

# Modelo Relacional e SQL

## Fundamentos e Projeto de Banco de Dados

Prof. Elton Sarmanho<sup>1</sup>  
eltonss@ufpa.br

<sup>1</sup>Faculdade de Sistemas de Informação - UFPA Campus Cametá

23 de dezembro de 2025



# Roteiro de Aprendizagem

O Modelo Relacional

Linguagem SQL

Consultas Avançadas (DQL)

Joins: Relacionando Tabelas

Agregação e Agrupamento

Conclusão e Exercícios

## Estrutura do Modelo Relacional

O modelo relacional organiza os dados em **relações** (tabelas), baseadas na teoria dos conjuntos.

- ▶ **Relação (Tabela):** Conjunto de registros sobre uma entidade.
- ▶ **Tupla (Linha):** Uma instância única (registro) da relação.
- ▶ **Atributo (Coluna):** Uma propriedade ou característica da entidade.
- ▶ **Domínio:** Conjunto de valores permitidos para um atributo.

Tupla

ID	Nome	Curso
101	Ana	SI
102	Beto	Eng

Atributos



# O Poder das Chaves

As chaves garantem a integridade e os relacionamentos entre as tabelas.

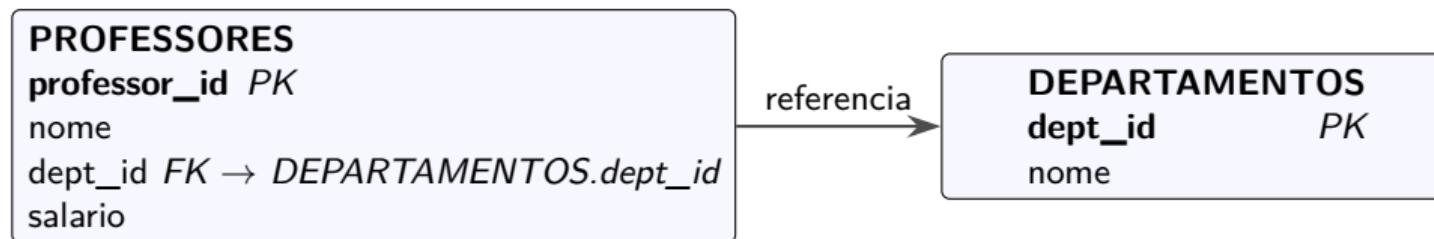
## Classificação

**Chave Primária (PK):** Identificador **único** de uma tupla. Não pode ser NULL.

**Chave Estrangeira (FK):** Atributo que cria um vínculo com a PK de outra tabela, estabelecendo o relacionamento.

**Chave Candidata:** Qualquer atributo capaz de ser uma PK (ex: CPF, E-mail).

# O Poder das Chaves



**Ideia:** uma **relação** (tabela) é descrita por um **esquema** (colunas/atributos). A **PK** identifica unicamente tuplas; a **FK** representa o relacionamento entre tabelas.



# Integridade Referencial

## Restrição de Integridade

Uma **Chave Estrangeira** deve sempre corresponder a uma **Chave Primária** existente na tabela referenciada ou ser NULA (se permitido).

## Consequências práticas:

- ▶ Não se pode excluir um departamento se houver professores vinculados a ele (a menos que use *CASCADE*).
- ▶ Não se pode cadastrar um aluno em um curso que não existe.



# Restrições de chave Estrangeira (FK)

*Inclusão de um registro na tabela que contém a FK:*

- ▶ **Deve ser garantido** que o valor da chave estrangeira **apareça** na coluna da chave primária (PK) referenciada.

*Alteração do valor da chave estrangeira:*

- ▶ **Deve ser garantido** que o novo valor da chave estrangeira **apareça** na coluna da chave primária referenciada.

## Restrições de chave Estrangeira (FK)

*Exclusão de um registro da tabela que contém a PK referenciada pela FK (deixar registros órfãos):*

- ▶ **Deve ser garantido** que na coluna chave estrangeira **não apareça** o valor da chave primária que está sendo excluída.

*Alteração do valor da PK referenciada pela FK*

- ▶ **Deve ser garantido** que na coluna chave estrangeira **não apareça** o antigo valor da chave primária que está sendo alterada.

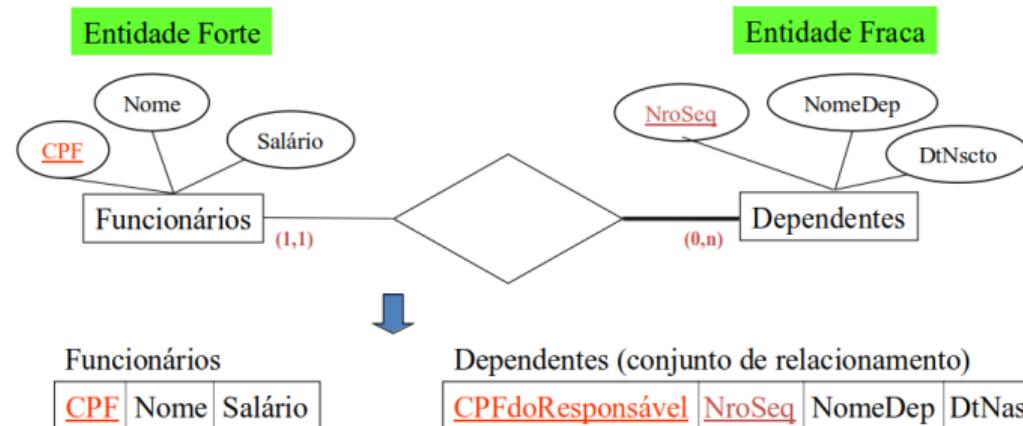


## Conceitos do Modelo E-R:Entidade Fraca X Forte

- ▶ Entidades Fracas: são aquelas que "não" possuem atributos suficientes para formar uma chave primária.
- ▶ Entidades Fortes: são aquelas que "possuem" atributos para formar uma chave primária.
  - ▶ Identificador, ou discriminador: é o conjunto de atributos da entidade que tem a propriedade de determinar de forma única cada instância da entidade.

# Chave Entidades Fortes X Entidades Fracas

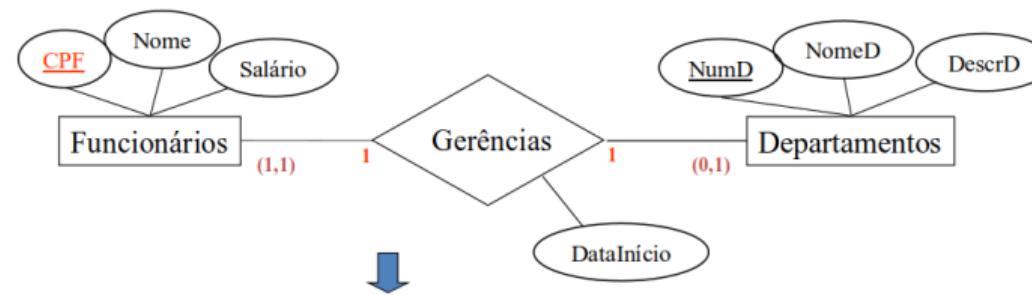
- **Conjunto de entidades fortes:** “parte” da chave primária (*CPFdoResponsável*) do conjunto de relacionamento é a chave primária da relação Funcionários (*CPF*).
- **Conjunto de entidades fracas:** a relação Dependentes é formada pelos atributos da entidade fraca (*NroSeq*, *NomeDep* e *DtNscto*) e a chave primária do conjunto de entidades fortes (*CPFdoResponsável*). *Chave composta* (*CPFdoResponsável* + *NroSeq*).



O atributo *NroSeq* é uma chave parcial, que distingue os vários dependentes de um dado funcionário. Para se determinar um dependente precisa-se também do *CPF* do funcionário.

# Chave Conjuntos de Relacionamentos (Binários) Um para Um

- Com duas tabelas, uma para cada conjunto de entidades que participa do relacionamento.
- Na entidade que participa do relacionamento deve-se incluir como chave estrangeira (**CPFdoGerente**) a chave primária da entidade que participa parcialmente do relacionamento.
- Incluir também colunas com os atributos do relacionamento.

