Análise e Desenvolvimento de Sistemas
Programação Java
Profª Ana Luiza Scharf
ana.scharf@ifsc.edu.br

Laboratório: Exemplo 05 - Conexão e Consulta MySQL

Introdução

Neste laboratório, vamos aprender como conectar um programa Java a um banco de dados MySQL usando uma **fábrica de conexões** (*ConnectionFactory*), além de executar consultas SQL para recuperar dados da tabela `Departamento`.

O padrão da fábrica de conexões ajuda a centralizar e organizar o acesso ao banco, facilitando a manutenção e a reutilização do código.

O banco de dados para este exemplo deve ser criado a partir do arquivo `resources/lab01mysqldmlddl.sql`, que contém as instruções para criar as tabelas e inserir dados necessários.

Faremos a conexão lendo as configurações (host, porta, usuário, senha, banco) de um arquivo de propriedades, `database.properties`, que deve estar na pasta `resources`.

Este exemplo é uma base importante para entender como sistemas Java podem interagir com bancos MySQL de forma segura e organizada. **Organização das pastas esperada:**

```
\begin{verbatim}
src/
 java/
    exemplo01/
       ExemploMuitoSimples.java
    exemplo02/
       PadroesDeProjeto.java
       db/
           ConnectionFactory.java
    exemplo03/
        UsandoPreparedStmt.java
    exemplo04/
       UsandoDAO.java
       entities/
           Pessoa.java
           PessoaDAO.java
    exemplo05/
       EemploMySQL.java
       db/
           ConnectionFactory.java
    bcd/
        Principal.java
    resources
        database.properties
        lab01-mysql-dml-ddl.sql
 lab01.sqlite
```

Classe `ConnectionFactory` - Criando a Conexão com MySQL

Código comentado

```
package exemplo05mysql.db;

import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

/**
 * Classe responsável por criar conexões com o banco MySQL.
 * As configurações ficam no arquivo database.properties dentro da pasta resources.
 */
public abstract class ConnectionFactory {

    // Nome do arquivo de configuração com as propriedades de conexão
    private static final String DB_PROPERTIES_FILE = "database.properties";

    private static Connection cnx; // Instância única de conexão (singleton simplificado)

    /**
     * Carrega o arquivo de propriedades do diretório resources.
     * Funciona dentro da IDE, testes de unidade e arquivos JAR.
     *
     * @return InputStream do arquivo properties
     * @throws IOException se o arquivo não for encontrado
     */
    private static InputStream getInputStream() throws IOException {
        InputStream is = ConnectionFactory.class.getClassLoader().getResourceAsStream(DB_PROPERTIES_FILE);

        if (is == null) {
            throw new IOException("arquivo não encontrado " + DB_PROPERTIES_FILE);
        } else {
            return is;
        }
    }

    /**
     * Cria e retorna uma conexão ativa com o banco MySQL.
     * Usa as configurações do arquivo database.properties para host, porta, banco, usuário e senha.
     *
     * @return Connection ativa com o banco de dados
     * @throws IOException se falhar ao carregar as configurações
     * @throws SQLException se falhar a conexão com o banco
     */
    public static synchronized Connection getDBConnection() throws IOException, SQLException {
        Properties properties = new Properties();

        try {
            // Carrega as propriedades do arquivo
            properties.load(getInputStream());

            // Obtém os dados necessários para a URL de conexão
            String host = properties.getProperty("host");
            String port = properties.getProperty("port");
            String dbname = properties.getProperty("database");

            // Monta a URL para conexão com MySQL
            String url = "jdbc:mysql://" + host + ":" + port + "/" + dbname;

            // Cria a conexão usando a URL e as propriedades (que incluem usuário e senha)
            cnx = DriverManager.getConnection(url, properties);

        } catch (SQLException ex) {
            throw new SQLException("erro com instrução SQL", ex);
        } catch (IOException ex) {
            throw new IOException("arquivo properties não encontrado", ex);
```

```
        }
        return cnx; // Retorna a conexão criada
    }
}
```

Explicação detalhada

- **Pacote e importações:** organiza a classe no pacote `exemplo05mysql.db` e importa as bibliotecas necessárias para conexão, manipulação de arquivos e exceções.
- **Constante DB_PROPERTIES_FILE:** define o nome do arquivo de configuração que contém host, usuário, senha, etc.
- **Método `getInputStream()`:** carrega o arquivo de propriedades do classpath, para funcionar em IDEs, testes e arquivos JAR.
- **Método `getDBConnection()`:**
  - Carrega as propriedades do arquivo.
  - Monta a URL de conexão para MySQL com host, porta e banco.
  - Cria a conexão com DriverManager usando as propriedades.
  - Usa `synchronized` para evitar problemas em múltiplas threads.
  - Lança exceções claras para problemas de SQL ou arquivo não encontrado.

