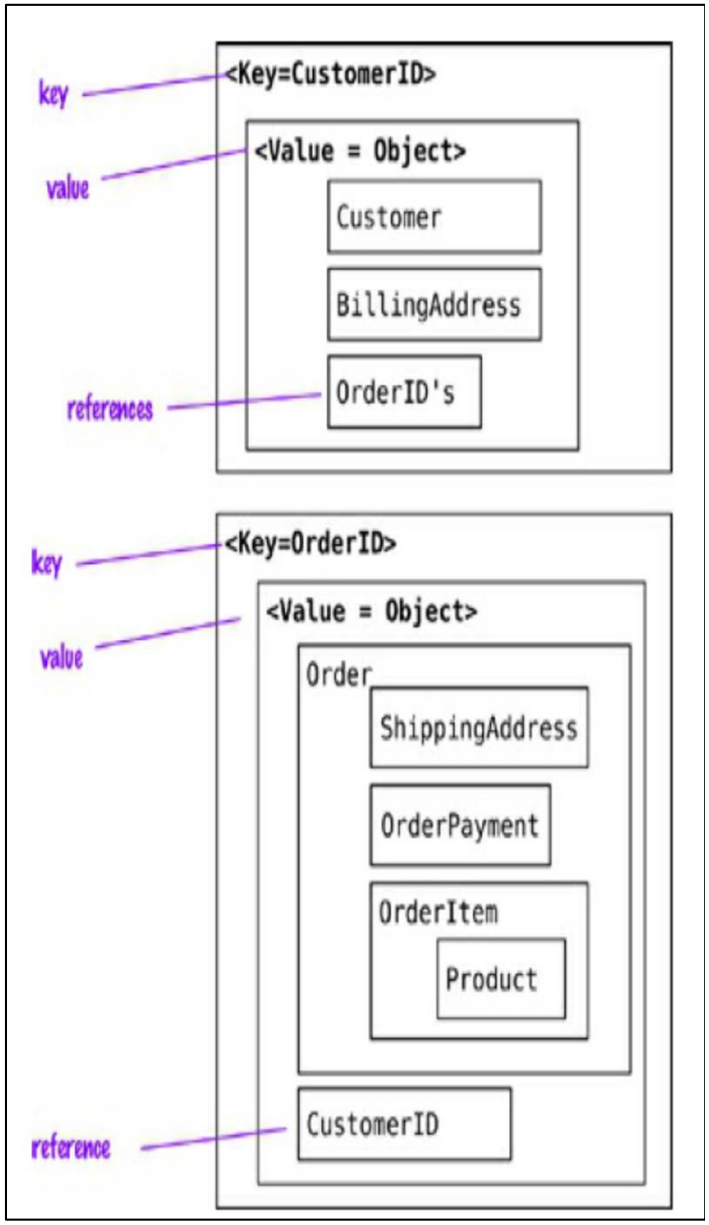
Modelando para el acceso a datos

- Modelado de los agregados
  - Tener en cuenta como va a ser el acceso
    - Ejemplo: Leer los datos de los productos necesita procesar todos los clientes



Modelando para el acceso a datos

- Modelado de los agregados
  - Tener en cuenta como va a ser el acceso
    - Ejemplo: Leer los datos de los productos necesita procesar todos los clientes
- Alternativa: Dividir el agregado y usar referencias
  - BD de documentos no necesitan almacenar las referencias en ambas direcciones
- Calcular agregados para realizar analítica concreta
  - Ejemplo: Almacenar en que pedidos se encuentra cada producto
  - BD documentos permiten realizar la consulta sin calcular el agregado, pero no es eficiente
- Column-families
  - Lecturas eficientes. Denormalizar en las escrituras



Bases de datos NoSQL: Modelado

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: https://citius.gal/team/jose-ramon-rios-viqueira

Curso 2023/2024