



# SQL

DEVinHouse

Parcerias para desenvolver a sua carreira

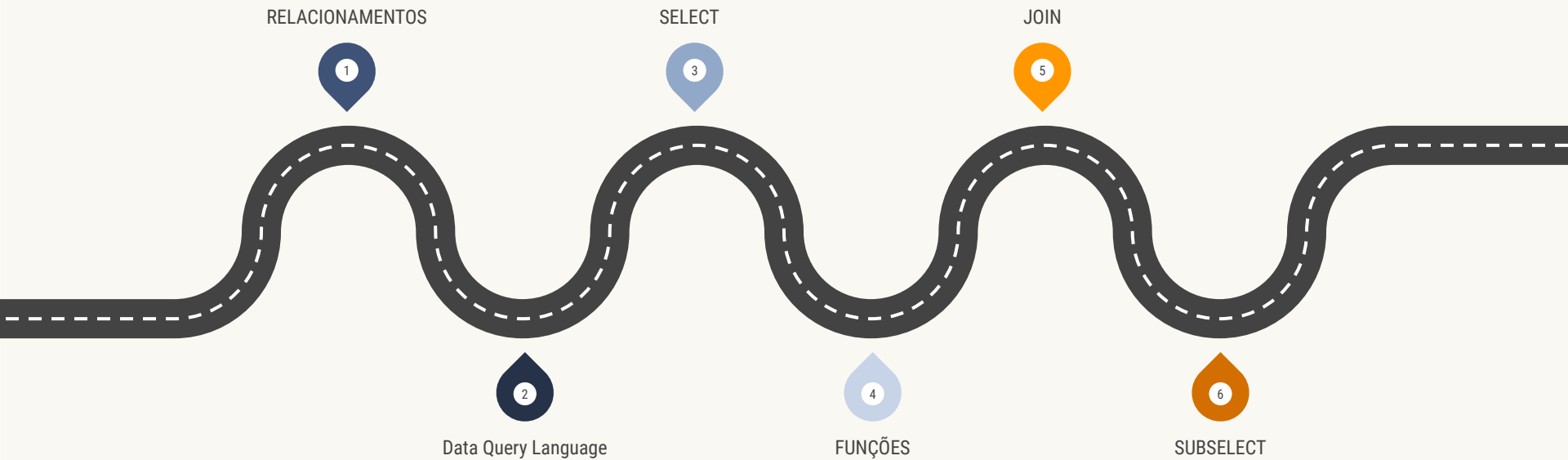
**SENAI**

<LAB365>

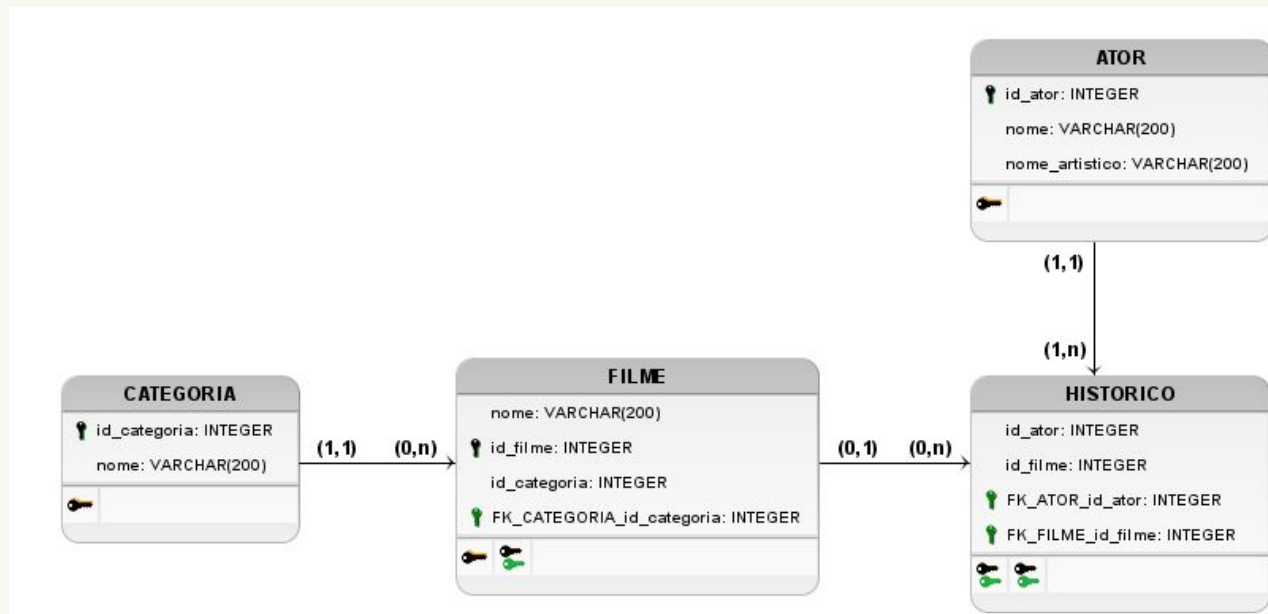
# OBSERVAÇÕES

- A interação é crucial para um melhor entendimento;
- Qualquer dúvida, favor levantar a mão ou enviar no chat;
- Caso queira, pode enviar sua dúvida via Slack;
- Utilize os materiais complementares;

# AGENDA



# EXERCÍCIO 8



## EXERCÍCIO 8

- A partir do modelo lógico apresentado anteriormente, realize:
  - Criação de schema;
  - Criação das tabelas contendo os tipos de dados informados no modelo.
  - Insira dados nas tabelas;
  - Atualize pelo menos três linhas de dados em cada uma das tabelas;
  - Remova pelo menos um dado em cada uma das tabelas;

# DATA DEFINITION LANGUAGE



DEVinHouse

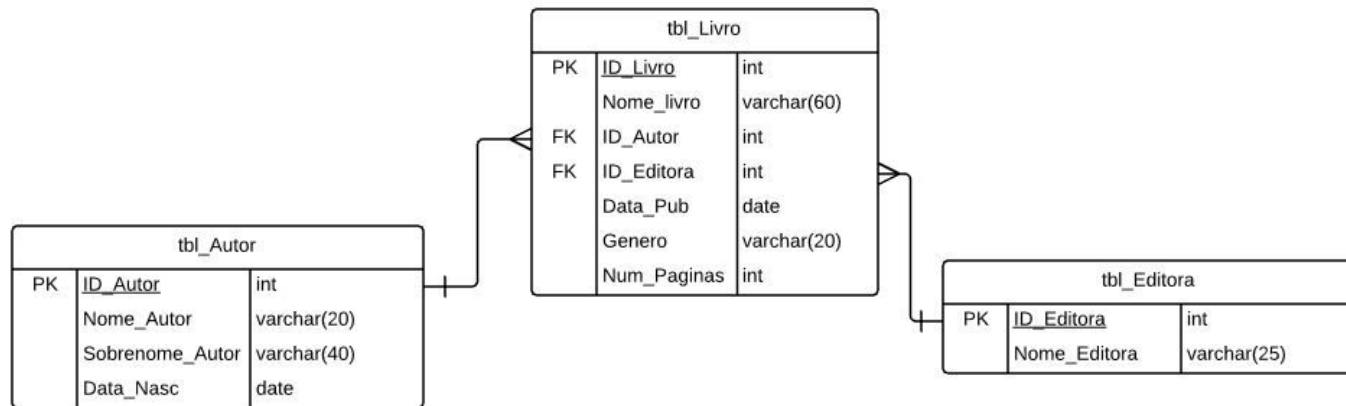
Parcerias para desenvolver a sua carreira

**SENAI**

<LAB365>

# FOREIGN KEY - FK

- A FK é o identificador da tabela A quando se referencia a tabela B;
- É uma coluna em uma tabela que se referencia à chave primária de outra tabela;
- Utilizamos um padrão de nomenclatura para diferenciar um PK de uma FK;
  - schema\_name.table\_a = PK -> id\_table\_a
  - schema\_name.table\_b = FK -> fk\_id\_table\_a



# FOREIGN KEY - FK

```
create table if not exists sql_class.tbl_autor (  
  id_autor int primary key,  
  nome_autor varchar(30),  
  sobrenome_autor varchar(40),  
  data_nasc date  
);
```

```
create table if not exists sql_class.tbl_livro (  
  id_livro int primary key ,  
  nome_livro varchar(40),  
  id_autor int not null,  
  id_editora int not null,  
  data_pub date,  
  genero varchar(25),  
  num_paginas int,  
  foreign key (id_autor) references tbl_Autor (id_autor)  
);
```



# FK APÓS DEFINIÇÃO DA TABELA

- Adicionar uma restrição de chave estrangeira após a criação da tabela

```
ALTER TABLE schema_name.child_table_name  
ADD CONSTRAINT constraint_name FOREIGN KEY (column)  
REFERENCES father_table_name(column);
```

- Exemplo:

```
CREATE TABLE ist.tbl_editora (  
    id_editora int not null,  
    nome_editora varchar(25),  
    PRIMARY KEY (id_editora)  
);
```

```
ALTER TABLE ist.tbl_livro  
ADD CONSTRAINT fk_id_editora FOREIGN KEY (id_editora)  
REFERENCES tbl_editora(id_editora);
```

# RELACIONAMENTOS - (1,1)


O relacionamento **One to One** indica que um registro da entidade **A** vai ter **apenas um** registro da entidade **B**:

```
create table pessoas (  
  id SERIAL PRIMARY KEY,  
  nome VARCHAR(80) NOT NULL,  
  sobrenome VARCHAR(80) NOT NULL,  
  endereco_id INT REFERENCES enderecos (id)  
);
```

**A**

```
create table enderecos (  
  id SERIAL PRIMARY KEY,  
  rua VARCHAR(255) NOT NULL,  
  numero INT,  
  complemento VARCHAR(40) NOT NULL,  
  bairro VARCHAR(80) NOT NULL,  
  cidade VARCHAR(80) NOT NULL,  
  uf CHAR(2) NOT NULL  
);
```

**B**



# RELACIONAMENTOS - (1,N)

O relacionamento **One to Many** indica que um registro da entidade **A** vai ter **um ou mais** registros da entidade **B**:

```
create table pedidos (  
  id SERIAL PRIMARY KEY,  
  data TIMESTAMP NOT NULL DEFAULT current_timestamp,  
  cliente VARCHAR(255) NOT NULL,  
  valor_total NUMERIC NOT NULL DEFAULT 0,  
);
```

```
create table pedido_itens (  
  id SERIAL PRIMARY KEY,  
  pedido_id VARCHAR(80) NOT NULL,  
  produto_titulo VARCHAR(80) NOT NULL,  
  descricao VARCHAR(255) NOT NULL,  
  valor NUMERIC,  
  FOREIGN KEY (pedido_id) REFERENCES pedidos (id)  
);
```



Vários **Pedido**itens para  
um mesmo **Pedido**

# RELACIONAMENTOS - (N,N)

O relacionamento **Many to Many**, só pode ser definido caso tenhamos uma **tabela intermediária** onde irá armazenar os relacionamentos múltiplos, nela constarão todas as chaves primárias da entidade **A** e da entidade **B**:

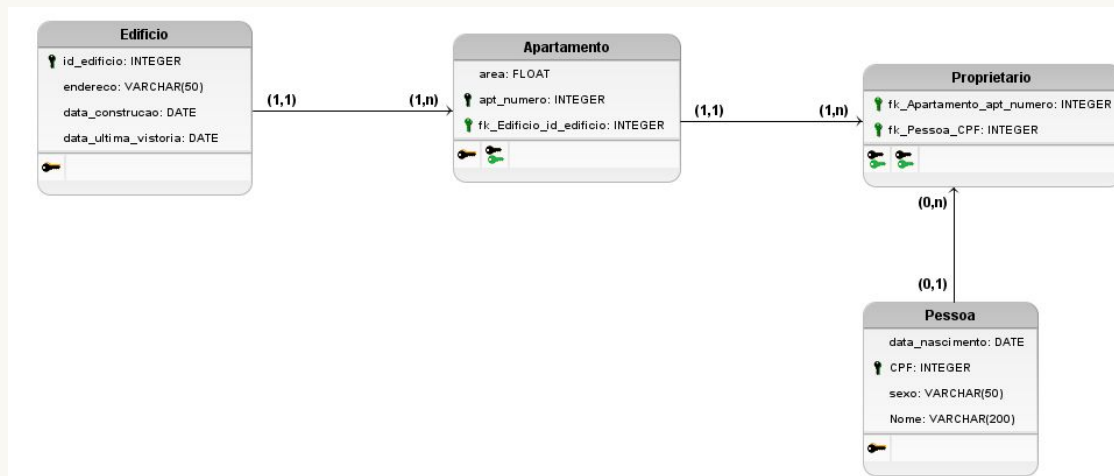
```
create table depositos (  
    id SERIAL PRIMARY KEY,  
    endereco VARCHAR(255) NOT NULL,  
    nome VARCHAR(80),  
);
```

```
create table deposito_produto (  
    id SERIAL PRIMARY KEY,  
    produto_id INT,  
    deposito_id INT REFERENCES depositos(id) ON DELETE SET NULL,  
    FOREIGN KEY (produto_id) REFERENCES produtos(id) ON DELETE CASCADE  
);
```

```
create table produtos (  
    id SERIAL PRIMARY KEY,  
    produto_titulo VARCHAR(80) NOT NULL,  
    descricao VARCHAR(255) NOT NULL,  
);
```

# EXERCÍCIO 9

- Informadas na modelagem abaixo:
  - Criar as tabelas;
  - Definir os relacionamentos entre elas;
  - Insiram dados em cada uma das tabelas;



# DATA QUERY LANGUAGE



DEVinHouse

Parcerias para desenvolver a sua carreira

**SENAI**

<LAB365>

# SELECT

- Para consultarmos dados utilizamos o SELECT, dessa forma conseguimos retornar dados presentes nas tabelas que queremos consultar, para utilizá-lo, basta seguir o comando:

Retornar todas as colunas

```
SELECT * FROM schema_name.child_table_name;
```

Selecionar quais colunas serão retornadas:

```
SELECT column_a, column_b ... FROM schema_name.child_table_name;
```

# FUNÇÕES DE AGREGAÇÃO

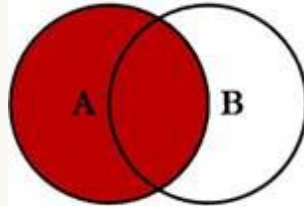
- COUNT ( column )
  - Recupera o número de elementos de um determinado grupo.
- SUM ( column )
  - Calcula o somatório dos valores de uma coluna.
- AVG ( column )
  - Calcula a média entre os valores de uma coluna.
- MIN ( column )
  - Retorna o menor valor contido em uma coluna.
- MAX ( column )
  - Retorna o maior valor armazenado em uma coluna.
- Os valores nulos não são considerados para o cálculo do resultado da função.
  - A única exceção é o COUNT(\*);



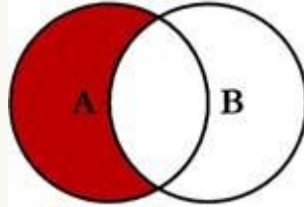
# FUNÇÕES DE AGRUPAMENTO

- A cláusula agrupada pode ser combinada com a cláusula WHERE, sendo que os dados só são agrupados depois de filtrados;
- Quando o critério de agrupamento possui mais de um item de grupo, a classificação dos dados nos vários grupos levará em consideração a combinação dos valores das colunas que compõem a cláusula GROUP BY;
- A cláusula HAVING só pode ser definida combinada com a cláusula GROUP BY
- Somente itens de grupo e funções de agregação podem ser utilizadas nas expressões condicionais do HAVING;
- As funções de agregação utilizadas na cláusula HAVING não precisam ser as mesmas utilizadas na projeção;

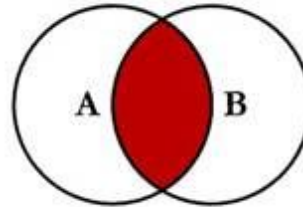
## SQL JOINS



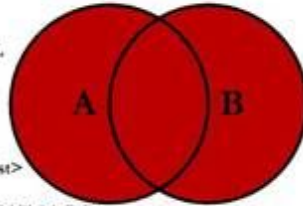
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



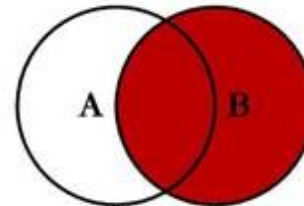
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



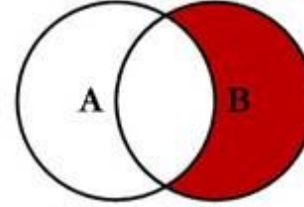
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



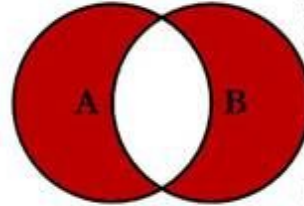
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
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
```

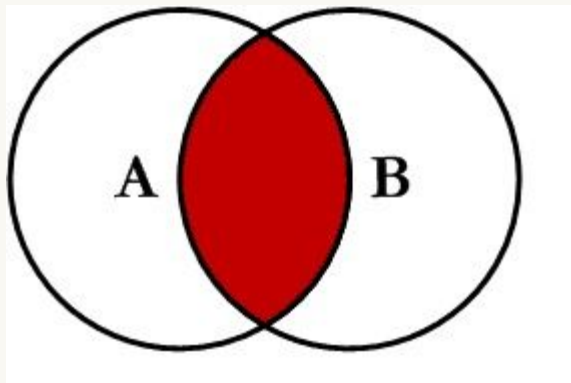


```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# INNER JOIN

- Retorna registros comuns às duas tabelas.

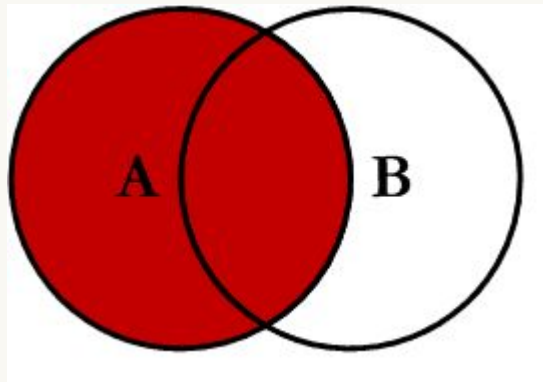
```
SELECT * FROM cliente as A  
INNER JOIN endereco as B ON cliente.codigo = endereco.codigo
```



# LEFT JOIN

- Retorna registros que estão na tabela A (cliente) e os registros comuns de B (endereço) e A.

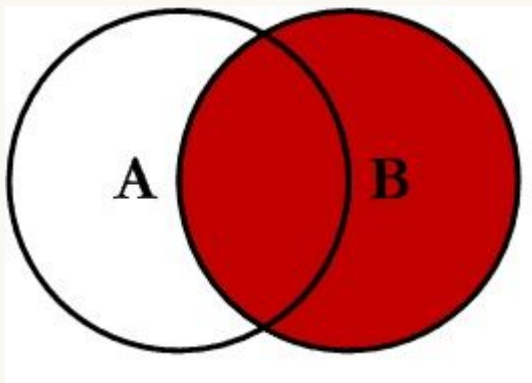
```
SELECT * FROM cliente as A  
LEFT JOIN endereco as B ON cliente.codigo = endereco.codigo
```



# RIGHT JOIN

- Retorna registros que estão na tabela B (endereco) e os registros comuns de A (cliente) e B.

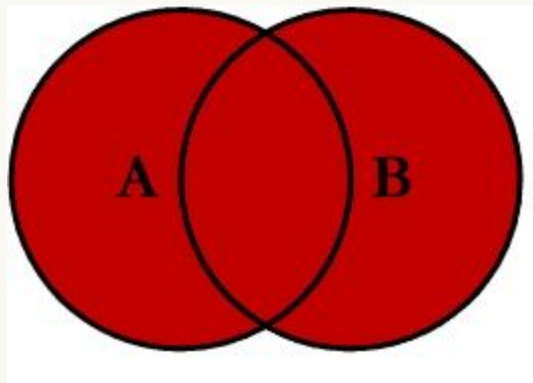
```
SELECT * FROM cliente as A  
RIGHT JOIN endereco as B ON cliente.codigo = endereco.codigo
```



# OUTER JOIN

- Retorna todos registros que estão na tabela A (cliente) e todos os registros na tabela B (endereço).

```
SELECT * FROM cliente as A  
FULL OUTER JOIN endereco as B ON cliente.codigo = endereco.codigo
```



# SUBSELECT

- É um recurso oferecido pelo SQL que permite a um comando SELECT obter e utilizar dinamicamente os dados de um outro para composição do resultado final;
- Consulta Principal:
  - Quais são os alunos que possuem a MGP maior que a do Aluno X (matrícula 123)?
    - Subconsulta
      - Qual a média do aluno X?



# DEVinHouse

Parcerias para desenvolver a sua carreira

**OBRIGADO!**



<LAB365>