Bases de Datos
Tema 7: Introducción a modelos no relacionales

Dpto. de Ingeniería Informática

Contenidos

- Objetivos
- Introducción
- Evolución en el modelo relacional
  - Modificación de tablas
  - Modificación de atributos
  - Modificación de restricciones
- Claves surrogadas
- Modelos NoSQL
  - Modelos de datos agregados versus modelo en grafo
  - Características
- Referencias

Objetivos

- Entender la modificación de las estructuras estáticas en el modelo relacional
- Entender las ventajas e inconvenientes del uso de claves surrogadas
- Conocer las alternativas disponibles al modelo relacional: NoSQL

Introducción

- Una relación se representa como una tabla bidimensional donde los datos se muestran utilizando la estructura registro para formar su cuerpo
  - Pueden cambiar en el tiempo (por ejemplo, quiero almacenar más atributos de cada tupla)
- Toda BD está sujeta a un gran número de estas reglas y son de dos tipos:
  - *Generales* para toda BD: que se cumplan las que implican las claves primarias y foráneas (no se modifican)
  - *Específicas* para cada BD: todo hijo tiene una sola madre (estas pueden cambiar y por ejemplo tener dos madres)

Evolución de los diferentes elementos

- No existen acuerdos sobre todo lo que se puede cambiar en las estructuras del modelo relacional
  - Y menos en qué consecuencias tiene
- Nos apoyaremos principalmente en el manual de MySQL
- Iremos viendo elemento por elemento del modelo relacional
- Normalmente se hace una réplica de la BDD en producción en un entorno de pruebas para comprobar la propuesta
  - Y automatizar su implementación con *scripts*
- Cada cambio puede requerir determinados privilegios sobre la BDD
  - Y lo normal es efectuarlos bloqueando el acceso a la tabla con LOCK (o sólo dejar leer si procede)
  - Además de planificarlo y notificarlo a los afectados ;)

Cambios en tablas

- Las tablas son estáticas, y se definen en un esquema que indica atributos y restricciones
  - Creación de tablas
    - Se explica en prácticas
  - Si no necesito más una tabla:
    - *DROP TABLE tabla1;*
    - Se eliminan sus datos, restricciones asociadas y la estructura de la tabla de la BDD
  - Normalmente lo que necesito es cambiar su estructura: atributos, restricciones, etc
    - *ALTER TABLE*

Cambios en tablas

- A veces quiero “dividir” una tabla en dos
  - ¿Es algo común?

Cambios en tablas

- A veces quiero “dividir” una tabla en dos
  - Muy común para normalizar
    - Aunque puede deberse a cambios en requisitos
  - Hay que hacerlo en varios pasos
    - Preparar esquemas (con atributos y restricciones)
      - Suele requerir una secuencia propia
      - Se puede optar por crear 2 (o N) tablas, o “reutilizar” la original
    - Después migrar datos
      - Completar datos si procede: repeticiones, NULL, etc
      - Eliminar campos o tablas innecesarios

Cambios en atributos

- ¿Qué problemas pueden darse si deseo añadir atributos nuevos?
  - En principio ninguno a nivel *lógico*
  - Pero sí puede haberlos a nivel *físico*. Ej: he configurado para tener la tabla entera en un disco X y al añadir más atributos necesita dos discos
- Ejemplo
  - ALTER TABLE tabla1 ADD columna8 *def_columna*
    - La *def_columna* es igual que para definir una columna de una tabla que creo: tipo de dato, restricciones, etc

Cambios en atributos

- Por defecto la columna va al final de la tabla
  - A menos que añada el parámetro *FIRST*
    - Entonces irá como primera columna de la tabla
  - O ponga *AFTER columna4*
    - Para que vaya tras la columna indicada
- ¿Qué valor tendrá inicialmente el nuevo atributo en las tuplas de la relación?
  - *NULL*
  - Hay que plantearse si es un valor correcto para el caso concreto

Cambios en atributos

- ¿Qué problemas pueden darse si deseo modificar atributos?
- Pero ¿para qué lo queremos hacer?
  - ¿Cambiar su nombre? Para eso usamos vistas
  - ¿Reordenarlos? Si es por comodidad, usar vistas. Si es por eficiencia, de eso se encarga el nivel físico
  - ¿Cambiar su tipo de datos (dominio)? Cuidado: si es a un tipo de menor capacidad o precisión puede dar lugar a truncados y pérdida de información
    - Considerar crear un atributo nuevo, hacer el trasvase manualmente y eliminar el anterior
  - MySQL tiene CHANGE y MODIFY, fuera del estándar

Cambios en atributos

- ¿Qué problemas pueden darse si deseo eliminar atributos?
  - En principio ninguno por el hecho de ser un dato
    - Otra cosa es que tenga restricciones asociadas (lo tratamos un poco más adelante)
- Ejemplo:
  - *ALTER TABLE clientes DROP apellidos;*

Cambios en atributos

- Hay muchas más elementos que se pueden cambiar en las tablas por ejemplo:
  - Forma de almacenarse en disco/memoria
  - Compresión de datos
  - Encriptación
  - Etc
- Se escapan del objetivo de la asignatura, para eso está ABBDD

Consideraciones

- Los cambios en la BD pueden afectar a sus usuarios (o no) dependiendo del tipo de cambio:
  - Evidentemente, si añado o elimino tablas se ven afectados todos los SELECT sobre ellas
  - Si añado atributos los “SELECT * ...” siguen funcionando, pero probablemente los atributos nuevos no aparezcan en los informes
  - Si elimino atributos los “SELECT * ...” siguen funcionando, pero es probable que a la hora de acceder a los campos de error
    - Y los “SELECT atributosX” puede fallar directamente

Consideraciones

- Un correcto uso de las vistas me puede ahorrar muchos (que no todos) los problemas
  - Si los datos siguen estando, se pueden rehacer las vistas para que los usuarios no sean conscientes de los cambios subyacentes

Ejemplos

- Unir dos tablas en una
- Dividir una tabla en dos
- Dividir los elementos de una columna en dos
- Cambiar tipos de datos
- Un buen ejemplo sería el *Brexit*: dejar de usar moneda, de tener la misma capacidad de circulación, su seguro médico cambia, etc.

