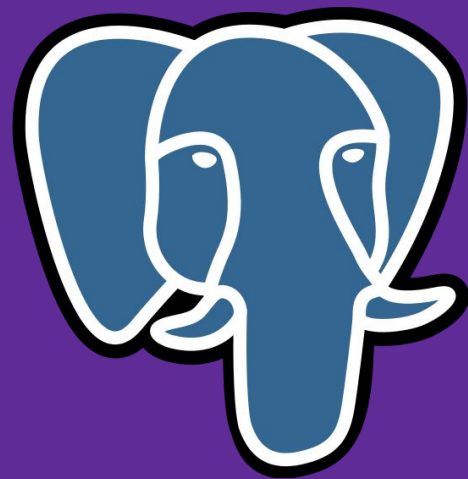


# PostgreSQL



# O Que é um Banco de Dados Relacional?

Um banco de dados relacional (BDR) é um sistema de armazenamento de dados que organiza informações em tabelas, permitindo uma estrutura clara e relacionamentos entre diferentes conjuntos de dados. A base dessa tecnologia é o modelo relacional, que utiliza tabelas (também chamadas de relações) para armazenar dados. Cada tabela contém linhas (tuplas) e colunas (atributos). As tabelas podem se relacionar umas com as outras por meio de chaves primárias e estrangeiras, permitindo consultas complexas e manipulação de dados eficiente.

## Elementos Básicos dos Bancos de Dados Relacionais

- **Tabelas:** Coleções de dados organizados em linhas e colunas.
- **Linhas:** Também chamadas de registros, representam uma única entrada em uma tabela.
- **Colunas:** Também chamadas de campos, representam um atributo da tabela.
- **Chave Primária:** Um identificador único para cada linha em uma tabela.
- **Chave Estrangeira:** Um campo em uma tabela que cria um vínculo com a chave primária de outra tabela, estabelecendo um relacionamento entre as tabelas.

# O Que é Cardinalidade em um Banco de Dados Relacional

A cardinalidade em um banco de dados relacional refere-se à natureza das relações entre duas tabelas. Ela define quantas instâncias de uma entidade podem ou devem estar relacionadas a quantas instâncias de outra entidade. A cardinalidade é uma parte fundamental do design de banco de dados, pois influencia a estrutura do banco e como as tabelas interagem entre si.

## Tipos de Cardinalidade

Existem três tipos principais de cardinalidade em bancos de dados relacionais:

1. **Um-para-Um (1:1)**
2. **Um-para-Muitos (1:N)**
3. **Muitos-para-Muitos (N:N)**

# Relacionamentos em Bancos de Dados Relacionais

Os relacionamentos entre tabelas em um banco de dados relacional são fundamentais para a organização e integridade dos dados. Existem três tipos principais de relacionamentos: um-para-um, um-para-muitos e muitos-para-muitos. Vamos explorar cada um desses relacionamentos em detalhes, com exemplos práticos usando PostgreSQL.

## 1. Relacionamento Um-para-Um (1:1)

**Definição:** Um registro em uma tabela está relacionado a, no máximo, um registro em outra tabela.

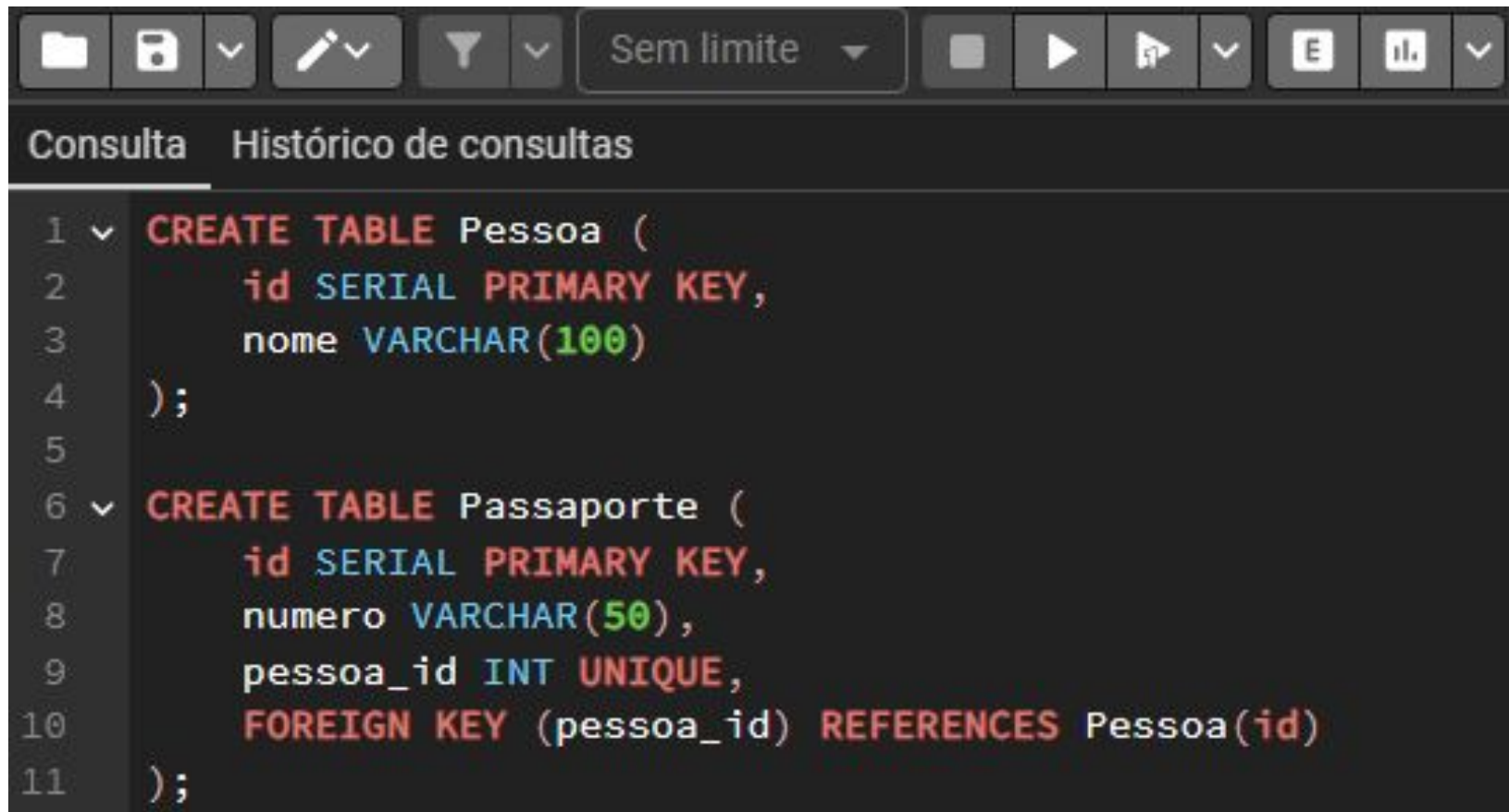
**Exemplo Prático:**

- **Tabelas:** **Pessoa** e **Passaporte**
- **Relacionamento:** Cada pessoa pode ter um único passaporte, e cada passaporte pertence a uma única pessoa.

