
Newton
Quem se prepara, não para.

Programação Orienta a Objetos

3º período

Professora: Michelle Hanne

Sumário

- ✓ Instalando o Driver de conexão com banco de dados
- ✓ Conectando com o Banco de Dados
- ✓ Realizando consultas

Instalando o Driver de conexão com banco de dados

Newton
Quem se prepara, não para.

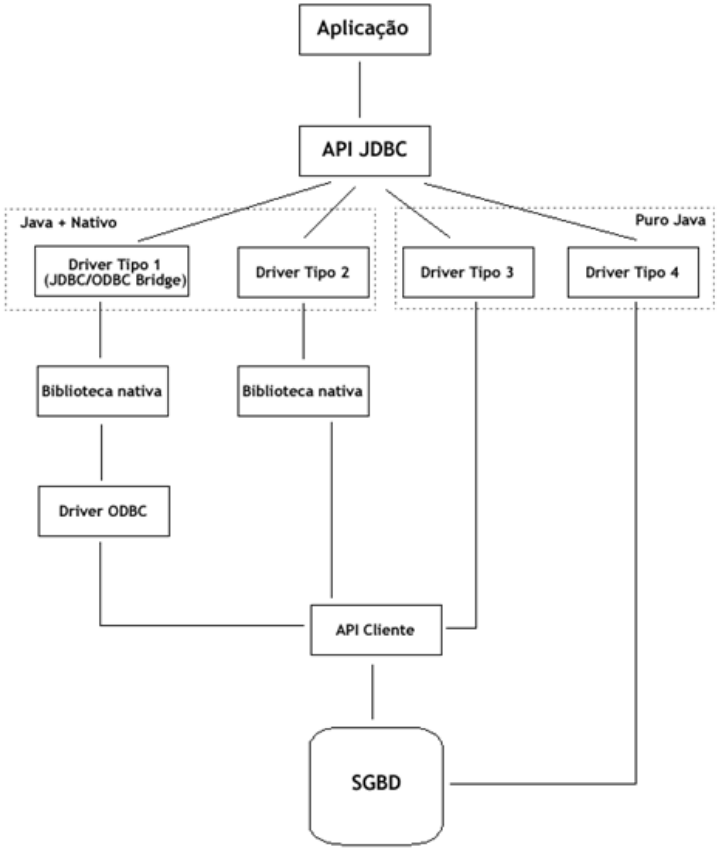
API JDBC (Java Database Connectivity) no JDK utilizada para conectar com tipos específicos de bases de dados relacionais, como MySQL e PostgreSQL.

JDBC é semelhante ao ODBC, e no principio usava justamente ODBC para conectar-se com o banco de dados. A partir de um código nativo as aplicações Java podiam utilizar qualquer banco de dados que tivesse um driver ODBC disponível. Isso contribuiu bastante para a popularização do JDBC uma vez que existe um driver ODBC para praticamente qualquer banco de dados de mercado.

Assim como ODBC, JDBC também funciona através de drivers que são responsáveis pela conexão com o banco e execução das instruções SQL.

Fonte: https://www.devmedia.com.br/jdbc-tutorial/6638

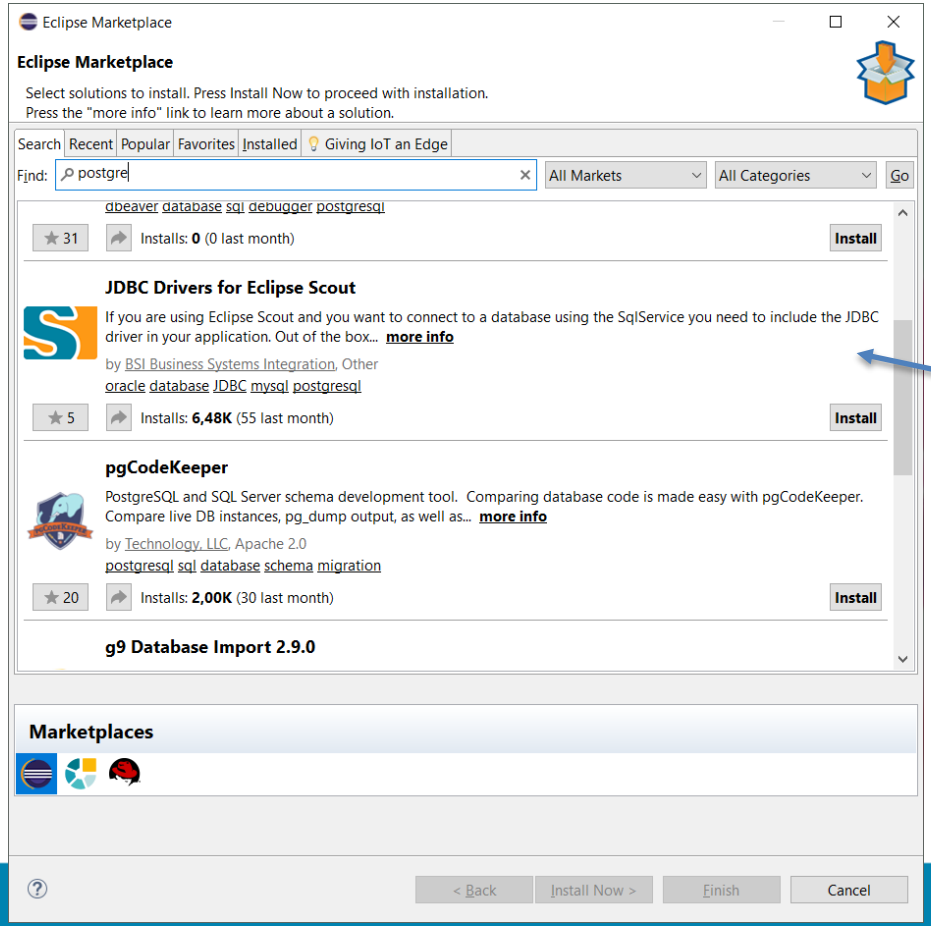
Instalando o Driver de conexão com banco de dados



