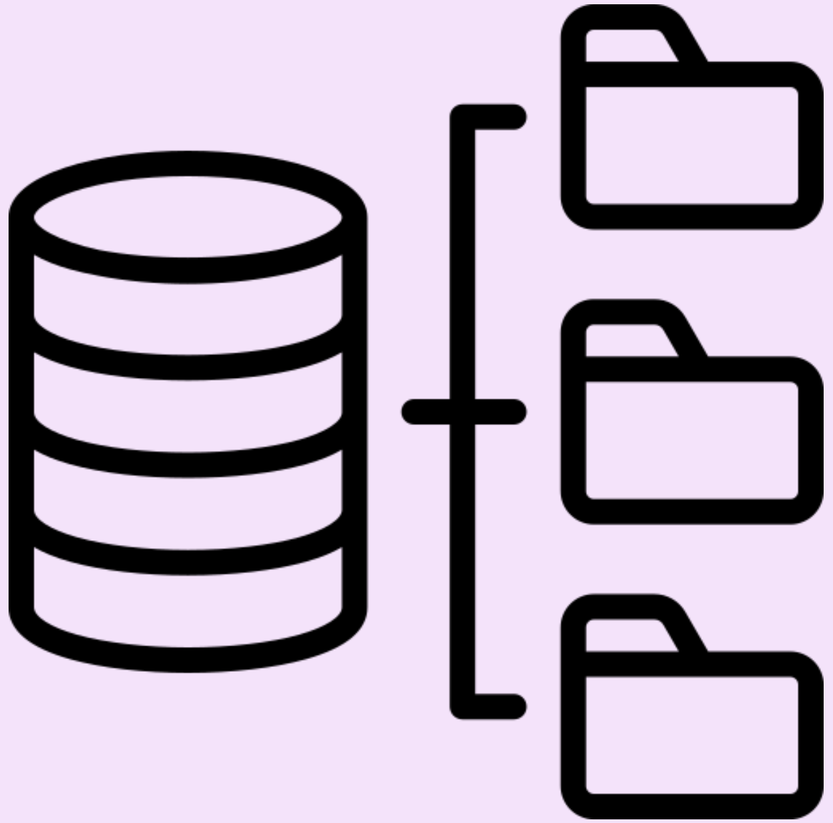
Bando de dados

O que é um Banco de Dados?

Um banco de dados é uma coleção organizada de informações - ou dados - estruturadas, normalmente armazenadas eletronicamente em um sistema de computador.

Os dados podem ser facilmente acessados, gerenciados, modificados, atualizados, controlados e organizados. A maioria dos bancos de dados usa a **linguagem de consulta estruturada (SQL)** para escrever e consultar dados.



Importância dos Bancos de Dados nos Dias de Hoje

1. **Gerenciamento de informações:** Os bancos de dados permitem armazenar e gerenciar grandes volumes de informações de maneira organizada, facilitando o acesso, a atualização e a análise dos dados.

2. **Produtividade:** Os bancos de dados aumentam a produtividade dos funcionários, automatizando tarefas rotineiras e permitindo o rápido acesso às informações necessárias para realizar suas atividades.

3. **Segurança e conformidade:** Bancos de dados bem projetados e gerenciados podem ajudar a garantir a segurança e a privacidade das informações, bem como atender aos requisitos de conformidade regulatória.

Importância dos Bancos de Dados nos Dias de Hoje

4. **Controle de estoque e logística:** Os bancos de dados ajudam no gerenciamento de estoques, rastreamento de produtos e planejamento de logística, garantindo que os produtos estejam disponíveis quando necessário e otimizando a cadeia de suprimentos.

5. **Análise e inteligência de negócios:** A análise de dados em bancos de dados permite que as empresas identifiquem tendências, padrões e oportunidades de negócios, melhorando a tomada de decisões estratégicas.

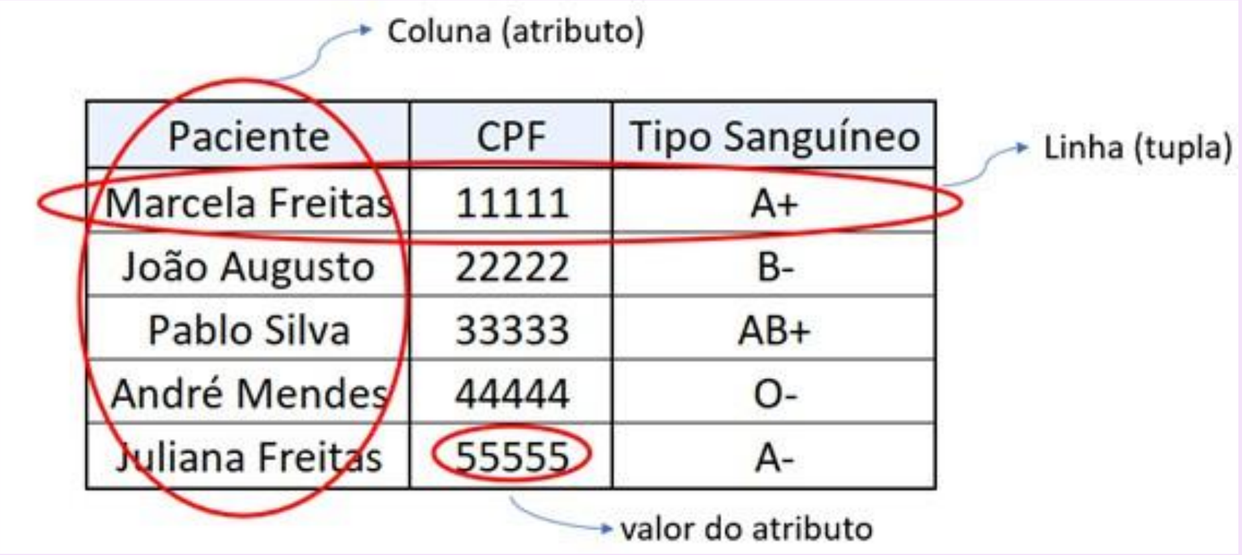
Tipos de bancos de dados

Existem muitos tipos diferentes de bancos de dados. O melhor banco de dados para uma organização específica depende de como a organização pretende usar os dados.

