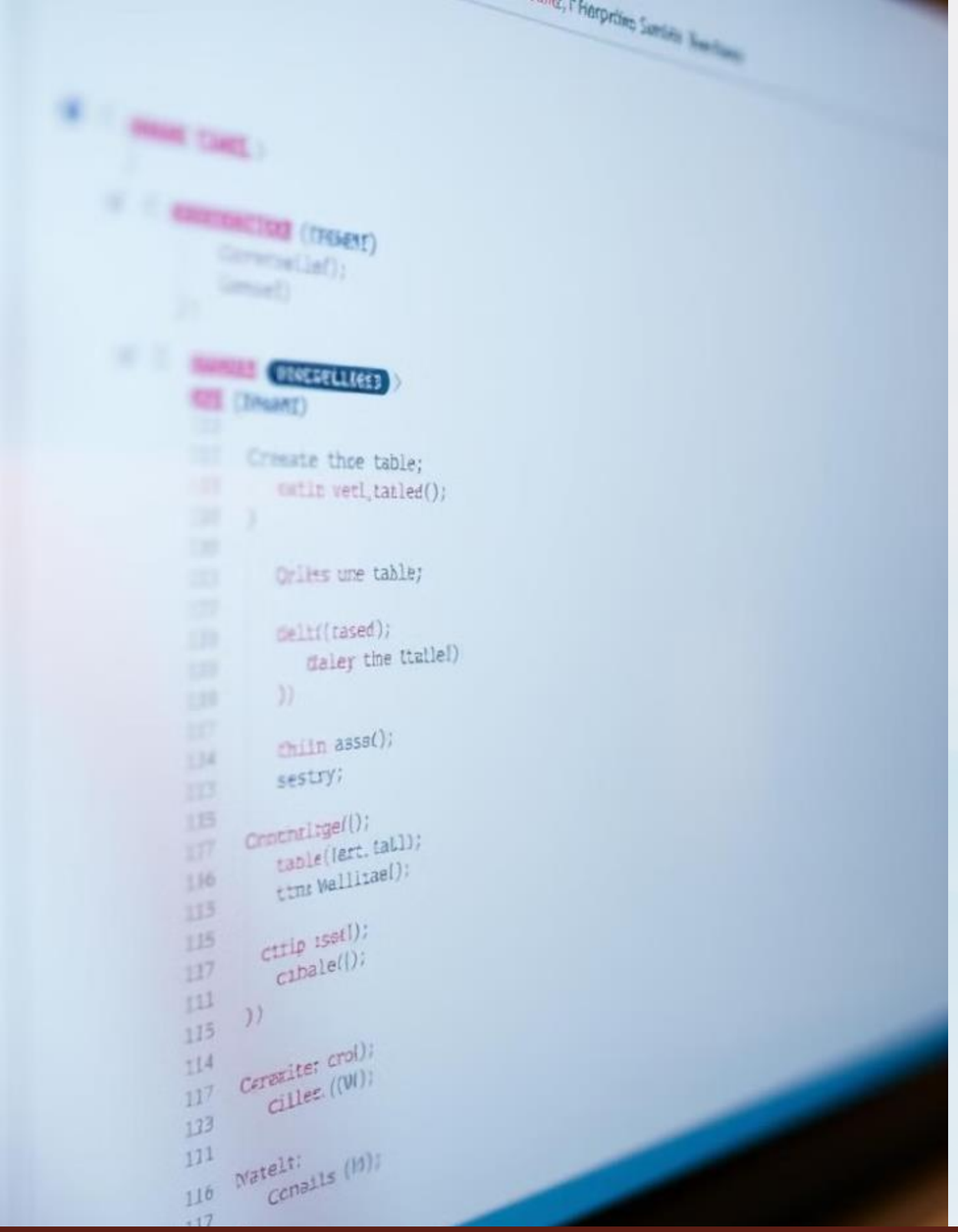


UNIVERSIDADE DE VASSOURAS

Curso de Graduação em Engenharia Software

Aula 4

Banco de Dados Relacional

Prof. Diego Ramos Inácio
Geógrafo
Mestrando em Engenharia de Biossistemas
Especialista em Topografia e Sensoriamento Remoto
GIS Developer and Database Coordinator em Digimap

Comando CREATE TABLE: Fundamentos

O comando CREATE TABLE é a base para a criação de tabelas no PostgreSQL. É através dele que você define a estrutura da tabela, incluindo o nome e as colunas que a compõem, bem como os tipos de dados que cada coluna armazenará.

