pentaho

Version 5.3

Install DI) Server with Your Own DI Repository

Copyright Page

Help and Support Resources

To view the most up-to-date help content, visit https://help.pentaho.com.

If you do not find answers to your questions here, please contact your Pentaho technical support representative.

Support-related questions should be submitted through the Pentaho Customer Support Portal at http://support.pentaho.com.

For information about how to purchase support or enable an additional named support contact, please contact your sales representative, or send an email to sales@pentaho.com.

For information about instructor-led training, visit http://www.pentaho.com/training.

Liability Limits and Warranty Disclaimer

The author(s) of this document have used their best efforts in preparing the content and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, express or implied, with regard to these programs or the documentation contained in this book.

The author(s) and Pentaho shall not be liable in the event of incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the programs, associated instructions, and/or claims.

Trademarks

The trademarks, logos, and service marks ("Marks") displayed on this website are the property of Pentaho Corporation or third party owners of such Marks. You are not permitted to use, copy, or imitate the Mark, in whole or in part, without the prior written consent of Pentaho Corporation or such third party. Trademarks of Pentaho Corporation include, but are not limited, to "Pentaho", its products, services and the Pentaho logo.

Trademarked names may appear throughout this website. Rather than list the names and entities that own the trademarks or inserting a trademark symbol with each mention of the trademarked name, Pentaho Corporation states that it is using the names for editorial purposes only and to the benefit of the trademark owner, with no intention of infringing upon that trademark.

Third-Party Open Source Software

For a listing of open source software used by each Pentaho component, navigate to the folder that contains the Pentaho component. Within that folder, locate a folder named licenses. The licenses folder contains HTML.files that list the names of open source software, their licenses, and required attributions.

Contact Us

Global Headquarters Pentaho Corporation Citadel International, Suite 460

5950 Hazeltine National Drive Orlando, FL 32822

Phone: +1 407 812-OPEN (6736)

Fax: +1 407 517-4575

http://www.pentaho.com

Sales Inquiries: sales@pentaho.com

Introduction

This section explains how to install Data Integration (DI) Server and configure it to use the DI Repository database of your choice. The DI Repository contains solution content, scheduling, and audit tables needed for the DI Server to operate. You can house the DI Repository on PostgreSQL, MySQL, or Oracle. With this installation option, you must supply, install, and configure your chosen database yourself.

Prerequisites

Read Select DI Installation Option to make sure that this is the best installation option for you. Then, check the Supported Technologies tables to make sure that your server computer, DI Repository database, and web browser meet Pentaho's requirements for the this version of the software.

Expertise

The topics in this section are written for IT administrators, evaluators, and analysts who have who have access to the server on which the DI Server will be installed. You should know where data is stored, how to connect to it, details about the computing environment, and how to use the command line to issue commands for Microsoft Windows or Linux. You should also know how to install a database.

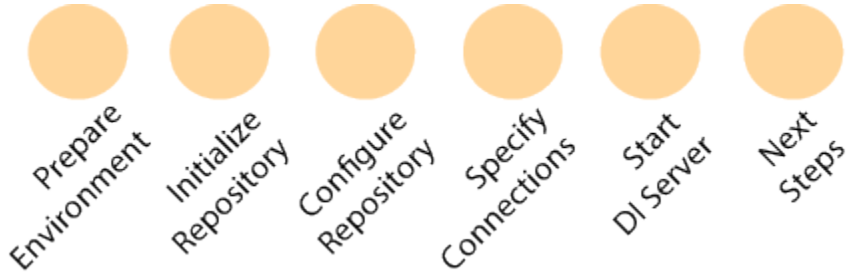
Tools

You will need a text editor and a zip tool to complete some of the steps in this installation process.

Login Credentials

All of the tasks in this section require that you have the appropriate permissions and accesses required to install software on servers and workstations.

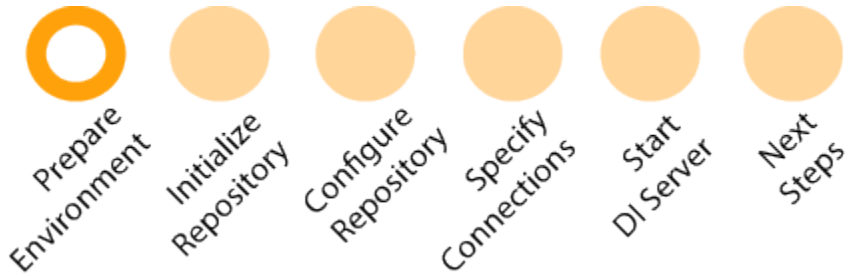
Overview of the Installation Process



To install the DI Server, perform the tasks indicated in the *guidepost*.

- **Prepare Environment:** Explains how to create a pentaho user account, create the directory structure, download DI Server files from our web site, install your selected DI Repository database, and set up system environment variables.
- **Initialize Repository:** Provides information about how to run DDL scripts that create tables for the DI Repository.
- **Configure Repository:** Provides information about how to configure the DI Repositories on your selected database.
- **Specify Connections:** Explains how to specify JDBC connections to the DI Repository.
- **Start DI Server:** Explains how to start the DI Server.
- **Next Steps:** Indicates what to do after the DI Server has been installed.

Prepare Environment



To prepare the computer on which you plan to install the DI Server, complete these tasks.

Create User Account

Create Windows User Account

If you plan to install on a server that runs Windows, you do not need to create a special user account on the server. However, you should use an account that has administrator privileges to complete the tasks in this section.

Create Linux User Account

If you plan to install the DI Server in a Linux environment you must create a user account named **pentaho** on the server computer. By default, license information for Pentaho products is stored in the home directory for this account. Use this account to perform installation tasks that do not require root access. Also, use this account to run start and stop server scripts.

1. Open a **Terminal** window on the server. If you plan to install the DI Server on a remote computer, establish an OpenSSH session to the remote server.
2. In the **Terminal** window, log in as the **root** user by typing this command.

   ```
   su root
   ```

3. When prompted, type the password in the **Terminal** window.
4. In the **Terminal** window, create a new user account called **pentaho**, along with the `pentaho` home directory, by typing this line.

   ```
   sudo useradd -s /bin/bash -m pentaho
   ```

   Note: `/bin/bash` indicates that the user account should be created using the Bash shell. In many Linux distributions, the default new user shell is `/bin/sh` or some equivalent, such as Dash, that might not use the `~/.bashrc` configuration file by default. If you don't have or want to use Bash, adjust the instructions throughout this section accordingly.
5. In the **Terminal** window, assign a password for the **pentaho** user by typing this line.

```
sudo passwd pentaho
```

6. Verify that you can log in using the newly-created **pentaho** user account.
    a. In the **Terminal** window, attempt to log in by typing this line.

    ```
    su pentaho -
    ```

    b. Type the password for the **pentaho** user account if you are prompted.
    c. Use the **Terminal** window to navigate to the pentaho directory to verify that it has been created. By default, it is in the `/home` directory.
    d. Close the **Terminal** window.

Create Directory Structure

Create Windows Directory Structure

1. Log into the machine on which you will run the DI Server.
2. Create this directory path.

    ```
    pentaho/server
    ```

3. Verify that you have the appropriate permissions to read, write, and execute commands in the directories you created.
    a. Open **Windows Explorer** and right-click the `pentaho` directory.
    b. Select the **Properties** option in the security tab to verify that you have read, write, and execute permissions.
    c. In **Windows Explorer** and navigate to the `data-integration-server` directory and right-click it.
    d. Select the **properties** option in the security tab to verify that you have read, write, and execute permissions.

Create Linux Directory Structure

1. Log into the machine on which you will run the DI Server. Make sure that you are logged in as the **pentaho** user.
2. Create this directory path from home directory (`pentaho`).

    ```
    pentaho/server
    ```

3. Verify that you have the appropriate permissions to read, write, and execute commands in the directories you created.
    a. In Linux check the permissions of the `pentaho`, `server`, and `data-integration-server` directories by opening a **Terminal** window, navigating to the `pentaho` directory, then typing this command.

