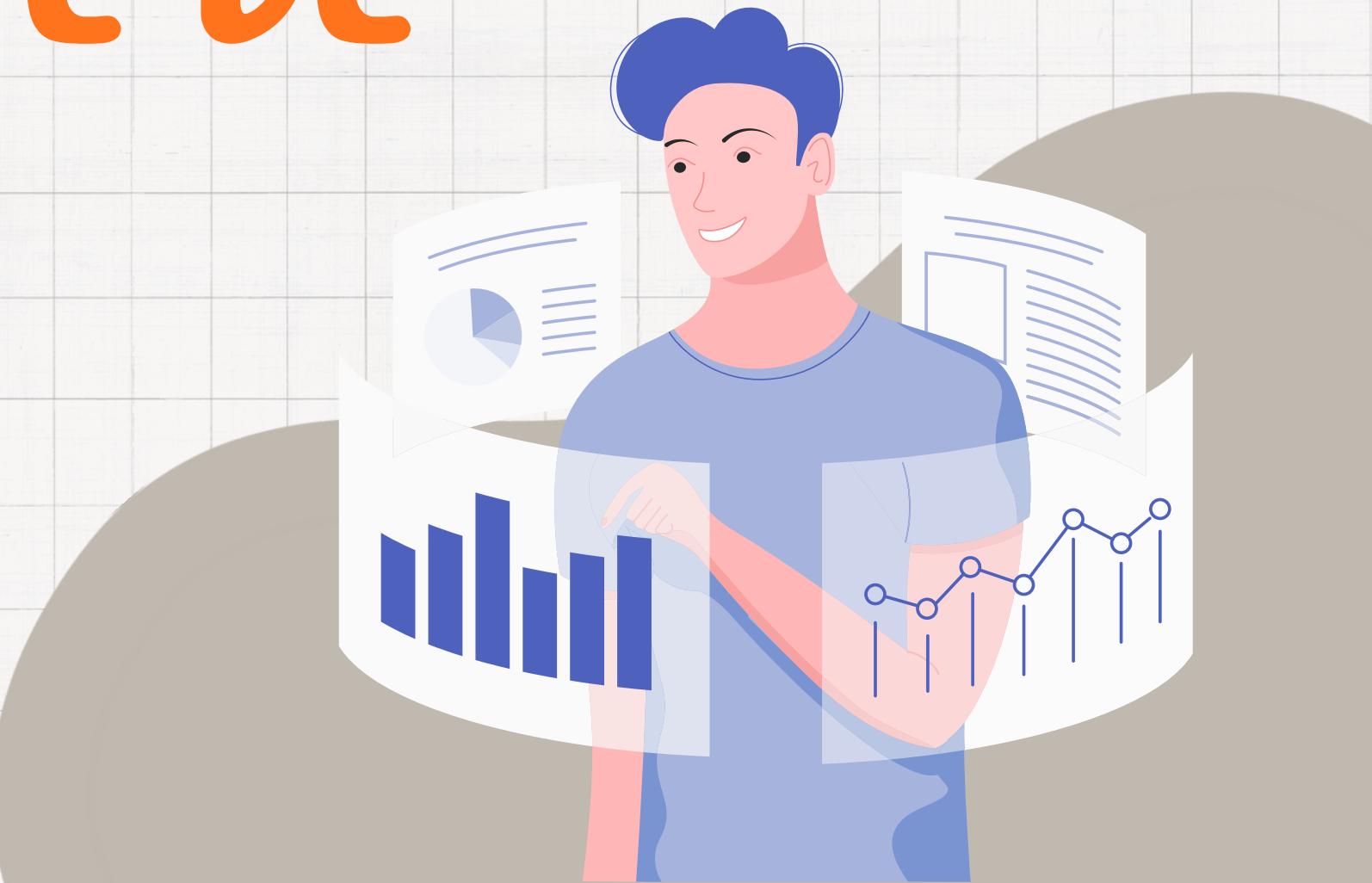


BANCO DE DADOS PARA ANALISTAS DE TESTE DE SOFTWARE

por Ingrid Lorrane



IMPORTÂNCIA DOS BANCOS DE DADOS EM TESTES DE SOFTWARE

Banco de dados desempenham um papel crítico no teste de software. Eles são responsáveis por armazenar e gerenciar os dados essenciais que as aplicações usam. Erros de banco de dados podem resultar em falhas no software, perda de dados e impacto negativo na experiência do usuário."



BANCOS DE DADOS

Um banco de dados é um sistema de armazenamento organizado e estruturado para coletar, gerenciar e recuperar informações.

