INTRODUÇÃO AO SQL

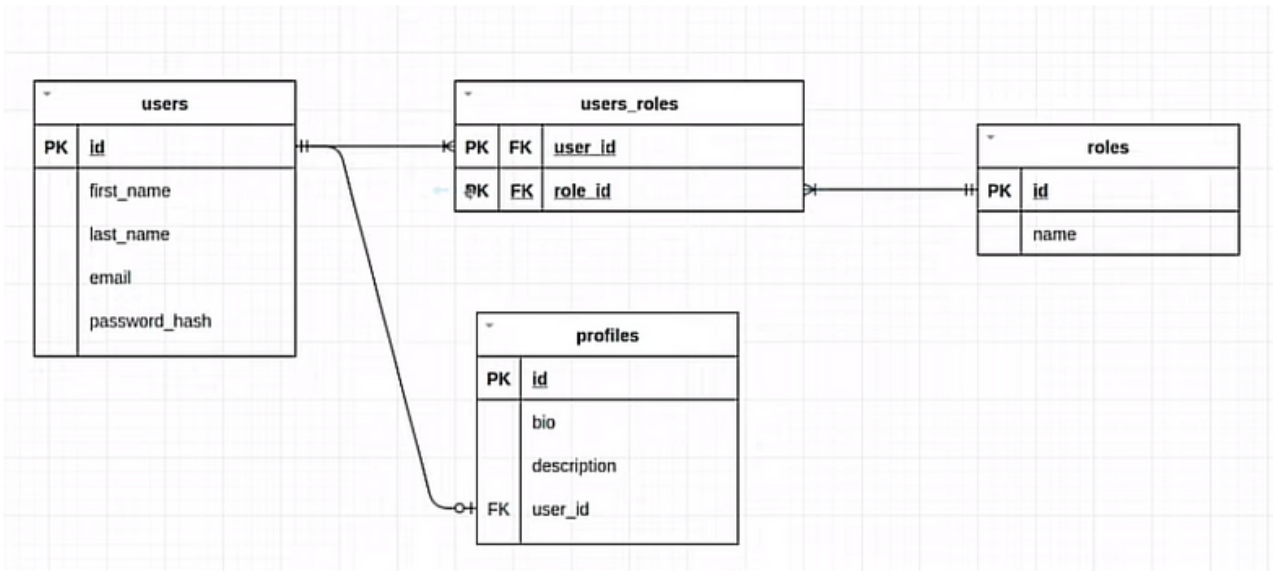
Introdução ao SQL

O que é SQL (Structured Query Language)?

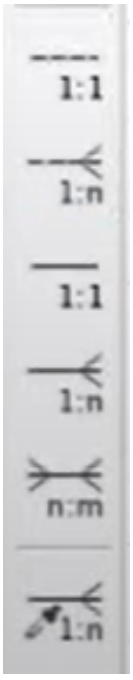
- SQL é uma linguagem de programação usada para garantir e manipular bancos de dados relacionais. Porem aos usuários criar, modificar, deletar e extrair dados de bancos de dados, além de gerenciar estruturas de banco de dados e controlar o acesso aos dados.

- A SQL é crucial no mundo dos bancos de dados pois facilita a gestão e manipulação de dados em sistemas relacionais. Permite aos usuários executar consultas complexas de maneira eficiente, garantir a integridade dos dados e estabelecer controle de acesso seguro aos dados, sendo uma habilidade essencial para profissionais de TI e análise de dados.

Tipos de Relacionamentos de tabelas



No contexto de bancos de dados relacionais, os relacionamentos One-to-One, One-to-Many e Many-to-Many referem-se à maneira como as tabelas em um banco de dados estão conectadas entre si. Vou explicar cada um deles:



Um para Um com apenas 1 chave primaria

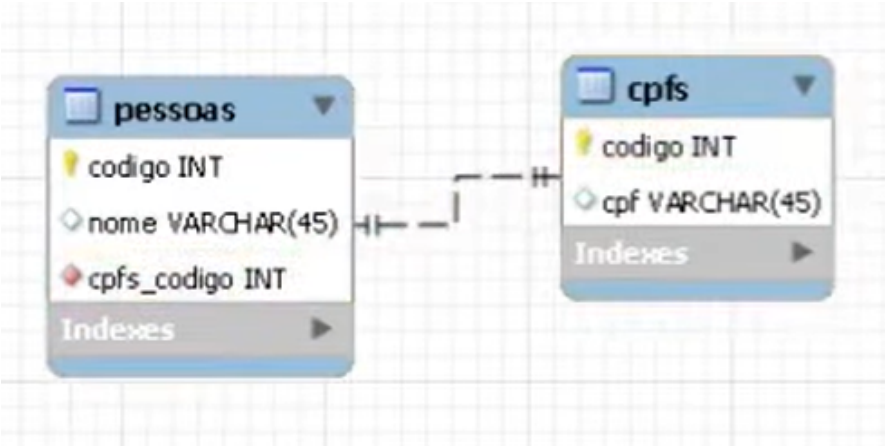
Um para Muitos

Um para Um, com 2 chaves primarias

Muitos para Muitos

1. Um para Um(1:1):

Neste tipo de relacionamento, uma linha em uma tabela está associada a no máximo uma linha em outra tabela. Isso significa que para cada registro na tabela A, há apenas um registro correspondente na tabela B, e vice-versa.



