

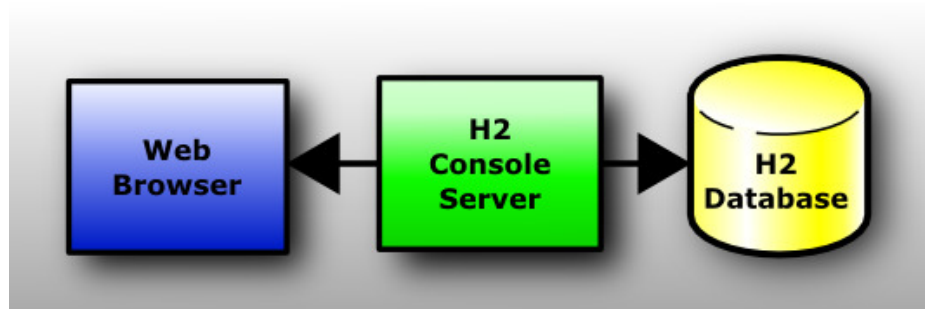
[Translate](#)**Search:****Home**[Download](#)[Cheat Sheet](#)**Documentation**[Quickstart](#)[Installation](#)[Tutorial](#)[Features](#)[Performance](#)[Advanced](#)**Reference**[SQL Grammar](#)[Functions](#)[Data Types](#)[Javadoc](#)[PDF \(1 MB\)](#)**Support**[FAQ](#)[Error Analyzer](#)[Google Group \(English\)](#)[Google Group \(Japanese\)](#)[Google Group \(Chinese\)](#)**Appendix**[JaQu](#)[Build](#)[History & Roadmap](#)[Links](#)[License](#)

## Tutorial

[Starting and Using the H2 Console](#)[Special H2 Console Syntax](#)[Settings of the H2 Console](#)[Connecting to a Database using JDBC](#)[Creating New Databases](#)[Using the Server](#)[Using Hibernate](#)[Using TopLink and Glassfish](#)[Using EclipseLink](#)[Using Databases in Web Applications](#)[CSV \(Comma Separated Values\) Support](#)[Upgrade, Backup, and Restore](#)[Command Line Tools](#)[The Shell Tool](#)[Using OpenOffice Base](#)[Java Web Start / JNLP](#)[Using a Connection Pool](#)[Fulltext Search](#)[User-Defined Variables](#)[Date and Time](#)[Using Spring](#)


## Starting and Using the H2 Console

The H2 Console application lets you access a SQL database using a browser interface. This can be a H2 database, or another database that supports the JDBC API.



This is a client / server application, so both a server and a client (a browser) are required to run it.

Depending on your platform and environment, there are multiple ways to start the application:

OS	Start
Windows	Click [Start], [All Programs], [H2], and [H2 Console (Command Line)] When using the Sun JDK 1.5, a window with the title 'H2 Console ' should appear. When using the Sun JDK 1.6, an icon will be added to the system tray:  If you don't get the window and the system tray icon, then maybe Java is not installed correctly (in this case, try another way to start the application). A browser window should open and point to the Login page at <a href="http://localhost:8082">http://localhost:8082</a> .
Windows	Open a file browser, navigate to <code>h2/bin</code> , and double click on <code>h2.bat</code> . A console window appears. If there is a problem, you will see an error message in this window. A browser window will open and point to the Login page (URL: <a href="http://localhost:8082">http://localhost:8082</a> ).
Any	Double click on the <code>h2*.jar</code> file. This only works if the <code>.jar</code> suffix is associated with java.
Any	Open a console window, navigate to the directory <code>h2/bin</code> and type: <div><pre>java -cp h2*.jar org.h2.tools.Server</pre></div>

## Firewall

If you start the server, you may get a security warning from the firewall (if you have installed one). If you don't want other computers in the network to access the application on your machine, you can let the firewall block those connections. The connection from the local machine will still work. Only if you want other computers to access the database on this computer, you need allow remote connections in the firewall.

