SQL Workbench/J User's Manual

Table of Contents

1. General Information 8
    1.1. Program version 8
    1.2. Feedback and support 8
    1.3. Credits and thanks 8
    1.4. Third party components 8
2. Software license 11
3. Change log 12
4. Installing and starting SQL Workbench/J 14
    4.1. Pre-requisites 14
    4.2. First time installation 14
    4.3. Upgrade installation 14
    4.4. Starting the program from the commandline 14
    4.5. Starting the program using the shell script 15
    4.6. Starting the program using the Windows® launcher 15
    4.7. Configuration directory 16
    4.8. Copying an installation 17
    4.9. Increasing the memory available to the application 18
5. Command line parameters 19
    5.1. Specify the directory for configuration settings 19
    5.2. Specify a base directory for JDBC driver libraries 19
    5.3. Specify the file containing connection profiles 19
    5.4. Setting configuration properties 20
    5.5. Defining variables 20
    5.6. Prevent updating the .settings file 21
    5.7. Connect using a pre-defined connection profile 21
    5.8. Connect without a profile 21
6. JDBC Drivers 24
    6.1. Configuring JDBC drivers 24
    6.2. Specifying a library directory 25
    6.3. Popular JDBC drivers 25
7. Connecting to the database 26
    7.1. Connection profiles 26
    7.2. Managing profile groups 26
    7.3. JDBC related profile settings 27
    7.4. PostgreSQL connections 28
    7.5. Extended properties for the JDBC driver 28
    7.6. SQL Workbench/J specific settings 28
    7.7. Connecting through a SSH tunnel 34
    7.8. Connect to Oracle with SYSDBA privilege 35
    7.9. Using the quick filter 35
8. Using workspaces 37
    8.1. Overview 37
    8.2. Creating a copy of the current workspace 37
    8.3. Load a different workspace 37
    8.4. Workspace and external files 38
    8.5. Workspace variables 38
9. Editing SQL Statements 39
    9.1. Editing files 39
    9.2. Code completion 39

9.3. Show hints for INSERT statements 40
9.4. Customizing keyword highlighting 40
9.5. Reformat SQL 41
9.6. Create SQL value lists 41
9.7. Programming related editor functions 42
10. Working with bookmarks 45
10.1. Defining bookmarks 45
10.2. Jumping to a bookmark 45
10.3. Configuring the display of the bookmark list 45
11. Creating stored procedures and triggers 46
11.1. PostgreSQL 46
11.2. Oracle PL/SQL 46
11.3. Using a static alternate delimiter 47
11.4. Using a dynamic delimiter 48
12. Using SQL Workbench/J 50
12.1. Displaying help 50
12.2. Resizing windows 50
12.3. Executing SQL statements 50
12.4. Displaying results 52
12.5. Dealing with BLOB and CLOB columns 53
12.6. Performance tuning when executing SQL 55
12.7. Saving and loading SQL scripts 55
12.8. Displaying the structure of tables 55
12.9. Viewing server messages 56
12.10. Editing data 57
12.11. Deleting rows from the result 58
12.12. Sorting the result 58
12.13. Filtering the result 58
12.14. Running stored procedures 59
12.15. Export result data 60
12.16. Copy data to the clipboard 60
12.17. Import data into the result set 61
13. Using SQL Workbench/J specific annotations in SQL comments 63
13.1. Naming result tabs 63
13.2. Adding macros to the result's context menu 63
13.3. Re-using an existing named result tab 64
13.4. Scrolling the result 65
13.5. Appending a results 65
13.6. Suppressing empty results 65
13.7. Automatic refresh of the result 65
13.8. Optimize row height 65
13.9. Show BLOB column as image 66
13.10. Keep result 66
13.11. Result as text 66
13.12. Create crosstab (pivot) from a result 67
14. Using macros and text clips 69
14.1. Loading and saving macro sets 69
14.2. Defining Macros 69
14.3. Executable macros 70
14.4. Expandable macros 71
14.5. DbTree macros 71
15. Working with foreign keys 73
15.1. Navigating referenced rows 73
15.2. Generating JOIN conditions 73
15.3. Selecting foreign key values in referencing tables 74

15.4. Deleting rows with foreign keys 75
16. DBMS specific features 76
16.1. PostgreSQL specific features 76
16.2. Oracle specific features 77
17. Variable substitution in SQL statements 81
17.1. Defining variables 81
17.2. Populate a variable from a SELECT statement 81
17.3. Populate a variable from a file 82
17.4. Editing variables 82
17.5. Using variables in SQL statements 82
17.6. Prompting for values during execution 83
17.7. Controlling the order of variables during prompting 83
17.8. Scope of variables 84
18. Using SQL Workbench/J in batch files 85
18.1. Specifying the connection 85
18.2. Specifying the script file(s) 85
18.3. Specifying a SQL command directly 85
18.4. Specifying a delimiter 86
18.5. Specifying an encoding for the file(s) 86
18.6. Specifying a logfile 86
18.7. Handling errors 86
18.8. Specify a script to be executed on successful completion 87
18.9. Specify a script to be executed after an error 87
18.10. Ignoring errors from DROP statements 87
18.11. Changing the connection 87
18.12. Controlling console output during batch execution 88
18.13. Running batch scripts interactively 88
18.14. Defining variables 88
18.15. Setting configuration properties 89
18.16. Examples 89
19. Using SQL Workbench/J in console mode 90
19.1. Entering statements 90
19.2. Exiting console mode 90
19.3. Setting or changing the connection 91
19.4. Displaying result sets 91
19.5. Running SQL scripts that produce a result 92
19.6. Controlling the number of rows displayed 92
19.7. Controlling the query timeout 93
19.8. Managing connection profiles 93
19.9. Using an external pager 94
19.10. PostgreSQL psql commands 94
20. Export data using WbExport 96
20.1. Memory usage and WbExport 96
20.2. Exporting Excel files 96
20.3. General WbExport parameters 97
20.4. Parameters for text export 103
20.5. Parameters for XML export 105
20.6. Parameters for type SQLUPDATE, SQLINSERT or SQLDELETEINSERT 106
20.7. Parameters for Spreadsheet types (ods, xslm, xls, xlsx) 108
20.8. Parameters for HTML export 110
20.9. Parameters for JSON export 110
20.10. Compressing export files 110
20.11. Examples 111
21. Import data using WbImport 114
21.1. Importing spreadsheet files 114

21.2. General parameters 114
21.3. Parameters for the type TEXT 123
21.4. Text Import Examples 126
21.5. Parameters for the type XML 129
21.6. Parameters for spreadsheet import 130
21.7. Update mode 131
21.8. Native UPSERT mode 132
21.9. Native insertIgnore mode 132
22. Copy data across databases 134
22.1. General parameters for the `WbCopy` command. 134
22.2. Copying data from one or more tables 137
22.3. Copying data based on a SQL query 140
22.4. Update mode 140
22.5. Synchronizing tables 140
22.6. Examples 141
23. Comparing databases 143
23.1. Compare two database schemas - WbSchemaDiff 143
23.2. Compare data across databases - WbDataDiff 146
24. Search data and code in the database 152
24.1. Search source of database objects - WbGrepSource 152
24.2. Search data in multiple tables - WbGrepData 153
25. SQL Workbench/J to generate DDL commands 157
25.1. Generate SQL script for database objects - WbGenerateScript 157
25.2. Show the source of a table - WbTableSource 158
25.3. Show the source of a view - WbViewSource 158
25.4. Show the source of a stored procedures - WbProcSource 158
25.5. Show the source of a trigger - WbTriggerSource 159
25.6. Generate DROP statement with dependencies - WbGenerateDrop 159
25.7. Generate SQL script for foreign key constraints - WbGenerateFKScript 160
25.8. Generate a table definition from an import file - WbGenerateImpTable 160
26. SQL Workbench/J to generate DML statements 162
26.1. Generate DELETE statements with dependencies - WbGenerateDelete 162
26.2. Generate INSERT statements with dependencies - WbGenerateInsert 163
27. Show information about database objects 164
27.1. Create a report of the database objects - WbSchemaReport 164
27.2. Show table structure - WbDescribe 166
27.3. List tables - WbList 166
27.4. List indexes - WbListIndexes 166
27.5. List stored procedures - WbListProcs 167
27.6. List triggers - WbListTriggers 167
27.7. List catalogs - WbListCat 167
27.8. List schemas - WbListSchemas 167
27.9. Show dependency information - WbListDependencies 168
28. Manage macros with SQL Workbench/J command 169
28.1. Define a new macro - WbDefineMacro 169
28.2. Delete a macro - WbDeleteMacro 169
28.3. List available macros - WbListMacros 169
29. Manage variables with SQL Workbench/J 170
29.1. Define a script variable - WbVarDef 170
29.2. Delete a script variable - WbVarDelete 171
29.3. Show defined script variables - WbVarList 171
30. Other SQL Workbench/J specific commands 172
30.1. Confirm script execution - WbConfirm 172
30.2. Display a message box - WbMessage 172
30.3. Print a text - WbEcho 172

30.4. Run a stored procedure with OUT parameters - WbCall 173
30.5. Execute a SQL script - WbInclude 174
30.6. Conditional execution 176
30.7. Run a result set as a SQL script - WbRunResult 177
30.8. Extract and run SQL from a Liquibase ChangeLog - WbRunLB 178
30.9. Handling tables or updateable views without primary keys 179
30.10. Change the default fetch size - WbFetchSize 180
30.11. Run statements as a single batch - WbStartBatch, WbEndBatch 180
30.12. Extracting BLOB content - WbSelectBlob 180
30.13. Control feedback messages - WbFeedback 181
30.14. Change the current schema - WbSetSchema 181
30.15. Change the current database/catalog - WbSwitchDB 181
30.16. Setting connection properties - SET 181
30.17. Changing Oracle session behavior - SET 182
30.18. Changing read only mode - WbMode 183
30.19. Count rows for all tables - WbRowCount 183
30.20. Change the connection for a script - WbConnect 184
30.21. Show the history of SQL statements - WbHistory 186
30.22. Run an XSLT transformation - WbXslt 186
30.23. Running operating system commands - WbSysExec 187
30.24. Opening a file with the default application - WbSysOpen 187
30.25. Change an internal configuration parameter - WbSetConfig 188
31. DataPumper 189
31.1. Overview 189
31.2. Selecting source and target connection 189
31.3. Copying a complete table 189
31.4. Advanced copy tasks 191
32. Database Object Explorer 192
32.1. Objects tab 192
32.2. Table details 194
32.3. Modifying the definition of database objects 195
32.4. Table data 195
32.5. Changing the display order of table columns 196
32.6. Customize data retrieval 196
32.7. Customizing the generation of the table source 197
32.8. View details 197
32.9. Procedure tab 197
32.10. Search table data 198
33. Working with the Database Object tree 200
33.1. Filtering the elements in the tree 200
33.2. Drag and drop support 200
33.3. Finding elements in the tree 200
33.4. Features available through the context menu 200
34. Common problems 203
34.1. The driver class was not found 203
34.2. Syntax error when creating stored procedures 203
34.3. The SQL source code for tables or indexes is incorrect 203
34.4. Timestamps with timezone information are not displayed correctly 203
34.5. Some of the dialogs are too small 204
34.6. Excel export not available 204
34.7. Generated XLSX files can't be opened with Excel 204
34.8. Out of memory errors 204
34.9. High CPU usage when executing statements 204
34.10. High CPU usage when displaying results 204
35. Common DBMS problems 205

35.1. Oracle 205
35.2. MySQL 207
35.3. Microsoft SQL Server 207
35.4. IBM DB2 209
35.5. Amazon Redshift 210
35.6. PostgreSQL 210
35.7. Sybase SQL Anywhere 210
36. Options dialog 212
36.1. General options 212
36.2. Editor options 214
36.3. SQL Excecution options 216
36.4. Macro options 217
36.5. Bookmark options 218
36.6. Editor colors 218
36.7. Font settings 219
36.8. Auto-completion options 219
36.9. Workspace options 220
36.10. Backup options 221
36.11. Options for displaying data 222
36.12. Options for formatting data 225
36.13. Data display colors 227
36.14. Options for data editing 228
36.15. DbExplorer options 229
36.16. DbTree options 231
36.17. File tree options 231
36.18. SQL Formatting 232
36.19. External SQL Formatter 235
36.20. SQL Generation 235
36.21. Window Title 237
36.22. SSH Hosts 237
36.23. External tools 238
36.24. Look and Feel 238
36.25. Customize the FlatLaf Look & Feel 239
36.26. Logging 239
37. Configuring keyboard shortcuts 240
37.1. Assign a shortcut to an action 240
37.2. Removing a shortcut from an action 240
37.3. Reset to defaults 240
38. Advanced configuration options 241
38.1. DBID 241
38.2. GUI related settings 241
38.3. Editor related settings 242
38.4. Controlling code generation for code snippets 243
38.5. DbExplorer Settings 244
38.6. General configuration settings 245
38.7. Database related settings 245
38.8. Configuring JDBC's clientInfo values 251
38.9. Configuring the check for the update table 251
38.10. DBMS specific settings 252
38.11. SQL Execution related settings 254
38.12. Default settings for Export/Import 255
38.13. Controlling the log file 257
38.14. Configure Log4j logging 258
38.15. Configuring the logfile viewer 259
38.16. Configuring SQL keywords 259

38.17. Settings related to SQL statement generation 260
38.18. Customize table source retrieval 261
38.19. Customize index source retrieval 262
38.20. Filter settings 263
39. Simple properties based profile storage 264
Index 266

1. General Information

1.1. Program version

This document describes build 133 of SQL Workbench/J

1.2. 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.eu>`

SQL Workbench/J can be downloaded from https://www.sql-workbench.eu

If you want to contact other users of SQL Workbench/J you can do this using the online forum: https://forum.sql-workbench.eu/

1.3. Credits and thanks

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

1.4. Third party components

1.4.1. JLine

SQL Workbench/J includes the JLine library to support command line editing for the console mode on Unix style operating systems. The JDK on Windows® supports full editing of the command line including the usual Windows® keyboard shortcuts to show the list of commands, so JLine is not used when SQL Workbench/J is running under Windows®.

The copyright notice for JLine follows:

Copyright (c) 2002-2006, Marc Prud'hommeaux <mwp1@cornell.edu> All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of JLine nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.4.2. Editor

The editor is based on the JEdit Syntax package: https://sourceforge.net/projects/jedit-syntax/

The jEdit 2.2.1 syntax highlighting package contains code that is Copyright 1998-1999 Slava Pestov, Artur Biesiadowski, Clancy Malcolm, Jonathan Revusky, Juha Lindfors and Mike Dillon.

1.4.3. Charset detector

SQL Workbench/J uses the Java port of Mozilla's universal charset detector from https://code.google.com/p/juniversalchardet/

1.4.4. JSch - Java Secure Chanel

SQL Workbench/J includes the JSch library to support SSH tunneling.

The copyright notice for JSch follows:

Copyright (c) 2002-2015 Atsuhiko Yamanaka, JCraft,Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JCRAFT, INC. OR ANY CONTRIBUTORS TO THIS SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.4.5. Icons

Some icons are taken from Yusuke Kamiyamane's Fugue Icons: https://p.yusukekamiyamane.com/

Some icons are taken from glyFX Image Library: http://www.glyfx.com

Some icons are taken from Aha Soft: https://www.aha-soft.com/free-icons/

The DbExplorer icon is from the icon set "Mantra" by Umar Irshad: https://umar123.deviantart.com/

2. Software license

Copyright 2002-2025, Thomas Kellerer

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License.

You may obtain a copy of the License at

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

3. Change log

Changes from build 132 to build 133

Enhancements

- For WbExport it's now possible to specify the font name to be used for Excel exports.
- SQL Workbench/J now requires Java 21
- References to files can now be stored as relative paths inside a workspace.
- For WbExport it's now possible to define the number of rows per INSERT statement when using multi row inserts.
- It's now possible to show BLOB columns as images.
- WbExport now supports opening the generated file
- The "Single record dialog" now also supports selecting foreign key values
- WbInclude now supports sending the entire script without splitting it up into separate statements.
- In the DbTree, result parameters are now shown for stored procedures (if supported by the JDBC driver).
- WbGenerateScript can now write one file per database object, instead of one single large file for everything.
- For Postgres, partitioned tables and regular tables are now shown as the same type in the DbTree and DbExplorer.
- WbGenerateInsert can now optionally retrieve the parent tables for the provided tables to build the entire dependency tree.
- The DbTree will now also show the WHERE condition for filtered indexes (if supported by the DBMS).
- For Postgres, include column level STATISTICS settings in the generated DDL for a table.
- It's now possible to make results read-only by default, even if the connection profile is not set to read only.
- A new annotation @WbReadOnly is available to make a result read only.

Bug fixes

- In the dialog prompt to enter a connection's password the input field didn't initially have the focus.
- For Postgres, the connection wasn't properly put into read only mode when enabling this in the profile
- For Db2 on z/OS, the source code for triggers and views was not retrieved.
- Connection profiles stored in a properties file with UTF-8 encoding, weren't loaded correctly.
- Fixed quoting of command line arguments when running external formatters
- Copying data using the DataPumper did no longer work.
- For SQL Server 2012 and later, the system view sql_modules is now used to retrieve procedure sources instead of sp_helptext.
- Generating empty/default statements from the DbExplorer or DbTree did not longer work.

- Resetting the read-only state after disabling it by clicking on the "lock" icon, didn't work anymore.
- Code completion for TRUNCATE TABLE did not work for schema prefixed names.
- For Postgres 9.6 and earlier, sequences that needed quoting were not retrieved and their DDL was not generated.
- Selecting a foreign key value while editing did not work if the foreign key constraint did not reference the primary key of a table.
- For Postgres, triggers on a partitioned table were shown once for each partition in the DbTree.
- Identity columns where not marked as "auto increment" in the table definition display.
- For Oracle, column level constraints IS JSON where not generated.
- The statement timeout was not applied when running DDL commands
- For Postgres, tables where no longer retrieved correctly when using JDBC driver 42.7.4
- For Oracle, the generated DDL for global temp tables was not correct if LOB storage parameters where specified.
- For Informix, date and datetime values are sent again using the old JDBC 3 compliant data types.

The full release history is available at the SQL Workbench/J homepage

4. Installing and starting SQL Workbench/J

4.1. Pre-requisites

SQL Workbench/J requires a Java 11 (or higher) runtime environment. You can either use a JRE ("Runtime") or a JDK ("Development Kit").

For Windows® and Linux a batch file is provided to automatically download a Java runtime environment

- For Windows®: `download_jre.cmd`
- For Linux/Unix and MacOS: `download_jre.sh`

For Windows® and Linux, it will download the a JRE from the SQL Workbench/J homepage. For MacOS, it will download a Java runtime from Adoptium. The Java runtime will be stored in the sub-directory `jre` which is automatically used by the Windows® launcher and the shell scripts to start SQL Workbench/J

You can manually download Windows®, Linux or MacOS binaries from one of these sites:

- Adoptium
- Azul Systems

Starting with build 126 the Windows® launcher can also download a Java runtime, if none was found.

4.2. First time installation

Once you have downloaded the application's distribution package, unzip the archive into a directory of your choice. Apart from that, no special installation procedure is needed.

You will need to configure the necessary JDBC driver(s) for your database before you can connect to a database. Please refer to the chapter JDBC Drivers for details on how to make the JDBC driver available to SQL Workbench/J

When starting SQL Workbench/J for the first time, it will create a directory called `.sqlworkbench` in the current user's home folder to store all its configuration information.

The "user's home directory" is `$HOME` on a Linux or Unix based system, and `%HOMEPATH%` on a Windows® system. (Technically speaking it is using the contents of Java system property `user.home` to find the user's home directory).

4.3. Upgrade installation

When upgrading to a newer version of SQL Workbench/J simply overwrite the old `sqlworkbench.jar`, the `exe` files and shell scripts that start the application.

If you are using the bundle with the "optional libraries", please delete all existing `jar` files in the `ext` sub-directory, before extracting the archive of the new version.

4.4. Starting the program from the commandline

To start the program a shell/batch script is provided. When using Linux or MacOS SQL Workbench/J can be started using `sqlworkbench.sh`.

When using Windows®, you can either use the batch file `sqlworkbench.cmd>` or `SQLWorkbench.exe` to start SQL Workbench/J.

Details on using the Windows® launcher can be found here.

4.5. Starting the program using the shell script

To run SQL Workbench/J under an Unix-type operating system, the supplied shell script `sqlworkbench.sh` can be used.

4.5.1. Specifying the Java runtime for the shell script

The shell scripts (and the batch files) first check if a Java runtime is available in the sub-directory `jre`. If that is available it will be used. The shell script `download_jre.sh` will automatically extract the Java runtime into that directory.

If no "local" Java runtime is found, the environment variable `WORKBENCH_JDK` is checked. If that variable is defined and points to a Java runtime installation, the shell script will use `$WORKBENCH_JDK/bin/java` to run the application.

If `WORKBENCH_JDK` is not defined, the shell script will check for the environment variable `JAVA_HOME`. If that is defined, the script will use `$JAVA_HOME/bin/java` to run the application.

If neither `WORKBENCH_JDK` nor `JAVA_HOME` is defined, the shell script will simply use `java` to start the application, assuming that a valid Java runtime is available on the path.

All parameters that are passed to the shell scripts are passed to the application, not to the Java runtime. If you want to change the memory or other system settings for the JVM, you need to edit the shell script.

4.6. Starting the program using the Windows® launcher

To start SQL Workbench/J on the Windows® platform, the supplied `SQLWorkbench.exe` for a 32bit Java runtime or `SQLWorkbench64.exe` for a 64bit Java runtime can be used to start the program. The file `sqlworkbench.jar` has to be located in the same directory as the exe files, otherwise it does not work.

By default the launcher will initialize the Java runtime in the same process as long as the "bitness" of the used Java runtime matches the one from the launcher. If `SQLWorkbench64.exe` is used, but a 32bit Java runtime is configured, the launcher will start Java as a second process. Likewise, if `SQLWorkbench.exe` is used and a 64bit Java runtime is configured, the launcher will also start Java as a second process.

Note that "pinning" SQL Workbench/J to the Windows® taskbar only works if the launcher can initialize the Java runtime as part of its own process and thus no second process is started.

SQL Workbench/J does not need a "fully installed" runtime environment, you can also copy the `jre` directory from an existing Java installation into the directory where SQL Workbench/J is installed. Creating a symbolic link (using `mklink`) also works. A "local" Java installation in the `jre` subdirectory will have priority over any JRE or JDK registered in the Windows® registry.

The launcher searches for the Java runtime in the following order. The first Java installation found will be used.

1. it first checks if a command line parameter `-javaHome` was specified. If it was provided and points to a Java installation, that will be used.
2. then it checks, if the configuration file `SQLWorkbench.cfg` contains a `javaHome` entry. If it exists and points to a Java installation, that will be used.

3. then it checks if a Java runtime is available in the sub-directory `jre`.
4. if the environment variable `WORKBENCH_JDK` is defined, the launcher will use that to run the application.
5. if the environment variable `JAVA_HOME` is defined, the launcher will use that to run the application.
6. then it checks if the Windows registry contains information about a Java runtime.
7. if still no Java was found, all directories defined through the `PATH` variable are searched for a Java installation. The first one found will be used.

If neither of those steps found a Java runtime, the launcher offers to download a Java runtime or select an existing installation. The Java runtime is downloaded from Adoptium, as the the provided batch file does.

When using a 32bit Java runtime the default memory available to the application is set to 1GB. When using a 64bit Java runtime the default is set to 65% of the available physical memory.

4.6.1. Parameters for the Windows® launcher

Additional parameters to the Windows® launcher can be defined in a configuration file that needs to be created in the directory where `SQLWorkbench.exe` is located. The name of the file has to be `sqlworkbench.cfg`.



The format of `sqlworkbench.cfg` is different to the `.ini` file of previous versions. The launcher will convert an existing `SQLWorkbench.ini` or `SQLWorkbench64.ini` to the new format if started for the first time. It is no longer necessary to have different configuration files for the 32bit and 64bit version of the launcher.

Specifying the Java location

If the launcher cannot find a Java runtime, you can specify the location of the JRE in the configuration file in the section `[Workbench]` with the following parameter:

```
[Workbench]
javaHome=c:\Program Files\Java
```

You need to specify the root directory of the Java runtime (or JDK). This can either be a 32bit or a 64bit runtime, but a 64bit runtime is highly recommended (and therefor using `SQLWorkbench64.exe`).

Defining the memory for the application

The memory available to the Java runtime is specified through the configuration file by providing the corresponding JVM paramter. The following entry would set the maximum memory to 4GB (the `-Xmx` part is passed to the Java runtime)

```
[Java]
vmarg.mem=-Xmx4g
```

You can verify the available memory in the about dialog: Help » About

4.7. Configuration directory

The configuration directory is the directory where all config (`workbench.settings`, `WbProfiles.xml`, `WbDrivers.xml`) files are stored.

If no configuration directory has been specified on the commandline, SQL Workbench/J will identify the configuration directory by looking at the following places

1. The current directory
2. The directory where `sqlworkbench.jar` is located
3. In the user's home directory (e.g. `$HOME/.sqlworkbench` on Unix based systems or `%HOMEPATH%\.sqlworkbench` on Windows® systems)

If the file `workbench.settings` is found in one of those directories, that directory is considered the configuration directory.

If no configuration directory can be identified, it will be created in the user's home directory (as `.sqlworkbench`).

The above mentioned search can be overridden by supplying the configuration directory on the commandline when starting the application.

The following files are stored in the configuration directory:

- General configuration settings (`workbench.settings`)
- Connection profiles (`WbProfiles.xml`)
- JDBC Driver definitions (`WbDrivers.xml`)
- Customized shortcut definitions (`WbShortcuts.xml`). If you did not customize any of the shortcuts, this file does not exist
- Macro definitions (`WbMacros.xml`)
- Saved column orders from the DbExplorer(`WbColumnOrder.xml`)
- Log file (`workbench.log`)
- Workspace files (`*.wksp`)
- The cache directory for the code completion

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 command line. 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.



The location of the settings file (and therefor the configuration directory) is shown through Help » About and on the "General" page of the Options dialog which can be displayed using Tools » Options.

4.8. Copying an installation

To copy an installation to a different computer, simply copy all the files from the configuration directory to the other computer (the log file does not need to be copied). When a profile is connected to a workspace, the workspace file should be specified without a directory name (or using the `%ConfigDir%` placeholder). In that case it is always loaded from the configuration directory. If the workspace file is given with an absolute directory, this needs to be adjusted after the copying the files.

You will need to edit the driver definitions (stored in `WbDrivers.xml`) because the *full path* to the driver's jar file(s) is stored in the file.

If you store all JDBC drivers in a common directory (or below a common root directory) you can define the libdir variable. In that case the paths to the driver's jar file are stored relative to the `%LibDir%` directory. After copying the installation you only need to adjust the `%LibDir%` variable on the other computer.

4.9. Increasing the memory available to the application

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

SQL Workbench/J reads **all** the data that is returned by a SQL statement into 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 using the Windows® launcher (e.g. `SQLWorkbench64.exe`), the available memory is defined in the INI file.

When using the shell or batch scripts, the available memory is defined through the `-Xmx` parameter for the `java` command. In the following example, the parameter `-Xmx4g` sets the available memory to 4GB

```
java -Xmx4g -jar sqlworkbench.jar
```

If you are using the supplied shell scripts to start SQL Workbench/J, you can edit the scripts and change the value for the -Xmx parameter in there.



With a 32bit Java runtime, you can not use (or assign) more than approx. 1.5GB for the application. If you need to process results that require more memory that that, you will have to use a 64bit Java runtime.

5. 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 command line (especially in a Windows® environment) you have to use single quotes, as the double quotes won't be passed to the application.

5.1. 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.

If you want to control the location where SQL Workbench/J 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.

On the Windows® platform you can use a forward slash to separate directory names in the parameter.

5.2. 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 placeholder `%LibDir%` The value for this parameter can also be set in the file `workbench.settings`.

5.3. Specify the file containing connection profiles

SQL Workbench/J stores the connection profiles in a file called `wb-profiles.properties`. If you want to use a different filename, or use different set of profiles for different purposes the parameter `-profileStorage` can be used to select a different file or directory. parameter.

If the value of the parameter does not contain a path, the file will be expected (and stored) in the configuration directory. To specify a file in the current directory use `-profileStorage=./my-profiles.properties`

Before Build 126, the profiles were stored in an XML file named `WbProfiles.xml` which is not intended to be edited manually.

When specifying a properties file with `-profileStorage` the file extension **must** be `.properties`

The value of `-profileStorage` may specify a directory. In that case all profile files (XML and .properties) from that directory will be included.

This parameter can be specified multiple times to specify multiple files or directories.

5.4. Setting configuration properties

Configuration properties that are stored in `workbench.settings` can be overwritten on the command line by setting them as Java system properties. This is done using the `-D` parameter. When using the Windows launcher, the position of this does not matter. On Linux this parameter must be specified before the main class starting the application, e.g.:

```
java -Dworkbench.some.property=42 -cp sqlworkbench.jar:libs/* workbench.WbStarter
```

5.5. Defining variables

You can define variables when starting SQL Workbench/J by either passing the variable definition directly or by passing a file that contains the variable definitions.

Defining variable values in this way can also be used when running in batch mode.

Variables defined through a command line parameter have a priority over variables defined in the connection profile or the workspace. If a variable is defined on the command line and a variable with the same name is defined in the profile or the workspace, the value from the profile (or workspace) will **not** be used.

Providing a file with variable definitions

With the `-varFile` 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
```

If the above file was saved under the name `vars.txt`, you can use those variables by starting SQL Workbench/J using the following command line:

```
java -jar sqlworkbench.jar -varFile=vars.txt
```

Specifying variables directly

A single variable can be defined by passing the parameter `-variable`. This parameter can be supplied multiple times to define multiple variables:

```
java -jar sqlworkbench.jar -variable=foo=42 -variable=bar='xyz'
```

Note that the variable definition does not need to be quoted even though it contains the = character. Using `-variable=bar='xyz'` will include the single quotes in the variable value. The variable definition only needs to be quoted if it contains a space:

```
java -jar sqlworkbench.jar -variable="foo=hello world"
```

5.6. 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.



If this parameter is supplied, the workspace will not be saved automatically as well!

5.7. Connect using a pre-defined connection profile

You can specify the name of an already created connection profile on the command line with the `-profile=<profile name>` parameter. The name has to be passed exactly like it appears in the profile dialog (case sensitive!). 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='PostgreSQL - 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='PostgreSQL - Test'
     -profilegroup='Local'
     -script='test.sql'
```

You can also store the connection profiles in a properties file and specify this file using the `-profileStorage` parameter.

5.8. Connect without a profile

You can also specify the full connection parameters on the command line, 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.

Specifying all connection attributes

Parameter	Description
-url	The JDBC connection URL
-username	Specify the username for the DBMS
-password	Specify the password for the user If this parameter is not specified (but `-url` and `-username`) then you will be prompted to enter the password. To supply an empty password use `-password=` in the command line when starting SQL Workbench/J
-driver	Specify the full class name of the JDBC driver

Parameter	Description
-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.
-checkUncommitted	If this parameter is set to true, SQL Workbench/J will try to detect uncommitted changes in the current transaction when the main window (or an editor panel) is closed. If the DBMS does not support this, this argument is ignored. It also has no effect when running in batch or console mode.
-trimCharData	Turns on right-trimming of values retrieved from `CHAR` columns. See the description of the profile properties for details.
-removeComments	This parameter corresponds to the Remove comments setting of the connection profile.
-fetchSize	This parameter corresponds to the Fetch size setting of the connection profile.
-ignoreDropError	This parameter corresponds to the Ignore DROP errors setting of the connection profile.
-altDelimiter	This parameter corresponds to the Alternate delimiter setting of the connection profile.
-emptyStringIsNull	This parameter corresponds to the Empty String is NULL setting of the connection profile. This will only be needed when editing a result set in GUI mode.
-connectionProperties	This parameter can be used to pass extended connection properties if the driver does not support them e.g. in the JDBC URL. The values are passed as key=value pairs, e.g. `-connectionProperties=someProp=42` If either a comma or an equal sign occurs in a parameter's value, it must be quoted. This means, when passing multiple properties the whole expression needs to be quoted: `-connectionProperties='someProp=42,otherProp=24'`. As an alternative, a colon can be used instead of the equals sign, e.g `-connectionProperties=someProp:42,otherProp:24`. In this case no quoting is needed (because no delimiter is part of the parameters value). If any of the property values contain a comma or an equal sign, then the whole parameter value needs to be quoted again, even when using a colon. `-connectionProperties='someProp:"answer=42",otherProp:"2,4"'` will define the value `answer=42` for the property `someProp` and the value 2,4 for the property `otherProp`.
-altDelim	The alternate delimiter to be used for this connection. e.g. `-altDelimiter=GO\l` 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.
-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`.
-connectionName	When specifying a connection without a profile (only using `-username`, `-password` and so on) then the name of the connection can be defined using this parameter. The connection name will be shown in the title of the main window if SQL Workbench/J is started in GUI mode. The parameter does not have any visible effect when running in batch or console mode.
-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.
-readOnly	Puts the connection into read-only mode.

