SOFTWARE ARCHITECTURES

Outline

Quick reminder

- Prerequisites for data persistence
- Connecting, starting and creating a database
- Creating JDBC Resource and Connection Pool
- Creating tables and inserting records to a database
- Creating entity classes from database

Requirements for this exercise session

Netbeans 11

JDK 8

Payara Server

Database server

Using a database on NetBeans

For this exercise sessions, we will use H2 as a database.

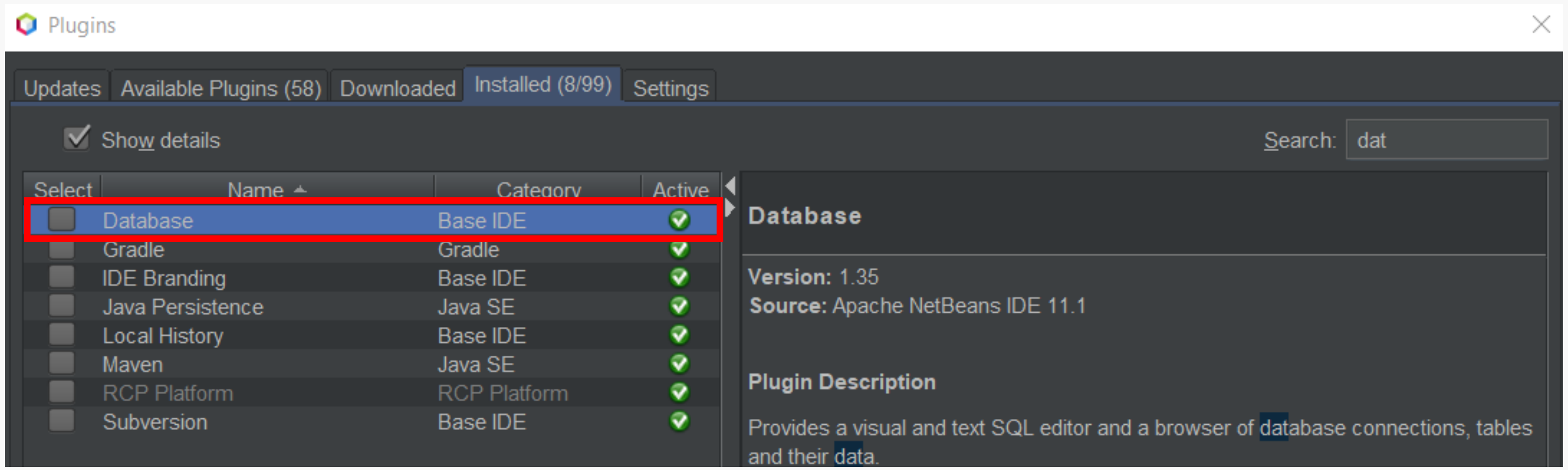
H2 is a relational database management system written in Java. It replace Derby as the default database in Payara 5.

H2 databases can be Embedded or can run in client-server mode.

Connecting to a database

Before we start, make sure you have «Database» plugin. If you don't have by default, please install it.

Tools > Plugins > Installed > Search «Database»



Plugins

Updates | Available Plugins (58) | Downloaded | Installed (8/99) | Settings

☑ Show details

Search: dat

Select	Name	Category	Active
☐	Database	Base IDE	✓
☐	Gradle	Gradle	✓
☐	IDE Branding	Base IDE	✓
☐	Java Persistence	Java SE	✓
☐	Local History	Base IDE	✓
☐	Maven	Java SE	✓
☐	RCP Platform	RCP Platform	✓
☐	Subversion	Base IDE	✓

Database

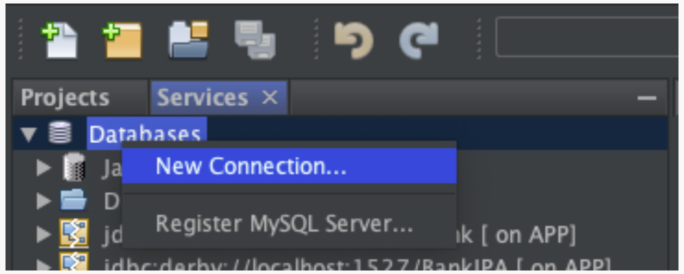
Version: 1.35
Source: Apache NetBeans IDE 11.1

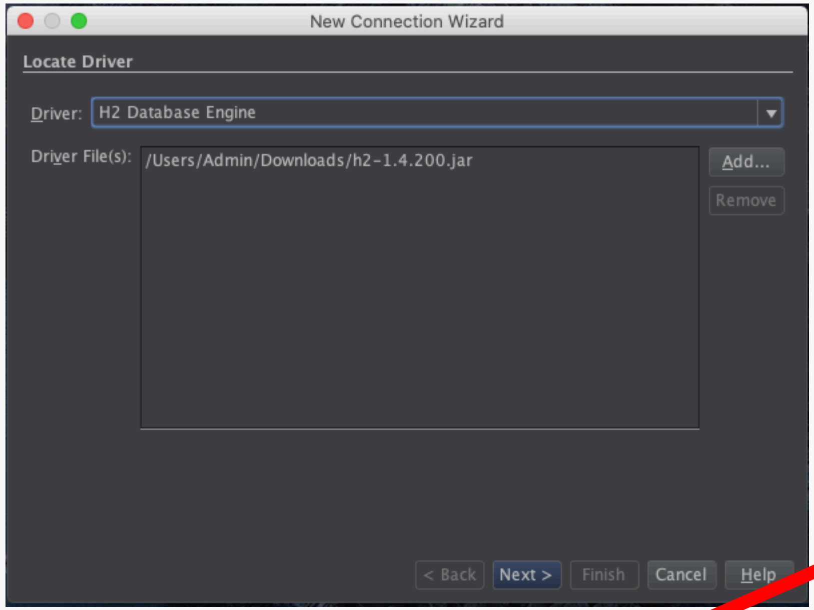
Plugin Description

Provides a visual and text SQL editor and a browser of database connections, tables and their data.

Connecting to a database

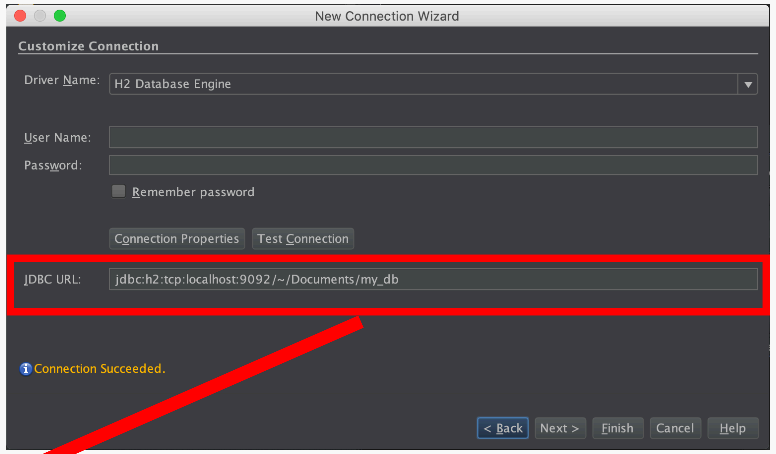
For our project, we will use an H2 embedded database. Before using it, we need to set it up in the Services section:





Driver: H2 Database Engine

Driver File(s): /Users/Admin/Downloads/h2-1.4.200.jar



Driver Name: H2 Database Engine

User Name:

Password:

JDBC URL: jdbc:h2:tcp:localhost:9092/~/Documents/my_db

Connection Succeeded.

Access the generated database that you downloaded.
E.g: */Users/Admin/Downloads/soar-tp-master/week7/my_db*

Recurrent issues

H2 Driver file missing: To solve this issue, you need to download the .jar file on the following link: http://repo2.maven.org/maven2/com/h2database/h2/1.4.200/h2-1.4.200.jar

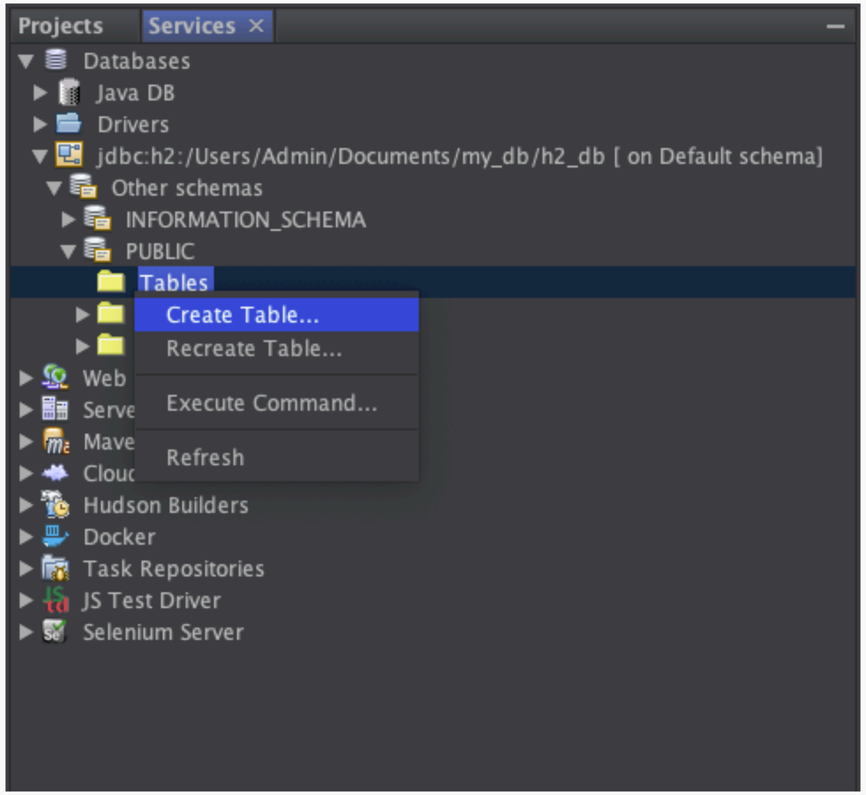
Cannot connect to the H2 database: Make sure that you have all the rights on the folder identified in the JBDC URL

Unable to access the database: To solve this issue, you need to start the H2 database manually:

```
java -cp <h2.jar location> org.h2.tools.Server -t -tcpAllowOthers
```

Creating a table in the database

Expand the connection and right-click on Tables, then click on Create Table



Create a Table called "EMPLOYEE" with the following columns:

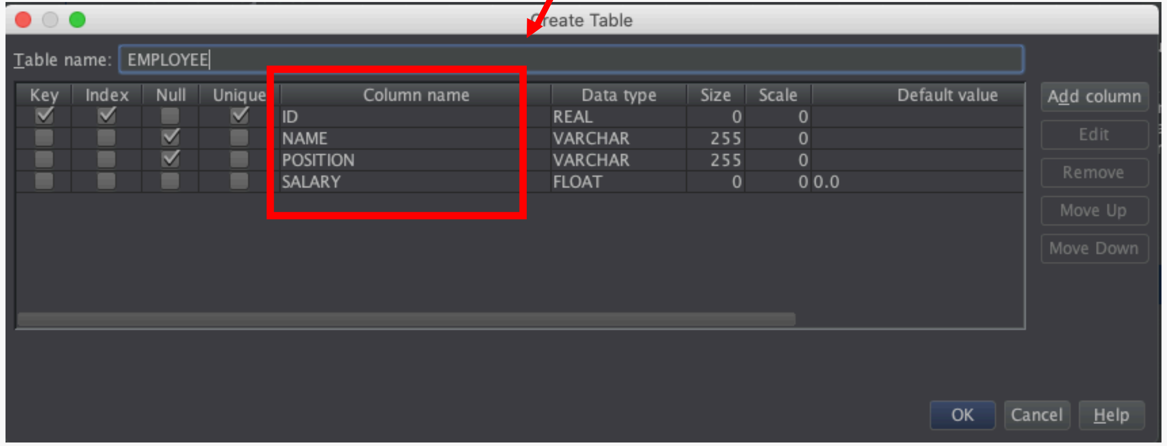


Table name: EMPLOYEE

Key	Index	Null	Unique	Column name	Data type	Size	Scale	Default value
✓	✓		✓	ID	REAL	0	0	
		✓		NAME	VARCHAR	255	0	
		✓		POSITION	VARCHAR	255	0	
				SALARY	FLOAT	0	0	0.0

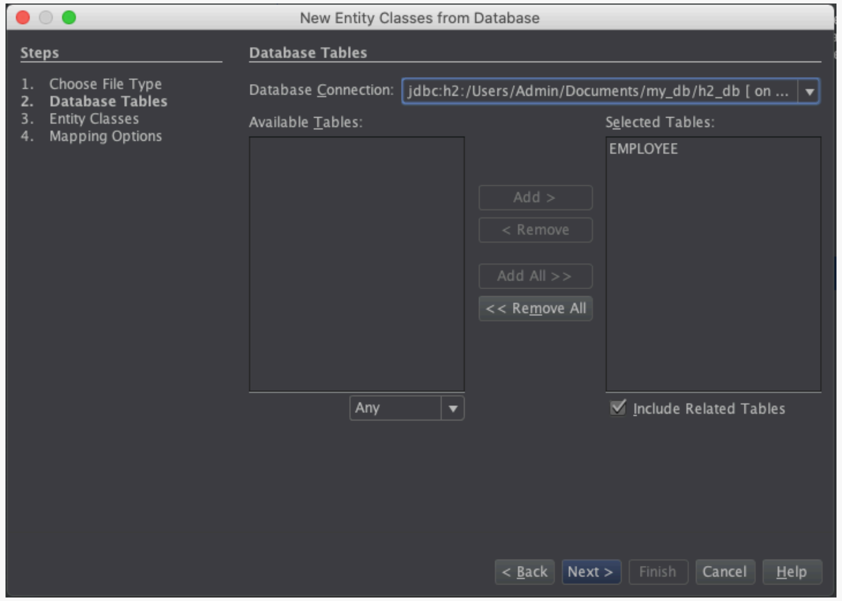
Make sure to define **ID** as a Primary Key

Creating a new Entity

An **entity** is a java object representing data from a relational database

In order to do the persistence, we need to create an object which reflect the attributes of our table.

