Aula 12 - Linguagem SQL: DML

UC10 - Criar banco de dados

Comando INSERT

- O comando INSERT é usado em banco de dados, quando queremos inserir dados na base. Podemos especificar uma linha a ser inserida ou escrever uma consulta cujo resultado é um conjunto de linhas a inserir.
- Os valores de atributos a serem inseridos devem estar na ordem que foram definidos no CREATE TABLE. Caso não estejam (ou o usuário não lembre a ordem correta), é necessário especificar a ordem.

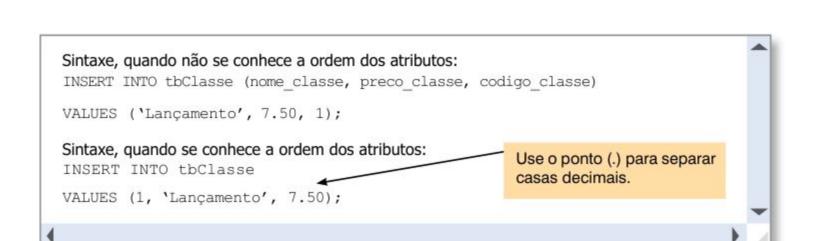
```
Sintaxe quando não se conhece a ordem dos atributos:
INSERT INTO nome da tabela (atributo_1, ..., atributo_n)
```

VALUES (valor 1, ..., valor n);

Sintaxe quando se conhece a ordem dos atributos (neste caso, podem-se omitir os atributos):

INSERT INTO nome da tabela

VALUES (valor_1, ..., valor_n);



Explicitando null e default:

INSERT INTO tbCliente (codigo_cli, CPF_cli, nome_cli, data_cadastro, cidade_cli, UF_cli)

VALUES (1, '12345678911','Pedro', null, null, default);

Explicitando apenas os atributos que serão preenchidos:

INSERT INTO tbCliente (codigo_cli, CPF_cli, nome_cli)
VALUES (1, '12345678911','Pedro');

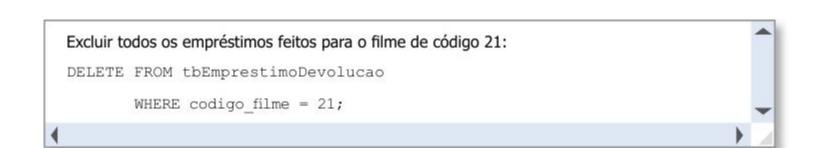
• Como foi dito anteriormente, além de inserir uma linha em uma tabela, o comando INSERT pode ser utilizado junto a um comando de consulta para retornar um conjunto de linhas que serão inseridos em outra tabela.

```
Criando a nova tabela:
CREATE TABLE tbTituloCategoria
(nome titulo varchar (50),
 nome categoria varchar (20)
);
Fazendo uma consulta em que as linhas do resultado serão inseridas na tabela
tbTituloCategoria:
INSERT INTO tbTituloCategoria(nome titulo, nome categoria)
SELECT nome titulo, nome categoria
FROM tbtitulo
INNER JOIN tbCategoria ON tbCategoria.codigo_categoria = tbTitulo.
codigo categoria;
```

Comando DELETE

- O comando DELETE é utilizado para excluir registros de uma tabela. Este comando não exclui dados de atributos específicos e sim linhas inteiras da tabela.
- Se a sua intenção for excluir valores de um atributo específico (ou seja, deixar o atributo com valor null), você deve utilizar o comando UPDATE e mudar o valor do atributo para NULL.
- Após remover os registros usando o comando DELETE, você não poderá desfazer a operação.







- Se você não usou o ON DELETE CASCADE na criação da tabela, não será possível excluir registros de uma tabela que tenham uma PK que é referenciada por uma FK em outra tabela. Isso daria um erro porque a tabela que tem a chave estrangeira perderia a referência, e isso comprometeria a integridade da base de dados.
- Por exemplo, suponha que você queira excluir o cliente Pedro da sua base de dados. Se você fizer simplesmente um DELETE na tabela *tbCliente*, o SGBD vai emitir uma mensagem de erro, porque o cliente Pedro é referenciado na tabela *tbEmprestimoDevolucao*. Nesse caso, você terá que excluir primeiramente todos os registros referentes ao Pedro na tabela *tbEmprestimoDevolucao* e em seguida excluir o registro da tabela *tbCliente*.

```
1º passo:
    DELETE FROM tbEmprestimoDevolucao
    WHERE codigo_cli = (SELECT codigo_cli FROM tbCliente WHERE nome_cli = 'Pedro');

2º passo:
    DELETE FROM tbCliente
    WHERE nome_cli = 'Pedro';
```