Bancos de dados relacionais

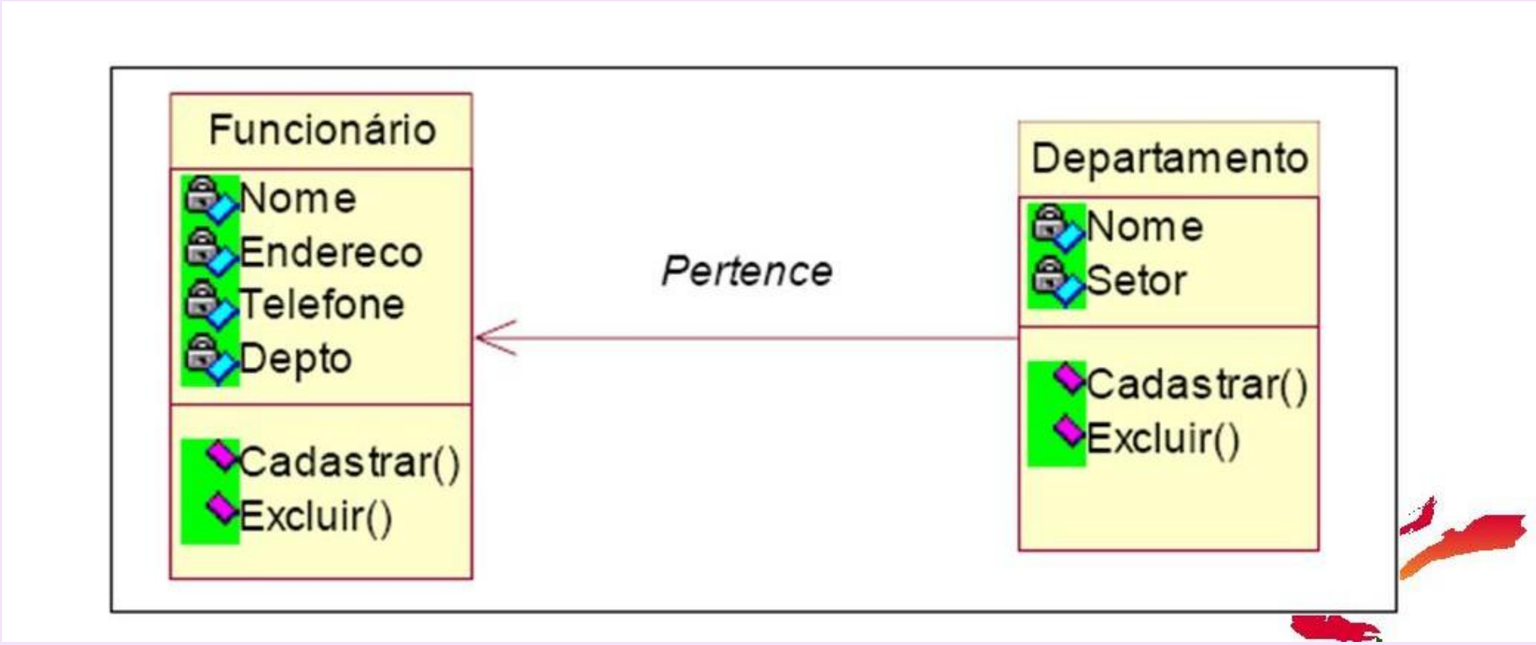
Os bancos de dados relacionais são baseados no modelo relacional. Nesse modelo, os dados são organizados em tabelas com linhas e colunas. Cada tabela representa uma entidade, e a relações entre as entidades são estabelecidas por meio de chaves primárias e chaves estrangeiras.

Exemplos populares de bancos de dados relacionais incluem MySQL, PostgreSQL, SQL Server e Oracle.



Bancos de Dados Orientados a Objetos:

Os bancos de dados orientados a objetos foram projetados para lidar com objetos complexos e suas inter-relações. Eles permitem armazenar não apenas dados primitivos, mas também estruturas de dados complexas, como objetos, herança e encapsulamento. Isso é especialmente útil em cenários de programação orientada a objetos, onde os objetos do programa podem ser diretamente mapeados para objetos no banco de dados. Alguns exemplos são ObjectDB e db4o.