It has been reported that when using Kaspersky 7.0 with firewall, the H2 Console is very slow when connecting over the IP address. A workaround is to connect using localhost, however this only works on the local machine.

A small firewall is already built into the server: other computers may not connect to the server by default. To change this, go to 'Preferences' and select 'Allow connections from other computers'.

## Testing Java

To find out which version of Java is installed, open a command prompt and type:

```
java -version
```

If you get an error message, you may need to add the Java binary directory to the path environment variable.

## Error Message 'Port may be in use'

You can only start one instance of the H2 Console, otherwise you will get the following error message: "The Web server could not be started. Possible cause: another server is already running...". It is possible to start multiple console applications on the same computer (using different ports), but this is usually not required as the console supports multiple concurrent connections.

## Using another Port

If the port is in use by another application, you may want to start the H2 Console on a different port. This can be done by changing the port in the file `.h2.server.properties`. This file is stored in the user directory (for Windows, this is usually in `Documents and Settings/<username>`). The relevant entry is `webPort`.

## Connecting to the Server using a Browser

If the server started successfully, you can connect to it using a web browser. JavaScript needs to be enabled. If you started the server on the same computer as the browser, open the URL `http://localhost:8082`. If you want to connect to the application from another computer, you need to provide the IP address of the server, for example: `http://192.168.0.2:8082`. If you enabled SSL on the server side, the URL needs to start with `https://`.

## Multiple Concurrent Sessions

Multiple concurrent browser sessions are supported. As that the database objects reside on the server, the amount of concurrent work is limited by the memory available to the server application.

## Login

At the login page, you need to provide connection information to connect to a database. Set the JDBC driver class of your database, the JDBC URL, user name and password. If you are done, click [Connect].

You can save and reuse previously saved settings. The settings are stored in a properties file (see [Settings of the H2 Console](#)).

## Error Messages

Error messages are shown in red. You can show/hide the stack trace of the exception by clicking on the message.

## Adding Database Drivers

Additional database drivers to connect to other databases (MySQL, PostgreSQL, HSQLDB,...) can be registered by adding the Jar file location of the driver to the environment variables `H2DRIVERS` or `CLASSPATH`. Example (Windows): to add the database driver library `C:\Programs\hsqldb\lib\hsqldb.jar`, set the environment variable `H2DRIVERS` to `C:\Programs\hsqldb\lib\hsqldb.jar`.

Multiple drivers can be set; each entry needs to be separated with a `;` (Windows) or `:` (other operating systems). Spaces in the path names are supported. The settings must not be quoted.

## Using the H2 Console

The H2 Console application has three main panels: the toolbar on top, the tree on the left, and the query / result panel on the right. The database objects (for example, tables) are listed on the left panel. Type in a SQL command on the query panel and click 'Run'. The result of the command appears just below the command.

## Inserting Table Names or Column Names

The table name and column names can be inserted in the script by clicking them in the tree. If you click on a table while the query is empty, then `SELECT * FROM ...` is added as well. While typing a query, the table that was used is automatically expanded in the tree. For example if you type `SELECT * FROM TEST T WHERE T.` then the table `TEST` is automatically expanded in the tree.

## Disconnecting and Stopping the Application

To log out of the database, click 'Disconnect' in the toolbar panel. However, the server is still running and ready to accept new sessions.

To stop the server, right click on the system tray icon and select [Exit]. If you don't have the system tray icon, navigate to [Preferences] and click [Shutdown], press [Ctrl]+[C] in the console where the server was started (Windows), or close the console window.

## Special H2 Console Syntax

The H2 Console supports a few built-in commands. Those are interpreted within the H2 Console, that means they work with any database. They need to be at the beginning of a statement (before any remarks), otherwise they are not parsed correctly. If in doubt, add ';' before the command.