By specifying a simple connection string

A connection string is specified using the `-connection` parameter:

Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile).

The connection is specified with a comma separated list of key value pairs:

- `username` - the username for the connection
- `password` - the password for the connection
- `url` - the JDBC URL
- `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL
- `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured

e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"`

If an approriate driver is already configured the driver's classname or the JAR file don't have to be specified.

If an approriate driver is not configured, the driver's jar file must be specified:

```
-connection="username=foo,password=bar,url=jdbc:postgresql://localhost/mydb,driverjar=/etc
```

SQL Workbench/J will try to detect the driver's classname automatically (based on the JDBC URL).

If this parameter is specified, `-profile` is ignored.

The individual parameters controlling the connection *behaviour* can be used together with `-connection`, e.g. `-autocommit` or `-fetchSize`

In addition to `-connection` parameter, the following parameters are also supported to specify connections for `WbCopy`, `WbDataDiff` or `WbSchemaDiff`:

- -sourceConnection
- -targetConnection
- -referenceConnection

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

6. JDBC Drivers

6.1. 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 JDBC driver is a file with the extension `.jar` (some drivers need more than one file). See the end of this section for a list of download locations. Once you have downloaded the driver you can store the driver's .jar file anywhere you like.

JDBC drivers should **not** be stored in the `ext` directory.

To register a driver with SQL Workbench/J you need to specify the following details:

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

After you have selected the .jar file(s) for a driver (by clicking on the  button), SQL Workbench/J will scan the jar file looking for a JDBC driver. If only a single driver is found, the class name is automatically put into the entry field for the class name. If more than one JDBC driver implementation is found, you will be prompted to select one. In that case, please refer to the manual of your driver or database to choose the correct one.



SQL Workbench/J is **not** using the system's `CLASSPATH` definition (i.e. the environment variable named `CLASSPATH`) to load the driver classes. Changing the `CLASSPATH` environment variable to include your driver's library will have no effect.

If you enter the class name of the driver manually, remember that it's case-sensitive: `org.postgresql.driver` is something different than `org.postgresql.Driver`

Files that are not found are displayed in red and italics.

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. Some drivers are distributed in several jar files. In that case, select all necessary files in the file open dialog, or add them one after the other. If an entry is selected in the list of defined jar files when adding a new jar file, the selected entry will be overwritten.

For drivers that require a license file, you have to include the license jar to the list of files for that driver.

If the driver requires files that are not contained in the jar library, you have to include the directory containing those files as part of the library definition (e.g: `"c:\etc\TheDriver\jdbcDriver.jar;c:\etc\TheDriver"`).

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.

The JDBC/ODBC bridge is no longer available in **Java 8** and therefor it is not possible to connect through an ODBC data source when using SQL Workbench/J.

6.2. 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.

6.3. Popular JDBC drivers

Here is an overview of common JDBC drivers, and the class name that need to be used. SQL Workbench/J contains predefined JDBC drivers with sample URLs for connecting to the database.

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-42.2.8.jar` (exact name depends on the driver version) https://jdbc.postgresql.org/download
Firebird SQL	`org.firebirdsql.jdbc.FBDriver`	`firebirdsql-full.jar` https://www.firebirdsql.org/
H2 Database Engine	`org.h2.Driver`	`h2.jar` https://www.h2database.com
HSQLDB	`org.hsqldb.jdbcDriver`	`hsqldb.jar` https://hsqldb.org/
Apache Derby	`org.apache.derby.jdbc.EmbeddedDriver`	`derby.jar` https://db.apache.org/derby/
IBM DB2	`com.ibm.db2.jcc.DB2Driver`	`db2jcc4.jar` https://www.ibm.com/support/pages/db2-jdbc-driver-versions-and-downloads
IBM DB2 for iSeries	`com.ibm.as400.access.AS400JDBCDriver`	`jt400.jar` https://jt400.sourceforge.net/
SQL Server (Microsoft driver)	`com.microsoft.sqlserver.jdbc.SQLServerDriver`	`mssql-jdbc-6.2.2.jre8.jar` (exact name depends on the driver version) https://github.com/Microsoft/mssql-jdbc
Oracle	`oracle.jdbc.OracleDriver`	`ojdbc8.jar` https://www.oracle.com/technetwork/database/features/jdbc/index-091264.html
MariaDB	`org.mariadb.jdbc.Driver`	`mariadb-java-client-1.3.6.jar` (exact name depends on the driver version) https://downloads.mariadb.org/connector-java/
MySQL	`com.mysql.jdbc.Driver`	`mysql-connector-java-5.1.36-bin.jar` (exact name depends on the driver version) https://www.mysql.com/downloads/connector/j/

7. Connecting to the database

7.1. Connection profiles

SQL Workbench/J uses the concept of 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 ()

7.2. 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.

7.3. JDBC related profile settings

7.3.1. Driver

This is the class name 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.

7.3.2. URL

The connection URL for your DBMS. This value is 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.

7.3.3. Username

This is the name of the DBMS user account

You can use placeholders in the username property that get replaced with operating system environment variables or Java properties. E.g. `${user.name}` will be replaced with the current operating system user - this works on any operating system as the variable is supplied by the Java runtime. `${USERNAME}` would be replaced with the current username on Windows®. you can combine this with fixed text, e.g. `DEV_${user.name}` or `TEST_${user.name}`.

7.3.4. Password

This is the password for your DBMS user account. You can choose not to store the password in the connection profile.

7.3.5. Autocommit

This check box enables/disables the "auto commit" 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.

7.3.6. 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 especially on slow network connections.

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 (while disabling auto commit at the same time.

Especially when exporting large results using WbExport or WbCopy it is recommended to turn off the caching in the driver. A good starting point is setting this value to 100.

You can change the fetch size for the current connection manually by running the SQL Workbench/J specific command WbFetchSize

7.3.7. Timeout

This property defines a timeout in seconds that is applied when establishing the connection to the database server. If no connection is possible in that time, the attempt will be aborted. If this is empty, the default timeout defined by the JDBC driver is used.

7.4. PostgreSQL connections

When connecting to a PostgreSQL database it's not necessary to specify username and password. Username and password will then be resolved according to the rules as `psql` or any other libpq application would do:

Username

If no username is specified in the connection profile, SQL Workbench/J will first check the environment variable `PGUSER`, if that is not defined, the current operating system user will be used.

Password

If no password is specified and the saving of the password is disabled, SQL Workbench/J will first check the environment variable `PGPASSWORD`. If that is not defined, SQL Workbench/J will look for a Postgres password file. If that exists and the host, database, port and user are matched in the password file, the stored password will be used.

7.5. Extended properties for the JDBC driver

JDBC drivers support additional connection properties where you can fine tune the behavior of the driver or enable special features that are not switched on by default. Most drivers support passing properties as part of the URL, but sometimes they need to be passed to the driver using a different method called extended properties.

If you need to pass an additional parameter 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 double click in the table cell that you want to edit. To delete an existing property click on the `Delete` button ().

Some driver require those properties to be so called "System properties" (see the manual of your driver for details). If this is the case for your driver, check the option `Copy to system properties before connecting`.

7.6. SQL Workbench/J specific settings

7.6.1. Prompt for username

If this option is enabled (i.e. checked) you will be asked to enter the username and password each time you connect to the specified database server. If this is checked, the input fields for username and password are disabled (but the values will still be saved in the profile).

This is useful if you have many different usernames for the same DBMS server and don't want to create a connection profile for each user.

7.6.2. 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.

To enable the use of PostgreSQL's password file this option needs to be disabled.

7.6.3. Separate connection per tab

If this option is enabled, then each tab in the main window will open a separate (phyiscal) 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.

7.6.4. 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 18, “Using SQL Workbench/J in batch files” for details.

7.6.5. 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.

7.6.6. Hide warnings

When a SQL statement returns warnings from the DBMS, these are usually displayed after the SQL statement has finished. By enabling this option, warnings that are returned from the DBMS are never displayed.

Note that for some DBMS (e.g. MS SQL Server) server messages (`PRINT 'Hello, world'`) are also returned as a warning by the driver. If you disable this property, those messages will also not be displayed.

If you hide warnings when connected to a PostgreSQL server, you will also not see messages that are returned e.g. by the `VACUUM` command.

7.6.7. 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 detecting comments, this does **not** work for non-standard MySQL comments using the # character.

7.6.8. 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.

The read only state of the connection can temporarily be changed (without modifying the profile) using the WbMode command.

7.6.9. 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 altogether.

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.

The read only state of the connection can temporarily be changed (without modifying the profile) using the WbMode command.

7.6.10. 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.

7.6.11. Empty string is NULL

If this option is enabled, then a NULL value will be sent to the database for an empty (zero length) string. Everything else will be sent to the database as entered.

Empty values for non-character values (dates, numbers etc) are always treated as `NULL`.

If this option is disabled you can still set a column's value to `NULL` while editing a result set. Please see Editing data [57] for details

7.6.12. 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.

7.6.13. Check for uncommitted changes

This option is (currently) only available for PostgreSQL, HSQLDB 2.x and Oracle

When closing the application (or a SQL panel) SQL Workbench/J will check if the current transaction has changes that were not committed and will issue a warning.

For more details see the description of DBMS specific features.

7.6.14. 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.

7.6.15. Store completion cache locally

If this option is enabled, the cache that is used for the code completion is stored locally when the connection is closed. When connecting to the database the next time the cache is pre-filled with the information from the local cache file.

The cache files will be created in a directory named `.cache` inside the configuration directory.

7.6.16. 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.

7.6.17. 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. Please refer to using the alternate delimiter for details on this property.

7.6.18. 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.

If you specify a filename that does not contain a directory or is a relative filename, it is assumed the workspace is stored in workspace directory.



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.

7.6.19. Default directory

The default directory controls from where scripts are loaded through WbInclude, or where files are written by WbExport if filenames without directories are used. If nothing is specified, the "current" directory of the application is used for these files.

Note that once a an external file is loaded in a SQL tab, that file's directory becomes the default directory for that tab (the "Default directory" from the profile is not used).

File references stored in a a workspace can be made relative to this directory.

7.6.20. Main window icon

You can assign an icon file for each connection profile. The icon will then be used for the main window instead of the default application icon.

The icon file can **only** be a `png` or `gif` file.

Do **not** use an animated GIF file because that will hang the application!

At least one file with an icon of 16x16 pixel should be selected. You can select multiple files with different icon sizes (e.g. a 16x16 and a 32x32 icon). Whether or not the additional sizes (i.e. bigger than 16x16) will be used depends on your operating system and desktop theme.

7.6.21. macros

For each connection profile, a separate set of macros can be used. If no macro file is specified, the global macro file is used.

7.6.22. Tagging connection profiles

To organize a large number of profiles it is possible to supply tags for each profile. These tags are then used by the profile filter to quickly show only certain profiles.

The tags for a profile are entered as a comma separated list. To see a list of already defined tags, press **Ctrl-Space** in the input field.

7.6.23. 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.

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).

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.

7.6.24. Schema and Catalog filters

If your database contains a lot of schema or catalogs that you don't want to be listed in the dropdown of the DbExplorer, you can define filter expressions to hide certain entries.

The filters are defined by clicking on the Schema/Catalog Filter button. The filter dialog contains two input fields, one to filter schema name and one to filter catalog names.

Each line of the filter definition defines a single regular expression of schema/catalog names to be *excluded* from the dropdown, i.e. if a schema/catalog matches the defined name, it will not be listed in the dropdown.

The filter items are treated as regular expressions, so the standard SQL wildcards will not work here. The basic expression is just a name (e.g. MDSYS). Comparison is always done case-insensitive. So *mdsys* and *MDSYS* will achieve the same thing.

If you want to filter all schemas that start with a certain value, the regular expression would be: `^pg_toast.*`. Note the dot followed by a * at the end. In a regular expression the dot matches any character, and the * will allow any number of characters to follow. The ^ specifies that the whole string must occur at the beginning of the value.

The regular expression must match completely in order to exclude the value from the drop down.

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

7.6.25. Variables

It is possible to setup variables that are automatically defined when the connection for the profile is created. These variables will be removed after the connection is closed.

Variables defined in the connection profile are overwritten by variables defined for the workspace.

7.7. Connecting through a SSH tunnel

It is possible to setup a SSH tunnel to be used to connect to a database server that is not directly reachable (without using a external tool like Putty to create the tunnel).

To configure a SSH tunnel click on the SSH button in the connection profile. The configuration of the SSH tunnel consists of three parts:

- Login information for the computer from which the database server can be reached (the "SSH host")
- The mapping definition of a local port to the remote port of the database server
- Rewriting the JDBC URL to make the driver connect to "localhost" and the local port established by the SSH tunnel

7.7.1. Configure the remote login

Option	Description
Global SSH Host Configuration	You can select a pre-defined SSH Hosts to connect to. If a global SSH host is selected the individual elements of the SSH host can not be edited.
SSH Hostname	This is the remote computer that is used to establish the SSH connection and from which the remote database server is reachable
SSH Port	The port for the SSH connection. If this is empty the default port 22 is used.
Username	The username on the SSH host to be used for the login.
Private key file	If the SSH host is configured to use authentication through a SSH key file, the path to the key file can be entered here
Password	This is either the password for the remote user login, or, if a private key file is specified, the passphrase for that key file. If no password is provided, you will be prompted to enter one when connecting. If a password is entered, it is **always** stored in the profile. If no password is entered, you will be prompted for one when the connection is established. If the password is used as a passphrase for a private key file, the entered password will be remembered until the application is closed. It will be (re)used for other tunnels using the same private key file.
Authenticate through SSH agent	If this option is enabled SQL Workbench/J will try to use a running SSH agent to authenticate against the SSH host. Supported agents are `pageant.exe` for Windows® and `ssh-agent` for Linux systems. In this case supplying a private key file or a password is not necessary, but the user name is still required. In order to be able to communicate with the SSH agent, additional libraries are needed, which can be downloaded from https://www.sql-workbench.eu/jna.zip. Please unzip the archive and put the two jar files into the `ext` sub-directory where `sqlworkbench.jar` is located.

7.7.2. Configure the tunnel

Option	Description
Local port	The local port to be used for the tunnel. If this is empty a port will be chose automatically. Automatically choosing a local port only works for certain JDBC drivers where the structure of the JDBC URL is know and can be automatically rewritten.
DB Hostname	The hostname (or IP address) of the database server. This is the name as the SSH host sees the database server.
DB Port	The port of the database server.

If the option "Rewrite JDBC URL" is selected, SQL Workbench/J will try to adjust the JDBC URL of the connection profile to reflect the SSH settings. This only works for the following JDBC drivers:

- PostgreSQL
- Oracle
- Microsoft SQL Server using the Microsoft JDBC driver or the jTDS driver
- MySQL
- All drivers that use the format `jdbc:xxxxx://servername:port/...`

If rewriting the JDBC URL was not successful, a local port **must** be specified in the SSH configuration and the JDBC URL must be changed manually to use that port.

If no local port was given, the URL will contain the port 0 (zero) which will be replaced with the actual port that was chosen when the SSH tunnel is established.

Setting up the SSH tunnel is essentially the same as manually creating a SSH tunnel using:

```
ssh -p [SSHPort] -L[LocalPort]:[DB Hostname]:[DB Port] [SSH User]@[SSH Host]
```

on the commandline and then changing the JDBC URL to use `localhost` and the local port.



If you use a private key file with a strong encryption you might need to enable your Java installation to use that encryption. To do this, download the JCE extension for strong encryption from https://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html and install it into your Java runtime.

If multiple profiles use the same SSH tunnel definition SQL Workbench/J will only open one tunnel even if multiple windows using different profiles are used.

7.8. Connect to Oracle with SYSDBA privilege

Connecting to Oracle with SYSDBA privilege can be done by checking the option `as SYSDBA` next to the username. When using this option, you have to use an Oracle user account that is allowed to connect as SYSDBA (e.g. `SYS`).

7.9. Using the quick filter

The behaviour of the quick filter depends on whether tags are defined or not.

If no tags are defined at all, the quick filter will only search in the profile name. The search is done case-insensitive. Search for `prod` will match any profile that has `PROD` or `prod` anywhere in the profile's name

If tags are defined, the input is first checked if its one or more tags. If that is the case, the profiles are only filtered based on the tags defined. If there is a tag named prod, and the filter value is `prod`, only profiles with that tag are displayed. The profile name is not taken into account then. If the value in the filter field is not a tag, then the profiles are filtered based on name.

Multiple tags are separated by a comma. To see a list of defined tags, press the **Ctrl-Space** key.

8. Using workspaces

8.1. Overview

A workspace is a collection of editor tabs that group scripts or statements together. A workspace stores the name of each editor tab, the cursor position for each editor, the selection and the statement history.

Each connection profile is associated with a workspace. If no workspace is explicitely chosen for a connection profile, a workspace with the name `Default` is used. If not specified otherwise, workspaces are stored in the configuration directory.

A workspace file has the extension `.wksp` and is a regular ZIP archive that can be opened with any ZIP tool. It contains one text file for each editor in the workspace and some property files that store additional settings like the divider location, the Max. Rows value or the selected catalog and schema of the DbExplorer.

Starting with build 132 workspaces can be stored inside directories. When specifying a workspace without the `.wksp` extension, it is assumed that the name (or full path) is a directory. The content of each editor tab is stored in a separate text file. The workspace directory also includes several `.properties` files that contain additional information for each tab.



It is recommended to use a different workspace for each connection profile.

Workspaces can be used to reduce the number of editor tabs being used. You can create different workspaces for different topics you work on. One workspace that contains queries to monitor a database, one workspace that contains everything related to a specific feature you are working on, one workspace to initialize a new environment and so on.

The workspace is saved when the window using that workspace is closed. If the option "Auto-Save workspace" is enabled the workspace is automatically saved when a SQL statement is executed.

8.2. Creating a copy of the current workspace

To create a copy of the current workspace, use Workspace » Save workspace as After saving the workspace, the new workspace becomes the current workspace (the old one will not be changed). You will be asked if the new workspace should be the default profile's workspace, so that if you connect using that connection profile the new workspace will be loaded automatically.

If the new workspace is not made the profile's workspace, the next time you connect using that connection profile, the old workspace file will be loaded.

If you chose not to assign the new workspace right after saving it, you can later assign the currently loaded workspace to be used by the current connection profile using: Workspace » Assign Workspace.

This feature can be used if you have a workspace that contains statements that you want to use for a new topic, but you don't want to lose the original set of statements (that were used for a previous work).

This can also be used to convert workspaces from the WKSP format to the directory based format and vice versa.

8.3. Load a different workspace

If you want to load an existing workspace e.g. because you want to work on a different topic, you can use Workspace » Load Workspace Again you are asked if you want to use the newly loaded workspace as the default workspace.

Workspaces loaded through this will be put into the Workspace » Recent Workspaces menu so that you can quickly switch between workspaces you often use.

If you have a workspace loaded other than the default workspace of the current connection profile, you can quickly re-load the default workspace through Workspace » Re-Load Profile Workspace If you do that, the current workspace will be saved and the workspace assigned to the current connection profile will be loaded.

8.4. Workspace and external files

By default a workspace "remembers" the external files that were loaded. The content of the loaded file will also be stored in the workspace file. This can be configured in the Options dialog.

The reference to the external file can also be stored as relative paths, to make sharing of workspaces across multiple computers easier.

8.5. Workspace variables

It is possible to setup variables that are automatically defined when the workspace is loaded. These variables will be removed when the workspace is closed or a different workspace is loaded.

Variables defined in the workspace will overwrite variables defined in the connection profile.

9. Editing SQL Statements

9.1. 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 15MB) 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.

9.2. Code 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
  JOIN address a ON p.id = a.person_id;
```

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 behavior 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.

If the cursor is located inside the assignment of an `UPDATE` statement (`set foo = |, `) or in the `VALUES` part of an `INSERT` statement, the popup will contain an item `(Select FK value)`. When selecting this item the dialog to select a value from a referenced table will be displayed if the current column is referencing another table. For performance reasons the check if the current column is referencing another table is only done *after* the item has been selected. If no foreign key could be found, a message is displayed in the status bar.

The editor assumes that the standard semicolon is used to separate statements when doing auto-completion or using the "Execute current" function. This can be changed to a non-standard behaviour through the options dialog so that the editor also recognizes empty lines as a statement delimiter.

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.

9.3. Show hints for INSERT statements

When writing (long) `INSERT` statements it is often helpful to check if a specific value is actually written into the intended column. To check the column a value corresponds to (or the vice versa), press **Ctrl-#** while in the column or values list. A tool tip will appear to show the corresponding element from the "other" part of the statement. Consider the following statement:

```
INSERT INTO some_table (column1, column2, column3)
VALUES
('hello', 'world', 42, 'foobar');
```

When the cursor is located at `column1`, pressing **Ctrl-#** will show a tool tip containing the text `'hello'` as that is the value that corresponds to column1. When the cursor is located at the number `42` pressing **Ctrl-#** will show the text `column3` in the tool tip.

When no matching column or value can be found, the tool tip will contain a hint that the "other" element is missing. In the above example, pressing **Ctrl-#** while the cursor is located inside the string `foobar` it will show the message "No column for this value".

If the values inserted are the result of a `SELECT` statement, the tool tip in the insert column list will show the corresponding column name from the SELECT statement.

9.4. Customizing keyword highlighting

The keywords that the editor can highlight are based on an internal list of keywords and information obtained from the JDBC driver. You can extend the list of known keywords using text files located in the config directory.

SQL Workbench/J reads four different types of keywords: regular keywords (e.g. SELECT), data types (e.g. VARCHAR), functions (e.g. upper()) and operators (e.g. JOIN). Each keyword type is read from a separate file: `keywords.wb`, `datatypes.wb`, `functions.wb` and `operators.wb`.

The files contain one keyword per line. Case does not matter (`SELECT` and `select` are treated identically). If you want to add a specific word to the list of global keywords, simply create a plain text file `keywords.wb` in the config directory and put one keyword per line into the file, e.g:

```
ALIAS
ADD
ALTER
```

If you want to define keywords specific for a DBMS, you need to add the DBID as a prefix to the filename, e.g. `oracle.datatypes.wb`.

To add the word `geometry` as a datatype for the editor when connected to a PostgreSQL database, create the file `postgresql.datatypes.wb` in the config directory with the following contents:

```
geometry
```

The words defined for a specific database are added to the globally recognized keywords, so you don't need to repeat all existing words in the file.

The color for each type of keyword can be changed in the options dialog.

9.5. 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 being. 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.

9.6. 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 » Code tools » 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 » Code tools » 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.

9.7. 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).

9.7.1. 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 » Code tools » 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.

9.7.2. 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 » Code tools » 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;
```

9.7.3. 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 » Settings » 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 preserved, 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 11g driver (`ojdbc6.jar`, `ojdbc7.jar`)
- Microsoft SQL Server 2000/2005 driver (`sqljdbc4.jar`)

10. Working with bookmarks

10.1. Defining bookmarks

A bookmark inside the editor is defined by adding the keyword `@WbTag` followed by the name of the bookmark into a SQL comment:

```
-- @WbTag delete everything
truncate table orders,order_line,customers;
commit;
```

The keyword is not case sensitive, `@wbtag` will work just as wel as `@WBTAG`, or `@WbTag`. A multiline comment can be used as well as a single line comment.

The annotations for naming a result can additionally be included in the bookmark list. This is enabled in the options panel for the editor.

The names of procedures and functions can also be used as bookmarks if enabled in the bookmark options

10.2. Jumping to a bookmark

To jump to a bookmark select Tools » Bookmarks. A dialog box with all defined bookmarks will be displayed. You can filter the list of displayed bookmarks by entering a value in the input field. Depending on the option Filter on name only the value will either be compared only against the bookmark name. If that option is disabled then the bookmark name and the name of the SQL tab will be checked for the entered value.



The selection in the bookmark list can be moved with the **UP/DOWN** keys even when the cursor is located in the filter input field.

If the option Only for current tab is enabled, then the dialog will open showing only bookmarks for the current tab.

10.3. Configuring the display of the bookmark list

There are two options to influence how the list of bookmarks is displayed. Both options are available when displaying the context menu for the list header (usually through a click with the right mouse button):

- Remember column widths - if this is enabled, the columns are not automatically adjusted to the width of the content. Instead the list remembers whatever width was adjusted.
- Remember sort order - by default the list is sorted by the name of the bookmark. When this option is selected, whatever sort order is selected (by clicking on the column headers) will be saved and restored the next time the dialog is opened.

11. Creating stored procedures and triggers

SQL Workbench/J will split statements based on the SQL terminator `;` and send each statement unaltered to the DBMS.

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

To be able to run DDL statements with embedded `;` characters, SQL Workbench/J needs to know where a statements ends. To specify the end of a statement with embedded `;` the so called "alternate delimiter" is used. This chapter describes how the alternate delimiter is used by SQL Workbench/J

11.1. PostgreSQL

The body of a function in Postgres is a character literal. Because a delimiter inside a character literal does not define the end of the statement, no special treatment is needed for Postgres.

11.2. Oracle PL/SQL

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
  l_result INTEGER;
BEGIN
  SELECT max(col1) INTO l_result FROM sometable;
  RETURN l_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 that is called the "alternate delimiter" which can be defined in the connection profile. To be compatible with SQL Developer and SQL*Plus it is recommended to set the alternate delimiter to a forward slash (`/`).

The script needs to be written like this:

```
CREATE OR REPLACE FUNCTION proc_sample RETURN INTEGER
IS
  l_result INTEGER;
BEGIN
  SELECT max(col1) INTO l_result FROM sometable;
  RETURN l_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.

The standard delimiter (the semicolon) and the alternate delimiter can be mixed in a single script. Whenever a PL/SQL block (either a stored procedure or an anonymous block) is encountered, SQL Workbench/J expects the alternated delimiter to terminate that block. This follows the same rules as used in SQL*Plus.

The following script will therefore work when connected to an Oracle database:

```
drop table sometable cascade constraints;
create table sometable
(
  col1 integer not null
);

create or replace function proc_sample return integer
is
  l_result integer;
begin
  select max(col1) into l_result from sometable;
  return l_result;
end;
/
```

11.3. Using a static alternate delimiter

When is the alternate delimiter used?

For all other DBMS, the use of the alternate delimiter is defined by the **last delimiter** used in the script.

As soon as the statement (or script) that you execute ends with the alternate delimiter, the alternate delimiter is used to separate all 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).

This means a script must use the alternate delimiter for **all** statements in the script. The following script will not work, because the last statement is terminated with the alternate delimiter and thus SQL Workbench/J assumes *all* statements are delimited with that. As the `CREATE TABLES` statements are delimited with the standard delimiter, they are not recognized as a separate statement and thus the script is sent as a single statement to the server.

```
create table orders
(
  order_id    integer not null primary key,
  customer_id integer not null,
  product_id  integer not null,
  pieces      integer not null,
  order_date  date    not null
);

create table orders_audit_log
(
  order_id    integer not null,
  delete_date timestamp not null
);

create trigger orders_audit_log
  for orders
  before delete
as
begin
  insert into audit_log (id, delete_date) values (old.order_id, current_timestamp);
end;
```

```
/
```

The solution is to terminate **every** statement with the alternate delimiter:

```
create table orders
(
  order_id    integer not null primary key,
  customer_id integer not null,
  product_id  integer not null,
  pieces      integer not null,
  order_date  date    not null
)
/

create table orders_audit_log
(
  order_id    integer not null,
  delete_date timestamp not null
)
/

create trigger orders_audit_log
  for orders
  before delete
as
begin
  insert into audit_log (id, delete_date) values (old.order_id, current_timestamp);
end;
/
```

11.4. Using a dynamic delimiter

The SQL Workbench/J specific command `WbDelimiter` can be used to change the currently used delimiter dynamically inside a script.



Using `WbDelimiter` only works if no alternate delimiter is configured for the current connection - neither through the connection profile nor through the global options.

The `WbDelimiter` keyword must be written as the only command in a line and followed by delimiter to be used after this line. The specified delimiter must then be used on a line of its own to terminate a statement.

Using `WbDelimiter` is not necessary when running Oracle scripts that mix SQL and PL/SQL as the correct delimiter is automatically detected already.

