MAC0439
Modelagem de BDs

Uma Breve Introdução aos Sistemas NoSQL

Profª. Kelly Rosa Braghetto

14 de março de 2018

NoSQL – o nome

- “SQL” = SGBD **relacional** tradicional
- Na última década, reconheceu-se que:
  - **Nem todo problema de gerenciamento e análise de dados é melhor solucionado usando um SGBDR tradicional**
- **“NoSQL” = “No SQL” = não usar um SGBDR tradicional**
- **“NoSQL” ≠ não usar a linguagem SQL**

NoSQL – o nome

- Uma segunda interpretação possível:
  - **Nem todo problema de gerenciamento e análise de dados é melhor solucionado usando exclusivamente um SGBDR tradicional**
- **NoSQL = “Not Only SQL”**

SGBDR ou não SGBDR – eis a questão...

- Um SGBDR tradicional provê
  - Eficiência
  - Confiabilidade
  - Conveniência
  - Armazenamento e acesso seguros (para multiusuários)

para quantidades massivas de dados persistentes

Sobre a Conveniência

- Modelo de dados simples (relacional)
- Linguagem de consulta declarativa (SQL)
- Garantias transacionais

Sobre a Conveniência

- Modelo de dados simples (relacional)
- Linguagem de consulta declarativa (SQL)
- Garantias transacionais

Todas essas características parecem ótimas!

Por que então, para algumas aplicações, é vantajoso abrir mão do SBGD?

Na opinião de Michael Stonebraker (*)...

- Há duas razões principais que motivam a troca de um SGBDR por um sistema NoSQL:
  - **Necessidade de um desempenho melhor**
  - **Necessidade de mais flexibilidade**

(*) em “SQL Databases vs. NoSQL Databases”, Communications of the ACM, abril de 2010.

http://dl.acm.org/citation.cfm?id=1721659

Detalhe importante: o Stonebraker é o ganhador do prêmio Turing de 2015!

Sobre a Conveniência

- As “conveniências” são garantidas por funcionalidades que vêm embaladas em um pacote indivisível num SGBDR.

  E elas têm os seus respectivos custos.

  - Uma aplicação não consegue “abrir mão” de alguma(s) dessas conveniências, a fim de se livrar do custo associado.

Dados X Modelo Relacional

- Dados ficam organizados em relações
- Há uma álgebra “compreensível” sobre relações
  - Base da linguagem de consulta (a SQL)
  - Tudo se encaixa muito bem em um mesmo pacote!
- Problema: E quando os dados não se encaixam muito bem nesse pacote? Ou seja, o que acontece quando os dados não são relacionais?
  - Antes que possam ser carregados no BD relacional, os dados terão que passar por um processo de reorganização

O Poder (e o Custo) da SQL

- A SQL é uma linguagem de consulta e modificação de dados muito poderosa. Ela inclui:
  - Seleções, projeções, junções, agregações, operações sobre conjuntos, predicados

- **Às vezes, a SQL oferece muito mais funcionalidades do que uma dada aplicação precisa de fato**
  - Exemplo: há aplicações que só precisam fazer recuperações simples, baseadas na chave do registro
  - Nesses casos, usar um sistema que implementa uma linguagem de consulta complicada é um custo alto demais para se arcar

A Importância (e o Custo) das Garantias Transacionais

- Transações são muito importantes quando há vários usuários acessando os mesmos dados e os requisitos de consistência são rígidos
- Mas em algumas aplicações em que esses requisitos são menos rígidos, até mesmo as garantias mais fracas impostas pelos SGBDs tradicionais podem não ser apropriadas

Confiabilidade

- Confiabilidade é algo que queremos em qualquer sistema de BD
- Mas é possível lidar com ela de forma diferente em aplicações executadas em modo “batch” ou aplicações de análise de dados
  - Nesses casos, na ocorrência de uma falha, uma estratégia viável para garantir a confiabilidade é simplesmente refazer o processamento todo
  - Essa estratégia não é aplicável, por exemplo, às transações “online” realizadas em um site de vendas de produtos que interage com um SGBD relacional

Dados em grande volume

- É uma das razões para o surgimento dos sistemas NoSQL
- **O volume dos dados manipulados hoje em dia é muito maior do que o volume para o qual os SGBDs tradicionais foram projetados**
- Alguns motivos para o aumento do volume de dados:
  - Grande queda do custo de dispositivos de armazenamento secundário
  - Coletas de dados a altas taxas, feitas por sensores em variados tipos de dispositivos (como celulares, câmeras, etc.) e por web sites (como Facebook, Twitter, etc.)

Eficiência

- **As aplicações da atualidade têm requisitos de desempenho que são muito mais rígidos que antigamente**
- Para aplicações web, eficiência no tempo de resposta é crucial
  - Milhões (e até bilhões) de registros
  - Mesmo para consultas envolvendo operações complexas sobre essa quantidade de registros, o tempo da resposta deve ser menos de 1 segundo

+ Dados, + Flexibilidade, + Eficiência

- Principais motivações para o desenvolvimento dos sistemas NoSQL
- **A proposta dos sistemas NoSQL é "sacrificar" alguns dos benefícios providos pelos SGBDRs tradicionais** em prol de mais:
  - capacidade de armazenamento
  - flexibilidade na representação dos dados
  - eficiência de acesso ao dados

Sistemas NoSQL – uma definição mais completa

- São uma alternativa aos SGBDs relacionais tradicionais, para lidar com grandes quantidades de dados
- Possuem como **vantagens** sobre o SGBDRs tradicionais:
  - **Esquema mais flexível**
  - **Inicialização mais rápida e barata**
  - **Escalabilidade para grandes volumes de dados (tanto para armazenamento, quanto para a eficiência no acesso aos dados)**
  - **Consistência relaxada, que resulta em melhor desempenho e disponibilidade**

Sistemas NoSQL – uma definição mais completa

- Possuem como **desvantagens** sobre o SGBDRs tradicionais:
  - **A ausência de uma linguagem de consulta declarativa e padronizada**
    - Maior esforço para os desenvolvedores, que precisam implementar as operações de manipulação dos dados usando linguagens de programação
  - **Consistência "relaxada"**
    - Menos garantias com relação à consistência dos dados

