SQL vs NoSQL

Termos de Uso

Propriedade Growdev

Todo o conteúdo deste documento é propriedade da Growdev. O mesmo pode ser utilizado livremente para estudo pessoal.

É proibida qualquer utilização desse material que não se enquadre nas condições acima sem o prévio consentimento formal, por escrito, da Growdev. O uso indevido está sujeito às medidas legais cabíveis.

SQL vs NoSQL

Bancos de dados SQL (relacionais) e NoSQL (não relacionais) atendem a diferentes necessidades e casos de uso, baseados em suas características, arquitetura e formas de modelar dados.

Escolher um desses tipos – ou até ambos – em um projeto depende das necessidades da aplicação, além da natureza dos dados armazenados.

Estrutura de dados

SQL – Estrutura dos dados

Os bancos de dados SQL possuem um modelo estruturado e baseado em esquema bem definidos.

- Os dados são armazenados em tabelas com **linhas** e **colunas**.
- Cada tabela possui um **esquema fixo** que define os tipos e restrições dos dados.
- A **integridade** dos dados é garantida por meio de chaves primárias, estrangeiras e restrições.

Cliente

ID	Nome	Idade	Cidade
123	José	20	Porto Alegre
999	Joana	50	Santo Amaro
321	Astrogildo	43	Porto Alegre

NoSQL - Estrutura dos dados

Os bancos de dados NoSQL possuem um modelo flexível e sem necessidade de esquema:

- Os dados podem ser armazenados como **documentos** (JSON/BSON), pares **chave-valor**, **colunas** ou **grafos**.
- Não há necessidade de esquema fixo, o que permite uma **modelagem adaptável**.

Queries

SQL – Queries

Predominância da linguagem **SQL (Structured Query Language)** para consultas e manipulação de dados de forma padronizada.

Suporta operações complexas, como joins, sub consultas e transações.

Em um banco relacional, os dados são frequentemente **normalizados** para evitar redundância.

Por isso, para unir dados de entidades relacionadas é necessário realizar JOINs, que podem ter um custo muito elevado em determinados casos.

```
SELECT
    Name, Id, Email, Password, Age
FROM Users
WHERE
    Email = "daphne@dog.com"
    AND Password = "12345"
```

NoSQL - Queries

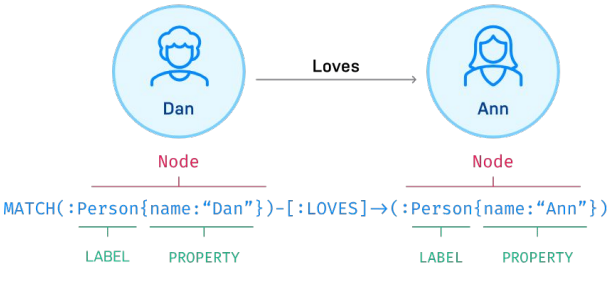
Não há uma padronização na forma de realizar consultas e manipulações.

Cada tipo de banco NoSQL possui sua própria linguagem de consulta ou APIs para manipulação de dados.

Exemplos: MongoDB usa comandos **JSON**, enquanto Cassandra utiliza **CQL** e o Neo4j usa **Cypher**.

```
primary  test → const nameRegex = /max/i
primary  test → db.users.find({name:
nameRegex}, {_id: 0, name: 1})
[
        { name: 'Maximo Heathcote' },
        { name: 'Maximus Borer' },
        { name: 'Maximo Heathcote' },
        { name: 'Maximus Borer' },
]
primary  test → db.users.fnd()
TypeError: db.users.fnd is not a function
primary  test → db.users.fnd({age: {$gt
db.users.fnd({age: {$gt  db.users.fnd({age: {$gte

primary  test → db.users.fnd({age: {$gt
```



NoSQL - Queries

Queries em BDs relacionais podem ter um custo muito elevado em cenários onde há um grande volume de dados

Exemplo: tabela com Posts, Comentários e Usuários em uma grande rede social, como o Instagram.

Em um BD relacional, os joins podem se tornar um gargalo de desempenho, já que o banco precisa combinar grandes volumes de dados, gerando alta latência.