Exemplo:

Suponha que você tenha uma tabela de funcionários e uma tabela de detalhes de contato. Cada funcionário tem apenas um número de telefone associado a ele, e cada número de telefone está vinculado a apenas um funcionário.

Tabela `Funcionários`:

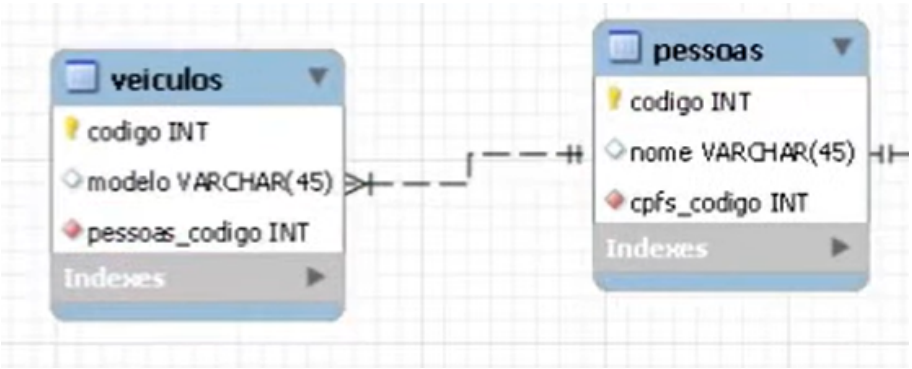
ID	Nome	Cargo
1	Alice	Gerente
2	Bob	Desenvolvedor

Tabela `Detalhes de Contato`:

ID	FuncionárioID	Telefone
1	1	123-456-7890
2	2	987-654-3210

2. Um para Muitos (1:N) / Muitos para Um (N:1):

Neste tipo de relacionamento, uma linha em uma tabela está associada a uma ou mais linhas em outra tabela. Ou seja, para cada registro na tabela A, pode haver vários registros correspondentes na tabela B, mas para cada registro na tabela B, há apenas um registro correspondente na tabela A.



Exemplo:

Considere uma tabela de departamentos e uma tabela de funcionários. Um departamento pode ter vários funcionários, mas cada funcionário pertence apenas a um departamento.

Tabela `Departamentos`:

ID	Nome
1	Vendas
2	TI

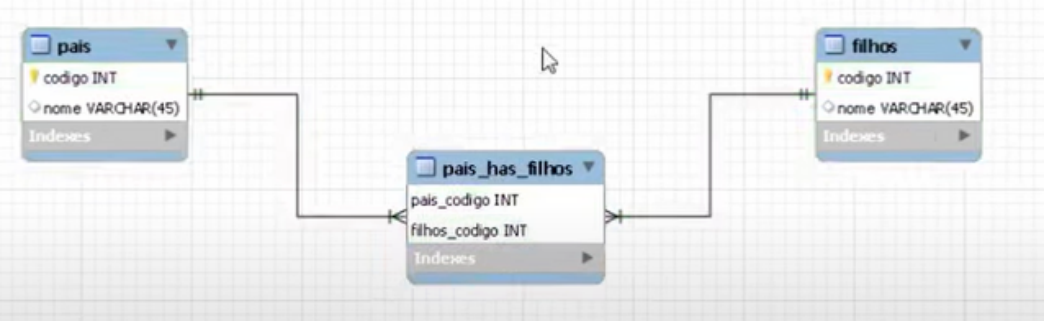
Tabela `Funcionários`:

ID	Nome	DepartamentoID
1	Alice	1
2	Bob	2
3	Charlie	1

No exemplo acima, o departamento de Vendas (ID 1) possui dois funcionários (Alice e Charlie), enquanto o departamento de TI (ID 2) possui apenas um funcionário (Bob).

4. Many-to-Many (Muitos para Muitos):

Neste tipo de relacionamento, várias linhas em uma tabela estão associadas a várias linhas em outra tabela. Isso é alcançado usando uma tabela de junção (também conhecida como tabela de associação) que mapeia as relações entre as outras duas tabelas.



Exemplo:

Considere uma situação em que os estudantes podem se inscrever em vários cursos, e cada curso pode ter vários estudantes matriculados.

Tabela `Estudantes`:

ID	Nome
1	Alice
2	Bob
3	Charlie

Tabela `Cursos`:

ID	Nome
1	Matemática
2	História
3	Ciências

Tabela de Junção `Matrículas`:

EstudanteID	CursoID
1	1
1	2
2	2
3	1
3	3

Neste exemplo, a tabela **Matrículas** mapeia os relacionamentos muitos-para-muitos entre estudantes e cursos. Alice está matriculada em Matemática e História, Bob está matriculado em História, e Charlie está matriculado em Matemática e Ciências.

