Linux on System z Web 2.0

Web 2.0 samples for accessing DB2 on Red Hat Enterprise Linux 5.2 February 2009

Linux on System z Web 2.0

Web 2.0 samples for accessing DB2 on Red Hat Enterprise Linux 5.2 February 2009

Note

Before using this document, be sure to read the information in “Notices” on page 69.

First Edition – February 2009

This edition applies to Red Hat Enterprise Linux 5.2 only.

Contents

Chapter 1. Introduction

Web 2.0 applications display a large amount of user supplied information. As back-ends to store the information, databases are used to provide a stable and structured environment.

The Red Hat Enterprise Linux 5.2 distribution includes several database servers such as MySQL and PostgreSQL, which are used by many Web 2.0 applications.

Today's Wiki's, Blogs and Content Management Systems are implemented in various programming languages. In addition to Java™, scripting languages such as Perl, Python, PHP and Ruby are used. This requires that database connectors are available for the related database to connect to.

In this document, the setup and configuration of programming languages is demonstrated to connect to IBM®'s database DB2 Enterprise Server Edition Version 9.5 LUW.

Selected DB2 Client configurations

The number of programming languages that are used for Web 2.0 applications is challenging system administrators to select the right programming language, which offers support for all requirements. Since Web 2.0 applications make use of databases that serve the user supplied information, connecting to IBM's database DB2 Enterprise Server Edition Version 9.5 might also become such a requirement to be addressed.

The programming languages and frameworks which are used in this document are described in another white paper 'Setting up a Web 2.0 stack on Red Hat Enterprise Linux 5.2' which is available at http://www.ibm.com/developerworks/linux/linux390/web20_rh5.html. The following is an overview about the explained setups in this document:

Accessing DB2® using Perl

- Using Perl DBI
- Using Perl DBIx::Class ORM

Accessing DB2 using Python

- Using Python with PyDB2
- Using Jython with zxJDBC

Accessing DB2 using PHP

- Using PHP with ibm_db2

Accessing DB2 using Ruby

- Using Ruby with IBM_DB
- Using JRuby with JDBC
- Using Ruby with ActiveRecord ORM
- Using Ruby on Rails with IBM_DB

Accessing DB2 using Java

- Using Java with JDBC
- Using Java with Hibernate

Accessing DB2 using Groovy

- Using Groovy with Groovy.sql
- Using Groowy with GORM

Requirements

System administrators who read this document are familiar with, how to set up YUM to include the Red Hat Enterprise Linux 5.2 DVD image and the related supplementary ISO image as software repositories. Also, the latest available security updates must have been applied to the system.

As required by any application, a specific environment must be set up. Also for a database application, the setup of some components must be in place before installation and execution of the database application. The basic setup of programming languages is described in another white paper 'Setting up a Web 2.0 stack on Red Hat Enterprise Linux 5.2', which is available at http://www.ibm.com/developerworks/linux/linux390/web20_rh5.html.

In this document, a DB2 Enterprise Server Edition Version 9.5 LUW installation with the default database instance "db2inst1" is used to run the examples. The setup of this database instance is not covered in this document. For more details, refer to the DB2 documentation.

Where to find this document

The latest version of this document and other Web 2.0 related documentation are available on the developerWorks® Web site ’Web 2.0 with RHEL5’.

Chapter 2. Setting up DB2 Client Interfaces

This chapter introduces two different client interfaces for accessing a DB2 database. These client interfaces are the base for programming language related DB2 client implementations. These interfaces are described in this chapter:

- DB2 Call Level Interface (DB2 CLI)
- DB2 Universal JDBC driver

Setting up the DB2 Call Level Interface

The DB2 Call Level Interface (CLI) provides a C-based interface for interacting with a DB2 database. It is used by all programming languages implemented in C that are covered in this document, specifically:

- Perl with DBD::DB2
- Python with PyDB2
- PHP with ibm_db2
- Ruby with IBM_DB

Installing DB2 CLI

DB2 CLI is included as an optional component in the DB2 Data Server Client and all DB2 Server and DB2® Connect™ editions. It is part of the "Base application development tools" component which can be chosen during the installation process.

The following steps are based on DB2 Enterprise Server Edition Version 9.5 and might vary for different editions and versions:

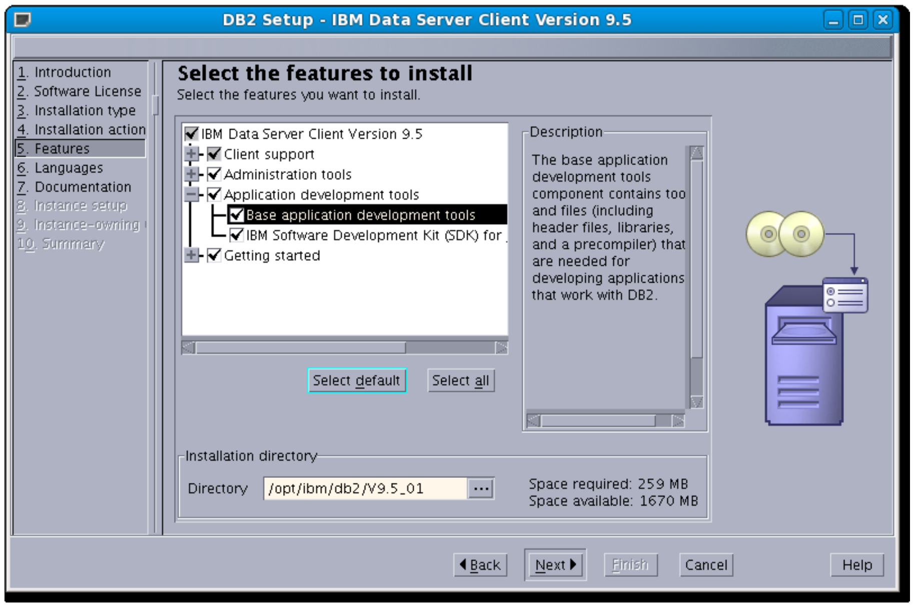
1. Navigate to the DB2 installation media directory and start the **db2setup** program by issuing this command:

   ```
   # ./db2setup
   ```

   Note: **db2setup** is a graphical setup tool and therefore requires access to an X-Server.
2. Select "Install a Product" and navigate to the product to be installed, either DB2 Enterprise Server Edition Version 9.5 or IBM Data Server Client Version 9.5.

   Note: In the following steps, the installation of IBM Data Server Client Version 9.5 is chosen.
3. Make the appropriate choice of **Install New** or **Work with Existing**.

   Note: In the following steps, the processing of the **Install New** procedure is explained.
4. Proceed through the installation dialog. When prompted to "Select the installation type", choose **custom**.
5. Proceed until prompted to "Select the features to install". Make sure to select the "Base application development tools" component below the "Application development tools" entry. See Figure 1 on page 4 for reference.



Select the features to install

Select the features you want to install.

- [✓] IBM Data Server Client Version 9.5
  - [✓] Client support
  - [✓] Administration tools
  - [✓] Application development tools
    - [✓] Base application development tools
    - [✓] IBM Software Development Kit (SDK) for
  - [✓] Getting started

Description

The base application development tools component contains too and files (including header files, libraries, and a precompiler) that are needed for developing applications that work with DB2.

Installation directory

Directory /opt/ibm/db2/V9.5_01

Space required: 259 MB
Space available: 1670 MB

Figure 1. DB2 Data Server Client: Selecting the "Base application development tools" for installation

6. Proceed with installation.

Basic setup for establishing secure connections

By default, all communication between DB2 clients and servers takes place unencrypted. Connections can be secured by enabling Secure Socket Layer (SSL) support, allowing for encrypted and signed communication over untrusted networks such as the Internet. This chapter provides a walkthrough for setting up client-side SSL support within CLI based on an existing server certificate.

After the DB2 server certificate has been imported into the client environment, secure connections can be initiated by any application based on CLI. See the "Establishing a secure connection" sections of the respective programming language in this document for details.

Prerequisites

The following prerequisites must be met to enable support for secure connection:

- DB2 Version 9.1 Fix Pack 5 or later
- IBM Java JRE 1.5
- DB2 Global Security Kit (GSKit)
- A valid DB2 server certificate in Base64–encoded ASCII format

Importing the DB2 server certificate

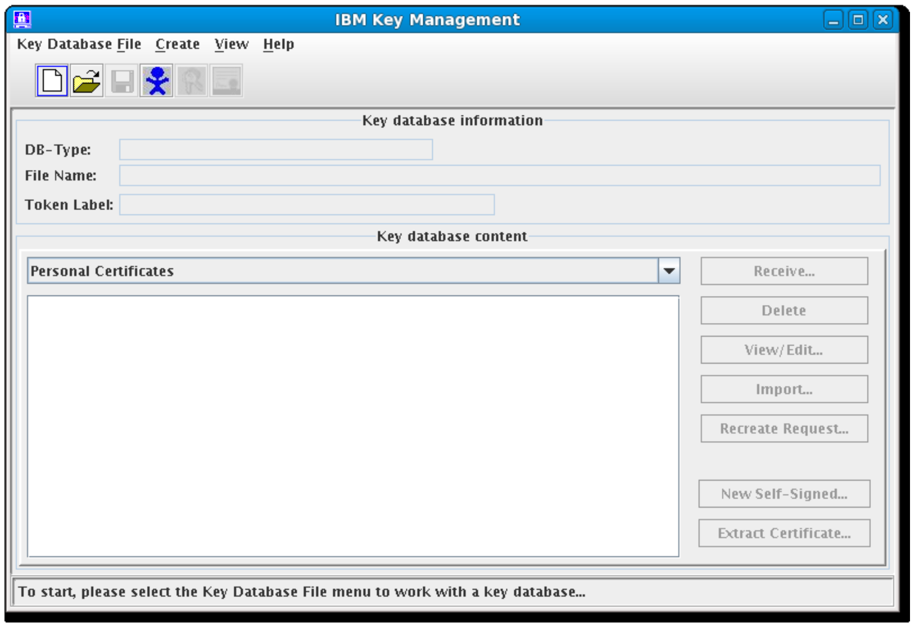
The DB2 server certificate is imported using the IBM Key Management tool, which is part of the DB2 Global Security Kit (GSKit). If the GSKit is not available on the system, follow the instructions given in *Configuring Secure Socket Layer (SSL) support in the DB2 client* to install it.

The following steps create a new keystore database and import a DB2 server certificate:

1. Issue the following command to start the IBM Key Management tool:

```
# gsk7ikm_64
```

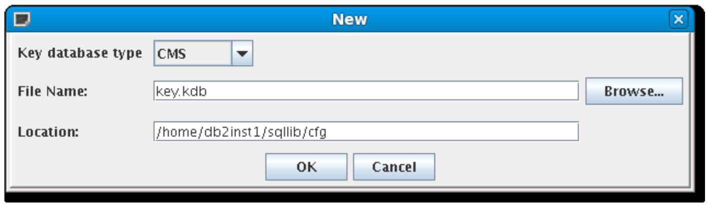
After issuing this command, the IBM Key Management window opens:



To start, please select the Key Database File menu to work with a key database...

Figure 2. IBM Key Management tool

2. To create a new keystore database, click **Key Database File**, then click **New**. The dialog that now opens has three input fields:



Key database type: CMS

File Name: key.kdb

Location: /home/db2inst1/sqllib/cfg

Figure 3. IBM Key Management: Create a new keystore

- The "Key database type" must be set to "CMS".
- The "File Name" specifies the name of the keystore. In this example “key.kdb” is used.
- The "Location" specifies the directory of the keystore. In this example “/home/db2inst1/sqllib/cfg/” is used.

  Note: The DB2 instance owner requires read access to this directory.

3. To create the new keystore database, click **OK**. Another dialog to enter the keystore database password opens. Make sure to select the **Stash the password to a file?** check box when creating the keystore password

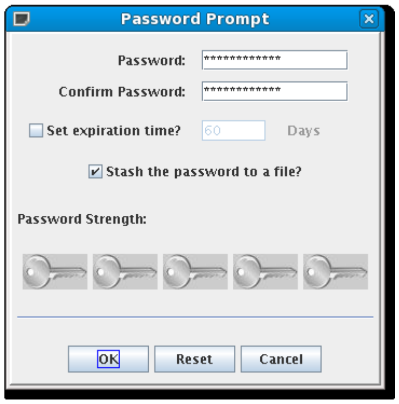
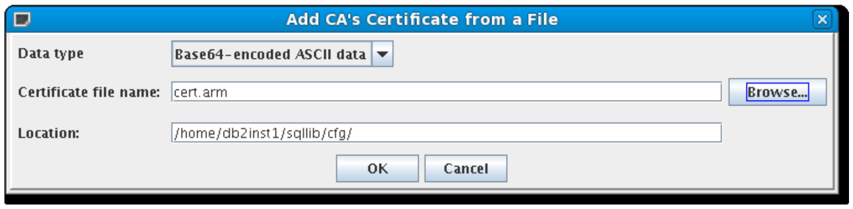


Figure 4. IBM Key Management: Set the password

4. To import the DB2 server certificate, make sure that **Signer certificates** in the **Key database content** area is selected, then click **Add...**. A dialog with three input fields appears:



Data type: Base64-encoded ASCII data
Certificate file name: cert.arm
Location: /home/db2inst1/sqllib/cfg/

Figure 5. IBM Key Management: Add CA's Certificate from a File

- The "Data type" must be set to "Base64-encoded ASCII data".
- The "Certificate file name" specifies the name of the keystore. In this example “cert.arm” is used.
- The "Location" specifies the directory of the keystore. In this example “/home/db2inst1/sqllib/cfg/” is used.

Click **OK** to complete the import.