Sistemas NoSQL – Resumo das características

+ Esquema mais flexível
+ Inicialização mais rápida e barata
+ Escalabilidade para grandes volumes de dados
+ Consistência relaxada → melhor desempenho e alta disponibilidade

- Ausência de linguagens de consulta declarativas → mais programação
- Consistência relaxada → menos garantias

Seis características comuns de sistemas NoSQL (*)

1) A habilidade de escalar horizontalmente *operações simples* em vários servidores
2) A habilidade de replicar e distribuir (particionar) dados em vários servidores
3) Substituição da “comunicação” via SQL por APIs simples
4) Um modelo de controle de concorrência mais fraco que o das transações ACID usadas nos SGBDRs
5) Uso eficiente de índices distribuídos e memória RAM para armazenamento dos dados
6) A habilidade de adicionar dinamicamente novos atributos aos registros de dados (ausência de esquema fixo – *schemaless*)

(*) Segundo Rick Cattel em “Scalable SQL and NoSQL Data Stores”, SIGMOD Record, December 2010

Definições importantes

- **“Operações simples”**
  - Busca de um registro por meio de sua chave (*key lookup*), leituras e escritas de um registro ou de um número pequeno de registros
  - Constituem o tipo de operação mais frequente nas aplicações web atuais
- **“Escalabilidade horizontal”**
  - Habilidade de distribuir os dados e o processamento de operações simples em vários servidores (nós), sem o compartilhamento de RAM ou disco entre os servidores → **arquitetura *“shared nothing”***
  - Esse tipo de escalabilidade geralmente é mais barata que a vertical, principalmente quando usa *commodity hardware*

Obs.: Escalabilidade vertical → ocorre quando o SGBD usa vários *cores* que compartilham a mesma RAM e disco, ou seja, o SGBD está em uma única máquina

Escalabilidade horizontal

- Pode ser feita de duas maneiras:
  - **Particionando (*sharding*) os dados** entre os nós
    - Particionamento horizontal: cada nó armazena um subconjunto das “linhas” do BD
    - Particionamento vertical: cada nó armazena um subconjunto das “colunas” do BD
  - **Replicando os dados** nos nós
- Possibilita que um grande número de operações simples (principalmente consultas) sejam realizadas a cada segundo → **paralelização**

Replicação de dados

- Pode ter dois objetivos diferentes (e não mutuamente exclusivos):
  - Tolerância a falhas
  - Balanceamento de carga de operações de acesso a dados
- A propagação das atualizações entre as réplicas pode ser:
  - **Assíncrona** → garante **"eventual consistency":** não há a garantia de que um dado lido é o mais atual, mas há a garantia de que as atualizações serão propagadas para todos os nós em algum momento do tempo
  - **Síncrona** → garante consistência
- Quando a replicação é assíncrona, uma falha em um nó pode causar perda irreversível de dados!

Armazenamento dos dados

- Os dados e os índices podem ser mantidos
  - No disco (HD, SSD)
  - Na memória RAM, com tolerância a falhas por meio de persistência em disco ou replicação
    - “Inspiração” veio do sucesso do ***memcached***

memcached

*“**memcached** is a high-performance, distributed memory object caching system, generic in nature, but originally intended for use in speeding up dynamic web applications by alleviating database load.”*

*“**memcached** is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.”*

“a short-term memory for your applications. “

http://memcached.org/

Problemas de Consistência

- Tipos de conflitos
  - **Escrita-escrita** → quando dois clientes tentam escrever sobre os mesmos dados ao mesmo tempo
  - **Leitura-escrita** → quando um cliente lê dados inconsistentes no meio da escrita de outro cliente
- Sistemas distribuídos têm conflitos do tipo leitura-escrita porque alguns nós podem ter recebido uma atualização de dados enquanto outros não

Conflitos e Consistência

- Há duas abordagens para evitar conflitos:
  - **Pessimista** → bloqueia os registros de dados, para evitar os conflitos
  - **Otimista** → detectam os conflitos e depois tratam deles
- Para obter boa consistência, é preciso envolver muitos nós nas operações de dados
  - Mas isso aumenta a latência (tempo de resposta das operações)

Controle de concorrência (Consistência de alterações)

Mecanismos empregados nos sistemas NoSQL:

- **Bloqueios** (*locks*) [locais ou globais], no nível de registro
- ***Multiversion-Concurrency Control* (MVCC)**
- Sem controle algum

Sistemas NoSQL – Modelos de Dados

Uma Visão Geral

baseada no livro

NoSQL Distilled – A Brief Guide to the Emerging World of Polyglot Persistence,

de Pramod Sadalage e Martin Fowler

Relembrando: Modelo Relacional

- Um **BD relacional** é um **conjunto de relações**
- Cada **relação** é um **conjunto tuplas**
- Cada **tupla** representa uma **entidade** ou **relacionamento**, descrito por meio de **atributos** atômicos (que admitem um único valor)
- Um **atributo** em uma tupla **pode fazer referência a um outro atributo** de uma tupla da mesma relação ou de uma outra relação (estabelecendo assim **relacionamentos**)