Espero que esses exemplos clarifiquem os conceitos de relacionamentos One-to-One, One-to-Many e Many-to-Many no contexto de bancos de dados SQL! Se você tiver mais perguntas ou precisar de mais exemplos, sinta-se à vontade para perguntar.

Exemplo de tudo interligado:

Descrição do site Filmes na Estante:

O site é como uma vitrine onde a pessoa pode buscar informações sobre **filmes**, **diretores** e **elenco**. Cada filme deve ter pelo menos um diretor e deve pertencer a uma **categoria** (ação, comédia, drama etc.). A página de cada filme deverá ter uma sinopse desse filme, pode conter imagens do filme selecionado (5 no máximo) e um trailer se existir. Também deve ter um link para a página do diretor do filme.

Na página do diretor, deve ter um pequeno resumo biográfico e pode conter uma foto dele.

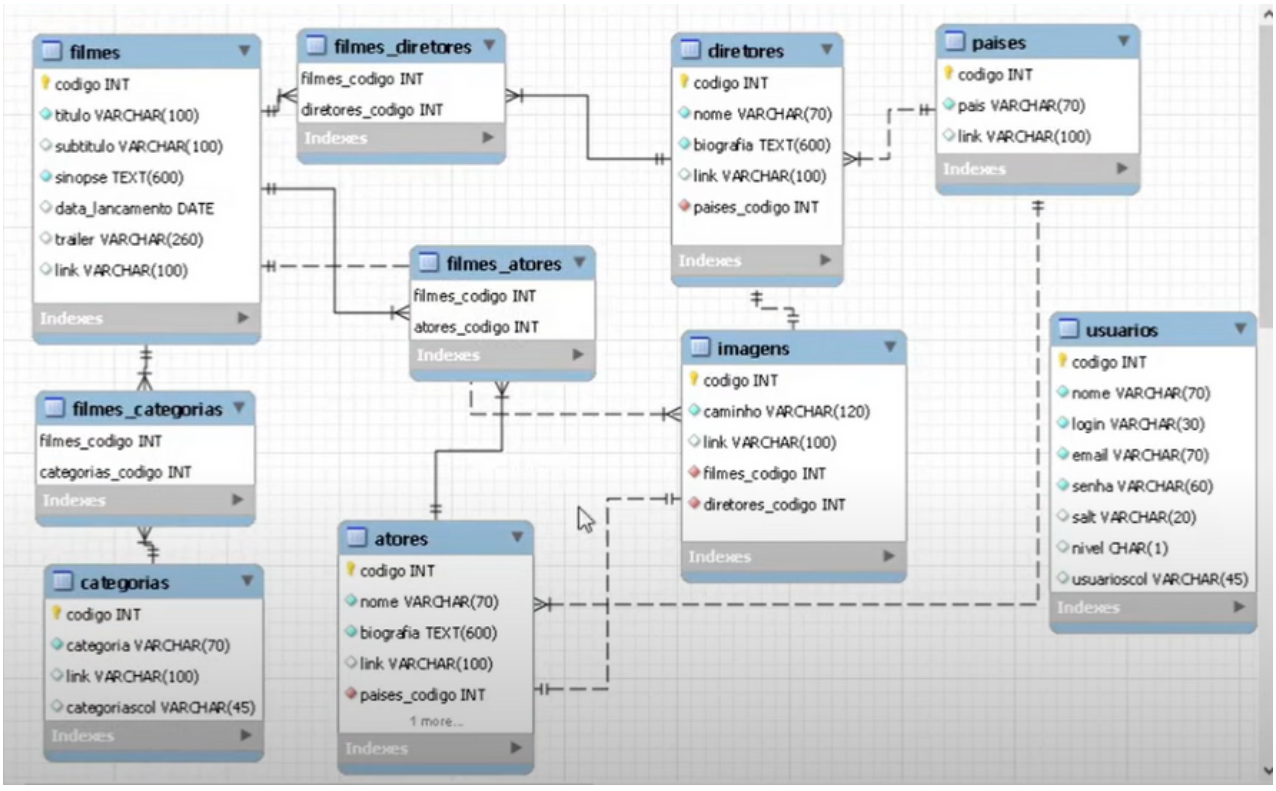
Na página do(a) ator/atriz também deve ter um pequeno resumo biográfico e pode conter uma foto dele(a).

Tanto na página do diretor quanto na página do ator/atriz, deve conter uma lista de filmes que eles trabalharam.

Seria interessante se o usuário pudesse fazer uma buscar pelo **país** do diretor ou ator/atriz.