```
create table orders
(
  order_id    integer not null primary key,
  customer_id integer not null,
  product_id  integer not null,
  pieces      integer not null,
  order_date  date    not null
);

create table orders_audit_log
```

```
(
  order_id    integer not null,
  delete_date timestamp not null
);

WbDelimiter ^

create trigger orders_audit_log
  for orders
  before delete
as
begin
  insert into audit_log (id, delete_date) values (old.order_id, current_timestamp);
end;
^

WbDelimiter ;

create table customer
(
  customer_id integer not null primary key,
  name        varchar(100) not null
);

insert into customer (customer_id, name)
values
(1, 'Arthur'),
(2, 'Ford');
```

12. Using SQL Workbench/J

12.1. Displaying help

You have two possibilities to display help for SQL Workbench/J: a HTML a PDF version of the manual.

The HTML help is available through the menu item Help » Contents It is expected that the HTML manual is stored in a directory called `manual` in the same directory where `sqlworkbench.jar` is located. This is automatically the case when you extract the distribution archive with sub-directories.

You can choose to display a single-page version of the HTML help (easier to search) or a multi-page version of the help that is easier to navigate. This can be changed in the options dialog, that is accessible from Tools » Option.

The PDF manual can be displayed 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 section of the options dialog.

The file `SQLWorkbench-Manual.pdf` must be available in the directory where `sqlworkbench.jar` is located.

When connected to a database, the menu item Help » DBMS Manual will display the online manual for the current DBMS (if there is one). Where possible the link will display the manual that corresponds to the version of the current connection.

The URL that is used to display the manual can be changed in the configuration file `workbench.settings`.

12.2. Resizing windows

Every window that is opened by SQL Workbench/J for the first time is displayed with a default size. In certain cases it can happen that not all labels are readable or all controls are visible on the window. This can happen, e.g. when a large default font is selected (or defined through the look and feel).

Every window in SQL Workbench/J can be resized and will remember its size. So in case not everything is readable on a dialog, just resize the window so that the missing parts become visible, and that size will be kept for the future.

12.3. Executing SQL statements

12.3.1. 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 the editor to use the statement that is defined by the current *line* rather than the cursor location when using Execute current.

Consider the following editor contents:

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

If the option to use the current line is **disabled** and the cursor is located after the semicolon in the third line, Execute current will execute the `SELECT` statement because the cursor is logically located in the statement after the `select`.

If that option is **enabled** and the cursor is located after the semicolon in the third line, Execute current will execute the `DELETE` statement because the statement in the current line is the `select` statement. If there are multiple SQL statements in the current line, the first statement will be executed.

You can configure SQL Workbench/J to automatically jump to the next statement, after executing the current statement. Simply select SQL » Settings » Auto advance to next 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

When executing all statements in the editor you have to delimit each statement, so that SQL Workbench/J can identify each statement. If your statements are not delimited using a semicolon, the whole editor text is sent as a single statement to the database. Some DBMS support this (e.g. Microsoft SQL Server), but most DBMS will throw an error in that case.

A script with two statements could look like this:

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

or:

```
DELETE FROM person;
DELETE FROM address;
COMMIT;

INSERT INTO person
(id, firstname, lastname)
VALUES
(1, 'Arthur', 'Dent');

INSERT INTO person
(id, firstname, lastname)
VALUES
(4, 'Mary', 'Moviestar');

INSERT INTO person
(id, firstname, lastname)
VALUES
(2, 'Zaphod', 'Beeblebrox');

INSERT INTO person
(id, firstname, lastname)
```

```
VALUES
(3, 'Tricia', 'McMillian');

COMMIT;
```

You can specifiy an alternate delimiter that can be used instead of the semicolon. See the description of the alternate delimiter for details. This is also needed when running DDL scripts (e.g. for stored procedures) that contain semicolons that should not delimit the statements.

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.

JDBC drivers do not support multi-threaded execution of statements on the same physical connection. If you want to run two statements at the same time, you will need to enable the Separate connection per tab option in your connection profile. In this case SQL Workbench/J will open a physical connection for each SQL tab, so that statements in the different tabs can run concurrently.

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 » Editor history back statement from the menu. Once the previous statement(s) have been recalled the next statement can be shown using **Alt-Right** or choosing SQL » Editor history next 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.

12.4. 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 (labeled "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.



SQL Workbench/J will read all rows returned by your statement into memory. When retrieving large results you might run out of memory. To adjust the memory available to SQL Workbench/J please refer to this chapter.

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 » Settings » 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 configure the default behavior for new editor tabs in the options dialog.

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

12.4.1. Limiting result sizes

To prevent retrieving an large amount of rows (and possibly running out of memory), the maximum number of rows that are retrieved can be defined for each SQL panel in the "Max. Rows" input field of the status bar. This value will be stored in the workspace that is associated with the connection profile.

A default value that will be used for newly opened SQL tabs can be defined in the options dialog.

12.4.2. Displaying values with embedded newlines

Data from `VARCHAR` or `CHAR` columns is displayed as a single-line if the column's max. size is below 250 characters. If you have data in smaller columns that contains newlines (line breaks) and you want to display directly in the result set, please adjust the limit to match your needs. The limit can be changed in the Data Display Options.

12.4.3. Naming result tabs

There are two ways to assign a name to the result tab of a query:

- By specifying the annotation `@WbResult`. For details please see the chapter about annotations
- You can automatically use the first table name from a `SELECT` statement to be used for the result name in the Data display options.

12.4.4. Displaying results as text

By default results are displayed in a scrollable table. You can enable the display of results as text. In that case the results will be displayed in the messages area, formatted in the same way as the console mode. This display can be enabled through SQL » Settings » Show results as text or through the annotation @WbResultAsText.

12.5. 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.

12.5.1. Updating BLOB 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 appropriate `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`

12.5.2. Updating CLOB 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.

12.5.3. 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).

12.5.4. 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 drop down. To open the BLOB content with one of the tools, select the tool from the drop down list, then click on the button Open with next to the external tools drop down. SQL Workbench/J will then retrieve the BLOB data from the server, store it in a temporary file on your hard disk, 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.

12.5.5. Displaying the contents of a BLOB column as an image

By default the content of BLOB columns won't be displayed in a result. If it is known that a column contains image data, a preview of the image can be enabled. In the context-menu (Right-Click) of a BLOB column's header, the option Show content as image will enable displaying that column as an image. When selecting that menu item, a prompt to specify the preview size is shown. If a single value is given it is assumed to be width and height of the preview. To specify width and height individually, enter two values separated by a comma: e.g.: `75,50`.

If this is enabled, the row height will be optimized automatically, even if this has been been disable globally.

The image preview can also be turned on using an annotation .

12.6. 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.

12.7. 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. You can prevent files from being put into the history by unchecking the option "Files in history" in the Editor section of the options dialog.

12.8. Displaying the structure of tables

The command `describe` can be used to display the structure of a view or table. You can also display information about the database object at the cursor by using SQL » Object info. This function is also available in the context menu of the editor.

When the menu item is invoked using the mouse, holding down the **CTRL** key will return dependent object information as well (e.g. indexes, foreign keys).

You can configure this function to always include dependent objects by adding a configuration property.

12.9. Viewing server messages

12.9.1. PostgreSQL

PostgreSQL supports sending of messages to the client using the `RAISE` statement in PL/pgSQL. The following function will display a result set (with the number 42) and the message area will contain the message *Thinking hard...*

```
CREATE OR REPLACE FUNCTION the_answer()
  RETURNS integer
  LANGUAGE plpgsql
AS
$body$
BEGIN
  RAISE NOTICE 'Thinking hard...';
  RETURN 42;
END;
$body$
```

12.9.2. 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 `set serveroutput on` 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
```

12.9.3. 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.
```

Due to the way the JDBC API works, the messages are only show *after* the statement has finished (this is different to e.g. SQL Server Management Studio where the messages are displayed as soon as `PRINT` is called, even when the overall script or procedure is still running.

12.9.4. 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.eu>.

12.10. Editing 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 is found, the unique indexes for the table will be retrieved. The first unique index found that only consists of columns defined as NOT NULL will be used.

If no PK or unique index can be found, the custom PK Mapping will be checked. 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, new or deleted rows) will not be saved to the database until you choose Data » Save Changes to Database.

If the update is successful (no database errors) a `COMMIT` will automatically be sent to the database (if autocommit is turned off).

If your `SELECT` was based on more than one table, you will be prompted to specify which table should be updated. It cannot be detected reliably which column belongs to which of the tables from the select statement. When updating a result from multiple tables, only columns from the chose update table should be changed, otherwise incorrect SQL statements will be generated.

If no primary (or unique) key can be found for the update table, you will be prompted to select the columns that should be used to uniquely identify a row in the update table.

If an error is reported during the update, a `ROLLBACK` will automatically 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

If the option Empty String is NULL is disabled for the current connection profile, you can still set a column's value to null when editing it. To do this, double click the current value, so that you can edit it. In the context menu (right mouse button) the option "Set to NULL" is available. This will clear the value and set it to `NULL`. You can assign a shortcut to this action, but the shortcut will only be active when editing a value inside a column.

12.11. 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.

12.12. 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. After the first click 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 additional column. To switch 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.

By default SQL Workbench/J will use "ASCII" sorting which is case-sensitive and will not sort special characters according to your language. You can change the locale that is used for sorting data in the options dialog under the category "Data Display". Sorting using a locale is a bit slower than "ASCII" sorting.

12.13. Filtering the result

Once the data has been retrieved from the server it can be filtered without re-retrieving it. You can define the filter in two ways: either select the filter columns and their filter values manually, or create a filter from the currently selected values in the result set.



The filter is applied on the data that is retrieved from the database. The data will **not** be reloaded from the database when you define a filter.

12.13.1. 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 drop down. If you select `*` for the column, the filter condition will be applied to all columns of the result set.



The value expression for a column does not accept SQL expressions! You can only compare the column to a constant, not to the result of a SQL function (such as `CURRENT_DATE` or `now()`) If you need this kind of filter, you have to use a SQL statement with the approriate `WHERE` condition.

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.

12.13.2. 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.

12.14. Running stored procedures

Stored procedures can be executed by using the SQL Workbench/J command `WbCall` which replaces the standard commands available for the DBMS (e.g. `CALL` or `EXECUTE`). By using a special command, additional checks can be carried out by SQL Workbench/J. This is especially necessary when dealing with OUT parameters or REF CURSORS.

The simplest way to run a stored procedure is:

```
WbCall my_proc();
```

When using Microsoft SQL Server, WbCall is not necessary as long as the stored procedure does not have OUT or REF CURSOR parameters. So with SQL Server you can simply write:

```
sp_who2;
```

To run the stored procedure `sp_who2` and to display it's results.

For more details on running a stored procedure with `OUT` parameters or `REF CURSORS` please refer to the description of the WbCall command.

12.15. 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)

In order to write the proprietary Microsoft Excel format, additional libraries are needed. Please refer to Exporting Excel files for details.

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.

12.16. Copy data to the clipboard

You can also copy the data from the result into the system clipboard in four different formats.

- 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)

- DbUnit XML

  For this option to be available `DbUnit`, `Log4j` and `slf4j` libraries must be copied into the same directory where `sqlworkbench.jar` is located.

  The following libraries are needed:

  - `dbunit-2.3.0.jar` (or later)
  - `slf4j-api-1.7.7.jar` (or later)
  - `slf4j-log4j12-1.7.7.jar` (or later)
  - `log4j-1.2.15.jar` (or later)

  You can also use WbExport together with the -stylesheet parameter and the suppplied stylesheets `wbexport2dbunit.xslt` and `wbexport2dbunitflat.xslt` to generate DbUnit XML files from data already present in the database (in that case no DbUnit libraries are needed).

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.

12.17. Import data into the result set

12.17.1. 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 appropriate `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)

Option	Description
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.

12.17.2. 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 Clipboard 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 mainly necessary if the delimiter is not the tab character. You can manually open that dialog, by holding down the **Ctrl** key when clicking on the menu item.

13. Using SQL Workbench/J specific annotations in SQL comments

By adding special comments to a SQL statement, you can influence the way the result is displayed in SQL Workbench/J. These comments are called annotations and must be included in a comment preceding the statement for which it is intended. The comment can be a single line or multi-line SQL comment



Annotations are used for the statement that **immediately** follows the comment that contains the annotation.

13.1. Naming result tabs

You can change the name of the result tab associated with a statement.

To give a resultset a name, use the annotation `@WbResult` followed by the name that should appear as the result's name in a comment directly preceding the statement.

The following examples executes two statements. The result for the first will be labelled "List of contacts" and the second will be labelled "List of companies":

```
-- @WbResult List of contacts
SELECT * FROM person;

/*
@WbResult List of companies
this will retrieve all companies from the database
*/
SELECT * FROM company;
```

The result name that is used, will be everything after the annotation's keyword until the end of the line.

For the second select (with the multi-line comment), the name of the result tab will be `List of companies`, the comment on the second line will not be considered.

13.2. Adding macros to the result's context menu

The annotation `@WbMacro` can be used to add macros to the context menu of the result of the query that immediately follows the comment with the annotation.

When such a macro is executed, the values of all columns of the currently selected row will be defined as variables that are usable in the macro. If the macro produces a result, it will always be appended as a new tab, regardless of the setting in the macro definition.



Variables which are defined by invocation of a macro from this menu item will not be deleted after the macro has been executed.

This annotation has no effect in console mode.

Assume the following macro with the name `"Person Address"`:

```
select *
from address
where person_id = $[id];
```

and the following SQL query:

```
-- @WbMacro name="Person Address"
select id, firstname, lastname
from person;
```

The context menu of the result will then contain a new submenu: Macros » Person Address. The variables `$[id]`, `$[firstname]` and `$[lastname]` will contain the values of the currently selected row when the macro is executed.

It is also possible to re-map the column names to different variable names.

```
-- @WbMacro name="Person Address" map="id:PersonID"
select id, firstname, lastname
from person;
```

In this case a variable named `PersonID` will be created with the value of the `id` column from the selected row.

The `map` parameter can be repeated several times to re-map multiple columns, e.g. `map=p_id:PersonID map=o_id:OrderID`

It is possible to specify more than one macro for the context menu:

```
-- @WbMacro name="Person Address" map="id:PersonID"
-- @WbMacro name="Customer Orders" map="id:PersonOrderID"
select id, firstname, lastname
from person;
```

A macro can only be executed from the menu when exactly one row is selected in the result.

You can assign a title to the result by using the `@WbResult` annotation based on a variable in the macro:

```
-- @WbResult Addresses for $[firstname] $[lastname]
select *
from address
where person_id = $[PersonID];
```

13.3. Re-using an existing named result tab

If the result of a query should be displayed in an existing result tab, the annotation `@WbUseTab` together with a tab name can be used. If this annotation is present in the comment immediately preceding a query and a result tab with that name already exists, the existing result will be replaced with the new result. If no result tab with that name exists, a new tab (with the supplied name) will be created.



Re-using a result tab only works if SQL » Append new results is enabled. You can combine `@WbUseTab` with the `@WbAppendResult` annotation to force re-using an existing result even though the option is turned off.

If the following query is run for the second time, the existing data will be replaced with the newly retrieved data:

```
-- @WbUseTab List of contacts
SELECT * FROM person;
```

This annotation has no effect in console mode.

13.4. Scrolling the result

The annotation `@WbScrollTo` can be used to automatically scroll a result set after it has been retrieved to a specific row number. The row number has to be supplied using a # sign:

```
-- @WbScrollTo #100
SELECT *
FROM person;
```

In addition to a row number, the special values `end` or `last` (without a #) are also recognized. When they are supplied, the result is automatically scrolled to the last row. This is useful when displaying the contents of log tables.

```
-- @WbScrollTo end
SELECT *
FROM activity_log;
```

This annotation has no effect in console mode.

13.5. Appending a results

The annotation `@WbAppendResult` can be used to always append the result of the associated query regardless of the current setting of SQL » Append new results.

This annotation has no effect in console mode.

13.6. Suppressing empty results

To suppress an empty result, the annotation `@WbRemoveEmpty` can be used. If a query returns no rows and contains this annotation, no result tab will be created. No warning or message will be shown if this happens!

This annotation has no effect in console mode.

13.7. Automatic refresh of the result

To automatically refresh a result in a defined interval, the `@WbRefresh` annotation can be used. The interval is specified as a parameter to the annotation:

```
-- @WbRefresh 15s
SELECT *
FROM pg_stat_activity;
```

The automatic refresh can also be enabled through the context menu of the result tab.

This annotation also works in console mode.

13.8. Optimize row height

If the automatic optimization of the row height is disabled, the annotation `@WbOptimizeRowHeight` can be used to optimize the height of the rows for a specific SQL query.

The annotation supports the parameter `lines` that defines the maximum number of lines to display, even if the content was longer.

```
-- @WbOptimizeRowHeight lines=10
select id, json_payload
from messages
order by id;
```

This annotation has no effect in console mode.

13.9. Show BLOB column as image

If a BLOB column of a query result contains image data (PNG, JPEG, ...) this annotation can be used to automatically display a preview of that image within the result. The annotation has two parameters.

The parameter `column` is mandatory and defines which column of the result should be treated as an image column.

The second parameter `previewSize` defines the size of the preview image. If a single value is given it is assumed as the size of a square, e.g. `previewSize=50`. To specify width and height of the preview, separate the values with a comma: `previewSize=75,50`.

Using this annotation does not automatically adjust the row height if that hasn't been enabled globally. The automatic row height can also be enabled using @WbOptimizeRowHeight.

```
-- @WbImagePreview column=image previewSize=50
select id, image
from some_table
order by id;
```

This annotation has no effect in console mode.

13.10. Keep result

The annotation `@WbKeepResult` will "lock" a result tab, i.e. the result will never be closed automatically. This can also be enabled through the context menu of a result.

This annotation has no effect in console mode.

13.11. Result as text

The annotation `@WbResultAsText` will display the result of the SQL query in text mode in the messages area rather than a scrollable table.

The annotation can be used with or without a parameter. If no parameter is specified, it will turn text mode on for the statement immediately following the annotation.

If the parameter `on` is specified, the text mode will be kept until the script finishes or it's turned off again using `@WbResultAsText off`.

Assuming the following script:

```
-- @WbResultAsText
select * from table_one;
```

```
select * from table_two;

-- @WbResultAsText on
select * from table_three;
select * from table_four;

-- @WbResultAsText off
select * from table_five;
select * from table_six;
```

The first result ("table_one") will be shown in text mode. The second result ("table_two") will be shown as a table. The results for table_three and table_four will again be shown in text mode and the last two results will be shown as tables again.

This annotation has no effect in console mode.

13.12. Create crosstab (pivot) from a result

The annotation `@WbCrossTabl` can automatically transpose rows to columns from a result of a query. This is also known as "crosstab" or "pivot".

Assuming the following statement

```
select department_id as dept,
       count(*) as "Employees",
       avg(salary)::int as "Avg. Salary",
       min(salary)::int as "Min. Salary"
from employees
where department_id in (10,20,30)
group by department_id
order by department_id;
```

Returns the following result:

```
dept | Employees | Avg. Salary | Min. Salary
-----+-----------+-------------+------------
  10 |         1 |        4400 |        4400
  20 |         2 |        9500 |        6000
  30 |         6 |        4150 |        2500
```

Then the result can be turned into a crosstab by adding the annotation `@WbCrossTab`. The annotation supports two parameters:

`labelColumn` - to define the column that should be used for the header's label
`addLabel` - an additional text that will be added to the value of the labels

```
-- @WbCrossTab labelColumn=dept addLabel="Dept "
select department_id as dept,
       count(*) as "Employees",
       avg(salary) as "Avg. Salary",
       min(salary) as "Min. Salary"
from employees
where department_id in (10,20,30)
group by department_id
order by department_id;
```

Returns the following result:

```
             | Dept 10 | Dept 20 | Dept 30
-------------+---------+---------+--------
Employees    | 1       | 2       | 6
Avg. Salary | 4400    | 9500    | 4150
Min. Salary | 4400    | 6000    | 2500
```

If this is used to export a crosstab using `WbExport`, the data is **always** read completely into memory in order to be able to generate the crosstab - even if the query would normally be streamed directly to the export file.

14. Using macros and text clips

SQL macros and text clips can help writing and executing SQL statements that you use frequently.

There are two types of macros:

- executable macros
- expandable macros

Executable macros are intended for complete SQL statements that are executed once you select/run the macro. They can also be used as an abbreviated SQL statement, by typing the macro's name and executing this as a SQL statement.

Expandable macros are intended for SQL fragments (or "clips"). The text of the macro is inserted into the editor if the name is typed or the macro is selected from the menu.

14.1. Loading and saving macro sets

By default SQL Workbench/J will use a file with the name `WbMacros.xml` stored in the configuration directory to save and load the macros.

It is possible to use a different set of macros for each connection profile. In that case the global/default macros are not loaded, only those specified in the connection profile.

To create a copy of the currently loaded macros, use Macros » Save Macros as.... To load previously saved macros, use Macros » Load Macros....

The currently loaded file is displayed as a tool tip of the Save Macros as... menu item and and the bottom of the Manage Macros dialog.

Starting with build 129 macros can optionally be stored as SQL scripts in a directory, to make sharing of the SQL statements in the macros easier. Each sub-directory in the selected directory represents one macro group.

When using the directory based storage, it's possible to use symbolic links in the filesystem to make a macro group available for multiple sets of macros. This can be used to e.g. include multi-purpose scripts in a set of macros specific to one DBMS.

14.2. Defining Macros

There are three 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. You can put macros into separate groups (e.g. one for PostgreSQL macros, one for Oracle etc). If you have only one group defined (or only one visible group), all macros of that group will be listed in the menu directly. If you define more than one group, each group will appear as a separate sub-menu.

Macros can also be defined using the command `WbDefineMacro`.

When the dialog is closed using the OK button the macros are automatically saved to the current file.

The order in which the macros (or groups) appear in the menu can be changed by dragging them to the desired position in the manage macro dialog.

14.3. Executable macros

There are two ways to run an executable macro: use it's name as a SQL command by typing it into the editor and executing it like any other SQL statement. Or by selecting the corresponding menu entry from the Macros menu.

Note that the macro name needs to be unique to be used as a "SQL Statement". If you have two different macros in two different macro groups with the same name, it is undefined (i.e. "random") which of them will be executed.

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.

In console mode you can use the command `WbListMacros` to show the complete list of macros (of course this can also be used in GUI mode as well.



Macros will not be evaluated when running in batch mode.

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 [50]
${text}$	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 parameters cannot correctly be executed by entering the macro alias in the SQL editor (and then executing the "statement").



The parameter keywords are case sensitive, 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
${current_statement}$
;

-- @wbResult Execution plan
select plan_table_output
from table(dbms_xplan.display(format => 'ALL'));
```

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 (analyze true, verbose true, buffers true) ${current_statement}$;
```

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
(
${current_statement}$
)
```

14.4. Expandable macros

Expandable macros are not intended to be run directly. They serve as code templates for writing statements.

When typing the name of the macro in the editor and completing this name with the "Macro expansion key", the typed word will be replaced with the macro's text. The name of a such a macro is not case sensitive. So `slt` and `SLT` are detected as the same macro name.

The macro expansion is only triggered if the macro expansion key is typed quickly after the word. If there is a longer pause between typing the last character of the macro's name and typing the expansion key, the macro will not be expanded.

For expandable macros, two special place holders in the macro text are supported. Both place holders are deleting when the macro text is inserted.

Parameter	Description
${c}	This parameter marks the location of the cursor after the macro is expanded.
${s}	This parameter also marks the position of the cursor after expansion. Additionally the word on the right hand side of the parameter will automatically be selected.

14.5. DbTree macros

The third type of macros is intended to be used in the DbTree. The macros will be available from the context menu for tables and columns in the DbTree.

Inside the macro definition the following parameters will be replaced:

Parameter	Description
${table}$	This will be replaced with the qualified name of the selected table. Depending on the context, this might be replaced with a fully qualified name (including schema or database).
${table_name}$	This will be replaced with the name of the table. No qualifying prefix (schema, database) will be used.
${schema_name}$	This will be replaced with the name of the schema in which the table is defined.

Parameter	Description
${catalog_name}$	This will be replaced with the name of the catalog in which the table is defined. `${db_name}` is an alternative writing for this placeholder.
${column_list}$	This will be replaced with a comma separated list of the names of all selected columns. The order in which the columns are listed might depend on the used Java version and operating system.
${column}$	This will be replaced with the name of selected column if exactly one column is selected.
${column_N}$	This will be replaced with the name of n-th select column if more than one column is selected. So `${column_1}$` will be replaced with the first column, `${column_2}$` with the second column and so on. The order in which the columns are listed might depend on the used Java version and operating system.

15. Working with foreign keys

15.1. 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

15.2. Generating JOIN conditions

When using ANSI JOIN syntax to create table joins with tables linked by foreign keys in the database, the command `JOIN completion` can be used to automatically generate the necessary join condition. Consider the following statement

```
SELECT ord.amount, ord.order_date, prod.name
FROM orders ord
  JOIN product prod ON |
```

(the | denoting the location of the cursor).

When the cursor is located behind the `ON` keyword and you select SQL » JOIN completion, SQL Workbench/J will retrieve the foreign key and corresponding primary key definitions between the tables `orders` and `product`. If such constraints exist, the corresponding condition will be generated and written into the editor. After executing JOIN completion, the SQL statement will look like this:

```
SELECT ord.amount, ord.order_date, prod.name
FROM orders ord
  JOIN product prod ON prod.id = ord.product_id
```

This feature requires the usage of the JOIN keyword. Joining tables in the `WHERE` clause is not supported.

By default SQL Workbench/J tries to create a join condition on the table from the "previous" `JOIN` condition (or the `FROM`) clause. If no foreign key constraint is found linking the "current" and the "previous" table, a popup window with all tables in the select statement that could be used for completion is displayed. This popup merely looks at the tables in the statement, no test for foreign key constraints is done when displaying this list.

You can configure this feature to generate a `USING` operator if the column names match. The case of the keywords in the generated condition is determined by the settings of the SQL Formatter.

15.3. Selecting foreign key values in referencing tables

SQL Workbench/J supports the selection of foreign key values (i.e. the primary key values of the referenced table) in two different situations: while editing a result set and while writing a DML statement.

15.3.1. Editing foreign key values

After starting to edit a cell, the context menu contains an item Select FK value. Once this item is selected SQL Workbench/J will detect the table that the current column references. If a foreign key is detected a dialog window will be shown containing the data from the referenced table. For performance reasons the check if the current column is referencing another table is only done *after* the menu item has been invoked. If no foreign key could be found, a message is displayed in the status bar.



This is only supported for result sets that are based on a single table.

By default the dialog will not load more than 150 rows from that table. The number of retrieved rows can be configured through the "Max. Rows" input field.

There are two ways to find the desired target row which can be selected using the radio buttons above the input field.

- Applying a filter

  This mode is intended for small lookup tables. All rows are loaded into memory and the rows are filtered immediately when typing. The typed value is searched in all columns of the result set. Clicking on the reload button will always re-retrieve all rows.

- Retrieving data

  This mode is intended for large tables where not all rows can be loaded into memory. After entering a search term and hitting the **ENTER** key (or clicking on the reload button), a SQL statement to retrieve the rows containing the search statement will be executed. The returned rows are then displayed.

Once you have selected the desired row, clicking the OK will put the value(s) of the corresponding primary key column(s) into the currently edited row.

15.3.2. Selecting FK values when editing DML statements

When invoking code-completion inside a DML (`UPDATE, DELETE, INSERT, SELECT`) statement, an additional entry `(Select FK value)` is available in the popup if the cursor is located inside the value assignment or condition, e.g. in the following example:

```
update film_category
    set category_id = |
  where film_id = 42;
```

(the | denoting the location of the cursor).

When that menu item is selected, the statement is analyzed and if the column of the current expression is a foreign key to a different table, the lookup dialog will appear and will let you select the appropriate PK value from the referenced table.

Foreign key lookup for DML statement is currently only supported for single column primary keys.

15.4. Deleting rows with foreign keys

To delete rows from the result set 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.

If you want to generate a SQL script to delete all dependent rows, you can also use the SQL Workbench/J command WbGenerateDelete.

16. DBMS specific features

16.1. PostgreSQL specific features

16.1.1. Checking for un-committed changes

Before a SQL panel (or the application) is closed, SQL Workbench/J will check if the current connection has any un-committed changes (e.g. an `INSERT` without a `COMMIT`). This is done by checking the `pg_locks` system view. The information in this view might not always be 100% correct and can report open transactions even though there are none.

The checking for un-committed changes can be controlled through the connection profile.

16.1.2. Using the COPY API for client side files

WbImport can make use of PostgreSQL's `COPY` API to send client side files to the server. The SQL statement `COPY from stdin` does not work when executed using the JDBC driver. But `WbImport` can make use of the COPY API by using the parameter `-usePgCopy`

16.1.3. Using .pgpass

If username, password or both are empty in a connection profile, SQL Workbench/J will try to use the information stored in the password file file or the environment variables (`PGPASS`, `PGUSER`) the same way as libpq uses them.

16.1.4. Using savepoints for single statements

PostgreSQL marks a complete transaction as failed if a only single statement fails. In such a case the transaction cannot be committed, e.g. consider the following script:

```
INSERT INTO person (id, firstname, lastname) VALUES (1, 'Arthur', 'Dent');
INSERT INTO person (id, firstname, lastname) VALUES (2, 'Zaphod', 'Beeblebrox');
INSERT INTO person (id, firstname, lastname) VALUES (2, 'Ford', 'Prefect');
COMMIT;
```

As the ID column is the primary key, the third insert will fail with a unique key violation. In PostgreSQL you cannot commit anyway and thus persist the first two `INSERT`s.

This problem can only be solved by using a SAVEPOINT before and after each statement. In case that statement fails, the transaction can be rolled back to the state before the statement and the reminder of the script can execute.

Doing this manually is quite tedious, so you can tell SQL Workbench/J to do this automatically for you by setting the properties:

```
workbench.db.postgresql.ddl.usesavepoint=true
workbench.db.postgresql.sql.usesavepoint=true
```

in the file workbench.settings. If this is enabled, SQL Workbench/J will issue a `SET SAVEPOINT` before running each statement and will release the savepoint after the statement. If the statement failed, a rollback to the savepoint will be issued that will mark the transaction as "clean" again. So in the above example (with `sql.usesavepoint` set to `true`), the last statement would be rolled back automatically but the first two `INSERT`s can be committed (this will also required to turn on the "Ignore errors" option is enabled).

If you want to use the modes `update/insert` or `insert/update` for WbImport, you should also add the property:

```
workbench.db.postgresql.import.usesavepoint=true
```

to enable the usage of savepoints during imports. This setting also affects the `WbCopy` command.

This is not necessary when the using the mode `upsert` or `insertIgnore` with Postgres 9.5

You can also use the parameter `-useSavepoint` for the `WbImport` and `WbCopy` commands to control the use of savepoints for each import.



Using savepoints can slow down the import substantially.

16.1.5. Preventing connections with "idle in transaction" state

Postgres has a very strict transaction concept which means that even a simple `SELECT` statement starts a transaction. This has some implications on concurrency, the most obvious one is that tables that are "used" in a transaction (because a query has retrieved some values) cannot be modified using DDL statements (`ALTER TABLE`). Connections to the server that do this have the status `idle in transaction` as opposed to just "idle".

There are two ways to prevent this:

- Working with auto-commit enabled, so that each query is automatically committed. This is how pgAdmin and psql work by default
- If you don't want to work with auto-commit enabled, these transactions need to be ended properly by running a `rollback` or `commit` when the query is finished

SQL Workbench/J can be configured to do the second approach automatically, by setting the configuration property `workbench.db.postgresql.transaction.readonly.end` to one of the following values:

- never
- rollback
- commit

The feature is disabled if the value `never` is configured. The other two values control how the transaction is ended: either by running a `rollback` or a `commit`

The statement to end the transaction will **only** be sent to the server, if the current transaction has not modified anything in the database. Once a real change has been done by running an DML or DDL statement, nothing will be sent automatically.

16.2. Oracle specific features

16.2.1. Checking for un-committed changes

Before a SQL panel (or the application) is closed, SQL Workbench/J will check if the current connection has any un-committed changes (e.g. an `INSERT` without a `COMMIT`). This is done by checking the `V$TRANSACTION` system view.



By default a regular user does not have `SELECT` privilege on `V$TRANSACTION`, please grant the privilege before enabling this feature.

The checking for un-committed changes can be controlled through the connection profile.

16.2.2. SQL*Plus autotrace mode

SQL Workbench/J supports the a mode similar to "autotrace" mode in SQL*Plus. The command to turn on autotrace is the same as in SQL*Plus and supports the same options. For details see the description of the SET command.

The current user needs to have the PLUSTRACE role in order to be able to see statement statistics (which is required by SQL*Plus as well). The PLUSTRACE role grants the SELECT privilege on the system views: V$SESSTAT, V$STATNAME and V$MYSTAT. The role is not required for the `traceonly explain` option.

As an extension to the Oracle syntax, SQL Workbench/J supports the keyword `realplan` as a substitute for `explain`. In that case the execution plan is also displayed but not by using EXPLAIN PLAN but by retrieving the actual execution plan that is available via `dbms_xplan.display_cursor()`. In order to use that package, the execute SQL will be changed by SQL Workbench/J. It will prepend it with a unique identifier so that the SQL can be found again in Oracle's system views and it will add the `gather_plan_statistics` hint to the statement in order to get more detailed statistics in the execution plan.

In order to see the "real" execution plan, use `set autotrace traceonly realplan` instead of `set autotrace traceonly explain`.

When using `statistics` together with `explain` or `realplan`, SQL Workbench/J will have to retrieve the generated SQL_ID in order to get the execution plan using `dbms_xplan.display_cursor()`. To use that function the SQL_ID is required which is retrieved from V$SQL using a unique comment that is added to the SQL statement before it is sent to the database. Querying V$SQL based on the column SQL_TEXT is quite an expensive operation and might create unwanted latch contention on the server. If you want to avoid that overhead do not use the `statistics` option when also displaying the execution plan.

Examples

Show statistics without retrieving the actual data:

```
set autotrace traceonly statistics
```

Retrieve the data and show statistics

```
set autotrace on statistics
```

Display the statistics and the execution plan but do not retrieve the data

```
set autotrace traceonly explain statistics
```

Display the statistics and the actual execution plan but do not retrieve the data

```
set autotrace traceonly realplan statistics
```

16.2.3. Using SQL*Plus' SHOW command

SQL Workbench/J supports most of the parameters and options the SHOW from SQL*Plus does.

SHOW option	Description
ERRORS	Displays errors from the last PL/SQL compilation.
PARAMETERS	Displays configuration parameters. Unlike SQL*Plus you can supply multiple parameters separated with a comma: `show parameter memory,sga` will show the values for all parameters that contain the string `memory` or `sga`.

SHOW option	Description
	As with SQL*Plus, you need the `SELECT` privilege on `V_$PARAMETER` to use the `PARAMETERS` option.
SGA	Displays memory information. As with SQL*Plus, you need `SELECT` privilege on `V_$SGA` to use the `sga` (or `sgainfo` option.
SGAINFO	Displays extended memory information not available in SQL*Plus.
RECYCLEBIN	Shows the content of the recyclebin.
USER	Shows the current user.
AUTOCOMMIT	Shows the state of the autocommit property.
LOGSOURCE	Displays the location of the archive logs.
EDITION	Shows the edition of the current database.
CON_ID	Displays the id of the current container database (only for 12c)
PDBS	Displays the list of pluggable databases (only for 12c)

16.2.4. Using Oracle's DBMS_OUTPUT package

To turn on support for Oracle's `DBMS_OUTPUT` package you have to use the (SQL Workbench/J specific) command `ENABLEOUT`. As an alternative you can also use the SQL*Plus command `set serveroutput on`.