    ```
    ls -ld ../pentaho ../pentaho/server
    ```

    b. Make sure that permissions for the directories allow you to read, write, and execute in those directories.

Install the DI Repository Host Database

The DI Repository houses data needed for Pentaho tools to provide scheduling and security functions, as well as metadata and models for reports that you create.

You can choose to host the DI Repository on the PostgreSQL, MySQL, or Oracle database. By default, Pentaho software is configured to use the PostgreSQL Database. If you already have a DI Repository database installed, you can skip this section.

NOTE:

If you are using Redshift, you will need to change the DI repository from PostgreSQL to MySQL, or another supported database, and replace our PostgreSQL 9.x driver with the PostgreSQL 8.4 driver that is recommended by Redshift. Follow the directions for initializing and configuring MySQL, then specify the data connection.

1. To download and install the DI Repository database, use the instructions in the documentation for the database of your choice. It does not matter where you install the database.
2. Verify that the DI Repository database is installed correctly. You can do this by connecting to your database and viewing the contents of any default databases that might have been created upon installation. Consult the user documentation for the database that you installed for further details. **Note:** If you are not familiar with SQL, consider using a visual database design tool to connect to the database and view its contents.
   - PGAdminIII is bundled with PostgreSQL.
   - MySQL Workbench can be used with MySQL. It is available as a separate download. Check the MySQL site for details on how to obtain this design tool.
   - Oracle SQL Developer can be used with Oracle. It is available as a separate download. Check the Oracle site for details on how to obtain this design tool.

Install Java JRE or JDK

Make sure that the version of the Java Runtime Environment (JRE) or the Java Development Kit (JDK) that Pentaho needs to run is installed on your system. You do not need to uninstall other versions of Java if you already have them running on your system. These instructions explain how to check for your default version of Java that is running on your computer and where to get the required version if you need one.

1. Check the supported technologies list to see which version of the JRE or JDK is needed for the software.
2. If you have not done so already, log into the computer on which you plan to install the software. Ensure that you have the appropriate permissions to install software.
3. Open a **Terminal** or a **Command Prompt** window. Enter this command.

   ```
   java -version
   ```

4. If the version of Java does not match the version needed to run Pentaho software, check your system to see if there are other versions of Java installed.
5. If the version of Java that you need to run Pentaho software is not on your system, download it from the Oracle site and install it.

Download and Unpack Installation Files

If you want to install specific Pentaho software, obtain the installation packages from the Pentaho Customer Support Portal. Consult your Welcome Kit if you need more information about how to access the portal.

1. Make sure the web application server on which you plan to deploy the DI Server has been stopped.
2. Download `pdi-ee-server-5.0.0-dist.zip` file. This file can be found on the Pentaho Customer Support Portal in the archive build folder.
3. Unpack the file by completing these steps.
   a. Use a zip tool to extract the `pdi-ee-server-5.0.0-dist.zip` file you just downloaded.
   b. Open a **Command Prompt** or **Terminal** window and navigate to the folder that contains the files you just extracted.
   c. Enter one of the following at the prompt.
   Windows:

   ```
   install.bat
   ```

   Linux:

   ```
   ./install.sh
   ```

   d. Read the license agreement that appears. Select **I accept the terms of this license agreement**, then click **Next**. **Note:** If you are unpacking the file in a non-graphical environment, open a **Terminal** or **Command Prompt** window and type `java -jar installer.jar -console` and follow the instructions presented in the window.
   e. Indicate where you want the file to be unpacked. It doesn't matter where because you will be manually placing the files in the appropriate directories later in these instructions.
   f. Click the **Next** button.
   g. The **Installation Progress** window appears. Progress bars indicate the status of the installation. When the installation progress is complete, click **Quit** to exit the Unpack Wizard.
4. Navigate to the directory where you installed the files. Verify that the directory structure has been correctly created by comparing the following directory structure with yours.
   - `pentaho/server/data-integration-server`
   - `pentaho/server/data-integration-server/data`
   - `pentaho/server/data-integration-server/licenses`
   - `pentaho/server/data-integration-server/pentaho-solutions`
   - `pentaho/server/data-integration-server/<your tomcat installation directory>`

   Note: If your web application server is not in the `pentaho/server/data-integration-server` directory, the `<your tomcat installation directory>` directory appears where you've chosen to install your web application server.

Set Environment Variables

Set the `PENTAHO_JAVA_HOME` variable to indicate the path to the Java JRE or JDK that Pentaho should use. Also, set the `PENTAHO_INSTALLED_LICENSE_PATH` variable so that when you start Pentaho, the licenses can install.

If you do not set these variables, Pentaho will not start correctly. To set environment variables, you should be logged into an account that has administrator-level privileges. For Linux systems, you must be logged into the **root** user account.

Set Windows PENTAHO_JAVA_HOME and PENTAHO_INSTALLED_LICENSE_PATH Variables

1. Open a **Command Prompt**.
2. At the prompt, set the path of the PENTAHO_JAVA_HOME variable to the path of your Java 7 installation. Here is an example.

```
SET PENTAHO_JAVA_HOME=C:\Program Files\Java\jre7
```

3. At the prompt, set the path of the `PENTAHO_INSTALLED_LICENSE_PATH` variable to point to `.installedLicenses.xml`. Here is an example.

```
SET PENTAHO_INSTALLED_LICENSE_PATH=C:\Users\joe\.pentaho\.installedLicenses.
xml
```

4. Log out, then log back in.
5. To verify that the variable has been properly set, open a **Command Prompt** window, then type this.

```
echo %PENTAHO_JAVA_HOME% %PENTAHO_INSTALLED_LICENSE_PATH%
```

6. The paths for the variables should appear. If they do not, try to set the environment variables again.

Set Linux PENTAHO_JAVA_HOME and PENTAHO_INSTALLED_LICENSE_PATH Variables

1. Open a **terminal** window and log in as **root**.
2. Open the `/etc/environment` file with a text editor. **Note:** The vi and gedit text editors are available on most Linux machines. For example, to open the `/etc/environment` file with gedit, type this.

```
gedit /etc/environment
```

3. Indicate where you installed Java in your `/etc/environment` file by typing this. **Note:** Substitute `/usr/lib/jvm/java-7-sun` with the location of the JRE or JDK you installed on your system.

```
export PENTAHO_JAVA_HOME=/usr/lib/jvm/java-7-sun
```

4. Indicate the location of the `.pentaho` directory by typing this.

```
export PENTAHO_INSTALLED_LICENSE_PATH=/<your home folder>/.pentaho/.
installedLicenses.xml
```

5. Save and close the file.
6. Log out, then log back in for the change to take effect.
7. Verify that the variables are properly set by opening a **Terminal** window and typing this.

```
env | grep PENTAHO_JAVA_HOME
```

8. The path to the variable should appear. If it does not, try setting the environment variable again.
9. Verify that the `PENTAHO_INSTALLED_LICENSE_PATH` variable is properly set by opening a **Terminal** window and typing this.

```
env | grep PENTAHO_INSTALLED_LICENSE_PATH
```

10. The path to the variable should appear. If it does not, try setting the environment variable again.

Advanced Topics

Complete the instructions in this section only if you have a headless node or if you plan to install on a Mac OS.

Prepare a Headless Linux or Solaris Server

There are two headless server scenarios that require special procedures on Linux and Solaris systems. One is for a system that has no video card; the other is for a system that has a video card, but does not have an X server installed. In some situations -- particularly if your server doesn't have a video card -- you will have to perform both procedures in order to properly generate reports with the BA Server.

Systems without video cards

The **java.awt.headless** option enables systems without video output and/or human input hardware to execute operations that require them. To set this application server option when the BA Server starts, you will need to modify the startup scripts for either the BA Server, or your Java application server. You do not need to do this now, but you will near the end of these instruction when you perform the Start BA Server step. For now, add the following item to the list of CATALINA_OPTS parameters: **-Djava.awt.headless=true**.

The entire line should look something like this:

```
export CATALINA_OPTS="-Djava.awt.headless=true -Xms4096m -Xmx6144m -
XX:MaxPermSize=256m -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.
gcInterval=3600000"
```

If you intend to create a BA Server service control script, you must add this parameter to that script's CATALINA_OPTS line.
Note: If you do not have an X server installed, you must also follow the below instructions.

Systems without X11

To generate charts, the Pentaho Reporting engine requires functionality found in X11. If you are unwilling or unable to install an X server, you can install the **xvfb** package instead. xvfb provides X11 framebuffer emulation, which performs all graphical operations in memory instead of sending them to the screen.

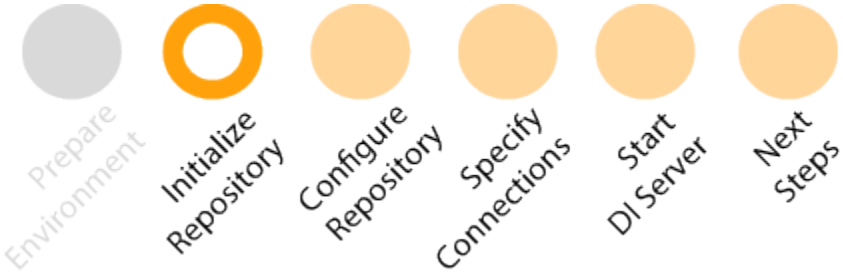
Use your operating system's package manager to properly install xvfb.

Adjust Amount of Memory Mac OS Allocates for PostgreSQL

If you plan to install the software on a Mac OS, and you choose to use PostgreSQL, you need to increase the amount of memory that the Mac OS allocates for PostgreSQL. You can skip these instructions if you plan to install the software on Windows or Linux.

PostgreSQL is the name of the default database that contains audit, schedule and other data that you create. PostgreSQL starts successfully only if your computer has allocated enough memory. Go to http://www.postgresql.org/docs/devel/static/kernel-resources.html and follow the instructions there on how to adjust the memory settings on your computer.

Initialize Repository



Before you prepare the DI Repository complete the tasks in Prepare Environment.

Pentaho stores content about reports that you create, examples we provide, report scheduling data, and audit data in the DI Repository. The DI Repository resides on the database that you installed during the Prepare Environment step. The DI Repository consists of three repositories: *Jackrabbit*, *Quartz*, and *Hibernate*.

- *Jackrabbit* contains the solution respository, examples, security data, and content data from reports that you use Pentaho software to create.
- *Quartz* holds data that is related to scheduling reports and jobs.
- *Hibernate* holds data that is related to audit logging.

This step only consists of one task: Initialize the database. In this task you run DDLs that contain SQL commands that create the Jackrabbit, Quartz, and Hibernate databases, as well as the Operations Mart schema.

Initialize PostgreSQL DI Repository Database

To initialize PostgreSQL so that it serves as the DI Repository, run SQL scripts to create the Hibernate, Quartz and Jackrabbit (also known as the JCR) databases.

Note: Your PostgreSQL configuration must support logins from all users. This is not always the default configuration, so you may have to edit your `pg_hba.conf` file to support this option. If you do need to make changes to pg_hba.conf, you must restart the PostgreSQL server before proceeding.

1. To make the databases that you create more secure, Pentaho recommends that you change the default passwords in the SQL script files to ones that you specify. If you are evaluating Pentaho, you might want to skip this step. If you do decide to make the databases more secure, use a text editor to change the passwords in these files:
   - `pentaho/server/data-integration-server/data/postgresql/create_jcr_postgresql.sql`
   - `pentaho/server/data-integration-server/data/postgresql/create_quartz_postgresql.sql`
   - `pentaho/server/data-integration-server/data/postgresql/create_repository_postgresql.sql`

   Here is an example of a password change made in the `create_jcr_postgresql.sql` file.

```
CREATE USER jcr_user PASSWORD 'myNewPassword'
```

5. **Windows:** The commands you use to run the SQL scripts depends on your operating system. For windows, do this.
    a. Open a **SQL Shell** window. The **SQL Shell** window is installed with PostgreSQL.
    b. When prompted for the server enter the name of the server if you are not using the default (localhost). If you are using the default, do not type anything and press Enter.
    c. When prompted for the database enter the name of the database if you are not using the default (postgres) If you are using the default, do not type anything and press Enter.
    d. When prompted for the port enter the name of the port if you are not using the default (5432). If you are using the default port, do not type anything and press Enter.
    e. When prompted for the username, accept the default, then press Enter.
    f. When prompted for the password, enter the password that you indicated when you installed PostgreSQL.
    g. Run the script to create the Jackrabbit database by typing this.

    ```
    \i /pentaho/server/data-integration-server/data/postgresql/create_jcr_
    postgresql.sql
    ```

    Note: If necessary, change the `\pentaho\server\data-integration-server` to the place where you unpacked your pentaho files.
    h. Run the script to create the hibernate database by typing this.

    ```
    \i /pentaho/server/data-integration-server/data/postgresql/create_
    repository_postgresql.sql
    ```

    i. Run the script to create the Quartz database by typing this.

    ```
    \i /pentaho/server/data-integration-server/data/postgresql/create_
    quartz_postgresql.sql
    ```

    j. To switch to the Hibernate database, type this.

    ```
    \c postgres
    ```

    k. Run the script to create the Operations Mart database by typing this.

    ```
    \i /pentaho/server/data-integration-server/data/postgresql/pentaho_
    mart_postgresql.sql
    ```

    l. Exit from the window by pressing the **CTRL** + **C** keys.
6. **Linux:** To run the SQL scripts on a Linux system, do this.
    a. Open a **Terminal** window. You should be logged in as the **pentaho** user.
    b. Sign into PostgreSQL by typing `psql -U postgres -h localhost` at the prompt.
    c. Run the script to create the Jackrabbit database by typing this.

```
\i ~/pentaho/server/data-integration-server/data/
postgresql/create_jcr_postgresql.sql
```

Note: If necessary, change the `~/pentaho/server/data-integration-server` directory path to the place where you unpacked your pentaho files.

d. Run the script to create the hibernate database by typing this.

```
\i ~/pentaho/server/data-integration-server/data/postgresql/create_
repository_postgresql.sql
```

Note: If necessary, change the `~/pentaho/server/data-integration-server` to the place where you unpacked your pentaho files.

e. Run the script to create the Quartz database by typing this.

```
\i ~/pentaho/server/data-integration-server/data/postgresql/create_
quartz_postgresql.sql
```

Note: If necessary, change the `~/pentaho/server/data-integration-server` directory path to the place where you unpacked your pentaho files.

f. To switch to the Hibernate database, type this.

```
\c postgres
```

g. Run the script to create the Operations Mart database by typing this.

```
\i ~/pentaho/server/data-integration-server/data/postgresql/pentaho_
mart_postgresql.sql
```

h. Exit from the window by pressing the **CTRL** + **C** keys.

7. To verify that databases and user roles have been created, do this.
   a. Open the pgAdminIII tool. pgAdminIII is bundled with both the Windows and Linux versions of PostgreSQL.
   b. To view the contents of PostgreSQL, click the PostgreSQL folder in the **Object Browser**, then enter the password when prompted.
   c. In the **Object Browser**, click the **Databases** folder. The Jackrabbit, Postgres, Hibernate and Quartz databases appear.
   d. In the **Object Browser**, click the **Login Roles** folder. The jcr_user, pentaho_user, hibuser, and postgres user accounts appear.
   e. If the databases and login roles do not appear, go to the beginning of these instructions and try running the scripts again.
   f. Select **File > Exit** to exit from pgAdminIII.

Initialize MySQL DI Repository Database

To initialize MySQL so that it serves as the DI Repository, run SQL scripts to create the Hibernate, Quartz and Jackrabbit (also known as the JCR) databases.

NOTE:

Use the ASCII character set when you run these scripts. Do not use UTF-8 because there are text string length limitations that might cause the scripts to fail.

If you are using Redshift, after you finish this section, move on to the configuring MySQL tasks.

These sections take you through the steps to initialize the MySQL DI repository database.