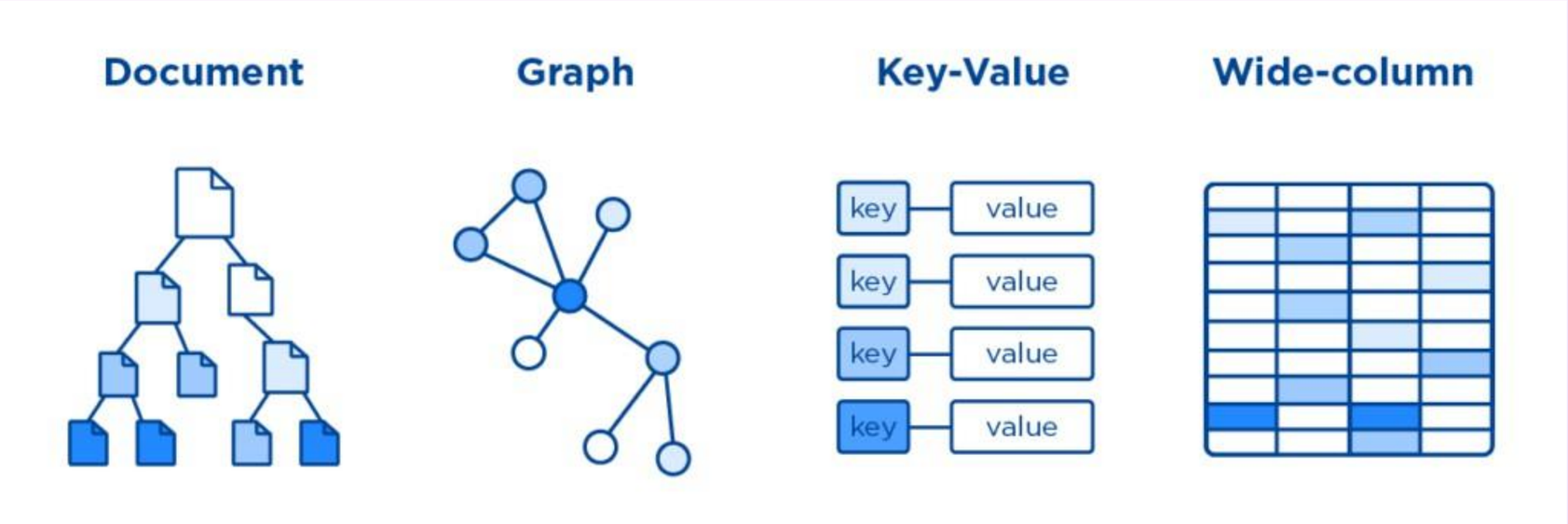


Bancos de Dados em Nuvem:

Bancos de dados em nuvem são hospedados e gerenciados em plataformas de computação em nuvem, como **Amazon Web Services (AWS)**, **Microsoft Azure** e **Google Cloud Platform**. Eles oferecem recursos escaláveis e serviços gerenciados para armazenar e acessar dados. Isso elimina a necessidade de gerenciar a infraestrutura subjacente, permitindo que as equipes se concentrem mais no desenvolvimento de aplicativos. Exemplos incluem Amazon RDS, Azure SQL Database e Google Cloud SQL.

Bancos de Dados NoSQL:

A categoria NoSQL (Not Only SQL) engloba uma variedade de bancos de dados que não se encaixam no modelo relacional tradicional. Eles foram projetados para atender a diferentes necessidades, como escalabilidade, flexibilidade no esquema de dados e baixa latência. Os bancos de dados NoSQL incluem várias subcategorias, como bancos de dados de documentos, colunas, chave-valor e grafos. Exemplos populares são MongoDB (documento), Cassandra (coluna), Redis (chave-valor) e Neo4j (grafo).



SQL x NoSQL:

A diferença entre bancos de dados SQL (relacionais) e bancos de dados NoSQL é substancial e envolve vários aspectos, **incluindo esquema de dados**, **modelo de armazenamento**, **consultas** e características como baixa **latência, flexibilidade e escalabilidade.**

SQL: Bancos de dados relacionais têm esquemas de dados rígidos, o que significa que você precisa definir a estrutura da tabela com antecedência. Cada coluna tem um tipo de dado predefinido.

NoSQL: Bancos de dados NoSQL, como bancos de dados de documentos ou chave-valor, oferecem esquemas flexíveis. Isso permite que você adicione campos aos documentos sem uma estrutura rígida.

Baixa Latência:

Geralmente, bancos de dados NoSQL são projetados para oferecer baixa latência, o que significa que eles podem recuperar e fornecer dados rapidamente. Isso é especialmente importante em aplicativos que exigem respostas rápidas, como sistemas de tempo real e aplicativos web interativos.

Flexibilidade:

Bancos de dados NoSQL, devido à sua natureza flexível de esquema, permitem que você adicione, altere e remova campos sem a necessidade de migrações complexas de esquema. Isso é útil quando os requisitos de dados estão em constante evolução.

Escalabilidade:

- NoSQL: Muitos bancos de dados NoSQL foram projetados para escalar horizontalmente, o que significa que você pode adicionar mais servidores para lidar com volumes crescentes de dados e tráfego sem grandes interrupções. Isso é especialmente útil em cenários onde a demanda por recursos pode variar significativamente.

- SQL: Bancos de dados relacionais também podem ser escalados, mas frequentemente envolvem mais complexidade em termos de modelagem de dados e gerenciamento de esquema.

Quando usar NoSQL e SQL?

- Se você estiver construindo um aplicativo com requisitos bem definidos e um esquema de dados estável, um banco de dados SQL pode ser mais apropriado.

- Se você precisar de flexibilidade no esquema e tiver dados semi-estruturados ou em constante mudança, um banco de dados NoSQL pode ser uma melhor escolha.

- Se seu aplicativo precisa de recuperação rápida de dados em tempo real, como em aplicativos de mídia social, os bancos de dados NoSQL podem ser benéficos.

- Se o aplicativo tem requisitos de conformidade regulatória e precisa garantir a segurança e o controle rigoroso dos dados, um banco de dados SQL pode ser mais apropriado.

O que é SQL?

Structured Query Language, Linguagem de consulta estruturada

É uma linguagem de programação para armazenar e processar informações em um banco de dados relacional. Um banco de dados relacional armazena informações em formato tabular, com linhas e colunas representando diferentes atributos de dados e as várias relações entre os valores dos dados

A linguagem SQL foi criada justamente para facilitar o gerenciamento de dados armazenados em bancos que seguem esse padrão.