After performing these steps, the DB2 CLI can now be configured to use the keystore. Login as the DB2 instance owner and create the SSL configuration file “/home/db2inst1/sqllib/cfg/SSLClientconfig.ini” with the following content:

```
DB2_SSL_KEYSTORE_FILE=/home/db2inst1/sqllib/cfg/key.kdb
DB2_SSL_KEYRING_STASH_FILE=/home/db2inst1/sqllib/cfg/key.sth
```

References

The following URLs provides more detailed information:

- Types of clients - DB2 Runtime Client and DB2 Client: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.uprun.doc/doc/c0022612.htm
- Configuring Secure Socket Layer (SSL) support in the DB2 client: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.uprun.doc/doc/t0053518.htm
- Configuring Secure Socket Layer (SSL) support in a DB2 instance: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.uprun.doc/doc/t0025241.htm

- DB2 technical tip: Set up Secure Sockets Layer (SSL) for DB2 on Windows®: http://www.ibm.com/developerworks/db2/library/techarticle/dm-0806sogalad/index.html

Setting up the DB2 Universal JDBC driver

The DB2 Universal JDBC driver provides an interface based on the Java Database Connectivity (JDBC) API used for interacting with a DB2 database. It is used by all programming languages running on a Java Virtual Machine (JVM) that are covered in this document, specifically:

- Jython using the DBI zxJDBC driver
- JRuby using JDBC or the ActiveRecord JDBC adapter
- Java using JDBC or Hibernate
- Groovy using Groovy.sql or GORM

Installing DB2 Universal JDBC driver

The DB2 Universal JDBC driver is included as a default component in the DB2 Data Server Client package and all DB2 Server and DB2 Connect editions.

Basic setup for establishing secure connections

By default, all communication between DB2 clients and servers takes place unencrypted. Connections can be secured by enabling Secure Socket Layer (SSL) support, allowing for encrypted and signed communication over untrusted networks such as the Internet. This chapter provides a walkthrough for setting up client-side SSL support based on an existing server certificate.

After the DB2 server certificate has been imported, secure connections can be initiated by any application using the DB2 Universal JDBC driver. See the "Establishing a secure connection" sections of the respective programming language in this document for details.

Prerequisites

The following prerequisites must be met in order to enable support for secure connection:

- DB2 Version 9.1 Fix Pack 5 or later
- IBM Java JRE 1.5
- A valid DB2 server certificate in Base64–encoded ASCII format

Importing the DB2 server key

Import the DB2 server certificate into the client keystore by running the **keytool** command. Replace “cert.arm” with the file name of the DB2 server's certificate:

```
# keytool -import -file cert.arm -keystore /root/.keystore
```

Ensure that Java is using the correct store and password by setting the javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword properties. The following example assumes that the DB2 server certificate was imported into /root/.keystore and that the password of that keystore is "client":

```
# export JAVA_OPTS="-Djavax.net.ssl.trustStore=/root/.keystore \
-Djavax.net.ssl.trustStorePassword=client $JAVA_OPTS"
```

Applications using JDBC can now use SSL by either setting the property sslConnection to true or by specifying sslConnection=true in the connection URL.

References

The following URLs provide more detailed information:

- Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver: http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/ad/cjvjt2cn.htm
- Properties for the IBM DB2 Driver for JDBC and SQLJ: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.java.doc/doc/c0024688.htm

Chapter 3. Accessing DB2 using Perl

This chapter introduces several ways to access a DB2 database developed with the Perl programming language as shown in Figure 6.

- “Using Perl DBI” provides instructions for connecting to a DB2 database using the Perl Database Interface (DBI) module, based on the DBD::DB2 driver developed and supported by IBM.
- “Using Perl DBIx::Class ORM” on page 12 builds on the preceding chapter, introducing Perl DBIx::Class, an object-relational mapper (ORM) that uses DBI and the DBD::DB2 driver for database access.

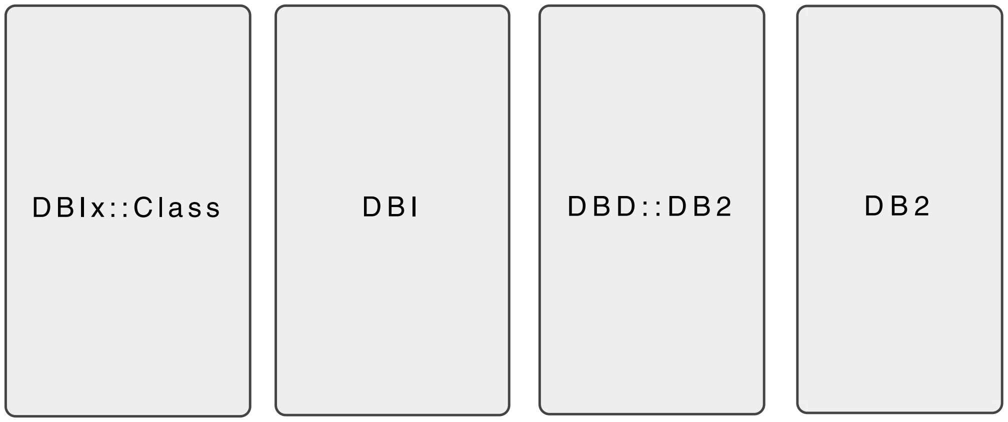


Figure 6. Accessing DB2 using Perl

There are other ways to connect to a DB2 database from Perl that are not covered in this document, including:

- Class::DBI (http://search.cpan.org/dist/Class-DBI/lib/Class/DBI.pm), a database abstraction layer.

Using Perl DBI

The Perl Database Interface API (DBI) module provides database access for client applications written in Perl based on an abstract interface that is implemented by database driver (DBD) modules. DB2 support is implemented by the DBD::DB2 driver developed and supported by IBM. This driver relies on the DB2 Call Level Interface (DB2 CLI) for database access.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the DBD::DB2 driver:

- DB2 Call Level Interface (DB2 CLI) (see “Setting up the DB2 Call Level Interface” on page 3 for details)
- Perl 5.006 or later
- Perl DBI 1.2.1 or later (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)
- Installation of Perl DBD::DB2 (as outlined below)

Installation

Before installing the DBD::DB2 driver, make sure that the DB2_HOME environment variable is set and points to the “sqllib” directory of the DB2 instance. The following example assumes that the DB2 instance is named "db2inst1":

```
# export DB2_HOME=/home/db2inst1/sqllib
```

The DBD::DB2 driver for Perl is available on the standard Perl package repository CPAN and can be installed by running the following command:

```
# cpan DBD::DB2
```

Note: Installing DBD::DB2 using the above command requires Internet access.

Connection setup

Establishing a connection

This Perl code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
use DBI;

my $dbh = DBI->connect("dbi:DB2:DATABASE=SAMPLE;HOSTNAME=db2.example.com; " .
 "PORT=50000;PROTOCOL=TCPIP;UID=db2inst1;PWD=db2inst1;", "", "" ) ||
 die "Can't connect to SAMPLE database: $DBI::errstr";
```

Establishing a secure connection

Secure connections using SSL can be established by adding the SECURITY=SSL keyword to the connection string.

This Perl code segment gestablishes a secure TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
use DBI;

my $dbh = DBI->connect("dbi:DB2:DATABASE=SAMPLE;HOSTNAME=db2.example.com; " .
 "PORT=50000;PROTOCOL=TCPIP;SECURITY=SSL;UID=db2inst1;PWD=db2inst1;", "", "" ) ||
 die "Can't connect to SAMPLE database: $DBI::errstr";
```

Note: CLI must be configured to use the correct certificate to create an SSL connection. See “Basic setup for establishing secure connections” on page 4 for details.

Example

This example connects to the "SAMPLE" DB2 database and creates a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

1. "Establish connection" obtains a connection to the DB2 database by calling connect on the DBI module using the connection string given in “Establishing a connection.”
2. "Create the PERSON table" uses a prepared statement to run a SQL query against the system catalog. The result of that query determines whether the

″PERSON″ table is created. If the fetch call on the statement handler evaluates to false, indicating that no records were found, the appropriate SQL statements for creating that table are generated and run.

3. "Insert record" demonstrates binding variables to a prepared statement as well as querying keys generated automatically through the IDENTITY column. The key created by that column is read into the last_id variable by binding the returned column value to the variables' reference.
4. "Select record" uses a prepared statement to load a given person by id, traversing through the returned list by calling fetchrow on the statement handler, which returns an array representation of the record that was found.
5. "Update record" changes the name attribute of the previously created record using another prepared statement.

Copy the following code into a new file called “db2sample-perl-dbi.pl”:

```
#!/usr/bin/env perl

use DBI;
use warnings;
use strict;

###############################################################################
# 1. Establish connection
my $dbh = DBI->connect("dbi:DB2:DATABASE=SAMPLE;HOSTNAME=db2.example.com; " .
   "PORT=50000;PROTOCOL=TCPIP;UID=db2inst1;PWD=db2inst1;", "", "" ) ||
   die "Can't connect to SAMPLE database: $DBI::errstr";

###############################################################################
# 2. Create the person table
my $stmt = "SELECT NAME FROM SYSIBM.SYSTABLES WHERE NAME='PERSON'";
my $sth = $dbh->prepare($stmt) or die "Cannot prepare: ".$dbh->errstr;
$sth->execute() || die "Cannot execute: ".$sth->errstr;
unless ($sth->fetch()) {
   print "Creating new person table...\n";
   $stmt = "CREATE TABLE person (
      id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
      name VARCHAR(50) NOT NULL,
      birthdate DATE )";
   $sth = $dbh->prepare($stmt) or die "Cannot prepare: ".$dbh->errstr;
   $sth->execute() || die "Cannot execute: ".$sth->errstr;
   print "Person table created\n";
}

###############################################################################
# 3. Insert record
print "Adding person...\n";
my $last_id;
$stmt = "SELECT id FROM NEW TABLE (
   INSERT INTO person (name, birthdate) VALUES (?, ?))";
$sth = $dbh->prepare($stmt) || die "Cannot prepare: ".$dbh->errstr;
$sth->bind_param(1, "hugo");
$sth->bind_param(2, "2008-08-14");
$sth->execute() || die "Cannot execute: ".$sth->errstr;
# getting the last inserted ID
$sth->bind_col(1, \$last_id);
$sth->fetch() || die "Cannot get last id: ".$sth->errstr;
print "Person added\n";

###############################################################################
# 4. Select record
print "Selecting person with id = $last_id\n";
my ($id, $name, $birthdate);
$stmt = "SELECT * FROM person WHERE id = ?";
$sth = $dbh->prepare($stmt) or die "Cannot prepare: ".$dbh->errstr;
$sth->bind_param(1, $last_id);
```

```
$sth->execute() || die "Cannot execute: ".$sth->errstr;
while (($id, $name, $birthdate) = $sth->fetchrow()){
   print "ID: $id, NAME: $name, BIRTHDATE: $birthdate\n";
}

################################################################################
# 5. Update record
print "Updating person with id = $last_id\n";
$stmt = "UPDATE person SET name = ? WHERE id = ?";
$sth = $dbh->prepare($stmt) or die "Cannot prepare: ".$dbh->errstr;
$sth->bind_param(1, "hugo2");
$sth->bind_param(2, $last_id);
$sth->execute() || die "Cannot execute: ".$sth->errstr;

################################################################################
# 4. Select record
print "Selecting person with id = $last_id\n";
$stmt = "SELECT * FROM person WHERE id = ?";
$sth = $dbh->prepare($stmt) || die "Cannot prepare: ".$dbh->errstr;
$sth->bind_param(1, $last_id);
$sth->execute() || die "Cannot execute: ".$sth->errstr;
while (($id, $name, $birthdate) = $sth->fetchrow()){
   print "ID: $id, NAME: $name, BIRTHDATE: $birthdate\n";
}

$sth->finish();
$dbh->disconnect();
```

Running the example

To start the example, issue the following command:

```
# perl db2sample-perl-dbi.pl
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using Perl DBIx::Class ORM

DBIx::Class is an object-relational mapper (ORM) that provides a high-level object-oriented abstraction layer by mapping model classes to tables with columns represented as attributes. This approach allows for database vendor independence and helps writing clean, maintainable code.

DBIx::Class builds on top of DBI and provides database connectivity by relying on DBD drivers, with additional database specific handling provided by DBIx::Class::Storage::DBI modules, which are loaded automatically based on the database used.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using Perl DBIx::Class:

- DB2 Call Level Interface (DB2 CLI) (see “Setting up the DB2 Call Level Interface” on page 3 for details)
- Perl DBI together with the DBD::DB2 driver (see “Using Perl DBI” on page 9 for installation instructions)
- Installation of Perl DBIx::Class (as outlined below)

Installation

Perl DBIx::Class is available on the standard Perl package repository CPAN and can be installed by running the following command:

```
# cpan install DBIx::Class
```

Note: Installing DBIx::Class using the above command requires Internet access. To run the example given in this chapter, the SQL::Translator is required to be available on the system also. Issue the following command to install the SQL::Translator:

```
# cpan install Module::Build::Compat
# cpan install SQL::Translator
```

Example

This example contains the creation of a database schema and table model. The Perl script will connect to the DB2 database named "SAMPLE" and create a table named ″PERSON″. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

Create the sample database schema

The “Sample.pm” file extends DBIx::Class::Schema and is responsible for loading the class files that map to the respective tables, in this example Sample::Person.

Copy the following code into a new file named “Sample.pm”:

```
package Sample;
use base qw/DBIx::Class::Schema/;

__PACKAGE__->load_classes(qw/Person/);

1;
```

Create the Person table model

The Person model represents the ″PERSON″ table. Instance attributes are mapped to columns of that table and allow automatic table creation based on the information stored in the model.

First, prepare the Sample package structure by creating the Sample directory:

```
# mkdir Sample
```

Copy the following code into a new file named “Sample/Person.pm” :

```
package Sample::Person;
use base qw/DBIx::Class/;

__PACKAGE__->load_components(qw/PK::Auto Core/);
__PACKAGE__->table('person');

__PACKAGE__->add_columns(
   id => {
      data_type => 'integer',
      is_auto_increment => 1
   },
   name => {
      data_type => 'varchar',
      size => 50
   },
   birthdate => {
      data_type => 'date'
   }
);
__PACKAGE__->set_primary_key('id');

1;
```

Create the sample script

This sample script demonstrates the use of the above sample schema and Person model by connecting to the "SAMPLE" database on "db2.example.com":

1. "Establish connection" uses the connection setup shown in “Establishing a connection” on page 10 to obtain a database connection based on the "Sample" schema to the "SAMPLE" DB2 database on "db2.example.com", port 50000.
2. "Create the PERSON table" uses the deploy method to create the ″PERSON″ table based on the column information stored in Person.pm. If previous examples have been run and the table already exists, it is dropped before table creation.
3. "Insert record" creates a new Person object based on the attributes supplied, and stores the associated record in the ″PERSON″ table.
4. "Select record" prints the id, name and birthdate values of the created record to the console.
5. "Update record" uses the Person instance name modifier to change the name attribute and makes permanent that change by calling the persons' update method.

Copy the following Perl code into a new filed named “db2sample-perl-dbix.pl”:

```
#!/usr/bin/env perl

use strict;
use warnings;
use Sample;

################################################################################
# 1. Establish connection
my $conn = Sample->connect(
   "dbi:DB2:DATABASE=SAMPLE;HOSTNAME=db2.example.com; " .
   "PORT=50000;PROTOCOL=TCPIP;UID=db2inst1;PWD=db2inst1;", "", "" );

################################################################################
# 2. Create the person table
print "Creating new person table...\n";
$conn->deploy({ add_drop_table => 0 });
print "Person table created\n";

################################################################################
```

```
# 3. Insert record
print "Adding person...\n";
my $person = $conn->resultset('Person')->create({
   name => "hugo",
   birthdate => "2008-08-14"
});
print "Person added\n";

################################################################################
# 4. Select record
print "Selecting person with id = ".$person->id."\n";
print "ID: ".$person->id.", NAME: ".$person->name.
   ", BIRTHDATE: ".$person->birthdate."\n";

################################################################################
# 5. Update record
print "Updating person with id =".$person->id."\n";
$person->name('hugo2');
$person->update;

################################################################################
# 4. Select record
print "Selecting person with id = ".$person->id."\n";
print "ID: ".$person->id.", NAME: ".$person->name.
   ", BIRTHDATE: ".$person->birthdate."\n";
```

Running the example

To start the example, issue the following command:

```
# perl db2sample-perl-dbix.pl
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

References

The following URLs provides more detailed information:

- DBD::DB2 API documentation: http://search.cpan.org/%7Eibmtordb2/DBD-DB2-1.1/DB2.pod
- DBD::DB2 Tutorial: http://www.ibmdatabasemag.com/showArticle.jhtml?articleID=59301551
- DBIx::Class Tutorial: http://search.cpan.org/dist/DBIx-Class/lib/DBIx/Class/Manual/Intro.pod

Chapter 4. Accessing DB2 using Python

This chapter introduces two ways to access a DB2 database from applications written with the Python programming language. Both methods conform to the Python Database API Specification v2.0 (PEP-249), thus providing a common interface for database access:

- "Using Python with PyDB2" provides instructions for connecting to a cataloged DB2 database with PyDB2, a database driver for Python built on the DB2 Call Level Interface (DB2 CLI).
- "Using Jython with zxJDBC" on page 20 details database connectivity for Jython applications using the Java Database Connectivity API (JDBC) in conjunction with the DB2 Universal JDBC driver.

There are other ways to connect to a DB2 database from Python that are not covered in this white paper, including:

- ibm_db (http://pypi.python.org/pypi/ibm_db/) is a Python Database API Specification v2.0 (PEP-249) compliant driver built on the DB2 Call Level Interface (DB2 CLI).

Using Python with PyDB2

PyDB2, a database driver for Python, provides a Python Database API Specification v2.0 (PEP-249) compliant driver built on the DB2 Call Level Interface (DB2 CLI) that is released as Open Source Software under the LGPL and available from SourceForge. This database driver works with catalog databases.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the PyDB2 driver:

- DB2 Call Level Interface (DB2 CLI) (See "Setting up the DB2 Call Level Interface" on page 3 for details)
- Python 2.4 or later
- Installation of PyDB2 driver (as outlined below)

Installation

To install the PyDB2 driver, complete the following steps:

1. Before installing PyDB2, the profile of the DB2 instance must be sourced. The following example assumes that the DB2 instance is named ″db2inst1″:

   ```
   # . /home/db2inst1/sqllib/db2profile
   ```

2. Download the latest version from http://sourceforge.net/project/showfiles.php?group_id=67548, version 1.1.1-1 at time of writing:

   ```
   # wget http://downloads.sourceforge.net/pydb2/PyDB2_1.1.1-1.tar.gz
   ```

3. Extract the downloaded file by running the following command:

   ```
   # tar xzf PyDB2_1.1.1-1.tar.gz
   ```

4. Change into the PyDB2_1.1.1 directory that was created by unpacking the archive:

```
# cd PyDB2_1.1.1
```

5. Build and install the PyDB2 driver by issuing the following commands:

```
# python setup.py build
# python setup.py install
```

Note: If the build process fails with warnings about differing signedness, the wrong library files might have been used. Ensure that the 64–bit DB2 Call Level Interface (DB2 CLI) library is used by creating a softlink from the lib64 directory to lib. This can be accomplished by issuing the following command (replace V9.5 with the version of DB2 installed on the system):

```
# ln -s /opt/ibm/db2/V9.5/lib64 /opt/ibm/db2/V9.5/lib
```

After the link has been created, retry the build process.

Connection setup

Connection prerequisites

Establishing catalog connections to a DB2 database with Python requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

Establishing a cataloged connection

This Python code segment establishes a connection to the cataloged "SAMPLE" database, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
import DB2

conn = DB2.connect(dsn='SAMPLE', uid='db2inst1', pwd='db2inst1')
```

Note: To use the cataloged connection with SSL, refer to the DB2 documentation how to setup the cataloged connection with SSL.

Example

This example will connect to the catalog "SAMPLE" DB2 database and create a table named ″PERSON″ unless, that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

- "Establish connection" opens a connection to the DB2 database using the connection parameters given in “Establishing a cataloged connection” and stores a database cursor into the curs variable.
- "Create the PERSON table" uses the execute method of that curs variable to run a query against the system catalog. The results of that query determine whether the ″PERSON″ table is created. If no rows are returned, the appropriate SQL statements for creating that table are generated.
- "Insert record" demonstrates the use of prepared statements by binding name and birthdate columns to placeholders, which are replaced with contents from the

person array defined beforehand. The fetchone method call returns the primary key that was automatically generated by the IDENTITY column as the first field of the first row, and stores that value into the id variable.

- ″Select record″ uses the id variable created in the previous step to retrieve the added person record using the fetchone method, and prints the resulting data set to the console.
- "Update record" uses another prepared statement to change the name attribute.

Copy the following code into a new file named “db2sample-python-pydb2.py”:

```
#!/usr/bin/env python

import DB2

################################################################################
# 1. Establish connection
conn = DB2.connect(dsn='SAMPLE', uid='db2inst1', pwd='db2inst1')
curs = conn.cursor()

################################################################################
# 2. Create the person table
curs.execute("SELECT NAME FROM SYSIBM.SYSTABLES WHERE NAME='PERSON'")
if not curs.fetchone():
    print "Creating new person table..."
    sql = """CREATE TABLE person (
        id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        name VARCHAR(50) NOT NULL,
        birthdate DATE )"""
    curs.execute(sql)
    print "Person table created"

################################################################################
# 3. Insert record
print "Adding person..."
person = ( "hugo", "2008-08-14" )
sql = """SELECT id FROM NEW TABLE (
    INSERT INTO person (name, birthdate) VALUES (?, ?) )"""
curs.execute(sql, person)
id = curs.fetchone()[0]
print "Person added"

################################################################################
# 4. Select record
print "Selecting person with id = " + str(id)
curs.execute('SELECT * FROM person WHERE id = ' + str(id))
row = curs.fetchone()
print "ID: " + str(row[0]) + ", NAME: " + str(row[1])  + \
      ", BIRTHDATE: " + str(row[2])

################################################################################
# 5. Update record
print "Updating person with id = " + str(id)
curs.execute('UPDATE person SET name = ? WHERE id = ?', ("hugo2", id))

################################################################################
# 4. Select record
print "Selecting person with id = " + str(id)
curs.execute('SELECT * FROM person WHERE id = ' + str(id))
row = curs.fetchone()
print "ID: " + str(row[0]) + ", NAME: " + str(row[1])  + \
      ", BIRTHDATE: " + str(row[2])

curs.close()
conn.close()
```

Running the example

Before running the example, the DB2 instance profile must be loaded as shown in "Connection prerequisites" on page 18.

Run the example by issuing the following command:

```
# python db2sample-python-pydb2.py
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using Jython with zxJDBC

The zxJDBC package for Jython detailed in this chapter provides a Python Database API Specification v2.0 (PEP-249) compliant driver built on the Java Database Connectivity API (JDBC). This allows use of the zxJDBC package in conjunction with the DB2 Universal JDBC driver for accessing a DB2 database.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using zxJDBC for Jython:

- DB2 Universal JDBC driver (see "Setting up the DB2 Universal JDBC driver" on page 7 for details)
- Jython (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site 'Web 2.0 with RHEL5')

Connection setup

Connection prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This includes the DB2 Universal JDBC driver into the class path.

Establishing a connection

This example Jython code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
from com.ziclix.python.sql import zxJDBC

conn = zxJDBC.connect(
  "jdbc:db2://db2.example.com:50000/SAMPLE", # url
  "db2inst1", # username
  "db2inst1", # password
  "com.ibm.db2.jcc.DB2Driver" # driver
)
```

Establishing a secure connection

This example Jython code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
from com.ziclix.python.sql import zxJDBC

conn = zxJDBC.connect(
  "jdbc:db2://db2.example.com:40397/SAMPLE:sslConnection=true;", # url
  "db2inst1", # username
  "db2inst1", # password
  "com.ibm.db2.jcc.DB2Driver" # driver
)
```

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

The following example will connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

- "Establish connection" opens a connection to the DB2 database using the connection parameters given in “Establishing a connection” on page 20, and stores a database cursor into the curs variable.
- "Create the PERSON table" uses the execute method of that curs variable to run a query against the system catalog. The results of that query determine whether the ″PERSON″ table is created. If no rows are returned, the appropriate SQL statements for creating that table are generated.
- "Insert record" demonstrates the use of prepared statements by binding name and birthdate columns to placeholders, which are replaced with contents from the person array defined beforehand. The fetchone method call returns the primary key that was automatically generated by the IDENTITY column as the first field of the first row, and stores that value into the id variable.
- ″Select record″ uses the id variable created in the previous step to retrieve the added person record using the fetchone method, and prints the resulting data set to the console.
- "Update record" uses another prepared statement to change the name attribute.

Copy the following code into a new file named “db2sample-python-zxjdbc.py”:

```
from com.ziclix.python.sql import zxJDBC

################################################################################
# 1. Establish connection
conn = zxJDBC.connect(
  "jdbc:db2://db2.example.com:50000/SAMPLE", # url
  "db2inst1", # username
  "db2inst1", # password
  "com.ibm.db2.jcc.DB2Driver" # driver
)
curs = conn.cursor()

################################################################################
# 2. Create the person table
curs.execute("SELECT NAME FROM SYSIBM.SYSTABLES WHERE NAME='PERSON'")
if not curs.fetchone():
 print "Creating new person table..."
        sql = """CREATE TABLE person (
   id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
   name VARCHAR(50) NOT NULL,
   birthdate DATE )"""
 curs.execute(sql)
 print "Person table created"

################################################################################
# 3. Insert record
print "Adding person..."
person = ( "hugo", "2008-08-14" )
sql = """SELECT id FROM NEW TABLE (
  INSERT INTO person (name, birthdate) VALUES (?, ?) )"""
curs.execute(sql, person)
id = curs.fetchone()[0]
print "Person added"

################################################################################
# 4. Select record
print "Selecting person with id = " + str(id)
curs.execute('SELECT * FROM person WHERE id = ' + str(id))
row = curs.fetchone()
print "ID: " + str(row[0]) + ", NAME: " + str(row[1])  + \
      ", BIRTHDATE: " + str(row[2])

################################################################################
# 5. Update record
print "Updating person with id = " + str(id)
curs.execute('UPDATE person SET name = ? WHERE id = ?', ("hugo2", id))

################################################################################
# 4. Select record
print "Selecting person with id = " + str(id)
curs.execute('SELECT * FROM person WHERE id = ' + str(id))
row = curs.fetchone()
print "ID: " + str(row[0]) + ", NAME: " + str(row[1]) + \
      ", BIRTHDATE: " + str(row[2])

curs.close()
conn.commit()
conn.close()
```

Running the example

Before running the example, ensure that the Java class path includes the DB2 Universal JDBC driver as shown in “Connection prerequisites” on page 20.

Run the example by issuing the following command:

```
# jython db2sample-python-zxjdbc.py
```

Note: If Jython is used with a secured connection, submit the javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword properties directly to the Jython call:

```
jython -Djavax.net.ssl.trustStore=/root/.keystore \
       -Djavax.net.ssl.trustStorePassword=client db2sample-python-zxjdbc.py
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

References

The following URLs provides more detailed information:

- PyDB2 project page: http://sourceforge.net/projects/pydb2
- Using Python to access DB2 for Linux®: http://www.ibm.com/developerworks/edu/i-dw-db2pylnx-i.html
- Jython project page: http://www.jython.org
- Jython User Guide: Database connectivity in Jython: http://www.jython.org/Project/userguide.html#database-connectivity-in-jython

Chapter 5. Accessing DB2 using PHP

This chapter introduces the ibm_db2 driver for PHP that enables applications written in the PHP programming language to access DB2 databases.

Using PHP with ibm_db2

The ibm_db2 driver for PHP provides database access using an API built on the DB2 Call Level Interface (DB2 CLI). It supports multiple connections, prepared statements and stored procedures.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the ibm_db2 driver for PHP:

- DB2 Call Level Interface (DB2 CLI) (see “Setting up the DB2 Call Level Interface” on page 3 for details)
- PHP and the PHP Extension Community Library (PECL) (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)
- Installation of ibm_db2 driver for PHP (as outlined below)

Installation

Before installing ibm_db2 driver for PHP, the profile of the DB2 instance must be sourced. The following example assumes that the DB2 instance is named "db2inst1":

```
# . /home/db2inst1/sqllib/db2profile
```

Install the ibm_db2 driver for PHP PECL extension by running the following command:

```
# pecl install ibm_db2
```

Note: Installing the ibm_db2 driver for PHP using the above command requires Internet access.

During this process, the directory of the DB2 Client Library installation must be specified:

```
DB2 Installation Directory? : /home/db2inst1/sqllib
```

After the compilation and installation process is finished, the following output is displayed:

```
Build process completed successfully
Installing '/usr/lib64/php5/extensions/ibm_db2.so'
install ok: channel://pear.php.net/ibm_db2-1.8.0
```

Create a new configuration file in “/etc/php.d” named “db2.ini” with the following content, replacing "db2inst1" with the name of the DB2 instance to be used:

```
extension=ibm_db2.so
ibm_db2.instance_name=db2inst1
```

If the Apache HTTP server is used to run the PHP script, the changes to the PHP configuration must be propagated. Therefore, restart the Apache HTTP server by running:

```
# service httpd restart
```

Connection setup

Connection prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

Establishing a connection

This PHP code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
$conn = db2_connect(
        "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=SAMPLE;" .
        "HOSTNAME=db2.example.com;PORT=50000;" .
        "PROTOCOL=TCPIP;UID=db2inst1;PWD=db2inst1;", "", "");
```

Establishing a secure connection

Secure connections using SSL can be established by adding the SECURITY=SSL keyword to the connection string.