To generate a new Entity class, we can simply right click on our project > New > Entity Classes from Database



New Entity Classes from Database

Steps

1. Choose File Type
2. **Database Tables**
3. Entity Classes
4. Mapping Options

Database Tables

Database Connection: jdbc:h2:/Users/Admin/Documents/my_db/h2_db [on ...

Available Tables:

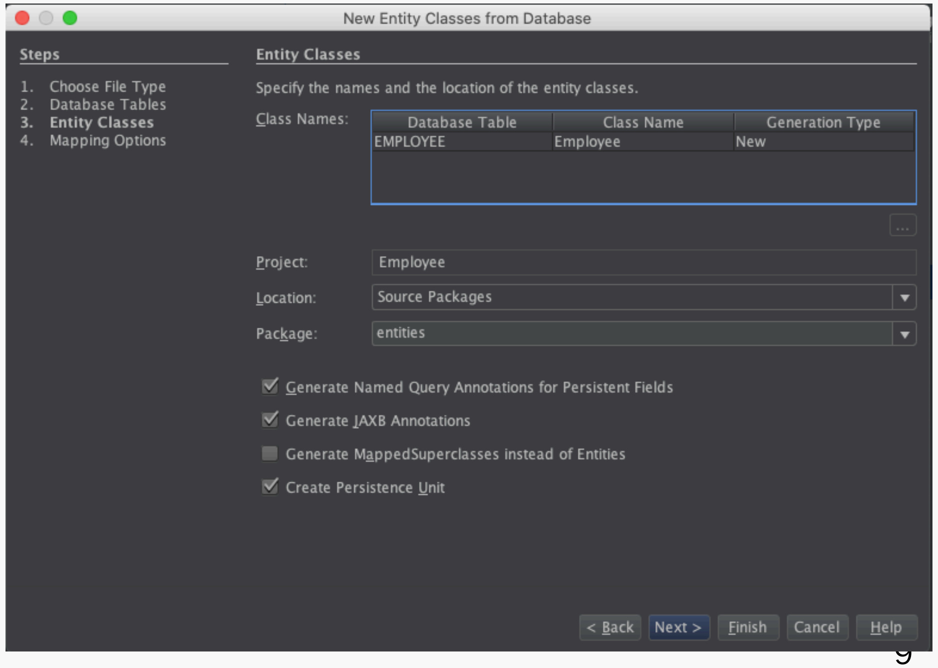
Selected Tables: EMPLOYEE

Add > | < Remove | Add All >> | << Remove All

Any

[✓] Include Related Tables

< Back | Next > | Finish | Cancel | Help



New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
3. **Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
EMPLOYEE	Employee	New

Project: Employee

Location: Source Packages

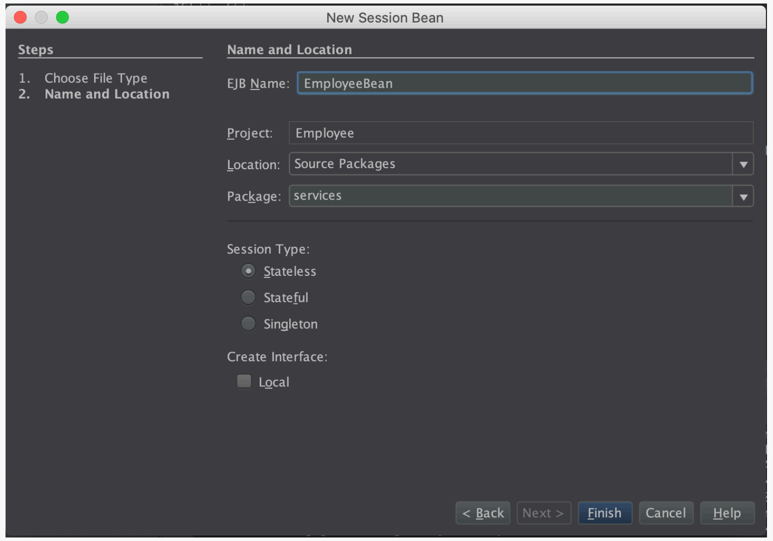
Package: entities

[✓] Generate Named Query Annotations for Persistent Fields
[✓] Generate JAXB Annotations
[ ] Generate MappedSuperclasses instead of Entities
[✓] Create Persistence Unit

< Back | Next > | Finish | Cancel | Help

Create a new session bean for Entity

Our bean will help us to make all the request on the database. For this exercise session, we will create one method which allows us to create a new employee and to save it in the database.


New Session Bean

Steps
1. Choose File Type
2. Name and Location

Name and Location

EJB Name: EmployeeBean

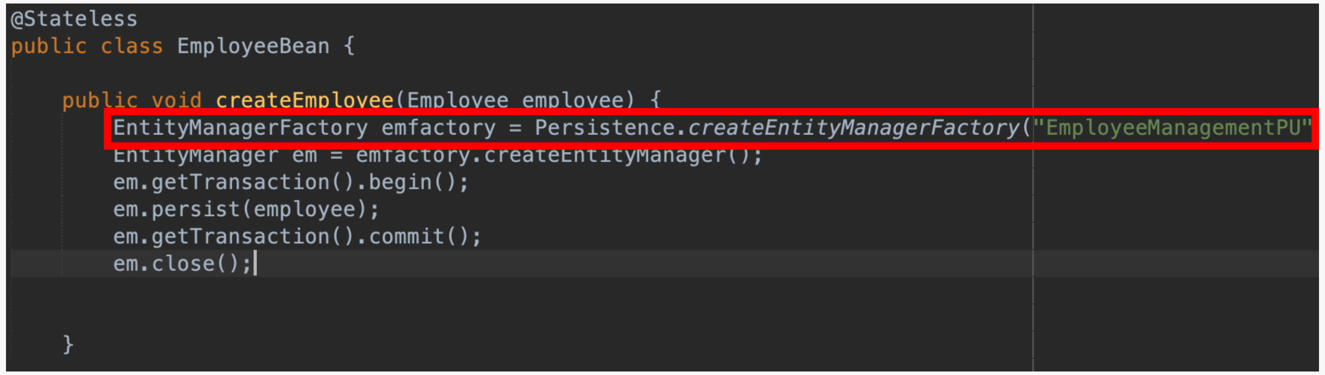
Project: Employee

Location: Source Packages

Package: services

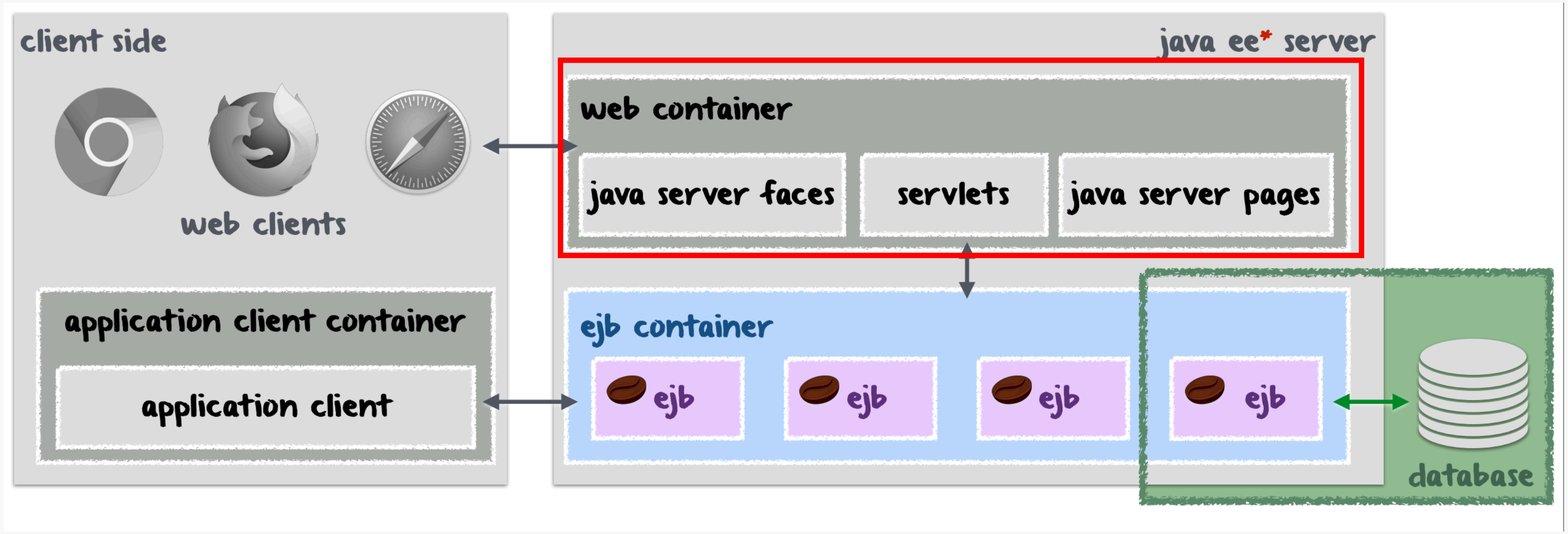
Session Type:
- [✓ Stateless]
- Stateful
- Singleton

Create Interface:
- Local


```
@Stateless
public class EmployeeBean {

    public void createEmployee(Employee employee) {
        EntityManagerFactory emfactory = Persistence.createEntityManagerFactory("EmployeeManagementPU"
        EntityManager em = emfactory.createEntityManager();
        em.getTransaction().begin();
        em.persist(employee);
        em.getTransaction().commit();
        em.close();


    }
```

CreateEntityManagerFactory() creates and return an *EntityManagerFactory* by providing the same unique name provided in ***persistence.xml***

Create a new servlet



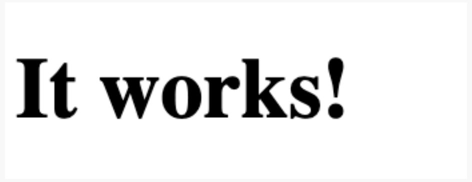
Servlets allow us to make a link between our EJBs and the web clients. To create a new servlet, right click on the package called **servlets** > New > Servlet.

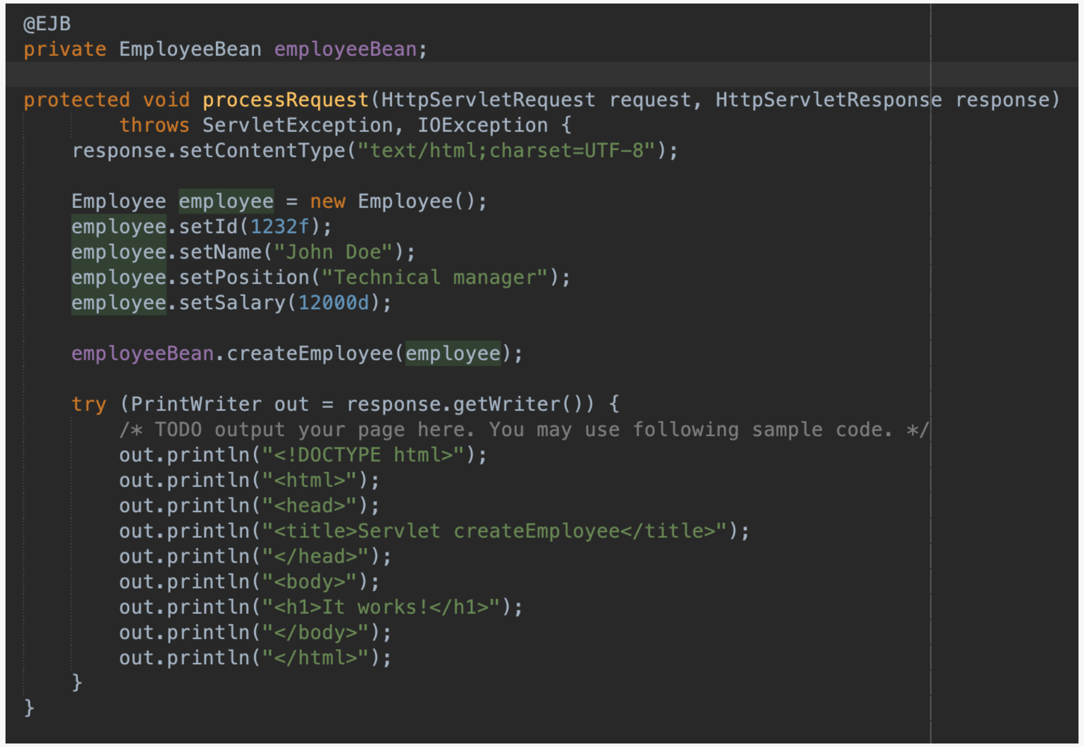
Run the program

In our servlet, we have to import our EJB first (*EmployeeBean*). Then we instanciate a new object called employee (*Employee employee = new Employee()*)

Finally, you have to call the ***createEmployee*** method of our EJB with ***employee*** as argument.