Sistemas NoSQL – Modelos de Dados

- Cada sistema NoSQL usa um modelo de dados diferente
- Esses modelos podem ser divididos nas seguintes categorias:
  - **Chave-valor**
  - **De documentos**
  - **Famílias de colunas**
  - **Grafos**

orientados a **agregados**

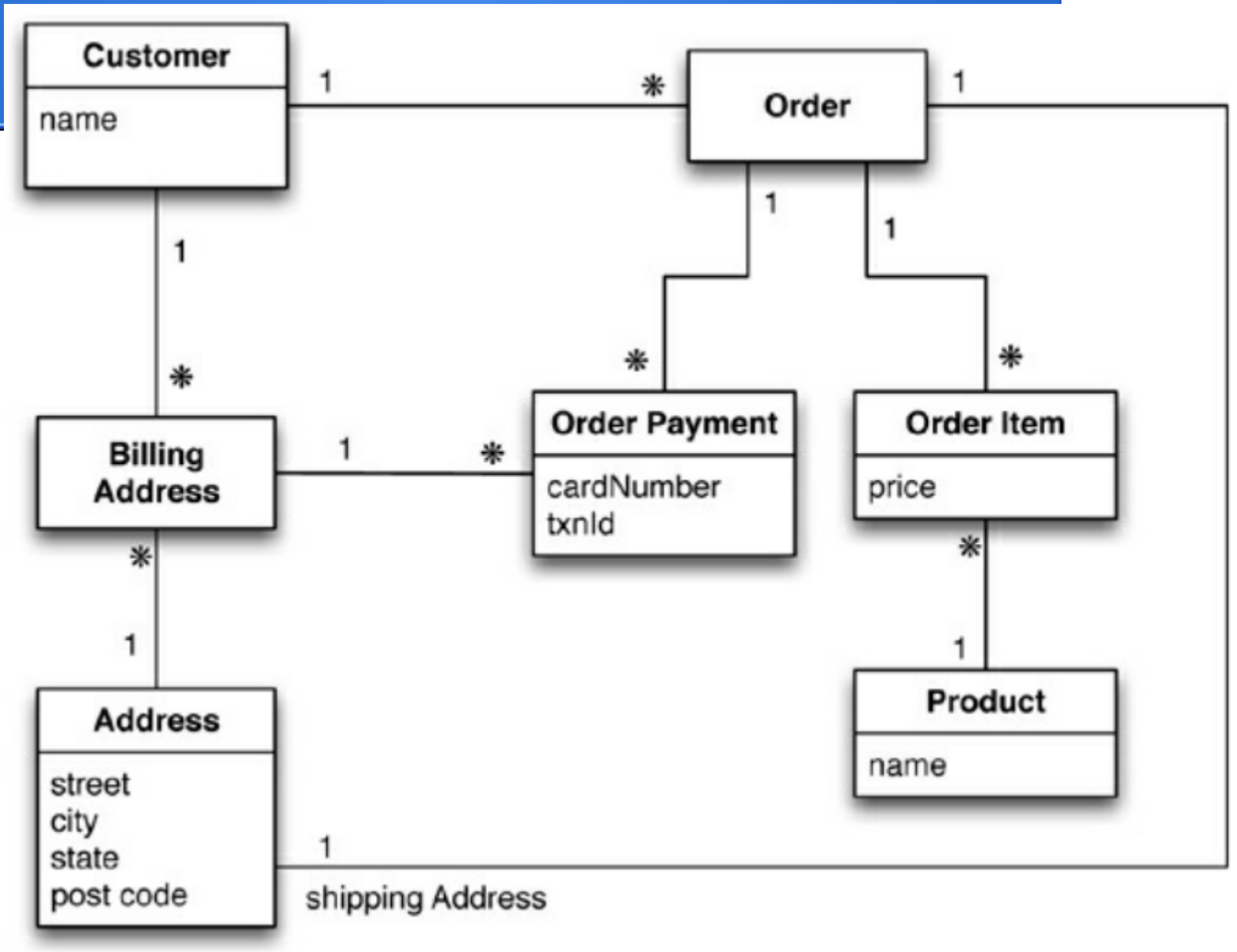
Agregados

- Um agregado é um conjunto de objetos relacionados, os quais queremos tratar como uma **unidade de dados**
  - É uma **unidade para manipulação dos dados e gerenciamento da consistência**
- Geralmente
  - Um agregado tem uma estrutura mais complexa que a de uma tupla do modelo relacional
  - A modificação de um agregado é implementada como uma operação atômica nos sistemas NoSQL
  - A comunicação com o componente de armazenamento dos dados é feita em termos de agregados

Agregados

- Agregados facilitam o funcionamento dos Sistemas NoSQL em aglomerados de máquinas
  - Nesses casos, deseja-se minimizar os número de nós que precisam ser acessados para se responder uma consulta
  - Os agregados indicam quais dados vão ser manipulados juntos e que, portanto, deveriam ficar num mesmo nó
  - Eles constituem uma unidade natural para replicação e particionamento
- Agregados também podem facilitar a vida dos programadores, já que eles têm uma estrutura mais próxima da dos objetos de dados em memória manipulados nas aplicações