This PHP code segment establishes a secure TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
$conn = db2_connect(
        "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=SAMPLE;" .
        "HOSTNAME=db2.example.com;PORT=40397;" .
        "PROTOCOL=TCPIP;SECURITY=SSL;UID=db2inst1;PWD=db2inst1;", "", "");
```

Note: CLI must be configured to use the correct certificate to create an SSL connection. See “Basic setup for establishing secure connections” on page 4 for details.

Example

This example will connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

1. "Establish connection" obtains a connection to the "SAMPLE" DB2 database by calling db2_connect using the connection string given in “Establishing a connection.”
2. "Create the PERSON table" first queries the system catalog to determine whether the ″PERSON″ table exists. If the db2_fetch_row call on the statement handler evaluates to false, indicating that no records were found, the appropriate SQL statements for creating that table are generated and executed.

3. "Insert record" demonstrates the usage of prepared statements and automatically generated primary keys. After the record is inserted, its key is retrieved by calling db2_fetch_row to load the first row and db2_result to get the value of the id column.
4. "Select record" uses a prepared statement to load a given person by id, traversing through the returned list by calling db2_fetch_object on the statement handle, which returns an object representation of the found record with column names mapped to fields.
5. "Update record" changes the name attribute of the previously created record through another prepared statement.

Copy the following code into a new file named “db2sample-php-ibm_db2.php”:

```
<?php
header("Content-type: text/plain");

//=============================================================================
// 1. Establish connection
$conn = db2_connect(
    "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=SAMPLE;" .
    "HOSTNAME=db2.example.com;PORT=50000;" .
    "PROTOCOL=TCPIP;UID=db2inst1;PWD=db2inst1;", "", "");
if (!$conn) {
    echo "Failed to connecto to the database.\n";
    echo "Error message: " . db2_conn_errormsg();
}

//=============================================================================
// 2. Create the person table
$result = db2_exec($conn,
                   "SELECT NAME FROM SYSIBM.SYSTABLES WHERE NAME='PERSON'");
if (!db2_fetch_row($result)) {
    echo "Creating new person table...\n";
    $result = db2_exec($conn, "CREATE TABLE person (
        id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        name VARCHAR(50) NOT NULL,
        birthdate DATE )");
    if (!$result) {
        echo "Person table not created.\n";
        die();
    }
    echo "Person table created\n";
}

//=============================================================================
// 3. Insert record
echo "Adding person...\n";
$stmt = db2_prepare($conn,
                    "INSERT INTO person (name, birthdate) VALUES (?, ?)");
$result = db2_execute($stmt, array('hugo', '2008-08-14'));
if (!$result) {
    echo "Could not add person:\n";
    echo "Message: " . db2_stmt_errormsg();
    echo "SQLSTATE: " . db2_stmt_error();
    die();
}
echo "Person added\n";
$result = db2_exec($conn,
                   "SELECT SYSIBM.IDENTITY_VAL_LOCAL() AS id FROM person");
if (db2_fetch_row($result)) {
    $last = db2_result($result, "ID");
} else {
    echo "Could not retrieve IDENTITY value:\n";
    echo "Message: " . db2_stmt_errormsg();
    echo "SQLSTATE: " . db2_stmt_error();
    die();
}

//==========================================================================
// 4. Select record
echo "Selecting person with id = $last\n";
$stmt = db2_prepare($conn,
                    "SELECT * FROM person WHERE id = ?");
$result = db2_execute($stmt, array($last));
while ($person = db2_fetch_object($stmt)) {
    echo "ID: {$person->ID}, NAME: {$person->NAME}, ";
    echo "BIRTHDATE: {$person->BIRTHDATE}\n";
}

//==========================================================================
// 5. Update record
echo "Updating person with id = $last\n";
$stmt = db2_prepare($conn,
                    "UPDATE person SET name = ? WHERE id = ?");
$result = db2_execute($stmt, array('hugo2', $last));
if (!$result) {
    echo "Could not modify person with id $last.\n";
    echo "Message: " . db2_stmt_errormsg();
    echo "SQLSTATE: " . db2_stmt_error();
    die();
}

//==========================================================================
// 4. Select record
echo "Selecting person with id = $last\n";
$stmt = db2_prepare($conn,
                    "SELECT * FROM person WHERE id = ?");
$result = db2_execute($stmt, array($last));
while ($person = db2_fetch_object($stmt)) {
    echo "ID: {$person->ID}, NAME: {$person->NAME}, ";
    echo "BIRTHDATE: {$person->BIRTHDATE}\n";
}

db2_close($conn);
?>
```

Running the example

Run the example on the command line by issuing the following command:

```
# php db2sample-php-ibm_db2.php
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

To run the example in a Web browser, setup the Apache HTTP Server with PHP support (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site 'Web

2.0 with RHEL5'). Copy the file "db2sample-php-ibm_db2.php" into the Web server directory and set the executable flag by issuing the following commands:

```
# cp db2sample-php-ibm_db2.php /var/www/html
# chmod 755 /var/www/html/db2sample-php-ibm_db2.php
```

Open a Web browser and navigate to http://<server-name>/db2sample-php-ibm_db2.php. The resulting page displays the following content:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

References

The following URLs provides more detailed information:

- PHP IBM DB2 reference: http://php.net/ibm_db2
- Connecting to a DB2 database with PHP (ibm_db2): http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.php.doc/doc/t0023132.htm
- Installing/Configuring ibm_db2: http://php.net/manual/en/ibm-db2.setup.php
- Developing PHP Applications for IBM Data Servers: http://www.redbooks.ibm.com/abstracts/sg247218.html

Chapter 6. Accessing DB2 using Ruby

This chapter introduces several ways to access a DB2 database from applications developed with the Ruby programming language, as shown in Figure 7.

- "Using Ruby with IBM_DB" provides instructions for connecting to a DB2 database using the reference Ruby implementation, commonly called Ruby MRI. The IBM_DB adapter and driver RubyGem package is developed and supported by IBM and includes both a driver providing a direct interface, and an ActiveRecord adapter built on top of that driver.
- "Using JRuby with JDBC" on page 35 details database connectivity for JRuby applications using the Java Database Connectivity API (JDBC) through the DB2 Universal JDBC driver.
- "Using Ruby with ActiveRecord ORM" on page 38 builds on the preceding chapters by introducing ActiveRecord, an object-relational mapper that can be used in both Ruby MRI and JRuby applications utilizing the IBM_DB adapter and driver RubyGem package or DB2 Universal JDBC driver respectively.
- "Using Ruby on Rails with IBM_DB" on page 44 demonstrates DB2 connectivity within Ruby on Rails, a popular web application framework that uses ActiveRecord for database access.

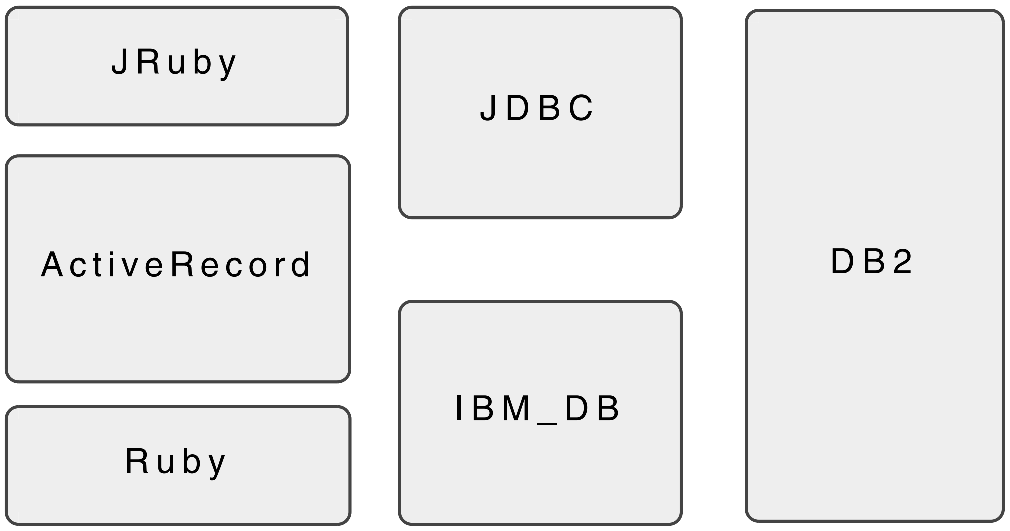


Figure 7. Accessing DB2 using Ruby

There are other ways to connect to a DB2 database from Ruby not covered in this white paper, including:

- rubyodbc (http://www.ch-werner.de/rubyodbc/) provides Unix ODBC bindings for Ruby.
- ruby-db2 (http://rubyforge.org/projects/ruby-dbi/) provides a DB2 adapter for the Ruby DBI project.

Using Ruby with IBM_DB

The IBM_DB adapter and driver RubyGem package provides database access using a relatively low-level API built on the DB2 Call Level Interface (DB2 CLI). It supports multiple connections, prepared statements and stored procedures.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the IBM_DB adapter and driver RubyGem package:

- DB2 Call Level Interface (DB2 CLI) (see “Setting up the DB2 Call Level Interface” on page 3 for details)
- Ruby 1.8.5 or later with RubyGems (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)
- Installation of IBM_DB adapter and driver RubyGem package (as outlined below)

Installation

Before installing the IBM_DB adapter and driver RubyGem package, the environment variables IBM_DB_DIR and IBM_DB_LIB must point to the appropriate directories. The following example assumes that the DB2 instance is named ″db2inst1″:

```
# export IBM_DB_DIR=/home/db2inst1/sqllib
# export IBM_DB_LIB=/home/db2inst1/sqllib/lib
```

The IBM_DB adapter and driver RubyGem package is available on the standard Ruby package repository named RubyForge, and is installed by running the following command:

```
# gem install ibm_db
```

Note: Installing the IBM_DB adapter and driver RubyGem package using RubyGems requires Internet access.

Connection setup

Establishing a connection

This Ruby code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
require 'rubygems'
require 'ibm_db'

conn = IBM_DB::connect(
  "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=SAMPLE;\
   HOSTNAME=db2.example.com;PORT=50000;PROTOCOL=TCPIP;\
   UID=db2inst1;PWD=db2inst1;", "", "")
```

Establishing a secure connection

Secure connections using SSL can be established by adding the SECURITY=SSL keyword to the connection string.

This Ruby code segment establishes a secure TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password that is secured using SSL:

```
require 'rubygems'
require 'ibm_db'

conn = IBM_DB::connect(
  "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=SAMPLE;\
   HOSTNAME=db2.example.com;PORT=40397;PROTOCOL=TCPIP;\
   SECURITY=SSL;UID=db2inst1;PWD=db2inst1;", "", "")
```

Note: CLI must be configured to use the correct certificate to create an SSL connection. See “Basic setup for establishing secure connections” on page 4 for details.

Example

This example will connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

1. "Establish connection" obtains a connection to the DB2 database by calling IBM_DB::connect using the connection string given in “Establishing a connection” on page 32.
2. "Create the PERSON table" uses IBM_DB::exec to run a direct SQL query against the system catalog. The results of that query determine whether the ″PERSON″ table is created. If the IBM_DB::fetch_array method invocation returns nil, indicating that no records were found, the appropriate SQL statements for creating that table are generated.
3. "Insert record" demonstrates the use of prepared statements, through the use of IBM_DB::prepare before IBM_DB::exec, which uses the previously created person array to bind the SQL parameters before processing the statement.
4. "Select record" uses a prepared statement to load a given person by id, traversing through the returned list by calling IBM_DB::fetch_assoc, which returns a hash representation of the database record.
5. "Update record" changes the name attribute of the previously created record through another prepared statement.

Copy the following Ruby code into a new file named “db2sample-ruby-ibm_db.rb”:

```
require 'rubygems'
require 'ibm_db'

##############################################################################
# 1. Establish connection
def connect_to_sample
  if @connection = IBM_DB::connect(
    "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=SAMPLE;\
     HOSTNAME=db2.example.com;PORT=50000;PROTOCOL=TCPIP;\
     UID=db2inst1;PWD=db2inst1;", "", "")
  else
    raise IBM_DB::conn_errormsg
  end
end

##############################################################################
# 2. Create the person table
def create_person_table
  stmt = IBM_DB::exec(@connection,
    "SELECT NAME FROM SYSIBM.SYSTABLES WHERE NAME='PERSON'")
  if IBM_DB::fetch_array(stmt)
    puts "Table person already exists"
  else
```

```
    puts "Creating person table"
    if stmt = IBM_DB::exec(@connection,
      "CREATE TABLE person (
        id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        name VARCHAR(50) NOT NULL,
        birthdate DATE )")
      puts "Table person created"
    else
      raise IBM_DB::stmt_errormsg
    end
  end
end

################################################################################
# 3. Insert record
def insert_person
  person = [ "hugo", "2008-08-14" ]
  puts "Adding person..."
  stmt = IBM_DB::prepare(@connection,
    "SELECT id FROM NEW TABLE (
       INSERT INTO person (name, birthdate) VALUES (?, ?))")
  if IBM_DB::execute(stmt, person)
    puts "Person added"
  else
    raise IBM_DB::stmt_errormsg
  end
  id = IBM_DB::fetch_array(stmt).first.to_s
end

################################################################################
# 4. Select record
def show_person(id)
  puts "Selecting person with id = #{id}"
  stmt = IBM_DB::prepare(@connection, "SELECT * FROM person WHERE id = ?")
  if IBM_DB::execute(stmt, [ id ])
    while row = IBM_DB::fetch_assoc(stmt)
      puts "ID: #{ row['ID'] }, " +
        "NAME: #{ row['NAME'] }, " +
        "BIRTHDATE: #{ row['BIRTHDATE'] }"
    end
  else
    raise IBM_DB::stmt_errormsg
  end
  IBM_DB::free_result(stmt)
end

################################################################################
# 5. Update record
def update_person(id)
  new_name = "hugo2"
  puts "Updating person with id = #{id}"
  stmt = IBM_DB::prepare(@connection, "UPDATE person SET name = ? WHERE id = ?")
  if IBM_DB::execute(stmt, [ new_name, id ])
    puts "Person updated"
  else
    raise IBM_DB::stmt_errormsg
  end
end

################################################################################
connect_to_sample

begin
  create_person_table
  id = insert_person
  show_person(id)
  update_person(id)
  show_person(id)
ensure
  IBM_DB::close(@connection)
end
```

Running the example

Run the example by issuing this command:

```
# ruby db2sample-ruby-ibm_db.rb
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating Person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using JRuby with JDBC

DB2 database connectivity in JRuby is accomplished by using the Java Database Connectivity API (JDBC) together with the DB2 Universal JDBC driver. JDBC can either be used directly, as described in this chapter, or by using a wrapper such as ActiveRecord shown in “Connection setup using the JDBC adapter” on page 41.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using JDBC with JRuby:

- DB2 Universal JDBC driver (see “Setting up the DB2 Universal JDBC driver” on page 7 for details)
- JRuby 1.1.6 or above (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)

Connection Setup

Connection prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This will set the class path environment variable accordingly to include the DB2 Universal JDBC driver.

