INSTITUTO FEDERAL
Rio Grande do Sul

JDBC – conexão com Banco de Dados

PROF. TIAGO MORAES

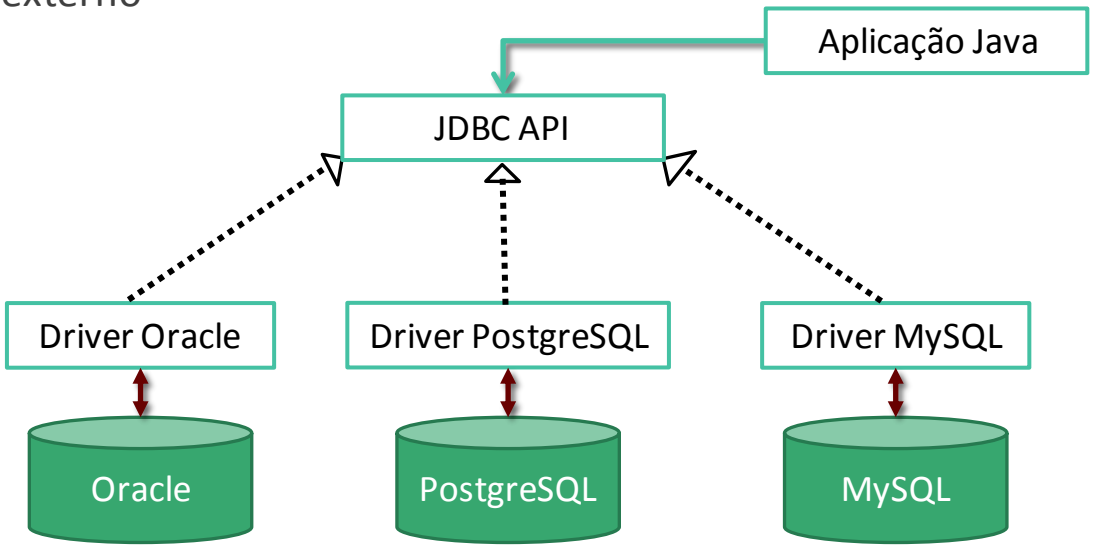
Roteiro

- Introdução ao JDBC
- Criando uma conexão
- Rodando uma *query*
- *Query parameters*
- Tratando a resposta de *queries*
- Toques finais

Introdução

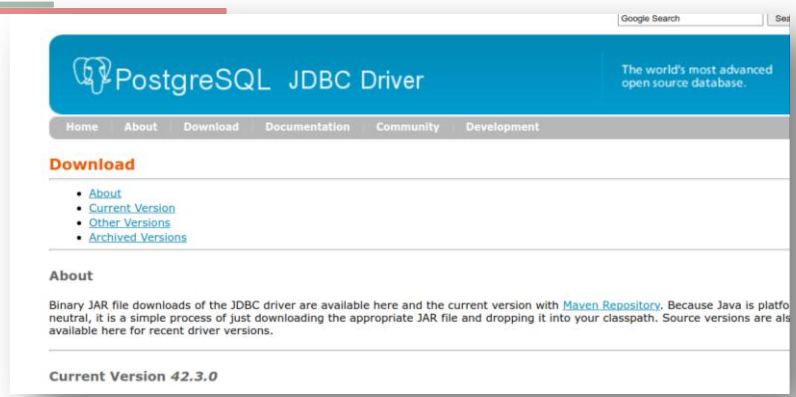
Arquitetura JDBC

- Conjunto de interfaces e classes para acesso a bancos de dados
  - Pacote java.sql
- Cada empresa oferece a implementação da API para o seu SGBD
  - Driver – JAR externo

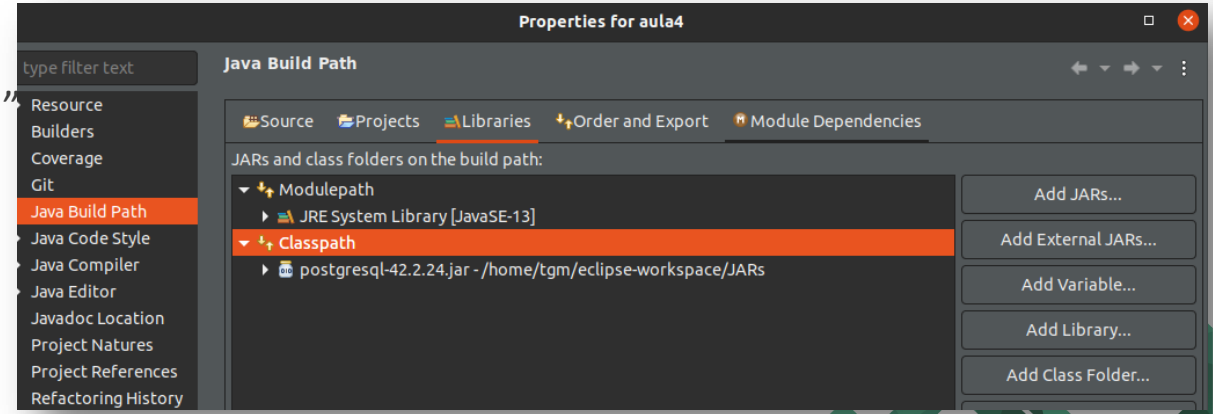


Introdução

- É necessário ter o driver
  - Jar externo do servidor
  - Para o PostgreSQL:
    - https://jdbc.postgresql.org/download.html