**Como Implementar:**

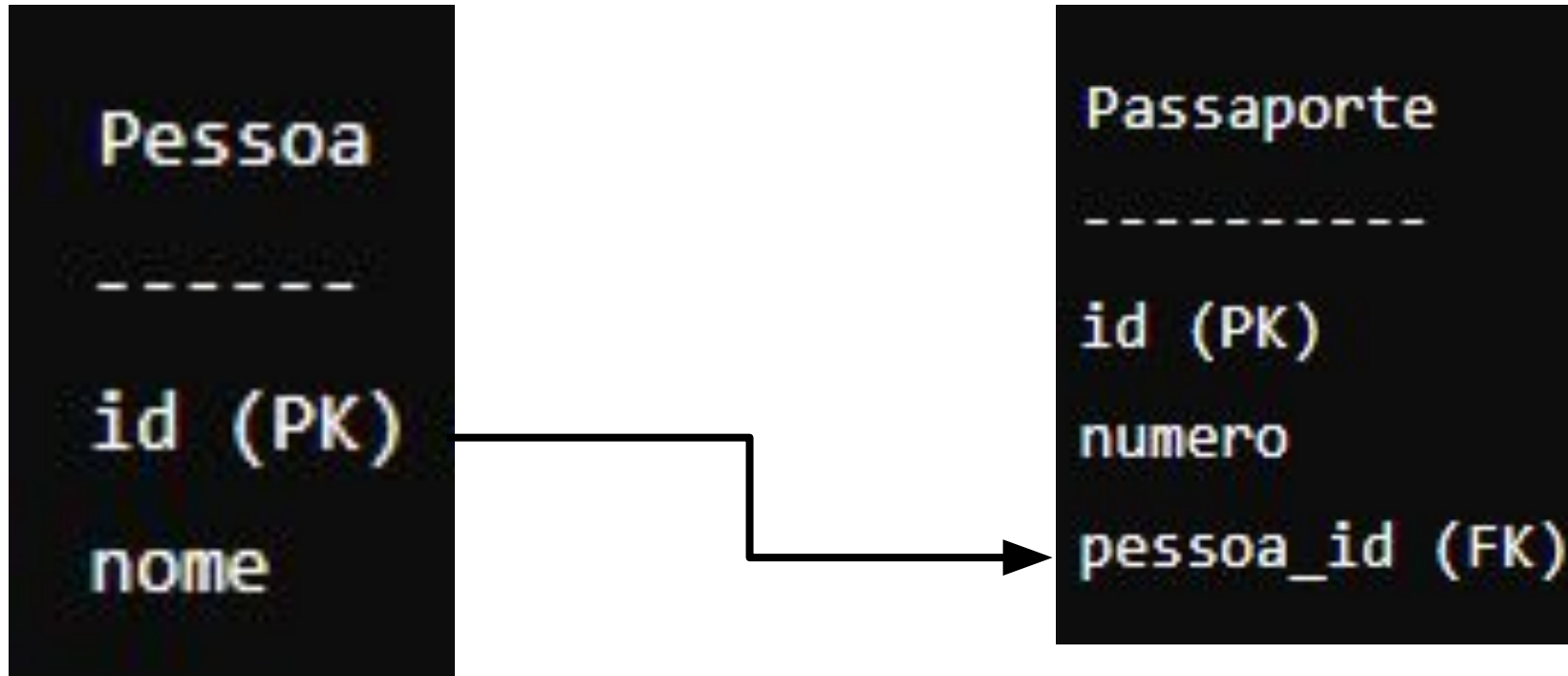
- Crie as tabelas **Pessoa** e **Passaporte**.
- Adicione uma chave estrangeira em uma das tabelas que referencia a chave primária da outra tabela.

# Relacionamentos em Bancos de Dados Relacionais



```
1  CREATE TABLE Pessoa (  
2      id SERIAL PRIMARY KEY,  
3      nome VARCHAR(100)  
4  );  
5  
6  CREATE TABLE Passaporte (  
7      id SERIAL PRIMARY KEY,  
8      numero VARCHAR(50),  
9      pessoa_id INT UNIQUE,  
10     FOREIGN KEY (pessoa_id) REFERENCES Pessoa(id)  
11 );
```

## Explicação Visual Um-para-Um (1:1)



# Relacionamentos em Bancos de Dados Relacionais

## 2. Relacionamento Um-para-Muitos (1:N)

**Definição:** Um registro em uma tabela pode estar relacionado a muitos registros em outra tabela, mas um registro na segunda tabela está relacionado a, no máximo, um registro na primeira tabela.

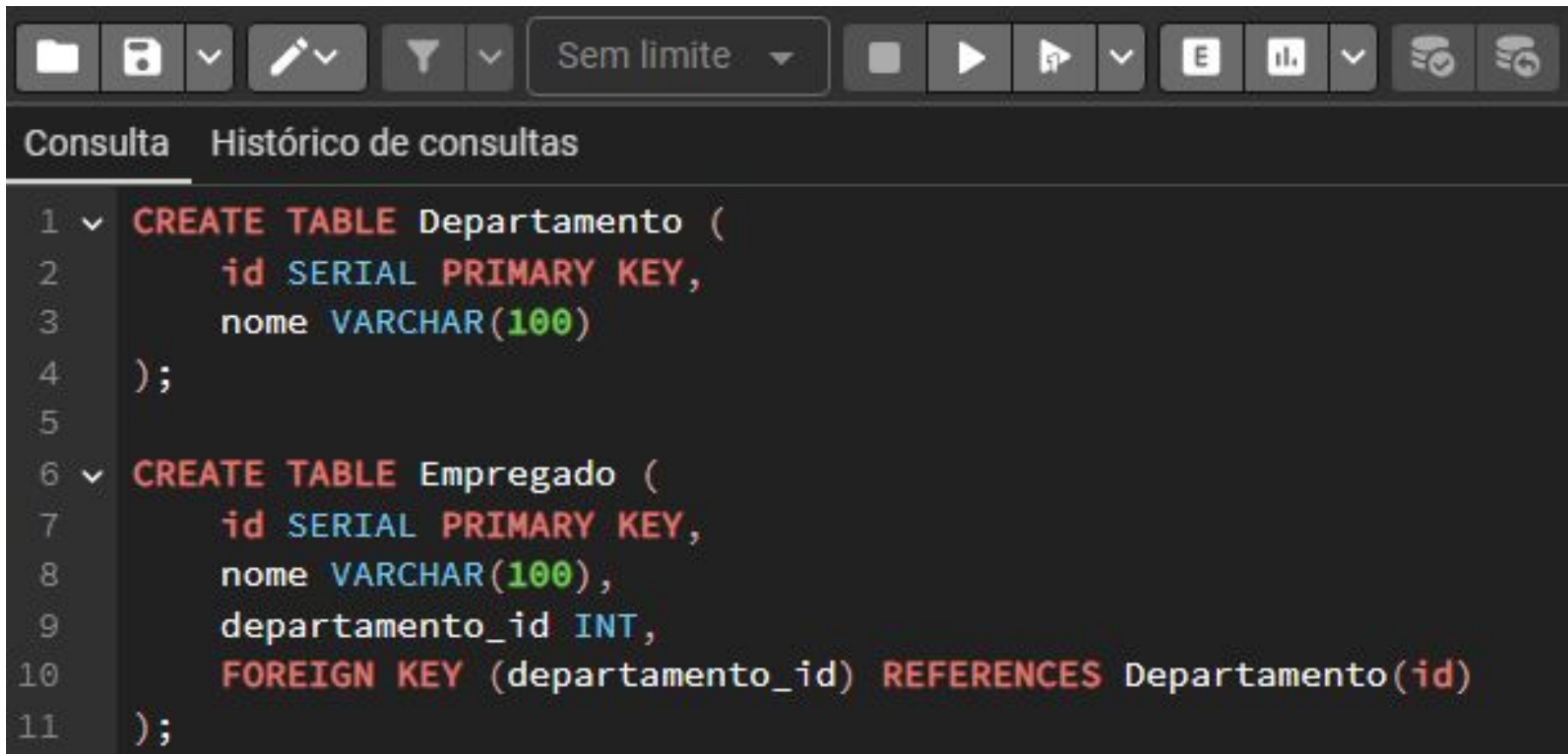
### Exemplo Prático:

- **Tabelas:** Departamento e Empregado
- **Relacionamento:** Um departamento pode ter muitos empregados, mas cada empregado pertence a apenas um departamento.

### Como Implementar:

- Crie as tabelas Departamento e Empregado.
- Adicione uma chave estrangeira em Empregado que referencia a chave primária de Departamento.

# Relacionamentos em Bancos de Dados Relacionais

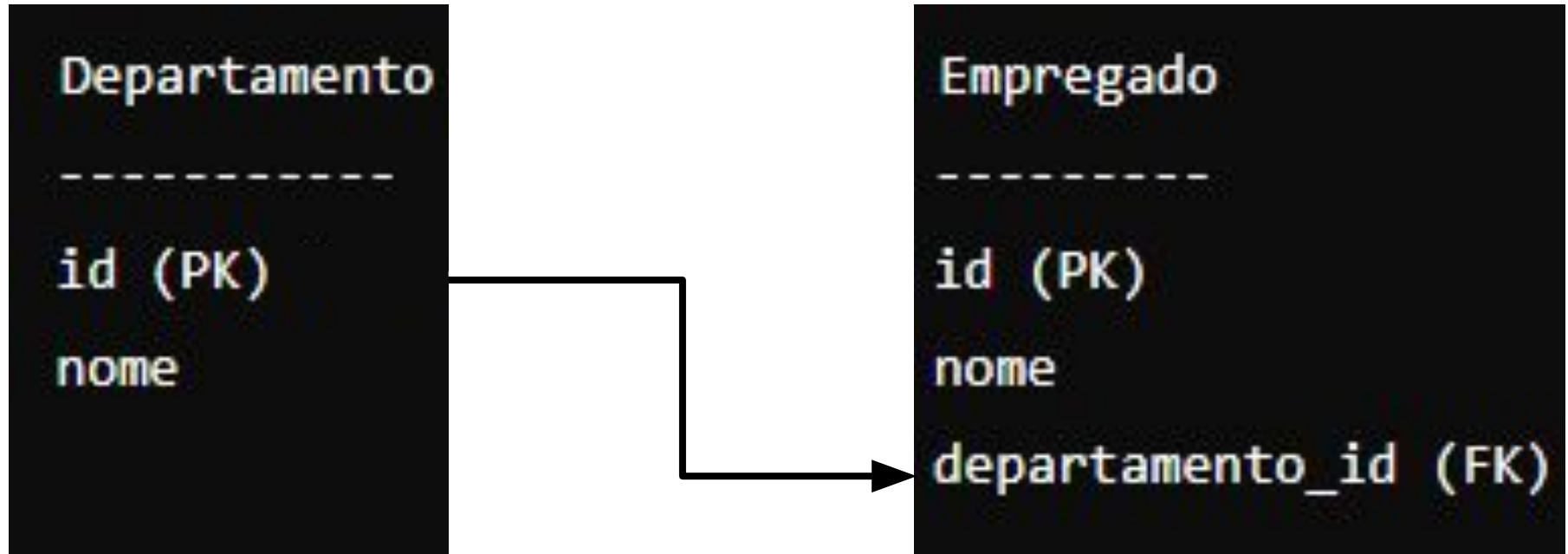


The image shows a database query editor interface. At the top, there is a toolbar with various icons for file operations, editing, and execution. Below the toolbar, there are two tabs: 'Consulta' (Query) and 'Histórico de consultas' (Query History). The 'Consulta' tab is active, displaying a SQL script. The script consists of two lines of code, each preceded by a line number and a dropdown arrow icon. The first line (1) is 'CREATE TABLE Departamento (' and the second line (2) is 'id SERIAL PRIMARY KEY, nome VARCHAR(100);'. The third line (3) is ');'. The fourth line (4) is 'CREATE TABLE Empregado (' and the fifth line (5) is 'id SERIAL PRIMARY KEY, nome VARCHAR(100), departamento\_id INT, FOREIGN KEY (departamento\_id) REFERENCES Departamento(id);'. The sixth line (6) is ');'.

```
1  CREATE TABLE Departamento (  
2      id SERIAL PRIMARY KEY,  
3      nome VARCHAR(100)  
4  );  
5  
6  CREATE TABLE Empregado (  
7      id SERIAL PRIMARY KEY,  
8      nome VARCHAR(100),  
9      departamento_id INT,  
10     FOREIGN KEY (departamento_id) REFERENCES Departamento(id)  
11 );
```