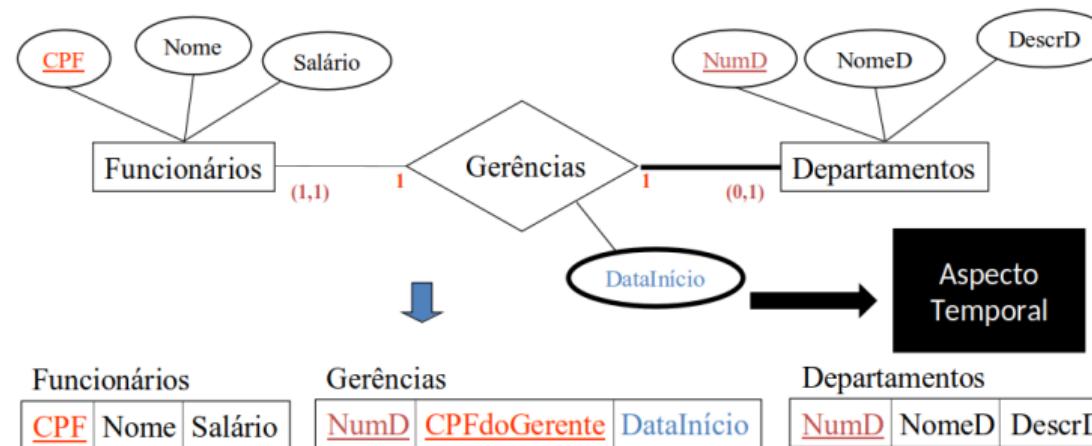


Funcionários
<b>CPF</b>   Nome   Salário

Departamentos
<b>NumD</b>   NomeD   DescriD   <b>CPFdoGerente</b>   DataInicio

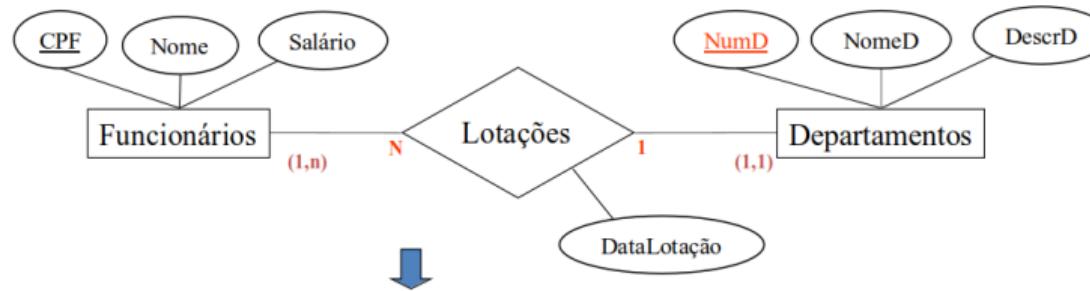
# Chave Conjuntos de Relacionamentos (Binários) Um para Um

Para manter o histórico indicando “todos” os funcionários que foram gerentes de um departamento ao longo da história. Usa-se o atributo descritivo que indica o “DataInício” da gerência como multivalorado.



# Chave Conjuntos de Relacionamentos (Binários) Muitos para Um

- Tabelas combinadas. Com duas tabelas, uma para cada conjunto de entidades que participa do relacionamento.
- Incluir como *chave estrangeira*, na tabela do “lado muitos” (ou “lado N”), a chave primária (**NumD**) da tabela do “lado um”.
- Incluir também colunas com os atributos do relacionamento.



Funcionários

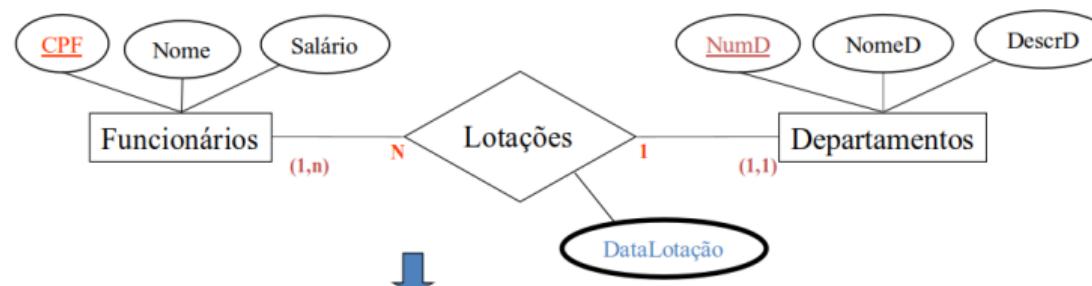
<u>CPF</u>	Nome	Salário	<u>NumD</u>	DataLotação
------------	------	---------	-------------	-------------

Departamentos

<u>NumD</u>	NomeD	DescriD
-------------	-------	---------

# Chave Conjuntos de Relacionamentos (Binários) Muitos para Um

Para manter o histórico indicando “todos” os departamentos pelos quais um funcionário esteve matriculado (ou lotado) ao longo da história. Usa-se o atributo descritivo que indica a “DataLotação” da gerência como multivalorado.



## Funcionários

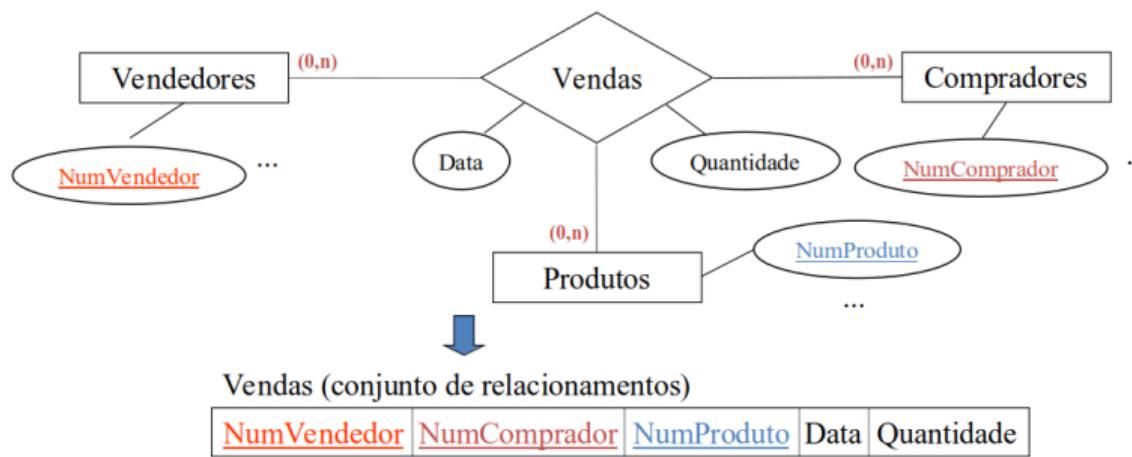
## Lotações

## Departamentos

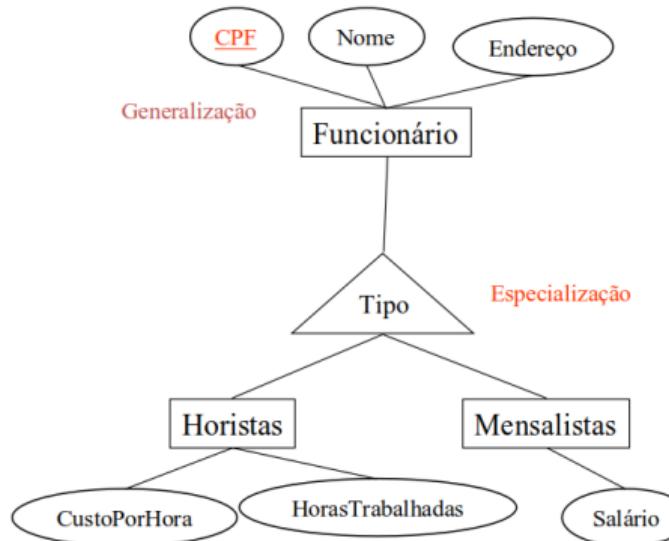


## Chave Conjuntos de Relacionamentos Ternário

- Criar uma nova tabela (Vendas) com a chave primária “composta” das chaves primárias das tabelas que representam os conjuntos de entidades participantes.
- Incluir também colunas com os atributos do relacionamento.



# Generalização / Especialização



Implementação Indicada: “2 tabelas”

Horistas

CPF	Nome	Endereço	CustoPorHora	HorasTrabalhadas
-----	------	----------	--------------	------------------

Mensalistas

CPF	Nome	Endereço	Salário
-----	------	----------	---------



## Linguagem de Consulta

- ▶ Uma linguagem de consulta (query language) é a linguagem por meio da qual os usuários obtêm informações do banco de dados.
- ▶ As linguagens de consulta podem ser categorizadas como **procedurais** ou **não-procedurais**.
  - ▶ Em uma linguagem procedural, o usuário deve "ensinar" ao sistema a realização de um "seqüência de operações" no banco de dados para obter o resultado desejado.
  - ▶ Em uma linguagem não-procedural (ou declarativa), o usuário "descreve a informação" desejada "sem fornecer um procedimento" específico para a obtenção dessas informações.



# O que é SQL?

## Definição de SQL

SQL significa *Structured Query Language*. É uma linguagem de programação especializada em gerenciar e manipular bancos de dados relacionais.

## Breve História e Evolução do SQL

Desenvolvido inicialmente nos anos 70 pela IBM, o SQL evoluiu e se tornou o padrão da indústria para gerenciamento de bancos de dados, mantido pela ANSI e pela ISO.