Establishing a connection

Obtaining a connection to a DB2 database requires two steps:

1. The JDBC driver must be instantiated:

```
java::lang::Class.for_name("com.ibm.db2.jcc.DB2Driver").new_instance
```

2. The connection to the DB2 database is obtained by invoking the DriverManager.get_connection method:

```
java::sql::DriverManager.get_connection(url, user, pass)
```

This example Ruby code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
include Java

url = 'jdbc:db2://db2.example.com:50000/SAMPLE'
props = java.util.Properties.new
props.set_property 'user', 'db2inst1'
props.set_property 'password', 'db2inst1'

driver = com.ibm.db2.jcc.DB2Driver
conn = driver.new.connect(url, props)
```

Establishing a secure connection

This example Ruby code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
include Java

url = 'jdbc:db2://db2.example.com:40397/SAMPLE;sslConnection=true'
props = java.util.Properties.new
props.set_property 'user', 'db2inst1'
props.set_property 'password', 'db2inst1'

driver = com.ibm.db2.jcc.DB2Driver
conn = driver.new.connect(url, props)
```

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

This example will connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

1. "Establish connection" uses the connection setup shown in “Establishing a connection” on page 35 to obtain a connection handle conn to the "SAMPLE" database on "db2.example.com", port 50000.
2. "Create the PERSON table" uses the connection handle method create_statement to instantiate a statement handle stmt that is then used to issue the necessary SQL statements for creating the ″PERSON″ table through the execute method. The ″PERSON″ table might already exist if other examples in this white paper have been run, leading to an SQLException being thrown by the JDBC driver.
3. "Insert record" reuses the statement handler stmt created before to insert a new record into the ″PERSON″ table, retrieving the automatically generated primary key into id by invoking the generated_keys method.

4. "Select record" retrieves the generated record using a prepared statement created by invoking prepareStatement on the connection handle, binding the primary key of the record stored in id to the placeholder using set_int, processing the query and looping through the result set.
5. "Update record" changes the name attribute of the previously created record to new_name using a prepared statement.

Copy the following Ruby code into a new file named “db2sample-ruby-jdbc.rb”:

```
include Java
import java.sql.Statement

url = 'jdbc:db2://db2.example.com:50000/SAMPLE'
props = java.util.Properties.new
props.set_property 'user', 'db2inst1'
props.set_property 'password', 'db2inst1'

################################################################################
# 1. Establish connection
driver = com.ibm.db2.jcc.DB2Driver
conn = driver.new.connect(url, props)

################################################################################
# 2. Create the person table
stmt = conn.create_statement
begin
  puts "Creating new person table..."
  stmt.execute("CREATE TABLE person ( " +
    "id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY," +
    "name VARCHAR(50) NOT NULL," +
    "birthdate DATE )")
  puts "Person table created"
rescue java::sql::SQLException
  puts "Could not create person table - already exists"
end

################################################################################
# 3. Insert record
puts "Adding person..."
stmt.execute("INSERT INTO person (name, birthdate) " +
  "VALUES ('hugo', '2008-08-14')", Statement::RETURN_GENERATED_KEYS)
results = stmt.generated_keys
results.next
puts "Person added"
id = results.get_int(1)

################################################################################
# 4. Select record
puts "Selecting Person with id = #{ id }"
select_person = conn.prepareStatement("SELECT * FROM person WHERE id = ?")
select_person.set_int(1, id)
results = select_person.execute_query
while (results.next) do
  puts "ID: #{ results.int("id") }, " +
    "NAME: #{ results.string("name") }, " +
    "BIRTHDATE: #{ results.date("birthdate") }"
end

################################################################################
# 5. Update record
new_name = "hugo2"
puts "Updating Person with id = #{ id }"
update_person = conn.prepareStatement(
  "UPDATE person SET name = ? WHERE id = ?")
update_person.set_string(1, new_name)
update_person.set_int(2, id);
update_person.execute_update
```

```
################################################################################
# 4. Select record
puts "Selecting Person with id = #{ id }"
results = select_person.execute_query
while (results.next) do
  puts "ID: #{ results.int("id") }, " +
    "NAME: #{ results.string("name") }, " +
    "BIRTHDATE: #{ results.date("birthdate") }"
end

# Connection cleanup
conn.close
```

Running the example

Before running this example, ensure that the Java class path includes the IBM DB2 Universal JDBC Driver as shown in "Connection prerequisites" on page 35.

Run the example by issuing this command:

```
# jruby db2sample-ruby-jdbc.rb
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating Person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using Ruby with ActiveRecord ORM

ActiveRecord is an object-relational mapper (ORM) that provides a high-level object-oriented database abstraction layer by mapping tables to model classes with columns represented as attributes that are automatically typecast between their database representation and the respective Ruby class. This approach allows for database vendor independence and helps writing clean, maintainable code. ActiveRecord is an integral part of the Ruby on Rails framework.

Database connectivity is accomplished by using adapters which interface between low-level drivers and ActiveRecord itself. Access to DB2 is possible through the IBM_DB adapter and driver RubyGem package or by using the DB2 Universal JDBC driver in conjunction with JRuby.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the ActiveRecord ORM with Ruby:

Using ActiveRecord ORM with IBM_DB

- Ruby IBM_DB gem (see "Using Ruby with IBM_DB" on page 31 for details)

Using ActiveRecord ORM with JDBC

- DB2 Universal JDBC driver (see “Setting up the DB2 Universal JDBC driver” on page 7 for details)
- JRuby 1.1.6 or above (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)
- Installation of ActiveRecord, ActiveSupport and the ActiveRecord JDBC adapter RubyGem packages (as outlined below)

Connection setup using the IBM_DB adapter

Establishing a connection

This Ruby code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
require 'rubygems'
require 'activerecord'

ActiveRecord::Base.establish_connection(
  :adapter => 'ibm_db',
  :host => 'db2.example.com',
  :port => 50000,
  :database => 'SAMPLE',
  :username => 'db2inst1',
  :password => 'db2inst1',
  :schema => 'db2inst1')
```

Establishing a secure connection

Secure connections using SSL can be established by adding the security key with value SSL to the connection hash.

This Ruby code segment establishes a secure TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
require 'rubygems'
require 'activerecord'

ActiveRecord::Base.establish_connection(
  :adapter => 'ibm_db',
  :host => 'db2.example.com',
  :port => 40397,
  :database => 'SAMPLE',
  :username => 'db2inst1',
  :password => 'db2inst1',
  :security => 'SSL')
```

Note: CLI must be configured to use the correct certificate to create an SSL connection. See “Basic setup for establishing secure connections” on page 4 for details.

Example

This example uses the IBM_DB adapter and driver RubyGem package to connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

1. "Declare Person class": The Person class represents the ″PERSON″ table. Columns of that table (id, name and birthdate) are mapped automatically to instance attributes by ActiveRecord::Base by querying table metadata.

2. "Establish connection" uses the connection setup shown in “Establishing a connection” on page 39 to obtain a database connection to the "SAMPLE" DB2 database on "db2.example.com", port 5000.
3. "Create the PERSON table" uses the ruby database migration method create_table, which produces the SQL statements necessary for creating the table in a database-independent way. Note that this method automatically adds an id column as primary key.
4. "Insert record" uses Person.create! to create a new person object as well as a database record with the person's attributes supplied in a hash, and stores that record into the person variable.
5. "Select record" retrieves that record through the Person.find static method by supplying the primary key value of the previously created person.
6. "Update record" changes the name attribute of the person object and stores the modified record in the ″PERSON″ table by calling ’person.save!’.
7. "Show record" prints the modified record to the console.

Copy the following code into a file named “db2sample-ruby-activerecord-ibm_db.rb”:

```
require 'rubygems'
require 'activerecord'

###############################################################################
# 1. Declare Person class
class Person < ActiveRecord::Base
  def to_s
    "ID: #{id}, NAME: #{name}, BIRTHDATE: #{birthdate.to_s(:date)}"
  end
end

###############################################################################
# 2. Establish connection
ActiveRecord::Base.establish_connection(
  :adapter => 'ibm_db',
  :host => 'db2.example.com',
  :port => 50000,
  :database => 'SAMPLE',
  :username => 'db2inst1',
  :password => 'db2inst1',
  :schema => 'db2inst1')
ActiveRecord::Base.pluralize_table_names = false

###############################################################################
# 3. Create the person table
ActiveRecord::Schema.define do
  create_table :person, :force => true do |t|
    t.string :name, :limit => 50
    t.date :birthdate
  end
end unless Person.table_exists?

###############################################################################
# 4. Insert record
puts "Adding person..."
person = Person.create!({
  :name => "hugo",
  :birthdate => Date.parse("2008-08-14") })
puts "Person added"

###############################################################################
# 5. Select record
puts "Selecting person with id = #{person.id}"
puts Person.find(person.id)

###############################################################################
```

```
# 6. Update record
puts "Updating person with id = #{person.id}"
person.name = "hugo2"
person.save!

################################################################################
# 7. Show record
puts person
```

Running the example

Run the example by issuing the following command:

```
# ruby db2sample-ruby-activerecord-ibm_db.rb
```

Running the example produces the following output:

```
-- create_table(:person, {:force=>true})
   -> 0.0488s
Adding person...
Person added
Selecting person with id = 100
ID: 100, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 100
ID: 100, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Connection setup using the JDBC adapter

Installation

ActiveRecord, ActiveSupport and the ActiveRecord JDBC adapter must be installed:

```
# jruby -S gem install activesupport activerecord activerecord-jdbc-adapter
```

Note: Installing ActiveRecord, ActiveSupport and the ActiveRecord JDBC adapter using the above command requires Internet access.

Connection prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This will set the class path environment variable accordingly to include the DB2 Universal JDBC driver.

Establishing a connection

This Ruby code segment obtains a connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password, using the DB2 Universal JDBC driver:

```
require 'rubygems'
require 'activerecord'

ActiveRecord::Base.establish_connection(
  :adapter => 'jdbc',
  :driver => 'com.ibm.db2.jcc.DB2Driver',
  :url => "jdbc:db2://db2.example.com:50000/SAMPLE",
  :username => 'db2inst1',
  :password => 'db2inst1'
)
```

Establishing a secure connection

This Ruby code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
require 'rubygems'
require 'activerecord'

ActiveRecord::Base.establish_connection(
  :adapter => 'jdbc',
  :driver => 'com.ibm.db2.jcc.DB2Driver',
  :url => "jdbc:db2://db2.example.com:40397/SAMPLE:sslConnection=true;",
  :username => 'db2inst1',
  :password => 'db2inst1'
)
```

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

This example uses the DB2 Universal JDBC driver to connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated, and displayed again:

1. "Declare Person class": The Person class represents the ″PERSON″ table. Columns of that table (id, name and birthdate) are mapped automatically to instance attributes by ActiveRecord::Base by querying table metadata.
2. "Establish connection" uses the connection setup shown in “Establishing a connection” on page 39 to obtain a database connection to the "SAMPLE" DB2 database on "db2.example.com", port 5000.
3. "Create the PERSON table" uses the ruby database migration method create_table which produces the SQL statements necessary for creating the table in a database-independent way. Note that this method automatically adds an id column as primary key.
4. "Insert record" uses Person.create! to create a new person object as well as a database record with the persons' attributes supplied in a hash, and stores that record into the person variable.
5. "Select record" retrieves that record through the Person.find static method by supplying the primary key value of the previously created person.
6. "Update record" changes the name attribute of the person object and stores the modified record in the ″PERSON″ table by calling ’person.save!’.
7. "Show record" prints the modified record to the console.

Copy the following code into a file named “db2sample-ruby-activerecord-jdbc.rb”:

```
require 'rubygems'
require 'activerecord'

#####################################################################
# 1. Declare Person class
class Person < ActiveRecord::Base
  def to_s
    "ID: #{id}, NAME: #{name}, BIRTHDATE: #{birthdate.to_s(:date)}"
  end
end

#####################################################################
# 2. Establish connection
ActiveRecord::Base.establish_connection(
  :adapter => 'jdbc',
  :driver => 'com.ibm.db2.jcc.DB2Driver',
  :url => "jdbc:db2://db2.example.com:50000/SAMPLE",
  :username => 'db2inst1',
  :password => 'db2inst1'
)
ActiveRecord::Base.pluralize_table_names = false

#####################################################################
# 3. Create the person table
ActiveRecord::Schema.define do
  create_table :person, :force => true do |t|
    t.string :name, :limit => 50
    t.date :birthdate
  end
end unless Person.table_exists?

