

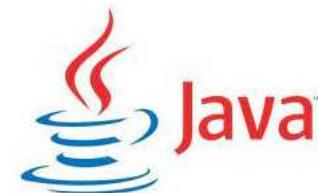


Residência em Tecnologia da Informação e Comunicação

JDBC: CRUD

Professor:

Alvaro Degas Coelho



INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



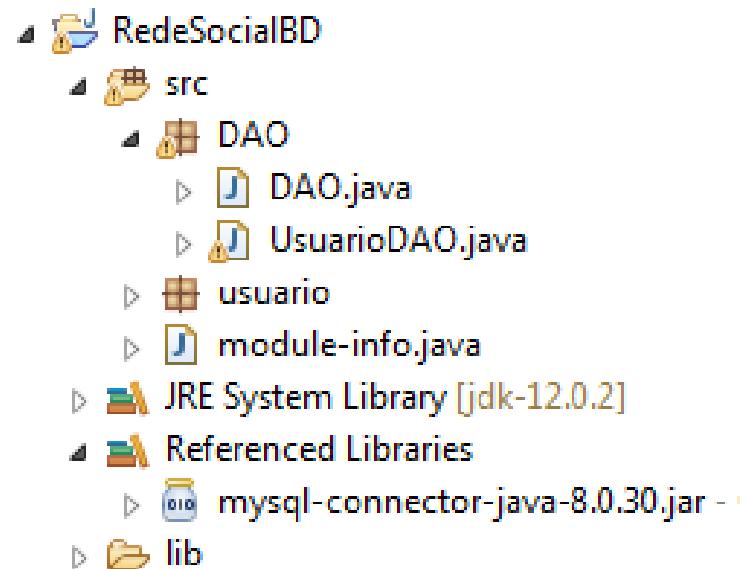


Residência
em Software

CRUD da Classe Usuário (da rede social)

Relembrando a configuração

- Projeto RedeSocial
 - Packages
 - Dao
 - usuario
 - Classes
 - dao.DAO
 - dao.UsuarioDAO
 - usuario.Usuario





A classe DAO

- Precisamos do método estático que cria um objeto do tipo Connection
 - Implementa a conexão com o BD
 - URL = jdbc(+URI)

```
package DAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DAO {

    private static final String URL = "jdbc:mysql://localhost:3306/teste";

    public static Connection criarConexao() {
        Connection con = null;
        try {
            con = DriverManager.getConnection(URL);
        } catch (SQLException e) {
            throw new RuntimeException("Falha ao conectar com o BD: ", e);
        }
        return con;
    }
}
```



- Já a implementamos como exercício
- Atributos (todos String)
 - Login, Senha e Email
- Métodos
 - getLogin(), setlogin()
 - getSenha(), setSenha()
 - getEmail(), setEmail()

