Translate
Search:

Home
Download
Cheat Sheet

Documentation
Quickstart
Installation
Tutorial
Features
Security
Performance
Advanced

Reference
Commands
Functions
• Aggregate • Window

Data Types
SQL Grammar
System Tables
Javadoc
PDF (2 MB)

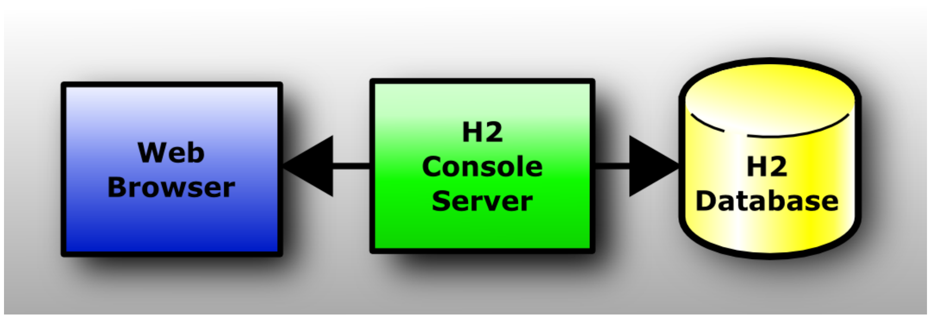
Support
FAQ
Error Analyzer
Google Group

Appendix
History
License
Build
Links
MVStore
Architecture
Migration to 2.0

Tutorial

Starting and Using the H2 Console

The H2 Console application lets you access a database using a browser. This can be a H2 database, or anoth 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 H2 Console:

OS	Start
Windows	Click [Start], [All Programs], [H2], and [H2 Console (Command Line)] 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 http://localhost:8082 .
Windows	Open a file browser, navigate to h2/bin , and double click on h2.bat . 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: http://localhost:8082).
Any	Double click on the h2*.jar file. This only works if the .jar suffix is associated with Java.
Any	Open a console window, navigate to the directory h2/bin , and type:

▲

```
java -jar h2*.jar
```

If the console startup procedure is unable to locate the default system web browser, an error message may be displayed. It is possible to explicitly tell H2 which program/script to use when opening a system web browser b setting either the BROWSER environment variable, or the h2.browser java property.

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 connectin over the IP address. A workaround is to connect using 'localhost'.

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 W 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 conso supports multiple concurrent connections.

Using another Port

If the default port of the H2 Console is already in use by another application, then a different port needs to be configured. The settings are stored in a properties file. For details, see Settings of the H2 Console. The releva entry is `webPort` .

If no port is specified for the TCP and PG servers, each service will try to listen on its default port. If the default port is already in use, a random port is used.

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. I you started the server on the same computer as the browser, open the URL `http://localhost:8082` . If you want connect to the application from another computer, you need to provide the IP address of the server, for examp `http://192.168.0.2:8082` . If you enabled TLS 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 in are shown in red. You can show/hide the stack trace of the exception by clicking on the message.

Adding Database Drivers

▲

To register additional JDBC drivers (MySQL, PostgreSQL, HSQLDB,...), add the jar file names to the environment variables `H2DRIVERS` or `CLASSPATH`. Example (Windows): to add the HSQLDB JDBC driver `C:\Programs\hsqldb\lib\hsqldb.jar`, set the environment variable `H2DRIVERS` to `C:\Programs\hsqldb\lib\hsqldb.jar`.

Multiple drivers can be set; entries need to be separated by `;` (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. Type a SQL command in th query panel and click [Run]. The result appears just below the command.

Inserting Table Names or Column Names

To insert table and column names into the script, click on the item in the tree. If you click on a table while the query is empty, then `SELECT * FROM ...` is added. While typing a query, the table that was used is expandec in the tree. For example if you type `SELECT * FROM TEST T WHERE T.` then the table TEST is expanded.

Disconnecting and Stopping the Application

To log out of the database, click [Disconnect] in the toolbar panel. However, the server is still running and read 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, so they work with any database. Built-in commands 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 of the line are set to null. The complete list of metadata commands is: `@attributes, @best_row_identifier, @catalogs, @columns, @column_privileges @cross_references, @exported_keys, @imported_keys, @index_info, @primary_keys, @procedures, @procedure_columns, @pseudo_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();
@generated(1) insert into test() values();
@generated(ID, "TIMESTAMP") insert into test() values();
	Show the result of `Statement.getGeneratedKeys()`. Names or one based indexes of required columns can be optionally specified.

@history;	List 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 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 statemer 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).

@prof_start;
@sleep 10;
@prof_stop;
	Sleep for a number of seconds. Used to profile a long running quer or operation that is running in another session (but in the same process).