Dentro da área de **administração**, o usuário deverá ser capaz de cadastrar, editar ou excluir:

- Filmes
- Diretores
- Atores/atrizes
- Banners
- Categorias



Operações Básicas do SQL

```
CREATE TABLE jovem_aprendiz (
        id SERIAL PRIMARY KEY,
        nome VARCHAR (255) NOT NULL,
        idade INT NOT NULL,
        departamento VARCHAR (255)
);
```

```
INSERT INTO jovem_aprendiz
(nome, idade, departamento)
VALUES ('Ana Silva',18,'Recursos Humanos'),
            ('João Souza',19,'Financeiro'),
            ('Marta Oliveira',20,'Marketing');
```

```
SELECT * FROM jovem_aprendiz;
```

```
UPDATE jovem_aprendiz
SET departamento='TI'
WHERE nome='Ana Silva';
```

```
DELETE FROM jovem_aprendiz
WHERE nome='Marta Oliveira';
```

Funções e Operadores no SQL

O SQL fornece uma ampla gama de funções e operadores que ajudam na manipulação e análise de dados armazenados em bancos de dados relacionais. Abaixo estão algumas das categorias mais importantes.

Funções de Agregação:

Funções de agregação são usadas para computar valores derivados de colunas em um conjunto de linhas, fornecendo insights estatísticos úteis.

- COUNT: Conta o número de linhas.
- SUM: Soma os valores de uma coluna.
- AVG: Calcula a média dos valores de uma coluna.
- MAX e MIN: Retorna o maior e o maior valor de uma coluna, respectivamente.

- Exemplo:

SELECT COUNT(*) AS total, AVG(idade) AS media_idade FROM jovem_aprendiz;

Operadores Lógicos:

- Operadores lógicos são utilizados para comparar valores e fazer controle condicional dentro das queries.
- AND: Verifica se ambas as condições são verdadeiras.
- OR: Verifica se pelo menos uma das condições é verdadeira.
- NOT: Nega a condição fornecida.

- Exemplo:

SELECT * FROM jovem_aprendiz WHERE idade >=18 AND departamento = ‘Marketing’;

Operadores de Comparação

Operadores de comparação são usados para comparar valores.

- =: Igual a;
- != ou <>; Diferente de;
- <: Menor que;
- >: Maior que;
- <=: Menor ou igual a.
- >=: Maior ou igual a.

- Exemplo:
SELECT * FROM jovem_aprendiz WHERE idade >=20;

Funções de Data e Hora:

Funções de data e hora ajudam a manipular e formatar valores de data e hora.

- NOW(): Retorna a data e hora atuais;
- DATE_PART(): Extrai uma parte da data ou hora;
- AGE(): Calcula a idade baseada em datas.

- Exemplo:
SELECT NOW(), AGE(NOW(), data_nascimento) FROM jovem_aprendiz;

Postgresql

O POSTGRESQL É UM SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS RELACIONAL DE CÓDIGO ABERTO, SUA ORIGEM REMONTA A 1986 COMO PARTE DO PROJETO POSTRES NA UNIVERSIDADE DA CALIFÓRNIA EM BERKELEY. MICHAEL STONEBRAKER, QUE TAMBÉM ESTAVA ENVOLVIDO NO DESENVOLVIMENTO INICIAL DO INGRES, LIDEROU ESTE PROJETO. O POSTRES TORNOU-SE POSTGRESQL EM 1996 COM A ADIÇÃO DE SUPORTE AO PADRÃO SQL.

Características Princípais

- Código Aberto;
- Suporta propriedades ACID (atomicity, consistency, isolation, durability) Garantindo transações seguras e confiáveis
- Oferece suporte a definiçõa de tipos de dados customizados, permitindo que os usuários definam seus próprios tipos de dados;
- É ALTAMENTE EXTENSÍVEL, PERMITINDO QUE OS USUÁRTOS ADICIOMEM NOVAS FUNÇÕES, OPERADORES, TIPOS DE DADOS E MUITO MAIS.
- INDEXAÇÃO E OTIMIZAÇÃO DE CONSULTAS: - POSSUI UM PLANEJADOR DE CONSULTAS SOFISTICADO E SUPORTE A INDEXAÇÃO AVANÇADA, O QUE AJUDA A OTIMIZAR O DESEMPENHO DAS CONSULTAS,

- SUPORTE A LINGUAGENS DE PROCEDIMENTOS: - OFERECE SUPORTE A VÁRIAS LINGUAGENS DE PROCEDIMENTOS, INCLUINDO PL/PGSOL, PL/PYTHON, PL/PERL E PL/TCL.

- CONFORMIDADE COM SQL: - OFERECE ALTA CONFORMIDADE COM OS PADRÕES SOL, FACILITANDO A MIGRAÇÃO DE OUTROS SISTEMAS DE BANCO DE DADOS RELACIONAL.

- RECURSOS DE SEGURANÇA: - INCLUI VÁRIOS RECURSOS DE SEGURANCA, COMO CONTROLE DE ACESSO BASEADO EM FUNÇÃO, CRIPTOGRAFIA DE DADOS E AUTENTICAÇÃO DE CERTIFICADO SSL.

