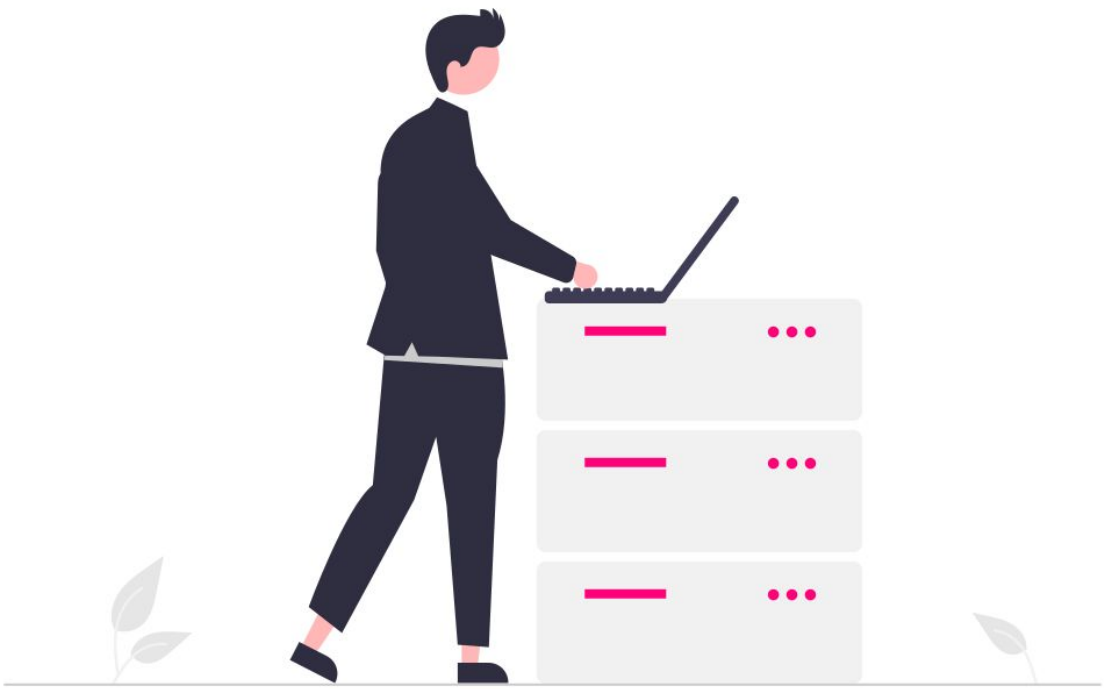
SQL



Relacionamento entre Tabelas

Relacionamento entre Tabelas

Modelo Relacional

O conceito de **banco de dados relacional** surge com a proposta de **facilitar a organização dos dados** trazendo algumas características importantes como: **integridade, segurança e escalabilidade**.

- **Integridade**: com um modelo relacional, os dados são armazenados em **tabelas conectadas por chaves primárias e secundárias**, o que garante que haja uma consistência de dados em toda a base.
- **Segurança**: banco de dados relacional permite a implementação de mecanismos de **segurança** para **proteger os dados** contra usuários não autorizados e perda de informações.
- **Escalabilidade**: o modelo relacional também permite a **expansão do banco de dados** de forma **fácil e eficiente**, garantindo um **alto desempenho** mesmo com **grandes volumes de dados**.

Sendo assim, o modelo relacional permite **implementar** uma **estrutura bem definida**, com **tabelas**, **colunas** e **relacionamentos** entre elas, o que torna mais **fácil e eficiente** armazenar e recuperar informações.