@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 you user home directory. For Windows installations, the user home directory is usually C:\Documents and Settings\[username or C:\Users\[username] . The configuration file contains the settings of the application and is automatically created when the H2 Console is first started. Supported settings are:

- webAllowOthers : allow other computers to connect.
- webPort : the port of the H2 Console
- webSSL : use encrypted TLS (HTTPS) connections.
- webAdminPassword : hash of password to access preferences and tools of H2 Console, use org.h2.server.web.WebServer.encodeAdminPassword(String) to generate a hash for your password. Always use long complex passwords, especially when access from other hosts is enabled.

In addition to those settings, the properties of the last recently used connection are listed in the form <number>=<name>|<driver>|<url>|<user> using the escape character \ . Example: 1=Generic H2 (Embedded)|org.h2.Driver|jdbc\:h2\:~/test|sa

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

This code 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 embedded URL does not yet exist, a new (empty) database is creat automatically. The user that created the database automatically becomes the administrator of this database.

Auto-creation of databases can be disabled, see Opening a Database Only if it Already Exists.

H2 Console does not allow creation of databases unless a browser window is opened by Console during its startup or from its icon in the system tray and remote access is not enabled. A context menu of the tray icon ca also be used to create a new database.

You can also create a new local database from a command line with a Shell tool:

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

Welcome to H2 Shell
Exit with Ctrl+C
[Enter]   jdbc:h2:mem:2
URL       jdbc:h2:./path/to/database
[Enter]   org.h2.Driver
Driver
[Enter]   sa
User      your_username
Password  (hidden)
Type the same password again to confirm database creation.
Password  (hidden)
Connected

sql> quit
Connection closed
```

By default remote creation of databases from a TCP connection or a web interface is not allowed. It's not recommended to enable remote creation of databases due to security reasons. User who creates a new database becomes its administrator and therefore gets the same access to your JVM as H2 has and the same access to your operating system as Java and your system account allows. It's recommended to create all databases locally using an embedded URL, local H2 Console, or the Shell tool.

If you really need to allow remote database creation, you can pass `-ifNotExists` parameter to TCP, PG, or We servers (but not to the Console tool). Its combination with `-tcpAllowOthers`, `-pgAllowOthers`, or `-webAllowOthers` effectively creates a remote security hole in your system, if you use it, always guard your por with a firewall or some other solution and use such combination of settings only in trusted networks.

H2 Servlet also supports such option. When you use it always protect the servlet with security constraints, see Using the H2 Console Servlet for example; don't forget to uncomment and adjust security configuration for you needs.

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). Please note that only the web server supports brows connections. The servers can be started in different ways, one is using the `Server` tool. Starting the server doesn't open a database - databases are opened as soon as a client connects.

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. Please note that you can't connection with a web browser to this URL. You can only connect using a H2 client (over JDBC).

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 -tcpPassword password
```

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

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

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

When using Hibernate, try to use the `H2Dialect` if possible. When using the `H2Dialect`, compatibility modes such as `MODE=MySQL` are not supported. When using such a compatibility mode, use the Hibernate dialect for the corresponding database instead of the `H2Dialect`; but please note H2 does not support all features of databases.

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, us 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 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 .

To use H2 within Glassfish, copy the h2*.jar to the directory glassfish/glassfish/lib .

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 Apache ActiveMQ

When using H2 as the backend database for Apache ActiveMQ, please use the TransactDatabaseLocker instead of the default locking mechanism. Otherwise the database file will grow without bounds. The problem i that the default locking mechanism uses an uncommitted UPDATE transaction, which keeps the transaction l from shrinking (causes the database file to grow). Instead of using an UPDATE statement, the TransactDatabaseLocker uses SELECT ... FOR UPDATE which is not problematic. To use it, change the ApacheMQ configuration element <jdbcPersistenceAdapter> element, property databaseLocker="org.apache.activemq.store.jdbc.adapter.TransactDatabaseLocker" . However, using the MVCC mode will again result in the same problem. Therefore, please do not use the MVCC mode in this case Another (more dangerous) solution is to set useDatabaseLock to false.