Ele é projetado para armazenar dados de forma eficiente e permitir o acesso a esses dados quando necessário.

PRINCIPAIS ELEMENTOS

- Tabela: Os dados são organizados em tabelas, que consistem em linhas (registros) e colunas (campos). Cada linha representa uma entrada única e cada coluna contém um tipo específico de informação.
- Registros: São instâncias individuais de dados armazenados em uma tabela. Cada registro é uma coleção de informações relacionadas.
- Campos: São os elementos individuais de dados em uma tabela. Cada campo contém um tipo específico de informação, como texto, números, datas, etc.

FUNÇÕES CHAVES:

- Armazenamento de Dados: Os bancos de dados armazenam informações de forma persistente, tornando-as acessíveis a aplicativos e usuários.
- Recuperação de Dados: Os dados podem ser recuperados e consultados de maneira eficiente.
- Gerenciamento de Dados: Os bancos de dados permitem a organização, manutenção e atualização de informações de forma estruturada.

TIPOS DE BANCOS DE DADOS

Existem vários tipos de bancos de dados, incluindo bancos de dados SQL e NoSQL.

Bancos de dados SQL seguem um modelo relacional e são ideais para dados estruturados. Por outro lado, bancos de dados NoSQL são flexíveis e adequados para dados não estruturados ou semiestruturados

MODELAGEM DE DADOS E ENTIDADES

A modelagem de dados é o processo de criar uma representação visual da estrutura de dados em um banco de dados. Ela inclui a definição de tabelas, colunas, chaves primárias, chaves estrangeiras e relacionamentos entre entidades.

SQL (STRUCTURED QUERY LANGUAGE)

LÍNGUAGEM PADRÃO PARA LIDAR COM BANCO DE DADOS RELACIONAL

O SQL é uma linguagem de consulta estruturada usada para interagir com bancos de dados relacionais. Os analistas de teste de software frequentemente precisam escrever consultas SQL para verificar a integridade dos dados e garantir que o software funcione corretamente.

DDL (DATA DEFINITION LANGUAGE):

O DDL, ou Linguagem de Definição de Dados, é uma parte fundamental do SQL que se concentra na definição e gerenciamento da estrutura do banco de dados. Suas principais funções incluem a criação, modificação e exclusão de objetos de banco de dados, como tabelas, índices, visões e procedimentos armazenados. Os comandos DDL desempenham um papel crítico na definição da estrutura que determina como os dados serão armazenados no banco de dados.

Alguns dos principais comandos DDL incluem:

- CREATE: Usado para criar objetos de banco de dados, como tabelas, índices e visões.
- ALTER: Permite a modificação da estrutura de objetos existentes, como a adição ou remoção de colunas de uma tabela.
- DROP: Exclui objetos de banco de dados, como tabelas, índices ou procedimentos armazenados, permanentemente.

DDL é responsável por definir restrições de integridade, como chaves primárias e estrangeiras, para garantir a qualidade e consistência dos dados no banco de dados.

DML (DATA MANIPULATION LANGUAGE)

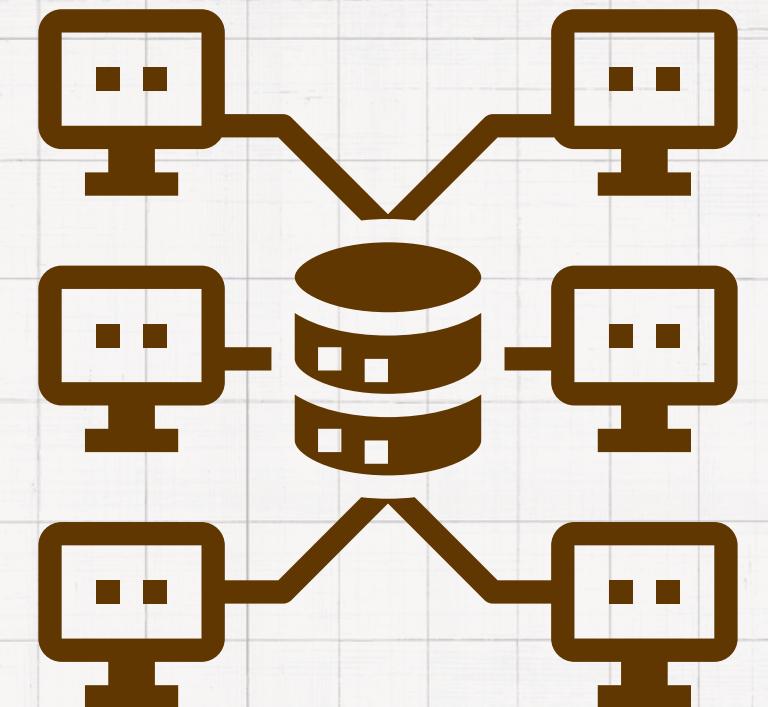
O DML, ou Linguagem de Manipulação de Dados, é a parte do SQL que lida com a manipulação dos dados armazenados em um banco de dados. Os comandos DML permitem a adição, modificação, exclusão e consulta de registros em tabelas. Eles desempenham um papel crucial na manutenção e atualização dos dados, garantindo que as informações permaneçam precisas e úteis.