Relacionamento entre Tabelas

Chaves primárias e estrangeiras

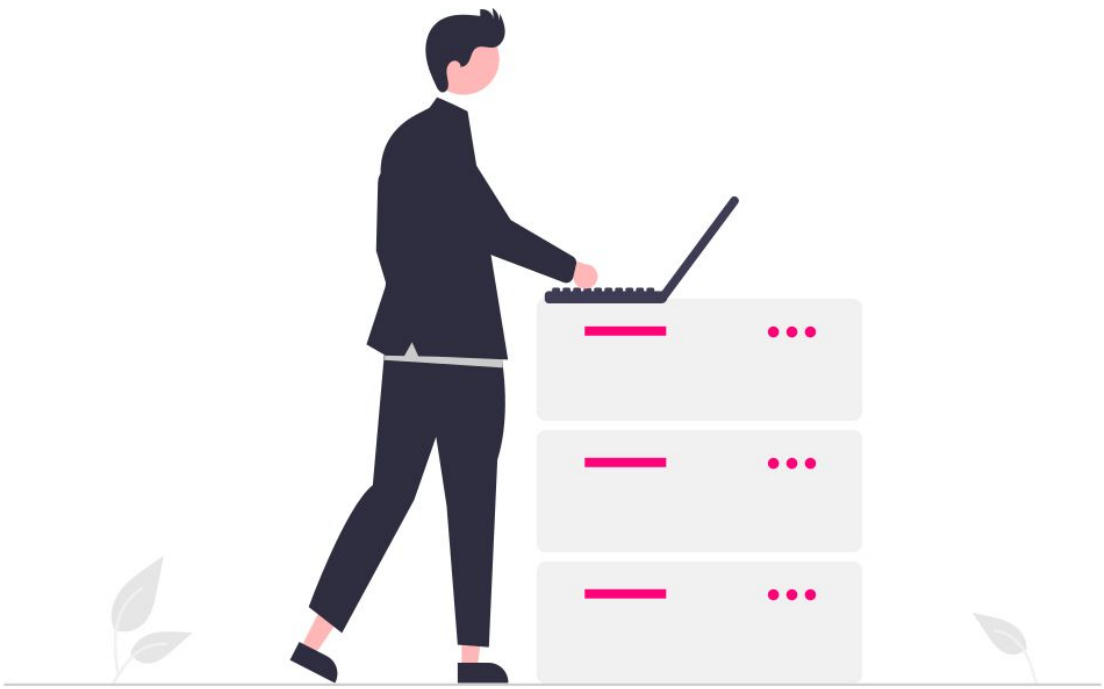
Para que um **relacionamento entre duas tabelas** aconteça é necessário que exista algum tipo de **ligação** entre elas. Essa ligação é feita através das **chaves primárias e estrangeiras**.

As **chaves primárias** são colunas ou conjuntos de colunas que **identificam unicamente cada registro em uma tabela**. Elas garantem que **não haja registros duplicados** e são importantes para garantir a integridade dos dados. **Uma chave primária não pode ser nula**.

As **chaves estrangeiras** são colunas em uma tabela que **se referem a uma chave primária em outra tabela**. Elas criam uma **relação entre duas tabelas** e são usadas para garantir a integridade referencial dos dados. As chaves estrangeiras garantem que **não haja valores nulos ou inconsistentes** em uma tabela relacionada a outra.

Existem **três tipos de relacionamentos entre tabelas**, os quais vamos explorar a seguir.

SQL



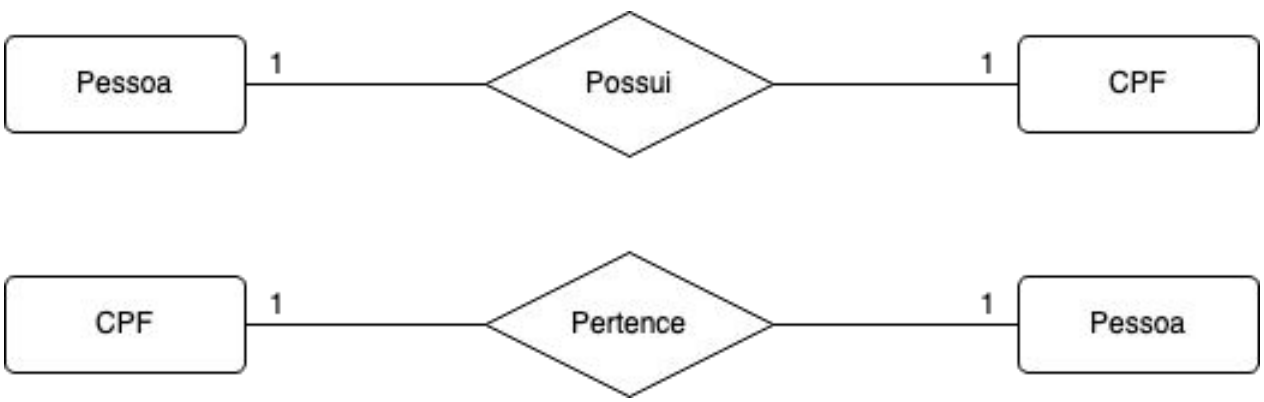
Cardinalidade das Relações

Cardinalidade das Relações

1:1 (um para um)

