

Disciplina: Desenvolvimento de Software

Aula 7: Java e Banco de Dados

Apresentação

Os bancos de dados relacionais são o meio mais comum para armazenamento de dados cadastrais. Esse tipo de banco é apoiado na álgebra relacional e no cálculo relacional, tratando de um ferramental consistente e amadurecido no mercado, e utilizando a linguagem de consulta SQL para efetuar as operações necessárias.

No entanto, há vários fornecedores de banco, como Oracle, IBM DB2 e SQL Server, e, para não criar uma versão do programa para cada fornecedor, as ferramentas de programação adotaram o uso de Middleware, como o JDBC.


Além do uso de Middleware, atualmente procuramos concentrar os comandos SQL em classes DAO, de forma a evitar a proliferação desses comandos ao longo de todo o código-fonte.

Objetivos

- Reconhecer as características de bancos relacionais e sintaxe SQL;
- Examinar o conceito de Middleware e características do JDBC;
- Usar o JDBC e padrão DAO para acesso ao banco de dados.

Bancos de dados relacionais

Podemos dizer que um banco de dados relacional é um repositório de dados capaz de manter relacionamentos consistentes entre os elementos armazenados.

 (Fonte: Shutterstock).

Os fundamentos matemáticos que definem a funcionalidade dos bancos de dados são a álgebra relacional e o cálculo relacional.

A álgebra relacional trata de conjuntos e operações que podem ser feitas sobre os mesmos, como união, interseção e subtração, além de operações específicas, como projeção, seleção e junção.

Podemos observar uma junção natural a seguir:



Seguindo a mesma linha, o cálculo relacional realiza consulta aos dados através de expressões formais, além de incluir algumas palavras novas, como “Existe” e “Para Todo”.

Como exemplo, poderíamos efetuar uma consulta do tipo “obter todos os coordenadores de cursos onde existe ao menos um aluno inscrito”:

Exemplo

A partir do conhecimento acerca desses elementos matemáticos que fundamentam os bancos relacionais, podemos compreender melhor a ferramenta de consulta e manipulação de dados comum a todos esses bancos, ou seja, a linguagem de consulta estruturada (SQL).

Os bancos relacionais não são os únicos existentes, mas são o tipo de banco mais comum em termos de cadastros, já que permitem uma estruturação de dados bem-organizada, com consultas muito ágeis, e fundamentadas em procedimentos matemáticos já consolidados e amadurecidos ao longo de décadas.

Linguagem SQL

 Fonte: <https://www.thinkwithgoogle.com>

Para trabalharmos com bancos de dados, é interessante conhecer ao menos o básico dos comandos do SQL, os quais podem ser divididos em três áreas:

- DDL (Data Definition Language);
- DML (Data Manipulate Language);
- Seleção ou consulta.

Os comandos DDL são responsáveis pela criação das estruturas que receberão os dados e manterão os relacionamentos de forma consistente, tendo como elementos principais as tabelas e índices. Basicamente, utilizamos os comandos CREATE, ALTER e DROP para a criação de uma tabela.

Exemplo

```
CREATE TABLE Curso (  
  ID_CURSO int NOT NULL Primary Key,  
  NOME_CURSO varchar(20),  
  COORDENADOR varchar(40)  
);
```

Nesse exemplo, nós criamos uma tabela chamada Curso, cujas colunas são ID, numérico que funciona como chave primária, NOME_CURSO e COORDENADOR, estas últimas do tipo texto.

Após a criação das estruturas, podemos utilizar os comandos **DML** para a inclusão, alteração e exclusão de registros, o que é efetuado, respectivamente, pelos comandos **INSERT, UPDATE e DELETE**.

Poderíamos criar um registro na tabela Curso com o comando apresentado a seguir.

Exemplo

```
INSERT INTO Curso (ID_CURSO, NOME_CURSO, COORDENADOR)  
VALUES (1, "Engenharia","Alberto")
```

Além dos comandos para manipulação estrutural e gerenciamento de dados, as seleções através do comando SELECT talvez tenham o papel mais relevante em termos se SQL.

Esse comando se divide, inicialmente, em duas partes principais, que são a projeção (campos) e a restrição (condições), como podemos ver no exemplo a seguir:

```
SELECT NOME_CURSO FROM Curso  
WHERE COORDENADOR LIKE 'A%'
```

O nome do curso é selecionado a partir da tabela Curso (projeção) apenas para os registros onde o nome do coordenador começa com a letra “A” (restrição).