Change Default Passwords

To make the databases that you create more secure, Pentaho recommends that you change the default passwords in the SQL script files to ones that you specify. If you are evaluating Pentaho, you might want to skip this step. If you do decide to make the databases more secure, use a text editor to change the passwords in these files:

- `pentaho/server/data-integration-server/data/mysql5/create_jcr_mysql.sql`
- `pentaho/server/data-integration-server/data/mysql5/create_quartz_mysql.sql`
- `pentaho/server/data-integration-server/data/mysql5/create_repository_mysql.sql`
- `pentaho/server/data-integration-server/data/mysql5/pentaho_mart_mysql.sql`

Run SQL Scripts

Once you change the passwords, you will need to run these SQL scripts. The process for running SQL scripts against MySQL is the same for Windows or Linux machines.

Run these scripts from the **MySQL Command Prompt** window or from **MySQL Workbench**.

Action	SQL Script
Create Quartz	`> source <your filepath>\create_quartz_mysql.sql`
Create Hibernate repository	`> source <your filepath>\create_repository_mysql.sql`
Create Jackrabbit	`> source <your filepath>\create_jcr_mysql.sql`
Create Pentaho Operations mart	`> source <your filepath>\pentaho_mart_mysql.sql`

Verify MySQL Initialization

After you run the scripts, this list will help you verify that databases and user roles have been created.

1. Open the **MySQL Workbench** tool. **MySQL Workbench** is freely available at the MySQL development site.
2. Make sure that the Quartz, Jackrabbit (JCR), Hibernate, and Pentaho Operations mart databases are present.
3. Exit from the **MySQL Workbench**.

Initialize Oracle DI Repository Database

To initialize Oracle so it serves as the DI Repository, run SQL scripts to create the Hibernate, Quartz and Jackrabbit (also known as the JCR) databases.

1. To make the databases that you create more secure, Pentaho recommends that you change the default passwords in the SQL script files to ones that you specify. If you are evaluating Pentaho, you might want to skip this step. If you do decide to make the databases more secure, use a text editor to change the passwords in these files. (Also, for each file, edit the **datafile** path with the path to your Oracle installation.)
   - `pentaho/server/data-integration-server/data/oracle10g/create_jcr_ora.sql`
   - `pentaho/server/data-integration-server/data/oracle10g/create_quartz_ora.sql`
   - `pentaho/server/data-integration-server/data/oracle10g/create_repository_ora.sql`

   Here is an example of a password change made in the `create_jcr_ora.sql` file.

   ```
   --conn admin/myNewPassword@pentaho
   create user di_jcr_user identified by "myNewPassword" default tablespace di_
   pentaho_tablespace quota unlimited on pentaho_tablespace temporary
   tablespace temp quota 5M on system;
   ```

   Note: The user name above (`jcr_user`) might be different in your `create_jcr_ora.sql` file.

5. Although there are several different methods for running SQL scripts, these instructions explain how to run SQL*Plus from a Terminal or Command Prompt window. These instructions are the same for both Windows and Linux. If you prefer to run SQL scripts using another method, modify instructions accordingly.
   a. Open a **Terminal** or **Command Prompt** window, start the **SQL*Plus** and log in.
   b. Run the script to create the Jackrabbit database by typing `START create_jcr_ora`. If necessary, append the path to the `create_jcr_ora.sql` path in the command.
   c. Run the script to create the repository database by typing `START create_repository_ora`. If necessary, append the path to the `create_repository_ora.sql` path in the command.
   d. Run the script to create the Quartz database and users by typing `START create_quartz_ora`. If necessary, append the path to the `create_quartz_ora.sql` path in the command.
   e. Run the script to create the Operations Mart database and users by typing `START pentaho_mart_ora`. If necessary, append the path to the `pentaho_mart_ora.sql` path in the command.
6. To verify user roles have been created, do this.
   a. In the **Terminal** or **Command Prompt** window that is running **SQL*Plus**, type SELECT USERNAME FROM DBA_USERS to see the users that have been created.
   b. If the usernames do not appear, go to the beginning of these instructions and try running the scripts again.
   c. Exit from **SQL*Plus**.

Configure Repository

Before you configure the DI Repository, complete the tasks in Initialize Repository.

Tasks performed during this step include configuring Audit, Quartz, and Hibernate properties. Tasks are grouped by the DI Repository database you have.

Configure PostgreSQL DI Repository Database

These instructions explain how to configure Quartz, Hibernate, Jackrabbit, and Pentaho Security for use with the PostgreSQL database. By default, the files edited in this section are configured for a PostgreSQL database that runs on port 5432. The default password is also in these files. If you have a different port, different password, or if had the system configured using a different database and now you want to change it back to PostgreSQL, complete all of the instructions in these steps.

Configure Quartz on PostgreSQL DI Repository Database

When you use Pentaho to schedule an event, such as a report to be run every Sunday at 1:00 a.m. EST, event information is stored in the Quartz JobStore. During the installation process, you must indicate where the JobStore is located. To do this, modify the `quartz.properties` file.

1. Open the `pentaho/server/data-integration-server/pentaho-solutions/system/quartz/quartz.properties` file in the text editor of your choice.
2. Make sure that in the `#_replace_jobstore_properties` section of the file, the `org.quartz.jobStore.driverDelegateClass` is set to `org.quartz.impl.jdbcjobstore.PostgreSQLDelegate`.
3. In the `# Configure Datasources` section of the file, set the `org.quartz.dataSource.myDS.jndiURL` equal to `Quartz`.

   ```
   org.quartz.dataSource.myDS.jndiURL = Quartz
   ```

4. Save the file and close the text editor.

Configure Hibernate Settings for PostgreSQL DI Repository Database

Modify the hibernate settings file to specify where Pentaho will find the DI Repository's hibernate configuration file. The hibernate configuration file specifies driver and connection information, as well as dialects and how to handle connection closes and timeouts.

1. Open `pentaho/server/data-integration-server/pentaho-solutions/system/hibernate/hibernate-settings.xml` in a text editor.
2. Verify that the location of the PostgreSQL hibernate configuration file appears. Make changes if necessary.

   ```
   <config-file>system/hibernate/postgresql.hibernate.cfg.xml</config-file>
   ```

3. Save the file if you had to make changes, then close it. Otherwise, just close it.
4. Open `pentaho/server/data-integration-server/pentaho-solutions/system/hibernate/postgresql.hibernate.cfg.xml` in a text editor.
5. Make sure that the password and port number match the ones you specified in your configuration. Make changes as necessary, then save and close the file.

Modify Jackrabbit DI Repository Information for PostgreSQL

Indicate which database houses the DI Repository as well as the port, url, username, and password. All of the information needed to configure the repository for the PostgreSQL, MySQL, and Oracle DI Repository databases appear. By default, the PostgreSQL sections are not commented out, but the MySQL and Oracle sections are. To modify this file so that it works for your DI Repository, you will need to make sure that the sections that refer to your DI Repository database are not commented out, and the sections refer to other DI Repository databases are commented out.

When code is commented out, it appears between the `<!--` and `-->` tags. The information in between the tags is commented out, and is therefore not executed by the software. In this example, the code `<!` `<FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">` is commented out.

```
<!--
<FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
-->
```

To make sure that the Jackrabbit repository is set so that PostgreSQL is the default database, do this.

1. Use a text editor to open the `pentaho/server/data-integration-server/pentaho-solutions/system/jackrabbit/repository.xml` file.
2. In the Repository part of the code, make sure that the PostgreSQL lines of code are not commented out, but the Oracle and MySQL lines are. The code should look like this.

   ```
   <!--
     <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
       <param name="driver" value="com.mysql.jdbc.Driver"/>
   ```

```
    ...
    <param name="schemaObjectPrefix" value="fs_repos_"/>
  </FileSystem>
  <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    ...
    <param name="schemaObjectPrefix" value="fs_repos_"/>
    <param name="tablespace" value="jackrabbit"/>
  </FileSystem>
-->
  <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
    <param name="user" value="jcr_user"/>
    <param name="password" value="password"/>
    <param name="schema" value="postgresql"/>
    <param name="schemaObjectPrefix" value="fs_repos_"/>
  </FileSystem>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly.

3. In the DataStore section of the code, verify that the PostgreSQL lines of code are not commented out, but the Oracle and MySQL lines are. The code should look like this.