Using H2 within NetBeans

There is a known issue when using the Netbeans SQL Execution Window: before executing a query, another query in the form SELECT COUNT(*) FROM <query> is run. This is a problem for queries that modify state, such as SELECT NEXT VALUE FOR SEQ . In this case, two sequence values are allocated instead of just or

Using H2 with jOOQ

jOOQ adds a thin layer on top of JDBC, allowing for type-safe SQL construction, including advanced SQL, stored procedures and advanced data types. jOOQ takes your database schema as a base for code generatio If this is your example schema:

```
CREATE TABLE USER (ID INT, NAME VARCHAR(50));
```

then run the jOOQ code generator on the command line using this command:

```
java -cp jooq.jar;jooq-meta.jar;jooq-codegen.jar;h2-1.4.199.jar;.
org.jooq.util.GenerationTool /codegen.xml
```

...where codegen.xml is on the classpath and contains this information

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration xmlns="http://www.jooq.org/xsd/jooq-codegen-3.11.0.xsd">
  <jdbc>
    <driver>org.h2.Driver</driver>
    <url>jdbc:h2:~/test</url>
    <user>sa</user>
    <password></password>
  </jdbc>
  <generator>
    <database>
      <includes>.*</includes>
      <excludes></excludes>
      <inputSchema>PUBLIC</inputSchema>
    </database>
    <target>
      <packageName>org.jooq.h2.generated</packageName>
      <directory>./src</directory>
    </target>
    </generator>
</configuration>
```

Using the generated source, you can query the database as follows:

```
DSLContext dsl = DSL.using(connection);
Result<UserRecord> result =
dsl.selectFrom(USER)
    .where(NAME.like("Johnny%"))
    .orderBy(ID)
    .fetch();
```

See more details on jOOQ Homepage and in the jOOQ Tutorial

Using Databases in Web Applications

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

Embedded Mode

The (currently) simplest solution is to use the database in the embedded mode, that means open a connection
your application when it starts (a good solution is using a Servlet Listener, see below), or when a session start
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. I
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, anc
close it when the web application stops. If using multiple applications, only one (any) of them needs to do that.
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>
```

If your servlet container is already Servlet 5-compatible, use the following snippet instead:

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

For details on how to access the database, see the file `DbStarter.java`. By default this tool opens an embedde
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 i 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>
<!--
<security-role>
    <role-name>admin</role-name>
</security-role>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>H2 Console</web-resource-name>
        <url-pattern>/console/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
-->
```

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

If your application is already Servlet 5-compatible, use the servlet class org.h2.server.web.JakartaWebServlet instead.

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.

Reading a CSV File from Within a Database

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

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

Please note for performance reason, CSVREAD should not be used inside a join. Instead, import the data firs (possibly into a temporary table), create the required indexes if necessary, and then query this table.

Importing Data from a CSV File

A fast way to load or import data (sometimes called 'bulk load') from a CSV file is to combine table creation wit import. Optionally, the column names and data types can be set when creating the table. Another option is to u INSERT INTO ... SELECT .

```
CREATE TABLE TEST AS SELECT * FROM CSVREAD('test.csv');
CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255))
  AS SELECT * FROM CSVREAD('test.csv');
```

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');
```

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");
        new Csv().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 = new Csv().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 usi the new engine.

Backup using the Script Tool

The recommended way to backup a database is to create a compressed SQL script file. This will result in a small, human readable, and database version independent backup. Creating the script will also verify the checksums of the database file. The Script tool is ran as follows:

```
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 t 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 usin the server mode, the references script files need to be available on the server side.

If the script was generated by H2 1.4.200 or an older version, add VARIABLE_BINARY option to import it into more recent version.

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

Online Backup

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

The resulting backup is transactionally consistent, meaning the consistency and atomicity rules apply.

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

The Backup tool (org.h2.tools.Backup) can not be used to create a 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 fi systems support creating snapshots. With other file systems, it can't be guaranteed that the data is copied in th 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 t 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 als 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 or 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 connection from the built-in connection pool than to get one using DriverManager.getConnection() .The build-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 f
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 (comma separated) is option
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 drop an index on a table:

```
CALL FT_DROP_INDEX('PUBLIC', 'TEST');
```

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 Apache Lucene Fulltext Search

To use the Apache Lucene full text search, you need the Lucene library in the classpath. Apache Lucene 8.5.2
binary compatible version is required. How to do that depends on the application; if you use the H2 Console, y
can add the Lucene jar file to the environment variables `H2DRIVERS` or `CLASSPATH` . To initialize the Luce
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
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 (comma separated) is optiona
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 drop an index on a table (be warned that this will re-index all of the full-text indices for the entire database):

```
CALL FTL_DROP_INDEX('PUBLIC', 'TEST');
```

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);
```