# Explicação Visual Um-para-Um (1:1)



# Relacionamentos em Bancos de Dados Relacionais

## 3. Relacionamento Muitos-para-Muitos (N:N)

**Definição:** Um registro em uma tabela pode estar relacionado a muitos registros em outra tabela, e vice-versa.

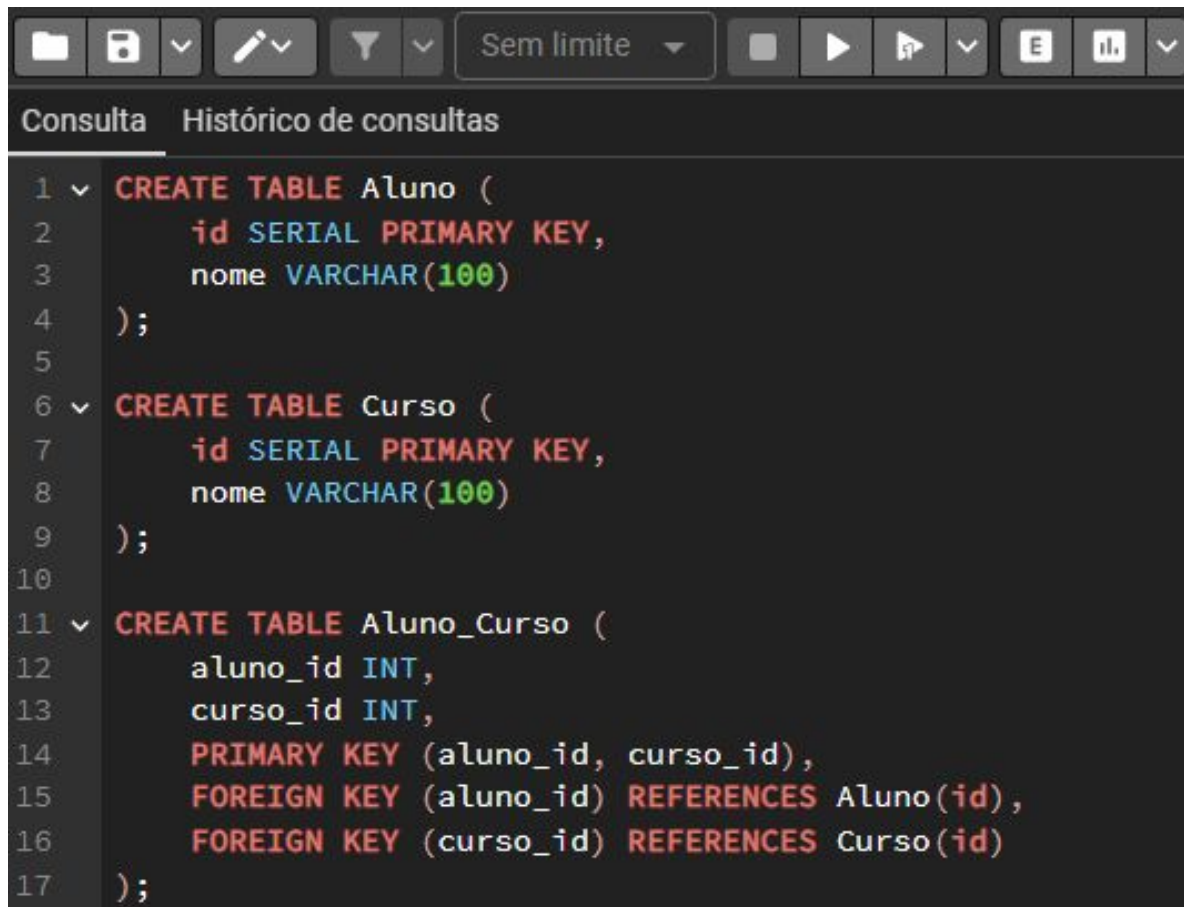
### Exemplo Prático:

- **Tabelas:** `Aluno` e `Curso`
- **Relacionamento:** Um aluno pode estar matriculado em muitos cursos, e um curso pode ter muitos alunos matriculados.

### Como Implementar:

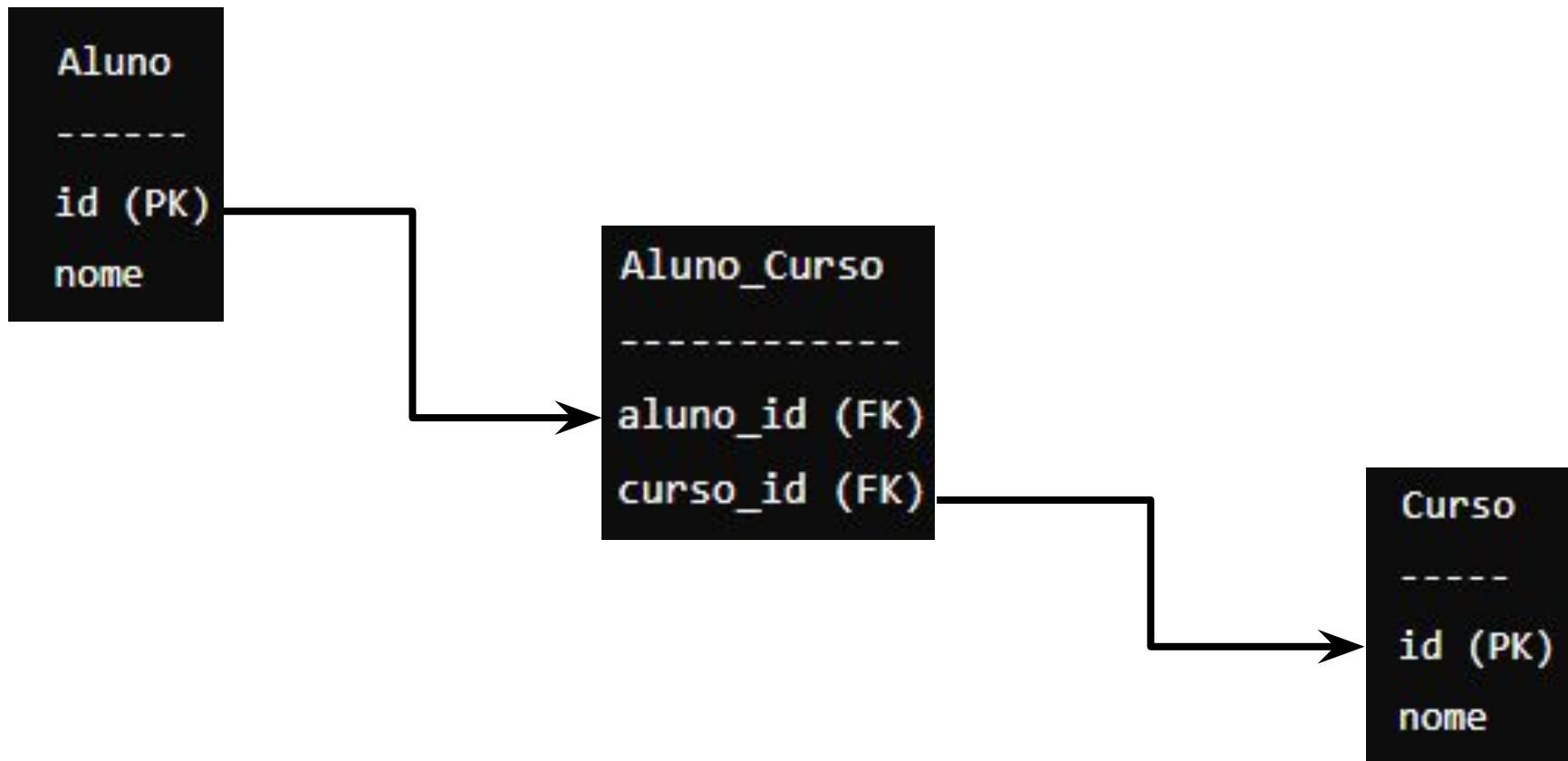
- Crie as tabelas `Aluno` e `Curso`.
- Crie uma tabela intermediária `Aluno_Curso` que contém chaves estrangeiras referenciando as chaves primárias de `Aluno` e `Curso`.