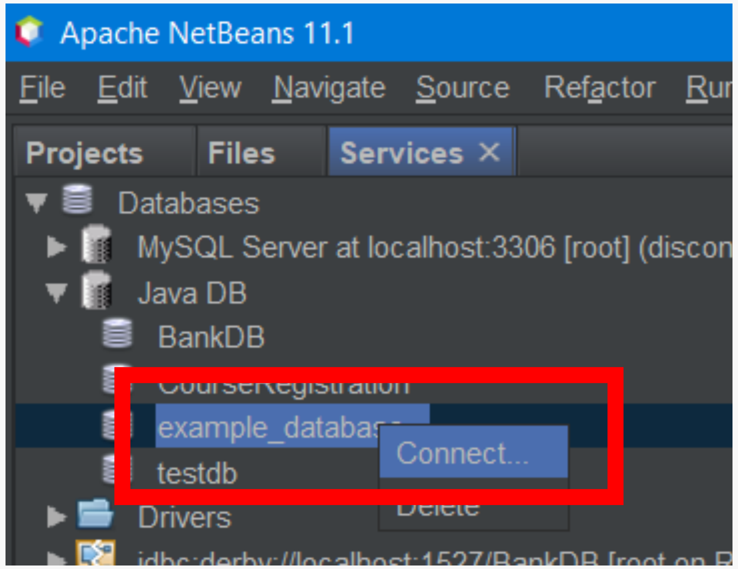
Right click on your servlet > **Run**


It works!