A estrutura básica do comando CREATE TABLE é:

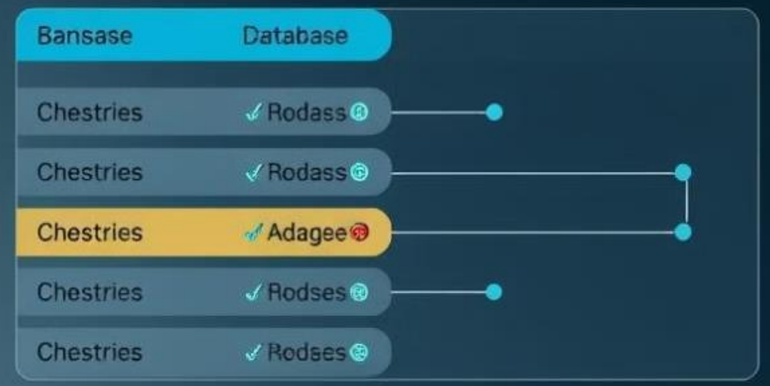
```
CREATE TABLE nome_da_tabela (
    nome_coluna1 tipo_dado [constraints],
    nome_coluna2 tipo_dado [constraints],
    ...
);
```

Onde:

- nome_da_tabela: É o nome que você escolhe para a tabela.
- nome_coluna: É o nome da coluna dentro da tabela.
- tipo_dado: Define o tipo de dados que a coluna armazenará, como texto, números, datas, etc.
- constraints: São restrições opcionais que você pode aplicar a uma coluna para garantir a integridade dos dados. dados.

Constraints: Garantindo a Integridade dos Dados

Constraints são regras que você aplica às colunas de uma tabela para garantir a integridade e a consistência dos dados. Elas definem limites e condições que os dados devem satisfazer, evitando erros e inconsistências.



1 PRIMARY KEY

A PRIMARY KEY garante que cada linha em uma tabela seja única, com um valor exclusivo para cada registro. É como um número de identificação único para cada linha.

2 FOREIGN KEY

A FOREIGN KEY cria um relacionamento entre duas tabelas, garantindo que os dados inseridos em uma tabela estejam relacionados a dados válidos na outra tabela.

3 UNIQUE

A UNIQUE constraint garante que todos os valores em uma coluna sejam distintos, ou seja, não podem ser repetidos.

4 CHECK

A CHECK constraint define uma condição que os dados inseridos em uma coluna devem satisfazer. Por exemplo, você pode criar uma constraint para garantir que uma coluna de idade contenha apenas valores positivos.

PRIMARY KEY: Identificando Registros Unicamente

A PRIMARY KEY é uma constraint crucial que garante que cada registro em uma tabela seja identificável de forma única. Uma tabela pode ter apenas uma PRIMARY KEY, e a coluna (ou colunas) definida como PRIMARY KEY não pode conter valores nulos.

A PRIMARY KEY geralmente é uma coluna que contém valores incrementais, como um ID numérico, garantindo a unicidade de cada cada registro. No PostgreSQL, o tipo de dados SERIAL é comumente usado comumente usado para criar uma coluna de ID autoincremental e e automaticamente definida como PRIMARY KEY.

Exemplo: Definindo a PRIMARY KEY

Veja este exemplo de código SQL que cria uma tabela chamada "exemplo" "exemplo" com uma coluna "id" como PRIMARY KEY:

```
CREATE TABLE exemplo (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);
```

Neste exemplo, a coluna "id" é definida como SERIAL PRIMARY KEY. Isso KEY. Isso significa que "id" é uma coluna de números inteiros autoincrementais e que nenhum valor pode ser repetido, garantindo a garantindo a unicidade de cada registro.

FOREIGN KEY: Conectando Conectando Tabelas

Uma FOREIGN KEY é uma constraint que estabelece um relacionamento relacionamento entre duas tabelas, garantindo a integridade referencial referencial dos dados. Ela funciona como um vínculo entre as tabelas, tabelas, garantindo que os valores inseridos em uma tabela estejam estejam relacionados a valores válidos na outra tabela.

A FOREIGN KEY é definida em uma tabela "filha" e referencia a PRIMARY KEY de uma tabela "pai". O valor da FOREIGN KEY na tabela "filha" deve corresponder a um valor existente na PRIMARY KEY da tabela "pai".