Relações X Agregados



Esquema relacional para o BD de um website de comércio eletrônico

Relações X 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

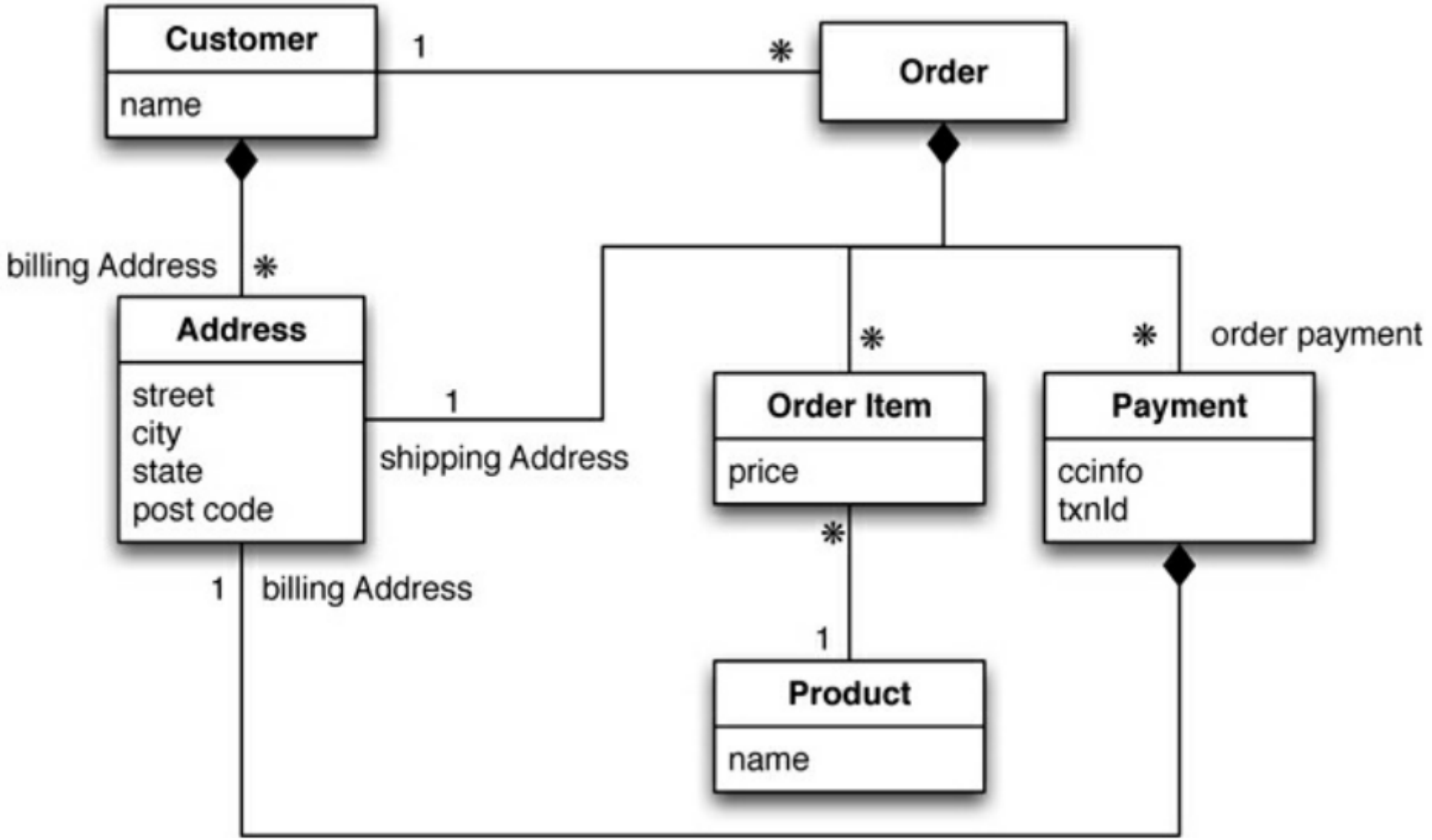
OrderItem			
Id	OrderId	ProductId	Price
100	99	27	32.45

Address	
Id	City
77	Chicago

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

Esquema relacional para o BD de um website de comércio eletrônico

Relações X **Agregados**



Um modelo baseado em agregados para um website de comércio eletrônico

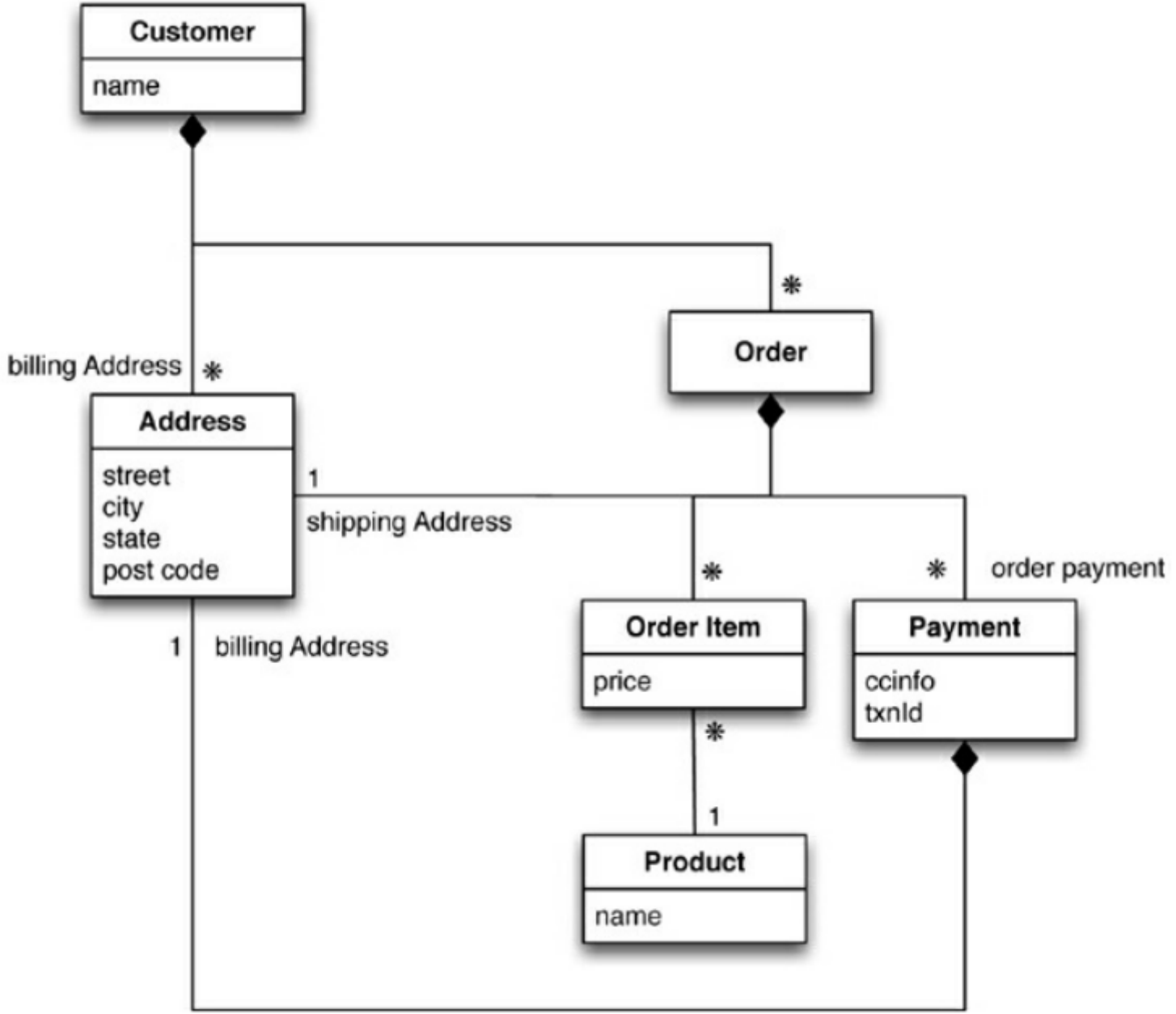
Relações X **Agregados**

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


Um modelo baseado em agregados para um website de comércio eletrônico

Relações X **Agregados**



Outro modelo baseado em agregados para um website de comércio eletrônico

Relações X **Agregados**

```
{
    "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"}
                    }
                ]
            }
        ]
    }
}
```