After running `ENABLEOUT` the `DBMS_OUTPUT` package is enabled, and any message written with `dbms_output.put_line()` is displayed in the message panel 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.

To disable the `DBMS_OUTPUT` package again, use the (SQL Workbench/J specific) command `DISABLEOUT`. This is equivalent to calling `dbms_output.disable()` procedure or using `set serveroutput off`

`ENABLEOUT` and `DISABLEOUT` support an additional parameter `quiet` to suppress the feedback message that the support for `DBMS_OUTPUT` has been enabled or disabled. `set serveroutput off` never gives a feedback message.

16.2.5. Using DBMS_METADATA for source retrieval

SQL Workbench/J uses the information returned by the JDBC driver to re-create the source of database objects (tables, views, ...). The source generated this way will not always match the source generated by Oracle itself.

The use of `DBMS_METADATA` for object source retrieval is controlled by configuration properties.

The property `workbench.db.oracle.use.dbmsmeta` can be used to controll the use for all object types. When set to `true` the source for all objects will be retrieved using `DBMS_METADATA`.

The use of `DBMS_METADATA` can also be controlled for a specific object type by appending the type name to the property name `workbench.db.oracle.use.dbmsmeta`. The following types can be configured:

- `workbench.db.oracle.use.dbmsmeta.table`
- `workbench.db.oracle.use.dbmsmeta.mview` (for MATERIALIZED VIEWs)
- `workbench.db.oracle.use.dbmsmeta.index`
- `workbench.db.oracle.use.dbmsmeta.view`
- `workbench.db.oracle.use.dbmsmeta.sequence`
- `workbench.db.oracle.use.dbmsmeta.synonynm`
- `workbench.db.oracle.use.dbmsmeta.procedure` (includes packages)
- `workbench.db.oracle.use.dbmsmeta.trigger`
- `workbench.db.oracle.use.dbmsmeta.constraint` (for FK and PK constraints)

The value of a specific object type overrides the global setting.

17. Variable substitution in SQL statements

17.1. 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`.

`WbVarDef myvar=42` 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.

Variable substitution is also done within Macros. If your macro definition contains a reference to a SQL Workbench/J variable, this will be treated the same way as in regular statements.

The definition of variables can also be read from a properties file. This can be done 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;
```

17.2. Populate a variable from a SELECT statement

A variable can also be defined as the result of a `SELECT` statement. The query can be defined through the parameter `-query`. Alternatively this can be done by by using `@` as the first character after the equal sign. The query needs to be enclosed in double quotes.

```
WbVarDef -variable=myvar -query="SELECT id FROM person WHERE name='Dent'"
```

or the old short syntax using `@`

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

If the `SELECT` returns more than one column, multiple variables can be defined by specifying a comma separated list of variable names. The following statement will define the variables `id` and `name` based on the values returned from the `SELECT` statement:

```
WbVarDef -variable=id,name -query="SELECT id,firstname FROM person WHERE lastname='Dent'"
```

When executing the statement, SQL Workbench/J only retrieves the first row of the result set. Subsequent rows are ignored. If the select returns more columns than variable names, the additional values are ignored. If more variables are listed than columns are present in the result set, the additional variables will be undefined.

The parameter `-nullHandling` can be used to control the result if the value retrieved through the query is `NULL`.

`-nullHandling=empty` - if the value from the query is NULL, the variable be set to an empty string.
`-nullHandling=ignore` - the value is ignored. An existing variable will not be changed. If the variable does not exist, it will not be created.
`-nullHandling=remove` - if the value from the query is NULL, the variable will be removed.

17.3. Populate a variable from a file

A variable can also be defined by reading the content of a file (this is different from reading the variable definition from a file).

```
WbVarDef -variable=somevar -contentFile=/temp/mydata.txt
```

When executing the statement, SQL Workbench/J will read the content of the file `mydata.txt` and use that as the value for the variable `somevar`.

If the file contents contains references to variables, these are replaced after the content has been loaded. To disable replacement, use the parameter `-replaceVars=false`.

Consider the following sequence of statements, where the file `select.txt` contains the statement `SELECT * FROM person WHERE id = $[person_id]`

```
WbVarDef person_id=42;
WbVarDef -variable=my_select -contentFile=select.txt;
$[my_select];
```

After running the above script, the variable `my_select`, will have the value of `SELECT * FROM person WHERE id = 42`. When "running" `$[my_select]`, the row with id=42 will be retrieved.

17.4. 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.

17.5. 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 `]`, 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]';`



Variables will be replaced inside string literals (e.g. `'$[foo]'`) and comments (e.g. `-- $[foo]` or `/* $[foo] */`)

If you are using values in your regular statements that actually need the prefix (`$[` or suffix `]`) characters, please make sure that you have no variables defined. Otherwise you will have 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 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.

17.6. 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, reference the value with a question 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 implicitly 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 question 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 command line) 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.

When defining a variable, you can specify a list of values that should be entered in the dialog.

```
WbVardef -variable=status -values='active,pending,closed';
```

17.7. Controlling the order of variables during prompting

By default the variables shown in the prompt dialog are sorted alphabetically. This behavior can be changed by setting the configuration property `workbench.sql.parameter.prompt.sort` to true, e.g. using WbSetConfig

```
WbSetConfig workbench.sql.parameter.prompt.sort=false
```

If the property is set to `false`, the variables are shown in the order they were declared:

```
WbVarDef zzz='';
WbVarDef vvv='';
WbVarDef aaa='';

select *
from foobar
where col1 = $[?aaa]
  and col2 = $[?vvv]
  and col3 > $[?zzz]
```

The dialog to enter the variables will show them in the order `zzz`, `vvv`, `aaa`

17.8. Scope of variables

By default all windows opened through File » New Window will share the same set of variables. If you change a variable in the first window, the new value is also immediately visible in all other windows.

The default behaviour can be changed in the "General" section of the options dialog. If enabled, each window will have its own set of variables. Setting or changing a variable in one window will not affect variables in the other windows.

18. 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.

To start SQL Workbench/J in batch mode, either the -script or -command must be passed as an argument on the command line.

If neither of these parameters is present, SQL Workbench/J will run in GUI mode.



When running SQL Workbench/J on Windows®, you either need to use `sqlwbconsole` or start SQL Workbench/J using the `java` command. When using the Windows® launcher `SQLWorkbench64.exe` no output (or console window) will be shown when the SQL scripts are processed.

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 by 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 specify the connection and the SQL script to be executed have to be passed on the command line.

18.1. 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 .

18.2. Specifying the script file(s)

The script that should be run is specified with the parameter `-script=<filename>` Multiple scripts can be specified by separating them with a comma. The scripts will then 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.

18.3. Specifying a SQL command directly

If you do not want to create an extra SQL script just to run one or more short SQL commands, you can specify the commands to be executed directly with the `-command` parameter. To specifiy more than on SQL statement use the standard delimiter to delimit them, e.g. `-command='delete from person; commit;'`

If a script has been specified using the `-script` parameter, the `-command` parameter is ignored.

When using Linux (or Unix-Based operating systems) the command can also be passed using a "Here Document". In this case the `-command` parameter has to be used without a value:

```
$ java -jar sqlworkbench.jar -profile=PostgresProduction -command <<SQLCMD
insert into some_table values (42);
delete from other_table where id = 42;
commit;
SQLCMD
```

The position of the `-command` parameter does not matter. The following will also work:

```
$ java -jar sqlworkbench.jar \
     -profile=PostgresProduction \
     -command \
     -displayResult=true \
     -showTiming=true <<SQLCMD
select *
from person;
SQLCMD
```

18.4. 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.

18.5. 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 command line. If you need to run several script files with different encodings, you have to create one "master" file, which calls the individual files using the WbInclude command together with its `-encoding` parameter.

18.6. 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`

18.7. 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.

If a script resulted in an error from the database, the last error message, error code and "SQL state" will be available in the special variables:

- `wb_last_error_code` - the vendor specific error number
- `wb_last_error_state` - the "SQL state" code
- `wb_last_error_msg` - the error message

Those variables can also be used for conditional aggregation in subsequent commands.

You can also specify whether errors from `DROP` commands should be ignored. To enable this, pass the parameter `-ignoreDropErrors=true` on the command line. 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.

18.8. 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.

If the filename is specified as a relative file, it is assumed to be in the current working directory.

18.9. 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.

If the filename is specified as a relative file, it is assumed to be in the current working directory.

18.10. 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.

18.11. Changing the connection

You can change the current connection inside a script using the command `WbConnect`.

18.12. 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.

18.12.1. 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
```

18.12.2. 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 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).

When using `-feedback=false`, informational messages like the total number of statements executed, or a successful connection are not logged either.

18.12.3. Show timing information

The parameter `-showTiming=true` can be used to display the execution time of each query. This is applicable for batch and console mode.

18.12.4. 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 specify `-showProgress=true` these messages will also be displayed on the console.

18.13. Running batch scripts interactively

By default neither parameter prompts nor execution confirmations ("Confirm Updates") are processed when running in batch mode. If you have batch scripts that contain parameter prompts and you want to enter values for the parameters while running the batch file, you have to start SQL Workbench/J using the parameter `-interactive=true`.

18.14. Defining variables

The definition of variables can be read from a properties file, either by specifying `-file=filename` for the `WbVarDef` command, or by passing the `-varFile` or `-variable` parameter when starting SQL Workbench/J. Please see the description for the command line parameters for details.

18.15. 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
```

18.16. Examples



For readability the examples in this section are displayed on several lines. If you enter them manually on the command line 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=PostgreSQL
     -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

Run a SQL command in batch mode without using a script file

The following example exports the table "person" without using the -script parameter:

```
java -jar sqlworkbench.jar
     -profile='TestData'
     -command='WbExport -file=person.txt -type=text -sourceTable=person'
```

The following example shows how to run two different SQL statements without using the -script parameter:

```
java -jar sqlworkbench.jar
     -profile='TestData'
     -command='delete from person; commit;'
```

19. Using SQL Workbench/J in console mode

SQL Workbench/J can also be used from the command line without starting the GUI, e.g. when you only have a console window (Putty, SSH) to access the database. In that case you can either run scripts using the batch mode, or start SQL Workbench/J in console mode, where you can run statements interactively, similar to the GUI mode (but of course with less comfortable editing possibilities).

When using SQL Workbench/J in console mode, you **cannot** use the Windows® launcher. Please use the supplied scripts `sqlwbconsole.cmd` (Windows® batch file) or `sqlwbconsole.sh` (Unix shell script) to start the console mode.

When starting SQL Workbench/J in console mode, you can define the connection using a profile name or specifying the connection properties directly . Additionally you can specify all parameters that can be used in batch mode.

The following batch mode parameters will be ignored in console mode:

`script` - you cannot specify a script to be run during startup. If you want to run a script in console mode, use the command WbInclude.
`encoding` - as you cannot specify a script, the encoding parameter is ignored as well
`displayResult` - always true in console mode
`cleanupSuccess` and `cleanupError`- as no script is run, there is no "end of script" after which a "cleanup" is necessary
`abortOnError`

19.1. Entering statements

After starting the console mode, SQL Workbench/J displays the prompt `SQL>` where you can enter SQL statements. The statement will not be sent to the database until it is either terminated with the standard semicolon, or with the alternate delimiter (that can be specified either in the used connection profile or on the commandline when starting the console mode).

As long as a statement is not complete, the prompt will change to `..>`. Once a delimiter is identified the statement(s) are sent to the database.

```
SQL> SELECT * [enter]
..>FROM person;
```

A delimiter is only recognized at the end of the input line, thus you can enter more than one statement on a line (or multiple lines) if the intermediate delimiter is not at the end of one of the input lines:

```
SQL> DELETE FROM person; rollback;
DELETE executed successfully
4 row(s) affected.

ROLLBACK executed successfully
SQL>
```

19.2. Exiting console mode

To exit the application in console mode, enter `exit` when the default prompt is displayed. If the "continuation prompt" (`..>`) is displayed, this will not terminate the application. The keyword `exit` must not be terminated with a semicolon.

19.3. Setting or changing the connection

If you did not specify a connection on the command line when starting the console, you can set or change the current connection in console mode using the `WbConnect` command. Using `WbConnect` in console mode will automatically close the current connection, before establishing the new connection.

To disconnect the current connection in console mode, run the statement `WbDisconnect`. Note that this statement is only available in console mode.

19.4. Displaying result sets

If you are running `SELECT` statements in console mode, the result is displayed on the screen in "tabular" format. Note that SQL Workbench/J reads the whole result into memory in order to be to adjust the column widths to the displayed data.

You can disable the buffering of the results using the command line parameter `bufferResults=false`. In that case, the width of the displayed columns will not be adjusted properly. The column widths are taken from the information returned by the driver which typically results is a much larger display than needed.

The output in tabular format (if results are buffered) looks like this:

```
SQL> select id, firstname, lastname, comment from person;
id | firstname | lastname   | comment
---+-----------+------------+--------------------
1  | Arthur    | Dent       | this is a comment
2  | Zaphod    | Beeblebrox |
4  | Mary      | Moviestar  | comment
3  | Tricia    | McMillian  | test1

(4 Rows)
SQL>
```

If the size of the column values exceed the console's width the display will be wrapped, which makes it hard to read. In that case, you can switch the output so that each column is printed on a single line.

This is done by running the statement: `WbDisplay record`

```
SQL> WbDisplay record;
Display changed to single record format
Execution time: 0.0s
SQL> select id, firstname, lastname, comment from person;
---- [Row 1] -------------------------------
id        : 1
firstname : Arthur
lastname  : Dent
comment   : this is a very long comment that would not fit onto the screen when printed as
---- [Row 2] -------------------------------
id        : 2
firstname : Zaphod
lastname  : Beeblebrox
comment   :
---- [Row 3] -------------------------------
id        : 4
firstname : Mary
lastname  : Moviestar
```

```
comment   :
---- [Row 4] ------------------------------
id        : 3
firstname : Tricia
lastname  : McMillian
comment   :

(4 Rows)
SQL>
```

To switch back to the "tabular" display, use: `WbDisplay tab`.

19.5. Running SQL scripts that produce a result

Normally when executing a SQL script using WbInclude, the result of such a script (e.g. when it contains a SELECT statement) is not displayed on the console.

To run such a script, use the command `WbRun` instead of `WbInclude`. If you have the following SQL script (named select_person.sql):

```
SELECT *
FROM person;
```

and execute that using the WbInclude command:

```
SQL> WbInclude -file=select_person.sql;
SQL> Execution time: 0.063s
```

If you execute this script using `WbRun` the result of the script is displayed:

```
SQL> WbRun select_people.sql;
select *
from person;

id | firstname | lastname
---+-----------+------------
1  | Arthur    | Dent
4  | Mary      | Moviestar
2  | Zaphod    | Beeblebrox
3  | Tricia    | McMillian

(4 Rows)
Execution time: 0.078s
SQL>
```

19.6. Controlling the number of rows displayed

In the SQL Workbench/J GUI window, you can limit the reusult of a query by entering a value in the "Max. Rows" field. If you want to limit the number of rows in console mode you can do this by running the statement

```
SQL> set maxrows 42;
MAXROWS set to 42
Execution time: 0.0s
SQL>
```

This will limit the number of rows retrieved to 42.

SET MAXROWS has no effect when run as a post-connect script.

19.7. Controlling the query timeout

To set the query timeout in console mode, you can run the following statement

```
SQL> set timeout 42;
TIMEOUT set to 42
Execution time: 0.0s
SQL>
```

This will set a query timeout of 42 seconds. Note that not all JDBC drivers support a query timout.

SET TIMEOUT has no effect when run as a post-connect script.

19.8. Managing connection profiles

Connection profiles can be managed through several SQL Workbench/J specific commands. They are primarily intended to be used in console mode, but can also be used when running in GUI mode.

19.8.1. List available profiles - WbListProfiles

The command `WbListProfiles` will display a list of all displayed profiles

19.8.2. Delete a profile - WbDeleteProfile

You can delete an existing profile using the command `WbDeleteProfile`. The command takes one argument, which is the name of the profile. If the name is unique across all profile groups you don't have to specify a group name. If the name is not unique, you need to include the group name, e.g.

```
SQL> WbDeleteProfile {MyGroup}/SQL Server
Do you really want to delete the profile '{MyGroup}/SQL Server'? (Yes/No) yes
Profile '{MyGroup}/SQL Server' deleted
SQL>
```

As the profile name is the only parameter to this command, no quoting is necessary. Everything after the keyword `WbDeleteProfile` will be assumed to be the profile's name

All profiles are automatically saved after executing `WbDeleteProfile`.

19.8.3. Save the current profile - WbStoreProfile

Saves the currently active connection as a new connection profile. This can be used when SQL Workbench/J if the connection information was passsed using individual parameters (`-url`, `-username` and so on) either on the commandline or through `WbConnect`.

```
SQL> WbStoreProfile {MyGroup}/PostgreSQL Production
Profile '{MyGroup}/PostgreSQL Production' added
SQL>
```

If no parameter switch is given, everything after the keyword `WbDeleteProfile` will be assumed to be the profile's name. By default the password is not saved.

Alternatively the command supports the parameters `name` and `savePassword`. If you want to store the password in the profile, the version using parameters must be used:

```
SQL> WbStoreProfile -name="{MyGroup}/DevelopmentServer" -savePassword=true
Profile '{MyGroup}/DevelopmentServer' added
SQL>
```

If the current connection references a JDBC driver that is not already defined, a new entry for the driver defintions is created referencing the library that was passed on the commandline.

All profiles are automatically saved after executing `WbStoreProfile`.

19.8.4. Create a new connection profile - WbCreateProfile

`WbCreateProfile` can be used to create a new profile without an existing connection. It accepts the same parameters as WbConnect plus an additional parameter to define the name of the new profile.

```
SQL> WbCreateProfile -name="Postgres" -profileGroup=DBA -savePassword=true -username=postg
..> -url=jdbc:postgresql://localhost/postgres
..> -driver=org.postgresql.Driver
..> -driverJar=c:/etc/libs/postgres/postgresql-9.4-1206-jdbc42.jar;
Profile '{DBA}/Postgres' added
SQL>
```

19.9. Using an external pager

By default all rows of the result of a query will be printed to the console without pausing. To scroll through the result, configure the console to enable scrolling back to previous rows.

Alternatively, it's possible to enable the use of an external pager tool, that controls the output of the result. Popular tools for this are the command line tools `more`, or `less`

To enable the use of an external pager, set the property `workbench.console.pager` to the full path of the pager's executable. This can also be done through `WbSetConfig`. `pager` is recognized as an abbreviation of `workbench.console.pager`.

```
SQL> WbSetConfig pager=/usr/bin/more;
```

If the specified executable does not exist, the definition of the pager will be ignored.

19.10. PostgreSQL psql commands

Some of the SQL Workbench/J specific commands can be abbreviated using the command syntax from PostgreSQL's command line client `psql`. This is only implemented for very few commands and most of them don't work exactly the same way as the PostgreSQL command.

The following commands are available:

Command	Description / SQL Workbench/J command
`\q`	Quit console mode (equivalent to `exit`

Command	Description / SQL Workbench/J command
`\s`	`WbHistory` - display the statement history
`\i`	`WbRun` - Run a SQL script
`\d`	`WbList` - Show the list of available tables
`\l`	`WbListCat` - Show the list of databases
`\dn`	`WbListSchemas` - Show the list of schemas
`\dt`	`DESCRIBE` - Show the structure of a table
`\df`	`WbListProcs` - Show the list of stored procedures
`\sf`	`WbProcSource` - Show the source code of a stored procedure or function
`\g`	Run the last entered statement again
`\!`	`WbSysExec` - Run a commandline program

Even though those commands look like the `psql` commands, they don't work exactly like them. Most importantly they don't accept the parameters that `psql` supports. Parameters need to be passed as if the regular SQL Workbench/J command had been used.

20. Export data using WbExport

The `WbExport` exports contents of the database into external files, e.g. plain text ("CSV") or XML.

The `WbExport` command can be used like any other SQL command (such as `UPDATE` or `INSERT`). This includes the usage in scripts that are 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 statement 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 `-formatFile=oracle` a control file will be created that contains the appropriate definitions to read the BLOB data from the external file.



Oracles's `BFILE`, PostgreSQL's `large object` and SQL Server's `filestream` types are not real `BLOB` datatypes (from a JDBC point of view) and are currently not exported by `WbExport`. Only columns that are reported as `BLOB`, `BINARY`, `VARBINARY` or `LONGVARBINARY` in the column "JDBC type" in the DbExplorer will be exported correctly into a separate file.

20.1. Memory usage and WbExport

WbExport is designed to directly write the rows that are retrieved from the database to the export file without buffering them in memory (except for the XLS and XLSX formats)

Some JDBC drivers (e.g. PostgreSQL, jTDS and the Microsoft driver) read the full result obtained from the database into memory. In that case, exporting large results might still require a lot of memory. Please refer to the chapter Common problems for details on how to configure the individual drivers if this happens to you.

20.2. Exporting Excel files

If you need to export data for Microsoft Excel, additional libraries are required to write the native Excel formats (`xls` and the new `xlsx` introduced with Office 2007). Exporting the "SpreadsheetML" format introduced with Office 2003 does not require additional libraries.

SQL Workbench/J supports three different Excel file formats:

Value for `-type` parameter	Description
xlsm	This is the plain XML ("SpreadsheetML") format introduced with Office 2003. This format is always available and does not need any additional libraries.

Value for `-type` parameter	Description
	Files with this format should be saved with the extension **xml** (otherwise Office is not able to open them properly)
xls	This is the old binary format using by Excel 97 up to 2003. To export this format, only `poi.jar` is needed. If the library is not available, this format will not be listed in the export dialog ("Save data as...") Files with this format should be saved with the extension **xls**
xlsx	This is the "new" XML format (OfficeOpen XML) introduced with Office 2007. To create this file format, additionaly libraries are required. If those libraries are not available, this format will not be listed in the export dialog ("Save data as...") Files with this format should be saved with the extension **xlsx**

For a comparison of the different Microsoft Office XML formats please refer to: https://en.wikipedia.org/wiki/Microsoft_Office_XML_formats

All needed Libraries are included in the "Generic package including all optional libraries".



To write the file formats XLS and XLSX the entire file needs to be built in memory. When exporting results with a large number of rows this will require a substantial amount of memory.

WbExport and the "Max. Rows" option

When you use the `WbExport` command together with a `SELECT` query, the "Max. Rows" setting will be **ignored** for the export.

20.3. General WbExport parameters

`WbExport` supports conditional execution

Parameter	Description
-type	Possible values: `text, sqlinsert, sqlupdate, sqldeleteinsert, sqlmerge, xml, ods, xlsm, xls, xlsx, html, json` 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. `sqlmerge` will generate statements that will result in `INSERT` or `UPDATE` type of statements. The exact syntax depends on the current database. To select a syntax for a different DBMS, use the parameter `-mergeType`. 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.

Parameter	Description
	`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 be opened e.g. with OpenOffice.org. `xlsm` will generate a spreadsheet file in the Microsoft Excel 2003 XML format ("XML Spreadsheet"). When using Microsoft Office 2010, this export format should should be saved with a file extension of `.xml` in order to be identified correctly. `xls` will generate a spreadsheet file in the proprietary (binary) format for Microsoft Excel (97-2003). The file `poi.jar` is required. `xlsx` will generate a spreadsheet file in the default format introduced with Microsoft Office 2007. Additional external libraries are required in order to be able to use this format. Please read the note at the beginning of this section. This parameter supports auto-completion.
-file	The output file to which the exported data is written. This parameter is ignored if `-outputDir` is also specified.
-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. This parameter supports auto-completion.
-schema	Define the schema in which the table(s) specified with `-sourceTable` are located. This parameter only accepts a single schema name. If you want to export tables from more than one schema, you need to fully qualify them as shown in the description of the `-sourceTable` parameter. This parameter supports auto-completion.
-types	Selects the object types to be exported. By default only TABLEs are exported. If you want to export the content of VIEWs or SYNONYMs as well, you have to specify all types with this parameter. `-sourceTable=* -types=VIEW,SYNONYM` or `-sourceTable=T% -types=TABLE,VIEW,SYNONYM`

Parameter	Description
	This parameter supports auto-completion.
-excludeTables	The tables listed in this parameter will not be exported. This can be used when all but a few tables should be exported from a schema. First all tables specified through `-sourceTable` will be evaluated. The tables specified by -excludeTables can include wildcards in the same way, `-sourceTable` allows wildcards. `-sourceTable=* -excludeTables=TEMP*` will export all tables, but not those starting with `TEMP`. This parameter supports auto-completion.
-sourceTablePrefix	Define a common prefix for all tables listed with `-sourceTable`. When this parameter is specified the existence of each table is not tested any longer (as it is normally done). When this parameter is specified the generated statement for exporting the table is changed to a `SELECT * FROM [prefix]tableName` instead of listing all columns individually. This can be used when exporting views on tables, when for each table e.g. a view with a certain prefix exists (e.g. table `PERSON` has the view `V_PERSON` and the view does some filtering of the data. This parameter can **not** be used to select tables from a specific schema. The prefix will be prepended to the table's **name**.
-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, execute `WbExport` with the parameter `-showEncodings`. This parameter is ignored for XLS, XLSX and ODS exports. This parameter supports auto-completion and if it is invoked for this parameter, it will show a list of encodings defined through the configuration property `workbench.export.defaultencodings` This is a comma-separated list that can be changed using `WbSetConfig`
-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 are: `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. This parameter supports auto-completion.
-header	Possible values: `true`, `false`

Parameter	Description
	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 output file 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.
-tableWhere	Defines an additional `WHERE` clause that is appended to **all** SELECT queries to retrieve the rows from the database. No validation check will be done for the syntax or the columns in the where clause. If the specified condition is not valid for all exported tables, the export will fail.
-clobAsFile	Possible values: `true`, `false` 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 exports (`-type=sqlinsert`) generated with `-clobAsFile=true` can only be used 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

Parameter	Description
-lobsPerDirectory	When exporting CLOB or BLOB columns as external files, the generated files can be distributed over several directories to avoid an excessive number of files in a single directory. The parameter `lobsPerDirectory` defines how many LOB files are written into a single directory. When the specified number of files have been written, a new directory is created. The directories are always created as a sub-directory of the target directory. The name for each directory is the base export filename plus "_lobs" plus a running number. So if you export the data into a file "the_big_table.txt", the LOB files will be stored in "the_big_table_lobs_1", "the_big_table_lobs_2", "the_big_table_lobs_3" and so on. The directories will be created if needed, but if the directories already exist (e.g. because of a previous export) **their contents will not be deleted**!
-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 name based on the row and column numbers). This parameter only makes sense if exactly one BLOB column of a table is exported.
-append	Possible values: `true`,`false` Controls whether results are appended to an existing file, or overwrite an existing file. This parameter is only supported for text, SQL, XLS and XLSX export types. When used with XLS oder XSLX exports, a new worksheet will be created.
-dateFormat	The date format to be used when writing date columns into the output file. This parameter is ignored for SQL exports.
-timestampFormat	The format to be used when writing datetime (or timestamp) columns into the output file. This parameter is ignored for SQL exports.
-timestampTZFormat	The format to be used when writing `TIMESTAMP WITH TIME ZONE` columns into the output file. This parameter is ignored for SQL exports. If this is not specified, the format for timestamp is used.
-locale	The locale (language) to be used when formatting date and timestamp values. The language will only be relevant if the date or timestamp format contains placeholders that are language dependent (e.g. the name of the month or weekday). This parameter supports code-completion.
-blobType	Possible values: `file`, `dbms`, `ansi`, `base64`, `pghex` 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. It is only valid for Text, SQL and XML exports, although not all parameter values make sense for all export types. The type `base64` is primarily intended for Text exports. The type `pghex` is intended to be used for export files that should be imported using PostgreSQL's `COPY` command. The types `dbms` and `ansi` are intended for SQL exports and generate a representation of the binary data as part of the SQL statement. `DBMS` will use a format that is understood by the DBMS you are exporting from, while `ansi` will generate a standard hex based representation of the binary data. The syntax generated by the `ansi` format is *not* understood by all DBMS!

Parameter	Description
	Two additional SQL literal formats are available that can be used together with PostgreSQL: `pgDecode` and `pgEscape`. `pgDecode` will generate a hex representation using PostgreSQL's decode() function. Using `decode` is a very compact format. `pgEscape` will use PostgreSQL's escaped octets, and generates much bigger statements (due to the increase escaping overhead). When using `file`, `base64` or `ansi` the file can be imported using `WbImport` 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.  Note that SQL scripts generated with `-blobType=file` can only be used with SQL Workbench/J 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. This parameter supports auto-completion.
-replaceExpression -replaceWith	Using these parameters, arbitrary text can be replaced during the export. `-replaceExpression` defines the regular expression that is to be replaced. `-replaceWith` defines the replacement value. `-replaceExpression='(\n|\r\n)' -replaceWith=' '` will replace all newline characters with a blank. The search and replace is done on the "raw" data retrieved from the database before the values are converted to the corresponding output format. In particular this means replacing is done before any character escaping takes place. Because the search and replace is done before the data is converted to the output format, it can be used for all export types (text, xml, Excel, ...). Only character columns (`CHAR`, `VARCHAR`, `CLOB`, `LONGVARCHAR`) are taken into account.
-trimCharData	Possible values: `true`, `false` If this parameter is set to true, values from `CHAR` columns will be trimmed from trailing whitespace. This is equivalent to the Trim CHAR data in the connection profile.
-literalFalse -literalTrue	These parameters define the values that should be exported for real `boolean` values. Both parameters must be specified. If only one of is provided, it is ignored.
-showProgress	Valid values: `true`, `false`, `<numeric value>` Control the update frequence in the status bar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specify a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).
-writeErrorToOutput	Possible values: `true`, `false`

Parameter	Description
	If this parameter is set to `true`, any error that occurs during the processing of the export will be written "as-is" into the export file. This will most likely cause a subsequent import of that file to fail, as the error message will mess up the formatting of the export file.
-openFile	Possible values: `true`, `false` If this parameter is provided, the generated output file will be opened if the export is successful. This parameter is ignored when writing multiple files. The same can be achieved by using WbSysOpen.

20.4. Parameters for text export

Parameter	Description
-delimiter	The given string sequence will be used to separate the values between two columns. The default is a tab character, equivalent to specifying: `-delimiter=\t` The character can also be specified as a unicode escape sequence, e.g. `-delimiter='\u0009'`
-rowNumberColumn	If this parameter is specified with a value, the value defines the name of an additional column that will contain the row number. The row number will always be exported as the first column. If the text file is not created with a header (`-header=false`) a value must still be provided to enable the creation of the additional column.
-quoteChar	The character (or sequence of characters) to be used to enclose column values that contain the delimiter. No quote character is used, unless one is specified. To set the double quote as the quote character you have to enclose it in single quotes: `-quotechar='"'` The character can also be specified as a unicode escape sequence, e.g. `-quoteChar='\u0022'`
-quoteCharEscaping	Possible values: `none`, `escape`, `duplicate` Defines how quote characters that appear in the actual data are written to the output file. If no quote character has been with the -quoteChar switch, this option is ignored. If `escape` is specified, a quote character that is embedded in the exported data is written as `here is a \" quote character`. If `duplicate` is specified, a quote character that is embedded in the exported data is written as two quotes e.g. `here is a "" quote character`. This parameter supports auto-completion.
-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.

Parameter	Description
	NULL values will not be quoted even if this parameter is set to true. This is useful to distinguish between NULL values and empty strings.
-quoteHeader	Possible values: `true`, `false` This parameter controls quoting of the column names in the header row. If `-quoteAlways=true` then this will cause all header columns to be quoted. If `-quoteAlways=false` then only column names that contain the delimiter will be quoted (this follows the same rules as the quoting for values). If this parameter is provided without a value (i.e. `-quoteAlways`) then this is the same as specifying the value `true`. The default value is: `false`
-decimal	The decimal symbol to be used for numbers. The default is a dot e.g. the number Pi would be written as `3.14152` When using `-decimal=','` the number Pi would be written as: `3,14152`
-decimalFormat	Specifies a format mask to format decimal numbers. If this is specified, then `-maxDigits` and `-fixedDigits` are ignored. The format string supports the same syntax as the decimal format for displaying values in the data formatting options.
-maxDigits	Defines a maximum number of decimal digits. If this parameter is not specified decimal values are exported according to the global formatting settings Specifying a value of 0 (zero) results in exporting as many digits as available.
-fixedDigits	Defines a fixed number of decimal digits. If this parameter is not specified decimal values are exported according to the `-maxDigits` parameter (or the global default). If this parameter is specified, all decimal values are exported with the defined number of digits. If `-fixedDigits=4` is used, the value `1.2` to be written as `1.2000`. This parameter is ignored if `-maxDigits` is also provided.
-escapeText	This parameter controls the escaping of non-printable or non-ASCII characters. Valid options are • `control` 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 • `extended` which will escape everything outside the range [32-126] and [161-255] • `pgcopy` which is the same as `control` but will write the characters as two byte hex values which are compatible with the input format for PostgreSQL's `COPY`. This will write a "short-hand" representation of control characters (e.g. `\n` for a newline) and a unicode representation for characters above ASCII 126 (e.g. `\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).

Parameter	Description
	To import a text file with escaped values using WbImport, the `-decode=true` must be used. Note that `WbImport` can not handle the `pgcopy` encoding. This parameter supports auto-completion.
-nullString	Defines the string value that should be written into the output file for a `NULL` value. This value will be enclosed with the specified quote character only if `-quoteAlways=true` is specified as well.
-formatFile	Possible values: `postgres`, `oracle`, `sqlserver`, `db2`, `mysql` This parameter controls the creation of a control file for the bulk load utilities of some DBMS. • `postgres` will create a SQL script with the necessary `COPY` syntax to import the generated text file • `oracle` will create a control file (.ctl) for Oracle's SQL*Loader utility • `sqlserver` will create a format file (.fmt) for Microsoft's `bcp` utility • `db2`will create a SQL script with a DB2 `IMPORT` command • `mysql`will create a SQL script with a MySQL `LOAD DATA INFILE` command You can specify more than one format (separated by a comma). In that case one control file for each format will be created.  The generated format file(s) are intended as a starting point for your own adjustments. Don't expect them to be complete. This parameter supports auto-completion.

20.5. Parameters for XML export

Parameter	Description
-table	The given tablename will be put into the `<table>` tag as an attribute.
-decimal	The decimal symbol to be used for numbers. The default is a dot (e.g. 3.14152)

-useCDATA	Possible values: `true`, `false`
	
	Normally all data written into the xml file will be written with escaped 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.
	
	With `-useCDATA=true` a HTML value would be written like this:
	
	`<![CDATA[<b>This is a title</b>]]>`
	
	With `-useCDATA=false` (the default) a HTML value would be written like this:
	
	`<b>This is a title</b>`

Parameter	Description
-xsltParameter	A list of parameters (key/value pairs) that should be passed to the XSLT processor. When using e.g. the `wbreport2liquibase.xslt` stylesheet, the value of the `author` attribute can be set using `-xsltParameter="authorName=42"`. This parameter can be provided multiple times for multiple parameters, e.g. when using `wbreport2pg.xslt`: `-xsltParameter="makeLowerCase=42" -xsltParameter="useJdbcTypes=true"`
-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	This parameter defines the output file for the XSLT transformation specified through the `-styleSheet` parameter
-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 harder to read for a human but is smaller in size which could be important for exports with large result sets.

20.6. 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.
-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.

Parameter	Description
	`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. PostgreSQL, 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. This parameter supports auto-completion.
-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. Note that this will only create the table including its primary key. This will **not** create other constraints (such as foreign key or unique constraints) nor will it create indexes on the target table.
-useSchema	Possible values: `true`, `false` If this parameter is set to `true`, all table names are prefixed with the appropriate schema. The default is taken from the global option Include owner in export
-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.
-includeAutoIncColumns	Possible values: `true`, `false` Default value: defined by global option

Parameter	Description
	With this parameter you can override the global option to include identity and auto-increment column for `INSERT` statements.
-includeReadOnlyColumns	Possible values: `true`, `false` Default value: `false` By default, columns that are marked as read-only by the JDBC driver or are defined as a computed column are not part of generated SQL statements. By setting this parameter to `true`, those columns will be included in `INSERT` statements.
-useMultiRowInserts	Possible values: `true`, `false` or a number Default value: `false` This option is only valid when using `-type=sqlinsert` When generating `INSERT` statements, one statement per row will be generated. To generate more efficient multi-row inserts, specify `true` for this parameter. If the parameter value is a number, it will define how many rows per generated INSERT statement are written. If the number is smaller than the the total number of rows, multiple INSERT statements will be generated.

20.7. Parameters for Spreadsheet types (ods, xslm, xls, xlsx)

Parameter	Description
-title	The name to be used for the worksheet
-infoSheet	Possible values: `true`, `false` Default value: `false` If set to true, a second worksheet will be created that contains the generating SQL of the export. For ods exports, additional export information is available in the document properties.
-fixedHeader	Possible values: `true`, `false` Default value: `true` unless a target sheet is specified If set to true, the header row will be "frozen" in the Worksheet so that it will not scroll out of view.
-autoFilter	Possible values: `true`, `false` Default value: `true` unless a target sheet is specified If set to true, the "auto-filter" feature for the column headers will be turned on.
-autoColWidth	Possible values: `true`, `false` Default value: `true` unless a target sheet is specified If set to true, the width of the columns is adjusted to the width of the content.
-targetSheet -targetSheetName	Possible values: any valid index or name for a worksheet in an existing Excel file This parameter is only available for XLS and XLSX exports

Parameter	Description
	When using this parameter, the data will be written into an existing file and worksheet without changing the formatting in the spreadsheet. No formatting is applied as it is assumed that the target worksheet is properly set up.  The parameters `-autoFilter`, `-fixedHeader` and `-autoColWidth` can still be used. If `-targetSheet` or `-targetSheetName` are specified they default to `false` unless they are explicitely passed as `true`. If the parameters `-dateFormat` or `-timestampFormat` are specified together with a target sheet, the format for date/timestamp columns in the Excel sheet will be overwritten. To overwrite the format in the Excel sheet, those parameters must be specified explicitely. If this parameter is used, the target file specified with the `-file` parameter **must** already exist If `-targetSheet` is supplied, the value for `-targetSheetName` is ignored These parameters support auto-completion if the `-file` parameter is already supplied.
-offset	Possible values: either a column/row combination or a cell reference in Excel format ("D3") This parameter is only available for XLS and XLSX exports When this parameter is specified the data is written starting at the specified location. No data will be written above or to the left of the specified cell. The values can be given as a numeric row column combination, e.g. `-offset=5,6`. Data will then be written starting with the fifth column in the sixth row. Alternatively the reference can be specified as an Excel reference: `-offset=C5`.
-formulaColumns	A comma separated list of column names (from the export result) that should be treated as formulas in the generated spreadsheet. The content of the value retrieved from the database is taken "as-is" into the spreadsheet cell which is then marked as containing a formula. Errors in the formula are not reported. The SQL on which the export is based, needs to generate the correct syntax for the formula. No formatting will be applied for those cells.
-maxDigits	This parameter is only valid for XLS or XLSX exports and defines the maximum number of digits to be displayed for decimal values. This is only applied to the display formatting, the actual values will contain the original value as retrieved from the database.
-fixedDigits	This parameter is only valid for XLS or XLSX exports and defines a fixed number of decimal digits to be used when applyin an Excel format for decimal columns. This is only applied to the display formatting, the actual values will contain the original value as retrieved from the database. This parameter is ignored if `-maxDigits` is also provided.
-fontName	This parameter is only valid for XLS or XLSX exports and defines the font name to be used for all cells. No validation of the font name will be done!
-fontSize	This parameter is only valid for XLS or XLSX exports and defines the font size (in "Points") to be used for all cells.

Parameter	Description
	This parameter is only used if `-fontName` is also provided.

20.8. Parameters for HTML export

Parameter	Description
-createFullHTML	Possible values: `true`, `false` Default value: `true` If this is set to true, a full HTML page (including <html>, <body> tags) will be created.
-escapeHTML	Possible values: `true`, `false` Default value: `true` If this is set to true, values inside the data will be escaped (e.g. the < sign will be written as <) so that they are rendered properly in an HTML page. If your data contains HTML tag that should be saved as HTML tags to the output, this parameter must be false.
-title	The title for the HTML page (put into the <title> tag of the generated output)
-preDataHtml	With this parameter you can specify a HTML chunk that will be added before the export data is written to the output file. This can be used to e.g. create a heading for the data: `-preDataHtml='<h1>List of products</h1>'`. The value will be written to the output file "as is". Any escaping of the HTML must be provided in the parameter value.
-postDataHtml	With this parameter you can specify a HTML chunk that will be added after the data has been written to the output file.

20.9. Parameters for JSON export

Parameter	Description
-nullString	Defines the string value that should be written into the output file for a `NULL` value.

20.10. Compressing export files

The `WbExport` command supports compressing of the generated output files. This includes the "main" export file as well as 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.

20.11. Examples

20.11.1. Simple plain text export

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

20.11.2. Exporting multiple tables