# SQL: Structured Query Language

A linguagem padrão para interagir com Bancos de Dados Relacionais. Divide-se em sub-linguagens:

- ▶ **DDL (Data Definition Language):** Define a estrutura (CREATE, ALTER, DROP).
- ▶ **DML (Data Manipulation Language):** Manipula os dados (INSERT, UPDATE, DELETE).
- ▶ **DQL (Data Query Language):** Consulta os dados (SELECT).
- ▶ **DCL/TCL:** Controle de acesso e transações (GRANT, COMMIT).



# Conceitos Básicos de Bancos de Dados

## O que é um Banco de Dados Relacional?

Um banco de dados relacional é um tipo de banco de dados que armazena e fornece acesso a dados relacionados entre si. Os dados são organizados em tabelas compostas por linhas e colunas.

## Tabelas, Linhas e Colunas

- ▶ **Tabelas:** Estruturas que armazenam dados sobre um determinado assunto.
- ▶ **Linhas:** Também conhecidas como registros, contêm os dados individuais.
- ▶ **Colunas:** Representam os diferentes atributos dos dados.



# Conceitos Básicos de Bancos de Dados

## Chaves Primárias e Estrangeiras

- ▶ **Chaves Primárias:** Um campo único que identifica cada linha (registro) de forma exclusiva em uma tabela.
- ▶ **Chaves Estrangeiras:** Um campo em uma tabela que é a chave primária em outra tabela, usado para estabelecer e impor um relacionamento entre os dados.



# SGBDs

## Introdução a Sistemas de Gerenciamento de Banco de Dados (SGBDs)

SGBDs são softwares que permitem criar, gerenciar e manipular bancos de dados. Eles fornecem uma interface para os usuários interagirem com os dados utilizando SQL.

## Exemplos de SGBDs

- ▶ **MySQL:** Um sistema de gerenciamento de banco de dados relacional de código aberto amplamente utilizado.
- ▶ **PostgreSQL:** Um SGBD relacional avançado, conhecido por sua robustez e conformidade com padrões.
- ▶ **SQLite:** Uma biblioteca em linguagem C que fornece um SGBD relacional leve e autônomo.



# SGBDs

## Papel do SQL nos SGBDs

SQL é a linguagem padrão para interagir com SGBDs. É usada para realizar tarefas como a criação de tabelas, a inserção de dados, a realização de consultas e a atualização de registros.



# Introdução aos Tipos de Dados em SQL

## O que são Tipos de Dados?

Tipos de dados em SQL definem o tipo de valor que pode ser armazenado em cada coluna de uma tabela. É crucial escolher o tipo de dado correto para cada coluna para garantir precisão e eficiência.

# Tipos Numéricos

## Tipos Numéricos Inteiros

- ▶ **INT / INTEGER:** Armazena números inteiros.
- ▶ **SMALLINT:** Para números inteiros menores.
- ▶ **BIGINT:** Para números inteiros muito grandes.

## Tipos Numéricos de Ponto Flutuante

- ▶ **FLOAT, REAL, DOUBLE PRECISION:** Para números reais com precisão variável.
- ▶ **NUMERIC, DECIMAL:** Números de ponto fixo, úteis para valores monetários.



# Tipos de Dados de Caracteres

## Tipos de Caracteres

- ▶ **CHAR(n)**: Caracteres de comprimento fixo.
- ▶ **VARCHAR(n)**: Caracteres de comprimento variável.
- ▶ **TEXT**: Para strings longas de texto.

# Tipos de Dados de Data e Hora

## Trabalhando com Data e Hora

- ▶ **DATE:** Armazena datas.
- ▶ **TIME:** Armazena horários.
- ▶ **TIMESTAMP:** Combinação de data e hora.

# Outros Tipos de Dados

## Tipos Diversos

- ▶ **BOOLEAN:** Armazena valores verdadeiros ou falsos.
- ▶ **BINARY, VARBINARY:** Para armazenar dados binários (Ex: vídeo, áudio ou Figuras).
- ▶ **ENUM:** Lista predefinida de valores.

## Outros Tipos de Dados

- ▶ O valor nulo (`null`) é um membro de todos os domínios. Para certos atributos, entretanto, valores nulos podem ser inadequados.
- ▶ Como por exemplo, no valor de chaves primárias ou de um atributo como o CPF.
- ▶ A SQL permite que a declaração de domínios de um atributo inclua a especificação de **not null**, proibindo, assim, a inserção de valores nulos para esse tipo de atributo. Qualquer modificação que possa resultar na inserção de um valor nulo em um domínio **not null** gera um diagnóstico de erro.

# Comentários

## Como usar ?

A utilização de comentários em sentenças SQL faz-se utilizando a combinação /\* e \*/ para delimitar uma ou mais linhas de texto considerado como um comentário.

```
1 /*  
2 CREATE DATABASE 'databaseTest';  
3 */
```

# Criando o Banco Universitário (PostgreSQL)

Vamos criar um cenário real de Universidade com relacionamentos.

```
1 -- Tabela "Pai" (lado 1 do relacionamento)
2 CREATE TABLE departamentos (
3     id SERIAL PRIMARY KEY,
4     nome VARCHAR(100) NOT NULL
5 );
6
7 -- Tabela "Filha" (lado N do relacionamento)
8 CREATE TABLE professores (
9     id SERIAL PRIMARY KEY,
10    nome VARCHAR(100) NOT NULL,
11    salario NUMERIC(10,2),
12    data_admissao DATE,
13    departamento_id INT,
14    -- Definicao da Chave Estrangeira
15    CONSTRAINT fk_depto FOREIGN KEY (departamento_id)
16    REFERENCES departamentos(id));
```



# Comando CREATE DOMAIN

## Uso do Comando

A SQL-92 permite a definição de domínios usando a cláusula `create domain` usando comando *create domain*

```
1 create domain dom_sexo as char(1)
2 check(value in('M', 'F'))
3 not null;
4
5 CREATE TABLE estudantes (
6     id SERIAL PRIMARY KEY,
7     nome VARCHAR(100),
8     sexo dom_sexo,
9     idade INT
10 );
```



```
create domain dom_mes as smallint
check(value between 1 and 12)
```

```
create domain dom_sexo as char(1)
check(value in('M', 'F'))
```

```
create domain dom_sexo as char(1)
check(value in('M', 'F'))
not null
```

para apagar um domínio criado:  
`drop domain nomeDoDomínio`

```
create domain dom_salario as numeric(8, 2)
default 380.00
check(value >= 380.00)
```

onde:

value

representa o valor atribuído ao atributo

default

define o valor padrão para o atributo

check

verifica se o valor informado para o atributo

satisfaz a condição especificada



# Comando ALTER

## Uso do Comando ALTER

Permite modificar a estrutura de uma tabela existente, como adicionar, remover ou modificar colunas.

```
1 ALTER TABLE estudantes  
2 ADD COLUMN email VARCHAR(100);
```



# Comando DROP

## Uso do Comando DROP

Usado para remover tabelas ou outros objetos do banco de dados. A ação é permanente e não pode ser desfeita.

1

2 **DROP TABLE estudantes;**

# Comandos DML - Data Manipulation Language

## Definição de DML

DML (Data Manipulation Language) é composto por comandos utilizados para inserir, modificar e excluir dados nas tabelas.

## Principais Comandos DML

- ▶ **INSERT:** Insere novos dados em tabelas.
- ▶ **UPDATE:** Atualiza dados existentes.
- ▶ **DELETE:** Remove dados de tabelas.



# Comandos DML - Data Manipulation Language

## Comando INSERT

Usado para adicionar novos registros a uma tabela.

- ▶ Sintaxe Básica: `INSERT INTO tabela (coluna1, coluna2) VALUES (valor1, valor2);`
- ▶ Exemplo: `INSERT INTO clientes (nome, idade) VALUES ('Ana', 28);`

# Inserindo Dados (INSERT)

Lembre-se: Devemos inserir primeiro nas tabelas fortes (independentes).

```
1 -- 1. Inserir Departamentos
2 INSERT INTO departamentos (nome) VALUES
3     ('Ciencias Exatas'),
4     ('Ciencias Humanas'),
5     ('Saude');
6
7 -- 2. Inserir Professores (referenciando departamentos criados)
8 INSERT INTO professores (nome, salario, departamento_id) VALUES
9     ('Prof. Elton', 12000.00, 1),
10    ('Prof. Carlos', 9500.50, 2),
11    ('Prof. Fabricio', 11000.00, 3);
```



# Comandos DML - Data Manipulation Language

## Comando UPDATE

Permite modificar registros existentes.

- ▶ Sintaxe Básica: UPDATE tabela SET coluna1 = valor1 WHERE condição;
- ▶ Exemplo: UPDATE clientes SET idade = 29 WHERE nome = 'Ana';

# Comandos DML - Data Manipulation Language

## Comando DELETE

Remove registros de uma tabela.