Esses drivers são implementações das interfaces do pacote **java.sql.** Geralmente são disponibilizados na forma de um arquivo **JAR (Java ARchive)** pelo fabricante do banco de dados ou terceiros.

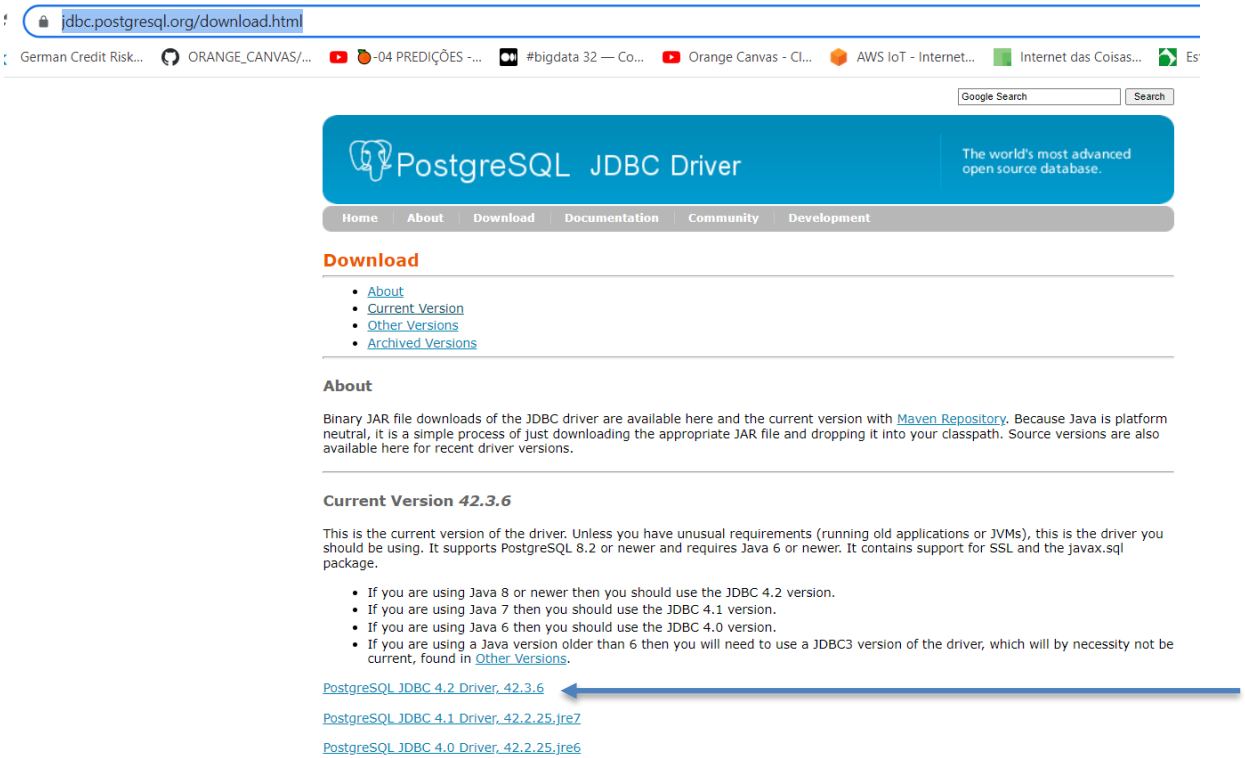
Fonte: https://www.devmedia.com.br/jdbc-tutorial/6638

