

Bancos de Dados

Roteiro de Exercícios 4

Versão: 0.2

Data: nov/2024

Autor: Oclair Prado

Oclair Prado

oclairprado@gmail.br

Objetivos:

- O objetivo fundamental deste material de apoio complementar é fornecer uma oportunidade adicional para o aluno praticar os conceitos estudados em sala de aula.
- Os exercícios estão divididos em ordem crescente de dificuldade e foram distribuídos em seções.
- Para melhor aproveitamento do material fornecido, o aluno deve resolver os exercícios propostos em casa e trazer os resultados obtidos para serem discutidos em sala de aula.

Orientações iniciais:

- Este material é continuação do GADS2022_BD_roteiro_1.pdf, GADS2022_BD_roteiro_2 e do GADS2022_BD_roteiro_3.pdf, que foram criados considerando as vendas em uma loja de alguns produtos para alguns clientes realizadas por alguns vendedores.
- Consulte os documentos anteriores para orientações sobre a inicialização do SGBD MySQL (Maria DB) e também para o preparo das tabelas com os dados necessários para a realização dos exercícios deste roteiro.

i) Inserir registros (direto e extraídos de outra tabela):

Considerando que a tabela destino ainda não existe. Deve ser criada na operação.

Primeiro, crie a tabela de origens para conter algumas cidades, com os comandos:

```
USE aulas;  
CREATE TABLE Origem (nome_cidade varchar(50) PRIMARY KEY);
```

Verifique como ficou a tabela Origem com o comando:

```
DESCRIBE Origem;
```

Agora insira algumas cidades, seguindo os próximos exemplos:

```
insert into Origem (nome_cidade) values ('Americana');  
insert into Origem (nome_cidade) values ('Atibaia');  
insert into Origem (nome_cidade) values ('Campinas');  
insert into Origem (nome_cidade) values ('Itatiba');  
insert into Origem (nome_cidade) values ('Itu');  
insert into Origem (nome_cidade) values ('Paulinia');  
insert into Origem (nome_cidade) values ('Piracicaba');  
insert into Origem (nome_cidade) values ('Sumare');  
insert into Origem (nome_cidade) values ('Tiete');
```

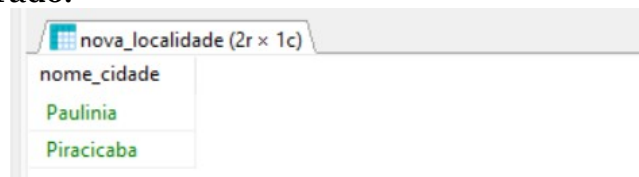
Considerando que a tabela de localidades ainda não existe, o próximo comando cria e esta tabela e insere alguns registros vindos da tabela de Origem:

```
CREATE TABLE Localidade (  
    SELECT a.nome_cidade  
    FROM Origem a  
    WHERE nome_cidade like 'A%');
```

Verifique como ficou a tabela Localidade com o comando:

```
DESCRIBE Localidade;
```

i.1) Crie uma nova tabela de localidades chamada Nova_localidade contendo as cidades de Origem com nomes começando com a letra P:

Resultado esperado:

nova_localidade (2r x 1c)	
nome_cidade	
Paulinia	
Piracicaba	

Também é possível inserir dados em uma tabela que já existe no banco de dados com dados vindos de outras tabelas:

```
INSERT INTO Localidade (nome_cidade)
SELECT (nome_cidade)
FROM Origem
WHERE nome_cidade like 'S%';
```

i.2) Para praticar, insira na tabela Localidade os nomes de cidades de Origem que começam com C.

Resultado esperado:



nome_cidade
Americana
Atibaia
Sumare
Campinas

j) Com SQL também é possível alterar dados de uma tabela usando UPDATE.

Vamos trocar Campinas por Valinhos na tabela de localidades:

```
USE aulas;
UPDATE Localidade SET nome_cidade = 'Valinhos'
WHERE nome_cidade = 'Campinas';
```

j.1) Na tabela de localidades, troque Atibaia por Alfenas:

Resultado esperado:



nome_cidade
Americana
Alfenas
Valinhos
Sumare

j.2) Altere o nome da cidade de Americana para Amparo:

Resultado esperado:



nome_cidade
Amparo
Alfenas
Valinhos
Sumare

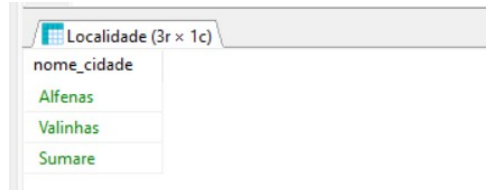
k.1) SQL também pode ser usado para remover registros de tabelas. Para isto usamos o comando DELETE:

Como exemplo, vamos remover a linha de Localidade com Amparo:

```
USE aulas;
```

```
DELETE from Localidade WHERE nome_cidade = 'Amparo';
```

Resultado esperado:



nome_cidade
Alfenas
Valinhas
Sumare

k.2) Apague da tabela Localidade a linha com a cidade Sumare:

Resultado esperado:



nome_cidade
Alfenas
Valinhas

l) Transação é como chamamos um conjunto de operações no banco de dados que devem ser completadas com sucesso para que sejam efetivadas.

