Java Database Connectivity (JDBC)

Instituto de computação

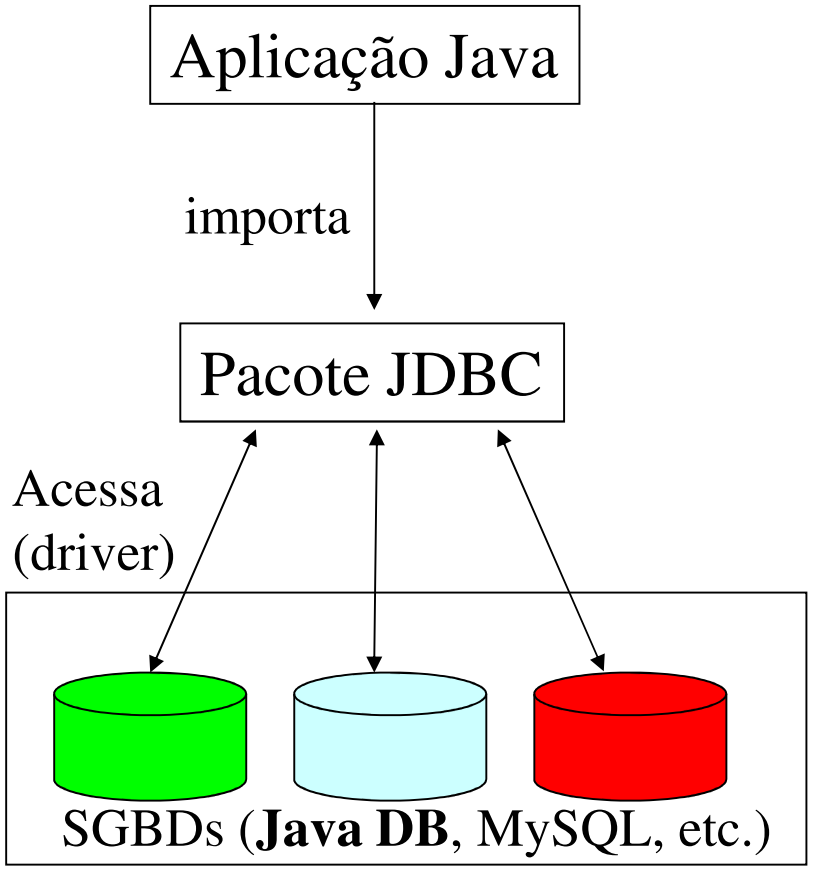
Prof. Lauro Eduardo Kozovits, D.Sc.

lauro@ic.uff.br

Utilizando material original do
Prof.Leonardo Gresta Paulino Murta

O Que é JDBC?

- API de acesso a banco de dados relacional do Java
- Faz uso do driver provido pelo banco de dados
- Roda SQL (create, insert, update, etc.) no banco



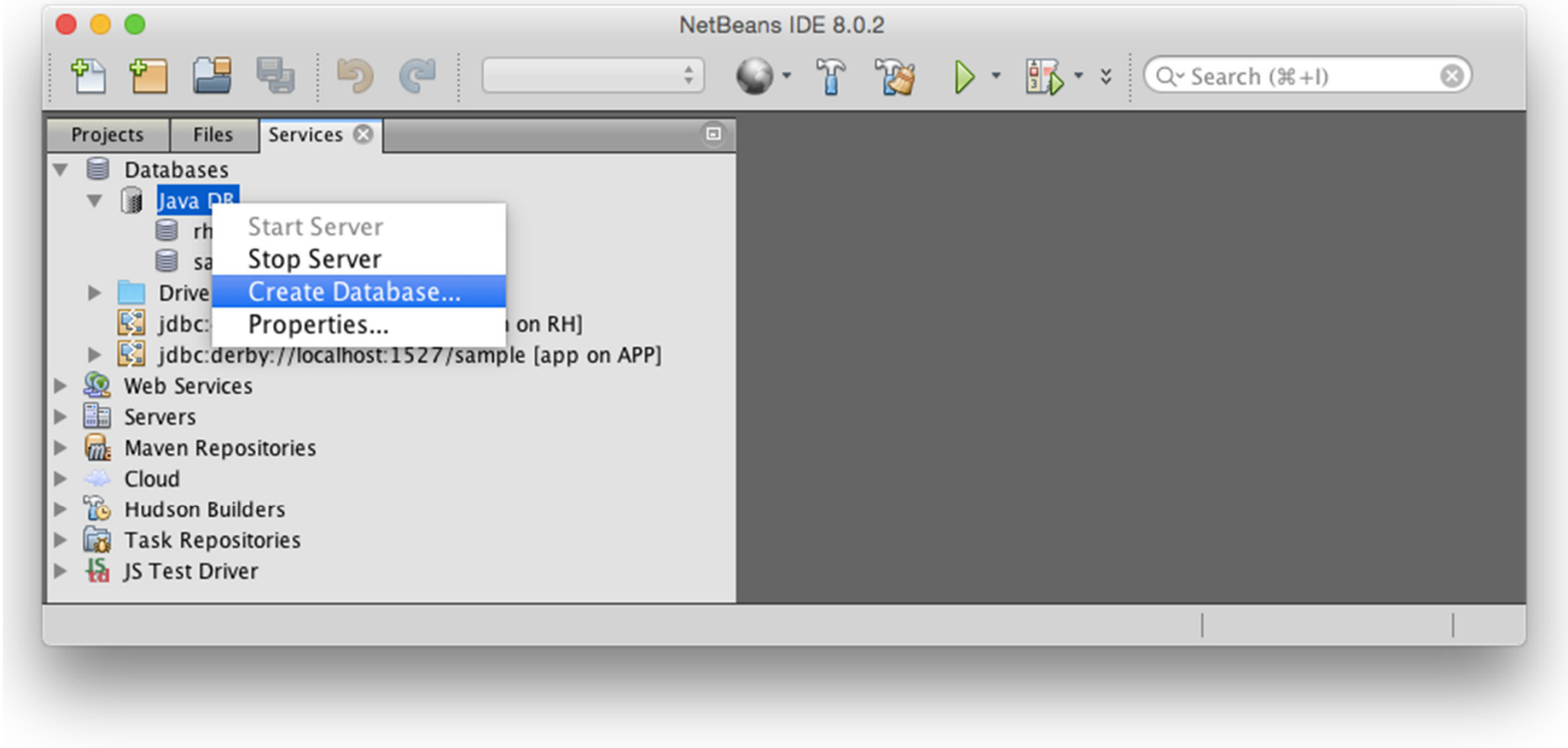
Como usar JDBC?