Command(s)	Description
@autocommit_true; @autocommit_false;	Enable or disable autocommit.
@cancel;	Cancel the currently running statement.
@columns null null TEST; @index_info null null TEST; @tables; @tables null null TEST;	Call the corresponding DatabaseMetaData.get method. Patterns are case sensitive (usually identifiers are uppercase). For information about the parameters, see the Javadoc documentation. Missing parameters at the end are set to null. The complete list of commands is: @attributes @best_row_identifier @catalogs @columns @column_privileges @cross_references @exported_keys @imported_keys @index_info @primary_keys @procedures @procedure_columns @schemas @super_tables @super_types @tables @table_privileges @table_types @type_info @udts @version_columns
@edit select * from test;	Use an updatable result set.
@generated insert into test() values();	Show the result of Statement.getGeneratedKeys().
@history;	Show the command history.
@info;	Display the result of various Connection and DatabaseMetaData methods.
@list select * from test;	Show the result set in list format (each column on its own line, with row numbers).
@loop 1000 select ?, ?/*rnd*/; @loop 1000 @statement select ?;	Run the statement this many times. Parameters ( ? ) are set using a loop from 0 up to x - 1. Random values are used for each ?/*rnd*/. A Statement object is used instead of a PreparedStatement if @statement is used. Result sets are read until ResultSet.next() returns false. Timing information is printed.
@maxrows 20;	Set the maximum number of rows to display.
@memory;	Show the used and free memory. This will call System.gc().
@meta select 1;	List the ResultSetMetaData after running the query.
@parameter_meta select ?;	Show the result of the PreparedStatement.getParameterMetaData() calls. The statement is not executed.
@prof_start; call hash('SHA256', '', 1000000); @prof_stop;	Start / stop the built-in profiling tool. The top 3 stack traces of the statement(s) between start and stop are listed (if there are 3).
@transaction_isolation; @transaction_isolation 2;	Display (without parameters) or change (with parameters 1, 2, 4, 8) the transaction isolation level.

## Settings of the H2 Console

The settings of the H2 Console are stored in a configuration file called .h2.server.properties in your user home directory. For Windows installations, the user home directory is usually C:\Documents and Settings\[username]. The configuration file contains the settings of the application and is automatically created when the H2 Console is first started.

## Connecting to a Database using JDBC

To connect to a database, a Java application first needs to load the database driver, and then get a connection. A simple way to do that is using the following code:

```
import java.sql.*;  
public class Test {  
    public static void main(String[] a)  
        throws Exception {
```

```
Class.forName("org.h2.Driver");
Connection conn = DriverManager.
    getConnection("jdbc:h2:~/test", "sa", "");
// add application code here
conn.close();
}
}
```

This code first loads the driver ( `Class.forName(...)` ) and then opens a connection (using `DriverManager.getConnection()` ). The driver name is "org.h2.Driver". The database URL always needs to start with `jdbc:h2:` to be recognized by this database. The second parameter in the `getConnection()` call is the user name ( `sa` for System Administrator in this example). The third parameter is the password. In this database, user names are not case sensitive, but passwords are.

## Creating New Databases

By default, if the database specified in the URL does not yet exist, a new (empty) database is created automatically. The user that created the database automatically becomes the administrator of this database.

Auto-creating new database can be disabled, see [Opening a Database Only if it Already Exists](#).

## Using the Server

H2 currently supports three server: a web server (for the H2 Console), a TCP server (for client/server connections) and an PG server (for PostgreSQL clients). The servers can be started in different ways, one is using the Server tool.

### Starting the Server Tool from Command Line

To start the Server tool from the command line with the default settings, run:

```
java -cp h2*.jar org.h2.tools.Server
```

This will start the tool with the default options. To get the list of options and default values, run:

```
java -cp h2*.jar org.h2.tools.Server -?
```

There are options available to use other ports, and start or not start parts.

### Connecting to the TCP Server

To remotely connect to a database using the TCP server, use the following driver and database URL:

- JDBC driver class: `org.h2.Driver`
- Database URL: `jdbc:h2:tcp://localhost/~/test`

For details about the database URL, see also in Features.

