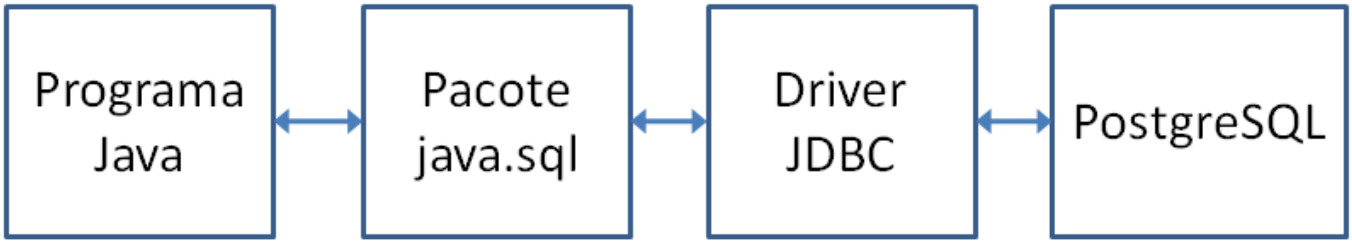
Bases de dados e Java

jorge.leao@ufrj.br

O objetivo destas notas é mostrar algumas maneiras básicas de usar bases de dados em Java.
Nos exemplos que serão mostrados a seguir, utiliza-se Java 11 e o SGBD PostgreSQL 11. O driver JDBC usado é o JDBC 4.2 (https://jdbc.postgresql.org/download.html).
Os exemplos de programas completos para os quais são fornecidos links foram desenvolvidos com o Apache Netbeans 11.1, mas podem ser adaptados para qualquer outro IDE.
A arquitetura básica de uma aplicação Java que acessa uma base de dados (em PostgreSQL ou qualquer outro SGBD convencional) é mostrada abaixo:



A seguir, são mostrados quatro exemplos de acesso a uma base de dados.

1. **Acesso com um driver simples feito por uma aplicação Java Desktop.**
   O exemplo está na página https://jorgeleao.github.io/eel418/index.2019.2.html ,
   no arquivo database20_2019-2.rar
   Desempacotando-se este arquivo .rar, e abrindo o projeto no Netbeans, pode-se ver 4 programas. Primeiro, execute o programa CriarArquivo.java para criar um arquivo de dados. Depois, execute o programa PopulaTabela.java que vai carregar os dados do arquivo criado, dando INSERTs na base de dados. Depois, execute o programa ConsultaDriverSimples.java que vai fazer um conjunto de acessos com SELECTs em posições aleatórias e medir o tempo total de acesso.
   Observe, tanto no PopulaTabela.java como no ConsultaDriverSimples.java, que o acesso à base de dados começa com o pedido de uma conexão com a base de dados através a instrução Java:

```
con = DriverManager.getConnection(
            "jdbc:postgresql://localhost:5434/fichas",
            "postgres",
            "abracadabra");
```

Esta é a forma mais simples de se conseguir uma conexão com a base de dados.
A partir da conexão obtida, todos os outros comando da biblioteca **`java.sql`** podem ser executados.

2. Acesso com um driver DataSource feito por uma aplicação Java Desktop.

O mesmo projeto, obtido do arquivo database20_2019-2.rar, possui ainda um outro programa, o ConsultaDriverPool.java. Este programa utiliza uma outra classe para obter as conexões com uma base de dados: org.postgresql.ds.PGConnectionPoolDataSource. Esta classe cria um conjunto de conexões que serão reutilizadas, aumentando o desempenho do sistema.
O exemplo do programa ConsultaDriverPool.java mostra como fazer uma conexão, mas não explora os recursos que permitem obter mais desempenho.
Ver mais sobre o assunto em:
https://jdbc.postgresql.org/documentation/publicapi/org/postgresql/ds/PGConnectionPoolDataSource.html
https://devcenter.heroku.com/articles/database-connection-pooling-with-java

3. Acesso com um driver DataSource feito por uma WebApp Java.

O servidor Apache Tomcat (assim como a maioria dos outros servlet containers) já possui todos os recursos para facilitar o uso de um "Pooling DataSource".
O projeto databaseWEB_2019_2 do arquivo databaseWEB_2019_2.rar mostra como se pode configurar uma WebApp Java para rodar no Tomcat e acessar uma base de dados com um "Pooling DataSource".
Embora a base de dados até pudesse ser acessada da mesma maneira que a mostrada no exemplo 1, a maneira mais aconselhada é usar um "Pooling DataSource" configurado no arquivo context.xml na pasta META-INF do projeto.
Neste exemplo, tem-se uma classe, BaseDAO.java, que serve de classe base (mãe) para a classe databaseDAO.java. A classe base possui o acesso à base de dados e só serve para obter uma conexão. A classe databaseDAO.java é a classe que possui as funcionalidades propriamente ditas de acesso à BD. Poderiam existir várias classes como a databaseDAO.java que agrupassem funcionalidades correlatas. Todas herdariam o método getConnection() da classe base.

4. Acesso à base de dados usando funções simples (métodos static) com injeção de dependência.

Outro método de obter uma conexão com um "Pooling DataSource" é usando injeção de dependência. No exemplo mostrado no projeto miniscada, do arquivo miniscada.rar, usa-se uma classe ConnectionFactory.java para obter uma conexão da base de dados. Na inicialização da aplicação (contexto) o método contextInitialized() da classe IniciadorDeContexto é executado, obtém uma nova instância da ConnectionFactory e a coloca (injeta) no campo static cf da classe Leituras01DAO. Depois disto, a classe Leituras01DAO, e outras semelhantes a ela, podem pedir conexões à sua ConnectionFactory.

Outro comentário genérico é sobre onde se deve colocar o driver JDBC. Ele pode ser colocado no próprio projeto e, neste caso, cada aplicação que se executa no servidor Tomcat pode ter um driver diferente, para conectar-se a SGBDs diferentes.
Mas o driver JDBC também pode ser colocado no diretório .../lib do próprio Tomcat. Como este diretório está automaticamente na variável classpath do Tomcat, não é necessário colocar qualquer referência no próprio projeto. Assim, todas as aplicações que não tiverem uma referência explícita a algum JDBC vão usar automaticamente o JDBC referenciado no diretório lib.