- É necessário adicionar o Jar no projeto
  - No Eclipse: Precisamos adicionar o jar ao classpah:
    - Click (botão direito) no projeto
    - "Build Path"->"Configure Build Path"
    - Em "libraries"
    - Classpath
    - Add External Jar



Criando uma conexão

- Método getConnection de DriverManager (java.sql.DriverManager)
  - Retorna uma conexão: objeto de Connection (java.sql.Connection)
  - Parâmetros (String):
    - **Login**: login do usuário do BD
    - **Senha**: do usuário do BD
    - URLcon: url de conexão com os valores de: **driver**, **host**, **porta** e **nome do BD**
      - "jdbc:<**driver**>://<**host**>:<**porta**>/<**nomeBD**>"
  - Exemplo:

```
String login = "postgres";
String senha = "postgres";
String urlcon = "jdbc:postgresql://localhost:5432/testebd";

Connection con = DriverManager.getConnection(urlcon, login, senha);
```

Rodando uma *query*

- Existem diferentes formas
  - Todas passam por rodar uma comando SQL passado como parâmetro (como String)
  - Por exemplo:
    - Criar um PreparedStatement (java.sql.PreparedStatement) a partir da conexão e da query
    - Rodar a query: execute(), executeQuery() ou executeUpdate()

```
Connection con = getConexao(); //constroi uma conexão

String query = "SELECT * FROM teste";
PreparedStatement pstm = con.prepareStatement(query);
pstm.execute();
```

Rodando uma *query*

- Diferenças entre os métodos para rodar uma query

método	retorno	Comentários
execute()	Boolean (se retornou dados)	Pode-se obter o ResultSet com o getResultSet()
executeQuery()	ResultSet com dados	Usar para SELECT
executeUpdate()	Quantidade de linhas afetadas	Usar para INSERT, DELETE, UPDATE

- Podemos (devemos) também parametrizar a consulta SQL
  - Definir *parameters* (caractere '?') para cada parte variável da query.
  - Exemplo:

"SELECT * FROM pessoa WHERE id = ?"

Query parameters

- Setando os *parameters*
  - Utilizar métodos set*Tipo* do preredStatment, por exemplo:
    - Para inteiro: setInt()
    - Para string: setString()
    - Para LocalDate: setObject()
    - Parâmetros (2): número do *parameter* (começando em um) e valor
  - Exemplo:

Boa Prática!
-evita SQL injection!
-código mais claro

```
Connection con = getConexao(); //constroi uma conexão
int id = 9;
String query = "SELECT * FROM pessoa WHERE id =?";
PreparedStatement pstm = con.prepareStatement(query);
pstm.setInt(1, id);
//consulta ficou: "SELECT * FROM pessoa WHERE id = 9"
pstm.execute();
```

Tratando a resposta de *queries*

- Quando a *query* retorna dados (normalmente SELECT) deve-se tratar a resposta:
  - A resposta será tratada com a classe ResultSet (java.sql.ResultSet)
  - De maneira geral a resposta estará apresentada como um conjunto de linhas
  - Assim, iteramos linha a linha: método next()
    - Passa o cursor para a próxima linha
    - retorna falso quando termina de iterar – após a última linha
    - Para se acessar os valores de cada coluna da linha se utiliza os métodos get*Tipo*()
      - getInteger() para coluna com valores inteiros
      - getString() para colunas com valores textuais
      - Parametro pode ser **int** (número da coluna – começa em 1) ou **string** (nome da coluna)

Tratando a resposta de *queries*

- Exemplo: considerando os dados da tabela Pessoa:

Pessoa

id	nome
2	João
5	Márcio
12	Ana

```
Connection con = getConexao(); //constroi uma conexão
String query = "SELECT id, nome FROM pessoa";
PreparedStatement pstm = con.prepareStatement(query);
ResultSet resp = pstm.executeQuery();
while(resp.next()){
    System.out.println(resp.getInt("id") +"-"+ resp.getString(2));
}
```

- Irá printar:

```
2 - João
5 - Márcio
12 - Ana
```

Toques finais

- Exceções:
  - Os métodos que acessam o BD para rodar consultas SQL ou estabelecer conexão, podem gerar uma SQLException
    - Exceção verificada
    - Deve-se utilizar o throws ou tratar a exceção
    - Um tratamento comum é transformar a exceção verificada em não verificada, pois:
      - Normalmente o programador que usa códigos que acessam banco de dados não consegue se recuperar desse tipo de erro
      - Deixa os códigos menos atrelados um ao outro, principalmente quando temos uma solução dividida em camadas

```
try{
    Connection con = DriverManager.getConnection(urlcon, login, senha);
}catch (SqlException e){
    throw new RuntimeException(e);
}
```

Toques finais

- Fechando os recursos abertos:
  - Durante o acesso ao banco de dados utilizando JDBC abrimos alguns recursos:
    - Conexão (Connection), PreparedStatment e ResultSet
  - Esses recursos devem ser fechados após sua utilização:
    - Todos tem o método close()
    - Caso tenhamos utilizado algum desses recursos em um try with resources podemos não necessitamos fechar:

```
Try(Connection con = DriverManager.getConnection(urlcon, login, senha)){
    PreparedStatement pstm = con.prepareStatement(query);
    ResultSet resp = pstm.executeQuery();
    ...
    pstm.close();
    resp.close();
}catch (SqlException e){
    throw new RuntimeException(e);
}
```