```
<!--
  <DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
   ...
    <param name="schemaObjectPrefix" value="ds_repos_"/>
  </DataStore>
  <DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
   ...
    <param name="schemaObjectPrefix" value="ds_repos_"/>
  </DataStore>
  -->
  <DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="user" value="jcr_user"/>
```

```
    <param name="password" value="password"/>
    <param name="databaseType" value="postgresql"/>
    <param name="minRecordLength" value="1024"/>
    <param name="maxConnections" value="3"/>
    <param name="copyWhenReading" value="true"/>
    <param name="tablePrefix" value=""/>
    <param name="schemaObjectPrefix" value="ds_repos_"/>
  </DataStore>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly.

4. In the Workspaces section of the code, make sure that the PostgreSQL lines of code are not commented out, but the Oracle and MySQL lines are. This code should look like this.

```
<!--
    <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="com.mysql.jdbc.Driver"/>
     ...
      <param name="schemaObjectPrefix" value="fs_ws_"/>
    </FileSystem>
    <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
      ...
      <param name="schemaObjectPrefix" value="fs_ws_"/>
      <param name="tablespace" value="jcr_user"/>
    </FileSystem>
-->
    <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="org.postgresql.Driver"/>
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schema" value="postgresql"/>
      <param name="schemaObjectPrefix" value="fs_ws_"/>
    </FileSystem>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly.

5. In the Persistence Manager section of the code, verify that the PostgreSQL lines of code are not commented out, but the Oracle and MySQL lines are. The code should look like this.

```
<!--
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.MySqlPersistenceManager">
      <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
      ...
      <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
    </PersistenceManager>
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.OraclePersistenceManager">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
      ...
      <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
      <param name="tablespace" value="jackrabbit"/>
    </PersistenceManager>
   -->
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.PostgreSQLPersistenceManager">
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      <param name="driver" value="org.postgresql.Driver"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schema" value="postgresql"/>
      <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
    </PersistenceManager>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly.

6. In the Versioning section of the code, verify that the PostgreSQL lines of code are not commented out, but the MySQL and Oracle lines are. The code should look like this.

```
<!--
    <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="com.mysql.jdbc.Driver"/>
     ...
      <param name="schemaObjectPrefix" value="fs_ver_"/>
    </FileSystem>
    <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
     ...
      <param name="schemaObjectPrefix" value="fs_ver_"/>
      <param name="tablespace" value="jackrabbit"/>
```

```
    </FileSystem>
    -->
    <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="org.postgresql.Driver"/>
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schema" value="postgresql"/>
      <param name="schemaObjectPrefix" value="fs_ver_"/>
    </FileSystem>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly.

7. In the Persistence Manager section of the code that is near the end of the file, verify that PostgreSQL lines of code are not commented out, but the MySQL and Oracle lines are. The codes should look like this.

```
<!--
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.MySqlPersistenceManager">
      <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
     ...
      <param name="schemaObjectPrefix" value="pm_ver_"/>
    </PersistenceManager>

    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.OraclePersistenceManager">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
     ...
      <param name="schemaObjectPrefix" value="pm_ver_"/>
      <param name="tablespace" value="jackrabbit"/>
    </PersistenceManager>
    -->
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.PostgreSQLPersistenceManager">
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      <param name="driver" value="org.postgresql.Driver"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schema" value="postgresql"/>
```

```
      <param name="schemaObjectPrefix" value="pm_ver_"/>
    </PersistenceManager>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly.

8. Close and save the file.

Configure MySQL DI Repository Database

These instructions explain how to configure Quartz, Hibernate, Jackrabbit, and Pentaho Security for a MySQL database.

NOTE:

By default, the examples in this section are for a MySQL database that runs on port 3306. The default password is also in these examples. If you have a different port, different password complete all of the instructions in these steps.

Set Up Quartz on MySQL DI Repository Database

Event information, such as scheduled reports, is stored in the Quartz JobStore. During the installation process, you must indicate where the **JobStore** is located. You do this by modifying the `quartz.properties` file.

1. Open the `pentaho/server/biserver-ee/pentaho-solutions/system/quartz/quartz.properties` file in any text editor.
2. Locate the `#_replace_jobstore_properties` section and set the `org.quartz.jobStore.driverDelegateClass` as shown here.

```
org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.StdJDBCDelegate
```

3. Save the file and close the text editor.

Set Hibernate Settings for MySQL

Modify the hibernate settings file to specify where Pentaho should find the DI Repository's hibernate-settings config file. The hibernate-settings config file specifies driver and connection information, as well as dialects and how to handle connection closes and timeouts. The files in this section are located in the `pentaho/server/data-integration/pentaho-solutions/system/hibernate` directory.

1. Open the `hibernate-settings.xml` file in a text editor. Find the `<config-file>` tags and change `postgresql.hibernate.cfg.xml` to `mysql5.hibernate.cfg.xml` as shown.

From:

```
<config-file>system/hibernate/postgresql.hibernate.cfg.xml</config-file>
```

To:

```
<config-file>system/hibernate/mysql5.hibernate.cfg.xml</config-file>
```

2. Save and close the file.
3. Open the `mysql5.hibernate.cfg.xml` file in a text editor.
4. Make sure that the password and port number match the ones you specified in your configuration. Make changes if necessary, then save and close the file.

Replace Default Version of Audit Log File with MySQL Version

Since you are using MySQL to host the DI Repository, you need to replace the `audit_sql.xml` file with one that is configured for MySQL.

1. Locate the `pentaho-solutions/system/dialects/mysql5/audit_sql.xml` file.
2. Copy it into the `pentaho-solutions/system` directory.

Modify Jackrabbit Repository Information for MySQL

There are parts of code that you will need to alter in order to change the default jackrabbit repository to MySQL.

1. Navigate to the `pentaho/server/data-integration-server/pentaho-solutions/system/jackrabbit` and open the `repository.xml` file with any text editor.
2. Following the table below, locate and change the code so that the MySQL lines are **not** commented out, but the PostgreSQL and Oracle lines **are** commented out.

NOTE:

If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters in each section accordingly.

Item:	Code Section:

Repository	`<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">`
	`  <param name="driver" value="com.mysql.jdbc.Driver"/>`
	`  <param name="url" value="jdbc:mysql://localhost:3306/jackrabbit"/>`
	`…`
	`</FileSystem>`

Item:	Code Section:
DataStore	<DataStore class="org.apache.jackrabbit.core.data.db. DbDataStore"> <param name="url" value="jdbc:mysql://localhost:3306/ jackrabbit"/> … </DataStore>
Workspaces	<FileSystem class="org.apache.jackrabbit.core.fs.db. DbFileSystem"> <param name="driver" value="com.mysql.jdbc. Driver"/> <param name="url" value="jdbc:mysql://localhost:3306/jackrabbit"/> … </FileSystem>
PersistenceManager (1st part)	<PersistenceManager class="org.apache.jackrabbit.core. persistence.bundle.MySqlPersistenceManager"> <param name="url" value="jdbc:mysql://localhost:3306/jackrabbit"/> … </PersistenceManager>

Item:	Code Section:

Versioning	`<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">`
	`  <param name="driver" value="com.mysql.jdbc.Driver"/>`
	`  <param name="url" value="jdbc:mysql://localhost:3306/jackrabbit"/>`
	`…`
	`</FileSystem>`

PersistenceManager
(2nd part)
	`<PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.MySqlPersistenceManager">`
	`  <param name="url" value="jdbc:mysql://localhost:3306/jackrabbit"/>`
	`  …`
	`</PersistenceManager>`

Prepare Oracle DI Repository Database

These instructions explain how to configure Quartz, Hibernate, Jackrabbit, and Pentaho Security. By default, the examples in this section are for a Oracle database that runs on port 1521. The default password is also in these examples. If you have a different port, different password complete all of the instructions in these steps.

Configure Quartz on Oracle DI Repository Database

When you use Pentaho to schedule an event, such as a report to be run every Sunday at 1:00 a.m. EST, event information is stored in the Quartz JobStore. During the installation process, you must indicate where the JobStore is located. To do this, modify the `quartz.properties` file.