- ▶ Sintaxe Básica: `DELETE FROM tabela WHERE condição;`
- ▶ Exemplo: `DELETE FROM clientes WHERE nome = 'Ana';`

# Atualizando e Excluindo (UPDATE / DELETE)

## Cuidado!

Sempre use a cláusula WHERE ao fazer UPDATE ou DELETE. Sem ela, você afetará **toda** a tabela.

```
1 -- Aumentar salario em 10% para o departamento 1
2 UPDATE professores
3 SET salario = salario * 1.10
4 WHERE departamento_id = 1;
5
6 -- Remover professores com salario muito baixo
7 DELETE FROM professores
8 WHERE salario < 1500;
```



# Comando SELECT

## Uso do Comando SELECT

O comando SELECT é usado para consultar e recuperar dados de uma ou mais tabelas.

```
1 SELECT nome, idade FROM estudantes WHERE idade > 18;
```

# Filtrando Resultados com WHERE

## Uso da Cláusula WHERE

A cláusula WHERE é usada com o comando SELECT para filtrar registros de acordo com condições específicas.

```
1 SELECT * FROM estudantes WHERE curso = 'Ciencia da Computacao';
```



# Anatomia do SELECT

A ordem que escrevemos é diferente da ordem que o banco executa.

## Ordem de Escrita:

1. SELECT (Colunas)
2. FROM (Tabelas)
3. WHERE (Filtros de linha)
4. GROUP BY (Agrupamento)
5. HAVING (Filtros de grupo)
6. ORDER BY (Ordenação)

## Ordem Lógica de Execução:

1. FROM (Pega os dados)
2. WHERE (Filtra linhas)
3. GROUP BY (Agrupa)
4. HAVING (Filtra grupos)
5. SELECT (Escolhe colunas)
6. ORDER BY (Ordena o final)



## Filtrando com WHERE e LIKE

O operador LIKE permite busca por padrões em texto.

- ▶ %: Substitui qualquer sequência de caracteres.
- ▶ \_: Substitui um único caractere.

```
1 -- Nomes que começam com 'A'  
2 SELECT * FROM alunos WHERE nome LIKE 'A%';  
3  
4 -- Nomes que terminam com 'Silva'  
5 SELECT * FROM alunos WHERE nome LIKE '%Silva';  
6  
7 -- Nomes que tem 'r' na segunda letra (ex: Bruno, Breno)  
8 SELECT * FROM alunos WHERE nome LIKE '_r%';
```



## Cláusula LIKE

As operações em strings mais usadas são as checagens para verificação de coincidências de pares, usando o operador **like** combinado com os caracteres especiais: porcentagem (%) e sublinhado (\_).

Expressão	Resultado
<b>LIKE 'A%</b> '	Qualquer string que iniciem com a letra A.
<b>LIKE '%A'</b>	Qualquer string que terminem com a letra A.
<b>LIKE '%A%</b> '	Qualquer string que tenha a letra A em qualquer posição.
<b>LIKE 'A_'</b>	String de dois caracteres que tenham a primeira letra A e o segundo caractere seja qualquer outro.
<b>LIKE '_A'</b>	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja a letra A.
<b>LIKE '_A_'</b>	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.
<b>LIKE '%A_'</b>	Qualquer string que tenha a letra A na penúltima posição e a última seja qualquer outro caractere.
<b>LIKE '_A%</b> '	Qualquer string que tenha a letra A na segunda posição e o primeiro caractere seja qualquer outro caractere.



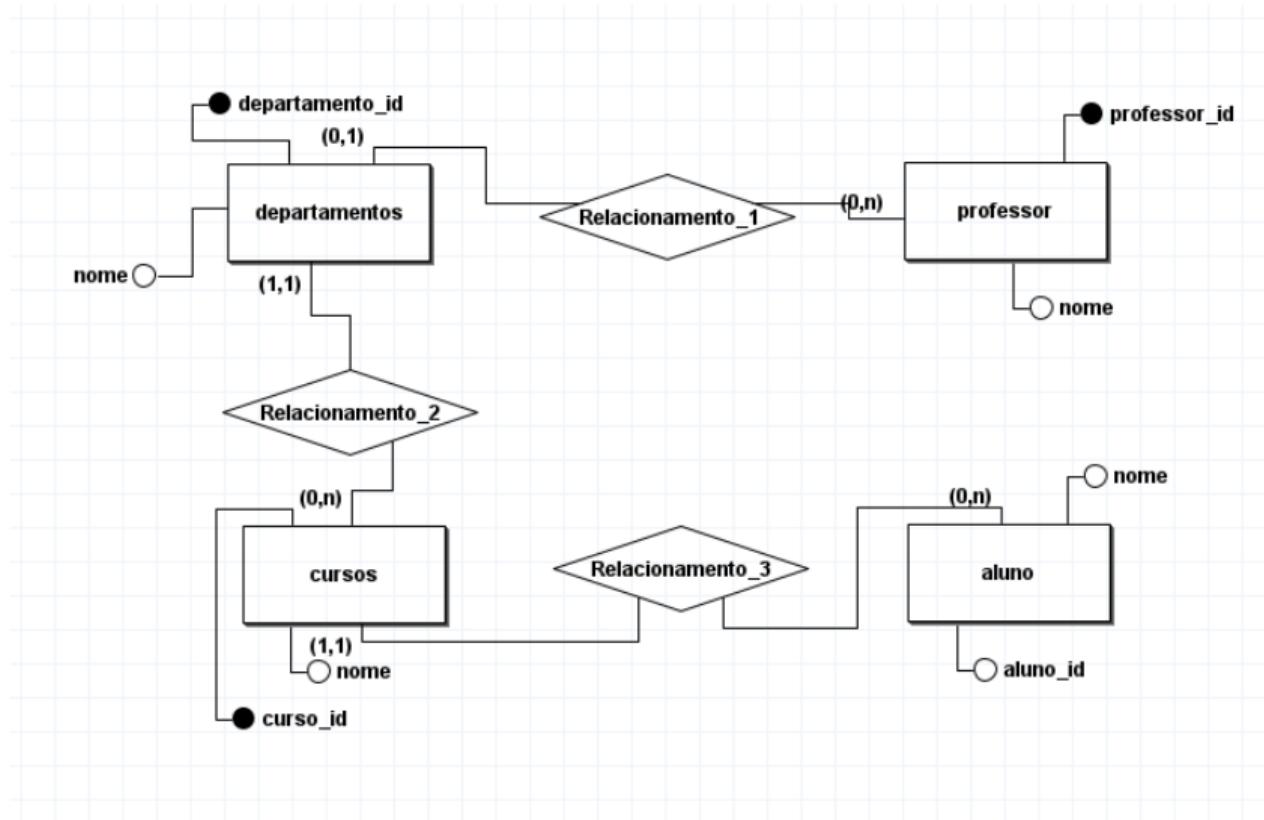
## Cláusula LIKE (2/3)

