



Project no. 004758

GORDA

Open Replication Of Databases

Specific Targeted Research Project

Software and Services

Database Support Description and Configuration Guide

GORDA Deliverable D4.6

Due date of deliverable: 2006/09/30

Actual submission date: 2007/09/29

Revision date: 2008/03/30

Start date of project: 1 October 2004

Duration: 42 Months

Continuent

Revision 1.1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contributors

Nuno Carvalho, U. Lisbon
Emmanuel Cecchet, Continuent
Alfrânio Correia, U. Minho
Rui Oliveira, U. Minho
José Pereira, U. Minho
Luís Rodrigues, U. Lisbon
Luís Soares, U. Minho
Ricardo Vilaça, U. Minho



(C) 2007 GORDA Consortium. Some rights reserved.

This work is licensed under the Attribution-NonCommercial-NoDerivs 2.5 Creative Commons License.
See <http://creativecommons.org/licenses/by-nc-nd/2.5/legalcode> for details.

Abstract

This document describes how to install and configure the GORDA compliant versions of the Apache Derby, PostgreSQL and Sequoia database management systems.

Contents

1	Introduction	3
1.1	Objectives	3
1.2	Relationship with other deliverables	3
2	Apache Derby/G	4
2.1	Introduction	4
2.2	Downloading, installing and configuring Apache Derby	4
2.2.1	Download and extract Derby	4
2.2.2	Download and install Apache Ant 1.6.3 or higher	5
2.2.3	Download and install Java Development Kit (JDK) 1.4.x and 1.3.x	5
2.2.4	Download JDBC 2.0 extension for Java Development Kit release 1.3.x	5
2.2.5	Download Java Cryptography Extension (JCE) version 1.2.2 for Java Development Kit release 1.3.x	6
2.2.6	Create Ant property file	6
2.2.7	Build Derby	7
2.2.8	Configure Derby	8
2.3	Downloading, installing and configuring the Derby/G Toolkit	8
2.3.1	Download and extract Derby/G	8
2.3.2	Install Derby/G	9
2.3.3	Configure Derby/G	10
3	Sequoia	11
3.1	Introduction	11
3.2	Downloading and installing Sequoia	11
3.2.1	Sequoia toolkit	11
3.2.2	Installing a database	12
3.3	Configuring Sequoia	13
3.3.1	MySQL	13
3.3.2	Sequoia toolkit	13
4	PostgreSQL/G	15
4.1	Introduction	15
4.2	Downloading, installing and configuring PostgreSQL/G	15
4.2.1	Download and build PostgreSQL/G toolkit	16
4.2.2	Download and build PostgreSQL	17

4.2.3	Configure PostgreSQL/G and PostgreSQL	17
4.3	Replica configuration	18
5	Using PostgreSQL/G Demos	19

Chapter 1

Introduction

This document describes how to install and configure the packages implemented in the scope of the GORDA project, related to Database Support. This includes the changes made into three different database management systems, namely Apache Derby version 10.2, PostgreSQL version 8.1 and Sequoia version 3.0.

Each of these solutions represents one possible configuration of the GORDA architecture. The solution provided by Apache Derby is totally in-core and allows the replication process to run in the same Java Virtual machine as the Database management System itself. It also allows portability between different operating systems. PostgreSQL/G provides a Hybrid solution of the GORDA architecture. Sequoia is a Middleware solution that wraps the Database management System and replicates client requests. This solution is needed to use the GORDA replication architecture with non-compliant Database management Systems, such as Oracle.

1.1 Objectives

The goal of this document is to provide a detailed guideline on how to build and configure the three GORDA compliant DBMS: Apache Derby/G, PostgreSQL/G and Sequoia.

1.2 Relationship with other deliverables

This deliverable describes and shows how to configure the packages that resulted from the work reported in deliverables D4.3 - In-Core Proof-of-concept, D4.4 - Middleware Proof-of-concept and D4.5 - Hybrid Proof-of-concept. The DBMS described in the previous deliverables and subject of the current document can be used with all protocols described in deliverable D3.3 - Replication modules reference implementation.