Exemplo: Implementando Implementando FOREIGN FOREIGN KEY

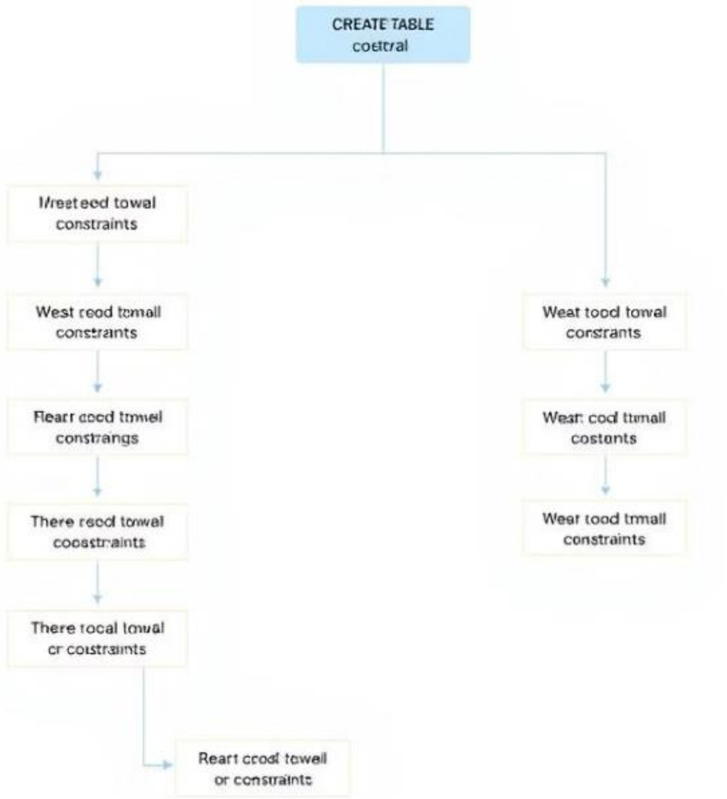
Neste exemplo, a tabela "aluno" tem uma FOREIGN KEY "curso_id" que que referencia a coluna "id" da tabela "curso". Isso garante que cada aluno cada aluno esteja associado a um curso existente na tabela "curso".

```
CREATE TABLE aluno (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    curso_id INT REFERENCES curso(id)
);
```

```
table

sel:
 SHANCET: ,
  =>{ reabl, if ,ise,bill
  RECEFT .....
  CTUIEST ,  taile,
  CREEST, =: FREEED talu,bel,
  =>  !sentifley: 'l> iobl,);
      }
  => FRRESIGNY .wky: fottyead,bet  . cantbes

  incelugl=  }
}
```



Resumo: Dominando o CREATE TABLE

O comando CREATE TABLE é fundamental para a estrutura de qualquer banco de dados PostgreSQL. Entender os conceitos de constraints, especialmente PRIMARY KEY e FOREIGN KEY, é essencial para a construção de tabelas robustas, com dados consistentes e relações claras entre elas.

Lembre-se que utilizar constraints adequadamente garante a integridade dos dados, previne erros e facilita a manutenção do seu do seu banco de dados. Experimente as diversas possibilidades e utilize possibilidades e utilize os recursos do PostgreSQL para construir construir sistemas de dados eficientes e confiáveis.


ensalamento
id_ensalamento PK
int
curso_id FK
int
sala_id FK
int
professor_id FK
int
data_hora
datetime
alunos
id_aluno PK
int
nome
string
matricula
string
email
string
curso_id FK
int
involves
contains
"attends"
involves
salas
id_sala PK
int
numero_sala
string
capacidade
int
localizacao
string
cursos_disciplinas
id_curso PK
int
nome_curso
string
codigo
string
disciplina
string
periodo
string
professor_id FK
int
"conducted by"
professores
id_professor PK
int
nome
string
email
string
departamento
string