Onde é utilizado SQL?

Alguns dos principais sistemas que utilizam essa tecnologia são:

- **MySQL**: criado pela Oracle, é uma opção que oferece serviços gratuitos e pagos;
- **PostgreSQL:** muito usado em aplicações web, essa é uma opção gratuita de código aberto;
- **Oracle:** conhecido pela sua segurança, esse sistema é um dos mais usados por grandes corporações;
- **SQL Server:** desenvolvido pela Microsoft, ele oferece tanto serviços pagos quanto versões gratuitas para download.

Quais as vantagens e desvantagens da linguagem SQL?

Principais vantagens

- **Padronização:** O SQL foi padronizado pelo ANSI. Por isso, ela oferece uma ampla documentação e, aprendendo a linguagem, você vai conseguir lidar facilmente com diferentes bancos de dados que a implementam;

- **Rápido acesso aos dados:** SQL permite a recuperação de múltiplos registros com um único comando, tudo feito de forma rápida e eficiente;

- **Portabilidade:** o SQL pode ser usado em diversas aplicações para diferentes plataformas;

- **Múltiplas visões de dados:** com SQL é possível definir diferentes visualizações da estrutura do banco para diferentes pessoas usuárias;

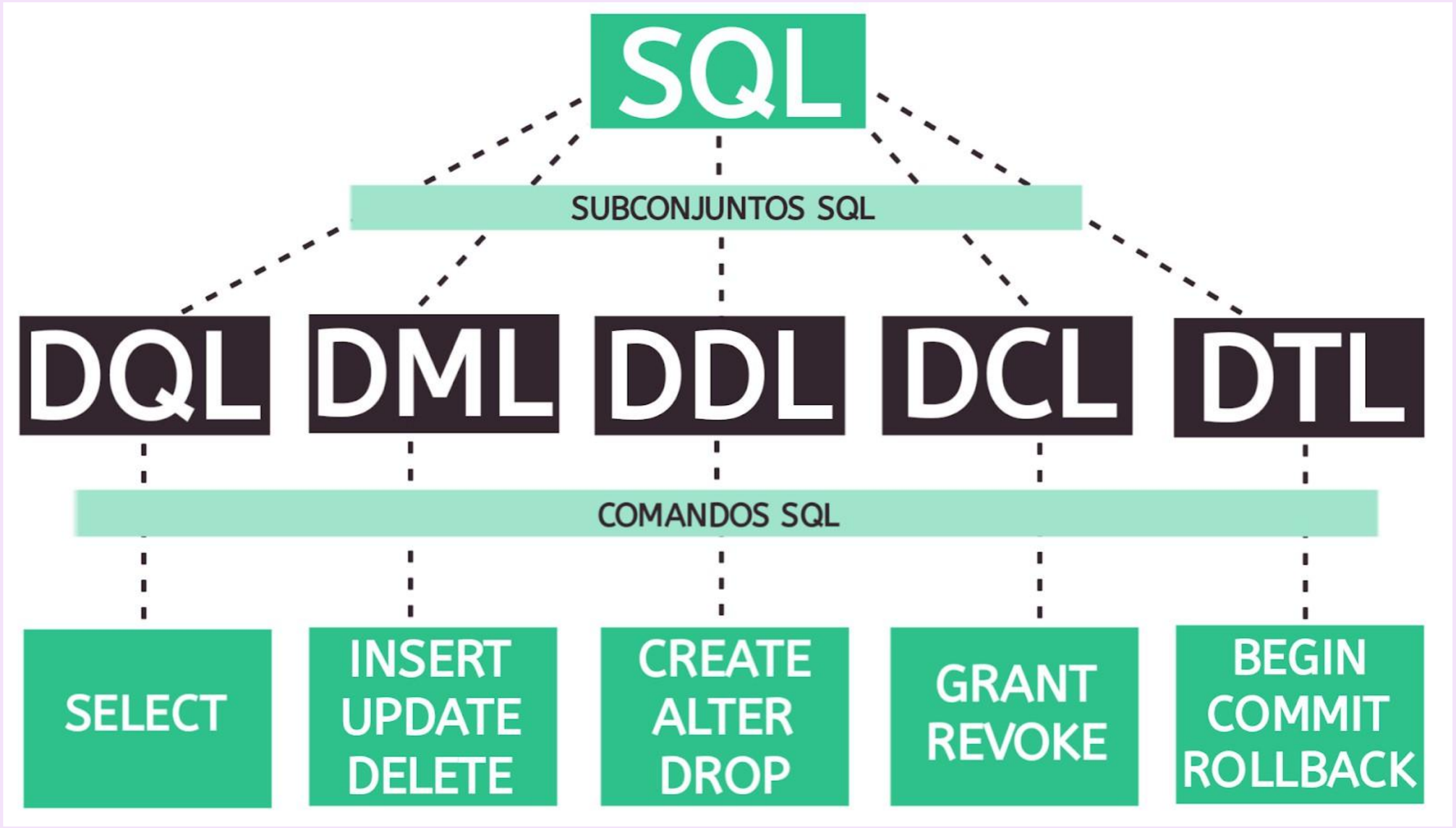
- **Linguagem cliente/servidor**: com SQL é possível implementar arquitetura cliente-servidor.

Quais as vantagens e desvantagens da linguagem SQL?

Principais desvantagens

- **Dificuldade com a interface:** para quem não está acostumado, a interface de um banco de dados SQL pode ser mais difícil de lidar;

- **Custo:** o custo operacional de alguns serviços oferecidos por bancos que utilizam SQL é caro e dificulta o acesso dos desenvolvedores.