Já uma consulta em BD NoSQL pode ser mais eficiente pois os dados estão armazenados juntos.

SQL

```sql
SELECT * FROM Posts Po
JOIN Comments Co
    ON Co.PostId = Po.Id
JOIN Users Us
    ON Us.Id = Po.UserId
JOIN PostCategories Pa
    ON Pa.PostId = Po.Id
JOIN Categories Ca
    On Ca.Id = Pa.CategoryId
```

MongoDB

```
db.users.find()

db.users.find({name: "Daphne Dog"})

db.users.find({}, {_id: 0, name: 1, posts: 1})
```

Consistência

SQL – Consistência

Os dados **são sempre consistentes**, o que é essencial em aplicações que necessitam essa rigidez, como em sistemas financeiros e transacionais.

Propriedades **ACID**: Atomicidade, Consistência, Isolamento e Durabilidade garantem **integridade** em transações.

Em sistemas com milhões de usuários simultâneos, garantir consistência forte pode ser um gargalo.

NoSQL - Consistência

O NoSQL não garante as propriedades ACID devido ao seu modelo flexível de dados.

Por outro lado, o NoSQL segue o padrão **BASE**:

- **Basicamente Disponível**;
- **Estado Flexível** (soft state); e
- **Consistência Eventual**.

Nesse modelo, a disponibilidade e a flexibilidade dos dados, além da a tolerância a falhas, são prioridades.

Técnicas como replicação garantem que, mesmo com falhas em alguns servidores, o sistema continue operacional.

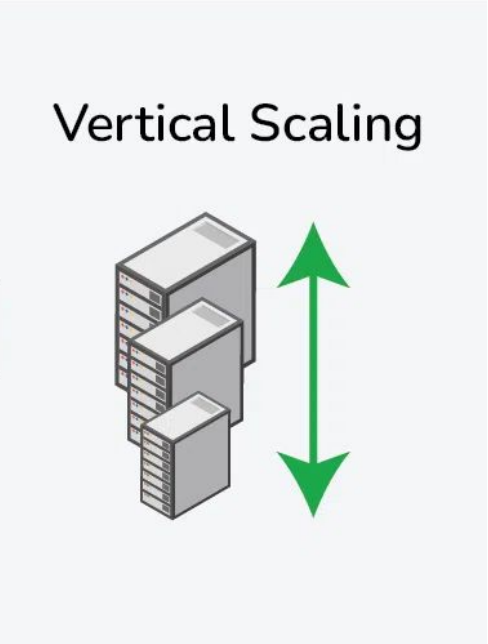
Escalabilidade

SQL – Escalabilidade

Permite **escalabilidade vertical**: Para melhorar o desempenho e lidar com cargas maiores, é necessário aumentar os recursos do servidor (mais memória, processadores, etc.).

Essa é uma técnica mais simples do ponto de vista arquitetural, mas é limitada fisicamente e pode se tornar caro.

A garantia do modelo ACID e a estrutura de relacionamentos dificultam a escalabilidade horizontal em bancos SQL.

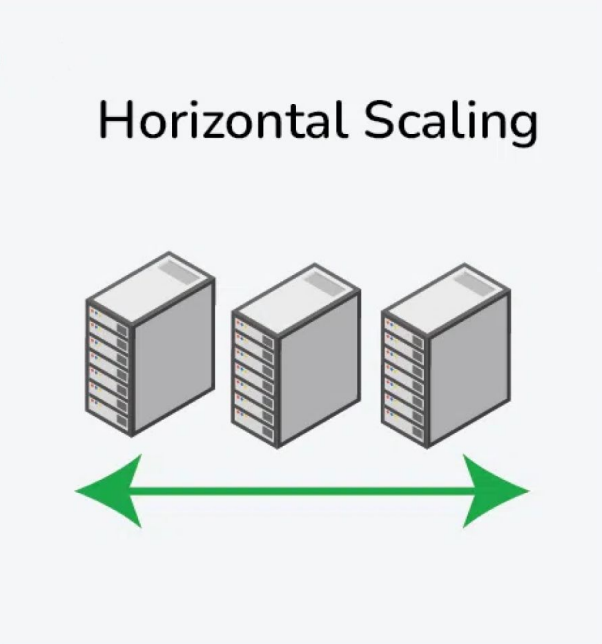


NoSQL – Escalabilidade

Possui facilidade para aplicar **escalabilidade horizontal**: permite ao banco expandir poder computacional com mais servidores.