```
WbExport -type=text
         -outputDir='c:/data'
         -delimiter=';'
         -header=true
         -sourcetable=table_1, table_2, table_3, table_4;
```

This will export each specified table into a text file in the specified directory. The files are named "table_1.txt", "table_2.txt" and so on. To export all tables of a schema, the -sourceTable parameter supports wildcards:

```
WbExport -type=text
         -outputDir='c:/data'
         -delimiter=';'
         -header=true
         -sourcetable=my_schema.*;
```

Limiting the export data when using a table based export, can be done using the -tableWhere argument. This requires that the specified WHERE condition is valid for all tables, e.g. when every table has a column called MODIFIED_DATE

```
WbExport -type=text
         -outputDir='c:/data'
         -delimiter=';'
         -header=true
         -tableWhere="WHERE modified_date > DATE '2009-04-02'"
         -sourcetable=table_1, table_2, table_3, table_4;
```

This will add the specified where clause to **each** SELECT, so that only rows are exported that were changed after April 2nd, 2009

20.11.3. Export based on a SELECT statement

```
WbExport -type=text
         -file='c:/data/data.txt'
         -delimiter=','
```

```
         -decimal=','
         -dateFormat='yyyy-MM-dd';
SELECT * FROM data_table;
```

20.11.4. Export a complete schema

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
         -extension=zip
         -checkDependencies=true;
```

20.11.5. Export as SQL INSERT script

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 script which that contains statements like `INSERT INTO newtable (...) VALUES (...);` and the list of columns are all columns that are defined by the SELECT statement.

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).

20.11.6. Exporting LOB data



To extract the contents of CLOB columns you have to specify the parameter `-clobAsFile=true`, otherwise the contents of the CLOB columns will be written directly into the export file. BLOB columns will always be exported into separate tables.

When exporting tables that contain BLOB columns, one file for each blob column and row will be created. By default the generated filenames will contain the row and column number to make the names unique. You can however control the creation of filenames when exporting LOB columns using several different approaches. If a unique name is stored within the table you can use the `-filenameColumn` parameter to generate the filenames based on the contents of that column:

```
WbExport -file='c:/temp/blob_table.txt'
         -type=text
```

```
         -delimiter=','
         -filenameColumn=file_name;
```

Will create the file `blob_table.txt` and for each blob a file where the name is retrieved from the column `BLOB_TABLE.FILE_NAME`. Note that if the filename column is not unique, blob files will be overwritten without an error message.

You can also base the export on a SELECT statement and then generate the filename using several columns:

```
WbExport -file='c:/temp/blob_table.txt'
         -type=text
         -delimiter=','
         -filenameColumn=fname;
SELECT blob_column, 'data_'||id_column||'_'||some_name||'.'||type_column as fname
FROM blob_table;
```

This examples assumes that the following columns are part of the table `blob_table`: `id_column`, `some_name` and `type_column`. The filenames for the blob of each row will be taken from the computed column `fname`. To be able to reference the column in the `WbExport` you must give it an alias.

This approach assumes that only a single blob column is exported. When exporting multiple blob columns from a single table, it's only possible to create unique filenames using the row and column number (the default behaviour).

20.11.7. Replace data during export

When writing the export data, values in character columns can be replaced using regular expressions.

```
WbExport -file='/path/to/export.txt'
         -type=text
         -replaceExpression='(\n|\r\n)' -replaceWith='*'
         -sourceTable=export_table;
```

This will replace each newline (either Windows®' CR/LF or Unix LF) with the character *.

The value for `-replaceExpression` defines a regular expression. In the example above multiple new lines will be replace with multiple `*` characters. To replace consecutive new lines with a single `*` character, use the regular expression `-replaceExpression='(\n|\r\n)+'`. (Note the + sign after the brackets)

21. Import data using WbImport

The `WbImport` command can be used to import data from text, XML or Spreadsheet (ODS, XLS, XLSX) 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 be used like any other SQL command (such as `UPDATE` or `INSERT`), including scripts that are 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 appropriate 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 [57].

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 to create and import compressed exports, please refer to compressing export files



If you use `continueOnError=true` and expect a substantial number of rows to fail, it is highly recommended to also use a "bad file" to log all rejected records. Otherwise the rejected records are stored in memory (until the import finishes) which may lead to an out of memory error.

21.1. Importing spreadsheet files

In order to import Microsoft Excel (XSL, XSLT) or OpenOffice Calc (ODS) files, additional libraries are needed. These are included in the download "Generic package including all optional libraries".

You can tell if the needed libraries are installed if you invoke the code-completion after typing the `-type=` parameter. If the types XLS or ODS are presented in the drop down, the libraries installed.

The Excel import supports XLS and XLSX, it does **not** support the "SpreadsheetML" format.



To import XLS or XLSX files, the entire file needs to be read into memory. When importing large files this will require a substantial amount of memory.

21.2. General parameters

`WbImport` supports conditional execution

`WbImport` supports the following parameters

Parameter	Description
-type	Possible values: `xml`, `text`, `ods`, `xls`, `xlsx` Defines the type of the input file. This is only needed if the input file has a non-standard file extensions. If this parameter is not specified, the import type is derived from the input file's extension. This parameter supports code-completion. If invoked, it will only display available import types.

Parameter	Description
-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. For some DBMS, the additional modes: '`upsert`' and '`insertIgnore`' are supported. For details please refer to the native upsert and native insertIgnore explanation. This parameter supports code-completion.
-file	Defines the full name of the input file. Alternatively you can also specify a directory (using `-sourcedir`) from which all files are imported.
-table	Defines the table into which the data should be imported This parameter is ignored, if the files are imported using the `-sourcedir` parameter This parameter supports code-completion.
-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.
-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-sensitive (i.e. `TXT` is something different than `txt`
-ignoreOwner	If the file names imported with from the directory specified with -sourceDir contain the owner (schema) information, this owner (schema) information can be ignored using this parameter. Otherwise the files might be imported into a wrong schema, or the target tables will not be found.
-excludeFiles	Using -excludeFiles, files from the source directory (when using -sourceDir) can be excluded from the import. The value for this parameter is a comma separated list of partial names. Each file that contains at least one of the values supplied in this parameter is ignored. `-excludeFiles=back,data` will exclude any file that contains the value `back` or `data` in it, e.g.: `backup`, `to_back`, `log_data_store` etc.
-skipTargetCheck	Normally `WbImport` will check if the specified target table exists and will retrieve the column definition for the target table through the JDBC API. However, some JDBC drivers do not always return all table information e.g. for temporary tables. If you know that the target table exists, the parameter `-skipTargetCheck=true` can be used to tell `WbImport` to not verify the existence of the table through the JDBC API. However, `WbImport` needs to know the data types of the table columns. When `-skipTargetCheck` is enabled, the data types are detected by running a `SELECT *` statement to fetch the definition of the columns. For JDBC drivers supporting meta data for prepared statements, the query is only prepared, but not executed. For drivers not supporting that, the query is executed and the result is limited to a single row.

Parameter	Description
-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. Note that this will not check dependencies in the data. This means that e.g. the data for a self-referencing table (parent/child) will not be order so that it can be imported. To import self-referencing tables, the foreign key constraint should be set to "initially deferred" in order to postpone evaluation of the constraint until commit time.
-commitEvery	If your DBMS neeeds frequent commits to improve performance and reduce locking on the import table you can control the number of rows after which a COMMIT is sent to the server. `-commitEvery` is 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 the import is run as a single transaction that is committed at the end. When using batch import and your DBMS requires frequent commits to improve import performance, the `-commitBatch` option should be used instead. You can turn off the use of a commit or rollback during import completely by using the option `-transactionControl=false`. Using `-commitEvery` means, that in case of an error the already imported rows cannot be rolled back, leaving the data in a potential invalid state.
-transactionControl	Possible values: `true`, `false` Controls if 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, 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.
-continueOnError	Possible values: `true`, `false` This parameter controls the behavior 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. 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 using `-useSavepoint=true`.
-emptyFile	Possible values: `ignore`, `warning`, `fail` This parameter controls the behavior when an empty file (i.e. with a length of zero bytes) is used for the input file. `ignore` means the file is ignored, no warning will be shown or written to the log file. `warning` means the file is ignored, but a warning will be shown and logged. With `fail` an empty file will be treated as an error unless `-continueOnError=true` is specified. The default value is `fail`
-useSavepoint	Possible values: `true`, `false`

Parameter	Description
	Controls if SQL Workbench/J guards every insert or update statement with a savepoint to recover from individual error during import, when `continueOnError` is set to true. Using a savepoint for each DML statement can drastically reduce the performance of the import.
-keyColumns	Defines the key columns for the target table. This parameter is only necessary if import is running in `UPDATE` mode. It is assumed that the values for the key columns will never be `NULL`. This parameter is ignored if files are imported using the `-sourcedir` parameter.
-ignoreIdentityColumns	Possible values: `true`, `false` Controls if identity or auto-increment columns will be included in the import. If this is used, the JDBC driver must correctly report the column to be excluded as an *AUTOINCREMENT* column. This can be verified in the table definition display of the DbExplorer. If the column is reported with `YES` for the *AUTOINCREMENT* property, then this column will be excluded during the import.
-overrideIdentity	Possible values: `system`, `user` Controls the option to override identity columns, for DBMS that support the standard SQL's `OVERRIDE` option. If the option is specified, values from the import file are sent to the database, effectively disabling the automatic generation of identity columns. The option `system` will use `OVERRIDING SYSTEM VALUE` in the generated `INSERT` statement. The option `user` will use `OVERRIDING USER VALUE`. No check is done, if the target DBMS supports the option. The default is, to not use any overriding option.
-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) If code-completion is invoked for this parameter, it will show a list of encodings defined through the configuration property `workbench.export.defaultencodings` This is a comma-separated list that can be changed using `WbSetConfig`. If this property is not defined, all available encodings will be shown.
-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. This parameter is ignored for spreadsheet imports.
-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.

Parameter	Description
-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 (using `-batchSize`, then `-commitBatch` will also be ignored.
-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 physical line 2 is row 1 (one) for this parameter. When importing text files, empty lines in the input file are silently ignored and do not add to the count of rows for this parameter. So if your input file has two lines to be ignored, then one empty line and then another line to be ignored, `startRow` must be set to 4.
-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.
-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.
-badFile	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 file 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).

Parameter	Description
-maxLength	With the parameter `-maxLength` you can truncate data for character columns (`VARCHAR`, `CHAR`) 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 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` In case you are importing a boolean column (containing "true", "false") 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. To store different values than 0/1 in the target column, use the parameters `-numericFalse` and `-numericTrue`
-numericFalse -numericTrue	These parameters control the conversion of boolean literals into numbers. If these parameters are used, any text input that is identified as a "false" literal, will be stored with the number specified with `-numericFalse`. Any text input that is identified as "true" will be stored as the number specified with `-numericFalse`. To use -1 for false and 1 for true, use the following parameters: `-numericFalse='-1' -numericTrue='1'`. Note that '-1' must be quoted due to the dash. If these parameters are used, `-booleanToNumber` will be assumed true implicitely. These parameters can be combined with `-literalsFalse` and `-listeralsTrue`. Please note: • This conversion is only applied for "text" input values. Valid numbers in the input file will **not** be converted to the values specified with `-numericFalse` or `-numericTrue`. This means that you cannot change a `0` (zero) in the input file into a `-1` in the target column.
-literalsFalse -literalsTrue	These parameters control the conversion of boolean literals into boolean values. These two switches define the text values that represent the (boolean) values `false` and `true` in the input file. This conversion is applied when storing the data in a column that is of type `boolean` in the database. 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!

Parameter	Description
	• You always have to specify both switches, otherwise the definition will be ignored
-dateFormat	The format for date columns. This is only used for text imports and when using -stringDates=true with spreadsheet imports
-timestampFormat	The format for datetime (or timestamp) columns in the input file. This is only used for text imports and when using -stringDates=true with spreadsheet imports.
-timestampTZFormat	The format for `TIMESTAMP WITH TIME ZONE` columns in the input file. If this is not specified, the format for timestamps is used. This is only used for text imports and when using -stringDates=true with spreadsheet imports.
-defaultTimeZone	Defines a default time zone for values of `TIMESTAMP WITH TIME ZONE` columns in case the input values don't contain a time zone. Only used, if `-timestampTZFormat` is also specified.
-locale	The locale (language) to be used for parsing date and timestamp values. This is only used for text imports and spreadsheet imports using -stringDates=true This parameter supports code-completion.
-illegalDateIsNull	If this is set to `true`, illegal dates (such as February, 31st) or malformed dates inside the input file will be treated as a null value. This is only used for text and spreadsheet imports
-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. This is only used for text and spreadsheet imports
-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`. This is only used for text and spreadsheet imports

Parameter	Description
-columnExpression	This parameter can be used to define SQL expressions to be used instead of a plain column reference for the INSERT statement. This is useful to apply SQL functions directly on the server to the value retrieved from the input file. The format is `-columnExpression=column_name:expression`. The parameter can be repeated multiple times for multiple columns. The position of the input value is denoted by a question mark. The question mark must **not** be quoted, even if the input is a string/character value. As an example, this can be used to convert a comma separated string into a Postgres array: `-columnExpression=tags:string_to_array(?, ',')`. Or this can be used to round a number value: `-columnExpression=price:round(?, 3)`
-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 only be used when inserting rows (e.g. using `-mode=insert`) The format of the values is `-constantValues="column1=value1,column2=value2"`. The parameter can be repeated multiple times, to make quoting easier: `-constantValues="column1=value1" -constantValues="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='${some.value}'"`. You can also specify a `SELECT` statement that retrieves information from the database based on values from the input file. This is useful when the input file contains e.g. values from a lookup table (but not the primary key from the lookup table). The syntax to specify a SELECT statement is similar to a function call: `-constantValues="$@{SELECT type_id FROM type_definition WHERE type_name = $4"` where `$4` references the fourth column from the input file. The first column is $1 (not $0). The parameter for the SELECT statement do not need to be quoted as internally a prepared statement is used. However the values in the input file must be convertible by the JDBC driver. If the input column from the source file is not part of the target table, the value will be passed as a string to the statement. This means that the SQL query should cast the parameter to the appropriate data type if needed, e.g. `where some_id = casst($5 as integer)`.

Parameter	Description
	In addition to the function call or `SELECT` statements, WbImport provides four variables that can be used to access the name of the currently imported file. This can be used to store the source file of the data in the target table. The following three variables are supported • `_wb_import_file_path` this contains the full path and file name of the current import file • `_wb_import_file_name` this contains only the file name (without the path) • `_wb_import_file_dir` this contains the name of the directory in which the file is stored • `_wb_import_line_number` this will be replaced with the current line number of the input file during the import Please refer to the examples for more details on the usage.
-insertSQL	Define the statement to be used for inserting rows. This can be used to use hints or customize the generated INSERT statement. The parameter may only contain the `INSERT INTO` part of the statement (i.e. `INSERT INTO` is the default if nothing is specified). This can be used to pass special hints to the database, e.g. to specify an append hint for Oracle:  You have to quote the parameter value using single quotes, otherwise comments will be removed from the SQL statement! `-insertSQL='INSERT /*+ append */ INTO'`
-adjustSequences	Possible values: `true`, `false` For DBMS that support sequences which are associated with a column, this parameter can be used to adjust the next value for the sequence to the maximum value of the imported data. This can also be used to synchronize identity columns for DBMS that allow overriding the generated values. Currently this is implemented for PostgreSQL, DB2 (LUW), H2 Database and HyperSQL (aka HSQLDB).

-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`.

Parameter	Description
	These statements are only used if more than one table is processed.
-runTableStatementOnError	Possible values: `true`, `false` Controls the execution of the post-table statement in case an error occurred while importing the data. By default the post-table statement is executed even if the import was not successful. If this is should not happen, use `-runTableStatementOnError=false`.
-ignorePrePostErrors	Possible values: `true`, `false` Controls handling of errors for the SQL statements defined through the `-preTableStatement` and `-postTableStatement` parameters. If this is set to true (the default), errors resulting from executing the supplied statements are ignored. If set to false then error handling depends on the parameter `-continueOnError`.
-showProgress	Valid values: `true`, `false`, `<numeric value>` Control the update frequence in the status bar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specify a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).

21.3. 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 appropriate 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.
-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 The character can also be specified as a unicode escape sequence, e.g. `-delimiter='\u0009'` Default value: \t (a tab character)

Parameter	Description
-columnWidths	In order to import files that do not have a delimiter but have 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 in characters for each column. If this parameter is given, the `-delimiter` parameter is ignored. The order of the columns in the input file must still be defined using the `-fileColumns` parameter. e.g.: `-fileColumns=custid,actcode,regioncd,flag -columnWidths='custid=10,actcode=5,regioncd=3,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.
-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. The character can also be specified as a unicode escape sequence, e.g. `-quoteChar='\u0022'`
-quoteAlways	Possible values: `true`, `false` WbImport will always handled quoted values correctly, if a quote character is defined through -quoteChar. Using `-quoteAlways=true` enables the distinction between NULL values and empty strings in the import file, but only if `-quoteAlways=true` has also been used when running WbExport. Remember to also use `-emptyStringIsNull=false`, as by default empty string values are treated as NULLs
-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 backslash, 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.

Parameter	Description
	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.
-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 WbExport's 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.
-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.
-nullString	Defines the string value that in the input file to denote a `NULL` value. The value of this is case-sensitive, so `-nullString=NULL` is different to `-nullString=null`
-blobIsFilename	Possible values: `true`, `false` This is a deprecated parameter. Please use `-blobType` instead. 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 parameter 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` This parameter is ignored, if `-blobType` is also specified.
-blobType	Possible values: `file`, `ansi`, `base64` Specifies how BLOB data is stored in the input file. If `file` is specified, it is assumed that the column value contains a filename that in turn contains the real blob data. This is the default format when using WbExport. For the other two type, `WbImport` assumes that the blob data is stored as encoded character data in the column.

Parameter	Description
	If this parameter is specified, `-blobIsFilename` is ignored.
-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.
-usePgCopy	This parameter has no value, its presence turns the feature on. If this parameter is specified, then the input file is sent to the PostgreSQL server using PostgreSQL's JDBC support for COPY The specified file(s) must conform to the format expected by PostgreSQL's COPY command. SQL Workbench/J creates a `COPY tablename (column, ...) FROM stdin WITH (format csv, delimiter '|', header true)` statement and then executes this, passing the actual file contents through the JDBC API. As `COPY` does not support "merging" of data, the only allowed import mode is `insert`. If a different mode is specified through the `-mode` parameter, an error will be reported. The options defined in the `WITH (...)` part are influenced by the parameters passed to `WbImport`. However `COPY` does not support all options that `WbImport` does. To control the format of the input file(s) **only** the following parameters are relevant when using `-usePgCopy`: • -header • -encoding • -delimiter Especially the formatting options for dates/timestamps and numbers will have **no** effect. So the input file must be formatted properly. All parameters controlling the target table(s), the columns, the source directory and so on still work. Including the import directly from a ZIP archive.

21.4. Text Import Examples

21.4.1. Importing date columns

```
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 `birthday`. 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).

21.4.2. Excluding input columns from the import

```
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.

21.4.3. Importing a file with fixed column widths

```
WbImport -file=cust_data.txt
         -table=customer
         -filecolumns=custnr,accountid,region_code
         -columnWidths='custnr=10,accountid=10,region_code=2';
```

This will import a file with three columns. The first column named `custnr` is taken from the characters 1-10, the second column named `accountid` is taken from the characters 21-30 and the third the column `region_code` is taken from characters 31 and 32

21.4.4. Filtering rows during import

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 https://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.

21.4.5. Importing several files

```
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.

21.4.6. Storing the name of the source file

The following statement will import all `.txt` files from the directory `/data/import` and store them in the appropriate tables. Each table that is being imported has to have a column named `source_file` and the complete path to the import file will be stored in that column (for each imported row).