Propriedades ACID:

- **Atomicidade:** Se as operações forem confirmadas, elas acontecerão todas sem exceção. Não ocorre a execução de apenas parte das operações contidas na transação.
- **Consistência:** As alterações no banco de dados somente acontecem caso o novo estado do sistema for válido. Se uma mudança inválida for executada, ela irá falhar, e o sistema permanece no seu estado anterior, válido.
- **Isolamento:** A transação não é visível a outros processos até que ela seja confirmada e persistida no banco.
- **Durabilidade:** Após a transação ter sido confirmada, ou seja, que suas operações tenham todas sido executadas com sucesso, as alterações são automaticamente persistidas (gravadas no banco), e não é necessário se preocupar em gravar as mudanças.

Neste exemplo será mostrado como podemos remover uma linha de uma tabela dentro de uma transação sem que efetivamente seja apagada, porque depois do ROLLBACK todos os ajustes efetuados dentro da transação são descartados.

```
SET @@autocommit = OFF;  
START TRANSACTION;  
DELETE FROM Localidade WHERE nome_cidade LIKE 'A%';  
SELECT * FROM Localidade;  
ROLLBACK;
```

Localidade (1r x 1c)	
nome_cidade	
Valinhas	

Verifique que nada mudou na tabela Localidade com o comando:

```
SELECT * FROM Localidade;
```

Localidade (2r x 1c)	
nome_cidade	
Alfenas	
Valinhas	

1.1) Mostre a sequência de comandos seguros (transação) para inserir a cidade Diadema nas tabelas Origem e Localidade:

Resolução:

```
SET @@autocommit = OFF;  
START TRANSACTION;  
INSERT INTO Origem (nome_cidade) VALUES ('Diadema');  
INSERT INTO Localidade (nome_cidade) VALUES ('Diadema');  
  
COMMIT; # Caso tudo tenha corrido bem  
ROLLBACK; # Caso tudo tenha corrido algum problema
```

Note que neste caso o controle está todo nas suas mãos. Você decide se deve usar o COMMIT, caso tudo tenha ocorrido bem ou se deve usar o ROLLBACK, caso tenha ocorrido algum problema em algum dos comandos anteriores.

Resultado esperado considerando tudo correu bem:

Localidade (3r x 1c)	
nome_cidade	
Alfenas	
Valinhas	
Diadema	

l.2) Mostre os comandos para remover com segurança (Transação) a cidade Diadema das tabelas Origem e Localidade.

Resultado esperado:



Localidade (2r x 1c)	
nome_cidade	
Alfenas	
Valinhas	

m) Views, visões, vista, tabela virtual.

Uma view é uma tabela lógica baseada em uma tabela ou mais tabelas ou em outra view. Uma view não contém dados próprios mas é como uma janela através da qual os dados das tabelas podem ser vistos e, em casos especiais, até alterados.

Sintaxe:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

m.1) Crie uma view para mostrar apenas as cidades que começam com A na tabela de Origem.

Resolução:

```
USE aulas;
CREATE VIEW view_origem AS
SELECT nome_cidade
FROM origem
WHERE nome_cidade LIKE 'A%';
```

Resultado esperado:



```
1 USE aulas;
2 SELECT * FROM view_origem
3
```

view_origem (2r x 1c)	
nome_cidade	
Americana	
Atibaia	

m.2) Mostre os comandos para criar uma view que mostre apenas as cidades que comecem com a letra C da tabela Origem.

Resultado esperado:



The screenshot shows a SQL query editor with two lines of code: `1 USE aulas;` and `2 SELECT * FROM view_origem_c`. Below the editor, a results pane displays the output of the query. It shows a table with one row and one column. The column is named `nome_cidade` and has a primary key icon. The value in the row is `Campinas`.

view_origem_c (1r x 1c)
nome_cidade
Campinas

Agradecimentos:

Agradecemos a colaboração e atenção dos amigos que contribuíram para a elaboração deste material de apoio.

Toda e qualquer contribuição será sempre muito bem-vinda.