# Relacionamentos em Bancos de Dados Relacionais



```
1  CREATE TABLE Aluno (  
2      id SERIAL PRIMARY KEY,  
3      nome VARCHAR(100)  
4  );  
5  
6  CREATE TABLE Curso (  
7      id SERIAL PRIMARY KEY,  
8      nome VARCHAR(100)  
9  );  
10  
11 CREATE TABLE Aluno_Curso (  
12     aluno_id INT,  
13     curso_id INT,  
14     PRIMARY KEY (aluno_id, curso_id),  
15     FOREIGN KEY (aluno_id) REFERENCES Aluno(id),  
16     FOREIGN KEY (curso_id) REFERENCES Curso(id)  
17 );
```

# Explicação Visual Um-para-Um (1:1)



**ESTADO CIVIL**  
**EM UM RELACIONAMENTO SÉRIO**  
**COM NOÉ...**



**NOÉ DA SUA CONTA!**

# Introdução às Categorias de SQL: DDL, DML, DCL e TCL

SQL (Structured Query Language) é a linguagem padrão para gerenciar e manipular bancos de dados relacionais. SQL é dividido em várias sub categorias de comandos, cada uma focada em aspectos diferentes do gerenciamento de dados e da estrutura do banco de dados. As principais categorias são:

- **DDL (Data Definition Language)** - Linguagem de Definição de Dados
- **DML (Data Manipulation Language)** - Linguagem de Manipulação de Dados
- **DCL (Data Control Language)** - Linguagem de Controle de Dados
- **TCL (Transaction Control Language)** - Linguagem de Controle de Transações

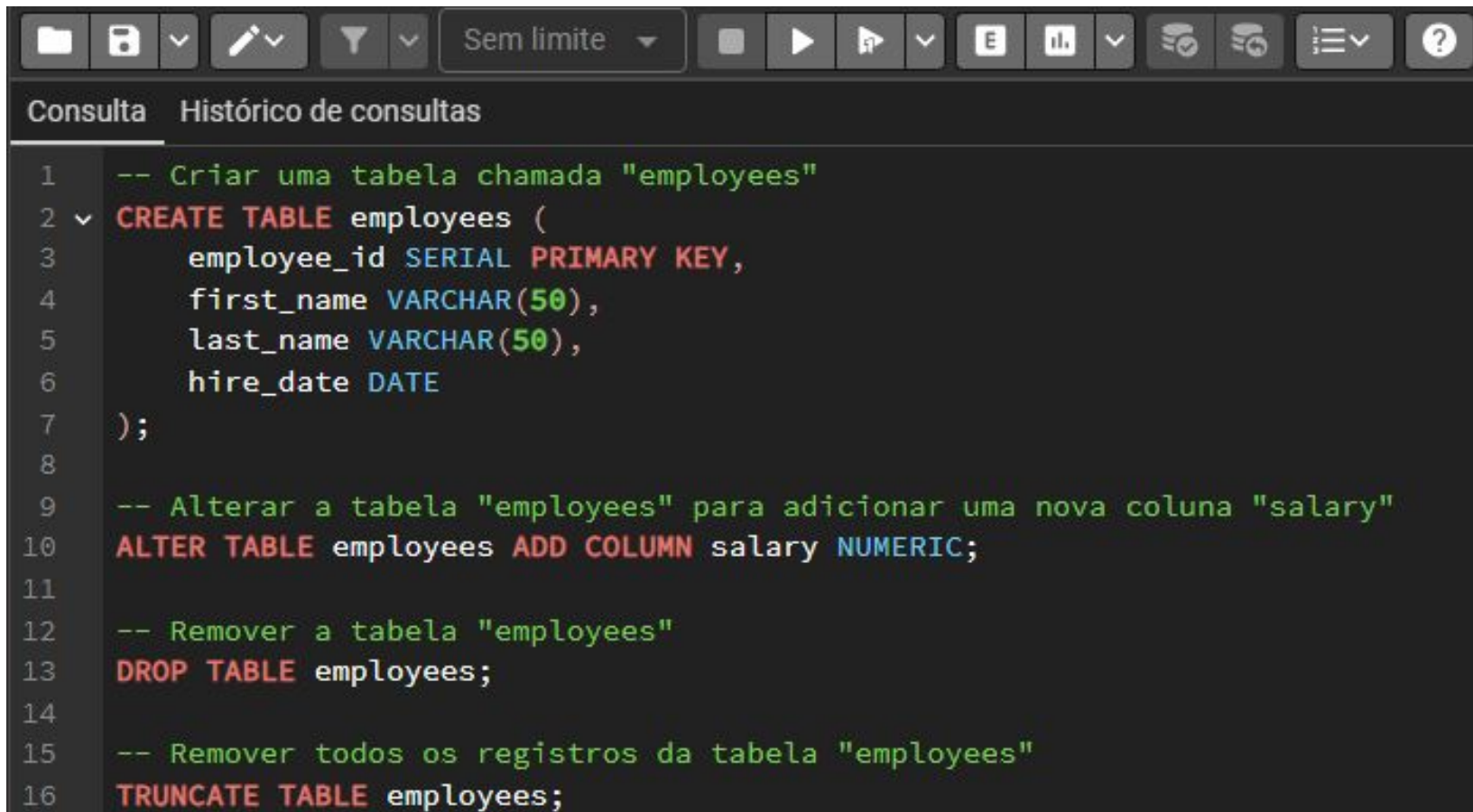
# 1. DDL (Data Definition Language)

**Função:** DDL é usada para definir e modificar a estrutura dos objetos do banco de dados, como tabelas, índices, e esquemas.

## Comandos Principais:

- **CREATE:** Cria novos objetos no banco de dados, como tabelas, índices, etc.
- **ALTER:** Modifica a estrutura de objetos existentes.
- **DROP:** Remove objetos do banco de dados.
- **TRUNCATE:** Remove todos os registros de uma tabela, sem registrar cada linha removida individualmente.