### Starting the TCP Server within an Application

Servers can also be started and stopped from within an application. Sample code:

```
import org.h2.tools.Server;
...
// start the TCP Server
Server server = Server.createTcpServer(args).start();
...
// stop the TCP Server
server.stop();
```

### Stopping a TCP Server from Another Process

The TCP server can be stopped from another process. To stop the server from the command line, run:

```
java org.h2.tools.Server -tcpShutdown tcp://localhost:9092
```

To stop the server from a user application, use the following code:

```
org.h2.tools.Server.shutdownTcpServer("tcp://localhost:9092");
```

This function will only stop the TCP server. If other server were started in the same process, they will continue

to run. To avoid recovery when the databases are opened the next time, all connections to the databases should be closed before calling this method. To stop a remote server, remote connections must be enabled on the server. Shutting down a TCP server can be protected using the option `-tcpPassword` (the same password must be used to start and stop the TCP server).

## Using Hibernate

This database supports Hibernate version 3.1 and newer. You can use the HSQLDB Dialect, or the native H2 Dialect. Unfortunately the H2 Dialect included in some versions of Hibernate is buggy. A [patch for Hibernate](#) has been submitted and is now applied. The dialect for the newest version of Hibernate is also available at `src/tools/org/hibernate/dialect/H2Dialect.java.txt`. You can rename it to `H2Dialect.java` and include this as a patch in your application, or upgrade to a version of Hibernate where this is fixed.

## Using TopLink and Glassfish

To use H2 with Glassfish (or Sun AS), set the Datasource Classname to `org.h2.jdbcx.JdbcDataSource`. You can set this in the GUI at Application Server - Resources - JDBC - Connection Pools, or by editing the file `sun-resources.xml`: at element `jdbc-connection-pool`, set the attribute `datasource-classname` to `org.h2.jdbcx.JdbcDataSource`.

The H2 database is compatible with HSQLDB and PostgreSQL. To take advantage of H2 specific features, use the `H2Platform`. The source code of this platform is included in H2 at `src/tools/oracle/toplink/essentials/platform/database/DatabasePlatform.java.txt`. You will need to copy this file to your application, and rename it to `.java`. To enable it, change the following setting in `persistence.xml`:

```
<property
  name="toplink.target-database"
  value="oracle.toplink.essentials.platform.database.H2Platform"/>
```

In old versions of Glassfish, the property name is `toplink.platform.class.name`.

## Using EclipseLink

To use H2 in EclipseLink, use the platform class `org.eclipse.persistence.platform.database.H2Platform`. If this platform is not available in your version of EclipseLink, you can use the `OraclePlatform` instead in many case. See also [H2Platform](#).

## Using Databases in Web Applications

There are multiple ways to access a database from within web applications. Here are some examples if you use Tomcat or JBoss.

### Embedded Mode

The (currently) simplest solution is to use the database in the embedded mode, that means open a connection in your application when it starts (a good solution is using a Servlet Listener, see below), or when a session starts. A database can be accessed from multiple sessions and applications at the same time, as long as they run in the same process. Most Servlet Containers (for example Tomcat) are just using one process, so this is not a problem (unless you run Tomcat in clustered mode). Tomcat uses multiple threads and multiple classloaders. If multiple applications access the same database at the same time, you need to put the database jar in the `shared/lib` or `server/lib` directory. It is a good idea to open the database when the web application starts, and close it when the web application stops. If using multiple applications, only one (any) of them needs to do that. In the application, an idea is to use one connection per Session, or even one connection per request (action). Those connections should be closed after use if possible (but it's not that bad if they don't get closed).

### Server Mode

The server mode is similar, but it allows you to run the server in another process.