O comando SELECT é bastante amplo, e aceita elementos para ordenação, agrupamento, combinação, operações de conjunto, entre diversos outros.

Alguns dos operadores específicos do SQL são apresentados na tabela seguinte.

Operador	Utilização
IN	Condiciona à ocorrência do valor do campo em um conjunto de valores
NOT IN	Condiciona à inexistência do valor do campo em um conjunto de valores
LIKE	O valor do campo deve estar de acordo com um padrão, sendo tipicamente utilizado em situações do tipo “começado com”
EXISTS	Verifica uma condição de existência relacionada ao campo
NOT EXISTS	Verifica uma condição de inexistência relacionada ao campo
BETWEEN	Verifica se o valor se encontra entre dois limites
ALL	Retorna o valor caso todos os elementos do conjunto satisfaçam à condição
ANNY	Retorna o valor caso algum elemento do conjunto satisfaça à condição.

Podemos ainda utilizar os operadores de comparação tradicionais, como **menor que, igualdade e maior que**. Também são permitidas combinações lógicas com o uso de **AND, OR e NOT**.

Comentário

É possível agrupar campos com **GROUP BY**, e utilizar operações de sumarização, como **MAX, MIN e COUNT**, além da possibilidade de aplicar restrições aos grupos formados com o uso de **HAVING**.

Não menos importante, podemos ordenar os resultados por quaisquer campos, de forma ascendente ou descendente, com o uso de **ORDER BY**.

Middleware

Para definirmos **middleware**, devemos entender os conceitos de **front-end e back-end**.

Caracterizamos o front-end como uma camada de software responsável pelo interfaceamento do sistema, normalmente utilizando uma linguagem de programação para viabilizar a criação desta interface.	Já o back-end se refere ao conjunto de tecnologias que podem ser acessadas a partir de nosso front-end, mas que não são programadas no seu nível, como os bancos de dados e as mensagerias.
---	--

Em nosso contexto, teremos como front-end os programas criados em Java e como back-end um banco de dados. Nossos programas Java deverão enviar comandos SQL para que o banco processe e retorne os dados que deverão ser exibidos para o usuário.

As mensagerias são outro exemplo de back-end, viabilizando a comunicação assíncrona entre sistemas corporativos através do envio de mensagens. É uma tecnologia crucial para alguns ramos, como a rede bancária.

Um grande problema enfrentado pelas linguagens mais antigas é que deveríamos ter versões específicas do programa para acesso a cada tipo de servidor de banco de dados, e há muitos no mercado, como Oracle, Informix, DB2, SQL Server, MySQL, Jasmine, Btrieve, entre diversos outros.

Com diferentes componentes de acesso e programações diferenciadas, a probabilidade de ocorrência de erros é simplesmente enorme. Pensando nisso, surge o conceito de middleware, que trata de uma camada de software intermediária para promover a comunicação entre o front-end e o back-end, de forma a deixar essa integração transparente.

Com o uso de um middleware, o programador irá enviar os comandos necessários para esse componente a partir do front-end e, com a configuração correta, o middleware irá assumir a responsabilidade de enviar os comandos para o back-end escolhido.

Com isso, teremos apenas uma versão do sistema e diversas configurações de conexão para o middleware, diminuindo enormemente a probabilidade de erros.

JDBC

Podemos dizer que o JDBC (**Java Database Connectivity**) é o middleware do Java para acesso a bancos de dados. Permite que utilizemos os mais diversos bancos, sem modificações no código Java, desde que aceitem o uso de SQL ANSI.

Antes de começar a programar no Java, temos de criar o banco que será empregado. Para isso, utilizaremos a interface do NetBeans e acesso ao **Derby** Database.

Apache Derby é um subprojeto do Apache DB, constituindo um banco de dados relacional implementado totalmente em Java. Disponível sob licença Apache, pode ser embutido em programas Java, bem como utilizado para transações on-line.

Na guia de **Serviços** do NetBeans é possível efetuar diversos controles associados ao tempo de execução, como o acompanhamento de servidores, acesso a repositórios e manipulação de bases de dados diversas, entre elas o Derby (**Java DB**).

Podemos observar a guia de Serviços com o Java DB selecionado a seguir.

Exemplo

A criação de um banco de dados novo nesse ambiente é bem simples. Basta clicar com o botão direito sobre o driver Java DB, selecionável com a abertura da árvore de Bancos de Dados, e escolher a opção “Criar Banco de Dados...”. Com isso, teremos a abertura da janela apresentada a seguir.

