Criando Bancos e Tabelas no PostgreSQL

16.5

https://www.guru99.com/pt/postgresql-joins-left-right.html

Aula 7

Introdução a linguagem SQL

Sub-linguagens do SQL

Vamos estudar algumas "sub-linguagens" do SQL:

- **Linguagem de Definição de Dados – DDL**
- Linguagem de Manipulação de Dados – DML
- Linguagem de Controle de Dados - DCL

DDL - Tipos de dados)

tipo de dado pode ser:

SMALLINT | INTEGER | BIGINTEGER | FLOAT | DOUBLE PRECISION

I {DECIMAL | NUMERIC}[(precision [, scale])]

I DATE I TIME I TIMESTAMP

I {CHAR | CHARACTER I CHARACTER VARYING I VARCHAR}[(int)]

I CLOB I BLOB

DDL - Comando ALTER TABLE - Padrão ISO

Modifica tabelas já definidas

Sintaxe geral de um Comando ALTER TABLE

```
ALTER TABLE <nome da tabela>
    ADD <definição de Coluna>
    ADD <Restrição de integridade> - Chaves primária,Candidatas, Estrangeiras.
    ALTER <definição de Coluna>  SET DATA TYPE <Tipo>
    ALTER <definição de Coluna> {SETlDROP} DEFAULT<default>
    ALTER <definição de Coluna> {SETlDROP} NOT NULL
    DROP <definição de Coluna>
    DROP CHECK       I DROP <nome da chave>
    DROP UNIQUE (<definição de Coluna>, ...)
    DROP PRIMARY KEY
    DROP FOREIGN KEY
    RENAME <novo nome> - Renomeia a tabela
    RENAME <Atributo> TO <novo atributo>
```

onde <definição de coluna> pode ser:

```
    <Nome Atributo> <Tipo de Dado>
    {SETlDROP} [NULL] I [DEFAULT default-value ]
```

ADD

Adicionando uma coluna.

O comando então será:

```
-- adicionar telefone VARCHAR(30)
ALTER TABLE aluno
        ADD telefone VARCHAR(30);
```

DDL - Comando ALTER TABLE - Padrão ISO

Modifica tabelas já definidas

Sintaxe geral de um Comando ALTER TABLE

```
ALTER TABLE <nome da tabela>
        ADD <definição de Coluna>
        ADD <Restrição de integridade> - Chaves primária,Candidatas, Estrangeiras.
        ALTER <definição de Coluna>  SET DATA TYPE <Tipo>
        ALTER <definição de Coluna> {SETlDROP} DEFAULT<default>
        ALTER <definição de Coluna> {SETlDROP} NOT NULL
        DROP <definição de Coluna>
        DROP CHECK       I DROP <nome da chave>
        DROP UNIQUE (<definição de Coluna>, ...)
        DROP PRIMARY KEY
        DROP FOREIGN KEY
        RENAME <novo nome> - Renomeia a tabela
        RENAME <Atributo> TO <novo atributo>
```

onde <definição de coluna> pode ser:

```
        <Nome Atributo> <Tipo de Dado>
        {SETlDROP} [NULL] I [DEFAULT default-value ]
```

ALTER

O comando então será:

```
-- idade DECIMAL(3)  para INTEGER(2)
ALTER TABLE aluno
        ALTER idade SET DATA TYPE INTEGER (2);
```

DDL - Comando ALTER TABLE - Padrão ISO

Modifica tabelas já definidas

Sintaxe geral de um Comando ALTER TABLE

```
ALTER TABLE <nome da tabela>
        ADD <definição de Coluna>
        ADD <Restrição de integridade> - Chaves primária,Candidatas, Estrangeiras.
        ALTER <definição de Coluna>  SET DATA TYPE <Tipo>
        ALTER <definição de Coluna> {SETlDROP} DEFAULT<default>
        ALTER <definição de Coluna> {SETlDROP} NOT NULL
        DROP <definição de Coluna>
        DROP CHECK
        DROP <nome da chave>
        DROP UNIQUE (<definição de Coluna>, ...)
        DROP PRIMARY KEY
        DROP FOREIGN KEY
        RENAME <novo nome> - Renomeia a tabela
        RENAME <Atributo> TO <novo atributo>
```

onde <definição de coluna> pode ser:

```
        <Nome Atributo> <Tipo de Dado>
        {SETlDROP} [NULL] I [DEFAULT default-value ]
```

DROP

Apagando uma coluna.

O comando então será:

```
-- apagar a coluna telefone
ALTER TABLE aluno
        DROP telefone;
```

pgAdmin 4

pgAdmin File Object Tools Help

Browser

- Tables (1)
  - circuitos
    - Columns (9)
      - circuitid
      - circuitref
      - name
      - location
      - country
      - lat
      - lng
      - alt
      - url
    - Constraints
    - Indexes
    - RLS Policies
    - Rules
    - Triggers
  - Trigger Functions
  - Types
  - Views
- Subscriptions
- postgres
- tes01
- teste
- usp

Dashboard Properties SQL Statistics Dependencies Dependents formula1/post... formula1/post... formula1/postgr

formula1/postgres@PostgreSQL 14

No limit

Query Query History

```
1  SELECT * FROM circuitos;
```

Data output Messages Notifications

	circuitid [PK] numeric (4)	circuitref character varying (20)	name character varying (60)	location character varying (40)	country character varying (20)	lat real	lng real	alt real
1	1	albert_park	Albert Park Grand Pri...	Melbourne	Australia	-37.8497	144.968	
2	2	sepang	Sepang International ...	Kuala Lumpur	Malaysia	2.76083	101.738	
3	3	bahrain	Bahrain International ...	Sakhir	Bahrain	26.0325	50.5106	
4	4	catalunya	Circuit de Barcelona-...	Montmeló	Spain	41.57	2.26111	
5	5	istanbul	Istanbul Park	Istanbul	Turkey	40.9517	29.405	
6	6	monaco	Circuit de Monaco	Monte-Carlo	Monaco	43.7347	7.42056	
7	7	villeneuve	Circuit Gilles Villeneuve	Montreal	Canada	45.5	-73.5228	
8	8	magny_cours	Circuit de Nevers Ma...	Magny Cours	France	46.8642	3.16361	
9	9	silverstone	Silverstone Circuit	Silverstone	UK	52.0786	-1.01694	

DDL - Comando ALTER TABLE - Padrão ISO

Modifica tabelas já definidas

Sintaxe geral de um Comando ALTER TABLE

```
ALTER TABLE <nome da tabela>
        ADD <definição de Coluna>
        ADD <Restrição de integridade> - Chaves primária,Candidatas, Estrangeiras.
        ALTER <definição de Coluna>  SET DATA TYPE <Tipo>
        ALTER <definição de Coluna> {SETlDROP} DEFAULT<default>
        ALTER <definição de Coluna> {SETlDROP} NOT NULL
        DROP <definição de Coluna>
        DROP CHECK
        DROP <nome da chave>
        DROP UNIQUE (<definição de Coluna>, ...)
        DROP PRIMARY KEY
        DROP FOREIGN KEY
        RENAME <novo nome> - Renomeia a tabela
        RENAME <Atributo> TO <novo atributo>
```

onde <definição de coluna> pode ser:

```
        <Nome Atributo> <Tipo de Dado>
        {SETlDROP} [NULL] I [DEFAULT default-value ]
```

RENAME

Mudando o nome de uma coluna.

