SQL Workbench/J User Manual

Table of Contents

1. General Information

1.1. Software license

1.2. Program version

This document describes build 100.1 of SQL Workbench/J

1.3. Feedback and support

Feedback regarding this program is more then welcome. Please report any problems you find, or send your ideas to improve the usability to: `<support@sql-workbench.net>`

SQL Workbench/J can be downloaded from http://www.sql-workbench.net

If you want to contact other users of SQL Workbench/J you can do this using an online forum at Google Groups: http://groups.google.com/group/sql-workbench

1.4. Credits and thanks

Thanks to Christian (and his team) for his thorough testing, his patience and his continous ideas to improve this tool. His input has influenced and driven a lot of features and has helped reduce the number of bugs drastically!

2. Installing and starting SQL Workbench/J

Starting with build 99 the jar file has been renamed to `sqlworkbench.jar`. Please delete the old file `Workbench.jar`. Windows user should also delete the old `JWorkbench.exe` and use the most recent `SQLWorkbench.exe`

To run SQL Workbench/J a Java runtime environment version 1.5 (or above) is required. You can either use a JRE ("Runtime") or a JDK ("Development Kit") to run SQL Workbench/J. Once you have downloaded the application, unzip the SQL Workbench/J archive into a directory of your choice. Apart from that, no special installation procedure is needed. When upgrading to a newer version of SQL Workbench/J simply overwrite the old `sqlworkbench.jar` and the exe launcher and shell scripts that start the application.

2.1. Starting the program

sqlworkbench.jar is a self executing JAR file. This means, that if your JDK is installed properly, a double click (on the Windows® platform) on sqlworkbench.jar will execute the application. To run the application manually use the command:

```
java -jar sqlworkbench.jar
```

If you want to create a shortcut on your desktop use the above line. On Windows® systems - if you don't use the native launcher- it is recommended to use javaw instead of java, so that the console window is not displayed.

```
javaw -jar sqlworkbench.jar
```

2.1.1. Specifying the location of configuration files

If no configuration directory has been specified on the commandline, SQL Workbench/J will check the current directory for a `workbench.settings` file. If no file is present in the current directory, SQL Workbench/J will check the directory where `sqlworkbench.jar` is located. If no `workbench.settings` is found in that directory, SQL Workbench/J creates a sub-directory called `.sqlworkbench` in the user's home folder (This is `$HOME` on a Linux or Unix based system, or `%HOMEPATH%` on a Windows system.

Before Build 98 the default configuration directory was the program's directory and not a directory in the user's home directory.

The configuration files are:

- General configuration settings (`workbench.settings`)
- Connection profiles (`WbProfiles.xml`)
- JDBC Driver definitions (`WbDrivers.xml`)
- Customized shortcut definitions (`WbShortcuts.xml`)
- Macro definitions (`WbMacros.xml`)
- Log file (`workbench.log`)
- Workspace files (`*.wksp`)

If you want to use a different file for the connection profile than WbProfiles.xml then you can specify the location of the profiles with the `-profilestorage` parameter on the commandline. Thus you can create different shortcuts on your desktop pointing to different sets of profiles. The different shortcuts can still use the same main configuration file.

If you want to control the location where stores the configuration files, you have to start the application with the parameter -configDir to specify an alternate directory:

```
java -jar sqlworkbench.jar -configDir=/export/configs/SQLWorkbench
```

or if you are using the Windows® launcher:

```
SQLWorkbench -configDir=c:\ConfigData\SQLWorkbench
```

The placeholder `${user.home}` will be replaced with the current user's home directory (as returned by the Operating System), e.g.:

```
java -jar sqlworkbench.jar -configDir=${user.home}/.sqlworkbench
```

If the specified directory does not exist, it will be created.

To copy an installation to a different computer, simply copy all the above files to the other computer (the log file does not need to be copied). When a profile is connected to a workspace, the filename of the workspace file is usually stored with a placeholder for the configuration directory (%configDir%) so that the profiles don't need to be adjusted.

You will need to edit the driver definitions (stored in `WbDrivers.xml`) as the full path to the driver's jar file(s) is stored in the file (unless you define the location of the drivers using the libdir variable.

2.2. Displaying help

You have two options to display help for SQL Workbench/J. Either the built-in help file, which is accessible through Help » Contents or the PDF manual by selecting Help » Manual. In order to be able to display the PDF manual, you need to define the path to the executable for the PDF reader in the General options and the file `SQLWorkbench-Manual.pdf` must be available in the directory where `sqlworkbench.jar` is located.

2.3. Increasing the memory available to the application

SQL Workbench/J is a Java application and thus runs inside a so-called virtual machine (JVM). The virtual machine limits the memory independently from the installed memory that is available to the operating system.

SQL Workbench/J reads the data that is returned by a `SELECT` statement into the main memory. When retrieving large result sets, you might get an error message, indicating that not enough memory is available. In this case you need to increase the memory that the JVM requests from the operating system (or change your statement to return fewer rows).

When you use the Windows® Launcher to start SQL Workbench/J you need to pass the parameter -J-Xmx512m to the executable:

```
SQLWorkbench.exe -J-Xmx512m
```

This example will increase the maximum memory to 512MB. The recommended way is to create Windows® shortcut (e.g. on the desktop) and add the above parameter to the shortcut definition. The launcher sets the available heap size for SQL Workbench/J to 256MB.

If you are running SQL Workbench/J on a non-Windows® operating system or do not want to use the launcher, then you need to pass this parameter directly to the JVM

```
java -Xmx512m -jar sqlworkbench.jar
```

The default heap size for your Java environment depends on your operating system and your JDK implementation. Most JDKs use a default of 96MB as the default maximum heap size.

The -Xmx parameter increases the **maximum** memory to the given value. This does not mean that the application will use that much memory

2.4. Display Problems, Crashes, Bluescreens when running under Windows®

If you experience problems when running SQL Workbench/J (or other Java/Swing based applications) on the Windows® platform, this might be due to problems with the graphics driver and/or the DirectDraw installation. If upgrading the graphics driver or the DirectDraw/DirectX version is not an option (or does not solve the problem), try to run SQL Workbench with the direct draw feature turned off:

```
java -Dsun.java2d.noddraw=true -jar sqlworkbench.jar
```

When using the exe launcher, you can use the following syntax:

```
SQLWorkbench -noddraw
```

If you run SQL Workbench/J through a program that enables remote access to a Windows® workstations (PC-Duo, VNC, NetMeeting, etc), you may need to disable the use of DirectDraw for Java as well.

2.5. High CPU usage when executing statements

If you experience a high CPU usage when running a SQL statement, this might be caused by a combination of the graphics driver, the JDK and the Windows® version you are using. This is usually caused by the animated icon which indicates a running statement (the yellow smiley). This animation can be turned off in Tools » Options See Enable animated icons for details. A different icon (not animated) will be used if that option is disabled.

2.6. Using the Windows launcher

On the Windows® platform the supplied SQLWorkbench.exe can be used to start the program when using a Sun JDK. The native launcher searches for an installed JDK (querying the registry) and then starts the SQL Workbench/J. The file sqlworkbench.jar has to be located in the same directory as the SQLWorkbench.exe, otherwise it doesn't work.

The launcher only works with JDK's from Sun, as it directly calls the JDK dll's to start the virtual machine. If you are using a different JDK, you cannot use the launcher to start SQL Workbench/J (unless it uses the same directory layout and filenames as the Sun JDK).

By default the launcher increases the maximum JVM heap size to 128 MB. If you need more heap memory, you need to pass the appropriate JVM parameter to the launcher.

2.6.1. How the Windows launcher searches for a Sun JDK

First the launcher checks for a system variable `WORKBENCH_JDK`. If that is defined, the JDK specified by that directory is used. If `WORKBENCH_JDK` is not found, `JAVA_HOME` is used. If `JAVA_HOME` is not defined, then the launcher checks if a sub-directory JRE exists in the folder where SQLWorkbench.exe is located. If that sub-directory exists, it is assumed that it contains a valid JRE. If the sub-directory does not exist, or if it is not a JRE installation, then the registry key `HKLM\Software\JavaSoft\Java Runtime Environment` is queried. If that is not defined, `HKLM\Software\JavaSoft\Java Development Kit` is queried.

In the registry key, a subkey for the version 1.5 is retrieved, and the directory specified by that key is used as the base JDK directory.

If your JDK/JRE installation cannot be found by the launcher, but you do have a JDK available, you can specify the location of the JDK with the -jdk parameter

The launcher assumes the layout of the Sun jdk in the specified directory. If you specify `c:\jdk` as the JDK directory, the launcher looks for the file `c:\jdk\bin\client\jvm.dll` (the specified directory would actually be a JRE then). If that is not found, it looks for `c:\jdk\jre\bin\client\jvm.dll` (that would be a "true" JRE installation). If the `-sever` parameter is specified, it will look for a sub-directory `server` instead of `client`. If your non-Sun JDK/JRE follows the same directory layout and filename conventions, you can use the launcher for that JDK as well.

2.6.2. Parameters for the Windows launcher

To distinguish parameters for the launcher and parameters to the JVM, JVM parameter need to be prefixed with -J. If you want to pass the parameter `-Xmx256m` to the JVM, pass the parameter `-J-Xmx256m` to the launcher. To define a system property you need to pass the parameter `-J-Dproperty.name=property_value`.

The following parameters are recognized:

Parameter	Description
-jdk	Specify the installation directory of the JDK e.g.: -jdk=c:\jdk1.5 When this parameter is specified the launcher will not look for a JDK installation as described here
-J	Pass a parameter to the JVM e.g: -J-Xms128m
-server	Select the server JVM (instead of the default client JVM). This switch only works with the Sun JVM.
-client	Select the client JVM. This switch only works with the Sun JVM.
-jvmtype	Select the JVM type to be loaded. For the Sun JVM this may be either `client` or `server` (equivalent to the -server or -client switches). If the JDK identified with the -jdk switch points to BEA's JRockit JVM, this should be `jrockit` (i.e. -jvmtype=jrockit). Basically the value of this switch is used to locate the `jvm.dll` in the base directory specified with the -jdk switch.
-noddraw	Disable the use of DirectDraw routines for the JVM. Use this parameter when you are running SQL Workbench/J through PC-Duo or a similar program, or if you are experiencing crashes when starting SQL Workbench/J
-debug	Write debug information to the file workbench.dbg to identify problems when using the launcher
-help	Display a message with the list of parameters

All other parameters are passed unchanged to the program. See command line parameters for details.

The following call to the launcher:

```
SQLWorkbench -noddraw -configDir=c:\MyConf
```

is the same as directly starting sqlworkbench.jar with these parameters:

```
java -Dsun.java2d.noddraw=true -jar sqlworkbench.jar -configDir=c:\MyConf
```

2.6.3. Windows Vista

With Windows Vista, Microsoft changed the way needed DLLs are searched when an executable is loaded. This affects the SQL Workbench/J launcher due to the (new?) Microsoft C runtime distribution model. If you want to run SQL Workbench/J under Windows Vista, please copy the file `msvcr71.dll` into the directory where `SQLWorkbench.exe` is located.

This file can be found at `%SystemRoot%\System32\msvcr71.dll` (usually this is `c:\Windows\System32\msvcr71.dll`)

Thanks to Jon for this tip.

2.7. Command line parameters

Command line parameters are **not** case sensitive. The parameters `-PROFILE` or `-profile` are identical. The usage of the command line parameters is identical between the launcher or starting SQL Workbench/J using the `java` command itself.

When quoting parameters on the commandline (especially in a Windows environment) you have to use single quotes, as the double quotes won't be passed to the application.

2.7.1. Specify the directory for configuration settings

The parameter `-configDir` specifies the directory where SQL Workbench/J will store all its settings. If this parameter is not supplied, the directory where the default location is used. The placeholder `${user.home}` will be replaced with the current user's home directory (as returned by the Operating System). If the specified directory does not exist, it will be created.

```
java -jar sqlworkbench.jar -configDir=${user.home}/wbconfig
SQLWorkbench -configDir='c:\Configurations\SQLWorkbench'
```

Note that even on the Windows platform you can use a forward slash to separate directory names in the parameter.

2.7.2. Specify a base directory for JDBC driver libraries

The `-libdir` parameter defines the base directory for your JDBC drivers. The value of this parameter can be referenced when defining a driver library using the placholder `%LibDir%` The value for this parameter can also be set in the file `workbench.settings`.

2.7.3. Specify the file containing connection profiles

SQL Workbench/J stores the connection profiles in a file called `WbProfiles.xml`. If you want to use a different filename, or use different set of profiles for different purposes you can define the file where the profiles are stored with the `-profilestorage` parameter.

If the value of the parameter does not contain a path, the file will be expected (and stored) in the configuration directory.

2.7.4. Defining variables

With the `-vardef` parameter a definition file for internal variables can be specified. Each variable has to be listed on a single line in the format `variable=value`. Lines starting with a # character are ignored (comments). the file can contain unicode sequences (e.g. `\u00fc`. Values spanning multiple lines are not supported. When reading a file during startup the default encoding is used. If you need to read the file in a specific encoding please use the `WbVarDef` command with the `-file` and `-encoding` parameter.

```
#Define some values
var_id=42
person_name=Dent
another_variable=24
```

You can also define a list of variables with this parameter. In this case, the first character after the = sign, has to be # (hash sign) to flag the value as a variable list:

```
java -jar sqlworkbench.jar -vardef=#var_id=42,person_name=Dent
```

2.7.5. Prevent updating the .settings file

If the `-nosettings` parameter is specified, SQL Workbench/J will not write its settings to the file `workbench.settings` when it's beeing closed. Note that in batch mode, this file is never written.

Note that if this parameter is supplied, the workspace will not be saved automatically as well!

2.7.6. Connect using a pre-defined connection profile

You can specify the name of an already created connection profile on the commandline with the `-profile=<profile name>` parameter. The name has to be passed exactly like it appears in the profile dialog (case sensitiv!). If the name contains spaces or dashes, it has to be enclosed in quotations marks. If you have more than one profile with the same name but in different profile groups, you have to specify the desired profile group using the `-profilegroup` parameter, otherwise the first profile matching the passed name will be selected.

Example (on one line):

```
java -jar sqlworkbench.jar
     -profile='Oracle - Test'
     -script='test.sql'
```

In this case the file `WbProfiles.xml` must be in the current (working) directory of the application. If this is not the case, please specify the location of the profile using either the `-profilestorage` or `-configDir` parameter.

If you have two profiles with the names `"Oracle - Test"` you will need to specify the profile group as well (in one line):

```
java -jar sqlworkbench.jar
     -profile='Oracle - Test'
     -profilegroup='Local'
     -script='test.sql'
```

2.7.7. Connect without a profile

You can also specify the full connection parameters on the commandline, if you don't want to create a profile only for executing a batch file. The advantage of this method is, that SQL Workbench/J does not need the files `WbProfiles.xml`, `WbDrivers.xml` to be able to connect to the database.

The connection can be specified with the following parameters:

Parameter	Description
-url	The JDBC connection URL
-username	Specify the username for the DBMS
-password	Specify the password for the user
-driver	Specify the full class name of the JDBC driver
-driverJar	Specify the full pathname to the .jar file containing the JDBC driver
-autocommit	Set the autocommit property for this connection. You can also control the autocommit mode from within your script by using the `SET AUTOCOMMIT` command.
-rollbackOnDisconnect	If this parameter is set to true, a `ROLLBACK` will be sent to the DBMS before the connection is closed. This setting is also available in the connection profile.
-separateConnection	If this parameter is set to true, and SQL Workbench/J is run in GUI mode, each SQL tab will use it's own connection to the database server. This setting is also available in the connection profile. The default is `true`.
-emptyStringIsNull	This parameter corresponds to the Empty String is NULL setting of the connection profile. This will only be used in GUI mode.
-removeComments	This parameter corresponds to the Remove comments setting of the connection profile.
-workspace	The workspace file to be loaded. If the file specification does not include a directory, the workspace will be loaded from the configuration directory. If this parameter is not specified, the default workspace (`Default.wksp`) will be loaded.
-altDelim	The alternate delimiter to be used for this connection. To define a single line delimiter append the characters ;nl to the parameter value: e.g. `-altDelimiter=GO;nl` to define a SQL Server like GO as the alternate delimiter. Note that when running in batchmode you can also override the default delimiter by specifying the `-delimiter` parameter.
-trimCharData	Turns on right-trimming of values retrieved from `CHAR` columns. See the description of the profile properties for details.
-readOnly	Puts the connection into read-only mode.

If a value for one of the parameters contains a dash or a space, you will need to quote the parameter value.

A disadvantage of this method is, that the password is displayed in plain text on the command line. If this is used in a batch file, the password will be stored in plain text in the batch file. If you don't want to expose the password, you can use a connection profile and enable password encryption for connection profiles.

3. JDBC Drivers

3.1. Configuring JDBC drivers

Before you can connect to a DBMS you have to configure the JDBC driver to be used. The driver configuration is available in the connection dialog or through File » Manage Drivers

The configuration of a specific driver requires at least two properties:

- the driver's class name
- the library ("JAR file") where to find the driver class

The name of the library has to contain the full path to the driver's jar file, so that SQL Workbench/J can find it. If a driver requires more than one library, select all necessary libraries in the file open dialog, or enter all the filenames separated by a semicolon (or a colon on Unix style operating systems). This is also true for drivers that require a license file that is contained in a jar file. In this case you have to include the license jar in the list of files. Basically this list defines the classpath for the classloader that is used to load and instantiate the driver.

If the driver accesses files through its classpath definition that are not contained in a jar library, you have to include that directory as part of the library definition (e.g: `"c:\etc\TheDriver\jdbcDriver.jar;c:\etc\TheDriver"`). The file selection dialog will not let you select a directory, so you have to add it manually to the library definition.

SQL Workbench/J is **not** using the system CLASSPATH definition (i.e. environment variable) to load the driver classes. Changing the CLASSPATH environment variable to include your driver's library will not work.

You do not need to specify a library for the JDBC-ODBC bridge, as the necessary drivers are already part of the Java runtime.

You can assign a sample URL to each driver, which will be put into the URL property of the profile, when the driver class is selected.

SQL Workbench/J comes with some sample URLs pre-configured. Some of these sample URLs use brackets to indicate a parameters that need to be replaced with the actual value for your connection: (servername)

In this case the entire sequence including the brackets need to be replaced with the actual value.

3.2. Specifying a library directory

When defining the location of the driver's .jar file, you can use the placeholder `%LibDir%` instead of the using the directory's name directly. This way your `WbDrivers.xml` is portable across installations. To specify the library directory, either set it in the `workbench.settings` file, or specify the directory using the -libdir switch when starting the application.

3.3. Popular JDBC drivers

Here is an overview of common JDBC drivers, and the classname that needs to be used. SQL Workbench/J contains predefined JDBC drivers that also contain sample URLs for connecting to the database. Note that most drivers accept additional configuration parameters either in the URL or through the extended properties. Please consult the manual of your driver for more detailed information on these additional parameters.

DBMS	Driver class	Library name
PostgreSQL	org.postgresql.Driver	postgresql-8.3-603.jdbc3.jar (exact name depends on PostgreSQL version) http://jdbc.postgresql.org
Firebird SQL	org.firebirdsql.jdbc.FBDriver	firebirdsql-full.jar http://www.firebirdsql.org/
Apache Derby	org.apache.derby.jdbc.EmbeddedDriver	derby.jar http://db.apache.org/derby/

DBMS	Driver class	Library name
h2 Database Engine	org.h2.Driver	h2.jar http://www.h2database.com
HSQLDB	org.hsqldb.jdbcDriver	hsqldb.jar http://hsqldb.sourceforge.net
MySQL	com.mysql.jdbc.Driver	mysql-connector-java-5.1.5-bin.jar (exact name depends on version) http://www.mysql.com
Oracle	oracle.jdbc.OracleDriver	ojdbc5.jar http://otn.oracle.com/software/tech/java/sqlj_jdbc/content.html
SQL Server 2000/2005	com.microsoft.sqlserver.jdbc.SQLServerDriver	sqljdbc.jar http://www.microsoft.com/sql/technologies/jdbc/default.mspx
SQL Server	net.sourceforge.jtds.jdbc.Driver	jtds.jar http://jtds.sourceforge.net
ODBC Bridge	sun.jdbc.odbc.JdbcOdbcDriver	Included in the JDK

4. Connecting to the database

4.1. Connection profiles

SQL Workbench/J uses the concept of database profiles to store connection information. A connection profile stores two different types of settings:

- JDBC related properties such as the JDBC driver class, the connection URL, the username etc.
- SQL Workbench/J related properties such as the profile name the associated workspace, etc.

After the program is started, you are prompted to choose a connection profile to connect to a database. The dialog will display a list of available profiles on the left side. When selecting a profile, its details (JDBC and SQL Workbench/J settings) are displayed on the right side of the window.

To create a new profile click on the `New Profile` button (). This will create a new profile with the name "New Profile". The new profile will be created in the currently active group. The other properties will be empty. To create a copy of the currently selected profile click on the `Copy Profile` button (). The copy will be created in the current group. If you want to place the copy into a different group, you can either choose to Copy & Paste a copy of the profile into that group, or move the copied profile, once it is created.

To delete an existing profile, select the profile in the list and click on the `Delete Profile` button ()

4.2. Managing profile groups

Profiles can be organized in groups, so you can group them by type (test, integration, production) or customer or database system. When you start SQL Workbench/J for the first time, no groups are created and the tree will only display the default group node. To add a new group click on the `Add profile group` () button. The new group will be appended at the end of the tree. If you create a new profile, it will be created in the currently selected group. If a profile is selected in the tree and not a group node, the new profile will be created in the group of the currently selected profile.

Empty groups are discarded (i.e. not saved) when you restart SQL Workbench/J

You can move profiles from one group to another but right clicking on the profile, then choose Cut. Then right-click on the target group and select Paste from the popup menu. If you want to put the profile into a new group that is not yet created, you can choose Paste to new folder. You will be prompted to enter the new group name.

If you choose Copy instead of Cut, a copy of the selected profile will be pasted into the target group. This is similar to copying the currently selected profile.

To rename a group, select the node in the tree, then press the F2 key. You can now edit the group name.

To delete a group, simply remove all profiles from that group. The group will then automatically be removed.

4.3. JDBC related profile settings

Property	Description
Driver	This is the classname for the JDBC driver. The exact name depends on the DBMS and driver combination. The documentation for your driver should contain this information. SQL Workbench/J has some drivers pre-configured. See JDBC drivers for details on how to configure your JDBC driver for SQL Workbench/J.
URL	The connection URL for your DBMS. This value is highly DBMS specific. The pre-configured drivers from SQL Workbench/J contain a sample URL. If the sample URL (which gets filled into the text field when you select a driver class) contains words in brackets, then these words (including the brackets) are placeholders for the actual values. You have to replace them (including the brackets) with the appropriate values for your DBMS connection.

Property	Description
Username	This is the name of the DBMS user account
Password	This is the password for your DBMS user account. You can choose not to store the password in the connection profile.
Fetch size	This setting controls the default fetch size for data retrieval. This parameter will directly be passed to the setFetchSize() method of the `Statement` object. For some combinations of JDBC driver and DBMS, setting this parameter to a rather large number can improve retrieval performance because it saves network traffic. The JDBC driver for PostgreSQL controls the caching of ResultSets through this parameter. As the results are cached by SQL Workbench/J anyway, it is suggested to set this parameter to a value greater then zero to disable the caching in the driver. Especially when exporting large results using WbExport or WbCopy it is recommended to turn off the caching in the driver (e.g. by setting the value for this property to 1).
Autocommit	This checkbox enables/disables the Autocommit property for the connection. If Autocommit is enabled, then each SQL statement is automatically committed on the DBMS. If this is disabled, any DML statement (`UPDATE, INSERT, DELETE, ...`) has to be committed in order to make the change permanent. Some DBMS require a commit for DDL statements (`CREATE TABLE, ...`) as well. Please refer to the documentation of your DBMS.

4.4. Extended properties for the JDBC driver

Most JDBC drivers support additional connection properties where you can fine tune the behaviour of the driver or enable special features that are not switched on by default.

If you need to pass an additional paramter to your driver you can do that with the Extended Properties button. After clicking that button, a dialog will appear with a table that has two columns. The first column is the name of the property, the second column the value that you want to pass to the driver.

To create a new property click on the new button. A new row will be inserted into the table, where you can define the property. To edit an existing property, simply doubleclick in the table cell that you want to edit. To delete an existing property click on the `Delete` button ().

4.5. SQL Workbench/J specific settings

4.5.1. Save password

If this option is enabled (i.e. checked) the password for the profile will also be stored in the profile file. If the global option Encrypt Passwords is selected, then the password will be stored encrypted, otherwise it will be stored in plain text!

If you choose not to store the password, you will be prompted for it each time you connect using the profile.

4.5.2. Separate connection per tab

If this option is enabled, then each tab in the main window will open a separate connection to the database server. This is useful, if the JDBC driver is not multi-threaded and does not allow to execute two statements concurrently on the same connection.

The connection for each tab will not be opened until the tab is actually selected.

Enabling this option has impact on transaction handling as well. If only one connection for all tabs (including the Database Explorer) is used, then a transaction that is started in one tab is "visible" to all other tabs (as they share the same connection). Changes done in one tab via `UPDATE` are seen in all other tabs (including the Database Explorer). If a separate connection is used for each tab, then each tab will have its own transaction context. Changes done in one tab will not be visible in other tabs until they are committed (depending on the isolation level of the database of course)

If you intend to execute several statements in parallel then it's strongly recommended to use one connection for each tab. Most JDBC drivers are not multi-threaded and thus cannot run more then on statement on the same connection. SQL Workbench/J does try to detect conflicting usages of a single connection as far as possible, but it is still possible to lock the GUI when running multiple statements on the same connection

When you disable the use of separate connections per tab, you can still create new a (physical) connection for the current tab later, by selecting File » New Connection. That menu item will be disabled if `Separate connection per tab` is disabled or you have already created a new connection for that tab.

4.5.3. Ignore DROP errors

If this option is enabled, any error reported by the database server when issuing a statement that begins with DROP, will be ignored. Only a warning will be printed into the message area. This is useful when executing SQL scripts to build up a schema, where a `DROP TABLE` is executed before each `CREATE TABLE`. If the table does not exist the error which the `DROP` statement will report, is not considered as an error and the script execution continues.

When running SQL Workbench/J in batchmode this option can be defined using a separate command line parameter. See Section 8, “Using SQL Workbench/J in batch files” for details.

4.5.4. Rollback before disconnect

Some DBMS require that all open transactions are closed before actually closing the connection to the server. If this option is enabled, SQL Workbench/J will send a `ROLLBACK` to the backend server before closing the connection. This is e.g. required for Cloudscape/Derby because executing a `SELECT` query already starts a transaction. If you see errors in your log file while disconnecting, you might need to enable this for your database as well.

4.5.5. Confirm updates

If this option is enabled, then SQL Workbench/J will ask you to confirm the execution of any SQL statement that is updating or changing the database in any way (e.g. UPDATE, DELETE, INSERT, DROP, CREATE, COMMIT, ...).

If you save changes from within the result list, you will be prompted even if Confirm result set updates is disabled.

This option cannot be selected together with the "Read only" option.

4.5.6. Read only

If this option is enabled, then SQL Workbench/J will never run any statements that might change the database. Changing of retrieved data is also disabled in this case. This option can be used to prevent accidental changes to important data (e.g. a production database)

SQL Workbench/J cannot detect all possible statements that may change the database. Especially when calling stored procedures SQL Workbench/J cannot know if they will change the database. But they might be needed to retrieve data, this cannot be disabled alltogether.