Alguns dos principais comandos DML incluem:

- **INSERT INTO**: Adiciona novos registros a uma tabela com dados específicos.
- **UPDATE**: Modifica registros existentes em uma tabela com novos valores.
- **DELETE FROM**: Remove registros de uma tabela com base em condições específicas.
- **SELECT**: Recupera dados de uma ou mais tabelas, permitindo a filtragem, projeção e ordenação dos resultados.

O DML é fundamental para a interação com o banco de dados e é amplamente utilizado em aplicativos, sistemas de gerenciamento de conteúdo, análise de dados e muito mais.

DDL trata da definição da estrutura do banco de dados, enquanto o DML se concentra na manipulação dos dados dentro desse banco de dados. Ambos desempenham papéis essenciais no gerenciamento de dados e no suporte a aplicativos que dependem de bancos de dados para armazenar e acessar informações.



EXEMPLOS DE COMANDOS DDL NO SQL

CREATE TABLE:

- Comando: CREATE TABLE NomeDaTabela
- Descrição: Cria uma nova tabela no banco de dados.
- Exemplo:

```
CREATE TABLE Clientes  
( ID INT PRIMARY KEY, Nome VARCHAR(50),  
Email VARCHAR(100)  
);
```

EXEMPLOS DE COMANDOS DDL NO SQL

ALTER TABLE:

- Comando: ALTER TABLE NomeDaTabela
- Descrição: Modifica a estrutura de uma tabela existente.
- Exemplo:

```
ALTER TABLE Produtos  
ADD Desconto INT;
```

EXEMPLOS DE COMANDOS DDL NO SQL

DROP TABLE:

- Comando: `DROP TABLE NomeDaTabela`
- Descrição: Exclui uma tabela e todos os seus dados permanentemente.
- Exemplo:

`DROP TABLE Pedidos;`

EXEMPLOS DE COMANDOS DML NO SQL

INSERT INTO:

- Comando: `INSERT INTO NomeDaTabela (Colunas) VALUES (Valores)`
- Descrição: Adiciona novos registros a uma tabela com dados específicos.
- Exemplo:

```
INSERT INTO Clientes (Nome, Email)  
VALUES ('João', 'joao@email.com');
```

EXEMPLOS DE COMANDOS DML NO SQL

UPDATE:

- Comando: UPDATE NomeDaTabela SET Coluna = NovoValor WHERE Condição
- Descrição: Modifica registros existentes em uma tabela com novos valores.
- Exemplo:

```
UPDATE Produtos  
SET Preco = 29.99  
WHERE Nome = 'ProdutoXYZ';
```

EXEMPLOS DE COMANDOS DML NO SQL

DELETE FROM:

- Comando: `DELETE FROM NomeDaTabela WHERE Condição`
- Descrição: Remove registros de uma tabela com base em condições específicas.
- Exemplo:

```
DELETE FROM Pedidos  
WHERE Data < '2023-01-01';
```

EXEMPLOS DE COMANDOS DML NO SQL

SELECT:

- Comando: `SELECT Colunas FROM NomeDaTabela WHERE Condição`
- Descrição: Recupera dados de uma ou mais tabelas, permitindo a filtragem, projeção e ordenação dos resultados.
- Exemplo:

```
SELECT Nome, Email  
FROM Clientes
```

SELECT - COMANDOS DE COMPARAÇÃO

OR - O operador OR é usado para combinar condições onde pelo menos uma delas deve ser verdadeira.

`SELECT * FROM Produtos`

`WHERE Categoria = 'Eletrônicos' OR Categoria = 'Acessórios';`

Este exemplo retorna todos os produtos que pertencem à categoria 'Eletrônicos' OU 'Acessórios'.

AND - O operador AND é usado para combinar condições onde todas as condições devem ser verdadeiras

`SELECT * FROM Pedidos`

`WHERE ValorTotal > 1000 AND Status = 'Processado';`

BETWEEN - O operador BETWEEN é usado para filtrar resultados dentro de um intervalo especificado.