```
WbImport -sourceDir=/data/import
         -header=true
         -schema=staging
         -extension=txt
         -constantValues="source_file=$[_wb_import_file_path]"
         -type=text;
```

21.4.7. Populating columns from the database

When your input file does not contain the actual values to be stored in the target table, but e.g. lookup values, you can specify a SELECT statement to retrieve the necessary primary key of the lookup table.

Consider the following tables:

```
contact (contact_id, first_name, last_name, type_id)
contact_type (type_id, type_name)
```

The table `contact_type` contains: (1, 'business'), (2, 'private'), (3, 'other').

Your input file only contains `contact_id, first_name, last_name, type_name`. Where `type_name` references an entry from the `contact_type` table.

To import this file, the following statement can be used:

```
WbImport -file=contacts.txt
         -type=text
         -header=true
         -table=contact
         -importColumns=contact_id, first_name, last_name
         -constantValues="type_id=$@{SELECT type_id FROM contact_type WHERE type_name = $4
```

For every row from the input file, SQL Workbench/J will run the specified SELECT statement. The value of the first column of the first row that is returned by the SELECT, will then be used to populate the `type_id` column. The SELECT statement will use the value of the third column of the row that is currently being inserted as the value for the WHERE condition.

You must use the -importColumns parameter as well to make sure the type_name column is not processed! As an alternative you can also use `-fileColumns=contact_id, first_name, last_name, $wb_skip$` instead of -importColumns.



The "placeholders" with the column index must not be quoted (e.g. '$1' for a character column will not work)!

If the column `contact_id` should be populated by a sequence, the above statement can be extended to include a function call to retrieve the sequence value (PostgreSQL syntax:)

```
WbImport
  -file=contacts.txt
  -type=text
  -header=true
  -table=contact
  -importColumns=first_name, last_name
  -constantValues="id=${nextval('contact_id_seq'::regclass)}"
  -constantValues="type_id=$@{SELECT type_id FROM contact_type WHERE type_name = $4}"
```

As the ID column is now populated through a constant expression, it may not appear in the `-importColumns` list. Again you could alternatively use `-fileColumns=$wb_skip$, first_name, last_name, $wb_skip$` to make sure the columns that are populated through the -constantValue parameter are not taken from the input file.

21.5. 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.

Parameter	Description
	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 omited. 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`.

21.6. Parameters for spreadsheet import

Both spreadsheet imports (Microsoft Excel, OpenOffice) support a subset of the parameters that are used for flat file imports.

These parameters are:

- `-header`
- `-fileColumns`
- `-importColumns`
- `-nullString`
- `-emptyStringIsNull`
- `-illegalDateIsNull`

The spreadsheet import does not support specifying a date or timestamp format unless `-stringDates=true` is used. It is expected that those columns are formatted in such a way that they can be identified as date or timestamps.

The spreadsheet import also does not support importing BLOB files that are referenced from within the spreadsheet. If you want to import this kind of data, you need to convert the spreadsheet into a text file.

The spreadsheet import supports the following additional parameters that arer not available for the text imports:

Parameter	Description
-sheetNumber	Selects the spread sheet inside the file to be imported. If this is not specified the first sheet is used. The first sheet has the number 1. All sheets can be imported with a single command when using `-sheetNumber=*`. In that case, it is assumed that each sheet has the same name as the target table. If all sheets are imported, the parameters `-table`, `-fileColumns` and `-importColumns` are ignored. For sheets where the sheet name does not match the target table, the correct table name can be specified using the -sheetTableName parameter.
-sheetName	Defines the name of the spreedsheet inside the file to be imported. If this is not specified the first sheet is used. As with `-sheetNumber`, all sheets can be imported by using `-sheetName=*`. In that case, it is assumed that each sheet has the same name as the target table.

Parameter	Description
	If all sheets are imported, the parameters `-table`, `-fileColumns` and `-importColumns` are ignored.
-sheetTableName	If importing multiple sheets from a single file, it is assumed that the name of each sheet defines the target table into which the sheet's data should be imported. For those sheets, where the names don't match the name of the target table, it's possible to specify the target table using the parameter `-sheetTableName`. The value of the parameter is a key/value pair delimited with a colon. The first element is the sheet name the second is the target table, e.g. `-sheetTableName=PDataSheet:public.person` will import the sheet named `PDataSheet` into the table `person`. The parameter can be repeated multiple times, to specify mappings for different sheets and tables. For all sheets that are not mapped, the sheet name will be assumed to be the target table.
-stringDates	Possible values: `true`, `false` By default WbImport tries to read "native" date and timestamp values from an Excel Worksheet. When this parameter is set to `true`, the values for date and timestamp values will be retrieved as a (formatted) string value and then converted using the format specified through the `-timestampFormat` and `-dateFormat` parameters.
-recalculateFormulas	Possible values: `true`, `false` By default WbImport will recalculate all formulas in an Excel workbook after loading it. This can be controlled with this parameter. `-recalculateFormulas=false` will prevent the recalculation and use the cached values stored in the workbook.

21.7. 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 key columns. 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.

To use insert/update or update/insert with PostgreSQL, make sure you have enabled savepoints for the import (which is enabled by default).

21.8. Native UPSERT mode

When using a DBMS that supports an "update or insert" functionality directly, this can be selected using `-mode=upsert`. In this case SQL Workbench/J will only use a single statement instead of two statements as described in the previous chapter. The advantage of using this mode over e.g. `insert,update` is that fewer statements are sent to the database, and that this mode supports the use of batching, which is not possible when using `insert,update`.

For the following database systems, native UPSERT is available:

- PostgreSQL 9.5 or later, using `INSERT ... ON CONFLICT`: https://www.postgresql.org/docs/current/static/sql-insert.html
- Firebird 2.1, using `UPDATE OR INSERT`: https://www.firebirdfaq.org/faq220/
- H2 Database, using `MERGE INTO`: https://www.h2database.com/html/grammar.html#merge
- Oracle, Microsoft SQL Server, HSQLDB 2.x and DB2 (LUW and z/OS) using a `MERGE` statement
- SQL Anyhwere using `INSERT ... ON EXISTING UPDATE` (this requires a primary key)
- SQLite using `INSERT OR REPLACE ...` (this requires a primary key)
- SAP HANA using a `UPSERT` statement
- MySQL using `INSERT ... ON DUPLICATE`: https://dev.mysql.com/doc/refman/5.1/en/insert-on-duplicate.html As MySQL does not allow to specify the key columns for the "ON DUPLICATE" part, this is only supported when the table has a primary key.

21.9. Native insertIgnore mode

The `-mode=insertIgnore` will use the built in feature of the DBMS to (silently) ignore inserts that would result in a violation of a unique key constraint but not update existing rows. Using `-mode=insertIgnore` has the same effect as using `-mode=insert -continueOnError=true` but will perform better (especially when *many* collisions are expected) because this can be combined with batching and it does not require the use of savepoints (e.g. for Postgres)

This mode is supported for the following DBMS:

- PostgreSQL 9.5 or later, using `INSERT ... ON CONFLICT`: https://www.postgresql.org/docs/current/static/sql-insert.html
- Oracle, using the `IGNORE_ROW_ON_DUPKEY_INDEX` hint: https://docs.oracle.com/cd/E11882_01/server.112/e41084/sql_elements006.htm#CHDEGDDG
- Microsoft SQL Server, HSQLDB 2.x and DB2 (LUW and z/OS) using a `MERGE` statement without a `WHEN NOT MATCHED` clause.

- SQLite using `INSERT OR IGNORE ...` (this requires a primary key)
- SQL Anyhwere using `INSERT ... ON EXISTING SKIP` (this requires a primary key)
- MySQL using `INSERT ... ON DUPLICATE`: https://dev.mysql.com/doc/refman/5.1/en/insert-on-duplicate.html with a dummy update setting one column to its current value. As MySQL does not allow to specify the key columns for the "ON DUPLICATE" part, this is only supported when the table has a primary key.

22. 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 basically 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 from the computer running SQL Workbench/J



Some JDBC drivers (e.g. PostgreSQL, jTDS and the Microsoft Driver) read the full result obtained from the database into memory. In that case, copying large results might require a lot of memory. Please refer to the chapter Common problems for details on how to configure the individual drivers if this happens to you.

22.1. General parameters for the `WbCopy` command.

`WbCopy` supports conditional execution

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. This parameter supports auto-completion
-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.
-sourceConnection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs: • `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` For a sample connection string please see the documentation for WbConnect. If this parmeter is specified, `-sourceProfile` is ignored
-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. This parameter supports auto-completion

Parameter	Description
-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.
-targetConnection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs: • `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` If this parmeter is specified, `-sourceProfile` is ignored
-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 using a `DELETE` statement before copying the data.
-truncateTable	Possible values: `true`, `false` If this parameter is set to true, all rows are removed from the target table using a `TRUNCATE` statement before copying the data. Not all DBMS support the `TRUNCATE` command. If this option is used for a DBMS that does not support `TRUNCATE` or where the target table(s) cannot be truncated due to other restrictions, an error will occur.
-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.

Parameter	Description
	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. It is assumed that the values for the key columns will never be `NULL`.
-ignoreIdentityColumns	Possible values: `true`, `false` Controls if identity or auto-increment columns will be included in the import. If this is used, the JDBC driver (of the target database) must correctly report the column to be excluded as an *AUTOINCREMENT* column. This can be verified in the table definition display of the DbExplorer. If the column is reported with `YES` for the *AUTOINCREMENT* property, then this column will be excluded during the import.
-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. 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 `-batchSize` 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 using `-useSavepoint=true`.
-transactionControl	Possible values: `true`, `false` Controls if SQL Workbench/J handles the transaction for the copy, or if the copy must be committed (or rolled back) manually. If `-transactionControl=false` is specified, SQL Workbench/J will neither send a `COMMIT` nor a `ROLLBACK` at the end. This can be used when multiple copy statements should be run in a single transaction. This can be combined with the cleanup and error scripts in batch mode.
-useSavepoint	Possible values: `true`, `false` Controls if SQL Workbench/J guards every insert or update statement with a savepoint to recover from individual error during import, when `continueOnError` is set to true. Using a savepoint for each DML statement can drastically reduce the performance of the import.

Parameter	Description
-maxRows	Limit the number of rows read from the source table(s) if the value for this parameter is greater than 0 (zero). This limit is always applied to all tables which are specified. The limiting of rows is done without adding an `ORDER BY` to the source query. This means that the sample of the rows is not deterministic, when running the statement again, a different set of rows might be selected. This option is primarily intended for testing purposes.
-trimCharData	Possible values: `true`, `false` If this parameter is set to true, values from `CHAR` columns will be trimmed from trailing whitespace. This is equivalent to the Trim CHAR data in the connection profile.
-showProgress	Valid values: `true`, `false`, `<numeric value>` Control the update frequence in the status bar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specify a value of 0 (zero) or the value `false`. `true` will set the progress interval to 1 (one).

22.2. Copying data from one or more tables

Parameter	Description
-sourceSchema	The name of the schema to be copied. When using this parameter, all tables from the specified schema are copied to the target. You must specify either `-sourceSchema`, `-sourceTable` or `-sourceQuery`
-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.
-includeViews	Valid values are `true` or `false`. By default only regular tables are taken into account if the source tables are specified through a wildcard. Using `-includeViews=true`, the search for matching names will include views (in addition to regular tables).
-excludeTables	The tables listed in this parameter will not be copied. This can be used when all but a few tables should be copied from one database to another. First all tables specified through `-sourceTable` will be evaluated. The tables specified by `-excludeTables` can include wildcards in the same way, `-sourceTable` allows wildcards. `-sourceTable=* -excludeTables=TEMP*` will copy all tables, but not those starting with `TEMP`. This parameter supports auto-completion.
-checkDependencies	Valid values are `true` or `false`. 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`
-targetSchema	The name of the target schema into which the tables should be copied. When this parameter is not specified, the default schema of the target connection is used.

Parameter	Description
-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`.  Using `-createTarget=true` is intended as a quick and dirty way of creating a target table "on the fly" during the copy process. Tables created this way should not be considered "production-ready". The created tables will only have the primary key and not-null constraints created. All other constraints from the source table are ignored. Because the automatic mapping of table definitions will only work in the most simple cases this feature is not suited to synchronize the table definitions between two different DBMS products. Because of these limitations this feature can not considered a replacement for a proper schema management. If you have the requirement to keep the schema definition of different DBMS in sync please consider a tool like Liquibase or Flyway. Do not try to use `WbCopy` for this. If you want to migrate a table (or several tables) from one DBMS to another, consider using WbSchemaReport together with an XSLT transformation When using this option with different source and target DBMS, the information about the data types to be used in the target database are retrieved from the JDBC driver. In some cases this information might not be accurate or complete. You can enhance the information from the driver by configuring your own mappings in workbench.settings. Please see the section Customizing data type mapping for details. If the automatic mapping generates an invalid `CREATE TABLE` statement, you will need to create the table manually in the target database.
-removeDefaults	Valid values are `true` or `false`. This parameter is only valid in combination with `-createTarget=true`. If set to `true`, any default value (or expression) defined on the source table will be ignored when creating the target table. This is useful if the source and target DBMS use different syntax for default values.
-tableType	When `-createTarget` is set to `true`, this parameter can be used to control the SQL statement that is generated to create the target table. This is useful if the target table should e.g. be a temporary table When using the auto-completion for this parameter, all defined "create types" that are configured in workbench.settings (or are part of the default settings) are displayed together with the name of the DBMS they are used for. The list is **not** limited to definitions for the target database! The specified type must nonetheless match a type defined for the target connection. If you specify a type that does not exist, the default `CREATE TABLE` will be used. For details on how to configure a CREATE TABLE template for this parameter, please refer to the chapter Settings related to SQL statement generation

Parameter	Description
-skipTargetCheck	Normally WbCopy will check if the specified target table does exist. However, some JDBC drivers do not always return all table information correctly (e.g. temporary tables). If you know that the target table exists, the parameter `-skipTargetCheck=true` can be used to tell WbCopy, that the (column) definition of the source table should be assumed for the target table and not further test for the target table will be done.
-dropTarget	Possible values: `false`, `true`, `cascade` If this parameter is set to `true` the target table will be dropped before it is created. For database systems that support it (Oracle, PostgreSQL), a `DROP ... CASCADE` is used when the value `cascade` is specified. If the DBMS does not support cascading drops, this is the same as specifying `true`.
-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 column (i.e. one with the same name) in the target table. If `-createTarget=true` is specified then this list also defines the columns of the target table to be created. The names have to be separated by comma: `-columns=firstname, lastname, zipcode` 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 `-createTarget=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. When using a SQL *query* as the data source a *mapping* cannot be specified. Please check Copying data based on a SQL query for details.
-adjustSequences	Possible values: `true`, `false` For DBMS that support sequences which are associated with a column, this parameter can be used to adjust the next value for the sequence to the maximum value of the imported data. This can also be used to synchronize identity columns for DBMS that allow overriding the generated values. Currently this is implemented for PostgreSQL, DB2 (LUW), H2 Database and HyperSQL (aka HSQLDB).

-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"`

Parameter	Description
	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`. These statements are only used if more than one table is processed.
-runTableStatementOnError	Possible values: `true`, `false` Controls the execution of the post-table statement in case an error occurred while importing the data. By default the post-table statement is executed even if the import was not successful. If this is should not happen, use `-runTableStatementOnError=false`.
-ignorePrePostErrors	Possible values: `true`, `false` Controls handling of errors for the SQL statements defined through the `-preTableStatement` and `-postTableStatement` parameters. If this is set to true (the default), errors resulting from executing the supplied statements are ignored. If set to false then error handling depends on the parameter `-continueOnError`.

22.3. Copying data based on a SQL query

Parameter	Description
-sourceQuery	The SQL query to be used as the source data (instead of a table). This parameter is ignored if `-sourceSchema` or `-sourceTable` is specified.
-columns	The *list* of columns from the target table, in the order in which they appear in the source query. If the column names in the query match the column names in the target table, this parameter is not necessary. If you do specify this parameter, note that this is **not** a column mapping. It only lists the columns in the correct order .

22.4. 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.

22.5. 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 [131] from `update,insert` or `insert,update` mode to `insert` mode applies for synchronizing tables using `WbCopy` as well.

22.6. Examples

22.6.1. Copy one table to another where all column names match

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

22.6.2. 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;
```

22.6.3. Copy only selected rows

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

This example will run the statement `SELECT * FROM the_table WHERE lastname like 'D%'` and copy all corresponding columns to the target table `the_other_table`.

22.6.4. Copy data between tables with different columns

This example copies only selected columns from the source table. The column names in the two 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=contacts
```

```
       -deleteTarget=true
       -columns=firstname/surname, lastname/name, birthday/dob;
```

22.6.5. Copy data based on a SQL query

When using a query as the source for the `WbCopy` command, the column mapping is specified by simply supplying the order of the target columns as they appear in the `SELECT` statement.

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

This copies the data based on the SELECT statement into the table `CONTACTS` of the target database. The `-columns` parameter defines that the first column of the SELECT (firstname) is copied into the target column with the name `surname`, the second result column (lastname) is copied into the target column `name` and the last source column (birthday) is copied into the target column `dob`.

This example could also be written as:

```
WbCopy -sourceProfile=ProfileA
       -targetProfile=ProfileB
       -sourceQuery="SELECT firstname as surname, lastname as name, birthday as dob FROM p
       -targetTable=contacts
       -deleteTarget=true
```

23. Comparing databases



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

There are two SQL Workbench/J specific commands that can compare either the structure of two databases or the data contained in them.

These commands (`WbSchemaDiff` and `WbDataDiff`) can be used like any other SQL command as long as they are run using SQL Workbench/J This includes the usage in scripts that are run in batch mode.

23.1. Compare two database schemas - WbSchemaDiff

`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 WbXSLT Command). Sample XSLT transformations are stored in the `xslt` subdirectory of the SQL Workbench/J installation directory. All scripts that are part of the download can also be found on the SQL Workbench/J homepage



This feature should only be considered as a one-off solution to quickly compare two database schemas. Is not intended to replace a proper schema (script) management. You should consider tools like Liquibase or Flyway to manage a database schema. Those scripts should also be stored in a version control system (Subversion, Git, ...)

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.
-referenceConnection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs: • `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` For a sample connection string please see the documentation for WbCopy. If this parameter is specified `-referenceProfile` will be ignored.

Parameter	Description
-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.
-targetConnection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs: • `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` For a sample connection string please see the documentation for WbConnect. If this parameter is specified `-targetProfile` will be ignored.
-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.
-excludeTables	A comma separated list of tables or views that should not be compared. If tables from several schemas are compared (using `-referenceTables=schema_one.*, schema_two.*`) then the listed tables must be qualified with a schema, e.g. `-excludeTables=schema_one.foobar, schema_two.fubar`
-encoding	The encoding to be used for the XML file. The default is UTF-8

Parameter	Description
-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`.
-includeTriggers	Select whether table triggers are compared as well. The default value is `true`.
-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. The default is to compare table constraints (`true`) Valid values are `true` or `false`.
-useConstraintNames	When including check constraints this parameter controls whether constraints should be matched by name, or only by their expression. If comparing by names the diff output will contain elements for constraint modification otherwise only drop and add entries will be available. The default is to compare by names(`true`) Valid values are `true` or `false`.
-includeViews	Select whether views should also be compared. Note that this comparison is very unreliable, because this compares the source code, not the logical representation of the view definition. The source code is compared the way it is returned by 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. Aslo different indention or empty lines will result in the views being reported as "different". A comparison across different DBMS will not work. 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-sensitive. A comparison across different DBMS will also not work! 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 being different to a `VARCHAR(100)` column in PostgreSQL which is not really true As both drivers report the column as java.sql.Types.VARCHAR, they would be considered as identical when using `-useJdbcType=true`. Valid values are `true` or `false`.
-additionalTypes	Select additional object types that are not compared by default (using the `-includeXXX` parameters) such as Oracle `TYPE` definitions. Those objects are compared on source code level (like procedures) rather than on attribute level.

Parameter	Description
	Valid values are object type names as shown in the "Type" drop down in the DbExplorer.
-xsltParameter	A list of parameters (key/value pairs) that should be passed to the XSLT processor. When using e.g. the `wbreport2liquibase.xslt` stylesheet, the value of the `author` attribute can be set using `-xsltParameter="authorName=42"`. This parameter can be provided multiple times for multiple parameters, e.g. when using `wbreport2pg.xslt`: `-xsltParameter="makeLowerCase=42" -xsltParameter="useJdbcTypes=true"`

WbSchemaDiff Examples

Compare all tables between two connections, and write the output to the file `migrate_prod.xml` and convert the XML to a series of SQL statements for PostgreSQL

```
WbSchemaDiff -referenceProfile="Staging"
             -targetProfile="Production"
             -file=migrate_prod.xml
             -styleSheet=wbdiff2pg.xslt
             -xsltOutput=migrate_prod.sql
```

Compare a list of matching tables between two databases and write the output to the file `migrate_staging.xml` ignoring all tables that start with `TMP_` and exclude any index definition from the comparison. Convert the output to a SQL script for Oracle

```
WbSchemaDiff -referenceProfile="Development"
             -targetProfile="Staging"
             -file=migrate_stage.xml
             -excludeTables=TMP_*
             -includeIndex=false
             -styleSheet=wbdiff2oracle.xslt
             -xsltOutput=migrate_stage.sql
```

23.2. 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 performance 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.



Comparing data across different DBMS might not be accurate or correct. Different JDBC drivers use different Java represenations of the values retrieved. `WbDataDiff` only works reliably if the source and target are the same DBMS and both connections use the same version of the JDBC driver.

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
-referenceConnection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs: • `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` For a sample connection string please see the documentation for WbCopy. If this parameter is specified `-referenceProfile` will be ignored.
-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.
-targetConnection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs:

Parameter	Description
	• `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` For a sample connection string please see the documentation for WbConnect. If this parameter is specified `-targetProfile` will be ignored.
-tableWhere	This defines an additional WHERE condition to be used for **all** tables that are compared. This requires the columns referenced in the condition to exist in all tables. It is not necessary to provide the WHERE keyword. This can be used to limit the number of rows to be compared based on e.g. a timestamp: `-tableWhere="modified_at >= timestamp '2023-07-31 19:30:00'"`. Note that the column must not be prefixed with a table name or table alias. If the provided condition is invalid for a table, the command will fail.
-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. You can enable writing a single file that includes all statements for all tables by using the parameter `-singleFile=true`
-singleFile	If this parameter's value is `true`, then only one single file containing all statements will be written.
-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.

Parameter	Description
-tableTypes	Defines a list of "table types" that are used to retrieve the tables to be compared. By default the list of types reported by the JDBC driver are used. This parameter can be used when data in tables having a non-default type should be compared. The necessary values can be seen in the `TYPE` column in the DbExplorer's table list.
-excludeTables	A comma separated list of tables that should not be compared. If tables from several schemas are compared (using `-referenceTables=schema_one.*, schema_two.*`) then the listed tables must be qualified with a schema, e.g. `-excludeTables=schema_one.foobar, schema_two.fubar`
-ignoreMissing	If this parameter is true, `WbDataDiff` will still generate INSERT statements for tables that are not found in the target database.
-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. Note that enabling this option will result in additional overhead reading the target table's data and will slow down the overal comparison speed. The default is `false`.
-type	Valid values are `sql`, `xml` Defines the type of the generated files.
-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` XML files are always stored in UTF-8
-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.
-includeIdentityColumns	Valid values: `true`, `false` Controls if "identity" or "auto increment" columns are included in the generated generated SQL statements. This overwrites the global option that controls this behaviour. This parameter is ignored if the output type is XML.
-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. They will however be part of generated `INSERT` or `UPDATE` statements unless `-excludeIgnored=true` is also specified.
-excludeIgnored	Valid values: `true`, `false`

Parameter	Description
	If this is set to `true`, the ignored columns will also be removed from any `INSERT` or `UPDATE` statement The default is `false`.
-alternateKey	With this parameter alternate keys can be defined for the tables that are compared. To specify alternate keys for multiple tables, the parameter must be repeated for each table. `-alternateKey='table_1=column_1,column_2' -alternateKey='table_2=other_col1,other_col2'` Note that each value has to be enclosed in either single or double quotes to mask the equals sign embedded in the parameter value. Once an alternate (primary) key has been defined, the primary key columns defined on the tables are ignored. By default the real PK columns will however be included in `INSERT` statement that are generated. To avoid this, set the parameter `-excludeRealPK` to true.
-excludeRealPK	Valid values are `true`, `false`. This parameter controls the usage of the real PK columns in case alternate PK columns are defined. If set to `true` the real PK columns are excluded from generated `INSERT` statements (assuming that a new PK value will be generated during inserting the rows). Note that this parameter will enable/disable the use of the real PK columns for **all** tables for which alternate key columns were defined. This parameter has no effect if no alternate keys were specified using the `-alternateKey` option.
-showProgress	Valid values: `true`, `false`, `<numeric value>` Control the update frequence in the status bar (when running in GUI mode). The default is every 10th row is reported. To disable the display of the progress specify 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
```

24. Search data and code in the database



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

24.1. Search source of database objects - WbGrepSource

The command `WbGrepSource` can be used to search in the source code of the specified database objects.

The command basically retrieves the source code for all selected objects and does a simple search on that source code. The source code that is searched is identical to the source code that is displayed in the "Source" tab in the various DbExplorer panels.

The search values can be regular expressions. When searching the source code the specified expression must be found somewhere in the source. The regex is *not* used to match the *entire* source.

The command supports the following parameters:

Parameter	**Description**
-searchValues	A comma separated list of values to be searched for.
-useRegex	Valid values are `true`, `false`. If this parameter is set to true, the values specified with `-searchValues` are treated as regular expression The default for this parameter is `false`.
-matchAll	Valid values are `true`, `false`. This specifies if all values specified with `-searchValues` have to match or only one. The default for this parameter is `false`.
-ignoreCase	Valid values are `true`, `false`. When set to true, the comparison is be done case-insesitive ("ARTHUR" will match "Arthur" or "arthur"). The default for this parameter is `true`.
-types	Specifies if the object types to be searched. The values for this parameter are the same as in the "Type" drop down of DbExplorer's table list. Additionally the types `function`, `procedure` and `trigger` are supported. When specifying a type that contains a space, the type name neeeds to be enclosed in quotes, e.g. `-types="materialized view"`. When specifying multiple types, the whole argument needs to be enclosed in quotes: `-types='table, materialized view'` The default for this parameter is `view, procedure, function, trigger, materialized view`. To search in all available object types, use `-types=*`. This parameter supports auto-completion.
-objects	A list of object names to be searched. These names may contain SQL wildcards, e.g. `-objects=PER%,NO%`

Parameter	Description
	This parameter supports auto-completion.
-schemas	Specifies a list of schemas to be searched (for DBMS that support schemas). If this parameter is not specified the current schema is searched. This parameter supports auto-completion.

The functionality of the `WbGrepSource` command is also available through a GUI at Tools » Search in object source

24.2. Search data in multiple tables - WbGrepData

The command `WbGrepData` can be used to search for occurrences of a certain value in all columns of multiple tables. It is the command line version of the (client side) Search Table Data tab in the DbExplorer. A more detailed description on how the searching is performed is available that chapter.

`WbGrepData` supports searching on the client side or on the server using SQL expression. Searching on the client side is more flexible, but is typically slower and might lead to memory problems when searching large tables.



The search on the client side retrieves all the data of a table using `SELECT * FROM the_table` and processes that result row-by-row. Although SQL Workbench/J only keeps one row at a time in memory, it is possible that the JDBC drivers caches the full result set in memory. Please see the chapter Common problems for your DBMS to check if the JDBC driver you are using caches result sets.

The following parameters are valid for the client side and server side search:

Parameter	Description
-searchValue	The value to be searched for. This parameter is ignored when using `isNull` for the compare type.
-tables	A list of table names to be searched. These names may contain SQL wildcards, e.g. `-tables=PER%,NO%`. If you want to search in different schemas, you need to prefix the table names, e.g. `-tables=schema1.p%,schema2.n%`. This parameter supports auto-completion.
-types	By default `WbGrepData` will search all tables and views (including materialized views). If you want to search only one of those types, this can be specified with the `-types` parameter. Using `-types=table` will only search table data and skip views in the database. This parameter supports auto-completion.
-excludeTables	A list of table names to be excluded from the search. If e.g. the wildcard for -tables would select too many tables, you can exclude individual tables with this parameter. The parameter values may include SQL wildcards. `-tables=p% -excludeTables=product_details,product_images` would process all tables starting with P but not the `product_detail` and the `product_images` tables.

Parameter	Description
-columns	By default `WbGrepData` will search all columns of a table (or view). If you want to limit the search to some columns only, you can define a list of column names to be included. Wildcards are not supported.
-retrieveCLOB	By default `CLOB` columns will be retrieved and searched. If this parameter is set to `false`, CLOB columns will not be retrieved. If the search value is not expected in columns of that type, excluding them from the search will speed up data retrieval (and thus the searching). Only columns reported as `CLOB` by the JDBC driver will be excluded. If the driver reports a column as `VARCHAR` this parameter will not exclude that column.
-serverSearch	When this is true, the server side search is enabled. The default is false, which enables the client side search.

The following parameters are valid only for the client side search:

Parameter	Description
-ignoreCase	Valid values are `true`, `false`. When set to true, the comparison is be done case-insensitive ("ARTHUR" will match "Arthur" or "arthur"). The default for this parameter is `true`.
-compareType	Valid values are `contains`, `equals`, `matches`, `startsWith`, `isNull` When specifying `matches`, the search value is used as a regular expression. A column is included in the search result if the regular expression is *contained* in the column value (not when the column value matches the regular expression entirely). The default for this parameter is `contains`.
-retrieveBLOB	By default `BLOB` columns will not be retrieved for those rows that match the criteria to avoid excessive memory usage. If `BLOB` columns should be retrieved, this parameter needs to be set to `true`. Enabling this will not search inside the binary data. If `BLOB` columns should be searched (and treated as character data), use the `-treatBlobAs` parameter
-treatBlobAs	If this parameter specifies a valid encoding, binary (aka "BLOB") columns will be retrieved and converted to a character value using the specified encoding. That character value is then searched.

Parameter	Description
	`-treatBlobAs="UTF-8"` would convert **all** `BLOB` columns in **all** tables that are searched to a character value using UTF-8 as the encoding. Therefore using this option usually only makes sense if a single table is searched.

The following parameters are valid only for the server side search:

Parameter	Description
-sqlOperator	The SQL operator to be used for searching. The default operator is `LIKE`. The value provided is used "as-is", so it has to be valid for the current DBMS. Note that the search value must include wildcards that match the chosen operator e.g. `-searchValue="%foobar%"` if `LIKE` is used.

24.2.1. Examples

Search for a specific value in a single table

The following statement will search for the text `Arthur` in all columns and all rows of the table `person`. It will find values `foobar`, `somefoo` or `notfoobar`:

```
WbGrepData -searchValue=foo -tables=person -ignoreCase=true
```

`-ignoreCase=true` is the default behavior and can be omitted.

Search for a specific value in all tables

The following statement will search for the text `foobar` in all columns and all tables.

```
WbGrepData -searchValue=foobar -tables=*
```

Search for a specific value at the beginning of a column value

The following statement will search for the text `foo` in all columns and all tables. It will match the value `foobar`, but not `barfoo`

```
WbGrepData -searchValue=foo -compareType=startsWith -tables=*
```

Search for a specific value with an exact match

The following statement will search for the text `foo` in all columns and all tables. It will only match the value `foo` or `FOO` but not `somefoobar`

```
WbGrepData -searchValue=foo -compareType=equals -tables=*
```

Search for patterns

The following statement will search for any value where three characters are followed by two numbers. It will match `foo42`, `bar12`

```
WbGrepData -searchValue="[a-z]{2}[0-9]{2}" -compareType=contains -tables=person
```

As the column values are only tested if the regular expression is *contained*, not if it is an exact match. The above search will also return `foo999`.

To get an exact match using the `contains` type, the regular expression needs to be anchored at the start and the end. The following will only find only values that start with (exactly) two characters and are followed by (exactly) two digits.

```
WbGrepData -searchValue="^[a-z]{2}[0-9]{2}$" -compareType=contains -tables=person
```

Search for multiple values

The following statement will return rows where any column either contains the value `foo` or the value `bar`:

```
WbGrepData -searchValue="foo|bar" -compareType=contains -tables=person
```

As the column values are only tested if the regular expression is *contained*, not if it is an exact match. The above search will also return `foo999`.

For more information about regular expressions please visit: Regular-Expressions.info

Search on the server side using a Postgres regular expression

The following statement will use the condition `column *~ '(foo|bar)'` for each column in the table `person`.

```
WbGrepData -serverSearch=true -searchValue="(foo|bar)" -sqlOperator="*~" -tables=person
```

25. SQL Workbench/J to generate DDL commands



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

25.1. Generate SQL script for database objects - WbGenerateScript

`WbGenerateScript` re-creates the SQL for objects in the database. It is the command line version of the Generate Script option in the DbExplorer

The command supports the following parameters.

Parameter	Description
-objects	A comma separated list of table (views or other objects), e.g. `-objects=customer,invoice,v_turnover,seq_cust_id`. The parameter supports specifying tables using wildcards `-objects=cust%,inv%`. If this is not specified, all objects are included, i.e. it's equivalent to `-objects=*`. The parameter supports auto-completion and will show a list of the available objects in the current schema.
-exclude	A comma separated list of object names to be excluded from the generated script. The parameter supports wildcards `-exclude=foo*,bar*`.
-schemas	A comma separated list of schemas. If this is not specified then the current (default) schema is used. If this parameter is provided together with the `-objects` parameter, then the objects for each schema are retrieved. e.g. `-objects=person -schemas=prod,test` will show generate the SQL for the `person` table in both schemas. The parameter supports auto-completion and will show a list of the available schemas.
-types	A comma separated list of object types e.g. `-types=VIEW,TABLE`. The possible values for this parameter are the types listed in the drop down of the "Objects" tab in the DbExplorer. The parameter supports auto-completion and will show a list of the available object types for the current DBMS.
-file	Defines the outputfile into which all statements are written. If neither this nor `-outputFile` is specified, the generated SQL statements are shown in the message area. file. This can't be used if `-outputDir` is specified.
-outputDir	If this parameter is specified, then the DDL statements for each object will be written into a file with the object's name inside the specified directory. This can't be used if `-file` is specified.
-encoding	The encoding to be used when writing the output file.
-includeForeignKeys	By default `WbGenerateScript` will add all foreign key constraints of the processed tables to the end of the generated script. If this parameter is set to `false` no foreign key constraints will be generated. The foreign keys can be generated in a separate step using the `WbGenerateFKScript` command.
-includeTriggers	If this parameter is is present (or set to true), then all triggers (for the selected schemas) will be retrieved as well. The default is `false`.

Parameter	Description
-includeProcedures	If this parameter is present (or set to true), then all procedures and functions (for the selected schemas) will be retrieved as well. The default is `false`.
-includeDrop	If this parameter is present (or set to true) a `DROP` statement will be generated for each object in the list.
-includeTableGrants	This parameter controls the generation of table grants. The default value is `true`.
-useSeparator	If this parameter is present (or set to true), comments will be added that identify the start and end of each object. The default is `false`.

25.2. Show the source of a table - WbTableSource

This command will show the source for a single table. The name of the table is given as an argument to the command:

```
WbTableSource person;
```

It is possible to provide wildcards for the table name:

```
WbTableSource p%;
```

This command is mainly intended to quickly look at a table's source, e.g. when running in console mode. To generate the DDL for multiple tables including depending objects or to save the generated SQL into a file, please use WbGenerateScript.

