Aula 03 – 03/jun.

Introdução ao uso de banco de dados

O banco de dados é um repositório de dados, é onde os dados são armazenados ou persistidos em nosso computador. Para que um programa desenvolvido em uma determinada linguagem possa se comunicar com o SGBD – Sistema de Gerenciamento de Banco de Dados (MySQL, MS Server, Oracle,...) é preciso uma plataforma ou framework de conexão. No caso do Java temos o JDBC , assim como no C# , o ADO.NET e assim por diante. Utilizaremos o JDBC do java e o MySQL como banco de dados.

JDBC (Java Data Base Connectivity)

> *Java Database Connectivity* ou JDBC é um conjunto de classes e interfaces (API) escritas em Java que fazem o envio de instruções SQL para qualquer banco de dados relacional; Api de baixo nível e base para api's de alto nível; Amplia o que você pode fazer com Java; Possibilita o uso de bancos de dados já instalados; Para cada banco de dados há um driver JDBC que pode cair em quatro categorias.
>
> (Wikipedia)

Antes de qualquer coisa, o JDBC precisa conhecer o caminho até o banco de dados. Para isso ele precisa de duas coisas: String de conexão e Driver de conexão.

Classe base:

```
import com.mysql.jdbc.Connection;
import java.sql.*;

public class Conexao {

    public static Connection getConn(){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            return (Connection)
                    DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "usuario", "senha.");
        } catch (ClassNotFoundException ex) {
            System.out.println(ex);
            return null;
         } catch (SQLException s) {
            System.out.println(s);
            return null;
         }
```

Na linha:

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "usuario", "senha.");
```

Deve ser colocado o nome do banco de dados, o nome do usuário, que muitas vezes vem por padrão "root" e a senha do SGBD.

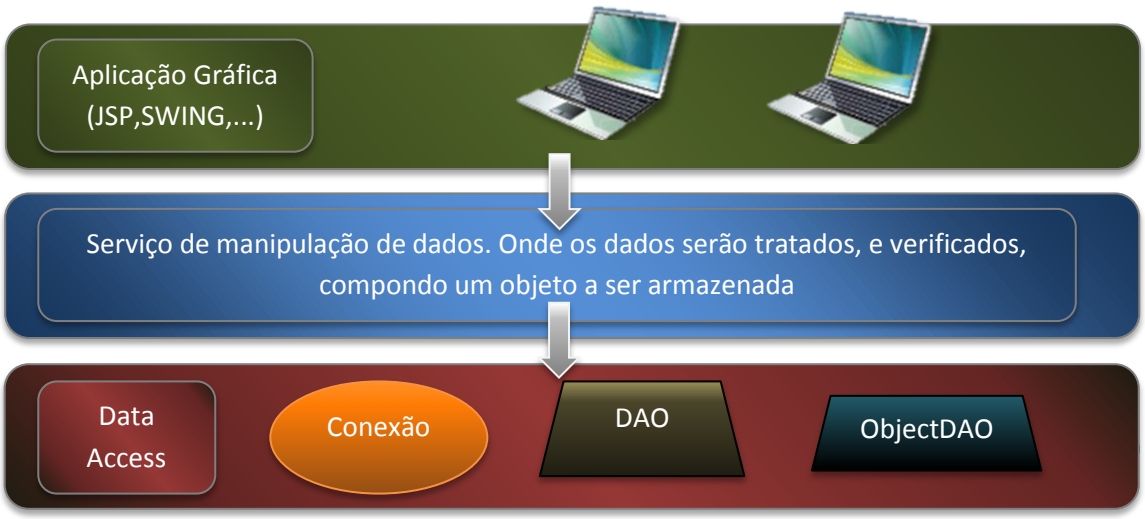
Explicando o código acima:

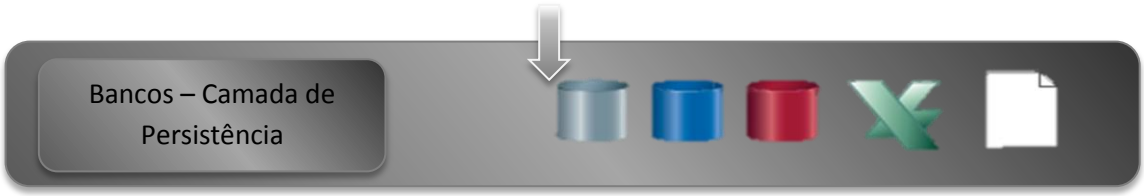
```java
//criando uma classe de conexao,
//chamada conexao para ser chamada sempre que  for preciso abrir uma conexao com o banco de dados.
public class Conexao {

