Relatório - Desenvolvimento da Aplicação para Utilização do Banco de Dados.

Durante o desenvolvimento do trabalho prático da disciplina, utilizamos o banco de dados relacional MySQL por ser uma boa opção tanto no quesito de instalação da base de dados, quanto também por ser uma tecnologia open-source.

Para o desenvolvimento da aplicação que utiliza a base de dados elaborada. Utilizamos a linguagem de programação JAVA e para o desenvolvimento a IDE IntelliJ.

Após a criação da base de dados no MySQL Workbench, executamos os scripts sql para criação das tabelas, views, relacionamentos, inserções e todos os demais processos necessários para popular a base de dados.

Em seguida tratamos da implementação da aplicação que irá consumir a base de dados gerada nesse trabalho conforme descrito abaixo:

1) ***Conexão à Base de Dados:***

Utilizamos os seguintes imports para construção da aplicação:

- java.sql.Connection
- java.sql.DriverManager
- java.sql.ResultSet
- java.sql.Statement

Além disso também utilizamos do pacote JAR MySQL connector como principal biblioteca de conexão.

O método de conexão utilizado é mostrado na imagem abaixo:

```
public void connect() {

    String servidor = "jdbc:mysql://localhost:3306/Skype" +
            "?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";

    String user = "root";
    String password = "LeIrBaG123";
    String driver = "com.mysql.cj.jdbc.Driver";

    try {
        Class.forName(driver);
        this.connection = DriverManager.getConnection(servidor, user, password);
        this.statement = this.connection.createStatement();
        System.out.println("CONNECTED");
    } catch (Exception e) {
        System.out.println("Erro: " + e.getMessage());
    }
}
```

O método connect() descrito acima é responsável pela conexão da aplicação com a base de dados, e seus componentes incluem:

A String servidor contém o endereço de conexão do banco de dados. Tudo que segue após o símbolo de interrogação(?) é referente ao tratamento do formato de hora da máquina local.

A String user contém o nome do usuário que a aplicação utiliza no banco de dados.

A String password contém a senha do usuário utilizado.

Ambos User e Password foram definidos durante a criação do banco de dados dentro do servidor criado na ferramenta MySQL Workbench.

A String driver contém o caminho até o driver JDBC, que foi utilizado no trabalho.

A aplicação conta com uma estrutura baseada em try-catch durante todo o seu desenvolvimento, visando identificar e tratar possíveis exceções que possam surgir quando tentamos nos conectar a base de dados. Dentro deste try-catch temos:

- Class.forName(driver)

O primeiro passo é a tentativa de inicialização dessa classe que representa nosso driver jdbc através da utilização do Class Loader. A classe em questão possui um bloco inicializador estático que irá registrá-la como um driver JDBC, avisando o java.sql.DriverManager, através da utilização do método registerDriver.

- this.connection = DriverManager.getConnection(servidor, user, password)

Neste ponto é realizada a conexão com o banco de dados. São enviados os parâmetros o servidor, usuário e senha necessários para a conexão.

- this.statement = this.connection.createStatement();

É criado um objeto Statement que irá executar as instruções SQL de acesso ao banco de dados.

2) Acessando a base de dados:

Como exemplo de acesso a base de dados, iremos relatar aqui o método de verificação de login de um usuário, como nossa aplicação consiste em uma representação do Skype que é uma aplicação de comunicação entre pessoas. Fizemos uma verificação inicial onde se deve entrar com os dados de algum usuário presente na tabela USUARIO da base de dados para começar a simular o uso do skype, dando a esse usuário as possibilidades de funções semelhantes a da aplicação original restrito apenas pelas limitações impostas pelo nosso escopo de representação do trabalho.

```
public boolean verifyLogin(String nome, String password) {

    try {
        String query = "select distinct nome, senha\r\n" +
                "from usuario;";

        this.resultset = this.statement.executeQuery(query);
        this.statement = this.connection.createStatement();
        while (this.resultset.next()){

            if(this.resultset.getString( columnLabel: "nome").equals(nome) && this.resultset.getString( columnLabel: "senha").equals(password)){
                return true;
            }else {
                return false;
            }
        }
    } catch (Exception e) {
        System.out.println("Erro: " + e.getMessage());
        return false;
    }
    return false;
}
```

Como mencionado anteriormente possuímos durante nossa implementação a utilização de try-catch visando identificar caso tenhamos alguma exceção ocorrendo durante nosso processamento.

Dentro desse try-catch possuímos:

- **String Query -** Que representa a query a ser executada no banco de dados.

- **This.resultset = this.statement.executeQuery(query) -**
  Comando utilizado para executar a query especificada anteriormente na base de dados e guardar todas as tuplas encontradas como resposta em resultSet.

- **while(this.resultset.next()) -**
  Essa parte nos garante que iremos processar o que estamos buscando para todos os resultados retornados pelo nosso banco de dados, para esse exemplo em questão buscamos os usuários registrados na tabela USUARIO da aplicação e vemos se o usuário que está utilizando a aplicação inseriu corretamente o username e password de algum dos usuarios registrados como utilizadores da aplicação, ou seja, se quem está

acessando a aplicação possui um registro correspondente na tabela de usuários da aplicação.

Outro exemplo interessante de utilização das consultas pela nossa aplicação é dado pelo método getIdByName(String nome). Esse método utiliza de um parâmetro passado pelo usuário para fazer uma consulta ao banco, ou seja, nesse caso não temos um select estático toda vez mas sim algo que pode ser alterado a cada iteração que o usuário tiver com a aplicação e querer saber o id de algum outro usuário da aplicação.

Esse método foi criado visando identificar o id de quem está logando na aplicação para facilitar consultas futuras à base de dados. Abaixo segue o método que procura o id do usuário tendo como entrada o seu nome:

```
public String getIdByName(String nome) {

    try{
        String query = "select usuario_id\r\n" +
                "from usuario\r\n" +
                "where nome = '" + nome + "';";
        this.resultset = this.statement.executeQuery(query);
        this.statement = this.connection.createStatement();
        while(this.resultset.next()){
            if(this.resultset.getString( columnLabel: "usuario_id").equals(nome)){
                return this.resultset.getString( columnLabel: "usuario_id");
            }
        }
    }catch (Exception e){
        System.out.println("Erro: " + e.getMessage());
    }

    return "Id não encontrado :( ";
}
```