A classe Usuario

```
package usuario;

public class Usuario {

    private String login, senha, email;

    public Usuario() {
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

Entendendo o PreparedStatement

- Conceito: ter uma sentença (statement) em SQL para enviar ao Banco de Dados
- Enviar ao BD <-> Haver uma conexão com o BD
 - PreparedStatement só pode existir a partir de uma Conexão
- Roteiro de uso típico
 - Usar a conexão para criar o PreparedStatement
 - Configurar o PreparedStatement de acordo com a operação desejada
 - Executar o PreparedStatement (Lembre que ele é linkado ao BD pelo objeto do tipo Conexão)
 - Receber os resultados



CRUD do Usuário

- C_{reate} R_ea_d U_pdate D_elete
- Primeira operação: Create
- Sentença SQL “padrão”
 - INSERT INTO Usuario (Login, Senha, Email) VALUES (**(1)**, **(2)**, **(3)**)
- Mas temos de substituir
 - (1) por getLogin()
 - (2) por getSenha()
 - (3) por getEmail()

Criando o PreparedStatement

- É necessário criar uma instância do PreparedStatement
 - Um construtor : usando o objeto Connection passando uma sentença SQL
 - Método prepareStatement(String query)
- É necessário haver pontos (chamaremos de Metacaracteres) para ligar a query (String) com os valores do objeto Java
 - Usa-se o caracter ‘?’ para isto
 - String query = “INSERT INTO Usuario (Login, Senha, Email)
VALUES (?, ?, ?)”

Criando o PreparedStatement

- É necessário substituir os Metacaracteres pelos valores corretos
 - Método `setString()`, `setDate()`, `setFloat()`, `setInt()`, `setNull()`, `setTime()` ...
- Cada um desses métodos (exceto `setNull()`) tem dois parâmetros
 - Um número indicando a posição correspondente do metacaracter
 - Começa de 1 (porque?)
 - O valor que será inserido naquela posição
- Como fica
 - `preparedStatement.setString(1, usuario.getLogin());`
 - `preparedStatement.setString(2, usuario.getSenha());`
 - `preparedStatement.setString(3, usuario.getEmail());`



Crie seu método **create** (da classe **UsuarioDAO**)

- Estático (sugerido)
- Lembre de instanciar um objeto do tipo Connection
 - DAO.criarConexao()
- Lembre de criar uma Query (String) usando os metacaracteres
 - String query = “INSERT ... VALUES (?, ?, ?)”
- Lembre de criar um PreparedStatement (usando o objeto Connection)
 - Isto vai precisar de TryCatch para as exceções do SQL (java.sql)
 - Com.PrepareStatement(query)
- Lembre de substituir os metacaracteres
 - preparedStatement.setString(...)



Residência
em Software

**Ainda não está pronto
para executar**

Verifique sua sentença SQL

- Método `toString()` do objeto `PreparedStatement`
 - Use para ver se a sentença está ok
- Na classe `UsuarioDao`
 - Crie o método `main`
 - Instancie um objeto `Usuario`
 - Execute `UsuarioDAO.create`
 - Veja (usando o `toString()` do `preparedStatement`) a sentença sql
 - Certifique-se de que a sentença está correta

Executar a sentença

- Abra o módulo de administração de seu BD
 - PHP Admin
- Observe se a tabela Usuario está perfeitamente criada (#Login, Senha, Email)
- No método UsuarioDAO.create()
 - Comande a execução da sentença SQL
 - preparedStatement.execute()
- Verifique se tudo correu bem
- Cheque se o dado foi inserido corretamente

READ Recuperando (Lendo) dados

- Ler dados tem vários formatos possíveis
- Os mais típicos
 - Ler Todos (Read All)
 - Retorna a lista completa de algum repositório (Tabela, View, etc.)
 - Localizar um
 - Normalmente pela chave primária

Como?

- O método readAll() vai selecionar todos os usuários
 - SELECT Login, Senha, Email FROM Usuario
 - Select * também serviria. Mas...
- Temos que instanciar um PreparedStatement
 - Portanto temos que ter um objeto Connection
 - Portanto temos que executar DAO.criarConexão()
- Veja no método create() como fizemos

PreparedStatement: Select

- Desta vez estamos trazendo Dados
 - Onde guardá-los?
- Objeto ResultSet (java.sql)
 - Recebe o resultado de uma consulta executada por um objeto PreparedStatement
 - Select
 - Método executeQuery() do objeto PreparedStatement
- ResultSet implementa um CURSOR

CURSORES

- Do padrão SQL
 - Um conjunto de Tuplas (Linhas)
 - Indexado
 - É possível iterar
- Uso padrão (SQL)
 - Declara o cursor (resultado de um Select)
 - OPEN – Abre o cursor
 - FETCH – Recupera a primeira tupla e localiza o cursor na tupla seguinte
 - Checa o Status SQL para verificar se ainda há dados
 - CLOSE – Fecha o cursor

ResultSet em Java

- Implementa o cursor
- Instanciado pelo método executeQuery() do objeto PreparedStatement
- Métodos importantes
 - next(): avança o cursor para a tupla seguinte (iterator)
 - Retorna true ou false se há ou não dados a serem recuperados
 - getString(), getDate(), getFloat(), getInt(), getNull(), getTime() ...
 - Cada um destes métodos recebe um String identificando qual a Coluna (atributo) está sendo referenciado



Crie seu método `readAll(da classe UsuarioDAO)`

- Estático (sugerido)
- Lembre de instanciar um objeto do tipo Connection