SQL
SUBCONJUNTOS SQL
DQL
DML
DDL
DCL
DTL
COMANDOS SQL
SELECT
INSERT
UPDATE
DELETE
CREATE
ALTER
DROP
GRANT
REVOKE
BEGIN
COMMIT
ROLLBACK

DML: linguagem de manipulação de dados

O **DML (Data Manipulation Language)** é o subconjunto do SQL que define os comandos usados para manipular os dados armazenados em um banco. Esse é um dos conjuntos mais utilizados, pois ele fornece operadores que nos permitem inserir, excluir e alterar os registros de uma tabela, por exemplo. Os comandos mais importantes desse subconjunto são: INSERT, DELETE e UPDATE.

DQL: linguagem de consulta de dados

O **DQL (Data Query Language)** é o conjunto SQL que define o comando mais popular da linguagem, o SELECT. Esse comando é essencial para que possamos consultar os dados que armazenamos em nosso banco.

DDL: linguagem de definição de dados

O **DDL (Data Definition Language)** é o subconjunto SQL que apresenta comandos usados para gerenciar as estruturas do banco de dados. Com ele, podemos criar, atualizar e remover objetos da base, como tabelas e índices. Os comandos definidos pelo DDL são: CREATE, DROP e ALTER.

DCL: linguagem de controle de dados

O **DCL (Data Control Language)** é o subconjunto no qual encontramos comandos para controlar o acesso aos dados da nossa base. Utilizando esse conjunto, conseguimos estabelecer restrições e permissões para quem acessa o banco por meio dos comandos GRANT e REVOKE.

DTL ou TCL: linguagem de transação de dados

O **DTL (Data Transaction Language)** — também conhecido como TCL (Transaction Control Language) — é o subconjunto SQL que define comandos que utilizamos quando é necessário gerenciar transações feitas no banco. Isso significa que eles permitem iniciar, confirmar e desfazer determinadas alterações. Os comandos estabelecidos pelo conjunto são COMMIT, BEGIN e ROLLBACK.

Consultas em Bando de Dados

O comando **SELECT** permite recuperar os dados de um objeto do banco de dados, como uma tabela, view

ID	Nome	Preço	Estoque
1	Produto A	50	20
2	Produto B	75	15
3	Produto C	30	5
4	Produto D	100	8
5	Produto E	45	30

Para selecionar apenas o nome e o preço dos produtos, você usaria o seguinte comando:

Exemplo 1: Selecionando Campos Específicos

```sql
SELECT Nome, Preço
FROM Produtos;
```

Nome	Preço
Produto A	50
Produto B	75
Produto C	30
Produto D	100
Produto E	45

Exemplo 2: Selecionando Todos os Campos

```sql
SELECT *
FROM Produtos;
```

ID	Nome	Preço	Estoque
1	Produto A	50	20
2	Produto B	75	15
3	Produto C	30	5
4	Produto D	100	8
5	Produto E	45	30

Exemplo 3: Filtrando com a Cláusula WHERE

```
SELECT Nome, Preço
FROM Produtos
WHERE Preço > 50;
```

```
|    Nome    | Preço |
|------------|-------|
| Produto B|   75    |
| Produto D|   100   |
```

Exemplo 4: Ordenando com a Cláusula ORDER BY

```
SELECT Nome, Preço
FROM Produtos
ORDER BY Preço DESC;
```

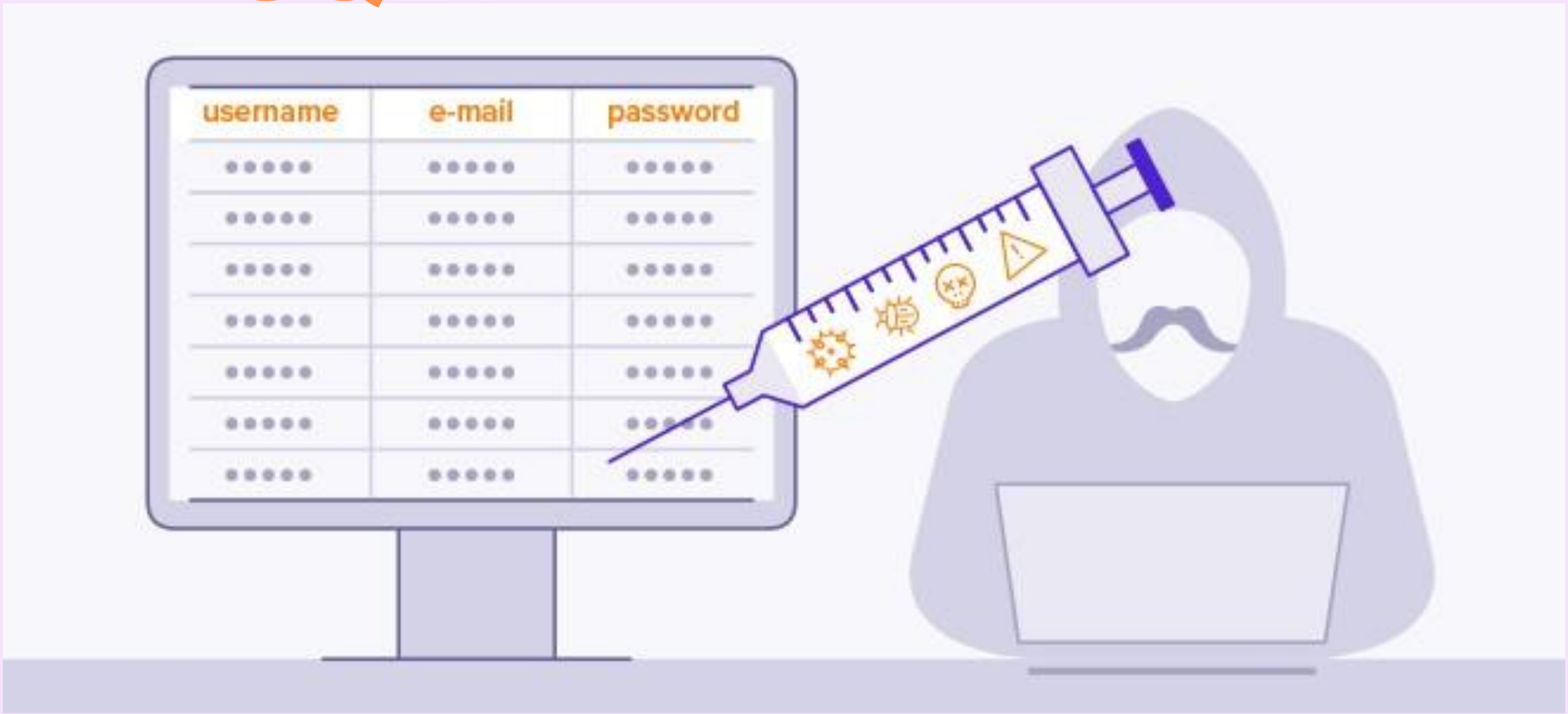
```
|    Nome    | Preço |
|------------|-------|
| Produto D|   100   |
| Produto B|   75    |
| Produto A|   50    |
| Produto E|   45    |
| Produto C|   30    |
```