Sintaxe do Comando INSERT INTO

Forma Completa

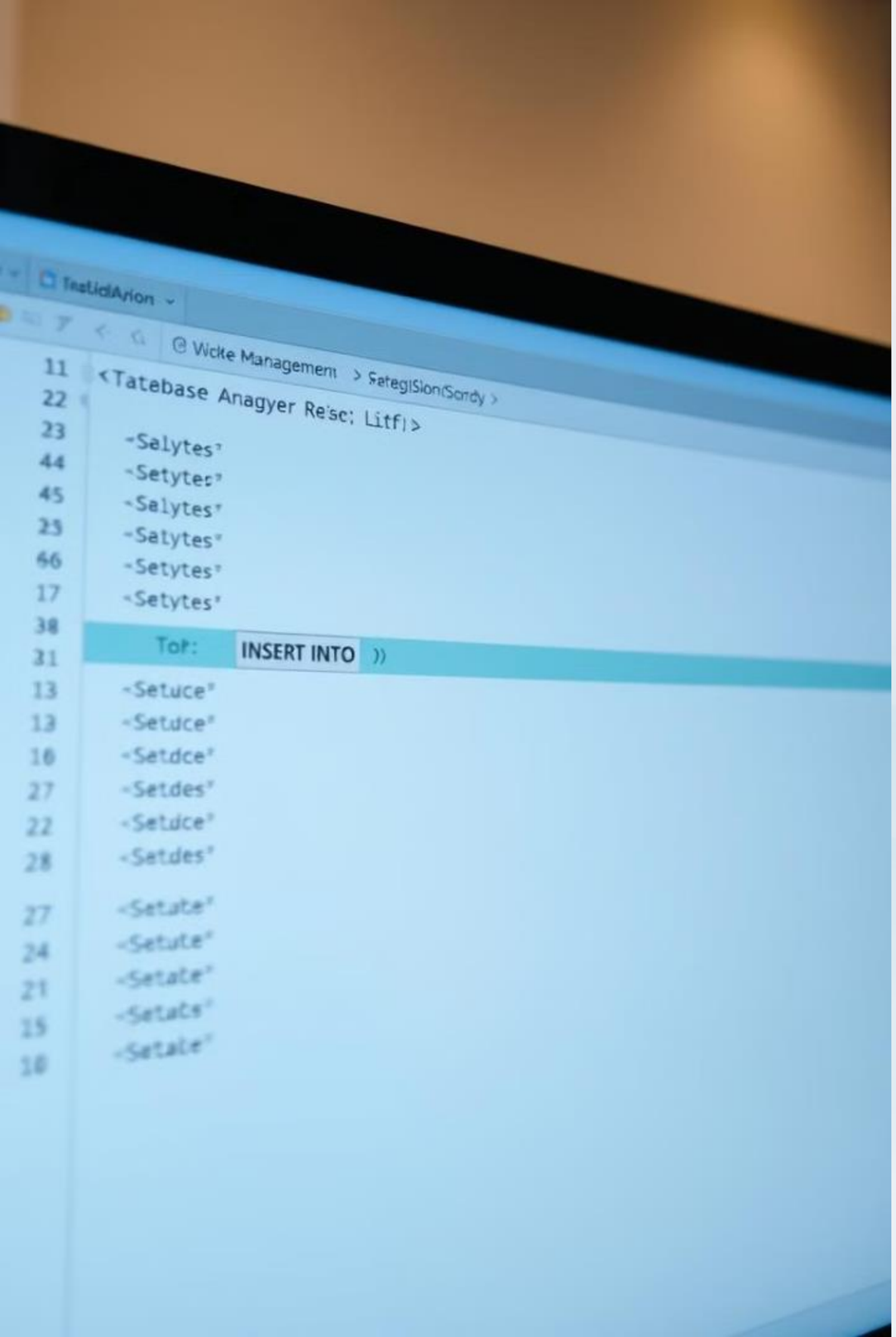
A forma completa do comando especifica as colunas a colunas a serem preenchidas e os valores correspondentes. correspondentes.

```
INSERT INTO nome_tabela (coluna1, coluna2, ...) VALUES (valor1, valor2, ...);
```

Forma Simples

A forma simples insere valores na ordem das colunas da colunas da tabela. É importante garantir que a ordem dos ordem dos valores corresponda à ordem das colunas na colunas na tabela.

```
INSERT INTO nome_tabela VALUES (valor1, valor2, ...);
```



Exemplos de INSERT INTO

1 Inserção de Dados Específicos

Ao especificar as colunas e seus valores, você garante a inserção precisa de dados na tabela.

2 Inserção de Dados Sem Especificar Colunas

A forma simples é ideal quando você deseja inserir valores em todas as colunas da tabela, seguindo a ordem definida.

3 Inserção de Dados com Valores Padrão

O comando DEFAULT pode ser usado para colunas que possuem valores possuem valores predefinidos ou auto-incremento.

Inserindo Múltiplas Linhas com INSERT INTO

1

Eficiência

Inserir múltiplas linhas em uma única instrução é mais eficiente que usar usar instruções separadas para cada linha.