 - DAO.criarConexao()
- Lembre de criar uma Query (String)
 - String query = “SELECT ...”
- Lembre de criar um PreparedStatement (usando o objeto Connection)
 - Isto vai precisar de TryCatch para as exceções do SQL (java.sql)
 - Com.PrepareStatement(query)

Crie seu método `readAll(da classe UsuarioDAO)`

- Lembre de executar (como consulta) seu prepared Statement
 - `preparedStatement.executeQuery()`
 - O resultado fica num objeto (cursor) `ResultSet`
- Lembre de percorrer seu `ResultSet` (`while ... Next()`)
 - Recuperar o Login: `getString("Login")`
 - Recuperar a senha: `getString("Senha")`
 - Recuperar o email: `getString("Email")`
 - Usar isto para criar um objeto do tipo Usuário
 - Incluir o Usuário numa lista de usuários
- Ao final, retorne a lista de usuários criada



Residência
em Software

**Execute sua consulta
(método main) e veja se
retornou a lista correta**

Recuperando um dado específico

- Segunda “versão” do Read
 - `readById(String login)`
 - Busca pela chave primária (Login)
- Conceitualmente é igual
- Duas diferenças fundamentais
 - Usar metacaracteres: “SELECT Login, Senha, Email FROM Usuario where Login = ?”
 - Retorna um objeto do tipo Usuario (e não uma lista)
 - Mas precisa criar um ResultSet do mesmo jeito



Residência
em Software

**Execute sua consulta
(método main) e veja se
retorna o usuário correto**

UPDATE: alterando dados

- A alteração de dados normalmente impede que se altere a chave primária
 - Custo com chaves estrangeiras e índices
 - Nestes casos é recomendável excluir e incluir novamente
- Todos os demais campos são potencialmente (mas não necessariamente) alteráveis
- A alteração tipicamente se refere a uma tupla especificamente
 - Identificada pela chave primária (cláusula WHERE)



UPDATE: alterando dados

- Tanto os novos valores (possivelmente iguais) a serem alterados quanto a chave primária usada no WHERE serão metacaracteres
 - No nosso caso
 - “UPDATE Usuario SET Senha=?, Email=? WHERE Login=?”

Crie seu método update(Usuario usuario)

- Connection (criado pelo DAO)
 - Query com metacaracteres
- PreparedStatement (criado pelo Connection)
 - Alterar os metacaracteres usando os gets() do objeto Usuário
- Executar o PreparedStatement
 - Try Catch – observar erros por violação de regras de integridade

DELETE: excluindo dados

- Dica: não é comum se excluir realmente dados. Normalmente eles são tornados “inativos”
- Assim como o update: normalmente é excluída uma tupla específica
 - Identificada por chave primária
- Duas opções possíveis
 - update(String Login) ou update(Usuario usuario)
 - A diferença não interfere substancialmente na lógica.
 - A segunda opção é mais encapsulada
 - Considere a hipótese de a chave primária mudar

Crie seu método delete(Usuario usuario)

- Connection (criado pelo DAO)
 - Query com metacaracter (chave primária)
- PreparedStatement (criado pelo Connection)
 - Alterar os metacaracter usando o get() do objeto Usuário
- Executar o PreparedStatement
 - Try Catch – observar erros por violação de regras de integridade
 - Aqui não é o caso. Mas se houvesse uma chave estrangeira?



Residência
em Software

Falando em chave estrangeira...

Crie uma lista de postagens para a classe Usuario

- private ArrayList<String> postagens
- No construtor
 - Postagens = new ...
- Métodos
 - getPostagens() – recupera toda a lista de postagens do usuário
 - setPostagens() – insere uma lista inteira de postagens para um usuário
 - addPostagem(String postagem) – acrescenta uma postagem específica

E no BD

- No modelo Relacional não há atributos multivalorados
 - Por padrão: Há implementações
- Solução mais utilizada: ORM (veremos)
- Solução Raiz (e mais leve)
 - Criar a implementação
- Padrão
 - Criar uma tabela relacionamento



Residência
em Software

Usuario(#Login, Senha, Email) Postagem(#id, \$Login, Texto)

Tõe	Toentrando	toe@tutu.com
Sara	Sarada	sara@tutu.com
Tico	TucoTuca	tico@tutu.com
Peu	Peucador	peu@tutu.com
Lia	Liamando	lia@tutu.com
Mara	Maravilhosa	lara@tutu.com

1	Sara	Alô Mundo
2	Sara	Hoje comi frango com Farofa
3	Peu	Meu amor não me quer
4	Lia	Viajei para a Europa
5	Tõe	Viajei para os Estados Humildes

Não esquecer de implementar a Chave Estrangeira no seu BD!!!

O que muda?

- Método create de Usuário
 - Novo PreparedStatement para inserir as postagens
 - Insert na Tabela Postagem
 - Percorrer todas as postagens
 - Criar a sentença correta
 - Login <- usuario.getLogin()
 - Postagem vem da lista
 - Executar o preparedStatement

O que muda?

- Método readById e readAll de Usuário
 - Novo PreparedStatement para recuperar as postagens
 - Select na Tabela Postagem (usando o login do usuário atual)
 - Novo resultset
 - Percorrer todas as postagens
 - Recuperar a postagem
 - usuario.addPostagem()
- Método delete de Usuário

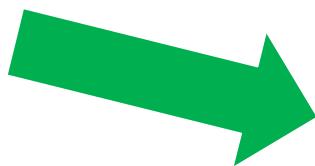


Residência
em Software

Alternativa seguindo o Padrão: Um DAO para Postagens



Residência
em Software