O comando então será:

```
-- mudar a coluna cidade para cidades
ALTER TABLE aluno
        RENAME cidade TO cidades;
```

DDL - Comando DROP TABLE

Remove completamente uma tabela e sua definição

Sintaxe geral de um Comando DROP TABLE

```
DROP TABLE [IF EXISTS] <nome da tabela> [, ...]
        [CASCADE | RESTRICT];
```

CASCADE: todas as visões e restrições (constraints) que referenciam o atributo são removidas automaticamente

RESTRICT: O atributo só é removido se não houver nenhuma visão ou restrição que o referencie

DROP TABLE

Exemplo: (não fazer, ou se fizer recrie a tabela alunos)

```
-- apagar a tabela aluno
DROP TABLE aluno;
```

Visões views

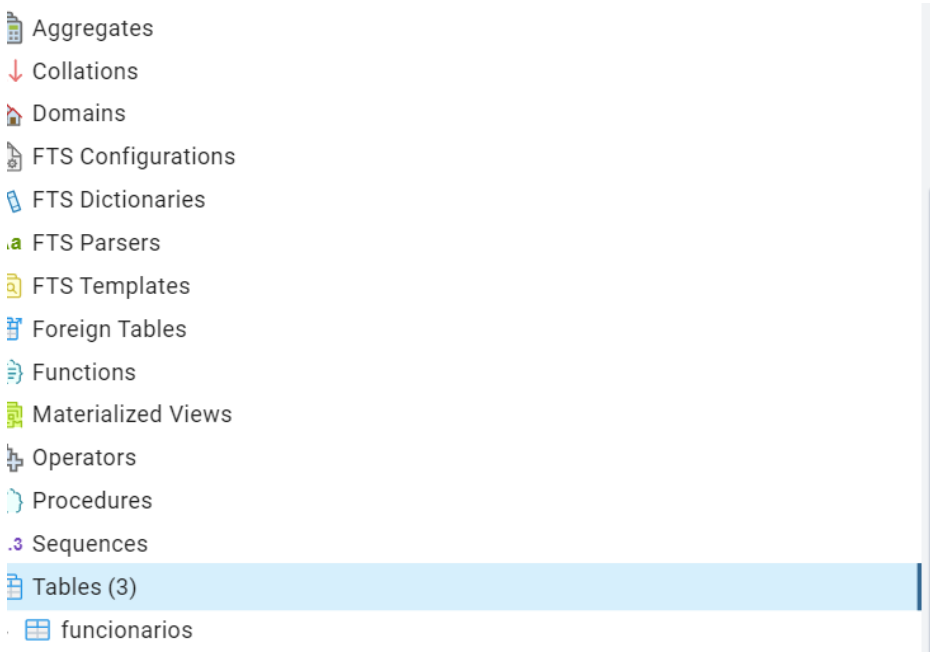
Views

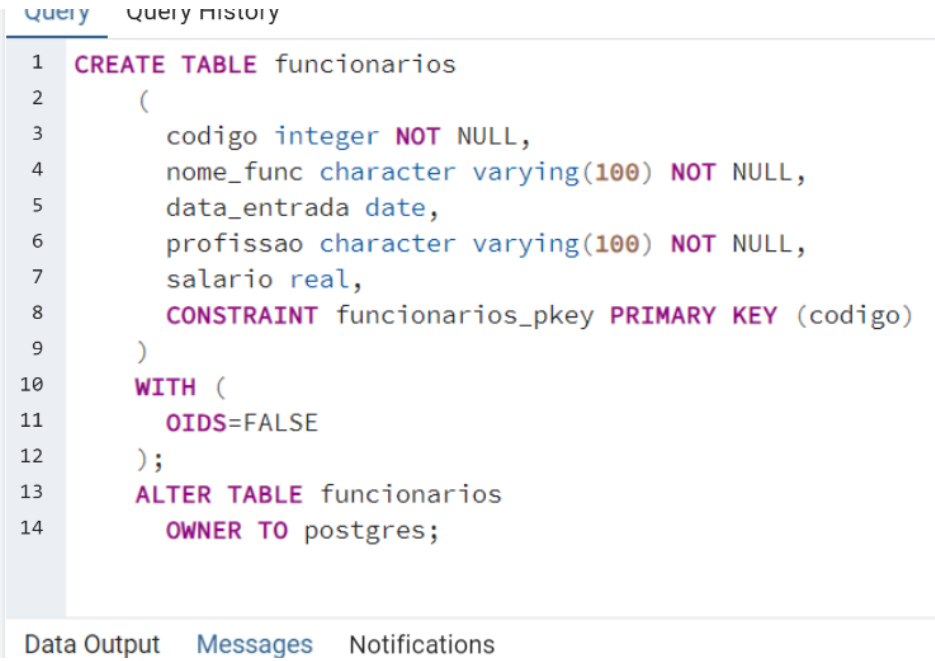
Primeiramente, precisamos entender o que são as Views, que são consideradas pseudo-tables, ou seja, elas são usadas junto a instrução **SELECT** para apresentar subconjuntos de dados presentes em tabelas reais.

Assim, podemos apresentar as colunas e linhas que foram selecionadas da tabela original ou associada.

E como as Views possuem permissões separadas, podemos utilizá-las para restringir mais o acesso aos dados pelos usuários, para que veja apenas o que é necessário.

```
CREATE TABLE funcionarios
    (
      codigo integer NOT NULL,
      nome_func character varying(100) NOT NULL,
      data_entrada date,
      profissao character varying(100) NOT NULL,
      salario real,
      CONSTRAINT funcionarios_pkey PRIMARY KEY (codigo)
    )
    WITH (
      OIDS=FALSE
    );
    ALTER TABLE funcionarios
      OWNER TO postgres;
```

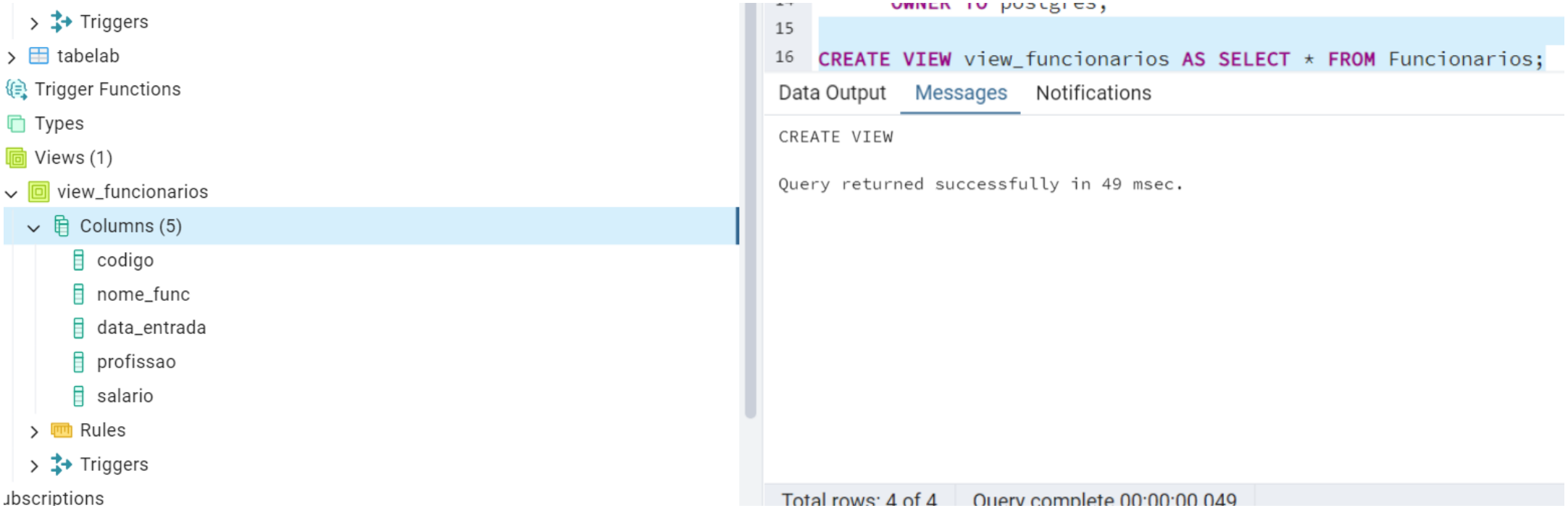




View

- Vamos criar uma view para essa tabela usando a sintaxe a seguir:

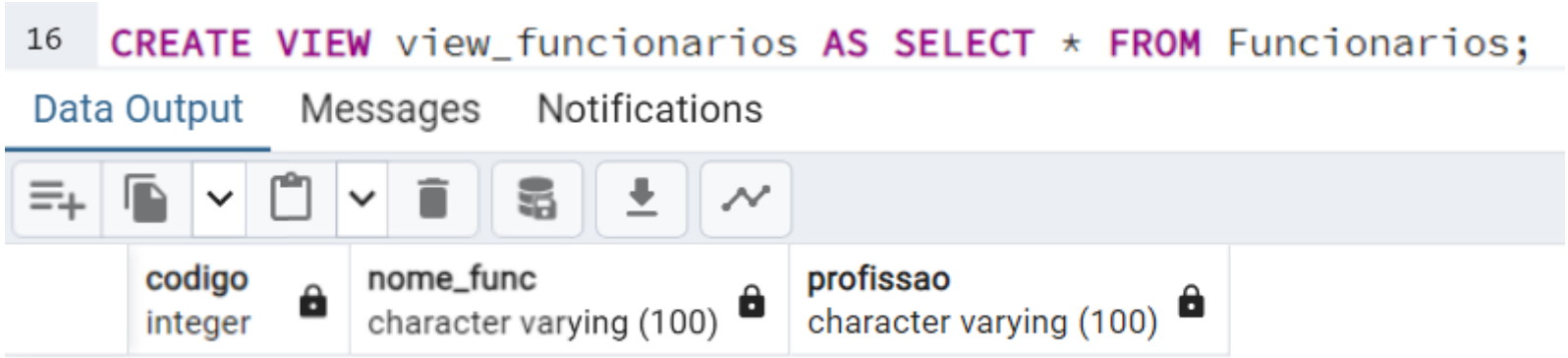
```
CREATE VIEW view_funcionarios AS SELECT * FROM Funcionarios;
```



