O comando SQL INSERT é essencial para inserir dados em uma tabela. Com ele, podemos adicionar um ou mais registros ao mesmo tempo, indicar em quais campos os dados serão inseridos e muito mais.

Entretanto, é preciso atenção ao utilizar esse recurso, pois ele pode retornar erro se algum campo obrigatório não estiver na instrução SQL para a inserção dos dados. Por ser um dos principais comandos para a manipulação de dados, as pessoas programadoras precisam entender como ele funciona.

Por isso, preparamos este post com os seguintes tópicos:

Índice

Boa leitura!

O que é o comando SQL INSERT?

O comando SQL INSERT é utilizado para inserirmos dados em uma tabela. Entretanto, existem diferentes formas de utilizar esse recurso. Confira quais são elas nos próximos tópicos.

SQL INSERT INTO

O comando SQL INSERT INTO é utilizado para inserir dados em uma tabela. Para isso, devemos indicar quais campos serão inseridos e seus valores correspondentes.

Qual a sintaxe do comando SQL INSERT INTO?

A sintaxe do comando SQL INSERT INTO é:

```
INSERT INTO <nome_tabela> (coluna1, coluna2, ...., colunaN)
VALUES
(valor_coluna1, valor_coluna2, ..., valor_colunaN)
```

Em que:

- **nome_tabela**: indica a tabela do banco de dados em que os dados serão inseridos;
- **coluna1, coluna2, ..., colunaN**: representam os nomes das colunas da tabela que terá os dados inseridos;
- **valor_coluna1, valor_coluna2**, …, valor_colunaN: indicam os dados correspondentes aos campos declarados na cláusula INSERT.

Quando declaramos os campos da tabela na cláusula INSERT, podemos inserir apenas os dados desejados e os que sejam obrigatórios. Além disso, não precisamos seguir a mesma ordem em que os campos estão definidos no banco de dados. O importante é seguir a mesma ordem apenas na disposição dos campos e valores na instrução SQL.

Quando utilizamos a sintaxe dessa forma, não precisamos informar o valor do ID, caso ele tenha sido declarado para ser incrementado automaticamente. Falaremos mais sobre isso no tópico com os exemplos práticos.

Confira outra sintaxe do SQL INSERT:

```
INSERT INTO <nome_tabela>
VALUES
(valor_coluna1, valor_coluna2, ...., valor_colunaN)
```