You can extend the list of keywords known to update the data in the `workbench.settings` file.

SQL Workbench/J will not guarantee that there is no way (accidentally or intended) to change data when this option is enabled. Please do not rely on this option when dealing with important data that must not be changed.

If you really need to guarantee that no data is changed, you have to do this with the security mechanism of your DBMS, e.g. by creating a read-only user.

This option cannot be selected together with the "Confirm updates" option.

4.5.7. Treat empty strings as NULL

When entering data into the result set (grid) this setting controls, how empty values (for character columns) should be treated. If this option is enabled, then a NULL value will be sent to the database for an empty (zero length) string. Empty values for other datatypes (dates, numbers etc) are always considered NULL. If this option is not enabled, then an empty string ('') will be sent to the database.

4.5.8. Include NULL columns in INSERT

This setting controls whether columns where the value from the result grid is null are included in INSERT statements. If this setting is enabled, then columns for new rows that have a null value are listed in the column list for the `INSERT` statement (with the corresponding `NULL` value passed in the `VALUES` list). If this property is un-checked, then those columns will not be listed in `INSERT` statements. This is useful if you have e.g. auto-increment columns that only work if the columns are not listed in the DML statement.

4.5.9. Remove comments

If this option is checked, then comments will be removed from the SQL statement before it is sent to the database. This covers single line comments using -- or multi-line comments using /* .. */ As an ANSI compliant SQL Lexer is used for this, this does **not** include comments for MySQL that are entered using the # character.

4.5.10. Remember DbExplorer Schema

If this option is enabled, the currently selected schema in the DbExplorer will be stored in the workspace associated with the current connection profile. If this option is not enabled, the DbExplorer tries to pre-select the current schema when it's opened.

4.5.11. Trim `CHAR` data

For columns defined with the CHAR datatype, some DBMS pad the values to the length defined in the column definition (e.g. a CHAR(80) column will always contain 80 characters). If this option is enabled, SQL Workbench/J will remove trailing spaces from the values retrieved from the database. When running SQL Workbench/J in batch mode, this flag can be enabled using the `-trimCharData` switch.

4.5.12. Connect scripts

You can define a SQL script that is executed immediately after a connection for this profile has been established, and a script that is executed before a connection is about to be closed. To define the scripts, click on the button Connect scripts. A new window will be opened that contains two editors. Enter the script that should be executed upon connecting into the upper editor, the script to be executed before disconnecting into the lower editor. You can put more than one statement into the scripts. The statements have to be separated by a semicolon.

The statements that are executed will be logged in the message panel of the SQL panel where the connection is created. You will not see the log when a connection for the DbExplorer is created.

Execution of the script will stop at the first statement that throws an error. The error message will also be logged to the message panel. If the connection is made for a DbExplorer panel, the errors will only be visible in the log file.

4.5.13. Keep alive script

Some DBMS are configured to disconnect an application that has been idle for some time. You can define an idle time and a SQL statement that is executed when the connection has been idle for the defined interval. This is also available when clicking on the Connect scripts.

The keep alive statement can not be a script, it can only be a **single** SQL statement (e.g. `SELECT version()` or `SELECT 42 FROM dual`). You may not enter the trailing semicolon.

The idle time is defined im milliseconds, but you can also enter the interval in seconds or minutes by appending the letter 's' (for seconds) or 'm' (for minutes) to the value. e.g.: `30000` (30 seconds), or `45s` (45 seconds), or `10m` (10 minutes).

When only one of the fields contains a value, You can disable the keep alive feature by deleting the entry for the interval but keeping the SQL statement. Thus you can quickly turn off the keep alive feature but keep the SQL statement for the next time.

4.5.14. Info Background

Once a connection has been established, information about the connection are display in the toolbar of the main window. You can select a color for the background of this display to e.g. indicate "sensitive" connections. To use the default background, click on the `Reset` () button. If no color is selected this is indicated with the text `(None)` next to the selection button. If you have selected a color, a preview of the color is displayed.

4.5.15. Alternate delimiter

If an alternate delimiter is defined, and the statement that is executed ends with the defined delimiter, this one will be used instead of the standard semicolon. The profile setting will overwrite the global setting for this connection. This way you can define the `GO` keyword for SQL Server connections, and use the forward slash in Oracle connections. The delimiter can be defined as a "single line delimiter", which means that it will only be recognized if put on a single line. Please refer to using the alternate delimiter for details on this property.

4.5.16. Workspace

For each connection profile, a workspace file can (and should) be assigned. When you create a new connection, you can either leave this field empty or supply a name for a new profile.

If the profile that you specify does not exist, you will be prompted if you want to create a new file, load a different workspace or want to ignore the missing file. If you choose to ignore, the association with the workspace file will be cleared and the default workspace will be loaded.

If you choose to leave the workspace file empty, or ignore the missing file, you can later save your workspace to a new file. When you do that, you will be prompted if you want to assign the new workspace to the current connection profile.

To save you current workspace choose Workspace » Save Workspace as to create a new workspace file.

When specifying the location of the workspace file, you can use the placeholder `%ConfigDir%` as part of the filename. The file will then be stored in the same directory as SQL Workbench/J's configuration files e.g.: `%ConfigDir%/oracle.wksp`

When you use the `%ConfigDir%` placeholder, you can move the profiles and workspaces to a different computer, without changing the location of the workspace files.

The placeholder will be put automatically into the filename when you select the location of the profile using the file dialog. The file dialog will be opened when you click the button with `...` to the right of the input field.

As the workspace stores several settings that are related to the connection (e.g. the selected schema in the DbExplorer) it is recommended to create one workspace for each connection profile.

4.6. Connect to Oracle with SYSDBA privilege

Connecting to Oracle with SYSDBA privilege can be done by supplying an additional property to the driver when connecting.

In the profile dialog, click on the Extended Properties button. Add a new property in the following window with the name `internal_logon` and the value `sysdba`. Now close the dialog by clicking on the OK button. This property will be passed on to the JDBC driver, which will enable the SYSDBA role when connecting to the server.

The profile itself has to use an Oracle user account that is allowed to connect as SYSDBA (e.g. SYS).

4.7. ODBC connections without a data source

On Microsoft Windows® you can use the ODBC bridge to connect to ODBC datasources. For some drivers you don't need to create an ODBC datasource in order to be able to use the ODBC driver. The following URLs can be used to connect to data files directly.

The class name of the driver is `sun.jdbc.odbc.JdbcOdbcDriver`

ODBC Connection	**URL to be used**
Excel	jdbc:odbc:DRIVER={Microsoft Excel Driver (*.xls)};DBQ=<filename>
Access	jdbc:odbc:DRIVER={Microsoft Access Driver (*.mdb)};DBQ=<filename>
dBase	jdbc:odbc:DRIVER={Microsoft dBase Driver (*.dbf)};DefaultDir=<directory where the .dbf files are located>

5. Editing SQL Statements

5.1. Editing files

You can load and save the editor's content into external files (e.g. for re-using) them in other SQL tools.

To load a file use File » Open... or right click on the tab's label and choose Open... from the popup menu.

The association between an editor tab and the external file will be saved in the workspace that is used for the current connection. When opening the workspace (e.g. by connecting using a profile that is linked to that workspace) the external file will be loaded as well.

If you want to run very large SQL scripts (e.g. over 100MB) it is recommended to execute them using WbInclude rather than loading them completely into the editor. `WbInclude` will not load the script into memory, thus you can even run scripts that would not fit into memory.

5.2. Command completion

The editor can show a popup window with a list of available tables (and views) or a list of available columns for a table. Which list is displayed depends on the position of the cursor inside the statement.

If the cursor is located in the column list of a `SELECT` statement and the `FROM` part already contains the necessary tables, the window will show the columns available in the table. Assuming you are editing the following statement (the `|` indicating the position of the caret):

```
SELECT p.|, p.firstname, a.zip, a.city
FROM person p, address a;
```

then pressing the **Ctrl-Space** key will show a list of columns available in the `PERSON` table (because the cursor is located after the `p.` alias). If you put the cursor after the `a.city` column and press the **Ctrl-Space** the popup window will list the two tables that are referenced in the `FROM` part of the statement. The behaviour when editing the `WHERE` part of an statement is similar.

When editing the list of tables in the `FROM` part of the statement, pressing the **Ctrl-Space** will pop up a list of available tables.

Usually a semicolon is used to separate statements in the editor. However for the auto completion of object names, this behaviour can be configured to also accept an empty line as a separator.

Parameters for SQL Workbench/J specific commands are also supported by the command completion. The parameters will only be shown, if you have already typed the leading dash, e.g. `WbImport -`. If you press the shortcut for the command completion while the cursor is located after the dash, a list of available options for the current comand is shown. Once the parameter has been added, you can display a list of possible values for the parameter if the cursor is located after the equals sign. for `WbImport -mode=` will display a list of allowed values for the `-mode` parameter. For parameters where table names can be supplied the usual table list will be shown.

5.3. Reformat SQL

When you analyze statements from e.g. a log file, they are not necessarily formatted in a way that can be easily read, let alone understood. The editor of the SQL Workbench/J can reformat SQL statements into a format that's easier to read and understand for a human beeing. This feature is often called pretty-printing. Suppose you have the following statement (pasted from a log file)

```
select user.* from user, user_profile, user_data
where user.user_id = user_profile.user_id and
user_profile.user_id = uprof.user_id and user_data.user_role = 1
and user_data.delete_flag = 'F' and not exists
(select 1 from data_detail where data_detail.id = user_data.id and
data_detail.flag = 'X' and data_detail.value > 42)
```

this will be reformatted to look like this:

```
SELECT user.*
FROM user,
     user_profile,
     user_data
WHERE user.user_id = user_profile.user_id
AND   user_profile.user_id = uprof.user_id
AND   user_data.user_role = 1
AND   user_data.delete_flag = 'F'
AND   NOT EXISTS (SELECT 1
                  FROM data_detail
                  WHERE data_detail.id = user_data.id
                  AND   data_detail.flag = 'x'
                  AND   data_detail.value > 42)
```

You can configure a threshold up to which sub-`SELECT`s will not be reformatted but put into one single line. The default for this threshold is 80 characters. Meaning that any subselect that is shorter than 80 characters will not be reformatted as the sub-`SELECT` in the above example. Please refer to Formatting options for details.

5.4. Create SQL value lists

Sometimes when you Copy & Paste lines of text from e.g. a spreadsheet, you might want to use those values as a condition for a SQL `IN` expression. Suppose you a have a list of ID's in your spreadsheet each in one row of the same column. If you copy and paste this into the editor, each ID will be put on a separate line. If you select the text, and then choose SQL » Create SQL List the selected text will be converted into a format that can be used as an expression for an `IN` condition:

```
Dent
Beeblebrox
Prefect
Trillian
Marvin
```

will be converted to:

```
('Dent',
 'Beeblebrox',
 'Trillian',
 'Prefect',
 'Marvin')
```

The function SQL » Create non-char SQL List is basically the same. The only difference is, that it assumes that each item in the list is a numeric value, and no single quotes are placed around the values.

The following list:

```
42
43
44
45
```

will be converted to:

```
(42, 43, 44, 45)
```

These two functions will only be available when text is selected which spans more then one line.

5.5. Programming related editor functions

The editor of the SQL Workbench/J offers two functions to aid in developing SQL statements which should be used inside your programming language (e.g. for SQL statements inside a Java program).

5.5.1. Copy Code Snippet

Suppose you have created the SQL statement that you wish to use inside your application to access your DBMS. The menu item SQL » Copy Code Snippet will create a piece of code that defines a String variable which contains the current SQL statement (or the currently selected statement if any text is selected).

If you have the following SQL statement in your editor:

```
SELECT p.name,
       p.firstname,
       a.street,
       a.zipcode,
       a.phone
FROM person p,
     address a
WHERE p.person_id = a.person_id;
```

When copying the code snippet, the following text will be placed into the clipboard

```
String sql="SELECT p.name, \n" +
"       p.firstname, \n" +
"       a.street, \n" +
"       a.zipcode, \n" +
"       a.phone \n" +
"FROM person p, \n" +
"     address a \n" +
"WHERE p.person_id = a.person_id; \n";
```

You can now paste this code into your application.

If you don't like the `\n` character in your code, you can disable the generation of the newline characters in you `workbench.settings` file. See Manual settings for details. You can also customize the prefix (`String sql =`) and the concatenation character that is used, in order to support the programming language that you use.

5.5.2. Clean Java code

When using the Copy Code Snippet feature during development, the SQL statement usually needs refinement after testing the Java class. You can Copy & Paste the generated Java code into SQL Workbench/J, then when you select the pasted text, and call SQL » Clean Java Code the selected text will be "cleaned" from the Java stuff around it. The algorithm behind that is as follows: remove everything up to the first " at the beginning of the line. Delete everything up to the first " searching backwards from the end of the line. Any trailing white-space including escaped characters such as \n will be removed as well. Lines starting with // will be converted to SQL single line comments starting with -- (keeping existing quotes!). The following code:

```
String sql="SELECT p.name, \n" +
"       p.firstname, \n" +
"       a.street, \n" +
//"       a.county, \n" +
"       a.zipcode, \n" +
"       a.phone \n" +
"FROM person p, \n" +
"     address a \n" +
"WHERE p.person_id = a.person_id; \n"
```

will be converted to:

```
SELECT p.name,
       p.firstname,
       a.street,
--"       a.county, " +
       a.zipcode,
```

```
       a.phone
FROM person p,
     address a
WHERE p.person_id = a.person_id;
```

5.5.3. Support for prepared statements

For better performance Java applications usually make use of prepared statements. The SQL for a prepared statement does not contain the actual values that should be used e.g. in the WHERE clause, but uses quotation marks instead. Let's assume the above example should be enhanced to retrieve the person information for a specific ID. The code could look like this:

```
String sql="SELECT p.name, \n" +
"       p.firstname, \n" +
"       a.street, \n" +
"       a.zipcode, \n" +
"       a.phone \n" +
"FROM person p, \n" +
"     address a \n" +
"WHERE p.person_id = a.person_id; \n" +
"  AND p.person_id = ?";
```

You can copy and clean the SQL statement but you will not be able to execute it, because there is no value available for the parameter denoted by the question mark. To run this kind of statements, you need to enable the prepared statement detection using SQL » Detect prepared statements

Once the prepared statement detection is enabled, SQL Workbench/J will examine each statement to check whether it is a prepared statement. This examination is delegated to the JDBC driver and does cause some overhead when running the statement. For performance reasons you should disable the detection, if you are not using prepared statements in the editor (especially when running large scripts).

If a prepared statement is detected, you will be prompted to enter a value for each defined parameter. The dialog will list all parameters of the statement together with their type as returned by the JDBC driver. Once you have entered a value for each parameter, clicking OK will execute the statement using those values. When you execute the SQL statement the next time, the old values will be presevered, and you can either use them again or modify them before running the statement.

Once you are satisfied with your SQL statement, you can copy the statement and paste the Java code into your program.

Prepared statements are supported for `SELECT`, `INSERT`, `UPDATE` and `DELETE` statements.

This feature requires that the getParameterCount() and getParameterType() methods of the `ParameterMetaData` class are implemented by the JDBC driver and return the correct information about the available parameters.

The following drivers have been found to support (at least partially) this feature:

- PostgreSQL, driver version 8.1-build 405
- H2 Database Engine, Version 1.0.73
- Apache Derby, Version 10.2
- Firebird SQL, Jaybird 2.0 driver
- HSQLDB, version 1.8.0

Drivers known to **not** support this feature:

- Oracle 10g driver (ojdbc14.jar)
- Microsoft SQL Server 2000 driver (mssqlserver.jar;msbase.jar;msutil.jar)
- Microsoft SQL Server 2005 driver (sqljdbc.jar)

6. Using SQL Workbench/J

6.1. Executing SQL statements

6.1.1. Statement history

When executing a statement the contents of the editor is put into an internal buffer together with the information about the text selection and the cursor position. Even when you select a part of the current text and execute that statement, the whole text is stored in the history buffer together with the selection information. When you select and execute different parts of the text and then move through the history you will see the selection change for each history entry.

The previous statement can be recalled by pressing **Alt-Left** or choosing SQL » Previous Statement statement from the menu. Once the previous statement(s) have been recalled the next statement can be shown using **Alt-Right** or choosing SQL » Next Statement from the menu. This is similar to browsing through the history of a web browser.

You can clear the statement history for the current tab, but selecting SQL » Clear history

When you clear the content of the editor (e.g. by selecting the whole text and then pressing the **Del** key) this will not clear the statement history. When you load the associated workspace the next time, the editor will automatically display the last statement from the history. You need to manually clear the statement history, if you want an empty editor the next time you load the workspace.

6.1.2. Control the statement to be executed

There are three different ways to execute a SQL command

Execute the selected text

When you press **Ctrl-E** or select SQL » Execute selected the currently selected text will be send to the DBMS for execution. If no text is selected the complete contents of the editor will be send to the database.

Execute current statement

When you press **Ctrl-Enter** or select SQL » Execute current the current statement will be executed. The "current" statement will be the text between the next delimiter before the current cursor position and the delimiter after the cursor position.

Example (| indicating the cursor position)

```
SELECT firstname, lastname FROM person;

DELETE FROM person| WHERE lastname = 'Dent';
COMMIT;
```

When pressing Ctrl-Enter the `DELETE` statement will be exectuted

You can configure SQL Workbench/J to automatically jump to the next statement, after executing the current statement. Simply select SQL » Jump to next statement The check mark next to the menu item indicates if this option is enabled. This option can also be changed through the Options dialog

Execute All

If you want to execute the complete text in the editor regardless of the current selection, use the Execute all command. Either by pressing **Ctrl-Shift-E** or selecting SQL » Execute All

As long as at least one statement is running the title of the main window will be prefixed with the » sign. Even if the main window is minimized you can still see if a statement is running by looking at the window title.

You can use variables in your SQL statements that are replaced when the statement is executed. Details on how to use variables can be found in the chapter Variable substitution.

6.1.3. Displaying results

When you run SQL statements that produce a result (such as a `SELECT` statement) these results will be displayed in the lower pane of the window, next to the message panel. For each result that is returned from the server, one tab (labelled "Result") will be created. If you select and execute three `SELECT` statements, the lower pane will show three result tabs and the message tab. If your statement(s) did not produce any result, only the messages tab will be displayed.

When you run a SQL statement, the current results will be cleared and replaced by the new results. You can turn this off by selecting SQL » Append new results. Every result that is retrieved while this option is turned on, will be added to the set of result tabs, until you de-select this option. This can also be toggled using the button on the toolbar. Additional result tabs can be closed using Data » Close result

You can also run stored procedures that return result sets. These result will be displayed in the same way. For DBMS's that support mulitple result sets from a single stored procedure (e.g. Microsoft SQL Server), one tab will be displayed for each result returned.

Due to a bug in Oracle's JDBC driver, you cannot retrieve columns with the `LONG` or `LONG RAW` data type if the `DBMS_OUTPUT` package is enabled. In order to be able to display these columns, the support for `DBMS_OUTPUT` has to be switched off.

6.1.4. Executing DML Statements

SQL statements can be entered in the upper part of the window. Please refer to Editing SQL Statements for details on the editing features of SQL Workbench/J.

The tabbed display allows you to keep more than one statement accessible (without needing to use the history functions). Each tab has its own result set and message panel. When you switch to a different statement tab, the result list in the lower part of the window will change to either the last result or the last message of that statement.

For JDBC drivers which do not support multi-threaded execution (e.g. Oracle's JDBC driver), you can configure your Connection Profile so that SQL Workbench/J will open a new connection for each tab.

If you enter more than one statement in the editor and want to execute all statements as a batch script, you need to delimit each statement. The SQL standard for terminating a SQL statement is the semicolon.

You can specifiy an alternate delimiter that can be used instead of the semicolon. See the description of the alternate delimiter for details.

Valid "scripts" are:

```
UPDATE person SET numheads = 2 WHERE name='Beeblebrox';
COMMIT;
```

You can run any statement that is valid for the current DBMS. If the statement returns a result set, it will be displayed, otherwise any messages from the server will be displayed in the messages tab. To add or remove editor pages, right click on the tab header and choose Add tab or Close tab). You can re-order the tabs by right clicking on the tab label, then choose Move left or Move right to change the position of the tab.

To execute the statement in which the cursor is currently located use **Ctrl-Enter**. The current statement is defined as the text between the previous SQL delimiter and the next SQL delimiter.

The font that is used for the SQL editor can be defined in the system preferences.

6.2. Using the alternate delimiter to execute DDL statements

SQL Workbench/J will send the contents of the editor unaltered to the DBMS, so executing DDL statements (`CREATE TABLE`, ...) is possible.

However when executing statements such as `CREATE PROCEDURE` which in turn contain valid SQL statement, delimited with a ; the SQL Workbench/J will send everything up to the first semicolon to the backend. In case of a `CREATE PROCEDURE` statement this will obviously result in an error as the statement is not complete.

This is an example of a `CREATE PROCEDURE` which will **not** work due to the embedded semicolon in the procedure source itself.

```
CREATE OR REPLACE FUNCTION proc_sample RETURN INTEGER
IS
  result INTEGER;
BEGIN
  SELECT max(col1) INTO result FROM sometable;
  RETURN result;
END;
```

When executing this script, Oracle would return an error because SQL Workbench/J will send everything up to the keyword `INTEGER` to the database. Obviously that fragment would not be correct.

The solution is to terminate the script with a character sequence called the "alternate delimiter". The value of this sequence can be configured in the options dialog as a global default, or per connection profile (so you can use different alternate delimiters for different database systems). The default is the forward slash / defined as a single line delimiter.

If a SQL statement is terminated with the alternate delimiter, that delimiter is used instead of a semicolon. This way the semicolons embedded in `CREATE PROCEDURE` statements will be sent correctly to the backend DBMS.

So the solution to the above problem is the following script:

```
CREATE OR REPLACE FUNCTION proc_sample RETURN INTEGER
IS
  result INTEGER;
BEGIN
  SELECT max(col1) INTO result FROM sometable;
  RETURN result;
END;
/
```

Note the trailing forward slash (`/`) at the end in order to "turn on" the use of the alternate delimiter. If you run scripts with embedded semicolons and you get an error, please verify the setting for your alternate delimiter.

When is the alternate delimiter used?

As soon as the statement (or script) that you execute is terminated with the alternate delimiter, the alternate delimiter is used to separate the individual SQL statements. When you execute selected text from the editor, be sure to select the alternate delimiter as well, otherwise it will not be recognized (if the alternate delimiter is not selected, the statement to be executed does not end with the alternate delimiter).

You cannot mix the standard semicolon and the alternate delimiter inside one script.

If you use the alternate delimiter (by terminating the whole script with it), then **all** statements have to be delimited with it. You cannot mix the use of the normal semicolon and the alternate delimiter for one execution. The following statement (when executed completely) would produce an error message:

```
SELECT sysdate FROM DUAL;

CREATE OR REPLACE FUNCTION proc_sample RETURN INTEGER
IS
  result INTEGER;
BEGIN
  SELECT max(col1) INTO result FROM sometable;
  RETURN result;
END;
/
```

SQL Workbench/J will use the alternate delimiter present, the `SELECT` statement at the beginning will also be sent to the database together with the `CREATE` statement. This of course is an invalid statement. You will need to either select and run each statement individually or change the delimiter after the `SELECT` to the alternate delimiter.

6.3. Dealing with BLOB and CLOB columns

SQL Workbench/J supports reading and writing `BLOB` (Binary Large OBject) or `CLOB` (Character Large OBject) columns from and to external files. BLOB clumns are sometimes also referred to as binary data. CLOB columns are sometimes also referred to as `LONG VARCHAR`. The exact data type depends on the DBMS used.

To insert and update LOB columns the usual `INSERT` and `UPDATE` statements can be used by using a special placeholder to define the source for the LOB data. When updating the LOB column, a different placeholder for BLOB and CLOB columns has to be used as the process of reading and sending the data is different for binary and character data.

When working with Oracle, only the 10g driver supports the standard JDBC calls used by SQL Workbench/J to read and write the LOB data. Earlier drivers will not work as described in this chapter.

6.3.1. Updating binary data through SQL

To update a BLOB (or binary) column, use the placeholder `{$blobfile=path_to_file}` in the place where the actual value has to occur in the `INSERT` or `UPDATE` statement:

```
UPDATE theTable
  SET blob_col = {$blobfile=c:/data/image.bmp}
WHERE id=24;
```

SQL Workbench/J will rewrite the UPDATE statement and send the contents of the file located in `c:/data/image.bmp` to the database. The syntax for inserting BLOB data is similar. Note that some DBMS might not allow you to supply a value for the blob column during an insert. In this case you need to first insert the row without the blob column, then use an `UPDATE` to send the blob data. You should make sure to update only one row by specifying an approriate `WHERE` clause.

```
INSERT INTO theTable
(id, blob_col)
VALUES
(42,{$blobfile=c:/data/image.bmp});
```

This will create a new record with id=42 and the content of `c:/data/image.bmp` in the column `blob_col`

6.3.2. Updating character data through SQL

The process of updating or inserting `CLOB` data is identical to the process for `BLOB` data. The only difference is in the syntax of the placeholder used to specify the source file. Firstly, the placeholder has to start with `{$clobfile=` and can optionally contain a parameter to define the encoding of the source file.

```
UPDATE theTable
  SET clob_col = {$clobfile=c:/data/manual.html encoding=utf8}
WHERE id=42;
```

If you ommit the encoding parameter, SQL Workbench/J will leave the data conversion to the JDBC driver (technically, it will use the `PreapredStatement.setAsciiStream()` method whereas with an encoding it will use the `PreparedStatement.setCharacterStream()` method).

The format of the `{$clobfile=}` or `{$blobfile=}` parameter has to be entered exactly as described here. You may not put e.g. spaces before or after the equal sign or the braces. If you do this, SQL Workbench/J will not recognize the parameter and will pass the statement "as is" to the JDBC driver.

6.3.3. Saving BLOB data to a file using SQL

To save the data stored in a BLOB column, the command WbSelectBlob can be used. The syntax of this command is similar to the regular `SELECT` command except that a target file has to be specified where the read data should be stored.

You can also use the WbExport command to export data. The contents of the BLOB columns will be saved into separate files. This works for both export formats (XML and Text).

6.3.4. BLOB data in the result set

When the result of your `SELECT` query contains BLOB columns, they will be displayed as `(BLOB)` together with a button. When you click on the button a dialog will be displayed allowing you to save the data to a file, view the data as text (using the selected encoding), display the blob as an image or display a hex view of the blob.

When displaying the BLOB content as a text, you can edit the text. When saving the data, the entered text will be converted to raw data using the selected encoding.

The window will also let you open the contents of the BLOB data with a predefined external tool. The tools that are defined in the options dialog can be selected from a dropdown. To open the BLOB content with one of the tools, select the tool from the dropdown list, then click on the button Open with next to the external tools dropdown. SQL Workbench/J will then retrieve the BLOB data from the server, store it in a temporary file on your harddisk, and run the selected application, passing the temporary file as a parameter.

From within this information dialog, you can also upload a file to be stored in that BLOB column. The file contents will not be sent to the database server until you actually save the changes to your result set (this is the same for all changes you make directly in the result set, for details please refer to Editing the data)

When using the upload function in the BLOB info dialog, SQL Workbench/J will use the file content for any subsequent display of the binary data or the the size information in the information dialog. You will need to re-retrieve the data, in order to use the blob data from the server.

