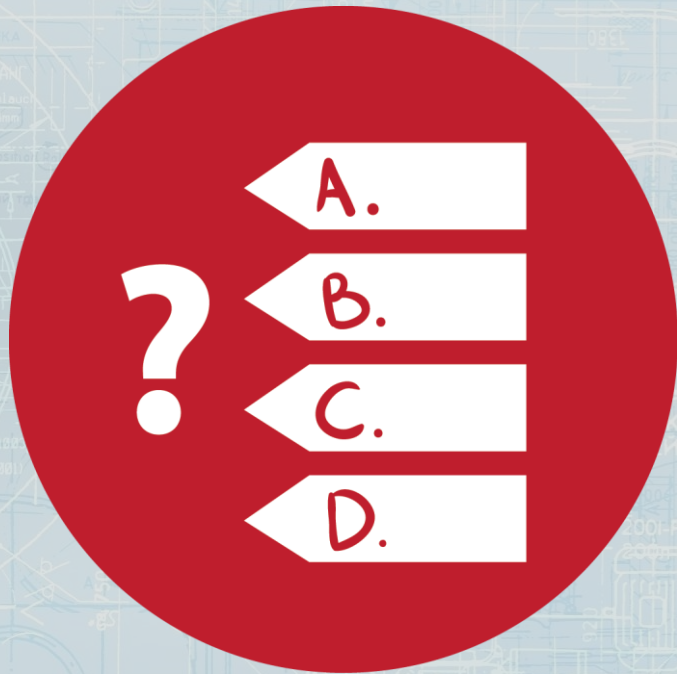


SGBD relacionais vs. NosQL

O MODELO RELACIONAL E SEUS CUSTOS



Por que (ainda) usar um SGBDR tradicional



- ❖ Eficiência
- ❖ Confiabilidade
- ❖ **Conveniência**
 - **Modelo de dados simples (relacional)**
 - **Linguagem de consulta declarativa (SQL)**
 - **Garantias transacionais**
- ❖ Armazenamento e acesso seguros
 - para multiusuários
 - para quantidades massivas de dados persistentes



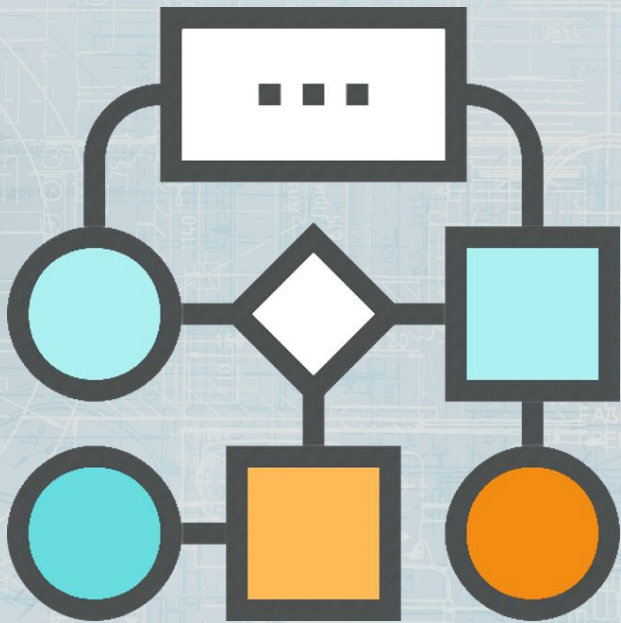
Conveniência × custo



- ❖ As conveniências de um SGBDR são garantidas por funcionalidades que vêm embaladas em um “pacote indivisível”
 - No entanto, cada conveniência tem um custo associado
 - Não é possível abrir mão de apenas algumas conveniências que não são necessárias para uma aplicação, a fim de se livrar do respectivo custo
 - **É tudo ou nada!**



Custo: modelo relacional



- ❖ Os dados
 - são organizados em relações
 - têm um formato padronizado, definido pelo esquema
- ❖ Há uma álgebra “compreensível” (operações em conjuntos) sobre as relações
 - Base da linguagem de consulta (SQL) – “JOINS”
- ❖ **Mas, e quando os dados não guardam relação entre si ou não se encaixam em um formato previsível?**
 - Antes que possam ser carregados em um BD relacional, os dados deverão ser reorganizados



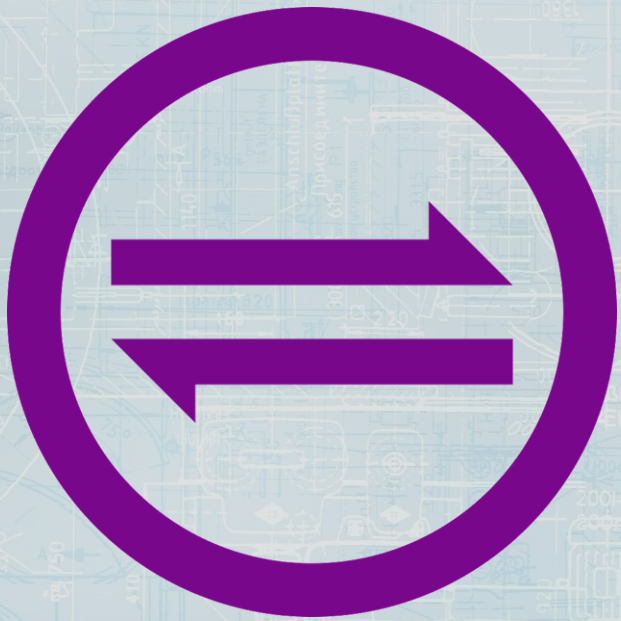
Custo: SQL



- ❖ A SQL é uma linguagem de consulta e modificação de dados muito poderosa
 - 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



Custo: 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 SGBDRs podem não ser apropriadas
- ❖ **As garantias transacionais, em qualquer nível, representam uma carga extra de processamento para o servidor**



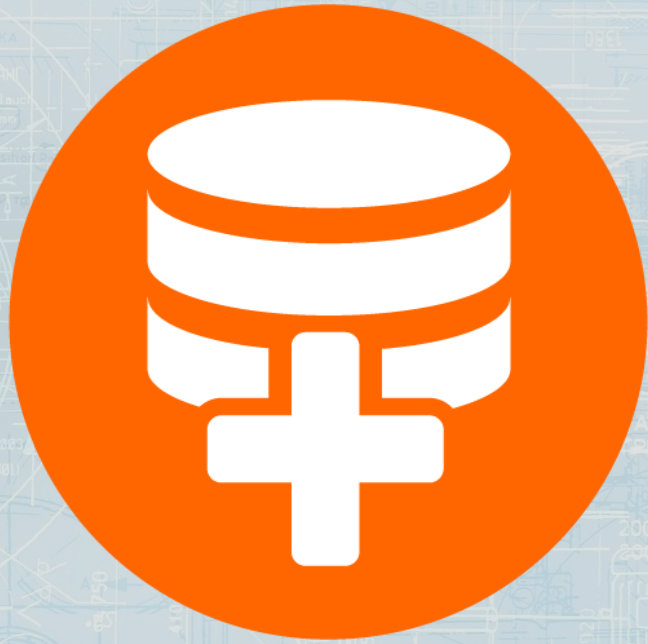
Custo: confiabilidade



- ❖ 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



Custo: 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 SGBDRs tradicionais foram projetados**
- ❖ Alguns motivos para o aumento do volume de dados:
 - Grande queda do custo de dispositivos de armazenamento secundário
 - Coletas de dados em altas taxas, feitas por sensores em variados tipos de dispositivos (como celulares, câmeras, etc.) e por *web sites* (como Facebook, Twitter, etc.)



Custo: 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



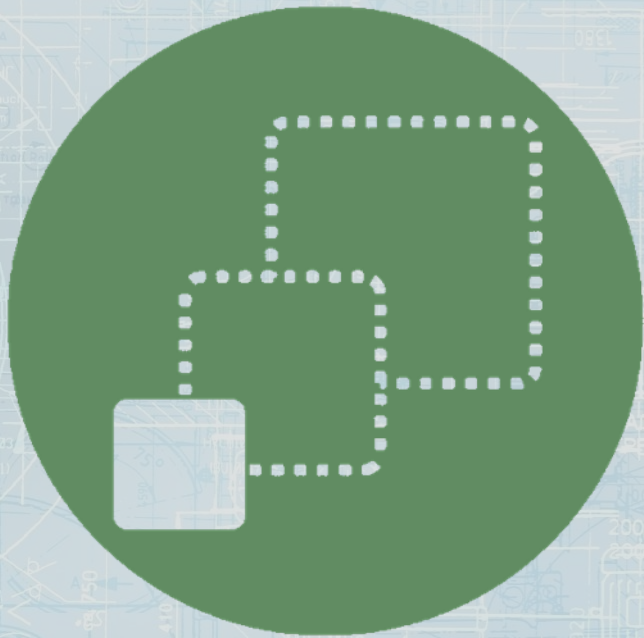
Custo: sobrecargas



❖ Principais sobrecargas de SGBDs relacionais:

- *Logging*: tudo é gravado duas vezes → uma no disco e outra no log do SGBD
- *Locks*: bloqueios para gravação de item de dados (para garantir propriedades transacionais)
- *Latches*: bloqueios especiais, para alteração de estruturas de dados auxiliares do SGBD (ex.: tabela de bloqueios)
- Gerenciamento de *buffer*: páginas de disco ↔ memória principal

Escalamento em SGBDRs



- ❖ **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?**

O MODELO NOSQL



Origem

❖ 1998 (primeira aparição)

- Carlo Strozzi
- SGBD relacional leve, para SOs derivados do UNIX
- Não implementava a linguagem SQL

❖ 2009 (reintrodução)