### Using a Servlet Listener to Start and Stop a Database

Add the `h2*.jar` file to your web application, and add the following snippet to your `web.xml` file (between the `context-param` and the `filter` section):

```
<listener>
  <listener-class>org.h2.server.web.DbStarter</listener-class>
</listener>
```

For details on how to access the database, see the file `DbStarter.java`. By default this tool opens an embedded connection using the database URL `jdbc:h2:~/test`, user name `sa`, and password `sa`. If you want to use this

connection within your servlet, you can access as follows:

```
Connection conn = getServletContext().getAttribute("connection");
```

DbStarter can also start the TCP server, however this is disabled by default. To enable it, use the parameter `db.tcpServer` in the file `web.xml`. Here is the complete list of options. These options need to be placed between the `description` tag and the `listener` / `filter` tags:

```
<context-param>
  <param-name>db.url</param-name>
  <param-value>jdbc:h2:~/test</param-value>
</context-param>
<context-param>
  <param-name>db.user</param-name>
  <param-value>sa</param-value>
</context-param>
<context-param>
  <param-name>db.password</param-name>
  <param-value>sa</param-value>
</context-param>
<context-param>
  <param-name>db.tcpServer</param-name>
  <param-value>-tcpAllowOthers</param-value>
</context-param>
```

When the web application is stopped, the database connection will be closed automatically. If the TCP server is started within the DbStarter, it will also be stopped automatically.

## Using the H2 Console Servlet

The H2 Console is a standalone application and includes its own web server, but it can be used as a servlet as well. To do that, include the `h2*.jar` file in your application, and add the following configuration to your `web.xml`:

```
<servlet>
  <servlet-name>H2Console</servlet-name>
  <servlet-class>org.h2.server.web.WebServlet</servlet-class>
  <!--
  <init-param>
    <param-name>webAllowOthers</param-name>
    <param-value></param-value>
  </init-param>
  <init-param>
    <param-name>trace</param-name>
    <param-value></param-value>
  </init-param>
  -->
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>H2Console</servlet-name>
  <url-pattern>/console/*</url-pattern>
</servlet-mapping>
```

For details, see also `src/tools/WEB-INF/web.xml`.

To create a web application with just the H2 Console, run the following command:

```
build warConsole
```

## CSV (Comma Separated Values) Support

The CSV file support can be used inside the database using the functions `CSVREAD` and `CSVWRITE`, or it can be used outside the database as a standalone tool.

### Writing a CSV File from Within a Database

The built-in function `CSVWRITE` can be used to create a CSV file from a query. Example:

```
CREATE TABLE TEST(ID INT, NAME VARCHAR);
INSERT INTO TEST VALUES(1, 'Hello'), (2, 'World');
CALL CSVWRITE('test.csv', 'SELECT * FROM TEST');
```

## Reading a CSV File from Within a Database

A CSV file can be read using the function `CSVREAD`. Example:

```
SELECT * FROM CSVREAD('test.csv');
```

## Writing a CSV File from a Java Application

The `Csv` tool can be used in a Java application even when not using a database at all. Example:

```
import java.sql.*;
import org.h2.tools.Csv;
import org.h2.tools.SimpleResultSet;
public class TestCsv {
    public static void main(String[] args) throws Exception {
        SimpleResultSet rs = new SimpleResultSet();
        rs.addColumn("NAME", Types.VARCHAR, 255, 0);
        rs.addColumn("EMAIL", Types.VARCHAR, 255, 0);
        rs.addRow("Bob Meier", "bob.meier@abcde.abc");
        rs.addRow("John Jones", "john.jones@abcde.abc");
        Csv.getInstance().write("data/test.csv", rs, null);
    }
}
```

## Reading a CSV File from a Java Application

It is possible to read a CSV file without opening a database. Example:

```
import java.sql.*;
import org.h2.tools.Csv;
public class TestCsv {
    public static void main(String[] args) throws Exception {
        ResultSet rs = Csv.getInstance().
            read("data/test.csv", null, null);
        ResultSetMetaData meta = rs.getMetaData();
        while (rs.next()) {
            for (int i = 0; i < meta.getColumnCount(); i++) {
                System.out.println(
                    meta.getColumnLabel(i + 1) + ": " +
                    rs.getString(i + 1));
            }
            System.out.println();
        }
        rs.close();
    }
}
```

## Upgrade, Backup, and Restore

### Database Upgrade

The recommended way to upgrade from one version of the database engine to the next version is to create a backup of the database (in the form of a SQL script) using the old engine, and then execute the SQL script using the new engine.