View

A view_funcionarios contém uma instrução SELECT para receber os dados da tabela. Para que possamos ver as views que criamos, podemos utilizar a seguinte declaração:

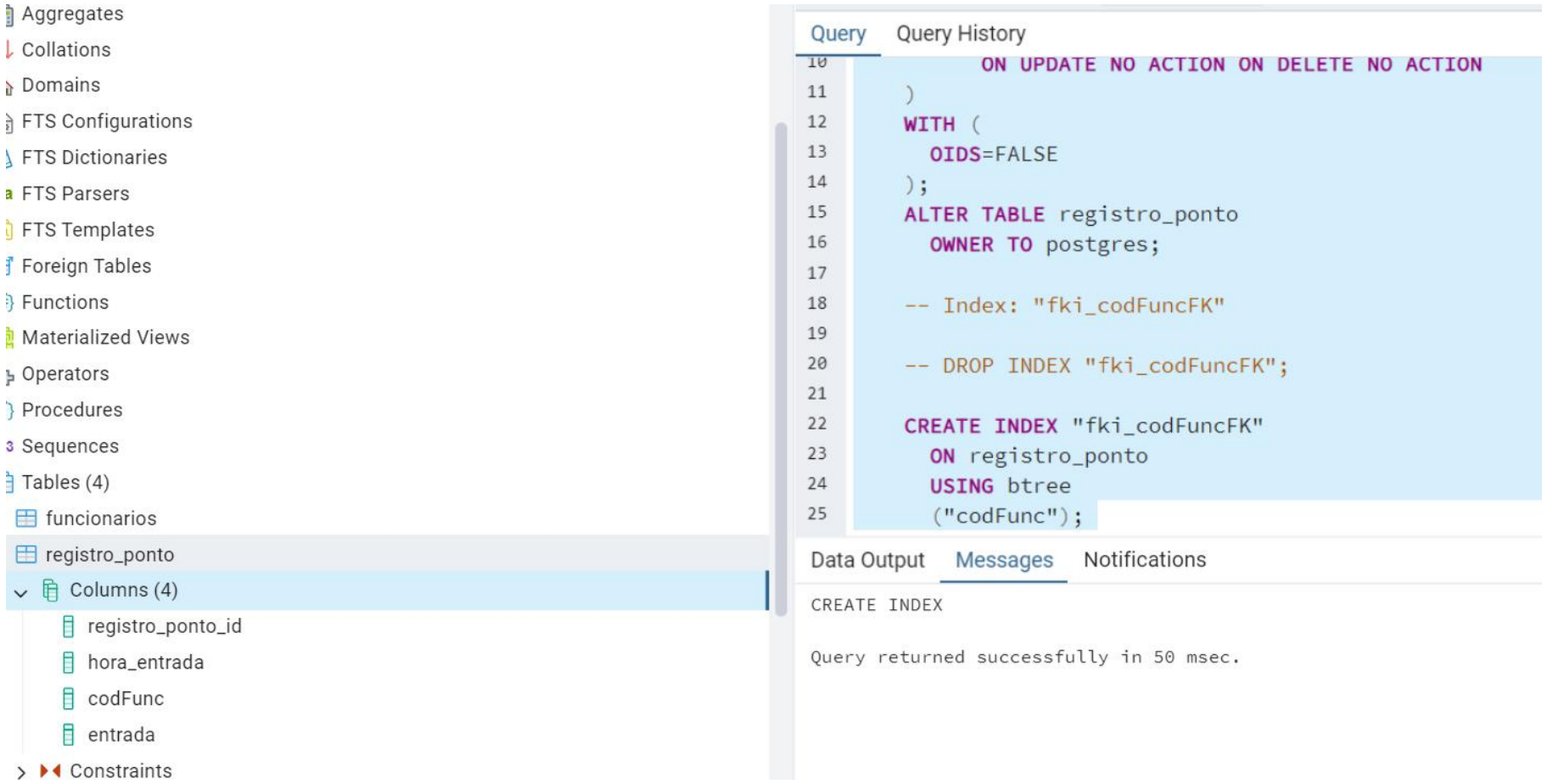
```
SELECT codigo, nome_func, profissao FROM view_funcionarios;
```



Outro exemplo: Registro de ponto

```
CREATE TABLE registro_ponto
   (
    registro_ponto_id integer NOT NULL,
    hora_entrada time without time zone,
    "codFunc" integer NOT NULL,
    entrada date,
    CONSTRAINT registro_ponto_pkey PRIMARY KEY (registro_ponto_id),
    CONSTRAINT "codFuncFK" FOREIGN KEY ("codFunc")
        REFERENCES funcionarios (codigo) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
   )
   WITH (
    OIDS=FALSE
   );
   ALTER TABLE registro_ponto
    OWNER TO postgres;
   -- Index: "fki_codFuncFK"
   -- DROP INDEX "fki_codFuncFK";
   CREATE INDEX "fki_codFuncFK"
    ON registro_ponto
    USING btree
    ("codFunc");
```

Resultado


Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (4)
funcionarios
registro_ponto
Columns (4)
registro_ponto_id
hora_entrada
codFunc
entrada
Constraints
Query
Query History
ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
OIDS=FALSE
);
ALTER TABLE registro_ponto
OWNER TO postgres;
-- Index: "fki_codFuncFK"
-- DROP INDEX "fki_codFuncFK";
CREATE INDEX "fki_codFuncFK"
ON registro_ponto
USING btree
("codFunc");
Data Output
Messages
Notifications
CREATE INDEX
Query returned successfully in 50 msec.

Visão de ponto de funcionário

Query | Query History

```
1  CREATE VIEW View_Ponto_funcionario
2  AS SELECT nome_func, profissao, entrada, hora_entrada
3  FROM funcionarios, registro_ponto
4  WHERE funcionarios.codigo = registro_ponto."codFunc";
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 48 msec.
```

CREATE VIEW View_Ponto_funcionario
AS SELECT nome_func, profissao, entrada, hora_entrada
FROM funcionarios, registro_ponto
WHERE funcionarios.codigo = registro_ponto."codFunc";

```
SELECT * FROM View_Ponto_funcionario;
```

SELECT * FROM View_Ponto_funcionario;

a Output | Messages | Notifications

nome_func	profissao	entrada	hora_entrada
character varying (100)	character varying (100)	date	time without time zone

Junções SQL

O que são junções PostgreSQL?

São usados para recuperar dados de mais de uma tabela.

Com Joins, é possível combinar as instruções SELECT e JOIN em uma única instrução.

Uma condição JOIN é adicionada à instrução e todas as linhas que atendem às condições são retornadas.