# 1. DDL (Data Definition Language)



The image shows a screenshot of a SQL IDE interface. At the top, there is a toolbar with various icons for file operations, editing, and execution. Below the toolbar, there are two tabs: "Consulta" (selected) and "Histórico de consultas". The main area displays a list of SQL queries, each preceded by a line number. The queries are as follows:

```
1  -- Criar uma tabela chamada "employees"
2  CREATE TABLE employees (
3      employee_id SERIAL PRIMARY KEY,
4      first_name VARCHAR(50),
5      last_name VARCHAR(50),
6      hire_date DATE
7  );
8
9  -- Alterar a tabela "employees" para adicionar uma nova coluna "salary"
10 ALTER TABLE employees ADD COLUMN salary NUMERIC;
11
12 -- Remover a tabela "employees"
13 DROP TABLE employees;
14
15 -- Remover todos os registros da tabela "employees"
16 TRUNCATE TABLE employees;
```



## 2. DML (Data Manipulation Language)

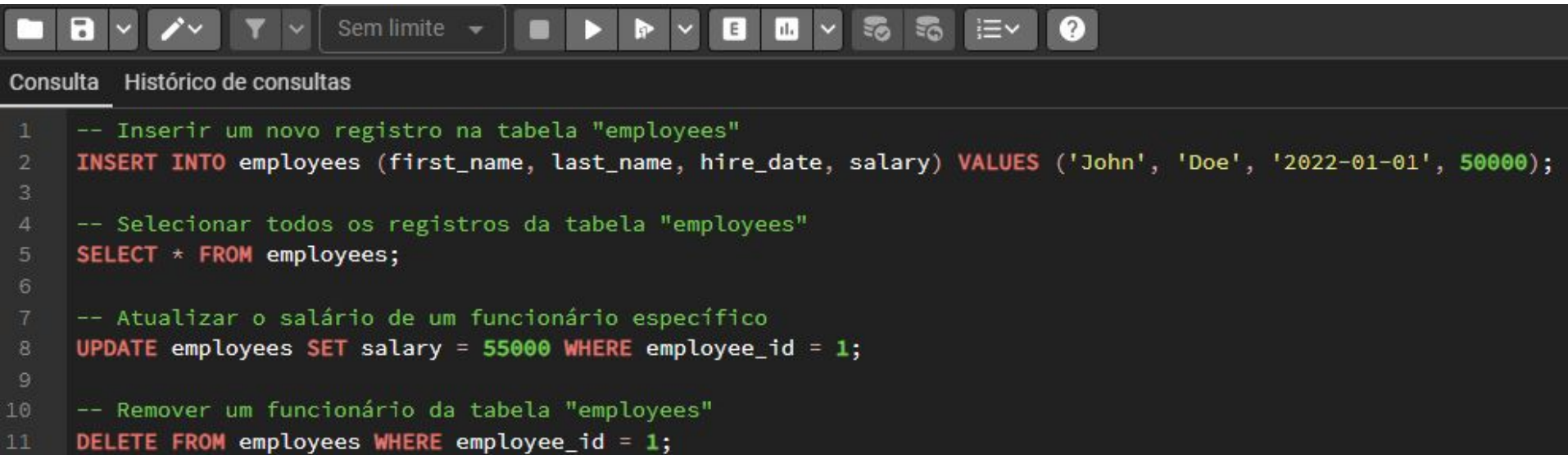
### 2. DML (Data Manipulation Language)

**Função:** DML é usada para manipular os dados dentro dos objetos do banco de dados.

#### Comandos Principais:

- **SELECT:** Recupera dados das tabelas.
- **INSERT:** Insere novos dados nas tabelas.
- **UPDATE:** Modifica dados existentes nas tabelas.
- **DELETE:** Remove dados das tabelas.

## 2. DML (Data Manipulation Language)



The image shows a screenshot of a SQL IDE interface. At the top, there is a toolbar with various icons for file operations, editing, and execution. Below the toolbar, there are two tabs: "Consulta" (active) and "Histórico de consultas". The main area displays a SQL script with line numbers on the left. The script contains four DML statements: an INSERT, a SELECT, an UPDATE, and a DELETE, each preceded by a comment in Portuguese.

```
1  -- Inserir um novo registro na tabela "employees"
2  INSERT INTO employees (first_name, last_name, hire_date, salary) VALUES ('John', 'Doe', '2022-01-01', 50000);
3
4  -- Selecionar todos os registros da tabela "employees"
5  SELECT * FROM employees;
6
7  -- Atualizar o salário de um funcionário específico
8  UPDATE employees SET salary = 55000 WHERE employee_id = 1;
9
10 -- Remover um funcionário da tabela "employees"
11 DELETE FROM employees WHERE employee_id = 1;
```

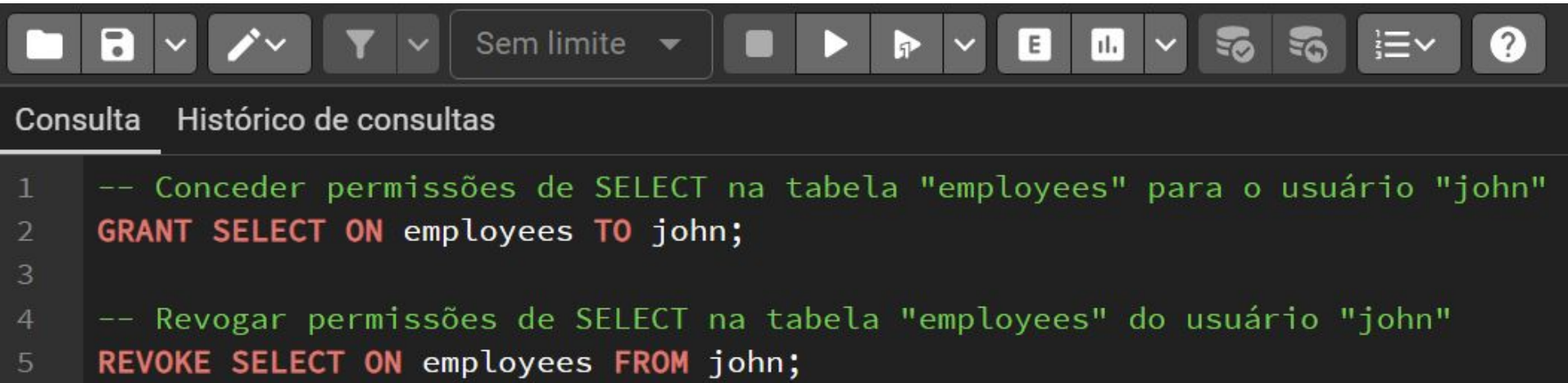
### 3. DCL (Data Control Language)

**Função:** DCL é usada para controlar o acesso aos dados no banco de dados.

**Comandos Principais:**

- **GRANT:** Concede permissões a usuários ou papéis.
- **REVOKE:** Revoga permissões de usuários ou papéis.