6.4. Performance tuning when executing SQL

There are some configuration settings that affect the performance of SQL Workbench/J. On slow computers it is recommended to turn off the usage of the animated icon as the indicator for a running statement.

When running large scripts, the feedback which statement is executed can also slow down the execution. It is recommended to either turn off the feedback using WBFEEDBACK OFF or by consolidating the script log

When running imports or exports it is recommended to turn off the progress display in the statusbar that shows the current row that is imported/exported because this will slow down the process as well. In both cases you can use `-showProgress` to turn off the display (or set it to a high number such as 1000) in order to reduce the overhead caused by updating the screen.

6.5. SQL Macros

SQL Workbench/J offers so called SQL macros, or abbreviations. You can define macros for often used SQL statements. Once defined, you only need to enter the defined macro name and the underlying SQL statement will be executed.

6.5.1. Defining Macros

There are two ways to define a SQL macro.

If the current statement in the editor should be defined as a macro, select (highlight) the statement's text and select Macros » Add SQL macro from the main menu. You will be prompted to supply a name for the new macro. If you supply the name of an existing macro, the existing macro will be overwritten.

Alternatively you can add a new macro through Macros » Manage Macros.... This dialog can also be used to delete and and edit existing macros.

Once a macro is defined, you can execute it by simply typing the macro's name in the editor and execute it like any other SQL command.

6.5.2. Executing macros

To execute a macro, you can either type the alias you have defined, or select the macro from the Macros menu. The first 10 macros will be listed there directly. To view the complete list of macros select Macros » Manage Macros... After selecting a macro, it can be executed by clicking on the Run Run button. If you check the option "Replace current SQL", then the text in the editor will be replaced with the text from the macro when you click on the run button.

Macros will no be evaluated when running in batch mode!

6.5.3. Parameters in macros

Apart from the SQL Workbench/J script variables for SQL Statements, additional "parameters" can be used inside a macro definition. These parameters will be replaced *before* replacing the script variables.

Parameter	Description
${selection}$	This parameter will be replaced with the currently selected text. The selected text will not be altered in any way.
${selected_statement}$	This behaves similar to `${selection}$` except that any trailing semicolon will be removed from the selection. Thus the macro definition can always contain the semicolon (e.g. when the macro actually defines a script with multiple statements) but when selecting the text, you do not need to worry whether a semicolon is selected or not (and would potentially break the script).
${current_statement}$	This key will be replaced with the current statement (without the trailing delimiter). The current statement is defined by the cursor location and is the statement that would be executed when using SQL » Execute current [22]
${text}$	This key will be replaced with the complete text from the editor (regardless of any selection).

The SQL statement that is eventually executed will be logged into the message panel when invoking the macro from the menu. Macros that use the above paramters cannot correctly be executed by entering the macro alias in the SQL editor (and then executing the "statement").

The parameter keywords are case sensitiv, i.e. the text `${SELECTION}$` will not be replaced!

This feature can be used to create SQL scripts that work only with with an additional statement. e.g. for Oracle you could define a macro to run an explain plan for the current statement:

```
EXPLAIN PLAN FOR
${current_statement}$
;

COMMIT;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

When you run this macro, it will run an `EXPLAIN PLAN` for the statement in which the cursor is currently located, and will immediately display the results for the explain. Note that the `${current_statement}$` keyword is terminated with a semicolon, as the replacement for `${current_statement}$` will never add the semicolon. If you use `${selection}$` instead, you have to pay attention to not select the semicolon in the editor before running this macro.

For PostgreSQL you can define a similar macro that will automatically run the `EXPLAIN` command for a statemet:

```
explain ${current_statement}$
```

Another usage of the parameter replacement could be a SQL Statement that retrieves the rowcount that would be returned by the current statement:

```
SELECT count(*) FROM
(
  ${current_statement}$
)
```

6.6. Using workspaces

The complete history for all editor tabs is saved and loaded into one file, called a workspace. These workspaces can be saved and loaded to restore a specific editing context. You can assign a saved workspace to a connection profile. When the connection is established, the workspace is loaded into SQL Workbench/J. Using this feature you can maintain a completely different set of statements for different connections.

If you do not assign a workspace to a connection profile, a workspace with the name `Default.wksp` will be used for storing the statement history. This default workspace is shared between **all** profiles that have no workspace assigned.

To save the current SQL statement history and the visible tabs into a new workspace, select Workspace » Save Workspace as....

The default file extension for workspaces is `wksp`.

Once you have loaded a workspace, you can save it with Workspace » Save Workspace. The current workspace is automatically saved, when you exit SQL Workbench/J.

An existing workspace can be loaded with Workspace » Load Workspace

If you have an external file open in one of the editor tabs, the filename itself will be stored in workspace. When loading the workspace SQL Workbench/J will try to load the external file again. If the file does not exist, the last history entry from the saved history for that tab will be displayed.

The workspace file itself is a normal ZIP file, which contains one file with the statement history for each tab. The individual files can be extracted from the workspace using your favorite UNZIP tool.

6.7. Saving and loading SQL scripts

The text from the current editor can be saved to an external file, by choosing File » Save or File » Save as. The filename for the current editor will be remembered. To close the current file, select File » Discard file **(Ctrl-F4)** or use the context menu on the tab label itself.

Detaching a file from the editor will remove the text from editor as well. If you only want to detach the filename from the editor but keep the text, then press **Ctrl-Shift-F4** or hold down the **Shift** key while selecting the Discard menu item.

When you load a SQL script and execute the statements, be aware that due to the history management in SQL Workbench/J the content of the external file will be placed into the history buffer. If you load large files, this might lead to massive memory consumption. Currently only the *number* of statements put into the history can be controlled, but not the total size of the history itself. This might change with a later version.

6.8. Viewing server messages

6.8.1. MS SQL Server

For MS SQL Server, any message written with the `PRINT` command will be displayed in the `Messages` tab after the SQL command has finished. The `PRINT` command is usually used in stored procedures for logging purposes, but it can also be used as a command on its own:

```
PRINT "Deleting records...";
DELETE from my_table WHERE value = 42;
PRINT "Done."
```

This will execute the `DELETE`. Once this script has finished, the `Messages` tab will contain the text:

```
Deleting records...
Done.
```

6.8.2. Oracle

For Oracle the `DBMS_OUTPUT` package is supported. Support for this package can be turned on with the ENABLEOUT command. If this support is not turned on, the messages will not be displayed. This is the same as using the `SET SERVEROUTPUT ON` command in SQL*Plus.

If you want to turn on support for `DBMS_OUTPUT` automatically when connecting to an Oracle database, you can put the `ENABLEOUT` command into the pre-connect script.

Any message "printed" with `DBMS_OUTPUT.put_line()` will be displayed in the message part after the SQL command has finished. Please refer to the Oracle documentation if you want to learn more about the `DBMS_OUTPUT` package.

```
dbms_output.put_line("The answer is 42");
```

Once the command has finished, the following will be displayed in the `Messages` tab.

```
The answer is 42
```

6.8.3. PostgreSQL

Postgres supports a similar mechanism. Any text returned by a function or stored procedure with the `RAISE` keyword, will be displayed in the message tab as well

6.8.4. Other database systems

If your DBMS supports something similar, please let me know. I will try to implement it - provided I have free access to the DBMS. Please send your request to support@sql-workbench.net

6.9. Editing the data

Once the data has been retrieved from the database, it can be edited directly in the result set. SQL Workbench/J assumes that enough columns have been retrieved from the table so that at a unique identifier is available to identify the rows to be updated.

If you have primary keys defined for the underlying tables, those primary key columns will be used for the `WHERE` statements for `UPDATE` and `DELETE`. If no primary key columns are found, the JDBC driver is asked for a *best row identifier*. If that doesn't return any information, your defined PK Mapping will be queried. If still no PK columns can be found, you will be prompted to select the key columns based on the current result set.

The changes (modified rows, new rows or deleted rows) will not be saved to the database until you choose Data » Save. If the update is successful (no database errors) a `COMMIT` will be sent to the database automatically (if needed).

If your `SELECT` was based on more than one table, you will be prompted to specify which table should be updated. Only columns for the chosen table will be included in the `UPDATE` or `INSERT` statements. If no primary key can be found for the update table, you will be prompted to select the columns that should be used to uniquel identify a row in the update table.

If an error is reported during the update, a `ROLLBACK` will be sent to the database. The `COMMIT` or `ROLLBACK` will only be sent if autocommit is turned off.

Columns containing BLOB data will be displayed with a ... button. By clicking on that button, you can view the blob data, save it to a file or upload the content of a file to the DBMS. Please refer to BLOB support for details.

When editing, SQL Workbench/J will highlight columns that are defined as `NOT NULL` in the database. You can turn this feature off, or change the color that is used in the options dialog.

When editing date, timestamp or time fields, the format specified in the options dialog is used for parsing the entered value and converting that into the internal representation of a date. The value entered must match the format defined there.

If you want to input the current date and time you can use `now, today, sysdate, current_timestamp, current_date` instead. This will then use the current date & time and will convert this to the approriate data type for that column. e.g. `now` will be converted to the current time for a time column, the current date for a date column and the current date/time for a timestamp column. These keywords also work when importing text files using WbImport or importing a text file into the result set. The exact keywords that are recognized can be configure in the settings file

6.10. Deleting rows from the result

To delete a row from the result, select Data » Delete Row from the menu. This will remove the currently selected row(s) from the result and will mark them for deletion once the changes are saved. No foreign key checks will be done when using this option.

The generated DELETE statements will fail if the deleted row(s) are still referenced by another table. In that case, you can use Delete With Dependencies.

6.11. Deleting rows with foreign keys

To delete rows including all dependent rows, choose Data » Delete With Dependencies. In this case SQL Workbench/J will analyze all foreign keys referencing the update table, and will generate the necessary DELETE statements to delete the dependent rows, before sending the DELETE for the selected row(s).

Delete With Dependencies might take some time to detect all foreign key dependencies for the current update table. During this time a message will be displayed in the status bar. The selected row(s) will not be removed from the result set until the dependency check has finished.

Note that the generated SQL statements to delete the dependent rows will only be shown if you have enabled the preview of generated DML statements in the options dialog

You can also generate a script to delete the selected and all depending rows through Data » Generate delete script. This will not remove any rows from the current result set, but instead create and display a script that you can run at a later time.

6.12. Navigating referenced rows

Once you have retrieved data from a table that has foreign key relations to other tables, you can navigate the relationship for specific rows in the result set. Select the rows for which you want to find the data in the related tables, then right click inside the result set. In the context menu two items are available:

Referenced rows
Referencing rows

Consider the following tables:

BASE (b_id, name)
DETAIL (d_id, base_id, description) with base_id referencing BASE(b_id)
MORE_DETAIL (md_id, detail_id, description) with detail_id referencing DETAIL (d_id)

The context menu for the selected rows will give you the choice in which SQL tab you want the generated SELECT to be pasted. This is similar to the Put SELECT into feature in the table list of the DbExplorer.

Once you have obtained a result set from the table BASE, select (mark) the rows for which you want to retrieve the related rows, e.g. the one where id=1. Using Referencing rows » DETAIL SQL Workbench/J will create the following statement:

```
SELECT *
FROM DETAIL
WHERE base_id = 1;
```

The result of the generated statement will always be added to the existing results of the chosen SQL panel. By default the generated SQL statement will be appended to the text editor. If you don't want the generated statement to be appended to the editor, hold down the Ctrl key while selecting the desired menu item. In that case, the generated statement will only be written to the messages panel of the SQL tab. If the target tab contains an external file, the statement will never be appended to the editor's text.

To navigate from the child data to the "parent" data, use Referenced rows

The additional result tabs can be closed using Data » Close result

6.13. Sorting the result

The result will be displayed in the order returned by the DBMS (i.e. if you use an ORDER BY in your SELECT the display will be displayed as sorted by the DBMS). You can change the sorting of the displayed data by clicking on the header of the column that should be used for sorting. Initially the data will be sorted ascending (lower values at the top). If you click on the column again the sort order will be reversed. The sort order will be indicated by a little triangle in the column header. If the triangle points upward the data is sorted ascending, if it points downward the data is sorted descending. Clicking on a column will remove any previous sorting (including the secondary columns) and apply the new sorting.

If you want to sort by more than one column, hold down the **Ctrl** key will clicking on the (second) header. The initial sort order is ascending for that column. To change the sort order hold down the **Ctrl** key and click on the column header again. The sort order for all "secondary" sort columns will be indicated with a slightly smaller triangle than the one for the primary sort column.

To define a different secondary sort column, you first have to remove the current secondary column. This can be done by holding down the **Shift** key and clicking on the secondary column again. Note that the data will not be resorted. Once you have removed the secondary column, you can define a different secondary sort column.

6.14. Filtering the result

Once the data has been retrieved from the Server it can be filtered with the need to re-retrieve the data. You can define the filter in two ways: either enter column and their filter values manually, or create a filter from the currently selected values in the result set.

6.14.1. Defining a filter manually

To define a filter, click on the `Filter` button () in the toolbar or select Data » Filter data. A dialog will appear where you can define a filter for the current result set. Each line in the filter dialog defines an expression that will be applied to the column selected in the first dropdown. If you select `*` for the column, the filter condition will be applied to all columns of the result set.

To add a multi-column expression, press the More button, to create a new line. To remove a column expression from the filter, click the `Remove` () button. For character based column data, you can select to ignore the case of the column's data when applying the expression, i.e. when `Ignore case` is selected, the expression `'NAME = arthur'` will match the column value '`Arthur`', and '`ARTHUR`'.

By default, the column expressions are combined with an `OR`, i.e. that a row will be displayed if at least one of the column expressions evaluates to true. If you want to view only rows where **all** column expressions must match, select the `AND` radio button at the top of the dialog.

Once you have saved a filter to an external file, this filter will be available in the pick list, next to the filter icon. The list will show the last filters that were saved. The number of items displayed in this drop down can be controlled in the settings file.

6.14.2. Defining a filter from the selection

You can also quickly filter the data based on the value(s) of the currenlty selected column(s). To apply the filter, select the column values by which you want to filter then click on the `Quickfilter` button () in the toolbar or select Data » Filter by value from the menu bar.

Using the **Alt** key you can select individual columns of one or more rows. Together with the **Ctrl** key you can select e.g. the first, third and fourth column. You can also select the e.g. second column of the first, second and fifth row.

Whether the quick filter is available depends on the selected rows and columns. It will be enabled when:

- You have selected one or more columns in a single row
- You have selected one column in multiple rows

If only a single row is selected, the quick filter will use the values of the selected columns combined with `AND` to define the filter (e.g. username = 'Bob' AND job = 'Clerk'). Which columns are used depends on the way you select the row and columns. If the whole row in the result is selected, the quick filter will use the value of the focused column (the one with the yellow rectangle), otherwise the individually selected columns will be used.

If you select a single column in multiple rows, this will create a filter for that column, but with the values will be combined with `OR` (e.g. name = 'Dent' OR name = 'Prefect'). The quick filter will not be available if you select more than one column in multiple rows.

Once you have applied a quick filter, you can use the regular filter definition dialog to check the definition of the filter or to further modify it.

6.15. Export result data

You can export the data of the **result set** into local files of the following formats:

- HTML
- SQL statements (INSERT, UPDATE or DELETE & INSERT)
- XML format
- Tab separated text file. Columns are separated with a tab, rows are separated with a newline character
- Spreadsheet Format (OpenDocument, Microsoft Excel)

To save the data from the current result set into an external file, choose Data » Save Data as You will be prompted for the filename. On the right side of the file dialog you will have the possibility to define the type of the export. The export parameters on the right side of the dialog are split into two parts. The upper part defines parameters that are available for all export types. These are the encoding for the file, the format for date and date/time data and the columns that should be exported.

All format specific options that are available in the lower part, are also available when using the WbExport command. For a detailed discussion of the individual options please refer to that section.

The options `SQL UPDATE` and `SQL DELETE/INSERT` are only available when the current result has a single table that can be updated, and the primary key columns for that table could be retrieved. If the current result does not have key columns defined, you can select the key columns that should be used when creating the file. If the current result is retrieved from multiple tables, you have to supply a table name to be used for the SQL statements.

Please keep in mind that exporting the data from the result set requires you to load everything into memory. If you need to export data sets which are too big to fit into memory, you should use the WbExport command to either create SQL scripts or to save the data as text or XML files that can be imported into the database using the WbImport command. You can also use SQL » Export query result to export the result of the currently selected SQL statement.

6.16. Copy data to the clipboard

You can also copy the data from the result into the system clipboard in four different formats. In any case default settings are used for the various options of the respective format.

- Text (tab separated)

  This will use a tab as the column separator, and will not quote any values. The end-of-line sequence will be a newline (Unix style) and the column headers will be part of the copied data. Special characters (e.g. newlines) in the actual data will not be replaced (as it is possible with the WbExport command).

  When you hold down the **Shift** key when you select the menu item, the column headers will not be copied to the clipboard. When you hold down the **Ctrl** key when selecting the menu item, you can choose which columns should be copied to the clipboard. Pressing **Shift** and **Ctrl** together is also supported.

- SQL (INSERT, UPDATE, or DELETE & INSERT)

  The end-of-line sequence will be a newline (Unix style). No cleanup of data will be done as it is possible with the WbExport command, apart from correctly quoting single quotes inside the values (which is required to generate valid SQL)

As with the `Save Data as` command, the options `SQL UPDATE` and `SQL DELETE/INSERT` are only available when the current result set is updateable. If no key columns could be retrieved for the current result, you can manually define the key columns to be used, using Data » Define key columns

If you do not want to copy all columns to the clipboard, hold down the the **CTRL** key while selecting one of the menu items related to the clipboard. A dialog will then let you select the columns that you want to copy.

Alternatively you can hold down the **Alt** key while selecting rows/columns in the result set. This will allow you to select only the columns and rows that you want to copy. If you then use one of the formats available in the Copy selected submenu, only the selected cells will be copied. If you choose to copy the data as `UPDATE` or `DELETE/INSERT` statements, the generated SQL statements will not be correct if you did not select the primary key of the underlying update table.

6.17. Import data into the result set

6.17.1. Import a file into the current result set

SQL Workbench/J can import tab separated text files into the current result set. This means, that you need to issue the approriate `SELECT` statement first. The structure of the file has to match the structure of the result set, otherwise an error will occur. To initiate the import select Data » Import file

When selecting the file, you can change some parameters for the import:

Option	Description
Header	if this option this is checked, the first line of the import file will be ignored
Delimiter	the delimiter used to separate column values. Enter \t for the tab character
Date Format	The format in which date fields are specified.
Decimal char	The character that is used to indicate the decimals in numeric values (typically a dot or a comma)
Quote char	The character used to quote values with special characters. Make sure that each opening quote is followed by a closing quote in your text file.

You can also import text and XML files using the WbImport command. Using the `WbImport` command is the recommended way to import data, as it is much more flexible, and - more important - it does not read the data into memory.

6.17.2. Import the clipboard into the current result

You can import the contents of the clipboard into the current result, if the format matches the result set. When you select Data » Import from Clipboar SQL Workbench/J will check if the current clipboard contents can be imported into the current result. The data can automatically be imported if the first row of the data contains the column names. One of the following two conditions must be true in order for the import to succeed

- The columns are delimited with a tab character and the first row contains column names. All matching columns will then be imported
- If no column name matches (i.e. no header row is present) but the number of columns (identified by the number of tab characters in the first row) is identical to the number of columns in the current result.

If SQL Workbench/J cannot identify the format of the clipboard a dialog will be opened where you can specify the format of the clipboard contents. This is maily necessary if the delimiter is not the tab character.

7. Variable substitution in SQL statements

7.1. Defining variables

You can define variables within SQL Workbench/J that can be referenced in your SQL statements. This is done through the internal command `WbVarDef`, e.g.: `wbvardef myvar=42` This example defines a variable with the name `myvar` and the value `42`. If the variable does not exist, it will be created. If it exists its value will be overwritten with the new value. To remove a variable simply set its value to nothing: `wbvardef myvar=`. Alternatevily you can use the command `wbvardelete myvar` to remove a variable definition.

Variables are case sensitive.

Variables can also be read from a properties file, either by specifying `-file=filename` for the `WbVarDef` command, or by passing the `-vardef` parameter when starting SQL Workbench/J. Please see the description for the command line parameters for details.

```
wbvardef -file=/temp/myvars.def
```

This file has to be a standard Java "properties" file. Each variable is listed on a single line in the format `variable=value`. Lines starting with a # character are ignored (comments). Assuming the file `myvars.def` had the following content:

```
#Define the ID that we need later
var_id=42
person_name=Dent
another_variable=24
```

After executing `wbvardef -file=/temp/myvars.def` there would be three variables available in the system: `var_id, person_name, another_variable`, that could be used e.g. in a SELECT query:

```
SELECT * FROM person where name='$[person_name]' or id=$[var_id];
```

SQL Workbench/J would expand the variables and send the following statement to the server:

```
SELECT * FROM person where name='Dent' or id=42;
```

A variable can also be defined as the result of a `SELECT` statement. This indicated by using @ as the first character after the equal sign. The `SELECT` needs to be enclosed in double quotes, if you are using single quotes e.g. in the where clause:

```
wbvardef myvar=@"SELECT id FROM person WHERE name='Dent'"
```

When executing the statement, SQL Workbench/J uses the first column of the first row of the result set for retrieving the value for the variable. Everything else (additional columns, additional rows) will be ignored.

You can also use PreparedStatements in the SQL editor. In this case the parameters are denoted by quotation marks and you will be prompted for a value each time you run the statement (which is different to using SQL Workbench/J variables. For details on how to use prepared statements refer to support for prepared statements

7.2. Editing variables

To view a list of currently defined variables execute the command `WBVARLIST`. This will display a list of currently defined variables and their values. You can edit the resulting list similar to editing the result of a `SELECT` statement. You can add new variables by adding a row to the result, remove existing variables by deleting rows from the result, or edit the value of a variable. If you change the name of a variable, this is the same as removing the old, and creating a new one.

7.3. Using variables in SQL statements

The defined variables can be used by enclosing them in special characters inside the SQL statement. The default is set to $[and] thus you can use a variable this way:

```
SELECT firstname, lastname FROM person WHERE id=$[id_variable];
```

If you have a variable with the name `id_variable` defined, the sequence `$[id_variable]` will be replaced with the current value of the variable.

Variables will be replaced *after* replacing macro parameters.

If the SQL statement requires quotes for the SQL literal, you can either put the quotes into the value of the variable (e.g. `wbvardef name="'Arthur'"`) or you put the quotes around the variable's placeholder, e.g.: `WHERE name='$[name]';`

As you can see the variable substitution is also done inside quoted literals.

If you are using values in your regualar statements that actually need the prefix (`$[` or suffix (`]`) characters, please make sure that you have no variables defined. Otherwise you will unpredictable results. If you want to use variables but need to use the default prefix for marking variables in your statements, you can configure a different prefix and suffix for flagging variables. To change the the prefix e.g. to %# and the suffix (i.e end of the variable name) to #, add the following lines to your `workbench.settings` file:

```
workbench.sql.parameter.prefix=%#
workbench.sql.parameter.suffix=#
```

You may leave the suffix empty, but the prefix definition may not be empty.

7.4. Prompting for values during execution

You can also use variables in a way that SQL Workbench/J will prompt you during execution of a SQL statement that contains a variable.

If you want to be prompted for a value, simply reference the value with a quotation mark in front of its name:

```
SELECT id FROM person WHERE name like '$[?search_name]%'
```

If you execute this statement, SQL Workbench/J will prompt you for the value of the variable `search_name`. If the variable is already defined you will see the current value of the variable. If the variable is not yet defined it will be implicitely defined with an empty value.

If you use a variable more then once in your statement it is sufficient to define it once as a prompt variable. Prompting for a variable value is especially useful inside a macro definition.

You can also define a conditional prompt with using an ampersand instead of a quotation mark. In this case you will only be prompted if no value is assigned for the variable:

```
SELECT id FROM person WHERE name like '$[&search_name]%'
```

The first time you execute this statement (and no value has been assigned to `search_name` before using `WBVARDEF` or on the commandline) you will be prompted for a value for `search_name`. Any subsequent execution of the statement (or any other statement referencing `$[&search_name]`) will re-use the value you entered.

8. Using SQL Workbench/J in batch files

SQL Workbench/J can also be used from batch files to execute SQL scripts. This can be used to e.g. automatically extract data from a database or run other SQL queries or statements. If the -script parameter is passed on the commandline, SQL Workbench/J will run in batch mode, otherwise in GUI mode.

When using SQL Workbench/J from a batch file (or the commandline) you should **not** use the Windows launcher, as it will immediately return to the commandline while starting the JVM (and thus SQL Workbench/J) in the background.

Please refer to Starting SQL Workbench/J for details on how to start SQL Workbench/J with the `java` command.

When you need to quote parameters inside batch or shell scripts, you have to use single quotes (`'test-script.sql'`) to quote these values. Most command line shells (including Windows®) do not pass double quotes to the application and thus the parameters would not be evaluated correctly ba SQL Workbench/J

If you want to start the application from within another program (e.g. an `Ant` script or your own program), you will need to start SQL Workbench/J's main class directly.