Exemplo 5: Combinando Condições com a Cláusula WHERE

```sql
SELECT Nome, Preço
FROM Produtos
WHERE Preço > 50 AND Estoque < 10;
```

Nome	Preço
Produto D	100
Produto C	30

O que é injeção SQL?



O que é injeção SQL?

A **injeção de SQL**, também conhecida como SQLI, é um vetor de ataque comum que usa **código SQL** malicioso para manipulação de banco de dados de back-end para acessar informações que não deveriam ser exibidas.

Essas informações podem incluir qualquer número de itens, incluindo dados confidenciais da empresa, listas de usuários ou detalhes particulares de clientes.

Um ataque bem-sucedido pode resultar em:

- Visualização não autorizada de listas de usuários
- Exclusão de tabelas inteiras
- Obtenção de direitos administrativos sobre um banco de dados pelo invasor

Tipos de injeções SQL

As injeções de SQL normalmente se enquadram em três categorias:

- **SQLi dentro da banda (clássico)**
- **SQLi inferencial (cego)**
- **SQLi fora da banda.**

Você pode classificar os tipos de injeções SQL com base nos métodos usados para acessar dados de back-end e seu potencial de dano.

SQLi em banda

O invasor utiliza o **mesmo canal de comunicação** para lançar seus ataques e coletar seus resultados. A simplicidade e eficiência do SQLi em banda o tornam um dos tipos mais comuns de ataque SQLi.

SQLi baseado em erros — o invasor executa ações que fazem com que o banco de dados produza mensagens de erro. O invasor pode potencialmente usar os dados fornecidos por essas mensagens de erro para coletar informações sobre a estrutura do banco de dados.

SQLi baseado em união — esta técnica aproveita o operador UNION SQL, que funde várias instruções select geradas pelo banco de dados para obter uma única resposta HTTP. Esta resposta pode conter dados que podem ser aproveitados pelo invasor.

SQLi inferencial (cego)

O invasor envia cargas de dados ao servidor e observa a resposta e o comportamento do servidor para aprender mais sobre sua estrutura. Este método é chamado de SQLi cego porque os dados não são transferidos do banco de dados do site para o invasor, portanto, o invasor não pode ver informações sobre o ataque dentro da banda.

As injeções cegas de SQL **dependem da resposta e dos padrões de comportamento do servidor**, portanto, normalmente são mais lentas de executar, mas podem ser igualmente prejudiciais.

SQLi fora de banda

O invasor só pode realizar esta forma de ataque quando determinados recursos estiverem habilitados no servidor de banco de dados utilizado pela aplicação web. Esta forma de ataque é usada principalmente como uma alternativa às técnicas SQLi in-band e inferenciais.

O SQLi fora de banda é executado quando o invasor não consegue usar o mesmo canal para lançar o ataque e coletar informações ou quando um servidor é muito lento ou instável para que essas ações sejam executadas. Essas técnicas contam com a capacidade do servidor de criar solicitações DNS ou HTTP para transferir dados a um invasor.

Exemplo de injeção SQL

Imagine um cenário em que um site de comércio eletrônico possui uma página que exibe detalhes de um item, onde os detalhes são buscados com base no ID do item na URL. A URL normal para acessar um item específico seria algo como:

http://www.estore.com/items/items.asp?itemid=123

Agora, um invasor pode tentar explorar uma vulnerabilidade de SQL Injection inserindo um valor malicioso no parâmetro "itemid" na URL. No exemplo que você forneceu:

http://www.estore.com/items/items.asp?itemid=999+or+1%3D1+--

Nessa URL, o invasor adicionou "or 1=1 --" ao parâmetro "itemid".

"or 1=1": Isso é uma condição lógica que é sempre verdadeira. Ou seja, ela sempre se avalia como verdadeira, não importando o valor de "itemid".

"--": Isso é usado para iniciar um comentário em SQL, o que faz com que o restante da consulta seja ignorado.

O resultado disso é que o invasor está essencialmente enganando o sistema para pensar que a condição é verdadeira, fazendo com que a consulta SQL se pareça com algo como:

```
SELECT * FROM items WHERE itemid=999 OR 1=1 --;
```

SQL Injection: como prevenir?