- Se você utilizar o comando ON DELETE CASCADE na criação da tabela e se a linha da tabela que tem a PK for apagada, o SGBD se encarregará de apagar também a linha da tabela que tem a FK correspondente. Nesse caso, só teríamos de executar o DELETE em tbCliente (2º passo).
- Para que fique mais claro, imagine que você queira alterar o valor de *codigo_categoria* de filmes de terror de 2 para 200 na tabela *tbCategoria*. Como o código da categoria é chave estrangeira na tabela *tbTitulo*, você teria que alterar o código da categoria também nessa tabela. No entanto, se você não usou UPDATE CASCADE, o SGBD não fará isso automaticamente para você. Nesse caso, você teria que inserir um novo registro na tabela *tbCategoria* com o código 200, depois alterar na tabela *tbTitulo* todos os códigos de 2 para 200 e finalmente apagar na tabela *tbCategoria* o registro que tem o código 2.

```
1º passo:
INSERT INTO tbCategoria
VALUES (200, 'Terror');
2º passo:
UPDATE tbTitulo
SET codigo_categoria = 200
WHERE codigo_categoria = 2;
3º passo:
DELETE FROM tbCategoria
WHERE codigo_categoria = 2;
```

Comando UPDATE

- O comando UPDATE modifica valores inseridos dentro de uma tabela.
- O comando SET é um comando de atribuição. Deve-se especificar quais atributos terão seus valores alterados e o novo valor.
- Pode-se utilizar ainda uma condição especificando qual (ou quais) registro deverá ser alterado por meio do comando WHERE. Com o WHERE será feita uma procura por registros que satisfazem a condição e as alterações serão aplicadas apenas nesses registros.

Sintaxe do comando UPDATE: UPDATE nome da tabela SET nome do atributo1 = Novo valor [{, nome do atributo_n = novo valor};] [WHERE condição;]

```
UPDATE tbClasse
SET preco_classe = 9.50
WHERE nome_classe = 'Lançamento';
```

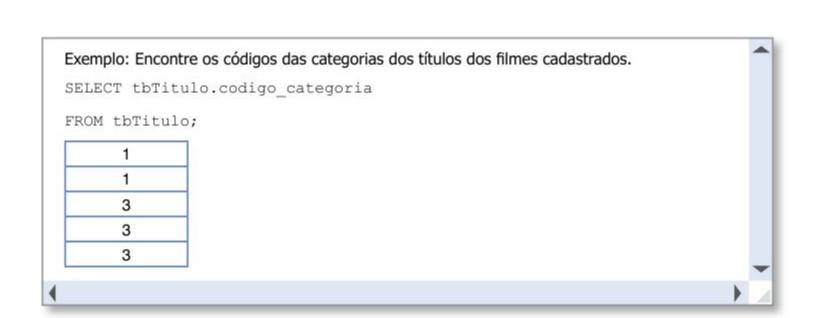
• Se você não usou o ON UPDATE CASCADE na criação da tabela e tentar alterar o valor de uma PK que é referenciada por uma FK, o SGBD irá emitir um erro. Se ele permitisse a alteração, estaria ferindo a regra de integridade referencial entre as tabelas.

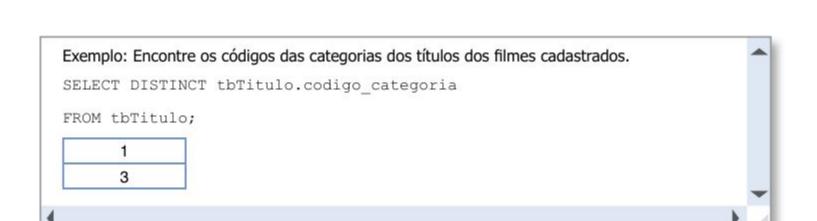
```
1º passo:
INSERT INTO tbCategoria
VALUES (200, 'Terror');
2º passo:
UPDATE tbTitulo
SET codigo_categoria = 200
WHERE codigo_categoria = 2;
3º passo:
DELETE FROM tbCategoria
WHERE codigo_categoria = 2;
```

Comando SELECT

- Depois que inserimos dados em uma tabela, podemos começar a fazer consultas nessa tabela. A
 estrutura básica de uma consulta em SQL consiste em três comandos: SELECT, FROM e WHERE.
- O comando SELECT é usado para selecionar os atributos desejados no resultado de uma consulta.
- O comando FROM define quais tabelas serão usadas na consulta.
- O comando WHERE descreve a condição da consulta e não é obrigatório. Por exemplo, se você quiser retornar o nome de todos os clientes da base de dados tbLocadora, você não precisará usar o comando WHERE.







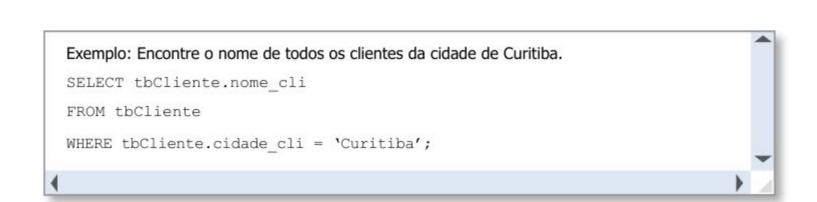
Exemplo: Verifique como ficariam os preços de filmes se aumentássemos em 5% o preço para cada classe.

SELECT tbClasse.nome_classe, tbClasse.preco_classe * 1.05

FROM tbClasse;

Comando WHERE

• O comando WHERE sempre terá associado a ele uma condição que determina quais registros deverão ser retornados pela consulta.