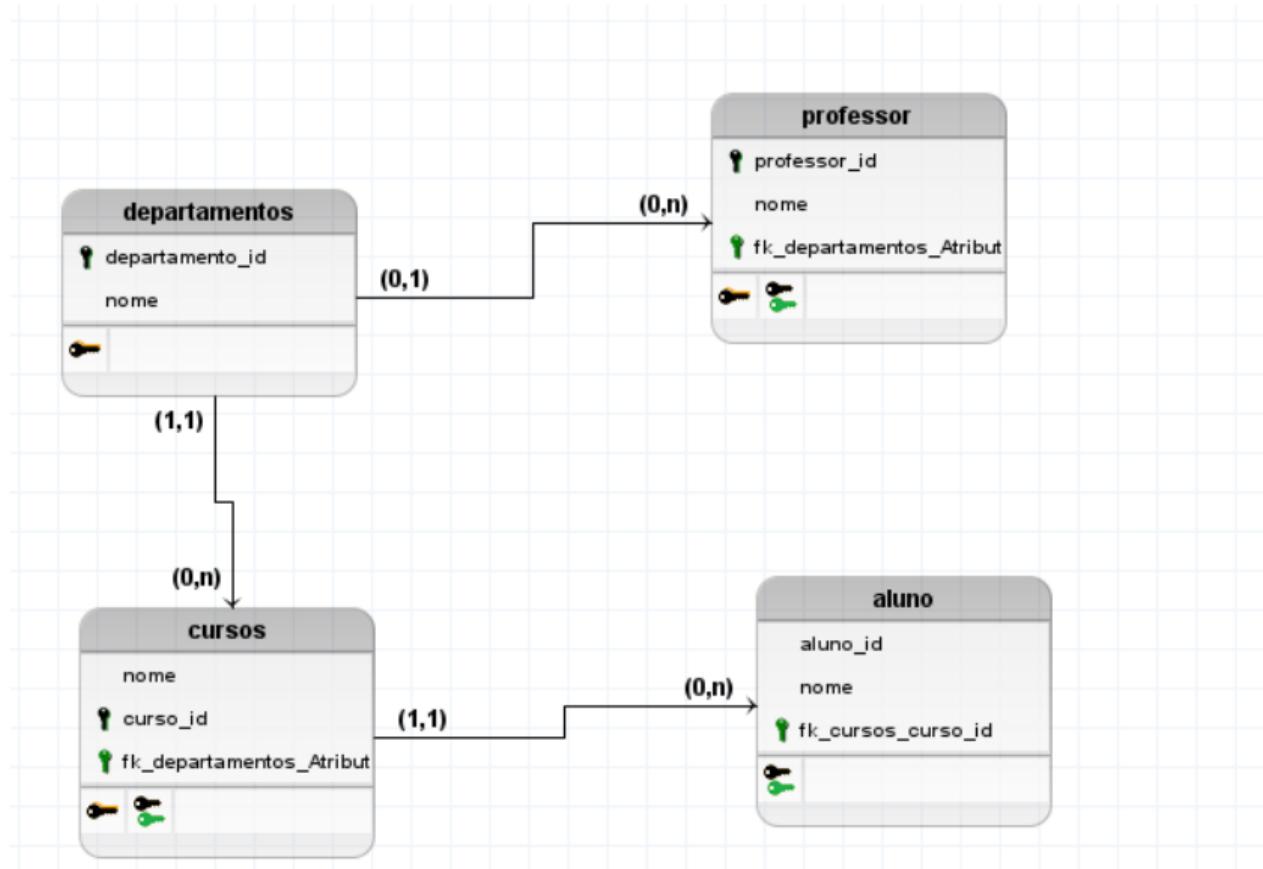
Expressão	Resultado
<u>LIKE</u> ‘ <u>__</u> ’	Qualquer string com exatamente três caracteres.
<u>LIKE</u> ‘ <u>__%</u> ’	Qualquer string com pelo menos três caracteres.

Comparações com strings são sensíveis ao tamanho da letra; isto é, minúsculas não são iguais a maiúsculas, e vice-versa.

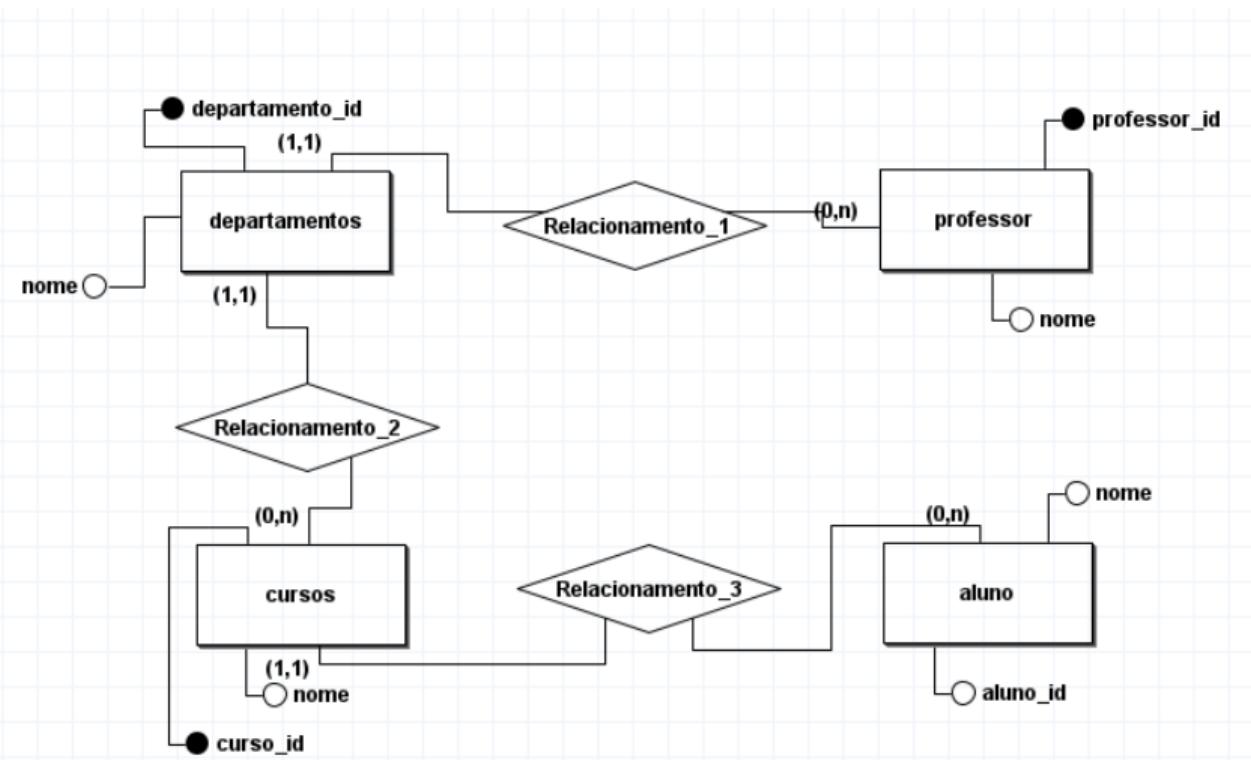
```
select full_name  
from employee  
where full_name like 'Be%'
```

```
select full_name  
from employee  
where full_name like '%BE%'
```

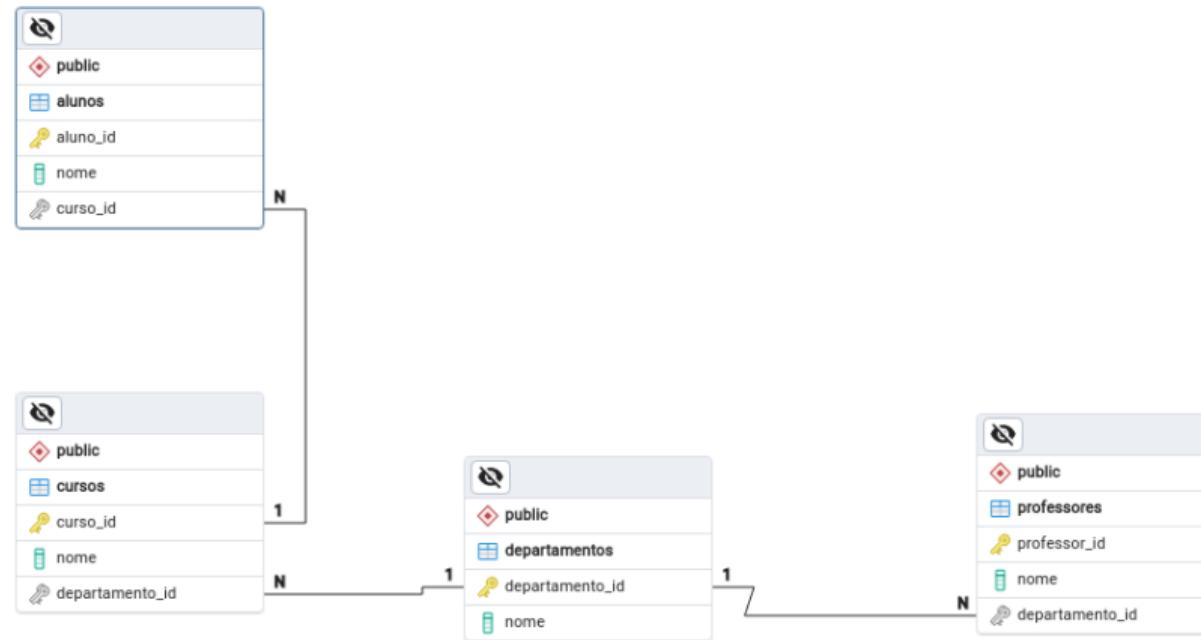




## Veja o detalhe da Cardinalidade Professor em relação a Departamentos



## Modelo Desenvolvido no PostgreSQL



# Script no PostgreSQL

```
1 -- Criacao da base de dados
2 CREATE DATABASE universidade;
3
4 \c universidade
```



## Script no PostgreSQL

- ▶ Novos Comandos e Conceitos irão aparecer nos próximos slides.
- ▶ Alguns serão detalhados.