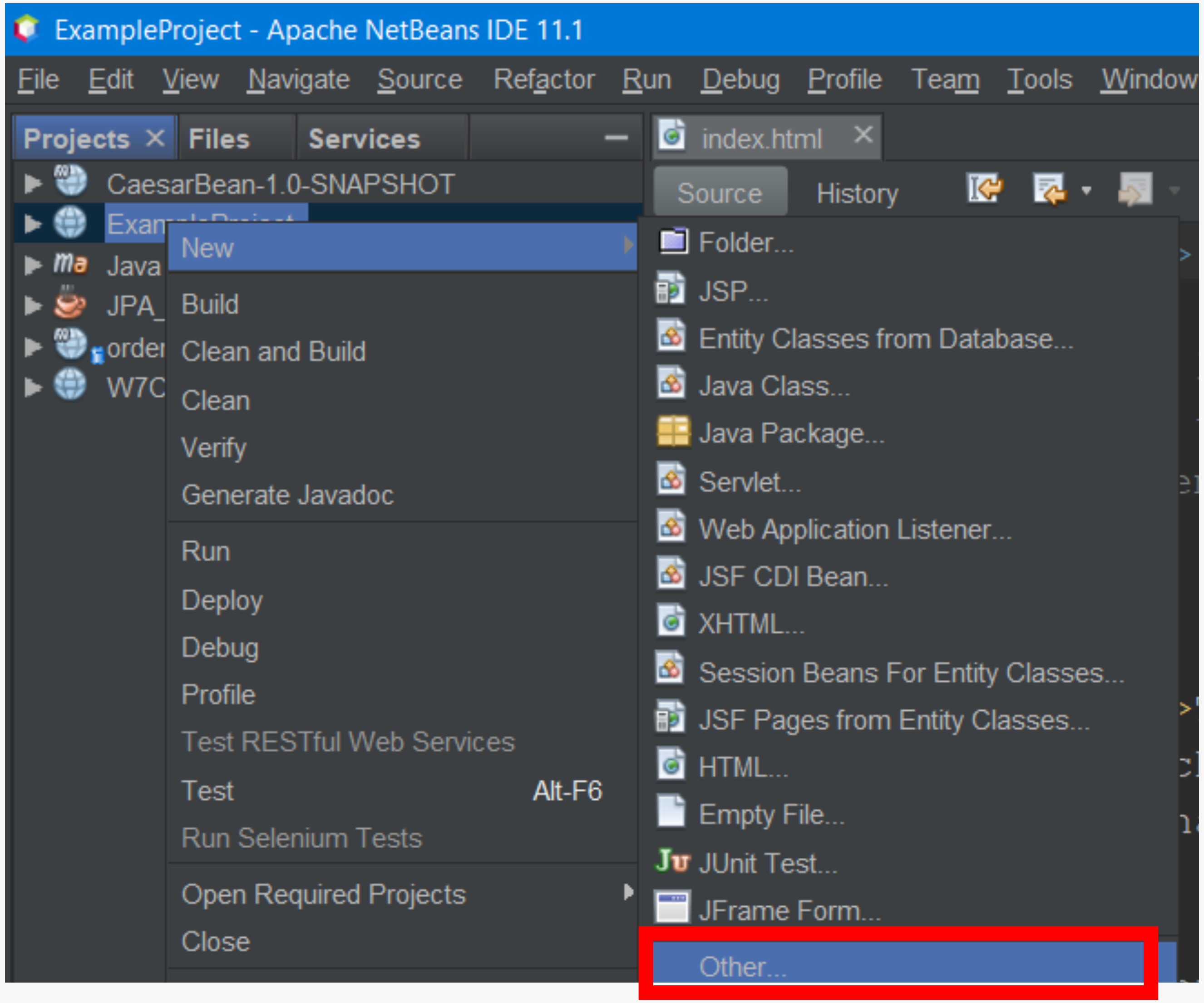

```
@EJB
private EmployeeBean employeeBean;

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");

    Employee employee = new Employee();
    employee.setId(1232f);
    employee.setName("John Doe");
    employee.setPosition("Technical manager");
    employee.setSalary(12000d);

    employeeBean.createEmployee(employee);

    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet createEmployee</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>It works!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Creating JDBC Resource and Connection Pool

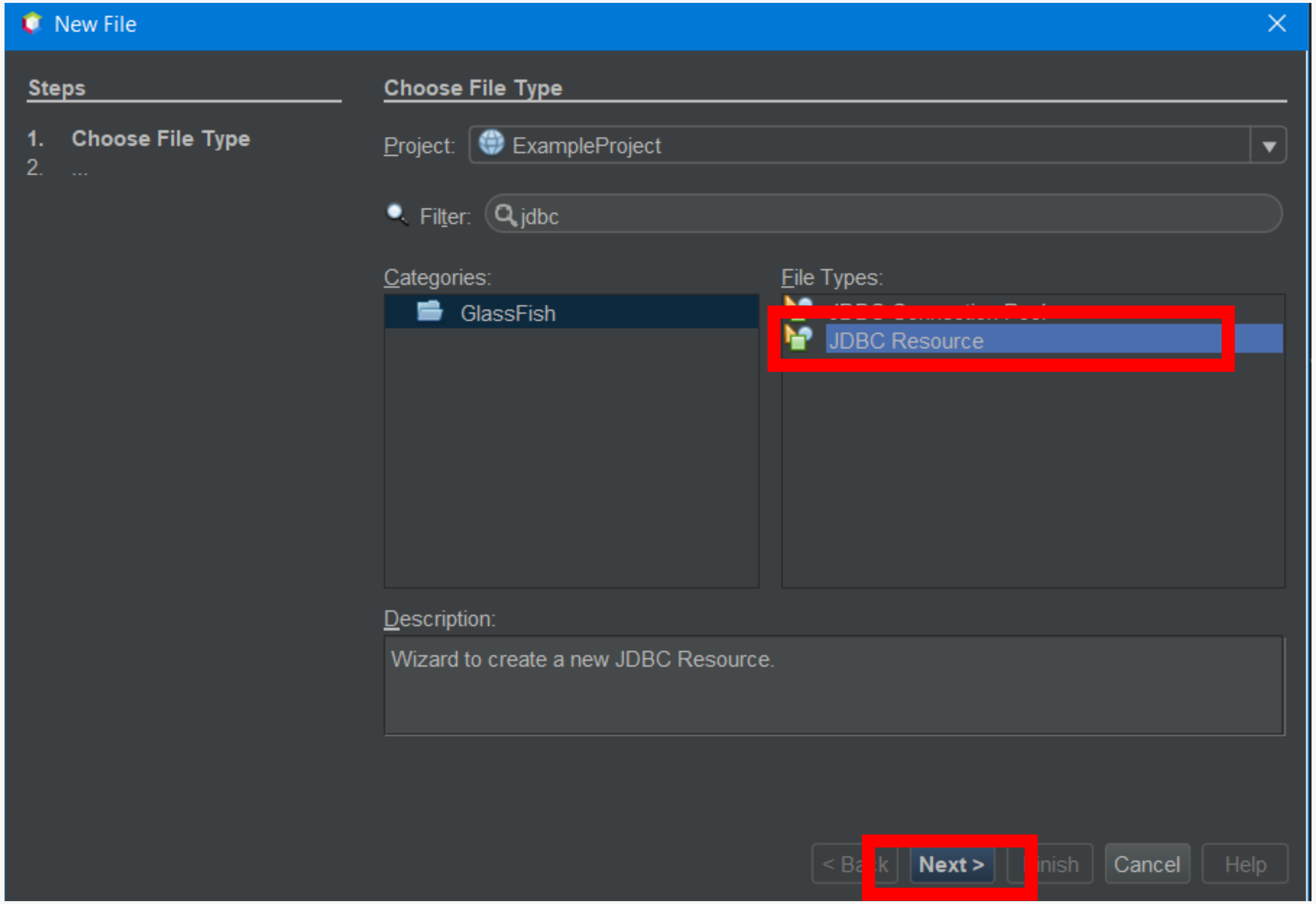
Expand Databases >> Right-click on your database >> Click on Connect …


Apache NetBeans 11.1
File Edit View Navigate Source Refactor Run
Projects Files Services
Databases
MySQL Server at localhost:3306 [root] (discon
Java DB
BankDB
CourseRegistration
example_database
testdb
Drivers
jdbc:derby://localhost:1527/BankDB [root on
Connect...
Delete

Right-click on your project >> New >> Other ...



Select JDBC Resource and click on Next


New File

Steps
1. Choose File Type
2. ...

Choose File Type

Project: ExampleProject

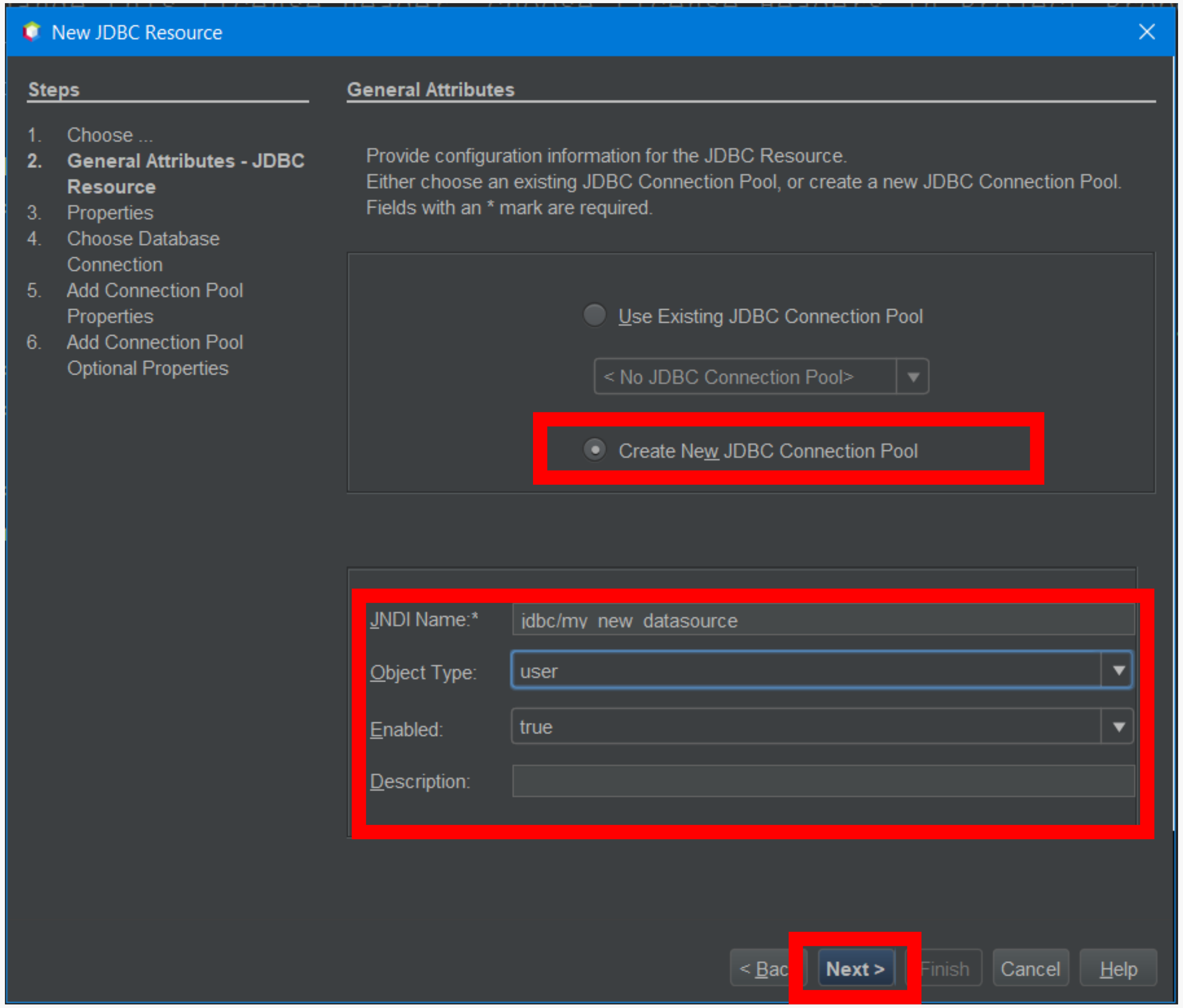
Filter: jdbc

Categories: GlassFish

File Types: JDBC Connection Pool, JDBC Resource

Description:
Wizard to create a new JDBC Resource.

Choose New JDBC Connection Pool and enter a JNDI Name, then click on Next



Steps

1. Choose ...
2. **General Attributes - JDBC Resource**
3. Properties
4. Choose Database Connection
5. Add Connection Pool Properties
6. Add Connection Pool Optional Properties

General Attributes

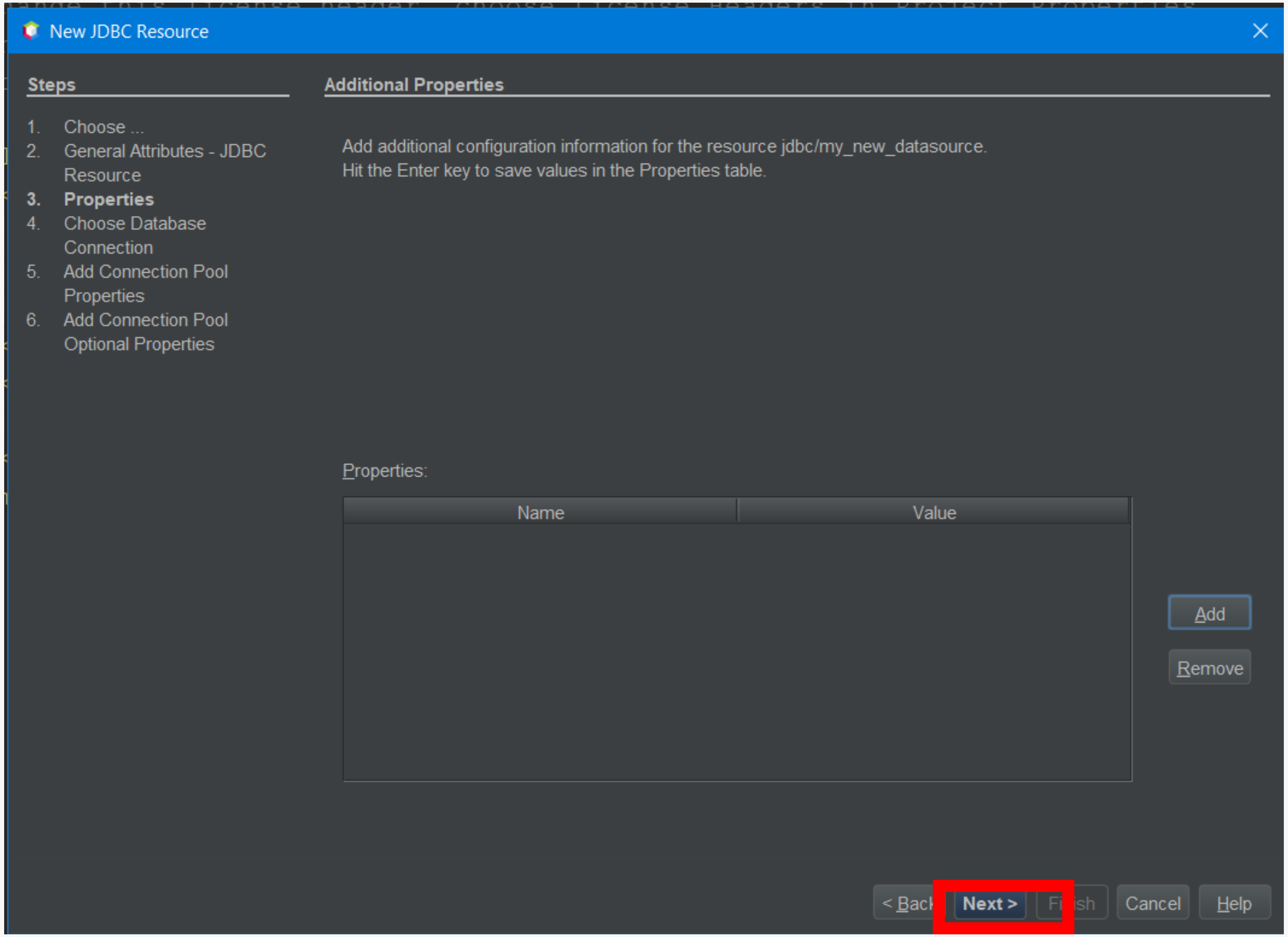
Provide configuration information for the JDBC Resource.
Either choose an existing JDBC Connection Pool, or create a new JDBC Connection Pool.
Fields with an * mark are required.

○ Use Existing JDBC Connection Pool

< No JDBC Connection Pool>

● Create New JDBC Connection Pool

JNDI Name:*	jdbc/my_new_datasource
Object Type:	user
Enabled:	true
Description:	



New JDBC Resource

Steps

1. Choose ...
2. General Attributes - JDBC Resource
3. **Properties**
4. Choose Database Connection
5. Add Connection Pool Properties