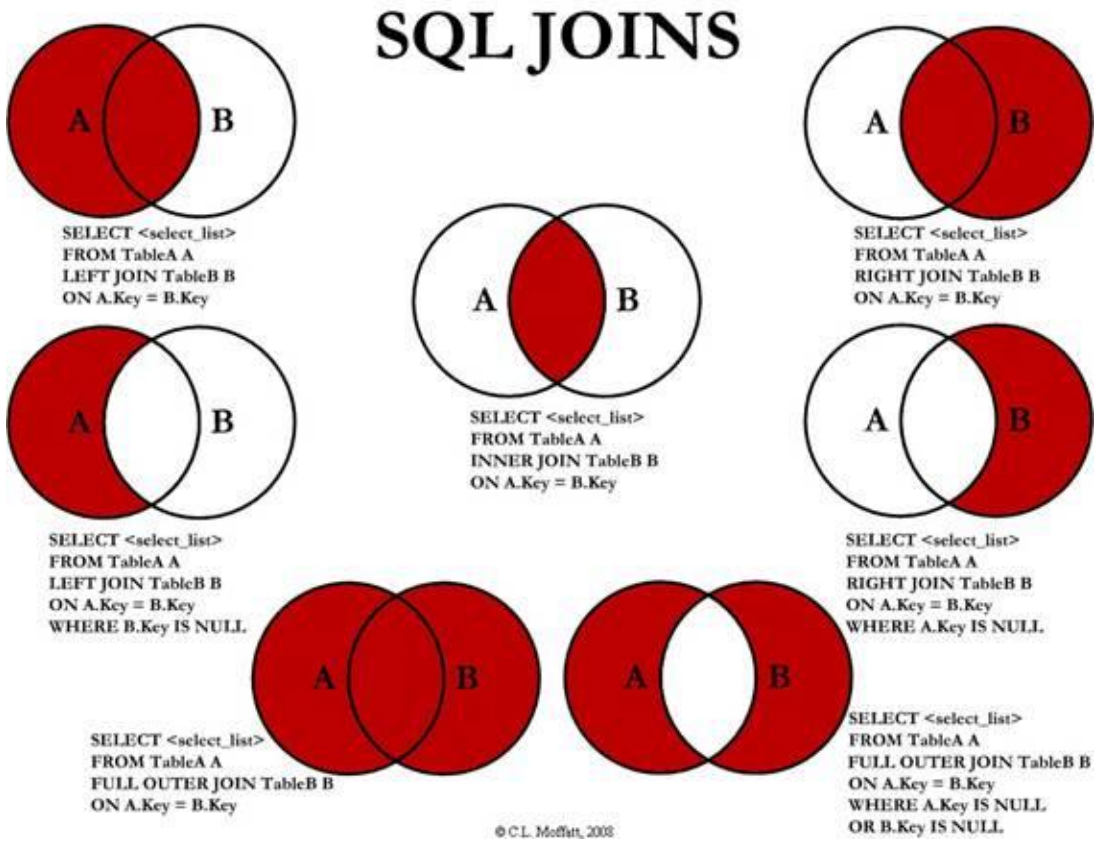
Os valores de diferentes tabelas são combinados com base em colunas comuns.

A coluna comum é principalmente uma chave primária na primeira tabela e uma chave estrangeira na segunda tabela.

Join

A figura ao lado traz uma representação gráfica, baseada na Teoria dos Conjuntos.

Nessa imagem, temos a representação de duas tabelas (A e B) e o resultado esperado por cada tipo de join (a área em vermelho representa os registros retornados pela consulta).



Tipos de Join

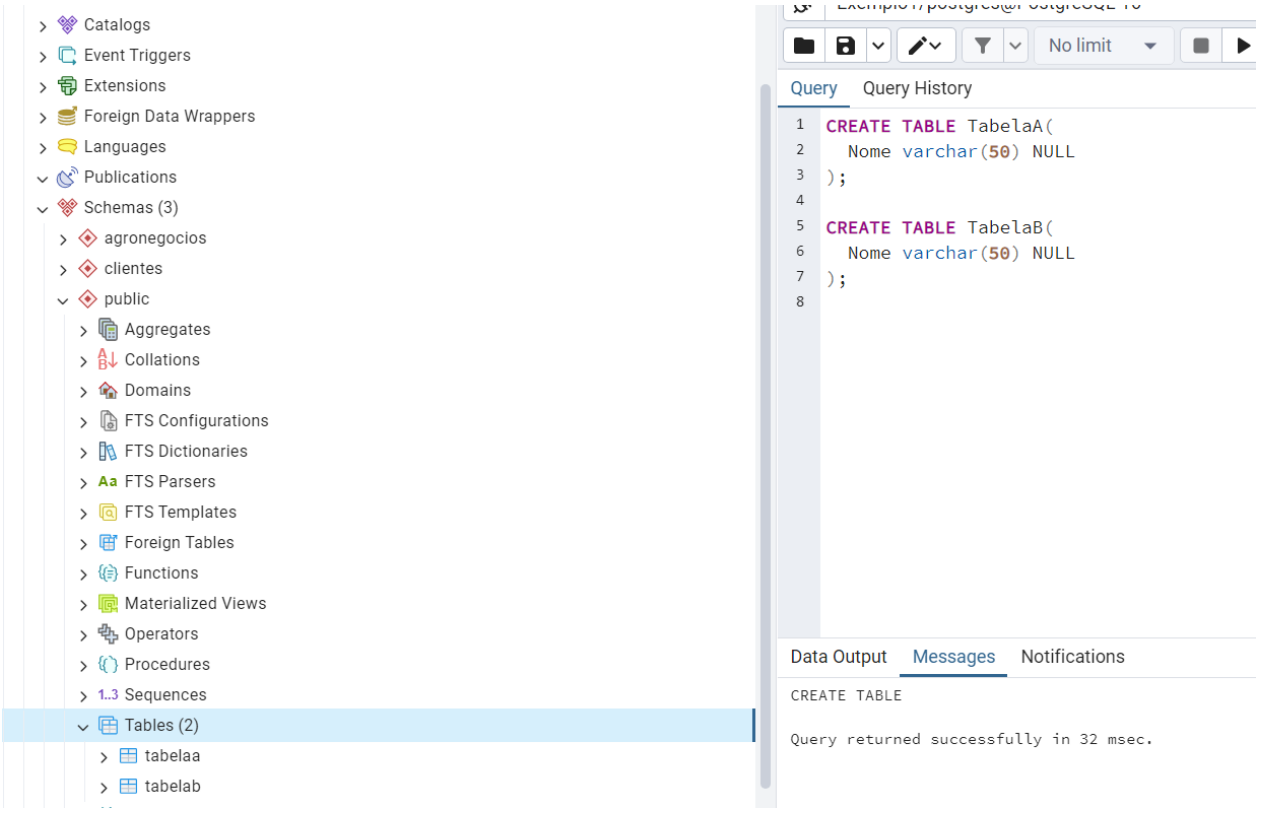
Exemplo

Para demonstrar o funcionamento dos métodos de junção (joins), precisaremos criar duas tabelas entre as quais deve haver algum relacionamento para que possamos "cruzar" os dados.

Como o objetivo aqui não é concentrar em boas práticas, modelagem ou normalização, criaremos apenas duas tabelas contendo uma coluna Nome, que será comum entre elas.

Exemplo

```
CREATE TABLE TabelaA(
  Nome varchar(50) NULL
);
CREATE TABLE TabelaB(
  Nome varchar(50) NULL
);
```



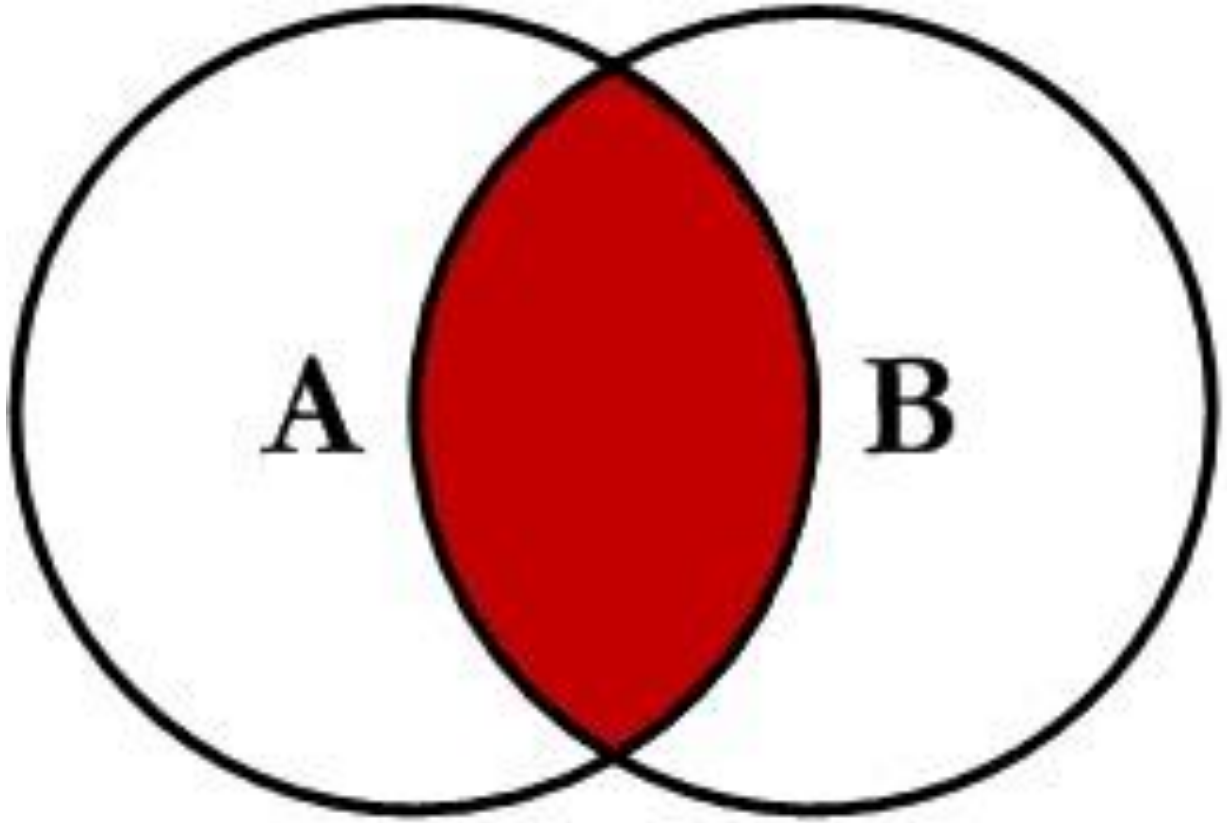
Inserir os valores

```
INSERT INTO TabelaA VALUES('Fernanda');
INSERT INTO TabelaA VALUES('Josefa');
INSERT INTO TabelaA VALUES('Luiz');
INSERT INTO TabelaA VALUES('Fernando');

INSERT INTO TabelaB VALUES('Carlos');
INSERT INTO TabelaB VALUES('Manoel');
INSERT INTO TabelaB VALUES('Luiz');
INSERT INTO TabelaB VALUES('Fernando');
```