O **relacionamento de um para um** (**1:1**) acontece quando **um registro de uma tabela** se **relaciona com um e apenas um registro em outra tabela** e vice-versa

Observe a imagem a seguir:



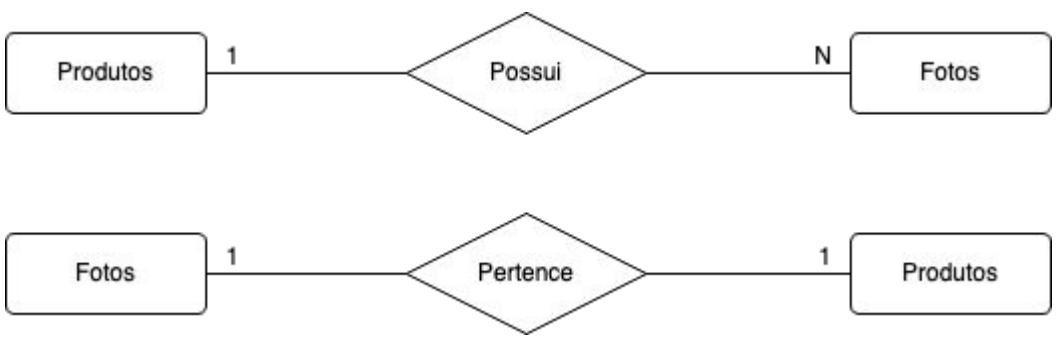
No diagrama acima temos um **relacionamento de um para um**, onde **um registro da tabela pessoa** possui ligação com **apenas um registro da tabela CPF** e, ao contrário, **um registro da tabela CPF** pertence **apenas a um registro da tabela pessoa**.

Cardinalidade das Relações

1:N (um para n ou um para muitos)

O **relacionamento de um para muitos** (**1:N**) acontece quando **um registro de uma tabela** se **relaciona com zero, um ou vários registros em outra tabela** e **cada registro** relacionado na tabela que armazena a **chave estrangeira** pertence a **apenas um registro na tabela com a chave primária**.

Observe a imagem a seguir:



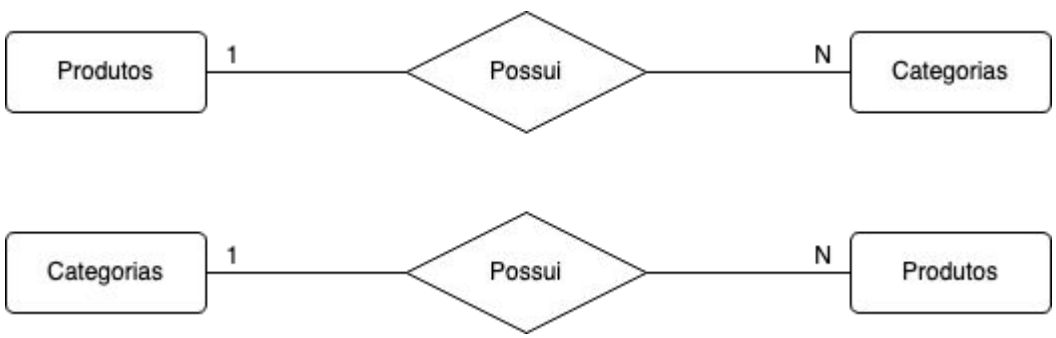
No diagrama acima temos um **relacionamento de um para muitos**, onde **um registro da tabela produtos** possui ligação com **N's registros da tabela fotos**. "**N's**" Significa que pode não ter nenhuma, uma ou várias. Ao contrário, **um registro da tabela fotos** pertence **apenas a um registro da tabela produtos**.