### Backup using the Script Tool

There are different ways to backup a database. For example, it is possible to copy the database files. However, this is not recommended while the database is in use. Also, the database files are not human readable and quite large. The recommended way to backup a database is to create a compressed SQL script file. This can be done using the `Script` tool:

```
java org.h2.tools.Script -url jdbc:h2:~/test -user sa -script test.zip -options compression zip
```

It is also possible to use the SQL command `SCRIPT` to create the backup of the database. For more information about the options, see the SQL command `SCRIPT`. The backup can be done remotely, however the file will be created on the server side. The built in FTP server could be used to retrieve the file from the server.

### Restore from a Script

To restore a database from a SQL script file, you can use the `RunScript` tool:

```
java org.h2.tools.RunScript -url jdbc:h2:~/test -user sa -script test.zip -options compression zip
```

For more information about the options, see the SQL command `RUNSCRIPT`. The restore can be done remotely, however the file needs to be on the server side. The built in FTP server could be used to copy the file to the server. It is also possible to use the SQL command `RUNSCRIPT` to execute a SQL script. SQL script files may contain references to other script files, in the form of `RUNSCRIPT` commands. However, when using the server mode, the references script files need to be available on the server side.

## Online Backup

The `BACKUP` SQL statement and the `Backup` tool both create a zip file with all database files. However, the contents of this file are not human readable.

Unlike than the `SCRIPT` statement, the `BACKUP` statement does not lock the database objects when using the multi-threaded mode, and therefore does not block other users. The resulting backup is transactionally consistent:

```
BACKUP TO 'backup.zip'
```

The `Backup` tool (`org.h2.tools.Backup`) can not be used to create an online backup; the database must not be in use while running this program.

Creating a backup by copying the database files while the database is running is not supported, except if the file systems support creating snapshots. The problem is that it can't be guaranteed that the data is copied in the right order.

## Command Line Tools

This database comes with a number of command line tools. To get more information about a tool, start it with the parameter `'-?'`, for example:

```
java -cp h2*.jar org.h2.tools.Backup -?
```

The command line tools are:

- `Backup` creates a backup of a database.
- `ChangeFileEncryption` allows changing the file encryption password or algorithm of a database.
- `Console` starts the browser based H2 Console.
- `ConvertTraceFile` converts a `.trace.db` file to a Java application and SQL script.
- `CreateCluster` creates a cluster from a standalone database.
- `DeleteDbFiles` deletes all files belonging to a database.
- `Recover` helps recovering a corrupted database.
- `Restore` restores a backup of a database.
- `RunScript` runs a SQL script against a database.
- `Script` allows converting a database to a SQL script for backup or migration.
- `Server` is used in the server mode to start a H2 server.
- `Shell` is a command line database tool.

The tools can also be called from an application by calling the `main` or another public method. For details, see the Javadoc documentation.

## The Shell Tool

The `Shell` tool is a simple interactive command line tool. To start it, type:

```
java -cp h2*.jar org.h2.tools.Shell
```

You will be asked for a database URL, JDBC driver, user name, and password. The connection setting can also be set as command line parameters. After connecting, you will get the list of options. The built-in commands don't need to end with a semicolon, but SQL statements are only executed if the line ends with a semicolon `;`. This allows to enter multi-line statements:

```
sql> select * from test
...> where id = 0;
```

By default, results are printed as a table. For results with many column, consider using the list mode:

```
sql> list
Result list mode is now on
```



```
sql> select * from test;
ID : 1
NAME: Hello

ID : 2
NAME: World
(2 rows, 0 ms)
```

## Using OpenOffice Base

OpenOffice.org Base supports database access over the JDBC API. To connect to a H2 database using OpenOffice Base, you first need to add the JDBC driver to OpenOffice. The steps to connect to a H2 database are:

- Start OpenOffice Writer, go to [Tools], [Options]
- Make sure you have selected a Java runtime environment in OpenOffice.org / Java
- Click [Class Path...], [Add Archive...]
- Select your h2 jar file (location is up to you, could be wherever you choose)
- Click [OK] (as much as needed), stop OpenOffice (including the Quickstarter)
- Start OpenOffice Base
- Connect to an existing database; select [JDBC]; [Next]
- Example datasource URL: jdbc:h2:~/test
- JDBC driver class: org.h2.Driver

Now you can access the database stored in the current users home directory.