Classe `ExemploMySQL` - Listando Dados do Banco

Código comentado

```
package exemplo05mysql;

import exemplo05mysql.db.ConnectionFactory;

import java.io.IOException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * Classe que demonstra como conectar ao banco MySQL e listar dados da tabela Departamento.
 * O script para criação do banco está em resources/lab01-mysql-dml-ddl.sql.
 */
public class ExemploMySQL {

    /**
     * Método que lista todos os departamentos, formatando a saída.
     * @return String formatada com dados da tabela Departamento.
     * @throws IOException caso haja erro na conexão.
     */
    public String listarDadosDeTodosDepartamentos() throws IOException {
        StringBuilder sb = new StringBuilder(); // Para construir a string com a saída formatada

        // Consulta SQL para buscar todos os dados da tabela Departamento
        String query = "SELECT * FROM Departamento";

        // Try-with-resources para garantir fechamento automático de recursos
        try (PreparedStatement stmt = ConnectionFactory.getDBConnection().prepareStatement(query);
             ResultSet rs = stmt.executeQuery()) {

            // Verifica se existe algum registro retornado
            if (rs.next()) {
                sb.append("-----------------------------------------------------\n");
                sb.append(String.format("|%-5s|%-35s|%10s|\n", "ID", "Nome", "Orçamento"));
                sb.append("-----------------------------------------------------\n");

                // Percorre todos os registros do ResultSet
                do {
                    int idDepto = rs.getInt("idDepartamento");   // Obtém idDepartamento
                    String dNome = rs.getString("dNome");        // Obtém nome do departamento
                    double orcamento = rs.getDouble("Orcamento"); // Obtém orçamento

                    // Adiciona linha formatada à saída
                    sb.append(String.format("|%-5d|%-35s|%10.2f|\n", idDepto, dNome, orcamento));
                } while (rs.next());

                sb.append("-----------------------------------------------------\n");
            } else {
                sb.append("Não há registros no banco de dados\n");
            }

        } catch (SQLException e) {
            e.printStackTrace();  // Exibe o erro caso ocorra na consulta
        }

        return sb.toString(); // Retorna a string com a tabela formatada
    }
}
```

Explicação detalhada

- **Pacote e importações:** está no pacote `exemplo05mysql` e importa a `ConnectionFactory` para obter conexões, além das classes JDBC para manipulação de SQL.
- **Classe:** contém método para listar todos os dados da tabela `Departamento`.

- **Método** `listarDadosDeTodosDepartamentos()`**:**
  - Cria um `StringBuilder` para montar a saída formatada.
  - Define a query SQL para buscar todos os registros da tabela.
  - Usa try-with-resources para abrir o `PreparedStatement` e executar a consulta, garantindo fechar recursos automaticamente.
  - Verifica se o `ResultSet` tem registros. Se sim, monta o cabeçalho da tabela.
  - Percorre todos os registros do `ResultSet`, obtendo os valores das colunas pelo nome.
  - Adiciona cada linha formatada na saída.
  - Caso não tenha registros, informa que o banco está vazio.
  - Captura exceções SQL e exibe a pilha de erros para ajudar no debug.