Cardinalidade das Relações

N:N (n para n ou muitos para muitos)

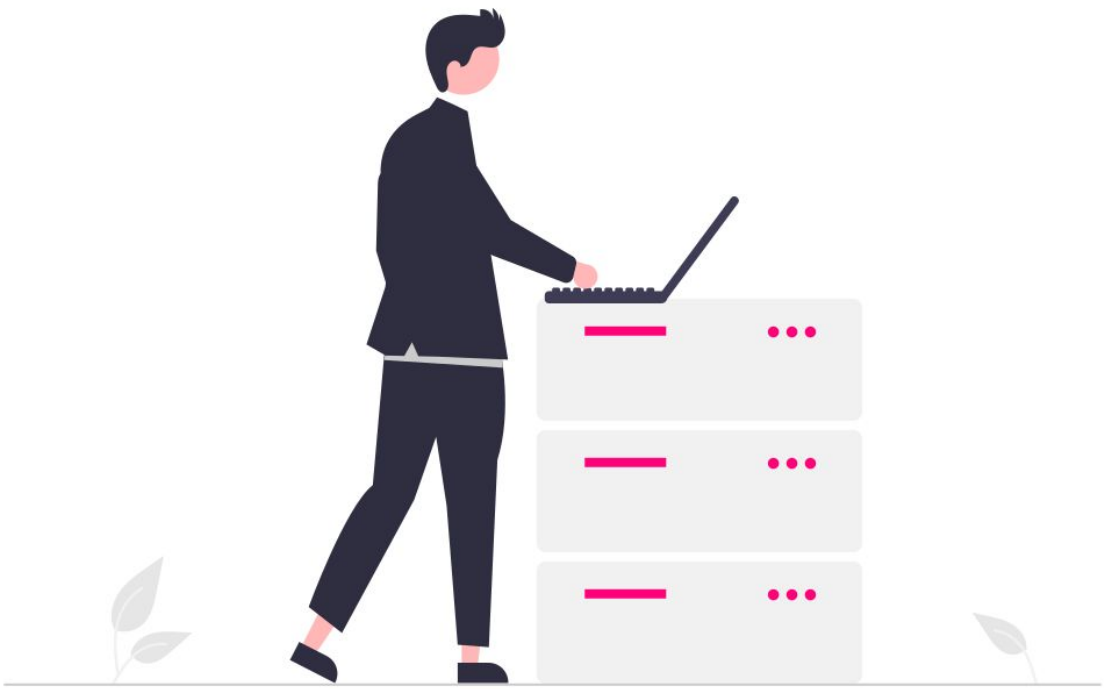
O **relacionamento de muitos para muitos** (**N:N**) acontece quando **um registro de uma tabela** se **relaciona com zero, um ou vários registros em outra tabela**, mas **cada registro** relacionado **nesta outra tabela** também se **relaciona com zero, um ou vários registros da outra tabela**. Uma **característica importante** do relacionamento de N:N é que **precisa existir uma tabela de ligação** entre as tabelas relacionadas.

Observe a imagem a seguir:



No diagrama acima temos um **relacionamento de muitos para muitos**, onde **um registro da tabela produtos** possui ligação com **N's registros da tabela categorias**. Ao contrário, **um registro da tabela categorias** também possui ligação com **N's registros da tabela produtos**.

SQL



Combinando Tabelas

Combinando Tabelas

JOIN

JOIN é uma **declaração SQL** que **combina linhas de duas ou mais tabelas** com base em uma condição de correspondência. Existem **diferentes tipos de Joins**, entre eles, **INNER JOIN**, **LEFT JOIN**, e **RIGHT JOIN**, cada um com sua própria função e resultado.

Observe a imagem a seguir:

Usando o **JOIN** para **combinar as duas tabelas**, podemos obter os seguintes resultados em uma consulta:

P.ID	P.NAME	P.EMAIL	A.ID	A.ADDRESS
1	Hudson Borer	borer-hudson@yahoo.com	23	9611-9809 West Rosedale Road
1	Hudson Borer	borer-hudson@yahoo.com	47	101 4th Street
1	Hudson Borer	borer-hudson@yahoo.com	49	1243 West Whitney Street
1	Hudson Borer	borer-hudson@yahoo.com	56	495 Juniper Road

Combinando Tabelas

ON

A **condição de correspondência** usada para combinar tabelas é a cláusula **ON** e ela é **obrigatória** para que a junção entre tabelas aconteça. A correspondência **deve atender a condição informada**, como no exemplo a seguir:

```
select ... on colunaA.tabela1 = colunaA.tabela2;
```

A **cláusula ON** pode parecer com a **WHERE**, mas **ambas são distintas**. A cláusula **WHERE** é usada para **filtrar registros** e a **ON** é **exclusivamente usada em junções de tabelas**.

Observe a imagem a seguir:

A **condição de correspondência** para que as duas tabelas acima pudessem ser combinadas seria:

```
select ... on People.id = Address.people_id;
```

Combinando Tabelas

Construção da query com JOIN

Partindo da imagem a seguir:

Para obter a **combinação entre as duas tabelas acima**, podemos usar a seguinte query:

```sql
select people.*, address.*
from people
join address
on people.id = address.people_id
where people.id = 1;
```

O resultado seria algo do tipo:

ID	NAME	EMAIL	ID	PEOPLE_ID	ADDRESS
1	Hudson Borer	borer-hudson@yahoo.com	23	1	9611-9809 West Rosedale Road
1	Hudson Borer	borer-hudson@yahoo.com	47	1	101 4th Street
1	Hudson Borer	borer-hudson@yahoo.com	49	1	1243 West Whitney Street

Apesar de conseguir fazer a junção, podemos ter **alguns problemas**. Observe que a query acima retornou **dois campos "ID"** (o da tabela people e o da tabela address). É possível **personalizar** a query **selecionando colunas específicas** e **criar apelidos** para o nome **das tabelas**.

Combinando Tabelas

Construção da query com JOIN

Para **evitar os problemas** como, **nome de colunas duplicados**, a query abaixo **seleciona apenas as colunas específicas**, que fazem sentido para o resultado esperado e **cria apelidos, tanto para nome de colunas quanto para os nomes das tabelas**. Observe:

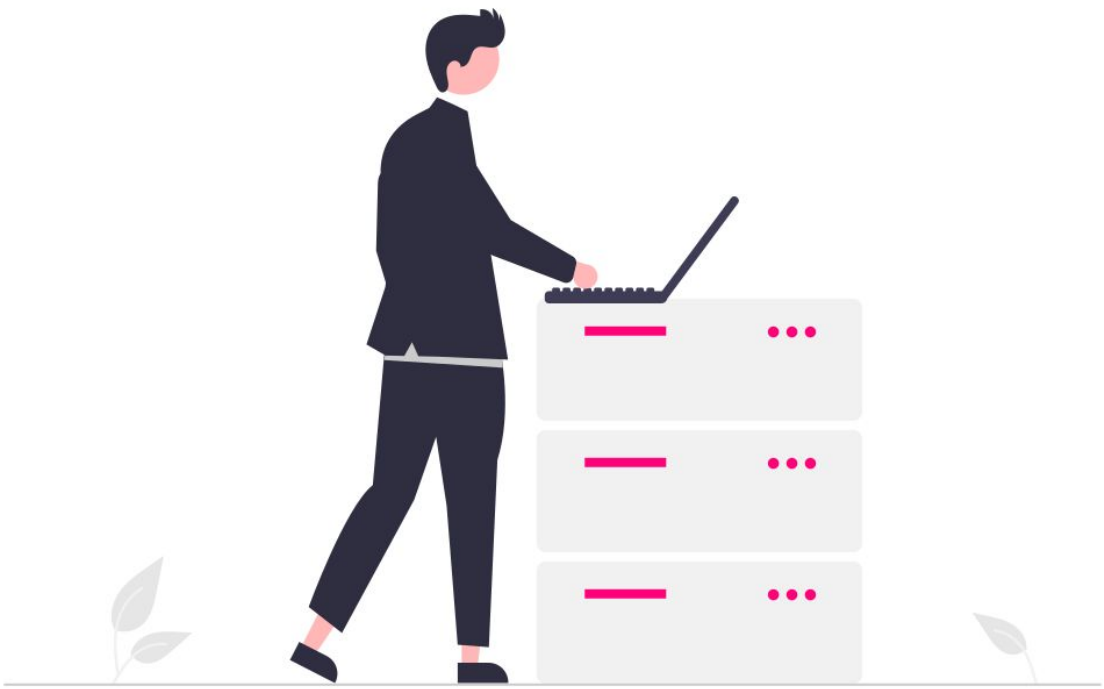
```
select p.id, p.name, p.email, a.id as address_id, a.address
from people as p
join address as a
on p.id = a.people_id
where p.id = 1;
```

Agora o resultado seria algo do tipo:

ID	NAME	EMAIL	ADDRESS_ID	ADDRESS
1	Hudson Borer	borer-hudson@yahoo.com	23	9611-9809 West Rosedale Road
1	Hudson Borer	borer-hudson@yahoo.com	47	101 4th Street
1	Hudson Borer	borer-hudson@yahoo.com	49	1243 West Whitney Street

Aliases SQL (AS) pode ser usado para **criar apelidos** e tornar a query **mais legível**. Conforme o exemplo anterior, podemos observar que esses apelidos podem ser criados para os **nomes de colunas** e também **para nome de tabelas**.

SQL



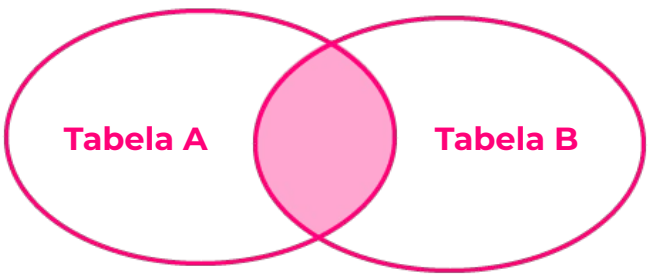
Tipos de JOIN's

Tipos de JOIN's

INNER JOIN

O **INNER JOIN**, que também pode ser escrito somente **JOIN**, **retorna apenas as linhas que têm valores correspondentes** em **ambas as tabelas**. Isso significa que apenas as linhas onde as **colunas das duas tabelas são iguais** serão retornadas.

Se algum registro de uma das duas tabelas não possuir registros relacionados na outra tabela, o registro não é retornado.



Exemplo:

```sql
select p.id, p.name, p.email, a.id as address_id, a.address
from people as p
inner join address as a on p.id = a.people_id
where p.id = 1;
```

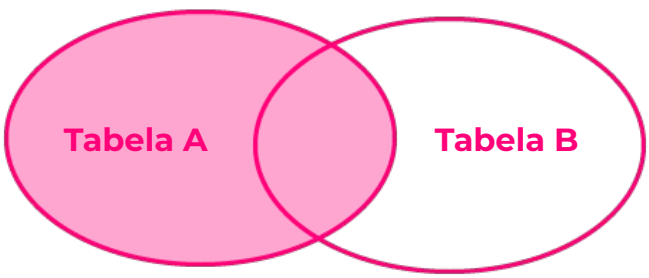
O resultado seria algo do tipo:

ID	NAME	EMAIL	ADDRESS_ID	ADDRESS
1	Hudson Borer	borer-hudson@yahoo.com	23	9611-9809 West Rosedale Road
1	Hudson Borer	borer-hudson@yahoo.com	47	101 4th Street
1	Hudson Borer	borer-hudson@yahoo.com	49	1243 West Whitney Street

Tipos de JOIN's

LEFT JOIN

O **LEFT JOIN** retorna todas as linhas da **tabela à esquerda** (**primeira tabela na cláusula JOIN**) e as linhas correspondentes da **tabela à direita** (**segunda tabela na cláusula JOIN**). Se **não houver correspondência** na tabela à direita, o valor **NULL** será retornado.



Exemplo:

```sql
select p.id, p.name, p.email, a.id as address_id, a.address
from people as p
left join address as a on p.id = a.people_id
where p.id = 1;
```

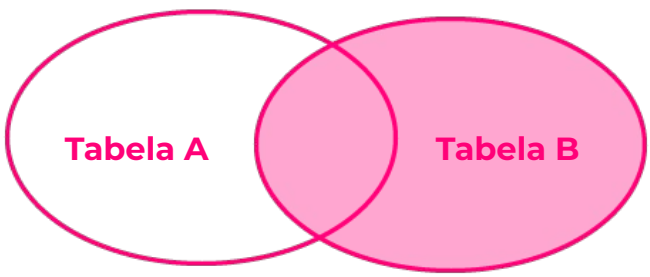
O resultado seria algo do tipo:

ID	NAME	EMAIL	ADDRESS_ID	ADDRESS
1	Hudson Borer	borer-hudson@yahoo.com	23	9611-9809 West Rosedale Road
1	Hudson Borer	borer-hudson@yahoo.com	NULL	NULL
1	Hudson Borer	borer-hudson@yahoo.com	49	1243 West Whitney Street

Tipos de JOIN's

RIGHT JOIN

O **RIGHT JOIN** é semelhante ao **LEFT JOIN**, mas retorna todas as linhas da **tabela à direita** e as linhas correspondentes da **tabela à esquerda**. Se **não houver correspondência** na tabela à esquerda, o valor **NULL** será retornado.



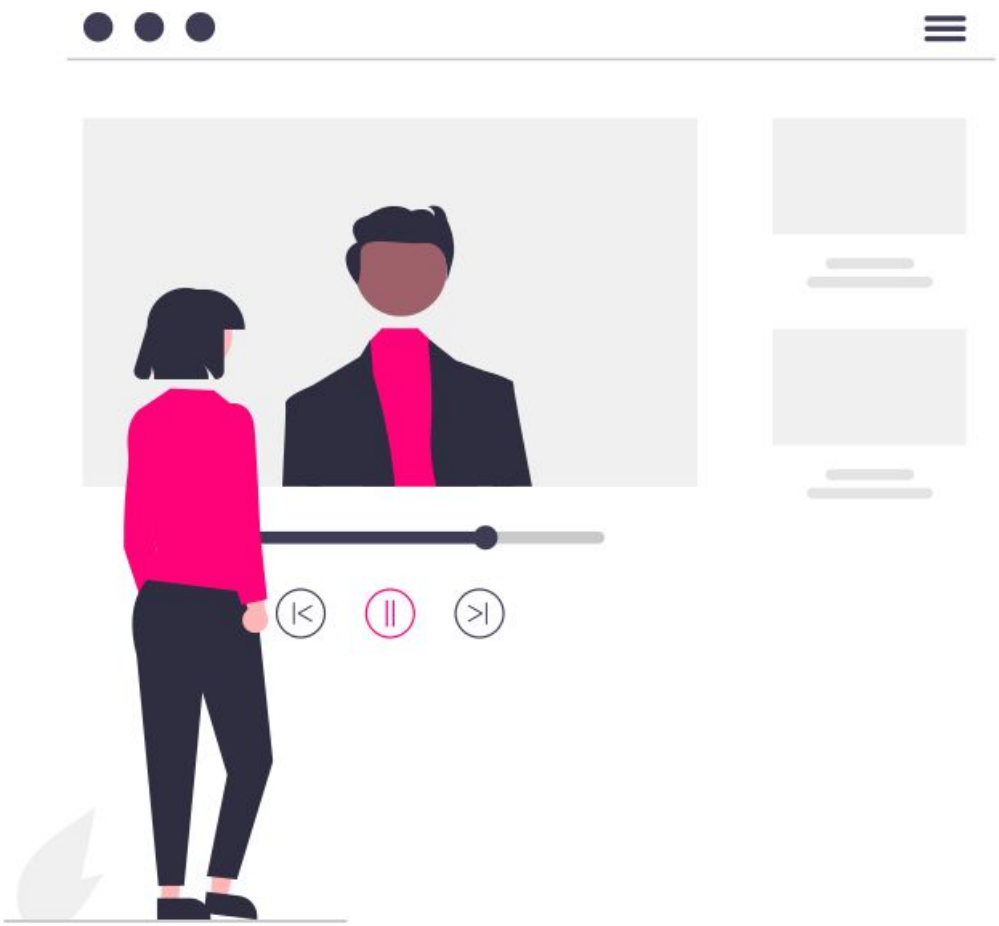
Exemplo:

```sql
select p.id, p.name, p.email, a.id as address_id, a.address
from address as a
right join people as p on p.id = a.people_id
where p.id = 1;
```

O resultado seria algo do tipo:

ID	NAME	EMAIL	ADDRESS_ID	ADDRESS
1	Hudson Borer	borer-hudson@yahoo.com	23	9611-9809 West Rosedale Road
1	Hudson Borer	borer-hudson@yahoo.com	NULL	NULL
1	Hudson Borer	borer-hudson@yahoo.com	49	1243 West Whitney Street

Guido Cerqueira
Professor Cubos Academy



cubos //academy//

Guido Cerqueira
Professor Cubos Academy

cubos
// academy //

www.cubos.academy