- SUPORTE A OBJETOS E JSON: - ALEM DE SER UM ROBNS, POSSUI SUPORTE PARA ARMAZENAMENTO E MANIPULAÇÃO DE OBJETOS E DADOS JSON, PERMITINDO FLEXIBILIDADE NA GESTÃO DE DADOS,

- PARTICIONAMENTO DE TABELA: - OFERECE SUPORTE A PARTICIONAMENTO DE TABELAS, AJUDANDO NA ORGANIZAÇÃO DE GRANDES CONJUNTOS DE DADOS E MELHORANDO O DESEMPENHO,

- REPLICAÇAO E BACKUP: - SUPORTE A REPLICAÇÃO SÍNCRONA E ASSÍNCRONA, ALÉM DE ROBUSTAS SOLUÇÕES DE BACKUP E RECUPERAÇÃO.

TIPOS DE DADOS ESPECIAIS EM POSTGRESQL

- JSON -> JSON (JavaScript Object Notation). Mantém o formato original do texto JSON

- JSONB -> Uma versão binárla do tipo JSON. Permite indexação e é mais eficiente em operações de busca e atualização. Embora as inserções sejam um pouco mais lentas do que o tipo JSON

- HSTORE • tipo de dado "chave-valor". É átil para armazenar coleções de pares chave-valor em uma única coluna PostgreSQL

- Arays -› você armazene arrays de qualquer tipo de dado base, incluindo números, strings e outros tipos de dados

- UUID -> Usado para armazenar identificadores universais únicos.

- Range Types -› Representam uma fala de valores.

- Geometric Types -> Permitem representar pontos, linhas, polígonos, etc., para operações geométricas.

- Network Address -› Especializado no armazenamento de endereços IP e CIDR.

- Bit Strings -> Usados para armazenar strings de bits.

ÍNDICES E PERFORMANCE

Índices são estruturas de dados que permitem uma busca mais rápida por registros em uma tabela, funcionando como uma espécie de "mapa" que aponta diretamente para os locais onde os dados estão armazenados

- Melhoria na Velocidade de Consulta;
- Eficiência em Operações de JOIN;
- Ordenação e Agrupamento;
- Restrições de Unicidade;
- Manutenção de Índices.

Imagine que você tem uma planilha Excel enorme com muitas colunas e linhas, repleta de dados variados. Agora, suponha que você precise encontrar informações específicas nessa planilha. Uma abordagem seria percorrer cada linha e cada coluna até encontrar o que procura, mas isso seria muito demorado e ineficiente.

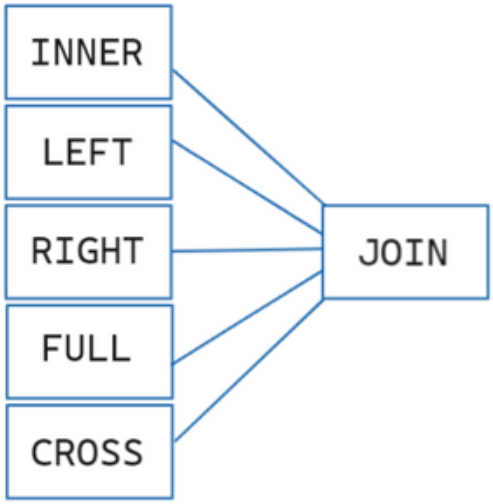
Agora, pense na função de "filtro" no Excel, onde você pode escolher uma coluna específica e aplicar um filtro para mostrar apenas as linhas que correspondem a certo critério. Ao fazer isso, você elimina uma grande quantidade de dados irrelevantes e foca apenas nos dados que são Importantes para sua busca, agilizando significativamente o processo de encontrar o que precisa.

Essa funcionalidade de filtragem no Excel é semelhante à maneira como os Índices funcionam em um banco de dados. Um Índice em um banco de dados é como um filtro que é aplicado a uma coluna específica (ou conjunto de colunas) para acelerar as buscas. Ao invés de percorrer todas as linhas de uma tabela, o banco de dados pode usar o índice para ir diretamente às linhas que satisfazem a consulta, economizando tempo e recursos computacionais.

Join

(https://www.alura.com.br/artigos/join-e-seus-tipos)

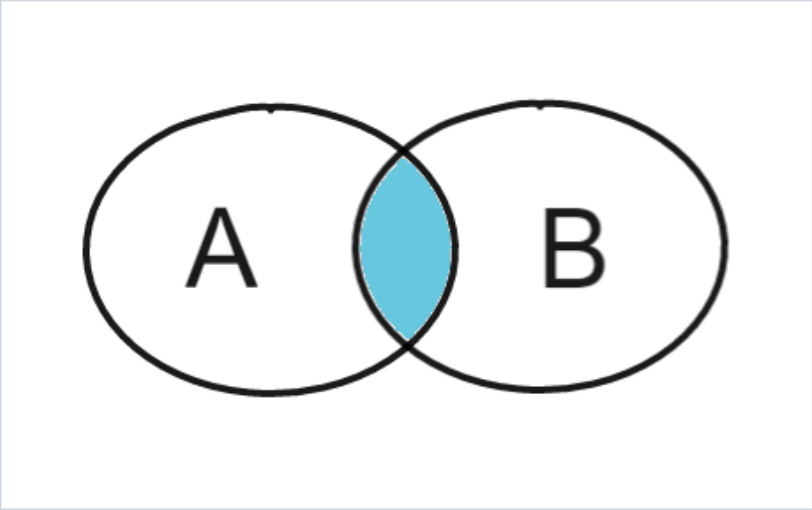
"Join" é um termo comumente utilizado em bancos de dados e linguagens de consulta, como SQL (Structured Query Language), para combinar dados de duas ou mais tabelas com base em uma condição relacionada entre elas. Quando você utiliza a cláusula "JOIN" em uma consulta SQL, você está instruindo o banco de dados a combinar registros de diferentes tabelas com base em uma chave comum, criando assim um conjunto de resultados mais completo e significativo.