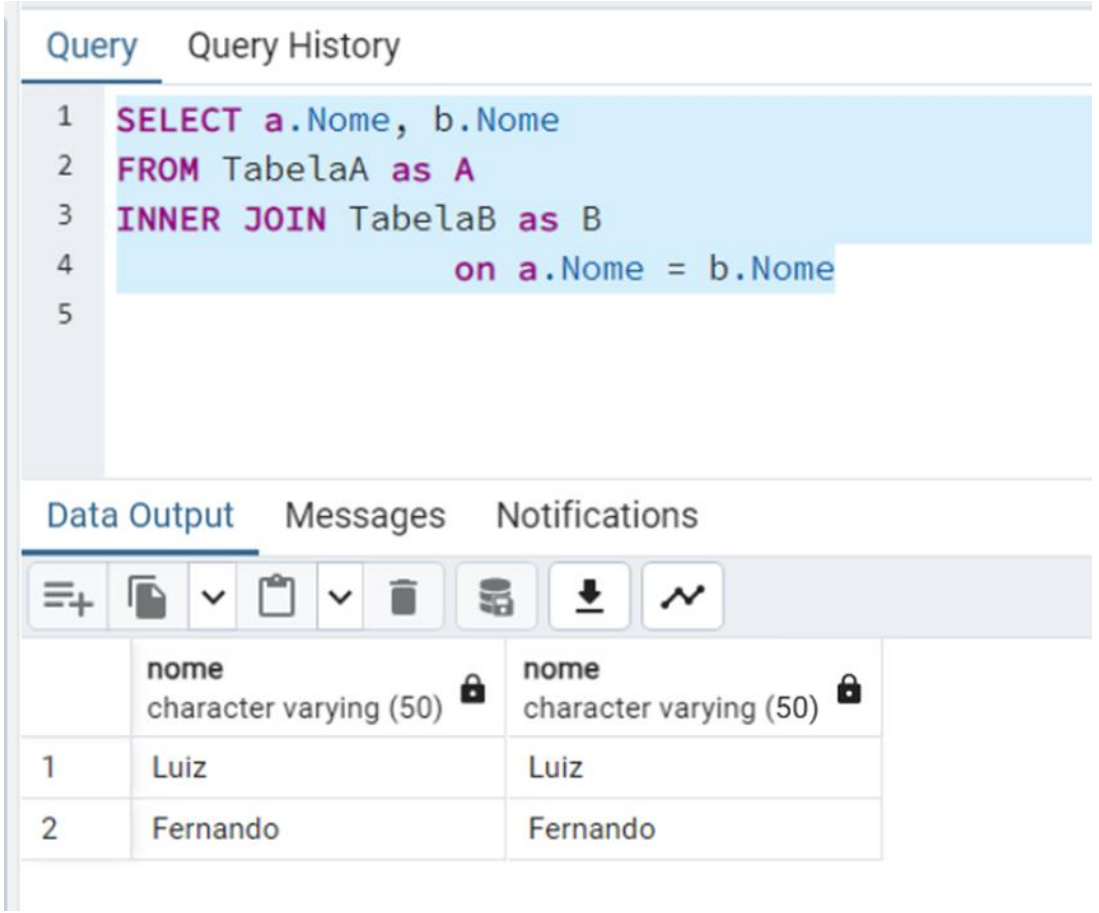
Inner Join

O Inner Join é o método de junção mais conhecido e, como ilustra a figura ao lado, retorna os registros que são comuns às duas tabelas.



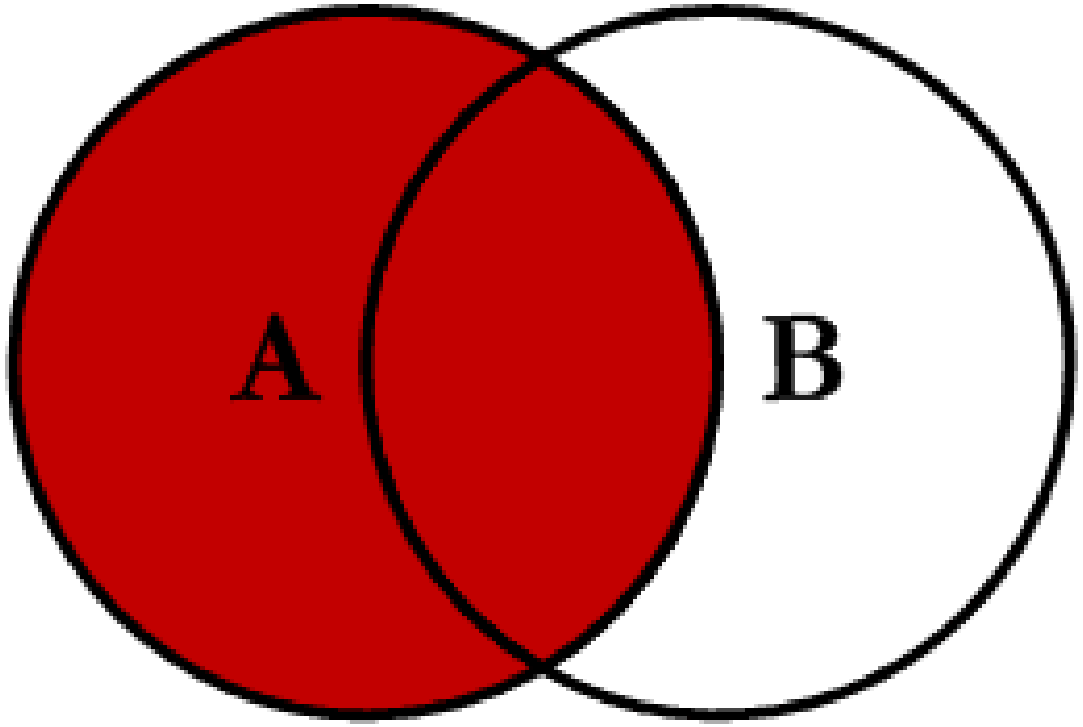
Inner Join

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
INNER JOIN TabelaB as B
                on a.Nome = b.Nome
```



Left Join

O Left Join, cujo funcionamento é ilustrado na figura ao lado, tem como resultado todos os registros que estão na tabela A (mesmo que não estejam na tabela B) e os registros da tabela B que são comuns à tabela A.



Left Join

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
                on a.Nome = b.Nome
```