Arquivo de Script SQL para criação e carga do banco de dados MySQL

Não al

Conteúdo do arquivo `lab01-mysql-dml-ddl.sql`

```
-- DROP SCHEMA IF EXISTS lab01;
-- CREATE SCHEMA lab01;
-- USE lab02;

-- MySQL dump 10.13  Distrib 5.7.17, for Linux (x86_64)
--
-- Host: localhost    Database: lab01
-- ------------------------------------------------------
-- Server version 5.7.17

/*!40101 SET @OLD_CHARACTER_SET_CLIENT = @@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS = @@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION = @@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE = @@TIME_ZONE */;
/*!40103 SET TIME_ZONE = '+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS = @@UNIQUE_CHECKS, UNIQUE_CHECKS = 0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS = @@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS = 0 */;
/*!40101 SET @OLD_SQL_MODE = @@SQL_MODE, SQL_MODE = 'NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES = @@SQL_NOTES, SQL_NOTES = 0 */;

--
-- Table structure for table `Departamento`
--

DROP TABLE IF EXISTS `Departamento`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Departamento`
(
    `idDepartamento` int(11)        NOT NULL,
    `dNome`          varchar(255)   NOT NULL,
    `Orcamento`      decimal(10, 0) NOT NULL,
    PRIMARY KEY (`idDepartamento`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Departamento`
--

LOCK TABLES `Departamento` WRITE;
/*!40000 ALTER TABLE `Departamento`
    DISABLE KEYS */;
INSERT INTO `Departamento`
VALUES (1, 'Financeiro', 15000),
       (2, 'TI', 60000),
       (3, 'Gestão de Pessoas', 150000),
       (4, 'Pesquisa e Desenvolvimento', 7500),
       (5, 'Jurídico', 1000);
/*!40000 ALTER TABLE `Departamento`
    ENABLE KEYS */;
UNLOCK TABLES;

--
```

```sql
-- Table structure for table `Funcionario`
--

DROP TABLE IF EXISTS `Funcionario`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Funcionario`
(
    `idFuncionario`  INT         NOT NULL,
    `Nome`           VARCHAR(45) NOT NULL,
    `Sobrenome`      VARCHAR(45) NOT NULL,
    `idDepartamento` INT         NOT NULL,
    PRIMARY KEY (`idFuncionario`),
    CONSTRAINT `fk_Funcionario_Departamento` FOREIGN KEY (`idDepartamento`)
        REFERENCES `Departamento` (`idDepartamento`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Funcionario`
--

LOCK TABLES `Funcionario` WRITE;
/*!40000 ALTER TABLE `Funcionario`
    DISABLE KEYS */;
INSERT INTO `Funcionario`
VALUES (123, 'Julio', 'Silva', 1),
       (152, 'Arnaldo', 'Coelho', 1),
       (222, 'Carol', 'Ferreira', 2),
       (326, 'João', 'Silveira', 2),
       (331, 'George', 'de la Rocha', 3),
       (332, 'José', 'Oliveira', 1),
       (546, 'José', 'Pereira', 4),
       (631, 'David', 'Luz', 3),
       (654, 'Zacarias', 'Ferreira', 4),
       (745, 'Eric', 'Estrada', 4),
       (845, 'Elizabeth', 'Coelho', 1),
       (846, 'Joaquim', 'Goveia', 1);
/*!40000 ALTER TABLE `Funcionario`
    ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE = @OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE = @OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS = @OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS = @OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT = @OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS = @OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION = @OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES = @OLD_SQL_NOTES */;

-- Dump completed on 2017-03-15 16:26:09
```

Listagem 3: Script SQL para criação das tabelas e inserção dos dados

Explicações detalhadas

- **Comentários iniciais e configurações:** As primeiras linhas comentadas e comandos como `SET @OLD_CHARACTER_SET_CLIENT` são configurações feitas automaticamente pelo `mysqldump` para preservar o estado do banco e evitar problemas com codificação, verificações e modo SQL.

- **Criação do esquema e uso do banco:** Os comandos comentados para `DROP SCHEMA`, `CREATE SCHEMA` e `USE` indicam que o banco de dados `lab01` deve ser criado e selecionado antes da execução. Ajuste conforme seu ambiente.
- **Tabela `Departamento`:** Define os campos `idDepartamento` (chave primária), `dNome` (nome do departamento) e `Orcamento` (valor numérico para orçamento).
- **Uso de `LOCK TABLES ... WRITE`:** O comando bloqueia a tabela para escrita, garantindo exclusividade na inserção dos dados e evitando problemas de concorrência.
- **Inserção dos dados na tabela `Departamento`:** Insere cinco registros exemplares para os departamentos.
- **Tabela `Funcionario`:** Similarmente, a tabela contém dados dos funcionários, incluindo chave primária `idFuncionario` e chave estrangeira `idDepartamento` que referencia a tabela de departamentos, garantindo integridade referencial.
- **Bloqueios e inserções na tabela `Funcionario`:** Novamente o bloqueio da tabela é feito antes da inserção dos dados para garantir integridade e evitar conflitos.
- **Liberação dos bloqueios:** Os comandos `UNLOCK TABLES;` liberam o acesso para outras conexões após a inserção.