```
public static boolean create(Usuario usuario) {
    DAO dao = new DAO();
    Connection con = dao.criarConexao();
    String query = "INSERT INTO Usuario (Login, Senha, Email) VALUES (?, ?, ?)";

    try (PreparedStatement preparedStatement = con.prepareStatement(query)) {
        preparedStatement.setString(1, usuario.getLogin());
        preparedStatement.setString(2, usuario.getSenha());
        preparedStatement.setString(3, usuario.getEmail());
        preparedStatement.execute();
        for (String post: usuario.getPostagens()) {
            PostagemDAO.create(usuario, post);
        }
        return true;
    } catch (SQLException e) {
        System.out.println("Erro ao adicionar o cliente: " + e);
        return false;
    } finally {
        dao.closeConnection(con);
    }
}
```

PostagemDAO

- Basicamente tem as mesmas funcionalidades de UsuarioDAO
- Observações
 - É necessário receber o usuário (ou pelo menos o Login) para os métodos insert, update e delete
 - É necessário um método ReadByUser pararetornar a lista de postagens de um usuário
 - Usar em setPostagend() da classe Usuário
 - É desnecessário métodos ReadAll() e ReadById()

Observações

- Seguir o Padrão: vantagens
 - Melhora a limpeza de seu código
 - Desacopla as operações de UsuarioDao e PostagensDao
 - Aumenta a coesão
- Engenharia de Software: muito bom!
- Desvantagens
 - Abre duas conexões com o BD para executar uma única operação
 - Pode gerar inconsistências de dados