25.3. Show the source of a view - WbViewSource

This command will show the source for a single view. The name of the view is given as an argument to the command:

```
WbViewSource current_orders;
```

The output of this command can be written to a file when using parameters:

```
WbViewSource -view=current_orders -file=/home/arthur/script/current_orders.sql;
```

The supported parameters are:

- `-view` - the name of the view
- `-file` - the name of the output file
- `-encoding` - the encoding to be used for the file

25.4. Show the source of a stored procedures - WbProcSource

This command will show the source for a single stored procedure (if the current DBMS is supported by SQL Workbench/J). The name of the procedure is given as an argument to the command:

```
WbProcSource the_answer;
```

The output of this command can be written to a file when using parameters:

```
WbProcSource -procedure=the_answer -file=/home/arthur/scripts/answer_proc.sql;
```

The supported parameters are:

- `-procedure` - the name of the procedure (or package)
- `-file` - the name of the output file
- `-encoding` - the encoding to be used for the file

25.5. Show the source of a trigger - WbTriggerSource

This command will show the source for a single trigger The name of the trigger is given as an argument to the command:

```
WbTriggerSource order_trigger;
```

The output of this command can be written to a file when using parameters:

```
WbTriggerSource -trigger=order_trigger -file=/home/arthur/scripts/order_trigger;
```

The supported parameters are:

- `-trigger` - the name of the trigger
- `-file` - the name of the output file
- `-encoding` - the encoding to be used for the file

25.6. Generate DROP statement with dependencies - WbGenerateDrop

The command `WbGenerateDrop` can be used to generate a SQL script for a table that will drop all foreign keys referencing that table, then a `DROP` statement for that table and the statements to re-created the foreign keys referencing that table.

This is useful if you need to re-create a table but don't want to manually delete all referencing foreign keys, especially if the DBMS does not support a cascading DROP.

This is also available in the DbExplorer's context menu as "Generate DROP script".

The command supports the following parameters.

Parameter	Description
-tables	A comma separated list of tables, e.g. `-tables=customer,invoice`. The parameter supports specifying tables using wildcards `-tables=cust%,inv%`.
-includeCreate	Valid values: `true`, `false` By default `WbGenerateDrop` will also add the statements to re-create the foreign keys of the table(s). By specifying `-includeCreate=false` only the statements to drop the foreign key and to drop the table(s) will be created.
-onlyForeignkeys	Valid values: `true`, `false` When using `-onlyForeignkeys=true`, then only `ALTER TABLE` statements will be generate that will drop the foreign keys of all selected tables. No `DROP TABLE` and no statements to re-create the foreign keys will be generated. Setting this parameter to `true` implies `-includeCreate=false`.
-sortByType	Valid values: `true`, `false` Usually the generated SQL script will be ordered by the type of statements. So first all statements to drop constraints will be listed, then the drop table statements, then the statements to re-create all foreign keys. When specifying `-sortByType=false`, then the script will be ordered by table: first all statements for the first table, then all statements for the second table and so on.

Parameter	Description
-outputFile	Defines the file into which all statements are written. If multiple tables are selected using the `-tables` parameter, all statements will be written into this file.
-outputDir	Specifies an output directory into which one script per selected table will be written. The script files are named `drop_XXX.sql`, where XXX is the name of the respective table. If this parameter is used, `-outputFile` is ignored.

If neither `-outputFile` nor `-outputDir` is specified, the output is written to the message panel.

25.7. Generate SQL script for foreign key constraints - WbGenerateFKScript

`WbGenerateFKScript` re-creates the foreign key constraints for the specified tables.

The command supports the following parameters.

Parameter	Description
-tables	The tables for which the foreign key constraints should be generated. This parameter accepts wildcards, e.g. `-tables=*`. To specify tables from multiple schemas, use `-tables=order_mgmt.*, customer_mgmt.*`
-exclude	A comma separated list of tables names to be excluded from the generated script. The parameter supports wildcards `-exclude=foo*,bar*`.
-file	Defines the output file into which all statements are written. If this is not specified, the generated SQL statements are shown in the message area. file.
-encoding	The encoding to be used when writing the output file.

25.8. Generate a table definition from an import file - WbGenerateImpTable

`WbGenerateImpTable` analyzes an import file and generates a suitable `CREATE TABLE` statement to create a table with a structure that matches the import file, so that the file can be imported into that table.

By default this command will only check the first 1000 lines of the input file, assuming that the values are distributed evenly. If the data types for the columns do not reflect the real data, the sample size needs to be increased.

The generated table definition is intended for a quick way to import the data and thus the column definitions are likely to be not completely correct or optimal.

The command supports the following parameters.

Parameter	Description
-file	Specifies the input file to be analyzed. The filename may contain wildcards. When specifying e.g.: `-file=/temp/export/*.txt` one table for each text file found in the directory `/temp/export` will be created.
-lines	Defines the number of lines to analyze. The default is 1000 if this parameter is not specified A value of 0 (zero) or less, results in parsing the entire file.
-type	Possible values: `text`, `ods`, `xls`, `xlsx` The type of the import file. The valid types are the same as for WbImport. The `XML` type is not supported.

Parameter	Description
	To import spreadsheet files, the necessary additional libraries must be installed.
-useVarchar	Possible values: `true`, `false` If enabled, all columns will be created as `varchar`. By default `WbGenerateImpTable` will try to create columns with a data type appropriate for the data found in the import file. If the input data is highly irregular or very skewed, creating all columns as `varchar` will make sure that all rows can be imported without any conversion error.
-delimiter	The delimiter for text files.
-quoteChar	The quote character for text files.
-encoding	The encoding for text files.
-header	Specifies if the input file contains column headers.
-dateFormat	The format for date columns.
-timestampFormat	The format for timestamp columns in the input file.
-decimal	The character used as the decimal separator.
-outputFile	By default the generated SQL statement is shown in the message area. If `-outputFile` is specified, the generated SQL statement will be written to that file.
-sheetNumber	If the input file is a spreadsheet, this parameter defines the sheet number to be analyzed. The first sheet has the number `1`. When specifying `*` for the sheet number a `CREATE TABLE` statement for every sheet will be created.
-table	The table name to use (or create)
-runScript	Possible values: `true`, `false` By default, the `CREATE TABLE` statement is only generated and displayed. If `-runScript=true` is specified, the generated SQL script will be executed immediately. By default, this will display a dialog to confirm the execution the `CREATE TABLE` statement. This confirmation can be suppressed using the parameter `-prompt=false`. In this case the generated statement will be run directly.

26. SQL Workbench/J to generate DML statements



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

26.1. Generate DELETE statements with dependencies - WbGenerateDelete

The command `WbGenerateDelete` can be used to generate a SQL script for one or more rows that should be deleted including all rows from referencing tables (if foreign key constraints are defined)

This is also available through the menu item Data » Generate delete script which will generate the delete for the selected row(s) in the current result.

The command supports the following parameters.

Parameter	Description
-table	Specifies the root table of the hierarchy from which to delete the rows. It is possible to define multiple tables separated by a comma. The table names may contain wildcards. If multiple tables are specified, specifying a column value for the WHERE clause is not supported.
-columnValue	Defines the expression for each PK column to select the rows to be deleted. The value for this parameter is the column name followed by a colon, followed by the value for this column or an expression. e.g.: `-columnValue="person_id:42"` will select rows where person_id has the value 42. You can also specify expressions instead: `-columnValue="id:<0"` or `-columnValue="id:in (1,2,3)"`. For a multi-column primary key, specify the parameter multiple times: `-columnValue="person_id:100" -columnValue="address_id:200"`. This parameter is ignored if more than one table is specified with the `-table` parameter.
-includeCommit	If `true` a `COMMIT` statement will be appended to the generated script.
-outputFile	The file into which the generated statements should be written. If this is omitted, the statements are displayed in the message area.
-appendFile	Valid values: `true`, `false` If `true`, the statements are appended to an existing file. Otherwise any existing file will be overwritten.
-formatSql	Valid values: `true`, `false` If `true`, the generated SQL will be formatted using the SQL formatter. If `false` each statement will be written as a single line.

To generate a script that deletes the person with ID=42 and all rows referencing that person, use the following statement:

```
WbGenerateDelete -table=person -columnValue="id:42";
```

To generate a script that deletes any person with an ID greater than 10 and all rows referencing those rows, use the following statement:

```
WbGenerateDelete -table=person -columnValue="id: > 10";
```

To generate a script that deletes rows from the `film_category` where the primary key consists of the columns `film_id` and `category_id`:

```
WbGenerateDelete -table=person -columnValue="film_id: in (1,2,5)" -columnValue="category_i
```

To generate a script that deletes rows from multiple tables in the correct order:

```
WbGenerateDelete -table=order*,customer*,invoice*;
```

26.2. Generate INSERT statements with dependencies - WbGenerateInsert

The command `WbGenerateInsert` can be used to generate a SQL script that shows the order in which `INSERT` statements should be executed with respect to declared foreign keys.

The command supports the following parameters.

Parameter	**Description**
-tables	Specify the list of tables for which the INSERT statements should be generated.
-retrieveFKParents	If this is true, all parent tables for the tables specified by `-tables` are retrieved before processing the list of tables to sort them in the correct order.
-fullInsert	If this is true the output will be complete `INSERT` statements, rather than just the list of tables in the correct order.
-file	Specifies an output file to which the result is written instead of displaying it.

27. Show information about database objects



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

27.1. Create a report of the database objects - WbSchemaReport

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 https://www.sql-workbench.eu/xstl.html. If you have XSLT stylsheets that you would like to share, please send them to `<support@sql-workbench.eu>`.



To see table and column comments with an Oracle database, you need to enable remarks reporting for the JDBC driver, otherwise the driver will not return comments. To see the "comment" values from SQL Server's extended properties, please setup the property retrieval as described here

The command supports the following parameters:

Parameter	Description
-file	The filename of the output file.
-objects	A (comma separated) list of objects to report. Default is all objects that are "tables" or views. The list of possible objects corresponds to the objects shown in the "Objects" tab of the DbExplorer. If you want to generate the report on tables from different schemas you have to use fully qualified names in the list (e.g. `-tables=shop.orders,accounting.invoices`) You can also specify wildcards in the table name: `-table=CONTRACT_%` will create an XML report for all tables that start with `CONTRACT_`. This parameter supports auto-completion.
-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=public,accounting` would generate a report for all tables in the schemas `public` and `accounting`. If you combine `-schemas` with `-objects`, the list of objects will be applied to every schema unless the object names are supplied **with** a schema: `-schemas=accounting,invoices -objects=o*,customers.c*` will select all objects starting with `O` from the schemas `accounting,invoices` and all objects starting with `C` from the schema `customers`. The possible values for this parameter correspond to the "Schema" dropdown in the DbExplorer. The parameter supports auto-completion and will show a list of available schemas.
-types	A (comma separated) list of "table like" object types to include. By default `TABLE`s and `VIEW`s are included. To include e.g. `SYSTEM VIEW`s and `TEMPORARY TABLE`s, use the following option: `-types='TABLE,VIEW,SYSTEM VIEW,TEMPORARY TABLE'`. If you include type names that contain a space (or e.g. a dash) you have to quote the whole list, not just the single value. The default for this parameter is `TABLE,VIEW`

Parameter	Description
	The values for this parameter correspond to the values shown in the "types" dropdown in the "Objects" tab of the DbExplorer. The parameter supports auto-completion and will show a list of the available object types for the current DBMS. You can include any type shown in the DbExplorer's Objects tab. To e.g. include `domain` and `enum` definitions for PostgreSQL use: `-types=table,view,sequence,domain,enum` This parameter supports auto-completion.
-excludeObjectNames	A (comma separated) list of tables to exclude from reporting. This is only used if -tables is also specified. To create a report on all tables, but exclude those that start with 'DEV', use `-tables=* -excludeTableNames=DEV*`

-objectTypeNames	This parameter can be repeated several times to define the object names per object type to be retrieved. The format of the argument is `-objectTypeNames=typename:namepattern`
	
	The following will select the tables `person` and `address`, all sequences starting with "T" and the view `v_person`. If the list of object names contains a comma, the whole parameter value needs to be quoted:
	
	`-objectTypeNames='table:person,address' -objectTypeNames=sequence:t* -o`
	
	The type names are the same ones that can be used with the `-types` parameter. This can be combined with schema qualified names:
	
	`-objectTypeNames='table:cust.person,accounting.address' -objectTypeName`
	
	This can also be used to restrict the retrieval of stored procedures: `-objectNameTypes=procedure:P*` will include all stored procedures (or functions) that start with a "P". In this case the parameter `-includeProcedures` is ignored.
	
	If this parameter is used at least once, all of the following parameters are ignored: `-types` and `-objects`, `-includeSequences`, `-includeTables` and `-includeViews` are ignored.
	
	The exclusion pattern defined through `-excludeObjectNames` is applied to *all* object types.

-includeTables	Controls the output of table information for the report. The default is `true`. Valid values are `true`, `false`.
-includeSequences	Control the output of sequence information for the report. The default is `false`. Valid values are `true`, `false`. Adding `sequence` to the list of types specified with the `-types` parameter has the same effect.
-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`.
-includeTriggers	This parameter controls if table triggers are added to the output. The default value is `true`.
-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.

Parameter	Description
-writeFullSource	By default the source code for views is written as retrieved from the DBMS into the XML file. This might not be a complete `create view` statement though. When `-writeFullSource=true` is specified SQL Workbench/J will generate a complete `create view` statement, similar to the code that is shown in the DbExplorer. The default is `false`. Valid values are: `true`, `false`.
-styleSheet	Apply a XSLT transformation to the generated XML file.
-xsltOutput	The name of the generated output file when applying the XSLT transformation.
-xsltParameter	A list of parameters (key/value pairs) that should be passed to the XSLT processor. When using e.g. the `wbreport2liquibase.xslt` stylesheet, the value of the `author` attribute can be set using `-xsltParameter="authorName=42"`. This parameter can be provided multiple times for multiple parameters, e.g. when using `wbreport2pg.xslt`: `-xsltParameter="makeLowerCase=42" -xsltParameter="useJdbcTypes=true"`

27.2. Show table structure - WbDescribe

`WbDescribe` shows the definition of the given table.

Alternatively it can be written as `DESC` or `DESCRIBE`.

The command expects the name of a table or view as a parameter. The output of the command will be several result tabs to show the table structure, indexes and triggers (if present). If the "described" object is a view, the message tab will additionally contain the view source (if available).

```
WbDescribe 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
```
WbDescribe otheruser.person;
```

27.3. 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.

You can limit the displayed objects by either specifying a wildcard for the names to be retrieved: `WbList P%` will list all tables or views starting with the letter "P"

The command supports two parameters to specify the tables and objects to be listed. If you want to limit the result by specifying a wildcard for the name **and** the object type, you have to use the parameter switches:

Parameter	Description
-objects	Select the objects to be returned using a wildcard name, e.g. `-objects=P%`
-types	Limit the result to specific object types, e.g. `WbList -objects=V% -types=VIEW` will return all views starting with the letter "V".

27.4. List indexes - WbListIndexes

This command will list all indexes defined on tables available to the current user.

The command supports two parameters to specify the tables and objects to be listed. If you want to limit the result by specifying a wildcard for the name **and** the object type, you have to use the parameter switches:

Parameter	Description
-schema	Show only indexes for the specified schema, e.g. `-schema=some_schema`
-catalog	Show only indexes for the specified catalog e.g. `-catalog=other_db`
-tableName	Show only indexes for the tables specified by the parameter. The parameter value can contain a wildcard, e.g. `-tableName=VP%` lists the indexes for all tables starting with VP
-indexName	Show only indexes with the specified name. The parameter value can contain a wildcard, e.g. `-indexName=PK%` lists only indexes that start with `PK`

If none of the above parameters are used, the provided command line parameter is assumed to be a single table name, e.g.:

```
WbListIndexes public.person;
```

27.5. 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.

You can limit the list by supplying a wildcard search for the name, e.g.:

```
WbListProcs public.p%
```

27.6. List triggers - WbListTriggers

This command will list all stored triggers available to the current user. The output of this command is equivalent to the Database Explorer's Triggers tab (if enabled)

27.7. List catalogs - WbListCat

Lists the available catalogs (or databases). It is the same information that is shown in the DbExplorer's "Database" drop down.

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 with the command `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.

This command ignores the filter defined for catalogs in the connection profile and always returns all databases.

27.8. List schemas - WbListSchemas

Lists the available schemas from the current connection. The output of this command depends on the underlying JDBC driver and DBMS. It is the same information that is shown in the DbExplorer's "Schema" drop down.

This command ignores the filter defined for schemas in the connection profile and always returns all schemas.

27.9. Show dependency information - WbListDependencies

Lists dependency information about database objects. It is the command line version of the "Used by" and "Uses" information in the DbExplorer and DbTree.

The command supports the following parameters:

Parameter	Description
-name	Then name of the database object for which the dependencies should be displayed. This parameter supports wildcards, e.g. `-name=p%` to display dependencies for all objects that start with a `P`. This parameter supports auto-completion.
-schema	Search objects in the specified schema. This parameter supports auto-completion.
-catalog	Search objects in the specified catalog (database). This parameter supports auto-completion.
-objectType	A (comma separated) list of objects to report. Default is all objects that are "tables" or views. The list of possible objects corresponds to the objects shown in the "Objects" tab of the DbExplorer. This parameter supports auto-completion.
-dependency	Define the type of dependency to display. Valid values are `uses` and `using`. This parameter supports auto-completion.

28. Manage macros with SQL Workbench/J command



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

28.1. Define a new macro - WbDefineMacro

Defines a new macro (or overwrites an existing one). This command is primarily intended for the console mode

Parameter	Description
-name	The name of the new macro
-group	The name of the macro group in which the new macro should be stored
-text	The text of the macro
-file	A file from which to read the macro text. If this parameter is supplied, `-text` is ignored
-encoding	The encoding of the input file specified with the `-file` parameter.
-autoExpand	If `true` then the new macro is a macro that is expanded while typing

28.2. Delete a macro - WbDeleteMacro

`WbDeleteMacro` is used to delete a macro, the macro name is passed with the command, e.g:

```
WbDeleteMacro showData
```

28.3. List available macros - WbListMacros

Display the defined macros. This command is primarily intended for the console mode.

29. Manage variables with SQL Workbench/J



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

More details about using variables can be found in the chapter Variable substitution.

29.1. Define a script variable - WbVarDef

This defines an internal variable which is used for variable substitution during SQL execution.

There are two possibilities to define a variable.

The short syntax is: `WbVarDef variable=value`

The long syntax allows to define variables in a different way:

`WbVarDef` supports conditional execution

Parameter	Description
-variable	The name of the variable to be defined.
-value	The value of the variable.
-file	Read the variable definitions from the specified file.
-contentFile	Read the contents of the variable from a the specified file.
-encoding	The encoding of the file specified with `-contentFile`.
-query	Define a SQL query to be used as the source for the variable(s).
-replaceVars	Valid values are: `true`, `false` If the variable content is defined through `-contentFile`, this parameter controls if variable references in the the content should be replaced. The default for this parameter is: `true`.
-cleanupValue	Valid values are: `true`, `false` If this is set to true, the value is "cleaned" up such that newlines (linefeeds) and SQL style comments are removed. The default for this parameter is: `false`.
-removeUndefined	Valid values are: `true`, `false` If this is set to true, any reference to non-existing variables inside the value to be defined through `-value`, `-query` or `-contentFile`) is removed. The default for this parameter is: `false`.
-nullHandling	Define how NULL values returned from a SQL query should be handled. `empty` - if the value from the query is NULL, the variable be set to an empty string. `ignore` - the value is ignored. An existing variable will not be changed. If the variable does not exist, it will not be created. `remove` - if the value from the query is NULL, the variable will be removed.

Parameter	Description
-values	Define a comma separated list of values that are used in the dialog that is shown when prompting for variable values.
-silent	If the option `-silent` is specified, no feedback about the variable definition will be shown.

More details and examples can be found in the chapter: Variable substitution

29.2. Delete a script variable - WbVarDelete

This removes a variable.

The syntax is `WbVarDelete variable`

`variable` is the name of the variable to be removed. The name can contain wildcards. `WbVarDelete *` will remove all defined variables. `WbVarDelete id*` will remove all variables starting with `id`.

29.3. Show defined script variables - WbVarList

This shows all defined variables when invoked without parameters. There are two ways to filter the result:

Supplying a simple wildcard value, e.g. `WbVarList foo*` to display variables where the name starts with `foo`

Using the parameter `-match` to define a regular expression only show variables where the name matches the regular expression, e.g. `WbVarList -match="^(foo|bar).*"` to list all variables where the name starts with `foo` or `bar`.

30. Other SQL Workbench/J specific commands



All SQL Workbench/J specific command can **only** be used from within SQL Workbench/J

30.1. 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.

`WbConfirm` can be called in three different ways:

- Without any parameter, then a default message will be displayed
- With just a message text, e.g. `WbConfirm Do you really want to drop everything?`
- Supplying parameters for the message, the text for the "Yes" choice and the text for the "No" choice using standard SQL Workbench/J parameters:

```
WbConfirm -message="Do you really want to drop everything?" -yesText="OK, go
ahead" -noText="No, please stop"
```

When using `WbConfirm` in console (or interactive batch) mode, the check if the "Yes" choice was typed by the user is done by testing if the "Yes" value starts with the text the user enters (ignoring upper/lowercase differences). So if the "Yes text" is set to `"Continue"`, the user can enter `c`, `co`, `cont` and so on. Because of that, the "No" text should not start with the same letters as the "Yes" text. When using `-yesText=Continue and -noText=Cancel` and the user enters `C`, this would be regarded as a "Yes".

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 unless the `-interactive` parameter was specified.

30.2. Display a message box - WbMessage

The `WbMessages` command pauses the execution of the current script and displays a message and waits until the dialog is closed. Unlike `WbConfirm` the script will always continue once the message dialog is closed.

`WbMessage` can be called in two different ways:

- With just a message text, e.g. `WbMessage Done!`
- Supplying parameters for the message and the dialog title:

```
WbMessage -message="Script finished" -title="SQL Script"
```

This command has no effect in batch or console mode.

30.3. Print a text - WbEcho

The command `WbEcho` can be used to print messages. The following statement:

```
WbEcho The answer is: 42
```

will print the text "The answer is: 42" to the message pane in GUI mode, or to the console in batch or console mode.

If `WbEcho` is used in a script that is executed with `WbInclude` and the `-verbose=false` option, the message will not be displayed. To display a message even if verbose mode is disabled, prefix the text with two exclamation marks: `!!`, e.g.:

```
WbEcho !! Dropping all tables!
```

30.4. Run a stored procedure with OUT parameters - WbCall

If you want to run a stored procedure that has `OUT` parameters, you have to use the `WbCall` command to correctly see the returned value of the parameters.

Consider the following (Oracle) procedure:

```
CREATE OR REPLACE procedure return_answer(answer OUT integer)
IS
BEGIN
  answer := 42;
END;
/
```

To call this procedure you need to supply a placeholder indicating that a parameter is needed.

```
SQL> WbCall return_answer(?);
PARAMETER | VALUE
----------+------
ANSWER    | 42

(1 Row)
Converted procedure call to JDBC syntax: {call return_answer(?)}
Execution time: 0.453s
SQL>
```

Stored procedures with REF CURSORS

If the stored procedure has a REF CURSOR (as an output parameter), `WbCall` will detect this, and retrieve the result of the ref cursors.

Consider the following (Oracle) stored procedure:

```
CREATE PROCEDURE ref_cursor_example(pid number, person_result out sys_refcursor, addr_resu
BEGIN
    OPEN person_result FOR
      SELECT *
      FROM person
      WHERE person_id = pid;

    OPEN addr_result FOR
      SELECT a.*
      FROM address a JOIN person p ON a.address_id = p.address_id
      WHERE p.person_id = pid;
END;
/
```

To call this procedure you use the same syntax as with a regular OUT parameter:

```
WbCall ref_cursor_example(42, ?, ?);
```

SQL Workbench/J will display two result tabs, one for each cursor returned by the procedure. If you use `WbCall ref_cursor_example(?, ?, ?)` you will be prompted to enter a value for the first parameter (because that is an IN parameter).

PostgreSQL functions that return a refcursor

When using ref cursors in PostgreSQL, normally such a function can simply be used inside a SELECT statement, e.g. `SELECT * FROM refcursorfunc();`. Unfortunately the PostgreSQL JDBC driver does not handle this correctly and you will not see the result set returned by the function.

To display the result set returned by such a function, you have to use `WbCall` as well

```
CREATE OR REPLACE FUNCTION refcursorfunc()
  RETURNS refcursor
AS
$$
DECLARE
    mycurs refcursor;
 BEGIN
    OPEN mycurs FOR SELECT * FROM PERSON;
    RETURN mycurs;
 END;
$$ LANGUAGE plpgsql;
/
```

You can call this function using

```
WbCall refcursorfunc();
```

This will then display the result from the SELECT inside the function.

30.5. 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 filenames (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 unless `-displayResult=true` is used.

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 that size will be processed statement by statement. In this case the automatic detection of the alternate delimiter [47] will **not work**. If your scripts exceed the maximum size and you do use the alternate delimiter you will have to use the long version of the command using the `-file` and `-delimiter` parameters.

`WbInclude` supports conditional execution

The command supports the following parameters:

Parameter	Description
-file	The filename of the file to be included.
-continueOnError	Defines the behavior 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 a delimiter to be used that is different from the standard `;` delimiter.
	
	A non-standard delimiter will be required to be on a line of its own. If you specify `-delimiter=/` the following will work:
	
	`select *`
	`from person`
	`/`
	
	but putting the delimiter at the end of a line will **not** work:
	
	`select *`
	`from person/`
	
	If this parameter is not specified, the SQL standard `;` delimiter will be used.

-splitStatements	Controls if the script should be split into separate statements that are sent individually to the database server. If this parameter is `false`, then the entire contents of the script file will be sent to the database as single statement. This useful for database systems that do not require a statement terminator. If this parameter is set to `false` the alternate delimiter defined in the connection profile, any delimiter defined with `WbDelimiter` and the value of the `-delimiter` parameter are ignored. The default value for this parameter is `true`.
-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 the statement `WbFeedback off` to the called script.
-displayResult	By default any result set that is returned e.g. by a `SELECT` statement in the script will not be displayed. By specfying `-displayResult=true` those results will be displayed.
-printStatements	If true, every SQL statement will be printed before execution. This is mainly intended for console usage, but works in the GUI as well.
-showTiming	If true, display the execution time of every SQL statement and the overall execution time of the script.
-useSavepoint	Control if each statement from the file should be guarded with a savepoint when executing the script. Setting this to `true` will make execution of the script more robust, but also slows down the processing of the SQL statements.
-ignoreDropErrors	Controls if errors resulting from DROP statements should be treated as an error or as a warning.
-searchFor -replaceWith -ignoreCase	Defines search and replace parameters to change the SQL statements before they are sent to the database. This can e.g. be used to replace the schema name in DDL script that uses fully qualified table names.

Parameter	Description
-useRegex	The replacement is done without checking the syntax of the statements. If the search value is contained in a string literal or a SQL comment, it is also replaced.

30.5.1. Examples

Execute `my_script.sql`

```
@my_script.sql;
```

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

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

Execute the script `create_tables.sql` and change all occurances of `oldschema` to `new_schema`

```
WbInclude -file=create_tables.sql -searchFor="oldschema." -replaceWith="new_schema."
```

Execute a large script that uses a non-standard statement delimiter:

```
WbInclude -file=insert_10million_rows.sql -delimiter='/';
```

30.6. Conditional execution

The following SQL Workbench/J commands support conditional execution based on variables:

- WbInclude
- WbCopy
- WbImport
- WbExport
- WbSysExec
- WbRunResult
- WbVarDef
- WbConfirm

Conditional execution is controlled using the following parameters:

Parameter	Description
-ifDefined	The command is only executed if the variable with the specified name is defined. `-ifDefined=some_var`
-ifNotDefined	The command is only executed if the variable with the specified name is not defined. `-ifNotDefined=some_var`
-ifEquals	The command is only executed if the specified variable has the specified value `-ifEquals='some_var=42'`
-ifNotEquals	The command is only executed if the specified variable does not have the specified value `-ifNotEquals='some_var=42'`
-ifEmpty	The command is only executed if the specified variable is defined but has an empty value `-ifEmpty=some_var`. This is essentially a shorthand for `-ifEquals="some_var=''"`

Parameter	Description
-ifNotEmpty	The command is only executed if the specified variable is defined and has a non empty value `-ifNotEmpty=some_var`. This is essentially a shorthand for `-ifNotEquals="some_var=''"`
-ifFileExists	The command is only executed if the specified file exists (on the computer running) `-ifFileExists=/data/export/text`.
-ifNotFileExists	The command is only executed if the specified file does not exist (on the computer running) `-ifFileExists=/data/export/text`.
-ifTableExists	The command is only executed if the specified table exists `-ifTableExists=person`.
-ifNotTableExists	The command is only executed if the specified table does not exist. `-ifNotTableExists=person`.
-isDBMS	The command is only executed if the current connection is against the specified DBMS. The value supplied is the DBID that is used internally by SQL Workbench/J, e.g. `-isDBMS=postgresql` This parameter supports auto-completion.
-isNotDBMS	The command is only executed if the current connection is not against the specified DBMS. The value supplied is the DBID that is used internally by SQL Workbench/J e.g. `-isNotDBMS=postgresql` This parameter supports auto-completion.

30.7. Run a result set as a SQL script - WbRunResult

`WbRunResult` can be used to execute the result of a query as a SQL script. Each row in the result is assumed to be a single statement. This is e.g. useful to generate scripts dynamically without having to use dynamic SQL in a procedural language like PL/pgSQL or PL/SQL.

From the result of the generating script, only the first column will be used. If the statement generated more columns, the additional columns are ignored. Each generated statement should end with a `;` character.

`WbRunResult` supports conditional execution

The command supports the following parameters:

Parameter	Description
-query	The query that generates the script. If this parameter is not provided, the SQL query following `WbRunResult` will be used as the input. This is similar to the way as WbExport works.
-dryRun	If true, the generated script won't be executed, only the result will be displayed. This can be used to inspect the generated statements for actually running them. This is useful when providing the query through the `-query` parameter, not as a separate statement that could be run on its own.
-continueOnError	If true, the script execution does not stop on the first error. This is the same as the `-continueOnError` parameter for WbInclude.
-verbose	Controls the feedback during script execution. This is the same as the `-verbose` parameter for WbInclude.
-ignoreDropErrors	Don't stop executing the script if a `DROP` statement results in an error. This is the same as the `-ignoreDropErrors` parameter for WbInclude.

Parameter	Description
-printStatements	Print each statement while the script is processed. This is the same as the `-printStatements` parameter for WbInclude.
-showTiming	Print execution statistics for each statement. This is the same as the `-showTiming` parameter for WbInclude.
-showProgress	Report the current statement nummer in the statusbar.

The following script will drop all tables that end with `_backup`:

```
WbRunResult -showProgress -showTiming;
SELECT concat('DROP TABLE ', table_schema, '.', table_name, ';') as ddl
FROM information_schema.tables
WHERE table_name LIKE '%\_backup' ESCAPE '\';
```

30.8. Extract and run SQL from a Liquibase ChangeLog - WbRunLB

If you manage your stored procedures in Liquibase ChangeLogs, you can use this command to run the necessary SQL directly from the XML file, without the need to copy and paste it into SQL Workbench/J. This is useful when testing and developing stored procedures that are managed by a Liquibase changeLog.



This is NOT a replacement for Liquibase.

`WbRunLB` will only extract SQL statements stored in `<sql>` or `<createProcedure>` tags or scripts referenced through the `<sqlFile>` tag.

It will not convert any of the Liquibase tags to "real" SQL.

`WbRunLB` will **NOT** update the Liquibase log table (`DATABASECHANGELOG`) nor will it check if the specified changeSet(s) have already been applied to the database.

It is merely a convenient way to extract and run SQL statements stored in a Liquibase XML file!

The attribute `splitStatements` for the `sql` tag is evaluated. The delimiter used to split the statements follows the usual SQL Workbench/J rules (including the use of the alternate delimiter).

`WbRunLB` supports the following parameters:

Parameter	Description
-file	The filename of the Liquibase changeLog (XML) file. The `<include>` tag is NOT supported! SQL statements stored in files that are referenced using Liquibase's `include` tag will not be processed.
-changeSet	A list of changeSet ids to be run. If this is omitted, then the SQL from all changesets (containing supported tags) are executed. The value specified can include the value for the author and the id, `-changeSet="Arthur::42"` selects the changeSet where `author="Arthur"` and `id="42"`. This parameter can be repeated in order to select multiple changesets: `-changeSet="Arthur::42" -changeSet="Arthur::43"`. You can specify wildcards before or after the double colon: `-changeSet="*::42"` will select all changesets with the id=42. `-changeSet="Author::*"` will select all changesets from "Arthur" If the parameter value does not contain the double colon it is assumed to be an ID only: `-changeSet="42"` is the same as `-changeSet="*::42"` If this parameter is omitted, all changesets are executed.

Parameter	Description
	This parameter supports auto-completion if the `-file` argument is specified.
-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`
-encoding	Specify the encoding of the input file. If no encoding is specified, UTF-8 is used.

30.9. Handling tables or updateable views without primary keys

30.9.1. 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.

30.9.2. List defined primary key columns - WbListPKDef

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

30.9.3. 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.

30.9.4. 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.

30.10. Change the default fetch size - WbFetchSize

The default fetch size for a connection can be defined in the connection profile. Using the command `WbFetchSize` you can change the fetch size without changing the connection profile.

The following script changes the default fetch size to 2500 rows and then runs a `WbExport` command.