```
java -cp sqlworkbench.jar workbench.WbStarter
```

Inside an Ant build script this would need to be done like this:

```
<java classname="workbench.WbStarter" classpath="sqlworkbench.jar" fork="true">
  <arg value="-profile='my profile'"/>
  <arg value="-script=load_data.sql"/>
</java>
```

The parameters to specifiy the connection and the SQL script to be executed have to be passed just like normally on the commandline.

8.1. Specifying the connection

When running SQL Workbench/J in batch mode, you can define the connection using a profile name or specifying the connection properties directly .

8.2. Specifying the script file(s)

With the parameter `-script=<filename>` the script file to be executed can be passed to the application. Multiple scripts can be specified by separating them with a comma. The scripts will be executed in the order in which they appear in the commandline. If the filenames contain spaces or dashes (i.e. `test-1.sql`) the names have to be quoted.

You can also execute several scripts by using the `WbInclude` command inside a script.

8.3. Specifying a delimiter

If your script files use a non-standard delimiter for the statements, you can specify an alternate delimiter through the profile or through the `-altDelimiter` parameter. The alternate delimiter should be used if you have several scripts that use the regular semicolon and the alternate delimiter. If your scripts exceed a certain size, they won't be processed in memory and detecting the alternate delimiter does not work in that case. If this is the case you can use the `-delimiter` switch to change the default delimiter for all scripts. The usage of the alternate delimiter will be disabled if this parameter is specified.

8.4. Specifying an encoding for the file(s)

In case your script files are not using the default encoding, you can specify the encoding of your script files with the `-encoding` parameter. Note that this will set for all script files passed on the commandline. If you need to run several scriptfiles with different encodings, you have to create one "master" file, which calls the individual files using the WbInclude command together with its `-encoding` parameter.

8.5. Specifying a logfile

If you don't want to write the messages to the default logfile which is defined in `workbench.settings` an alternate logfile can be specified with `-logfile=<filename>`

8.6. Handling errors

To control the behavior when errors occur during script execution, you can use the parameter `-abortOnError=[true|false]`. If any error occurs, and `-abortOnError` is `true`, script processing is completely stopped (i.e. SQL Workbench/J will be stopped). The only script which will be executed after that point is the script specified with the parameter `-cleanupError`.

If `-abortOnError` is false all statements in all scripts are executed regardless of any errors. As no error information is evaluated the script specified in -cleanupSuccess will be executed at the end.

If this parameter is not supplied it defaults to true, meaning that the script will be aborted when an error occurs.

You can also specify whether errors from `DROP` commands should be ignored. To enable this, pass the parameter `-ignoreDropErrors=true` on the commandline. This works when connecting through a profile or through a full connection specification. If this parameter is set to `true` only a warning will be issued, but any error reported from the DBMS when executing a DROP command will be ignored.

Note that this will not always have the desired effect. When using e.g. PostgreSQL with autocommit off, the current transaction will be aborted by PostgreSQL until a `COMMIT` or `ROLLBACK` is issued. So even if the error during the `DROP` is ignored, subsequent statements will fail nevertheless.

8.7. Specify a script to be executed on successful completion

The script specified with the parameter `-CleanupSuccess=<filename>` is executed as the last script if either no error occurred or AbortOnError is set to false.

If you update data in the database, this script usually contains a `COMMIT` command to make all changes permanent. The abort script usually contains a `ROLLBACK` command.

8.8. Specify a script to be executed after an error

The script specified with the parameter `-cleanupError=<filename>` is executed as the last script if `AbortOnError` is set to true and an error occurred during script execution.

The failure script usually contains a `ROLLBACK` command to undo any changes to the database in case an error occured.

8.9. Ignoring errors from DROP statements

When connecting without a profile, you can use the switch `-ignoredroperrors=[true|false]` to ignore errors that are reported from `DROP` statements. This has the same effect as connecting with a profile where the Ignore DROP errors property is enabled.

8.10. Changing the connection

You can change the current connection inside a script using the command `WbConnect`. You can either specify a profile name or specifying the connection properties directly.

The command accepts the same parameters as the commandline.

8.11. Controlling console output during batch execution

Any output generated by SQL Workbench/J during batch execution is sent to the standard output (stdout, System.out) and can be redirected if desired.

8.11.1. Displaying result sets

If you are running `SELECT` statements in your script without "consuming" the data through an `WbExport`, you can optionally display the results to the console using the parameter `-displayresult=true`. If this parameter is not passed or set to false, results sets will not be visible (for a `SELECT` statement you will simply see the message '`SELECT executed successfully`'.

8.11.2. Controlling execution feedback

When running statements, SQL Workbench/J reports success or failure of each statement. Inside a SQL script the WbFeedback command can be used to control this feedback. If you don't want to add a `WbFeedback` command to your scripts, you can control the feedback using the `-feedback` switch on the command line. Passing `-feedback=false` has the same effect as putting a `WbFeedback off` in your script.

As displaying the feedback can be quite some overhead especially when executing thousands of statements in a script file, it is recommended to turn off the result logging using `WbFeedback off` or `-feedback=false`

To you only log a summary of the script execution (per script file), specify the parameter `consolidateMessages=true`. This will then display the number of statements executed, the number of failed statements and the total number of rows affected (updated, deleted or inserted).

8.11.3. Controlling statement progress information

Several commands (like `WbExport`) show progress information in the statusbar. When running in batch mode, this information is usually not shown. When you specifiy `-showprogress=true` these messages will also be displayed on the console.

8.12. Setting configuration properties

When running SQL Workbench/J in batch mode, with no `workbench.settings` file, you can set any property by passing the property as a system property when starting the JVM. To change the loglevel to `DEBUG` you need to pass `-Dworkbench.log.level=DEBUG` when starting the application:

```
java -Dworkbench.log.level=DEBUG -jar sqlworkbench.jar
```

8.13. Examples

The examples in this section are displayed on several lines. If you enter them manually on the commandline you will need to put everything in one line, or use the escape character for your operating system to extend a single command over more then one input line.

Connect to the database without specifying a connection profile:

```
java -jar sqlworkbench.jar -url=jdbc:postgresql:/dbserver/mydb
     -driver=org.postgresql.Driver
     -username=zaphod
     -password=vogsphere
     -driverjar=C:/Programme/pgsql/pg73jdbc3.jar
     -script='test-script.sql'
```

This will start SQL Workbench/J, connect to the database server as specified in the connection parameters and execute the script `test-script.sql`. As the script's filename contains a dash, it has to be quoted. This is also necessary when the filename contains spaces.

Executing several scripts with a cleanup and failure script:

```
java -jar sqlworkbench.jar
     -script='c:/scripts/script-1.sql','c:/scripts/script-2.sql',c:/scripts/script3.sql
     -profile=Firebird
```

```
-abortOnError=false
-cleanupSuccess=commit.sql
-cleanupError=rollback.sql
```

Note that you need to quote each file individually (where it's needed) and not the value for the `-script` parameter

9. Export data using WbExport

The `WbExport` command can be used to export the contents of the database into external files. Several types of output files are supported. The `WbExport` command can either be used from within the GUI like any other command (such as `UPDATE` or `INSERT`), or it can be used as part of a SQL script that is run in batch mode.

The `WbExport` command exports either the result of the **next** SQL statement (which has to produce a result set) or the content of the table(s) specified with the `-sourceTable` parameter. The data is directly written to the output file and not loaded into memory. The export file(s) can be compressed ("zipped") on the fly. WbImport can import the zipped (text or XML) files directly without the need to unzip them.

If you want to save the data that is currently displayed in the result area into an external file, please use the Save Data as feature. You can also use the Database Explorer to export multiple tables.

When using a `SELECT` based export, you have to run both statements (`WbExport` and `SELECT`) as one script. Either select both statements in the editor and choose SQL » Execute selected, or make the two statements the only statements in the editor and choose SQL » Execute all.

You can also export the result of a `SELECT` statement, by selecting the statment in the editor, and then choose SQL » Export query result.

When exporting data into a Text or XML file, the content of BLOB columns is written into separate files. One file for each column of each row. Text files that are created this way can most probably only be imported using SQL Workbench/J as the main file will contain the filename of the BLOB data file instead of the actual BLOB data. The only other application that I know of, that can handle this type of imports is Oracle's `SQL*Loader` utility. If you run the text export together with the parameter `-writeoracleloader=true` the control file will contain the approriate definitions to read the BLOB data from the external file.

To be able to generate the binary Microsoft Excel format, you need to download the POI library from the Apache project's website: http://poi.apache.org/. Any version greater than 2.5 is fine. The distribution contains a jar file named e.g. `poi-3.0.1-FINAL-20070705.jar` (exact name depends on the version). This jar file needs to be copied into the same directory as `sqlworkbench.jar` under the name `poi.jar`.

If an export to an Excel spreadsheet is needed, it is recommended to use the Excel XML format, as that doesn't need an external library (just like exporting to the OpenDocument format).

The command supports the following parameters:

Parameter	Description
-type	Possible values: `text, sqlinsert, sqlupdate, sqldeleteinsert, xml, ods, xsl, xslt, html` Defines the type of the output file. `sqlinsert` will create the necessary `INSERT` statements to put the data into a table. If the records may already exist in the target table but you don't want to (or cannot) delete the content of the table before running the generated script, SQL Workbench/J can create a DELETE statement for every `INSERT` statement. To create this kind of script, use the `sqldeleteinsert` type. In order for this to work properly the table needs to have keycolumns defined, or you have to define the keycolumns manually using the `-keycolumns` switch. `sqlupdate` will generate UPDATE statements that update all non-key columns of the table. This will only generate valid `UPDATE` statements if at least one key column is present. If the table does not have key columns defined, or you want to use different columns, they can be specified using the `-keycolumns` switch. `ods` will generate a spreadsheet file in the OpenDocument format that can e.g. be opened with OpenOffice.org. `xlsx` will generate a spreadsheet file in the Microsoft Excel XML format and is the recommended format for creating Excel spreadsheets.

Parameter	Description
	`xls` will generate a spreadsheet file in the propriatary (binary) format for Microsoft Excel.
-createDir	If this parameter is set to true, SQL Workbench/J will create any needed directories when creating the output file.
-sourcetable	Defines a list of tables to be exported. If this switch is used, `-outputdir` is also required unless exactly one table is specified. If one table is specified, the -file parameter is used to generate the file for the table. If more then one table is specified, the `-outputdir` parameter is used to defined the directory where the generated files should be stored. Each file will be named as the exported table with the approriate extension (.xml, .sql, etc). You can specify * as the table name which will then export all tables accessible by the current user. If you want to export tables from a different user or schema you can use a schema name combined with a wildcard e.g. `-sourcetable=otheruser.*`. In this case the generated output files will contain the schema name as part of the filename (e.g. `otheruser.person.txt`). When importing these files, SQL Workbench/J will try to import the tables into the schema/user specified in the filename. If you want to import them into a different user/schema, then you have to use the `-schema` switch for the import command.
-outputDir	When using the `-sourcetable` switch with multiple tables, this parameter is mandatory and defines the directory where the generated files should be stored.
-continueOnError	When exporting more than one table, this parameter controls whether the whole export will be terminated if an error occurs during export of one of the tables.
-encoding	Defines the encoding in which the file should be written. Common encodings are ISO-8859-1, ISO-8859-15, UTF-8 (or UTF8). To get a list of available encodings, execut `WbExport` with the parameter `-showencoding`
-showEncodings	Displays the encodings supported by your Java version and operating system. If this parameter is present, all other parameters are ignored.
-lineEnding	Possible values: `crlf`, `lf` Defines the line ending to be used for XML or text files. `crlf` puts the ASCII characters #13 and #10 after each line. This is the standard format on Windows based systems. `dos` and `win` are synonym values for `crlf`, `unix` is a synonym for `lf`. `lf` puts only the ASCII character #10 at the end of each line. This is the standard format on Unix based systems (`unix` is a synonym value for this format). The default line ending used depends on the platform where SQL Workbench/J is running.
-header	Possible values: `true`, `false` If this parameter is set to true, the header (i.e. the column names) are placed into the first line of output file. The default is to not create a header line. You can define the default value for this parameter in the file workbench.settings. This parameter is valid for text and spreadsheet (OpenDocument, Excel) exports.
-compress	Selects whether the output file should be compressed and put into a ZIP archive. An archive will be created with the name of the specified outputfile but with the extension `zip`. The archive will then contain the specified file (e.g. if you specify `data.txt`, an archive `data.zip` will be created containing exactly one entry with the name `data.txt`). If the exported result set contains BLOBs, they will be stored in a separate archive, named `data_lobs.zip`. When exporting multiple tables using the `-sourcetable` parameter, then SQL Workbench/J will create one ZIP archive for each table in the specified output directory with the filename `"tablename".zip`. For any table containing BLOB data, one additional ZIP archive is created.
-clobAsFile	Possible values: `true`, `false`

Parameter	Description
	For SQL, XML and Text export this controls how the contents of CLOB fields are exported. Usually the CLOB content is put directly into the output file When generating SQL scripts with WbExport this can be a problem as not all DBMS can cope with long character literals (e.g. Oracle has a limit of 4000 bytes). When this parameter is set to true, SQL Workbench/J will create one file for each CLOB column value. This is the same behaviour as with BLOB columns. Text files that are created with this parameter set to true, will contain the filename of the generated output file instead of the actual column value. When importing such a file using `WbImport` you have to specify the `-clobIsFilename=true` parameter. Otherwise the filenames will be stored in the database and not the clob data. This parameter is not necessary when importing XML exports, as `WbImport` will automatically recognize the external files.  SQL scripts (`-type=sqlinsert`) generated with `-clobAsFile=true` can only be run with SQL Workbench/J! All CLOB files that are written using the encoding specified with the `-encoding` switch. If the `-encoding` parameter is not specified the default file encoding will be used.
-lobIdCols	When exporting CLOB or BLOB columns as external files, the filename with the LOB content is generated using the row and column number for the currently exported LOB column (e.g. data_r15_c4.data). If you prefer to have the value of a unique column combination as part of the file name, you can specify those columns using the `-lobIdCols` parameter. The filename for the LOB will then be generated using the base name of the export file, the column name of the LOB column and the values of the specified columns. If you export your data into a file called user_info and specify `-lobIdCols=id` and your result contains a column called `img`, the LOB files will be named e.g. user_info_img_344.data
-extensionColumn	When exporting CLOB or BLOB columns as external files, the extension of the generated filenames can be defined based on a column of the result set. If the exported table contains more than one type of BLOBs (e.g. JPEG, GIF, PDF) and your table stores the information to define the extension based on the contents, this can be used to re-generate proper filenames. This parameter only makes sense if exactly one BLOB column is exported.
-filenameColumn	When exporting CLOB or BLOB columns as external files, the complete filename can be taken from a column of the result set (instead of dynamically creating a new file based on the row and column numbers). This parameter only makes sense if exactly one BLOB column is exported.
-showProgress	Valid values: true, false, <numeric value> Control the update frequence in the statusbar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specifiy a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).

9.1. Parameters for the type TEXT

Parameter	Description
-delimiter	The given string sequence will be placed between two columns. The default is a tab character (`-delimiter=\t`
-dateFormat	The date format to be used when writing date columns into the output file.
-timestampFormat	The format to be used when writing datetime (or timestamp) columns into the output file.
-quoteChar	The character (or sequence of characters) to be used to enclose text (character) data if the delimiter is contained in the data. By default quoting is disabled until a quote character is defined. To set the double quote as the quote character you have to enclose it in single quotes: `-quotechar='"'`
-quoteCharEscaping	Possible values: `none`, `escape`, `duplicate`

Parameter	Description
	Defines how quote characters that appear in the actual data are written to the output file. If no quote character has been defined using the -quoteChar switch, this option is ignored. If `escape` is specified a quote character (defined through -quoteChar) that is embedded in the exported (character) data is written as e.g. `here is a \" quote character`. If `duplicate` is specified, a quote character (defined through -quoteChar) that is embedded in the exported (character) data is written as two quotes e.g. `here is a "" quote character`.
-quoteAlways	Possible values: `true`, `false` If quoting is enabled (via `-quotechar`, then character data will normally only be quoted if the delimiter is found inside the actual value that is written to the output file. If `-quoteAlways=true` is specified, character data will always be enclosed in the specified quote character. This parameter is ignored if not quote character is specified. If you expect the quote character to be contained in the values, you should enable character escaping, otherwise the quote character that is part of the exported value will break the quote during import.
-decimal	The decimal symbol to be used for numbers. The default is a dot (e.g. 3.14152)
-escapeText	This parameter controls the escaping of non-printable or non-ASCII characters. Valid options are `ctrl` which will escape everything below ASCII 32 (newline, tab, etc), `7bit` which will escape everything below ASCII 32 and above 126, `8bit` which will escape everything below ASCII 32 and above 255 and `extended` which will escape everything outside the range [32-126] and [161-255] This will write a unicode representation of the character into the text file e.g. \n for a newline, \u00F6 for ö. This file can only be imported using SQL Workbench/J (at least I don't know of any DBMS specific loader that will decode this properly) If character escaping is enabled, then the quote character will be escaped inside quoted values and the delimiter will be escaped inside non-quoted values. The delimiter could also be escaped inside a quoted value if the delimiter falls into the selected escape range (e.g. a tab character).
-formatFile	Possible values: `oracle`, `sqlserver` This parameter controls the creation of a control file for the bulk load utilities of Oracle and Microsoft SQL Server. `oracle` will create a control file for Oracle's SQL*Loader utility, `sqlserver` will create a format file for Microsoft's `bcp` utility. The format file has the same filename as the output file but with the ending `.ctl` for Oracle and `.fmt` for SQL Server. You can specify both formats. In that case two control files will be created. Note that the generated control file will most probably need some adjustments before you can actually use it.

9.2. Parameters for type XML

Parameter	Description
-table	The given tablename will be put into the <table> tag as an attribute.
-dateFormat	The date format to be used when writing date columns into the output file.
-timestampFormat	The format to be used when writing datetime (or timestamp) columns into the output file.
-decimal	The decimal symbol to be used for numbers. The default is a dot (e.g. 3.14152)
-useCDATA	Possible values: `true`, `false` Normally any character data written into the xml file will be processed to escape XML characters (e.g. < will be written as <). If you don't want that escaping, set `-usecdata=true` and all character data (VARCHAR, etc) will be enclosed in a CDATA section.

Parameter	Description

	With `-cdata=true` a HTML value would be written like this:
	
	`<![CDATA[<b>This is a title</b>]]>`
	
	With `-cdata=false` (the default) a HTML value would be written like this:
	
	`<b>This is a title</b>`

-stylesheet	The name of the XSLT stylesheet that should be used to transform the SQL Workbench/J specific XML file into a different format. If -stylesheet is specified, -xsltoutput has to be specified as well.
-xsltOutput	The resulting output file (specified with the -file parameter), can be transformed using XSLT after the export has finished. This parameter then defines the name of the outputfile of the transformation.
-verboseXML	Possible values: `true`, `false` This parameter controls the tags that are used in the XML file and minor formatting features. The default is -verboseXML=true and this will generate more readable tags and formatting. However the overhead imposed by this is quite high. Using -verboseXML=false uses shorter tag names (not longer then two characters) and does put more information in one line. This output is a harder to read for a human but is smaller in size which could be important for exports with large result sets.

9.3. Parameters for type SQLUPDATE, SQLINSERT or SQLDELETEINSERT

Parameter	Description
-table	Define the tablename to be used for the UPDATE or INSERT statements. This parameter is required if the SELECT statement has multiple tables in the FROM list. table.
-charfunc	If this parameter is given, any non-printable character in a text/character column will be replaced with a call to the given function with the ASCII value as the parameter. If -charfunc=chr is given (e.g. for an Oracle syntax), a CR (=13) inside a character column will be replaced with: `INSERT INTO ... VALUES ('First line'||chr(13)||'Second line' ... )` This setting will affect ASCII values from 0 to 31
-concat	If the parameter `-charfunc` is used SQL Workbench/J will concatenate the individual pieces using the ANSI SQL operator for string concatenation. In case your DBMS does not support the ANSI standard (e.g. MS ACCESS) you can specify the operator to be used: `-concat=+` defines the plus sign as the concatenation operator.
-blobType	Possible values: `file`, `dbms`, `ansi` This parameter controls how BLOB data will be put into the generated SQL statements. By default no conversion will be done, so the actual value that is written to the output file depends on the JDBC driver's implementation of the Blob interface. The parameter value `file`, will cause SQL Workbench/J to write the contents of each blob column into a separate file. The SQL statement will contain the SQL Workbench/J specific extension to read the blob data from the file. For details please refer to BLOB support. If you are planning to run the generated SQL scripts using SQL Workbench/J this is the recommended format.  When using `-blobType=file` the generated SQL script can only be run with SQL Workbench/J!

Parameter	Description
	The parameter value `ansi`, will generate "binary strings" that are compatible with the ANSI definition for binary data. MySQL and Microsoft SQL Server support these kind of literals. The parameter value `dbms`, will create a DBMS specific "binary string". MySQL, HSQLDB, H2 and PostgreSQL are known to support literals for binary data. For other DBMS using this option will still create an ansi literal but this might result in an invalid SQL statement.
-sqlDateLiterals	Possible values: `jdbc`, `ansi`, `dbms`, `default` This parameter controls the generation of date or timestamp literals. By default literals that are specific for the current DBMS are created. You can also choose to create literals that comply with the JDBC specification or ANSI SQL literals for dates and timestamps. **`jdbc`** selects the creation of JDBC compliant literals. These should be usable with every JDBC based tool, including your own Java code: `{d '2004-04-28'}` or `{ts '2002-04-02 12:02:00.042'}`. This is the recommended format if you plan to use SQL Workbench/J (or any other JDBC based tool) to run the generated statements. **`ansi`** selects the creation of ANSI SQL compliant date literals: `DATE '2004-04-28'` or `TIMESTAMP '2002-04-02 12:04:00'`. Please consult the manual of the target DBMS, to find out whether it supports ANSI compliant date literals. **`default`** selects the creation of quoted date and timestamp literals in ISO format (e.g. '2004-04-28'). Several DBMS support this format (e.g. Postgres, Microsoft SQL Server) **`dbms`** selects the creation of specific literals to be used with the current DBMS (using e.g. the `to_date()` function for Oracle). The format of these literals can be customized if necessary in `workbench.settings` using the keys `workbench.sql.literals.[type].[datatype].pattern` where [type] is the type specified with this parameter and [datatype] is one of `time, date, timestamp`. If you add new literal types, please also adjust the key `workbench.sql.literals.types` which is used to show the possible values in the GUI (auto-completion "Save As" dialog, Options dialog). If no type is specified (or *dbms*), SQL Workbench/J first looks for an entry where [type] is the current dbid. If no value is found, `default` is used. You can define the default literal format to be used for the WbExport command in the options dialog.
-commitEvery	A numeric value which identifies the number of `INSERT` or `UPDATE` statements after which a `COMMIT` is put into the generated SQL script. -commitevery=100 will create a `COMMIT;` after every 100th statement. If this is not specified one `COMMIT;` will be added at the end of the script. To suppress the final COMMIT, you can use `-commitEvery=none`. Passing `-commitEvery=atEnd` is equivalent to `-commitEvery=0`
-createTable	Possible values: `true`, `false` If this parameter is set to true, the necessary `CREATE TABLE` command is put into the output file. This parameter is ignored when creating `UPDATE` statements.
-keyColumns	A comma separated list of column names that occur in the table or result set that should be used as the key columns for `UPDATE` or `DELETE` If the table does not have key columns, or the source SELECT statement uses a join over several tables, or you do not want to use the key columns defined in the database, this key can be used to define the key columns to be used for the UPDATE statements. This key overrides any key columns defined on the base table of the SELECT statement.

9.4. Parameters for Spreadsheet types (ods, xslx, xls)

Parameter	Description
-pageTitle	The name to be used for the worksheet

9.5. Compressing export files

The `WbExport` command supports compressing of the generated output files. This includes the "main" export file and any associated LOB files.

When using `WbImport` you can import the data stored in the archives without unpacking them. Simply specify the archive name with the `-file` parameter. SQL Workbench/J will detect that the input file is an archive and will extract the information "on the fly". Assume the following export command:

```
WbExport -type=text -file=/home/data/person.txt -compress=true -sourcetable=person;
```

This command will create the file `/home/data/person.zip` that will contain the specified `person.txt`. To import this export into the table employee, you can use the following command:

```
WbImport -type=text -file=/home/data/person.zip -table=employee;
```

Assuming the `PERSON` table had a BLOB colum (e.g. a picture of the person), the `WbExport` command would have created an additional file called `person_blobs.zip` that would contain all BLOB data. The `WbImport` command will automatically read the BLOB data from that archive.

9.6. Examples

```
WbExport -type=text
         -file='c:/data/data.txt'
         -delimiter='|'
         -decimal=',';
SELECT * FROM data_table;
```

Will create a text file with the data from `data_table`. Each column will be separated with the character | Each fractional number will be written with a comma as the decimal separator. As the SELECT statement retrieves all rows and columns from the table, this could also be written as:

```
WbExport -type=text
         -file='c:/data/data.txt'
         -delimiter='|'
         -decimal=','
         -sourcetable=data_table;
```

To export all tables from the current connection into tab-separated files and compress the files, you can use the following statement:

```
WbExport -type=text
         -outputDir=c:/data/export
         -compress=true
         -sourcetable=*;
```

This will create one zip file for each table containing the exported data as a text file. If a table contains BLOB columns, the blob data will be written into a separate zip file.

The files created by the above statement can be imported into another database using the following command:

```
WbImport -type=text
         -sourceDir=c:/data/export
         -checkDependencies=true;
```

To generate a file that contains `INSERT` statements that can be executed on the target system, the following command can be used:

```
WbExport -type=sqlinsert
         -file='c:/data/newtable.sql'
         -table=newtable;
SELECT * FROM table1, table2
WHERE table1.column1 = table2.column1;
```

will create a SQL scripts which inserts the data from table1 and table2 into a table called newtable. If the parameter -table is omitted, the creation of SQL `INSERT` statements is only possible, if the SELECT is based on a single table (or view).

WbExport and the "Max. Rows" option

There is a difference in the behaviour of the command regarding the "Max. Rows" setting in the GUI. When you use the `WbExport` command together with a `SELECT` query, the Max. Rows setting will be respected by the `SELECT` statement (and you will see a warning that the result set was limited). When you use the `WbExport` with the `-sourcetable` switch, the "Max. Rows" setting will not be respected, and all rows from the table will be written into the specified file.

10. Import data using WbImport

The `WbImport` command can be used to import data from text or XML files into a table of the database. WbImport can read the XML files generated by the WbExport command's XML format. It can also read text files created by the WbExport command that escape non-printable characters.

The `WbImport` command can either be used from within the GUI like any other SQL command (such as `UPDATE` or `INSERT`), or it can be used as part of a SQL script that is run in batch mode.

During the import of text files, empty lines (i.e. lines which only contain whitespace) will be silently ignored.

`WbImport` recognizes certain "literals" to identify the current date or time when converting values from text files to the approriate data type of the DBMS. Thus, input values like `now`, or `current_timestamp` for date or timestamp columns are converted correctly. For details on which "literals" are supported, please see the description about editing data [29].

The DataPumper can also be used to import text files into a database table, though it does not offer all of the possibilities from the `WbImport` command.

Archives created with the `WbExport` command using the `-compress=true` parameter can be imported using `WbImport` command. You simply need to specifiy the archive file created by `WbExport`, and `WbImport` will automatically detect the archive. For an example on creating and importing compressed exports, please refer to compressing export files

10.1. General parameters

The WbImport command has the following syntax

Parameter	Description
-type	Possible values: `xml`, `text` Defines the type of the input file
-file	Defines the full name of the input file. Alternatively you can also specify a directory (using `-sourcedir`) from which all files are imported.
-sourceDir	Defines a directory which contains import files. All files from that directory will be imported. If this switch is used with text files and no target table is specified, then it is assumed that each filename (without the extension) defines the target table. If a target table is specified using the `-table` parameter, then all files will be imported into the same table. The `-deleteTarget` will be ignored if multiple files are imported into a single table.
-checkDependencies	When importing more than one file (using the `-sourcedir` switch), into tables with foreign key constraints, this switch can be used to import the files in the correct order (child tables first). When `-checkDependencies=true` is passed, SQL Workbench/J will check the foreign key dependencies for all tables.
-extension	When using the `-sourcedir` switch, the extension for the files can be defined. All files ending with the supplied value will be processed. (e.g. `-extension=csv`). The extension given is case-sensitiv (i.e. `TXT` is something different than `txt`
-commitEvery	A numeric value that defines the number of rows after which a `COMMIT` is sent to the DBMS. If this parameter is not passed (or a value of zero or lower), then SQL Workbench/J will commit when all rows have been imported. When using batch execution it is recommended to commit the batch using the `-commitBatch` parameter.
-mode	Defines how the data should be sent to the database. Possible values are '`INSERT`', '`UPDATE`', '`INSERT,UPDATE`' and '`UPDATE,INSERT`' For details please refer to the update mode explanation.
-continueOnError	Possible values: `true`, `false` This parameter controls the behaviour when errors occur during the import. The default is `true`, meaning that the import will continue even if an error occurs during file parsing or updating the database. Set this parameter to `false` if you want to stop the import as soon as an error occurs.

Parameter	Description
	The default value for this parameter can be controlled in the settings file and it will be displayed if you run `WbImport` without any parameters. With PostgreSQL `continueOnError` will only work, if the use of savepoints is enabled. This can be done using by setting the property `workbench.db.postgresql.import.usesavepoint=true` in in the configuration file `workbench.settings`. If this is enabled, then each INSERT (or UPDATE) statement will be "wrapped" between savepoints so that a statement error can be recovered.
-keyColumns	Defines the key columns for the target table. This parameter is only necessary if import is running in `UPDATE` mode. This parameter is ignored if files are imported using the `-sourcedir` parameter
-table	Defines the table into which the data should be imported This parameter is ignored, if the files are imported using the `-sourcedir` parameter
-schema	Defines the schema into which the data should be imported. This is necessary for DBMS that support schemas, and you want to import the data into a different schema, then the current one.
-encoding	Defines the encoding of the input file (and possible CLOB files)
-deleteTarget	Possible values: `true`, `false` If this parameter is set to true, data from the target table will be deleted (using `DELETE FROM ...`) before the import is started. This parameter will only be used if `-mode=insert` is specified.
-truncateTable	Possible values: `true`, `false` This is essentially the same as `-deleteTarget`, but will use the command `TRUNCATE` to delete the contents of the table. For those DBMS that support this command, deleting rows is usually faster compared to the `DELETE` command, but it cannot be rolled back. This parameter will only be used if `-mode=insert` is specified.
-batchSize	A numeric value that defines the size of the batch queue. Any value greater than 1 will enable batch mode. If the JDBC driver supports this, the INSERT (or UPDATE) performance can be increased drastically. This parameter will be ignored if the driver does not support batch updates or if the mode is not `UPDATE` or `INSERT` (i.e. if `-mode=update,insert` or `-mode=insert,update` is used).
-commitBatch	Possible values: `true`, `false` If using batch execution (by specifying a batch size using the `-batchSize` parameter) each batch will be committed when this parameter is set to `true`. This is slightly different to using `-commitEvery` with the value of the `-batchSize` parameter. The latter one will add a COMMIT statement to the batch queue, rather than calling the JDBC commit() method. Some drivers do not allow to add different statements in a batch queue. So, if a frequent `COMMIT` is needed, this parameter should be used. When you specify `-commitBatch` the parameter `-commitEvery` will be ignored. If no batch size is given, then `-commitBatch` will be ignored.
-transactionControl	Possible values: `true`, `false` Controls whether SQL Workbench/J handles the transaction for the import, or if the import must be committed (or rolled back) manually. If `-transactionControl=false` is specified, the SQL Workbench/J will neither send a `COMMIT` nor a `ROLLBACK` at the end. This can be used when multiple files need to be imported in a single transaction. This can be combined with the cleanup and error scripts in batch mode.

Parameter	Description
-updateWhere	When using update mode an additional `WHERE` clause can be specified to limit the rows that are updated. The value of the `-updatewhere` parameter will be added to the generated `UPDATE` statement. If the value starts with the keyword `AND` or `OR` the value will be added without further changes, otherwise the value will be added as an `AND` clause enclosed in brackets. This parameter will be ignored if update mode is not active.
-startRow	A numeric value to define the first row to be imported. Any row before the specified row will be ignored. The header row is not counted to determine the row number. For a text file with a header row, the pysical line 2 is row 1 (one) for this parameter.
-endRow	A numeric value to define the last row to be imported. The import will be stopped after this row has been imported. When you specify `-startRow=10` and `-endRow=20` 11 rows will be imported (i.e. rows 10 to 20). If this is a text file import with a header row, this would correspond to the physical lines 11 to 21 in the input file as the header row is not counted.
-badFile	Possible values: `true`, `false` If `-continueOnError=true` is used, you can specify a file to which rejected rows are written. If the provided filename denotes a directory a file with the name of the import table will be created in that directory. When doing multi-table inserts you **have** to specify a directory name. If a file with that name exists it will be deleted when the import for the table is started. The fill will not be created unless at least one record is rejected during the import. The file will be created with the same encoding as indicated for the input file(s).
-maxLength	With the parameter `-maxLength` you can truncate data for character (VARCHAR, CAHR) columns during import. This can be used to import data into columns that are not big enough (e.g. VARCHAR columns) to hold all values from the input file and to ensure the import can finish without errors. The parameter defines the maximum length for certain columns as using the following format: `-maxLength='firstname=30,lastname=20'` Where firstname and lastname are columns from the target table. The above example will limit the values for the column firstname to 30 characters and the values for the column lastname to 20 characters. If a non-character column is specified this is ignored. Note that you **have** quote the parameter's value in order to be able to use the "embedded" equals sign.
-booleanToNumber	Possible values: `true`, `false` When exporting data from a DBMS that supports the BOOLEAN datatype, the export file will contain the literals "true" or "false" for the value of the boolean columns. When importing this file into a DBMS that does not support the BOOLEAN datatype, the import would fail. In case you are importing the boolean column into a numeric column in the target DBMS, SQL Workbench/J will automatically convert the literal `true` to the numeric value 1 (one) and the literal `false` to the numeric value 0 (zero). If you do not want this automatic conversion, you have to specify `-booleanToNumber=false` for the import. The default values for the true/false literals can be overwritten with the -literalsFalse and -literalsTrue switches.
-literalsFalse -literalsTrue	When dealing with boolean values in the input file, these two switches define the literals that represent the value `false` and the value `true` when parsing the input data. The value to these switches is a comma separated list of literals that should be treated as the specified value, e.g.: `-literalsFalse='false,0' -literalsTrue='true,1'` will define the most commonly used values for true/false. Please note: • The definition of the literals is case sensitive! • You always have to specify both switches, otherwise the definition will be ignored

Parameter	Description
-constantValues	With this parameter you can supply constant values for one or more columns that will be used when inserting new rows into the database. The constant values will not be used when updating columns! The format of the values is `-constantValues="column1=value1,column2=value2"`. The values will be converted by the same rules as the input values from the input file. If the value for a character column is enclosed in single quotes, these will be removed from the value before sending it to the database. To include single quotes at the start or end of the input value you need to use two single quotes, e.g.`-constantValues="name=''Quoted'',title='with space'"` For the field `name` the value `'Quoted'` will be sent to the database. for the field `title` the value `with space` will be sent to the database. To specify a function call to be executed, enclose the function call in `${...}`, e.g. `${mysequence.nextval}` or `${myfunc()}`. The supplied function will be put into the `VALUES` part of the INSERT statement without further checking (after removing the ${ and } characters, of course). So make sure that the syntax is valid for your DBMS. If you do need to store a literal like `${some.value}` into the database, you need to quote it: `-constantValues="varname='${some.value}'"`.

-preTableStatement -postTableStatement	This parameter defines a SQL statement that should be executed before the import process starts inserting data into the target table. The name of the current table (when e.g. importing a whole directory) can be referenced using `${table.name}`.
	
	To define a statement that should be executed after all rows have been inserted and have been committed, you can use the `-postTableStatement` parameter.
	
	These parameters can e.g. be used to enable identity insert for MS SQL Server:
	
	`-preTableStatement="set identity_insert ${table.name} on"`
	`-postTableStatement="set identity_insert ${table.name} off"`
	
	Errors resulting from executing these statements will be ignored. If you want to abort the import in that case you can specify `-ignorePrePostErrors=false` and `-continueOnError=false`.

-ignorePrePostErrors=false	Controls handling of errors for the -preTableStatement and -postTableStatement parameters. If this is set to false, errors resulting from executing the supplied parameters are ignored. If set to true (default) then error handling depends on the parameter `-continueOnError`.
-showProgress	Valid values: true, false, <numeric value> Control the update frequence in the statusbar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specifiy a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).

10.2. Parameters for the type TEXT

Parameter	Description
-fileColumns	A comma separated list of the table columns in the import file Each column from the file should be listed with the approriate column name from the target table. This parameter also defines the order in which those columns appear in the file. If the file does not contain a header line or the header line does not contain the names of the columns in the database (or has different names), this parameter has to be supplied. If a column from the input file has no match in the target table, then it should be specified with the name wb_skip. You can also specify the wb_skip flag for columns which are present but that you want to exclude from the import. This parameter is ignored when the `-sourceDir` parameter is used.

Parameter	Description
-importColumns	Defines the columns that should be imported. If all columns from the input file should be imported (the default), then this parameter can be ommited. If only certain columns should be imported then the list of columns can be specified here. The column names should match the names provided with the -filecolumns switch. The same result can be achieved by providing the columns that should be excluded as `$wb_skip$` columns in the `-filecolumns` switch. Which one you choose is mainly a matter of taste. Listing all columns and excluding some using `-importcolumns` might be more readable because the structure of the file is still "visible" in the `-filecolumns` switch. This parameter is ignored when the `-sourcedir` parameter is used.
-delimiter	Define the character which separates columns in one line. Records are always separated by newlines (either CR/LF or a single a LF character) unless `-multiLine=true` is specified Default value: \t (a tab character)
-columnWidths	To import files that do not have a delimiter but a fixed width for each column, this parameters defines the width of each column in the input file. The value for this parameter is a comma separated list, where each element defines the width for a single column. If this parameter is given, the `-delimiter` parameter is ignored. e.g.: -columnWidths='name=10,lastname=20,street=50,flag=1' Note that the whole list must be enclosed in quotes as the parameter value contains the equal sign. If you want to import only certain columns you have to use `-fileColumns` and `-importColumns` to select the columns to import. You cannot use `$wb_skip$` in the -fileColumns parameter with a fixed column width import.
-dateFormat	The format for date columns.
-timestampFormat	The format for datetime (or timestamp) columns in the input file.
-quoteChar	The character which was used to quote values where the delimiter is contained. This parameter has no default value. Thus if this is not specified, no quote checking will take place. If you use `-multiLine=true` you **have** to specify a quote character in order for this to work properly.
-quoteCharEscaping	Possible values: `none`, `escape`, `duplicate` Defines how quote characters that appear in the actual data are stored in the input file. You have to define a quote character in order for this option to have an effect. The character defined with the -quoteChar switch will then be imported according to the setting defined by this switch. If `escape` is specified, it is expected that a quote that is part of the data is preceded with a backslas, e.g. the input value `here is a \" quote character` will be imported as `here is a " quote character` If `duplicate` is specified, it is expected that the quote character is duplicated in the input data. This is similar to the handling of single quotes in SQL literals. The input value `here is a "" quote character` will be imported as `here is a " quote character`
-multiLine	Possible values: `true`, `false` Enable support for records spanning more than one line in the input file. These records have to be quoted, otherwise they will not be recognized. If you create your exports with the WbExport command, it is recommended to encode special characters using the `-escapetext` switch rather then using multi-line records. The default value for this parameter can be controlled in the settings file and it will be displayed if you run `WbImport` without any parameters.

Parameter	Description
-decimal	The decimal symbol to be used for numbers. The default is a dot
-header	Possible values: `true`, `false` If set to true, indicates that the file contains a header line with the column names for the target table. This will also ignore the data from the first line of the file. If the column names to be imported are defined using the `-filecolumns` or the `-importcolumns` switch, this parameter has to be set to true nevertheless, otherwise the first row would be treated as a regular data row. This parameter is always set to true when the `-sourcedir` parameter is used. The default value for this option can be changed in the settings file and it will be displayed if you run `WbImport` without any parameters. It defaults to `true`
-decode	Possible values: `true`, `false` This controls the decoding of escaped characters. If the export file was e.g. written with escaping enabled then you need to set `-decode=true` in order to interpret string sequences like \t, \n or escaped Unicode characters properly. This is not enabled by default because applying the necessary checks has an impact on the performance.
-columnFilter	This defines a filter on column level that selects only certain rows from the input file to be sent to the database. The filter has to be defined as `column1="regex",column2="regex"`. Only Rows matching all of the supplied regular expressions will be included by the import. This parameter is ignored when the `-sourcedir` parameter is used.
-lineFilter	This defines a filter on the level of the whole input row (rather than for each column individually). Only rows matching this regular expression will be included in the import. The complete content of the row from the input file will be used to check the regular expression. When defining the expression, remember that the (column) delimiter will be part of the input string of the expression.
-emptyStringIsNull	Possible values: `true`, `false` Controls whether input values for character type columns with a length of zero are treated as `NULL` (value `true`) or as an empty string. The default value for this parameter is `true` Note that, input values for non character columns (such as numbers or date columns) that are empty or consist only of whitespace will always be treated as `NULL`.
-trimValues	Possible values: `true`, `false` Controls whether leading and trailing whitespace are removed from the input values before they are stored in the database. When used in combination with `-emptyStringIsNull=true` this means that a column value that contains only whitespace will be stored as `NULL` in the database. The default value for this parameter can be controlled in the settings file and it will be displayed if you run `WbImport` without any parameters. Note that, input values for non character columns (such as numbers or date columns) are always trimmed before converting them to their target datatype.
-blobIsFilename	Possible values: `true`, `false`

Parameter	Description
	When exporting tables that have BLOB columns using WbExport into text files, each BLOB will be written into a separate file. The actual column data of the text file will contain the file name of the external file. When importing text files that do not reference external files into tables with BLOB columns setting this paramter to false, will send the content of the BLOB column "as is" to the DBMS. This will of course only work if the JDBC driver can handle the data that in the BLOB columns of the text file. The default for this parameter is `true`
-clobIsFilename	Possible values: `true`, `false` When exporting tables that have CLOB columns using WbExport and the parameter `-clobAsFile=true` the generated text file will not contain the actual CLOB contents, but the a filename indicating the file in which the CLOB content is stored. In this case `-clobIsFilename=true` has to be specified in order to read the CLOB contents from the external files. The CLOB files will be read using the encoding specified with the `-encoding` parameter.

10.3. Text Import Examples

```
WbImport -file=c:/temp/contacts.txt
         -table=person
         -filecolumns=lastname,firstname,birthday
         -dateformat="yyyy-MM-dd";
```

This imports a file with three columns into a table named person. The first column in the file is `lastname`, the second column is `firstname` and the third column is `birhtday`. Values in date columns are formated as yyyy-MM-dd

A special timestamp format `millis` is availalbe to identify times represented in milliseconds (since January 1, 1970, 00:00:00 GMT).

```
WbImport -file=c:/temp/contacts.txt
       -table=person
       -filecolumns=lastname,firstname,$wb_skip$,birthday
       -dateformat="yyyy-MM-dd";
```

This will import a file with four columns. The third column in the file does not have a corresponding column in the table `person` so its specified as `$wb_skip$` and will not be imported.

```
WbImport -file=c:/temp/contacts.txt
         -table=person
         -filecolumns=lastname,firstname,phone,birthday
         -importcolumns=lastname,firstname;
```

This will import a file with four columns where all columns exist in the target table. Only `lastname` and `firstname` will be imported. The same effect could be achieved by specifying wb_skip for the last two columns and leaving out the -importcolumns switch. Using -importcolumns is a bit more readable because you can still see the structure of the input file. The version with `$wb_skip$` is mandatory if the input file contains columns that do not exist in the target table.

If you want to import certain rows from the input file, you can use regular expressions:

```
WbImport -file=c:/temp/contacts.txt
         -table=person
         -filecolumns=lastname,firstname,birthday
         -columnfilter=lastname="^Bee.*",firstname="^Za.*"
         -dateformat="yyyy-MM-dd";
```

The above statement will import only rows where the column `lastname` contains values that start with `Bee` and the column `firstname` contains values that start with `Za`. So `Zaphod Beeblebrox` would be imported, `Arthur Beeblebrox` would not be imported.

If you want to learn more about regular expressions, please have a look at http://www.regular-expressions.info/

If you want to limit the rows that are updated but cannot filter them from the input file using `-columnfilter` or `-linefilter`, use the `-updatewhere` parameter:

```
WbImport -file=c:/temp/contacts.txt
         -table=person
         -filecolumns=id,lastname,firstname,birthday
         -keycolumns=id
         -mode=update
         -updatewhere="source <> 'manual'"
```

This will update the table `PERSON`. The generated `UPDATE` statement would normally be: `UPDATE person SET lastname=?, firstname=?, birthday=? WHERE id=?`. The table contains entries that are maintained manually (identified by the value 'manual' in the column `source`) and should not be updated by SQL Workbench/J. By specifying the `-updatewhere` parameter, the above `UPDATE` statement will be extended to `WHERE id=? AND (source <> 'manual')`. Thus skipping records that are flagged as manual even if they are contained in the input file.

```
WbImport -sourceDir=c:/data/backup
         -extension=txt
         -header=true
```

This will import all files with the extension `txt` located in the directory `c:/data/backup` into the database. This assumes that each filename indicates the name of the target table.

```
WbImport -sourceDir=c:/data/backup
         -extension=txt
     -table=person
         -header=true
```

This will import all files with the extension `txt` located in the directory `c:/data/backup` into the table `person` regardless of the name of the input file. In this mode, the parameter `-deleteTarget` will be ignored.

10.4. Parameters for the type XML

The XML import only works with files generated by the WbExport command.

Parameter	Description
-verboseXML	Possible values: `true`, `false` If the XML was generated with `-verboseXML=false` then this needs to be specified also when importing the file. Beginning with build 78, the SQL Workbench/J writes the information about the used tags into the meta information. So it is no longer necessary to specify whether -verboseXML was true when creating the XML file.
-sourceDir	Specify a directory which contains the XML files. All files in that directory ending with ".xml" (lowercase!) will be processed. The table into which the data is imported is read from the XML file, also the columns to be imported. The parameters `-keycolumns`, `-table` and `-file` are ignored if this parameter is specified. If XML files are used that are generated with a version prior to build 78, then all files need to use either the long or short tag format and the `-verboseXML=false` parameter has to be specified if the short format was used. When importing several files at once, the files will be imported into the tables specified in the XML files. You cannot specify a different table (apart from editing the XML file before starting the import).
-importColumns	Defines the columns that should be imported. If all columns from the input file should be imported (the default), then this parameter can be ommited. When specified, the columns have to match the column names available in the XML file.
-createTarget	If this parameter is set to `true` the target table will be created, if it doesn't exist. Valid values are `true` or `false`.

10.5. Update mode

The `-mode` parameter controls the way the data is sent to the database. The default is `INSERT`. SQL Workbench/J will generate an `INSERT` statement for each record. If the `INSERT` fails no further processing takes place for that record.

If `-mode` is set to `UPDATE`, SQL Workbench/J will generate an `UPDATE` statement for each row. In order for this to work, the table needs to have a primary key defined, and all columns of the primary key need to be present in the import file. Otherwise the generated `UPDATE` statement will modify rows that should not be modified. This can be used to update existing data in the database based on the data from the export file.

To either update or insert data into the table, both keywords can be specified for the `-mode` parameter. The order in which they appear as the parameter value, defines the order in which the respective statements are sent to the database. If the first statement fails, the second will be executed. For `-mode=insert,update` to work properly a primary or unique key has to be defined on the table. SQL Workbench/J will catch any exception (=error) when inserting a record, then it will try updating the record, based on the specified keycolumns. The `-mode=update,insert` works the other way. First SQL Workbench/J will try to update the record based on the primary keys. If the DBMS signals that no rows have been updated, it is assumed that the row does not exist and the record will be inserted into the table. This mode is recommended when no primary or unique key is defined on the table, and an `INSERT` would always succeed.

The keycolumns defined with the `-keycolumns` parameter don't have to match the real primary key, but they should identify one row uniquely.

You cannot use the update mode, if the tables in question **only** consist of key columns (or if only key columns are specified). The values from the source are used to build up the `WHERE` clause for the `UPDATE` statement.

If you specify a combined mode (e.g.: `update,insert`) and one of the tables involved consists only of key columns, the import will revert to `insert` mode. In this case database errors during an `INSERT` are not considered as real errors and are silently ignored.

For maximum performance, choose the update strategy that will result in a succssful first statement more often. As a rule of thumb:

- Use `-mode=insert,update`, if you expect more rows to be inserted then updated.
- Use `-mode=update,insert`, if you expect more rows to be updated then inserted.

11. Copy data across databases

The `WbCopy` is essentially the command line version of the the DataPumper. For a more detailed explanation of the copy process, please refer to that section. It bascially chains a WbExport and a `WbImport` statement without the need of an intermediate data file. The `WbCopy` command requires that a connection to the source and target database can be made at the same time.

11.1. General parameters for the `WbCopy` command.

Parameter	Description
-sourceProfile	The name of the connection profile to use as the source connection. If -sourceprofile is not specified, the current connection is used as the source. If the profile name contains spaces or dashes, it has to be quoted.
-sourceGroup	If the name of your source profile is not unique across all profiles, you will need to specify the group in which the profile is located with this parameter. If the group name contains spaces or dashes, it has to be quoted.
-targetProfile	The name of the connection profile to use as the target connection. If `-targetprofile` is not specified, the current connection is used as the target. If the profile name contains spaces or dashes, it has to be quoted.
-targetGroup	If the name of your target profile is not unique across all profiles, you will need to specify the group in which the profile is located with this parameter. If the group name contains spaces or dashes, it has to be quoted.
-commitEvery	The number of rows after which a commit is sent to the target database. This parameter is ignored if JDBC batching (`-batchSize`) is used.
-deleteTarget	Possible values: `true`, `false` If this parameter is set to true, all rows are deleted from the target table before copying the data.
-mode	Defines how the data should be sent to the database. Possible values are `INSERT`, `UPDATE`, '`INSERT,UPDATE`' and '`UPDATE,INSERT`'. Please refer to the description of the WbImport command for details on.
-syncDelete	If this option is enabled `-syncDelete=true`, SQL Workbench/J will check each row from the target table if it's present in the source table. Rows in the target table that are not present in the source will be deleted. As this is implemented by checking each row individually in the source table, this can take some time for large tables. This option requires that each table in question has a primary key defined. Combined with an `UPDATE,INSERT` or `UPDATE,INSERT` mode this creates an exact copy of the source table. If more than one table is copied, the delete process is started after all inserts and updates have been processed. It is recommended to use the `-checkDependencies` parameter to make sure the deletes are processed in the correct order (which is most probably already needed to process inserts correctly). To only generate the SQL statements that would synchronize two databases, you can use the command WbDataDiff
-keyColumns	Defines the key columns for the target table. This parameter is only necessary if import is running in `UPDATE` mode. It is ignored when specifying more than one table with the `-sourceTable` argument. In that case each table must have a primary key.
-batchSize	Enable the use of the JDBC batch update feature, by setting the size of the batch queue. Any value greater than 1 will enable batch modee. If the JDBC driver supports this, the INSERT (or UPDATE) performance can be increased.

Parameter	Description
	This parameter will be ignored if the driver does not support batch updates or if the mode is not UPDATE or INSERT (i.e. if `-mode=update,insert` or `-mode=insert,update` is used).
-commitBatch	Valid values: `true`, `false` When using the `-batchSiez` parameter, the `-commitEvery` is ignored (as not all JDBC drivers support a `COMMIT` inside a JDBC batch operation. When using `-commitBatch=true` SQL Workbench/J will send a `COMMIT` to the database server after each JDBC batch is sent to the server.
-continueOnError	Defines the behaviour if an error occurs in one of the statements. If this is set to `true` the copy process will continue even if one statement fails. If set to `false` the copy process will be halted on the first error. The default value is `false`. With PostgreSQL `continueOnError` will only work, if the use of savepoints is enabled. This can be done using by setting the property `workbench.db.postgresql.import.usesavepoint=true` in in the configuration file `workbench.settings`. If this is enabled, then each INSERT (or UPDATE) statement will be "wrapped" between savepoints so that a statement error can be recovered.
-showProgress	Valid values: true, false, <numeric value> Control the update frequence in the statusbar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specifiy a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).

11.2. Copying data from one or more tables

Parameter	Description
-sourceTable	The name of the table(s) to be copied. You can either specifiy a list of tables: `-sourceTable=table1,table2`. Or select the tables using a wildcard: `-sourceTable=*` will copy all tables accessible to the user. If more than one table is specified using this parameter, the `-targetTable` parameter is ignored.
-checkDependencies	When copying more than one file into tables with foreign key constraints, this switch can be used to import the files in the correct order (child tables first). When `-checkDependencies=true` is passed, SQL Workbench/J will check the foreign key dependencies for the tables specified with `-sourceTable`
-sourceWhere	A `WHERE` condition that is applied to the source table.
-targetTable	The name of the table into which the data should be written. This parameter is ignored if more than one table is copied.
-createTarget	If this parameter is set to `true` the target table will be created, if it doesn't exist. Valid values are `true` or `false`.
-dropTarget	If this parameter is set to `true` the target table will be dropped before it is create. This parameter is ignored if `-createtarget=true` is specified.
-columns	Defines the columns to be copied. If this parameter is not specified, then all matching columns are copied from source to target. Matching is done on name **and** data type. You can either specify a list of columns or a column mapping. When supplying a list of columns, the data from each column in the source table will be copied into the corresponding (i.e. one with the same name) column in the target table. If `-createtarget=true`, then the list also defines the columns of the target table. The names have to be separated by comma: `-columns=firstname, lastname, zipcode`

Parameter	Description
	A column mapping defines which column from the source table maps to which column of the target table (if the column names do not match) If `-createtable=true` then the target table will be created from the specified target names: `-columns=firstname/surname, lastname/name, zipcode/zip` Will copy the column `firstname` from the source table to a column named `surname` in the target table, and so on. This parameter is ignored if more than one table is copied.

-preTableStatement	This parameter defines a SQL statement that should be executed before the copy process starts inserting data into the target table. The name of the current table (when e.g. importing a whole directory) can be referenced using `${table.name}`.
	
	To define a statement that should be executed after all rows have been inserted but before the data is committed, you can use the `-postTableStatement` parameter.
	
	These parameters can e.g. be used to enable identity insert for MS SQL Server:
	
	`-preTableStatement="set identity_insert ${table.name} on"`
	`-postTableStatement="set identity_insert ${table.name} off"`
	
	Errors resulting from executing these statements will be ignored. If you want to abort the copy in that case you can specify `-ignorePrePostErrors=false` and `-continueOnError=false`.

11.3. Copying data based on a SQL query

Parameter	Description
-sourcequery	The SQL query to be used as the source data (instead of a table).
-columns	The list of columns of the target table, in the order in which they appear in the source table.

11.4. Update mode

The `WbCopy` command understands the same update mode parameter as the `WbImport` command. For a discussion on the different update modes, please refer to the WbImport command.

11.5. Synchronizing tables

Using `-mode=update,insert` ensures that all rows that are present in the source table do exist in the target table and that all values for non-key columns are identical.

When you need to keep two tables completely in sync, rows that are present in the target table that do not exist in the source table need to be deleted. This is what the parameter `-syncDelete` is for. If this is enabled (`-syncDelete=true`) then SQL Workbench/J will check every row from the target table if it is present in the source table. This check is based on the primary keys of the target table and assumes that the source table as the same primary key.

Testing if each row in the target table exists in the source table is a substantial overhead, so you should enable this option only when really needed. `DELETE`s in the target table are batched according to the `-batchSize` setting of the `WbCopy` command. To increase performance, you should enable batching for the whole process.

Internally the rows from the source table are checked in chunks, which means that SQL Workbench/J will generate a `SELECT` statement that contains a `WHERE` condition for each row retrieved from the target table. The default chunk size is relatively small to avoid problems with large SQL statements. This approach was taken to minimize the number of statements sent to the server.

The automatic fallback [56] from `update,insert` or `insert,update` mode to `insert` mode applies for synchronizing tables using `WbCopy` as well.

11.6. Examples

11.6.1. Copy one table to another

```
WbCopy -sourceProfile=ProfileA
       -targetProfile=ProfileB
       -sourceTable=the_table
       -targetTable=the_other_table;
```

11.6.2. Synchronize the tables between two databases

This example will copy the data from the tables in the source database to the corresponding tables in the target database. Rows that are not available in the source tables are deleted from the target tables.

