HITACHI
Inspire the Next

Pentaho Server: Optimizing Connection Pools

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

This page intentionally left blank.

Overview

This document covers some best practices on the optimization of database connection pools within the Pentaho Server.

These limits apply to each Pentaho Server; if you are running more than one server, check with your database administrator (DBA) for guidance about the correct number of connections.

This document is not intended to dictate what the best options are, but rather to present some best practices for customers who are interested in optimizing performance and response time in their Pentaho installations. Some of the topics covered here include connection parameters and considerations for configuring database server resources.

Software	Version(s)
Pentaho	6.x, 7.x, 8.0

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document.

This document assumes that you have knowledge of database connections and Pentaho, and that you already have Pentaho installed and configured for your environment.

More information about related topics outside of this document can be found in the Pentaho Documentation.

Use Case

Fabiola's users have been complaining about painfully slow connections to the database, especially reports that are very slow to load or save. She's also noticed a spike in resource usage in general, reasons that the two issues could be related, and decides to investigate ways to optimize the database connection pools to see if that would help.

Pentaho Server Database Connections

The Pentaho Server is designed to perform analytic queries and data transformations against a wide variety of database connections.

Normally, the server uses relational databases via Java Database Connectivity (JDBC) 3 and 4 compliant drivers. Other types, such as HTTP/REST services or custom libraries are supported for reporting and transformations, but the built-in data sources must be relational.

You can find details on these topics in the following sections:

The focus of this document is to provide guidance on the built-in (Repository) database connections of the server, and relational JDBC connections in general.

Table 1: Defaults for Built-In Connections

Built-In Connection Name	Default Username	Default Database and Schema Name
Jackrabbit	jcr_user	jackrabbit: public
Hibernate	hibuser	hibernate: public
Audit	hibuser	hibernate: public
Quartz	pentaho_user	quartz: public
PDI_Operations_Mart	hibuser	hibernate: pentaho_operations_mart
pentaho_operations_mart	hibuser	hibernate: pentaho_operations_mart
live_logging_info	hibuser	hibernate: pentaho_dilogs

Table 2: Components and Terminology

Component	Description	Notes
JDBC	Java Database Connectivity Specification	JDBC is a Java specification for accessing databases. The Pentaho Server supports JDBC 3 and JDBC 4 levels of the specification, with SQL 92 access.
JDBC Driver	Typically a JAR file	See each database provider for **classname** and required usage properties. The best practice is to use the official driver for the database, not from a third party.

Component	Description	Notes
JNDI	Java Naming Directory Interface	JNDI provides a way for the system to look up resources via a centrally managed name or ID. The database network address, port, and other main properties are managed here.
DBCP	Database Connection Pool	The DBCP manages database connections in a central pool for the rest of the system. A primary purpose of the DBCP is to pre-open and recycle the connections so that the rest of the application does not have to wait. The Pentaho Server 7.1 ships with Apache Commons DBCP version 1.4.

JNDI and DBCP Parameters

We recommend you use JNDI to manage connections. Below is an example of the Hibernate data source configuration from the default `tomcat/webapps/pentaho/META-INF/context.xml` file.

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

In this configuration:

Table 3: JNDI Parameters

Parameter	Definition
`maxTotal`	This is the maximum number of connections that can be borrowed from the pool at one time. A negative value means no limit. Other requests for a connection will wait if this limit is reached.
`maxIdle`	This is the maximum number of connections that can remain ready and unused in the pool.
`maxWaitMillis`	This is the maximum number of milliseconds that the pool will wait for a connection to be returned before throwing an exception.