Nessa janela, preencheremos o nome de nosso novo banco de dados com o valor “LojaEAD”, bem como usuário e senha, onde sugiro que seja preenchido também com “LojaEAD” para ambos. Ao clicar em “OK”, será criado o banco e ficará disponível para conexão, como pode ser observado na figura seguinte.

O banco é identificado por sua **Connection String**, no caso indicando a própria máquina (**localhost**) através da porta padrão (**1527**) e instância **LojaEAD**.

O passo seguinte será abrir esse banco de dados clicando com o botão direito e escolhendo a opção “**Conectar...**”.

Com o banco aberto, poderemos criar uma tabela, navegando até o esquema **LOJAEAD** na divisão **Tabelas**, e clicando novamente com o botão direito para selecionar a opção “**Criar Tabela...**”, como podemos observar na figura seguinte.

Na janela que se abrirá, iremos configurar uma tabela de nome **Produto**, com os seguintes campos:

- código (inteiro, chave primária);
- nome (varchar, tamanho 40);
- quantidade (inteiro).

Para adicionar os campos, deve ser clicado o botão “**Adicionar coluna**”, o qual abrirá a tela que podemos observar a seguir.

Definindo o nome da tabela e adicionando os campos, teremos a configuração final que pode ser observada a seguir, faltando apenas clicar em “**OK**” para finalizar o processo.

Ao término, o NetBeans irá executar o comando **CREATE TABLE** necessário para a geração da tabela, que ficará disponível na divisão **Tabelas**, permitindo que acionemos a opção “**Exibir Dados...**” a partir do clique com o botão direito, como podemos observar na figura seguinte.

Uma janela de comandos SQL e visualização de dados será aberta na área do editor de código do NetBeans, e poderemos acrescentar registros com o uso Alt+I ou clique sobre o botão equivalente (). Podemos observar a tela para inserção de registro a seguir.

Nessa tela, podemos preencher os valores para um novo registro, e, se quisermos mais de um, basta clicar em “**Adicionar...**”.

Após o preenchimento, basta clicar em “**OK**” e o NetBeans executará os comandos **INSERT** necessários. Devemos incluir alguns produtos em nossa tabela, para que os testes com código sejam mais significativos.

Com relação à codificação Java, os componentes do JDBC estão presentes no pacote java.sql, e o processo para utilização segue quatro passos simples:

- 1 Instanciar a classe do **driver** de conexão;
- 2 Obter uma conexão (**Connection**) a partir da **Connection String**, usuário e senha;
- 3 Instanciar um executor de SQL (**Statement**) ;
- 4 Executar os comandos **DML**.

Atenção

Para o comando de seleção, há mais um detalhe, que seria a recepção da consulta em um ResultSet.

Isso pode ser observado no exemplo seguinte onde, aproveitando a base criada anteriormente, vamos efetuar uma consulta aos dados inseridos.

```
package testebase;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class TesteBase {

    public static void main(String[] args) throws Exception {

        Class.forName("org.apache.derby.jdbc.ClientDriver");

        // passo 1

        Connection c1 = DriverManager.getConnection(
            "jdbc:derby://localhost:1527/LojaEAD",
            "LojaEAD", "LojaEAD");

        // passo 2

        Statement st = c1.createStatement();

        // passo 3

        ResultSet r1 = st.executeQuery("SELECT * FROM PRODUTO");

        // passo 4 e recepção no ResultSet

        while(r1.next())

            System.out.println("Produto "+r1.getInt("codigo")+": "+
                r1.getString("nome")+ "::"+r1.getInt("quantidade"));

        r1.close();
        st.close();
        c1.close();

    }

}
```

Podemos acompanhar, nesse código, os quatro passos citados para a conexão e utilização do banco de dados, e, após a recepção dos dados da consulta no ResultSet, podemos nos movimentar pelos registros, acessando cada campo pelo nome, sempre lembrando de utilizar o método correto para o tipo.

```
while(r1.next())
System.out.println("Produto "+r1.getInt("codigo")+": "+
    r1.getString("nome")+ "::"+r1.getInt("quantidade"));
```

Ao efetuar a consulta, o ResultSet fica posicionado antes do primeiro registro, na posição **BOF (Beginning of File)**, e com o uso do comando **next** podemos mover sempre para os próximos registros, a partir do primeiro, até atingir a posição **EOF (End of File)**.

