



Cursos

➔ Listagem de disciplinas

Selecione uma disciplina

Aulas

- 01 Introdução a Banco de Dados
- 02 Modelo de Entidade e Relacionamento
- 03 Modelo Relacional
- 04 Transformações ER para MR
- 05 Transformações ER para MR e dicionário de dados
- 06 Normalização básica
- 07 Normalização avançada
- 08 Introdução à Linguagem SQL e Sistemas Gerenciadores de Banco de Dados
- 09 Linguagem SQL - criação, inserção e modificação de tabelas
- 10 Linguagem SQL - Consulta simples de tabelas
- 11 Linguagem SQL - Consulta avançada de tabelas
- 12 Linguagem SQL - Alteração da estrutura de tabelas e ambientes de múltiplas tabelas
- 13 Linguagem SQL - Subconsultas
- 14 Linguagem SQL - VISÕES
- 15 Linguagem SQL - STORED PROCEDURES
- 16 Linguagem SQL - Funções
- 17 Linguagem SQL - Segurança
- 18 Engenharia Reversa
- 19 Utilizando SQL em Java
- 20 Utilizando conceitos avançados de SQL em Java

⬅ Voltar 🖨 Imprimir ⬆ Topo

Sistemas de Banco de Dados

Aula 20 – Utilizando conceitos avançados de SQL em Java

Professores autores

Nélio Alessandro Azevedo Cacho
neliocacho@ect.ufrn.br
Xiankleber Cavalcante Benjamim
xianklebercb@gmail.com

Apresentacao

Nesta aula você irá aprender como inserir e atualizar registro no seu banco de dados MySQL através de aplicações em Java. Você também irá ver como passar parâmetros para comandos SQL. Finalmente, você irá aprender como chamar um procedimento armazenado através da sua aplicação em Java. Depois de aprender todos estes conceitos, você será capaz de realizar as principais operações SQL através da sua aplicação em Java.

Objetivo

Ao final desta aula, você será capaz de:

- Executar comandos de atualização, inserção e deleção de dados através da sua aplicação Java;
- Passar parâmetros para comandos SQL;
- Chamar procedimentos armazenados no banco de dados.

Executando comandos de inserção, deleção e atualização em Java

Na aula anterior você aprendeu como conectar sua aplicação em Java com o banco de dados MySQL. Ao final da aula passada, você terminou de implementar a classe `ConexaoMySQL`, que fornecia o método `getConexaoMySQL` responsável por criar uma conexão com o banco de dados MySQL. Nesta aula iremos continuar usando esta classe. Portanto, se você não implementou todas as funcionalidades da classe `ConexaoMySQL`, sugiro que você volte para a aula passada e conclua-a.

Bem, para começar, vamos definir na classe `ConexaoMySQL` mais um método chamado *inserir*. O objetivo deste método é permitir que dados sejam inseridos no banco de dados através de instruções SQL. A estrutura do método é mostrada a seguir:

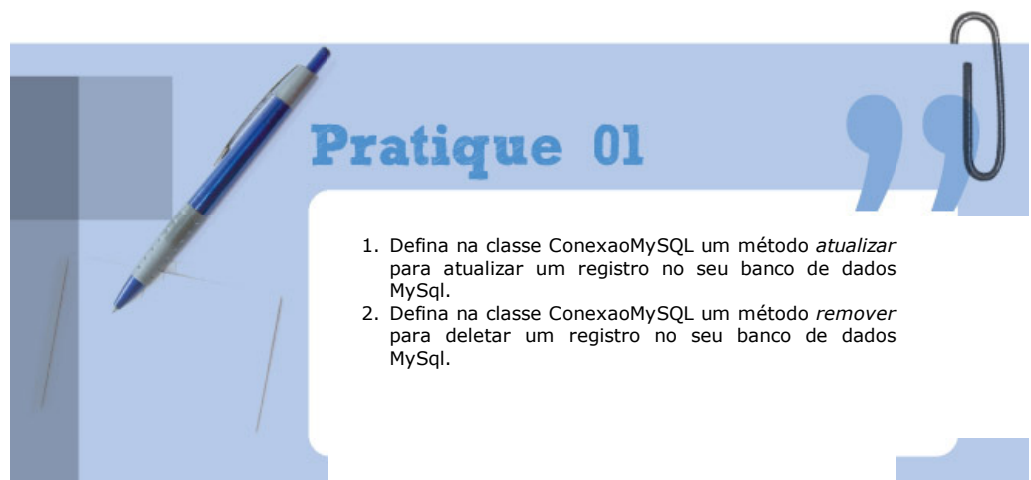
```
public void inserir(){
    Statement s = null;
    Connection connection = ConexaoMySQL.getConexaoMySQL();
    try {
        s = (Statement) connection.createStatement();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Note que até agora o método `inserir` usa exatamente as mesmas coisas que você já aprendeu para fazer o método `consultar`. Onde, então, estaria a diferença? A diferença é que, no método `consultar`, visto na aula passada, você invocava o método `executeQuery` da classe `Statement`. Agora, para realizar uma atualização (`update`), inserção (`insert`) ou deleção (`delete`) de um registro no banco de dados, você deve chamar o método `executeUpdate`. Por exemplo, para inserir um novo registro utilizando a instrução SQL `"INSERT INTO clientes (cli_codigo, cli_email, cli_nome, cli_cpf, cli_data_nasc, cli_sexo)VALUES (9, 'neliocado@ect.ufrn.br' , 'Nelio', '012345678-20', '1979-08-21', 'M')"` você deve invocar o método `executeUpdate` como mostrado abaixo:

```
int updateCount = s.executeUpdate ("INSERT INTO clientes (cli_codigo, cli_email, cli_nome,
cli_cpf, cli_data_nasc, cli_sexo)VALUES (9, 'neliocado@ect.ufrn.br' , 'Nelio', '012345678-20',
'1979-08-21', 'M')");
```

Note que o método `executeUpdate` recebe como parâmetro uma string que poderá conter um comando de atualização (`update`), inserção (`insert`) ou deleção (`delete`). O método `executeUpdate` retorna um valor inteiro informando quantos registros foram afetados pelo comando SQL. Veja que não existe o comando `executeInsert` e `executeDelete`. Você deve usar `executeUpdate` para todos os comandos SQL `INSERT`, `DELETE` e `UPDATE`. Abaixo você pode ver o código final do método `inserir`.

```
public void inserir(){
    Statement s = null;
    Connection connection = ConexaoMySQL.getConexaoMySQL();
    try {
        s = (Statement) connection.createStatement();
        int updateCount = s.executeUpdate("INSERT INTO clientes (cli_codigo, cli_email,
cli_nome, cli_cpf, cli_data_nasc, cli_sexo)VALUES (9, 'neliocado@ect.ufrn.br' , 'Nelio',
'012345678-20', '1979-08-21', 'M')");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



Passando parâmetros para comandos SQL

Em alguns casos, comandos SQL como consultas, atualizações, remoções ou inserções requerem muitos dados. Nestes casos, uma boa opção é utilizar parâmetros para simplificar a consulta e evitar trabalhar com a concatenação de Strings. Assim, você vai ver agora como passar parâmetros para comandos SQL.

Para começar vamos modificar o método `inserir` que você acabou de criar anteriormente. A modificação consiste em substituir a invocação do método `connection.createStatement()` por `connection.prepareStatement()`. Ao fazer isso, você terá que mudar o tipo da variável `s` para `PreparedStatement`. Veja a seguir como ficaria a parte inicial do seu novo método `inserir`.

```
public void inserir(){
    PreparedStatement s = null;
    Connection connection = ConexaoMySQL.getConexaoMySQL();
    try {

        s = (PreparedStatement) connection.prepareStatement("INSERT INTO clientes (cli_codigo,
cli_email, cli_nome, cli_cpf, cli_data_nasc, cli_sexo)VALUES (?, ?, ?, ?, ?, ?)");
```

Note que o comando SQL `INSERT` do `prepareStatement` usa `?` para determinar onde os parâmetros serão informados. Ou seja, agora, cada `?` no comando SQL representa um parâmetro. Com isso, você não precisa fornecer todos os valores quando definir um comando SQL. Estes valores poderão ser passados posteriormente na forma de parâmetros.

Para relacionar o valor com um parâmetro do seu comando SQL, você usa o método `set` da classe `PreparedStatement`. É importante você observar que para cada tipo de parâmetro você usa um método diferente. Por exemplo, para definir o valor do primeiro parâmetro do comando `INSERT` acima, você deve usar o comando `s.setInt(1, 11)`. Este comando informa que ao achar a primeira `?` no comando SQL, o Java deve substituí-la pelo valor `11`. O mesmo acontece se você executar o comando `s.setString(2, "ricardo@gmail.com")`, o Java vai substituir a segunda `?` pelo valor `"ricardo@gmail.com"`. Veja a seguir como definir o valor dos outros 4 parâmetros restantes.

```
s.setString(3, "Ricardo");
s.setString(4, "030585484-20");
s.setDate(5, new Date(1970,5,5));
s.setString(6, "M");
```

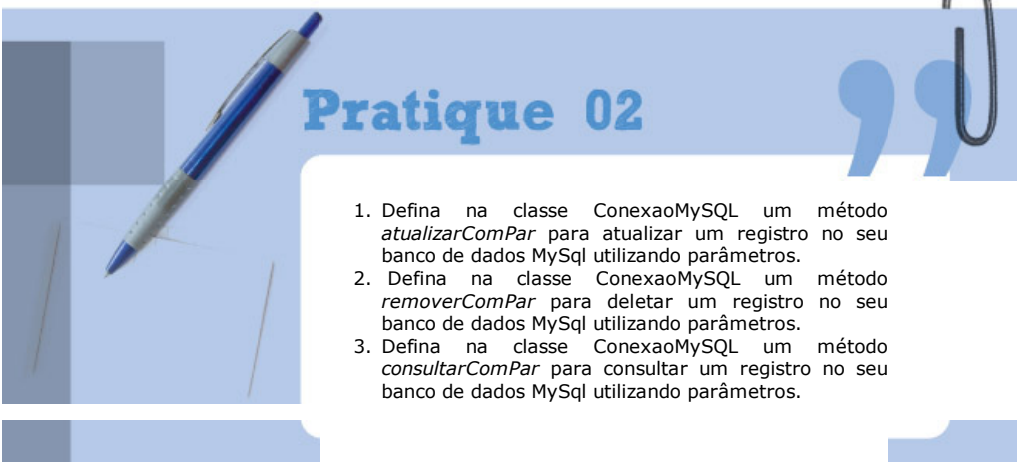
Depois de definir o valor de todos os parâmetros, você deve executar o comando `executeUpdate()`. Note que agora, diferentemente do exemplo mostrado na seção anterior, você não usa uma string de comando SQL. O novo método `inserir` pode ser visto logo a seguir:

```
public void inserir(){
    PreparedStatement s = null;
    Connection connection = ConexaoMySQL.getConexaoMySQL();
    try {
        s = (PreparedStatement) connection.prepareStatement("INSERT INTO clientes
(cli_codigo, cli_email, cli_nome, cli_cpf, cli_data_nasc, cli_sexo)VALUES (?, ?, ?, ?, ?, ?)");
        s.setInt(1, 11);
        s.setString(2, "ricardo@gmail.com");
        s.setString(3, "Ricardo");
        s.setString(4, "030585484-20");
        s.setDate(5, new Date(1970,5,5));
        s.setString(6, "M");
        int updateCount = s.executeUpdate();
    } catch (SQLException e) {
```

```

        e.printStackTrace();
    }
}

```



Pratique 02

1. Defina na classe `ConexaoMySQL` um método `atualizarComPar` para atualizar um registro no seu banco de dados MySQL utilizando parâmetros.
2. Defina na classe `ConexaoMySQL` um método `removerComPar` para deletar um registro no seu banco de dados MySQL utilizando parâmetros.
3. Defina na classe `ConexaoMySQL` um método `consultarComPar` para consultar um registro no seu banco de dados MySQL utilizando parâmetros.

Chamando procedimentos armazenados

Na Aula 16 você aprendeu como criar procedimentos armazenados no MySQL. Agora, você vai ver como executar um procedimento armazenado através de uma aplicação em Java. Para começar vamos criar um novo método na classe `ConexaoMySQL` com o nome `invocaProc`. Veja a seguir a parte inicial do método criado.

```

public void invocaProc(){
    PreparedStatement s = null;
    Connection connection = ConexaoMySQL.getConexaoMySQL();
}

```

Para chamar um procedimento armazenado você deve criar uma instância de uma classe `PreparedStatement` através do método `prepareCall()`. Ou seja, você deve invocar algo como: `s = (PreparedStatement) connection.prepareCall("{ call ValorID(?)})"`;

Note que o método `prepareCall` recebe como parâmetro a string `"call ValorID(?) "`, onde `ValorID` é o nome do procedimento armazenado que você deseja invocar no seu banco de dados MySQL. Veja também que temos uma interrogação (?) nesta chamada. Como você viu anteriormente, a interrogação equivale a um parâmetro. Ou seja, este procedimento armazenado possui um parâmetro que deve ser informado antes de chamar o procedimento armazenado. O valor do argumento é definido chamando-se o método `setString()` do objeto `CallableStatement`. Como este procedimento retorna um valor, temos que chamar o método `executeQuery()`, o qual retorna um `ResultSet` o resultado. Veja a seguir como fica o método `invocaProc`.


```

public void invocaProc(){
    PreparedStatement s = null;
    Connection connection = ConexaoMySQL.getConexaoMySQL();
    try {
        s = (PreparedStatement) connection.prepareCall("{ call ValorID(?)})";
        s.setString(1, "employee");
        s.executeQuery();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

Resumo

Nesta aula, aprendemos como executar comandos SQL (INSERT, DELETE e UPDATE) através da sua aplicação Java. Vimos também como utilizar parâmetros para reduzir os comandos SQL. Finalmente, você aprendeu como chamar um procedimento armazenado no seu banco de dados do MySQL.



Evolucao

Agora vamos relembrar e fixar nossa aula.

1. Quando eu devo utilizar parâmetros em comandos SQL?
2. Em qual situação eu devo usar o comando `executeQuery` e em qual eu devo utilizar `executeUpdate`?
3. Escreva uma aplicação em Java que realize as seguintes consultas ao seu banco de dados CursoX criado na Aula 10:
 - a. Remover todos os clientes do banco locadora.
 - b. Chamar um procedimento armazenado no banco locadora.

Referencias

CHAN, Mark C.; GRIFFITH, Steven W.; IASI, Anthony F. **Java**: 1001 dicas de programação. São Paulo: Makron Books, 1999.

DATE, C. J. **Introduction to Database Systems**. 7th ed. 1999.

DATE, Christopher J. **Introdução a sistemas de banco de dados**. Rio de Janeiro: Campus, 2000.

DEITEL, H. M.; DEITEL, P. J.; FURMANKIEWICZ, Edson. **Java, como programar**. 3. ed. Porto Alegre: Bookman, 2007.

ELMASRI, R. E.; NAVATHE, S. B. **Sistemas de banco de dados**. 4. ed. Rio de Janeiro : Addison-Wesley, 2005.

HEUSER, C. A. **Projeto de banco de dados**. 5. ed. Porto Alegre: Sagra-Luzzatto, 2004.

_____. **Projeto de banco de dados**. 6. ed. Porto Alegre: Editora Bookman, 2009.

GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de dados e algoritmos em Java**. 2. ed. Porto Alegre: Bookman, 2001.

[Voltar](#)[Imprimir](#)[Topo](#)