`SELECT * FROM Vendas`

`WHERE DataVenda BETWEEN '2023-01-01' AND '2023-02-28';`

GROUP BY NO SQL

O comando GROUP BY é utilizado em consultas SQL para agrupar linhas que têm o mesmo valor em uma ou mais colunas, geralmente seguido por funções de agregação, como COUNT, SUM, AVG, MAX ou MIN. Isso permite realizar operações sobre grupos específicos de dados em vez de sobre o conjunto inteiro.

Aqui estão alguns pontos-chave sobre o uso do GROUP BY:.

Sintaxe Básica:

```
SELECT coluna1, COUNT(coluna2)  
FROM tabela  
GROUP BY coluna1;
```

Este exemplo conta a quantidade de ocorrências de valores únicos na coluna1, agrupando os resultados por essa coluna.

GROUP BY NO SQL

Funções de Agregação:

Ao usar GROUP BY, frequentemente se acompanha com funções de agregação. Exemplos comuns incluem COUNT(), SUM(), AVG(), MAX(), MIN().

```
SELECT Categoria, AVG(Preco)  
FROM Produtos  
GROUP BY Categoria;
```

Este exemplo retorna a média dos preços dos produtos para cada categoria.

GROUP BY NO SQL

Agrupamento por Múltiplas Colunas:

Pode-se agrupar por mais de uma coluna, criando grupos mais específicos

```
SELECT Categoria, Fabricante, COUNT(*)  
FROM Produtos  
GROUP BY Categoria, Fabricante;
```

Este exemplo conta a quantidade de produtos por categoria e fabricante.

GROUP BY NO SQL

Filtragem com HAVING:

A cláusula HAVING é usada em conjunto com GROUP BY para filtrar os resultados após a agregação.

```
SELECT Categoria, AVG(Preco)  
FROM Produtos  
GROUP BY Categoria  
HAVING AVG(Preco) > 50;
```

Este exemplo retorna as categorias com uma média de preço superior a 50.

GROUP BY NO SQL

ORDER BY com GROUP BY

Pode-se ordenar os resultados usando ORDER BY junto com GROUP BY.

```
SELECT Categoria, COUNT(*)  
FROM Produtos  
GROUP BY Categoria  
ORDER BY COUNT(*) DESC;
```

Este exemplo retorna as categorias ordenadas pela quantidade de produtos em ordem decrescente.

JUNÇÕES NO SQL - JUNTANDO DADOS DE MÚLTIPAS TABELAS

Quando você tem dados espalhados em diferentes tabelas e precisa combiná-los para obter informações mais completas, você usa junções. É como unir peças de um quebra-cabeça para ter a imagem completa. Existem três tipos principais de junções:

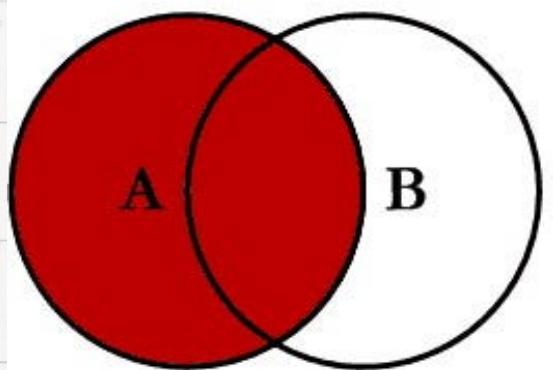
INNER JOIN (Junção Interna)

LEFT JOIN (Junção à Esquerda)

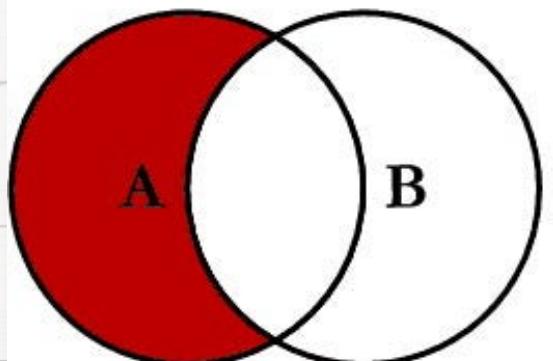
RIGHT JOIN (Junção à Direita)