#####################################################################
# 4. Insert record
puts "Adding person..."
person = Person.create!({
  :name => "hugo",
  :birthdate => Date.parse("2008-08-14") })
puts "Person added"

#####################################################################
# 5. Select record
puts "Selecting person with id = #{person.id}"
puts Person.find(person.id)

#####################################################################
# 6. Update record
puts "Updating person with id = #{person.id}"
person.name = "hugo2"
person.save!

#####################################################################
# 7. Show record
puts person
```

Running the example

Run the example by entering the following command:

```
# jruby db2sample-ruby-activerecord-jdbc.rb
```

Running the example produces the following output:

```
-- create_table(:person, {:force=>true})
   -> 0.0552s
   -> 0 rows
Adding person...
Person added
Selecting person with id = 42
ID: 42, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 42
ID: 42, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using Ruby on Rails with IBM_DB

The following example demonstrates DB2 connectivity within the Ruby on Rails framework using the IBM_DB adapter and driver RubyGem package.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the Ruby on Rails framework in conjunction with the IBM_DB adapter and driver RubyGem package:

- IBM_DB adapter and driver RubyGem package (see “Using Ruby with IBM_DB” on page 31 for details)
- Ruby on Rails (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)

Example

This example creates a new Ruby on Rails project, configures the database access, creates the scaffold for the Person model and finally creates the table by using the migration process:

Create the Ruby on Rails project

Create a new Ruby on Rails project named "db2sample-ruby-rails" and change the working directory to that project root directory:

```
# rails db2sample-ruby-rails
# cd db2sample-ruby-rails
```

Configure the database access

Database configuration for a Rails application is encapsulated in “config/database.yml”, which allows each environment (development, test and production) to use a different database. These definitions are specified as attribute-value pairs in YAML format that is used by ActiveRecord to establish the connection. For this example, change the database configuration in “config/database.yml” so that the development section reads:

```
development:
  adapter: ibm_db
  database: SAMPLE
  host: db2.example.com
  port: 50000
  username: db2inst1
  password: db2inst1
  schema: db2inst1
```

Create the Person scaffold

The scaffold command will automatically create a Person model based on ActiveRecord, which provides access to the people database table along with a migration. This migration adds the people table to the database with name and date of birth columns, a people controller with create, read, update and delete actions and views for these actions as well as unit and functional tests for both model and controller:

```
# script/generate scaffold Person name:string birthdate:date
```

Run the migration process

Run the migration created by the scaffold command to create the people table in the database:

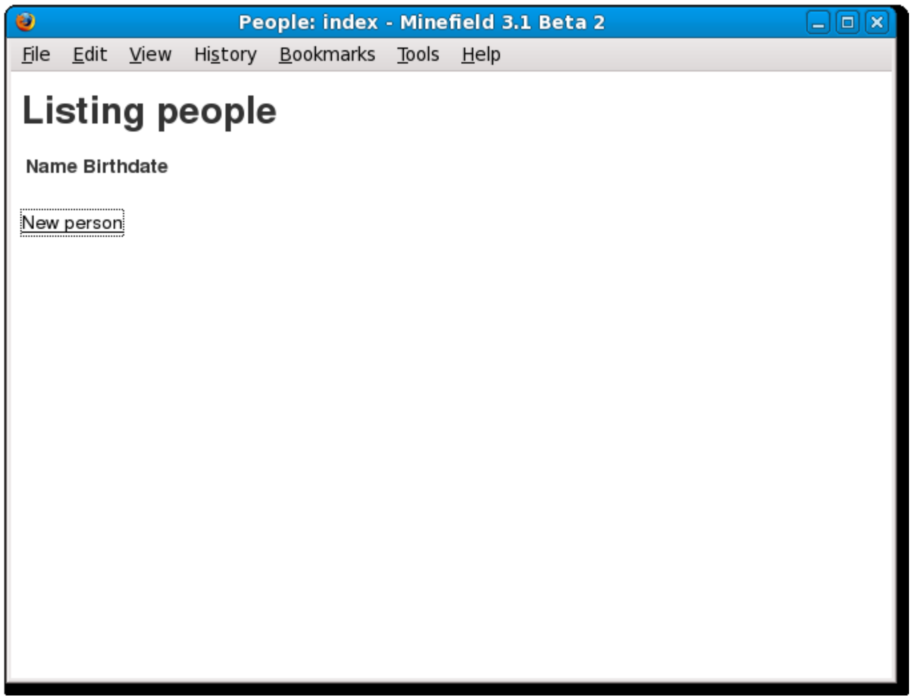
```
# rake db:migrate
```

Running the example

To run the Ruby on Rails example, start the development server on port 3000 by issuing this command:

```
# script/server
```

Open a Web browser and navigate to http://<server-name>:3000/people. This displays a screen similar to the one shown in Figure 8:


People: index - Minefield 3.1 Beta 2
File Edit View History Bookmarks Tools Help

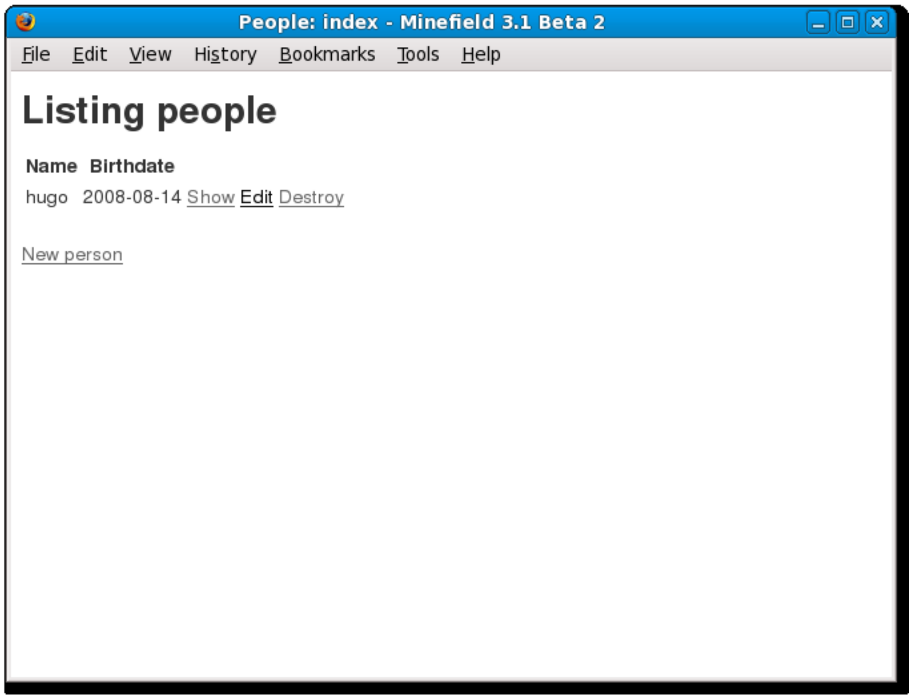
Listing people

Name Birthdate

New person

Figure 8. Rails welcome screen

Clicking on **New Person** enables the user to create a new record. After doing so, the people page looks similar to the one shown in Figure 9 on page 46.


People: index - Minefield 3.1 Beta 2
File Edit View History Bookmarks Tools Help

Listing people

Name	Birthdate			
hugo	2008-08-14	Show	Edit	Destroy

New person

Figure 9. Rails people list

References

The following URLs provides more detailed information:

- IBM_DB RubyForge home page: http://rubyforge.org/projects/rubyibm/
- IBM_DB driver API documentation: http://rubyibm.rubyforge.org/docs/driver/0.9.0/rdoc/
- IBM_DB driver tutorial: http://antoniocangiano.com/2008/02/08/essential-guide-to-the-ruby-driver-for-db2/
- IBM_DB ActiveRecord adapter tutorial: http://rubyforge.org/docman/view.php/2361/7682/IBM_DB_GEM.pdf
- DB2 on Rails: http://db2onrails.com
- JRuby home page: http://www.jruby.org
- java.sql: http://java.sun.com/javase/6/docs/api/java/sql/package-frame.html
- Properties for the IBM DB2 Driver for JDBC and SQLJ: http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.apdv.java.doc/doc/rjvdsprp.htm

Chapter 7. Accessing DB2 using Java

This chapter introduces two ways to access a DB2 database from applications developed in the Java programming language:

- "Using Java with JDBC" provides instructions for connecting to a DB2 database using the Java Database Connectivity API (JDBC).
- "Using Java with Hibernate" on page 50 details JPA, the Java Persistence API, based on the Hibernate implementation.

Using Java with JDBC

The Java Database Connectivity API (JDBC) is part of the default Java installation. It is a common API for accessing databases from different vendors.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using the DB2 Universal JDBC driver along with the JDBC API:

- DB2 Universal JDBC driver (see "Setting up the DB2 Universal JDBC driver" on page 7 for details)

Connection setup

Connection prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This will set the class path environment variable accordingly to include the DB2 Universal JDBC driver.

Establishing a connection

Obtaining a connection to a DB2 database consists of these two steps:

1. The JDBC driver must be instantiated:

   ```
   Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
   ```

2. The connection to the DB2 database must be opened by calling DriverManager.getConnection:

   ```
   Connection conn = DriverManager.getConnection(url, user, pass);
   ```

This Java code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
String url = "jdbc:db2://db2.example.com:50000/SAMPLE";
String user = "db2inst1";
String pass = "db2inst1";
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection conn = DriverManager.getConnection(url, user, pass);
```

Establishing a secure connection

This Java code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
String url = "jdbc:db2://db2.example.com:40397/SAMPLE:sslConnection=true;";
String user = "db2inst1";
String pass = "db2inst1";

Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection conn = DriverManager.getConnection(url, user, pass);
```

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

This example connects to the "SAMPLE" DB2 database and creates a table named "PERSON", unless that table already exists. After that, a record is inserted into the "PERSON" table, and then the record is displayed, updated, and displayed again:

1. "Establish connection" uses the connection setup shown in “Establishing a connection” on page 47 to obtain a connection handle conn to the "SAMPLE" database on "db2.example.com", port 50000.
2. "Create the PERSON table" uses the connection handle method createStatement to instantiate a statement handle stmt, which is then used to issue the necessary SQL statements for creating the ″PERSON″ table through the execute method. The ″PERSON″ table might already exist if other examples in this white paper have been run, leading to an SQLException being thrown by the JDBC driver.
3. "Insert record" reuses the statement handler stmt created before to insert a new record into the ″PERSON″ table, retrieving the automatically generated primary key into the variable id by invoking the getGeneratedKeys method.
4. "Select record" retrieves the generated record using a prepared statement created by invoking prepareStatement on the connection handle, binding the primary key of the record stored in id to the placeholder using setInt, processing the query and looping through the result set.
5. "Update record" changes the name attribute of the previously created record to newName using a prepared statement.

Create a directory tree for the package db2.jdbc.sample and change into that directory:

```
# mkdir -p db2/jdbc/sample
# cd db2/jdbc/sample
```

Copy the following code into a new file named “db2/jdbc/sample/DB2SampleJDBC.java”:

```
package db2.jdbc.sample;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
public class DB2SampleJDBC {
   public static void main(String[] args) throws Exception {
      // ######################################################################
      // 1. Establish connection
      String url = "jdbc:db2://db2.example.com:50000/SAMPLE";
      String user = "db2inst1";
      String pass = "db2inst1";

      Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
      Connection conn = DriverManager.getConnection(url, user, pass);

      // ######################################################################
      // 2. Create the person table
      Statement stmt = conn.createStatement();
      ResultSet rs;

      try {
         System.out.println("Creating new person table...");
         stmt.execute("CREATE TABLE person ( "
            + "id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,"
            + "name VARCHAR(50) NOT NULL," + "birthdate DATE )");
         System.out.println("Person table created");
      } catch (SQLException e) {
         System.err.println("Person table not created - might already exist!");
      }

      // ######################################################################
      // 3. Insert record
      System.out.println("Adding person...");
      stmt.execute("INSERT INTO person (name, birthdate) "
         + "VALUES ('hugo', '2008-08-14')",
         Statement.RETURN_GENERATED_KEYS);
      rs = stmt.getGeneratedKeys();
      rs.next(); // Move cursor into first row
      int id = rs.getInt(1);
      System.out.println("Person added");

      // ######################################################################
      // 4. Select record
      System.out.println("Selecting person with id = " + id);
      PreparedStatement selectPerson = conn
         .prepareStatement("SELECT * FROM person WHERE id = ?");
      selectPerson.setInt(1, id);
      rs = selectPerson.executeQuery();
      while (rs.next()) {
         System.out.println("ID: " + rs.getInt("id") + ", " + "NAME: "
            + rs.getString("name") + ", " + "BIRTHDATE: "
            + rs.getDate("birthdate"));
      }

      // ######################################################################
      // 5. Update record
      String newName = "hugo2";
      System.out.println("Updating person with id = " + id);
      PreparedStatement updatePerson = conn
         .prepareStatement("UPDATE person SET name = ? WHERE id = ?");
      updatePerson.setString(1, newName);
      updatePerson.setInt(2, id);
      updatePerson.executeUpdate();

      // ######################################################################
      // 4. Select record
      System.out.println("Selecting person with id = " + id);
      rs = selectPerson.executeQuery();
      while (rs.next()) {
         System.out.println("ID: " + rs.getInt("id") + ", " + "NAME: "
            + rs.getString("name") + ", " + "BIRTHDATE: "
```

```
                + rs.getDate("birthdate"));
            }
        }
    }
```

Running the example

Before running this example, ensure that the Java class path includes the DB2 Universal JDBC driver as shown in “Connection prerequisites” on page 47.