Com esse intuito, foi criada a cláusula JOIN ou junção que é utilizada para realizar a combinação de colunas de uma ou mais tabelas em um única query a partir de uma coluna em comum entre as tabelas. Existem cinco tipos de JOIN, que realizam consultas de formas diferentes nas tabelas do banco de dados, o INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN e CROSS JOIN. Neste artigo, vamos abordar como cada um deles funciona.

INNER JOIN

O `INNER JOIN` é utilizado quando queremos retornar os registros que tenham correspondência nas duas tabelas presentes na junção. Observe que, na imagem a seguir, é utilizado o `INNER JOIN`. Nesse caso apenas os registros que estão na interseção do conjunto A com o conjunto B são retornados:



Trazendo para o SQL, vamos utilizar o seguinte exemplo: em um banco de dados de uma empresa, existem duas tabelas que se relacionam entre si, sendo elas a tabela de funcionários e a tabela de cargos. A primeira possui os campos **código do funcionário**, **nome** e **código do cargo**:

COD_FUNCIONARIO	NOME	COD_CARGO
1	JOSE	3
2	DANIELLA	1
3	ANA	2
4	CARLOS	(null)

E a segunda possui os campos **código do cargo** e **descrição dos cargos**:

COD_CARGO	DESCRIÇÃO
1	VENDEDOR
2	CAIXA
3	GERENTE
4	ENTREGADOR

- **SELECT com INNER JOIN:**

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

```
SELECT A.nome, A.cod_cargo, B.descrição FROM funcionario A INNER JOIN cargo B ON A.cod_cargo = B.cod_cargo;
```

- Informando no `SELECT` os campos que serão retornados no resultado:

```
SELECT A.nome, A.cod_cargo, B.descrição
```

- Realizando a junção entre as tabelas:

```
FROM funcionario A INNER JOIN cargo B
```

- Condição que os registros precisam corresponder para serem retornados:

```
ON A.cod_cargo = B.cod_cargo;
```

Resultado

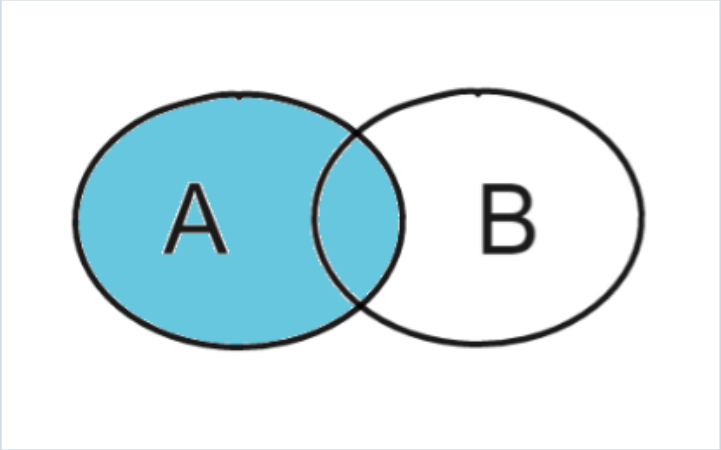
```
SELECT A.nome, A.cod_cargo, B.descrição
FROM funcionario A INNER JOIN cargo B
ON A.cod_cargo = B.cod_cargo;
```

NOME	COD_CARGO	DESCRIÇÃO
DANIELLA	1	VENDEDOR
ANA	2	CAIXA
JOSE	3	GERENTE

Foram retornados todos funcionários, com exceção do funcionário Carlos que não tem um cargo associado a ele e do cargo entregador, pois não está associado a nenhum funcionário.