“Contras” dos Agregados

- Não há um conceito de “chave estrangeira” para impor integridade aos relacionamentos entre agregados
- Um sistema NoSQL não consegue usar a estrutura interna do agregado para ajudar nas decisões relacionadas ao armazenamento e distribuição dos dados
- Geralmente, nos sistemas NoSQL não é possível manipular múltiplos agregados de forma atômica (ou seja, não há transações ACID envolvendo mais de um agregado)
- O agregado não representa uma organização lógica “intrínseca” aos dados; ele reflete a forma como as aplicações acessam os dados
  - Quando é preciso acessar os dados de diferentes formas, uma modelagem lógica tradicional (como no modelo relacional) pode ser mais apropriada

Bancos de Dados dos Tipos Chave-Valor e de Documentos

- São fortemente orientados a agregados
- Esses dois tipos de BDs são constituídos por conjuntos de agregados
  - cada agregado tem uma chave (ou ID), usada na recuperação dos dados
- Diferença entre eles:
  - **BD chave-valor**: o agregado é “opaco” para o BD
  - **BD de documentos:** o BD pode “enxergar” a estrutura do DB

Bancos de Dados dos Tipos Chave-Valor e de Documentos

- Num BD chave-valor, geralmente se acessa um agregado por meio de busca pela sua chave
  - O suporte à definição de índices/chaves secundárias não é implementado em todos os sistemas
- Num BD de documentos, é possível fazer buscas e criar índices sobre os campos do agregado. Também é possível recuperar partes dele (ao invés de recuperá-lo por inteiro)

Bancos de Dados Chave-Valor

- Interface extremamente simples
  - Modelo de dados: **pares (chave, valor)**
  - Operações:
    - **Insert**(chave,valor)
    - **Fetch**(chave) → busca/recuperação de um valor
    - **Update**(chave,valor)
    - **Delete**(chave)

Bancos de Dados Chave-Valor (variações)

- Interface
  - Modelo de dados: pares (chave, valor-estruturado)
  - Alguns permitem colunas (não-uniformes) dentro do valor
  - Alguns permitem que a operação de Fetch seja realizada sobre faixa de valores de chaves
    - Por exemplo, 2 < chave < 10

Bancos de Dados Chave-Valor

- Exemplos
  - Redis (http://redis.io/)
  - Memcached (http://memcached.org/)
  - Riak (http://basho.com/products/riak-kv/)
  - OrientDB (http://orientdb.com/)

Bancos de Dados de Documentos

- Interface
  - Modelo de dados: **pares (chave, documento)**
  - **Documento: dados em formatos semiestruturados (JSON, XML, etc.)**
  - Permite a definição de documentos aninhados
  - Documentos são agrupados em **coleções**
- Diferentemente dos BDs chave-valor, geralmente **nesses sistemas é possível definir índices secundários**

Bancos de Dados de Documentos

- Operações básicas:

  Insert(chave,documento), Fetch(chave),

  Update(chave,documento), Delete(chave)
- Também fazem Fetch de documentos buscando pelo seus conteúdos
  - mas não há uma linguagem de busca padrão
  - essa é uma funcionalidade que varia de implementação para implementação
- Geralmente não implementam bloqueios explícitos
  - Consequência: consistência ainda mais fraca