1. **Criar o banco de dados**
2. Incluir no projeto o driver de acesso ao banco de dados desejado
3. Conectar com o banco de dados
4. Preparar o SQL
5. Executar o SQL
6. Iterar sobre o resultado

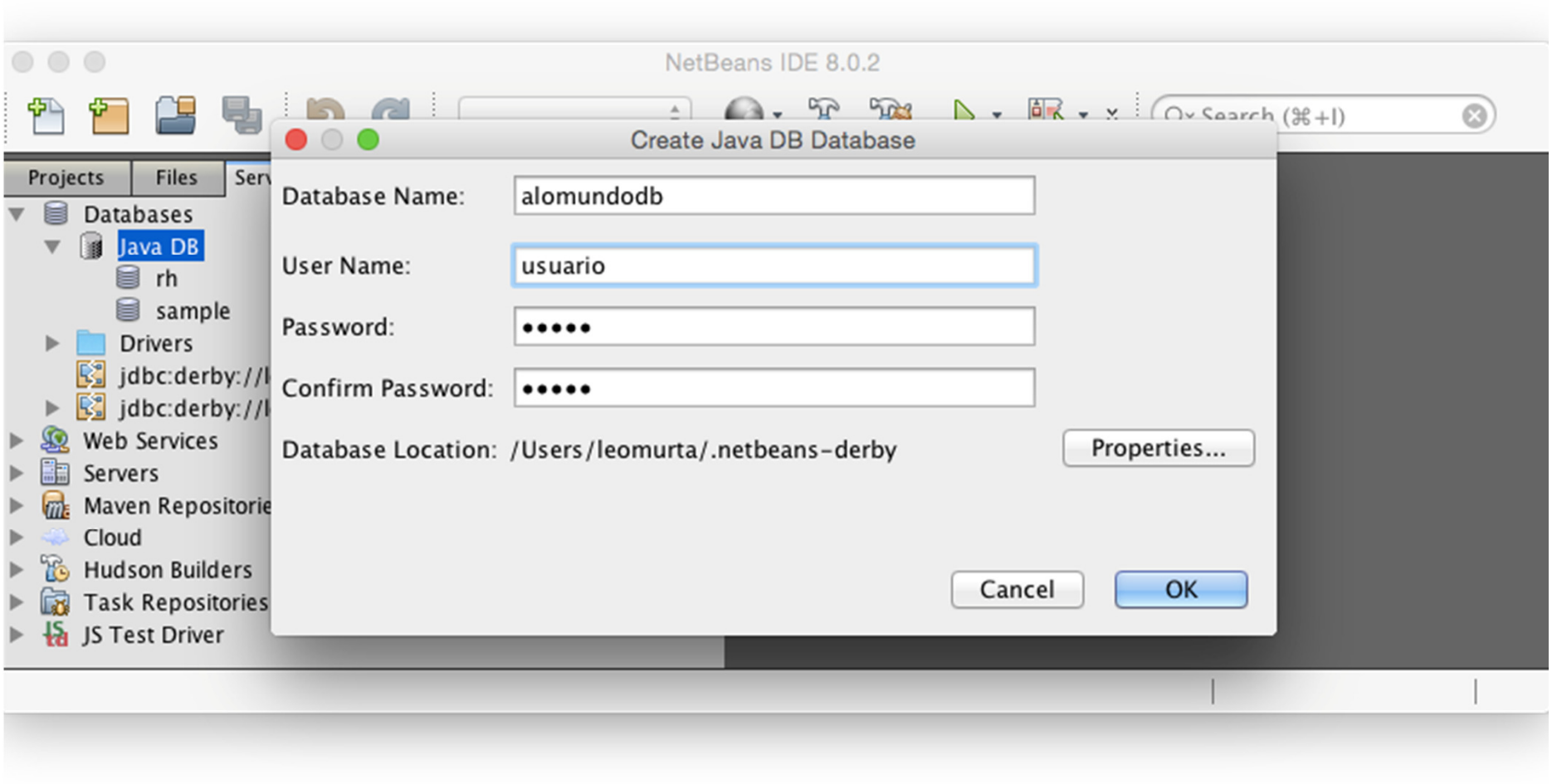
Criando o BD

- Exemplificaremos com o banco Java DB (Apache Derby que vem no Java), configurado pelo NetBeans
  - Contudo, é possível usar qualquer outro banco de dados que tenha driver JDBC
  - Também é possível usar outros clientes sem ser o NetBeans
- Dica importante
  - Manter o esquema (todos os create table) em um arquivo dentro do próprio projeto (esquema.sql)
  - Idealmente, o próprio programa poderia detectar se o BD existe e criar por demanda