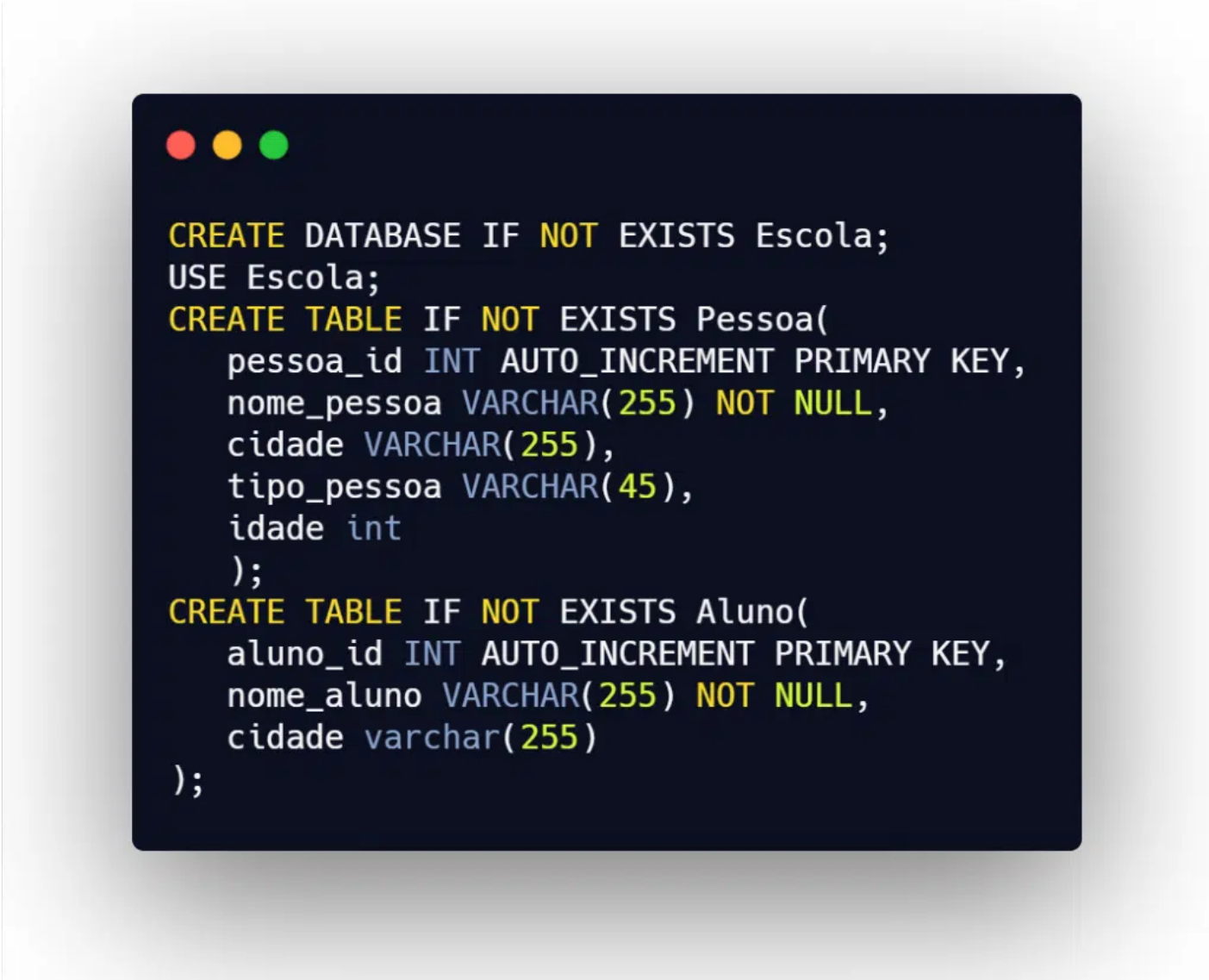
Perceba que, nessa sintaxe, não definimos os nomes de campos que serão inseridos. Isso significa que precisamos inserir todos os registros que fazem parte da tabela e na ordem existente no banco de dados.

Exemplos de uso do comando SQL INSERT INTO

Nada melhor que conferirmos exemplos práticos para observarmos o comportamento da instrução SQL INSERT. Para isso, vamos criar uma pequena base de dados chamada **"Escola"**, que contém duas tabelas necessárias para realizarmos os nossos testes, são elas:

- **Pessoa**: utilizada para armazenar as informações de qualquer tipo de pessoa que tenha alguma relação com a escola (professor, aluno, funcionário, por exemplo);
- **Aluno**: para armazenar apenas as informações sobre as pessoas estudantes.

Confira o script para a criação da base no banco de dados MySQL.



```
CREATE DATABASE IF NOT EXISTS Escola;
USE Escola;
CREATE TABLE IF NOT EXISTS Pessoa(
   pessoa_id INT AUTO_INCREMENT PRIMARY KEY,
   nome_pessoa VARCHAR(255) NOT NULL,
   cidade VARCHAR(255),
   tipo_pessoa VARCHAR(45),
   idade int
   );
CREATE TABLE IF NOT EXISTS Aluno(
   aluno_id INT AUTO_INCREMENT PRIMARY KEY,
   nome_aluno VARCHAR(255) NOT NULL,
   cidade varchar(255)
);
```

Exemplo 1. Inserir campos determinados

Vamos iniciar o nosso exemplo inserindo dados na tabela "Pessoa". Perceba que temos o campo "tipo_pessoa", que indica qual o

tipo de relacionamento a pessoa tem com a escola.

Se estivéssemos em um cenário real, esse campo deveria armazenar um valor do tipo inteiro e ser a chave estrangeira de outra tabela. Para facilitar o entendimento, utilizamos o campo preenchido com o seu conteúdo em texto.

Confira a instrução SQL para a inserção dos dados:

```
INSERT INTO Pessoa (nome_pessoa, tipo_pessoa, idade)
VALUES
   ('Maria de Lourdes', 'Professora', 30),
   ('José Pedro', 'Aluno', 25),
   ('Joana Soares', 'Funcionária', null);

SELECT * FROM Pessoa;
Resultado:
Pessoa_id / nome_pessoa / cidade / tipo_pessoa / idade
1 / Maria de Lourdes / null / Professora / 30
2 / José Pedro / null / Aluno / 25
3 / Joana Soares / null / Funcionária / null
```

Na cláusula INSERT, informamos três campos da tabela Pessoa e os valores correspondentes na cláusula VALUES. Por fim, utilizamos a cláusula SELECT para selecionar todos os dados da tabela para verificar o resultado. Perceba que o conteúdo do campo **"**cidade**"** está nulo (null) em todos os registros, pois ele não foi informado quando fizemos a inserção dos dados.