Na parte final, devemos fechar os componentes relacionados ao banco na ordem inversa daquela em que foram criados, já que existe dependência sucessiva entre eles.

Para implementar esse exemplo, devemos criar um novo projeto do tipo **Java..Aplicação Java** no NetBeans, e nomeá-lo como **TesteBase**, deixando todas as opções padrão marcadas.

Após digitar o código, devemos adicionar a biblioteca de acesso ao Derby no projeto, através do clique com o botão direito na seção **Bibliotecas** e escolher a opção **“Adicionar Biblioteca...”**.

A seguir, podemos observar a janela que será aberta, onde deveremos selecionar o item **Driver do Java DB** e clicar em **“Adicionar Biblioteca”**.

Por fim, podemos executar o projeto e observar a saída proporcionada pelo mesmo.

```
Produto 1: Banana::1000
Produto 2: Morango::150
Produto 3: Laranja::400
```


Padrão DAO

Agora que sabemos efetuar as operações com o banco de dados, seria interessante organizar a forma de programar, pois é fácil imaginar a dificuldade que seria dar a manutenção em um sistema com dezenas de milhares de linhas de código Java, com comandos SQL espalhados ao longo dessas linhas.

A linguagem Java é orientada a objetos, e torna-se mais fácil representar uma tabela através de uma classe, onde suas instâncias corresponderão aos registros existentes.

Exemplo

```
public class Produto {
    public int codigo;
    public String nome;
    public int quantidade;
    public Produto(){ }
    public Produto(int codigo, String nome, int quantidade) {
        this.codigo = codigo;
        this.nome = nome;
        this.quantidade = quantidade;
    }
}
```

Com base na observação de que o SQL espalhado ao longo do código traz grandes dificuldades, foi desenvolvido o padrão DAO (Data Access Object) com o objetivo de concentrar as instruções SQL em um único tipo de classe, agrupando e reutilizando os diversos comandos relacionados ao banco de dados. Podemos observar a classe ProdutoDAO a seguir:

Exemplo

```
public class ProdutoDAO {
    private Connection getConnection() throws Exception{
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        return DriverManager.getConnection(
            "jdbc:derby://localhost:1527/LojaEAD",
            "LojaEAD", "LojaEAD");
    }
    private Statement getStatement() throws Exception{
        return getConnection().createStatement();
    }
    private void closeStatement(Statement st) throws Exception
    {
        st.getConnection().close();
    }
    public List<Produto> obterTodos(){
        ArrayList<Produto> lista = new ArrayList<>();
        try {
            ResultSet r1 = getStatement().executeQuery(
                "SELECT * FROM PRODUTO");
            while(r1.next())
                lista.add(new Produto(r1.getInt("codigo"),
                    r1.getString("nome"),r1.getInt("quantidade")));
            closeStatement(r1.getStatement());
        }catch(Exception e){
        }
        return lista;
    }
}
```

Observe que iniciamos criando os métodos **getStatement** e **closeStatement**, com o objetivo de gerar executores de SQL e eliminá-los, efetuando também as conexões e desconexões nos momentos necessários.

Outro método que podemos observar é o **getConnection**, utilizado apenas para encapsular o processo de conexão com o banco.

O método **obterTodos** irá retornar todos os registros da tabela Produto no formato de um **ArrayList**. Inicialmente é executado o SQL necessário e, para cada registro obtido no cursor, é gerado um novo objeto da classe Produto e adicionado à lista de retorno.

```
ResultSet r1 = getStatement().executeQuery(
    "SELECT * FROM PRODUTO");
while(r1.next())
    lista.add(new Produto(r1.getInt("codigo"),
        r1.getString("nome"),r1.getInt("quantidade")));
```

Utilizando a classe ProdutoDAO, o código da classe principal será extremamente simplificado.

Exemplo

```
public class TesteBase {
    public static void main(String[] args)throws Exception {
        ProdutoDAO dao = new ProdutoDAO();
        dao.obterTodos().forEach((p) -> {
            System.out.println("Produto "+p.codigo+": "+p.nome+
                "::~"+p.quantidade);
        });
    }
}
```

Note que foi possível utilizar o operador funcional **forEach**, o qual navegará por todos os elementos do **ArrayList** fornecido a partir de **obterTodos**, imprimindo os dados de cada um dos produtos ali encontrados de uma forma muito mais simples e organizada.