JUNÇÕES NO SQL - JUNTANDO DADOS DE MÚLTIPAS TABELAS

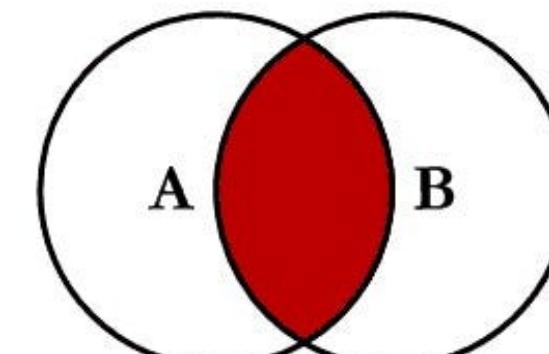
SQL JOINS



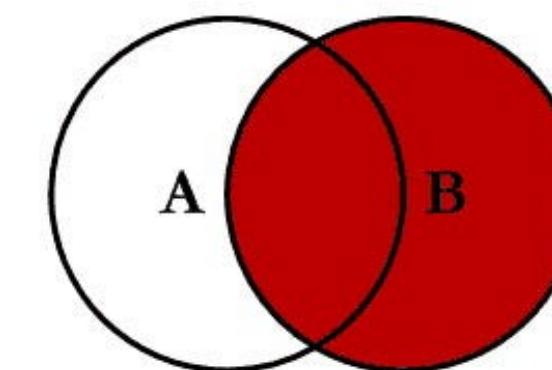
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



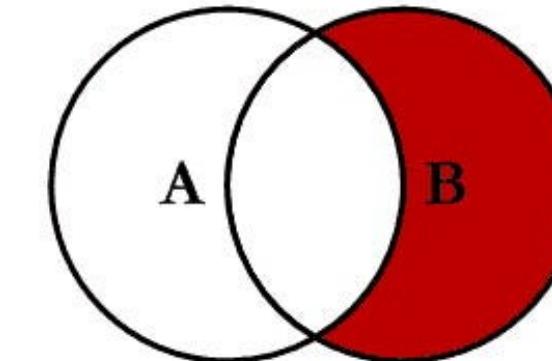
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



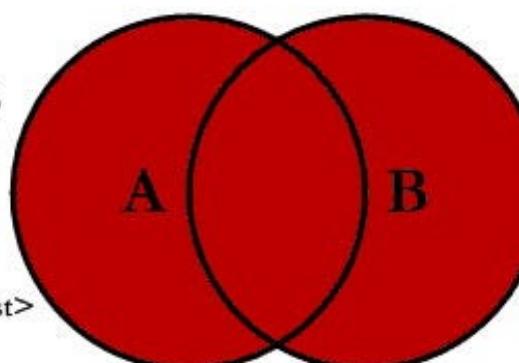
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



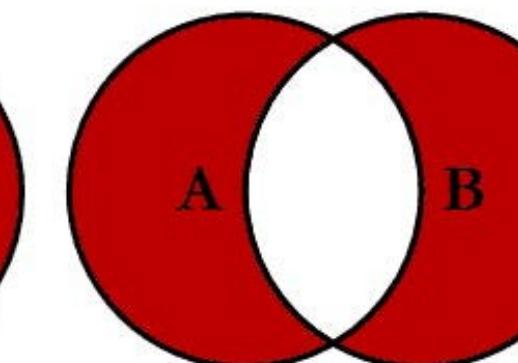
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

JUNÇÕES NO SQL - JUNTANDO DADOS DE MÚLTIPAS TABELAS

INNER JOIN (Junção Interna):

Imagine duas tabelas, A e B.

Um INNER JOIN retorna apenas as linhas que têm correspondências em ambas as tabelas.

É como pegar apenas os itens que estão em ambas as listas.

```
SELECT Clientes.Nome, Pedidos.Produto
```

```
FROM Clientes
```

```
INNER JOIN Pedidos ON (Clientes.ID = Pedidos.ClienteID);
```

Este exemplo retorna o nome do cliente e o produto que eles pediram, apenas para os clientes que fizeram pedidos.

JUNÇÕES NO SQL - JUNTANDO DADOS DE MÚLTIPAS TABELAS

RIGHT JOIN (Junção à Direita):

Um RIGHT JOIN é o oposto de um LEFT JOIN. Ele retorna todas as linhas da tabela à direita (tabela B) e as correspondências da tabela à esquerda (tabela A). Se não houver correspondência, os valores da tabela à esquerda serão NULL.

```
SELECT Clientes.Nome, Pedidos.Produto  
FROM Clientes  
RIGHT JOIN Pedidos ON Clientes.ID = Pedidos.ClienteID;
```

Este exemplo retorna o nome do cliente e o produto que eles pediram, incluindo todos os pedidos, mesmo que não haja um cliente correspondente (o campo Nome será NULL para esses casos).