Portanto, **sempre que um campo for omitido na cláusula INSERT e seu valor não for obrigatório, o conteúdo inserido será nulo, mesmo que ele seja do tipo numérico ou de outro formato.**

Vale ressaltar que **quando um campo é definido como NOT NULL, significa que ele é obrigatório.** Portanto, ele sempre deverá ser informado ao fazer um INSERT. Caso essa condição não seja atendida, o SQL retornará um erro para comunicar que o campo não está preenchido.

Exemplo 2. Inserir todos os campos

A cláusula INSERT pode ser utilizada para inserir todos os campos da tabela, ou seja, sem a necessidade de informar o nome de cada campo. Entretanto, **é preciso informar todos os valores (cláusula VALUES) de cada um e na ordem em que eles foram criados**. Os campos que tiverem conteúdos nulos ou vazios também devem ser informados, caso contrário, haverá um erro na execução do comando.

É importante observar esses detalhes ao utilizar o comando dessa forma, pois isso garante de que os dados serão inseridos corretamente, além de evitar erros de

valores incompatíveis.

Outra observação importante é em relação ao campo de identificação (ID), caso ele seja configurado como autoincremento. No banco de dados MySQL, por exemplo, ele pode ser inserido com um valor específico ou com o conteúdo igual a nulo. Nesse caso, o próprio banco verificará qual será o próximo número sequencial.

Já no SQL Server, não há a necessidade de informar esse campo se o identificador for definido como autoincremento, pois o banco fará a inserção do valor sequencial automaticamente. Portanto, **é importante observar essas particularidades conforme o banco de dados utilizado**.

Confira a instrução SQL abaixo feita para o MySQL:

```
INSERT INTO Pessoa
VALUES
   (null, 'Marcos Araújo', 'São Paulo', 'Professor', null),
   (5, 'Mariana Santos', null, 'Aluna', 28),
   (null, 'Patrícia Soares', 'Belo Horizonte', 'Aluna', 25);
```

A mesma instrução no SQL Server:

```
INSERT INTO [escola].[dbo].[Pessoa]
VALUES
   ('Marcos Araújo', 'São Paulo', 'Professor', null),
   ('Mariana Santos', null, 'Aluna', 28),
   ('Patrícia Soares', 'Belo Horizonte', 'Aluna', 25);
```

O resultado para as duas instruções considerando os dados já incluídos no exemplo anterior será:

```
SELECT * FROM pessoa

Resultado:
pessoa_id / nome_pessoa / cidade / tipo_pessoa / idade
1 / Maria de Lourdes / null / Professora / 30
2 / José Pedro / null /Aluno / 25
3 / Joana Soares / null / Funcionária / null
4 / Marcos Araújo / São Paulo / Professor / null
5 / Mariana Santos / null / Aluna / 28
6 / Patrícia Soares / Belo Horizonte / Aluna / 25
```

SQL INSERT INTO SELECT

O comando SQL INSERT INTO SELECT é utilizado para inserirmos os dados de uma tabela em outra. Na prática, é como se disséssemos para o banco de dados, selecione determinados campos da segunda tabela e insira esses valores na primeira.

Qual a sintaxe do comando SQL INSERT INTO SELECT?

A sintaxe do comado SQL INSERT INTO SELECT é:

```
INSERT INTO <nome_tabela_destino>
SELECT (coluna1, coluna2, ..., colunaN)
FROM <nome_tabela_origem>
WHERE <condição>
```

Em que:

- **nome_tabela_destino**: corresponde a tabela em que os dados selecionados serão inseridos;
- **coluna1, coluna2, ..., colunaN**: indicam os campos selecionados da tabela de origem que serão inseridos na tabela de destino;
- **nome_tabela_origem**: indica a tabela que contém os campos selecionados;
- **condição**: caso exista algum critério de seleção dos dados.