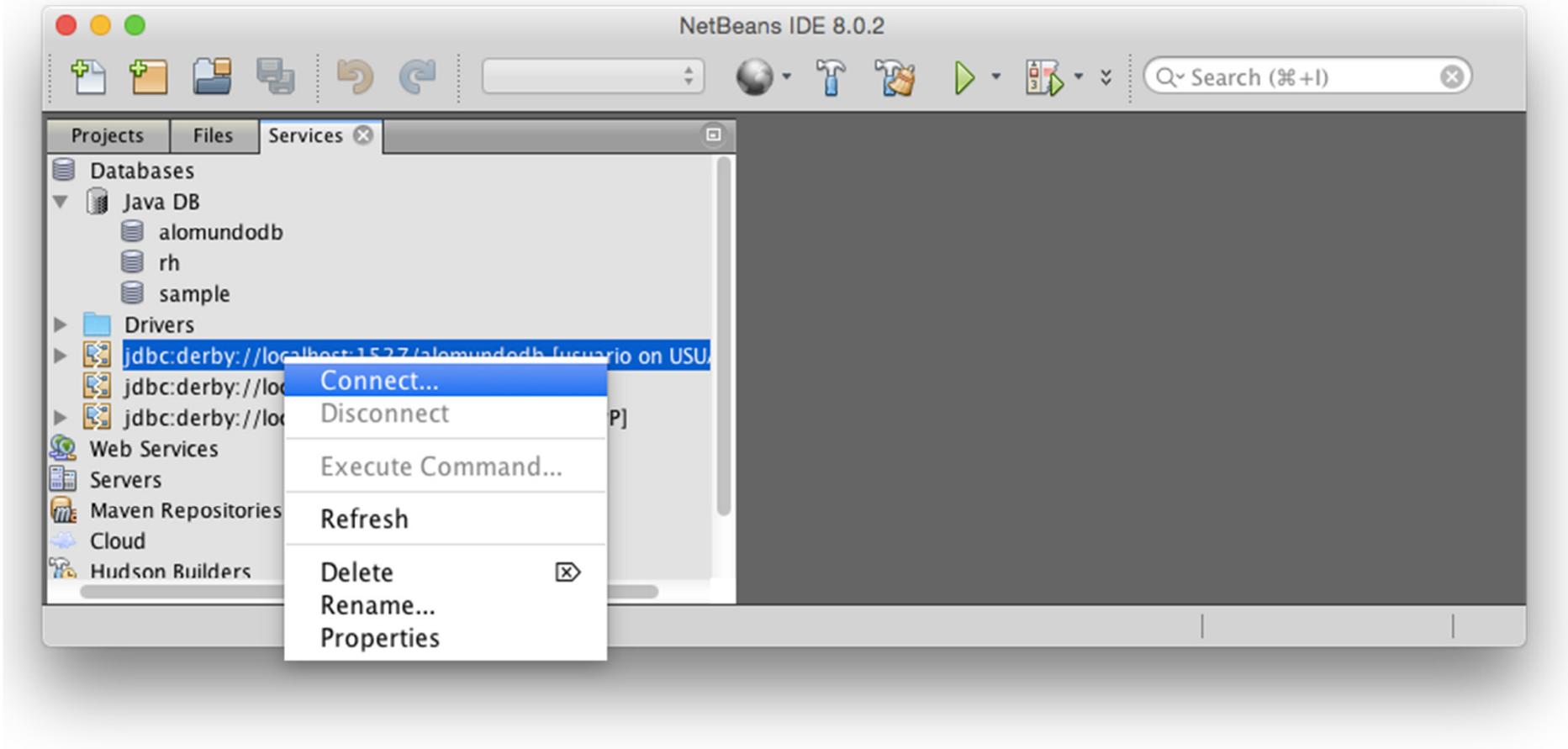
Criando o BD no NetBeans



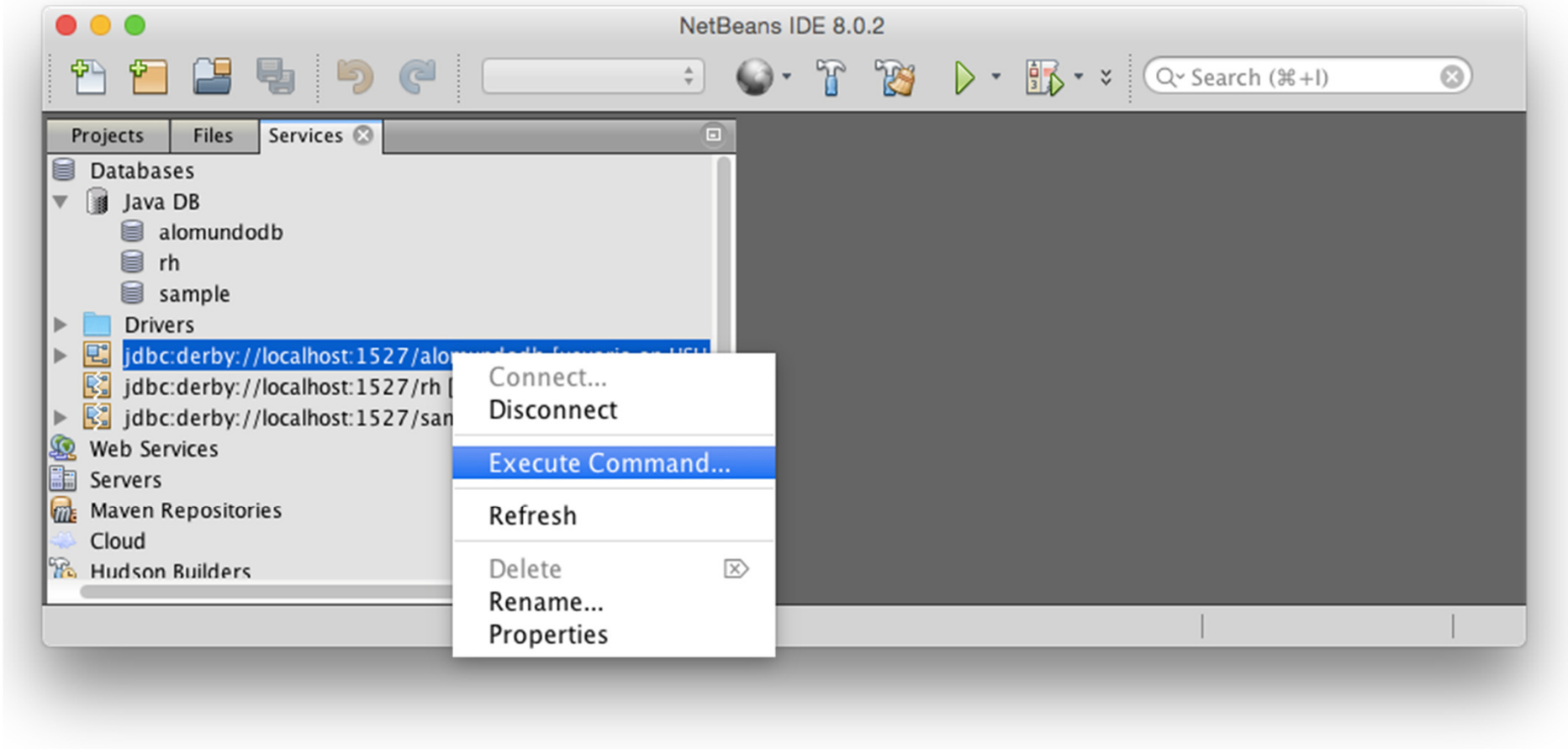
Criando o BD no NetBeans



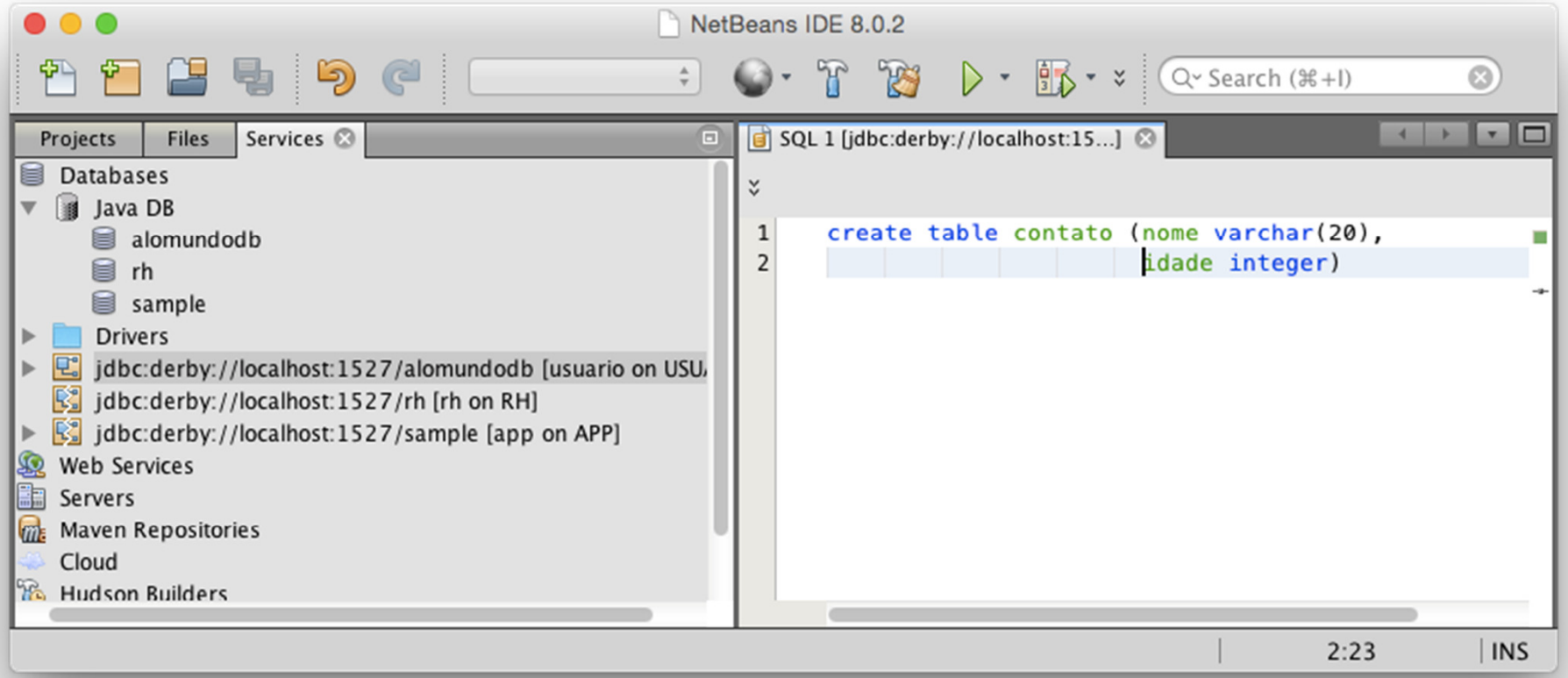