6. Add Connection Pool Optional Properties

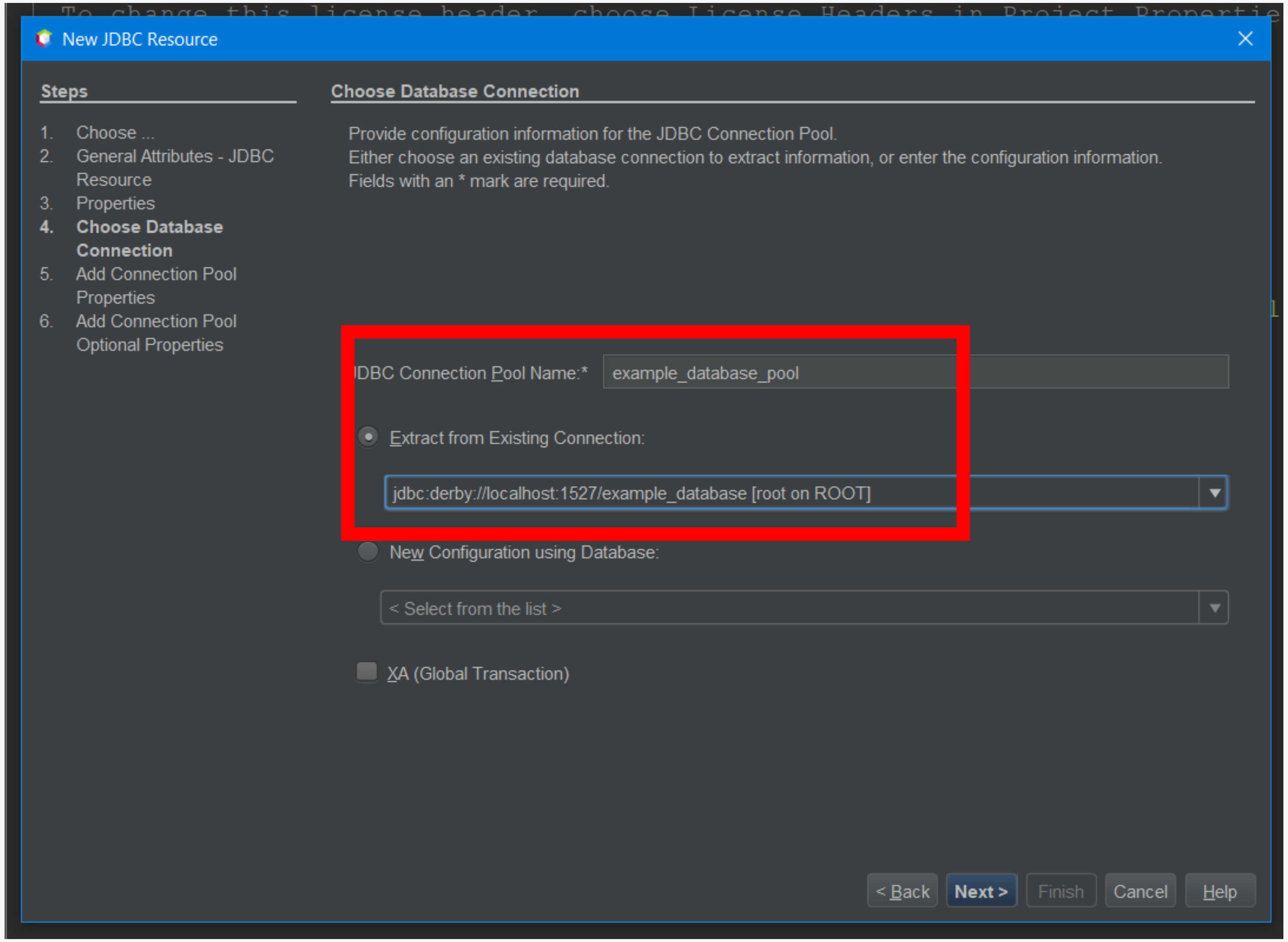
Additional Properties

Add additional configuration information for the resource jdbc/my_new_datasource.
Hit the Enter key to save values in the Properties table.

Properties:

Name	Value

Enter a pool name and select your database connection



To change this license header, choose License Headers in Project Propertie

New JDBC Resource

Steps

1. Choose ...
2. General Attributes - JDBC Resource
3. Properties
4. **Choose Database Connection**
5. Add Connection Pool Properties
6. Add Connection Pool Optional Properties

Choose Database Connection

Provide configuration information for the JDBC Connection Pool.
Either choose an existing database connection to extract information, or enter the configuration information.
Fields with an * mark are required.

JDBC Connection Pool Name:* example_database_pool

[✓ Extract from Existing Connection:]

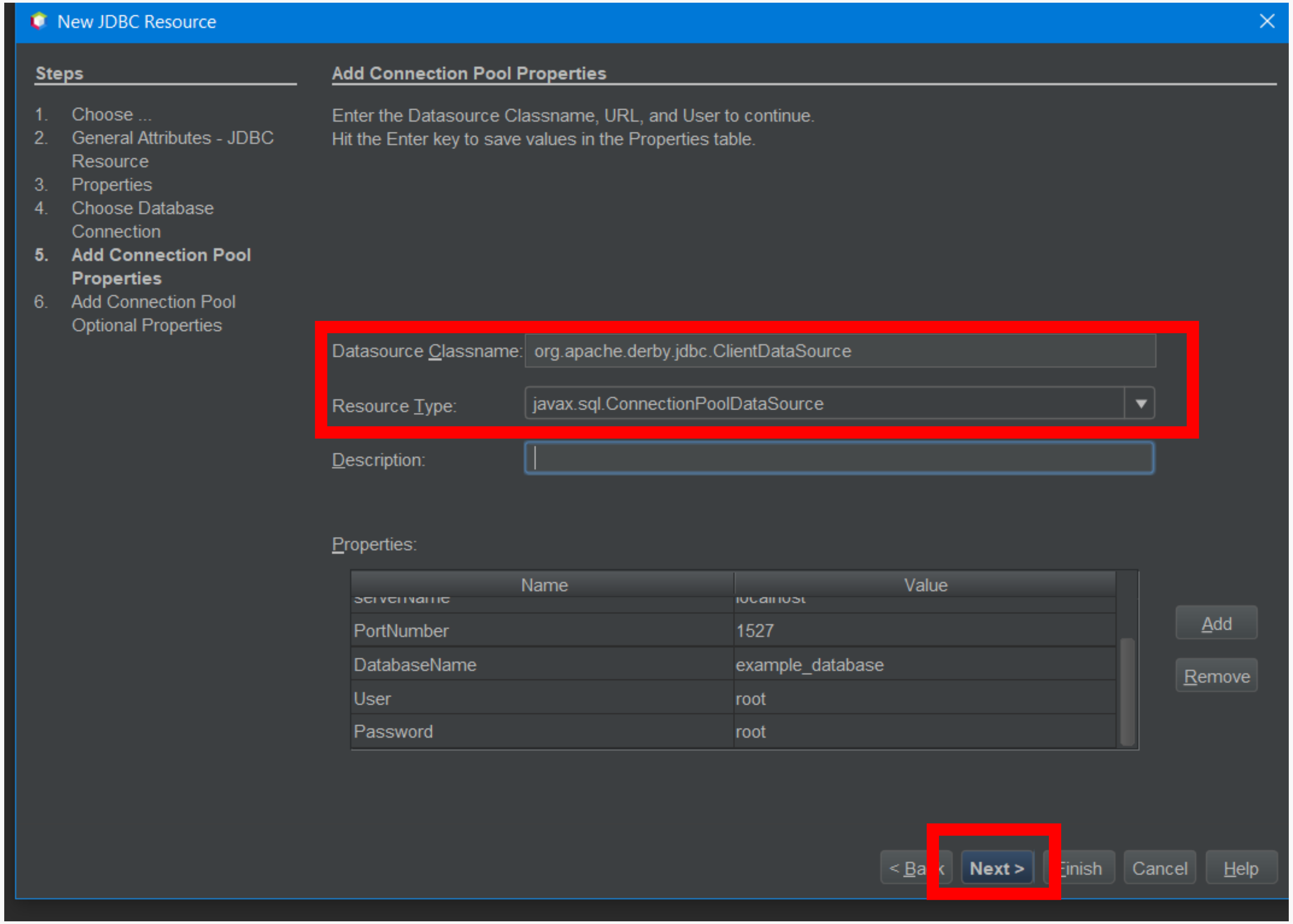
jdbc:derby://localhost:1527/example_database [root on ROOT]

New Configuration using Database:

< Select from the list >

XA (Global Transaction)

Check Resource Type and change if necessary



New JDBC Resource

Steps

1. Choose ...
2. General Attributes - JDBC Resource
3. Properties
4. Choose Database Connection
5. **Add Connection Pool Properties**
6. Add Connection Pool Optional Properties

Add Connection Pool Properties

Enter the Datasource Classname, URL, and User to continue.
Hit the Enter key to save values in the Properties table.

Datasource Classname: org.apache.derby.jdbc.ClientDataSource

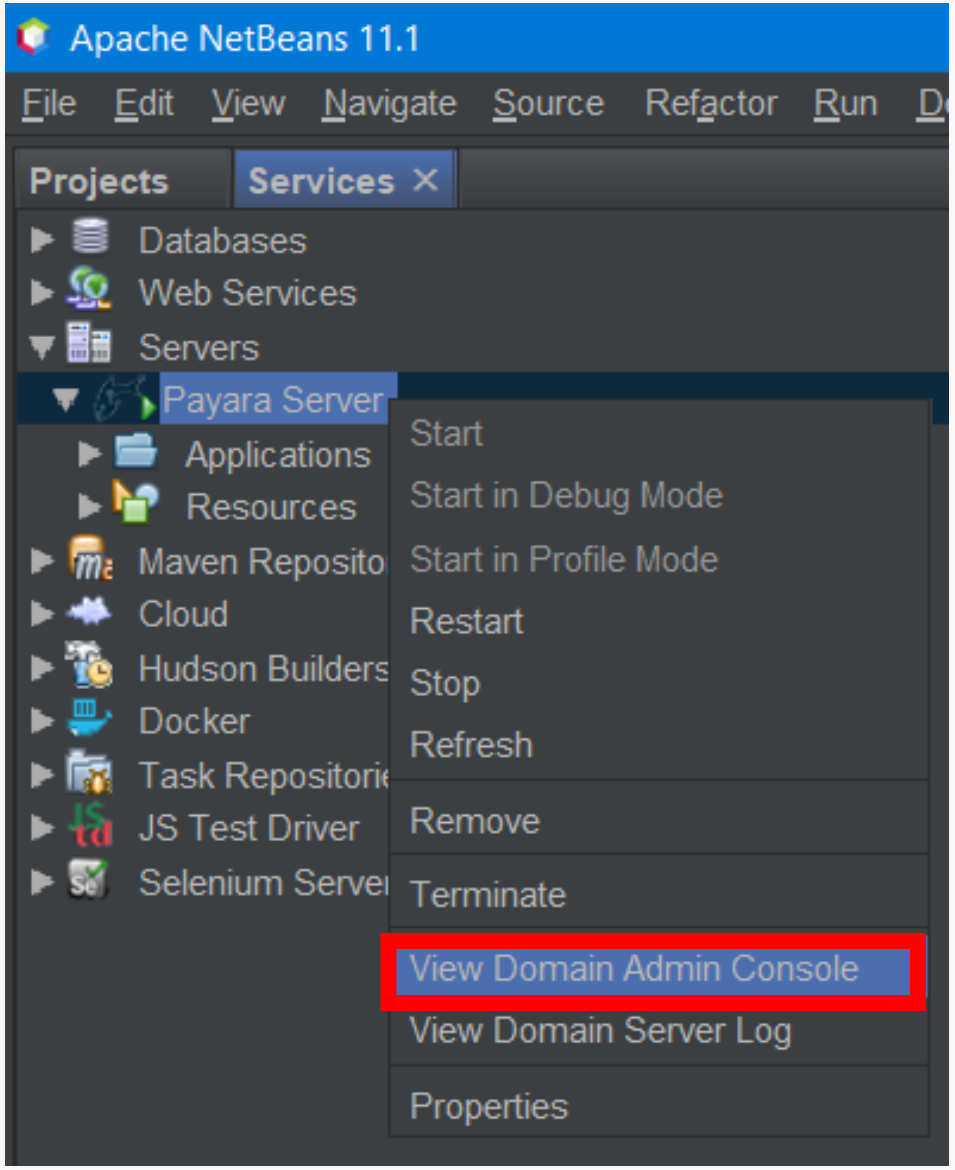
Resource Type: javax.sql.ConnectionPoolDataSource

Description:

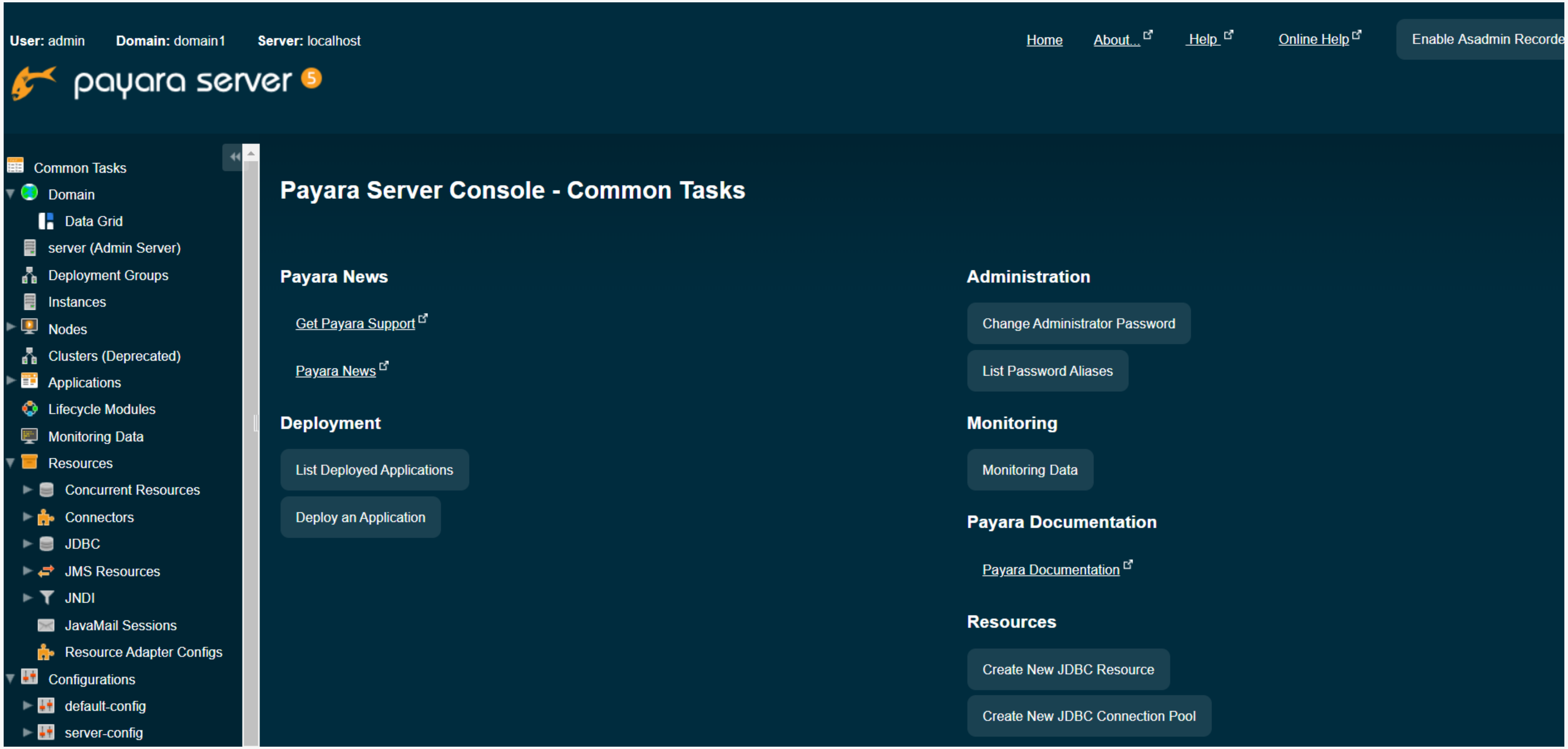
Properties:

Name	Value
serverName	localhost
PortNumber	1527
DatabaseName	example_database
User	root
Password	root

Payara Server >> Right-click >> View Domain Admin Console



Payara Admin Console



Payara Server Console - Common Tasks

Payara News

Get Payara Support

Payara News

Deployment

List Deployed Applications

Deploy an Application

Administration

Change Administrator Password

List Password Aliases

Monitoring

Monitoring Data