## Script no PostgreSQL

```
1 -- Criacao da tabela departamentos
2 CREATE TABLE departamentos (
3     departamento_id SERIAL PRIMARY KEY ,
4     nome VARCHAR(100) NOT NULL
5 );
6
7 -- Criacao da tabela professores
8 CREATE TABLE professores (
9     professor_id SERIAL PRIMARY KEY ,
10    nome VARCHAR(100) NOT NULL ,
11    departamento_id INT ,
12    FOREIGN KEY (departamento_id) REFERENCES departamentos(
13        departamento_id)
14 );
```



## Script no PostgreSQL

```
1 -- Criacao da tabela cursos
2 CREATE TABLE cursos (
3     curso_id SERIAL PRIMARY KEY,
4     nome VARCHAR(100) NOT NULL,
5     departamento_id INT,
6     FOREIGN KEY (departamento_id) REFERENCES departamentos(
7         departamento_id)
8 );
9
9 -- Criacao da tabela alunos
10 CREATE TABLE alunos (
11     aluno_id SERIAL PRIMARY KEY,
12     nome VARCHAR(100) NOT NULL,
13     curso_id INT,
14     FOREIGN KEY (curso_id) REFERENCES cursos(curso_id)
15 );
```



# Script no PostgreSQL

```
1 -- Insercao de dados na tabela departamentos
2 INSERT INTO departamentos (nome) VALUES ('EExatas');
3 INSERT INTO departamentos (nome) VALUES ('HHHumanas');
4 INSERT INTO departamentos (nome) VALUES ('Saude');
5
6 -- Fazer select para visualizar os registros
```



## Script no PostgreSQL

```
1 -- Insercao de dados na tabela professores
2 INSERT INTO professores (nome, departamento_id) VALUES ('Prof. Elton',
3   1);
4 INSERT INTO professores (nome, departamento_id) VALUES ('Prof. Carlos',
5   2);
6 INSERT INTO professores (nome, departamento_id) VALUES ('Prof.
7   Fabricio', 3);
```



## Script no PostgreSQL

```
1
2 -- Insercao de dados na tabela cursos
3 INSERT INTO cursos (nome, departamento_id) VALUES ('Sistemas de
   Informaxao', 1);
4 INSERT INTO cursos (nome, departamento_id) VALUES ('Historia', 2);
5 INSERT INTO cursos (nome, departamento_id) VALUES ('Biologiaaaa', 3);
6
7 -- Insercao de dados na tabela alunos
8 INSERT INTO alunos (nome, curso_id) VALUES ('Eva', 1);
9 INSERT INTO alunos (nome, curso_id) VALUES ('Ana', 2);
10 INSERT INTO alunos (nome, curso_id) VALUES ('Ide', 3);
```



## Script no PostgreSQL

- ▶ Vamos selecionar um registro conforme uma condição. Utilizaremos a Cláusula Select

```
1  
2 SELECT * from departamentos;
```



## Script no PostgreSQL

- ▶ Vamos consertar nome deste departamento usando Cláusula *update*

```
1  
2 UPDATE departamentos SET nome = 'Exatas' WHERE departamentos.  
    departamento_id = 1;
```

## Script no PostgreSQL

- ▶ Vamos consertar nome deste departamento usando Cláusula *update*

```
1  
2 UPDATE departamentos SET nome = 'Exatas' WHERE departamentos.  
    departamento_id = 1;
```

- ▶ Faça correção dos demais registros

## Script no PostgreSQL

- ▶ No modelo de negócio exige adição de uma nova informação na tabela alunos.  
Nesse caso seria **email**

```
1  
2 ALTER TABLE alunos ADD COLUMN email VARCHAR(255);
```

- ▶ Lembre-se de que, após adicionar esta coluna, ela estará vazia para os registros existentes. Se necessário, você pode atualizar esses registros com os endereços de e-mail correspondentes usando o comando **UPDATE**



## Script no PostgreSQL

- ▶ No modelo de negócio exige adição de uma nova informação na tabela professor.  
Nesse caso seria **estado civil**

## Script no PostgreSQL

- ▶ No modelo de negócio exige adição de um novo departamento chamado  
**Tecnologia da Informação**

```
1
2 INSERT INTO departamentos (nome) VALUES ('xxx');
```



## Script no PostgreSQL

- ▶ No modelo de negócio exige remoção do novo departamento chamado **Tecnologia da Informação**

```
1  
2 DELETE FROM departamentos WHERE departamento_id = ?;
```



## Script no PostgreSQL

- ▶ No modelo de negócio exige adição de um novo departamento chamado **DTI**.

## Script no PostgreSQL

- ▶ No modelo de negócio exige adição de um novo departamento chamado **DTI**.
- ▶ Os superiores consideraram que não existe necessidade desse departamento. Vamos Excluí-lo.

## Integridade Referencial: ideia central

### Definição

Integridade referencial garante que toda **chave estrangeira (FK)** aponte para uma **chave primária (PK)** existente (ou seja **não existam referências "quebradas"**).

- ▶ **Regra (inserção/atualização):** um valor em FK deve existir na tabela referenciada.
- ▶ **Regra (remoção/alteração da PK):** não é permitido "apagar" ou "trocar" uma PK que esteja sendo referenciada, **a menos que** você trate a ação (RESTRICT/NO ACTION, CASCADE, SET NULL, etc.).
- ▶ Em SQL isso é implementado com FOREIGN KEY ... REFERENCES ....



## Violações - Deletar um Registro Referenciado

- ▶ Tentar deletar um registro que é referenciado por outra tabela através de uma chave estrangeira. Por exemplo:

```
1  
2 -- Supondo que 'departamento_id' em 'professores' eh uma chave  
   estrangeira que referencia 'departamento_id' em 'departamentos'  
3 DELETE FROM departamentos WHERE departamento_id = 1;
```

Se houver professores associados ao departamento\_id = 1, este comando causará um erro de integridade referencial, a menos que você tenha definido a exclusão em cascata.



## Violações - Inserir um Registro com Chave Estrangeira Inexistente

- ▶ Inserir um registro em uma tabela com uma chave estrangeira que não existe na tabela referenciada. Por exemplo:

```
1  
2 -- Supondo que 'curso_id' em 'alunos' eh uma chave estrangeira que  
   referencia 'curso_id' em 'cursos'  
3 INSERT INTO alunos (nome, curso_id) VALUES ('Joao', 9999);
```

Se não houver um curso com curso\_id = 9999, esse comando falhará.



## Violações - Atualizar um Registro Violando a Chave Estrangeira

- ▶ Alterar o valor de uma chave estrangeira para um valor que não existe na tabela referenciada. Por exemplo:

```
1 -- Supondo que 'departamento_id' em 'professores' eh uma chave  
2   estrangeira que referencia 'departamento_id' em 'departamentos'  
3 UPDATE professores SET departamento_id = 9999 WHERE professor_id = 1;
```

Se não houver um departamento com departamento\_id = 9999, este comando causará um erro.



## Violações - Adicionar uma Chave Estrangeira Sem Correspondência

- ▶ Adicionar uma nova chave estrangeira a uma tabela existente sem garantir que todos os valores já presentes na tabela tenham correspondência na tabela referenciada. Por exemplo:

---

```
1 ALTER TABLE professores ADD CONSTRAINT fk_departamento FOREIGN KEY (
    departamento_id) REFERENCES departamentos(departamento_id);
```

---

Se houver valores em professores.departamento\_id que não existam em departamentos.departamento\_id, essa alteração falhará.



# Introdução à Cláusula SELECT

## Fundamentos do SELECT

A cláusula SELECT é usada para escolher e exibir dados de uma ou mais tabelas. É a operação mais básica e fundamental em SQL para consulta de dados.

# Cláusula FROM

## Usando a Cláusula FROM

A cláusula FROM especifica a(s) tabela(s) de onde os dados serão selecionados. É onde definimos a origem dos dados para a consulta SELECT.

## Exemplo de Uso do FROM

```
1 SELECT nome FROM alunos;
```



## Filtragem com WHERE

- ▶ WHERE é usado para filtrar registros.
- ▶ Permite a especificação de condições.

### Exemplo de Uso do WHERE

```
1 SELECT * FROM alunos WHERE curso_id = 3;
```



## Atualizar Tabela Alunos

```
1 update alunos set email = 'eva@ufpa.br' where aluno_id = 1;
2 update alunos set email = 'ana@ufpa.br' where aluno_id = 2;
3 update alunos set email = 'ide@ufpa.br' where aluno_id = 3;
```

# Ordenação com ORDER BY

- ▶ ORDER BY é usado para ordenar os resultados de uma consulta SQL.

## Exemplo de Uso do ORDER BY

```
1 SELECT nome, email FROM alunos ORDER BY nome;
```



## Ordenação com ORDER BY

- ▶ ORDER BY é usado para ordenar os resultados de uma consulta SQL.
- ▶ DESC ou ASC

### Exemplo de Uso do ORDER BY

---

```
1 SELECT nome, email FROM alunos ORDER BY nome DESC;
```

---



## Ordenação com ORDER BY

- ▶ Dupla ordenação
  - ▶ Primeiro pelo atributo **nome** e as tuplas com valor igual para atributo **nome** ficam ordenadas pelo atributo **email**

### Exemplo de Uso do ORDER BY

```
1 SELECT nome, email FROM alunos ORDER BY nome, email;
```



## Exercício para os alunos

1. Adicione uma nova coluna na tabela professores:
  - ▶ SQL: ALTER TABLE professores ADD COLUMN salario DECIMAL(8,2);
2. Atribua salários aos professores (exemplo com valor aleatório):
  - ▶ SQL: UPDATE professores SET salario = CAST((RANDOM() \* 12000 + 3000) AS DECIMAL(8,2));
3. Ordene os professores com base no salário (ASC ou DESC).

