Java DB Tools and Utilities Guide

Version 10.10

Derby Document build:
April 7, 2014, 2:20:42 PM (EDT)

Contents

Copyright.....5

License.....6

Relationship between Java DB and Apache Derby.....10

About this guide.....11
- **Purpose of this document**.....11
- **Audience**.....11
- **How this guide is organized**.....11

What are the Derby tools and utilities?.....13
- **Overview**.....13
  - Environment setup and the Derby tools.....13
- **About Derby databases**.....13
- **JDBC connection basics**.....14
  - JDBC drivers overview.....14
  - Database connection URLs.....14
- **Tools and localization**.....15
  - About locales.....15
  - Database locale.....15
  - Specifying an alternate codeset.....16
  - Formatting display of locale-sensitive data.....16

Using ij.....17
- **Starting ij**.....17
- **Creating a database using ij**.....18
- **Starting ij using properties**.....18
- **Getting started with ij**.....19
  - Connecting to a Derby database.....19
  - Using ij commands.....21
  - Running ij scripts.....21

ij properties reference.....23
- **ij.connection.connectionName property**.....23
- **ij.database property**.....23
- **ij.dataSource property**.....24
- **ij.driver property**.....25
- **ij.exceptionTrace property**.....25
- **ij.maximumDisplayWidth property**.....26
- **ij.outfile property**.....26
- **ij.password property**.....26
- **ij.protocol property**.....27
- **ij.protocol.protocolName property**.....27
- **ij.showErrorCode property**.....27
- **ij.showNoConnectionsAtStart property**.....28
- **ij.showNoCountForSelect property**.....28
- **ij.URLCheck property**.....29
- **ij.user property**.....30
- **derby.ui.codeset property**.....30
- **derby.ui.locale property**.....31

ij commands and errors reference.....33
- **ij commands**.....33
  - Conventions for ij examples.....33

ij SQL command behavior....33
Absolute command....34
After Last command....34
Async command....35
Autocommit command....35
Before First command....36
Close command....36
Commit command....37
Connect command....37
Describe command....38
Disconnect command....38
Driver command....39
Elapsedtime command....40
Execute command....40
Exit command....41
First command....42
Get Cursor command....42
Get Scroll Insensitive Cursor command....43
Help command....45
Last command....45
LocalizedDisplay command....45
MaximumDisplayWidth command....46
Next command....46
Prepare command....47
Previous command....47
Protocol command....48
Readonly command....48
Relative command....49
Remove command....49
Rollback command....50
Run command....50
Set Connection command....51
Show command....51
Wait For command....55
Syntax for comments in ij commands....55
Syntax for identifiers in ij commands....56
Syntax for strings in ij commands....57
ij errors....57
ERROR SQLState....57
WARNING SQLState....58
IJ ERROR....58
IJ WARNING....58
JAVA ERROR....58

sysinfo....59
sysinfo example....59
Using sysinfo to check the classpath....60

dblook....61
Using dblook....61
dblook options....61
Generating the DDL for a database....62
dblook examples....63

SignatureChecker....65
Using SignatureChecker....65

PlanExporter....67

Using PlanExporter.....67
PlanExporter XML format.....68
PlanExporter example.....69
Optional tools.....71
Using the databaseMetaData optional tool.....71
Using the foreignViews optional tool.....71
Trademarks.....73

Copyright

Copyright 2004-2014 The Apache Software Foundation

Copyright 2005, 2014, Oracle and/or its affiliates. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0.

Related information

License

License

The Apache License, Version 2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use,
      reproduction, and distribution as defined by Sections 1 through
      9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized
      by the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under
      common control with that entity. For the purposes of this
      definition, "control" means (i) the power, direct or indirect,
      to cause the direction or management of such entity, whether by
      contract or otherwise, or (ii) ownership of fifty percent (50%)
      or more of the outstanding shares, or (iii) beneficial ownership
      of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making
      modifications, including but not limited to software source code,
      documentation source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or
      Object form, that is based on (or derived from) the Work and
      for which the editorial revisions, annotations, elaborations,
      or other modifications represent, as a whole, an original work
      of authorship.  For the purposes of this License, Derivative
      Works shall not include works that remain separable from, or
      merely link (or bind by name) to the interfaces of, the Work
      and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or
      additions to that Work or Derivative Works thereof, that is
      intentionally submitted to Licensor for inclusion in the Work
      by the copyright owner or by an individual or Legal Entity
      authorized to submit on behalf of the copyright owner. For the
      purposes of this definition,
      "submitted" means any form of electronic, verbal, or written
      communication sent to the Licensor or its representatives,
      including but not limited to communication on electronic mailing
      lists, source code control systems, and issue tracking systems
```

```
    that are managed by, or on behalf of, the Licensor for the
    purpose of discussing and improving the Work, but excluding
    communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a
    Contribution."

    "Contributor" shall mean Licensor and any individual or Legal
    Entity on behalf of whom a Contribution has been received by
    Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions
   of this License, each Contributor hereby grants to You a
   perpetual, worldwide, non-exclusive, no-charge, royalty-free,
   irrevocable copyright license to reproduce, prepare Derivative
   Works of, publicly display, publicly perform, sublicense, and
   distribute the Work and such Derivative Works in Source or
   Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have
   made, use, offer to sell, sell, import, and otherwise transfer
   the Work, where such license applies only to those patent claims
   licensable by such Contributor that are necessarily infringed by
   their Contribution(s) alone or by combination of their
   Contribution(s) with the Work to which such Contribution(s) was
   submitted. If You institute patent litigation against any entity
   (including a cross-claim or counterclaim in a lawsuit) alleging
   that the Work or a Contribution incorporated within the Work
   constitutes direct or contributory patent infringement, then any
   patent licenses granted to You under this License for that Work
   shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute
       must include a readable copy of the attribution notices
       contained within such NOTICE file, excluding those notices
       that do not pertain to any part of the Derivative Works, in
       at least one of the following places: within a NOTICE text
       file distributed as part of the Derivative Works; within the
       Source form or documentation, if provided along with the
       Derivative Works; or, within a display generated by the
       Derivative Works, if and wherever such third-party notices
       normally appear. The contents of the NOTICE file are for
       informational purposes only and do not modify the License.
       You may add Your own attribution notices within Derivative
       Works that You distribute, alongside or as an addendum to
       the NOTICE text from the Work, provided that such additional
       attribution notices cannot be construed as modifying the
       License.

   You may add Your own copyright statement to Your modifications
```

```
   and may provide additional or different license terms and
   conditions for use, reproduction, or distribution of Your
   modifications, or for any such Derivative Works as a whole,
   provided Your use, reproduction, and distribution of the Work
   otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state
   otherwise, any Contribution intentionally submitted for
   inclusion in the Work by You to the Licensor shall be under the
   terms and conditions of this License, without any additional
   terms or conditions.  Notwithstanding the above, nothing herein
   shall supersede or modify the terms of any separate license
   agreement you may have executed with Licensor regarding such
   Contributions.

6. Trademarks. This License does not grant permission to use the
   trade names, trademarks, service marks, or product names of the
   Licensor, except as required for reasonable and customary use
   in describing the origin of the Work and reproducing the content
   of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or
   conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or
   FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
   determining the appropriateness of using or redistributing the
   Work and assume any risks associated with Your exercise of
   permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and
   grossly negligent acts) or agreed to in writing, shall any
   Contributor be liable to You for damages, including any direct,
   indirect, special, incidental, or consequential damages of any
   character arising as a result of this License or out of the use
   or inability to use the Work (including but not limited to
   damages for loss of goodwill, work stoppage, computer failure or
   malfunction, or any and all other commercial damages or losses),
   even if such Contributor has been advised of the possibility of
   such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by
   reason of your accepting any such warranty or additional
   liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.
```

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.  See the License for the specific language governing
permissions and limitations under the License.
```

Relationship between Java DB and Apache Derby

Java DB is a relational database management system that is based on the Java programming language and SQL. Java DB is the Oracle release of the Apache Derby project, the Apache Software Foundation's (ASF) open source relational database project.

The Java DB product includes Derby without any modification whatsoever to the underlying source code.

Because Java DB and Derby have the same functionality, the Java DB documentation refers to the core functionality as Derby.

The Java DB 10.10 documentation is based on the Derby 10.10 documentation. References to "Derby" in the Java DB documentation should be understood as synonyms for "Java DB."

Oracle has made changes to the Apache Derby documentation. This manual is identical to the *Derby Tools and Utilities Guide*, with the following exceptions:

- Oracle has added this topic, "Relationship between Java DB and Apache Derby".
- In the titles of manuals, "Derby" has been changed to "Java DB".
- Path names that refer to locations within a Derby installation have been changed to refer to locations within a Java Development Kit (JDK) installation as needed. Specifically, demos and samples are in a different location within the JDK from their location in a Derby installation.

About this guide

For general information about the Derby documentation, such as a complete list of books, conventions, and further reading, see *Getting Started with Java DB*.

For more information about Derby, visit the Derby website at http://db.apache.org/derby. The website provides pointers to the Derby Wiki and other resources, such as the derby-users mailing list, where you can ask questions about issues not covered in the documentation.

Purpose of this document

This book describes how to use the Derby tools and utilities. The tools and utilities covered in this book include:

- `ij`
- `sysinfo`
- `dblook`
- `SignatureChecker`
- `PlanExporter`

Audience

This book is for:

- Developers, who might use the tools when developing applications
- System administrators, who might use the tools to run scripts that perform administrative tasks such as backups or importing and exporting data
- End-users, who might use one of the tools to run ad-hoc queries against a database

How this guide is organized

This guide includes the following sections:

- What are the Derby tools and utilities?

  Overview of the tools and utilities, and Derby and JDBC basics for new or infrequent users.
- Using ij

  How to get started with `ij`, a JDBC and SQL scripting tool.
- ij properties reference

  Reference for `ij` properties.
- ij commands and errors reference

  Reference for `ij` commands and errors.
- sysinfo

  Reference information on the utility that provides information about your Derby environment.
- dblook

  Reference information for a utility that dumps the DDL of a user-specified database to either a console or a file.
- SignatureChecker

  Reference information for a tool that identifies any SQL functions and procedures in a database that do not follow the SQL Standard argument matching rules.

- PlanExporter

  Reference information for a tool that exports query plan data for further analysis.

What are the Derby tools and utilities?

The Derby tools and utilities are a set of routines supplied with Derby that are typically used to create, inspect, and update a Derby database.

For more complete information on developing a system using Derby, see the *Java DB Developer's Guide*.

Overview

Derby is a database management system (DBMS), accessed by applications through the JDBC API.

Included with the product are some standalone Java tools and utilities that make it easier to use and develop applications for Derby.

These tools and utilities include:

- *ij*

  `ij` is Derby's interactive JDBC scripting tool. It is a simple utility for running scripts against a Derby database. You can also use it interactively to run ad hoc queries. `ij` provides several commands for ease in accessing a variety of JDBC features.

  `ij` can be used in an embedded or a client/server environment.
- *sysinfo*

  `sysinfo` provides information about your version of Derby and your environment.
- *dblook*

  `dblook` is Derby's Data Definition Language (DDL) Generation Utility, more informally called a schema dump tool. It is a simple utility that dumps the DDL of a user-specified database to either a console or a file. The generated DDL can then be used for such things as recreating all or parts of a database, viewing a subset of a database's objects (for example, those which pertain to specific tables and schemas), or documenting a database's schema.
- *SignatureChecker*

  The `SignatureChecker` tool identifies any SQL functions and procedures in a database that do not follow the SQL Standard argument matching rules.

Environment setup and the Derby tools

The tools `ij`, `sysinfo`, `dblook`, and `SignatureChecker` can all be used in either an embedded or a client/server environment.

Java Platform, Standard Edition 5

All Derby tools require Java Platform, Standard Edition (Java SE) 5 or later or a platform that is running the Connected Device Configuration/Foundation Profile (JSR 169).

Classpath

The jar file `derbyrun.jar` simplifies the process of setting up the `CLASSPATH` environment variable to run Derby and the tools. Adding this jar file to your classpath has the effect of putting all the Derby jar files in your classpath.

For details on using the Derby jar files for deploying applications, see the sections on deploying Derby applications in the *Java DB Developer's Guide*.

About Derby databases

A Derby database consists of platform-independent files stored in a directory that has the same name as the database.

JDBC connection basics

Most of the Derby tools are JDBC applications. A JDBC application is one that uses the classes in the *java.sql* package to interact with a DBMS.

When you work with JDBC applications, you need to know about several concepts. The most basic is the *connection*. A *JDBC connection* is the object through which commands are sent to the Derby engine and responses are returned to the program. Establishing a connection to a specific database is done by specifying a appropriate database *URL*. The following sections provide background information to help in understanding the Derby database connection URL.

JDBC drivers overview

Before a JDBC application connects to a database, it must cause the proper JDBC driver to be loaded in the Java session. Derby provides the following JDBC drivers for use with the Derby database engine:

- *org.apache.derby.jdbc.EmbeddedDriver*

  For embedded environments, when Derby runs in the same JVM as the application. This is commonly referred to as the embedded drvier.
- *org.apache.derby.jdbc.ClientDriver*

  For client/server environments that use the Derby Network Server. This is commonly referred to as the Network Client driver.

You can use `ij` to connect to any database that supplies a JDBC driver. For those databases, you would need to load the supplied JDBC driver.

Database connection URLs

A JDBC URL provides a way of identifying a database so that the appropriate driver recognizes it and connects to it. In the Derby documents, a JDBC URL is referred to as a database connection URL.

After the driver is loaded, an application must specify the correct database connection URL to connect to a specific database. The Derby database connection URL allows you to accomplish tasks other than simply connecting. For more information about the Derby database connection URLs, see the *Java DB Developer's Guide*.

A JDBC URL always starts with *jdbc:*. After that, the format for the database connection URL depends on the JDBC driver.

Here is the format for the database connection URL for connecting to an existing Derby database using the embedded driver:

- jdbc:derby:*databaseName*;*URLAttributes*

The format for the database connection URL for connecting to an existing Derby database using the Network Client is:

- jdbc:derby://*host:port*/*databaseName*;*URLAttributes*

The italicized items stand for something the user fills in:

- *databaseName*

The name of the database you want to connect to. This might also include the file system path to the database.

- *URLAttributes*

  One or more of the supported attributes of the database connection URL, such as *upgrade=true*, *create=true* or *territory=ll_CC*. For more information, see "Setting attributes for the database connection URL" in the *Java DB Reference Manual*.
- *host*

  The name of the machine where the server is running. It can be the name of the machine or the address.
- *port*

  The port number used by the server framework

About Protocols

Officially, the portion of the database connection URL called the protocol is *jdbc:*, just as *http://* is a protocol in Web URLs. However, the second portion of the database connection URL (everything between *jdbc:* and *databaseName*), which is called the subprotocol, is informally considered part of the protocol. Later in this book you might see references to protocol. Consider protocol to be everything that comes before *databaseName*.

For complete information about the database connection URL, see the *Java DB Developer's Guide*.

Tools and localization

The Derby tools provide support for common localization features such as localized message files and GUI, locale-appropriate formatting of data, codesets, unicode identifiers and data, and database locales.

For general information about localizing Derby systems, see the *Java DB Developer's Guide*.

About locales

In the Derby documentation, we refer to three locales:

- *Java system locale*

  This is the locale of your machine, which is automatically detected by your JVM. For Derby and Derby tools, the Java system locale determines the default locale.
- *Database locale*

  This is the locale associated with your database when it is created. By default, this is the same as the Java system locale. The database locale determines the language of database errors.
- *Tools session locale*

  This locale is associated with your session, when using Derby tools such as `ij` or `dblook`. This locale determines the language of messages, as well as the localized display format for numbers, dates, times, and timestamps. You can use the `derby.ui.locale` property to specify the session locale that should be used.

Database locale

To specify a database locale, use the *territory=ll_CC* attribute on the URL connection when creating the database.

Note: You cannot modify a database's locale after the database has been created.

For information about database locales, see "Localizing Derby" in the *Java DB Developer's Guide*.

Specifying an alternate codeset

You can specify an alternate codeset for your tool session.

Use the *derby.ui.codeset* property when starting `ij` or `dblook`. This property can be useful when working with scripts created on a different system.

Formatting display of locale-sensitive data

To display dates, timestamps, numbers, and times in the format of the `ij` Session locale, use the *LocalizedDisplay* command.
Note: These options do not change how Derby*stores* locale-sensitive data, simply how the tool displays the data.

The following example demonstrates using *localizedDisplay* in an *en_US* locale:

```
ij> VALUES CURRENT_DATE;
1
----------
2011-05-23
1 row selected
ij> localizeddisplay on;
ij> VALUES CURRENT_DATE;
1
------------------
May 23, 2011
1 row selected
```

Using ij

ij is Derby's interactive JDBC scripting tool. It is a simple utility for running scripts or interactive queries against a Derby database.

ij is a Java application, which you start from a command window such as an MS-DOS Command Window or the UNIX shell. ij provides several non-SQL commands for ease in accessing a variety of JDBC features for testing.

Starting ij

Derby provides batch and shell scripts for users in Windows and UNIX environments that can be used to start ij. By calling the appropriate script you will start ij and be able to connect with a simple command. The scripts are found in the `bin` directory of your Derby installation. You can also customize the ij scripts to suit your environment.

If you are using Derby as a client/server environment, start the Network Server before connecting to the Derby database. (See "Starting the Network Server" in the *Java DB Server and Administration Guide* for details.) You can start ij by running the ij scripts for your environment. Follow the instructions in "Setting up your environment" in *Getting Started with Java DB* to set the `DERBY_HOME` and `JAVA_HOME` environment variables and to add `DERBY_HOME/bin` to your path. Then use the following command:

```
ij [-p propertyFile] [inputFile]
```

Alternatively, set the `DERBY_HOME` environment variable, then use one of these commands:

```
(UNIX) java [options] -jar $DERBY_HOME/lib/derbyrun.jar ij
    [-p propertyFile] [inputFile]
```

```
(Windows) java [options] -jar %DERBY_HOME%\lib\derbyrun.jar ij
    [-p propertyFile] [inputFile]
```

```
java [options] org.apache.derby.tools.ij
    [-p propertyFile] [inputFile]
```

If you use the last form of the command, be sure that `derbyrun.jar` is in your classpath (for pre-10.2 distributions `derbytools.jar` and usually `derby.jar` were required in the classpath).

If you need to use other classes in addition to `derbyrun.jar`, you cannot use the `-cp` argument or the CLASSPATH environment variable to set CLASSPATH variables when you are using the `-jar` argument to start the `ij` tool. If you want to run the `ij` tool with a custom classpath, you cannot use the `-jar` argument. Instead, you have to use the full class name to start the `ij` tool (`java org.apache.derby.tools.ij`).

The command line items are:

- **`java`**

  Start the JVM.
- ***`options`***

  The options that the JVM uses. You can use the -D option to set ij properties (see Starting ij using properties) or system properties, such as Derby properties.
- ***`propertyFile`***

A file you can use to set ij properties (instead of the -D option). The property file should be in the format created by the java.tools.Properties.save methods, which is the same format as the derby.properties file.

- ***inputFile***

  A file from which to read commands. The ij tool exits at the end of the file or an exit command. Using an input file causes ij to print out the commands as it runs them. If you reroute standard input, ij does not print out the commands. If you do not supply an input file, ij reads from the standard input.

For detailed information about ij commands, see ij commands and errors reference.

Creating a database using `ij`

You can create a Derby from within the `ij` tool.

1. To create a database with the `ij` tool, type the following command:

   ```
   ij> connect 'jdbc:derby:testdb;create=true';
   ```

   This command creates a database called `testdb` in the current directory, populates the system tables, and connects to the database. You can then run any SQL statements from the `ij` command line.

Starting ij using properties

You set ij properties in any of the following ways:

1. by using the -D option on the command line
2. by specifying a properties file using the `-p propertyfile` option on the command line

Remember: ij property names are case-sensitive, while commands are case-insensitive.

Examples

The following examples illustrate how to use ij properties.

To start ij by using a properties file called `ij.properties`, use a command like the following (with the addition of the file paths):

```
java -jar derbyrun.jar -p ij.properties
```

To start ij with a maximumDisplayWidth of 1000:

```
java -Dij.maximumDisplayWidth=1000 -jar derbyrun.jar
```

To start ij with an ij.protocol of **jdbc:derby:** and an ij.database of **sample**, use the following command:

```
java -Dij.protocol=jdbc:derby: -Dij.database=sample derbyrun.jar
```

To start ij with two named connections, using the ij.connection.connectionName property, use a command like the following (all on one line):

```
java -Dij.connection.sample=jdbc:derby:sample
-Dij.connection.History=jdbc:derby:History
-Dderby.system.home=c:\derby\demo\databases
-jar c:\derby\lib\derbyrun.jar
```

To see a list of connection names and the URLs used to connect to them, use the following command. (If there is a connection that is currently active, it will show up with an * after its name.)

```
ij version 10.10
ij(HISTORY)> show connections;
HISTORY* -    jdbc:derby:History
SAMPLE -     jdbc:derby:sample
* = current connection
ij(HISTORY)>
```

Getting started with ij

This section discusses the use of the ij tool.

Connecting to a Derby database

To connect to a Derby database, you need to perform the following steps:

1. Start the JVM
2. Load the appropriate driver.
3. Create a connection by providing a valid database connection URL.

When using ij interactively to connect to a Derby database connection information is generally supplied on the full database connection URL. ij automatically loads the appropriate driver based on the syntax of the URL. The following example shows how to connect in this manner by using the **Connect** command and the embedded driver:

```
D:>java org.apache.derby.tools.ij
ij version 10.10
ij> connect 'jdbc:derby:sample';
ij>
```

If the URL entered contains Network Client information the **Connect** command loads the Network Client driver:

```
D:>java org.apache.derby.tools.ij
ij version 10.10
ij> connect 'jdbc:derby://localhost:1527/sample';
ij>
```

Note: In these and subsequent examples the databases were created in the derby.system.home directory. For more information on the System Directory see the *Java DB Developer's Guide*.

ij provides alternate methods of specifying part or all of a connection URL (e.g. the ij.protocol, ij.database, or ij.connection.connectionName properties). These properties are often used when a script is being used and the path to the database or the driver name is not known until runtime. The properties can also to used to shorten the amount of information that must be provided with the connection URL. The following are some examples of different ways to supply the connection information:

- Supplying full connection information on the command line
  Specifying one of the following properties along with a valid connection URL on the ij command line starts ij with the connection already active. This is often used when running a SQL script so the database name or path can be specified at runtime.
  - ij.database - opens a connection using the URL provided
  - ij.connection.*connectionName* - Used to open one or more connections. The property can appear multiple times on the command line with different *connectionNames* and the same or different URLs.

  This example shows how to create the database *myTours* and run the script *ToursDB_schema.sql* by specifying the database URL using the ij.database property.

  ```
  C:\>java -Dij.database=jdbc:derby:myTours;create=true
  ```

```
     org.apache.derby.tools.ij
    %JAVA_HOME%\demo\db\programs\toursdb\ToursDB_schema.sql
ij version 10.10
CONNECTION0* -  jdbc:derby:myTours
* = current connection
ij> -- Licensed to the Apache Software Foundation (ASF) under one or
 more
-- contributor license agreements.  See the NOTICE file distributed
 with
     ...output removed...
ij> CREATE TRIGGER TRIG2 AFTER DELETE ON FLIGHTS FOR EACH STATEMENT
 MODE DB2SQL
INSERT INTO FLIGHTS_HISTORY (STATUS) VALUES ('INSERTED FROM TRIG2');
0 rows inserted/updated/deleted
ij>
```

- Defining a Protocol and using a "short form" URL

  A default URL protocol and subprotocol can be specified by setting the property ij.protocol or using the ij Protocol command. This allows a connection to be made by specifying only the database name. This "short form" of the database connection URL defaults the protocol (For more information, see About Protocols).

  This example uses the ij Protocol command and a "short form" connection URL:

```
D:>java org.apache.derby.tools.ij
ij version 10.10
ij> protocol 'jdbc:derby:';
ij> connect 'sample';
ij>
```

- Specifying an alternate Driver

  If you are using the drivers supplied by Derby, you can specify the driver names listed in JDBC drivers overview. However, the Derby drivers are implicitly loaded when a supported protocol is used so specifying them is probably redundant. Specifying a driver is required when ij is used with other JDBC drivers to connect to non-Derby databases. To use drivers supplied by other vendors explicitly specify the driver one of three ways

  - with an ij property ij.Driver
  - using the JVM system property `jdbc.drivers`
  - using the ij Driver command

  This example specifies the driver using the ij Driver command

```
D:>java org.apache.derby.tools.ij
ij version 10.10
ij> driver 'sun.jdbc.odbc.JdbcOdbcDriver';
ij> connect 'jdbc:odbc:myOdbcDataSource';
ij>
```

The ij Driver name and connection URL

The following table summarizes the different ways to specify the driver name and database connection URL.

Table 1. Specifying the driver name and database connection URL

Action	System Property	`ij` Property	`ij` Command
Loading the driver implicitly	None	*ij.connection.connectionName* (plus full URL), *ij.database* (plus full URL), *ij.protocol*, *ij.protocol.protocolName* (plus protocol clause in Connect command)	*Protocol*, *Connect* (plus full URL)

Action	System Property	`ij` Property	`ij` Command
Loading the driver explicitly	*jdbc.drivers*	-D*ij.Driver*	*Driver*
Specifying the database connection URL	None	*ij.connection.connectionName*, *ij.database*	*Connect*

Using ij commands

The primary purpose of ij is to allow the execution of Derby SQL statements interactively or via scripts. Since SQL statements can be quite long, ij uses the semicolon to mark the end of a statement or command. All statements and commands must be terminated with a semicolon. If you press Return before terminating a statement or command, ij places a continuation character (>) at the beginning of the next line.

ij uses properties, listed in ij properties reference, to simplify its use.

ij also recognizes specialized commands that provide additional features, such as the ability to create and test cursors and prepared statements, transaction control, and more. For complete information about ij commands, see ij commands and errors reference.

Other uses for ij

ij is a JDBC-neutral scripting tool with a small command set. It can be used to access any JDBC driver and database accessible through that driver.

The main benefit of a tool such as ij is that it is easy to run scripts for creating a database schema and automating other repetitive database tasks.

In addition, ij accepts and processes SQL commands interactively for ad hoc database access.

Running ij scripts

You can run scripts in ij in any of the following ways:

- Name an input file as a command-line argument.

  For example:

  ```
  java org.apache.derby.tools.ij <myscript.sql>
  ```
- Redirect standard input to come from a file.

  For example:

  ```
  java org.apache.derby.tools.ij < <myscript.sql>
  ```
- Use the Run command from the ij command line.

  For example:

  ```
  ij> run 'myscript.sql';
  ```

Note: If you name an input file as a command-line argument or if you use the Run command, ij echoes input from a file. If you redirect standard input to come from a file, ij does not echo commands.

You can save output in any of the following ways:

- By redirecting output to a file:

```
java org.apache.derby.tools.ij <myscript.sql> > <myoutput.txt>
```

- By setting the `ij.outfile` property:

```
java -Dij.outfile=<myoutput.txt> org.apache.derby.tools.ij
    <myscript.sql>
```

ij exits when you enter the Exit command or, if executing a script, when the end of the command file is reached. When you use the Exit command, ij automatically shuts down an embedded Derby system by issuing a `connect jdbc:derby:;shutdown=true` request. It does not shut down Derby if it is running in a server framework.

ij properties reference

When starting up `ij`, you can specify properties on the command line or in a properties file, as described in Starting ij using properties.

ij.connection.*connectionName* property

Function

Creates a named connection to the given database connection URL when `ij` starts up; it is equivalent to the Connect AS *Identifier* command. The database connection URL can be of the short form if an *ij.protocol* is specified. This property can be specified more than once per session, creating multiple connections. When `ij` starts, it displays the names of all the connections created in this way. It also displays the name of the current connection, if there is more than one, in the `ij` prompt.

Syntax

```
ij.connection.connectionName=databaseConnectionURL
```

When specified on the command line the *databaseConnectionURL* should not be enclosed in single quotations, however, if the database path contains special characters (e.g. a space) it must be enclosed in double quotes.

Example

This example connects to the existing database *sample* and creates, then connects to, the database *anotherDB*.

```
D:> java -Dij.connection.sample1=jdbc:derby:sample
-Dij.connection.anotherConn=jdbc:derby:anotherDB;create=true
    org.apache.derby.tools.ij
ij version 10.10
ANOTHERCONN* -  jdbc:derby:anotherDB;create=true
SAMPLE1 -        jdbc:derby:sample
* = current connection
ij(ANOTHERCONN)>
```

See also

- Connect command

ij.database property

Function

Creates a connection to the database name listed indicated by the property when `ij` starts up. You can specify the complete connection URL (including protocol) with this property or just the database name if you also specify *ij.protocol* on the command line. After it boots, `ij` displays the generated name of the connection made with this property.

Syntax

```
ij.database=databaseConnectionURL
```

When specified on the command line the *databaseConnectionURL* should not be enclosed in single quotations, however, if the database path contains special characters (e.g. a space) it must be enclosed in double quotes.

Example

```
java -Dij.protocol=jdbc:derby:
    -Dij.database=wombat;create=true org.apache.derby.tools.ij
ij version 10.10
CONNECTION0* -  jdbc:derby:wombat
* = current connection
ij>
```

ij.dataSource property

Function

The `ij.dataSource` property specifies the datasource to be used to access the database. When specifying a datasource, `ij` does not use the DriverManager mechanism to establish connections.

Syntax

When you set the `ij.dataSource` property `ij` will automatically try to connect to a database. To establish a connection to a specific database using `ij.dataSource`, set the `ij.dataSource.databaseName` property. If you do not set this property, `ij` will start with an error. If you want to create the database, specify the `ij.dataSource.createDatabase` property as well as `ij.dataSource.databaseName`. Do not specify `ij.protocol` when setting `ij.dataSource` as that would activate the DriverManager mechanism.

```
ij.dataSource=datasource class name
ij.dataSource.databaseName=databasename
[ij.dataSource.createDatabase=create]
```

If you do not specify `ij.dataSource.databaseName` and get an error indicating no database was found, you can still connect to a database by using `ij`'s `connect` command. You should not specify the protocol (for example jdbc:derby:) in the `connect` command when using `ij.dataSource`.

Example 1

In the following example, `ij` connects to a database named `sample` using an `EmbeddedDataSource`. The `sample` database is created if it does not already exist.

```
#
# If your application runs on JDK 1.6 or higher, then you should
# specify the JDBC4 variant of this DataSource:
# org.apache.derby.jdbc.EmbeddedDataSource40.
# If your application runs with a jvm supporting JSR169, you cannot use
# org.apache.derby.jdbc.EmbeddedDataSource, instead, use:
# org.apache.derby.jdbc.EmbeddedSimpleDataSource.
#
java -Dij.dataSource=org.apache.derby.jdbc.EmbeddedDataSource
-Dij.dataSource.databaseName=sample -Dij.dataSource.createDatabase=create
  org.apache.derby.tools.ij
ij version 10.10
CONNECTION0*
* = current connection
ij>
```

Example 2

In the following example, `ij` starts using an `EmbeddedSimpleDataSource`, without specifying `ij.dataSource.databaseName`. This results in an error indicating no database was found. After the error, the `connect` command is used to create and connect to a database named `smalldb`.

```
#
```

```
# Start ij using EmbeddedSimpleDataSource
#
java -Dij.dataSource=org.apache.derby.jdbc.EmbeddedSimpleDataSource
  org.apache.derby.tools.ij
ERROR XJ004: Database '' not found.
ij version 10.10
ij> connect 'smalldb;create=true';
ij>
```

For more information about DataSources, refer to the JDBC documentation and "Using Derby as a J2EE Resource Manager" in the *Java DB Developer's Guide*.

ij.driver property

Function

Loads the JDBC driver that the class specifies.

Syntax

```
ij.driver=JDBCDriverClassName
```

Notes

Example

```
D:>java -Dij.driver=sun.jdbc.odbc.JdbcOdbcDriver
 org.apache.derby.tools.ij
ij version 10.10
ij> Connect 'jdbc:odbc:MyODBCDataSource';
ij>
```

See also

- Driver command

ij.exceptionTrace property

Function

When the `ij.exceptionTrace` property is set to `true`, a full exception stack trace is printed when exceptions occur in `ij`. The default setting is `false`.

Syntax

```
ij.exceptionTrace={ false | true }
```

Example

In the following example, `ij` is started with the `ij.exceptionTrace` property set to true.

```
java -Dij.exceptionTrace=true org.apache.derby.tools.ij
ij version 10.10
ij> connect 'jdbc:derby:wombat';
ERROR XJ004: Database 'wombat' not found.
SQL Exception: Database 'wombat' not found.
    at
 org.apache.derby.impl.jdbc.SQLExceptionFactory.getSQLException(SQLExceptionFactory.jav
    at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Util.java:87)
    at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Util.java:93)
    at
 org.apache.derby.impl.jdbc.Util.generateCsSQLException(Util.java:172)
    at
 org.apache.derby.impl.jdbc.EmbedConnection.newSQLException(EmbedConnection.java:1955)
    at
 org.apache.derby.impl.jdbc.EmbedConnection.(EmbedConnection.java:254)
```

```
    at
 org.apache.derby.impl.jdbc.EmbedConnection30.(EmbedConnection30.java:72)
    at
 org.apache.derby.jdbc.Driver30.getNewEmbedConnection(Driver30.java:73)
    at
 org.apache.derby.jdbc.InternalDriver.connect(InternalDriver.java:200)
    at java.sql.DriverManager.getConnection(DriverManager.java:512)
    at java.sql.DriverManager.getConnection(DriverManager.java:140)
    at org.apache.derby.impl.tools.ij.ij.dynamicConnection(ij.java:873)
    at org.apache.derby.impl.tools.ij.ij.ConnectStatement(ij.java:723)
    at org.apache.derby.impl.tools.ij.ij.ijStatement(ij.java:553)
    at org.apache.derby.impl.tools.ij.utilMain.go(utilMain.java:289)
    at org.apache.derby.impl.tools.ij.Main.go(Main.java:207)
    at org.apache.derby.impl.tools.ij.Main.mainCore(Main.java:173)
    at org.apache.derby.impl.tools.ij.Main14.main(Main14.java:55)
    at org.apache.derby.tools.ij.main(ij.java:60)  ij
ij>
```

ij.maximumDisplayWidth property

Function

Specifies the maximum number of characters used to display any column. The default value is 128. Values with display widths longer than the maximum are truncated and terminated with an & character.

Syntax

```
ij.maximumDisplayWidth=numberOfCharacters
```

Example

```
java -Dij.maximumDisplayWidth=1000 org.apache.derby.tools.ij
```

See also

- MaximumDisplayWidth command

ij.outfile property

Function

Specifies a file to which the system should direct output for a session. Specify the file name relative to the current directory, or specify the absolute path.

Syntax

```
ij.outfile=fileName
```

Example

```
java -Dij.outfile=out.txt org.apache.derby.tools.ij myscript.sql
```

ij.password property

Function

Specifies the password used to make connections. This property is used in conjunction with the *ij.user* property to authenticate a connection. If authentication is not active then these properties are ignored.

Syntax

```
ij.password=password
```

Example

```
java -Dij.user=me -Dij.password=mine org.apache.derby.tools.ij
```

See the *Java DB Developer's Guide* for more information on Derby authentication and security.

ij.protocol property

Function

Specifies the default protocol and subprotocol portions of the database connection URL for connections. The Derby protocol is:

- jdbc:derby:

Allows you to use a short form of a database name in a connection URL.

Syntax

```
ij.protocol=protocolForEnvironment
```

Example

```
D:>java -Dij.protocol=jdbc:derby:
    org.apache.derby.tools.ij
ij version 10.10
ij> Connect 'newDB;create=true';
ij>
```

See also

- Protocol command

ij.protocol.*protocolName* property

Function

This property is similar to the *ij.protocol* property. The only difference is that it associates a name with the value, thus allowing you to define and use more than one protocol. (See Connect command.)

Syntax

```
ij.protocol.protocolName=protocolForEnvironment
```

Example

```
D:>java -Dij.protocol.derby=jdbc:derby:
-Dij.protocol.emp=jdbc:derby: org.apache.derby.tools.ij
ij version 10.10
ij> Connect 'newDB' protocol derby as new;
ij>
```

See also

- Protocol command

ij.showErrorCode property

Function

Set this property to *true* to have ij display the *SQLException ErrorCode* value with error messages. The default is *false*.

Error codes denote the severity of the error. For more information, see the *Java DB Reference Manual*.

Syntax

```
ij.showErrorCode={ false | true }
```

Example

```
java -Dij.showErrorCode=true -Dij.protocol=jdbc:derby:
    org.apache.derby.tools.ij
ij version 10.10
ij> Connect 'sample';
ij> VLUES 1;
ERROR 42X01: Syntax error: Encountered "VLUES"
at line 1, column 1. (errorCode = 30000)
ij>
```

ij.showNoConnectionsAtStart property

Function

The `ij.showNoConnectionsAtStart` property specifies whether the connections message should be displayed when `ij` is started. Default is false, that is, a message indicating the current connections, if any, is displayed.

Syntax

```
ij.showNoConnectionsAtStart={ false | true }
```

Example

In the following example, `ij` connects to a previously created database named `sample` using an `EmbeddedDataSource`. The property `ij.showNoConnectionsAtStart` is set to true in the first session of the example, and set to false in the second session.

```
java -Dij.dataSource=org.apache.derby.jdbc.EmbeddedDataSource
-Dij.dataSource.databaseName=sample -Dij.showNoConnectionsAtStart=true
 org.apache.derby.tools.ij
ij version 10.10
ij> disconnect;
ij> exit;

java -Dij.dataSource=org.apache.derby.jdbc.EmbeddedDataSource
-Dij.dataSource.databaseName=sample -Dij.showNoConnectionsAtStart=false
 org.apache.derby.tools.ij
ij version 10.10
CONNECTION0*
* = current connection
ij> disconnect;
ij> exit;
```

ij.showNoCountForSelect property

Function

The `ij.showNoCountForSelect` property specifies whether to display messages indicating the number of rows selected. Default is false, that is, if the property is not set, select count messages are displayed.

Syntax

```
ij.showNoCountForSelect={ false | true }
```

Example

In the following example, `ij` is first started with the `ij.showNoCountForSelect` property to true, then with the property set to false.

```
java -Dij.showNoCountForSelect=true org.apache.derby.tools.ij
ij version 10.10
CONNECTION0*
* = current connection
ij> create table t1 (c1 int);
0 rows inserted/updated/deleted
ij> insert into t1 values 1, 2, 3;
3 rows inserted/updated/deleted
ij> select * from t1;
C1
-----------
1
2
3
ij> disconnect;
ij> exit;

java -Dij.showNoCountForSelect=false org.apache.derby.tools.ij
ij version 10.10
CONNECTION0*
* = current connection
ij> select * from t1;
C1
-----------
1
2
3

3 rows selected

ij>
```

ij.URLCheck property

Function

This property determines whether `ij` checks for invalid or non-Derby URL attributes when you are using the embedded driver. Set this property to *false* to prevent `ij` from validating URL attributes. The default value is *true*.

When the *ij.URLCheck* property is set to *true*, you are notified whenever a connection URL contains an incorrectly specified attribute. For example if the attribute name is misspelled or cased incorrectly `ij` prints a message.
Note: `ij` checks attribute *values* if the attribute has pre-defined values. For example, the attribute *shutdown* has the pre-defined values of *true* or *false*. If you try to set the attribute *shutdown* to a value other than *true* or *false*, `ij` displays an error. For example:

```
ij> Connect 'jdbc:derby:anyDB;shutdown=rue';
ERROR XJ05B: JDBC attribute 'shutdown' has an invalid value 'rue',
valid values are '{true|false}'.
ij>
```

Syntax

```
ij.URLCheck={ false | true }
```

Example

By default, `ij` displays messages about invalid attributes:

```
java org.apache.derby.tools.ij
```

```
ij version 10.10
ij> connect 'mydb;uSer=naomi';
URL Attribute [uSer=naomi]
   Case of the Derby attribute is incorrect.
```

The following command line specifies to turn off URL attribute checking in `ij`.

```
java -Dij.URLCheck=false org.apache.derby.tools.ij
ij version 10.10
ij> connect 'mydb;uSer=naomi';
ij>
```

Typically, you would only explicitly turn off the URL checker if you were using `ij` with a non-Derby JDBC driver or database.

Notes

The URL checker does not check the validity of properties, only database connection URL *attributes*.

For a list of attributes, see "Setting attributes for the database connection URL" in the *Java DB Reference Manual*. Because the *ij.URLCheck* property is valid only with the embedded driver, it does not apply to attributes such as *securityMechanism=value*, *ssl=sslMode*, and the attributes related to tracing.

ij.user property

Function

Specifies the logon name used to establish the connection. This property is used in conjunction with the *ij.password* property to authenticate a connection. If authentication is not active then these properties are ignored.

When a username is supplied you need to be aware of the database schema. When you connect using `ij.user`, the default database schema applied to all SQL statements is the same as the user id provided even if the schema does not exist. Use the SET SCHEMA statement to change the default when the schema does not match the username. Alternately you can fully qualify the database objects referred to in the SQL statements . If no user is specified, no SET SCHEMA statement has been issued, or SQL statements do not include the schema name, all database objects are assumed to be under the APP schema.

Syntax

```
ij.user=username
```

Example

```
java -Dij.user=me -Dij.password=mine org.apache.derby.tools.ij
ij version 10.10
ij> connect 'jdbc:derby:sampleDB';
ij> set schema finance;
ij> select * from accounts;
```

See the *Java DB Developer's Guide* for more information on Derby and security.

derby.ui.codeset property

Function

Set this property to a supported character encoding value when using one of the Derby tools with a language not supported by your default system.

Syntax

```
derby.ui.codeset=derbyval
```

where *derbyval* is a supported character encoding value, for example, UTF8 (see the table later in this topic).

Example

The following command runs `ij` using the Japanese locale (*derby.ui.locale=ja_JP*) and Japanese Latin Kanji mixed encoding (*derby.ui.codeset=Cp939*):

```
java -Dderby.ui.locale=ja_JP -Dderby.ui.codeset=Cp939
    -Dij.protocol=jdbc:derby:
    org.apache.derby.tools.ij
```

The following table contains a sampling of character encodings. Supported encodings vary from product to product. For example, to see the full list of the character encodings that are supported by the Java Platform, Standard Edition 7 Software Development Kit, go to http://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html.

Table 2. Sample character encodings

Character Encoding	Explanation
8859_1	ISO Latin-1
8859_2	ISO Latin-2
8859_7	ISO Latin/Greek
Cp1257	Windows Baltic
Cp1258	Windows Vietnamese
Cp437	PC Original
EUCJIS	Japanese EUC
GB2312	GB2312-80 Simplified Chinese
JIS	JIS
KSC5601	KSC5601 Korean
MacCroatian	Macintosh Croatian
MacCyrillic	Macintosh Cyrillic
SJIS	PC and Windows Japanese
UTF8	Standard UTF-8

derby.ui.locale property

Function

Set this property to a supported locale name when using one of the Derby tools, if you want another locale than the system default locale. The locale determines the localized display format for numbers, dates, times and timestamps, as well as the language of the messages from the Derby tools.

Syntax

```
derby.ui.locale=derbyval
```

where *derbyval* is a supported locale name, in the form *ll_CC*, where *ll* is the two-letter language code, and *CC* is the two-letter country code; for example, `ja_JP`.

Example

The following command runs `ij` using the Japanese locale (*derby.ui.locale=ja_JP*) and Japanese Latin Kanji mixed encoding (*derby.ui.codeset=Cp939*):

```
java -Dderby.ui.locale=ja_JP -Dderby.ui.codeset=Cp939
    -Dij.protocol=jdbc:derby:
    org.apache.derby.tools.ij
```

Language codes consist of a pair of lowercase letters that conform to ISO 639-1. The following table shows some examples.

Table 3. Sample language codes

Language Code	Description
de	German
en	English
es	Spanish
ja	Japanese

To see a full list of ISO 639 codes, go to *http://www.loc.gov/standards/iso639-2/php/code_list.php*.

Country codes consist of two uppercase letters that conform to ISO 3166. The following table shows some examples.

Table 4. Sample country codes

Country Code	Description
DE	Germany
US	United States
ES	Spain
MX	Mexico
JP	Japan

A copy of ISO 3166 can be found at *http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html*.

ij commands and errors reference

This section describes the commands and errors within the ij tool.

ij commands

`ij` accepts several commands to control its use of JDBC. It recognizes a semicolon as the end of an `ij` or SQL command; it treats semicolons within SQL comments, strings, and delimited identifiers as part of those constructs, not as the end of the command. Semicolons are required at the end of an `ij` or SQL statement.

All `ij` commands, identifiers, and keywords are case-insensitive.

Commands can span multiple lines without any special escaping for the ends of lines. This means that if a string spans a line, the new lines will show up in the value in the string.

`ij` treats any command that it does not recognize as an SQL command to be passed to the underlying connection, so syntactic errors in `ij` commands will cause them to be handed to the SQL engine and will probably result in SQL parsing errors.

Conventions for ij examples

Examples in this document show input from the keyboard or a file in bold text and console output from the DOS prompt or the `ij` application in regular text.

```
C:\> REM This example is from a DOS prompt:
C:\> java -Dij.protocol=jdbc:derby: org.apache.derby.tools.ij
ij version 10.10
ij> connect 'menuDB;create=true';
ij> CREATE TABLE menu(course CHAR(10), item CHAR(20), price INTEGER);
0 rows inserted/updated/deleted
ij> disconnect;
ij> exit;
C:\>
```

ij SQL command behavior

Any command other than those documented in the `ij` command reference are handed to the current connection to execute directly. The statement's closing semicolon, used by `ij` to determine that it has ended, is not passed to the underlying connection. Only one statement at a time is passed to the connection. If the underlying connection itself accepts semicolon-separated statements (which Derby does not), they can be passed to the connection using `ij`'s Execute command to pass in a command string containing semicolon-separated commands.

`ij` uses the result of the JDBC execute request to determine whether it should print a number-of-rows message or display a result set.

If a JDBC execute request causes an exception, it displays the *SQLState*, if any, and error message.

Setting the `ij` property *ij.showErrorCode* to *true* displays the *SQLException*'s error code (see ij properties reference).

The number-of-rows message for inserts, updates, and deletes conforms to the JDBC specification for any SQL statement that does not have a result set. DDL (data definition

language) commands typically report "0 rows inserted/updated/deleted" when they successfully complete.

To display a result set, `ij` formats a banner based on the JDBC *ResultSetMetaData* information returned from *getColumnLabel* and *getColumnWidth*. Long columns wrap the screen width, using multiple lines. An & character denotes truncation (`ij` limits displayed width of a column to 128 characters by default; see MaximumDisplayWidth command).

`ij` displays rows as it fetches them. If the underlying DBMS materializes rows only as they are requested, ij displays a partial result followed by an error message if there is a error in fetching a row partway through the result set.

`ij` verifies that a connection exists before issuing statements against it and does not execute SQL when no connection has yet been made.

There is no support in `ij` for the JDBC feature multiple result sets.

ij command example

```
ij> INSERT INTO menu VALUES ('appetizer','baby greens',7),
('entree','lamb chops ',6),('dessert','creme brulee',14);
3 rows inserted/updated/deleted
ij> SELECT * FROM menu;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |6
appetizer |baby greens         |7

3 rows selected
ij>
```

Absolute command

Syntax

```
ABSOLUTE int Identifier
```

Description

Moves the cursor to the row specified by the *int*, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

Example

```
ij> autocommit off;
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> absolute 3 scrollCursor;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
```

After Last command

Syntax

```
AFTER LAST Identifier
```

Description

Moves the cursor to after the last row, then fetches the row. (Since there is no current row, it returns the message: `"No current row."`

The cursor must have been created with the Get Scroll Insensitive Cursor command.

Example

```
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> after last scrollcursor;
No current row
```

Async command

Syntax

```
ASYNC Identifier String
```

Description

The ASYNC command lets you execute an SQL statement in a separate thread. It is used in conjunction with the Wait For command to get the results.

You supply the SQL statement, which is any valid SQL statement, as a *String*. The *Identifier* you must supply for the async SQL statement is used in the Wait For command and is a case-insensitive `ij` identifier. An identifier that does not specify a *connectionName* must not be the same as any other identifier for an async statement on the current connection; an identifier that specifies a *connectionName* must not be the same as any other identifier for an async statement on the designated connection. You cannot reference a statement previously prepared and named by the `ij` Prepare command in this command.

`ij` creates a new thread in the current or designated connection to issue the SQL statement. The separate thread is closed once the statement completes.

Examples

```
ij> async aInsert 'INSERT into menu values (''entree'',''chicken'',11)';
ij> INSERT INTO menu VALUES ('dessert','ice cream',3);
1 rows inserted/updated/deleted.
ij> wait for aInsert;
1 rows inserted/updated/deleted.
-- the result of the asynchronous insert
```

```
ij> connect 'jdbc:derby:memory:dummy;create=true;user=john'
    as john_conn;
ij> create table john_tbl (c int);
0 rows inserted/updated/deleted
ij> insert into john_tbl values(1),(2),(3);
3 rows inserted/updated/deleted
ij> connect 'jdbc:derby:memory:dummy;user=fred' as fred_conn;
ij(FRED_CONN)> async john_async @ john_conn 'select * from john_tbl';
ij(FRED_CONN)> wait for john_async @ john_conn;
C
-----------
1
2
3

3 rows selected
ij(FRED_CONN)>
```

Autocommit command

Syntax

```
AUTOCOMMIT { ON | OFF }
```

Description

Turns the connection's auto-commit mode on or off. JDBC specifies that the default auto-commit mode is `ON`. Certain types of processing require that auto-commit mode be `OFF`. For information about auto-commit, see the *Java DB Developer's Guide*.

If auto-commit mode is changed from off to on when there is a transaction outstanding, that work is committed when the current transaction commits, not at the time auto-commit is turned on. Use Commit or Rollback before turning on auto-commit when there is a transaction outstanding, so that all prior work is completed before the return to auto-commit mode.

Example

```
ij> autocommit off;
ij> DROP TABLE menu;
0 rows inserted/updated/deleted
ij> CREATE TABLE menu (course CHAR(10), item CHAR(20), price INT);
0 rows inserted/updated/deleted
ij> INSERT INTO menu VALUES ('entree', 'lamb chop', 14),
('dessert', 'creme brulee', 6),
('appetizer', 'baby greens', 7);
3 rows inserted/updated/deleted
ij> commit;
ij> autocommit on;
ij>
```

Before First command

Syntax

```
BEFORE FIRST int Identifier
```

Description

Moves the cursor to before the first row, then fetches the row. (Since there is no current row, it returns the message `No current row`.)

The cursor must have been created with the Get Scroll Insensitive Cursor command.

Example

```
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> before first scrollcursor;
No current row
```

Close command

Syntax

```
CLOSE Identifier
```

Description

Closes the named cursor. The cursor must have previously been successfully created with the `ij`Get Cursor or Get Scroll Insensitive Cursor commands.

Example

```
ij> get cursor menuCursor as 'SELECT * FROM menu';
ij> next menuCursor;
COURSE     |ITEM                |PRICE
-----------------------------------------------
entree     |lamb chop           |14
```

```
ij> next menuCursor;
COURSE     |ITEM                |PRICE
---------------------------------------------
dessert    |creme brulee        |6
ij> close menuCursor;
ij>
```

Commit command

Syntax

```
COMMIT
```

Description

Issues a *java.sql.Connection.commit* request. Use this command only if auto-commit is off. A *java.sql.Connection.commit* request commits the currently active transaction and initiates a new transaction.

Example

```
ij> commit;
ij>
```

Connect command

Syntax

```
CONNECT ConnectionURLString [ PROTOCOL Identifier ]
    [ AS Identifier ] [ USER String
     PASSWORD String ]
```

Description

Connects to the database indicated by the *ConnectionURLString*. Specifically, takes the value of the string (the database connection URL) and issues a `getConnection` request using `java.sql.DriverManager` or a `javax.sql.DataSource` implementation (see the ij.dataSource property) to set the current connection to that database connection URL.

You have the option of specifying a name for your connection. Use the Set Connection command to switch between connections. If you do not name a connection, the system generates a name automatically.

You also have the option of specifying a named protocol previously created with the Protocol command or the ij.protocol.protocolName property.

If the connection requires a user name and password, supply those with the optional user and password parameters.

If the connect succeeds, the connection becomes the current one and ij displays a new prompt for the next command to be entered. If you have more than one open connection, the name of the connection appears in the prompt.

All further commands are processed against the new, current connection.

Examples

```
ij> connect 'jdbc:derby:menuDB;create=true';
ij> -- we create a new table in menuDB:
CREATE TABLE menu(course CHAR(10), item CHAR(20), price INTEGER);
ij> protocol 'jdbc:derby:';
ij> connect 'sample' as sample1;
ij(SAMPLE1)> connect 'newDB;create=true' as newDB;
```

```
ij(NEWDB)> show connections;
CONNECTION0 - jdbc:derby:menuDB
NEWDB* - jdbc:derby:anotherDB
SAMPLE1 - jdbc:derby:newDB
ij>
ij> connect 'jdbc:derby:sample' user 'sa' password 'cloud3x9';
ij>
```

```
ij> protocol 'jdbc:derby:';
ij> connect 'memory:sample;create=true';
```

```
ij> protocol 'jdbc:derby:memory:';
ij> connect 'sample;create=true';
```

Describe command

Syntax

```
DESCRIBE { table-Name | view-Name }
```

Description

Provides a decription of the specified table or view. For a list of tables in the current schema, use the Show Tables command. For a list of views in the current schema, use the Show Views command. For a list of available schemas, use the Show Schemas command.

If the table or view is in a particular schema, qualify it with the schema name. If the table or view name is case-sensitive, enclose it in single quotes. You can display all the columns from all the tables and views in a single schema in a single display by using the wildcard character '*'. See the examples below.

Examples

```
ij> describe airlines;
COLUMN_NAME
 |TYPE_NAME|DEC&|NUM&|COLUM&|COLUMN_DEF|CHAR_OCTE&|IS_NULL&
------------------------------------------------------------------------
-----
AIRLINE              |CHAR     |NULL|NULL|2     |NULL      |4         |NO
AIRLINE_FULL         |VARCHAR  |NULL|NULL|24    |NULL      |48        |YES
BASIC_RATE           |DOUBLE   |NULL|2   |52    |NULL      |NULL      |YES
DISTANCE_DISCOUNT    |DOUBLE   |NULL|2   |52    |NULL      |NULL      |YES
BUSINESS_LEVEL_FACT&|DOUBLE   |NULL|2   |52    |NULL      |NULL      |YES
FIRSTCLASS_LEVEL_FA&|DOUBLE   |NULL|2   |52    |NULL      |NULL      |YES
ECONOMY_SEATS        |INTEGER  |0   |10  |10    |NULL      |NULL      |YES
BUSINESS_SEATS       |INTEGER  |0   |10  |10    |NULL      |NULL      |YES
FIRSTCLASS_SEATS     |INTEGER  |0   |10  |10    |NULL      |NULL      |YES
```

```
   -- describe a table in another schema:
   describe user2.flights;
   -- describe a table whose name is in mixed-case:
   describe 'EmployeeTable';
   -- describe a table in a different schema, with a case-sensitive
 name:
   describe 'MyUser.Orders';
   -- describe all the columns from all the tables and views in APP
 schema:
   describe 'APP.*';
   -- describe all the columns in the current schema:
   describe '*';
```

Disconnect command

Syntax

```
DISCONNECT [ ALL | CURRENT | ConnectionIdentifier ]
```

Description

Disconnects from the database. Specifically issues a `java.sql.Connection.close` request against the connection indicated on the command line. There must be a current connection at the time the request is made.

If ALL is specified, all known connections are closed and there will be no current connection.

Disconnect CURRENT is the same as Disconnect without indicating a connection, the default connection is closed.

If a connection name is specified with an identifier, the command disconnects the named connection. The name must be the name of a connection in the current session provided with the ij.connection.connectionName property or with the Connect command.

If the ij.database property or the Connect command without the AS clause was used, you can supply the name the system generated for the connection. If the current connection is the named connection, when the command completes, there will be no current connection and you must issue a Set Connection or Connect command.

A Disconnect command issued against a Derby connection does not shut down the database or Derby (but the Exit command does).

Example

```
ij> connect 'jdbc:derby:menuDB;create=true';
ij> -- we create a new table in menuDB:
CREATE TABLE menu(course CHAR(10), ITEM char(20), PRICE integer);
0 rows inserted/updated/deleted
ij> disconnect;

ij> protocol 'jdbc:derby:';
ij> connect 'sample' as sample1;
ij> connect 'newDB;create=true' as newDB;
SAMPLE1 -    jdbc:derby:sample
NEWDB* -    jdbc:derby:newDB;create=true
* = current connection
ij(NEWDB)> set connection sample1;
ij> disconnect sample1;
ij> disconnect all;
ij>
```

Driver command

Syntax

```
DRIVER DriverNameString
```

Description

Takes the value of the *DriverNameString* and issues a *Class.forName* request to load the named class. The class is expected to be a JDBC driver that registers itself with *java.sql.DriverManager*.

If the Driver command succeeds, a new `ij` prompt appears for the next command.

Example

```
ij> -- load the Derby driver so that a connection
-- can be made:
driver 'org.apache.derby.jdbc.EmbeddedDriver';
```

```
ij> connect 'jdbc:derby:menuDB;create=true';
ij>
```

Elapsedtime command

Syntax

```
ELAPSEDTIME { ON | OFF }
```

Description

When `elapsedtime` is turned on, `ij` displays the total time elapsed during statement execution. The default value is OFF.

Example

```
ij> elapsedtime on;
ij> VALUES current_date;
1
----------
1998-07-15
ELAPSED TIME = 2134 milliseconds
ij>
```

Execute command

Syntax

```
EXECUTE { SQLString | PreparedStatementIdentifier }
[ USING { String | Identifier } ]
```

Description

Has several uses:

- To execute an SQL command entered as *SQLString*, using the Execute *String* style. The String is passed to the connection without further examination or processing by `ij`. *Normally, you execute SQL commands directly, not with the Execute command*.
- To execute a named command identified by *PreparedStatementIdentifier*. This must be previously prepared with the `ij` Prepare command.
- To execute either flavor of command when that command contains dynamic parameters, specify the values in the Using portion of the command. In this style, the *SQLString* or previously prepared *PreparedStatementIdentifier* is executed using the values supplied as *String* or *Identifier*. The *Identifier* in the USING clause must have a result set as its result. Each row of the result set is applied to the input parameters of the command to be executed, so the number of columns in the Using's result set must match the number of input parameters in the Execute's statement. The results of each execution of the Execute statement are displayed as they are made. If the Using's result set contains no rows, the Execute's statement is not executed.

  When auto-commit mode is on, the Using's result set is closed upon the first execution of the Execute statement. To ensure multiple-row execution of the Execute command, use the Autocommit command to turn auto-commit off.

Examples

```
ij> autocommit off;
ij> prepare menuInsert as 'INSERT INTO menu VALUES (?, ?, ?)';
ij> execute menuInsert using 'VALUES
    (''entree'', ''lamb chop'', 14),
    (''dessert'', ''creme brulee'', 6)';
1 row inserted/updated/deleted
1 row inserted/updated/deleted
```

```
ij> commit;
```

```
ij> connect 'jdbc:derby:firstdb;create=true';
ij> create table firsttable (id int primary key,
    name varchar(12));
0 rows inserted/updated/deleted
ij> insert into firsttable values
    (10,'TEN'),(20,'TWENTY'),(30,'THIRTY');
3 rows inserted/updated/deleted
ij> select * from firsttable;
ID         |NAME
------------------------
10         |TEN
20         |TWENTY
30         |THIRTY

3 rows selected
ij> connect 'jdbc:derby:seconddb;create=true';
ij(CONNECTION1)> create table newtable (newid int primary key,
    newname varchar(12));
0 rows inserted/updated/deleted
ij(CONNECTION1)> prepare src@connection0 as 'select * from firsttable';
ij(CONNECTION1)> autocommit off;
ij(CONNECTION1)> execute 'insert into newtable(newid, newname)
    values(?,?)' using src@connection0;
1 row inserted/updated/deleted
1 row inserted/updated/deleted
1 row inserted/updated/deleted
ij(CONNECTION1)> commit;
ij(CONNECTION1)> select * from newtable;
NEWID      |NEWNAME
------------------------
10         |TEN
20         |TWENTY
30         |THIRTY

3 rows selected
ij(CONNECTION1)> show connections;
CONNECTION0 -   jdbc:derby:firstdb
CONNECTION1* -  jdbc:derby:seconddb
ij(CONNECTION1)> disconnect connection0;
ij>
```

Exit command

Syntax

```
EXIT
```

Description

Causes the `ij` application to complete and processing to halt. Issuing this command from within a file started with the Run command or on the command line causes the outermost input loop to halt.

`ij` automatically shuts down a Derby database running in an embedded environment (issues a *Connect 'jdbc:derby:;shutdown=true*' request) on exit.

`ij` exits when the Exit command is entered or if given a command file on the Java invocation line, when the end of the command file is reached.

Example

```
ij> disconnect;
ij> exit;
C:\>
```

First command

Syntax

```
FIRST Identifier
```

Description

Moves the cursor to the first row in the *ResultSet*, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

Example

```
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> first scrollcursor;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
```

Get Cursor command

Syntax

```
GET [WITH {HOLD|NOHOLD}] CURSOR Identifier AS String
```

WITH HOLD is the default attribute of the cursor. For a non-holdable cursor, use the WITH NOHOLD option.

Description

Creates a cursor with the name of the *Identifier* by issuing a *java.sql.Statement.executeQuery* request on the value of the *String*.

If the *String* is a statement that does not generate a result set, the behavior of the underlying database determines whether an empty result set or an error is issued. If there is an error in executing the statement, no cursor is created.

`ij` sets the cursor name using a *java.sql.Statement.setCursorName* request. Behavior with respect to duplicate cursor names is controlled by the underlying database. Derby does not allow multiple open cursors with the same name.

Once a cursor has been created, the `ij` Next and Close commands can be used to step through its rows, and if the connection supports positioned update and delete commands, they can be issued to alter the rows.

Examples

```
ij> -- autocommit needs to be off so that the positioned update
ij> -- can see the cursor it operates against.
ij> autocommit off;
ij> get cursor menuCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> next menuCursor;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
ij> next menuCursor;
COURSE    |ITEM                |PRICE
----------------------------------------------
dessert   |creme brulee        |6
ij> UPDATE menu SET price=price+1 WHERE CURRENT OF menuCursor;
1 row inserted/updated/deleted
```

```
ij> next menuCursor;
COURSE    |ITEM                |PRICE
---------------------------------------------
appetizer |baby greens salad   |7
ij> close menuCursor;
ij> commit;
ij>
```

```
ij> connect 'jdbc:derby:memory:dummy;create=true;user=john'
    as john_conn;
ij> create table john_tbl(c int);
0 rows inserted/updated/deleted
ij> insert into john_tbl values(1),(2),(3);
3 rows inserted/updated/deleted
ij> connect 'jdbc:derby:memory:dummy;user=fred' as fred_conn;
ij(FRED_CONN)> get cursor john_cursor@john_conn
    as 'select * from john_tbl';
ij(FRED_CONN)> next john_cursor@john_conn;
C
-----------
1
ij(FRED_CONN)> next john_cursor@john_conn;
C
-----------
2
ij(FRED_CONN)> next john_cursor@john_conn;
C
-----------
3
ij(FRED_CONN)> next john_cursor@john_conn;
No current row
ij(FRED_CONN)> close john_cursor@john_conn;
ij(FRED_CONN)> disconnect all;
ij>
```

Get Scroll Insensitive Cursor command

Syntax

```
GET SCROLL INSENSITIVE [WITH {HOLD|NOHOLD}]
   CURSOR Identifier AS
   String
```

WITH HOLD is the default attribute of the cursor. For a non-holdable cursor, use the WITH NOHOLD option.

Description

Creates a scrollable insensitive cursor with the name of the *Identifier*. It does this by issuing a *createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)* call and then executing the statement with a *java.sql.StatementExecuteQuery* request on the value of the String.

If the *String* is a statement that does not generate a result set, the behavior of the underlying database determines whether an empty result set or an error is issued. If there is an error in executing the statement, no cursor is created.

`ij` sets the cursor name using a *java.sql.Statement.setCursorName* request. Behavior with respect to duplicate cursor names is controlled by the underlying database. Derby does not allow multiple open cursors with the same name.

Once a scrollable cursor has been created, you can use the following commands to work with the result set:

- Absolute command
- After Last command

- Before First command
- Close command
- First command
- Last command
- Next command
- Previous command
- Relative command

Examples

```
ij> autocommit off;
ij> get scroll insensitive cursor scrollCursor as
    'SELECT * FROM menu';
ij> absolute 5 scrollCursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
ij> after last scrollcursor;
No current row
ij> before first scrollcursor;
No current row
ij> first scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
ij> last scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
dessert   |creme brulee        |6
ij> previous scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
ij> relative 1 scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
dessert   |creme brulee        |6
ij>>previous scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
dessert   |creme brulee        |6
ij> next scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
dessert   |creme brulee        |6
```

```
ij> connect 'jdbc:derby:memory:dummy;create=true;user=john'
    as john_conn;
ij> create table john_tbl(c int);
0 rows inserted/updated/deleted
ij> insert into john_tbl values(1),(2),(3);
3 rows inserted/updated/deleted
ij> connect 'jdbc:derby:memory:dummy;user=fred' as fred_conn;
ij(FRED_CONN)> get scroll insensitive cursor john_cursor@john_conn
    as 'select * from john_tbl';
ij(FRED_CONN)> next john_cursor@john_conn;
C
-----------
1
ij(FRED_CONN)> getcurrentrownumber john_cursor@john_conn;
1
ij(FRED_CONN)> last john_cursor@john_conn;
C
-----------
3
ij(FRED_CONN)> previous john_cursor@john_conn;
C
```

```
-----------
2
ij(FRED_CONN)> first john_cursor@john_conn;
C
-----------
1
ij(FRED_CONN)> after last john_cursor@john_conn;
No current row
ij(FRED_CONN)> before first john_cursor@john_conn;
No current row
ij(FRED_CONN)> relative 2 john_cursor@john_conn;
C
-----------
2
ij(FRED_CONN)> absolute 1 john_cursor@john_conn;
C
-----------
1
ij(FRED_CONN)> close john_cursor@john_conn;
ij(FRED_CONN)> disconnect all;
ij>
```

Help command

Syntax

```
HELP
```

Description

Prints out a brief list of the `ij` commands.

Last command

Syntax

```
LAST Identifier
```

Description

Moves the cursor to the last row in the *ResultSet*, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

Example

```
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> absolute 5 scrollCursor;
COURSE    |ITEM                |PRICE
---------------------------------------------
entree    |lamb chop           |14
ij> last scrollcursor;
COURSE    |ITEM                |PRICE
---------------------------------------------
dessert   |creme brulee        |6
```

LocalizedDisplay command

Syntax

```
LOCALIZEDDISPLAY { on | off }
```

Description

Specifies to display locale-sensitive data (such as dates) in the native format for the `ij` locale. The `ij` locale is the same as the Java system locale.

Note: NUMERIC and DECIMAL values are not localized when using the J2ME/CDC/Foundation Profile because of platform limitations.

Example

The following demonstrates *LocalizedDisplay* in an English locale:

```
ij> VALUES CURRENT_DATE;
1
----------
2000-05-01
1 row selected
ij> localizeddisplay on;
ij> VALUES CURRENT_DATE;
1
------------------
May 1, 2000
1 row selected
```

MaximumDisplayWidth command

Syntax

```
MAXIMUMDISPLAYWIDTH integer_value
```

Description

Sets the largest display width for columns to the specified value. This is generally used to increase the default value in order to display large blocks of text.

Example

```
ij> maximumdisplaywidth 3;
ij> VALUES 'NOW IS THE TIME!';
1
---
NOW
ij> maximumdisplaywidth 30;
ij> VALUES 'NOW IS THE TIME!';
1
----------------
NOW IS THE TIME!
```

Next command

Syntax

```
NEXT Identifier
```

Description

Fetches the next row from the named cursor created with the Get Cursor command or Get Scroll Insensitive Cursor. It displays a banner and the values of the row.

Example

```
ij> get cursor menuCursor as 'SELECT * FROM menu';
ij> next menuCursor;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
ij>
```

Prepare command

Syntax

```
PREPARE Identifier AS String
```

Description

Creates a *java.sql.PreparedStatement* using the value of the String, accessible in `ij` by the *Identifier* given to it. If a prepared statement with that name already exists in `ij`, an error will be returned and the previous prepared statement will remain. Use the Remove command to remove the previous statement first. If there are any errors in preparing the statement, no prepared statement is created.

Any SQL statements allowed in the underlying connection's prepared statement can be prepared with this command.

If the *Identifier* specifies a *connectionName*, the statement is prepared on the specified connection.

Examples

```
ij> prepare seeMenu as 'SELECT * FROM menu';
ij> execute seeMenu;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |6

2 rows selected
ij>
```

```
ij> connect 'jdbc:derby:firstdb;create=true';
ij> create table firsttable (id int primary key,
    name varchar(12));
0 rows inserted/updated/deleted
ij> insert into firsttable values
    (10,'TEN'),(20,'TWENTY'),(30,'THIRTY');
3 rows inserted/updated/deleted
ij> select * from firsttable;
ID         |NAME
------------------------
10         |TEN
20         |TWENTY
30         |THIRTY

3 rows selected
ij> connect 'jdbc:derby:seconddb;create=true';
ij(CONNECTION1)> create table newtable (newid int primary key,
    newname varchar(12));
0 rows inserted/updated/deleted
ij(CONNECTION1)> prepare src@connection0 as 'select * from firsttable';
ij>
```

Previous command

Syntax

```
PREVIOUS Identifier
```

Description

Moves the cursor to the row previous to the current one, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

Example

```
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> last scrollcursor;
COURSE    |ITEM                |PRICE
---------------------------------------------
dessert   |creme brulee        |6
ij> previous scrollcursor;
COURSE    |ITEM                |PRICE
---------------------------------------------
entree    |lamb chop           |14
```

Protocol command

Syntax

```
PROTOCOL String [ AS Identifier ]
```

Description

Specifies the protocol, as a String, for establishing connections and automatically loads the appropriate driver. *Protocol* is the part of the database connection URL syntax appropriate for your environment, including the JDBC protocol and the protocol specific to Derby. For further information about the Derby database connection URL, see the *Java DB Developer's Guide*. Only Derby protocols are supported. Those protocols are listed in ij.protocol property.

Providing a protocol allows you to use a shortened database connection URL for connections. You can provide only the database name (and a subsubprotocol name if needed) instead of the full protocol. In addition, you do not need to use the Driver command or specify a driver at start-up, since the driver is loaded automatically.

If you name the protocol, you can refer to the protocol name in the Connect command.

Examples

```
ij> protocol 'jdbc:derby:';
ij> connect 'sample';
```

```
ij> protocol 'jdbc:derby:';
ij> connect 'memory:sample;create=true';
```

```
ij> protocol 'jdbc:derby:memory:';
ij> connect 'sample;create=true';
```

Readonly command

Syntax

```
READONLY { ON | OFF }
```

Description

Sets the current connection to a "read-only" connection, as if the current user were defined as a *readOnlyAccess* user. (For more information about database authorization, see the *Java DB Developer's Guide*.)

Example

```
ij> connect 'jdbc:derby:menuDB';
ij> readonly on;
ij> SELECT * FROM menu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |6
appetizer |baby greens         |7
entree    |lamb chop           |14
entree    |lamb chop           |14
dessert   |creme brulee        |6
6 rows selected
ij> UPDATE menu set price = 3;
ERROR 25502: An SQL data change is not permitted for a read-only
connection, user or database.
```

Relative command

Syntax

```
RELATIVE int Identifier
```

Description

Moves the cursor to the row that is *int* number of rows relative to the current row, then fetches the row. The cursor must have been created with the Get Scroll Insensitive Cursor command. It displays a banner and the values of the row.

Example

```
ij> -- autocommit needs to be off so that the positioned update
ij> -- can see the cursor it operates against.
ij> autocommit off;
ij> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
ij> last scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
dessert   |creme brulee        |6
ij> previous scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
ij> relative 1 scrollcursor;
COURSE    |ITEM                |PRICE
-----------------------------------------------
dessert   |creme brulee        |6
```

Remove command

Syntax

```
REMOVE Identifier
```

Description

Removes a previously prepared statement from ij. The identifier is the name by which the statement was prepared. The statement is closed to release its database resources.

Example

```
ij> prepare seeMenu as 'SELECT * FROM menu';
ij> execute seeMenu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
```

```
dessert   |creme brulee        |6

2 rows selected
ij> remove seeMenu;
ij> execute seeMenu;
IJ ERROR: Unable to establish prepared statement SEEMENU
ij>
```

Rollback command

Syntax

```
ROLLBACK
```

Description

Issues a *java.sql.Connection.rollback* request. Use only if auto-commit is off. A *java.sql.Connection.rollback* request undoes the currently active transaction and initiates a new transaction.

Example

```
ij> autocommit off;
ij> INSERT INTO menu VALUES ('dessert', 'rhubarb pie', 4);
1 row inserted/updated/deleted
ij> SELECT * from menu;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |7
appetizer |baby greens         |7
dessert   |rhubarb pie         |4

4 rows selected
ij> rollback;
ij> SELECT * FROM menu;
COURSE    |ITEM                |PRICE
----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |7
appetizer |baby greens         |7

3 rows selected
ij>
```

Run command

Syntax

```
RUN String
```

Description

Assumes that the value of the string is a valid file name, and redirects `ij` processing to read from that file until it ends or an Exit command is executed. If the end of the file is reached without `ij` exiting, reading will continue from the previous input source once the end of the file is reached. Files can contain Run commands.

`ij` prints out the statements in the file as it executes them.

Any changes made to the `ij` environment by the file are visible in the environment when processing resumes.

Example

```
ij> run 'setupMenuConn.ij';
ij> -- this is setupMenuConn.ij
-- ij displays its contents as it processes file
ij> connect 'jdbc:derby:menuDB';
ij> autocommit off;
ij> -- this is the end of setupMenuConn.ij
-- there is now a connection to menuDB and no autocommit.
-- input will now resume from the previous source.
;
ij>
```

Set Connection command

Syntax

```
SET CONNECTION Identifier
```

Description

Allows you to specify which connection to make current when you have more than one connection open. Use the Show Connections command to display open connections.

If there is no such connection, an error results and the current connection is unchanged.

Example

```
ij> protocol 'jdbc:derby:';
ij> connect 'sample' as sample1;
ij> connect 'newDB;create=true' as newDB;
ij (NEWDB)> show connections;
SAMPLE1 -    jdbc:derby:sample
NEWDB* -    jdbc:derby:newDB;create=true
* = current connection
ij(NEWDB)> set connection sample1;
ij(SAMPLE1)> disconnect all;
ij>
```

Show command

Syntax

```
SHOW
{
   CONNECTIONS |
   FUNCTIONS [ IN schemaName ] |
   INDEXES [ IN schemaName | FROM table-Name ] |
   PROCEDURES [ IN schemaName ] |
   ROLES |
   ENABLED_ROLES |
   SETTABLE_ROLES |
   SCHEMAS |
   SYNONYMS [ IN schemaName ] |
   TABLES [ IN schemaName ] |
   VIEWS [ IN schemaName ] |

}
```

Description

The SHOW command can be used to display information about active connections and database objects.

SHOW CONNECTIONS

If there are no connections, the SHOW CONNECTIONS command returns "No connections available".

Otherwise, the command displays a list of connection names and the URLs used to connect to them. The currently active connection, if there is one, is marked with an * after its name.

Example

```
ij> connect 'sample' as sample1;
ij> connect 'newDB;create=true' as newDB;
ij(NEWDB)> show connections;
SAMPLE1 -        jdbc:derby:sample
NEWDB* -        jdbc:derby:newDB;create=true
* = current connection
ij(NEWDB)>
```

SHOW FUNCTIONS

SHOW FUNCTIONS displays all functions in the database. By default, both system functions and user-defined functions appear in the output.

If **IN *schemaName*** is specified, then only the functions in the specified schema are displayed.

Example

If you created the TO_DEGREES function described in "CREATE FUNCTION statement" in the *Java DB Reference Manual*, the output of the CREATE FUNCTION and SHOW FUNCTIONS commands would look something like the following:

```
ij> connect 'jdbc:derby:firstdb';

ij> CREATE FUNCTION TO_DEGREES ( RADIANS DOUBLE )
> RETURNS DOUBLE
> PARAMETER STYLE JAVA
> NO SQL LANGUAGE JAVA
> EXTERNAL NAME 'java.lang.Math.toDegrees';
0 rows inserted/updated/deleted
ij> show functions in app;
FUNCTION_SCHEM|FUNCTION_NAME                |REMARKS
------------------------------------------------------------------------------
APP           |TO_DEGREES                   |java.lang.Math.toDegrees


1 row selected
```

SHOW INDEXES

SHOW INDEXES displays all the indexes in the database.

If **IN *schemaName*** is specified, then only the indexes in the specified schema are displayed.

If **FROM *table-Name*** is specified, then only the indexes on the specified table are displayed.

Example

```
ij> show indexes in app;
TABLE_NAME          |COLUMN_NAME         |NON_U&|TYPE|ASC&|CARDINA&|PAGES
------------------------------------------------------------------------------
---
AIRLINES            |AIRLINE             |false |3   |A   |NULL    |NULL
COUNTRIES           |COUNTRY_ISO_CODE    |false |3   |A   |NULL    |NULL
COUNTRIES           |COUNTRY             |false |3   |A   |NULL    |NULL
CITIES              |CITY_ID             |false |3   |A   |NULL    |NULL
FLIGHTS             |FLIGHT_ID           |false |3   |A   |NULL    |NULL
FLIGHTS             |SEGMENT_NUMBER      |false |3   |A   |NULL    |NULL
FLIGHTAVAILABILITY  |FLIGHT_ID           |false |3   |A   |NULL    |NULL
FLIGHTAVAILABILITY  |SEGMENT_NUMBER      |false |3   |A   |NULL    |NULL
```

```
FLIGHTAVAILABILITY  |FLIGHT_DATE         |false |3   |A   |NULL    |NULL
MAPS                |MAP_ID              |false |3   |A   |NULL    |NULL
MAPS                |MAP_NAME            |false |3   |A   |NULL    |NULL
FLIGHTS             |DEST_AIRPORT        |true  |3   |A   |NULL    |NULL
FLIGHTS             |ORIG_AIRPORT        |true  |3   |A   |NULL    |NULL
CITIES              |COUNTRY_ISO_CODE    |true  |3   |A   |NULL    |NULL
FLIGHTAVAILABILITY  |FLIGHT_ID           |true  |3   |A   |NULL    |NULL
FLIGHTAVAILABILITY  |SEGMENT_NUMBER      |true  |3   |A   |NULL    |NULL

16 rows selected
ij> show indexes from flights;
TABLE_NAME          |COLUMN_NAME         |NON_U&|TYPE|ASC&|CARDINA&|PAGES
------------------------------------------------------------------------------
---
FLIGHTS             |FLIGHT_ID           |false |3   |A   |NULL    |NULL
FLIGHTS             |SEGMENT_NUMBER      |false |3   |A   |NULL    |NULL
FLIGHTS             |DEST_AIRPORT        |true  |3   |A   |NULL    |NULL
FLIGHTS             |ORIG_AIRPORT        |true  |3   |A   |NULL    |NULL

4 rows selected
```

SHOW PROCEDURES

SHOW PROCEDURES displays all the procedures in the database that have been created with the CREATE PROCEDURE statement, as well as system procedures.

If **IN *schemaName*** is specified, only procedures in the specified schema are displayed.

Example

```
ij> show procedures in syscs_util;
PROCEDURE_SCHEM     |PROCEDURE_NAME                |REMARKS
------------------------------------------------------------------------
SYSCS_UTIL          |SYSCS_BACKUP_DATABASE         |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_BACKUP_DATABASE_AND_ENA&|org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_BACKUP_DATABASE_AND_ENA&|org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_BACKUP_DATABASE_NOWAIT  |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_BULK_INSERT             |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_CHECKPOINT_DATABASE     |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_COMPRESS_TABLE          |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_DISABLE_LOG_ARCHIVE_MODE|org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_EXPORT_QUERY            |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_EXPORT_TABLE            |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_FREEZE_DATABASE         |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_IMPORT_DATA             |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_IMPORT_TABLE            |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_INPLACE_COMPRESS_TABLE  |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_SET_DATABASE_PROPERTY   |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_SET_RUNTIMESTATISTICS   |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_SET_STATISTICS_TIMING   |org.apache.derby.ca&
SYSCS_UTIL          |SYSCS_UNFREEZE_DATABASE       |org.apache.derby.ca&

18 rows selected
```

SHOW ROLES, SHOW ENABLED_ROLES, SHOW SETTABLE_ROLES

SHOW ROLES displays the names of all roles created, whether settable for the current session or not.

SHOW ENABLED_ROLES displays the names of all the roles whose privileges are available for the current session. That is, it shows the current role and any role contained in the current role. (For a definition of role containment, see "Using SQL roles" in the *Java DB Developer's Guide*.)

SHOW SETTABLE_ROLES displays all the roles that the current session can set, that is, all roles that have been granted to the current user or to PUBLIC.

The roles shown by these commands are sorted in ascending order.

Example

```
ij> show roles;
ROLEID
------------------------------
ANYUSER
CASUALUSER
POWERUSER

3 rows selected
ij> show enabled_roles;
ROLEID
------------------------------
ANYUSER
CASUALUSER

2 rows selected
ij> show settable_roles;
ROLEID
------------------------------
CASUALUSER
POWERUSER

2 rows selected
```

In the examples above, both CASUALUSER and POWERUSER contain ANYUSER, but ANYUSER is not settable directly.

SHOW SCHEMAS

SHOW SCHEMAS displays all of the schemas in the current connection.

Example

```
ij> show schemas;
TABLE_SCHEM
------------------------------
APP
NULLID
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT

11 rows selected
```

SHOW SYNONYMS

SHOW SYNONYMS displays all the synonyms in the database that have been created with the CREATE SYNONYMS statement.

If **IN *schemaName*** is specified, only synonyms in the specified schema are displayed.

Example

```
ij> show synonyms;
TABLE_SCHEM         |TABLE_NAME                    |REMARKS
------------------------------------------------------------------------------
APP                 |MYAIRLINES                    |
```

SHOW TABLES

SHOW TABLES displays all of the tables in the current schema.

If **IN *schemaName*** is specified, the tables in the given schema are displayed.

Example

```
ij> show tables;
TABLE_SCHEM         |TABLE_NAME                    |REMARKS
------------------------------------------------------------------------
APP                 |AIRLINES                      |
APP                 |CITIES                        |
APP                 |COUNTRIES                     |
APP                 |FLIGHTAVAILABILITY            |
APP                 |FLIGHTS                       |
APP                 |FLIGHTS_HISTORY               |
APP                 |MAPS                          |

7 rows selected
```

SHOW VIEWS

SHOW VIEWS displays all of the views in the current schema.

If **IN *schemaName*** is specified, the views in the given schema are displayed.

Example

```
ij> show views;
TABLE_SCHEM         |TABLE_NAME                    |REMARKS
------------------------------------------------------------------------
APP                 |TOTALSEATS                    |

1 row selected
```

Wait For command

Syntax

```
WAIT FOR Identifier
```

Description

Displays the results of a previously started asynchronous command.

The identifier for the asynchronous command must have been used in a previous Async command on this connection. The Wait For command waits for the SQL statement to complete execution, if it has not already, and then displays the results. If the statement returns a result set, the Wait For command steps through the rows, not the Async command. This might result in further execution time passing during the result display.

Example

See Async command.

Syntax for comments in ij commands

Syntax

```
-- Text

/* Text */
```

Description

You can use a double dash to create a comment anywhere within an `ij` command line and as permitted by the underlying connection within SQL commands. The comment is ended at the first new line encountered in the text.

Comments are ignored on input and have no effect on the output displayed.

You can also enclose text in `/* */` characters to create either one-line or multi-line comments. Nested comments are permitted. For example, you could put lines like the following into a script named `comment.sql`:

```
/* start the file with a /* nested comment */ and see what happens */
connect 'jdbc:derby:newdb;create=true';
values 'hi!';
create table t (x int);
/* use a multi-line comment */
/*
insert into t values 1, 2, 3;
insert into t values 4, 5, 6;
*/
/* end the file with a comment */
values 'This is a test';
/* This is also a test */
```

Examples

```
ij> -- this is a comment;
-- the semicolons in the comment are not taken as the end
-- of the command; for that, we put it outside the --:
;
ij>
```

```
ij> run 'comment.sql';
ij> /* start the file with a /* nested comment */ and see what happens */
connect 'jdbc:derby:newdb;create=true';
ij> values 'hi!';
1
---
hi!

1 row selected
ij> create table t (x int);
0 rows inserted/updated/deleted
ij> /* use a multi-line comment */
/*
insert into t values 1, 2, 3;
insert into t values 4, 5, 6;
*/
/* end the file with a comment */
values 'This is a test';
1
--------------
This is a test

1 row selected
ij> /* This is also a test */
;
ij>
```

Syntax for identifiers in ij commands

Syntax

```
Identifier [ @ connectionName ]
```

Description

Some `ij` commands require identifiers. These `ij` identifiers are case-insensitive. They must begin with a letter in the range A-Z, and can consist of any number of letters in the range A-Z, digits in the range 0-9, and underscore (_) characters.

An identifier can optionally use an at sign (`@`) followed by a *connectionName*. Spaces on either side of the `@` sign are optional. If you specify a *connectionName*, you can refer to databases on different connections. This capability enables you to perform tasks such as copying data from one database to another. For an example of copying data between databases, see Execute command. For other examples, see Async command, Get Cursor command, and Get Scroll Insensitive Cursor command.

These identifiers exist within the scope of `ij` only and are distinct from any identifiers used in SQL commands, except in the case of the Get Cursor command. The Get Cursor command specifies a cursor name to use in creating a result set.

`ij` does not recognize or permit delimited identifiers in `ij` commands. They can be used in SQL commands.

Example

```
These are valid ij identifiers:
foo1
exampleIdentifier12345
another_one
myId@connection0
id2 @ connection1
```

Syntax for strings in ij commands

Syntax

```
'Text'
```

Description

Some `ij` commands require strings. `ij` strings are represented by the same literal format as SQL strings and are delimited by single quotation marks. To include a single quotation mark in a string, you must use two single quotation marks, as shown in the examples below. `ij` places no limitation on the lengths of strings, and will treat embedded new lines in the string as characters in the string.

Some `ij` commands execute SQL commands specified as strings. Therefore, you must double any single quotation marks within such strings, as shown in the second example below.

The cases of letters within a string are preserved.

Example

```
This is a string in ij      And this is its value
'Mary''s umbrella'           Mary's umbrella
'hello world'                hello world

--returns Joe's
execute 'VALUES ''Joe''''s''';
```

ij errors

ij might issue messages to inform the user of errors during processing of statements.

ERROR SQLState

When the underlying JDBC driver returns an *SQLException*, `ij` displays the *SQLException* message with the prefix "ERROR SQLState". If the *SQLException* has no SQLState associated with it, the prefix "ERROR (no SQLState)" is used.

WARNING SQLState

Upon completion of execution of any JDBC request, `ij` will issue a *getWarnings* request and display the SQLWarnings that are returned. Each *SQLWarning* message is displayed with the prefix "WARNING *SQLState*". If an *SQLWarning* has no SQLState associated with it, the prefix "WARNING (no SQLState)" is used.

IJ ERROR

When `ij` runs into errors processing user commands, such as being unable to open the file named in a Run command or not having a connection to disconnect from, it prints out a message with the prefix "IJ ERROR".

IJ WARNING

`ij` displays warning messages to let the user know if behavior might be unexpected. `ij` warnings are prefixed with "IJ WARNING".

JAVA ERROR

When an unexpected Java exception occurs, `ij` prints a message with the prefix "JAVA ERROR".

sysinfo

Use the `sysinfo` utility to display information about your Java environment and Derby (including version information). To use `sysinfo`, do one of the following:

- If you are relatively new to the Java programming language, follow the instructions in "Setting up your environment" in *Getting Started with Java DB* to set the `DERBY_HOME` and `JAVA_HOME` environment variables and to add `DERBY_HOME/bin` to your path. Then use the following command:

  ```
  sysinfo
  ```

- If you are a regular Java user but are new to Derby, set the `DERBY_HOME` environment variable, then use a `java` command to invoke the `derbyrun.jar` file:

  ```
  (UNIX) java [options] -jar $DERBY_HOME/lib/derbyrun.jar sysinfo
  ```

  ```
  (Windows) java [options] -jar %DERBY_HOME%\lib\derbyrun.jar sysinfo
  ```

- If you are familiar with both the Java programming language and Derby, you have already set `DERBY_HOME`. Set your classpath to include the Derby jar files. Then use a `java` command to invoke the `sysinfo` class directly.

  ```
  java org.apache.derby.tools.sysinfo
  ```

sysinfo example

When you run the `sysinfo` command using the `derbyrun.jar` file, the output looks something like this:

```
java -jar C:\db-derby-10.10.0.0-bin\lib\derbyrun.jar sysinfo
------------------ Java Information ------------------
Java Version:    1.7.0_07
Java Vendor:     Oracle Corporation
Java home:       C:\jdk1.7.0_07\jre
Java classpath:  C:\db-derby-10.10.0.0-bin\lib\derbyrun.jar
OS name:         Windows XP
OS architecture: x86
OS version:      5.1
Java user name:  user1
Java user home:  C:\Documents and Settings\user1
Java user dir:   C:\DERBYDBS
java.specification.name: Java Platform API Specification
java.specification.version: 1.7
java.runtime.version: 1.7.0_07-b11
--------- Derby Information --------
[C:\db-derby-10.10.0.0-bin\lib\derby.jar] 10.10.0.0 - (1384314)
[C:\db-derby-10.10.0.0-bin\lib\derbytools.jar] 10.10.0.0 - (1384314)
[C:\db-derby-10.10.0.0-bin\lib\derbynet.jar] 10.10.0.0 - (1384314)
[C:\db-derby-10.10.0.0-bin\lib\derbyclient.jar] 10.10.0.0 - (1384314)
------------------------------------------------------
----------------- Locale Information -----------------
Current Locale : [English/United States [en_US]]
Found support for locale: [cs]
         version: 10.10.0.0 - (1384314)
Found support for locale: [de_DE]
         version: 10.10.0.0 - (1384314)
Found support for locale: [es]
         version: 10.10.0.0 - (1384314)
Found support for locale: [fr]
         version: 10.10.0.0 - (1384314)
Found support for locale: [hu]
         version: 10.10.0.0 - (1384314)
Found support for locale: [it]
```

```
         version: 10.10.0.0 - (1384314)
Found support for locale: [ja_JP]
         version: 10.10.0.0 - (1384314)
Found support for locale: [ko_KR]
         version: 10.10.0.0 - (1384314)
Found support for locale: [pl]
         version: 10.10.0.0 - (1384314)
Found support for locale: [pt_BR]
         version: 10.10.0.0 - (1384314)
Found support for locale: [ru]
         version: 10.10.0.0 - (1384314)
Found support for locale: [zh_CN]
         version: 10.10.0.0 - (1384314)
Found support for locale: [zh_TW]
         version: 10.10.0.0 - (1384314)
-----------------------------------------------------
```

When you request help for a problem by posting to the derby-user mailing list, include a copy of the information provided by the `sysinfo` utility.

Using sysinfo to check the classpath

sysinfo provides an argument (*-cp*) which can be used to test the classpath.

```
java org.apache.derby.tools.sysinfo -cp
[ [ embedded ][ server ][ client] [ tools ] [ anyClass.class ] ]
```

If your environment is set up correctly, the utility shows output indicating success.

You can provide optional arguments with *-cp* to test different environments. Optional arguments to *-cp* are:

- embedded
- server
- client
- tools
- *classname*.class

If something is missing from your classpath, the utility indicates what is missing. For example, if you neglected to include the directory containing the class named *SimpleApp* to your classpath, the utility would indicate this when the following command line was issued (type all on one line):

```
$ java org.apache.derby.tools.sysinfo -cp embedded SimpleApp.class
FOUND IN CLASS PATH:

Derby embedded engine library (derby.jar)


NOT FOUND IN CLASS PATH:

user-specified class (SimpleApp)
(SimpleApp not found.)
```

dblook

Use the `dblook` utility to view all or parts of the Data Definition Language (DDL) for a given database. To use the `dblook` utility, do one of the following:

- If you are relatively new to the Java programming language, follow the instructions in "Setting up your environment" in *Getting Started with Java DB* to set the `DERBY_HOME` and `JAVA_HOME` environment variables and to add `DERBY_HOME/bin` to your path. Then use the following command:

  ```
  dblook -d connectionURL [options]
  ```
- If you are a regular Java user but are new to Derby, set the `DERBY_HOME` environment variable, then use a `java` command to invoke the `derbyrun.jar` file (all on one line):

  ```
  (UNIX) java [options] -jar $DERBY_HOME/lib/derbyrun.jar dblook
             -d connectionURL [options]
  ```

  ```
  (Windows) java [options] -jar %DERBY_HOME%\lib\derbyrun.jar dblook
             -d connectionURL [options]
  ```
- If you are familiar with both the Java programming language and Derby, you have already set `DERBY_HOME`. Set your classpath to include the Derby jar files. Then use a `java` command to invoke the `dblook` class directly.

  ```
  java org.apache.derby.tools.dblook -d connectionURL [options]
  ```

Using dblook

The syntax for the command to launch the `dblook` utility is:

```
dblook -d connectionURL [options]
```

The value for *connectionURL* is the complete URL for the database. Where appropriate, the URL includes any connection URL attributes that might be required to access the database. For complete information on connection URL attributes, see "Setting attributes for the database connection URL" in the *Java DB Reference Manual*.

For example, to connect to the database 'myDB', the URL would simply be `'jdbc:derby:myDB';` to connect using the Network Server to a database 'C:\private\tmp\myDB' on a remote server (port 1527), the URL would be:

```
'jdbc:derby://localhost:1527/
"C:\private\tmp\myDB";user=someusr;password=somepwd'
```

As with other Derby utilities, you must ensure that no other JVMs are started against the database when you call the `dblook` utility, or an exception will occur and will print to the `dblook.log`. If this exception is thrown, the `dblook` utility will quit. To recover, you must ensure that no other Derby applications running in a separate JVM are connected to the source database. These connections need to be shutdown. Once all existing JVMs running against the database have been shutdown, the `dblook` utility will execute successfully. You can also start the Derby Network server and run the `dblook` utility as a client application while other clients are connected to the server.

dblook options

The `dblook` utility options include:

-z <schemaName>

specifies the schema to which the DDL should be restricted. Only objects with the specified schema are included in the DDL file.

-t <tableOne> <tableTwo> ...
specifies the tables to which the DDL should be restricted. All tables with a name from this list will be included in the DDL file subject to `-z` limitations, as will the DDL for any keys, checks, or indexes on which the table definitions depend.

Additionally, if the statement text of any triggers or views includes a reference to any of the listed table names, the DDL for that trigger/view will also be generated, subject to `-z` limitations. If a table is not included in this list, then neither the table nor any of its keys, checks, or indexes will be included in the final DDL. If this parameter is not provided, all database objects will be generated, subject to `-z` limitations. Table names are separated by whitespace.

-td
specifies a statement delimiter for SQL statements generated by dblook. If a statement delimiter option is not specified, the default is the semicolon (';'). At the end of each DDL statement, the delimiter is printed, followed by a new line.

-o <filename>
specifies the file where the generated DDL is written. If this file is not specified, it defaults to the console (i.e. standard System.out).

-append
prevents overwriting the DDL output ('-o' parameter, if specified) and "dblook.log" files. If this parameter is specified, and execution of the `dblook` command leads to the creation of files with names identical to existing files in the current directory, `dblook` will append to the existing files. If this parameter is not set, the existing files will be overridden.

-verbose
specifies that all errors and warnings (both SQL and internal to `dblook`) should be echoed to the console (via System.err), in addition to being printed to the "dblook.log" file. If this parameter is not set, the errors and warnings only go to the "dblook.log" file.

-noview
specifies that CREATE VIEW statements should not be generated.

Generating the DDL for a database

The `dblook` utility generates all of the following objects when generating the DDL for a database:

- Checks
- Functions
- Indexes
- Jar files
- Keys (primary, foreign, and unique)
- Schemas
- Stored procedures
- Triggers
- Tables
- Views
- Roles
- User-defined types
- User-defined aggregates

Note: When dblook runs against a database that has jar files installed, it will create a new directory, called `DERBYJARS`, within the current directory, and that is where it will

keep copies of all of the jars it encounters. In order to run the generated DDL as a script, this `DERBYJARS` directory must either 1) exist within the directory in which it was created, or 2) be moved manually to another directory, in which case the path in the generated DDL file must be manually changed to reflect to the new location.

The `dblook` utility ignores any objects that have system schemas (for example, SYS, SYSIBM), since DDL is not able to directly create nor modify system objects.

dblook examples

The following examples demonstrate how the various `dblook` utility options can be specified from a command line. These examples use the `sample` database.

Note: The quotations marks shown in these examples are part of the command argument and must be passed to `dblook`. The way in which quotation marks are passed depends on the operating system and command line that you are using. With some systems it might be necessary to escape the quotation marks by using a forward slash before the quotation mark, for example: `"\"My Table"\"`

Status messages are written to the output (either a -o filename, if specified, or the console) as SQL script comments. These status messages serve as headers to show which types of database objects are being, or have been, processed by the `dblook` utility.

Writing the DDL to the console
You can write the DDL to the console for everything that is in the `sample` database. In this example, the database is in the current directory. For example:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample
```

Including error and warning messages in the dblook command
You can write error and warning messages when you write the DDL to the console. The messages are written using `System.err`. For example:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample -verbose
```

Writing the DDL to a file
You can write the DDL to a file called `myDB_DDL.sql` for everything that is in the `sample` database. In this example, the database and file are in the current directory. For example:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample -o myDB_DDL.sql
```

Specifying directory paths in the dblook command
If the database or file are not in the current directory, you must specify the directory paths. For example:

```
java org.apache.derby.tools.dblook -d
 'jdbc:derby:c:\private\stuff\sample'
   -o 'C:\temp\newDB.sql'
```

Specifying a schema in the dblook command
You can specify the schema for the database. To write the DDL to the console, for all of the objects in the `sample` database where the database is in the SAMP schema, use the following command:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample -z samp
```

Specifying a remote database and host

If the `sample` database is in the SAMP schema on localhost:1527, you must specify your user ID and password. For example, use the following command to write the DDL to the console:

```
java org.apache.derby.tools.dblook
   -d 'jdbc:derby://localhost:1527/"C:\temp\sample";
   user=someusername;password=somepassword' -z samp
```

Specifying a schema and a table within the database in the dblook command
You can specify that only the objects in the `sample` database that are associated with the SAMP and the `My Table` table are written to the console. For example:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample -z samp -t "My
 Table"
```

Specifying multiple tables in the dblook command
You can specify more than one table in the dblook command by separating the names of the tables with a space. For example, for objects in the `sample` database that are associated with either the `My Table` table or the `STAFF` table, use the following command:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample -t "My Table"
 staff
```

Writing DDL to a file without a statement delimiter
To write the DDL for all of the objects in`sample` database to the `myDB_DDL.sql` file without a statement delimiter, you must omit the default semi-colon. You can append the DDL to the output files if the files are already there. For example:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample
   -o myDB_DDL.sql -td '' -append
```

Excluding views from the DDL
To write the DDL to the console for all of the objects in the `sample` database except for views, use the following command:

```
java org.apache.derby.tools.dblook -d jdbc:derby:sample -noview
```

SignatureChecker

Use the `SignatureChecker` tool to identify any SQL functions and procedures in a database that do not follow the SQL argument matching rules described in "Argument matching" in the *Java DB Reference Manual*. If your application uses SQL functions and/or procedures, you should run this tool against your databases.

Using SignatureChecker

Before you run the `SignatureChecker` tool, make sure that your classpath contains the Derby jar files, including *derbytools.jar*.

On a Java SE platform, run the `SignatureChecker` tool as follows, where *connection-url-to-database* is the connection URL you would use in order to obtain a connection by calling *DriverManager.getConnection()*:

```
java org.apache.derby.tools.SignatureChecker connection-url-to-database
```

Alternatively, you can invoke the tool using *derbyrun.jar*. For example:

```
java -jar derbyrun.jar SignatureChecker "jdbc:derby:myDB"
```

On a Java ME platform, run the `SignatureChecker` tool as follows, where *database-name* is the database name you would set by calling *EmbeddedSimpleDataSource.setDatabaseName()*:

```
java org.apache.derby.tools.SignatureChecker database-name
```

The tool examines every routine registered in the database and displays results like the following:

```
Found a matching method for: "APP"."DOINSERT"( )
Found a matching method for: "APP"."DOINSERTANDCOMMIT"( )
Found a matching method for: "APP"."APPENDFOOANDBAR"( VARCHAR )
Unresolvable routine: "APP"."IDONTEXIST"( VARCHAR , INTEGER ).
Detailed reason: No method was found that matched the method call
 z.iDontExist(java.lang.String, int),
tried all combinations of object and primitive types and any possible
 type conversion for any parameters the method call may have.
The method might exist but it is not public and/or static, or the
 parameter types are not method invocation convertible.
Found a matching method for: "APP"."RUNDDL"( VARCHAR )
Unresolvable routine: "APP"."TABFUNCDOESNTEXIST"( VARCHAR , BIGINT ).
Detailed reason: No method was found that matched the method call
 org.apache.derbyTesting.functionTests.tests.lang.TableFunctionTest.
 appendFooAndBar(java.lang.String, long),
tried all combinations of object and primitive types and any possible
 type conversion for any parameters the method call may have.
The method might exist but it is not public and/or static, or the
 parameter types are not method invocation convertible.
```

In the example above, the `SignatureChecker` tool found matches for all routines except for the functions *app.iDontExist* and *app.tabFuncDoesntExist*. If the tool cannot find a match for one of your functions or procedures, it tells you what signature it expected to find. You need to adjust your application in one of the following ways:

- **Method:** Change the signature of your Java method to match the signature suggested by the `SignatureChecker` tool.

- **Routine:** Drop and recreate your function or procedure so that its arguments and return type match your Java method according to the SQL Standard rules described in "Argument matching" in the *Java DB Reference Manual*.

PlanExporter

Use the `PlanExporter` tool to export query plan data for further analysis. The query plan data can be exported in a variety of formats:

- XML, the base format for exported query plan data
- HTML, which helps you view graphically the execution plans of complex queries you have executed

By using this tool, you can avoid querying XPLAIN style tables to get a basic idea of the query plan followed by the optimizer.

You can specify other query plan export formats by specifying an appropriate XSL stylesheet to transform the query plan data, or you can export the query plan data as XML, then reformat as appropriate using any external XML-aware tool of your choice.

Note: The `PlanExporter` tool is in an experimental stage. The Derby team welcomes comments and feedback on how to improve it.

Using PlanExporter

Before you run the `PlanExporter` tool, make sure that your classpath contains the Derby jar files, including *derbytools.jar*.

Before you run the `PlanExporter` tool, you must capture the `stmt_id` of the query you have executed from `SYSXPLAIN_STATEMENTS system table`. To do so, follow these steps:

1. **Use XPLAIN styles to capture the runtime statistics.**

   Refer to "SYSCS_UTIL.SYSCS_SET_XPLAIN_SCHEMA system procedure" in the *Java DB Reference Manual* to see how to do this.

   Note: You must remember the `schema_name`.
2. **Query the SYSXPLAIN_STATEMENTS system table to obtain the `stmt_id` of the query you have executed.**

   Refer to "SYSXPLAIN_STATEMENTS system table" in the *Java DB Reference Manual* for information about the SYSXPLAIN_STATEMENTS system table.

You can run the tool as follows in the directory where your database is located.

```
java org.apache.derby.tools.PlanExporter derby_connection_URL schema_name
 stmt_id options
```

The *options* can be passed according to your requirements. Consider the following possible scenarios:

- To generate an XML file of the query plan, specify the following option:

  ```
  -xml path
  ```

  The *path* can be either absolute or relative. If the root filename does not have a suffix, the tool appends `.xml`.
- To generate a plain HTML file of the query plan, you can use the default simple style sheet provided with Derby. Specify the following options:

  ```
  -html path
  ```

  If the root filename does not have a suffix, the tool appends `.html`.

To generate the XML file as well, specify the following options:

```
-xml path -html path
```

To use a different style sheet that does not contain any *javascript* functions, specify the following options:

```
-xsl path -html path
```

To generate the XML file as well, specify the following options:

```
-xml path -xsl path -html path
```

- To generate an advanced view of the query plan, you can use advanced XSL style sheets provided with Derby inside *derbytools.jar/org/apache/derby/impl/tools/planexporter/resources/*, or you can specify a style sheet created by you. To do this, specify the following options:

  ```
  -adv -xsl fileName -xml path
  ```

  Note:
  - Before you use the `-adv` feature, you must copy the advanced XSL style sheet into the current directory. Thus, you must specify only the name of the style sheet, not the path.
  - Generating HTML is not supported when you use the `-adv` feature of the `PlanExporter` tool. But if you open the generated XML file in a web browser, the browser will do the necessary transformation.

PlanExporter XML format

The `PlanExporter` tool extracts the query plan of an executed query as a XML document by using the statistics captured from Derby XPLAIN style tables.

An XML document generated by the `PlanExporter` tool has the following structure.

- **The basic tree structure:**
  - `plan`: The root of the XML tree
    - `statement`: First child, the query executed
    - `time`: Second child, the time that this query executed
    - `stmt_id` : Third child, the STMT_ID of the query
    - `details`: Fourth child, containing the query plan
- **The `statement` element:**

  This element has only its value. That value implies the query executed, as retrieved from the STMT_TEXT row of SYSXPLAIN_STATEMENTS table.

  For example:

  ```
  <statement>select * from my_table</statement>
  ```
- **The `time` element:**

  This element has only its value. That value implies the date & time which the query executed, as retrieved from the XPLAIN_TIME row of SYSXPLAIN_STATEMENTS table.

  For example:

  ```
  <time>2010-07-13 14:27:59.405</time>
  ```
- **The `stmt_id` element:**

  This element has only its value. That value implies the statement id of the query executed, as retrieved from the STMT_ID row of SYSXPLAIN_STATEMENTS table.

For example:

```
<stmt_id>9ac8804c-0129-cc31-ca9a-00000047f1e8</stmt_id>
```

- **The `details` element:**

  This element contains the query plan, as a tree structure of plan nodes.

  For a particular query there is only one root plan node.
- **A `node` element:**

  Contains the details of a plan node of the query plan. This element can contain zero or many child elements of the same type (`node` elements).

  This element contains one or more attributes, given that they are not null. The possible attributes and their meanings are shown in the following table.

  Table 5. Attributes of the `node` element

Attribute Name	Meaning
`name`	Name of the plan node
`input_rows`	Retrieved from the INPUT_ROWS row of the SYSXPLAIN_RESULTSETS system table
`returned_rows`	Retrieved from the RETURNED_ROWS row of the SYSXPLAIN_RESULTSETS system table
`no_opens`	Retrieved from the NO_OPENS row of the SYSXPLAIN_RESULTSETS system table
`visited_pages`	Retrieved from the NO_VISITED_PAGES row of the SYSXPLAIN_SCAN_PROPS system table
`scan_qualifiers`	Retrieved from the SCAN_QUALIFIERS row of the SYSXPLAIN_SCAN_PROPS system table
`next_qualifiers`	Retrieved from the NEXT_QUALIFIERS row of the SYSXPLAIN_SCAN_PROPS system table
`scanned_object`	Retrieved from the SCAN_OBJECT_NAME row of the SYSXPLAIN_SCAN_PROPS system table
`scan_type`	Retrieved from the SCAN_TYPE row of the SYSXPLAIN_SCAN_PROPS system table
`sort_type`	Retrieved from the SORT_TYPE row of the SYSXPLAIN_SORT_PROPS system table
`sorter_output`	Retrieved from the NO_OUTPUT_ROWS row of the SYSXPLAIN_SORT_PROPS system table

  For example:

```
<node name="TABLESCAN" returned_rows="100000" no_opens="1"
visited_pages="2165" scan_qualifiers="None" scanned_object="USERS"
scan_type="HEAP" >
```

PlanExporter example

This example shows the steps that you must follow in order to use the `PlanExporter` tool.

1. Move to the directory where your database was created.

2. Run the `ij` tool:

```
java org.apache.derby.tools.ij
```

3. Create a connection to the database:

```
CONNECT 'jdbc:derby:myDb;create=false';
```

Note: You can use a Derby client/server database as well.

4. Use XPLAIN styles:

```
CALL SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(1);
CALL SYSCS_UTIL.SYSCS_SET_XPLAIN_SCHEMA('MY_SCHEMA');
select * from my_table;
CALL SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(0);
CALL SYSCS_UTIL.SYSCS_SET_XPLAIN_SCHEMA('');
```

5. Obtain the `stmt_id` of the query:

```
select stmt_text, stmt_id from MY_SCHEMA.SYSXPLAIN_STATEMENTS;
exit;
```

Now find the `stmt_id` of the executed query in the displayed results and note it down. It looks something like this:

```
9ac8804c-0129-cc31-ca9a-00000047f1e8
```

6. Run the `PlanExporter` tool in the same location:

```
java org.apache.derby.tools.PlanExporter jdbc:derby:myDb MY_SCHEMA
    9ac8804c-0129-cc31-ca9a-00000047f1e8 -html plain_html;
```

This command uses the default style sheet provided with Derby, and the HTML file will be generated at the same location, since the command does not specify a different path. The name of the HTML file generated is *plain_html.html*.

Optional tools

Derby supports optional tools, which you can load and unload by using the `SYSCS_UTIL.SYSCS_REGISTER_TOOL` system procedure as described in the *Java DB Reference Manual*.

Using the databaseMetaData optional tool

The `databaseMetaData` optional tool creates functions and table functions corresponding to most of the methods in the *java.sql.DatabaseMetaData* interface.

Before you run the `databaseMetaData` optional tool, make sure that your classpath contains the Derby jar files, including *derbytools.jar*.

You can load and unload the `databaseMetaData` tool by using the `SYSCS_UTIL.SYSCS_REGISTER_TOOL` system procedure. See the *Java DB Reference Manual* for information about this procedure.

To load the `databaseMetaData` tool, use the following statement:

```
call syscs_util.syscs_register_tool( 'databaseMetaData', true )
```

This command creates metadata functions and table functions in the current schema. The functions and table functions have the same names as the corresponding *java.sql.DatabaseMetaData* methods which they wrap. Once you have loaded this tool, you can filter and join these functions to create powerful metadata queries. For instance, the following query lists the column names and datatypes for all columns in tables created by users:

```
select t.table_schem, t.table_name, c.column_name, c.type_name
from table( getTables( null, '%', '%' ) ) t,
     table( getColumns( null, '%', '%', '%') ) c
where c.table_schem = t.table_schem
and c.table_name = t.table_name
and t.table_type = 'TABLE'
order by table_schem, table_name, column_name
```

A few *DatabaseMetaData* methods take array arguments. Because those arguments cannot be represented as Derby types, the arguments are eliminated. This means that the trailing *types* arguments to *getTables()* and *getUDTs()* have been eliminated. In addition, the following *DatabaseMetaData* methods do not have corresponding metadata routines:

- *getRowIdLifetime()* is eliminated because Derby does not provide an implementation of *java.sql.RowIdLifetime*.
- *getSchemas()* is eliminated because Derby does not support overloads. The more general *getSchemas(String, String)* method is included.
- *supportsConvert()* is eliminated because Derby does not support overloads. The more general *supportsConvert(int, int)* is included.

When you have finished joining metadata results, you can drop this package of functions and table functions as follows:

```
call syscs_util.syscs_register_tool( 'databaseMetaData', false )
```

Using the foreignViews optional tool

The `foreignViews` optional tool creates schemas, table functions, and convenience views for all user tables in a foreign database. This can be useful for bulk-importing foreign data.

Before you run the `foreignViews` optional tool, make sure that your classpath contains the Derby jar files, including *derbytools.jar*.

You can load and unload the `foreignViews` tool by using the `SYSCS_UTIL.SYSCS_REGISTER_TOOL` system procedure. See the *Java DB Reference Manual* for information about this procedure.

To load the `foreignViews` tool, use a statement like the following:

```
call syscs_util.syscs_register_tool( 'foreignViews', true,
    'foreignDatabaseURL', 'XYZ_' )
```

The two trailing arguments have the following meanings:

- *foreignDatabaseURL* is a URL suitable for creating a connection to the foreign database by calling *java.sql.DriverManager.getConnection()*. For example:

  ```
  'jdbc:derby:db3;user=fred;password=fredpassword'
  ```
- `'XYZ_'` is a string prefixed to the names of all schemas created by this tool. This argument may be omitted. If it is omitted, the tool will create schemas which have the same names as the schemas in the foreign database.

For example, suppose that the foreign database has two schemas, `S1` and `S2`. `S1` contains two user tables, `T1` and `T2`. `S2` contains two user tables, `U1` and `U2`. Loading the tool as shown above will create the following objects in your Derby database:

schema `XYZ_S1`
table function `XYZ_S1.T1`, which reads `S1.T1` from the foreign database
table function `XYZ_S1.T2`, which reads `S1.T2` from the foreign database
view `XYZ_S1.T1`, which wraps the corresponding table function
view `XYZ_S1.T2`, which wraps the corresponding table function
schema `XYZ_S2`
table function `XYZ_S2.U1`, which reads `S2.U1` from the foreign database
table function `XYZ_S2.U2`, which reads `S2.U2` from the foreign database
view `XYZ_S2.U1`, which wraps the corresponding table function
view `XYZ_S2.U2`, which wraps the corresponding table function

The views hide the arguments to the table functions. You can then populate your local schema by using the following SELECT statements:

```
insert into S1.T1 select * from XYZ_S1.T1
insert into S1.T2 select * from XYZ_S1.T2
insert into S2.U1 select * from XYZ_S2.U1
insert into S2.U2 select * from XYZ_S2.U2
```

When you have finished bulk-importing the foreign data, you can drop this package of schemas, table functions and views as follows:

```
call syscs_util.syscs_register_tool( 'foreignViews', false,
    'foreignDatabaseURL', 'XYZ_' )
```

Trademarks

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the Apache Derby documentation library:

Cloudscape, DB2, DB2 Universal Database, DRDA, and IBM are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.