Payara Documentation

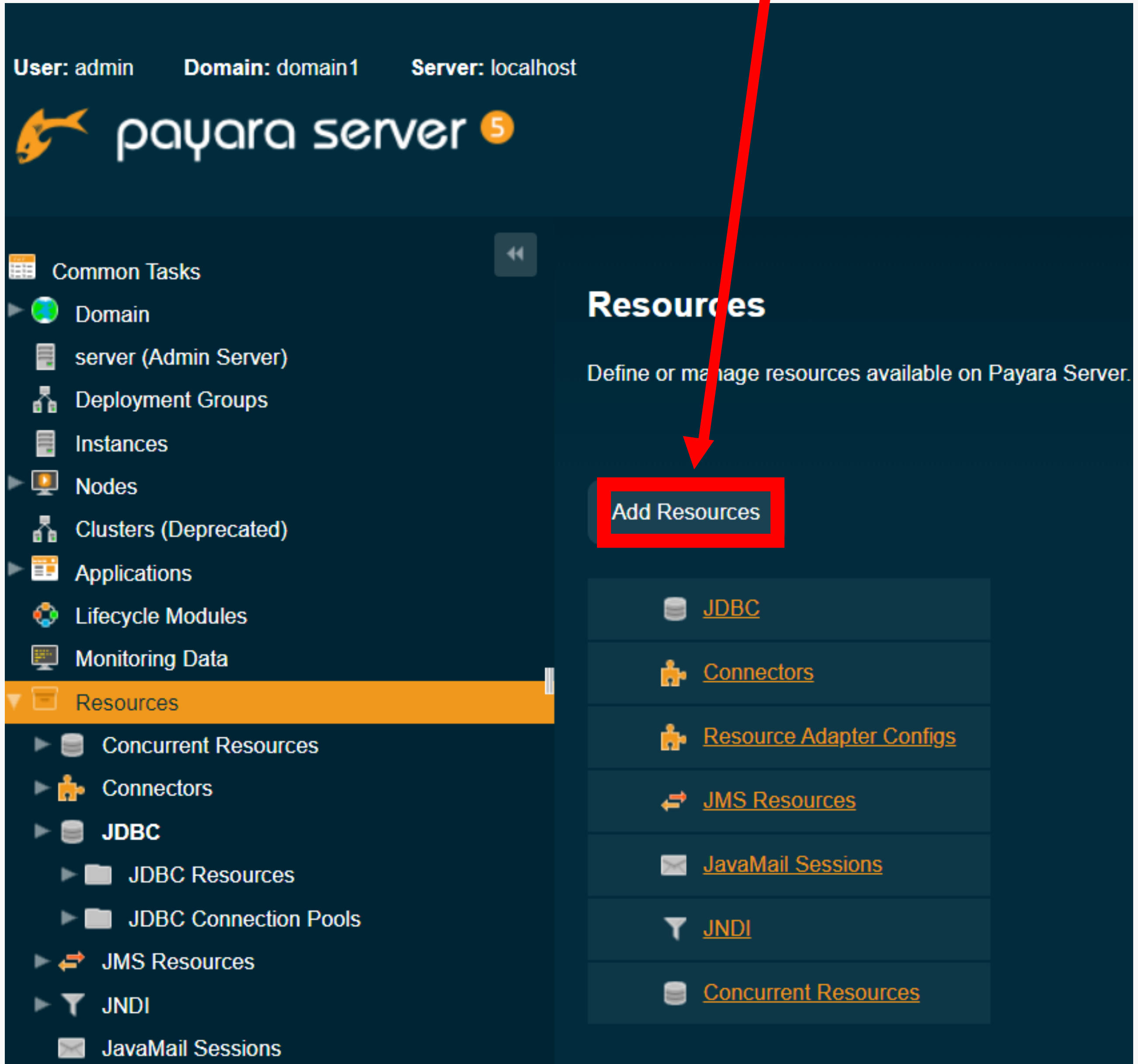
Payara Documentation

Resources

Create New JDBC Resource

Create New JDBC Connection Pool

Resources >> Add Resources

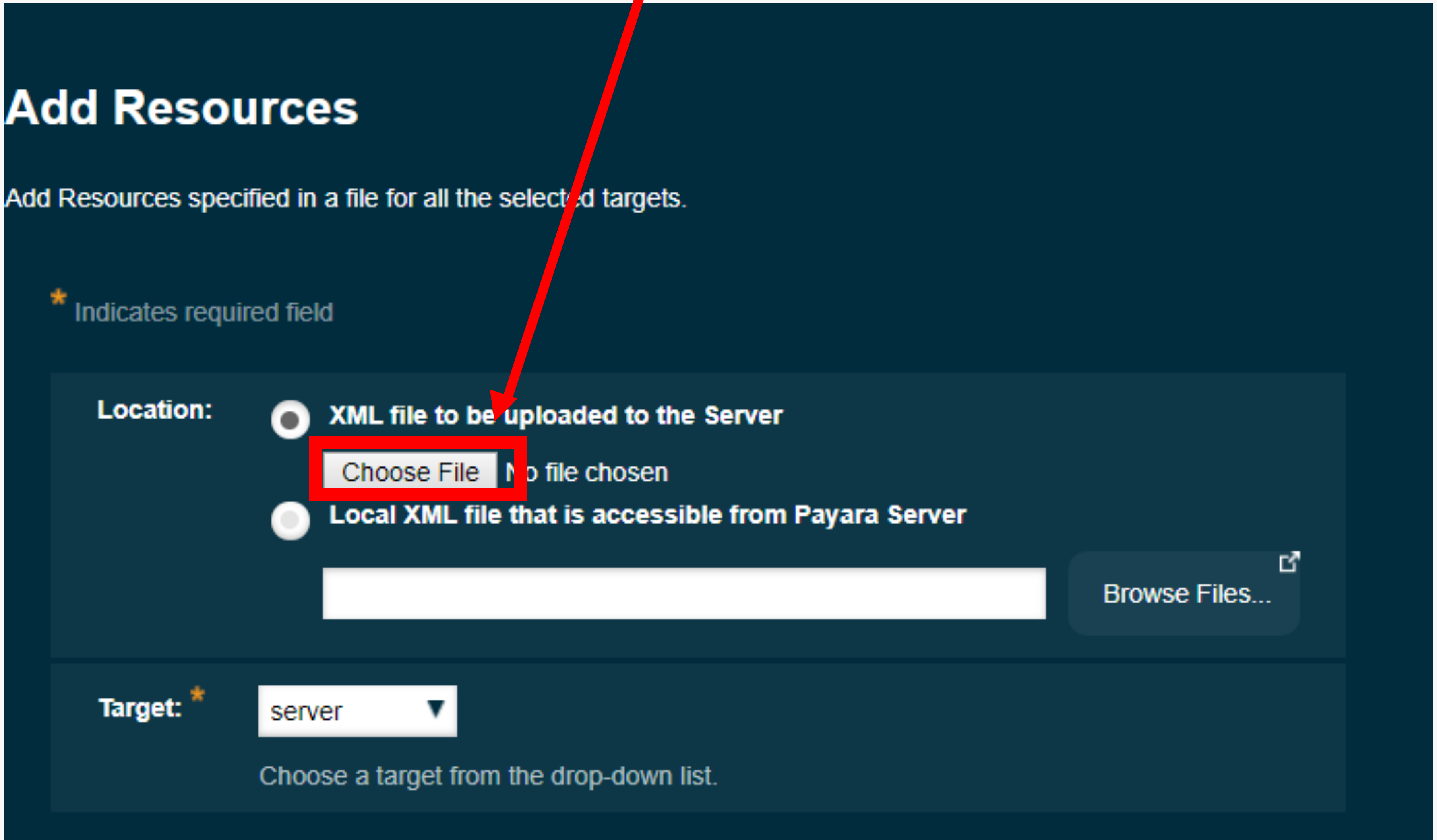


Resources

Define or manage resources available on Payara Server.

Click Choose File

...\<project_name>\setup\glassfish-resources.xml



Add Resources

Add Resources specified in a file for all the selected targets.

* Indicates required field

Location: ● XML file to be uploaded to the Server

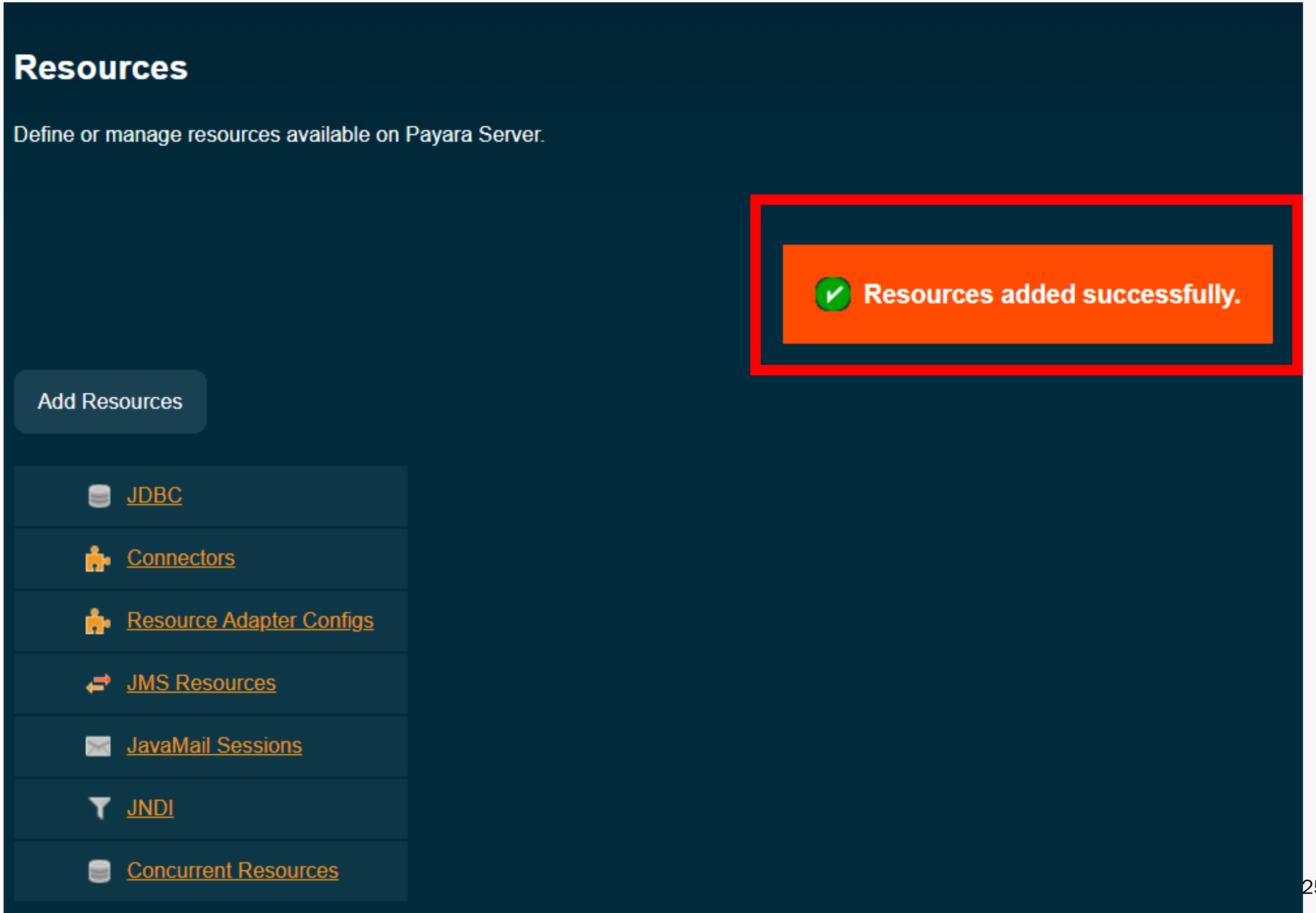
Choose File No file chosen

○ Local XML file that is accessible from Payara Server

Browse Files...

Target: * server

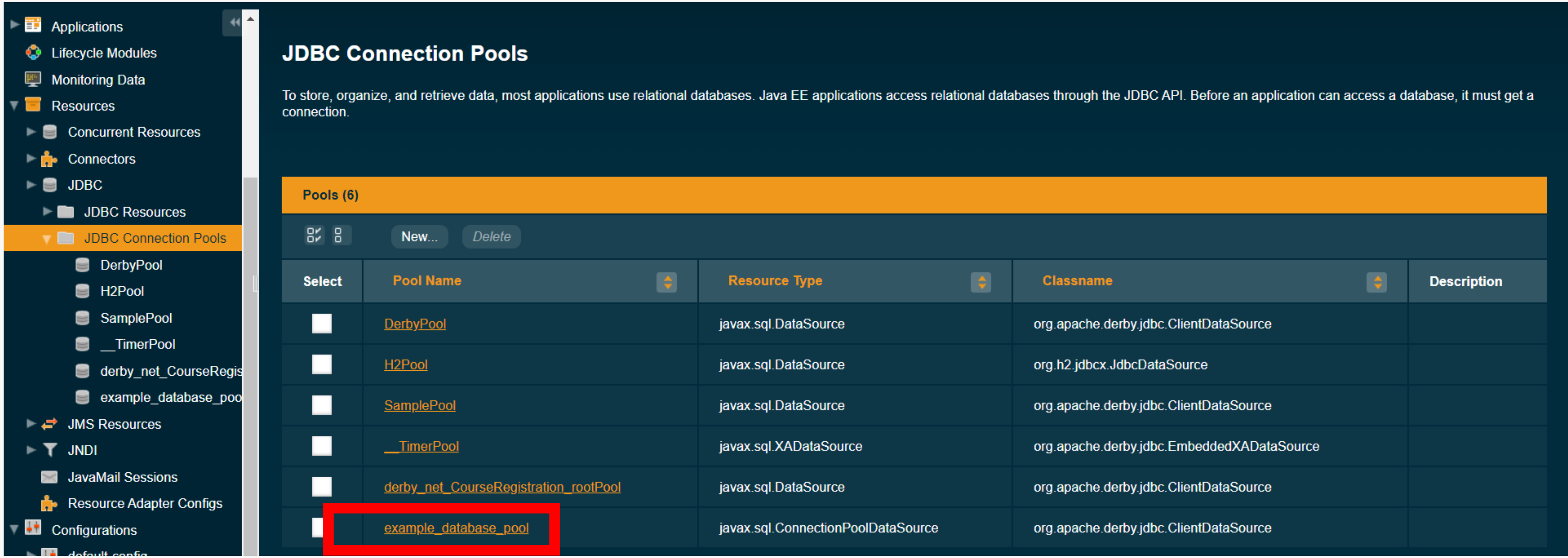
Choose a target from the drop-down list.


Resources

Define or manage resources available on Payara Server.

Resources added successfully.

We have our pool on the admin console, click on it



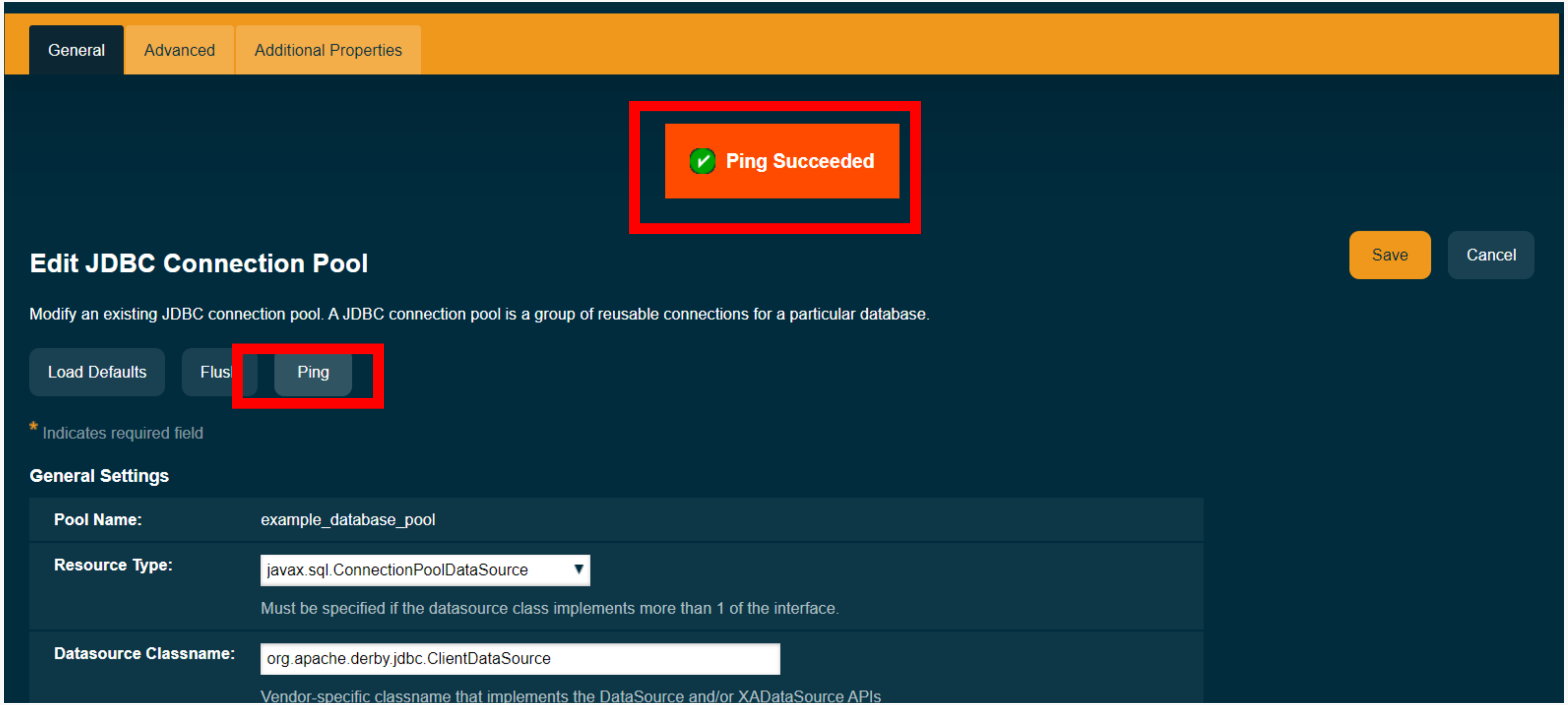
JDBC Connection Pools

To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access a database, it must get a connection.

Pools (6)

Select	Pool Name	Resource Type	Classname	Description
☐	DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
☐	H2Pool	javax.sql.DataSource	org.h2.jdbcx.JdbcDataSource	
☐	SamplePool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
☐	__TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource	
☐	derby_net_CourseRegistration_rootPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
☐	example_database_pool	javax.sql.ConnectionPoolDataSource	org.apache.derby.jdbc.ClientDataSource	

Ping it!



Edit JDBC Connection Pool

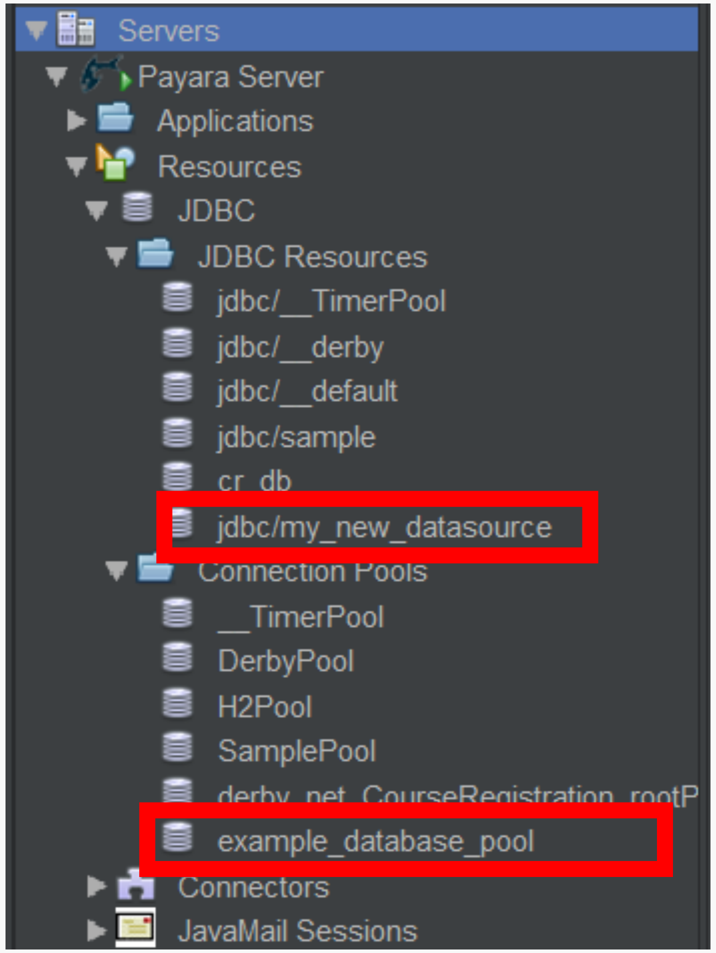
Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

* Indicates required field

General Settings

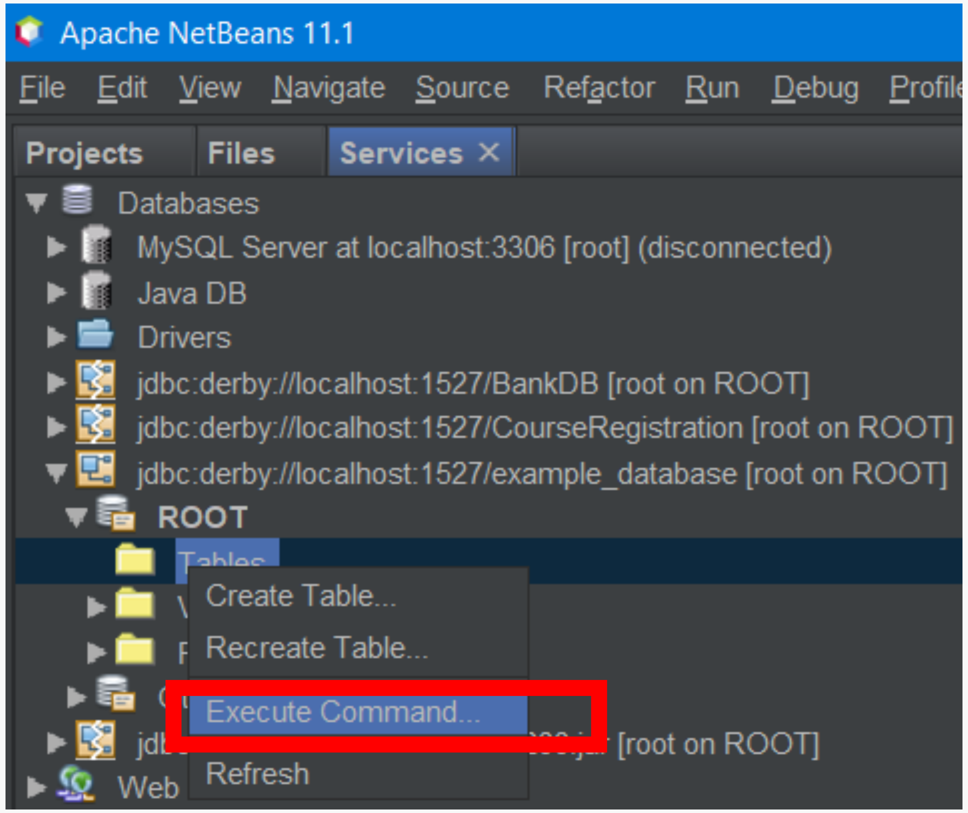
Pool Name:	example_database_pool
Resource Type:	javax.sql.ConnectionPoolDataSource ▼ Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:	org.apache.derby.jdbc.ClientDataSource Vendor-specific classname that implements the DataSource and/or XADataSource APIs

JDBC Resource and Connection Pool are under Payara Server >> Resources >> JDBC >> …

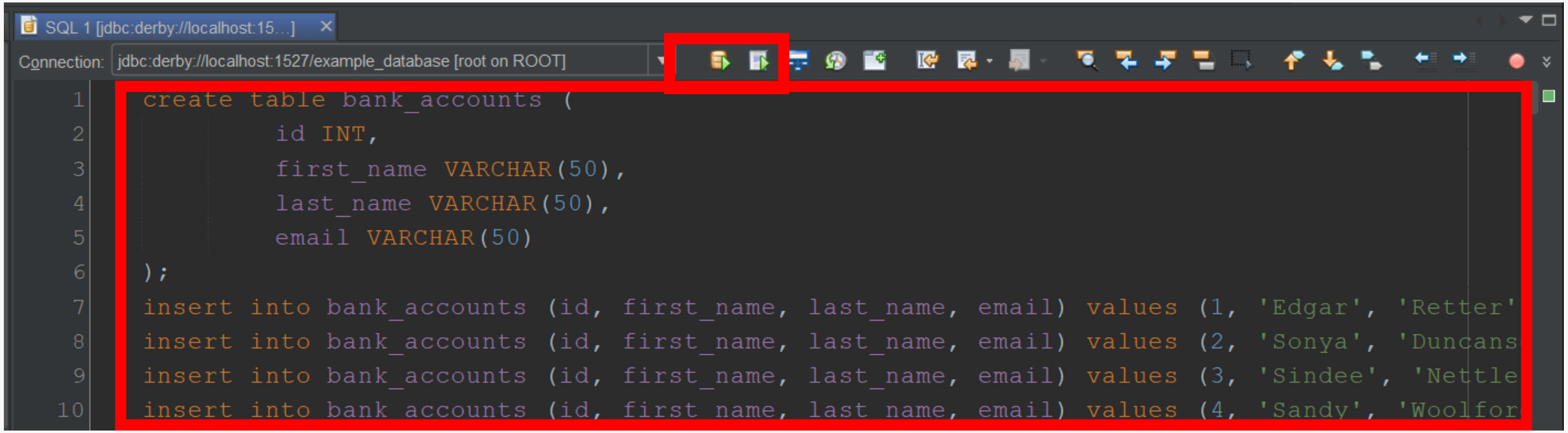