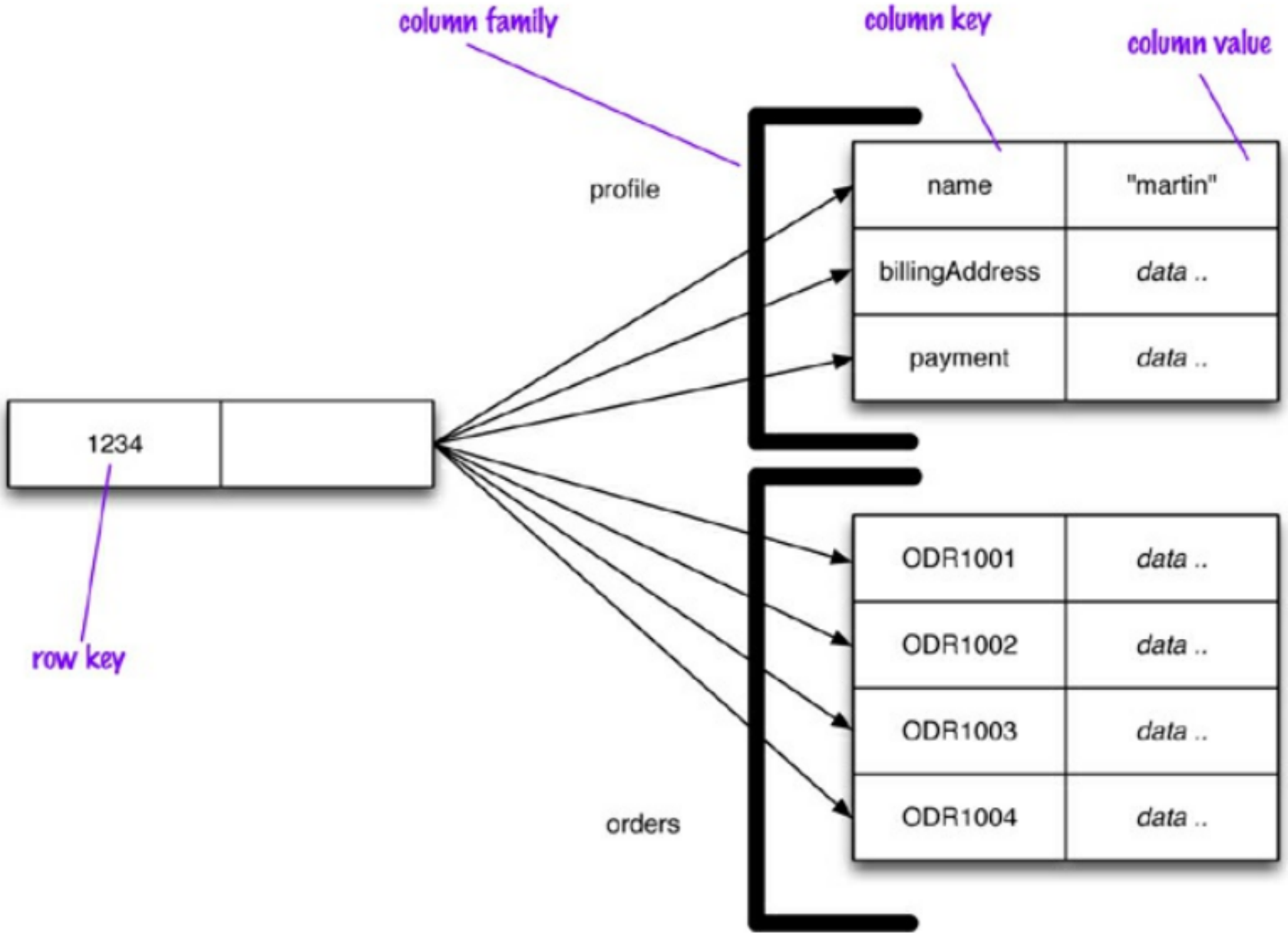
Bancos de Dados de Documentos

- Exemplos
  - MongoDB (http://www.mongodb.org/)
  - Couchbase (http://www.couchbase.com/)
  - Amazon DynamoDB (https://aws.amazon.com/pt/dynamodb/)
  - CouchDB (http://couchdb.apache.org/)

Bancos de Dados de Famílias de Colunas

- **BDs de famílias de colunas** armazenam dados como linhas, onde cada linha possui uma chave e muitas (uma família de!) colunas
- As linhas não precisam ter as mesmas colunas, e novas colunas podem ser adicionadas a qualquer linha a qualquer momento (sem a necessidades de serem adicionadas às demais linhas)
- Uma coluna pode ser uma ***supercoluna*** *(*= **mapa de colunas**), ou seja, um contêiner para outras colunas
- Uma família de colunas agrupa dados que geralmente são acessados juntos

Famílias de Colunas – Exemplo



Bancos de Dados de Famílias de Colunas

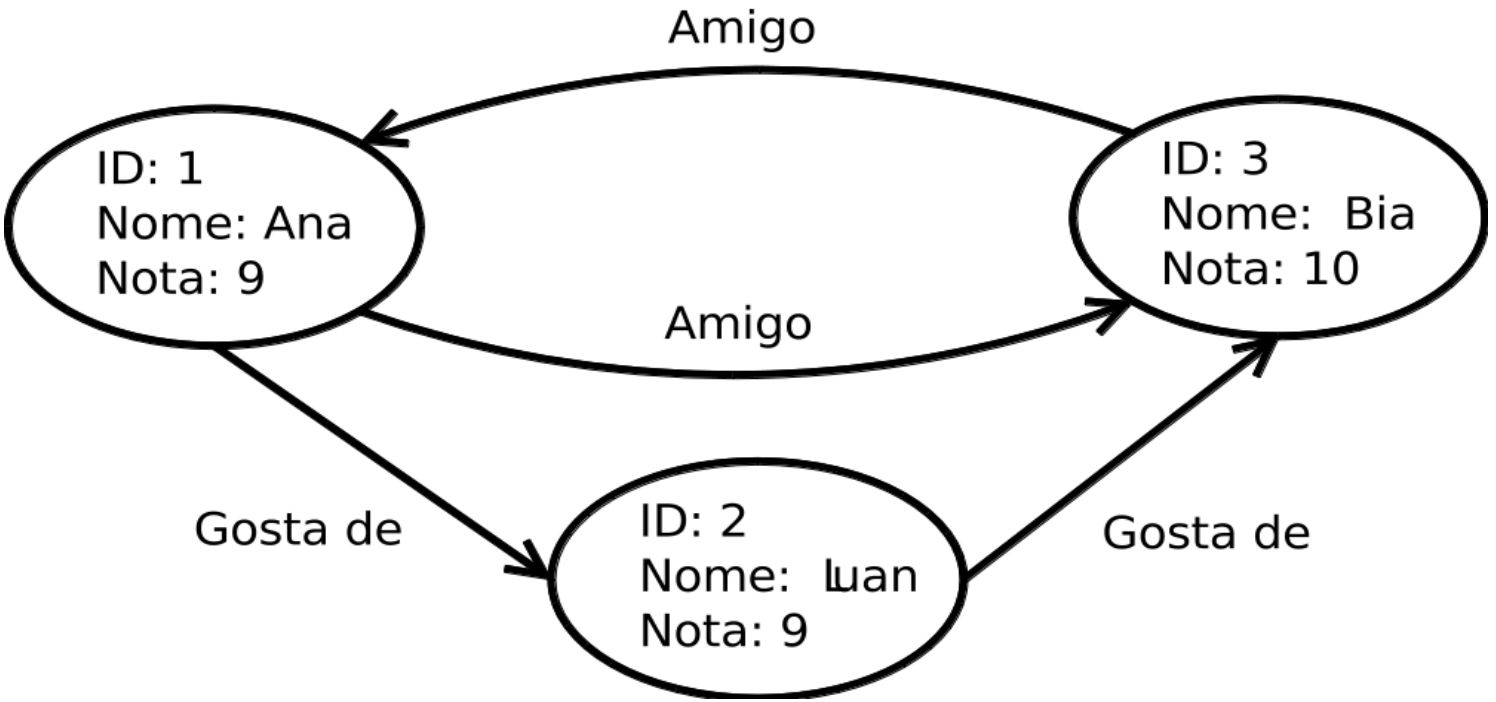
- Modelo de escalabilidade: dividir tanto linhas quanto colunas em diversos nós
  - Linhas são divididas entre os nós de acordo com o valor chave primária.
    - A divisão ocorre por faixa de valores (e não por hash) → maior eficiência para buscas de linhas segundo faixas de chaves
  - Colunas são distribuídas entre os nós de acordo com suas famílias (pré-definidas pelo programador)