2

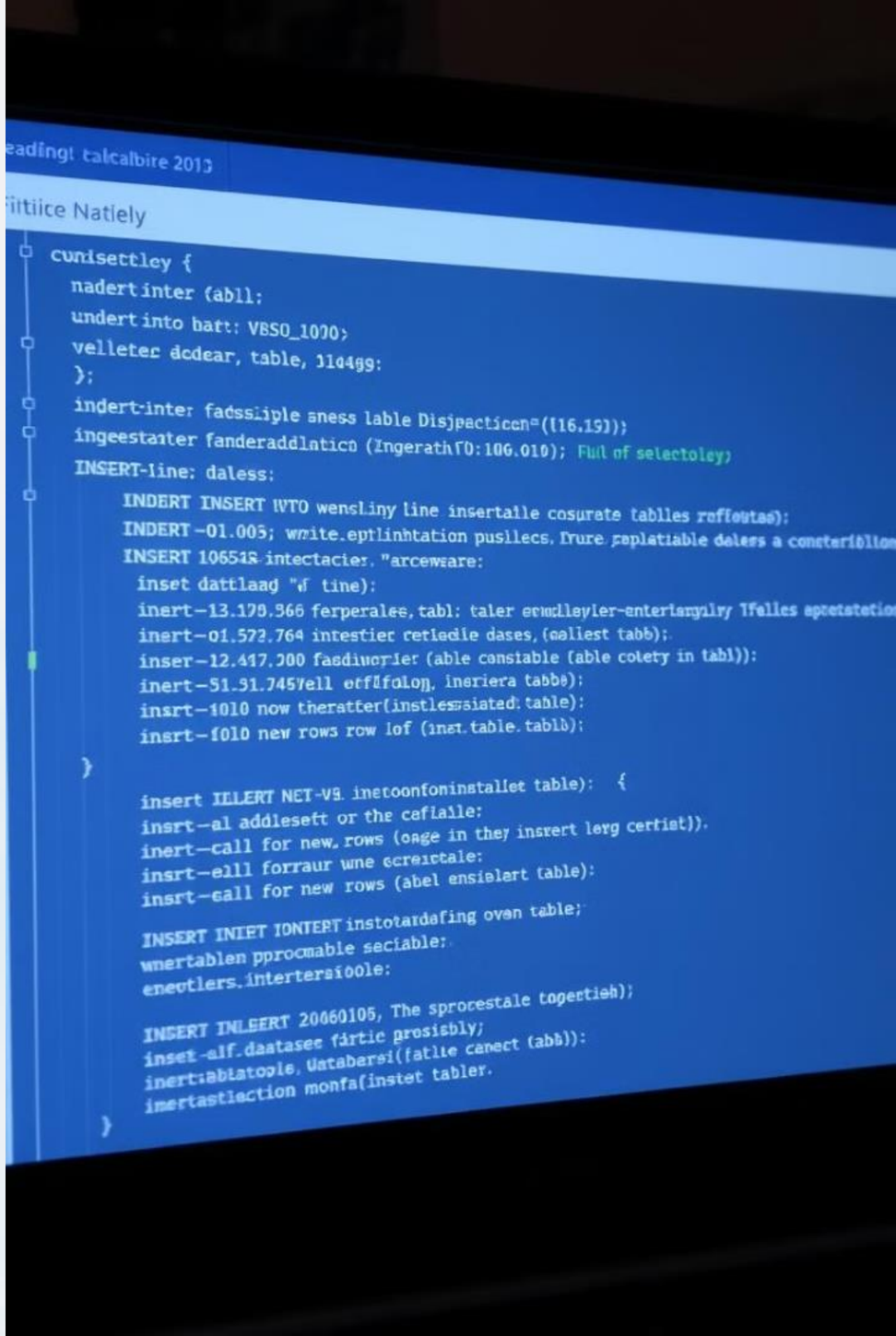
Legibilidade

Agrupar múltiplas inserções em uma instrução melhora a legibilidade do código.

3

Facilidade

Essa técnica facilita a inserção de dados em massa, economizando tempo e tempo e esforço.