Exemplo: Encontre o nome de todos os clientes da cidade de Curitiba em que o código seja menor que 200.

SELECT tbCliente.nome_cli
FROM tbCliente

WHERE tbCliente.cidade_cli = 'Curitiba' AND tbCliente.codigo_cli < 200;

Exemplo: Encontre o nome de todos os clientes da cidade de Curitiba que tenham data de cadastro no ano de 2009.

SELECT tbCliente.nome_cli

FROM tbCliente
WHERE tbCliente.cidade_cli = 'Curitiba'

AND tbCliente.data_cadastro BETWEEN '2009-01-01' AND '2009-12-31';

Exemplo: Encontre o nome de todos os clientes da cidade de Curitiba que tenham data de cadastro em qualquer ano, exceto 2009.

SELECT tbCliente.nome_cli

AND tbCliente.data_cadastro NOT BETWEEN '2009-01-01' AND '2009-12-31';

FROM tbCliente

WHERE tbCliente.cidade_cli = 'Curitiba'

Comando FROM

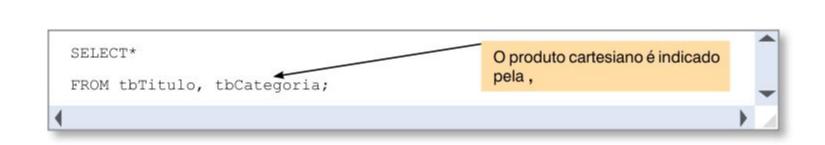
- O comando FROM define quais tabelas serão utilizadas em uma consulta, ou seja, de quais tabelas devemos buscar os dados.
- Até agora, usamos apenas uma tabela no FROM. No entanto, é muito comum termos que acessar dados de duas ou mais tabelas para que possamos obter informações interessantes.
- É importante sempre usar numa consulta o menor número de tabelas possível por questão de desempenho. Por isso, sempre faça uma análise de quais tabelas são realmente necessárias na sua consulta.
- Quando precisamos utilizar duas ou mais tabelas, podemos fazer o produto cartesiano dessas tabelas. O produto cartesiano permite combinar informações de várias tabelas fazendo a combinação de todos os dados de uma tabela com todos os dados da outra tabela.

codigo_categoria	nome_categoria
1	Terror
2	Drama
3	Comédia

tbCategoria

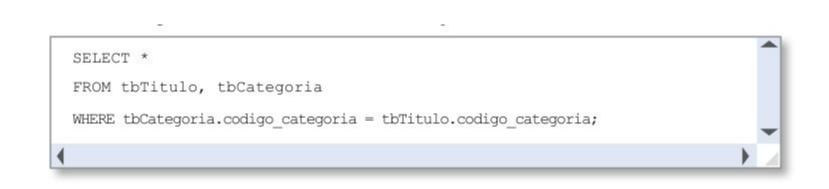
codigo_titulo	nome_titulo	codigo_categoria
1	Mortos Vivos	1
2	Superando Desafios	3
3	A Hilariante	2
4	O Bicho Papão	1

tbTitulo



codigo_titulo	nome_titulo	codigo_categoria	codigo_categoria	nome_categoria
1	Mortos Vivos	1	1	Terror
1	Mortos Vivos	1	2	Drama
1	Mortos Vivos	1	3	Comédia
2	Superando Desafios	3	1	Terror
2	Superando Desafios	3	2	Drama
2	Superando Desafios	3	3	Comédia
3	A Hilariante	2	1	Terror
3	A Hilariante	2	2	Drama
3	A Hilariante	2	3	Comédia
4	O Bicho Papão	1	1	Terror
4	O Bicho Papão	1	2	Drama
4	O Bicho Papão	1	3	Comédia

• Observe que a consulta deveria retornar apenas as linhas da tabela que estão pintadas de cinza. As outras informações retornadas não são verdadeiras. Por exemplo, o filme "Mortos Vivos" é apenas da categoria "Terror", mas no nosso resultado diz que ele pertence também a categoria "Drama" e "Comédia".



coulgo_titulo	nome_maio	coulgo_categoria	coulgo_categoria	nome_categoria
1	Mortos Vivos	1	1	Terror
2	Superando Desafios	3	3	Comédia

Drama

Terror

codigo titulo

4

A Hilariante

O Bicho Papão

• Atualmente, a forma mais utilizada de se escrever uma consulta que utilize duas ou mais tabelas é usando o comando INNER JOIN. A diferença é que esse comando possui uma condição de junção. Essa condição é a mesma que fizemos no WHERE (PK = FK). No entanto, como ela é aplicada no FROM ela permite que os dados sejam "filtrados" durante a execução do FROM, diminuindo o tempo de execução da consulta, uma vez que o número de registros pesquisados será menor agora que só temos os dados que são corretos.



• Essa consulta retornaria o mesmo resultado obtido anteriormente, porém o tempo de execução é mais rápido. É importante ressaltar que o otimizador de alguns SGBD (como por exemplo, o do SQLServer) quando se deparam com o produto cartesiano otimizam a consulta e executam o INNER JOIN. Portanto, não haverá diferença de desempenho nesse caso.