To compile the “DB2SampleJDBC.java” file, first update the class path variable to include the current directory and then run the compile:

```
# javac db2/jdbc/sample/DB2SampleJDBC.java
```

Run the example by running this command:

```
# java db2/jdbc/sample/DB2SampleJDBC
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using Java with Hibernate

Hibernate is an Open-Source object-relational mapper (ORM) for Java which implements the Java Persistence API.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using Hibernate:

- DB2 Universal JDBC driver (see “Setting up the DB2 Universal JDBC driver” on page 7 for details)
- Installation of Hibernate (as outlined below)

Installation

To install Hibernate on Red Hat Enterprise Linux 5.2, process the following steps:

1. Create a new directory to store the Hibernate components:

   ```
   # mkdir -p /opt/hibernate
   ```

2. Download the Hibernate Core package from http://www.hibernate.com.

   ```
   # wget -c http://downloads.sourceforge.net/hibernate/hibernate-distribution-3.3.1.GA-dist.tar.gz
   ```

Extract the TAR file and change into newly created directory:

```
# tar xzf hibernate-distribution-3.3.1.GA-dist.tar.gz
# cd hibernate-distribution-3.3.1.GA/
```

Copy these files into the directory "/opt/hibernate":

```
# cp hibernate3.jar /opt/hibernate
# cp lib/required/*.jar /opt/hibernate
```

Change into the previous directory:

```
# cd ..
```

3. Download the Hibernate Annotations package from http://www.hibernate.org:

```
# wget -c http://downloads.sourceforge.net/hibernate/hibernate-annotations-3.4.0.GA.tar.gz
```

Extract the TAR file and change into newly created directory:

```
# tar xzf hibernate-annotations-3.4.0.GA.tar.gz
# cd hibernate-annotations-3.4.0.GA/
```

Copy these files into the directory "/opt/hibernate":

```
# cp hibernate-annotations.jar /opt/hibernate
# cp lib/hibernate-commons-annotations.jar /opt/hibernate
# cp lib/hibernate-core.jar /opt/hibernate
# cp lib/ejb3-persistence.jar /opt/hibernate
```

Change into the previous directory:

```
# cd ..
```

4. Download the Simple Logging Facade for Java (SLF4J) from http://www.slf4j.org:

```
# wget -c http://www.slf4j.org/dist/slf4j-1.5.2.tar.gz
```

Note: The Simple Logging Facade must meet the version number of the slf4j-api JAR file which is part of the Hibernate Core package.

Extract the TAR file and change into newly created directory:

```
# tar xzf slf4j-1.5.2.tar.gz
# cd slf4j-1.5.2
```

Copy these files into the directory "/opt/hibernate":

```
# cp slf4j-jdk14-1.5.2.jar /opt/hibernate
```

Change into the previous directory:

```
# cd ..
```

Connection setup

Connection prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This will set the class path environment variable accordingly to include the DB2 Universal JDBC driver.

Establishing a connection

In the Hibernate documentation, an example is available that shows the Hibernate configuration file for accessing an HSQL database:

`http://www.hibernate.org/hib_docs/v3/reference/en-US/html_single/#tutorial-firstapp-configuration`

Use the following properties to establish a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

Property Name	Property Value
hibernate.dialect	org.hibernate.dialect.DB2Dialect
hibernate.connection.driver_class	com.ibm.db2.jcc.DB2Driver
hibernate.connection.url	jdbc:db2://db2.example.com:50000/SAMPLE
hibernate.connection.username	db2inst1
hibernate.connection.password	db2inst1

Establishing a secure connection

In the Hibernate documentation, an example is available that shows the Hibernate configuration file for accessing an HSQL database:

`http://www.hibernate.org/hib_docs/v3/reference/en-US/html_single/#tutorial-firstapp-configuration`

Use the following properties to establish a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

Property Name	Property Value
hibernate.dialect	org.hibernate.dialect.DB2Dialect
hibernate.connection.driver_class	com.ibm.db2.jcc.DB2Driver
hibernate.connection.url	jdbc:db2://db2.example.com:50000/ SAMPLE:sslConnection=true;
hibernate.connection.username	db2inst1
hibernate.connection.password	db2inst1

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

This example contains the setup of a database configuration file, the implementation of the Hibernate sessionFactory and the implementation of an entity bean. Finally, the DB2SampleHibernate class is implemented which makes use of the Hibernate API

Create the package

For this example create a new package named “db2.hibernate.sample” by using the following command:

```
# mkdir -p db2/hibernate/sample
```

Create the database configuration file hibernate.cfg.xml

In the Hibernate documentation, an example is available that shows the Hibernate configuration file for accessing an HSQL database:

`http://www.hibernate.org/hib_docs/v3/reference/en-US/html_single/#tutorial-firstapp-configuration`

Get the example as a template and save it to a file named “db2/hibernate/sample/hibernate.cfg.xml”. Use the following properties to establish a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

Property Name	Property Value
hibernate.dialect	org.hibernate.dialect.DB2Dialect
hibernate.connection.driver_class	com.ibm.db2.jcc.DB2Driver
hibernate.connection.url	jdbc:db2://db2.example.com:50000/SAMPLE
hibernate.connection.username	db2inst1
hibernate.connection.password	db2inst1
hbm2ddl.auto	update

Additionally add the mappings for the package "db2.hibernate.sample" and "db2.hibernate.sample.Person":

Mapping Tag	Mapping Value
package	db2.hibernate.sample
class	db2.hibernate.sample.Person

Create the Hibernate session factory HibernateUtil.java

HibernateUtil is a helper class serving as factory for the Hibernate session.

Copy the following code into a file called “db2/hibernate/sample/HibernateUtil.java”:

```
package db2.hibernate.sample;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class HibernateUtil {
 private static final SessionFactory sessionFactory;
 static {
  try {
   sessionFactory = new AnnotationConfiguration().addPackage(
     "db2.hibernate.sample").addAnnotatedClass(Person.class)
     .configure().buildSessionFactory();
  } catch (Throwable ex) {
   // Log exception!
   throw new ExceptionInInitializerError(ex);
  }
 }

 public static Session getSession() throws HibernateException {
  return sessionFactory.openSession();
 }
}
```

Create the entity bean Person.java

Person is the entity bean: its attributes map to the respective table columns using the Java Persistence API (JPA).

Copy the following code into a file called “db2/hibernate/sample/Person.java”:

```
package db2.hibernate.sample;

import java.io.Serializable;
import java.util.GregorianCalendar;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Person implements Serializable {
 Integer id;
 String name;
 GregorianCalendar birthdate;

 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 public Integer getId() {
  return id;
 }

 public void setId(Integer id) {
  this.id = id;
 }

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public GregorianCalendar getBirthdate() {
  return birthdate;
 }

 public void setBirthdate(GregorianCalendar birthdate) {
  this.birthdate = birthdate;
 }
}
```

Create the example implementation DB2SampleHibernate.java

DB2SampleHibernate is the class that is responsible for tying all strings together: After obtaining a session, a table named "PERSON" is created, unless that table already exists. After that, a record is inserted into the "PERSON" table, and then the record is displayed, updated, and displayed again:

1. "Establish connection" obtains a connection to the "SAMPLE" DB2 database by calling the method HibernateUtil.getSession. Also, this method creates the "PERSON" table, if it does not already exists.
2. "Insert record" starts a transaction and inserts the record into the database by using the saveOrUpate method. After that, the transaction is committed.
3. "Show record" retrieves a database record for the primary key id and displays the record.
4. "Update record" starts a transaction and changes the name attribute of the previously created record to newName. After that, the transaction is committed.

Copy the following code into a file called “db2/hibernate/sample/ DB2SampleHibernate.java”:

```
package db2.hibernate.sample;

import java.util.GregorianCalendar;

import org.hibernate.Session;

public class DB2SampleHibernate {

   public static void main(String[] args) {
      int personId;
      // ####################################################################
      // 1. Establish connection
      Session session = HibernateUtil.getSession();

      // ####################################################################
      // 2. Insert record
      session.beginTransaction();
      System.out.println("Adding person...");
      Person person = new Person();
      person.setName("hugo");
      person.setBirthdate(new GregorianCalendar(2008,07,14));
      session.saveOrUpdate(person);
      session.getTransaction().commit();
      personId = person.getId();
      System.out.println("Person added");
      person = null;

      // ####################################################################
      // 3. Show record
      System.out.println("Selecting person with id = " + personId);
      person = (Person) session.get(Person.class, personId);
      System.out.println("ID: " + person.getId() + ", " + "NAME: "
            + person.getName() + ", " + "BIRTHDATE: "
            + person.getBirthdate().getTime());
      person = null;

      // ####################################################################
      // 4. Update record
      session.beginTransaction();
      String newName = "hugo2";
      System.out.println("Updating Person with id = " + personId);
      person = (Person) session.get(Person.class, personId);
      person.setName(newName);
      session.update(person);
      session.getTransaction().commit();
      person = null;
```

```
        // ######################################################################
        // 3. Show record
        System.out.println("Selecting person with id = " + personId);
        person = (Person) session.get(Person.class, personId);
        System.out.println("ID: " + person.getId() + ", " + "NAME: "
              + person.getName() + ", " + "BIRTHDATE: "
              + person.getBirthdate().getTime());
        person = null;

        session.close();
    }

}
```

Running the example

Before running this example, ensure that the Java class path includes the DB2 Universal JDBC driver as shown in “Connection prerequisites” on page 52.

First update the class path variable to include the current directory and the package db2.hibernate.sample:

```
# export CLASSPATH=$CLASSPATH:db2/hibernate/sample
```

Now, all JAR files which have been copied into the directory “/opt/hibernate/” must be added to the class path variable:

```
# export CLASSPATH=$CLASSPATH:/opt/hibernate/antlr-2.7.6.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/commons-collections-3.1.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/dom4j-1.6.1.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/ejb3-persistence.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/hibernate-annotations.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/hibernate-commons-annotations.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/hibernate-core.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/javassist-3.4.GA.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/jta-1.1.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/slf4j-api-1.5.2.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/slf4j-jdk14-1.5.2.jar
# export CLASSPATH=$CLASSPATH:/opt/hibernate/slf4j-log4j12-1.5.2.jar
```

Compile the Java source files by issuing the following command:

```
# javac db2/hibernate/sample/*.java
```

Run the example by issuing this command:

```
# java db2/hibernate/sample/DB2SampleHibernate
```

Note: If Hibernate is used with a secured connection, submit the javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword properties directly to the Java call:

```
# java -Djavax.net.ssl.trustStore=/root/.keystore \
       -Djavax.net.ssl.trustStorePassword=client $JAVA_OPTS \
       db2/hibernate/sample/DB2SampleHibernate
```

Running the example produces the following output:

```
INFO: table not found: Person
INFO: schema update complete
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: Thu Aug 14 00:00:00 CEST 2008
Updating Person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: Thu Aug 14 00:00:00 CEST 2008
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

References

The following URLs provides more detailed information:

- DB2 and Java Database Connectivity (JDBC): http://www.ibm.com/developerworks/data/library/techarticle/0203zikopoulos/0203zikopoulos.html
- Introduction to Java application development for DB2: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.doc/ad/rjvjcdif.htm.
- Installing the IBM DB2 Driver for JDBC and SQLJ: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.doc/ad/rjvjcdif.htm.
- The Hibernate project page: http://www.hibernate.org/

Chapter 8. Accessing DB2 using Groovy

This chapter introduces two ways to access a DB2 database from applications written with the Groovy programming language, as shown in Figure 10. Both of the methods presented in this chapter rely on the Java Database Connectivity API (JDBC):

- "Using Groovy with Groovy.sql" provides instructions for connecting to a DB2 database using Groovy.sql, a wrapper for JDBC that provides database access with a higher integration of Groovy language features.
- "Using Grails with GORM" on page 62 demonstrates DB2 connectivity within Grails, a web application framework that uses Grails ORM (GORM) for database access.

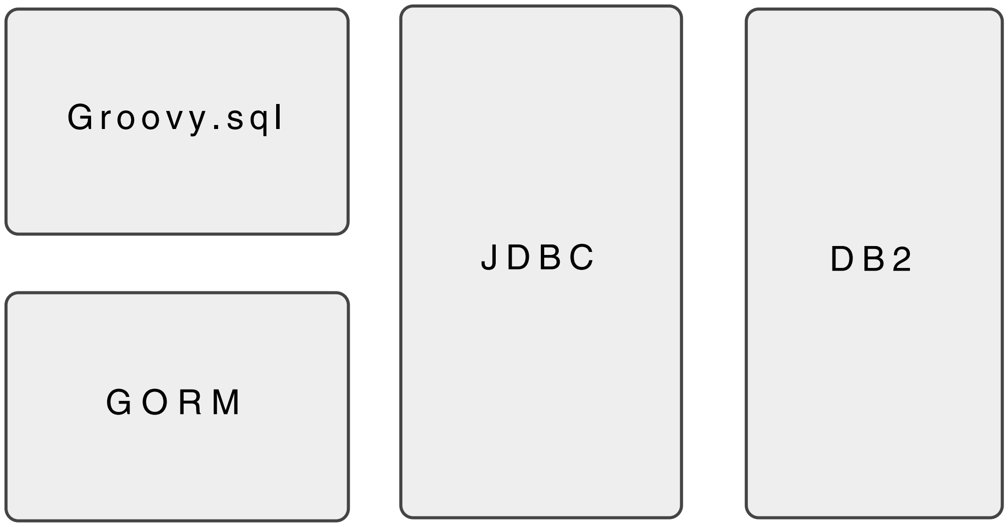


Figure 10. Accessing DB2 using Groovy

There are other ways to connect to a DB2 database from Groovy programs that are not covered in this white paper, including:

- Using JDBC directly.
- GSQL (http://groovy.codehaus.org/GSQL).

Using Groovy with Groovy.sql

The Groovy.sql package is part of the default Groovy installation, and provides a wrapper around the Java Database Connectivity (JDBC) API, introducing Groovy iterators, attribute mapping and convenience methods to simplify database access for Groovy-based applications.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using Groovy.sql:

- DB2 Universal JDBC driver (see "Setting up the DB2 Universal JDBC driver" on page 7 for details)
- Groovy (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site 'Web 2.0 with RHEL5')

Connection setup

Connection Prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This will set the class path environment variable accordingly to include the DB2 Universal JDBC driver.