Cambios en restricciones

- Si la CP de una relación deja de ser válida (porque se debe repetir datos o admitir nulos)
  - Mala elección hicimos en su momento :(
    - Por eso hay partidarios de usar siempre códigos que no se den a conocer al usuario final (sí a usuarios directos: programadores)
  - Si hay alguna clave candidata que sigue siendo válida, usarla
  - Si no la hay, crear un código (*autoincrement*)
- Afecta a las CF de otras relaciones sobre ella

Cambios en restricciones

- Cambio en atributos implicados en CF
- El caso más sencillo es que cambie la CP de una relación (que es CF en otra)
  - Si los tributos CF siguen existiendo y tienen UNIQUE normalmente no hay problema
  - Si la CP ha dejado de ser única / NOT NULL
    - Añadir a la tabla que tiene la CF los atributos de la otra que van a pasar a ser CP
    - Incluir contenido (valores/identificadores de la nueva CP)
    - Definir la nueva restricción de CF
    - Eliminar la antigua restricción de CF
    - Eliminar los atributos que mantenían la antigua CF

Cambios en restricciones

- Cambio en atributos implicados en CF
- Es bueno plantearse porqué cambian: las CF suelen ser relaciones (1:1, 1:N o M:N) del E/R
  - Si una relación 1:1 pasa a 1:N
    - Si ya tenía la CF en la entidad de la N sólo hay que quitarle el UNIQUE
    - En caso contrario hay que añadir el atributo CP de la relación que participa con 1 en la otra relación, establecer nueva CF, rellenar datos, eliminar restricción de CF y eliminar atributos
  - Si una relación 1:1 pasa a M:N
    - Crear relación nueva con las dos claves primarias y sus correspondientes CF, rellenar datos, eliminar restricción de CF y eliminar atributo

Cambios en restricciones

- Cambio en atributos implicados en CF
- Es bueno plantearse porqué cambian: las CF pueden ser relaciones (1:1, 1:N o M:N) del E/R
  - Si una relación 1:N pasa a 1:1
    - Comprobar que no haya repeticiones en los datos en la CF y añadir UNIQUE
  - Si una relación 1:N pasa a M:N
    - Igual que en el caso de 1:1 a M:N, crear relación nueva con las dos claves primarias y sus correspondientes CF, pasar datos, eliminar CF anterior y eliminar atributo

Cambios en restricciones

- Cambio en atributos implicados en CF
- Es bueno plantearse porqué cambian: las CF pueden ser relaciones (1:1, 1:N o M:N) del E/R
  - Si una relación M:N pasa a 1:N
    - Añadir la restricción UNIQUE a la clave primaria de la entidad con el 1
    - O se puede modificar esquemas para incorporar la CP de la entidad del 1 en la del N (como casos anteriores)
  - Si una relación M:N pasa a 1:1
    - Añadir la restricción UNIQUE a la clave primaria de cada entidad
    - O también incorporar la CP de una al otro esquema

Cambios en restricciones

- Cambio en atributos implicados en CF
- También puede deberse a
  - Cambios en DF
  - Cambios en otras restricciones: NOT NULL, UNIQUE, etc
  - Cambios en estructuras de una generalización
  - Cambios de participaciones: relación binaria a ternaria
    - No estudiaremos casos concretos, pero al igual que en los ejemplos anteriores hay que buscar la estructura destino que mejor respete el nuevo “mundo”, crearla y migrar datos
      - Puede implicar la delegación de control en programas externos (exactamente igual que al crear las bases de datos relaciones)

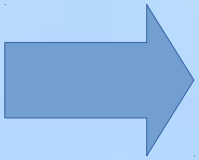
Claves surrogadas

- El concepto de clave surrogada se refiere a es un identificador único para una tupla que, al contrario que la clave “natural”, está fuera del dominio del sistema
  - En inglés se puede usar *surrogate key,* synthetic key, entity identifier, system-generated key, database sequence number, factless key, technical key, arbitrary unique identifier
    - Se puede encontrar traducida como clave sustituta o clave artificial

Claves surrogadas

- Ejemplo

DNI	**Nombre**
11111111	Antonio
12325431	Manuel
43284364	Raquel
33529751	Carlos
98765211	Salvador
11243525	Tatiana



ID	**DNI**	**Nombre**
1	11111111	Antonio
2	12325431	Manuel
3	43284364	Raquel
4	33529751	Carlos
5	98765211	Salvador
6	11243525	Tatiana

Claves surrogadas

- Los medios más comunes de implementarlo:
  - Definir el campo que será CP como INTEGER con AUTOINCREMENT
    - Cuando se hacen inserciones no se da valor el campo, de modo que el sistema lo añade: 1, 2, 3, etc
    - Es un entero único para esa tabla, y creciente
  - Usar un UUID (Universally Unique Identifier)
    - Es único a nivel global
    - Más costoso de generar, y ocupa más espacio
- Es probable que el SGBD lo tenga internamente ¿fuera de nuestro alcance?

Claves surrogadas

- ¿Y esto para qué?
  - Tener a los usuarios alejados de las CP
  - CP independientes de cambios en los requisitos
  - CP atómicas
  - CP siempre son enteros
  - Puede incrementar el rendimiento (según)
  - Uniformidad
  - Porque lo pide el framework X
  - Evitar problemas de FN2 y FNBC
  - Etc

Claves surrogadas

- Desventajas
  - Implica mayor uso de espacio
    - Espacio donde proceda: disco, memoria, caché, ...
  - No evita poner claves candidatas a UNIQUE y NOT NULL
  - No asegura FN3
  - Para optimizar las búsquedas hay que indexar por la clave surrogada y por la clave “del sistema”: más disco y tiempo (actualiz. y búsqueda)
  - Se complican las claves en M:N
  - Cuidado con usarlo para lo que no es. Ej: pensar que es un conteo de filas (si borro filas puede que ese valor de la clave se reutilice o no)
    - Si se reutiliza puedo creer que es la tupla anterior :(
    - Last_insert_id (concurrencia)
  - Complica el desarrollo: el mismo dato en producción y en prueba/auditoría puede tener distinta CP

Alternativas al modelo relacional

- En los últimos años han aparecido modelos alternativos al relacional con éxito
- Justificar qué violamos del modelo relacional para según qué queramos ganar
  - Eficiencia
  - Flexibilidad en modelos
  - Interoperabilidad
  - Tiempo de respuesta
  - ...