Bancos de Dados de Famílias de Colunas

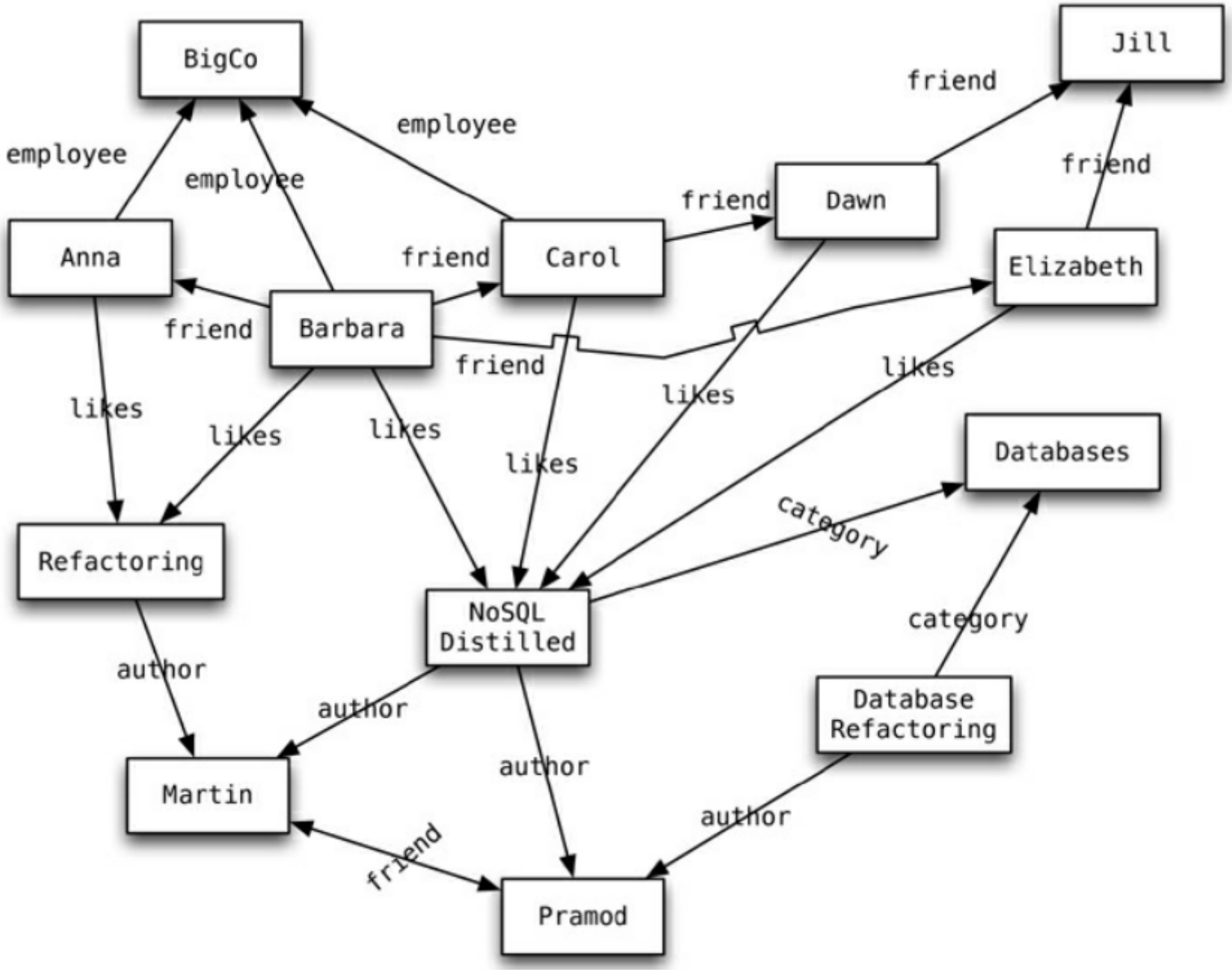
- Exemplos:
  - Precursor
    - Google Bigtable (https://cloud.google.com/bigtable/)
  - Cassandra (https://cassandra.apache.org/)
  - HBase (https://hbase.apache.org/)
  - HyperTable (http://hypertable.org/)
  - MonetDB (https://www.monetdb.org/)

Bancos de Dados de Grafos

- Modelos de dados: **nós (entidades)** e **arcos (relacionamentos)**
- Nós podem ter **propriedades** (incluindo **ID**)
- Arcos podem ter **rótulos** ou **papéis**



BD de Grafo - Exemplo



Bancos de Dados de Grafos

- Os nós podem ter diferentes tipos de relacionamentos entre si; também não há restrições quanto à quantidade de relacionamentos que os nós podem ter
- Consultas podem envolver:
  - um único passo, expressões de caminho ou recursões completas
  - condições sobre propriedades e rótulos
- Em BDs de grafos, percorrer os relacionamentos é uma operação bastante rápida (porque eles não precisam ser calculados no momento da consulta; eles já estão persistidos)
- BDs de grafos, diferentemente dos orientados a agregados, costumam:
  - implementar transações ACID
  - ser mantidos em uma única máquina

Bancos de Dados de Grafos

- Exemplos:
  - Neo4J (http://www.neo4j.org/)
  - InfiniteGraph (http://www.objectivity.com/infinitegraph/)
  - Giraph (http://giraph.apache.org/)

SGBDRs também podem ser escaláveis!