LEFT JOIN

O `LEFT JOIN` é utilizado quando queremos retornar apenas os registros da tabela da esquerda (tabela que está antes da cláusula *LEFT JOIN*) e os registros que tenham correspondência na tabela da direita. Observe que, na imagem a seguir, é utilizado o `LEFT JOIN`. Nesse caso, todos os registros do conjunto A e apenas os registros que estão na interseção do conjunto A com o conjunto B seriam retornados:



- **SELECT com LEFT JOIN:**

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

```
SELECT A.nome, A.cod_cargo, B.descrição FROM funcionario A LEFT JOIN cargo B ON A.cod_cargo = B.cod_cargo;
```

Resultado

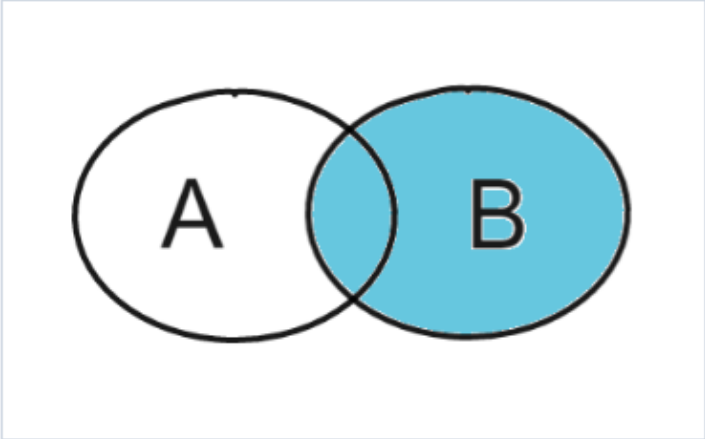
```
SELECT A.nome,A.cod_cargo,B.descrição
FROM funcionario A LEFT JOIN cargo B
ON A.cod_cargo = B.cod_cargo;
```

NOME	COD_CARGO	DESCRIÇÃO
DANIELLA	1	VENDEDOR
ANA	2	CAIXA
JOSE	3	GERENTE
CARLOS	(null)	(null)

Foram retornados todos funcionários, tem um cargo associado a ele, juntamente com todos os funcionários que não tem um cargo associado.

RIGHT JOIN

O `RIGHT JOIN` é utilizado quando queremos retornar apenas os registros da tabela da direita(tabela que está após a cláusula *RIGHT JOIN*) e os registros que tenham correspondência na tabela da esquerda. Observe que, na imagem a seguir, é utilizado o `RIGHT JOIN`. Nesse caso, todos os registros do conjunto B e apenas os registros que estão na interseção do conjunto A com o conjunto B seriam retornados:



- **SELECT com RIGHT JOIN:**

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

```
SELECT A.nome,A.cod_cargo,B.descrição FROM funcionario A RIGHT JOIN cargo B ON A.cod_cargo = B.cod_cargo;
```

Resultado

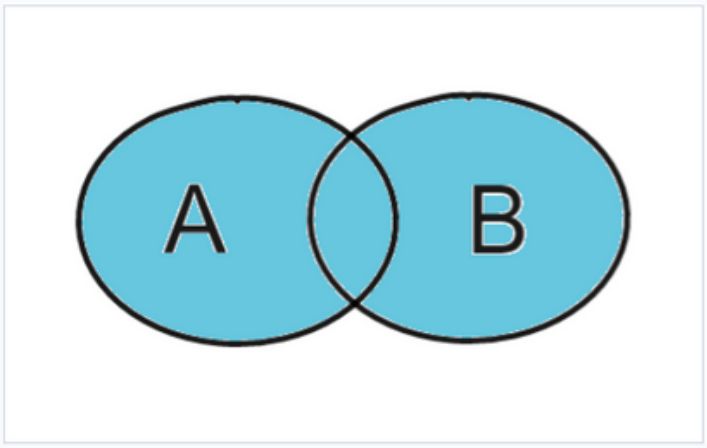
```
SELECT A.nome,A.cod_cargo,B.descrição
FROM funcionario A RIGHT JOIN cargo B
ON A.cod cargo = B.cod cargo;
```

NOME	COD_CARGO	DESCRIÇÃO
JOSE	3	GERENTE
DANIELLA	1	VENDEDOR
ANA	2	CAIXA
(null)	(null)	ENTREGADOR

Foram retornados todos os cargos que estão associados a um funcionário, juntamente com todos os cargos que não estão associados a funcionário.

FULL JOIN

O `FULL JOIN` é utilizado quando queremos retornar registros que tenham correspondência em qualquer uma das tabelas presentes na junção. Observe que, na imagem a seguir, é utilizado o `FULL JOIN`. Nesse caso, todos os registros do conjunto B e todos os registros do conjunto A seriam retornados:



- **SELECT com FULL JOIN:**

```
SELECT <select_list> FROM Tabela A FULL JOIN Tabela B ON A.Key = B.Key
```

```
SELECT A.nome, A.cod_cargo, B.descrição FROM funcionario A FULL JOIN cargo B ON A.cod_cargo = B.cod_cargo;
```