Modelos NoSQL

- En los últimos 30 años (sí, antes que nacierais los que estáis aquí ;) han cambiado muchas cosas en Informática:
  - Lenguajes de programación
  - Arquitecturas
  - Plataformas
  - Procesos
  - …
  - Pero los SGBD Relacionales siguen ahí
    - Es “bueno”, los datos son la esencia de la Informática

Modelos NoSQL

- NoSQL no tiene una definición consensuada
- Son diversos modelos famosos desde 2009 que no cumplen el modelo relacional porque
  - No tienen esquemas (o los tienen flexibles/implícitos)
  - Funcionan de manera distribuida “pura”
  - Proporcionan otras ventajas no funcionales
  - …
  - A cambio de dejar de garantizar la consistencia “total” del modelo relacional
- Se basan en principios documentados desde 1960s

Modelos NoSQL

- Tras unos años *heap* parece que se ha llegado a un status quo de convivencia entre SQL y NoSQL modelos, según tipo de proyecto, etc
- Normalmente los sistemas NoSQL permiten
  - Manejar un modelo de datos más cómodo para el desarrollador
    - Pero menos para el gestor de bases de datos
  - Escalar de manera cómoda a grandes volúmenes de datos (distribuidos)
    - A costa de “perder” algo de control sobre ellos

Puntos fuertes de SGBDR

- Los SGBDR han ofrecido cuatro pilares fuertes:
  - Respaldo de datos muy fiable
    - Gestionados de manera centralizada
  - Muy buen soporte de concurrencia
    - Totalmente transparente
  - Fácil de explotar desde distintas aplicaciones
  - El estándar SQL está muy soportado en herramientas (que lo suelen respetar mucho)
    - Y hay mucho personal formado

Puntos débiles de SGBDR

- Todo se representa como tuplas de tablas
  - Esto genera *Object-relational impedance mismatch*
- Los lenguajes OO han sido un éxito, y pueden trabajar con tipos abstractos (por ejemplo *factura*). Pero las SGBBD OO fracasaron :(
  - Ejemplo: una factura “real” pueden ser unas pocas de tuplas desperdigadas: cliente por una parte, artículos por otra, descuento por otras, etc.
    - Es más, puede ser que la factura como tal ni siquiera tenga representación en datos (si es entera información derivada)

Puntos débiles de SGBDR

- Todo se representa como tuplas de tablas
  - Esto genera *Object-relational impedance mismatch*
  - Es más, se supone que los datos de cada celda son atómicos, pero eso no siempre es así
    - A veces tenemos campos textuales donde se hacen búsquedas o incluso binarios
  - La solución hasta ahora ha sido usar *frameworks* de desarrollo que realizan un trabajo de mapeado para facilitar el trabajo a los programadores
    - Django, Ruby on Rails, CakePHP, CodeIgniter, Drupal, Gyroscope, Laravel, Symfony, TYPO3, ...

Puntos débiles de SGBDR

- Uso de BD como integradoras de aplicaciones
  - Decir a todas las aplicaciones que usen la misma BDD está bien porque todos usan SQL …
    - … siempre que satisfagan sus requisitos
  - Pero a veces alguna aplicación requiere algo de la BD que afecta a las demás
    - En empresas grandes los datos puede haber alguna aplicación que requiere respuesta más rápida, introducir muchos datos que sólo usa ella (y afecta al nivel físico), etc
  - Y los cambios que se hacen en la BD afectan a todas las aplicaciones

Puntos débiles de SGBDR

- Uso de BD como integradoras de aplicaciones
  - Por ahora la solución ha sido usar comunicación sobre HTTP, servicios web (SOA), etc
    - Y los datos se intercambian en XML, Json, etc
  - Esto es, se desacoplan las bases de datos que interactúan de las aplicaciones que las explotan

Puntos débiles de SGBDR

- Escalabilidad:
  - El modelo relacional está pensado para gestionar una BD centralizada
    - Puede ser que a nivel físico se redunde, distribuya, etc
    - Pero a nivel lógico es sólo una BD
  - Se comporta muy mal en la nube (no aporta la flexibilidad que a veces se requiere hoy en día)
    - Prefiere el uso de disco más grandes, más rápidos, con mejor soporte RAID, etc. siempre centralizado
      - El problema es que normalmente la respuesta o fiabilidad de un disco no aumenta proporcionalmente a su precio ...

Modelos NoSQL

- Ante esto la alternativa es NoSQL
  - Fenómeno famoso por el open-source
    - “Disruptivo”
  - Se usa a veces para referirse a “no sólo SQL”
    - Por lo que considera ambientes con SQL y “otras tecnologías”
  - Es más, siendo conscientes de que para tener éxito hay que ser aceptado intentan parecerse a SQL en la medida de lo posible

Meta-modelos de datos

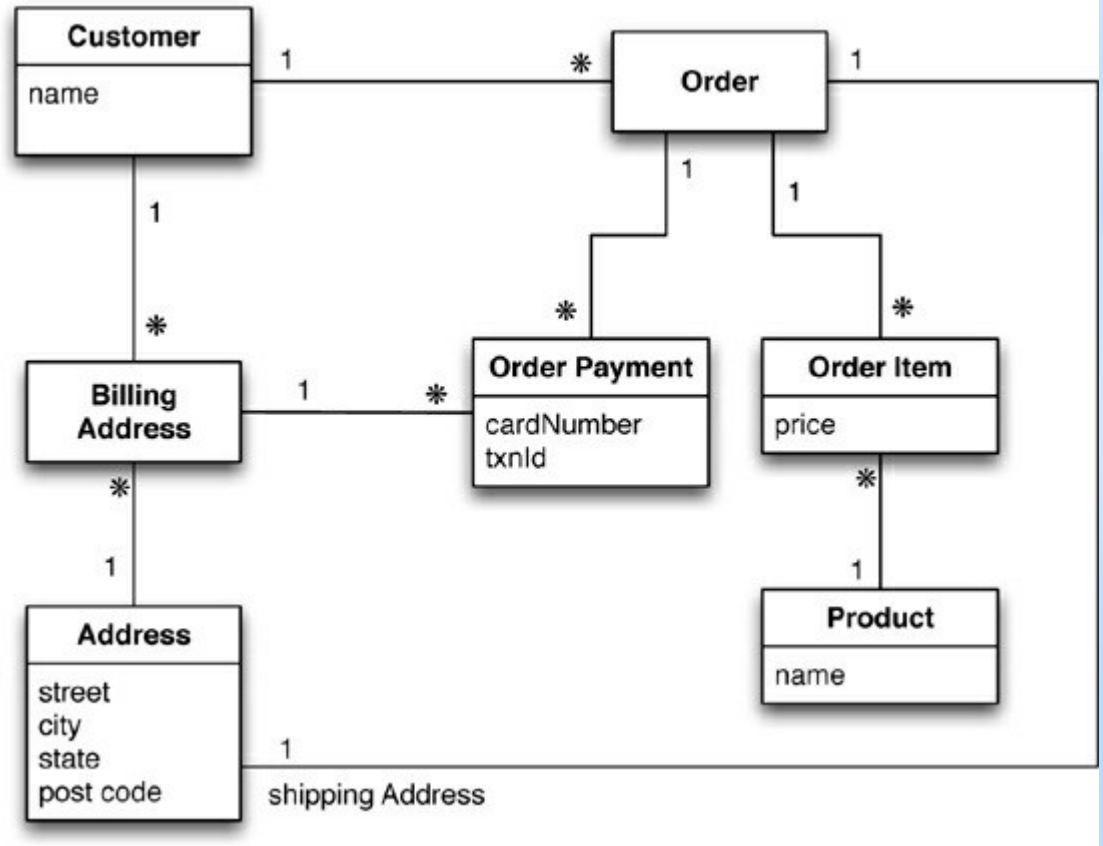
- A veces se denominan incorrectamente “modelos de datos” (nosotros lo haremos así)
- En los últimos años ha predominado el modelo relacional. Pero veremos otros 4 recientes:
  - Tres trabajan sobre “orientación a agregación de datos” (término de *Domain-driven design*):
    - Key-value meta-data model
    - Document meta-data model
    - Colum-family stores meta-data model
  - Otro es distinto:
    - Graph meta-data model