# CLÁUSULA WHERE (1/5)

- ▶ A SQL usa conectores lógicos **and**, **or** e **not** na cláusula where.
- ▶ Os operandos dos conectivos lógicos podem ser expressões envolvendo operadores de comparação <, <=, >, >=, = e <> (diferente de).

## CLÁUSULA WHERE(2/5)

- ▶ Encontrar professores com salários maior que X.

### Exemplo de Uso Where com Operadores de comparação

```
1 SELECT nome, salario FROM professores where (salario>3000);
2 SELECT nome, salario FROM professores where (salario>7000);
3 SELECT nome, salario FROM professores where (salario>10000);
```



## CLÁUSULA WHERE(3/5)

- ▶ Encontrar professores com salários que estejam no intervalo fechado X até Y.

Exemplo de Uso Where com conectores lógicos

```
1 SELECT nome, salario FROM professores where (salario>=7000) and (salario<=10000);
```

## CLÁUSULA WHERE(4/5)

- ▶ A SQL possui o operador de comparação **between** para simplificar a cláusula where que especifica que um atributo possa ter um valor maior ou igual a algum valor e menor ou igual a algum outro valor.

### Exemplo de Uso do comando BETWEEN

```
1 SELECT nome, salario FROM professores where salario between 7000 and  
10000;
```



## CLÁUSULA WHERE(5/5)

- A SQL possui o operador de teste de pertinência **IN** que verifica se um dado valor é membro (ou pertence) a um conjunto de valores. A cláusula IN no SQL é equivalente ao operador lógico **OR**

### Exemplo de Uso do comando BETWEEN

```
1 SELECT nome, salario FROM professores where professores.  
departamento_id IN (1,3)
```



## Vamos inserir algumas informação

- ▶ Add coluna data de data\_admissao na tabela professor

```
1 ALTER TABLE professores ADD COLUMN data_admissao DATE;
```

```
1 UPDATE professores SET data_admissao = '2024-01-01' WHERE professor_id  
= 1;
```



## Vamos inserir algumas informação

- ▶ Add coluna data de data\_admissao na tabela professor

```
1 ALTER TABLE professores ADD COLUMN data_admissao DATE;
```

```
1 UPDATE professores
2 SET data_admissao =
3   '2011-01-01'::DATE +
4   (RANDOM() * ('2024-01-01'::DATE - '2011-01-01'::DATE))::INTEGER;
```

## Trabalhando com datas (1/3)

- Ao realizar **select** com **Where** no contexto de datas, com objetivo de filtrar com base em um intervalo de tempo, use **BETWEEN**

```
1 select * from professores where data_admissao between ('01-01-2015')
      and ('01-01-2024');

2

3 select * from professores where data_admissao between ('2015-01-01')
      and ('2024-01-01');
```

professor_id	nome	departamento_id	salario	data_admissao
2	Prof. Carlos	2	11989.70	2023-09-11
1	Prof. Elton	1	11434.85	2023-10-01

Tabela: Resposta da Consulta



## Trabalhando com datas (2/3)

- ▶ Selecionar Professores Admitidos ANTES de uma Data Específica

```
1 select * from professores where data_admissao < '01-01-2015'
```



## Trabalhando com datas (3/3)

- ▶ Selecionar Professores Admitidos DEPOIS de uma Data Específica

```
1 select * from professores where data_admissao > '01-01-2015'
```



## Produto Cartesiano

- ▶ O produto cartesiano de duas tabelas é um conjunto **de todas as combinações possíveis** de registros entre elas.
- ▶ Ocorre quando duas tabelas são listadas no FROM sem condição específica de junção.
- ▶ Exemplo: `SELECT * FROM professores, departamentos;`
- ▶ Resulta em um número de linhas igual ao produto do número de linhas de cada tabela.



## Problemas do Produto Cartesiano

- ▶ Geralmente não é desejável ou útil, pois pode resultar em uma grande quantidade de dados não relacionados.
- ▶ Pode causar problemas de desempenho em grandes bases de dados.
- ▶ Deve ser usado com cautela e apenas quando necessário.

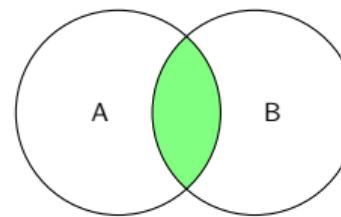
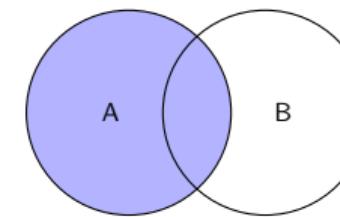
## Junção de Duas Tabelas

- ▶ A junção é o processo de combinar colunas de duas ou mais tabelas com base em uma relação entre elas.
- ▶ Utiliza-se uma condição de junção para especificar como as tabelas devem ser combinadas.

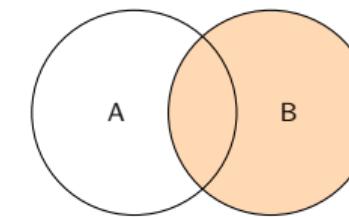


# Tipos de JOIN (Junções)

JOINS são usados para combinar linhas de duas ou mais tabelas baseados em uma coluna relacionada.

**INNER JOIN**Interseção ( $A \cap B$ )**LEFT JOIN**

Todo A + Match B

**RIGHT JOIN**

Todo B + Match A

## Sintaxe de JOIN (Padrão ANSI)

Evite usar a vírgula no FROM (produto cartesiano implícito). Use a sintaxe explícita JOIN.

```
1 -- Objetivo: Listar Nome do Professor e Nome do Departamento
2 SELECT
3     p.nome AS Professor,
4     d.nome AS Departamento
5 FROM professores as p
6 INNER JOIN departamentos as d ON p.departamento_id = d.departamento_id
;
```

- ▶ **INNER JOIN:** Traz apenas se houver vínculo. Se um professor não tem departamento (NULL), ele não aparece.
- ▶ **LEFT JOIN:** Traz todos os professores, mesmo os sem departamento.



## Sintaxe de JOIN (Padrão ANSI)

Evite usar a vírgula no FROM (produto cartesiano implícito). Use a sintaxe explícita JOIN.

```
1 INSERT INTO professores (nome) VALUES ('Prof. Allan');
2 SELECT
3     p.nome AS Professor,
4     d.nome AS Departamento
5 FROM professores as p
6 LEFT JOIN departamentos as d ON p.departamento_id = d.departamento_id;
```

- ▶ **LEFT JOIN:** Traz todos os professores, mesmo os sem departamento.



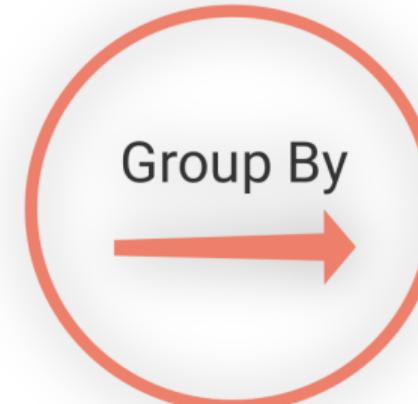
# Introdução ao GROUP BY

- ▶ GROUP BY é usado para agrupar linhas que têm os mesmos valores em colunas específicas.
  - ▶ O atributo ou atributos fornecidos na cláusula GROUP BY são usados para formar grupos.
  - ▶ Tuplas com os mesmos valores em todos os atributos da cláusula group by são colocadas em um grupo.
- ▶ Comumente usado com funções agregadas (COUNT, SUM, AVG, MIN, MAX).
- ▶ Exemplo: Agrupar professores por departamento.



# Introdução ao GROUP BY

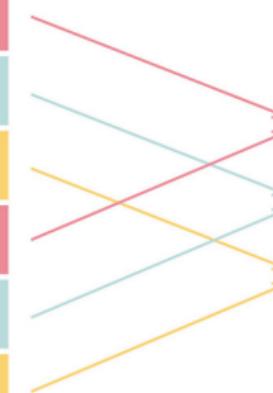
day	category	value
2020-03-01	Red	62
2020-12-01	Blue	63
2020-01-01	Green	55
2020-04-01	Blue	41
2020-06-01	Red	60
2020-10-01	Blue	60
2020-07-01	Green	41
2020-11-01	Green	60
2020-08-01	Red	46



day	category	value
2020-04-01	Blue	63
2020-10-01	Blue	41
2020-12-01	Blue	60
2020-01-01	Green	55
2020-07-01	Green	41
2020-11-01	Green	60
2020-03-01	Red	62
2020-06-01	Red	60
2020-08-01	Red	46

# Introdução ao GROUP BY

title	genre	price
book 1	adventure	11.90
book 2	fantasy	8.49
book 3	romance	9.99
book 4	adventure	9.99
book 5	fantasy	7.99
book 6	romance	5.88



genre	avg_price
adventure	$(11.90 + 9.99)/2$ 10.945
fantasy	$(8.49 + 7.99)/2$ 8.24
romance	$(9.99 + 5.88)/2$ 7.935

## Agrupando e Contando

- Exemplo: Contar o número de professores em cada departamento.

```
1   SELECT departamento_id, COUNT(*) FROM professores GROUP BY  
2     departamento_id;
```

- Mostra quantos professores existem em cada departamento.