1. Open the `pentaho/server/data-integration-server/pentaho-solutions/system/quartz/quartz.properties` file in the text editor of your choice.
2. In the `#_replace_jobstore_properties` section of the file, set the `org.quartz.jobStore.driverDelegateClass` equal to `org.quartz.impl.jdbcjobstore.oracle.OracleDelegate`.

   ```
   org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.oracle.OracleDelegate
   ```

3. In the `# Configure Datasources` section of the file, set the `org.quartz.dataSource.myDS.jndiURL` equal to `Quartz`.

```
org.quartz.dataSource.myDS.jndiURL = Quartz
```

4. Save the file and close the text editor.

Configure Hibernate Settings for Oracle

Modify the hibernate settings file to specify where Pentaho will find the DI Repository's hibernate configuration file. The hibernate configuration file specifies driver and connection information, as well as dialects and how to handle connection closes and timeouts.

1. Open `pentaho/server/data-integration-server/pentaho-solutions/system/hibernate/hibernate-settings.xml` in a text editor. By default, the location of the PostgreSQL database configuration file is specified.

```
<config-file>system/hibernate/postgresql.hibernate.cfg.xml</config-file>
```

2. Change the default to this to point to the Oracle configuration file.

```
<config-file>system/hibernate/oracle10g.hibernate.cfg.xml</config-file>
```

3. Save and close the file.
4. Open `pentaho/server/data-integration-server/system/hibernate/oracle10g.hibernate.cfg.xml` in a text editor.
5. Make sure that the password and port number match the ones you specified in your configuration. Make changes as necessary, then save and close the file.

Replace Default Version of Audit Log File with Oracle Version

The default audit_sql.xml file that is in the pentaho-solutions/system directory is configured for the PostgreSQL database. Since you are using Oracle to host the DI Repository, you need to replace the `audit_sql.xml` file with one that is configured for Oracle. To do this, copy the `pentaho-solutions/system/dialects/oracle10g/audit_sql.xml` file to the `pentaho-solutions/system` directory.

Modify Jackrabbit Repository Information for Oracle

You must indicate which database is used as the DI Repository as well as the port, url, username, and password. All of the information needed to configure the repository for the PostgreSQL, MySQL, and Oracle DI Repository databases appear. By default, the PostgreSQL sections are not commented out, but the MySQL and Oracle sections are. To modify this file so that it works for your DI Repository, you will need to make sure that the sections that refer to your DI Repository Database are not commented out, and the sections refer to other DI Repository databases are commented out.

When code is commented out, it appears between the `<!--` and `-->` tags. The information in between the tags is commented out, and is therefore not executed by the software. In this example, the code `<!` `<FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">` is commented out.

```
<!--
<FileSystem class=”org.apache.jackrabbit.core.fs.db.OracleFileSystem”>
-->
```

To modify the Jackrabbit repository so that Oracle is the default database, do this.

1. Use a text editor to open the `pentaho/server/data-integration-server/pentaho-solutions/system/jackrabbit/repository.xml` file.
2. In the Repository part of the code, change the code so that the Oracle lines of code are not commented out, but the PostgreSQL and MySQL lines are, like this.

```
<!--
<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="com.mysql.jdbc.Driver"/>
    <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
   ...
    <param name="schemaObjectPrefix" value="fs_repos_"/>
  </FileSystem>
-->
<FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <param name="user" value="jcr_user"/>
    <param name="password" value="password"/>
    <param name="schemaObjectPrefix" value="fs_repos_"/>
    <param name="tablespace" value="jackrabbit"/>
  </FileSystem>
<!--
  <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
    ...
    <param name="schemaObjectPrefix" value="fs_repos_"/>
  </FileSystem>
-->
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly. Also note that the `<param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>` parameter indicates that the database is named `orcl`. Modify the parameter if your database has a different name.

3. Change <param name="tablespace" value="jackrabbit"/> to <param name="tablespace" value="di_pentaho_tablespace">. Then change <param name="user" value="jcr_user"/> to <param name="user" value="di_jcr_user"/>.

4. In the DataStore section of the code, change the code so that the Oracle lines of code are not commented out, but the PostgreSQL and MySQL lines are, like this.

```
<!--
<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
    ...
    <param name="schemaObjectPrefix" value="ds_repos_"/>
  </DataStore>
  -->
 <DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <param name="driver" value="oracle.jdbc.driver.OracleDriver"/>
    <param name="user" value="jcr_user"/>
    <param name="password" value="password"/>
    <param name="databaseType" value="oracle"/>
    <param name="minRecordLength" value="1024"/>
    <param name="maxConnections" value="3"/>
    <param name="copyWhenReading" value="true"/>
    <param name="tablePrefix" value=""/>
    <param name="schemaObjectPrefix" value="ds_repos_"/>
  </DataStore>
 <!--
 <DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
    ...
    <param name="schemaObjectPrefix" value="ds_repos_"/>
  </DataStore>
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly. Also note that the `<param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>` parameter indicates that the database is named `orcl`. Modify the parameter if your database has a different name.

5. Change <param name="user" value="jcr_user"/> to <param name="user" value="di_jcr_user"/>.
6. In the Workspaces section of the code, change the code so that the Oracle lines of code are not commented out, but the PostgreSQL and MySQL lines are, like this.

```
<!--
<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="com.mysql.jdbc.Driver"/>
      <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
     ...
```

```
      <param name="schemaObjectPrefix" value="fs_ws_"/>
    </FileSystem>
-->
    <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schemaObjectPrefix" value="fs_ws_"/>
      <param name="tablespace" value="jcr_user"/>
    </FileSystem>
<!--
    <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="org.postgresql.Driver"/>
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      ...
      <param name="schemaObjectPrefix" value="fs_ws_"/>
    </FileSystem>
-->
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly. Also note that the `<param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>` parameter indicates that the database is named `orcl`. Modify the parameter if your database has a different name.

7. Change <param name="tablespace" value="jcr_user"/> to <param name="tablespace" value="di_pentaho_tablespace">. Then change <param name="user" value="jcr_user"/> to <param name="user" value="di_jcr_user"/>.
8. In the Persistence Manager section of the code, change the code so that the Oracle lines of code are not commented out, but the PostgreSQL and MySQL lines are, like this.

```
<!--
<PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.
MySqlPersistenceManager">
      <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
     ...
      <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
    </PersistenceManager>
 -->
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.OraclePersistenceManager">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
      <param name="driver" value="oracle.jdbc.driver.OracleDriver"/>
```

```
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schema" value="oracle"/>
      <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
      <param name="tablespace" value="jackrabbit"/>
    </PersistenceManager>
 <!--
   <PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.
PostgreSQLPersistenceManager">
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      ...
      <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
    </PersistenceManager>
 -->
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly. Also note that the `<param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>` parameter indicates that the database is named `orcl`. Modify the parameter if your database has a different name.

9. Change <param name="tablespace" value="jackrabbit"/> to <param name="tablespace" value="di_pentaho_tablespace"> Then change <param name="user" value="jcr_user"/> to <param name="user" value="di_jcr_user"/>.
10. In the Versioning section of the code, change the code so that the Oracle lines of code are not commented out, but the PostgreSQL and MySQL lines are, like his.

```
<!--
<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="com.mysql.jdbc.Driver"/>
      <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
      ...
      <param name="schemaObjectPrefix" value="fs_ver_"/>
    </FileSystem>
-->
    <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schemaObjectPrefix" value="fs_ver_"/>
      <param name="tablespace" value="jackrabbit"/>
    </FileSystem>
 <!--
```

```
   <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
      <param name="driver" value="org.postgresql.Driver"/>
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
      ...
      <param name="schemaObjectPrefix" value="fs_ver_"/>
    </FileSystem>
-->
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly. Also note that the `<param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>` parameter indicates that the database is named `orcl`. Modify the parameter if your database has a different name.

11. Change <param name="tablespace" value="jackrabbit"/> to <param name="tablespace" value="di_pentaho_tablespace">. Then change <param name="user" value="jcr_user"/> to <param name="user" value="di_jcr_user"/>.
12. In the Persistence Manager section of the code that is near the end of the file, change the code so that the Oracle lines of code are not commented out, but the PostgreSQL and MySQL lines are, like this.