Agregación de datos

- En el modelo relacional se trabaja con tuplas individuales
- En los modelos basados en agregaciones se hace con estructuras más complejas: registros que incluyen listas y otras estructuras de datos internamente
  - A estas agregaciones serán a las que se le apliquen las operaciones y se garantice consistencia
  - Estas agregaciones suelen ser más cercanas a las estructuras de datos que usan los programadores y facilitan el escalado usando clústeres

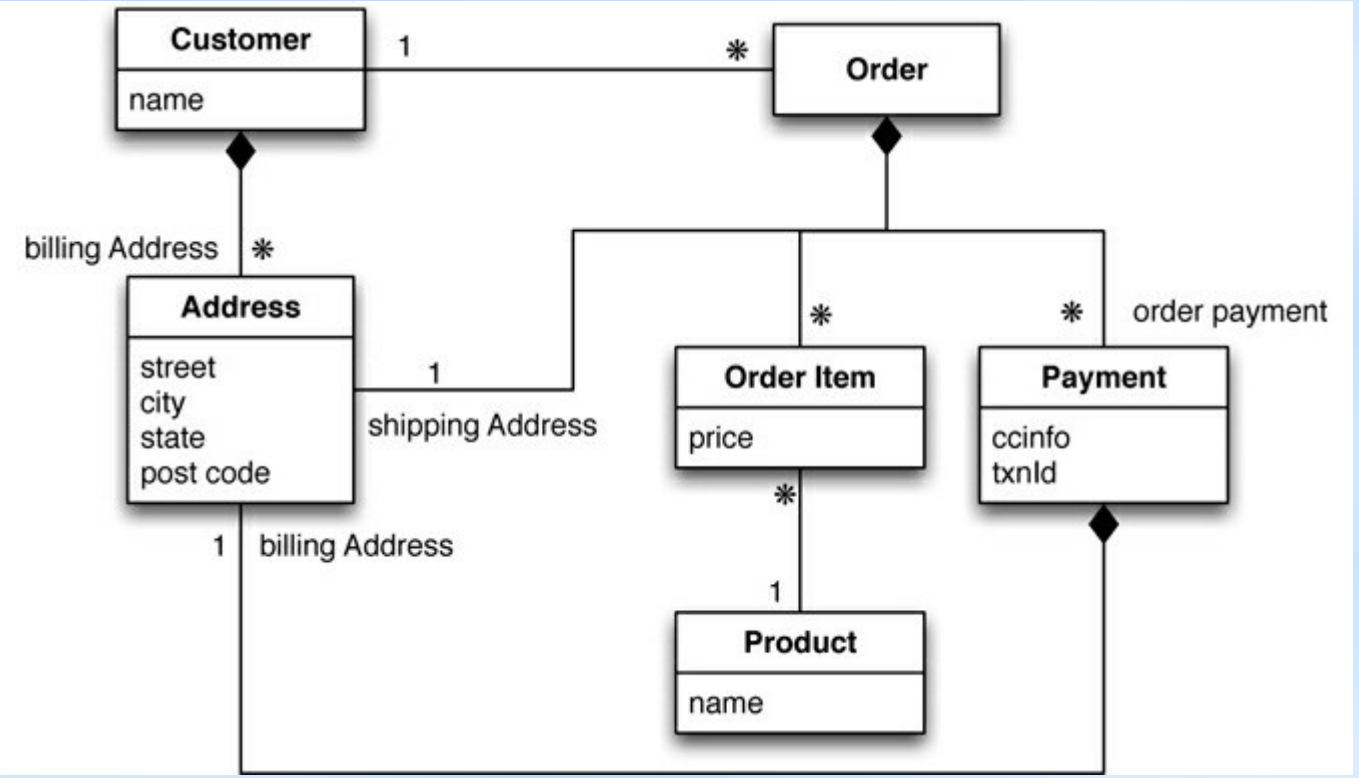
Agregación de datos

- Ejemplo de modelo relacional:



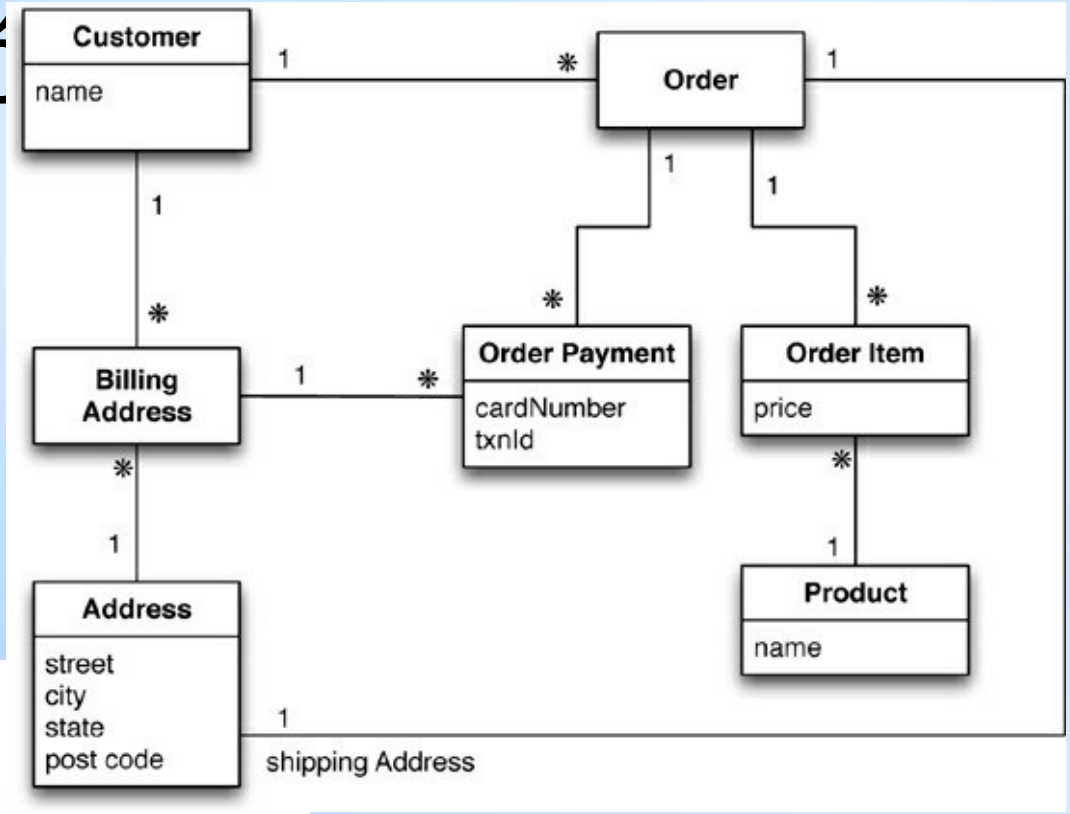
Agregación de datos

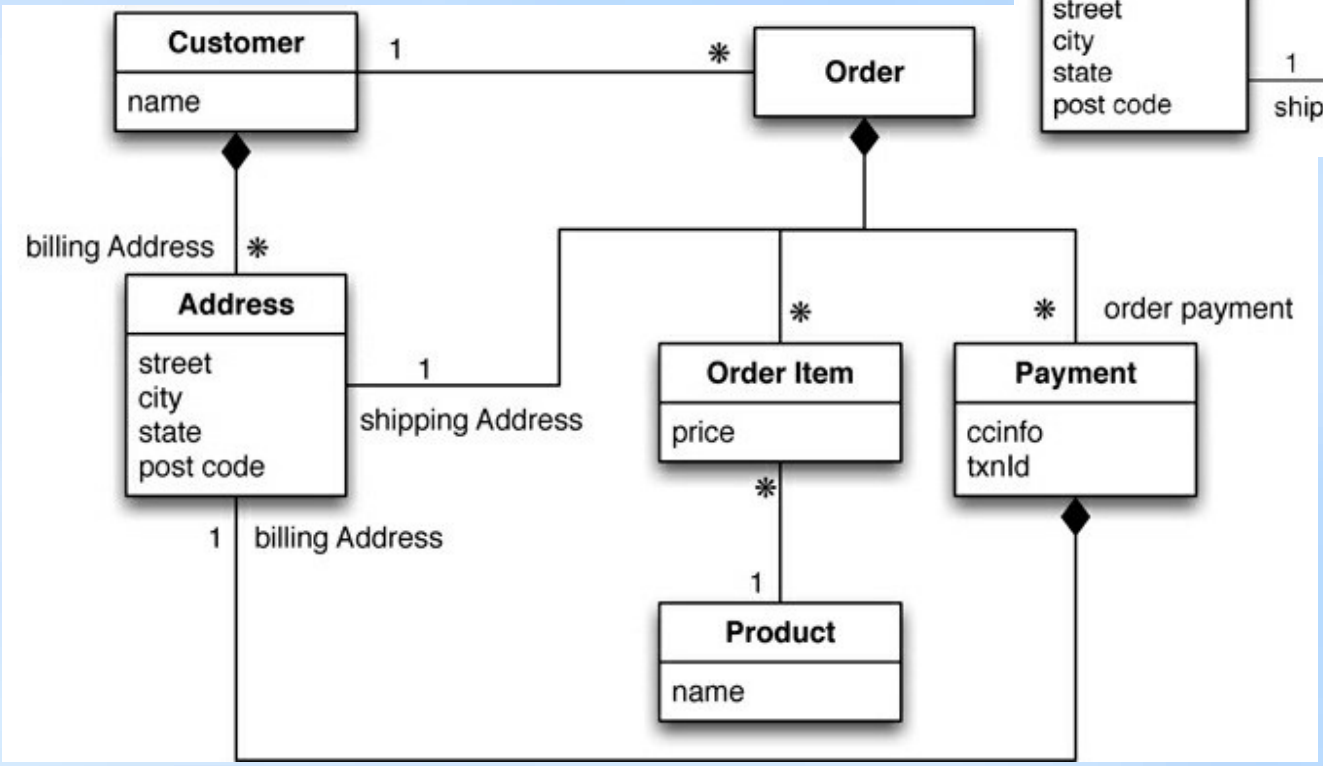
- Ejemplo de modelo agregado (notación UML, el diamante es una agregación):
  - No es necesario “billing address”
  - Address, order item y payment agregan juntos a Order



Agregació

- Los datos se copian por cada “order”





Relacional
Versus
Agregado

Agregación de datos

- Es cierto que el modelo relacional puede gestionar agregaciones, pero lo hace igual que cualquier otra relación, usando CF
  - No da herramientas ni semántica diferenciada
  - En los modelos agregados sí existen, y están al servicio de cada aplicación concreta que usa la BDD
    - Porque cada agregación será adecuada para determinados propósitos de una aplicación pero incómodos para otras

Agregación de datos

- Al informar al SGBD de esto la refleje como tal en los clústeres
  - De este modo se minimizan (o al menos se conoce) los clústeres a los que afecta cada operación
  - Tiene el problema de que dificulta garantizar transacciones ACID (Atomic, Consistent, Isolated, and Durable) en toda la BDD
  - Pero facilitar mucho hacerlas en un clúster