Criando o BD no NetBeans



Criando o BD no NetBeans



Criando o BD no NetBeans



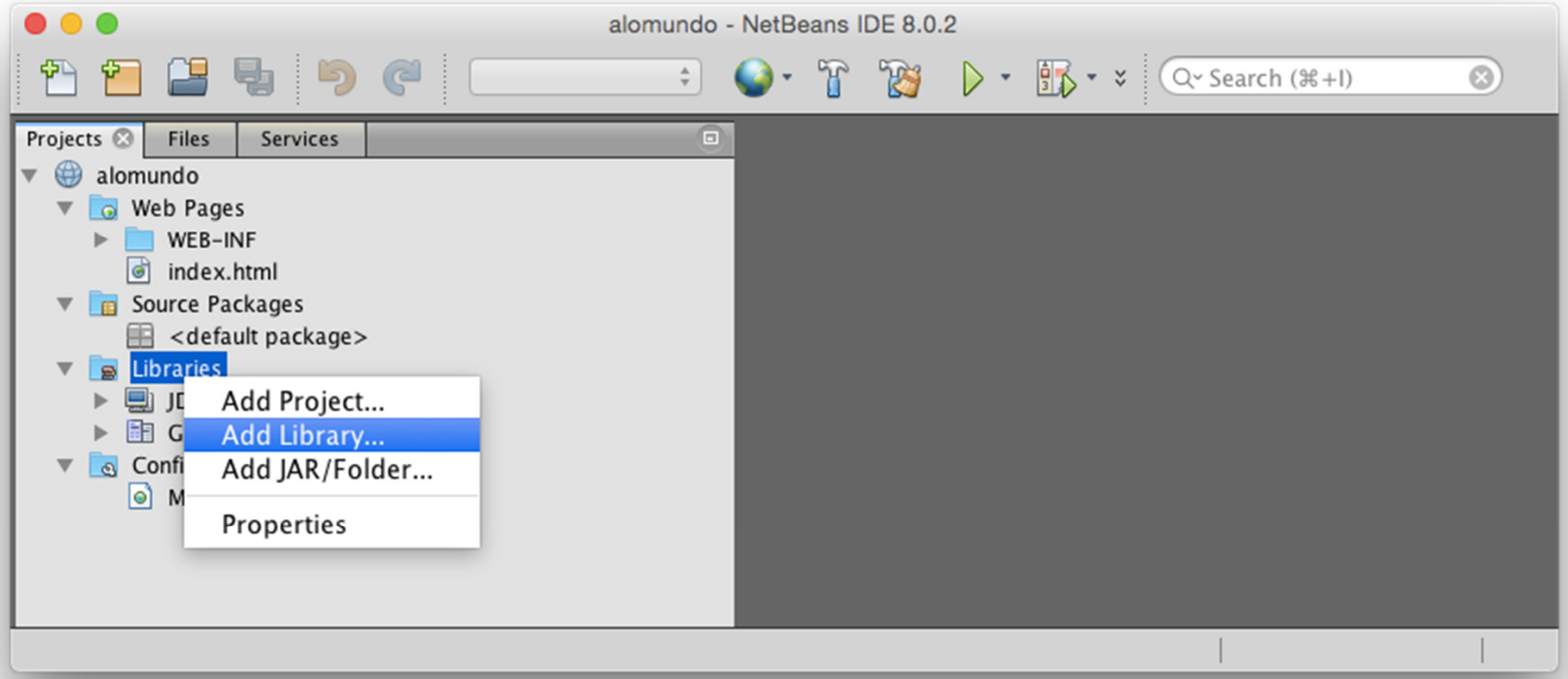
Como usar JDBC?

1. Criar o banco de dados
2. **Incluir no projeto o driver de acesso ao banco de dados desejado**
3. Conectar com o banco de dados
4. Preparar o SQL
5. Executar o SQL
6. Iterar sobre o resultado