Os dados podem ser **distribuídos** em vários servidores (shards), permitindo crescimento ilimitado com custos reduzidos.

Escalar horizontalmente com hardware commodity é mais barato do que aumentar os recursos de um único servidor. Mas essa abordagem traz mais desafios de **gerenciamento**.



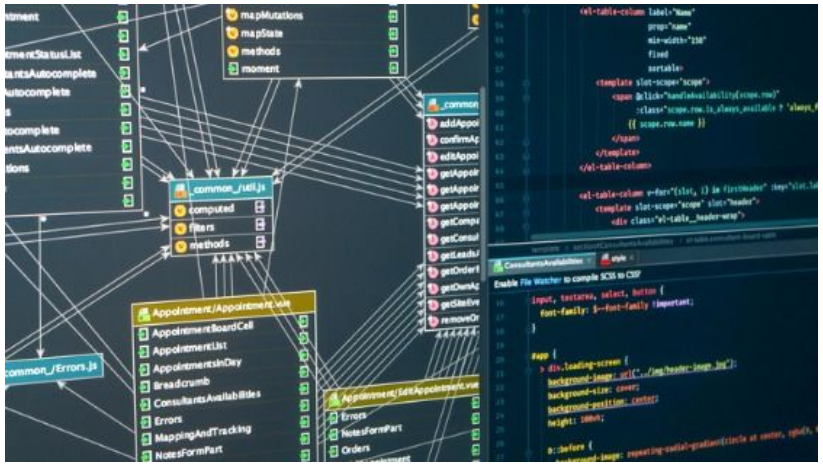
Modelagem

SQL - Modelagem

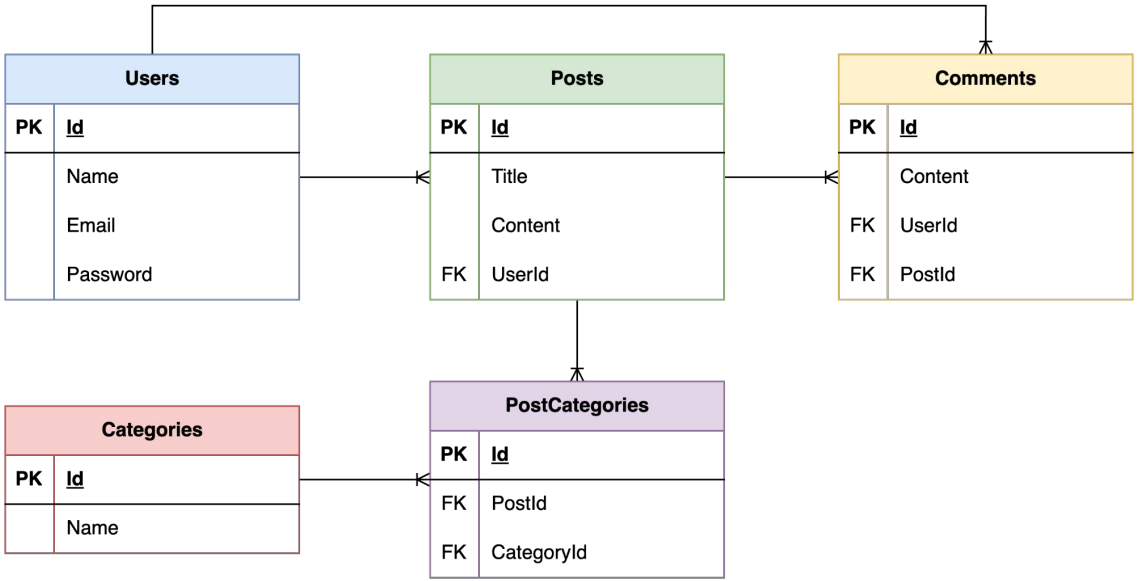
Bancos SQL priorizam a **normalização** dos dados através da modelagem. Os dados são organizados em várias tabelas relacionadas, reduzindo a redundância.