```
▼ Servers
  ▼ Payara Server
    ▶ Applications
    ▼ Resources
      ▼ JDBC
        ▼ JDBC Resources
            jdbc/__TimerPool
            jdbc/__derby
            jdbc/__default
            jdbc/sample
            cr_db
            jdbc/my_new_datasource
        ▼ Connection Pools
            __TimerPool
            DerbyPool
            H2Pool
            SamplePool
            derby_net_CourseRegistration_rootP
            example_database_pool
      ▶ Connectors
      ▶ JavaMail Sessions
```

Expand your connection, right-click on Tables, then click on Execute Command...

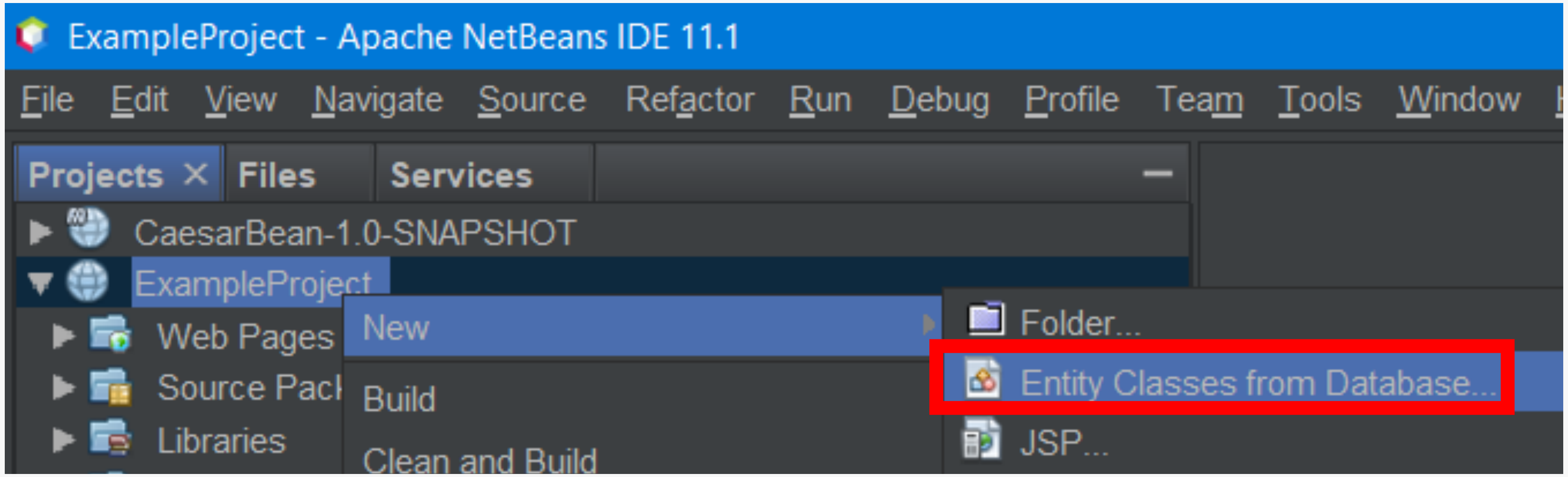

Apache NetBeans 11.1
File
Edit
View
Navigate
Source
Refactor
Run
Debug
Profile
Projects
Files
Services
Databases
MySQL Server at localhost:3306 [root] (disconnected)
Java DB
Drivers
jdbc:derby://localhost:1527/BankDB [root on ROOT]
jdbc:derby://localhost:1527/CourseRegistration [root on ROOT]
jdbc:derby://localhost:1527/example_database [root on ROOT]
ROOT
Tables
Create Table...
Recreate Table...
Execute Command...
Refresh
[root on ROOT]
Web

Copy and paste SQL queries in the file and click on execute

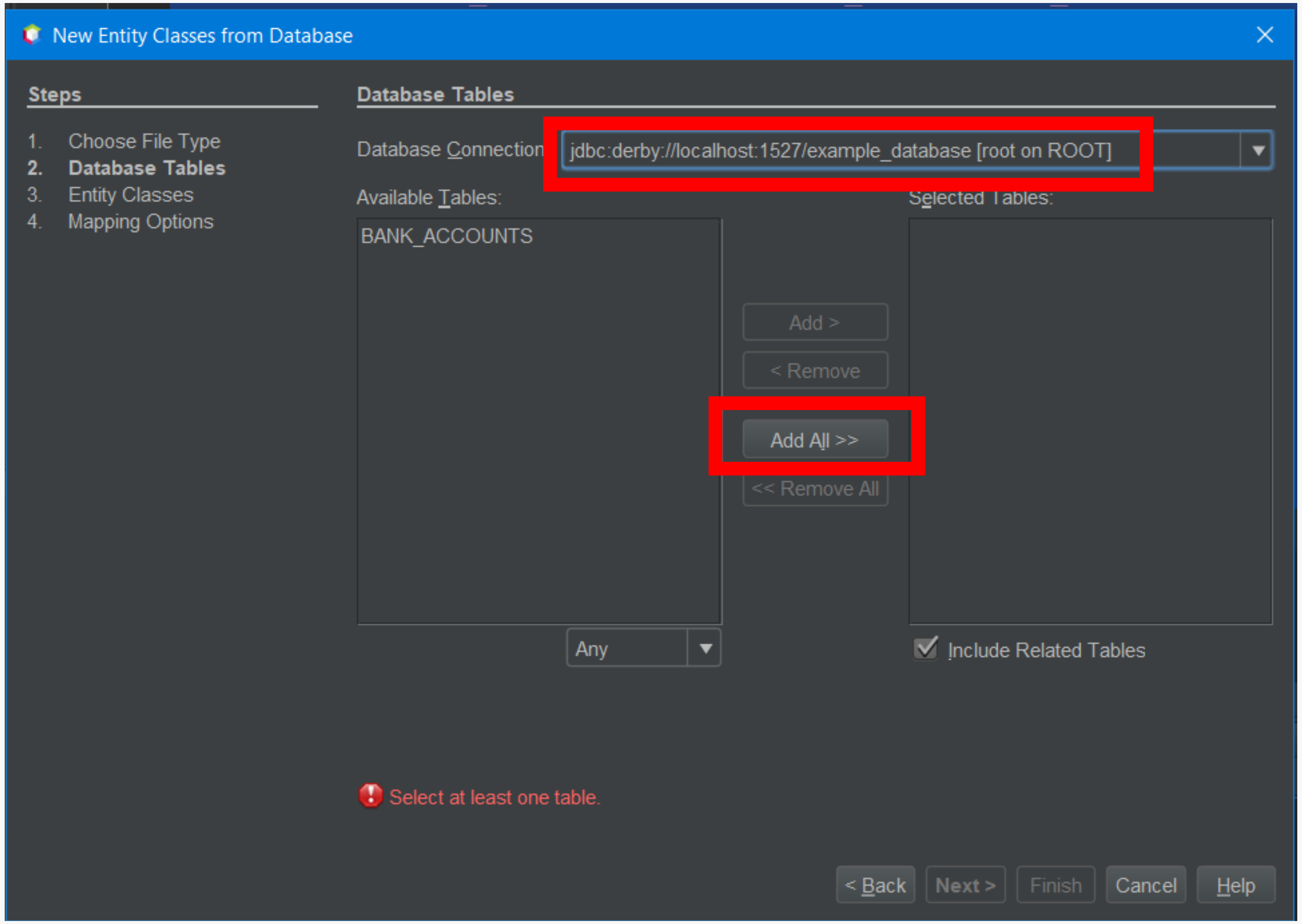

SQL 1 [jdbc:derby://localhost:15...]
Connection: jdbc:derby://localhost:1527/example_database [root on ROOT]