    //criando um método que realize e retorne essa conexão
    public static Connection getConn(){
       try{
           //aqui vamos mostrar qual é o driver que iremos utilizar,
           //o drive mostra como o JDBC deve se comportar
           //e comunicar com o SGBD, isso é,
           //ele indica qual é o SGBD que iremos nos conectar
           //e como será realizada essa conexão.
           //aqui utilizaremos o JDBC para o MySQL, mas existem outros, um para cada SGBD.
            Class.forName("com.mysql.jdbc.Driver");
            //aqui nós vamos retorna a conexão pronta.
            return (Connection)
                    DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "usuario", "senha.");
            //exceções são criadas para verificar se a classe do driver foi encontrado ou não,
            //e se ocorreu um erro de conexão com o banco de dados.
       } catch (ClassNotFoundException ex) {
            System.out.println(ex);
            return null;
        } catch (SQLException s) {
            System.out.println(s);
            return null;
        }
    }

}
```

Uma vez conectado, estamos prontos para a manipulação do banco , isto é , inserir comandos SQL.

Padrão DAO

O JDBC oferece uma interface com vários comandos para serem utilizados para manipular SQL e o banco. Mas utiliza-los no meio da aplicação pode ser cansativo, e difícil que detectar onde estão os erros. Fora que caso precise mudar algo será extremamente difícil reparar. Para facilitar esse nosso trabalho, foi criado um padrão de projetos adotado por toda a comunidade Java: O DAO – Data Access Object, ou Objeto de Acesso a Dados. Ele visa criar uma "película" entre a aplicação Java e a conexão com o banco de dados. Entre outras palavras, a aplicação Java só precisa mandar os dados a serem manipulados e dizer o que ela quer com ele (inserir, atualizar, consultar, deletar, ou seja, as 4 operação básicas de um sistema CRUD – Create, Read, Update e Delete). Como mostra a figura a seguir.





Vamos agora criar uma classe chamada DAO, que programa esse padrão. Nela, será concentrado todo o código JDBC para manipulação de dados, assim como o acesso a Classe Conectar.

```
import java.sql.*;

public abstract class Dao {

    public Connection getConnection(){
        return Conexao.getConn();
    }

    public Statement getStatement() throws SQLException{
        return getConnection().createStatement();
    }

    public PreparedStatement getStatement(String statement) throws SQLException{
        return getConnection().prepareStatement(statement);
    }

    public ResultSet executeQuery(String query, Object ... params) throws SQLException{
        PreparedStatement ps = getStatement(query);
        for(int i = 0; i< params.length; i++)
            ps.setObject(i+1, params[i]);
        return ps.executeQuery();
    }

    public int executeCommand(String query, Object... params) throws SQLException{
        PreparedStatement ps = getStatement(query);
        for(int i = 0; i< params.length; i++)
            ps.setObject(i+1, params[i]);
        int result = ps.executeUpdate();
        ps.close();
        return result;
    }

}
```

Como pode ver, ela é uma classe, mais complexa, que executa qualquer código SQL que for passada a ela. Agora vamos ver mais a fundo.

```
import java.sql.*;

public abstract class Dao {

    //esse método cria uma conexão com com a classe Conexao,
    //fazendo com que ela se conecte ao banco. Abrindo assim, uma nova sessão.
    public Connection getConnection(){
        return Conexao.getConn();
    }
```

```
    //ambos os métodos a seguir criam Statements, ou instruçoes semi prontas,
    //para rodar SQL. Elas já estão semi-preparadas.
    //O método mais usado é o getStatement(Stringstatement),
    //onde se passa uma String contendo uma instrução SQL.
    public Statement getStatement() throws SQLException{
        return getConnection().createStatement();
    }

    public PreparedStatement getStatement(String statement) throws SQLException{
        return getConnection().prepareStatement(statement);
    }


    // esses dois métodos a seguir recebem a String SQL desejada
    // e podem receber vários ou nenhum parâmetro do tipo Object, ou seja,
    // elas podem receber parâmetros que identificam valores a serem colocados nas SQL,
    // como os dados de uma inserção, ou atualização, ou remoção e parâmetros de pesquisa.