To use H2 in NeoOffice (OpenOffice without X11):

- In NeoOffice, go to [NeoOffice], [Preferences]
- Look for the page under [NeoOffice], [Java]
- Click [Class Path], [Add Archive...]
- Select your h2 jar file (location is up to you, could be wherever you choose)
- Click [OK] (as much as needed), restart NeoOffice.

Now, when creating a new database using the "Database Wizard" :

- Click [File], [New], [Database].
- Select [Connect to existing database] and the select [JDBC]. Click next.
- Example datasource URL: jdbc:h2:~/test
- JDBC driver class: org.h2.Driver

Another solution to use H2 in NeoOffice is:

- Package the h2 jar within an extension package
- Install it as a Java extension in NeoOffice

This can be done by create it using the NetBeans OpenOffice plugin. See also [Extensions Development](#).

## Java Web Start / JNLP

When using Java Web Start / JNLP (Java Network Launch Protocol), permissions tags must be set in the .jnlp file, and the application .jar file must be signed. Otherwise, when trying to write to the file system, the following exception will occur: java.security.AccessControlException : access denied ( java.io.FilePermission ... read ). Example permission tags:

```
<security>
  <all-permissions/>
</security>
```

## Using a Connection Pool

For H2, opening a connection is fast if the database is already open. Still, using a connection pool improves performance if you open and close connections a lot. A simple connection pool is included in H2. It is based on the [Mini Connection Pool Manager](#) from Christian d'Heureuse. There are other, more complex, open source connection pools available, for example the [Apache Commons DBCP](#). For H2, it is about twice as faster to get a connection from the built-in connection pool than to get one using `DriverManager.getConnection()`. The build-in connection pool is used as follows:

```
import java.sql.*;
import org.h2.jdbcx.JdbcConnectionPool;
public class Test {
```

```
public static void main(String[] args) throws Exception {
    JdbcConnectionPool cp = JdbcConnectionPool.create(
        "jdbc:h2:~/test", "sa", "sa");
    for (int i = 0; i < args.length; i++) {
        Connection conn = cp.getConnection();
        conn.createStatement().execute(args[i]);
        conn.close();
    }
    cp.dispose();
}
```

## Fulltext Search

H2 includes two fulltext search implementations. One is using Apache Lucene, and the other (the native implementation) stores the index data in special tables in the database.

### Using the Native Fulltext Search

To initialize, call:

```
CREATE ALIAS IF NOT EXISTS FT_INIT FOR "org.h2.fulltext.FullText.init";
CALL FT_INIT();
```

You need to initialize it in each database where you want to use it. Afterwards, you can create a fulltext index for a table using:

```
CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR);
INSERT INTO TEST VALUES(1, 'Hello World');
CALL FT_CREATE_INDEX('PUBLIC', 'TEST', NULL);
```

PUBLIC is the schema name, TEST is the table name. The list of column names (column separated) is optional, in this case all columns are indexed. The index is updated in realtime. To search the index, use the following query:

```
SELECT * FROM FT_SEARCH('Hello', 0, 0);
```

This will produce a result set that contains the query needed to retrieve the data:

```
QUERY: "PUBLIC"."TEST" WHERE "ID"=1
```

To get the raw data, use `FT_SEARCH_DATA('Hello', 0, 0)`. The result contains the columns SCHEMA (the schema name), TABLE (the table name), COLUMNS (an array of column names), and KEYS (an array of objects). To join a table, use a join as in: `SELECT T.* FROM FT_SEARCH_DATA('Hello', 0, 0) FT, TEST T WHERE FT.TABLE='TEST' AND T.ID=FT.KEYS[0]`;