Chapter 2

Apache Derby/G

2.1 Introduction

In this chapter we show the necessary steps to configure a GORDA replication system based on Apache Derby (Derby, for short) by means of the Derby/G toolkit. To proceed, one should be familiar with the following tools and concepts:

- Build and deploy Java programs using Ant [2];
- Basic understanding of Java Technology [6].

In the following we provide the information necessary to easily compile, install, and configure such packages and programs as long as it is required to enable Derby/G.

2.2 Downloading, installing and configuring Apache Derby

This section is intended for anyone interested in downloading and building a source distribution of Derby 10.2.2.0, including all its dependencies, and is based on Derby building instructions available at <http://db.apache.org/derby/>.

Derby is designed to work in JDK 1.3 (JDBC 2.0) and JDK 1.4 (JDBC 3.0) environments. In addition Derby's JDBC 2.0 implementation must compile against JDBC 2.0 class definitions and naturally the JDBC 3.0 implementation against JDBC 3.0 class definitions.

2.2.1 Download and extract Derby

1. Download Derby source code to any directory. This directory will be referred to as the `$derby.source` directory in the rest of this section.
2. After downloading the source code to directory `$derby.source`, there should be, at least, the following files and directories:

```
${derby.source}/build.xml  
${derby.source}/java
```



```
${derby.source}/tools
```

2.2.2 Download and install Apache Ant 1.6.3 or higher

1. Download a binary distribution of Ant 1.6.x from <http://ant.apache.org/bindownload.cgi>.
2. Install Ant 1.6.x in any directory. This directory will be referred to as \$ant.dir directory in the rest of this section.
3. Create environment variable ANT_HOME to point to the directory where you have installed Ant:

```
setenv ANT_HOME ${ant.dir} -- On unix(csh)
export ANT_HOME=${ant.dir} -- On unix(ksh)
```

4. Modify the PATH environment variable to include the directory \$ant.dir/bin in its list. This makes the 'ant' command line script available, which will be used to actually perform the build:

```
setenv PATH ${ant.dir}/bin:${PATH} -- On unix(csh)
export PATH=${ant.dir}/bin:${PATH} -- On unix(ksh)
```

2.2.3 Download and install Java Development Kit (JDK) 1.4.x and 1.3.x

1. Download JDK 1.4.x and JDK 1.3.x from <http://java.sun.com/j2se>.
2. Install both in any directories, according to the instructions included with the release. The directories where you have installed JDK 1.4.x and JDK 1.3.x will be referred to as \$jdk14.dir and \$jdk13.dir in the rest of this section.
3. Create environment variable JAVA_HOME to point to the directory where you have installed JDK 1.4.x:

```
setenv JAVA_HOME ${jdk14.dir} -- On unix(csh)
export JAVA_HOME=${jdk14.dir} -- On unix(ksh)
```

2.2.4 Download JDBC 2.0 extension for Java Development Kit release 1.3.x

1. Download JDBC 2.0 (*jdbc2_0-stdext.jar*) from <http://java.sun.com/products/jdbc/download.html> by selecting 'JDBC 2.0 Optional Package Binary'.
2. Save *jdbc2_0-stdext.jar* file in directory \$derby.source/tools/java .

2.2.5 Download Java Cryptography Extension (JCE) version 1.2.2 for Java Development Kit release 1.3.x

1. Download JCE 1.2.2 (*jce-1_2_2.zip*) from <http://java.sun.com/products/jce/index-122.html> by selecting 'Download JCE 1.2.2 software, policy files, and documentation.'. You will be asked to login or register in order to proceed with the download¹.
2. After downloading, unzip the file *jce-1_2_2.zip* to any directory. That directory will now contain documentation files and multiple jar files. Copy only *jce1.2.2/lib/jce1_2_2.jar* file to directory *\$derby.source/tools/java*

2.2.6 Create Ant property file

You will need to create a property file to specify your environment and some of your options. Do the following to specify your environment and options:

1. Determine the directory on your system that corresponds to the 'user.home' system property of the JVM referred to by JAVA_HOME. This directory will be referred to as the \$user.home directory in the rest of this section. In order to determine the correct value of \$user.home, you can do either of the following:

- Run ant diagnostics and look for 'user.home' in the list of System properties:

```
ant -diagnostics
```

- Write and run a small java program that prints the value of the 'user.home' system property, e.g. by including the following line in the program:

```
System.out.println(System.getProperty("user.home"));
```

- On Unix systems, \$user.home is often equivalent to the value of the environment variable \${HOME} or \$home.