Agregación de datos

- Por lo tanto, si afecta comúnmente a varias agregaciones hay que considerar gestionarlos a nivel de aplicación
  - O si no convence, “romper” en el nivel más grande que responda a los intereses de todas las aplicaciones
  - Ejemplo:
    - A una aplicación le interesa *datos_cli* y *pedidos*
    - A otra *pedido* y *tipo_pago*
    - Lo más sensato es agregar “pedidos” sólo, sin *datos_cli* ni *tipo_pago*

Agregación de datos en key-value y document

- Son dos modelos muy orientados a agregación
  - Los dos definen la BDD como lotes de agregaciones que se identifican con una clave
  - En *key-value* el contenido de las agregaciones es opaco para el SGBD (acepta imágenes, XLS, modelos 3D, etc)
  - En *document* la agregación tiene una estructura definida y el SGBD ofrece herramientas para su gestión (se puede consultar su contenido, etc)
    - Aunque en la práctica hay extensiones como Solr que dan a un SGBD key-value algunas caract de este tipo

Agregación de datos en column

- El tipo *column* nació de *Google Big Table*:
  - Matrices enormes con campos poco poblados y esquema no definido
    - Es el fundamento de Hbesa y Cassandra
      - No confundir como productos pre-SQL como C-store
  - Se pueden considerar un mapeado a 2 niveles
    - Cada fila tiene clave única y columnas concretas
    - Las columnas se agrupan en familias que se supone que (normalmente) se accederán a la vez
    - Ej: row-key es cod_cli, que da acceso a 2 familias:
      - Datos personales (únicos)
      - Pedidos (lista con posible orden interno)

Agregación de datos en column

- Esta estructura en familias se suele reflejar en el diseño físico y mejora el rendimiento
- A diferencia de la document, que ve cada documento como una unidad estructural, aquí cada fila es una serie de columnas independiente
  - Aunque Cassandra permite definir “supercolumnas” que con columnas de columnas ~BigTables

Agregación de datos en column

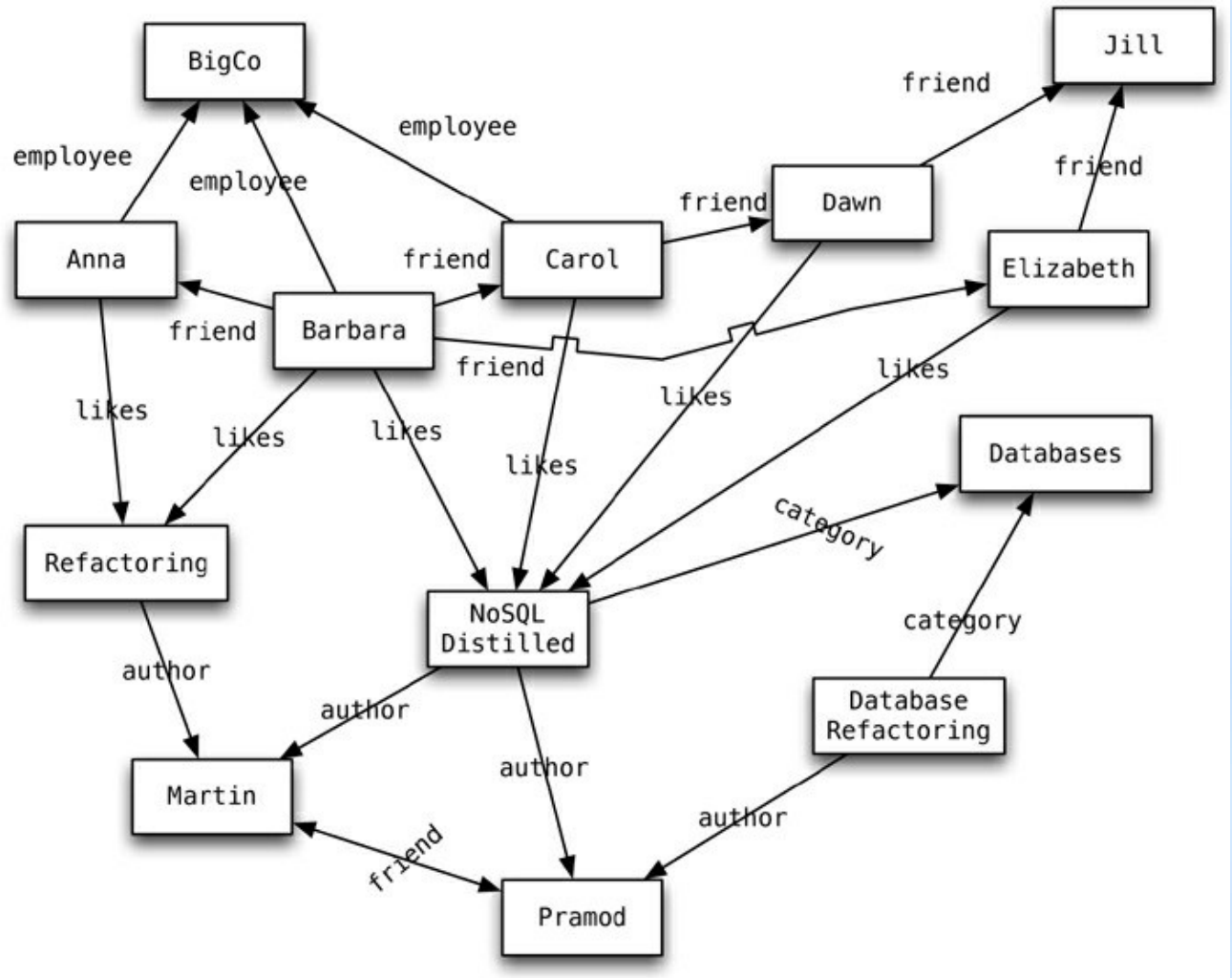
- Cuidado, las BD en columna son muy diferentes a las relacionales:
  - Las columnas se pueden añadir o eliminar dinámicamente
  - Cada fila puede tener distintas columnas
- De hecho, Cassandra permite definir
  - *skinny rows* con igual columnas para toda fila (parecido a relacional)
  - *wide rows* en las que cada fila tiene distintas columnas

Gestión de relaciones en NoSQL

- Distintos usos de un conjunto de datos pueden dar lugar a elegir un modelo de agregación que satisfaga a todos los usuarios
  - Suele ser más “fino” parecido a BD relacional
- Y después hay que relacionar las agregaciones:
  - Los SGBD NoSQL suele proveer mecanismos para informar de dichas relaciones
  - Pero su gestión es limitada: normalmente sólo garantizan las operaciones a nivel de agregación
    - Si necesitamos más garantías → Modelo relacional