- Valide sempre os dados digitados pelo usuário
- Crie usuários com permissões adequadas
- Nunca retorne as mensagens do servidor SQL para o usuário
- Habilite logs de segurança no servidor

NoSQL

Bancos de Dados NoSQL: Muitos bancos de dados NoSQL não usam a linguagem SQL tradicional. Eles podem ter suas próprias linguagens de consulta ou abordagens específicas para recuperar dados.

Por exemplo, o MongoDB utiliza uma linguagem de consulta chamada "MongoDB Query Language", que é diferente da SQL.

MongoDB(MQL)

select * from Produtos = db.Produtos.find()

A função find() na Linguagem de Consulta do MongoDB (MQL) fornece a maneira mais fácil de recuperar dados de vários documentos dentro de uma de suas coleções. Essa função é aquela que você usará frequentemente.

MongoDB(MQL)

SQL

select * from Produtos where preço>50;

MQL

db.Produtos.find({ preco: { $gt: 50 } });

Nesta consulta MQL, estamos usando o operador $gt (greater than) para buscar documentos na coleção "Produtos" onde o valor do campo "preco" seja maior do que 50. Isso é similar ao que você faria com a cláusula WHERE na consulta SQL.

Exploração dos Diferentes Tipos de Bancos de Dados NoSQL

1. Bancos de Dados de Documentos:

Os bancos de dados de documentos armazenam informações em documentos semelhantes a JSON. Cada documento é autônomo e pode conter diferentes estruturas de dados. Essa flexibilidade torna os bancos de dados de documentos adequados para aplicações com esquemas de dados variáveis.

Exploração dos Diferentes Tipos de Bancos de Dados NoSQL

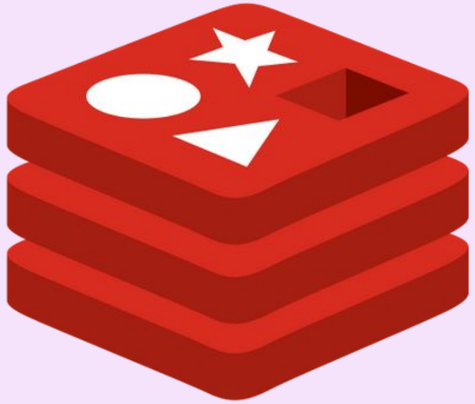
2. Bancos de Dados de Colunas:

Os bancos de dados de colunas armazenam dados em colunas, permitindo um desempenho eficiente em consultas que envolvem grandes volumes de dados e agregações. Cada coluna pode ser indexada separadamente, facilitando a recuperação seletiva de informações.

Exploração dos Diferentes Tipos de Bancos de Dados NoSQL

3. Bancos de Dados Chave-Valor:

Os bancos de dados chave-valor armazenam dados em pares chave-valor simples. São eficientes para armazenar dados simples e de acesso rápido, sendo amplamente usados em caches e sessões de usuários.