2. Create a file called 'ant.properties' in your \$user.home directory and define the following variables in 'ant.properties':

- (a) Define the location of JDK 1.4.X:

```
j14lib=${jdk14.dir}/jre/lib
```

- (b) Define the location of JDK 1.3.X:

```
j13lib=${jdk13.dir}/jre/lib
```

- (c) When set to true (default false), directs Ant to proceed past any build errors:

```
proceed=true
```

¹This package is now only available for users from US or Canada.

- (d) When set to false (default true), no extra asserts or debugging information is included in the class files, making Derby run faster as it generates smaller class files:
- ```
sane=false
```

## 2.2.7 Build Derby

1. Go to directory `${derby.source}`.
2. Run the following commands to build Derby:

```
ant -- build all classes into ${derby.source}/classes
ant testing -- build the Derby test framework and related files
ant buildjars -- build all jars
```

3. Check if the following directories were created:

```
${derby.source}/classes
${derby.source}/jars -- if you have built the jars in last step
```

4. Verify the Derby system info by executing the following command:

```
java -cp ${derby.source}/classes org.apache.derby.tools.sysinfo
```

The output should look like this:

```
----- Java Information -----
Java Version: 1.5.0_08
Java Vendor: Sun Microsystems Inc.
Java home: /usr/lib/jvm/java-1.5.0-sun-1.5.0.08/jre
Java classpath: /home/nunomrc/workspace/derby2-g/classes/
OS name: Linux
OS architecture: i386
OS version: 2.6.17-10-generic
Java user name: nunomrc
Java user home: /home/nunomrc
Java user dir: /home/nunomrc
java.specification.name: Java Platform API Specification
java.specification.version: 1.5
----- Derby Information -----
JRE - JDBC: J2SE 5.0 - JDBC 3.0
[/home/nunomrc/workspace/derby2-g/classes] 10.2.2.0 - (1)

```

5. If the installation was successful, you are ready to use Apache Derby. Please refer to the Derby Developer's Guide [16] and Derby Administration Guide [17] for more information about Apache Derby.

### 2.2.8 Configure Derby

In Derby you can set persistent system-wide properties in a text file called *derby.properties*. The file must be placed in the system directory `DERBY_HOME`, meaning that there should be one *derby.properties* file per system, not per database. In a client/server environment, that directory is on the server.

Derby does not provide *derby.properties*, does not create it automatically, and does not automatically write any properties or values to this file. Instead, you must create, write, and edit this file yourself. The file should be in the format created by the *java.util.Properties.save* method. Only properties set in the following ways have the potential to be dynamic:

- As database-wide properties
- As system-wide properties via a Properties object in the application in which the Derby engine is embedded

For more information about a Derby system and the system directory, see 'Derby System' in the Derby Developer's Guide [16].

## 2.3 Downloading, installing and configuring the Derby/G Toolkit

The Derby/G package provides the Java side of the GAPI implementation in Apache Derby 10.2.2.0. To use this package, you need Apache Derby 10.2.2.0 installed and building correctly (see section 2.2).

### 2.3.1 Download and extract Derby/G

1. Download Derby/G from <http://gorda.di.uminho.pt/community> to any directory.
2. Extract Derby/G using the command:

```
unzip derby-g-<version>.zip
```

3. The directory `derby-g-<version>` created with the extraction operation will be referred to as `$derby.G.source` directory in the rest of this section. This directory contains the Derby/G files including:

**config/** Directory with the configuration files.

**configure** This file configures Derby/G over an original Derby 10.2.2.0 version, on Unix systems.

**derby-g-\$version\$-patch.diff** Patch to apply to original Derby 10.2.2.0 containing Derby/G extensions.

**docs/** Directory with Derby/G documentation including Javadoc.

**java/** Directory with Derby/G source code including reflector, drda, client, and demo code.

**lib/** Directory with the required jars for Derby/G.

### 2.3.2 Install Derby/G

1. Derby/G patch will only be successfully installed if the Derby System is installed and building correctly. In the directory where you have installed Derby you may optionally verify if it is building correctly:

```
ant clean
ant all
```

2. Configure Derby/G using the *configure* script. This script expects an argument with the Derby home directory, from now on referred to as `$DERBY_HOME`:

```
./configure ${DERBY_HOME}
```

3. If you have Apache Derby 10.2.2.0 correctly installed you should now be ready to use the GORDA Derby Reflector. But, if you had any problems using the *configure* script and your Derby installation is working fine, please try the following steps:

- (a) Go to `$DERBY_HOME` to proceed with the configuration:

```
cd ${DERBY_HOME}
```

- (b) Create the directory to place Derby/G files:

```
mkdir -p ./patches/gordaReflection
```

- (c) Copy Derby/G files:

```
cp -r ${derby.G.source}/* ./patches/gordaReflection
```

- (d) Apply the DIFF:

```
cp ${derby.G.source}/derby-g-0.3-patch.diff .
patch -p1 < derby-g-0.3-patch.diff
rm -f derby-g-0.3-patch.diff
```

- (e) Recompile Derby with Derby/G extension:

```
ant clean
ant all
```

4. Derby/G toolkit is now installed. Extra functionalities like demos and reflection clients support are also available and may be built in the Derby/G directory:

```
cd ./patches/gordaReflection
ant all
```

### 2.3.3 Configure Derby/G

Although Derby/G is now correctly installed in Apache Derby 10.2.2.0 it is disabled by default. In order to use Derby/G one will need have the following properties configured in the *derby.properties* file:

```
derby.service.GordaReflection=gorda.reflector.derby.iapi.ReflectionFactory
derby.gorda.plugin=PluginClassName
derby.system.bootAll=true
```

In Derby/G *docs* directory you will find a sample *derby.properties* fully documented about Derby and Derby/G configuration. You can copy it to DERBY\_HOME directory:

```
cp ${derby.G.source}/docs/derby.properties ${DERBY_HOME}
```

# Chapter 3

## Sequoia

### 3.1 Introduction

In this chapter we show the necessary steps to configure a GORDA replication system based on a non-compliant DBMS by means of the Sequoia toolkit. As an example we will use MySQL as the underlying DBMS. To proceed, one should be familiar with the following tools and concepts:

- Install Mysql [13] ;
- Build and install Java programs using Ant [2];
- Basic understanding of Java Technology [6].
- Install and configure Sequoia [15].

Different client applications can use Sequoia providing that such applications access a database through either JDBC 3 or ODBC (For further details on the ODBC see Carob Project [8]).

### 3.2 Downloading and installing Sequoia

#### 3.2.1 Sequoia toolkit

Sequoia toolkit can be downloaded from

- <http://gorda.di.uminho.pt/community>.

Once downloaded, decompress the files by doing:

- `unzip sequoia-3.0-beta2-src.zip`

The sources should be compiled using the Ant tool [2]. To do so, execute the following commands:

- `ant clean`
- `ant compile`

To install Sequoia on Unix, define in which directory the installation should be placed and afterwards copy the files into that directory:

- `SEQUOIA_HOME=/usr/local/sequoia ; export SEQUOIA_HOME`
- `ant install`

One may also use a graphical installation:

- `java -jar sequoia-x.y-bin-installer.jar`

### 3.2.2 Installing a database

Sequoia needs to extract information on updated tuples thus requiring changes on database catalogs. To do so, it exploits the fact that triggers are available in most database systems. Unfortunately, each database has its proprietary language to create triggers and a binding for each database must be provided. Our current prototype has bindings for PostgreSQL and MySQL, but the code to provide the triggers is extremely simple as it only logs information on updated tuples and soon we will have binds to different database systems. Nevertheless, for these two DBMS bindings it is possible to use any of the GORDA implementations of the protocols in deliverables D3.3 - Replication modules reference implementation.

In what follows we quickly present how to install MySQL version 5.X. One should download it from:

- <http://www.mysql.com/>.

The MySQL should be installed as follows:

- `groupadd mysql`
- `useradd -g mysql mysql`
- `cd /usr/local`
- `gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar -xvf`
- `ln -s full-path-to-mysql-VERSION-OS mysql`
- `cd mysql`
- `chown -R mysql .`
- `chgrp -R mysql .`
- `scripts/mysql_install_db --user=mysql`
- `chown -R root .`
- `chown -R mysql data`

Of course, if one already has a MySQL installed and a database working, this section and the next might be skipped.



## 3.3 Configuring Sequoia

### 3.3.1 MySQL

The first step consists on running MySQL as follows:

- `bin/mysqld_safe --user=mysql &`

Then, we should create a database by using the MySQL utility “mysql”:

- `mysql`

Then by using the utility run the following SQL commands:

- `CREATE DATABASE test`
- `create table orders (key int not null default auto increment,  
sum float, description varchar(20),  
primary key (key));`
- `create table order_line (key int not null default auto increment,  
okey int, value float, unit int,  
description varchar(20), primary key (key));`

When you are done, quit the utility by typing:

- `\q`

### 3.3.2 Sequoia toolkit

Copy or create a “controller.xml”, move it to “sequoia/config/controller” and then edit it as follows:

```
<SEQUOIA-CONTROLLER>
 <Controller name="c1" jdbcIpAddress="127.0.0.1"
 jdbcPort="25322">
 <JmxSettings jmxIpAddress="0.0.0.0" jmxPort="1090"/>
 <EscadaReplicator/>
 </Controller>
</SEQUOIA-CONTROLLER>
```

- Include the public identifier in the beginning of the controller configuration file.
- If your controller host uses multiple IP addresses, include the `ipAddress` attribute of the `Controller` element to bind the controller to a specific IP address. We may also configure the port number by changing “`jdbcPort`”.
- Include a `JmxSettings` element. This allows the controller to be administered remotely using the Sequoia Command Line Console (CLC), a JMX client based on the standard RMI connector for JMX. The JMX/RMI IP address/port number is 0.0.0.0:1090 by default.

- Include a EscadaReplicator element. This allows to start the ESCADA replicator.

The next step consists on configure the virtual database by editing the file “backend-distribution.xml”. One controller may manage one or more virtual databases and a virtual database may have one or more database backends which are Database management Systems (DBMSs). In what follows, we present how to configure a backend attached to a virtual database named “GordaDatabase-01”:

```
<DatabaseBackend
 name=?mysql?
 driver=?com.mysql.jdbc.Driver?
 url=?jdbc:mysql://localhost/tpcc"?
 connectionTestStatement=?select 1?
 objectSetImplementation="gorda.db.sequoia.
 utils.objectset.MySQLObjectSet">
</DatabaseBackend>
```

- Specify a unique name for the database backend as the value of the name attribute.
- Specify the class name of your native database JDBC driver as the value of the driver attribute.
- Specify the JDBC URL used to connect to this database backend as the value of the url attribute.
- Specify a value for the connectionTestStatement attribute. This SQL statement is used by the controller after a failed request execution to check if the connection is still valid:
- Specify the class name of the binding for this database that extract information on updated tuples exploiting triggers. For PostgreSQL databases use objectSetImplementation=“gorda.db.sequoia.utils.objectset.PGObjectSet” and add a element <DatabaseSchema gatherSequences=“false”/>.

## Chapter 4

# PostgreSQL/G

### 4.1 Introduction

In this chapter we show the necessary steps to configure a replication system based on PostgreSQL by means of the PostgreSQL/G toolkit. To proceed, one should be familiar with the following tools and concepts:

- Build and deploy PostgreSQL [5] by using GNU Software [3];
- Build and deploy Java software using maven (version 2) [12];
- Basic understanding of Java Technology [6].

PostgreSQL/G has the following dependencies:

- PostgreSQL JDBC Driver [1].
- Log4J [4].
- junit Test [11]
- Bean Scripting Framework [7]
- Commons Beanutils [9]
- OpenJMS Server [14].

### 4.2 Downloading, installing and configuring PostgreSQL/G

Running PostgreSQL/G requires to apply a patch to PostgreSQL, compile and install it along with different components written in Java. The patch and Java components are available as an archive named PostgreSQL/G toolkit.

### 4.2.1 Download and build PostgreSQL/G toolkit

1. Download PostgreSQL/G toolkit from <http://gorda.di.uminho.pt/community> to any directory.
2. Extract PostgreSQL/G toolkit using the following command:
  - `unzip postgresql-g-0.4.zip`
3. The directory `postgresql-g-0.4` created with the extraction operation is referred to as `$gorda` directory in the rest of this document. This directory contains:

<code>csrc</code>	It has a patch for the PostgreSQL's source code.
<code>javasrc</code>	This has the GAPI's source code, its rendering on PostgreSQL and demos:
<code>gorda-interfaces</code>	which has the GAPI's source code;
<code>gorda-postgres</code>	which has the GAPI's rendering on PostgreSQL;
<code>interfaces-demos</code>	which has common files shared by different renderings of the GAPI to run simple demos;
<code>postgres-demos</code>	which has the PostgreSQL/G's code to start up simple demos.

4. Build the java component of the PostgreSQL/G. To do this, you need Apache Maven2 and Java SDK 1.5 installed.
  - `mvn compile`
    - compile (output: target/classes)
  - `mvn package`
    - package (output: target/)
  - `mvn install`
    - install (maven cache)
  - `mvn javadoc:javadoc`
    - documentation (output: ./target/site/apidocs )
  - `mvn clean`
    - clean
5. Change to the sub-directories in the “javasrc”, build and deploy each one with the following command:
  - `mvn clean install`
6. From the extracted directory, define the shell variable `$gorda` as follows:
  - `gorda=`pwd`; export gorda`

## 4.2.2 Download and build PostgreSQL

1. Download the PostgreSQL version 8.1.X from <http://www.postgresql.org/>. It is worth noticing that our current prototype has been only tested on PostgreSQL 8.1.X.
2. From the directory where the PostgreSQL source code was extracted, apply the proper patch from the PostgreSQL/G toolkit as follows:

- `patch -p1 < $gorda/csrc/postgresql-8.1.3-gorda-0.4.diff`

3. Right after, the PostgreSQL should be compiled and installed as follows<sup>1</sup>:

- `./configure --enable-depend --prefix=$gorda/install`
- `make`
- `make install`

## 4.2.3 Configure PostgreSQL/G and PostgreSQL

It is not necessary to dump and restore a database in order to use the PostgreSQL/G. In fact, after installing the patch to PostgreSQL, the database can be used transparently as the patch was never applied. To replicate a database, we should first create it by using the commands provided by the PostgreSQL. Let us assume that a PostgreSQL installation did not exist prior to this guide:

1. Create a database cluster which is a collection of databases managed by a single server instance:

- `$gorda/install/pgsql/bin/initdb -D $gorda/PostgreSQL.data.1`

2. After creating a database cluster, run the PostgreSQL as follows:

- `$gorda/install/pgsql/bin/postmaster -i -p 5432 \`  
`-D $gorda/PostgreSQL.data.1`

3. Change to the directory `src/reflector/scripts` and create a database as follows:

- `$gorda/install/pgsql/bin/createdb test`

4. Create a set of tables to be replicated. To do so, firstly, run the PostgreSQL utility `psql` as follows:

- `$gorda/install/pgsql/bin/psql -d test`

5. Then type the following SQL commands to create tables:

- `create sequence seqorders increment by 1 no minvalue`  
`no maxvalue start with 1 cache 20 no cycle;`
- `create table orders (key int not null default`  
`nextval('seqorders'), sum float, description varchar(20),`  
`primary key (key));`

---

<sup>1</sup>Make sure that the Flex and Bison are installed in your system.

- create sequence seqorderline increment by 1 no minvalue no maxvalue start with 1 cache 20 no cycle;
  - create table order\_line (key int not null default nextval('seqorderline'), okey int, value float, unit int, description varchar(20), primary key (key));
  - grant all on orders to public;
  - grant all on order\_line to public;
  - grant all on seqorders to public;
  - grant all on seqorderline to public;
6. To test the system, create an unprivileged user to update our database as follows:
    - `$gorda/install/pgsql/bin/createuser test --no-superuser \ --no-createdb --no-creatorole`
  7. When you are done, quit the utility by typing:
    - `\quit`

## 4.3 Replica configuration

The next step consists on enabling the database for replication. In order to avoid cluttering the text with unnecessary SQL commands, we use a set of scripts installed by the PostgreSQL's patch and available at its source code directory: `src/reflector/scripts`. There is a configuration file `configs` that should be edited if any of the steps presented were not strictly applied.

1. Change to the directory of the PostgreSQL source code and then to the script directory:
  - `cd src/reflector/scripts`
2. Enable replication on the database test by executing the script:
  - `sh ./enable-db-reflection.sh test`
3. For each table that needs to be replicated, run the script `enable-table -reflection.sh`. In our case, run what follows:
  - `sh ./enable-table-reflection.sh test orders`
  - `sh ./enable-table-reflection.sh test order_line`

To be able to replicate updates on a table, it should have a single primary key.
4. Specify where the replication engine is located. This should be done as follows:
  - `sh ./configure-reflector.sh test localhost 2000 tcp`
5. If one wants to use futures provided by the PL-J, execute the following script:
  - `sh ./enable-plj.sh`

## Chapter 5

# Using PostgreSQL/G Demos

This section briefly describes the demos available and its usefulness to demonstrate the ideas and the simplicity behind the PostgreSQL/G and the GAPI. For a full-fledged replicator see [10].

1. Change to the directory `gorda-postgres-demos` and run:

- `mvn assembly:assembly`

This command builds and installs all demos.

2. Execute a demo:

**AllInOne** This demo accepts mosts of the events specified on the GAPI, prints on the standard output what is going on and immediately allows a database to proceed with its execution. Thus, when a client connects to a PostgreSQL database using a regular driver (e.g. JDBC driver), messages are printed on the standard output showing its execution. To run it, one should use the script `demoAllInOne.sh`.

**Hotback Demo** This demo performs continuous backups to disk, while periodically performing full backups. One should use the script `demoBackup.sh` to run it.

**Naivepb Demo** This demo creates a naive asynchronous primary backup replication. One should use the script `demoPrimary.sh` and `demoBackup.sh`, to start the primary and backup replica, respectively.

**Pubsub Demo** This demo mimics a publisher/subscriber platform where the publisher propagates all updates performed by committed transactions using a JMS Server [14]. Thus right after starting a JMS Server,<sup>1</sup> one can start both the publisher and the subscriber, running `demoPublisher.sh` and `demoSubscriber.sh`, respectively.

**Selftune Demo** This demo implements an admission control on a transaction level. To start it, one should run the following script `demoSelftune.sh`.

3. Run some updates and see their effect on the demos:

- `insert into orders(sum, description) values(1.0,'test-01');`

---

<sup>1</sup>This demo was tested with OpenJMS [14].

- `insert into orders(sum, description) values(5.0,'test-02');`
- `insert into orders(sum, description) values(10.0,'test-03');`
- `insert into order_line(okey,value,unit,description)`  
`values(1,10.0,1,'prod-01');`



# Bibliography

- [1] Postgresql jdbc driver. <http://jdbc.postgresql.org/>.
- [2] Ant. <http://ant.apache.org/>, 2006.
- [3] GNU Software. <http://www.gnu.org/software/software.html>, 2006.
- [4] Log4J. <http://logging.apache.org/log4j/docs/>, 2006.
- [5] PostgreSQL. <http://www.postgresql.org>, 2006.
- [6] The Source for Java Technology. <http://www.java.sun.com>, 2006.
- [7] Bean scripting framework. <http://jakarta.apache.org/bsf/>, 2007.
- [8] Carob project. <http://carob.continuent.org/HomePage>, 2007.
- [9] Commons beanutils. <http://commons.apache.org/beanutils/>, 2007.
- [10] Escada replicator. <http://sourceforge.net/projects/escada/>, 2007.
- [11] Junit. <http://www.junit.org/>, 2007.
- [12] Maven. <http://maven.apache.org/>, 2007.
- [13] Mysql. <http://www.mysql.org/>, 2007.
- [14] Openjms. <http://openjms.sourceforge.net/>, 2007.
- [15] Sequoia. <http://sequoia.continuent.org/HomePage>, 2007.
- [16] The Apache Software Foundation. Derby developer's guide.  
<http://db.apache.org/derby/docs/10.3/devguide/>, 2007.
- [17] The Apache Software Foundation. Derby server and administration guide.  
<http://db.apache.org/derby/docs/10.3/adminguide/>, 2007.