The JNDI names that you add to `context.xml` become available to the Pentaho Server application, and referenceable under **Manage Data Sources** > **Database Connection** as shown in Figure 1:

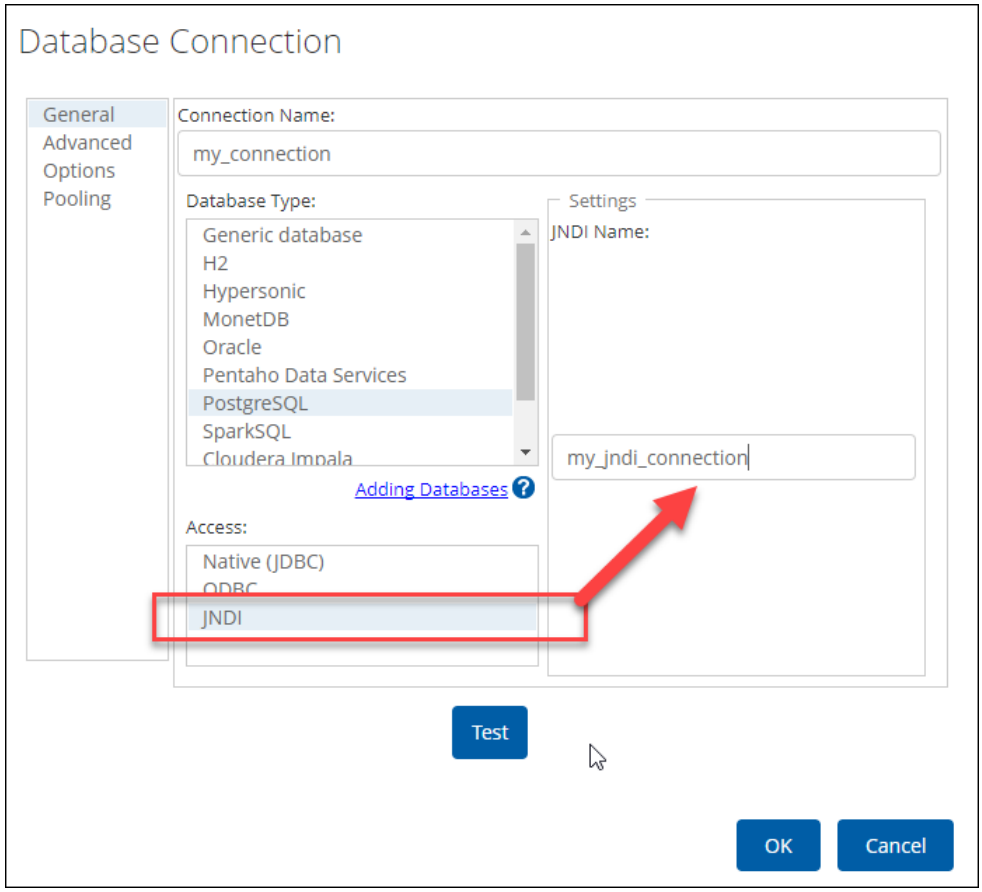

Figure 1: Pentaho Server > Manage Data Sources > Database Connection

Configure JNDI

Add your own JNDI definitions to the Pentaho Server for your regularly used database connections, and then reference them from your data source configuration with Pentaho Design tools and the Server Data Source configuration page.

- Configuring JNDI for Tomcat
- Configuring JNDI for JBoss EAP

Database Server Resources

Each type of database server has its own server-side internal resource pools and limits for managing connections.

For example, a standalone PostgreSQL server install has a default `max_connections` of 100, while it is common for Enterprise database servers to support 200 or more concurrent connections.

Consider this maximum connection limit when configuring your connection pool, remembering that:

- Read transactions are relatively fast, depending on the complexity of the query.
- Write transactions, such as with extract, transform, and load (ETL), require substantially more resources from the database server.
- Multiple concurrent writes to databases should be minimized.

Configure Connection Parameters

The best practice for Enterprise databases is to keep a minimal number of connections open to the database server. This constraint must be weighed against the need for application performance. Connections should be ready when they are likely to be demanded. The pool should also regularly close old connections and pre-open them before demand. The following parameters are key for this.