```
<!--
<PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.
MySqlPersistenceManager">
      <param name="url" value="jdbc:mysql://localhost:3306/di_jackrabbit"/>
      ...
      <param name="schemaObjectPrefix" value="pm_ver_"/>
    </PersistenceManager>
-->
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.
bundle.OraclePersistenceManager">
      <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
      <param name="driver" value="oracle.jdbc.driver.OracleDriver"/>
      <param name="user" value="jcr_user"/>
      <param name="password" value="password"/>
      <param name="schema" value="oracle"/>
      <param name="schemaObjectPrefix" value="pm_ver_"/>
      <param name="tablespace" value="jackrabbit"/>
    </PersistenceManager>
 <!--
   <PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.
PostgreSQLPersistenceManager">
      <param name="url" value="jdbc:postgresql://localhost:5432/di_
jackrabbit"/>
```

```
      ...
       <param name="schemaObjectPrefix" value="pm_ver_"/>
     </PersistenceManager>
-->
```

Note: If you changed your password when you initialized the database during the Prepare Environment step, or if your database is on a different port, edit the url and password parameters accordingly. Also note that the `<param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>` parameter indicates that the database is named `orcl`. Modify the parameter if your database has a different name.

13. Change <param name="tablespace" value="jackrabbit"/> to <param name="tablespace" value="di_pentaho_tablespace">. Then change <param name="user" value="jcr_user"/> to <param name="user" value="di_jcr_user"/>.

Specify Connections

After your repository has been configured, you must configure the web application servers to connect to the DI Repository. In this step, JDBC and JNDI connections are made to the Hibernate, Jackrabbit, and Quartz databases. These databases were installed on your DI Repository database during the Initialize Repository and Configure Repository sections of these instructions.

By default, the DI Server software is configured to be deployed and run on the Tomcat server. As such, connections have already been specified and only the Tomcat context.xml file must be modified. Complete the following tasks.

Perform Tomcat-Specific Connection Tasks

If you plan to run the DI Server on Tomcat, you must modify JDBC Connection information.

Copy Solution Database JDBC Drivers

For the DI Server to connect to the DI Repository database of your choice, add the DI Repository's database JDBC driver library to the appropriate place in the web application server on which the DI Server will be deployed. The default web application server for the archive installation process is Tomcat.

NOTE:

If you are using Redshift, you will need to replace our PostgreSQL 9.x driver with the PostgreSQL 8.4 driver that is recommended by Redshift.

1. Download a JDBC driver JAR from your database vendor or a third-party driver developer.
2. Copy the JDBC driver JAR you just downloaded to the `/tomcat/lib/` directory.
3. Follow the instructions on the JDBC utility tool to install the JDBC driver.
4. Verify that the driver is installed.

Modify JDBC Connection Information in the Tomcat context.xml File

Database connection and network information, such as the username, password, driver class information, IP address or domain name, and port numbers for your BA Repository database are stored in the `context.xml` file. Modify this file to reflect the database connection and network information to reflect your operating

environment. You also modify the values for the `validationQuery` parameters in this file if you have choosen to use an BA Repository database other than PostgreSQL.

1. Consult your database documentation to determine the JDBC class name and connection string for your BA Repository database.
2. Go to the `biserver-ee\tomcat\webapps\pentaho\META-INF` directory and open the `context.xml` file with any file editor.
3. Follow the directions below for your database. Be sure to adjust the port numbers and passwords to reflect your environment, if necessary.

   For PostgreSQL:

   Comment out the resource references that refer to databases other than PostgreSQL, such as MySQL and Oracle. Then, add the following code to the file if it does not already exist.

```
<Resource validationQuery="select 1" url="jdbc:postgresql://localhost:5432/
hibernate" driverClassName="org.postgresql.Driver" password="password"
username="hibuser" maxWait="10000" maxIdle="5" maxActive="20" factory="org.
apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.DataSource"
auth="Container" name="jdbc/Hibernate"/>
<Resource validationQuery="select 1" url="jdbc:postgresql://localhost:5432/
hibernate" driverClassName="org.postgresql.Driver" password="password"
username="hibuser" maxWait="10000" maxIdle="5" maxActive="20" factory="org.
apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.DataSource"
auth="Container" name="jdbc/Audit"/>
<Resource validationQuery="select 1" url="jdbc:postgresql://localhost:5432/
quartz" driverClassName="org.postgresql.Driver" password="password"
username="pentaho_user" maxWait="10000" maxIdle="5" maxActive="20"
factory="org.apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.
DataSource" auth="Container" name="jdbc/Quartz"/>
<Resource validationQuery="select 1" url="jdbc:postgresql://localhost:5432/
hibernate" driverClassName="org.postgresql.Driver" password="password"
username="hibuser" maxWait="10000" maxIdle="5" maxActive="20" factory="org.
apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.DataSource"
auth="Container" name="jdbc/pentaho_operations_mart"/>
<Resource validationQuery="select 1" url="jdbc:postgresql://localhost:5432/
hibernate" driverClassName="org.postgresql.Driver" password="password"
username="hibuser" maxWait="10000" maxIdle="5" maxActive="20" factory="org.
apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.DataSource"
auth="Container" name="jdbc/PDI_Operations_Mart"/>
```

   For MySQL:

   Comment out the resource references that refer to databases other than MySQL, such as PostgreSQL and Oracle. Then, add the following code to the file if it does not already exist.

```
<Resource validationQuery="select 1" url="jdbc:mysql://localhost:3306/
hibernate" driverClassName="com.mysql.jdbc.Driver" password="password"
username="hibuser" maxWait="10000" maxIdle="5" maxActive="20" factory="org.
apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.DataSource"
auth="Container" name="jdbc/Hibernate"/>
<Resource validationQuery="select 1" url="jdbc:mysql://localhost:3306/
hibernate" driverClassName="com.mysql.jdbc.Driver" password="password"
username="hibuser" maxWait="10000" maxIdle="5" maxActive="20" factory="org.
apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.DataSource"
auth="Container" name="jdbc/Audit"/>
<Resource validationQuery="select 1" url="jdbc:mysql://localhost:3306/
quartz" driverClassName="com.mysql.jdbc.Driver" password="password"
username="pentaho_user" maxWait="10000" maxIdle="5" maxActive="20"
factory="org.apache.commons.dbcp.BasicDataSourceFactory" type="javax.sql.
DataSource" auth="Container" name="jdbc/Quartz"/>
<Resource validationQuery="select 1" url="jdbc:mysql://localhost:3306/
pentaho_operations_mart" driverClassName="com.mysql.jdbc.Driver"
password="password" username="hibuser" maxWait="10000" maxIdle="5"
maxActive="20" factory="org.apache.commons.dbcp.BasicDataSourceFactory"
type="javax.sql.DataSource" auth="Container" name="jdbc/pentaho_operations_
mart"/>
<Resource validationQuery="select 1" url="jdbc:mysql://localhost:3306/
pentaho_operations_mart" driverClassName="com.mysql.jdbc.Driver"
password="password" username="hibuser" maxWait="10000" maxIdle="5"
maxActive="20" factory="org.apache.commons.dbcp.BasicDataSourceFactory"
type="javax.sql.DataSource" auth="Container" name="jdbc/PDI_Operations_
Mart"/>
```

For Oracle:

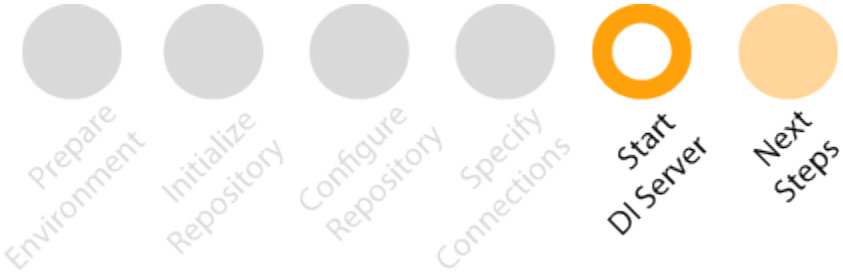
Comment out the resource references that refer to databases other than Oracle, such as PostgreSQL and MySQL. Then, add the following code to the file if it does not already exist.