Query | Query History

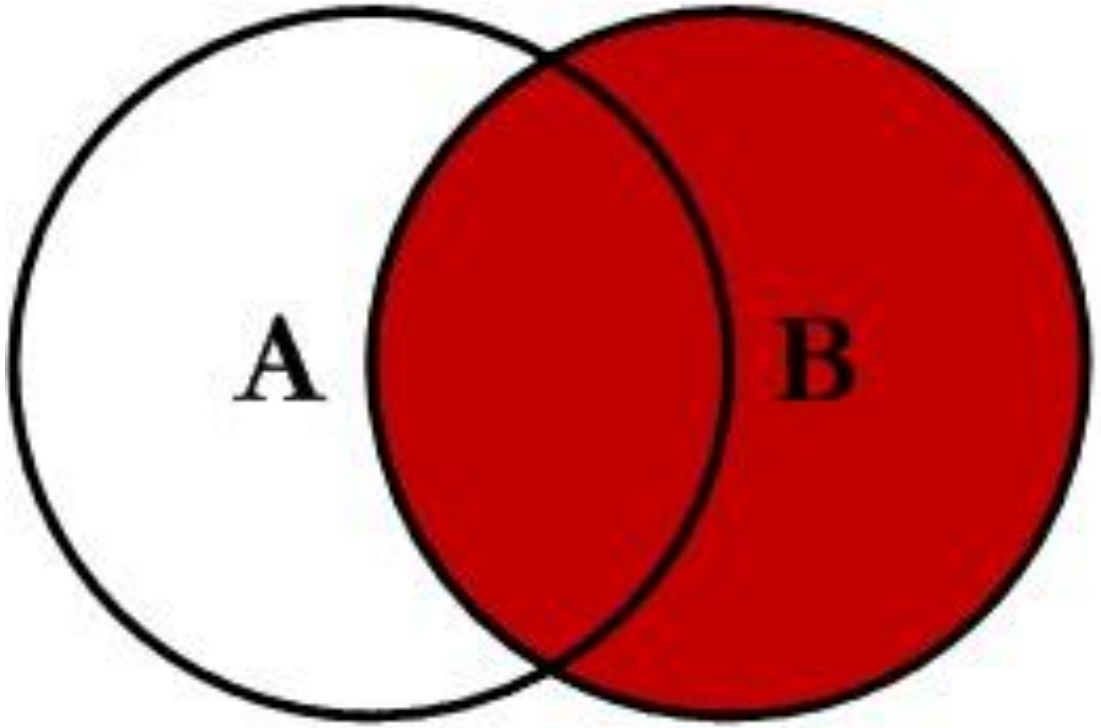
```
1  SELECT a.Nome, b.Nome
2  FROM TabelaA as A
3  LEFT JOIN TabelaB as B
4                  on a.Nome = b.Nome
5
```

Data Output | Messages | Notifications

	nome character varying (50)	nome character varying (50)
1	Fernanda	[null]
2	Josefa	[null]
3	Luiz	Luiz
4	Fernando	Fernando

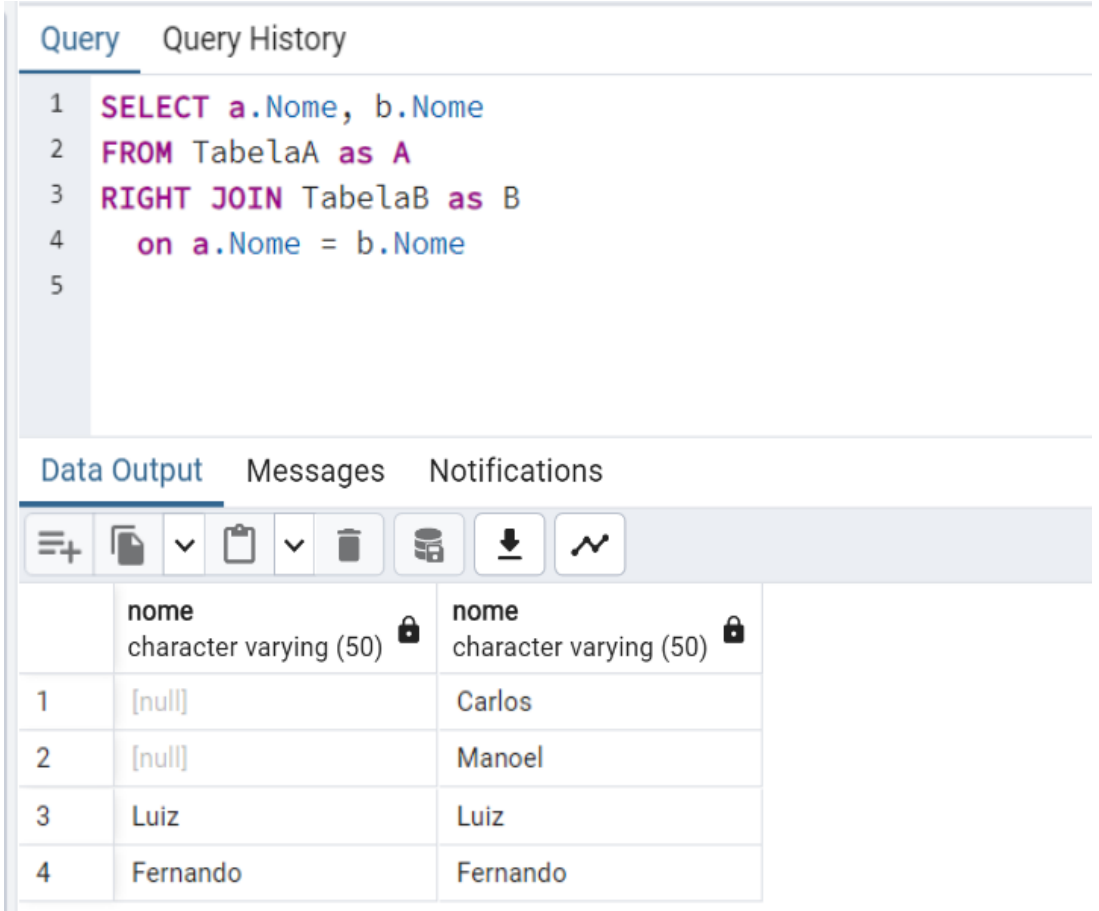
Right Join

Usando o Right Join, conforme mostra a figura ao lado, teremos como resultado todos os registros que estão na tabela B (mesmo que não estejam na tabela A) e os registros da tabela A que são comuns à tabela B.



Right Join

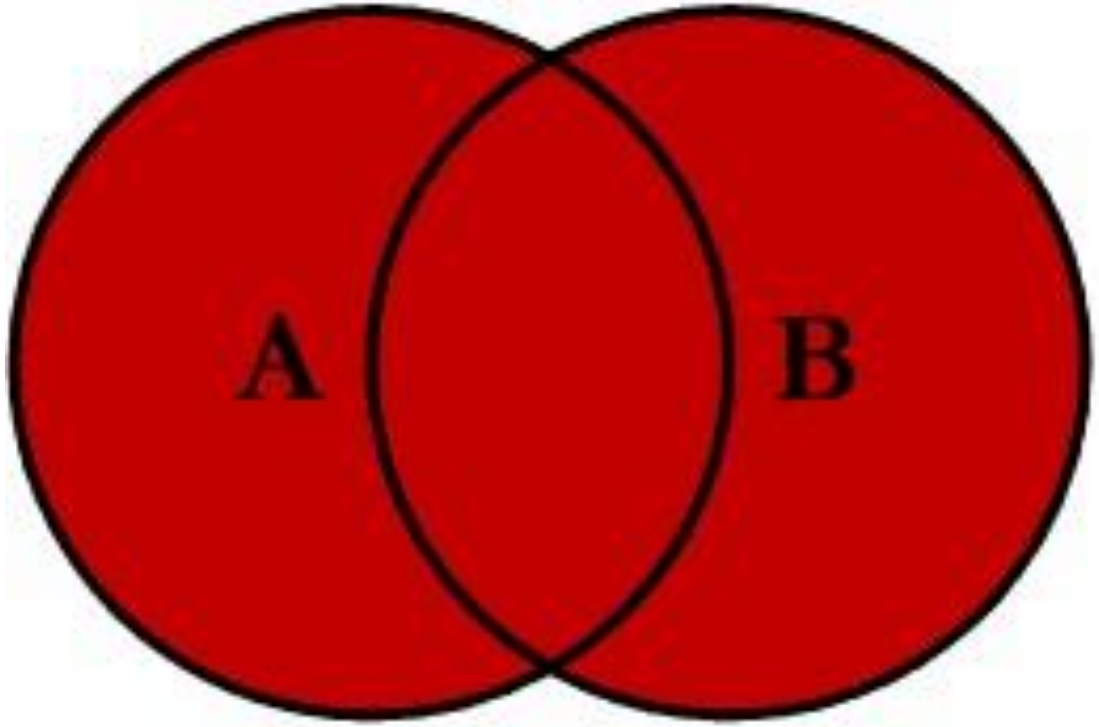
```
SELECT a.Nome, b.Nome
FROM TabelaA as A
RIGHT JOIN TabelaB as B
    on a.Nome = b.Nome
```



	nome character varying (50)	nome character varying (50)
1	[null]	Carlos
2	[null]	Manoel
3	Luiz	Luiz
4	Fernando	Fernando