Classe `Principal` - Integração do Exemplo 05 (MySQL)

Trecho descomentado do menu principal para o Exemplo 05

```
// No vetor EXEMPLOS, descomentamos a opção 5 para o exemplo 05 (MySQL)
private final String[] EXEMPLOS = {
    "\n..:: Pequenos exemplos com Java, SQLite e MySQL ::..\n",
    "1 - Exemplo 01",
    "2 - Exemplo 02 - uso de padrões de projeto",
    "3 - Exemplo 03 - uso de PreparedStatement",
    "4 - Exemplo 04 - uso do Data Access Object (DAO)",
    "5 - Exemplo 05 - MySQL",
    "6 - Sair do programa"
};
```

Explicação: Aqui incluímos a opção 5 no menu principal, para permitir que o usuário acesse as funcionalidades do exemplo 05, que utiliza conexão com banco MySQL.

Método `exemplo05()` descomentado e explicado

```
/**
 * Executará métodos da classe no pacote exemplo05mysql
 *
 * @throws IOException
 */
private void exemplo05() throws IOException {
    ExemploMySQL exemploMySQL = new ExemploMySQL();

    // Chama o método que lista todos os departamentos do banco MySQL
    System.out.println(exemploMySQL.listarDadosDeTodosDepartamentos());
}
```

Explicação detalhada:

- Criamos uma instância da classe `ExemploMySQL`, que contém a lógica para conexão e consulta ao banco MySQL.
- O método `listarDadosDeTodosDepartamentos()` executa uma consulta SQL que retorna os dados da tabela `Departamento`.
- Os resultados são formatados e impressos no console.
- Essa função é chamada quando o usuário escolhe a opção 5 no menu principal.

Alteração no `main()` para incluir o exemplo 05

```
public static void main(String[] args) throws Exception {
    Principal p = new Principal();
    int opcao = -1;
    do {
        opcao = p.menu(p.EXEMPLOS);
        switch (opcao) {
            case 1:
                p.exemplo01();
                break;
            case 2:
                p.exemplo02();
                break;
            case 3:
                p.exemplo03();
                break;
            case 4:
                p.exemplo04();
                break;
            case 5:   // Chamada ao exemplo05 adicionada
                p.exemplo05();
```

```
                break;
            }
        } while (opcao != 6);
}
```

Explicação: Agora, ao escolher a opção 5 no menu principal, o programa chama o método `exemplo05()` que executa a consulta no banco MySQL e imprime os dados de todos os departamentos.

Conexão com MySQL

Arquivo `database.properties`

```
user=user
password=1234
useSSL=false
host=127.0.0.1
port=3306
database=lab01
```

Observação: no IFSC, devemos utilizar o servidor MySQL fornecido via Docker. No seu computador pessoal, você pode manter as credenciais user=user e password=1234. Porém, ao usar o ambiente do IFSC, é necessário alterar para os usuários configurados no Docker, ou seja: user=root e password=senhaRoot.

Explicação:

- `user` e `password`: credenciais para acessar o banco de dados MySQL.
- `useSSL=false`: desativa o uso de SSL.
- `host` e `port`: indicam onde o MySQL está rodando. Nesse caso, localmente.
- `database`: nome do banco de dados que será utilizado no exemplo.

Esse arquivo é carregado pela classe `ConnectionFactory.java` para configurar a conexão com o MySQL.
—

Criar o banco de dados `lab01` no MySQL

Siga os passos abaixo para criar e preparar o banco de dados `lab01` com o MySQL Workbench:

1. Abra o **MySQL Workbench**.
2. Clique no botão + em **MySQL Connections** para criar uma nova conexão.
3. Na janela **Set up a New Connection**, preencha os campos da seguinte forma:
    - **Connection Name:** `lab01`
    - **Hostname:** `127.0.0.1`
    - **Port:** `3306`
    - **Username:** `user`
    - Clique em `Store in Vault...` e informe a senha: **1234**

    Observação: as variáveis `user` e `password` devem ser as mesmas do arquivo `database.properties`.
4. Clique em `Test Connection`. Caso o MySQL Workbench solicite uma senha de administrador (usuário `root`), informe: `Aluno`.
5. Após o teste bem-sucedido, clique em `OK` para salvar a conexão.
6. Agora, clique sobre a conexão recém-criada chamada `lab01` para acessá-la.
7. No editor SQL que abrir, execute o seguinte script SQL para criar o banco, o usuário e as permissões:

Observação: Após esses passos, o banco de dados `lab01` estará criado, com permissões completas atribuídas ao usuário `user`, e pronto para receber os dados via o script `lab01-mysql-dml-ddl.sql`.