Instalando o Driver de conexão com banco de dados



Driver genérico para conexão JDBC

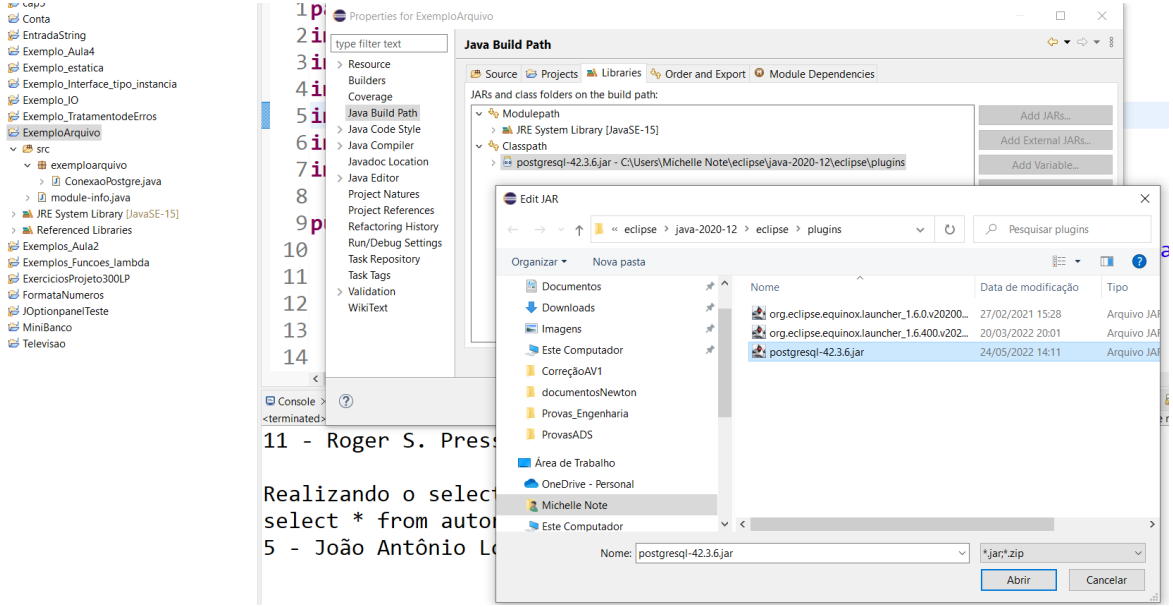
Instalando o Driver de conexão com banco de dados



Realizar o download do driver PostgreSQL

https://jdbc.postgresql.org/download.html

Instalando o Driver de conexão com banco de dados



Clique com o botão direito no projeto, vá em “Properties”, selecione “Java Build Path” na árvore esquerda, clique na aba “Libraries”, clique no botão “Add External JARs”, selecione o arquivo JAR do driver e clique em “OK”.

Conectando com o Banco de Dados

Newton
Quem se prepara, não para.

```
public class ConexaoPostgre {
 private final String url =
"jdbc:postgresql://localhost/BDlivrariaUniversitaria";
 private final String user = "postgres";
 private final String password = "b0aofmd0";
 Connection conn = null;

public Connection connect() {

        try {
            conn = DriverManager.getConnection(url, user, password);

            if (conn != null) {
                System.out.println("Connected to the PostgreSQL server
successfully.");
            } else {
                System.out.println("Failed to make connection!");
            }
            //versão do postgreeSQL
            Statement statement = conn.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT VERSION()");
            if (resultSet.next()) {
            System.out.println(resultSet.getString(1));
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return conn;
    }
```

Uma conexão é representada pela interface **java.sql.Connection.** É necessário um objeto de uma classe que implemente essa interface (essa classe é fornecida pelo driver do BD).

A classe **DriverManager** possui alguns métodos **getConnection(),** que são responsáveis por procurar dentre os drivers carregados um que seja compatível com a URL fornecida e solicitar a ele que abra uma conexão.

Comandos da interface Connection

Newton
Quem se prepara, não para.