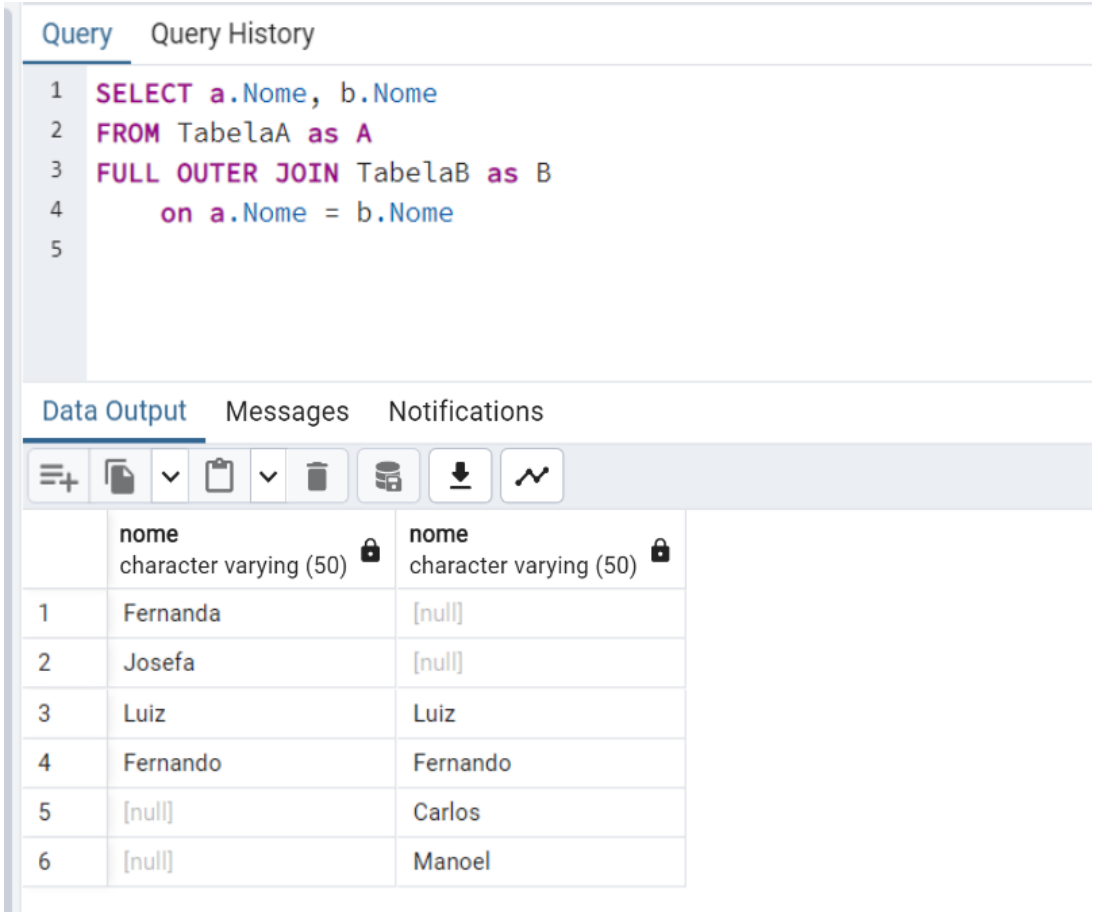
Outer Join

O Outer Join (também conhecido por Full Outer Join ou Full Join), conforme mostra a figura ao lado, tem como resultado todos os registros que estão na tabela A e todos os registros da tabela B.



Outer Join

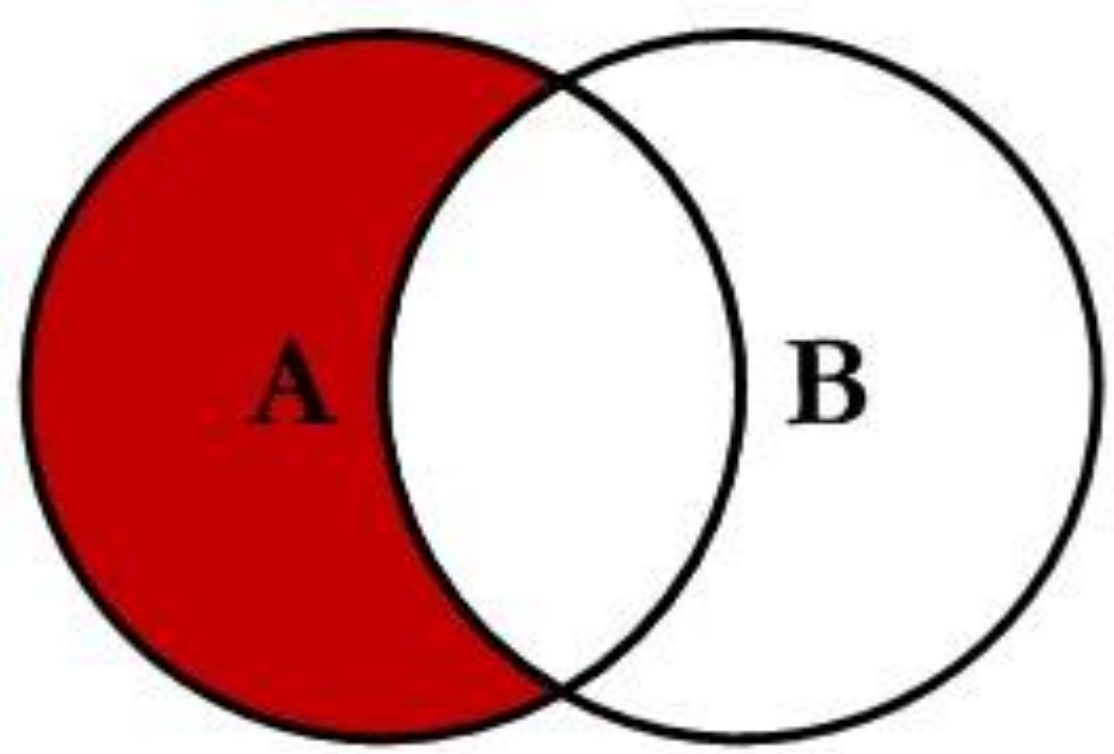
```
SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL OUTER JOIN TabelaB as B
    on a.Nome = b.Nome
```



	nome character varying (50)	nome character varying (50)
1	Fernanda	[null]
2	Josefa	[null]
3	Luiz	Luiz
4	Fernando	Fernando
5	[null]	Carlos
6	[null]	Manoel