```
WbCopy -sourceProfile=ProfileA
       -targetProfile=ProfileB
       -sourceTable=*
       -mode=update,insert
       -syncDelete=true;
```

11.6.3. Copy only selected rows

```
WbCopy -sourceProfile=ProfileA
       -targetProfile=ProfileB
       -sourceTable=the_table
       -sourceWhere="lastname LIKE 'D%'
       -targetTable=the_other_table;
```

11.6.4. Copy data between tables with different columns

This example copies only selected columns from the source table. The column names in the tables do not match and a column mapping is defined. Before the copy is started all rows are deleted from the target table.

```
WbCopy -sourceProfile=ProfileA
       -targetProfile=ProfileB
       -sourceTable=person
       -targetTable=contact
       -deleteTarget=true
       -columns=firstname/surname, lastname/name, birthday/bday;
```

11.6.5. Copy data based on a SQL query

When using a query as the source for the `WbCopy` command, the columns from the query need to be matched to the columns of the target table.

```
WbCopy -sourceProfile=ProfileA
       -targetProfile=ProfileB
       -sourceQuery="SELECT firstname, lastname, birthday FROM person"
       -targetTable=contact
       -deleteTarget=true
       -columns=surname, name, bday;
```

The order in the `-columns` parameter **must** match the order in the `SELECT` statement!

12. Other SQL Workbench/J specific commands

SQL Workbench/J implements a set of SQL commands. These commands can be used like any standard SQL command (such as `UPDATE` inside SQL Workbench/J, i.e. inside the editor or as part of a SQL script that is run through SQL Workbench/J in batch mode. As those commands are implemented by SQL Workbench/J you will not be able to use them when running your SQL scripts using a different client program (e.g. `psql`, SQL*Plus or `phpmyadmin`.

12.1. Create a report of the database objects - WbReport

Creates an XML report of selected tables. This report could be used to generate an HTML documentation of the database (e.g. using the XSLT command). This report can also be generated from within the Database Object Explorer

The resulting XML file can be transformed into a HTML documentation of your database schema. Sample stylesheets can be downloaded from http://www.sql-workbench.net/xstl.html. If you have XSLT stylsheets that you would like to share, please send them to `<support@sql-workbench.net>`.

Using this command you can reverse engineer an existing database. The XML file can then be used to generate a HTML documentation of the database or to be transformed into a format that is supported by your design tool.

To see table and column comments with an Oracle database, you need to enable remarks reporting for the JDBC driver in order to see the comments.

The command supports the following parameters:

Parameter	Description
-file	The filename of the output file.
-tables	A (comma separated) list of tables to report. Default is all tables. If this parameter is specified `-schemas` is ignored. If you want to generate the report on tables from different users/schemas you have to use fully qualified names in the list (e.g. `-tables=MY_USER.TABLE1,OTHER_USER.TABLE2`) You can also specify wildcards in the table name: `-table=CONTRACT_%` will create an XML report for all tables that start with `CONTRACT_`
-schemas	A (comma separated) list of schemas to generate the report from. For each user/schema all tables are included in the report. e.g. `-schemas=MY_USER,OTHER_USER` would generate a report for all tables in the schemas `MY_USER` and `OTHER_USER`.
-namespace	The namespace to be used for the XML tags. By default no namespace is used. If you supply a value for this e.g. `wb` the tag `<schema-report>` would be written as `<wb:schema-report>`
-includeTables	Control the output of table information for the report. The default is `true`. Valid values are `true`, `false`.
-includeTableGrants	If tables are included in the output, the grants for each table can also be included with this parameter. The default value is `false`.
-includeProcedures	Control the output of stored procedure information for the report. The default is `false`. Valid values are `true`, `false`.
-includeSequences	Control the output of sequence information for the report. The default is `false`. Valid values are `true`, `false`.
-reportTitle	Defines the title for the generated XML file. The specified title is written into the tag `<report-title>` and can be used when transforming the XML e.g. into a HTML file.

12.2. Compare two database schemas - WbSchemaDiff

The `WbSchemaDiff` analyzes two schemas (or a list of tables) and outputs the differences between those schemas as an XML file. The XML file describes the changes that need to be applied to the target schema to have the same structure as the reference schema, e.g. modify column definitions, remove or add tables, remove or add indexes.

The output is intended to be transformed using XSLT (e.g. with the XSLT Command). Sample XSLT transformations can be found on the SQL Workbench/J homepage

The command supports the following parameters:

Parameter	Description
-referenceProfile	The name of the connection profile for the reference connection. If this is not specified, then the current connection is used.
-referenceGroup	If the name of your reference profile is not unique across all profiles, you will need to specify the group in which the profile is located with this parameter.
-targetProfile	The name of the connection profile for the target connection (the one that needs to be migrated). If this is not specified, then the current connection is used. If you use the current connection for reference and target, then you should prefix the table names with schema/user or use the `-referenceschema` and `-targetschema` parameters.
-targetGroup	If the name of your target profile is not unique across all profiles, you will need to specify the group in which the profile is located with this parameter.
-file	The filename of the output file. If this is not supplied the output will be written to the message area
-referenceTables	A (comma separated) list of tables that are the reference tables, to be checked.
-targetTables	A (comma separated) list of tables in the target connection to be compared to the source tables. The tables are "matched" by their position in the list. The first table in the `-referenceTables` parameter is compared to the first table in the `-targetTables` parameter, and so on. Using this parameter you can compare tables that do not have the same name. If you omit this parameter, then all tables from the target connection with the same names as those listed in `-referenceTables` are compared. If you omit both parameters, then all tables that the user can access are retrieved from the source connection and compared to the tables with the same name in the target connection.
-referenceSchema	Compare all tables from the specified schema (user)
-targetSchema	A schema in the target connection to be compared to the tables from the reference schema.
-namespace	The namespace to be used for the XML tags. By default no namespace is used. If you supply a value for this e.g. `wb` the tag `<schema-report>` would be written as `<wb:modify-table>`
-encoding	The encoding to be used for the XML file. The default is UTF-8
-includePrimaryKeys	Select whether primary key constraint definitions should be compared as well. The default is `true`. Valid values are `true` or `false`.
-includeForeignKeys	Select whether foreign key constraint definitions should be compared as well. The default is `true`. Valid values are `true` or `false`.
-includeTableGrants	Select whether table grants should be compared as well. The default is `false`.
-includeConstraints	Select whether table and column (check) constraints should be compared as well. SQL Workbench/J compares the constraint definition (SQL) as stored in the database. When comparing schemas from different DBMS systems this will not return the desired results. The default is to not compare table constraints (`false`) Valid values are `true` or `false`.
-includeViews	Select whether views should also be compared. When comparing views, the source as it is stored in the DBMS is compared. This comparison is case-sensitiv, which means `SELECT * FROM foo;` will be reported as a difference to `select * from foo;` even if they are logically the same. A comparison across different DBMS will also not work properly! The default is `true` Valid values are `true` or `false`.
-includeProcedures	Select whether stored procedures should also be compared. When comparing procedures the source as it is stored in the DBMS is compared. This comparison is case-sensitiv. A comparison across different DBMS will also not work!

Parameter	Description
	The default is `false` Valid values are `true` or `false`.
-includeIndex	Select whether indexes should be compared as well. The default is to not compare index definitions. Valid values are `true` or `false`.
-includeSequences	Select whether sequences should be compared as well. The default is to not compare sequences. Valid values are `true`, `false`.
-useJdbcTypes	Define whether to compare the DBMS specific data types, or the JDBC data type returned by the driver. When comparing tables from two different DBMS it is recommended to use `-useJdbcType=true` as this will make the comparison a bit more DBMS-independent. When comparing e.g. Oracle vs. PostgreSQL a column defined as `VARCHAR2(100)` in Oracle would be reported as beeing different to a `VARCHAR(100)` column in PostgreSQL which is not really true As both drivers ropert the column as java.sql.Types.VARCHAR, they would be considered as identical when using `-useJdbcType=true`. Valid values are `true` or `false`.

12.3. Compare data across databases - WbDataDiff

The `WbDataDiff` command can be used to generate SQL scripts that update a target database such that the data is identical to a reference database. This is similar to the `WbSchemaDiff` but compares the actual data in the tables rather than the table structure.

For each table the command will create up to three script files, depending on the needed statements to migrate the data. One file for `UPDATE` statements, one file for `INSERT` statements and one file for `DELETE` statements (if `-includeDelete=true` is specified)

As this command needs to read every row from the reference and the target table, processing large tables can take quite some time, especially if `DELETE` statements should also be generated.

`WbDataDiff` requires that all involved tables have a primary key defined. If a table does not have a primary key, `WbDataDiff` will stop the processing.

To improve performance (a bit), the rows are retrieved in chunks from the target table by dynamically constructing a WHERE clause for the rows that were retrieved from the reference table. The chunk size can be controlled using the property `workbench.sql.sync.chunksize` The chunk size defaults to 25. This is a conservative setting to avoid problems with long SQL statements when processing tables that have a PK with multiple columns. If you know that your primary keys consist only of a single column and the values won't be too long, you can increase the chunk size, possibly increasing the performace when generating the SQL statements. As most DBMS have a limit on the length of a single SQL statement, be careful when setting the chunksize too high. The same chunk size is applied when generating `DELETE` statements by the `WbCopy` command, when syncDelete mode is enabled.

The command supports the following parameters:

Parameter	Description
-referenceProfile	The name of the connection profile for the reference connection. If this is not specified, then the current connection is used.
-referenceGroup	If the name of your reference profile is not unique across all profiles, you will need to specify the group in which the profile is located with this parameter. If the profile's name is unique you can omit this parameter
-targetProfile	The name of the connection profile for the target connection (the one that needs to be migrated). If this is not specified, then the current connection is used. If you use the current connection for reference and target, then you should prefix the table names with schema/user or use the `-referenceschema` and `-targetschema` parameters.
-targetGroup	If the name of your target profile is not unique across all profiles, you will need to specify the group in which the profile is located with this parameter.

Parameter	Description
-file	The filename of the main script file. The command creates two scripts per table. One script named `update_<tablename>.sql` that contains all needed `UPDATE` or `INSERT` statements. The second script is named `delete_<tablename>.sql` and will contain all `DELETE` statements for the target table. The main script merely calls (using WbInclude) the generated scripts for each table.
-referenceTables	A (comma separated) list of tables that are the reference tables, to be checked. You can specify the table with wildcards, e.g. `-referenceTables=P%` to compare all tables that start with the letter `P`.
-targetTables	A (comma separated) list of tables in the target connection to be compared to the source tables. The tables are "matched" by their position in the list. The first table in the `-referenceTables` parameter is compared to the first table in the `-targetTables` parameter, and so on. Using this parameter you can compare tables that do not have the same name. If you omit this parameter, then all tables from the target connection with the same names as those listed in `-referenceTables` are compared. If you omit both parameters, then all tables that the user can access are retrieved from the source connection and compared to the tables with the same name in the target connection.
-referenceSchema	Compare all tables from the specified schema (user)
-targetSchema	A schema in the target connection to be compared to the tables from the reference schema.
-checkDependencies	Valid values are `true`, `false`. Sorts the generated scripts in order to respect foreign key dependencies for deleting and inserting rows. The default is `true`.
-includeDelete	Valid values are `true`, `false`. Generates `DELETE` statements for rows that are present in the target table, but not in the reference table. The default is `false`. The default is `false`.
-encoding	The encoding to be used for the SQL scripts. The default depends on your operating system. It will be displayed when you run `WbDataDiff` without any parameters. You can overwrite the platform default with the property `workbench.encoding` in the file `workbench.settings`
-sqlDateLiterals	Valid values: `jdbc`, `ansi`, `dbms`, `default` Controls the format in which the values of DATE, TIME and TIMESTAMP columns are written into the generated SQL statements. For a detailed description of the possible values, please refer to the WbExport command.
-ignoreColumns	With this parameter you can define a list of column names that should not be considered when comparing data. You can e.g. exclude columns that store the last access time of a row, or the last update time if that should not be taken into account when checking for changes.
-showProgress	Valid values: true, false, <numeric value> Control the update frequence in the statusbar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specifiy a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).

WbDataDiff Examples

Compare all tables between two connections, and write the output to the file `migrate_staging.sql`, but do not generate `DELETE` statements.

```
WbDataDiff -referenceProfile="Production"
           -targetProfile="Staging"
           -file=migrate_staging.sql
           -includeDelete=false
```

Compare a list of matching tables between two databases and write the output to the file `migrate_staging.sql` including `DELETE` statements.

```
WbDataDiff -referenceProfile="Production"
           -targetProfile="Staging"
           -referenceTables=person,address,person_address
           -file=migrate_staging.sql
           -includeDelete=true
```

Compare three tables that are differently named in the target database and ignore all columns (regardless in which table they appear) that are named `LAST_ACCESS` or `LAST_UPDATE`

```
WbDataDiff -referenceProfile="Production"
           -targetProfile="Staging"
           -referenceTables=person,address,person_address
           -targetTables=t_person,t_address,t_person_address
           -ignoreColumns=last_access,last_update
           -file=migrate_staging.sql
           -includeDelete=true
```

12.4. Run an XSLT transformation - WbXslt

Transforms an XML file via a XSLT stylesheet. This can be used to format XML input files into the correct format for SQL Workbench/J or to transform the output files that are generated by the various SQL Workbench/J commands.

Parameters for the XSLT command:

Parameter	Description
-inputfile	The name of the XML source file.
-xsltoutput	The name of the generated output file.
-stylesheet	The name of the XSLT stylesheet to be used.

12.5. Define a script variable - WbVarDef

This defines an internal variable which is used for variable substitution during SQL execution. Details can be found in the chapter Variable substitution.

The syntax for defining a variable is: `WbVarDef variable=value`

The variable definition can also be read from a file. The file should list each variable definition on one line (this is the format of a normal Java properties file). Lines beginning with a # sign are ignored. The syntax is `WBVARDEF -file=<filename>`

You can also specify a file when starting SQL Workbench/J with the parameter `-vardef=filename.ext`. When specifying a filename you can also define an encoding for the file using the `-encoding` switch. The specified file has to be a regular Java properties file. For details see see Reading variables from a file.

12.6. Delete a script variable - WbVarDelete

This removes an internal variable from the variable list. Details can be found in the chapter Variable substitution.

12.7. Show defined script variables - WbVarList

This list all defined variables from the variable list. Details can be found in the chapter Variable substitution.

12.8. Confirm script execution - WbConfirm

The `WbConfirm` command pauses the execution of the current script and displays a message. You can then choose to stop the script or continue. The message can be supplied as a parameter of the command. If no message is supplied, a default message is displayed.

This command can be used to prevent accidental execution of a script even if confirm updates is not enabled.

This command has no effect in batch mode.

12.9. Execute a SQL script - WbInclude (@)

With the `WbInclude` command you run SQL scripts without actually loading them into the editor, or call other scripts from within a script. The format of the command is `WbInclude -file=filename;`. For DBMS other then MS SQL, the command can be abbreviated using the @ sign: `@filename;` is equivalent to `WbInclude -file=filename;`. The called script way may also include other scripts. Relative filens (e.g. as parameters for SQL Workbench/J commands) in the script are always resolved to the directory where the script is located, not the current directory of the application.

The reason for excluding MS SQL is, that when creating stored procedures in MS SQL, the procedure parameters are identified using the @ sign, thus SQL Workbench/J would interpret the lines with the variable definition as the WbInclude command. If you want to use the @ command with MS SQL, you can configure this in your `workbench.settings` configuration file.

If the included SQL script contains `SELECT` queries, the result of those queries will **not** be displayed in the GUI

The long version of the command accepts additional parameters. When using the long version, the filename needs to be passed as a parameter as well.

Only files up to a certain size will be read into memory. Files exceeding this size will be processes statement by statement. In this case the automatic detection of the alternate delimiter [24] will not work. If your scripts exceed the maximum size and do use the alternate delimiter you will have to use the "long" version so that you can specify the actual delimiter used in your script.

The command supports the following parameters:

Parameter	Description
-file	The filename of the file to be included.
-continueOnError	Defines the behaviour if an error occurs in one of the statements. If this is set to `true` then script execution will continue even if one statement fails. If set to `false` script execution will be halted on the first error. The default value is `false`
-delimiter	Specify the delimiter that is used in the script. This defaults to `;`. If you want to define a delimiter that will only be recognized when beeing on a single line, append `;nl` to the value, e.g.: `-delimiter=/;nl`
-encoding	Specify the encoding of the input file. If no encoding is specified, the default encoding for the current platform (operating system) is used.
-verbose	Controls the logging level of the executed commands. `-verbose=true` has the same effect as adding a `WbFeedback on` inside the called script. `-verbose=false` has the same effect as adding a `WbFeedback off` inside of the called script.

Execute `my_script.sql`

```
@my_script.sql;
```

Execute `my_script.sql` but abort on the first error

```
wbinclude -file="my_script.sql" -continueOnError=false;
```

12.10. Handling tables or updateable views without primary keys

12.10.1. Define primary key columns - WbDefinePK

To be able to directly edit data in the result set (grid) SQL Workbench/J needs a primary key on the underlying table. In some cases these primary keys are not present or cannot be retrieved from the database (e.g. when using updateable views). To still be able to automatically update a result based on those tables (without always manually defining the primary key) you can manually define a primary key using the `WbDefinePk` command.

Assuming you have an updateable view called `v_person` where the primary key is the column `person_id`. When you simply do a `SELECT * FROM v_person`, SQL Workbench/J will prompt you for the primary key when you try to save changes to the data. If you run

```
WbDefinePk v_person=person_id
```

before retrieving the result, SQL Workbench/J will automatically use the `person_id` as the primary key (just as if this information had been retrieved from the database).

To delete a definition simply call the command with an empty column list:

```
WbDefinePk v_person=
```

If you want to define certain mappings permanently, this can be done using a mapping file that is specified in the configuration file. The file specified has to be a text file with each line containing one primary key definition in the same format as passed to this command. The global mapping will automatically be saved when you exit the application if a filename has been defined. If no file is defined, then all PK mappings that you define are lost when exiting the application (unless you explicitely save them using WbSavePkMap

```
v_person=person_id
 v_data=id1,id2
```

will define a primary key for the view `v_person` and one for the view `v_data`. The definitions stored in that file can be overwritten using the `WbDefinePk` command, but those changes won't be saved to the file. This file will be read for all database connections and is not profile specific. If you have conflicting primary key definitions for different databases, you'll need to execute the `WbDefinePk` command each time, rather then specifying the keys in the mapping file.

When you define the key columns for a table through the GUI, you have the option to remember the defined mapping. If this option is checked, then that mapping will be added to the global map (just as if you had executed `WbDefinePk` manually.

The mappings will be stored with lowercase table names internally, regardless how you specify them.

12.10.2. List defined primary key columns - WbListPKDef

To view the currently defined primary keys, execute the command `WbListPkDef`.

12.10.3. Load primary key mappings - WbLoadPKMap

To load the additional primary key definitions from a file, you can use the the `WbLoadPKMap` command. If a filename is defined in the configuration file then that file is loaded. Alternatively if no file is configured, or if you want to load a different file, you can specify the filename using the `-file` parameter.

12.10.4. Save primary key mappings - WbSavePKMap

To save the current primary key definitions to a file, you can use the the `WbSavePKMap` command. If a filename is defined in the configuration file then the definition is stored in that file. Alternatively if no file is configured, or if you want to store the current mapping into a different file, you can specify the filename using the `-file` parameter.

12.11. Extracting BLOB content - WbSelectBlob

To save the contents of a `BLOB` or `CLOB` column into an external file the `WbSelectBlob` command can be used. Most DBMS support reading of `CLOB` (character data) columns directly, so depending on your DBMS (and JDBC driver) this command might only be needed for binary data.

The syntax is very similar to the regular `SELECT` statement, an additional `INTO` keyword specifies the name of the external file into which the data should be written:

```
WbSelectBlob blob_column
INTO c:/temp/image.bmp
FROM theTable
WHERE id=42;
```

Even if you specify more then one column in the column list, SQL Workbench/J will only use the first column. If the SELECT returns more then one row, then one outputfile will be created for each row. Additional files will be created with a counter indicating the row number from the result. In the above example, image.bmp, image_1.bmp, image_3.bmp and so on, would be created. If you want to export addtional columns together with the BLOB contents, please use the WbExport command.

You can fully manipulate (save, view, upload) the contents of BLOB columns in a result set. Please refer to BLOB support for details.

12.12. Enable Oracle's DBMS_OUTPUT package - ENABLEOUT

This command enables the `DBMS_OUTPUT` package when connected to an Oracle database. On other systems this command does nothing. After the `DBMS_OUTPUT` package is enabled, any message written with dbms_output.put_line() are displayed in the message pane after executing a SQL statement. It is equivalent to calling the dbms_output.enable() procedure.

You can control the buffer size of the `DBMS_OUTPUT` package by passing the desired buffer size as a parameter to the `ENABLEOUT` command:

```
ENABLEOUT 32000;
```

Due to a bug in Oracle's JDBC driver, you cannot retrieve columns with the `LONG` or `LONG RAW` data type if the `DBMS_OUTPUT` package is enabled. In order to be able to display these columns support for `DBMS_OUTPUT` has to be switched off.

12.13. Disable Oracle's DBMS_OUTPUT package - DISABLEOUT

This disables the `DBMS_OUTPUT` package for an Oracle database. This is equivalent to calling dbms_output.disable() procedure.

12.14. Control feedback messages - WbFeedback

Normally SQL Workbench/J prints the results for each statement into the message panel. As this feedback can slow down the execution of large scripts, you can disable the feedback using the `WbFeedback` command. When `WbFeedback OFF` is executed, only a summary of the number of executed statements will be displayed, once the script execution has finished. This is the same behaviour as selecting "Consolidate script log" in the options window. The only difference is, that the setting through `WbFeedback` is temporary and does not affect the global setting.

12.15. Setting connection properties - SET

The `SET` command is available to enable you to run SQL scripts that are designed to run with Oracle's SQL*Plus utility inside SQL Workbench/J as well. Most of the parameters of the `SET` are only valid inside SQL*Plus, and thus for Oracle any error message resulting from executing a `SET` command will only be logged as a warning. For all other DBMS the command is passed directly to the server, except for the parameters described in this chapter (because they have an equivalent JDBC call that will be executed instead).

12.15.1. FEEEDBACK

`SET feedback ON/OFF` is equivalent to the WbFeedback command, but mimics the syntax of Oracle's SQL*Plus utility.

12.15.2. SERVEROUTPUT

`SET serveroutput on` is equivalent to the ENABLEOUT command and `SET serveroutput off` is equivalent to DISABLEOUT command.

12.15.3. AUTOCOMMIT

With the command `SET autocommit ON/OFF` autocommit can be turned on or off for the current connection. This is equivalent to setting the autocommit property in the connection profile or toggling the state of the SQL » Autocommit menu item.

12.16. Show table structure - DESCRIBE

Describe shows the definition of the given table. It can be abbreviated with DESC. The command expects the table name as a parameter.

```
DESC person;
```

If you want to show the structure of a table from a different user, you need to prefix the table name with the desired user