    //  executeQuery é um método próprio para consultas, e retorna um resultSet,
    // ouseja, um conjunto de dados, ou mais abruptamente,a tabela do banco.
    // Usada com o comando "select", pode receber parâmetros para uma pesquisa filtrada,
    // ou não, é essa a função dos "..." depois de Object, eles indicam que se trata de
    // um parâmetro variável, ou seja, posso passar, um, dois, três, mil, ou nenhum.
    // Ele parâmetro é tratado como um vetor, um conjunto de dados.
    //  por isso usamos o for para passear pelo vetor, e setarmos as posições dos parâmetros.
    // Como veremos mais a frente, onde forem colocados parâmetros nas SQL,
    // podemos simplesmente colocar um "?", e depois passarmos o objeto cujo valor
    // representa o parâmetro. Por exemplo:
    // "select * from alunos where código=?"
    // a passamos o paramtero aluno.codigo, para recebermos uma tabela que contem o aluno
    // cujo código pe igual ao passado.
    // outro exemplo, usando o executeCommand, que faz a mesma coisa que o executeQuery,
    // mas não retorna nenhuma tabela de dados.
    // executeCommand("insert into aluno (nome,telefone) values (?,?)", aluno.nome,aluno.telefone) ;
    //  repare que a ordem dos parâmetros DEVE ser a MESMA      qua a ordem das suas "?" (posições)
    public ResultSet executeQuery(String query, Object ... params) throws SQLException{
        PreparedStatement ps = getStatement(query);
        for(int i = 0; i< params.length; i++)
            ps.setObject(i+1, params[i]);
        return ps.executeQuery();
    }

    public int executeCommand(String query, Object... params) throws SQLException{
        PreparedStatement ps = getStatement(query);
        for(int i = 0; i< params.length; i++)
            ps.setObject(i+1, params[i]);
        int result = ps.executeUpdate();
        ps.close();
        return result;
    }

}
```

Essa classe pode ser usada, para facilitar o trabalho de programar todas as conexões, abrir e fechar sessão, setar objetos nas SQL, prepara statements, entre outras.

Para utilizar essa classe, basta criar classes que estendem a ela, usando herança. Assim, criamos uma classe, que pode utilizar os métodos de Dao como seus, sem se preocupar com como estão programados. Exemplo:

A Classe Alunos com todos os seus atributos

```
/**
 *
 * @author Lucas Biason
 */
public class Produto {
    private String nome;
    private float valor;
    public Produto() {
    }
    public Produto(String nome, float valor) {
        this.nome = nome;
        this.valor = valor;
   }
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public float getValor() {
        return valor;
    }

    public void setValor(float valor) {
        this.valor = valor;
    }

    public String toString(){
        return nome + ", R$"+valor;
    }
}
```

Agora, para manipular a persistência.

```
public class ListaProdutos extends Dao{

    public ListaProdutos(){}

    public String addProduto(String nome, float valor){
        try {
            executeCommand("insert into produto (nome,valor) values (?,?)", nome, valor);
            return "sucesso";
        } catch (SQLException ex) {
            return "falhou";
        }
    }
    public String upProduto(String nome, float valor){
        try {
            executeCommand("update produto set valor=? where nome=?",valor,nome);
            return "sucesso";
        } catch (SQLException ex) {
            return "falhou";
        }
```