```
<Resource validationQuery="select 1 from dual"
url="jdbc:oracle:thin:@localhost:1521/XE" driverClassName="oracle.jdbc.
OracleDriver" password="password" username="hibuser" maxWait="10000"
maxIdle="5" maxActive="20" factory="org.apache.commons.dbcp.
BasicDataSourceFactory" type="javax.sql.DataSource" auth="Container"
name="jdbc/Hibernate"/>
<Resource validationQuery="select 1 from dual"
url="jdbc:oracle:thin:@localhost:1521/XE" driverClassName="oracle.jdbc.
```

```
OracleDriver" password="password" username="hibuser" maxWait="10000"
maxIdle="5" maxActive="20" factory="org.apache.commons.dbcp.
BasicDataSourceFactory" type="javax.sql.DataSource" auth="Container"
name="jdbc/Audit"/>
<Resource validationQuery="select 1 from dual"
url="jdbc:oracle:thin:@localhost:1521/XE" driverClassName="oracle.jdbc.
OracleDriver" password="password" username="quartz" maxWait="10000"
maxIdle="5" maxActive="20" factory="org.apache.commons.dbcp.
BasicDataSourceFactory" type="javax.sql.DataSource" auth="Container"
name="jdbc/Quartz"/>
<Resource validationQuery="select 1 from dual"
url="jdbc:oracle:thin:@localhost:1521/XE" driverClassName="oracle.jdbc.
OracleDriver" password="password" username="hibuser" maxWait="10000"
maxIdle="5" maxActive="20" factory="org.apache.commons.dbcp.
BasicDataSourceFactory" type="javax.sql.DataSource" auth="Container"
name="jdbc/pentaho_operations_mart"/>
<Resource validationQuery="select 1 from dual"
url="jdbc:oracle:thin:@localhost:1521/XE" driverClassName="oracle.jdbc.
OracleDriver" password="password" username="hibuser" maxWait="10000"
maxIdle="5" maxActive="20" factory="org.apache.commons.dbcp.
BasicDataSourceFactory" type="javax.sql.DataSource" auth="Container"
name="jdbc/PDI_Operations_Mart"/>
```

4. Modify the username, password, driver class information, IP address (or domain name), and port numbers so they reflect the correct values for your environment.
5. Make sure that the `validationQuery` variable for your database is set to this.
   - **PostgreSQL:** `validationQuery="select 1"`
   - **MySQL:** `validationQuery="select 1"`
   - **Oracle:** `validationQuery="select 1 from dual"`
6. Save the `context.xml` file, then close it.
7. To make sure that the changes that you made in the context.xml file take effect when Tomcat is started, navigate to the `tomcat\conf\Catalina` directory. If the `pentaho.xml` file is in the present, delete it. It will be generated again when you start the BA Server, but will contain the changes that you just made in the `context.xml` file.

Start DI Server



After you've complete the tasks in the Prepare Environment, Initialize Repository, and Specify Connections steps, you can now install license keys and start the DI Server.

Install License Keys Using the Command Line Interface

1. Download the .lic file you want to install.
2. Copy your .lic files to the DI Server.
3. Navigate to the `licenses` directory. pdi/pdi-ee/data-integration/licenses
4. Run the license installation script.
    a. **For Linux:** Run `install_license.sh` with the install switch and the location and name of your .lic file as a parameter. You can specify multiple .lic files separated by spaces. Be sure to use backslashes to escape any spaces in the path or file name. `install_license.sh install /home/dvader/downloads/Pentaho\ DI\ Enterprise\ Edition.lic`
    b. **For Windows:** Run `install_license.bat` with the install switch and the location and name of your license file as a parameter. `install_license.bat install "C:\Users\dvader\Downloads\Pentaho DI Enterprise Edition.lic"`

Modify Tomcat Startup Script

The Tomcat startup script must be modified to include the `CATALINA_OPTS` variable. `CATALINA_OPTS` indicates the amount of memory to allocate. It also indicates where Pentaho licenses are installed. Specific instructions on how to modify the startup script depend on your operating system.

Modify the Tomcat Windows Startup Script

1. Make sure the Tomcat web application server is not running by starting the Windows **Task Manager** and looking for **Tomcat** in the **Applications** tab. If the server is running, stop it.
2. Use a text editor to open the `startup.bat` file, which is in the `bin` subdirectory of the Tomcat home directory.
3. Add this line directly before the `call "%EXECUTABLE%" start %CMD_LINE_ARGS%` line, which is located near the end of the file. `set CATALINA_OPTS=-Xms4096m -Xmx6144m -`

```
XX:MaxPermSize=256m -Dsun.rmi.dgc.client.gcInterval=3600000 -
Dsun.rmi.dgc.server.gcInterval=3600000 -
Dpentaho.installed.licenses.file=%PENTAHO_INSTALLED_LICENSE_PATH%
```

4. Save and close the file.

Modifying the Tomcat Linux Startup Script

1. Make sure the Tomcat web application server is not running by opening a **Terminal** window and typing `ps -A` at the prompt. If the server is running, stop it.
2. Use a text editor to open the `startup.sh` file, which is in the `bin` subdirectory of the Tomcat home directory.
3. Add this line directly before the `exec "$PRGDIR"/"$EXECUTABLE" start "$@"` line near the end of the file. `export CATALINA_OPTS="-Xms4096m -Xmx6144m -XX:MaxPermSize=256m -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Dpentaho.installed.licenses.file=$PENTAHO_INSTALLED_LICENSE_PATH"`
4. Save and close the file.

Windows Pentaho Archive Installation

If you used the Archive Installation, we provide individual control scripts to start and stop the DI server, and DI repository. Here is where you can find the individual control scripts.

DI Repository

- The Archive Installation enables you to install PostgreSQL, MySQL, or Oracle as the solution repository. Consult the third-party documentation for the RDBMS to find more information about starting and stopping.

  The solution repository must be started before the DI Server.

DI Server

- `/pentaho/server/data-integration-server`

Linux and Macintosh OS Pentaho Installation Wizard

When you ran the Installation Wizard on Linux, the DI Server was deployed in an included Apache Tomcat application server. You can control the Tomcat server using the start and stop scripts that come with the Pentaho installation. This script is also used as an easy way to start and stop the DI Server and the PostgreSQL repository. You can find this script at `/pentaho/ctlscript.sh`.

Here is a list of the script arguments you can use with the `data-integration-server` service.

Arguments

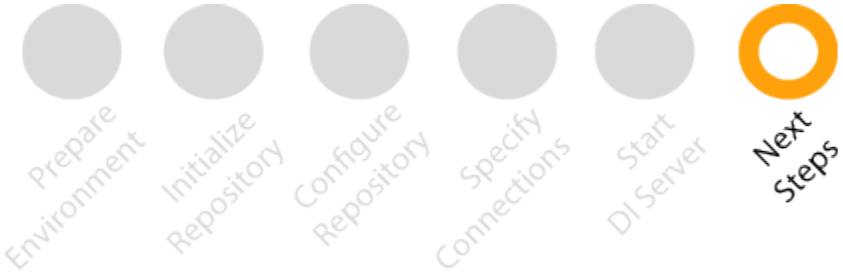
- start
- stop
- restart
- status
- help

```
./ctlscript.sh start data-integration-server
```

```
./ctlscript.sh status data-integration-server
```

```
./ctlscript.sh help
```

Next Steps



Now that you've installed the DI Server, do two things.

- Install the design tool and plugins.
- Configure the DI Server, design tool, and plugins.

Note: If you have installed the DI Server so that you can migrate content from the old system to this one, make sure that your license keys have been installed, then view the Upgrade BA and DI System instructions.

Learn More

- Learn about Big Data and Hadoop.