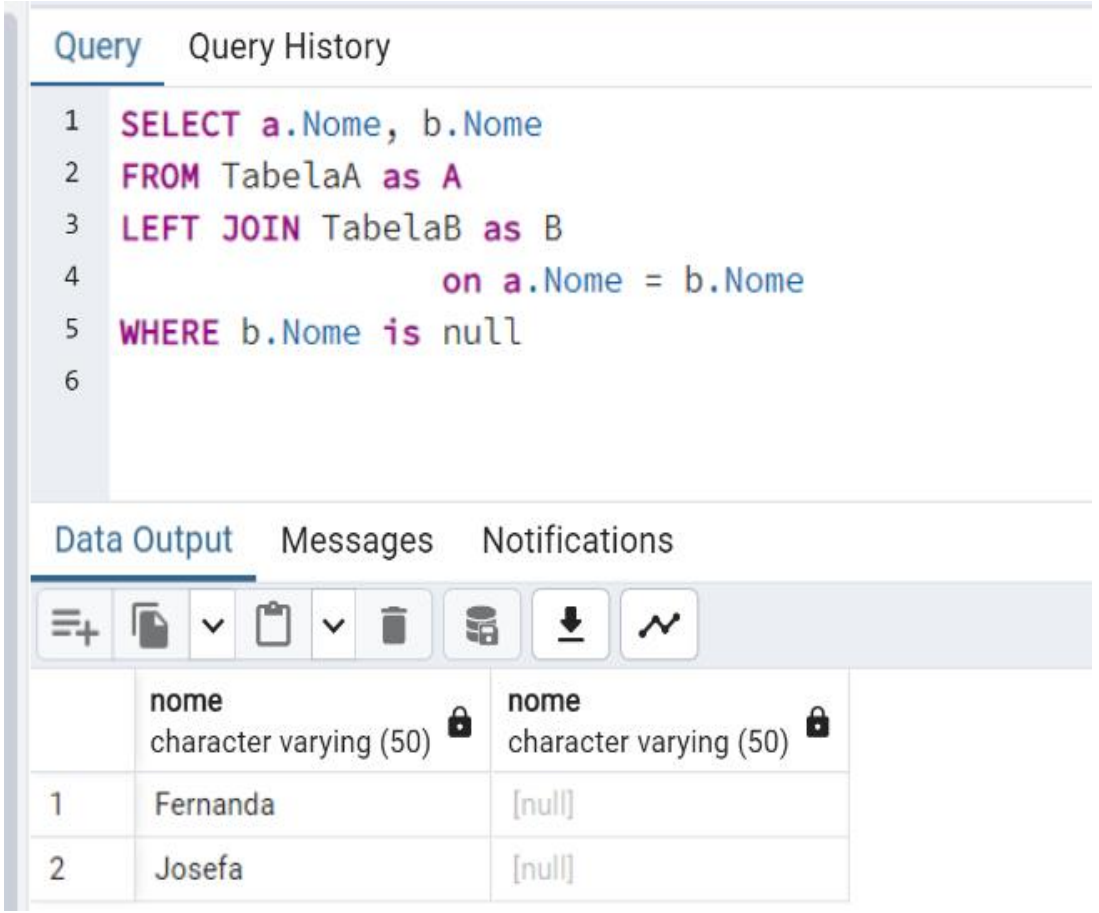
Left Excluding Join

Na figura ao lado temos a representação gráfica do Left Excluding Join, que retorna como resultado todos os registros que estão na tabela A e que não estejam na tabela B.



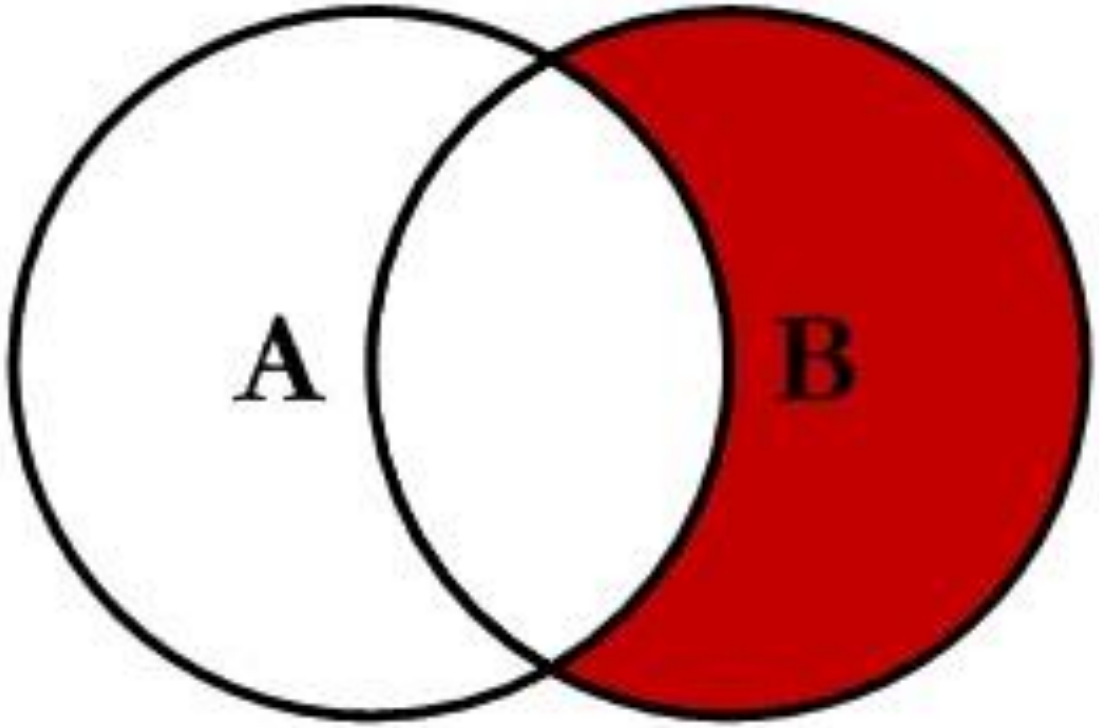
Left Excluding Join

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE b.Nome is null
```



Right Excluding Join

O Right Excluding Join, como ilustra a figura ao lado, retorna como resultado todos os registros que estão na tabela B e que não estejam na tabela A. ,



Left Excluding Join

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE b.Nome is null
```

Query | Query History

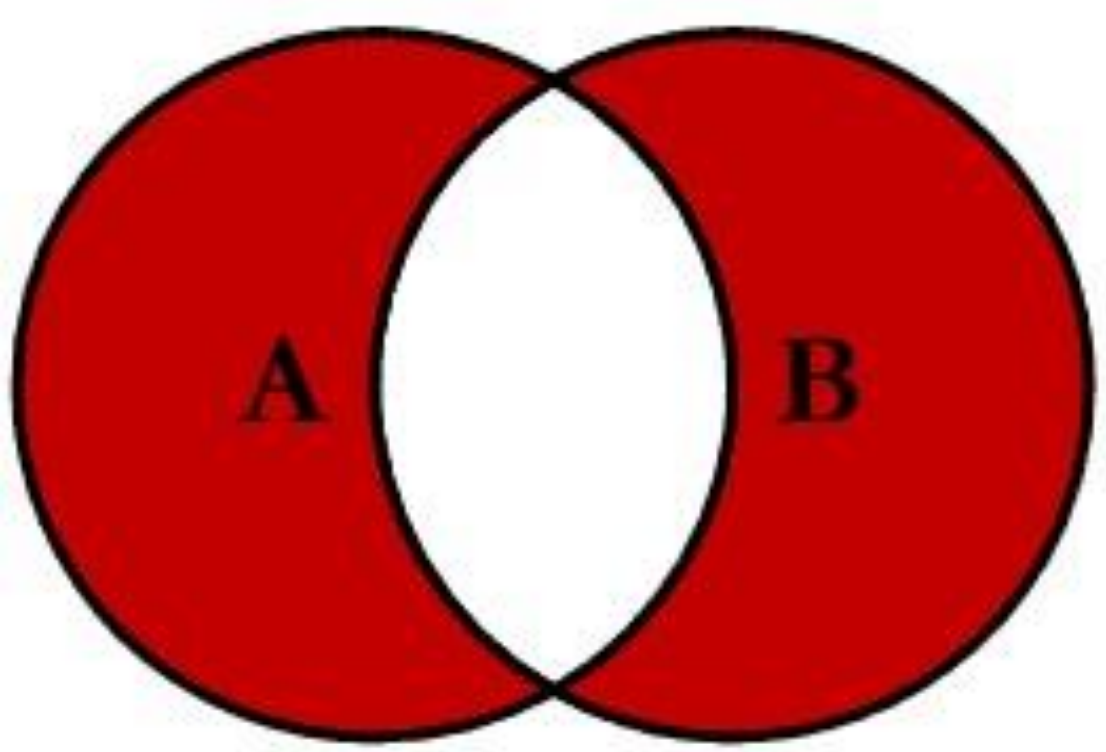
```
1  SELECT a.Nome, b.Nome
2  FROM TabelaA as A
3  RIGHT JOIN TabelaB as B
4      on a.Nome = b.Nome
5  WHERE a.Nome is null
```

Data Output | Messages | Notifications

	nome character varying (50)	nome character varying (50)
1	[null]	Carlos
2	[null]	Manoel

Outer Excluding Join

Usando o Outer Excluding Join, conforme mostra a figura ao ladoteremos como resultado todos os registros que estão na tabela B, mas que não estejam na tabela A, e todos os registros que estão na tabela A, mas que não estejam na tabela B.



```
SE
  SELECT a.Nome, b.Nome FROM TabelaA as A FULL OUTER JOIN TabelaB as B on a.Nome = b.Nome WHERE a.Nome is null or b.Nome is null
```

Inner Join

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
INNER JOIN TabelaB as B
        on a.Nome = b.Nome
```

Query | Query History

```
1  SELECT a.Nome, b.Nome
2  FROM TabelaA as A
3  FULL OUTER JOIN TabelaB as B
4      on a.Nome = b.Nome
5  WHERE a.Nome is null or b.Nome is null
```

Data Output | Messages | Notifications

	nome character varying (50)	nome character varying (50)
1	Fernanda	[null]
2	Josefa	[null]
3	[null]	Carlos
4	[null]	Manoel