### 3. DCL (Data Control Language)



The image shows a screenshot of a SQL IDE interface. At the top, there is a toolbar with various icons for file operations, editing, and execution. Below the toolbar, there are two tabs: "Consulta" (active) and "Histórico de consultas". The main area displays SQL code for Data Control Language (DCL) operations. The code is color-coded: green for comments, red for keywords, and black for identifiers and literals.

```
1  -- Conceder permissões de SELECT na tabela "employees" para o usuário "john"
2  GRANT SELECT ON employees TO john;
3
4  -- Revogar permissões de SELECT na tabela "employees" do usuário "john"
5  REVOKE SELECT ON employees FROM john;
```

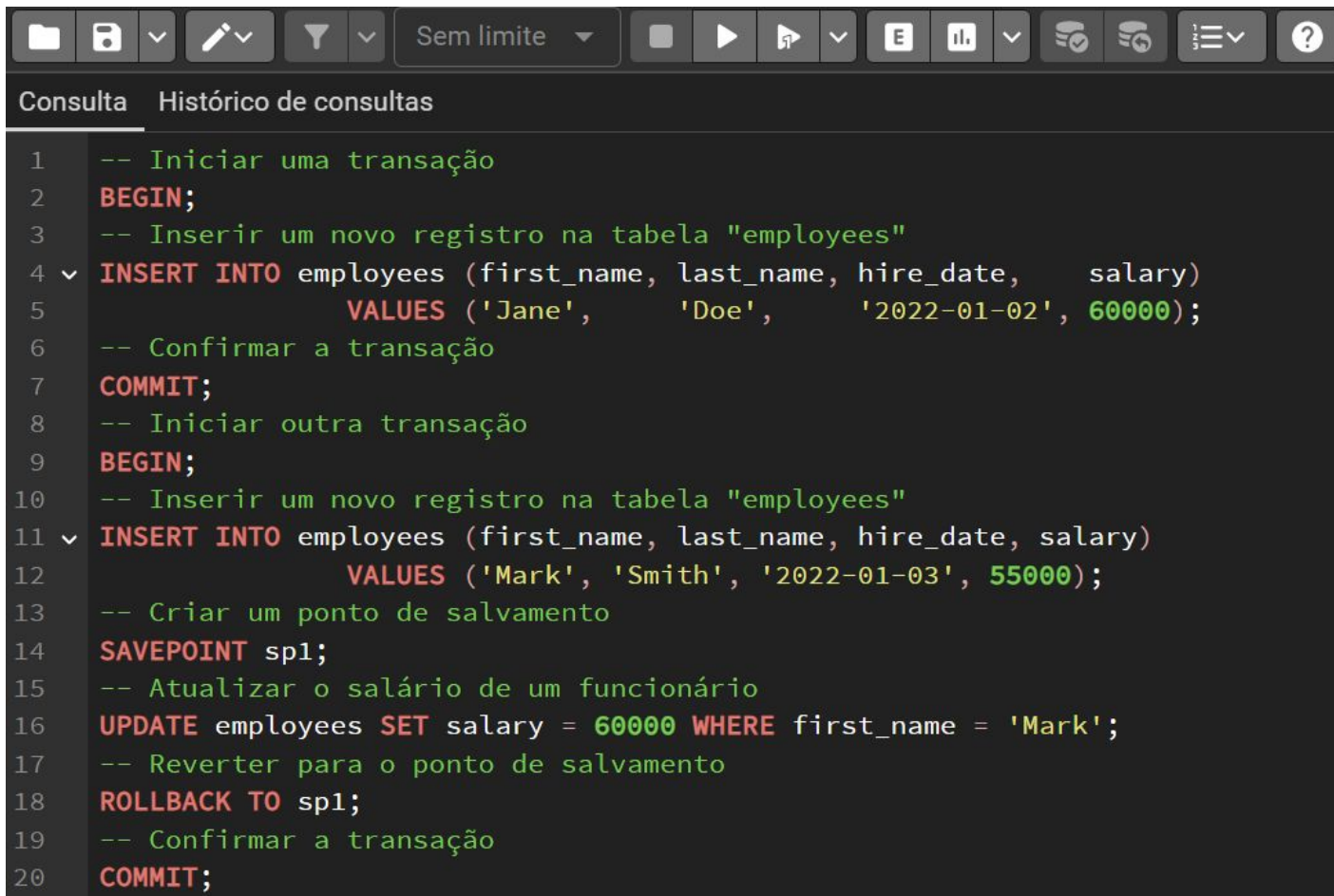
## 4. TCL (Transaction Control Language)

**Função:** TCL é usada para gerenciar transações no banco de dados, garantindo a consistência e a integridade dos dados.

### Comandos Principais:

- **COMMIT:** Confirma uma transação, tornando todas as alterações permanentes.
- **ROLLBACK:** Desfaz uma transação, revertendo todas as alterações feitas.
- **SAVEPOINT:** Cria um ponto de salvamento dentro de uma transação, permitindo reverter para esse ponto específico.
- **SET TRANSACTION:** Define características para a transação atual, como o nível de isolamento.

## 4. TCL (Transaction Control Language)



The screenshot shows a SQL IDE interface with a toolbar at the top containing icons for file operations, execution, and settings. Below the toolbar, there are tabs for 'Consulta' and 'Histórico de consultas'. The main area displays a SQL script with line numbers 1 through 20. The script uses SQL comments (--) to describe each step of the transaction control process. The first transaction starts with 'BEGIN;', inserts a record for 'Jane Doe' with a salary of 60,000, and ends with 'COMMIT;'. The second transaction starts with 'BEGIN;', inserts a record for 'Mark Smith' with a salary of 55,000, creates a savepoint 'sp1', updates the salary of 'Mark' to 60,000, rolls back to 'sp1', and ends with 'COMMIT;'. The script is color-coded: comments are green, keywords are red, and values are green.

```
1  -- Iniciar uma transação
2  BEGIN;
3  -- Inserir um novo registro na tabela "employees"
4  INSERT INTO employees (first_name, last_name, hire_date, salary)
5  VALUES ('Jane', 'Doe', '2022-01-02', 60000);
6  -- Confirmar a transação
7  COMMIT;
8  -- Iniciar outra transação
9  BEGIN;
10 -- Inserir um novo registro na tabela "employees"
11 INSERT INTO employees (first_name, last_name, hire_date, salary)
12 VALUES ('Mark', 'Smith', '2022-01-03', 55000);
13 -- Criar um ponto de salvamento
14 SAVEPOINT sp1;
15 -- Atualizar o salário de um funcionário
16 UPDATE employees SET salary = 60000 WHERE first_name = 'Mark';
17 -- Reverter para o ponto de salvamento
18 ROLLBACK TO sp1;
19 -- Confirmar a transação
20 COMMIT;
```

# **Introdução à Normalização em Bancos de Dados Relacionais**

**A normalização é um processo sistemático de organizar os dados em um banco de dados para minimizar a redundância e melhorar a integridade dos dados. Esse processo envolve a divisão de tabelas maiores em tabelas menores e a definição de relacionamentos entre elas. As formas normais (ou normas de normalização) são as diretrizes que ajudam a alcançar esse objetivo. Vamos explorar as três primeiras formas normais (1NF, 2NF, 3NF) e a Forma Normal de Boyce-Codd (BCNF), entender a necessidade da normalização, suas vantagens e os impactos no desenvolvimento de software.**

# Primeira Forma Normal (1NF)

**Definição:** Para que uma tabela esteja na Primeira Forma Normal (1NF), ela deve satisfazer os seguintes critérios:

1. **Atomicidade:** Todos os valores nas colunas devem ser atômicos (não divisíveis).
2. **Valores Únicos:** Cada campo deve ter um valor único. Não são permitidos grupos repetidos ou conjuntos de valores.