Establishing a connection

This example Groovy code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
def conn = Sql.newInstance(
    "jdbc:db2://db2.example.com:50000/SAMPLE", // url
    "db2inst1", // username
    "db2inst1", // password
    "com.ibm.db2.jcc.DB2Driver" // driver
)
```

Establishing a secure connection

This example Groovy code segment establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
def conn = Sql.newInstance(
    "jdbc:db2://db2.example.com:40397/SAMPLE;sslConnection=true", // url
    "db2inst1", // username
    "db2inst1", // password
    "com.ibm.db2.jcc.DB2Driver" // driver
)
```

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

This example will connect to the "SAMPLE" DB2 database and create a table named ″PERSON″, unless that table already exists. After that, a record is inserted into the ″PERSON″ table, and then the record is displayed, updated and displayed again:

1. "Establish connection" uses the connection setup shown in “Establishing a connection” to obtain a connection handle conn to the "SAMPLE" database on "db2.example.com", port 50000.
2. "Create the PERSON table" first performs a lookup for the ″PERSON″ table in the DB2 system catalog. If another example in this white paper has already been run, the table created by that example is used. Otherwise the necessary SQL statements for creating the ″PERSON″ table are produced and sent to the database by use of the execute method.

3. "Insert record" uses the executeInsert convenience method to add a new person record. This method call returns the primary key that was automatically generated by the IDENTITY column as the first field of the first row, and stores that value into the id variable.
4. "Select record" uses the id variable created in the previous step to retrieve the added person record using the firstRow method, and prints the resulting data set to the console.
5. "Update record" demonstrates the use of executeUpdate to change the person record's name attribute to the one specified in the newName variable.

Copy the following code into a file named “db2sample.groovy”:

```
import groovy.sql.Sql

// ##########################################################################
// 1. Establish connection
def conn = Sql.newInstance(
   "jdbc:db2://h4245006:50001/SAMPLE", // url
   "db2inst1", // username
   "db2inst1", // password
   "com.ibm.db2.jcc.DB2Driver" // driver
)

// ##########################################################################
// 2. Create the person table
if (!conn.firstRow("SELECT NAME FROM SYSIBM.SYSTABLES WHERE NAME='PERSON'")) {
   println "Creating new person table..."
   conn.execute("""CREATE TABLE person (
      id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
      name VARCHAR(50) NOT NULL,
      birthdate DATE )""")
   println "Person table created"
}

// ##########################################################################
// 3. Insert record
println "Adding person..."
def id = conn.executeInsert("INSERT INTO person (name, birthdate) \
   VALUES ('hugo', DATE('2008-08-14'))")[0][0]
println "Person added"

// ##########################################################################
// 4. Select record
println "Selecting person with id = $id"
def row = conn.firstRow("SELECT * FROM person WHERE id = $id")
println "ID: ${row.id}, NAME: ${row.name}, BIRTHDATE: ${row.birthdate}"

// ##########################################################################
// 5. Update record
def newName = "hugo2"
println "Updating person with id = $id"
conn.executeUpdate("UPDATE person SET name = $newName WHERE id = $id")

// ##########################################################################
// 4. Select record
println "Selecting person with id = $id"
row = conn.firstRow("SELECT * FROM person WHERE id = $id")
println "ID: ${row.id}, NAME: ${row.name}, BIRTHDATE: ${row.birthdate}"

conn.commit()
conn.close()
```

Running the example

Before running the example ensure that the Java class path includes the DB2 Universal JDBC driver as shown in “Connection Prerequisites” on page 60.

Run the example by issuing this command:

```
# groovy db2sample.groovy
```

Running the example produces the following output:

```
Creating new person table...
Person table created
Adding person...
Person added
Selecting person with id = 1
ID: 1, NAME: hugo, BIRTHDATE: 2008-08-14
Updating Person with id = 1
Selecting person with id = 1
ID: 1, NAME: hugo2, BIRTHDATE: 2008-08-14
```

Note: The output might be different than what is displayed here if other examples from this document have previously been run.

Using Grails with GORM

The following example project demonstrates DB2 connectivity within the Grails web application framework using GORM, an object-relational mapper (ORM) based on Hibernate.

Prerequisites

The following requirements must be met in order to connect to a DB2 database using GORM:

- DB2 Universal JDBC driver (see “Setting up the DB2 Universal JDBC driver” on page 7 for details)
- Grails (see *"Setting up a Web 2.0 stack"* on the developerWorks Web site ’Web 2.0 with RHEL5’)

Connection setup

Connection Prerequisites

Establishing connections to a DB2 database requires the DB2 instance profile to be loaded. Run the following command as the user initiating the connection:

```
# . /home/db2inst1/sqllib/db2profile
```

This will set the class path environment variable accordingly to include the DB2 Universal JDBC driver.

Establishing a connection

This code segment from the DataSource configuration file establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 50000, using "db2inst1" as the user name and ″db2inst1″ as the password:

```
dataSource {
 pooled = true
 driverClassName = "com.ibm.db2.jcc.DB2Driver"
 dialect = org.hibernate.dialect.DB2Dialect.class
 username = "db2inst1"
 password = "db2inst1"
}
hibernate {
    cache.use_second_level_cache=true
    cache.use_query_cache=true
    cache.provider_class='com.opensymphony.oscache.hibernate.OSCacheProvider'
}
// environment specific settings
environments {
 development {
  dataSource {
   dbCreate = "update" // one of 'create', 'create-drop','update'
   url = "jdbc:db2://db2.example.com:50000/SAMPLE"
  }
 }
[...]
```

Establishing a secure connection

This code segment from the DataSource configuration file establishes a TCP/IP connection to the "SAMPLE" database on host "db2.example.com" on port 40397, using "db2inst1" as the user name and ″db2inst1″ as the password, which is secured using SSL:

```
dataSource {
 pooled = true
 driverClassName = "com.ibm.db2.jcc.DB2Driver"
 dialect = org.hibernate.dialect.DB2Dialect.class
 username = "db2inst1"
 password = "db2inst1"
}
hibernate {
    cache.use_second_level_cache=true
    cache.use_query_cache=true
    cache.provider_class='com.opensymphony.oscache.hibernate.OSCacheProvider'
}
// environment specific settings
environments {
 development {
  dataSource {
   dbCreate = "update" // one of 'create', 'create-drop','update'
   url = "jdbc:db2://db2.example.com:40397/SAMPLE:sslConnection=true;"
  }
 }
[...]
```

Note: Java must be configured to use the correct certificate to establish an SSL connection. See “Basic setup for establishing secure connections” on page 7 for details.

Example

This example creates a new Grails project, configures the database access, creates the person domain class and the person controller:

Create the Grails project

Create a new Grails project named "db2grails" by issuing the following command:

```
# grails create-app db2sample-grails
```

Change into the newly created project directory:

```
# cd db2sample-grails
```

Configure the database access

Edit the file “grails-app/conf/DataSource.groovy” to use the "SAMPLE" database from this Grails project:

- Change the driverClassName in the data source closure to ″com.ibm.db2.jcc.DB2Driver″.
- Change the dialect in the data source closure to "org.hibernate.dialect.DB2Dialect.class".
- Change the development dataSource dbCreate variable to ″update″, as the "SAMPLE" database already exists.
- In the same closure, change the URL to ″jdbc:db2://db2.example.com:50000/SAMPLE″.

Note: It is certainly possible to use the default HSQL database for development and DB2 for production only.

After these modifications the “grails-app/conf/DataSource.groovy” file should resemble the one given below:

```
dataSource {
 pooled = true
 driverClassName = "com.ibm.db2.jcc.DB2Driver"
 dialect = org.hibernate.dialect.DB2Dialect.class
 username = "db2inst1"
 password = "db2inst1"
}
hibernate {
    cache.use_second_level_cache=true
    cache.use_query_cache=true
    cache.provider_class='com.opensymphony.oscache.hibernate.OSCacheProvider'
}
// environment specific settings
environments {
 development {
  dataSource {
   dbCreate = "update" // one of 'create', 'create-drop','update'
   url = "jdbc:db2://db2.example.com:50000/SAMPLE"
  }
 }
[...]
```

Create the Person domain class

The Person domain class establishes the mapping between person objects and records, where a Person instance corresponds to one row in the ″PERSON″ table with attributes mapped to that rows' column values.

Create the Person domain class by entering the following command:

```
# grails create-domain-class Person
```

Edit the generated class “grails-app/domain/Person.groovy” and add the name and the birthdate attributes:

```
class Person {
 String name
 Date birthdate
}
```

Create the Person controller

The Person controller forms the link between the Person domain objects and views associated with those objects, following the Model-View-Controller (MVC) pattern. In the example PersonController given below the actual implementation of both controller and corresponding views is provided by the scaffold helper that automatically provides create, read, update and delete (CRUD) actions for a domain class.

Finally, create the Person controller:

```
# grails create-controller Person
```

Edit the generated class “grails-app/controllers/PersonController.groovy” and replace its contents with the following code:

```
class PersonController {
 def scaffold = Person
}
```

Running the example

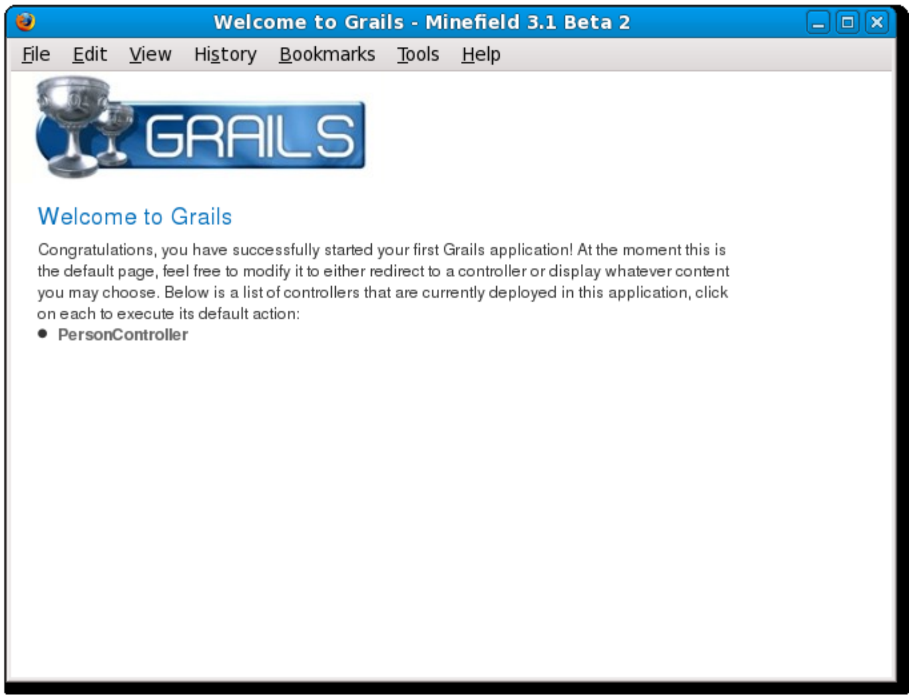
Before running the example ensure that the Java class path includes the DB2 Universal JDBC driver as shown in “Connection Prerequisites” on page 60.

Start the application server by issuing the following command:

```
# grails run-app
```

The run-app command starts an embedded Web server and automatically updates the database schema, creating the ″PERSON″ table if necessary.

Open a Web browser and navigate to http://<server-name>:8080/db2sample-grails. This displays a screen similar to the one shown in Figure 11 on page 66:


Welcome to Grails - Minefield 3.1 Beta 2
File Edit View History Bookmarks Tools Help
GRAILS

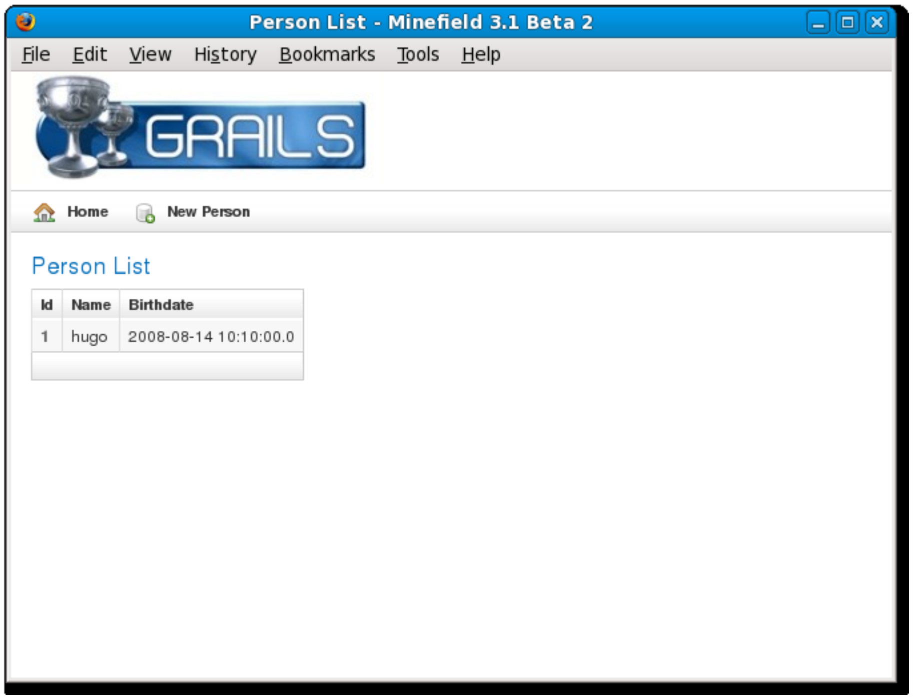
Welcome to Grails

Congratulations, you have successfully started your first Grails application! At the moment this is the default page, feel free to modify it to either redirect to a controller or display whatever content you may choose. Below is a list of controllers that are currently deployed in this application, click on each to execute its default action:

- PersonController

Figure 11. Grails welcome screen

Clicking on **PersonController** shows all entries in the ″PERSON″ table. After adding a person using the **New Person** link the page looks similar to the one shown in Figure 12:


Person List - Minefield 3.1 Beta 2
File Edit View History Bookmarks Tools Help
GRAILS
Home
New Person

Person List

Id	Name	Birthdate
1	hugo	2008-08-14 10:10:00.0

Figure 12. Grails person list

References

The following URLs provides more detailed information:

- Groovy home page: http://groovy.codehaus.org/
- Groovy Database overview: http://groovy.codehaus.org/Database+features
- Practically Groovy: JDBC programming with Groovy: http://www.ibm.com/developerworks/java/library/j-pg01115.html
- Mastering Grails: Build your first Grails application: http://www.ibm.com/developerworks/java/library/j-grails01158
- Patterns of Enterprise Application Architecture - Active Record: http://martinfowler.com/eaaCatalog/activeRecord.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks