## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copyright</td>
<td>5</td>
</tr>
<tr>
<td>License</td>
<td>6</td>
</tr>
<tr>
<td>About this guide</td>
<td>10</td>
</tr>
<tr>
<td>Purpose of this document</td>
<td>10</td>
</tr>
<tr>
<td>Audience</td>
<td>10</td>
</tr>
<tr>
<td>How this guide is organized</td>
<td>10</td>
</tr>
<tr>
<td>What are the Derby tools and utilities?</td>
<td>12</td>
</tr>
<tr>
<td>Overview</td>
<td>12</td>
</tr>
<tr>
<td>Environment setup and the Derby tools</td>
<td>12</td>
</tr>
<tr>
<td>About Derby databases</td>
<td>13</td>
</tr>
<tr>
<td>JDBC connection basics</td>
<td>13</td>
</tr>
<tr>
<td>JDBC drivers overview</td>
<td>13</td>
</tr>
<tr>
<td>Database connection URLs</td>
<td>13</td>
</tr>
<tr>
<td>Tools and localization</td>
<td>14</td>
</tr>
<tr>
<td>About locales</td>
<td>14</td>
</tr>
<tr>
<td>Database territory</td>
<td>15</td>
</tr>
<tr>
<td>Specifying an alternate codeset</td>
<td>15</td>
</tr>
<tr>
<td>Formatting display of locale-sensitive data</td>
<td>15</td>
</tr>
<tr>
<td>Using ij</td>
<td>16</td>
</tr>
<tr>
<td>Starting ij</td>
<td>16</td>
</tr>
<tr>
<td>Creating a database using ij</td>
<td>17</td>
</tr>
<tr>
<td>Starting ij using properties</td>
<td>17</td>
</tr>
<tr>
<td>Getting started with ij</td>
<td>18</td>
</tr>
<tr>
<td>Connecting to a Derby database</td>
<td>18</td>
</tr>
<tr>
<td>Using ij commands</td>
<td>20</td>
</tr>
<tr>
<td>Running ij scripts</td>
<td>20</td>
</tr>
<tr>
<td>ij properties reference</td>
<td>22</td>
</tr>
<tr>
<td>ij.connection.connectionName property</td>
<td>22</td>
</tr>
<tr>
<td>ij.database property</td>
<td>22</td>
</tr>
<tr>
<td>ij.dataSource property</td>
<td>23</td>
</tr>
<tr>
<td>ij.driver property</td>
<td>23</td>
</tr>
<tr>
<td>ij.exceptionTrace property</td>
<td>24</td>
</tr>
<tr>
<td>ij.maximumDisplayWidth property</td>
<td>25</td>
</tr>
<tr>
<td>ij.outfile property</td>
<td>25</td>
</tr>
<tr>
<td>ij.password property</td>
<td>25</td>
</tr>
<tr>
<td>ij.protocol property</td>
<td>26</td>
</tr>
<tr>
<td>ij.protocol.protocolName property</td>
<td>26</td>
</tr>
<tr>
<td>ij.showErrorCode property</td>
<td>26</td>
</tr>
<tr>
<td>ij.showNoConnectionsAtStart property</td>
<td>27</td>
</tr>
<tr>
<td>ij.showNoCountForSelect property</td>
<td>27</td>
</tr>
<tr>
<td>ij.URLCheck property</td>
<td>28</td>
</tr>
<tr>
<td>ij.user property</td>
<td>29</td>
</tr>
<tr>
<td>derby.ui.codeset property</td>
<td>29</td>
</tr>
<tr>
<td>derby.ui.locale property</td>
<td>30</td>
</tr>
<tr>
<td>ij commands and errors reference</td>
<td>32</td>
</tr>
<tr>
<td>ij commands</td>
<td>32</td>
</tr>
<tr>
<td>Conventions for ij examples</td>
<td>32</td>
</tr>
<tr>
<td>ij SQL command behavior</td>
<td>32</td>
</tr>
<tr>
<td>Absolute command</td>
<td>33</td>
</tr>
</tbody>
</table>
Storing jar files in a database ................................................................. 71
  Adding a Jar File.................................................................................. 71
  Removing a jar file.............................................................................. 71
  Replacing a jar file............................................................................... 71
  Installing a jar example...................................................................... 71

sysinfo ........................................................................................................... 73
  sysinfo example.................................................................................. 73
  Using sysinfo to check the classpath.................................................. 74

dblook ........................................................................................................... 75
  Using dblook....................................................................................... 75
  dblook options.................................................................................... 75
  Generating the DDL for a database.................................................... 76
  dblook examples.................................................................................. 77

SignatureChecker ..................................................................................... 79
  Using SignatureChecker...................................................................... 79

Trademarks ................................................................................................. 81
Copyright 2004-2010 The Apache Software Foundation
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
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.
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 Derby*.

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`
- the import and export utilities
- the jar file utilities
- `sysinfo`
- `dblook`
- `SignatureChecker`

Audience

This book is for:

- developers, who might use the tools when developing applications
- system administrators, who might use the tools to run backup scripts or to import large amounts of 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.
- **Using the bulk import and export procedures**
  
  Reference and how-to instructions for using bulk import and export.
- **Storing jar files in a database**
  
  Syntax for executing the built-in procedures for storing jar files in the database.
- **`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.
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 setup and update a Derby database.

For more complete information on developing a system using Derby, see the Derby 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.

- **The import and export utilities**
  
  These server-side utilities allow you to import data directly from files into tables and to export data from tables into files. Server-side utilities can be in a client/server environment but require that all files referenced be on the Server machine.

- **The jar file utilities**
  
  These utilities allow you to store jar files in a database.

- **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

*i j, sysinfo, dblook, and SignatureChecker* are tools that can be used in an embedded or a client/server environment. The import and export utilities and jar file utilities are database-side utilities, which means that they run in the same JVM as Derby (that is, on the server). This means when used in a client/server environment all files imported, exported, or loaded must be local to the server machine.


All Derby tools require the Java 2 Platform, Standard Edition, Version 1.4 or later.
**Classpath**

To simplify the process of setting up the `CLASSPATH` environment variable to run Derby and the tools, a new jar file, `derbyrun.jar`, has been added to the Derby distribution. 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 *Derby 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 an 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 driver.

- `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 *Derby 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 Derby 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 Derby 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 territories.

For general information about international Derby systems, see the Derby 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 territory**
  This is the territory associated with your database when it is created. By default, this is the same as the java system locale. The database territory 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 territory**

To specify a database territory, use the `territory` attribute on the URL connection when creating the database.

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

For information about database territories, see the Internationalization appendix in the *Derby 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
----------
2001-09-06
1 row selected
ij> localizeddisplay on;
ij> VALUES CURRENT_DATE;
1
------------------
September 6, 2001
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 Derby 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 Derby 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]`

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.

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 10.6
ij(HISTORY)> show connections;
```

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 10.6
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 10.6
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 Derby 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 be 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
%DERBY_HOME%\demo\programs\toursdb\ToursDB_schema.sql
```
### 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:

```java
D:> java org.apache.derby.tools.ij
ij version 10.6
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:

```java
D:> java org.apache.derby.tools.ij
ij version 10.6
ij> driver 'sun.jdbc.odbc.JdbcOdbcDriver';
ij> connect 'jdbc:odbc:myOdbcDataSource';
ij>
```

### The `ij Driver` name and connection URL

**Specifying the Driver Name and database connection URL**, summarizes the different ways to specify the driver name and database connection URL.

**Table 1. Specifying the Driver Name and database connection URL**

<table>
<thead>
<tr>
<th>Action</th>
<th>System Property</th>
<th><code>ij</code> Property</th>
<th><code>ij</code> Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>loading the driver implicitly</td>
<td>'ij.connection.connectionName' (plus full URL)</td>
<td><code>ij Driver</code> (plus full URL)</td>
<td><code>ProtocolConnect</code> (plus full URL)</td>
</tr>
</tbody>
</table>
### 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 `anohterDB`.

```
D:> java -Dij.connection.sample1=jdbc:derby:sample
-Dij.connection.anotherConn=jdbc:derby:anotherDB;create=true
org.apache.derby.tools.ij
ij version 10.6
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:
```
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 10.6
* = 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
```

---

Dij.database=wombat;create=true org.apache.derby.tools.ij
ij version 10.6
CONNECTION0*  jdbc:derby:wombat
* = current connection
ij>
Derby Tools and Utilities Guide

#
java -Dij.dataSource=org.apache.derby.jdbc.EmbeddedSimpleDataSource
org.apache.derby.tools.ij
ERROR XJ004: Database '' not found.
ij version 10.6
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 Derby 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 10.6
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 10.6
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.java:44)
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.EmbedConnection.newSQLException(EmbedConnection.java:1955)
at
org.apache.derby.impl.jdbc.EmbedConnection.(EmbedConnection.java:254)
```
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
java -Dij.user=me -Dij.password=mine org.apache.derby.tools.ij
```

See the *Derby 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**

```javascript
ij.protocol=protocolForEnvironment
```

**Example**

```bash
D:> java -Dij.protocol=jdbc:derby: org.apache.derby.tools.ij
ij version 10.6
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**

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

**Example**

```bash
ij version 10.6
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 *Derby Reference Manual*. 
Syntax

ij showErrorCode={ false | true }

Example

java -Dij.showErrorCode=true -Dij.protocol=jdbc:derby:
   org.apache.derby.tools.ij
ij version 10.6
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 10.6
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 10.6
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 set to `true`, then with the property set to `false`.

```java
java -Dij.showNoCountForSelect=true org.apache.derby.tools.ij
ij version 10.6
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 10.6
CONNECTION0*
  *= current connection
ij> select * from t1;
C1
--------
1
2
3
3 rows selected
ij>
```

**ij.URLCheck**

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

```
java -Dij.URLCheck=false org.apache.derby.tools.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 Derby 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
```

See the Derby 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 Sample Character Encodings).

**Example**

The following command specifies to run *ij* using the Japanese territory *(derby.ui.locale=ja_JP)* using Japanese Latin Kanji mixed encoding *(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 Java 2 Software Development Kit, Standard Edition, v. 1.4.2 go to http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html.

**Table 2. Sample Character Encodings**

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

**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. Note that some pages in the Derby documentation guides refer to a "locale" as a "territory".

**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 specifies to run \texttt{ij} using the Japanese territory \texttt{(derby.ui.locale=ja\_JP)} using Japanese Latin Kanji mixed encoding \texttt{(codeset=Cp939)}:

```shell
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.

<table>
<thead>
<tr>
<th>Language Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>en</td>
<td>English</td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
</tr>
</tbody>
</table>


Country codes consist of two uppercase letters that conform to ISO-3166.

<table>
<thead>
<tr>
<th>Country Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>Germany</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>ES</td>
<td>Spain</td>
</tr>
<tr>
<td>MX</td>
<td>Mexico</td>
</tr>
<tr>
<td>JP</td>
<td>Japan</td>
</tr>
</tbody>
</table>

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 10.6
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 SQLException, 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, i j 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 (i j limits displayed width of a column to 128 characters by default; see MaximumDisplayWidth command).

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

i j 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 i j 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
-----------------------------------------------
extree    |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
----------|-----------|-------
extree   |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**

```sql
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**

```sql
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 [Derby 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
```
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

```
i\texttt{ij} > \texttt{commit};
i\texttt{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

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

**Syntax**

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

**Description**

Provides a description 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   | NULL  | 52       | NULL      | YES
DISTANCE_DISCOUNT | DOUBLE | NULL| 2  | 52     | NULL      | NULL      | YES
BUSINESS_LEVEL_FACT& | DOUBLE | NULL| 2 | 52     | NULL      | NULL      | YES
FIRSTCLASS_LEVEL_FACT& | 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';
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
```
```sql
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
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

```
i\> get scroll insensitive cursor scrollCursor as 'SELECT * FROM menu FOR UPDATE OF price';
i\> first scrollCursor;
```

```
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>
```

**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.

i\> 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 i\> 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

```
i\> -- autocommit needs to be off so that the positioned update
i\> -- can see the cursor it operates against.
i\> autocommit off;
i\> get cursor menuCursor as 'SELECT * FROM menu FOR UPDATE OF price';
i\> next menuCursor;
```

```
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>
```

```
i\> next menuCursor;
```

```
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>
```

```
i\> UPDATE menu SET price=price+1 WHERE CURRENT OF menuCursor;
```

1 row inserted/updated/deleted
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.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 scrollable cursor has been created, you can use the following commands to work with the result set:
  • Absolute command
  • After Last command
Examples

```
ij> autocommit off;
ij> get scroll insensitive cursor scrollCursor as 'SELECT * FROM menu';
ij> absolute 5 scrollCursor;
ij> after last scrollCursor;
No current row
ij> before first scrollCursor;
No current row
ij> first scrollCursor;
ij> last scrollCursor;
ij> previous scrollCursor;
ij> relative 1 scrollCursor;
ij>>previous scrollCursor;
ij> next scrollCursor;
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;
ij(FRED_CONN)> getcurrentrownumber john_cursor@john_conn;
ij(FRED_CONN)> last john_cursor@john_conn;
ij(FRED_CONN)> previous john_cursor@john_conn;
```
Help command

Syntax

HELP

Description

Prints out a brief list of the i j 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

```
i j> get scroll insensitive cursor scrollCursor as 'SELECT * FROM menu FOR UPDATE OF price';
i j> absolute 5 scrollCursor;
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
</tbody>
</table>
```

```
i j> last scrollCursor;
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>
```

LocalizedDisplay command

Syntax

LOCALIZEDDISPLAY { on | off }

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

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

\textbf{Example}

The following demonstrates \texttt{LocalizedDisplay} in an English locale:

\begin{verbatim}
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
\end{verbatim}

\textbf{MaximumDisplayWidth command}

\textbf{Syntax}

\texttt{MAXIMUMDISPLAYWIDTH} integer\_value

\textbf{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.

\textbf{Example}

\begin{verbatim}
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!
\end{verbatim}

\textbf{Next command}

\textbf{Syntax}

\texttt{NEXT} Identifier

\textbf{Description}

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

\textbf{Example}

\begin{verbatim}
ij> get cursor menuCursor as 'SELECT * FROM menu';
ij> next menuCursor;
COURSE |ITEM                |PRICE
-----------------------------------------------
entree |lamb chop           |14
ij>
\end{verbatim}
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;

<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>6</td>
</tr>
</tbody>
</table>

2 rows selected

ij>

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

```
i> get scroll insensitive cursor scrollCursor as
'SELECT * FROM menu FOR UPDATE OF price';
i> last scrollcursor;
COURSE | ITEM                | PRICE
-----------------------------------------------
dessert | creme brulee        | 6
i> previous scrollcursor;
COURSE | ITEM                | PRICE
-----------------------------------------------
tree    | 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 Derby 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

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

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

```
i> protocol 'jdbc:derby:memory:';
i> 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 Derby Developer's Guide.)

Example
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

i$j> -- autocommit needs to be off so that the positioned update
i$j> -- can see the cursor it operates against.
1 jane commit off;
i$j> get scroll insensitive cursor scrollCursor as
1 SELECT * FROM menu FOR UPDATE OF price;
i$j> last scrollcursor;
---
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
i$j> previous scrollcursor;
---
entree lamb chop 14
i$j> relative 1 scrollcursor;
---
dessert creme brulee 6

Remove command

Syntax

REMOVE Identifier

Description

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

Example

i$j> prepare seeMenu as 'SELECT * FROM menu';
i$j> execute seeMenu;
---
entree lamb chop 14
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

```sql
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
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.

Example

```
i3> protocol 'jdbc:derby:';
i3> connect 'sample' as sample1;
i3> connect 'newDB;create=true' as newDB;
i3 (NEWDB)> show connections;
SAMPLE1 - jdbc:derby:sample
NEWDB* - jdbc:derby:newDB;create=true
*= current connection
i3(NEWDB)> set connection sample1;
i3(SAMPLE1)> disconnect all;
i3>
```

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 Derby 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_SCHEMA|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
-------------------------------------------------------------------------
---
AIRCRAFT            |AIRLINE             |true  |A   |NULL|NULL    |NULL
AIRCRAFT            |AIRCRAFT_COUNTRY    |false |A   |NULL|NULL    |NULL
AIRCRAFT            |AIRCRAFT_CITY       |false |A   |NULL|NULL    |NULL
COUNTRIES           |COUNTRY_ISO_CODE    |false |A   |NULL|NULL    |NULL
COUNTRIES           |COUNTRY             |false |A   |NULL|NULL    |NULL
CITIES              |CITY_ID             |false |A   |NULL|NULL    |NULL
FLIGHTS             |FLIGHT_ID           |false |A   |NULL|NULL    |NULL
FLIGHTS             |SEGMENT_NUMBER      |false |A   |NULL|NULL    |NULL
FLIGHTAVAILABILITY  |FLIGHT_ID           |false |A   |NULL|NULL    |NULL
FLIGHTAVAILABILITY  |SEGMENT_NUMBER      |false |A   |NULL|NULL    |NULL
```
### SHOW INDEXES

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NON_UNIQUE</th>
<th>TYPE</th>
<th>ASC</th>
<th>CARDINALITY</th>
<th>PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLIGHTS</td>
<td>FLIGHT_ID</td>
<td>false</td>
<td>3</td>
<td>A</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>FLIGHTS</td>
<td>SEGMENT_NUMBER</td>
<td>false</td>
<td>3</td>
<td>A</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>FLIGHTS</td>
<td>DEST_AIRPORT</td>
<td>true</td>
<td>3</td>
<td>A</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>FLIGHTS</td>
<td>ORIG_AIRPORT</td>
<td>true</td>
<td>3</td>
<td>A</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>CITIES</td>
<td>COUNTRY_ISO_CODE</td>
<td>true</td>
<td>3</td>
<td>A</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

16 rows selected

```sql
i#> show indexes from flights;
```

### 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

```sql
i#> show procedures in syscs_util;
```

<table>
<thead>
<tr>
<th>PROCEDURE_SCHEMA</th>
<th>PROCEDURE_NAME</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_BACKUP_DATABASE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_BACKUP_DATABASE_AND_ENA</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_BACKUP_DATABASE_AND_ENA_AND_NOWAIT</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_BULK_INSERT</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_CHECKPOINT_DATABASE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_COMPRESS_TABLE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_DISABLE_LOG_ARCHIVE_MODE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_EXPORT_QUERY</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_EXPORT_TABLE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_FREEZE_DATABASE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_IMPORT_DATA</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_IMPORT_TABLE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_INPLACE_COMPRESS_TABLE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_SET_DATABASE_PROPERTY</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_SET_RUNTIMESTATISTICS</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_SET_STATISTICS_TIMING</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYSCS_UNFREEZE_DATABASE</td>
<td>org.apache.derby.ca&amp;</td>
</tr>
</tbody>
</table>

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 Derby 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

```sql
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

```sql
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

```sql
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>
<thead>
<tr>
<th>TABLE_SCHEM</th>
<th>TABLE_NAME</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP</td>
<td>AIRLINES</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>CITIES</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>COUNTRIES</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>FLIGHT Avaibility</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>FLIGHTS</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>FLIGHTS_HISTORY</td>
<td></td>
</tr>
<tr>
<td>APP</td>
<td>MAPS</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>TABLE_SCHEM</th>
<th>TABLE_NAME</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP</td>
<td>TOTAL SEATS</td>
<td></td>
</tr>
</tbody>
</table>

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:

```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

```sql
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 i j 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.

i j does not recognize or permit delimited identifiers in i j commands. They can be used in SQL commands.

Example

These are valid i j identifiers:
  foo1
  exampleIdentifier12345
  another_one
  myId@connection0
  id2 @ connection1

Syntax for strings in i j commands

Syntax

' Text '

Description

Some i j commands require strings. i j 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. i j places no limitation on the lengths of strings, and will treat embedded new lines in the string as characters in the string.

Some i j 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 i j  And this is its value
'Mary''s umbrella'  Mary's umbrella
'hello world'  hello world
--returns Joe's
execute 'VALUES ''Joe'''s''';

i j errors

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

ERROR SQLState

When the underlying JDBC driver returns an SQLException, i j 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, i j 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 i j 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

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

JAVA ERROR

When an unexpected Java exception occurs, i j prints a message with the prefix "JAVA ERROR".
Using the bulk import and export procedures

You can import and export large amounts of data between files and the Derby database. Instead of having to use INSERT and SELECT statements, you can use Derby system procedures to import data directly from files into tables and to export data from tables into files.

The Derby system procedures import and export data in delimited data file format.

- Use the export system procedures to write data from a database to one or more files that are stored outside of the database. You can use a procedure to export data from a table into a file or export data from a SELECT statement result into a file.
- Use the import system procedures to import data from a file into a table. If the target table already contains data, you can replace or append to the existing data.

Methods for running the import and export procedures

You can run the import and export procedures from within an SQL statement using `ij` or any Java application.

The import and export procedures read and write text files, and if you use an external file when you import or export data, you can also import and export blob data. The import procedures do not support read-once streams (live data feeds), because the procedures read the first line of the file to determine the number of columns, then read the file again to import the data.

Note: The import and export procedures are server-side utilities that exhibit different behavior in client/server mode. Typically, you use these procedures to import data into and export data from a locally running Derby database. However, you can use the import and export procedures when Derby is running in a server framework if you specify import and export files that are accessible to the server.

Bulk import and export requirements and considerations

There are requirements and limitations that you must consider before you use the Derby import and export procedures.

Database transactions

You should issue either a COMMIT or ROLLBACK statement to complete all transactions and release all table-level locks before you invoke an import or export procedure. Derby issues a COMMIT or a ROLLBACK statement after each import and export procedure is run.

Note: Imports are transactional. If an error occurs during bulk import, all changes are rolled back.

Database connections

To invoke a Derby import or export procedure, you must be connected to the database into which the data is imported or from which the data is exported. Other user applications that access the table with a separate connection do not need to disconnect.

Classpath

You must have the `derbytools.jar` file in your classpath before you can use the import or export procedures from `ij`.

The table must exist
To import data into a table, the table must already exist in Derby. The table does not have to be empty. If the table is not empty, bulk import performs single row inserts which results in slower performance.

**Create indexes, keys, and unique constraints before you import**

To avoid a separate step, create the indexes, keys (primary and foreign), and unique constraints on tables before you import data. However, if your memory and disk spaces resources are limited, you can build the indexes and primary keys after importing data.

**Data types**

Derby implicitly converts the strings to the data type of the receiving column. If any of the implicit conversions fail, the whole import is aborted. For example, “3+7” cannot be converted into an integer. An export that encounters a runtime error stops.

**Note:** You cannot import or export the XML data type.

**Locking during import**

Import procedures use the same isolation level as the connection in which they are executed to insert data into tables. During import, the entire table is exclusively locked irrespective of the isolation level.

**Locking during export**

Export procedures use the same isolation level as the connection in which they are executed to fetch data from tables.

**Import behavior on tables with triggers**

The import procedures enables INSERT triggers when data is appended to the table. The REPLACE parameter is not allowed when triggers are enabled on the table.

**Restrictions on the REPLACE parameter**

If you import data into a table that already contains data, you can either replace or append to the existing data. You can use the REPLACE parameter on tables that have dependent tables. The replaced data must maintain referential integrity, otherwise the import operation will be rolled back. You cannot use the REPLACE parameter if the table has triggers enabled.

**Restrictions on tables**

You cannot use import procedures to import data into a system table or a declared temporary table.

**Bulk import and export of large objects**

You can import and export large objects (LOBs) using the Derby system procedures.

**Importing and exporting CLOB and BLOB data**

CLOB and BLOB can be exported to the same file as the rest of the column data, or the LOB column data can be exported to separate external file. When the LOB column data is exported to separate external file, reference to the location of the LOB data is placed in the LOB column in the main export file.

Importing and exporting LOB data using an separate external file might be faster than storing the LOB data in the same file as the rest of the column data:

- The CLOB data does not have to be scanned for the delimiters inside the data
- The BLOB data does not need to be converted into a hexadecimal format

**Importing and exporting other binary data**

When you export columns that contain the data types CHAR FOR BIT DATA, VARCHAR FOR BIT DATA, and LONG VARCHAR FOR BIT DATA, the column data is always exported to the main export file. The data is written in the hexadecimal format. To import data into a table that has columns of these data types, the data in the import file for those column must be in the hexadecimal format.

**Importing LOB data from a file that contains all of the data**
You can use the SYSCS_UTIL.SYSCS_IMPORT_TABLE and SYSCS_UTIL.SYSCS_IMPORT_DATA procedures to import data into a table that contains a LOB column. The LOB data must be stored in the same file as the other column data that you are importing. If you are importing data from a file that was exported from a non-Derby source, the binary data must be in the hexadecimal format.

**Importing LOB data from a separate external file**

You can use the SYSCS_UTIL.SYSCS_IMPORT_TABLE_LOBS_FROM_EXTFILE and SYSCS_UTIL.SYSCS_IMPORT_DATA_LOBS_FROM_EXTFILE procedures to import LOB data that is stored in a file that is separate from the main import file. These procedures read the LOB data using the reference that is stored in the main import file. If you are importing data from a non-Derby source, the references to the LOB data must be in the main import file in the format lobsFileName.Offset.length/. This is the same method that the Derby export procedures use to export the LOB data to a separate external file.

**Exporting LOB data to the same file as the other column data**

You can use the SYSCS_UTIL.SYSCS_EXPORT_TABLE and SYSCS_UTIL.SYSCS_EXPORT_QUERY procedures to write LOB data, along with rest of the column data, to a single export file.

CLOB column data is treated same as other character data. Character delimiters are allowed inside the CLOB data. The export procedures write the delimiter inside the data as a double-delimiter.

BLOB column data is written to the export file in the hexadecimal format. For each byte of BLOB data, two characters are generated. The first character represents the high nibble (4 bits) in hexadecimal and the second character represents the low nibble.

**Exporting LOB data to a separate external file from the other column data**

You can use the SYSCS_UTIL.SYSCS_EXPORT_TABLE_LOBS_TO_EXTFILE and SYSCS_UTIL.SYSCS_EXPORT_QUERY_LOBS_TO_EXTFILE procedures to write LOB data to a separate external file. These procedures include the lobFileName parameter, which specifies the name of external file for the LOB data.

When you use these procedures, the location of the LOB data is written to the main export file. The format of the reference to the LOB stored in the main export file is lobsFileName.Offset.length/.

- Offset is the position in the external file in bytes
- length is the size of the LOB column data in bytes

If a LOB column value is NULL, length is written as -1. No data conversion is performed when you export LOB data to an external file. BLOB data is written in binary format and CLOB data is written using the codeset that you specify.

See [Examples of bulk import and export](#) for examples using each of the import and export procedures.

**File format for input and output**

There are specific requirements for the format of the input and output files when you import and export data.

The default file format is a delimited text file with the following characteristics:

- Rows are separated by a new line
- Fields are separated by a comma (,)
- Character-based fields are delimited with double quotes ("")
**Restriction:** Before you perform import or export operations, you must ensure that the chosen delimiter character is not contained in the data to be imported or exported. If you chose a delimiter character that is part of the data to be imported or exported unexpected errors might occur. The following restrictions apply to column and character delimiters:

- Delimiters are mutually exclusive
- A delimiter cannot be a line-feed character, a carriage return, or a blank space
- The default decimal point (.) cannot be a character delimiter
- Delimiters cannot be hex decimal characters (0-9, a-f, A-F).

The record delimiter is assumed to be a new-line character. The record delimiter should not be used as any other delimiter.

Character delimiters are permitted with the character-based fields (CHAR, VARCHAR, and LONG VARCHAR) of a file during import. Any pair of character delimiters found between the enclosing character delimiters is imported into the database. For example, suppose that you have the following character string:

"what a ""great"" day!"

The preceding character string gets imported into the database as:

What a "great" day!

During export, the rule applies in reverse. For example, suppose you have the following character string:

"The boot has a 3" heel."

The preceding character string gets exported to a file as:

"The boot has a 3""heel."

The following example file shows four rows and four columns in the default file format:

1,abc,22,def
22,,,"a is a zero-length string, b is null"
13,"hello",454,"world"
4,"b and c are both null",

The export procedure outputs the following values:

1,"abc",22,"def"
22,,,"a is a zero-length string, b is null"
13,"hello",454,"world"
4,"b and c are both null",

---

**Importing data using the built-in procedures**

You can use the Derby import procedures to import all of the data from table or query, or to import LOB data separately from the other data.

1. Choose the correct procedure for the type of import that you want to perform. For examples of these procedures, see Examples of bulk import and export.

<table>
<thead>
<tr>
<th>Type of import</th>
<th>Procedure to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>To import all the data to a table, where the import file contains the LOB data</td>
<td>SYSCS_UTIL.SYSCS_IMPORT_TABLE (IN SCHEMANAME VARCHAR(128), IN TABLENAME VARCHAR(128), IN FILENAME VARCHAR(32672), IN COLUMNDELIMITER CHAR(1), IN CHARACTERDELIMITER CHAR(1))</td>
</tr>
</tbody>
</table>
To import the data to a table, where the LOB data is stored in a separate file and the main import file contains all of the other data with a reference to the LOB data

```
SYSCS_UTIL.SYSCS_IMPORT_TABLE_LOBS_FROM_EXTFILE
(IN SCHEMANAME VARCHAR(128),
  IN TABLENAME VARCHAR(128),
  IN FILENAME VARCHAR(32672),
  IN COLUMNDELMITER CHAR(1),
  IN CHARACTERDELMITER CHAR(1),
  IN CODESET VARCHAR(128),
  IN REPLACE SMALLINT)
```

The import utility looks in the main import file for a reference to the location of the LOB data. The format of the reference to the LOB stored in the main import file must be `lobsFileName.Offset.length/`.  

To import data from a file to a subset of columns in a table, where the import file contains the LOB data

```
SYSCS_UTIL.SYSCS_IMPORT_DATA
(IN SCHEMANAME VARCHAR(128),
  IN TABLENAME VARCHAR(128),
  INSERTCOLUMNS VARCHAR(32672),
  IN COLUMNINDEXES VARCHAR(32672),
  IN FILENAME VARCHAR(32672),
  IN COLUMNDELMITER CHAR(1),
  IN CHARACTERDELMITER CHAR(1),
  IN CODESET VARCHAR(128),
  IN REPLACE SMALLINT)
```

You must specify the `insertColumns` parameter on the table into which data will be imported. You must specify the `columnIndex` parameter to import data fields from a file to column in a table.

To import data to a subset of columns in a table, where the LOB data is stored in a separate file and the main import file contains all of the other data with a reference to the LOB data

```
SYSCS_UTIL.SYSCS_IMPORT_DATA_LOBS_FROM_EXTFILE
(IN SCHEMANAME VARCHAR(128),
  IN TABLENAME VARCHAR(128),
  INSERTCOLUMNS VARCHAR(32672),
  IN COLUMNINDEXES VARCHAR(32672),
  IN FILENAME VARCHAR(32672),
  IN COLUMNDELMITER CHAR(1),
  IN CHARACTERDELMITER CHAR(1),
  IN CODESET VARCHAR(128),
  IN REPLACE SMALLINT)
```

The import utility looks in the main import file for a reference to the location of the LOB data. The format of the reference to the LOB stored in the main import file must be `lobsFileName.Offset.length/`.  

### Parameters for the import procedures

The Derby import procedures use specific parameters.

**SCHEMANAME**

Specifies the schema of the table. You can specify a NULL value to use the default schema name. The `SCHEMANAME` parameter takes an input argument that is a `VARCHAR (128)` data type.

**TABLENAME**
Specifies the name of the table into which the data is to be imported. This table cannot be a system table or a declared temporary table. The string must exactly match case of the table name. Specifying a NULL value results in an error. The TABLENAME parameter takes an input argument that is a VARCHAR (128) data type.

**INSERTCOLUMNS**
Specifies the comma separated column names of the table into which the data will be imported. You can specify a NULL value to import into all columns of the table. The INSERTCOLUMNS parameter takes an input argument that is a VARCHAR (32672) data type.

**COLUMNINDEXES**
Specifies the comma separated column indexes (numbered from one) of the input data fields that will be imported. You can specify a NULL value to use all input data fields in the file. The COLUMNINDEXES parameter takes an input argument that is a VARCHAR (32762) data type.

**FILENAME**
Specifies the name of the file that contains the data to be imported. If the path is omitted, the current working directory is used. The specified location of the file should refer to the server side location if using the Network Server. Specifying a NULL value results in an error. The FILENAME parameter takes an input argument that is a VARCHAR (32672) data type.

**COLUMNDELMITER**
Specifies a column delimiter. The specified character is used in place of a comma to signify the end of a column. You can specify a NULL value to use the default value of a comma. The COLUMNDELMITER parameter takes an input argument that is a CHAR (1) data type.

**CHARACTERDELMITER**
Specifies a character delimiter. The specified character is used in place of double quotation marks to enclose a character string. You can specify a NULL value to use the default value of a double quotation mark. The CHARACTERDELMITER parameter takes an input argument that is a CHAR (1) data type.

**CODESET**
Specifies the code set of the data in the input file. The code set name should be one of the Java-supported character encoding sets. Data is converted from the specified code set to the database code set (UTF-8). You can specify a NULL value to interpret the data file in the same code set as the JVM in which it is being executed. The CODESET parameter takes an input argument that is a VARCHAR (128) data type.

**REPLACE**
A non-zero value for the replace parameter will import in REPLACE mode, while a zero value will import in INSERT mode. REPLACE mode deletes all existing data from the table by truncating the table and inserts the imported data. The table definition and the index definitions are not changed. You can only import with REPLACE mode if the table already exists. INSERT mode adds the imported data to the table without changing the existing table data. Specifying a NULL value results in an error. The REPLACE parameter takes an input argument that is a SMALLINT data type.

If you create a schema, table, or column name as a non-delimited identifier, you must pass the name to the import procedure using all uppercase characters. If you created a schema, table, or column name as a delimited identifier, you must pass the name to the import procedure using the same case that was used when it was created.

**Import into tables that contain identity columns**
You can use the either the SYSCS_UTIL.SYSCS_IMPORT_DATA procedure or the SYSCS_UTIL.SYSCS_IMPORT_DATA_LOBS_FROM_EXTFILE procedure to
import data into a table that contains an identity column. The approach that you take depends on whether the identity column is GENERATED ALWAYS or GENERATED BY DEFAULT.

Identity columns and the REPLACE parameter
If the REPLACE parameter is used during import, Derby resets its internal counter of the last identity value for a column to the initial value defined for the identity column.

Identity column is GENERATED ALWAYS
If the identity column is defined as GENERATED ALWAYS, an identity value is always generated for a table row. When a corresponding row in the input file already contains a value for the identity column, the row cannot be inserted into the table and the import operation will fail.

To prevent such failure, the following examples show how to specify parameters in the SYSCS_UTIL.SYSCS_IMPORT_DATA and SYSCS_UTIL.SYSCS_IMPORT_DATA_LOBS_FROM_EXTFILE procedures to ignore data for the identity column from the file, and omit the column name from the insert column list.

The following table definition contains an identity column, \( c2 \) and is used in the examples below:

```sql
CREATE TABLE tab1 (c1 CHAR(30), c2 INT GENERATED ALWAYS AS IDENTITY, c3 REAL, c4 CHAR(1))
```

- Suppose that you want to import data into \( \text{tab1} \) from a file \( \text{myfile.del} \) that does not have identity column information. The \( \text{myfile.del} \) file contains three fields with the following data:

```
Robert,45.2,J
Mike,76.9,K
Leo,23.4,I
```

To import the data, you must explicitly list the column names in the \( \text{tab1} \) table except for the identity column \( c2 \) when you call the procedure. For example:

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_DATA (NULL, 'TAB1', 'C1,C3,C4', null, 'myfile.del',null, null, null, 0)
```

- Suppose that you want to import data into \( \text{tab1} \) from a file \( \text{empfile.del} \) that also has identity column information. The file contains three fields with the following data:

```
Robert,1,45.2,J
Mike,2,23.4,I
Leo,3,23.4,I
```

To import the data, you must explicitly specify an insert column list without the identity column \( c2 \) and specify the column indexes without identity column data when you call the procedure. For example:

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_DATA (NULL, 'TAB1', 'C1,C3,C4', '1,3,4', 'empfile.del',null, null, null, 0)
```

Identity column is GENERATED BY DEFAULT
If the identity column is defined as GENERATED BY DEFAULT, an identity value is only generated for a table row if no explicit value is given. This means that you have several options, depending on the contents of your input file, and the desired outcome of the import processing:

- You may omit the identity column from the insert column list, in which case Derby will generate a new value for the identity column for each input row. You may use
this option whether or not the input file contains values for the identity column, but note that if the input file contains values for the identity column, you must also then omit the identity column from the column indexes when you call the procedure.

• You may include the identity column in the insert column list, in which case Derby will use the column values from the input file. Of course, this option is only available if the input file actually contains values for the identity column.

The following table definition contains an identity column, $c_2$ and is used in the examples below:

```
CREATE TABLE tab1 (c1 CHAR(30),
    c2 INT GENERATED BY DEFAULT AS IDENTITY,
    c3 REAL, c4 CHAR(1))
```

• Suppose that you want to import data into $\text{tab1}$ from a file $\text{myfile.del}$ that does not have identity column information. The $\text{myfile.del}$ file contains three fields with the following data:

  Robert,45.2,J  
  Mike,76.9,K  
  Leo,23.4,I

To import the data, you must explicitly list the column names in the $\text{tab1}$ table except for the identity column $c_2$ when you call the procedure. For example:

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_DATA (NULL, 'TAB1', 'C1,C3,C4',
null, 'myfile.del',null, null, null, 0)
```

• Suppose that you want to import data into $\text{tab1}$ from a file $\text{empfile.del}$ that also has identity column information. The file contains three fields with the following data:

  Robert,1,45.2,J  
  Mike,2,23.4,I  
  Leo,3,23.4,I

In this case, suppose that you wish to use the existing identity column values from the input file. To import the data, you may simply pass null for the insert column list and column indexes parameters when you call the procedure. For example:

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_DATA (NULL, 'TAB1', NULL,
null, 'empfile.del',null, null, null, 0)
```

• Suppose (again) that you want to import data into $\text{tab1}$ from a file $\text{empfile.del}$ that also has identity column information, but in this case, suppose that you do not wish to use the identity column values from the input file, but would prefer to allow Derby to generate new identity column values instead. In this case, to import the data, you must specify an insert column list without the identity column $c_2$ and specify the column indexes without identity column data when you call the procedure. For example:

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_DATA (NULL, 'TAB1', 'C1,C3,C4',
'1,3,4', 'empfile.del',null, null, null, 0)
```

### Exporting data using the built-in procedures

You can use the Derby export procedures to export all of the data from table or query, or to export LOB data separately from the other data.

1. Choose the correct procedure for the type of export that you want to perform. For examples of these procedures, see Examples of bulk import and export.

<table>
<thead>
<tr>
<th>Type of export</th>
<th>Procedure to use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To export all the data from a table to a single export file, including the LOB data

```
SYCS_UTIL.SYCS_EXPORT_TABLE (IN SCHEMANAME VARCHAR(128), IN TABLENAME VARCHAR(128), IN FILENAME VARCHAR(32672), IN COLUMNDELMITER CHAR(1), IN CHARACTERDELMITER CHAR(1), IN CODESET VARCHAR(128))
```

To export all the data from a table, and place the LOB data into a separate export file

```
SYCS_UTIL.SYCS_EXPORT_TABLE_LOBS_TO_EXTFILE (IN SCHEMANAME VARCHAR(128), IN TABLENAME VARCHAR(128), IN FILENAME VARCHAR(32672), IN COLUMNDELMITER CHAR(1), IN CHARACTERDELMITER CHAR(1), IN CODESET VARCHAR(128), IN LOBSFILENAME VARCHAR(32672))
```

A reference to the location of the LOB data is placed in the LOB column in the main export file.

To export the result of a SELECT statement to a single file, including the LOB data

```
SYCS_UTIL.SYCS_EXPORT_QUERY (IN SELECTSTATEMENT VARCHAR(32672), IN FILENAME VARCHAR(32672), IN COLUMNDELMITER CHAR(1), IN CHARACTERDELMITER CHAR(1), IN CODESET VARCHAR(128))
```

To export the result of a SELECT statement to a main export file, and place the LOB data into a separate export file

```
SYCS_UTIL.SYCS_EXPORT_QUERY_LOBS_TO_EXTFILE (IN SELECTSTATEMENT VARCHAR(32672), IN FILENAME VARCHAR(32672), IN COLUMNDELMITER CHAR(1), IN CHARACTERDELMITER CHAR(1), IN CODESET VARCHAR(128), IN LOBSFILENAME VARCHAR(32672))
```

A reference to the LOB data is written to the main export file.

### Parameters for the export procedures

The Derby export procedures use specific parameters.

**SCHEMANAME**

Specifies the schema of the table. You can specify a NULL value to use the default schema name. The SCHEMANAME parameter takes an input argument that is a VARCHAR (128) data type.

**SELECTSTATEMENT**

Specifies the SELECT statement query that returns the data to be exported. Specifying a NULL value will result in an error. The SELECTSTATEMENT parameter takes an input argument that is a VARCHAR (32672) data type.

**TABLENAME**

Specifies the table name of the table or view from which the data is to be exported. This table cannot be a system table or a declared temporary table. The string must exactly match the case of the table name. Specifying a NULL value results in an error. The TABLENAME parameter takes an input argument that is a VARCHAR (128) data type.

**FILENAME**

Specifies the file to which the data is to be exported. If the path is omitted, the current working directory is used. If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information. The specified location of the file should refer to the server-side location if you are using the Network Server. Specifying a NULL value results in an error. The FILENAME parameter takes an input argument that is a VARCHAR (32672) data type.

**COLUMNDELMITER**
Specifies a column delimiter. The specified character is used in place of a comma to signify the end of a column. You can specify a NULL value to use the default value of a comma. The COLUMNDELIMITER parameter must be a CHAR (1) data type.

**CHARACTERDELIMITER**

Specifies a character delimiter. The specified character is used in place of double quotation marks to enclose a character string. You can specify a NULL value to use the default value of a double quotation mark. The CHARACTERDELIMITER parameter takes an input argument that is a CHAR (1) data type.

**CODESET**

Specifies the code set of the data in the export file. The code set name should be one of the Java-supported character encoding sets. Data is converted from the database code page to the specified code page before writing to the file. You can specify a NULL value to write the data in the same code page as the JVM in which it is being executed. The CODESET parameter takes an input argument that is a VARCHAR (128) data type.

**LOBSFILENAME**

Specifies the file that the large object data is exported to. If the path is omitted, the lob file is created in the same directory as the main export file. If you specify the name of an existing file, the export utility overwrites the contents of the file. The data is not appended to the file. If you are using the Network Server, the file should be in a server-side location. Specifying a NULL value results in an error. The LOBSFILENAME parameter takes an input argument that is a VARCHAR (32672) data type.

If you create a schema, table, or column name as a non-delimited identifier, you must pass the name to the export procedure using all uppercase characters. If you created a schema or table name as a delimited identifier, you must pass the name to the export procedure using the same case that was used when it was created.

### Examples of bulk import and export

All of the examples in this section are run using the `ij` utility.

**Example importing all data from a file**

The following example shows how to import data into the STAFF table in a sample database from the `myfile.del` file. The data will be appended to the existing data in the table.

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE
    (null,'STAFF','myfile.del',null,null,null,0);
```

**Example importing all data from a delimited file**

The following example shows how to import data into the STAFF table in a sample database from a delimited data file `myfile.del`. This example defines the percentage character (%) as the string delimiter, and a semicolon as the column delimiter. The data will be appended to the existing data in the table.

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE
    (null,'STAFF','c:\output\myfile.del',';','%',null,0);
```

**Example importing all data from a table, using a separate import file for the LOB data**

The following example shows how to import data into the STAFF table in a sample database from a delimited data file `staff.del`. The import file `staff.del` is the main import file and contains references that point to a separate file which contains the
LOB data. This example specifies a comma as the column delimiter. The data will be appended to the existing data in the table.

CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE_LOBS_FROM_EXTFILE(
    null, 'STAFF', 'c:\data\staff.del', ',', '', '', 'UTF-8', 0);

Example importing data into specific columns, using a separate import file for the LOB data

The following example shows how to import data into several columns of the STAFF table. The STAFF table includes a LOB column in a sample database. The import file staff.del is a delimited data file. The staff.del file contains references that point to a separate file which contains the LOB data. The data in the import file is formatted using double quotation marks (") as the string delimiter and a comma (,) as the column delimiter. The data will be appended to the existing data in the STAFF table.

CALL SYSCS_UTIL.SYSCS_IMPORT_DATA_LOBS_FROM_EXTFILE
    (null, 'STAFF', 'NAME,DEPT,SALARY,PICTURE', '2,3,4,6',
     'c:\data\staff.del', ',', '', '', 'UTF-8', 0);

Example exporting all data from a table to a single export file

The following example shows how to export data from the STAFF table in a sample database to the file myfile.del.

CALL SYSCS_UTIL.SYSCS_EXPORT_TABLE
    (null, 'STAFF', 'myfile.del', null, null, null);

Example exporting data from a table to a single delimited export file

The following example shows how to export data from the STAFF table to a delimited data file myfile.del with the percentage character (%) as the character delimiter, and a semicolon as the column delimiter from the STAFF table.

CALL SYSCS_UTIL.SYSCS_EXPORT_TABLE
    (null, 'STAFF', 'c:\output\myfile.del', ';', '%', null);

Example exporting all data from a table, using a separate export file for the LOB data

The following example shows how to export data from the STAFF table in a sample database to the main file staff.del and the LOB export file pictures.dat.

CALL SYSCS_UTIL.SYSCS_EXPORT_TABLE_LOBS_TO_EXTFILE
    (null, 'STAFF', 'c:\data\staff.del', ',', '', '', 'UTF-8', 'c:\data\pictures.dat');

Example exporting data from a query to a single export file

The following example shows how to export employee data in department 20 from the STAFF table in a sample database to the file awards.del.

CALL SYSCS_UTIL.SYSCS_EXPORT_QUERY
    ('SELECT * FROM STAFF WHERE dept=20',
     'c:\output\awards.del', null, null, null);

Example exporting data from a query, using a separate export file for the LOB data

The following example shows how to export employee data in department 20 from the STAFF table in a sample database to the main file staff.del and the lob data to the file pictures.dat.

CALL SYSCS_UTIL.SYSCS_EXPORT_QUERY_LOBS_TO_EXTFILE
    ('SELECT * FROM STAFF WHERE dept=20',
     'c:\data\staff.del', ',', '', '', 'UTF-8', 'c:\data\pictures.dat');
Import and export procedures from JDBC

You can run import and export procedures from a JDBC program.

The following code fragment shows how you might call the SYSCS_UTIL.SYSCS_EXPORT_TABLE procedure from Java. In this example, the procedure exports the data in the staff table in the default schema to the staff.dat file. A percentage (%) character is used to specify the column delimiter.

```java
PreparedStatement ps=conn.prepareStatement("CALL SYSCS_UTIL.SYSCS_EXPORT_TABLE (?,?,?,?,?,?)");
ps.setString(1,null);
ps.setString(2,"STAFF");
ps.setString(3,"staff.dat");
ps.setString(4,"%");
ps.setString(5,null);
ps.setString(6,null);
ps.execute();
```

How the Import and export procedures process NULL values

In a delimited file, a NULL value is exported as an empty field. The following example shows the export of a four-column row where the third column is empty:

7,95,,Happy Birthday

The import procedures work the same way; an empty field is imported as a NULL value.

CODESET values for import and export procedures

Import and export procedures accept arguments to specify codeset values. You can specify the codeset (character encoding) for import and export procedures to override the system default.

The following table contains a sample of the character encoding that is supported by JDK 1.x. To review the complete list of character encodings, refer to your Java documentation.

**Table 5. Sample character encodings**
This table contains sample character encodings supported by JDK1.x.

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

**Examples of specifying the codeset in import and export procedures**

The following example shows how to specify UTF-8 encoding to export to the `staff.dat` table:

```sql
CALL SYSCS_UTIL.SYSCS_EXPORT_TABLE
  (NULL,'STAFF','staff.dat',NULL,NULL,'UTF-8')
```

The following example shows how to specify UTF-8 encoding to import from the `staff.dat` table:

```sql
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE
  (NULL,'STAFF','staff.dat',NULL,NULL,'UTF-8',0)
```
Storing jar files in a database

SQLJ.install_jar, SQLJ.remove_jar, and SQLJ.replace_jar, are a set of procedures in the SQL schema that allow you to store jar files in the database.

Your jar file has a physical name (the name you gave it when you created it) and a Derby name (the Derby identifier you give it when you load it into a particular schema). Its Derby name is an SQL92Identifier; it can be delimited and must be unique within a schema. A single schema can store more than one jar file.

Adding a Jar File

To add a jar file using SQL syntax:

```sql
CALL SQLJ.install_jar('jarFilePath', qualifiedJarName, 0)
```

- `jarFilePath`
  The path and physical name of the jar file to add or use as a replacement. For example:
  ```text
d:/todays_build/tours.jar
```
- `qualifiedJarName`
  The Derby name of the jar file, qualified by the schema name. Two examples:
  ```text
  MYSHEMA.Sample1
  MYSHEMA."Sample2"
  ```

Removing a jar file

To remove a jar file using SQL syntax:

```sql
CALL SQLJ.remove_jar (qualifiedJarName, 0)
```

Replacing a jar file

To replace a jar file using SQL syntax:

```sql
CALL SQLJ.replace_jar('jarFilePath', qualifiedJarName)
```

- `jarFilePath`
  The path and physical name of the jar file to add or use as a replacement. For example:
  ```text
d:/todays_build/tours.jar
```
- `qualifiedJarName`
  The Derby name of the jar file, qualified by the schema name. Two examples:
  ```text
  MYSHEMA.Sample1 -- a delimited identifier.
  MYSHEMA."Sample2"
  ```
Installing a jar example

- Complete SQL example for installing a jar:

```sql
CALL SQLJ.install_jar('d:\todays_build\tours.jar', 'APP."ToursLogic!"', 0);
```

For more information about storing classes in a database, see the *Derby Developer's Guide*. 
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 Derby to set the `DERBY_HOME` and `JAVA_HOME` environment variables and to add `DERBY_HOME/bin` to your path. Then use the following command:

```shell
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:

  ```shell
  (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.

```shell
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:

```shell
java -jar C:\db-derby-10.6.0.0-bin\lib\derbyrun.jar sysinfo
-------------------- Java Information --------------------
Java Version: 1.6.0_20
Java Vendor: Sun Microsystems Inc.
Java home: C:\jdk1.6.0_20\jre
Java classpath: C:\db-derby-10.6.0.0-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: 1.6
java.runtime.version: 1.6.0_20-b02
----------------- Derby Information -----------------
JRE - JDBC: Java SE 6 - JDBC 4.0
[C:\db-derby-10.6.0.0-bin\lib\derby.jar] 10.6.0.0 - (908506)
[C:\db-derby-10.6.0.0-bin\lib\derbytools.jar] 10.6.0.0 - (908506)
[C:\db-derby-10.6.0.0-bin\lib\derbynet.jar] 10.6.0.0 - (908506)
[C:\db-derby-10.6.0.0-bin\lib\derbyclient.jar] 10.6.0.0 - (908506)
-----------------------------------------------
----------------- Locale Information -----------------
Current Locale : [English/United States [en_US]]
Found support for locale: [cs]
  version: 10.6.0.0 - (908506)
Found support for locale: [de_DE]
  version: 10.6.0.0 - (908506)
Found support for locale: [es]
  version: 10.6.0.0 - (908506)
Found support for locale: [fr]
  version: 10.6.0.0 - (908506)
Found support for locale: [hu]
  version: 10.6.0.0 - (908506)
```
Found support for locale: [it]
  version: 10.6.0.0 - (908506)
Found support for locale: [ja_JP]
  version: 10.6.0.0 - (908506)
Found support for locale: [ko_KR]
  version: 10.6.0.0 - (908506)
Found support for locale: [pl]
  version: 10.6.0.0 - (908506)
Found support for locale: [pt_BR]
  version: 10.6.0.0 - (908506)
Found support for locale: [ru]
  version: 10.6.0.0 - (908506)
Found support for locale: [zh_CN]
  version: 10.6.0.0 - (908506)
Found support for locale: [zh_TW]
  version: 10.6.0.0 - (908506)

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 Derby to set the DERBY_HOME and JAVA_HOME environment variables and to add DERBY_HOME/bin to your path. Then use the following command:

```plaintext
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):

```plaintext
(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.

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

## Using dblook

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

```plaintext
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 Derby 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:

```plaintext
'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

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

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

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

```java
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
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 *Derby Reference Manual*. 
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.