## Agrupando e Calculando Médias

- ▶ Exemplo: Calcular a média salarial por departamento.

```
1  SELECT departamento_id, AVG(salario) FROM professores GROUP BY  
2    departamento_id;
```

- ▶ Fornece a média salarial para cada departamento.



# Funções de Agregação

Calculam um único valor a partir de um conjunto de linhas.

- ▶ COUNT(\*): Conta linhas.
- ▶ SUM(col): Soma valores.
- ▶ AVG(col): Média.
- ▶ MAX(col): Maior valor.
- ▶ MIN(col): Menor valor.

```
1 -- Total da folha salarial
2 SELECT SUM(salario) FROM
      professores;
3
4 -- Maior salario registrado
5 SELECT MAX(salario) FROM
      professores;
```



# Introdução à Cláusula HAVING

## O que é a Cláusula HAVING?

HAVING é usada em SQL para especificar um critério de filtragem que é aplicado aos grupos criados pela cláusula GROUP BY. Diferentemente do WHERE, que filtra linhas, HAVING filtra grupos.

# Diferença entre WHERE e HAVING

## WHERE vs. HAVING

- ▶ WHERE é usado para filtrar linhas antes da agregação.
- ▶ HAVING é usado para filtrar grupos após a agregação.



# Uso Básico do HAVING

## Exemplo de Uso do HAVING

```
SELECT departamento, COUNT(*)  
FROM funcionarios  
GROUP BY departamento  
HAVING COUNT(*) > 5;
```

*Este exemplo seleciona departamentos com mais de 5 funcionários.*



## Uso Básico do HAVING

### Exemplo de Uso do HAVING

```
UPDATE professores SET departamento_id = 1 where nome = 'Prof. Allan';
```

---

```
SELECT c.nome, count(p) FROM professores as p
INNER JOIN departamentos as d
ON d.departamento_id = p.departamento_id
INNER JOIN cursos c
ON c.departamento_id = d.departamento_id
GROUP BY c.nome
HAVING COUNT(p.professor_id)>1
```

*Este exemplo seleciona os cursos com 2 professores ou mais*



# HAVING com Funções de Agregação

## Exemplo de HAVING com Funções de Agregação

```
SELECT vendedor_id, SUM(valor_venda)
FROM vendas
GROUP BY vendedor_id
HAVING SUM(valor_venda) > 10000;
```

*Este exemplo mostra vendedores cujas vendas totais excedem 10.000.*



## HAVING em Conjunto com WHERE

### Exemplo de HAVING com WHERE

```
SELECT vendedor_id, SUM(valor_venda)
FROM vendas
WHERE data_venda > '2023-01-01'
GROUP BY vendedor_id
HAVING SUM(valor_venda) > 5000;
```

*Seleciona vendedores com vendas totais acima de 5.000 após 1º de janeiro de 2023.*



## Combinação de HAVING com OUTRAS Cláusulas

### HAVING com ORDER BY e LIMIT

```
SELECT departamento, AVG(salario)
FROM funcionarios
GROUP BY departamento
HAVING AVG(salario) > 3000
ORDER BY AVG(salario) DESC
LIMIT 3;
```

*Mostra os 3 departamentos com a maior média salarial acima de 3.000.*



## Introdução ao INNER JOIN com GROUP BY

- ▶ INNER JOIN é usado para combinar registros de duas tabelas com base em uma condição de correspondência.
- ▶ GROUP BY é utilizado para agregar resultados e aplicar funções agregadas a grupos de registros.
- ▶ A combinação dessas duas cláusulas permite análises detalhadas e complexas.

## Exemplo 1: Contagem de Professores por Departamento

- Objetivo: Contar quantos professores existem em cada departamento.

```
1  SELECT departamentos.nome , COUNT(professores.professor_id) FROM  
professores INNER JOIN departamentos ON professores.  
departamento_id = departamentos.departamento_id GROUP BY  
departamentos.nome;
```

2

## Exercício 1: Média salarial por Departamento

- ▶ Objetivo: Verificar média salarial dos professores em cada departamento.

## Exemplo 2: Número de Cursos por Departamento

- Objetivo: Determinar o número de cursos oferecidos por cada departamento.

```
1   SELECT departamentos.nome, COUNT(cursos.curso_id) FROM cursos  
2   INNER JOIN departamentos ON cursos.departamento_id =  
3   departamentos.departamento_id GROUP BY departamentos.nome;
```



## Exercício 2: Número de Alunos por Curso

- ▶ Objetivo: Determinar o número de Alunos oferecidos por cada Curso.

## Exemplo 3: Nome dos professores por curso

- Objetivo: Mostrar Nome dos professores em seus respectivos cursos.

```
1  Select p.nome,c.nome from departamentos as d  
2    join professores as p on d.departamento_id = p.departamento_id  
3    join cursos c on c.departamento_id = d.departamento_id;
```



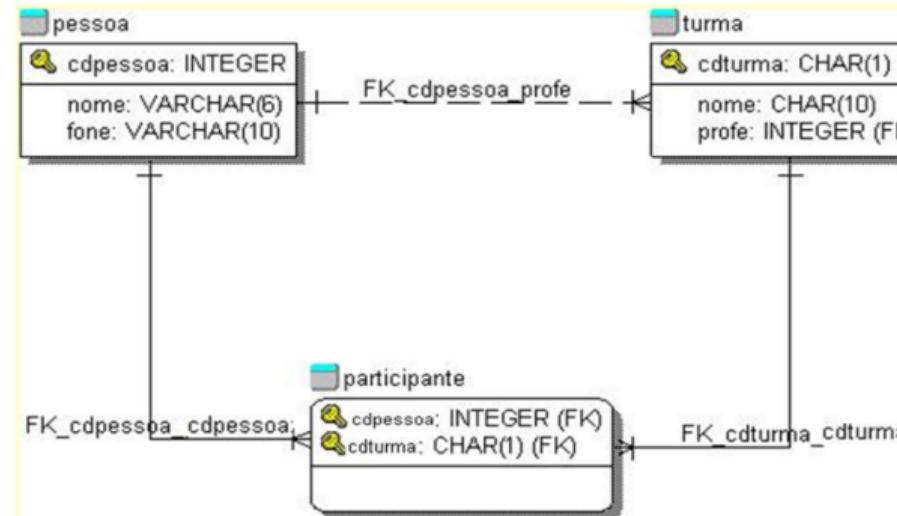
## Exercício

- ▶ Objetivo: Mostrar Nome dos alunos em seus respectivos Departamentos.

## Resumo e Boas Práticas

- ▶ **Modelagem:** Dedique tempo definindo PKs e FKs corretamente.
- ▶ **Performance:** Evite SELECT \* em tabelas grandes; selecione apenas as colunas necessárias.
- ▶ **Joins:** Prefira INNER JOIN quando não precisar de dados nulos; é geralmente mais performático.
- ▶ **Nulls:** Funções como SUM e AVG ignoram NULLs automaticamente, mas COUNT(\*) conta linhas com null.





Pessoa		
cd Pessoa	nome	fone
1	Obilac	260088
2	Silva	282677
3	Cabral	260088
4	Lobato	174590
5	Mateus	

Turma		
cd Turma	nome	prof
A	Volei	4
B	Karate	4
C	Natação	2

pessoa	turma
1	A
3	A
1	B
1	C
2	C

## Questões

1. Liste todos os nomes e telefones das pessoas cadastradas.
2. Quantos participantes existem em cada turma?
3. Liste os nomes dos professores junto com os nomes das turmas que eles ministram.
4. Quem são os participantes de cada turma?
5. Quantas turmas cada professor está ministrando?
6. Quais turmas têm mais de 2 participantes?
7. Quais professores estão ministrando 2 ou mais turmas?



## Referências I

- ❑ Elmasri, R., & Navathe, S. (2010). *Sistemas de banco de dados*. Pearson Addison Wesley.
- ❑ Heuser, C. A. (2009). *Projeto de banco de dados*. Bookman Editora.
- ❑ Documentação Oficial do PostgreSQL. <https://www.postgresql.org/docs/>