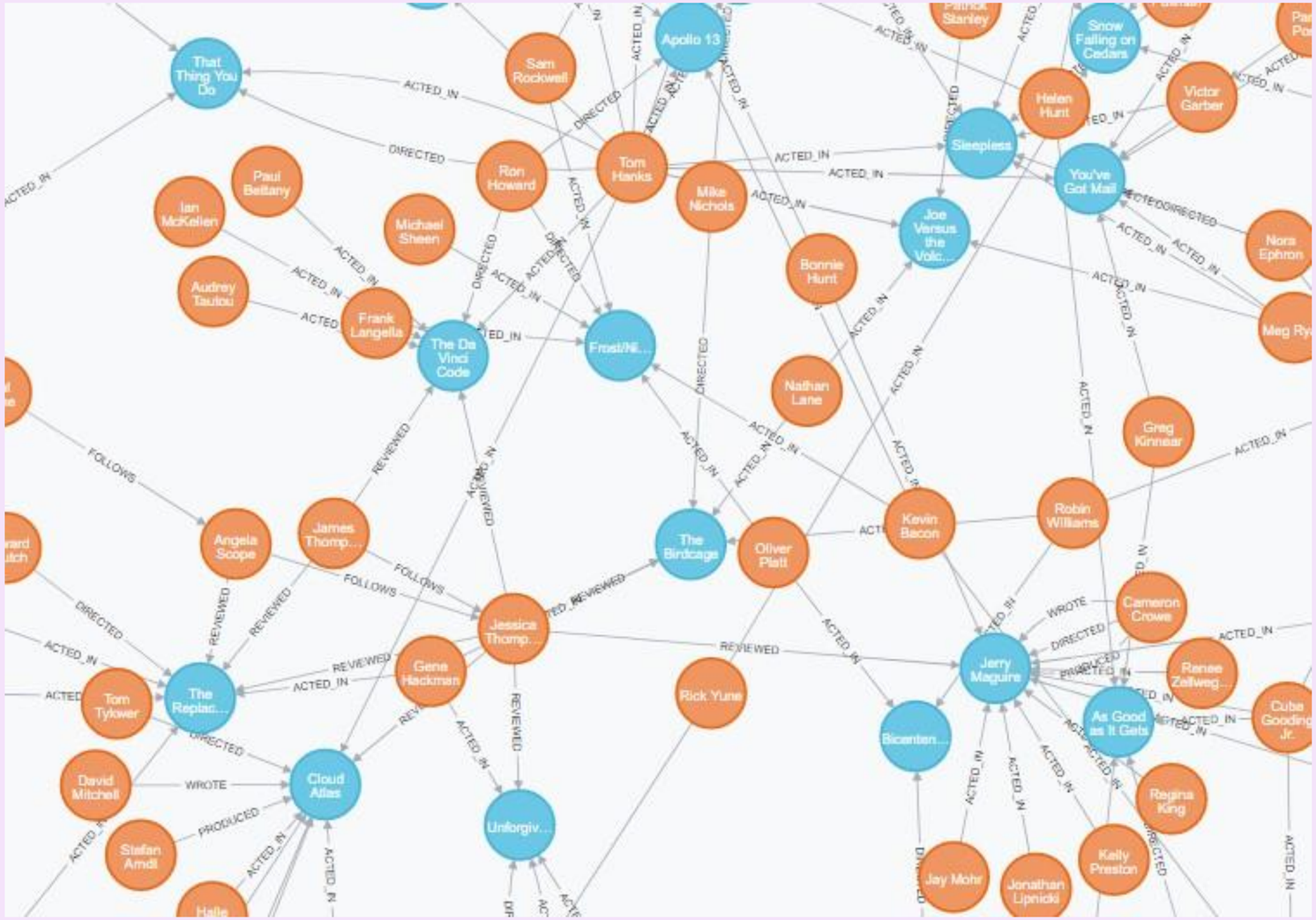
redis

Exploração dos Diferentes Tipos de Bancos de Dados NoSQL

4. Bancos de Dados de Grafos:

Os bancos de dados de grafos armazenam dados em forma de nós (entidades) e arestas (relacionamentos). Eles são usados para modelar e consultar dados altamente interconectados, como redes sociais e sistemas de recomendação.

Bancos de Dados de Grafos:



Pense em uma rede social. Cada usuário é um nó, e as conexões entre os usuários (amigos, seguidores) são as arestas. Um banco de dados de grafos permite consultas complexas para encontrar conexões e recomendações precisas.

NoSQL Injection

Assim como ocorre com as injeções SQL, as injeções NoSQL representam uma séria ameaça à segurança dos sistemas de banco de dados. Esses tipos de ataques possibilitam que invasores explorem brechas nas aplicações, contornem mecanismos de autenticação e executem ações maliciosas nos bancos de dados NoSQL.

Essas ações podem variar desde a extração de informações confidenciais até a manipulação indevida de dados armazenados, podendo inclusive resultar em comprometimento tanto do banco de dados quanto do servidor subjacente.

Diferenças entre Injeção SQL e Injeção NoSQL:

Estrutura de Dados:

A principal diferença é que os bancos de dados NoSQL não usam a linguagem SQL para consultas. Portanto, as técnicas de injeção não se baseiam em inserir comandos SQL maliciosos. Em vez disso, os ataques NoSQL exploram a estrutura e as regras de consulta específicas do banco de dados NoSQL.

Diferenças entre Injeção SQL e Injeção NoSQL:

Operadores Especiais:

Em injeções NoSQL, os atacantes podem explorar operadores especiais usados em bancos de dados NoSQL, como MongoDB.

Por exemplo, em vez de usar 1=1 como em injeções SQL, eles podem explorar operadores como $gt (greater than), $ne (not equal), $regex (regular expression), entre outros.

Como ocorre as invasões?

- **Entrada do Usuário Não Validada**: A injeção NoSQL geralmente ocorre quando as entradas do usuário não são devidamente validadas e sanitizadas antes de serem incorporadas às consultas NoSQL.

- **Exploração de Operadores**: Os invasores inserem operadores especiais ou estruturas de consulta maliciosas nas entradas do usuário. Esses operadores podem enganar o sistema e alterar o comportamento da consulta.

- **Manipulação da Lógica da Consulta**: Os atacantes podem manipular a lógica da consulta para obter mais informações do que o pretendido ou para contornar mecanismos de autenticação e acesso.

Mitigação de Injeção NoSQL:

- **Validação e Sanitização:** É crucial validar e sanitizar todas as entradas do usuário antes de usá-las em consultas NoSQL.
- **Uso de Bibliotecas Seguras:** Muitas linguagens de programação e bancos de dados NoSQL oferecem bibliotecas que ajudam a prevenir ataques de injeção.
- **Princípio do Menor Privilégio:** Conceda aos usuários apenas as permissões necessárias para acessar os recursos, reduzindo assim o risco de explorações.
- **Acesso Controlado:** Utilize mecanismos de autenticação e autorização rigorosos para restringir o acesso às consultas e ações específicas.

Exemplo de Vulnerabilidade: Injeção NoSQL

Suponha que uma aplicação use o MongoDB e permite que os usuários realizem consultas por nome

```
const userInput = req.query.name;
const query = { name: userInput };

db.collection('users').find(query).toArray((err, users) => {
  if (err) {
    // Tratamento de erro
  }

  // Processar os resultados
});
```

Se um invasor inserir como entrada **"userInput": { "$gt": " " }**, a consulta resultante pode retornar todos os documentos da coleção, porque o operador $gt (greater than) é interpretado pelo MongoDB como uma comparação de string e " " é maior que qualquer string vazia. Isso pode permitir que o invasor obtenha mais informações do que deveria.

Exemplo de Mitigação: Validação e Sanitização

Para mitigar esse tipo de vulnerabilidade, a entrada do usuário deve ser validada e sanitizada adequadamente:

```
const userInput = req.query.name;

if (userInput === undefined || userInput === null) {
  // Tratar entrada inválida
}

const query = { name: userInput };

db.collection('users').find(query).toArray((err, users) => {
  if (err) {
    // Tratamento de erro
  }

  // Processar os resultados
});
```

Neste exemplo, a validação é feita para garantir que a entrada do usuário não esteja indefinida ou nula. Isso ajuda a prevenir a injeção NoSQL explorando operadores especiais.