- É possível fazer particionamento e replicação de dados em SGBDRs
- Os SGBDRs, desde sua criação, já foram especializados para diferentes necessidades:
  - Data Warehouses (operações de leitura predominam)
  - BDs “in-memory”
  - BDs distribuídos
  - **BDs escaláveis horizontalmente**
    - Veja: MySQL Cluster, VoltDB, NuoDB, Clustrix
- Se o desempenho de um SGBDR for “competitivo” com o de um sistema NoSQL, porque abriríamos mão dos benefícios de se ter SQL + consistência ACID?

SGBDR ou NoSQL?

Como escolher?

(vamos discutir alguns problemas de exemplo!)

Exemplo 1 – Análise de logs Web

Cada registro no log possui:

IdUsuario, url, timestamp, infos-adicionais

- **Tarefa 1**: Carregar o log em um sistema de banco de dados
- Em um SGBD tradicional, isso requer
  - Limpeza dos dados: padronização dos formatos das datas, eliminação de registros incompletos, etc.
  - Extração dos dados: processamentos das informações adicionais, etc.
  - Verificação: verificar se os URLs são válidos, etc.
  - Especificação de um esquema
- Em um sistema NoSQL: não é preciso fazer nada!
  - Em algum momento, pode ser necessário fazer a limpeza dos dados, a verificação, etc. Mas isso pode ser feito como uma operação no modo “batch”, somente sobre os dados que serão usados de fato

Exemplo 1 – Análise de logs Web

Cada registro no log possui:

IdUsuario, url, timestamp, infos-adicionais

- **Tarefa 2**: Encontre todos os registros para...
  - Um dado IdUsuario
  - Uma dada URL
  - Um dado timestamp
- Nenhuma dessas operações requer SQL!
  - Todas se relacionam a encontrar um conjunto de registros com base em um único valor
- Se quisermos buscar alguma construção específica que apareça nas informações adicionais, a linguagem SQL não nos ajudará nisso
- Obs. importante: as operações listadas acima são altamente paralelizáveis
  - Cada uma delas envolve somente olhar para registros de forma individual
  - A uso de paralelismo para o cálculo de operações simples é uma das características mais importantes de muitos sistemas NoSQL

Exemplo 1 – Análise de logs Web

Cada registro no log possui:

IdUsuario, url, timestamp, infos-adicionais

- **Tarefa 3**: Encontre todos os pares de IdUsuarios que acessaram as mesmas URLs
- Essa consulta envolve uma operação relacional (SQL!) – uma auto-junção entre duas instâncias da tabela de log
- Entretanto, esse tipo de consulta não é frequente

Exemplo 1 – Análise de logs Web

Cada registro no log possui:

IdUsuario, url, timestamp, infos-adicionais

Registros separados para dados de usuários:

IdUsuario, nome, idade, sexo,...

- **Tarefa 4**: Encontrar a média de idade dos usuários que acessaram uma dada URL
- Essa é uma consulta com cara de SQL!
- Alguns aspectos dos sistemas NoSQL podem ser relevantes no cômpito da resposta para essa consulta
  - Consistência: quando temos um grande volume de dados e estamos interessados em informação estatísticas sobre esses dados (como a média, por exemplo), a consistência “absoluta” não precisa ser um requisito
  - Podemos contar somente alguns dos acessos a uma dada URL, enquanto outros podem ser deixados de fora (porque o BD está fragmentado ou inconsistente)

Exemplo 2 – Grafo de uma rede social

Cada registro possui:

IdUsuario1, IdUsuario2

Registros separados para dados de usuários:

IdUsuario, nome, idade, sexo,...

- **Tarefa 1**: Encontrar todos os amigos de um dado usuário
  - Essa operação não requer operações complicadas de uma linguagem de consulta; ela pode ser feita de modo bastante direto
- **Tarefa 2**: Encontrar todos os amigos dos amigos de um dado usuário
  - Essa operação requer a uma auto-junção
- **Tarefa 3**: Encontrar todas as “amigas mulheres” dos “amigos homens” de um dado usuário
  - Essa consulta requer um pouco mais do poder da SQL; mas pode ser que não seja necessário todo o poder da linguagem, porque é possível ver que há um padrão nos tipos de consultas que estão sendo realizadas

Exemplo 2 – Grafo de uma rede social

Cada registro possui:

IdUsuario1, IdUsuario2

Registros separados para dados de usuários:

IdUsuario, nome, idade, sexo,...

- **Tarefa 4**: Encontre todos os amigos dos amigos dos … amigos de um dado usuário
  - Requer um grande número de junções → algo pouco eficiente em um SGBDR, mesmo que sejam usadas consultas recursivas!
  - Nesse exemplo, a consistência também não é um requisito muito rígido, pois o banco de dados muda com frequência → soluções aproximadas são aceitáveis

Exemplo 2 – Grafo de uma rede social

Cada registro possui:

IdUsuario1, IdUsuario2

Registros separados para dados de usuários:

IdUsuario, nome, idade, sexo,...

- O tipo de operação do slide anterior sugere o uso de um sistema apropriado para realizar operações sobre grafos de grande escala
  - **banco de dados de grafos**

Exemplo 3: Páginas da Wikipedia

Grandes coleções de documentos

Combinação de dados estruturados e não estruturados

- **Tarefa**: Encontrar o parágrafo de introdução de todas as páginas sobre presidentes americanos de antes de 1900
- Esses tipos de dados e consulta não são apropriados para se manipular com um SGBDR
- E, novamente, consistência não parece ser um requisito fundamental nessa aplicação
- Solução: **bancos de dados de documentos**

Referências Bibliográficas

- Blog da ACM Sigmod (para quem gosta de assuntos de BD)

  http://wp.sigmod.org/

- Artigo: “Scalable SQL and NoSQL Data Stores”, Rick Cattell, 2010

  http://www.sigmod.org/publications/sigmod-record/1012/pdfs/04.surveys.cattell.pdf

- Livro: “NoSQL Distilled – A Brief Guide to the Emerging World of Polyglot Persistence“, de Pramod Sadalage e Martin Fowler

  http://martinfowler.com/books/nosql.html

- Site: “Ultimate Guide to the Non-Relational Universe!”

  http://nosql-database.org/

- Material do curso de banco de dados da Universidade de *Stanford*

  https://class.stanford.edu/courses/Engineering/db/2014_1