A partir de agora temos aberta uma conexão com o banco de dados e você pode manipulá-la através do objeto Connection criado. **Abaixo estão alguns métodos da interface Connection que são mais utilizados:**

- **close():** fecha a conexão.
- **commit():** realiza um commit em todas as alterações desde o último commit/rollback. Caso a conexão esteja em modo auto-commit não é necessário chamá-lo explicitamente, pois será executado a cada alteração.
- **createStatement():** um dos métodos mais importantes da conexão, ele cria um objeto Statement que será usado para enviar expressões SQL para o banco. O retorno é um objeto da interface java.sql.Statement.

Fonte: https://www.devmedia.com.br/jdbc-tutorial/6638

Comandos da interface Connection

Newton
Quem se prepara, não para.

- **getMetaData():** busca os metadados do banco de dados. Metadados seriam basicamente a estrutura do banco, nomes de tabelas, campos, tipos, etc. Retorna um objeto da interface java.sql.DatabaseMetaData.
- **isClosed():** verifica se a conexão está fechada (retorna true se estiver fechada e false se estiver aberta).
- **isReadOnly():** verifica se a conexão é somente leitura (retorna true se for somente leitura e false se permitir alterações).
- **prepareCall(String sql):** cria um objeto para execução de stored procedures, o objeto retornado implementa java.sql.CallableStatement.
- **prepareStatement(String sql):** Cria um objeto semelhante ao criado por createStatement(), porém permite trabalhar com queries parametrizadas.
- **rollback():** desfaz as alterações feitas desde o último commit/rollback, é o inverso de commit. Caso a conexão esteja em modo auto-commit não é possível usá-lo, pois a conexão não deixa transações não confirmadas que possam ser desfeitas.
- **setAutoCommit(boolean autoCommit):** altera o modo auto-commit da conexão (true para ativar e false para desativar). Caso o auto-commit seja desativado, é necessária a chamada explícita ao método commit(), caso contrário as alterações não terão efeito.

Fonte: https://www.devmedia.com.br/jdbc-tutorial/6638

Executando consultas no Banco de Dados

Newton
Quem se prepara, não para.

A execução de consultas e atualizações no banco de dados gira em torno da interface java.sql.Statement (e sub-interfaces). Para criar um objeto desse tipo, utilize os métodos da conexão que está aberta. Por exemplo:

```
Statment stmt = conexao.createStatement(); //conexao é o nome da variável que criamos acima
```

Com uma instância de Statment já podemos executar uma query no banco, como abaixo:

```
ResultSet resultado = stmt.executeQuery("select * from clientes");
```

O método **executeQuery()** é usado **para executar consultas apenas**, e não deve ser usado para comandos como update, delete, create, etc. Para isso temos o método **executeUpdate()**. Já o método **execute()** é utilizado em situações em que a query pode retornar mais de um resultado.

Executando consultas no Banco de Dados

Newton
Quem se prepara, não para.

```
private static final String QUERY = "select * from autor where id_autor =?";
private static final String SELECT_ALL_QUERY = "select * from autor";

public void getAllUsers() {
        // Step 1: Establishing a Connection
        try {
            // Step 2:Create a statement using connection object
            PreparedStatement preparedStatement = conn.prepareStatement(SELECT_ALL_QUERY);
            System.out.println(preparedStatement);
            // Step 3: Execute the query or update query
            ResultSet rs = preparedStatement.executeQuery();

            // Step 4: Process the ResultSet object.
            while (rs.next()) {
                int id = rs.getInt("id_autor");
                String name_autor = rs.getString("nm_autor");
                System.out.println(id + " - " + name_autor);
            }
        } catch (SQLException e) {
            printSQLException(e);
        }
    }
```

Fonte: https://www.devmedia.com.br/jdbc-tutorial/6638

Executando consultas no Banco de Dados

Newton
Quem se prepara, não para.

```
public void getUserById() {
    // Step 1: Establishing a Connection
        try {
            // Step 2:Create a statement using connection object
            PreparedStatement preparedStatement = conn.prepareStatement(QUERY);
            // Step 3: Execute the query or update query
            preparedStatement.setInt(1, 5);
            System.out.println(preparedStatement);
            ResultSet rs = preparedStatement.executeQuery();
            // Step 4: Process the ResultSet object.
            // Step 4: Process the ResultSet object.
            while (rs.next()) {
                int id = rs.getInt("id_autor");
                String name_autor = rs.getString("nm_autor");
                System.out.println(id + " - " + name_autor);
            }
        } catch (SQLException e) {
            printSQLException(e);
        }
    }
```

Referências

https://www.javaguides.net/2020/02/how-to-connect-to-postgresql-with-java.html
https://www.javaguides.net/2020/02/java-jdbc-postgresql-select-example.html

https://www.devmedia.com.br/jdbc-tutorial/6638

https://www.javaguides.net/p/java-postgresql-tutorial.html