```
create table bank_accounts (
        id INT,
        first_name VARCHAR(50),
        last_name VARCHAR(50),
        email VARCHAR(50)
);
insert into bank_accounts (id, first_name, last_name, email) values (1, 'Edgar', 'Retter'
insert into bank_accounts (id, first_name, last_name, email) values (2, 'Sonya', 'Duncans
insert into bank_accounts (id, first_name, last_name, email) values (3, 'Sindee', 'Nettle
insert into bank_accounts (id, first_name, last_name, email) values (4, 'Sandy', 'Woolfor
```

Right-click on your project >> New >> Entity Classes from Database...


ExampleProject - Apache NetBeans IDE 11.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window
Projects Files Services
CaesarBean-1.0-SNAPSHOT
ExampleProject
Web Pages
Source Pack
Libraries
New
Build
Clean and Build
Folder...
Entity Classes from Database...
JSP...

Choose your database connection and click on Add All, then click on Next


New Entity Classes from Database

Steps
1. Choose File Type
2. **Database Tables**
3. Entity Classes
4. Mapping Options

Database Tables

Database Connection: jdbc:derby://localhost:1527/example_database [root on ROOT]

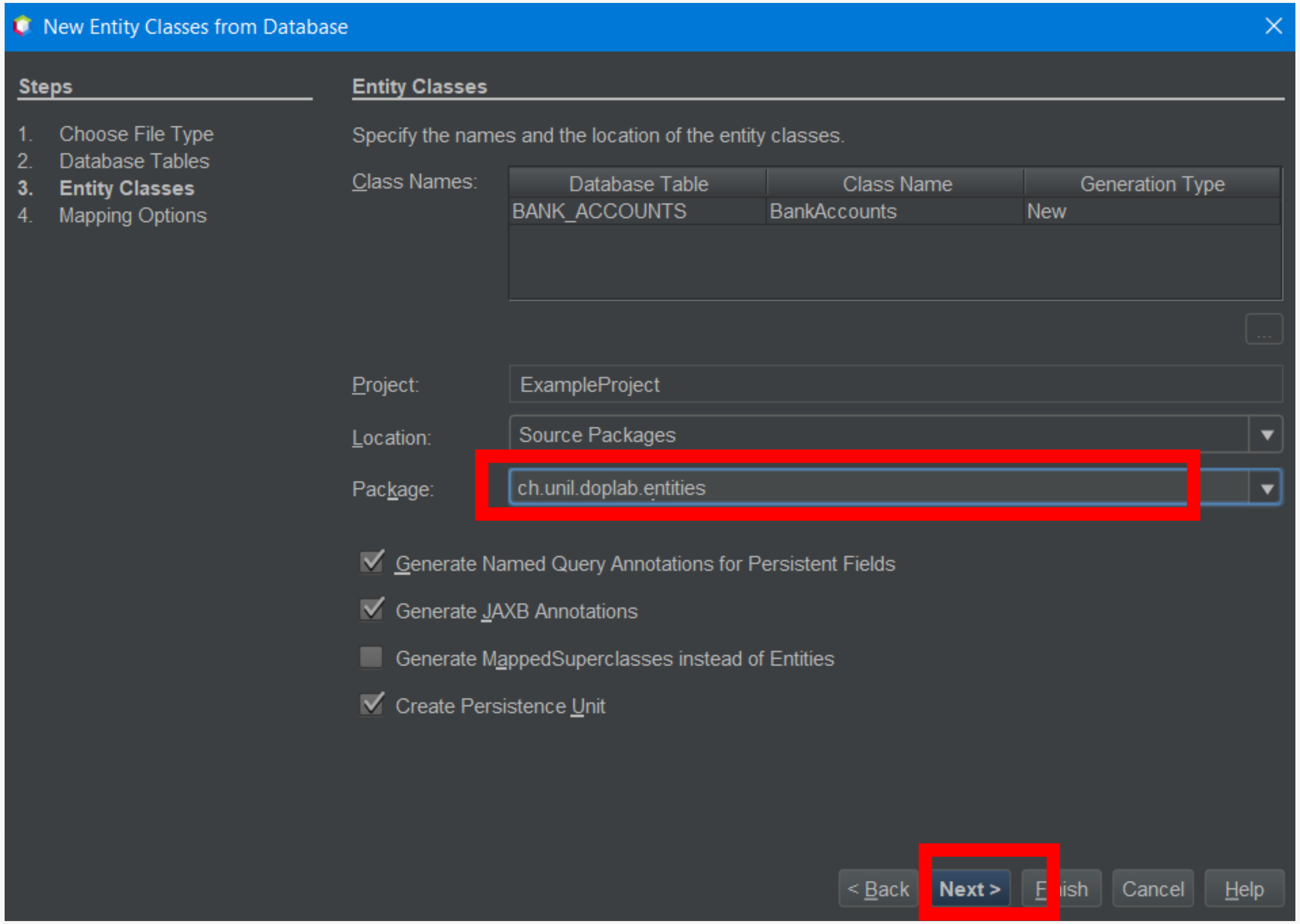
Available Tables:
BANK_ACCOUNTS

Selected Tables:

Include Related Tables ✓

Select at least one table.

Enter **ch.unil.doplab.entities** as the package name, then click on Next



New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
3. **Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
BANK_ACCOUNTS	BankAccounts	New

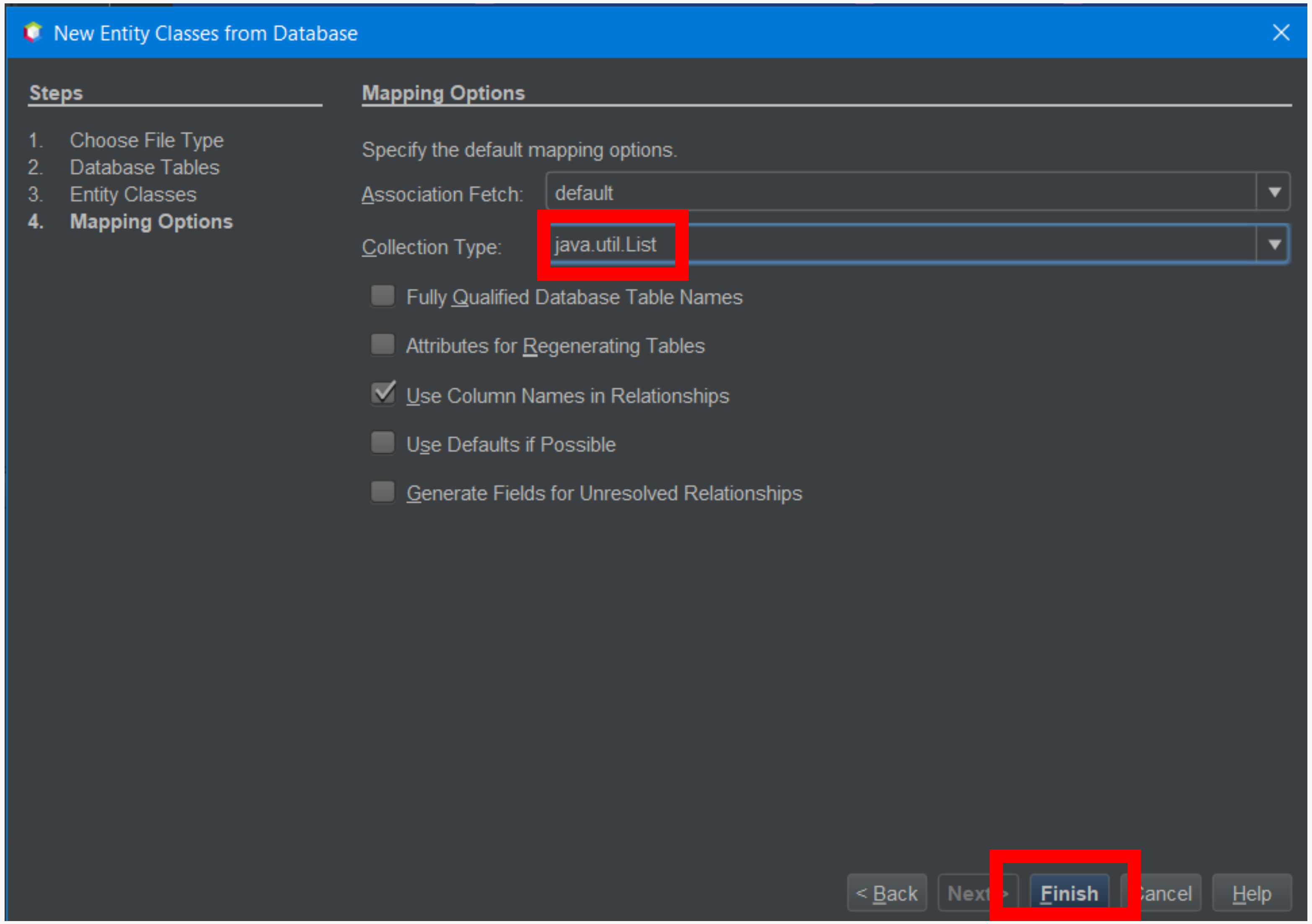
Project: ExampleProject

Location: Source Packages

Package: ch.unil.doplab.entities

- [x] Generate Named Query Annotations for Persistent Fields
- [x] Generate JAXB Annotations
- [ ] Generate MappedSuperclasses instead of Entities
- [x] Create Persistence Unit

Choose **java.util.List** as the Collection Type, then click on Finish



Mapping Options

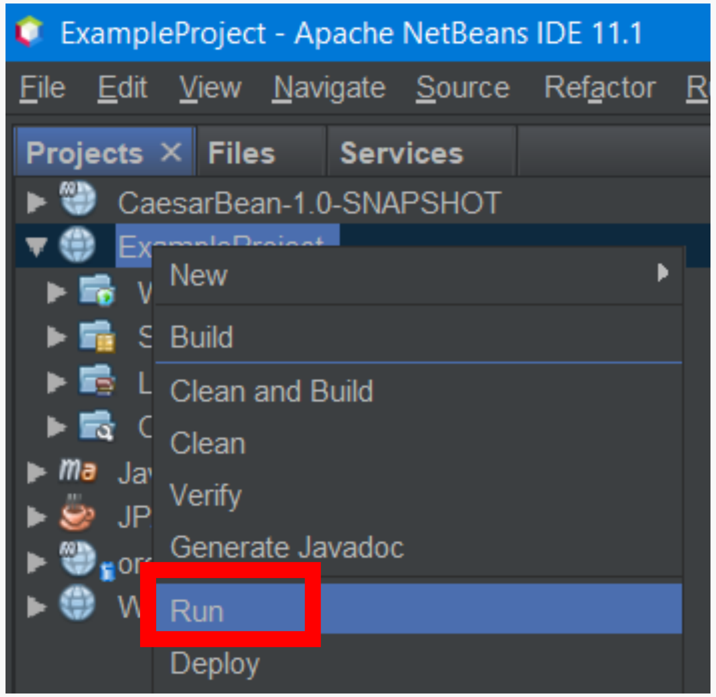
Specify the default mapping options.

Association Fetch: default

Collection Type: java.util.List

- [ ] Fully Qualified Database Table Names
- [ ] Attributes for Regenerating Tables
- [x] Use Column Names in Relationships
- [ ] Use Defaults if Possible
- [ ] Generate Fields for Unresolved Relationships

Right-click on your project and click on **Run!**



Now, it is your turn

1. Create A Database Connection (called **CourseRegistration**)
2. Create JDBC Resource (call it **cr_db**)
3. Create Connection Pool (call it **derby_net_CourseRegistration_rootPool**)
4. Create your tables and insert values using queries in ***course_registration_db_queries.sql*** file
5. Create Entity Classes From Database
6. Change toString() methods in Course and Student classes as they are shown in the next slide.
7. Run & Play with your project!

Course.java

```
...
    @Override
    public String toString() {
        return "Couse ID: " + this.courseId
                + " <br>Course Name: " + this.courseName
                + " <br>Course Credits: " + this.courseCredits;
    }
...
```

Student.java

```
...
    @Override
    public String toString() {
        return "Student ID: " + this.studentId
                + " <br>Student Name: " + this.studentFirstName
                + " <br>Student LastName: " + this.studentLastName;
    }
...
```