Para trabalhar com **JSF**, teremos de efetuar algumas alterações na classe **Produto**, adicionando **getters e setters**, além das anotações de escopo e mapeamento.

Exemplo

```
@Named(value="produto")
@RequestScoped
public class Produto implements Serializable{
    private int codigo;
    private String nome;
    private int quantidade;
    public Produto(){ }
    public Produto(int codigo, String nome, int quantidade) {
        this.codigo = codigo;
        this.nome = nome;
        this.quantidade = quantidade;
    }
    public int getCodigo() { return codigo; }
    public void setCodigo(int codigo) { this.codigo = codigo; }
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
    public int getQuantidade() { return quantidade; }
    public void setQuantidade(int quantidade)
{ this.quantidade = quantidade; }
}
```

Também é necessário alterar a classe **ProdutoDAO**, efetuando o acréscimo das anotações para mapeamento e escopo.

```
@Named(value="produtoDAO")
@SessionScoped
public class ProdutoDAO implements Serializable{
```

Além dessas mudanças, focando a utilização junto ao framework JSF, iremos adicionar os métodos para efetuar a inclusão e a exclusão.

```
public void excluir(int codigo){
    try {
        Statement st = getStatement();
        st.executeUpdate("DELETE FROM PRODUTO WHERE CODIGO = "+
            codigo);
        closeStatement(st);
    }catch(Exception e){
    }
}
```

Podemos observar que a deleção utiliza o Statement tradicional, apenas com a diferença de que em vez de utilizar **executeQuery**, como nas seleções de dados, utilizamos o método **executeUpdate**, por se tratar de um comando DML.

```
public void incluir(Produto p){
    try {
        PreparedStatement ps = getConnection().prepareStatement(
"INSERT INTO PRODUTO VALUES(?,?,?)");
        ps.setInt (1, p.getCodigo());
        ps.setString(2, p.getNome());
        ps.setInt (3, p.getQuantidade());
        ps.executeUpdate();
        closeStatement(ps);
    }catch(Exception e){
    }
}
```

Já na inclusão, foi utilizado um elemento do tipo **PreparedStatement**, que permite a definição de parâmetros, facilitando a escrita sem a preocupação com o uso de apóstrofe ou outro delimitador do SQL, particularmente útil quando tivermos de trabalhar com datas.

Para definir os parâmetros, utilizamos pontos de interrogação, os quais assumem valores posicionais, a partir de um, o que é um pouco diferente da indexação dos vetores, que começa em zero.

Os parâmetros são preenchidos, de acordo com seus tipos, antes da chamada, efetuada com **executeUpdate**.

```
ps.setInt (1, p.getCodigo());
ps.setString(2, p.getNome());
ps.setInt (3, p.getQuantidade());
```

Feitas essas mudanças, as classes Produto e ProdutoDAO estarão preparadas para o uso com JSF e, como o servidor GlassFish já traz as bibliotecas de acesso ao Derby, não precisam ser adicionadas ao projeto do tipo Web.

Para testar nossas classes, vamos criar um projeto Web (**WebTesteBanco**), com uso de framework **JSF** e adicionar Produto e ProdutoDAO ao projeto.

Em seguida, vamos modificar o index para a programação seguinte.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <h:form>
      Código:<br/>
      <h:inputText value="#{produto.codigo}"/><br/>
      Nome:<br/>
      <h:inputText value="#{produto.nome}"/><br/>
      Quantidade:<br/>
      <h:inputText value="#{produto.quantidade}"/>
      <p><h:commandButton
        action="#{produtoDAO.incluir(produto)}"
        value="Incluir"/>
      </p>
      <hr/>
      <h:dataTable value="#{produtoDAO.obterTodos()}" var="p" border="1">
        <h:column>#{p.codigo}</h:column>
        <h:column>#{p.nome}</h:column>
        <h:column>#{p.quantidade}</h:column>
        <h:column><h:commandButton
          action="#{produtoDAO.excluir(p.codigo)}"
          value="Excluir"/>
        </h:column>
      </h:dataTable>
    </h:form>
  </h:body>
</html>
```

Inicialmente, todo o conteúdo interno fica dentro de <h:form>, de forma a viabilizar o uso dos componentes de entrada e botões.

Na parte superior da janela teremos os campos para cadastro de produto, onde o value de cada campo estará associado à propriedade equivalente no objeto produto.

Esse objeto tem escopo de requisição, mantendo-se os valores na chamada até que a resposta seja enviada pelo cliente.