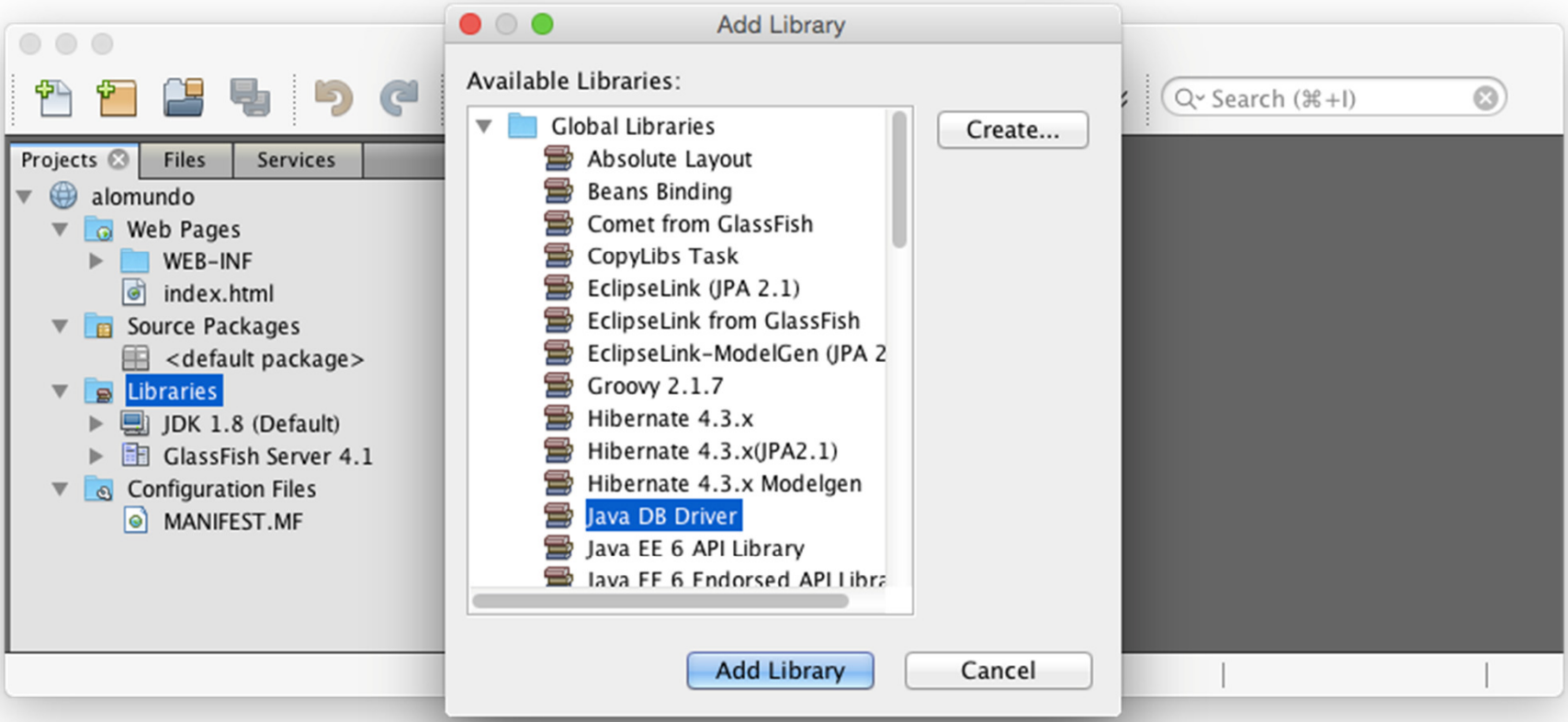
Incluindo driver do BD no projeto

- Primeiro, é necessário baixar o driver
  - MySQL: http://dev.mysql.com/downloads/connector/j/
  - PostgreSQL: https://jdbc.postgresql.org
  - Apache Derby (Java DB): http://db.apache.org/derby/derby_downloads.html
- Depois, basta copiar o arquivo .jar que contém o driver para a pasta WEB-INF/lib
- O NetBeans facilita nossa vida, pois já vem com alguns drivers
  - Basta adicionar no projeto