The Lucene fulltext search supports searching in specific column only. Column names must be uppercase (except if the original columns are double quoted). For column names starting with an underscore (_), another underscore needs to be added. Example:

```
CREATE ALIAS IF NOT EXISTS FTL_INIT FOR "org.h2.fulltext.FullTextLucene.init";
CALL FTL_INIT();
DROP TABLE IF EXISTS TEST;
CREATE TABLE TEST(ID INT PRIMARY KEY, FIRST_NAME VARCHAR, LAST_NAME VARCHAR);
CALL FTL_CREATE_INDEX('PUBLIC', 'TEST', NULL);
INSERT INTO TEST VALUES(1, 'John', 'Wayne');
INSERT INTO TEST VALUES(2, 'Elton', 'John');
SELECT * FROM FTL_SEARCH_DATA('John', 0, 0);
SELECT * FROM FTL_SEARCH_DATA('LAST_NAME:John', 0, 0);
CALL FTL_DROP_ALL();
```

User-Defined Variables

This database supports user-defined variables. Variables start with `@` and can be used wherever expression
or parameters are allowed. Variables are not persisted and session scoped, that means only visible from withi
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, COALESCE(@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. Rolling back a transaction does not affect the value of a user-defined variable.

Date and Time

Date, time and timestamp values support standard literals:

```
VALUES (
    DATE '2008-01-01',
    TIME '12:00:00',
    TIME WITH TIME ZONE '12:00:00+01:00',
    TIMESTAMP '2008-01-01 12:00:00',
    TIMESTAMP WITH TIME ZONE '2008-01-01 12:00:00+01:00'
);
```

ISO 8601-style datetime formats with T instead of space between date and time parts are also supported.

TIME and TIMESTAMP values are preserved without time zone information as local time. That means if you store the value '2000-01-01 12:00:00' in one time zone, then change time zone of the session you will also get '2000-01-01 12:00:00', the value will not be adjusted to the new time zone, therefore its absolute value in UTC may be different.

TIME WITH TIME ZONE and TIMESTAMP WITH TIME ZONE values preserve the specified time zone offset and if you store the value '2008-01-01 12:00:00+01:00' it also remains the same even if you change time zone the session, and because it has a time zone offset its absolute value in UTC will be the same. TIMESTAMP WITH TIME ZONE values may be also specified with time zone name like '2008-01-01 12:00:00 Europe/Berlin It that case this name will be converted into time zone offset. Names of time zones are not stored.

Using Spring

Using the TCP Server

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,-tcpPort,8043" />
</bean>
```

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

OSGi

The standard H2 jar can be dropped in as a bundle in an OSGi container. H2 implements the JDBC Service defined in OSGi Service Platform Release 4 Version 4.2 Enterprise Specification. The H2 Data Source Factory service is registered with the following properties: `OSGI_JDBC_DRIVER_CLASS=org.h2.Driver` and `OSGI_JDBC_DRIVER_NAME=H2 JDBC Driver`. The `OSGI_JDBC_DRIVER_VERSION` property reflects th version of the driver as is.

The following standard configuration properties are supported: `JDBC_USER, JDBC_PASSWORD, JDBC_DESCRIPTION, JDBC_DATASOURCE_NAME, JDBC_NETWORK_PROTOCOL, JDBC_URL, JDBC_SERVER_NAME, JDBC_PORT_NUMBER`. Any other standard property will be rejected. Non-standard properties will be passed on to H2 in the connection URL.

Java Management Extension (JMX)

Management over JMX is supported, but not enabled by default. To enable JMX, append `;JMX=TRUE` to the database URL when opening the database. Various tools support JMX, one such tool is the `jconsole`. When opening the `jconsole`, connect to the process where the database is open (when using the server mode, you need to connect to the server process). Then go to the `MBeans` section. Under `org.h2` you will find one entry per database. The object name of the entry is the database short name, plus the path (each colon is replaced with an underscore character).

The following attributes and operations are supported:

- `CacheSize`: the cache size currently in use in KB.
- `CacheSizeMax` (read/write): the maximum cache size in KB.

- Exclusive : whether this database is open in exclusive mode or not.
- FileReadCount : the number of file read operations since the database was opened.
- FileSize : the file size in KB.
- FileWriteCount : the number of file write operations since the database was opened.
- FileWriteCountTotal : the number of file write operations since the database was created.
- LogMode (read/write): the current transaction log mode. See SET LOG for details.
- Mode : the compatibility mode (REGULAR if no compatibility mode is used).
- MultiThreaded : true if multi-threaded is enabled.
- Mvcc : true if MVCC is enabled.
- ReadOnly : true if the database is read-only.
- TraceLevel (read/write): the file trace level.
- Version : the database version in use.
- listSettings : list the database settings.
- listSessions : list the open sessions, including currently executing statement (if any) and locked tables (if any).

To enable JMX, you may need to set the system properties com.sun.management.jmxremote and com.sun.management.jmxremote.port as required by the JVM.