```
    }
    public String delProduto(String nome){
        try {
            executeCommand("delete from produto where nome=?",nome);
            return "sucesso";
        } catch (SQLException ex) {
            return "falhou";
        }
    }
    public Produto getProduto(String nome){
        try {
            ResultSet rs = executeQuery("select * from produto where nome=?",nome);
            rs.next();
            Produto p = new Produto();
            p.setNome(rs.getString("nome"));
            p.setValor(rs.getFloat("valor"));
            return p;
        } catch (SQLException ex) {
            return null;
        }
    }
    public List<Produto> getProdutos(String filtro){
        try {
            ResultSet rs = executeQuery("select * from produto where nome like ?","%"+filtro+"%");
            List<Produto> lista = new LinkedList<Produto>();
            Produto p;
            while(rs.next()){
                p = new Produto(rs.getString("nome"),rs.getFloat("valor"));
                lista.add(p);
            }
            return lista;
        } catch (SQLException ex) {
            return null;
        }
    }
}
```

Projetos feitos em aula:

- TesteBanco e TesteBanco com servlets (o exemplo dados será apenas para cadastrar um novo, os alunos deveram fazer para casa as outras operações (atualizar e remover)).

Exercícios Complementares de Fixação

1) Refazer os projetos feitos em aula.

2) Fornecedores (cadastrar fornecedores e alterar a classe produto para que guarde um fornecedor do produto).Um fornecedor tem: código, nome, cnpj, email, endereço. Altere a tabela produto para que guarde o código do fornecedor que o fornece. Para cadastrar esse código, no formulário de adição e alteração de produto, insira um combobox com os fornecedores cadastrados. Dica: o valor das opções da caixa serão os códigos dos fornecedores e a palavra que irá aparecer na seleção será o nome do fornecedor.

Dica: um combox é feito da seguinte maneira:

```
<Select size=”1” name=”fornecedor”>
    …opções
</select>
```

Para criar as opções faça um “for” e use o commando a seguir para criar cada uma das opções do combobox.

```
<option value="<coloque o código do fornecedor aqui>">Nome do fornecedor
</option>
```

3) (Fazer o desafio proposto) Crie um sistema para controle dos produtos/fornecedores. Listando todos eles e indicando em vermelho os produtos com estoque baixo (menor que 5).

Dica: Primeiramente crie o banco de dados “estoque”. Com as seguintes tabelas:

Fornecedor:

- Codigo: Integer
- Nome: String
- Telefone: String

Produto:

- Codigo: Integer
- Nome: String
- Quantidade: String
- Valor: String
- Fornecedor_id : Integer

Após isso, crie uma pagina onde será listado (em forma de tabela) todos os produtos com seus respectivos fornecedores.

Deverá ser Listado:

Nome do Produto | Quantidade | Valor | Nome do Fornecedor | Telefone do Fornecedor

4) Refaça o exercício 2 da primeira mini apostila, dessa vez, crie um banco de dados e faça um sistema JSP com Servlets para realizar as seguintes operações para cada uma das tabelas : Inserir novo, Atualizar, Remover, Criar uma página inicial com uma tabela contendo os dados do banco. Faça um Sistema para cada uma das tabelas segundo a lista a seguir:

Sistema de venda:

a. Produto
b. Fornecedor
c. Cliente

Sistema de Escola:

a. Escola
b. Curso
c. Aluno

5) Controle de Alunos: Crie um banco de dados “escola” com a tabela:

Aluno:

a. Codigo : String
b. Nome: String
c. Idade: String

Crie uma página para listar todos os alunos.

Crie um sistema para Inserir, Atualizar e Deletar.

Crie uma outra página que mostre apenas os alunos de uma determinada idade recebida pelo usuário na página anterior.

Crie uma outra página que separa os alunos por série e mostra várias tabelas de série com seus alunos, de acordo com a relação a seguir:

- serie 1 - 6 a 10 anos;
- serie 2 – 11 a 15 anos;
- serie 3 – 16 a 18 anos;
- serie 4 – a partir de 19 anos;

6) Crie um sistema para gerar as notas finais dos alunos. Dica:

Crie um banco de dados com as tabelas:

Alunos :

Codigo: Integer;
Nome: String;

Notas:

Cod_aluno: Integer;
P1, p2, p3: Float;

Existirá uma tela para a entrada das notas dos alunos e seus respectivos pesos.

Crie uma Página que mostre uma tabela com os alunos e suas notas e sua média final. Caso a média seja menor que 5, escreva a média e o nome do aluno , na tabela, em vermelho, caso contrário normal (em preto);

!!!Faça todos os exercícios uma vez com e outra vez sem servlets, o intuito é ver a diferença de se programar usando JSP sem e com servlets.!!!

Exercícios de Pesquisa:

1) Explique o que é: JDBC. Pesquise sua arquitetura e seu funcionamento.
2) Explique com suas palavras o funcionamento dos métodos da classe Dao.
3) Explique sobre o padrão Dao.