Conectar o IntelliJ ao MySQL e executar o script `lab01-mysql-dml-ddl.sql`

1. No IntelliJ, abra a aba **Database** (geralmente na lateral direita).
2. Clique no botão **+** e escolha `Data Source > MySQL`.
3. Preencha os dados da conexão:
   - **Host**: `127.0.0.1`
   - **Port**: `3306`
   - **User**: `user` (ou seu usuário do MySQL)
   - **Password**: `1234`
   - **Database**:

   Observação: as variáveis `user` e `password` devem ser as mesmas do arquivo `database.properties`.
4. Clique em **Test Connection** para verificar se está tudo ok.
5. Clique em **OK** para salvar a conexão.
6. Clique com o botão direito na conexão criada e selecione `Open Console` para abrir o console SQL.
7. Agora, abra o arquivo `lab01-mysql-dml-ddl.sql` que está em `src/main/resources`.
8. Clique no ícone de **seta** (Execute) para rodar o script e criar as tabelas e dados.
9. Aguarde a execução e verifique a mensagem de sucesso.

Atenção: um possível erro é criar o banco `lab01` e depois executar `USE lab01`. Se ocorrer erro, adicione a criação do banco no arquivo `lab01-mysql-dml-ddl.sql`.

Executar o exemplo Java `ExemploMySQL`

1. Execute a classe `Principal.java`.
2. Escolha a opção correspondente ao **Exemplo 05 - MySQL**.
3. O programa listará os dados da tabela `Departamento`.

Teste de Conexão com o Banco MySQL

Nesta seção apresentamos um teste simples de unidade para verificar se a conexão com o banco MySQL está funcionando corretamente. Esse teste é importante para garantir que as configurações de conexão estão corretas e que o banco está acessível.

Classe de Teste `TesteConexaoMySQL` com comentários

```
package bcd;

// Importa a fábrica de conexões para o MySQL
import exemplo05mysql.db.ConnectionFactory;

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;

// Importações do JUnit 5 para testes unitários
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertNotNull;

/**
 * Classe para executar teste de unidade sobre a conexão com MySQL
 */
public class TesteConexaoMySQL {
```

```
    @Test
    public void testarConexao() throws IOException, SQLException {
        // Tenta obter uma conexão com o banco de dados
        Connection conexao = ConnectionFactory.getDBConnection();

        // Verifica se a conexão não é nula (conexão estabelecida com sucesso)
        assertNotNull(conexao, "Não foi possível conectar no servidor MySQL");
    }
}
```

Explicação detalhada

- A classe `TesteConexaoMySQL` usa o framework JUnit 5 para realizar testes automatizados.
- O método `testarConexao()` tenta obter uma conexão com o banco MySQL usando o `ConnectionFactory`.
- O teste verifica se o objeto `Connection` não é nulo, ou seja, se a conexão foi estabelecida com sucesso.
- Caso a conexão falhe, a mensagem de erro "Não foi possível conectar no servidor MySQL"será exibida.
- Esse teste ajuda a garantir que o ambiente e as configurações estejam corretos antes de rodar os demais exemplos.

Conclusão do Laboratório e Diferenças em Relação ao Exemplo 04

Neste laboratório, exploramos o uso do banco de dados MySQL em uma aplicação Java, utilizando uma abordagem semelhante à do exemplo 04, que usava SQLite com padrão DAO. A principal novidade aqui é a conexão com um sistema gerenciador de banco de dados mais robusto e amplamente utilizado em ambientes profissionais: o MySQL.

Banco de dados diferente: Enquanto o exemplo 04 trabalhava com SQLite, um banco leve e embutido, este laboratório utiliza MySQL, que é um banco cliente-servidor, demandando uma conexão via rede, configuração de usuário, senha e banco de dados.

Este projeto foi integralmente elaborado pelo professor **Emerson Ribeiro de Mello**, docente do IFSC – Campus São José.