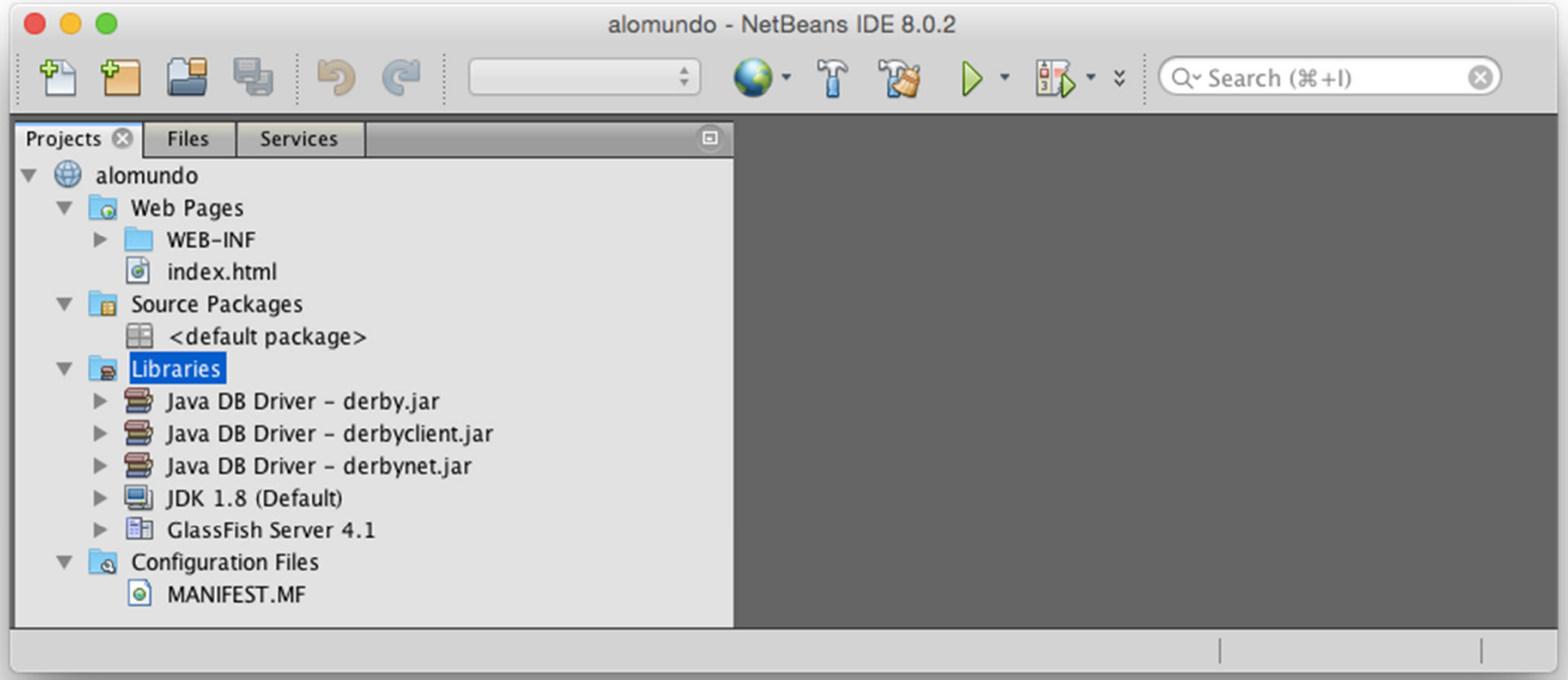
Incluindo driver do BD no projeto



Incluindo driver do BD no projeto



Incluindo driver do BD no projeto



Como usar JDBC?

1. Criar o banco de dados
2. Incluir no projeto o driver de acesso ao banco de dados desejado
3. **Conectar com o banco de dados**
4. Preparar o SQL
5. Executar o SQL
6. Iterar sobre o resultado

Conectando com o BD

- A partir deste momento já é possível criar uma conexão com o banco de dados

```
Connection conexao = DriverManager.getConnection(<URL>,
<USUARIO>, <SENHA>);
```

- Exemplo

```
Connection conexao =
      DriverManager.getConnection("jdbc:derby://localhost:1527
/alomundodb", "usuario", "12345");
```

Conectando com o BD

- A conexão com o BD pode ser estabelecida uma única vez, quando o Servlet é iniciado

```
Connection conn = null;

@Override
public void init() throws ServletException {
  try {
    conn =
DriverManager.getConnection("jdbc:derby://localhost:1527/alomun
dodb", "usuario", "12345");
  } catch (Exception ex) {
    ...
  }
}
```

Conectando com o BD

- Mas precisa ser fechada quando o servlet é destruído

```
@Override
public void destroy() {
  try {
    conn.close();
  } catch (Exception ex) {
    ...
  }
}
```

Como usar JDBC?

1. Criar o banco de dados
2. Incluir no projeto o driver de acesso ao banco de dados desejado
3. Conectar com o banco de dados
4. **Preparar o SQL**
5. Executar o SQL
6. Iterar sobre o resultado

Preparando o SQL

- Através da conexão é possível criar um objeto de execução de SQL

```
PreparedStatement sql = conexao.prepareStatement(<SQL>);
```

- Caso o SQL tenha parâmetros, usar “?” em cada parâmetro e atribuir os valores em seguida (importante para evitar injeção de SQL)

```
PreparedStatement sql = conn.prepareStatement("insert into
contato values (?, ?)")
sql.setString(1, nome);
sql.setInt(2, idade);
```

Preparando o SQL

- Também é necessário fechar o sql depois do seu uso
  - Isso pode ser feito de forma automática

```
try (PreparedStatement sql = conn.prepareStatement("...")) {
  // Execução da SQL
}
```

  - Ou pode ser feito manualmente, após a execução

```
// Execução da SQL
sql.close();
```

Como usar JDBC?

1. Criar o banco de dados
2. Incluir no projeto o driver de acesso ao banco de dados desejado
3. Conectar com o banco de dados
4. Preparar o SQL
5. **Executar o SQL**
6. Iterar sobre o resultado

Executando o SQL

- Para a executar atualizações na base (comandos create, drop, alter, insert, update, delete)

```
sql.executeUpdate();
```

- Para a executar consultas na base (comando select)

```
ResultSet resultado = sql.executeQuery();
```

Executando o SQL

- Exemplo de inserção de um contato (nome e idade) preenchido no fomulário HTML

```
String nome = request.getParameter("nome");
String idade = request.getParameter("idade");
if (nome != null && idade != null) {
  try (PreparedStatement sql = conexao.prepareStatement("insert
into contato values (?, ?)")) {
    sql.setString(1, nome);
    sql.setInt(2, Integer.parseInt(idade));
    sql.executeUpdate();
  } catch (Exception ex) {
    ...
  }
}
```

Como usar JDBC?

1. Criar o banco de dados
2. Incluir no projeto o driver de acesso ao banco de dados desejado
3. Conectar com o banco de dados
4. Preparar o SQL
5. Executar o SQL
6. **Iterar sobre o resultado**