Table 4: Connection Parameters

Parameter	Definition
`maxIdle`	The maximum number of connections that can remain idle in the pool.
`maxConnLifetimeMillis`	The maximum number of milliseconds that a connection can remain opened. The default (negative value) indicates an infinite time.
`timeBetweenEvictionRunMillis`	The number of milliseconds that the evictor thread waits between cycles. By default, no evictor thread is run.
`testWhileIdle`	Indicates whether objects will be validated by the evictor thread. Default = `false`.
`testOnBorrow`	Indicates whether a connection will be validated just before a borrow event. Default = `true`.
`removeAbandonedTimeout`	The number of seconds before an abandoned connection is considered for removal from the pool.
`removeAbandonedOnMaintenance`	Indicates that borrowed (inactive) connections can be removed if the timeout is reached.

The usage profile of your application determines the best combination of values for your purposes. For example, if you have many reports that are used throughout the day, many idle connections are reasonable. If you have sporadic numbers of users, with extended periods of no activity, a small number of idle connections is reasonable.

The DBCP connection pool will allow up to `maxTotal` connections if they are actively being used. The following settings will help keep the number of connections to a minimum (`maxIdle`) when the connections are not being used.

- `maxIdle = 5`
- `maxConnLifetimeMillis = 60000`
- `timeBetweenEvictionRunsMillis = 60000`
- `testWhileIdle = true`
- `removeAbandonedTimeout = 300`
- `removeAbandonedOnMaintenance = true`

To ensure fast response on initial requests, it is good to populate the `initialSize` and `minIdle` parameters with a small value. This will initially open connections and keep them open.

- `initialSize = 2`
- `minIdle = 2`

Monitoring

The application log files will report an exception if a connection is not available to the application. You should have a process that scans the application logs regularly for these types of `ERROR` events. Live monitoring of JNDI resources can be done via JConsole and MBeans as shown:

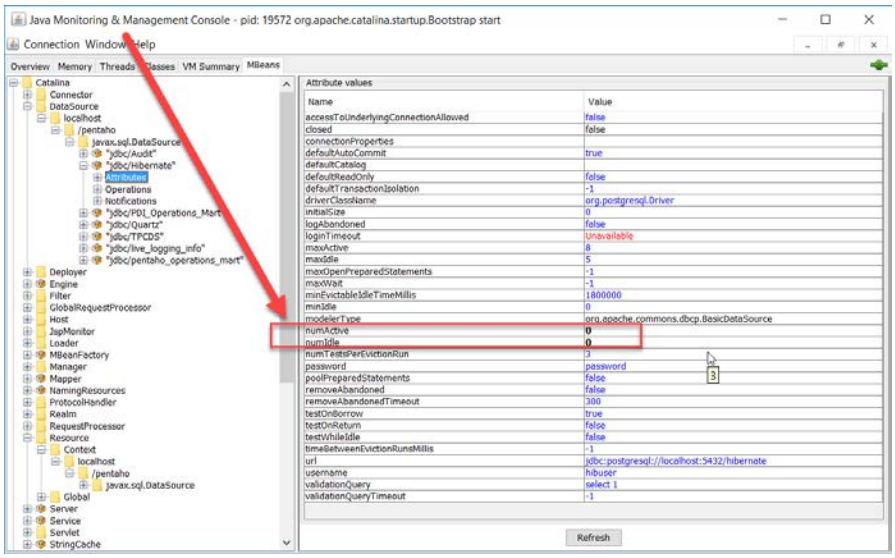

Figure 2: JConsole JNDI Monitoring

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- Apache Commons DBCP 1.4 Javadoc
- Apache Commons DBCP Configuration Parameters
- Configuring JNDI for Tomcat
- Configuring JNDI for JBoss EAP
- Logging and Monitoring Pentaho Servers
- Pentaho Components Reference
- Tomcat Performance Tuning for Pentaho

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project:_____

Date of the Review:_____

Name of the Reviewer:_____

Item	Response	Comments
Did you review the current database settings in your environment?	YES_____ NO_____	
Did you set up a process to monitor the utilization of resources?	YES_____ NO_____	
Did you develop a scaling plan to increase available resources?	YES_____ NO_____	