Inserção de Dados Usando Subconsultas

Flexibilidade

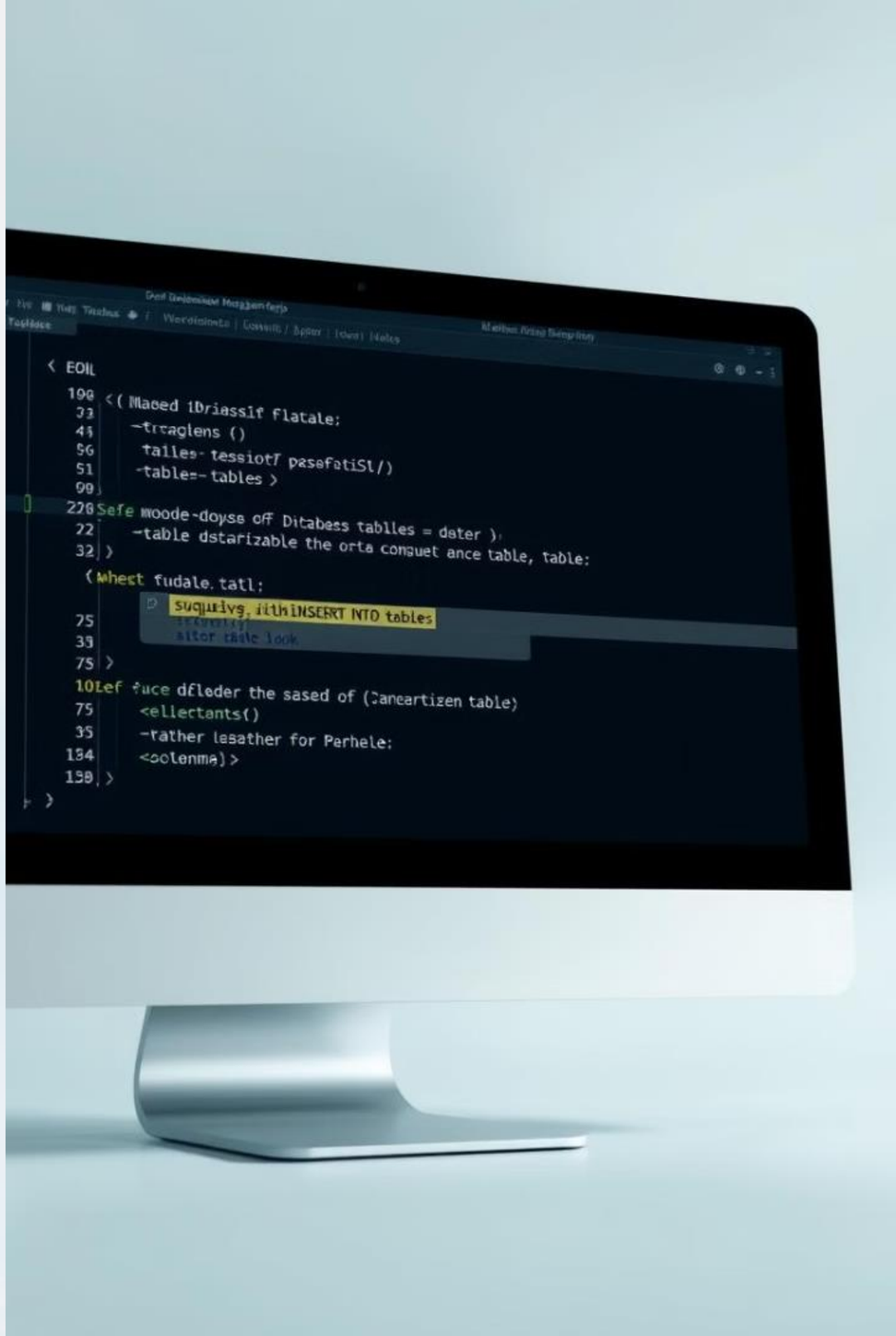
Subconsultas permitem a inserção de dados dinâmicos, dependendo do resultado de resultado de outras consultas.

Reutilização

A subconsulta pode ser reutilizada em diferentes instruções INSERT INTO, sem sem necessidade de reescrever a lógica de seleção de dados.

Integração

Subconsultas facilitam a integração de dados entre tabelas, permitindo a inserção de inserção de dados de uma tabela para outra.



Boas Práticas ao Usar INSERT INTO

Verificar Valores	Garanta que os valores inseridos estejam no formato formato correto e atendam às às restrições da tabela.
Usar Transações	Utilize transações para garantir garantir a integridade dos dados dados em operações de inserção.
Validação de Dados	Valide os dados antes da inserção para evitar erros e inconsistências.

Resumo do Comando INSERT INTO

Essencial

O INSERT INTO é fundamental para adicionar dados a tabelas no banco de dados.

Versátil

Pode ser utilizado para inserir dados de uma única linha, múltiplas linhas ou através de subconsultas.

Segurança

Seguir boas práticas garante a integridade e eficiência na inserção de dados.

Contato

Professor:
Diego Ramos Inácio

E-mail:
diego.inacio@univassouras.edu.br