Os **relacionamentos** complexos entre tabelas (1-1, 1-N e N-N) são representados explicitamente. Consultas que envolvem várias tabelas podem ser mais lentas devido ao custo computacional de **joins**.

Qualquer alteração no esquema (ex.: adicionar colunas) pode exigir migração de dados ou **reestruturação**.



SQL – Modelagem



NoSQL - Modelagem

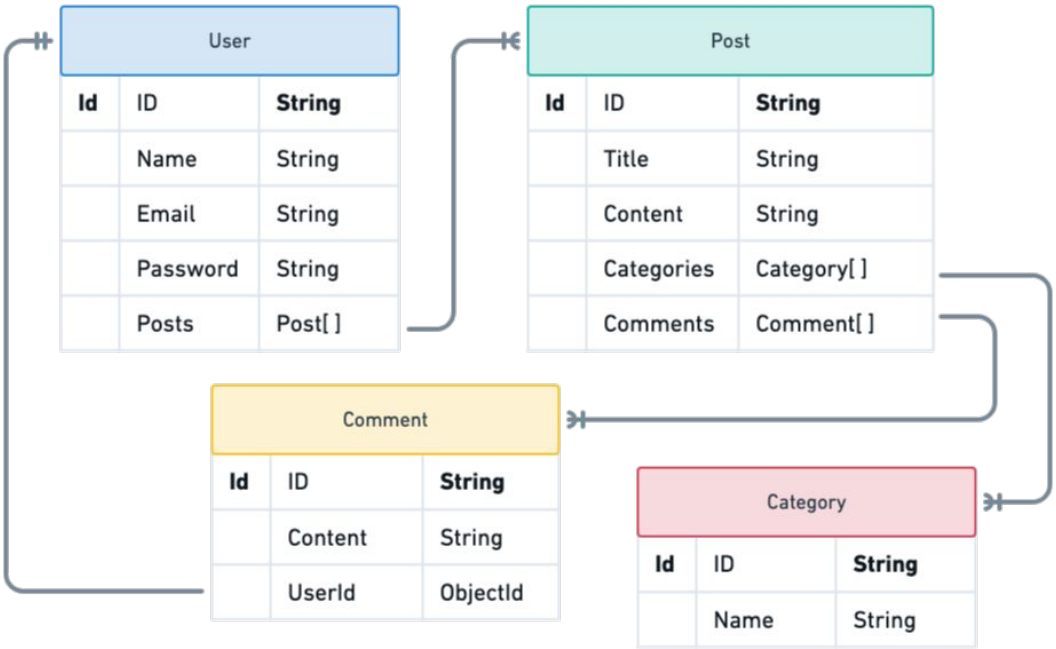
Não há necessidade de normalização em bancos NoSQL. Os dados frequentemente são armazenados de forma **embutida ou duplicados** em documentos únicos para evitar joins.

A modelagem é orientada pelas **necessidades de leitura e escrita** da aplicação.

Por exemplo, em vez de dividir dados em tabelas, você cria estruturas que já contenham todas as informações relevantes para uma consulta.

Dados de diferentes formatos podem coexistir no mesmo banco. Não há necessidade de esquema fixo.

NoSQL - Modelagem



Fixando ...

Aspecto	SQL	NoSQL
Aplicações Transacionais	Bancos, sistemas ERP e controle financeiro.	Não é ideal devido à consistência eventual.
Big Data	Menos eficiente em grandes volumes.	Ideal para dados massivos e distribuídos.
Esquema	Dados fixos e bem definidos.	Dados semi estruturados ou não estruturados.
Escalabilidade	Limitada a escalabilidade vertical.	Perfeito para escalabilidade horizontal.
Redes Sociais	Não lida bem com relações densas e grandes volumes.	Excelente para dados não estruturados e conexões.

Parabéns!
Nos vemos na próxima etapa!