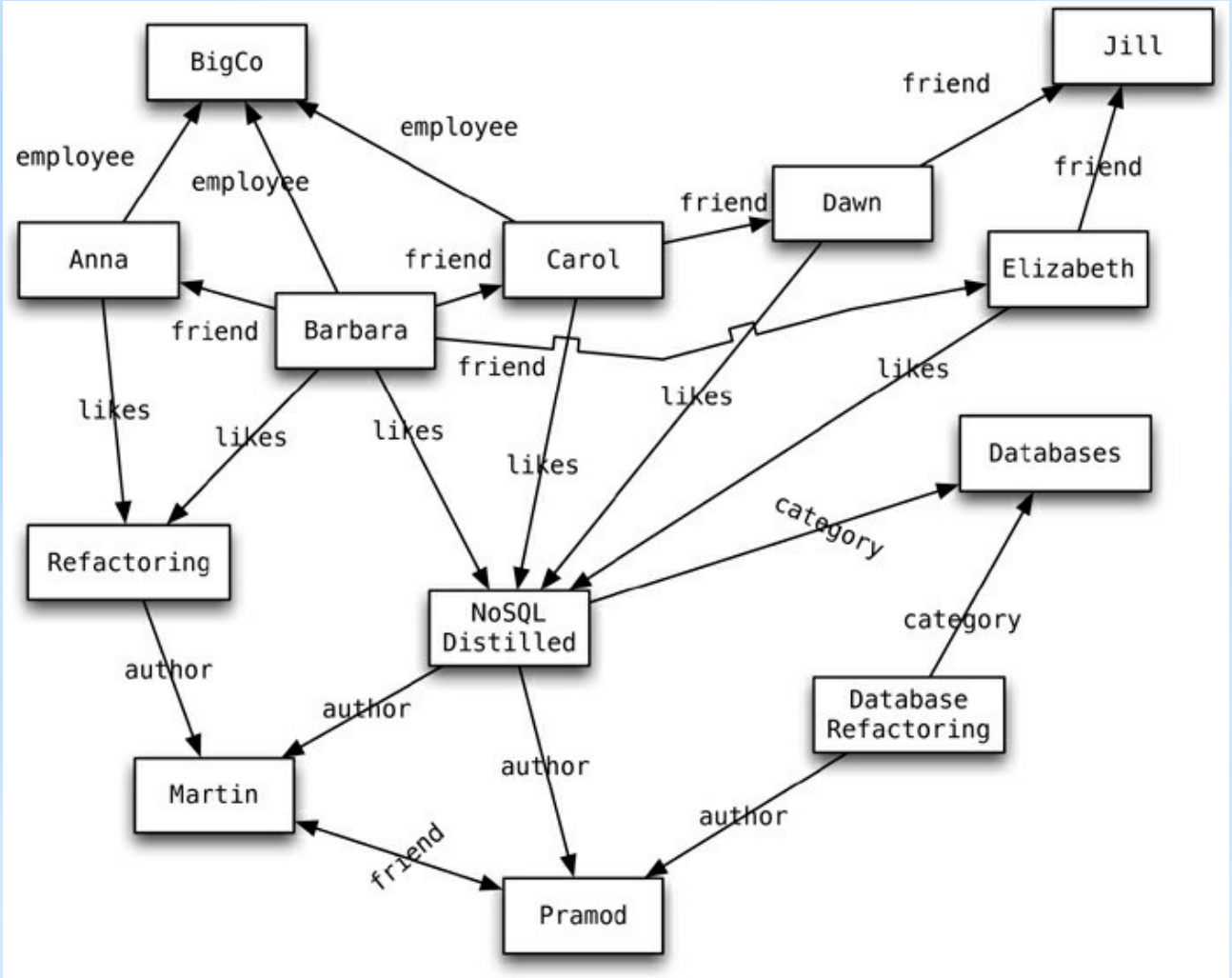
Bases de datos en grafo

- Cuando hay muchas relaciones su gestión se complica
  - Solución: BD en grafo
- Se basa en datos atómicos muy relacionados



Bases de datos en grafo

- Las consultas son recorridos de grafos en un lenguaje específico
  - Ej: amigos de Barbara que les gusta un libro escrito por un amigo de Martin



Bases de datos en grafo

- A diferencia de los otros modelos es muy potente para interrelacionar muchos datos
  - Que pueden estar dispersos, descubrirse, etc
- Sus lenguajes de consulta suelen ser mucho más rápidos que los JOIN de BDR
  - Pero las inserciones más lentas
- Funcionan mejor para garantizar operaciones en un servidor centralizado (a diferencia de los tres modelos basados en agregaciones)
- Base del Open-data

BDD sin esquema

- Los 4 modelos NoSQL vistos son muy flexibles a cambios de esquema
  - De hecho se puede ir definiendo a medida que se va desarrollando el proyecto
- El problema es que normalmente la información del esquema (significado de campos, tipos de datos almacenados, valores correlaciones, etc) no está centralizado en el SGBD …
  - Está en el código de (varias) aplicaciones

BDD sin esquema

- Distribuir la definición del esquema puede provocar problemas de coordinación
- Una solución que se usa para centralizar el acceso a la base de datos en una aplicación (*web service/s*)
  - Pero hay que mantenerlo/s
- Otra solución es dar permisos sobre elementos concretos de una agregación a cada aplicación
  - Similar a gestión de permisos sobre agregaciones en vez de sobre relaciones

Materialized views

- Las vistas en un SGBDR permiten controlar y facilitar el acceso a los datos
  - Por ejemplo, dando acceso a parte de los datos o resumiéndolos en datos derivados
- En NoSQL también existen, de 2 formas:
  - Con actualización en tiempo real: bueno si hay muchas lecturas y pocas escrituras
  - Con un *batch* que garantice validez en un intervalo temporal (similar a *snapshot* de SGBDR)

Modelos de distribución

- Existen distintas alternativas *combinables*:
  - Servidor único: es lo ideal
    - Probl: costoso, poco flexible, centraliza fallos y puede no llegar al rendimiento deseado
  - Sharding (escalabilidad horizontal): dividir los datos en *n* servidores, dejando en cada uno de ellos los datos a los que accede cada aplicación concreta
    - Los SGBD NoSQL suelen ofrecer sharding automático
      - (más estático) por localizaciones y (más dinámico) por workload
      - Junto con una buena caché dispara la respuesta
    - Probl: que el sharding no encaje con el uso real y se mantiene la centralización de fallos