Resultado

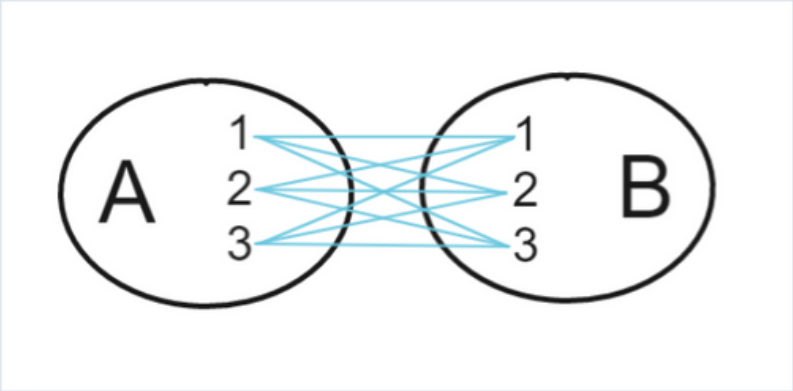
```
SELECT A.nome,A.cod_cargo,B.descrição
FROM funcionario A FULL JOIN cargo B
ON A.cod_cargo = B.cod_cargo;
```

NOME	COD_CARGO	DESCRIÇÃO
JOSE	3	GERENTE
DANIELLA	1	VENDEDOR
ANA	2	CAIXA
CARLOS	(null)	(null)
(null)	(null)	ENTREGADOR

Foram retornados todos os funcionários, mesmo os que não estão associados a um cargo e todos os cargos, mesmo os que não estão associados a um funcionário.

CROSS JOIN

O `CROSS JOIN` é utilizado quando queremos retornar os registros realizando um cruzamento entre os dados das tabelas presentes na junção. Observe que, na imagem a seguir, é utilizado o `CROSS JOIN`. Nesse caso, todos os registros do conjunto A realizam um cruzamento com todos os registros do conjunto B para serem retornados:



- **SELECT com CROSS JOIN:**

```
SELECT <select_list> FROM Tabela A CROSS  JOIN Tabela B
```

```
SELECT A.nome, A.cod_cargo, B.descrição, B.cod_cargo FROM funcionario A CROSS JOIN cargo B;
```

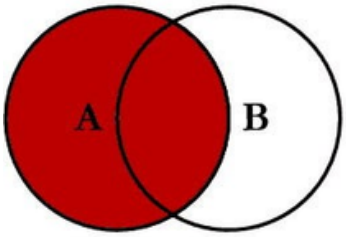
Resultado

```
SELECT A.nome,A.cod_cargo,B.descrição,B.cod_cargo
FROM funcionario A CROSS JOIN cargo B;
```

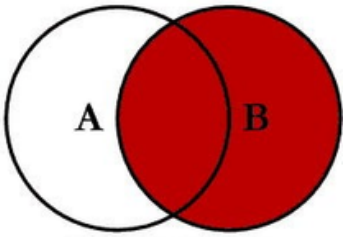
NOME	COD_CARGO	DESCRIÇÃO	COD_CARGO_1
JOSE	3	VENDEDOR	1
DANIELLA	1	VENDEDOR	1
ANA	2	VENDEDOR	1
CARLOS	(null)	VENDEDOR	1
JOSE	3	CAIXA	2
DANIELLA	1	CAIXA	2
ANA	2	CAIXA	2
CARLOS	(null)	CAIXA	2
JOSE	3	GERENTE	3
DANIELLA	1	GERENTE	3
ANA	2	GERENTE	3
CARLOS	(null)	GERENTE	3
JOSE	3	ENTREGADOR	4
DANIELLA	1	ENTREGADOR	4
ANA	2	ENTREGADOR	4
CARLOS	(null)	ENTREGADOR	4

Foi realizado o cruzamento entre todos os registros da tabela de funcionários com todos os registros da tabela de cargos.

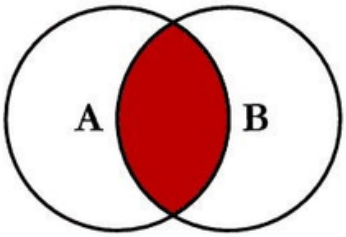
SQL JOINS



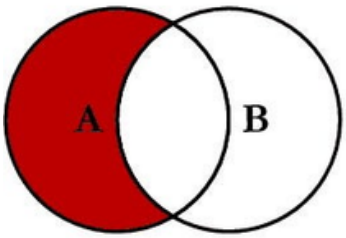

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



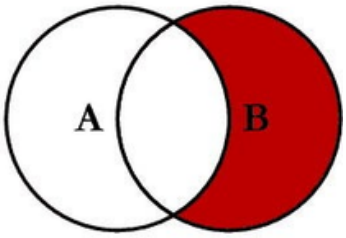

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



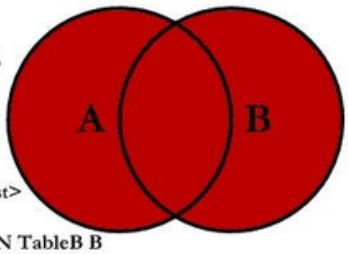

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



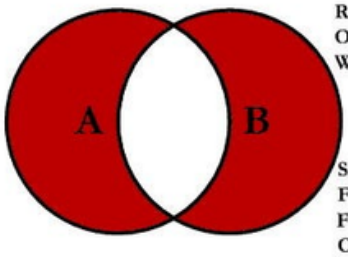

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
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
```