You can also call the index from within a Java application:

```
org.h2.fulltext.FullText.search(conn, text, limit, offset);
org.h2.fulltext.FullText.searchData(conn, text, limit, offset);
```

### Using the Lucene Fulltext Search

To use the Lucene full text search, you need the Lucene library in the classpath. How to do that depends on the application; if you use the H2 Console, you can add the Lucene jar file to the environment variables H2DRIVERS or CLASSPATH. To initialize the Lucene fulltext search in a database, call:

```
CREATE ALIAS IF NOT EXISTS FTL_INIT FOR "org.h2.fulltext.FullTextLucene.init";
CALL FTL_INIT();
```

You need to initialize it in each database where you want to use it. Afterwards, you can create a full text index for a table using:

```
CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR);
INSERT INTO TEST VALUES(1, 'Hello World');
CALL FTL_CREATE_INDEX('PUBLIC', 'TEST', NULL);
```

PUBLIC is the schema name, TEST is the table name. The list of column names (column separated) is optional, in this case all columns are indexed. The index is updated in realtime. To search the index, use the following query:

```
SELECT * FROM FTL_SEARCH('Hello', 0, 0);
```

This will produce a result set that contains the query needed to retrieve the data:

```
QUERY: "PUBLIC"."TEST" WHERE "ID"=1
```

To get the raw data, use `FTL_SEARCH_DATA('Hello', 0, 0);`. The result contains the columns `SCHEMA` (the schema name), `TABLE` (the table name), `COLUMNS` (an array of column names), and `KEYS` (an array of objects). To join a table, use a join as in: `SELECT T.* FROM FTL_SEARCH_DATA('Hello', 0, 0) FT, TEST T WHERE FT.TABLE='TEST' AND T.ID=FT.KEYS[0];`

You can also call the index from within a Java application:

```
org.h2.fulltext.FullTextLucene.search(conn, text, limit, offset);  
org.h2.fulltext.FullTextLucene.searchData(conn, text, limit, offset);
```

## User-Defined Variables

This database supports user-defined variables. Variables start with `@` and can be used wherever expressions or parameters are allowed. Variables are not persisted and session scoped, that means only visible from within the session in which they are defined. A value is usually assigned using the `SET` command:

```
SET @USER = 'Joe';
```

The value can also be changed using the `SET()` method. This is useful in queries:

```
SET @TOTAL = NULL;  
SELECT X, SET(@TOTAL, IFNULL(@TOTAL, 1.) * X) F FROM SYSTEM_RANGE(1, 50);
```

Variables that are not set evaluate to `NULL`. The data type of a user-defined variable is the data type of the value assigned to it, that means it is not necessary (or possible) to declare variable names before using them. There are no restrictions on the assigned values; large objects (LOBs) are supported as well.

## Date and Time

Date, time and timestamp values support ISO 8601 formatting, including time zone:

```
CALL TIMESTAMP '2008-01-01 12:00:00+01:00';
```

If the time zone is not set, the value is parsed using the current time zone setting of the system. Date and time information is stored in H2 database files in GMT (Greenwich Mean Time). If the database is opened using another system time zone, the date and time will change accordingly. If you want to move a database from one time zone to the other and don't want this to happen, you need to create a SQL script file using the `SCRIPT` command or `Script` tool, and then load the database using the `RUNSCRIPT` command or the `RunScript` tool in the new time zone.

## Using Spring

Use the following configuration to start and stop the H2 TCP server using the Spring Framework:

```
<bean id = "org.h2.tools.Server"  
      class="org.h2.tools.Server"  
      factory-method="createTcpServer"  
      init-method="start"  
      destroy-method="stop">  
    <constructor-arg value="-tcp,-tcpAllowOthers,true,-tcpPort,8043" />  
</bean>
```

The `destroy-method` will help prevent exceptions on hot-redeployment or when restarting the server.