```
Código:<br/>
<h:inputText value="#{produto.codigo}"/><br/>
Nome:<br/>
<h:inputText value="#{produto.nome}"/><br/>
Quantidade:<br/>
<h:inputText value="#{produto.quantidade}"/>
```

Em seguida, há um botão que aciona o método incluir, de **produtoDAO**, passando **produto** como parâmetro. É interessante observar como o JSF mantém a relação bilateral entre o campo visual e o atributo do bean.

```
<h:commandButton action="#{produtoDAO.incluir(produto)}"
value="Incluir"/>
```

Já na parte inferior, teremos um DataTable para exibir os valores, onde a lista de objetos da classe Produto são fornecidos através de obterTodos, do objeto produtoDAO, além da denominação, através do atributo var, de uma variável para percorrer a lista.

```
<h:dataTable value="#{produtoDAO.obterTodos()}" var="p" border="1">
```

Com o uso da variável, podemos acessar os atributos de cada objeto da lista, exibindo os dados no DataTable e até mesmo criando botões de exclusão para cada registro de forma dinâmica.

A exclusão, nesse caso, é feita com a chamada a **excluir**, de **produtoDAO**, passando como parâmetro o código do objeto corrente.

```
<h:column>#{p.quantidade}</h:column> <h:column><h:commandButton action="#{produtoDAO.excluir(p.codigo)}" value="Excluir"/>
</h:column>
```

Podemos observar o resultado final na figura seguinte.

Por meio da adoção do DAO, conseguimos concentrar os comandos SQL em classes específicas, facilitando muito a programação dos níveis superiores do sistema, além de promover grande reúso.

Isso ficou evidente no aproveitamento das classes para um projeto Web com JSF sem a necessidade de grandes modificações no código.

Atividade

1 - Segundo a estrutura básica de uma consulta com o comando SELECT, efetuando acesso a uma única tabela e sujeita a poucas condições, quais são os elementos mínimos necessários em termos da álgebra relacional e do cálculo relacional?

- a) União e subtração
- b) Projeção e restrição
- c) Junção e restrição.
- d) Projeção e junção
- e) União e restrição .

Atividade

Dentre as classes Java para acesso a banco de dados, com uso de JDBC, qual delas deve ser utilizada para a execução de comandos SQL parametrizados?

- a) Statement
- b)Connection
- c) DriverManager
- d) DriverManager
- e) PreparedStatement

Atividade

3 - Considerando uma tabela chamada **log** que guarda o id do usuário (inteiro), data e hora da ocorrência do erro (texto no formato “yyyy/mm/dd hh:mm:ss”) e mensagem do erro (texto), conforme o comando de criação e classe de entidade seguintes, implemente os métodos de uma classe **LogDAO** para efetuar consulta geral e inclusão de registro.

```
CREATE TABLE LOG (  
    ID_USUARIO int NOT NULL Primary Key,  
    DATA_HORA varchar(20),  
    MENSAGEM varchar(200)  
);  
  
public class Log {  
    public int idUsuario;  
    public String dataHora;  
    public String mensagem;  
    public Log() { }  
}
```

Notas

Título modal¹

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Título modal¹

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Referências

CASSATI, J. P.; FOSTER, George. Programação servidor em sistemas webRio de Janeiro: Estácio, 2016.

DEITEL, P.; DEITEL, H. Ajax, rich internet applications e desenvolvimento web para programadores. São Paulo: Pearson Education, 2009.

Java, como programar. 8. ed. São Paulo: Pearson, 2010.

MARINHO, A. L. Desenvolvimento de aplicações para internet. 1. ed. São Paulo: Pearson, 2016.

SANTOS, F. Tecnologias para internet II. 11. ed. Rio de Janeiro: Estácio, 2017.

Próxima aula

- O conceito de padrão arquitetural e principais arquiteturas;
- As características e benefícios da arquitetura MVC;
- A arquitetura MVC com Front Control em sistema Java Web.

Explore mais

- [Sintaxe SQL; <https://www.w3schools.com/sql/>](https://www.w3schools.com/sql/)
- [Tutorial de JDBC; <https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html>](https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html)
- [Tutorial de DAO com Generics; <https://www.baeldung.com/java-dao-pattern>](https://www.baeldung.com/java-dao-pattern)
- [Tutorial de JSF. <https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html target=>](https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html)