## Exemplo:

Suponha que temos uma tabela de clientes com as seguintes informações:

Cliente_ID	Nome	Telefones
1	João	12345, 67890
2	Maria	54321, 98765



# Primeira Forma Normal (1NF)

A tabela acima, não se enquadra na 1NF porque a coluna "Telefones" contém múltiplos valores. Para normalizar, devemos separar esses valores em linhas distintas:

Cliente_ID	Nome	Telefone
1	João	12345
1	João	67890
2	Maria	54321
2	Maria	98765

# Segunda Forma Normal (2NF)

**Definição:** Para que uma tabela esteja na Segunda Forma Normal (2NF), ela deve satisfazer os seguintes critérios:

1. **Estar em 1NF:** A tabela deve estar na Primeira Forma Normal.
2. **Dependência Parcial:** Todos os atributos não chave devem depender funcionalmente da chave primária inteira, não de uma parte dela (isso se aplica a tabelas com chaves compostas).

**Exemplo:** Suponha que temos uma tabela de itens do pedidos com as seguintes informações:

Pedido_ID	Cliente_ID	Nome_Cliente	Produto_ID	Quantidade
1	1	João	101	2
2	2	Maria	102	1

Essa tabela não está em 2NF porque "Nome\_Cliente" depende apenas de "Cliente\_ID" e não de "Pedido\_ID". Para normalizar, devemos separar as informações do cliente e do pedido:

# Tabela de Clientes:

Cliente_ID	Nome
1	João
2	Maria

# Terceira Forma Normal (3NF)

**Definição:** Para que uma tabela esteja na Terceira Forma Normal (3NF), ela deve satisfazer os seguintes critérios:

1. **Estar em 2NF:** A tabela deve estar na Segunda Forma Normal.
2. **Dependência Transitiva:** Todos os atributos não chave devem depender diretamente da chave primária, sem dependências transitivas.

**Exemplo:** Suponha que temos uma tabela de vendas com as seguintes informações:

Venda_ID	Cliente_ID	Nome_Cliente	Cidade	Produto_ID	Preço
1	1	João	SP	101	50
2	2	Maria	RJ	102	30

Essa tabela não está em 3NF porque "Cidade" depende de "Cliente\_ID" através de "Nome\_Cliente". Para normalizar, devemos separar essas informações: Tabela de Vendas:

Venda_ID	Cliente_ID	Produto_ID	Preço
1	1	101	50
2	2	102	30

# Tabela de Clientes:

Cliente_ID	Nome	Cidade
1	João	SP
2	Maria	RJ

# Forma Normal de Boyce-Codd (BCNF)

**Definição:** BCNF é uma versão mais rigorosa da 3NF. Para que uma tabela esteja em BCNF, ela deve satisfazer os seguintes critérios:

1. **Estar em 3NF:** A tabela deve estar na Terceira Forma Normal.
2. **Cada determinante é uma chave candidata:** Para toda dependência funcional  $A \rightarrow B$ , A deve ser uma chave candidata.

**Exemplo:** Suponha uma tabela de aulas com as seguintes informações:

Aula_ID	Professor	Sala	Horário
1	Prof_A	101	10:00
2	Prof_B	102	11:00

Se um professor pode ensinar em várias salas, mas em diferentes horários, podemos ter dependências onde "Professor" determina "Sala" e "Horário", mas não é uma chave candidata. Para estar em BCNF, a tabela deve ser reestruturada: Tabela de Aulas:

Aula_ID	Professor	Sala	Horário
1	Prof_A	101	10:00
2	Prof_B	102	11:00

## Tabela de Professores:

Professor	Sala
Prof_A	101
Prof_B	102

## Tabela de Horários:

Professor	Horário
Prof_A	10:00
Prof_B	11:00

# Necessidade da Normalização

## Por que Normalizar?

1. **Eliminar Redundância de Dados:** Evitar a duplicação desnecessária de dados.
2. **Aumentar a Consistência:** Garantir que as informações sejam atualizadas de maneira consistente.
3. **Melhorar a Integridade dos Dados:** Reduzir as chances de anomalias de inserção, atualização e exclusão.
4. **Otimizar o Uso de Armazenamento:** Reduzir o espaço necessário para armazenar dados.
5. **Facilitar a Manutenção:** Simplificar a manutenção e a atualização do banco de dados.

## Vantagens da Normalização:

- **Integridade dos Dados:** Dados mais precisos e consistentes.
- **Eficiência de Armazenamento:** Redução de redundância e economia de espaço.
- **Facilidade de Consulta e Atualização:** Estrutura de dados mais clara e organizada.
- **Redução de Anomalias:** Minimização de problemas ao inserir, atualizar ou excluir dados.



# Impactos no Desenvolvimento de Software

- **Projeto de Banco de Dados Eficiente:** Um banco de dados bem normalizado facilita o desenvolvimento de aplicativos, garantindo que as consultas sejam eficientes e os dados sejam consistentes.
- **Manutenção e Evolução:** Um esquema de banco de dados normalizado é mais fácil de manter e adaptar a novas necessidades, facilitando a adição de novas funcionalidades sem comprometer a integridade dos dados.
- **Desempenho:** Embora a normalização possa aumentar a complexidade das consultas devido à necessidade de junções (joins), os benefícios em termos de consistência e redução de redundância frequentemente superam os custos de desempenho.
- **Colaboração:** Um banco de dados bem estruturado facilita a colaboração entre desenvolvedores, analistas de dados e administradores de banco de dados, proporcionando uma base comum e clara para o trabalho conjunto.

# Conclusão

A normalização é um aspecto fundamental no design de bancos de dados relacionais, garantindo que os dados sejam armazenados de maneira eficiente, consistente e fácil de manter. Compreender e aplicar as formas normais é crucial para qualquer desenvolvedor ou administrador de banco de dados que deseja criar sistemas robustos e escaláveis.