Modelos de distribución

- Existen distintas alternativas:
  - Replicación maestro-esclavo:
    - Replica información en varios nodos: uno es el maestro (centraliza consistencia) y otros esclavos (resp. Rápida)
    - No centraliza fallo (los datos siempre están replicados)
    - Probl: las actualizaciones el maestro y propagación a esclavos pueden ser un cuello de botella
  - Replicación en pares (*peers*): red de servidores que todos admiten (lecturas y) escrituras
    - Soluciona el cuello de botella del Master
    - Probl: se complica mucho la consistencia de datos

Consistencia

- Los SGBDR ofrecen consistencia fuerte
- Los NoSQL suelen ofrecer otras más débiles:
  - Update consistency
  - Read consistency
  - Teorema de CAP: sólo se pueden tener dos de las tres propiedades siguientes:
    - Consistencia
    - Disponibilidad (*avalilability*)
    - Tolerancia a particionar (*partition tolerance*)
  - Compromisos:
    - Garantías temporales con *timestamps*
    - Uso de *map-reduce*

Ejemplos comerciales

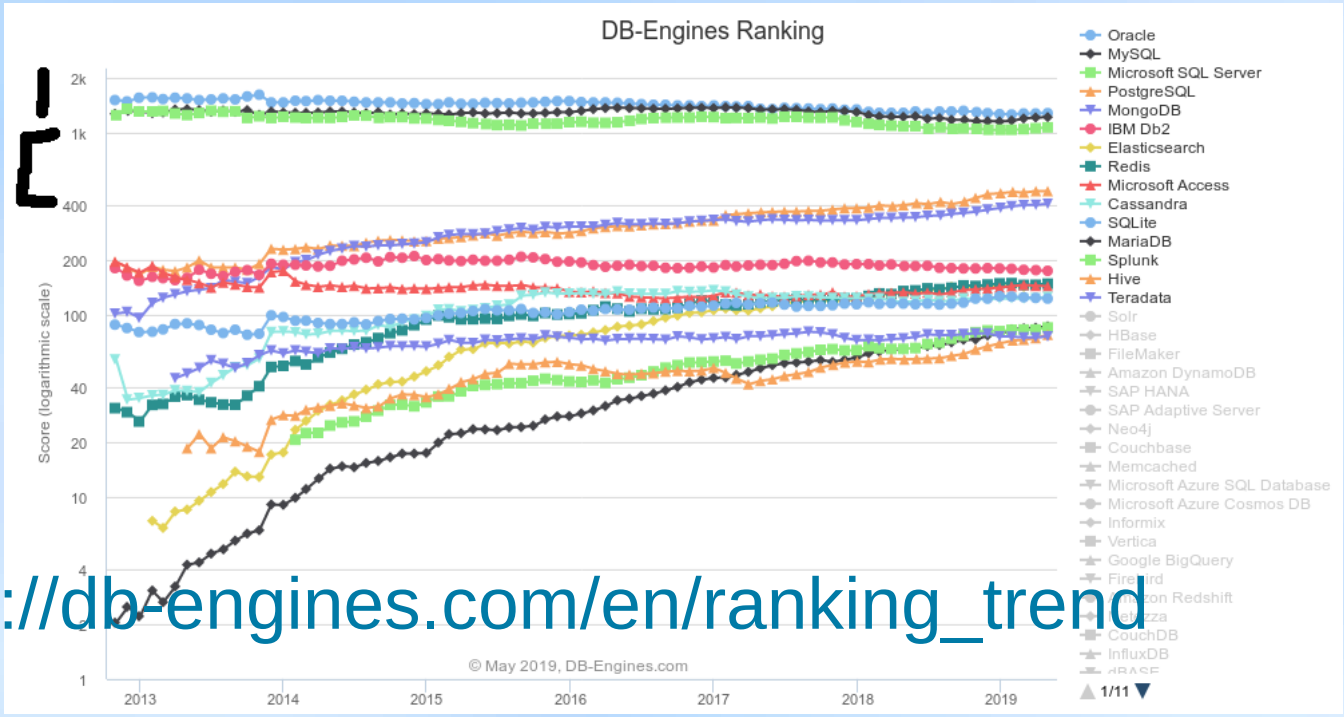
- Ejemplos:
  - Riak ejemplo key-value
  - MongoDB ejemplo de document
  - Cassandra ejemplo column-family
  - Neo4JSGBDR ejemplo de grafo

Resumen

- Entonces, ¿uso SQL o NoSQL?
  - Esa decisión la debe tomar la persona responsable de gestión del proyecto según:
    - perfil del personal que trabajará en el proyecto (considerando futuras contrataciones y rotación de personal)
    - el ecosistema tecnológico de la empresa (o del proyecto)
    - presupuesto para adquisición de software, certificación (si procede) y formación
    - políticas tecnológicas de la empresa
    - plazos de entrega
    - requisitos del proyecto:volumen de datos, tiempo de respuesta, ...
    - Etc.
  - Igual que para elegir el sistema operativo de los servidores, lenguaje(s) de programación o la metodología a usar

Resumen

- El objetivo en esta asignatura es conocer qué alternativas hay al modelo relacional
  - Hay otra asignatura para aprender a usarlas
- Tras una década del “boom”
  - Según https://db-engines.com/en/ranking_trend



Referencias

- Manual oficial de MySQL 5.7
  - 13.1.8 ALTER TABLE Syntax
    - https://dev.mysql.com/doc/refman/5.7/en/alter-table.html
- Documentación Oracle 11g.1:
  - ALTER TABLE
    - https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_3001.htm#SQLRF01001
- *Surrogate key https://en.wikipedia.org/wiki/Surrogate_key*
- *NoSQL Distilled*, P. J. Sadalage y M. Fowler. Editorial Addison-Wesley, 2012
- Why SQL is beating NoSQL, and what this means for the future of data
  - https://blog.timescale.com/why-sql-beating-nosql-what-this-means-for-future-of-data-time-series-database-348b777b847a
- TFM: Autoría de aplicaciones móviles para el análisis de datos (Tatiana Person)
  - http://rodin.uca.es/xmlui/handle/10498/19961

Gracias por la atención

¿Preguntas?