Iterando sobre o resultado

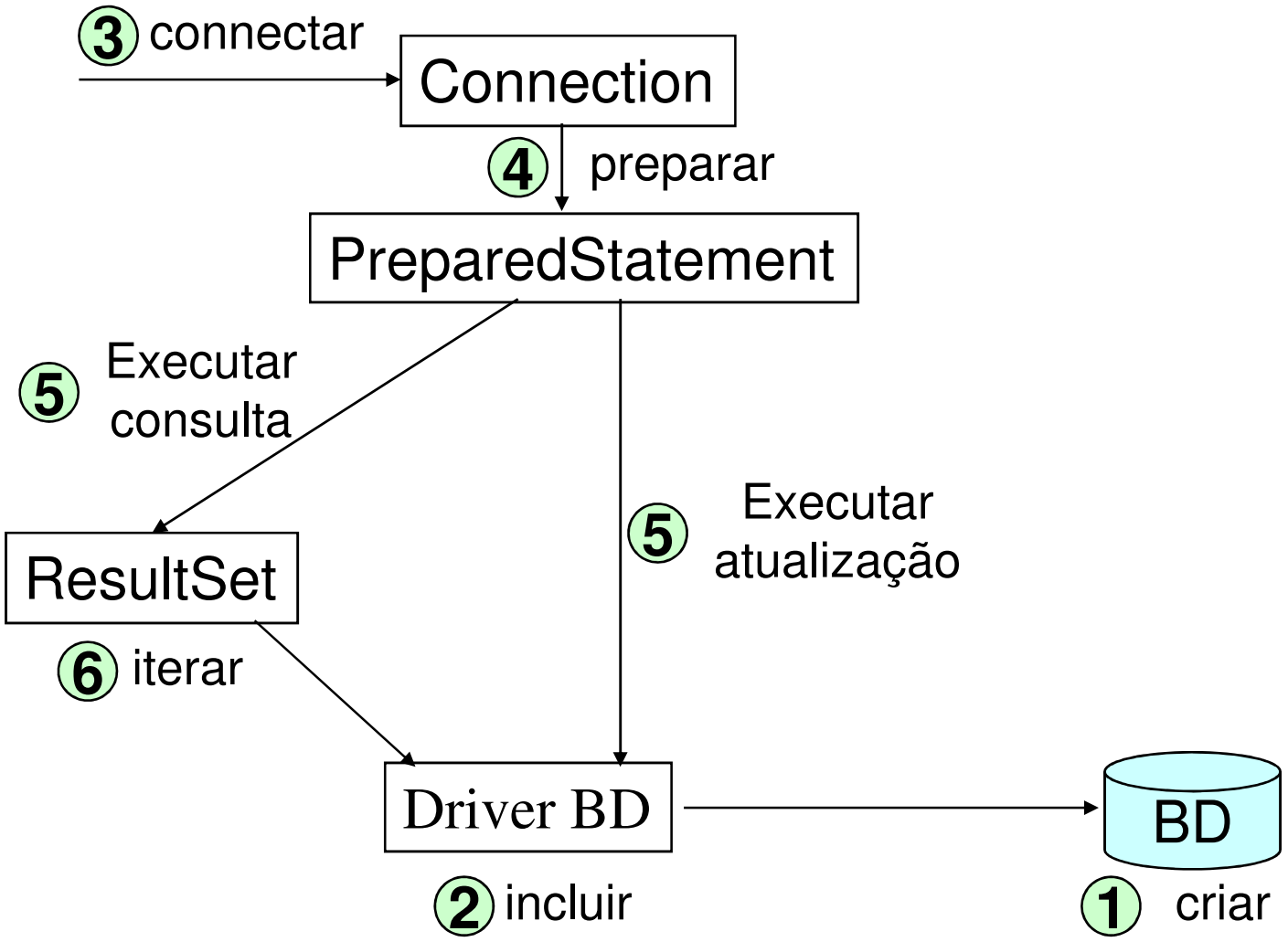
- O método *executeQuery()* retorna um objeto da classe *ResultSet*, que contém as tuplas que foram selecionadas na consulta
- O *ResultSet* permite acessar a próxima tupla
  - Método next()
  - Deve ser usado inclusive para pegar a primeira tupla
  - Retorna *false* se não houver próxima tupla
- O *ResultSet* também permite acessar os dados da tupla corrente
  - Método *getXXX(<NOME DO CAMPO>)*, onde *XXX* é o tipo do campo (ex.: *getInt("id")* ou *getString("nome")*)

Iterando sobre o resultado

- Exemplo de consulta a todos os contatos existentes no BD

```
try (PreparedStatement sql = conn.prepareStatement("select *
from contato")) {
  ResultSet resultado = sql.executeQuery();
  while (resultado.next()) {
    out.print("<p>" + resultado.getString("nome"));
    out.println(", " + resultado.getInt("idade") + ”</p>");
  }
} catch (SQLException ex) {
  ...
}
```

Tudo junto


3 connectar
Connection
4 preparar
PreparedStatement
5 Executar consulta
5 Executar atualização
ResultSet
6 iterar
Driver BD
BD
2 incluir
1 criar

Transações

- Executa um conjunto de comandos como se fossem atômicos
- Método *setAutoCommit(boolean)* de *Connection*
  - permite definir se cada comando SQL é "confirmado" (*Connection.commit()*) automaticamente, logo após ser executado ou não (default é *true*)
  - Se não for automático, o programador deve fazer chamar o método *commit()*
  - Caso seja necessário descartar a transação, basta chamar o método *rollback()*, também de *Connection*.

Transações

```
try {
  conexao.setAutoCommit(false);
  sql1.executeUpdate();
  sql2.executeUpdate();
  conexao.commit();
} catch (SQLException e) {
  conexao.rollback();
  ...
} finally {
  sql1.close();
  sql2.close();
  conexao.setAutoCommit(true);
}
```

Exercício

- Faça uma aplicação web de agenda eletrônica usando Servlet e JDBC
  - Permita o cadastro, listagem e remoção de contatos
  - Cada contato deve ter nome, endereço e telefone
  - A listagem dos contatos deve ser feita em uma tabela HTML
  - Ao lado de cada contato deve ter um link para a sua remoção

Java Database Connectivity (JDBC)

Instituto de computação

Prof. Lauro Eduardo Kozovits, D.Sc.

lauro@ic.uff.br

Utilizando material original do
Prof.Leonardo Gresta Paulino Murta