Exemplos de uso do SQL INSERT INTO SELECT

Exemplo 1. Inserir dados de outra tabela

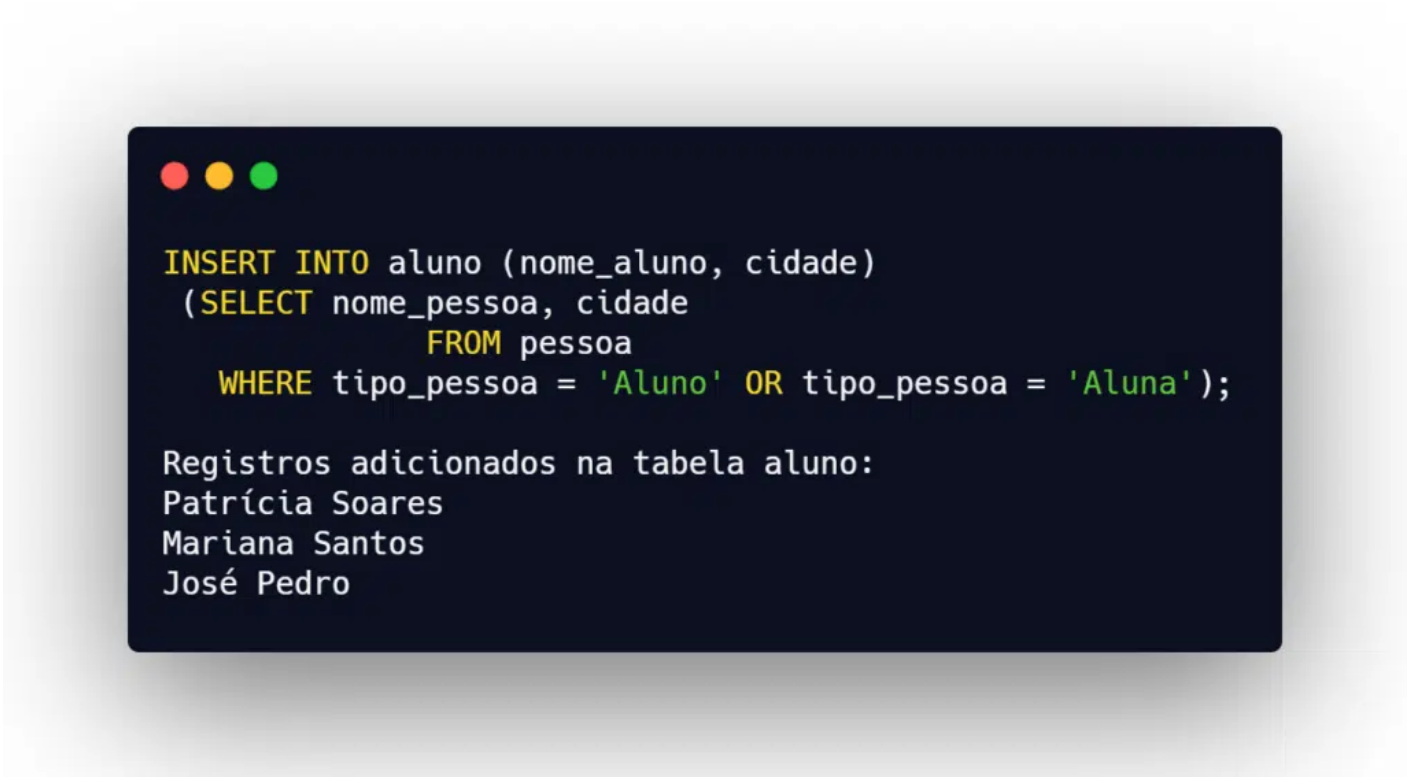
Outra forma de utilizar o comando INSERT é para trazer dados de outra tabela. Nesse caso, precisamos indicar quais campos receberão os dados selecionados na tabela de destino e selecionar os campos equivalentes na tabela de origem. Confira a instrução SQL a seguir:

```
INSERT INTO aluno (nome_aluno, cidade)
 (SELECT nome_pessoa, cidade FROM pessoa);

SELECT nome_pessoa FROM aluno;

Resultado:
```

```
Maria de Lourdes
José Pedro
Joana Soares
Marcos Araújo
Mariana Santos
Patrícia Soares
```

Exemplo 2. Inserir dados de outra tabela com alguma condição

No exemplo acima, selecionamos todas as pessoas da tabela **"Pessoa"** e inserimos na tabela "Aluno". Entretanto, queremos condicionar essa seleção, ou seja, recuperar apenas os registros de pessoas que sejam alunas. Para isso, basta adicionar a cláusula WHERE no momento de fazer a seleção dos dados. Veja a instrução SQL abaixo:



Como o INSERT é usado no SQL Server?