```
DESCRIBE otheruser.person;
```

12.17. List tables - WbList

This command lists all available tables (including views and synonyms). This output is equivalent to the left part of the Database Object Explorer's Table tab.

12.18. List stored procedures - WbListProcs

This command will list all stored procedures available to the current user. The output of this command is equivalent to the Database Explorer's Procedure tab.

12.19. List catalogs - WbListCat

Lists the available catalogs or databases. The output of this command depends on the underlying JDBC driver and DBMS. For MS SQL Server this lists the available databases (which then could be changed by USE <dbname>)

For Oracle this command returns nothing (as Oracle does not implement the concept of catalogs)

This command calls the JDBC driver's `getCatalogs()` method and will return its result. If on your database system this command does not display a list, it is most likely that your DBMS does not support catalogs (e.g. Oracle) or the driver does not implement this feature.

13. DataPumper

13.1. Overview

The export and import features are useful if you cannot connect to the source and the target database at once. If your source and target are both reachable at the same time, it is more efficient to use the DataPumper to copy data between two systems. With the DataPumper no intermediate files are necessary. Especially with large tables this can be an advantage.

To open the DataPumper, select Tools » DataPumper

The DataPumper lets you copy data from a single table (or SELECT query) to a table in the target database. The mapping between source columns and target columns can be specified as well

Everything that can be done with the DataPumper, can also be accomplished with the `WbCopy` command. The DataPumper can also generate a script which executes the `WbCopy` command with the correct parameters according to the current settings in the window. This can be used to create scripts which copy several tables.

The DataPumper can also be started as a stand-alone application - without the main window - by specifying -datapumper=true in the command line when starting SQL Workbench/J. You can also use the supplied Windows executable `DataPumper.exe` or the Linux/Unix shell script `datapumper`

When opening the DatPumper from the main window, the main window's current connection will be used as the initial source connection. You can disable the automatic connection upon startup with the property `workbench.datapumper.autoconnect` in the `workbench.settings` file.

13.2. Selecting source and target connection

The DataPumper window is divided in three parts: the upper left part for defining the source of the data, the upper right part for defining the target, and the lower part to adjust various settings which influence the way, the data is copied.

After you have opened the DataPumper window it will automatically connect the source to the currently selected connection from the main window. If the DataPumper is started as a separate application, no initial connection will be made.

To select the source connection, press the ellipsis right next to the source profile label. The standard connection dialog will appear. Select the connection you want to use as the source, and click OK. The DataPumper will then connect to the database. Connecting to the target database works similar. Simply click on the ellipsis next to the target profile box.

Instead of a database connection as the source, you can also select a text or XML file as the source for the DataPumper. Thus it can also be used as a replacement of the WbImport command.

The dropdown for the target table includes an entry labelled "(Create new table)". For details on how to create a new table during the copy process please refer to the advanced tasks section.

After source and target connection are established you can specify the tables and define the column mapping between the tables.

13.3. Copying a complete table

To copy a single table select the source and target table in the dropdowns (which are filled as soon as the connection is established)

After both tables are selected, the middle part of the window will display the available columns from the source and target table. This grid display represents the column mapping between source and target table.

13.3.1. Mapping source to target columns

Each row in the display maps a source column to a target column. Initially the DataPumper tries to match those columns which have the same name and data type. If no match is found for a target column, the source column will display `(Skip target column)` This means that the column from the target table will not be included when inserting data into the target table (technically speaking: it will be excluded from the column list in the INSERT statement).

13.3.2. Restricting the data to be copied

You can restrict the number of rows to be copied by specifying a `WHERE` clause which will be used when retrieving the data from the source table. The `WHERE` clause can be entered in the SQL editor in the lower part of the window.

13.3.3. Deleting all rows from the target table

When you select the option "Delete target table", all rows from the target table will be deleted before the copy process is started. This is done with a `DELETE FROM <tablename>;` When you select this option, make sure the data can be deleted in this way, otherwise the copy process will fail.

The `DELETE` will not be committed right away, but at the end of the copy process. This is obviously only of interest if the connection is not done with autocommit = true

13.3.4. Continuing when an insert fails

In some cases inserting of individual rows in the target table might fail (e.g. a primary key violation if the table is not empty). When selecting the option "Continue on error", the copy process will continue even if rows fail to insert

13.3.5. Committing changes

By default all changes are committed at the end, when all rows have been copied. By supplying a value in the field "Commit every" SQL Workbench/J will commit changes every time the specified number of rows has been inserted into the target. When a value of 50 rows has been specified, and the source table contains 175 rows, SQL Workbench/J will send 4 `COMMIT`s to the target database. After inserting row 50, row 100, row 150 and after the last row.

13.3.6. Batch execution

If the JDBC driver supports batch updates, you can enable the use of batch updates with this checkbox. The checkbox will be disabled, if the JDBC driver does not support batch updates, or if a combined update mode (insert,update, update,insert) is selected.

Batch execution is only available if either INSERT or UPDATE mode is selected.

13.3.7. Update mode

Just like the WbImport and WbCopy commands, the data pumper can optionally update the data in the target table. Select the approriate update strategy from the `Mode` drop down. The DataPumper will use the key columns defined in the column mapper to generate the `UPDATE` command. When using update you have to select at least one key column.

You cannot use the update mode, if you select **only** key columns, The values from the source are used to build up the `WHERE` clause for the `UPDATE` statement. If ony key columns are defined, then there would be nothing to update.

For maximum performance, choose the update strategy that will result in a succssful first statement more often. As a rule of thumb:

- Use `-mode=insert,upadte`, if you expect more rows to be inserted then updated.
- Use `-mode=update,insert`, if you expect more rows to be updated then inserted.

13.4. Advanced copy tasks

13.4.1. Populating a column with a constant

To populate a target column with a constant value. The name of the source columns can be edited in order to supply a constant value instead of a column name. Any expression understood by the source database can be entered there. Note that if (Skip target column) is selected, the field cannot be edited.

13.4.2. Creating the target table

You can create the target table "on the fly" by selecting `(Create target table)` from the list of target tables. You will be prompted for the name of the new table. If you later want to use a different name for the table, click on the button to the right of the drop down.

The target table will be created without any primary key definitions, indexes of foreign key constraints.

The DataPumper tries to map the column types from the source columns to data types available on the target database. For this mapping it relies on information returned from the JDBC driver. The functions used for this may not be implemented fully in the driver. If you experience problems during the creation of the target tables, please create the tables manually before copying the data. It will work best if the source and target system are the same (e.g. PostgreSQL to PostgreSQL, Oracle to Oracle, etc).

Most JDBC drivers map a single JDBC data type to more then one native datatype. MySql maps its `VARCHAR`, `ENUM` and `SET` type to java.sql.Types.VARCHAR. The DataPumper will take the first mapping which is returned by the driver and will ignore all subsequent ones. Any datatype that is returned twice by the driver is logged as a warning in the log file. The actual mappings used, are logged with type INFO.

If you want the datapumper to use a different native data type (and not the first match returned by the driver), you can specify a list of types that should completely be ignored during the mapping process. This can be specified in `workbench.settings`. The following keys are recognized:

Property key	**Description**
workbench.ignoretypes.postgres	Datatypes to be ignored for PostgreSQL. The default list is name,text.
workbench.ignoretypes.firebird	Datatypes to be ignored for FirebirdSQL
workbench.ignoretypes.hsqldb	Datatypes to be ignored for HSQLDB
workbench.ignoretypes.oracle	Datatypes to be ignored for Oracle
workbench.ignoretypes.sqlserver	Datatypes to be ignored for MS SQL Server
workbench.ignoretypes.mysql	Datatypes to be ignored for MySQL. The default list is ENUM,SET
workbench.ignoretypes.other	Datatypes to be ignored for all other DBMS'

The case of the datatype names is relevant, they should be entered the same way as they are reported in the log file!

If you want to override the default settings with an "empty" list, remove the value from the key (`workbench.ignoretypes.hsqldb=`) or use the value `no-ignore-list`.

13.4.3. Using a query as the source

If you want to copy the data from several tables into one table, you can use a SELECT query as the source of your data. To do this, select the option `Use SQL query as source` below the SQL editor. After you have entered you query into the editor, click the button Retrieve columns from query. The columns resulting from the query will then be put into the source part of the column mapping. Make sure, the columns are named uniquely when creating the query. If you select columns from different tables with the same name, make sure you use a column alias to rename the columns.

Creating the target table "on the fly" is not available when using a SQL query as the source of the data

14. Database Object Explorer

The Database Object Explorer displays the available database objects such as Tables, Views, Triggers and Stored Procedures.

There are three ways to start the DbExplorer

Using Tools » Database Explorer.
Passing the paramter `-dbexplorer` to the main program (sqlworkbench.sh or SQLWorkbench.exe)
When using Windows with the `DbExplorer.exe` executable or in Linux/Unix using shell script `dbexplorer.sh`

At the top of the window, the current schema (for MS SQL Server this would be the current database, for Oracle this is the user) can be selected.

The displayed list can be filtered using the quick filter above the list. To filter the list by the object name, simply enter the criteria in the filter field, and press `ENTER` or click the filter icon  . The criteria field will list the last 25 values that were entered in the dropdown. If you want to filter based on a different column of the list, right-click on the criteria field, and select the desired column from the `Filtercolumn` menu item of the popup menu. The same filter can be applied on the `Procedures` tab.

The list of tables can be pre-filtered to remove unwanted entries such as tables that have been deleted and now reside in Oracle's "Recycle Bin". The filtering is done through a regular expression on a per-database basis. By default this is only defined for Oracle and will filter out any table that starts with `BIN$`.

For Oracle (and other DBMS that support them), synonyms are displayed as well. You can filter out unwanted synonyms by specifying a regular expression in your `workbench.settings` file. This filter will also be applied when displaying the list of available tables when opening the command completion popup.

The first tab displays the structure of tables and views. The type of object displayed can be chosen from the drop down right above the table list. This list will be returned by the JDBC driver, so the available "Table types" can vary from DBMS to DBMS.

The menu item Database Explorer will either display the explorer as a new window, or a new panel, depending on the system options.

You can always open additional DbExplorer tabs or windows using

Tools » New DbExplorer panel.
Tools » New DbExplorer window.

14.1. Objects tab

The object list displays tables, views, sequences and synonyms (basically anyhting apart from procedures or functions). The context menu of the list offers several additional functions:

Export data

This will execute a `WbExport` command for the currently selected table(s). Choosing this option is equivalent to do a `SELECT * FROM table;` and then executing SQL » Export query result from the SQL editor in the main window. See the description of the WbExport command for details.

Put SELECT into

This will put a SELECT statement into the SQL editor to display all data for the selected table. You can choose into which editor tab the statement will be written. The currently selected editor tab is displayed in bold (when displaying the DbExplorer in a separate window. You can also put the generated SQL statement into a new editor tab, by selecting the item New tab

Create empty INSERT

This creates an empty `INSERT` statement for the currently selected table(s). This is intended for programmers that want to use the statement inside their code.

Create default SELECT

This creates a `SELECT` for the selected table(s) that includes all columns for the table. If you want to generate a SELECT statement to actually retreive data from within the editor, please use the `Put SELECT into` option.

Create DDL Script

With this command a script for multiple objects can be created. Select all the tables, views or other objects in the table list, that you want to create a script for. Then right click and select "Create DDL Script". This will generate one script for all selected items in the list.

When this command is selected, a new window will be shown. The window contains a statusbar which indicates the object that is currently processed. The complete script will be shown as soon as all objects have been processed. The objects will be processed in this order:

SEQUENCEs
TABLEs
VIEWs
SYNONYMS

Create schema report

This will create an XML report of the selected tables. You will be prompted to specify the location of the generated XML file. This report can also be generated using the WbReport command.

Drop

Drops the selected objects. If at least one object is a table, and the currently used DBMS supports cascaded dropping of constraints, you can enable cascaded delete of constraints. If this option is enabled SQL Workbench/J would generate e.g. for Oracle a `DROP TABLE mytable CASCADE CONSTRAINTS`. This is necessary if you want to drop several tables at the same time that have foreign key constraints defined.

Delete data

Deletes all rows from the selected table(s) by executing a `DELETE FROM table_name;` to the server for each selected table. If the DBMS supports `TRUNCATE` then this can be done with `TRUNCATE` as well. Using `TRUNCATE` is usually faster as no transation state is maintained.

The list of tables is sorted according to the sort order in the table list. If the tables have foreign key constraints, you can re-order them to be processed in the correct order by clicking on the Check foreign keys button.

14.2. Table details

When a table is selected, the right part of the window will display its column definition, the SQL statement to create the table, any index defined on that table (only if the JDBC driver returns that information), other tables that are referenced by the currently selected table, any table that references the currently selected table and any trigger that is defined on that table.

The column list will also display any comments defined for the column (if the JDBC driver returns the information). Oracle's JDBC driver does not return those comments by default. To enable the display of column comments (remarks) you have to supply an extended property in your connection profile. The property's name should be `remarksReporting` and the value should be set to `true`.

If the DBMS supports synonyms, the columns tab will display the column definition of the underlying table or view. The source tab will display the statement to re-create the synonym. If the underlying object of the synonym is a table, then indexes, foreign keys and triggers for that table will be displayed as well. Note that if the synonym is for a view, those tabs will still be displayed, but will not show any information.

14.3. Table data

The data tab will display the data from the currently selected table. There are several options to configure the display of this tab. The `Autoload` check box, controls the retrieval of the data. If this is checked, then the data will be retrieved from the database as soon as the table is selected in the table list (and the tab is visible).

The data tab will also display a total row count of the table. As this display can take a while, the automatic retrieval of the row count can be disabled. To disable the automatic calculation of the table's rowcount, click on the Settings button and deselect the checkbox `Autoload table row count`. To calculate the table's row count when this is not done automatically, click on the `Rows` label. You can cancel the row count retrieval while it's running by clicking on the label again.

The data tab is only available if the currently selected objects is recognized as an object that can can be "SELECTED". Which object types are included can be defined in the settings for SQL Workbench/J See selectable object types for details.

You can define a maximum number of rows which should be retrieved. If you enter 0 (zero) then all rows are retrieved. Limiting the number of rows is useful if you have tables with a lot of rows, where the entire table would not fit into memory.

In addition to the max rows setting, a second limit can be defined. If the total number of rows in the table exceeds this second limit, a warning is displayed, whether the data should be loaded.

This is useful when the max rows parameter is set to zero and you accidently display a table with a large number of rows.

If the automatic retrieval is activated, then the retrieve of the data can be prevented by holding down the Shift key while switching to the data tab.

The data in the tab can be edited just like the data in the main window. To add or delete rows, you can either use the buttons on the toolbar in the upper part of the data display, or the popup menu. To edit a value in a field, simply double click that field, start typing while the field has focus (yellow border) or hit F2 while the field has focus.

14.4. View details

When a database `VIEW` is selected in the object list the right will display the columns for the view, the source and the data returned by a select from that view.

The data details tab works the same way as the data tab for a table. If the view is updateable (depends on the view definition and the underlying DBMS) then the data can also be changed within the data tab

The source code is retrieved by customized SQL queries (this is not supported by the JDBC driver). If the source code of views is not displayed for your DBMS, please contact `<support@sql-workbench.net>`.

14.5. Procedure tab

The procedure tab will list all stored procedures and functions stored in the current schema. For procedures or functions returning a result set, the definition of the columns will be displayed as well.

To display the procedure's source code SQL Workbench/J uses its own SQL queries. For most popular DBMS systems the necessary queries are built into the application. If the procedure sourc is not displayed for your DBMS, please contact the author.

Functions inside Oracle packages will be listed separately on the left side, but the source code will contain all functions/ procedures from that package.

14.6. Search tables

This tab offers the ability to search for a value in all text columns of all tables which are selected. The results will be displayed on the right side of that tab. The result will always display the complete row where the search value was found. Any column that contains the entered value will be highlighted.

The value will be used to create a `LIKE 'value'` restriction for each text column on the selected tables. Therefore the value should contain a wildcard, otherwise the exact expression will be searched.

You can apply a function to each column as well. This is useful if you want to to do a case insensitive search on Oracle (Oracles VARCHAR comparison is case sensitive). In the entry field for the column the placeholder col is replaced with the actual column name during the search. To do a case insensitive search in Oracle, you would enter lower(col) in the column field and '%test%' in the value field.

The expression in the column field is sent to the DBMS without changes, except the replacement of col with the current column name. The above example would yield a `lower(<column_name>) like '%test%'` for each text column for the selected tables.

The generated SQL statements are logged in the second tab, labelled `SQL Statements`.

In the resulting tables, SQL Workbench/J tries to highlight those columns which match the criteria. This might not always work, if you apply a function to the column itself such as `to_upper()` SQL Workbench/J does not know that this will result in a case-insesitive search on the database. SQL Workbench/J tries to guess if the given function/value combination might result in a case insensitive search (especially on a DBMS which does a case sensitive search by default) but this might not work in all the cases and for all DBMS.

15. Configuring keyboard shortcuts

You can configure the keyboard shortcut to execute a specific action (=menu item) in the dialog which is displayed when you select Tools » Configure shortcuts.... The dialog lists the available actions together with their configured shortcut and their default shortcut.

15.1. Assign a shortcut to an action

To assign a (new) keyboard combination for a specific action, select (highlight) the action in the list and click on the Assign button. A small window will pop up, where you can press the key combination which you would like to assign to that action. Note that only F-Keys (F1, F2, ...) can be used without a modifier (Shift, Control, Alt). All other keys need be pressed together with one of the modifier keys.

After you have entered the desired keyboard shortcut, press the OK button. If the shortcut is already assigned to a different action, you will be prompted, if you want to override that definition. If you select to overwrite the shortcut for the other action, that action will then have **no** shortcut assigned

15.2. Removing a shortcut from an action

To remove a shortcut completely from an action, select (highlight) that action, and click on the Clear button. Once the shortcut has been cleared, the action is no longer accessible through a shortcut (only through the menu).

15.3. Reset to defaults

If you want to reset the shortcut for a single action to its default, select (highlight) the action in the list, and click on the Reset button. To reset all shortcuts click on the Reset all button.

16. Common problems

16.1. Oracle Problems

16.1.1. Error: "Stream has already been closed"

Due to a bug in Oracle's JDBC driver, you cannot retrieve columns with the `LONG` or `LONG RAW` data type if the `DBMS_OUTPUT` package is enabled. In order to be able to display these columns, the support for `DBMS_OUTPUT` has to be switched off using the DISABLEOUT command before runnnig a `SELECT` statement that returns `LONG` or `LONG RAW` columns.

16.1.2. BLOB support is not working properly

SQL Workbench/J supports reading and writing BLOB data in various ways. The implementation relies on standard JDBC API calls to work properly in the driver. If you experience problems when updating BLOB columns (e.g. using the enhanced `UPDATE, INSERT` syntax or the DataPumper) then please check the version of your Oracle JDBC driver. Only 10.x drivers implement the necessary JDBC functions properly. The version of your driver is reported in the log file when you make a connection to your Oracle server.

16.1.3. Table and column comments are not displayed

By default Oracle's JDBC driver does not return comments made on columns or tables (`COMMENT ON ..`). Thus your comments will not be shown in the database explorer.

To enable the display of column comments, you need to pass the property `remarksReporting` to the driver.

In the profile dialog, click on the Extended Properties button. Add a new property in the following window with the name `remarksReporting` and the value `true`. Now close the dialog by clicking on the OK button.

Turning on this features slows down the retrieval of table information e.g. in the Database Explorer.

When you have comments defined in your Oracle database and use the WbReport command, then you have to enable the remrks reporting, otherwise the comments will not show up in the report.

16.2. MySQL Problems

16.2.1. "Operation not allowed" error message

In case you receive an error message "`Operation not allowed after ResultSet closed`" please upgrade your JDBC driver to a more recent version. This problem was fixed with the MySQL JDBC driver version 3.1. So upgrading to that or any later version will fix this problem.

16.2.2. Problems with zero dates with MySQL

MySQL allows the user to store invalid dates in the database (0000-00-00). Since version 3.1 of the JDBC driver, the driver will throw an exception when trying to retrieve such an invalid date. This behaviour can be controlled by adding an extended property to the connection profile. The property should be named `zeroDateTimeBehavior`. You can set this value to either `convertToNull` or to `round`. For details see the MySQL site

16.3. SQL Server Problems

16.3.1. Can't start a cloned connection while in manual transaction mode

This error usually occurs in the DbExplorer if an older Microsoft JDBC Driver is used and the connection does not use autocommit mode. There are two ways to fix this problem:

- Upgrade to a newer Microsoft driver (e.g. the one for SQL Server 2005)

- Enable autocommit in the connection profile
- Add the parameter `;SelectMethod=Cursor` to your JDBC URL

This article in Microsoft's Knowledgebase gives more information regarding this problem.

The possible parameters for the SQL Server 2005 driver are listed here

16.4. DB2 Problems

16.4.1. "Connection closed" errors

When using the DB2 JDBC drivers it is important that the `charsets.jar` is part of the used JDK (or JRE). Apparently the DB2 JDBC driver needs this library in order to correctly convert the EBCDIC characterset (used in the database) into the Unicode encoding that is used by Java. The library `charsets.jar` is usually included in all multi-language JDK/JRE installations.

If you experience intermittent "Connection closed" errors when running SQL statements, please verify that `charsets.jar` is part of your JDK/JRE installation. This file is usually installed in `jre\lib\charsets.jar`.

17. Options dialog

The options dialog enables you to influence the behaviour and look of SQL Workbench/J to meet your needs. To open the options dialog choose Tools » Options.

17.1. General options

17.1.1. Check for updates

With this option you can enable an automatic update check when SQL Workbench/J is started. You can define the interval in days after which the application should check for updates on the homepage. If a newer version is found on the website this will be indicated with a little globe in the statusbar. Clicking on the icon will open your default internet browser with the application's homepage.

If you disable this option, you can manually check for updates using the menu Help » Check for updates....

17.1.2. Language

With this option you can select in which language the application is shown. The new value will only be in affect when you restart the application.

17.1.3. Encrypt passwords

If this option is enabled, the password stored within a connection profile will be encrypted. Whether the password should be stored at all can be selected in the profile itself.

Using this option only supplies very limited security. As the source code for SQL Workbench/J is freely available, the algorithm to decrypt the passwords stored in this way can easily be extracted to retrieve the plain text passwords.

17.1.4. Consolidate script log

Usually SQL Workbench/J reports the success and timings for each statement that is beeing executed in the message tab of the current SQL panel. For large scripts this can slow down script execution dramatically. If this option is enabled, only a summary of the execution is printed once the script has finished. You can turn off the log during script execution by using the WBFEEDBACK command.

17.1.5. Enable animated icons

Enable or disable the use of an animated icons in the SQL editor to indicate a running SQL statement. It has been reported, that the animated icon does have a severe (negativ) impact on the performance on some computers (depending on JDK/OS/Graphics driver). If you experience a high CPU usage during the execution of SQL statements, or if you find your SQL statements are running very slow, try to turn off the usage of the animated icons.

17.1.6. Standard font

The standard font that is used for menus, lables, buttons etc.

17.1.7. Message font

The font that is used in the message pane of the SQL window.

17.1.8. Field delimiter

The default delimiter used when exporting data to a text file. This can be overridden in the file selection dialog when actually exporting the data.

17.1.9. Quote character

The default character to be used to quote column data when exporting to a text file. This value will be used when you don't specify a quote character e.g. for the `WbExport` command and when you use SQL » Save data as to export a result set.

17.1.10. PDF Reader

If you want to display the PDF manual from inside the application, you need to specify the full path to your PDF reader application. If this is defined and the file `SQLWorkbench-Manual.pdf` is available in the directory where `sqlworkbench.jar` is located, you will be able to display the PDF manual by selecting Help » Manual.

17.1.11. Log Level

With this option you can control the level of information written to the application log. The most verbose level is `DEBUG`. With `ERROR` only severe errors (either resulting from running a user command or from an internal error) are written to the application log.

17.2. Editor options

17.2.1. Auto jump next statement

If this option is enabled, then the cursor will automatically jump to the next statement in the script, when you execute a single statement using **Ctrl-Enter** ("Run current statement"). This can also be toggled through the menu SQL » Jump to next statement

For more information on how you can execute statements in the editor, please refer to Executing Statements

17.2.2. Right click behaviour in the editor

Normally a right click in the SQL editor does not change the location of the cursor (caret). If this option is checked, then a right click will also change the caret's location (to where the mouse cursor is located)

17.2.3. Close completion with search

When using the quicksearch feature in the code completion this option controls the behaviour when hitting the ESC key. If this option is enabled, the ESC key will also close the popup window with the available choices. If this option is disabled, the ESC key will only close the quicksearch *input* field.

17.2.4. Paste completion in

With this option you can select how the selected object name from the code completion popup is pasted into the editor. `As is` means, that the values will be inserted into the editor as it was retrieved from the database. This option will also be used when SQL statements are generated internally (e.g. for updating the result set or when you export/copy data as SQL statements)

17.2.5. Sort pasted columns by

When selecting to paste all (or several columns) from the popup window, you can select with this option, in which order the columns should be written into the editor.

17.2.6. Line ending for DBMS

This property controls the line terminator used by the editor when sending SQL statements to the database. The value "Platform default" relates to the platform where you run SQL Workbench/J this is not the platform of the DBMS server.

The editor always uses "unix" line ending internally. If you select a different value for this property, SQL Workbench/J will convert the SQL statements to use the desired line ending before sending them to the DBMS. As this can slow down the execution of statements, it is highly recommended to leave the default setting of Unix line endings. You should only change this, if your DBMS does not understand the single linefeed character (ASCII value 10) properly.

17.2.7. File format

This property controls the line terminator used when a file is saved by the editor. Changing this property affects the next save operation.

17.2.8. Editor font

The font that is used in the SQL editor. This font is also used when displaying the SQL source for tables and other database objects in the DbExplorer.

17.2.9. Error highlight color

When a statement is not executed correctly (and the DBMS signals an error) it is highlighted in the editor. With this option you can select the color that is used to highlight the incorrect statement.

17.2.10. Selection color

The color that is used to highlight selected text.

17.2.11. Alternate Delimiter

This options defines the default alternate delimiter. You can override this default in the connection profile, to use different delimiters for different DBMS. For details see using the alternate delimiter

17.2.12. History size

The number of statements per tab which should be stored in the statement history. Remember that always the full text of the editor (together with the selection and cursor information) is stored in the history. If you have large amounts of text in the editor and set this number quite high, be aware of the memory consumption this might create.

17.2.13. Files in history

If this option is enabled, the content of external files is also stored in the statement history.

17.2.14. Electric scroll

Electric scrolling is the automatic scrolling of the editor when clicking into lines close to the upper or lower end of the editor window. If you click inside the defined number of lines at the upper or lower end, then the editor will scroll this line into the center of the visible area. The default is set to 3, which means that if you click into (visible) line 1,2 or 3 of the editor, this line will be centered in the display.

17.2.15. Editor tab width

The number of spaces that are assumed for the `TAB` character.

17.2.16. Additional word characters

The editor recognizes character sequences that consist of letters and characters only as "words". This influences the way word by word jumping is done, or when selecting text using a doubleclick. Every character that is entered for this option is considered a "word" character and thus does not mark a word boundary.

By putting e.g. an underscore into this field, the text `MY_TABLE` is recognized as a single word instead of two words (which is the default).

17.3. Options for displaying data

17.3.1. Data font

The font that is used to display result sets. This includes the object list and results in the DbExplorer.

17.3.2. Date, timestamp and time formats

Define the format for displaying date, date/time (timestamp) and time columns in the result set. For details on the format of this option, please refer to the documentation of the `SimpleDateFormat` class. This format is also used when parsing input for date or timestamp fields, so if you enter a date while editing the data, make sure you enter it the same way as defined with this option.

Here is an overview of the letters and their meaning that can be used to format the date and timestamp values. Be aware that case matters!

Letter	Description
G	Era designator (Text, e.g. AD)
y	Year (Number)
M	Month in year (Number)
w	Week in year (Number)
W	Week in month (Number)
D	Day in year (Number)
d	Day in month (Number)
F	Day of week in month (Number)
E	Day in week (Text)
a	AM/PM marker
H	Hour in day (0-23)
k	Hour in day (1-24)
K	Hour in am/pm (0-11)
h	Hour in am/pm (1-12)
m	Minute in hour
s	Second in minute
S	Milliseconds
z	General time zone (e.g. Pacific Standard Time; PST; GMT-08:00)
Z	RFC 822 time zone (e.g. -0800)

17.3.3. Decimal symbol

The character which is used as the decimal separator when displaying numbers.

17.3.4. Decimal digits

Define the maximum number of digits which will be displayed for numeric columns. This only affects the display of the number internally they are still stored as the DBMS returned them. To see the internal value, leave the mouse cursor over the cell. The tooltip which is displayed will contain the number as it was returned by the JDBC driver. When exporting data or copying it to the clipboard, the real value will be used.

17.3.5. Alternate row colors

If this option is selected, the rows in the data table will be displayed with alternating background color. You can choose the alternate color (the other color is defined by the used Look & Feel) with the font chooser next to the checkbox.

17.3.6. Allow row height resizing

If this option is enabled, you will be able to resize individual rows in the result set.

17.3.7. Auto adjust column widths

If this option is enabled, the widths of the result set columns are automatically adjusted to fit the largest value (respecting the min. and max. size settings)

17.3.8. Consider column headers

When calculating the optimal width for a column (either manually or if "Auto adjust column widths" is enabled, then the column's label will be included in the width calculation if this option is enabled. If this option is disabled, and the column contains very short values, the column width could be smaller than the column's label.

17.3.9. Max. column width

When the initial display size of a column is calculated, or if you optimize the column widths to fit the actual data, columns will not exceed this width. This is useful when displaying large character columns.

17.3.10. Min. column width

When the initial display size of a column is calculated, or if you optimize the column widths to fit the actual data, columns will not exceed this width.

17.4. Options for data editing

17.4.1. Confirm result set updates

When this option is enabled, the statements which are sent to the database when saving changes to result set table, are displayed before execution. The update can be cancelled at that point if the statements are not correct. The generated statements can also be saved to a file from that window.

The statement(s) that are displayed in the confirmation window can not be changed!

17.4.2. Highlight required fields

When editing data either in the result set or in the data tab of the DbExplorer, fields that are set to `NOT NULL` in the underlying table, will be displayed with a different background color if this option is selected.

17.4.3. Color for required fields

If required fields are highlighted during editing, this option defines the background color that is used.

17.4.4. Default PK Map

This property defines a mapping file for primary key columns. The information from that file is read whenever the primary keys for a table of cannot be obtained from the database. For a detailed description on how to define extra primary key columns, please refer to the `WbDefinePk` command.

17.5. DbExplorer options

17.5.1. DB Explorer as Tab

The Database Explorer can either be displayed as a separate window or inside the main window as a another tab. If this option is selected, the Db Explorer will be displayed inside the main window. If the option Retrieve DB Explorer is checked as well, the current database scheme will be retrieved upon starting SQL Workbench/J

17.5.2. Show trigger panel

By default triggers are shown only in the details of a table. If the option "Show trigger panel" is selected, an additional panel will be displayed in the DbExplorer that displays all triggers in the database independently of their table.

17.5.3. Automatically retrieve objects

If this option is enabled, the contents of the database schema is retrieved when the DB Explorer is displayed. If this option is not checked, either the Refresh button or selecting a schema or table type will load the list.

17.5.4. Remember object type

The list of objects can be filtered with the dropdown. If the option "Remember object type" is selected, the current object type will be stored in the workspace of the current connection, and will be restored the next time.

17.5.5. Remember sort column

When this option is selected, the sort column in the data display of the DbExplorer will be restored after reloading the table data.

17.5.6. Focus to data panel

When this option is selected, the focus inside the DbExplorer will be set to the data panel, after an object in the list has been selected (and the data panel is visible).

17.5.7. Show focus

When this option is selected, a rectangle indicating the currently focused panel will be displayed, to indicate the component that will received keystrokes e.g. shortcuts such as `Ctrl-R`.

17.5.8. Generate PK constraint name

When displaying the SQL source for a table, a name will be generated for primary key constraint if the current constraint has no name or a system generated name.

If this option is selected, the generated SQL does not reflect the real statement that was used to create the table!

System generated names are identified using a regular expression that can be configured.

17.6. Window Title

The title bar of the main window displays displays information about the current connection, workspace and editor file. Some of these elements can be enabled or disabled with the options on this page.

17.6.1. Application name at end

If this option is enabled, the Application name will be put at the end of the window title.

17.6.2. Show Workspace name

If this option is enabled, the currently loaded workspace name will be displayed in the main window's title.

17.6.3. Show Profile Group

If this option is enabled, the group of the current connection profile will be displayed in the main window's title. The name of the current connection profile will always be shown.

17.6.4. Enclose Group With

If you select to display the current profile's group, you can select a pair of characters to put around the group name.

17.6.5. Separator

If you select to display the current profile's name and group, you can select the character that separates the two names.

17.6.6. Editor Filename

If the current editor tab contains an external file, you can choose if and which information about the file should be displayed in the window title. You can display nothing, only the filename or the full path information about the current file. The information will be displayed behind the current profile and workspace name.

17.7. SQL Formatting

These options influence the behaviour of the SQL Formatter when reformatting a SQL statement in the editor.

17.7.1. Max. length for sub-select

When the SQL formatter hits a sub-SELECT while parsing it will not reformat any statement which is shorter then the length specified with this option, i.e. any sub-SELECT shorter then this value will be formatted as one single statement without line breaks or indention. See SQL Formatter for details on how the SQL formatting works.

17.7.2. Columns in SELECT

This property defines the number of columns the formatter puts in on line when formatting a `SELECT` statement.

17.7.3. Quoted elements per line

When using the editor to transform lines into lists suitable for an `IN` list, this option defines how many values will be put into a single line when creating quoted elements ("Create SQL List").

17.7.4. Other elements per line

This option defines how many values will be put into a single line when creating non-quoted elements ("Create non-char SQL List").

17.8. SQL Generation

17.8.1. Generated UPDATE statements

If formatting of `UPDATE` statements is enabled, the threshold defines how many columns have to be present for a single `UPDATE` statement in order to put each column into a separate line. If the number of columns is lower then this value they will remain on one line. The keywords (UPDATE, WHERE) will still be formatted into new lines.

17.8.2. Generated INSERT statements

If formatting of `INSERT` is enabled, the way they are formatted can be controlled with several values.

Column threshold

If the number of columns in the statement exceeds this value, the columns will be spread over several lines. The number of columns that are put into each line is controlled using the option "Columns per line".

Columns per line

If the number of columns in the option "Column threshold" is exceeded, this option controls how many columns are put into each line

17.8.3. Include owner in export

This setting controls whether SQL Workbench/J uses the onwer (schema) when creating SQL scripts during exporting data (through WbExport or "Save as"). When this option is selected, the usage of the schema depends on the ignore schema setting that controls ignoring certain schemas for specific DBMS. When this is option is not selected, the schema/owner will never be used for SQL scripts.

17.8.4. Date literals for clipboard

Defines the date literal format to be used when copying data as SQL statements *to the clipboard*. For a detailed description of the different formats please refer to the WbExport description. This option does not influence the default format used by the `WbExport` command.

When you copy data as "Text" (tab-separated) to the clipboard, the date and timestamp format from the general options is used.

17.8.5. Date literals for WbExport

Defines the date literal format to be used for the WbExport command. The value of this option is used if the `-sqlDateLiterals` switch is not supplied when running `WbExport`. This default value is reported when `WbExport` is executed without parameters.

17.9. External tools

On this page, you can define external tools (programs). Currently the only place where this is used, is in the blob info dialog, to open the BLOB data with one of the defined external tools. This could be a program to display images (like XnView), Acrobat Reader to display PDF documents, OpenOffice to display office documents or a text editor (like PSPad) to display text files.

17.10. Look and Feel

If you want to use additional Look and Feels that are not part of the JDK, you can specify them here.

A Look And Feel definition consists of a name, the class name to be used and the location of the JAR file that provides the look and feel implementation. The class name that has to be used should be available in the documentation of the look and feel of your choice. The name is SQL Workbench/J internal and is only used when displaying the list of available Look and Feels.

When you switch the current Look & Feel, you will need to restart the application to activate the new look and feel. Note that if you switch the current Look & Feel it will be changed, regardless whether you close the options dialog using Cancel or OK.

The current look and feel is only changed when you click on the Make current button. Simply selecting a different entry in the list on the left side will **not** change the look and feel.

18. Properties in the .settings file

This section describes the additional options for SQL Workbench/J which are not (yet) available in the options dialog.

The name of the setting refers to the entry in the file `workbench.settings` which is located in the configuration directory. Not all listed properties will be present in `workbench.settings`. In this case, simply create a new line with the property name and the value as described here. The position where you add this entry does not matter.

18.1. Database Identifier

Some parameters are used such that a list of "Database Identifiers" is expected. The identifier that needs to be put there can be obtained by hovering the mouse over the connection URL information in the main window, or from the log file. After a successful connect to a database, there will be an entry in the log file similar to this:

```
INFO  15.08.2004 10:24:42 Connected to: [HSQL Database Engine]
```

If the description for a property in this chapter refers to a "Database Identifier", the text between (but not including) the square brackets has to be used.

18.2. DBID

For some settings, where the ID is part of the property's key, a "clean" version of the Database Identifer, called the DBID, is used. This DBID is also reported in the log file:

```
INFO  15.08.2004 10:24:42 Using DBID=hsql_database_engine
```

If the description for a property in this chapter refers to the "DBID", then this value has to be used. If the DBID is part of the property key this will be referred to as `[dbid]` in the key.

18.3. GUI related settings

18.3.1. Showing accelerator in menu

Property: `workbench.gui.showmnemonics`

Usually the mnemonic (aka. Accelerator) for a menu item is not shown under Windows 2000 or later. It will only be shown, when you press the ALT key. With this settings, this JDK behaviour can be controlled.

Default: `true`

18.3.2. Save workspaces automatically

`workbench.workspace.autosave`

If this is enabled, the current workspace will be saved automatically, when a SQL statement is executed.

Default: `false`

18.3.3. Controlling workspace backups

Property: `workbench.workspace.createbackup`

If this is enabled, a backup file will be created before saving the workspace. This is done by renaming the current file (overwriting any existing backup) and then saving the workspace.

Default: `false`

18.3.4. Controlling the number of macros in the main menu

Property: `workbench.gui.macro.maxmenuitems`

With this setting the number of macros displayed in the Macro menu can be controlled. Note that only the first 9 macros will be accessible with a menu accelerator.

Default value: `15`

18.3.5. Controlling the type of print dialog

Property: `workbench.print.nativepagedialog`

When printing the contents of a table, this settings controls the type of print dialog to be used. The default setting will open the native print dialog of the operating system. If you experience problems when trying to print, set this property to false. SQL Workbench/J will then open a cross-platform print dialog.

Default value: `true`

18.4. Editor related settings

18.4.1. Include Oracle public synonyms in auto-completion of tables

Property: `workbench.editor.autocompletion.oracle.public_synonyms`

When using auto completion for table columns and table names, Oracle's public synonyms are not included by default. This has two reasons: first, the author believes that public synonyms shouldn't be used (it's just as bad as global variables in programming) and second, Oracle defines a huge number of public synonyms that would make the popup with all available tables very long and hard to use. Setting this property to `true`, will include public synonyms in the popup. Please refer to filtering synonyms for details on how to filter out unwanted synonyms from this list.

Default value: `false`

18.4.2. Empty line to terminate SQL statements

Property: `workbench.editor.autocompletion.sql.emptylineseparator`

When analysing statements in the editor, it is assumed that individual statements separated with a semicolon. When using auto completion, SQL Workbench/J can be configured to accept an empty line as the separator between two statements. This does not influence the behaviour when running scripts or for the "execute current" command.

Default value: `false`

18.4.3. Customizing the colors used for syntax highlighting

Property: `workbench.editor.color.[type]`

These properties control the various colors used for syntax highlighting in the editor. Each entry defines a color with the three RGB values that make up the color.

Available values for `[type]` and their defaults:

`comment1` - The color used for multi-line comments (128,128,128)
`comment2` - The color used for single-line comments (128,128,128)
`keyword1` - The color standard SQL keywords (0,0,255)
`keyword2` - The color used for WB specific commands (255,0,255)
`keyword3` - The color used for SQL functions (0,150,0)
`literal1` - The color used for literals (101,0,153)
`operator` - The color used for operators (0,0,0)

18.4.4. Set the modifier key for rectangular selections in the edior

Property: `workbench.editor.rectselection.modifier`

These properties control the modifier key that needs to be pressed to enable rectangular selections in the editor. Possible values are `alt` for setting the **Alt** key as the modifier, or `ctrl` for setting the **Ctrl** key as the modifier.

Default value: `alt`

18.4.5. Default file encoding

Property: `workbench.file.encoding`

Several internal commands use an encoding when writing external text files (e.g. WbExport). If no encoding is specified for those commands, the default platform encoding as reported by the Java runtime system is used. You can overwrite the default encoding that Java assumes by setting this property.

Default value: empty, the Java runtime default is used

18.4.6. Limitting size of the text put into the history

Property: `workbench.sql.history.maxtextlength`

When you execute a SQL statement in the editor, the current content of the editor is put into the history buffer. If you are editing large scripts, this can lead to memory problems. This property controls the max. size of the editor text that is put into the history.

If the current editor text is bigger than the size defined in this property the text is not put into the history.

Default value: 10485760 (10MB)

18.4.7. Controlling newlines in code snippets

Property: `workbench.clipcreate.includenewline`

When creating a Java code snippet, the newlines inside the editor are preserved by putting a `\n` character into the String declaration. Setting this property to false, will tell SQL Workbench/J not put any `\n` characters into the Java string.

Default: `true`

18.4.8. Controlling the concatenation character for code snippets

Property: `workbench.clipcreate.concat`

When creating a Java code snippet, each line is concatenated using the standard + operator. If your programming language uses a different concatenation character (e.g. &), this can be changed with this property.

Default: +

18.4.9. Controlling the prefix for code snippets

Property: `workbench.clipcreate.codeprefix`

When creating a Java code snippet, this is prefixed with `String sql =` . With this property you can adjust this prefix.

Default: `String sql =`

18.5. DbExplorer Settings

18.5.1. Controlling data display in the DbExplorer

Property: `workbench.db.objecttype.selectable.[dbid]=value1,value2,...`

The DbExplorer makes the "data" tab available based on the type of the selected object in the object list (second column). If the type returned by the JDBC driver is one of the types listed in this property, SQL Workbench/J assumes that it can issue a `SELECT * FROM` to retrieve data from that object.

Default values:

```
.defaultt=view,table,system view,system table
.postgres=view,table,system view,system table,sequence
.rdb=view,table,system,system view
```
The values in this property are not case-sensitiv (TABLE is the same as table)

18.5.2. Showing the currently focused control in the DbExplorer

Property: `workbench.gui.dbobjects.showfocus`

To indicate the table that has currently the focus in the DbExplorer, set this property to `true`. The currently focused table (or editor) will be enclosed with a yellow border to indicate this. To change the color of the border, define the color using the property `workbench.gui.focusindicator.bordercolor`. The value of that property are the three RGB values for the color.

Default value: `false`

18.6. Controlling sorting of data

When you sort data in the result set (by clicking on the column header) this is performed using Java built-in comparisons. For performance reasons the sorting of character values (Strings) is done based on the ASCII value of the characters which results in a case-sensitive sorting. Another disadvantage of this method is, that non-ascii characters might not be sorted correctly either.

The sorting of character values can be influenced with three different settings in `workbench.settings` that control the collation to be used. The first property enables the usage of a country and language aware collation sequence, the other two properties control the language and the country to be used for the collator.

18.6.1. Using language aware collations for sorting

Property: `workbench.sort.usecollator`

When you sort the result set, characters values will be sorted case-sensitiv by default. This is caused by the `compareTo()` method available in the Java environment which puts lower case characters in front of upper case characters when sorting. When setting `workbench.sort.usecollator` to `true` a language sensitive comparison is used to sort character values, that will treat lowercase and uppercase letters the same during sorting. As the policy on how special characters are sorted is different for each language, you can also define the language and country that should be used when initializing these sorting rules.

Default value: false

18.6.2. Sort language

Property: `workbench.sort.language`

If you want to use a language/country specific collation sequence, this property defines the sort language to be used.

Default value: en

18.6.3. Sort country

Property: `workbench.sort.country`

If you want to use a language/country specific collation sequence, this property defines the sort country to be used when setting up the collation sequence.

Default value: The country defined by your operating system

18.7. Database related settings

18.7.1. Verifying connection URLs

Property: `workbench.db.verifydriverurl`

Usually before connecting to the DBMS, SQL Workbench/J will call the acceptsUrl() method of the JDBC driver, to verify if the entered URL is correct and will be accepted by the driver. Some drivers have been reported to indicate an error in the URL even if they could connect successfully.

If you are seeing an error indicating that the URL is not accepted by the driver, but you are sure that the driver accepts the URL, then set this property to `false`

Default: `true`

18.7.2. Automatically connect the DataPumper

Property: `workbench.datapumper.autoconnect`

When opening the DataPumper it will connect to the current profile as the source connection. If you do not want the DataPumper to connect automatically set this property to `false`

Default: `true`

18.7.3. Controlling COMMIT for DDL statements

Property `workbench.db.ddlneedscommit`

A list of Database Identifiers that require a COMMIT after issuing DDL statements (such as `CREATE TABLE`

Default: `PostgreSQL,Firebird,Cloudscape,Apache Derby,DB2/NT,Frontbase,Microsoft SQL Server`

18.7.4. COMMIT/ROLLBACK behaviour

Property: `workbench.db.usejdbccommit`

Some DBMS return an error when `COMMIT` or `ROLLBACK` is sent as a regular command through the JDBC interface. If the DBMS is listed here, the JDBC functions `commit()` or `rollback()` will be used instead.

This is a comma separated list of Database Identifiers

Default: `Firebird,Cloudscape`

18.7.5. Generating constraints for SQL source

Property: `workbench.db.inlineconstraints`

This setting controls the generation of the `CREATE TABLE` source in the DbExplorer. This is a comma separated list of Database Identifiers that only support defining primary and foreign keys inside the `CREATE TABLE` statement.

Default: `FirstSQL/J`

18.7.6. Case sensitivity when comparing values

Property `workbench.db.casesensitive`

The search panel of the DbExplorer highlights matching values in the result tables. The highlighter needs to know whether string comparisons in the database are case sensitive in order to highlight the correct values. This is a comma separated list of Database Identifiers

Default: `Oracle`

18.7.7. Definining additional SQL keywords

Property: `workbench.db.keywordlist.[dbid]=`

For a DBID you can define a list of additional SQL keywords. This can be used if the JDBC driver does not implement getSQLKeywords() correctly.

No default

18.7.8. Definining SQL commands that may change the database

Property: `workbench.db.updatingcommands` for general SQL statements

Property: `workbench.db.[dbid].updatingcommands` for DBMS specific update statements

When enabling the read only or confirm update option in a connection profile, SQL Workbench/J assumes a default set of SQL commands that will change the database. With this property you can add additional keywords that should be considered as "updating commands". This is a comma separated list of keywords. The keywords may not contain whitespace.

No default

18.7.9. Database switch in DbExplorer

Property: `workbench.dbexplorer.switchcatalog`

When connected to a DBMS that supports multiple databases (catalogs) for the same connection, the DbExplorer displays a dropdown list with the available databases. Switching the selected catalog in the dropdown will trigger a switch of the current catalog/database if the DbExplorer uses its own connection. If you do not want to switch the database, but merely apply the new selection as a filter (which is always done, if the DbExplorer shares the connection with the other SQL panels) set this property to `false`.

Default: `true`

18.7.10. Filtering tables

Property: `workbench.db.[dbid].exclude.tables`

Whenever SQL Workbench/J retrieves a list of tables (e.g. the DbExplorer, auto completion, WbSchemaReport) certain tables can be filtered out by supplying a regular expression in this property. The default setting will filter Oracle tables that reside in the "Recycle bin". This setting can be applied on a per DBMS basis

Default value: `workbench.db.oracle.exclude.tables=^BIN\\$.*`

Note that you need to use two backslashes in the RegeEx.

18.7.11. Filtering synonyms

Property: `workbench.db.[dbid].exclude.synonyms`

The database explorer and the auto completion can display (Oracle public) synonyms. Some of these are usually not of interest to the end user. Therefor the list of displayed synonyms can be controlled. This property defines a regular expression. Each synonym that matches this regular expression, will be excluded from the list presented in the GUI.

Default value (for Oracle): `^AQ\\$.*|^MGMT\\$.*|^GV\\$.*|^EXF\\$.*|^KU\\$_.*|^WM\\$.*|^MRV_.*|^CWM_.*|^CWM2_.*|^WK\\$_.*|^CTX_.*`

Note that you need to use two backslashes in the RegeEx.

18.7.12. Support for Oracle materialized views (snapshots)

Property: `workbench.db.oracle.detectsnapshots`

When displaying the list of tables in the database explorer Oracle materialized views (snapshots) are identified as tables by the Oracle JDBC driver. To identify a specific "table" as a materialized view, a second request to the database is necessary (accessing the system view `ALL_MVIEWS`). As this request can slow down the retrieval performance, this feature can be turned off. If for any reason the `ALL_MVIEWS` view cannot be accessed, this feature will be turned off until you re-connect to the database.

Default value: `true`

18.7.13. Fix type display for VARCHAR columns in Oracle

Property: `workbench.db.oracle.fixcharsemantics`

The Oracle driver does not report the size of VARCHAR2 columns correctly if the character semantic has been set to "char". The JDBC driver always returns the length in bytes. When this property is set to true, the length for those columns will be displayed correctly in the DbExplorer. As this means SQL Workbench/J is using it's own query to retrieve the table definition, this might not always yield the same results as the original statement from the Oracle driver. If your table definitions are not displayed correcly, set this value to `false` so that the original driver methods are used. The statement used by SQL Workbench/J is a bit faster then then original Oracle statement, as it does not use a `LIKE` predicate (which is required to comply with the JDBC specs).

Default value: `true`

18.7.14. Fix type display for NVARCHAR2 columns in Oracle

Property: `workbench.db.oracle.fixnvarchartype`

The Oracle driver does not report the type of NVARCHAR2 columns correctly. They are returned as Types.OTHER. If this property is enabled, than SQL Workbench/J is also using it's own SELECT statement to retrieve the table definition.

Default value: `true`

18.7.15. Defining a base directory for JDBC libraries

Property: `workbench.libdir`

A directory that contains the .jar files for the JDBC drivers. The value of this property can be referenced using `%LibDir%` in the driver's definition. The value for this can also be specified on the commandline.

No default

18.7.16. Defining keywords for date or timestamp input

Property: `workbench.db.keyword.current_date`

The "literals" that are accepted for DATE columns to identify the current date. Default values are `current_date, today`

Property: `workbench.db.keyword.current_timestamp`

The "literals" that are accepted for TIMESTAMP columns to identify the current date/time. Default values are `current_timestamp,sysdate,systimestamp`

Property: `workbench.db.keyword.current_time`

The "literals" that are accepted for TIME columns to identify the current time. Default values are `current_time, now`

18.7.17. Use Savepoints to guard DDL statement execution

Property: `workbench.db.[dbid].ddl.usesavepoint`

Some DBMS (such as PostgreSQL) cannot continue inside a transaction when an error occurs. A script with multiple DDL statements can therefor not run completely if one statement fails, even if you choose to ignore the error. If this property is set to true, SQL Workbench/J will set a savepoint before executing a DDL statement. In case of an error the savepoint will be rolledback and the transaction can continue.

Default value: `true` for PostgreSQL, `false` for others

18.7.18. Use Savepoints to update/insert mode for WbImport

Property: `workbench.db.[dbid].import.usesavepoint`

Some DBMS (such as PostgreSQL) cannot continue inside a transaction when an error occurs. When running WbImport in `update,insert` or `insert,update` mode, the first of the two statements needs to be rolled back in order to be able to continue the import. If this property is set to true, SQL Workbench/J will set a savepoint before executing the first (insert or update) statement. In case of an error the savepoint will be rolledback and WbImport will try to execute the second statement.

Default value: `true` for PostgreSQL, `false` for others

18.7.19. Ignore errors during data retrieval

Property: `workbench.db.ignore.readerror`

When retrieving data (e.g. using a `SELECT` statement) errors that are reported by the driver will be displayed to the user. The retrieval will be terminated. If you want to ignore errors and replace the data that could not be retrieved with a `NULL` value, set this property to `true`.

Using this parameter is not recommended as it might produce results that do not reflect the data as it is stored in the database.

Default value: `false`

18.8. SQL Execution related settings

18.8.1. Maximum script size for in-memory script execution

Property: `workbench.sql.script.inmemory.maxsize`

This setting controls the size up to which files that are executed in batch mode or via the WbInclude command are read into memory. Files exceeding this size are not read into memory but processed statement by statement. When a file is not read into memory the automatic detection of the alternate delimiter does not work any longer. The size is given in bytes.

Default: `1048576`

18.8.2. Ignoring certain SQL commands

Property: `workbench.db.ignore.[dbid]=`

For a DBMS identifier you can define a list of commands that are simply ignored by SQL Workbench/J. This is useful e.g. for Oracle, when you want to run scripts that are intended for SQL*Plus. If those scripts contain special SQL*Plus commands (that are not understood by the Oracle server as SQL*Plus executes these commands directly) they would fail in SQL Workbench/J. If those commands are simply ignored and not send to the server, the scripts can run without modification.

The DBID is the lower case DBMS identifier (see above) with all special characters like `\/,.$[]()` and spaces removed. The actual ID that is used to retrieve this list is reported as an informational message in the log file (`Using DBID=`)

Default: `workbench.db.ignore.oracle=prompt,exit,whenever`

18.8.3. Enabling short WbInclude

Property: `workbench.db.supportshortinclude`

By default the WbInclude command can be shortened using the @ sign. This behaviour is disabled for MS SQL to avoid conflicts with parameter definitions in stored procedures. This property contains a list of DBIDs for which this should be enabled. To enable this for all DBMS, simply use * as the value for this property.

Default: `oracle, rdb, hsqldb, postgresql, mysql, adaptive_server_anywhere, cloudscape, apache_derby`

18.8.4. Check for single line commands without delimiter

Property: `workbench.db.checksinglelinecmd`

When parsing a SQL script, SQL Workbench/J supports statements that are put into a single line without a delimiter. This is primarily intended for compatibility with Oracle's SQL*Plus and is not enabled for other database systems (starting with build 86).

Default: `oracle`

18.9. Default settings for Export/Import

For some switches of the WbExport and WbImport command, you can override the default values used by SQL Workbench/J in case you do not provide the parameter. The default values mentioned in this chapter apply, if no property is defined in the `workbench.settings` file. The current default for these properties is displayed in the help message when you run the corresponding command without any parameters.

18.9.1. Controlling header lines in text exports

Property: `workbench.export.text.default.header`

This property controls whether default value for the `-header` parameter of the WbExport command.

Default: `false`

18.9.2. Controlling XML export format

Property: `workbench.export.xml.default.verbose`

This property controls whether XML exports are done using verbose XML or short tags and only basic formatting. This property sets the default value of the `-verbosexml` parameter for the WbExport command.

Default: `true`

18.9.3. Setting default for WbImport's -continueOnError parameter

Property: `workbench.import.default.continue`

This property controls the default value for the parameter `-continueOnError` of the WbImport command.

18.9.4. Setting default for WbImport's -header parameter

Property: `workbench.import.default.header`

This property controls the default value for the parameter `-header` of the WbImport command.

Default: `true`

18.9.5. Setting default for WbImport's -multiLine parameter

Property: `workbench.import.default.multilinerecord`

This property controls the default value for the parameter `-multiLine` of the WbImport command.

Default: `false`

18.9.6. Setting default for WbImport's -trimValues parameter

Property: `workbench.import.default.trimvalues`

This property controls the default value for the parameter `-trimValues` of the WbImport command.

Default: `false`

18.10. Controlling the log file

18.10.1. Log level

Property: `workbench.log.level`

Set the log level for the log file. Valid values are

- DEBUG
- INFO
- WARN
- ERROR

Default: `INFO`

18.10.2. Log format

Property: `workbench.log.format`

Define the format of the log messages. The following placeholders are supported:

- {type}
- {timestamp}
- {message}
- {error}
- {source}
- {stacktrace}

The order of the placeholders defines the order in the log file, except for the stacktrace which will always be printed after the message. If the log level is set to debug, the stacktrace will always be displayed even if it is not included in the format string.

Default: `{type} {timestamp} {message} {error}`

18.10.3. Logging to the console

Property: `workbench.log.console`

Defines whether SQL Workbench/J logs messages additionally to the standard error output

Default: `false`

18.10.4. Logging SQL used for retrieving metadata

Property: `workbench.dbmetadata.logsql`

If this is set to `true`the SQL queries used to retrieve DBMS specific meta data (such as view/procedure/trigger source, defined triggers/views) will be logged with level INFO. This can be used to debug customized SQL statements for DBMS's which are not (yet) preconfigured.

Default: `false`

18.11. Settings related to SQL statement generation

18.11.1. Controlling schema usage in generated SQL statements

Property: `workbench.sql.ignoreschema.[dbid]=schema1,...`

Define a list of schemas that should be ignored for the DB ID When SQL Workbench/J creates DML statements and the current table is reported to belong to any of the schemas listed in this property, the schema will not be used to qualify the table. To ignore all schemas use a *, e.g. `workbench.sql.ignoreschema.rdb=*`. In this case, table names will never be prefixed with the schema name reported by the JDBC driver. The values specified in this property are case sensitiv.

Note that for Oracle, tables that are owned by the current user will never be prefixed with the owner.

Default values:

```
.oracle=PUBLIC
.postgres=public
.rdb=*
```

18.11.2. System generated names for contraints

Property: `workbench.db.[dbid].constraints.systemname`

Defines a regular expression to identify system generated constraint names. If a constraint name is identified as beeing system generated, it is treated as if no name was defined, when e.g. creating the SQL for a table. Whether or not SQL Workbench/J then generates a name for the constraint can be controlled in the options for the DbExplorer.

Default values:

oracle: `^SYS_.*`
mysql: `PRIMARY`

18.11.3. Controlling the chunk size for WbDataDiff

Property: `workbench.sql.sync.chunksize`

Controls the number of rows that are retrieved from the target table when running `WbDataDiff` or `WbCopy` with the `-syncDelete=true` parameter.

Default value: `25`

18.12. Filter settings

18.12.1. Controlling the number of items in the pick list

Property: `workbench.gui.filter.mru.maxsize`

When saving a filter to an external file, the pick list next to the filter icon will offer a drop down that contains the most recently used filter definitions. This setting will control the maximum size of that dropdown.

Default value: `15`