- Evento organizado por Johan Oskarsson (Last.fm)
- Popularidade do BigTable/MapReduce (Google) e do DynamoDB (Amazon).
- Novos sistemas de banco de dados distribuídos e de código aberto

nosql



Nomenclatura

nosql

- ❖ Com o tempo, “**SQL**” tornou-se sinônimo de SGBDs **relacionais**
- ❖ **NoSQL** é diferente de não usar a linguagem SQL; refere-se a não usar um SGBD relacional
 - Seria até melhor chamar de “**NoRel**”
- ❖ As últimas duas décadas mostraram que nem todo problema de gerenciamento e análise de dados tem por melhor resposta a adoção de um SGBDR tradicional
 - Em outras palavras, nem todo problema de gerenciamento e análise de dados é melhor solucionado usando **exclusivamente** um SGBDR tradicional
- ❖ Atualmente, **NoSQL** → “**Not Only SQL**”



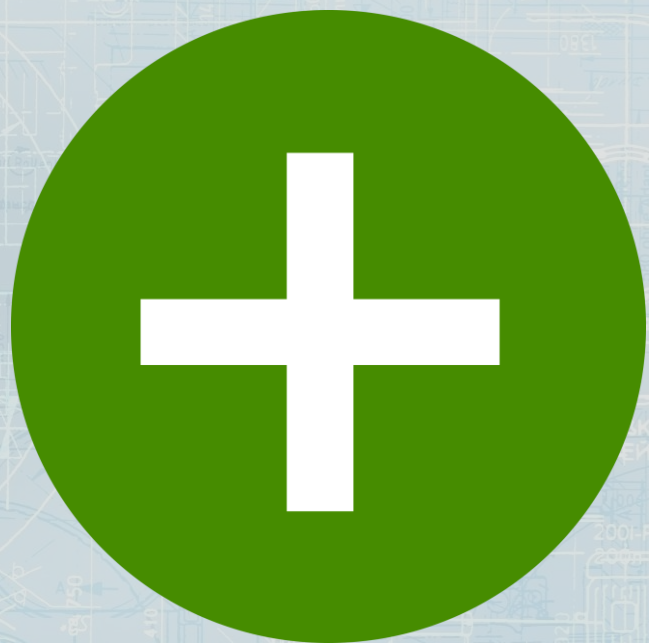
Por que não usar um SGBDR tradicional



- ❖ Necessidade de melhor desempenho
- ❖ Necessidade de maior flexibilidade



+ Dados, + Eficiência, + Flexibilidade



- ❖ 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



NoSQL: vantagens



- ❖ 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



NoSQL: desvantagens



- ❖ A ausência de uma linguagem de consulta declarativa e padronizada
 - Implica em 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



NoSQL: características comuns



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



Definições importantes



❖ “Operações simples”

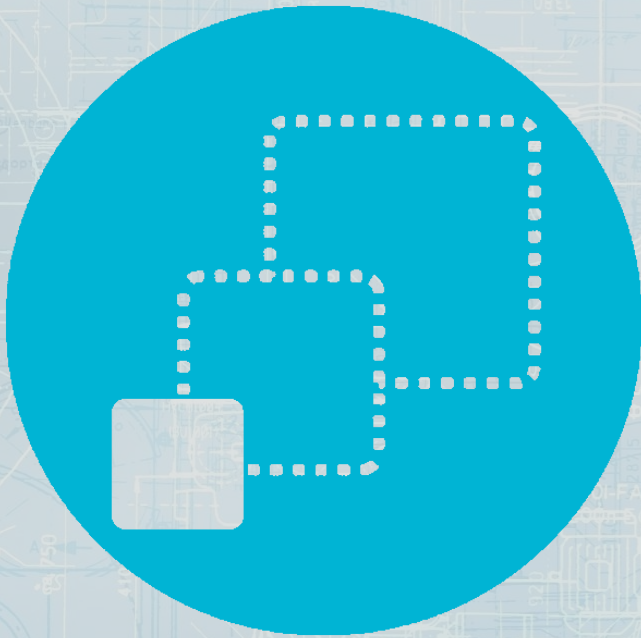
- Busca de um registro por meio de sua chave (*key lookup*), leituras e escritas de um registro ou de um número pequeno de registros
- Constituem o tipo de operação mais frequente nas aplicações web atuais

❖ “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 os dados (*sharding*)** 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 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

Problemas de consistência

❖ Tipos de conflitos que podem ocorrer em SGBDRs:

- **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 também têm esses conflitos. Ex.:

- Alterações simultâneas em cópias diferentes de um dado registro
- Leitura de uma cópia desatualizada





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 os tratam
- ❖ Para obter boa consistência, é preciso envolver muitos nós nas operações sobre os dados
 - Mas isso aumenta a latência (tempo de resposta das operações)



Conclusões



- ❖ O **modelo relacional** tradicional não está superado e se adapta bem a aplicações
 - com dados em **formato padronizado**
 - em que a **consistência** dos dados é crucial
 - que podem tolerar alguma **latência** de E/S
- ❖ Por outro lado, o **modelo NoSQL** é indicado quando
 - os dados **não** se moldam bem ao **modelo relacional**
 - **não** é necessária **consistência** absoluta
 - **velocidade de I/O** é crucial



Para saber mais



- ❖ **ELMASRI, R.; NAVATHE, S. B.**
Sistemas de Banco de Dados:
Fundamentos e Aplicações. 7. ed.
São Paulo: Pearson, 2019, p. 795-820