Para adicionar dados utilizando o SQL Server, é utilizada a seguinte sintaxe formal:

```
INSERT [INTO] tab_name [(column_list)] VALUES (value_1, value_2, ... value_N)
```

Primeiro vem a expressão INSERT INTO. Depois, entre colchetes, você pode especificar a lista de colunas (nome da tabela) e, nela, podem ser adicionados dados. No final, após a palavra VALUES é possível listar os valores adicionados às colunas.

Por exemplo, considere o seguinte banco de dados a ser criado:

```
CREATE DATABASE productsdb_1;
USE productsdb_1;
CREATE TAB Prods
(
    Id INT PRIMARY KEY,
    ProductName_1 VARCHAR(30) NOT NULL,
    Manufacturer_1 VARCHAR(20) NOT NULL,
    ProductCount_1 INT DEFAULT 0,
    Price_1 DECIMAL (10, 2) NOT NULL
)
```

Vamos adicionar uma linha a ela usando o comando INSERT:

```
INSERT Prods VALUES ('iPhone 7', 'Apple', 5, 52000)
```

Observe que os valores das colunas entre colchetes após a palavra-chave VALUES são passados na ordem em que são declarados. Por exemplo, na expressão CREATE TABLE acima, você pode observar que a primeira coluna é Id.

A segunda coluna representa ProductName, então o primeiro valor (a linha “iPhone 7”) será passado para esta coluna. O segundo valor (a linha “Apple”) será passado para a terceira coluna “Fabricante” e assim por diante. Ou seja, os valores são passados para as colunas da seguinte forma:

- Nome do Produto: ‘iPhone 7’.;
- Fabricante: ‘Apple’;
- Contagem de produtos: 5;
- Preço: 52.000.

Ao inserir valores, você também pode especificar colunas diretas às quais os valores serão adicionados:

```
INSERT INTO Prods (ProductName_1, Price_1, Manufacturer_1)

VALUES ('iPhone 6S', 41000, 'Apple')
```

Aqui o valor é especificado apenas para três colunas. E agora os valores são passados na ordem das colunas:

- Nome do Produto: ‘iPhone 6S’.
- Fabricante: ‘Apple’
- Preço: 41.000

Para colunas não especificadas (neste caso ProductCount), um valor padrão será adicionado se o atributo DEFAULT ou o valor NULL for definido. As colunas não especificadas devem ser NULL ou ter o atributo DEFAULT.

Também podemos adicionar mais de uma linha por vez:

```
INSERT INTO Prods VALUES

('iPhone 6', 'Apple', 3, 36000),

('Galaxy S8', 'Samsung', 2, 46000),

('Galaxy S8 Plus', 'Samsung', 1, 56000)
```

Nesse caso, três linhas serão adicionadas à tabela.

Além disso, ao adicionar, podemos especificar o uso do valor padrão para a coluna usando a palavra-chave DEFAULT ou o valor NULL:

```
INSERT INTO Prods (ProductName_1, Manufacturer_1, ProductCount_1, Price_1)

VALUES ('Mi6', 'Xiaomi', DEFAULT, 28000)
```

Nesse caso, o valor padrão da coluna ProductCount será usado (se estiver definido, se não estiver, será NULL).

Boas práticas ao usar o comando SQL INSERT

O comando SQL INSERT é um dos mais importantes da linguagem SQL, pois permite a inserção de dados nas tabelas. Entretanto, é preciso atenção ao utilizá-lo, pois se a instrução não estiver no formato adequado, as informações incluídas no banco podem estar erradas ou, até mesmo, retornar um erro de execução.

Por isso, **é importante observar a sintaxe da instrução utilizada e seguir sua indicação para garantir que os dados serão inseridos corretamente**. Vale lembrar que, apesar de a linguagem SQL ser comum aos bancos de dados relacional, cada banco tem suas particularidades. Portanto, **sempre consulte a documentação oficial do banco utilizado para evitar problemas.**

O comando SQL Insert é um dos principais comandos que você utilizará em sua jornada como desenvolvedor ou desenvolvedora. **Ele é uma das quatro possíveis operações que podemos fazer em bancos de dados, além de atualização, leitura e deleção de alguma informação.**