```
WbFetchSize 2500;
WbExport -sourceTable=person -type=text -file=/temp/person.txt;
```

`WbFetchSize` will not change the current connection profile.

30.11. Run statements as a single batch - WbStartBatch, WbEndBatch

To send several SQL Statements as a single "batch" to the database server, the two commands WbStartBatch and WbEndBatch can be used. All statements between these two will be sent as a single statement (using `executeBatch()`) to the server.

Note that not all JDBC drivers support batched statements, and the flexibility what kind of statements can be batched varies between the drivers as well. Most drivers will not accept different types of statements e.g. mixing DELETE and INSERT in the same batch.

To send a group of statements as a single batch, simply use the command `WbStartBatch` to mark the beginning and `WbEndBatch` to mark the end. You have to run all statements together either by using "Execute all" or by selecting all statements (including WbStartBatch and WbEndBatch) and then using "Execute selected". The following example sends all INSERT statements as a single batch to the database server:

```
WbStartBatch;
INSERT INTO person (id, firstname, lastname) VALUES (1, 'Arthur', 'Dent');
INSERT INTO person (id, firstname, lastname) VALUES (2, 'Ford', 'Prefect');
INSERT INTO person (id, firstname, lastname) VALUES (3, 'Zaphod', 'Beeblebrox');
INSERT INTO person (id, firstname, lastname) VALUES (4, 'Tricia', 'McMillian');
WbEndBatch;
COMMIT;
```

30.12. 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 output file 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.

`WbSelectBlob` is intended for an ad-hoc retrieval of a single LOB column. If you need to extract the contents of several LOB rows and columns it is recommended to use the WbExport command.

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

30.13. 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.

`WbFeedback traceOn` can be used to enable printing of every executed statement to the screen. The SQL statement printed will be the one after variable substitution and macro expansion. `WbFeedback traceOff` will turn tracing of statements off.

30.14. Change the current schema - WbSetSchema

The `WbSetSchema` command will use the JDBC driver's API to change the current schema.

Whether this command works and actually changes the current schema depends on the JDBC driver and the underlying DBMS. The driver is free to ignore the call to `setSchema()` API call.

30.15. Change the current database/catalog - WbSwitchDB

The `WbSwitchDB` can be used to switch the current catalog (or database) if the DBMS supports this. The details on what is done in the background depend on the DBMS.

- For Postgres this is similar to the `psql`'s `\connect` command and will use the current username and password to create a new physical connection to the specified database.
- For Oracle this is used to switch to a different PDB (pluggable database). This will be done by running `ALTER SESSION SET CONTAINER = ...`.
- For SQL Server, MySQL and MariaDB this will run a `USE ...` command to switch the database.

30.16. Setting connection properties - SET

The `SET` command is passed on directly to the driver, except for the parameters described in this chapter because they have an equivalent JDBC call which will be executed instead.

Oracle does not have a SQL set command. The SET command that is available in SQL*Plus is a specific SQL*Plus command and will not work with other client software. Most of the SQL*Plus SET commands only make sense within SQL*Plus (e.g. formatting of the results). To be able to run SQL scripts that are intended for Oracle SQL*PLus, any error reported from the SET command when running against an Oracle database will silently be ignored and only logged as a warning.

30.16.1. FEEDBACK

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

30.16.2. 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.

30.16.3. MAXROWS

Limits the number of rows returned by the next statement. The behaviour of this command is a bit different between the console mode and the GUI mode. In console mode, the maxrows stay in effect until you explicitely change it back using `SET maxrows` again.

In GUI mode, the maxrows setting is only in effect for the script currently being executed and will only temporarily overwrite any value entered in the "Max. Rows" field.

30.17. Changing Oracle session behavior - SET

The following options for the SET command are only available when being connected to an Oracle database.

30.17.1. SERVEROUTPUT

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

30.17.2. AUTOTRACE

This enables or disables the "autotrace" feature similar to the one in SQL*Plus. The syntax is equivalent to the SQL*Plus command and supports the following options:

Option	Description
ON	Turns on autotrace mode. After running a statement, the statement result (if it is a query), the statistics and the execution plan for that statement are displayed as separate result tabs.
OFF	Turns off the autotrace mode.
TRACEONLY	Like `ON`, but does not display the result of a query.
REALPLAN	This is an extension to the SQL*Plus `EXPLAIN` mode. Using `EXPLAIN`, SQL Workbench/J will simply run an `EXPLAIN PLAN` for the statement (and the statement will not be executed) - this is the same behavior as SQL*Plus' `EXPLAIN` mode.

Option	Description
	Using `REALPLAN`, SQL Workbench/J will run the statement and then retrieve the execution plan that was generated while running the statement. This might yield a different result than regular `EXPLAIN` mode. The actual plan also contains more details about estimated and actual row counts. This plan is retrieved using `dbms_xplan.display_cursor()`. If `REALPLAN` is used, the actual SQL statement sent to the server will be changed to include the `GATHER_PLAN_STATISTICS` hint.

The information shown in autotrace mode can be controlled with two options after the `ON` or `TRACEONLY` parameter. `STATISTICS` will fetch the statistics about the execution and `EXPLAIN` which will display the execution plan for the statement. If not additional parameter is specified, `EXPLAIN STATISTICS` is used.

If statistics are requested, query results will be fetched from the database server but they will not be displayed.

Unlike SQL*Plus, the keywords (`AUTOTRACE, STATISTICS, EXPLAIN`) cannot be abbreviated!

For more information about the prerequisites for the autotrace mode, see the description of DBMS specific features.

30.18. Changing read only mode - WbMode

In the connection profile two options can be specified to define the behavior when running commands that might change or update the database: a "read only" mode that ignores such commands and a "confirm all" mode, where you need to confirm any statement that might change the database.

These states can temporarily be changed without changing the profile using the `WbMode` command.



This changes the mode for all editor tabs, not only for the one where you run the command.

Parameters for the `WbMode` command are:

Parameter	Description
reset	Resets the flags to the profile's definition
normal	Makes all changes possible (turns off read only and confirmations)
confirm	Enables confirmation for all updating commands
readonly	Turns on the read only mode

The following example will turn on read only mode for the current connection, so that any subsequent statement that updates the database will be ignored

```
WbMode readonly;
```

To change the current connection back to the settings from the profile use:

```
WbMode reset;
```

30.19. Count rows for all tables - WbRowCount

This command retrieves the row counts for several tables at once. If called without parameters the row counts for all tables accessible to the current user are counted.

The command supports the following parameters to specify the tables (or views) to be counted.

Parameter	Description
-objects	Show only the row counts for the tables (or views) specified by the parameter. The parameter value can contain wildcards, e.g. `-objects=PR%,ORD%` will count the rows for all tables with names that either start with `PR` or `ORD` To count all tables from a schema (or catalog), the schema can be part of the parameter, e.g. `-objects=some_schema.%` will include all tables from the schema `some_schema`. The parameter supports auto-completion and will show a list of available tables.
-types	Define the types of objects which should be selected. By default only tables are considered. If you also want to count the rows for views, use `-types=table,view` The parameter supports auto-completion and will show a list of available object types.
-orderBy	Defines how the resulting table should be sorted. By default it will be sorted alphabetically by table name. The `-orderBy` parameter specifies the columns to sort the result by. By default, sorting is done ascending, if you want a descending sort, append `:desc` to the column name, e.g.: `-orderBy="rowcount:desc"`. So sort by multiple columns separate the column names with a comma: `-orderBy="rowcount:desc,name:desc"` or `-orderBy="rowcount,name:desc"`
-excludeColumns	Possible values: `catalog`, `schema`, `type` By default `WbRowCount` will display the same columns as the table list in the DbExplorer. If not all columns are needed or wanted, this parameter can be used to exclude certain columns. You can specify a comma separated list of columns to be excluded, e.g. `-excludeColumns=type,catalog`. The columns `ROWCOUNT` and `NAME` can not be excluded. The name `database` can be used instead of `catalog`.

If none of the above parameters are used, `WbRowCount` assumes that a list ot table names was specified. `WbRowCount person,address,orders`is equivalent to `WbRowCount -objects=person,address,orders`. When called without any parameters the row counts for all tables accessible to the current user will be displayed.

Unlike the Count rows item in the DbExplorer, `WbRowCount` displays the result for all tables once it is finished. It does not incrementally update the output.

30.20. Change the connection for a script - WbConnect

With the `WbConnect` command, the connection for the currently running script can be changed.

When this command is run in GUI mode, the connection is **only** changed for the remainder of the script execution. Therefor at least one other statement should be executed together with the `WbConnect` command. Either by running the complete script of the editor or selecting the `WbConnect` command together with other statements. Once the script has finished, the connection is closed and the "global" connection (selected in the connect dialog) is active again. This also applies to scripts that are run in batch mode or scripts that are started from within the console using `WbInclude`.

When this command is entered directly in the command line of the console mode, the current connection is closed and the new connection is kept open until the application ends, or a new connection is established using `WbConnect` on the command line again.

There are three different ways to specify a connection:

30.20.1. By specifying a profile

Parameter	Description
-profile	Specifies the profile name to connect to. This parameter is ignored if either `-connection` or the detailed connection informations are supplied individually.
-profileGroup	Specifies the group in which the profile is stored. This is only required if the profile name is not unique

30.20.2. By specifying a simple connection string

Parameter	Description
-connection	Allows to specify a full connection definition as a single parameter (and thus does not require a pre-defined connection profile). The connection is specified with a comma separated list of key value pairs: • `username` - the username for the connection • `password` - the password for the connection • `url` - the JDBC URL • `driver` - the class name for the JDBC driver. If this is not specified, SQL Workbench/J will try to determine the driver from the JDBC URL • `driverJar` - the full path to the JDBC driver. This not required if a driver for the specified class is already configured e.g.: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb"` If an appropriate driver is already configured the driver's classname or the JAR file don't have to be specified. If an appropriate driver is not configured, the driver's jar file must be specified: `"username=foo,password=bar,url=jdbc:postgresql://localhost/mydb,driverjar=/etc/drivers/postgresql.jar"` SQL Workbench/J will try to detect the driver's classname automatically (based on the JDBC URL). If this parameter is specified, `-profile` is ignored. The individual parameters controlling the connection *behavior* can be used together with `-connection`, e.g. `-autocommit` or `-fetchSize`

30.20.3. By specifying all connection attributes

Parameter	Description
-url	The JDBC connection URL
-username	Specify the username for the DBMS

Parameter	Description
-password	Specify the password for the user If this parameter is not specified (but `-url` and `-username`) then you will be prompted to enter the password. To supply an empty password use `-password=` in the command line when starting SQL Workbench/J
-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.
-checkUncommitted	If this parameter is set to true, SQL Workbench/J will try to detect uncommitted changes in the current transaction when the main window (or an editor panel) is closed. If the DBMS does not support this, this argument is ignored. It also has no effect when running in batch or console mode.
-trimCharData	Turns on right-trimming of values retrieved from `CHAR` columns. See the description of the profile properties for details.
-removeComments	This parameter corresponds to the Remove comments setting of the connection profile.
-fetchSize	This parameter corresponds to the Fetch size setting of the connection profile.
-ignoreDropError	This parameter corresponds to the Ignore DROP errors setting of the connection profile.
-altDelimiter	This parameter corresponds to the Alternate delimiter setting of the connection profile.

If none of the parameters is supplied when running the command, it is assumed that any value after `WbConnect` is the name of a connection profile, e.g.:

```
WbConnect production
```

will connect using the profile name `production`, and is equivalent to

```
WbConnect -profile=production
```

30.21. Show the history of SQL statements - WbHistory

This command is primarily intended for console mode to show the statements that have been executed. In console mode the number of any of the listed statements can be entered to re-execute that statement from the history directly.

30.22. 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.

Parameter	Description
-xsltParameter	A list of parameters (key/value pairs) that should be passed to the XSLT processor. When using e.g. the `wbreport2liquibase.xslt` stylesheet, the value of the `author` attribute can be set using `-xsltParameter="authorName=42"`. This parameter can be provided multiple times for multiple parameters, e.g. when using `wbreport2pg.xslt`: `-xsltParameter="makeLowerCase=42" -xsltParameter="useJdbcTypes=true"`

30.23. Running operating system commands - WbSysExec

To run an operating system command use `WbSysExec` followed by a valid command for your operating system.

To run the program `ls` the following call can be used:

```
WbSysExec ls
```

To run Windows® commands that are internal to `cmd.exe` such as `DIR`, you must call `cmd.exe` with the `/c` switch to make sure cmd.exe is terminated:

```
WbSysExec cmd /c dir /n
```

If you need to specify a working directory for the program, or want to specify the command line arguments individually, a second format is available using the standard SQL Workbench/J parameter handling:

`WbSysExec` supports conditional execution

Parameter	Description
-program	The name of the executable program
-argument	One command line argument for the program. This parameter can be repeated multiple times.
-dir	The working directory to be used when calling the external program
-env	A definition for an environment variable for the process running the program. This parameter can be repeated multiple times.

To run an internal Windows® command using the second format, use the following syntax:

```
WbSysExec -program='cmd.exe' -argument='/c' -argument='dir /n' -dir='c:\temp\'
```

To define multiple environment variables for the process, repeat the `-env` parameter for each variable e.g. `-env="var_one=value_one" -env="var_two=value_two"`

30.24. Opening a file with the default application - WbSysOpen

`WbSyOpen` can be used to open a file with the default application of the operating system.

```
WbExport -file=c:/temp/person.txt -sourceTable=person -type=text -header=true;
WbSysOpen c:/temp/person.txt;
```



Due to limitations of the Java console mode, neither `WbSysExec` nor `WbSysOpen` can be used to run a text editor (vi, vim) in console mode (and putting SQL Workbench/J into the background).

30.25. Change an internal configuration parameter - WbSetConfig

Not all configuration parameters are available through the Options Dialog and have to be changed manually in the file workbench.settings. Editing the file requires to close the application.

When using `WbSetConfig`, configuration properties can be changed permanently without restarting SQL Workbench/J.

`WbSetProp` is an alias for `WbSetConfig` with the difference that the change will **not** be saved permanently. Any property set through `WbSetProp` is lost when the SQL Workbench/J is closed.

Any value that is changed through this command will be saved automatically in `workbench.settings` when the application is closed.

If you want to e.g. disable the use of Savepoints in the SQL statements entered interactively, the following command will turn this off for PostgreSQL:

```
WbSetConfig workbench.db.postgresql.sql.usesavepoint=false
```

For a list of configuration properties that can be changed, please refer to Advanced configuration options

If you supply only the property key, the current value will be displayed. If no argument is supplied for `WbSetConfig` all properties are displayed. You can also supply a partial property key. `WbSetConfig workbench.db.postgresql` will list all PostgreSQL related properties. You can directly edit the properties in the result set.

The value `[dbid]` inside the property name will get replaced with the current DBID to change database specific settings.

The following command changes the property named `workbench.db.postgresql.ddlneedscommit` if the current connection is against a PostgreSQL database:

```
WbSetConfig workbench.db.[dbid].ddlneedscommit=true
```

31. DataPumper

31.1. 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.

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.

31.2. 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 drop down for the target table includes an entry labeled "(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.

31.3. 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.

31.3.1. 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).

31.3.2. 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.

31.3.3. 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

31.3.4. 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

31.3.5. 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.

31.3.6. Batch execution

If the JDBC driver supports batch updates, you can enable the use of batch updates with this check box. The check box 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.

31.3.7. 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 successful 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.

31.4. Advanced copy tasks

31.4.1. 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.

31.4.2. 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.

To customize the mapping of generic JDBC datatypes to DBMS specific datatypes, please refer to Customizing data type mapping

31.4.3. 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

32. 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, SQLWorkbench.exe or SQLWorkbench64.exe)

At the top of the window, the current schema and/or catalog can be selected. Whether both drop downs are available depends on the current DBMS. For Microsoft SQL Server, both the schema and the database can be changed. The labels next to the drop down are retrieved from the JDBC driver and should reflect the terms used for the current DBMS (Schema for PostgreSQL and Oracle, Owner and Database for SQL Server, Database for MySQL).

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 drop down. 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.

Synonyms are displayed if the current DBMS supports them. 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. If a DbExplorer is already open (either a window or a tab), the existing one is made visible (or active), when using this menu item.

You can open any number of additional DbExplorer tabs or windows using Tools » New DbExplorer panel or Tools » New DbExplorer window

32.1. 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.

When using this function, the customization for data types is **not** applied to the generated `SELECT` statement.

Count rows

This will count the rows for each selected table object. The rowcounts will be opened in a new window. This is the same functionality as the `WbRowCount` command.

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

When using this function, the customization for data types **will be** applied to the generated `SELECT` statement.

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 empty UPDATE

This creates an empty `UPDATE` 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. This feature is intended for programmers who want to put a SELECT statement into their code.

If you want to generate a SELECT statement to actually retrieve data from within the editor, please use the Put SELECT into option.

When using this function, the customization for data types is **not** applied to the generated `SELECT` statement.

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 status bar 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 the order: `SEQUENCES`, `TABLES`, `VIEWS`, `SYNONYMS`.

The same script can also be generated using the WbGenerateScript command.

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 WbSchemaReport 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.

If the current DBMS does not support a cascading drop, you can order the tables so that foreign keys are detected and the tables are dropped in the right order by clicking on the Check foreign keys button.

If the checkbox "Add missing tables" is selected, any table that should be dropped before any of the selected tables (because of foreign key constraints) will be added to the list of tables to be dropped.

Generate DROP script

This creates a script that first removes all incoming foreign keys to the selected tables, the necessary `DROP` statements and the statements to re-create the foreign keys.

For more details, please refer to the description of the WbGenerateDrop statement.

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 transaction 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.

If the check box "Add missing tables" is selected, any table that should be deleted before any of the selected tables (because of foreign key constraints) will be added to the list of tables.

ALTER script

After you have changed the name of a table in the list of objects, you can generate and run a SQL script that will apply that change to the database.

For details please refer to the section Changing table definitions

32.2. 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 not for a view, those tabs will still be displayed, but will not show any information.

32.3. Modifying the definition of database objects



Applying changes to the definition of a table (or other database objects) is only possible if the necessary `ALTER` statements have been configured. For most of the major DBMS these statements are already built into SQL Workbench/J.

If your changes are rejected (e.g. while changing a table name or the datatype of a column), please make sure that you have enabled the option Allow table altering. If that option is enabled and your DBMS *does* support the change you were trying to do, please send a mail with the necessary information to the support email address.

32.3.1. Changing the table definition

You can edit the definition of the columns, add new columns or delete existing columns directly in the list of columns. To apply the changes, click on the Apply DDL button.

32.3.2. Renaming objects

You can change the name of a table (or other objects if the DBMS supports that) directly in the object list. For DBMS that support it, you can also edit the remarks column of the table to change the documentation.

Once you have changed a name (or several) the menu item "ALTER Script" in the context menu of the object list will be enabled. Additionally a button Apply DDL will appear in the status bar of the object list. Both will bring up a window with the necessary SQL statements to apply your changes. You can save the generated script to a file or run the statements directly from that window.

32.4. 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 row count, click on the Settings button and deselect the check box `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.

32.5. Changing the display order of table columns

You can re-arrange the display order of the columns in the data tab using drag & drop. If you want to apply that column order whenever you display the table data, you can save the column order by right-clicking in the table header and then using the menu item Save column order. If the column order has not been changed, the menu item is disabled.

The column order will be stored using the fully qualified table name and the current connection's JDBC URL as the lookup key.

To reset the column order use the menu item Reset column order from the popup menu. This will revert the column order to the order in which the columns appear in the source table. The saved order will be deleted as well.

32.6. Customize data retrieval

When displaying the data for a table, SQL Workbench/J generates a `SELECT` statement that will retrieve all rows and columns from the database. In some cases the data for certain data types cannot be displayed correctly as the JDBC drivers might not implement a proper "`toString()`" method that converts the data into a readable format.

You can customize the SELECT statement that is generated by SQL Workbench/J when retrieving table data in the DbExplorer in the configuration file `workbench.settings`. For each DBMS you can define an expression for specific data types that are used when building the `SELECT` statement.

To configure this, you need to add one line per data type and DBMS to the file `workbench.settings`:

```
workbench.db.[dbid].selectexpression.[type]=expression(${column})
```

When building the `SELECT` statement, the placeholder `${column}` will be replaced with the actual column name. `[dbid]` is the DBID of the DBMS for which the replacement should be done.

The whole key (the part to the left of the equal sign) must be in lowercase.

`[type]` is the datatype of the column without any brackets or parameters: `varchar` instead of `varchar(10)`, or `number` instead of `number(10,2)`

To convert e.g. the `geometry` datatype of Postgres to a readable format, one would use the following expression `astext(transform(geo_column,4326))`.

To tell the DbExplorer to replace the retrieval of columns of type `geometry` in PostgreSQL with the above expression, the following line in `workbench.settings` is necessary:

```
workbench.db.postgres.selectexpression.geometry=astext(transform(${column},4326))
```

For e.g. the table `geo_table (id integer, geo_col geometry)` SQL Workbench/J will generate the following `SELECT` statement:

```
SELECT id, astext(transform(geo_col,4326))
FROM geo_table
```

to retrieve the data of that table.

Note that the data of columns that have been "converted" through this mechanism, might not be updateable any more. If you intend to edit such a column you will have to provide a column alias in order for SQL Workbench/J to generate a correct UPDATE or INSERT statement.

Another example is to replace the retrieval of `XML` columns. To configure the DbExplorer to convert Oracle's `XMLTYPE` a string, the following line in `workbench.settings` is necessary:

```
workbench.db.oracle.selectexpression.xmltype=extract(${column}, '/').getClobVal()
```

To convert DB2's `XML` type to a string, the following configuration can be used:

```
workbench.db.db2.selectexpression.xml=xmlserialize(${column} AS CLOB)
```

The column name (as displayed in the result set) will usually be generated by the DBMS and will most probably not contain the real column name. In order to see the real column name you can supply a column alias in the configuration.

```
workbench.db.oracle.selectexpression.xmltype=extract(${column}, '/').getClobVal() AS ${col
```

In order for SQL Workbench/J to parse the SQL statement correctly, the `AS` keyword **must** be used.

You can check the generated SELECT statement by using the Put SELECT into feature. The statement that is generated and put into the editor, is the same as the one used for the data retrieval.

The defined expression will also be used for the Search table data feature, when using the server side search. If you want to search inside the data that is returned by the defined expression you have to make sure that you DBMS supports the result of that expression as part of a `LIKE` expression. E.g. for the above Oracle example, SQL Workbench/J will generate the following WHERE condition:

```
WHERE to_clob(my_clob_col) LIKE '%searchvalue%'
```

32.7. Customizing the generation of the table source

SQL Workbench/J re-generates the source of a table based on the information about the table's metadata returned by the driver. In some cases the driver might not return the correct information, or not all the information that is necessary to build the correct syntax for the DBMS. In those cases, a SQL query can be configured that can use the built-in functionality of the DBMS to return a table's definition.

This DBMS specific retrieval of the table source is defined by three properties in `workbench.settings`. Please refer to Customize table source retrieval for details.

32.8. 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.eu>`.

32.9. 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 source 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.

32.10. Search table data

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 results displayed here are not editable. If you want to modify the results after a search, you have to use the WbGrepData command

Two different implementations of the search are available: server side and client side.

32.10.1. Server side search

To server side search is enabled by selecting the check box labeled "`Server side search`".

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, labeled `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-insensitive 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.

The `SELECT` statement that is built to display the table's data will list all columns from the table. If the table contains BLOB columns this might lead to a substantial memory consumption. To avoid loading too many data into memory, you can check the option "Do not retrieve LOB columns". In that case columns of type CLOB or BLOB will not be retrieved.

SQL Workbench/J is building a `SELECT` that "searches" for data using a `LIKE` expression. Only columns of type `CHAR` and `VARCHAR` are included in the LIKE search, because that is what most DBMS support. If the DBMS you are using supports `LIKE` expressions for other datatypes as well, you can configure this datatypes to be included in the search feature of the DbExplorer.

32.10.2. Client side search

To client side search is enabled by un-checking the check box labeled "`Server side search`".

The client side search retrieves every row from the server, compares the retrieved values for each row and keeps the rows where at least one column matches the defined search criteria.

As opposed to the server side search, this means that **every** row from the selected table(s) will be sent from the database server to the application. For large tables were only a small number of the rows will match the search value this can increase the processing time substantially.

As the searching is done on the client side, this means that it can also "search" data types that cannot be using for a `LIKE` query such as `CLOB, DATE, INTEGER`.

The search criteria is defined similar to the definition of a filter for a result set. For every column, its value will be converted to a character representation. The resulting string value will then be compared according to the defined comparator and the entered search value. If at least one column's value matches, the row will be displayed. The comparison is always done in a case-insensitively. The contents of BLOB columns will never be searched.

The character representation that is used is based on the default formatting options from the Options Window. This means that e.g. a DATE column will be compared according to the standard formatting options before the comparison is done.

The client side search is also available through the WbGrepData command

33. Working with the Database Object tree

The Database Object Tree offers a similar functionality as the Database Object Explorer but can be displayed alongside the SQL editor tabs. The DB Object tree offers a subset of the features the DbExplorer offers. The Database Object Tree can be displayed using Tools » Show DbTree.



The DB Object tree always uses a separate connection regardless of the configuration of the current connection profile.

The elements of each part of the tree are only loaded when the node is expanded for the first time.

33.1. Filtering the elements in the tree

The quick filter above the object tree can be used to quickly search for objects with a specific name. The filtering will only be done on **already loaded** elements of the tree.

The behavior of the quick filter is the same as the filter in the DbExplorer

33.2. Drag and drop support

33.2.1. Dropping elements into the SQL editor

In general, dropping an element into the editor, will insert the element's name into the editor. There are however two exceptions to this rule:

- dropping the "Columns" node of the tree will insert a comma separated list of all columns
- dropping a table into a position where no current SQL statement is located, a `SELECT` statement for the table will be inserted
- in all other cases, the name of the dropped element will be inserted (e.g. the name of the table, column or index)

33.2.2. Displaying a table's data

To display the data of a table, drag the table node from the Database Object Tree to the result panel of the current SQL editor. SQL Workbench/J will then generate an appropriate `SELECT` statement for the table and execute it immediately.

33.3. Finding elements in the tree

When the object tree is displayed, the context menu of the editor contains a new item Find in object tree. This will try to find and select the identifier at the cursor location in the Object Tree.

If the schema (or catalog) that contains the object has not yet been loaded, it will be loaded in order to be able to display the current identifier.

33.4. Features available through the context menu

The Database Object Tree offers some of the features that are available in the DbExplorer

Export data

This exports the data from the selected table(s). This is identical to the function in the DbExplorer.

Find in object tree

This menu choice is only available when right clicking on a table in the "References" or "Referenced by" nodes. It will then find and select the referenced (or referencing) table in the DbTree.

Count rows

This will count the rows for each selected table object. The row counts will be shown in parentheses next to the table name. This is the same functionality as the `WbRowCount` command.

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

The generated SELECT statement is always inserted into the editor's text at the current cursor position.

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.

The generated SELECT statement is always inserted into the editor's text at the current cursor position.

Create empty UPDATE

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

The generated SELECT statement is always inserted into the editor's text at the current cursor position.

Create default SELECT

This creates a `SELECT` for the selected table(s) that includes all columns for the table. This feature is intended for programmers who want to put a SELECT statement into their code.

The generated SELECT statement is always inserted into the editor's text at the current cursor position.

Create DDL Script

This shows the SQL source for the selected object(s). This is identical to the function in the DbExplorer.

Drop

Drops the selected objects. This is identical to the function in the DbExplorer.

Generate DROP script

This creates a script to DROP the selected object(s) including dependent objects. This is identical to the function in the DbExplorer.

Delete data

Deletes all rows from the selected table(s) by executing a `DELETE FROM table_name;` to the server for each selected table. This is identical to the function in the DbExplorer.

34. Common problems

34.1. The driver class was not found

If you get an error `"Driver class not registered"` or `"Driver not found"` please check the following settings:

- Make sure you have specified the correct location of the jar file. Some drivers (e.g. for IBM DB2) may require more than one jar file.
- Check the spelling of the driver's class name. Remember that it's case sensitive. If you don't know the driver's class name, simply press the **Enter** key inside the input field of the jar file location. SQL Workbench/J will then scan the jar file(s) to find the JDBC driver

34.2. Syntax error when creating stored procedures

When creating a stored procedure (trigger, function) it is necessary to use a delimiter other than the normal semicolon because SQL Workbench/J does not know if a semicolon inside the stored procedure ends the procedure or simply a single statement inside the procedure.

Therefor you must use an alternate delimiter when running a DDL statement that contains "embedded" semicolons. For details please refer to using the alternate delimiter.

34.3. The SQL source code for tables or indexes is incorrect

SQL Workbench/J re-creates the source code for tables and indexes based on the information returned by the JDBC driver. This does not alway match the original DDL used to create the table or index due to the limited information available by the JDBC API.

If the DBMS supports a SQL query to retrieve the real (native) source of a table or index, the query can be configured to be used instead of the generic reverse engineering built into SQL Workbench/J

Please see the chapter Customize table source retrieval for details on how to configure the query.

34.4. Timestamps with timezone information are not displayed correctly

When using databases that support timestamps or time data with a timezone, the display in SQL Workbench/J might not always be correct. Especially when daylight savings time (DST) is in effect.

This is caused by the handling of time data in Java and is usually *not* caused by the database, the driver or SQL Workbench/J

If your time data is not displayed correctly, you might try to explicitely specify the time zone when starting the application. This is done by passing the system property `-Duser.timezone=XYZ` to the application, where XYZ is the time zone where the computer is located that runs SQL Workbench/J

The time zone should be specified relativ to GMT and not with a logical name. If you are in Germany and DST is active, you need to use `-Duser.timezone=GMT+2`. Specifying `-Duser.timezone=Europe/Berlin` does usually **not** work.

When using the Windows® launcher you have to prefix the paramter with -J to identify it as a parameter for the Java runtime not for the application.

34.5. Some of the dialogs are too small

When using non-default font sizes in the operating system, it can happen that the windows shown in SQL Workbench/J are sometimes too small and some GUI elements are cut off or not visible at all.

All windows and dialogs can be resized and will remember their size. If GUI controls are not visible or are cut-off simply resize the window until everything is visible. The next time the dialog is opened, the chose size will be restored.

34.6. Excel export not available

In order to write the proprietary Microsoft Excel format, additional libraries are needed. Please refer to Exporting Excel files for details.

34.7. Generated XLSX files can't be opened with Excel

When using an Oracle JDBC driver together with the needed XDB libraries, the generated XLSX files can't be opened with Excel. This is caused by Oracle's library `xmlparserv2.jar` that conflicts with the libraries used to generate the XLSX file.

The only fix for this is to not use the Oracle XDB files (i.e. only use the `ojdbc8.jar` driver file, not the others.

34.8. Out of memory errors

The memory that is available to the application is limited by the Java virtual machine to ensure that applications don't use all available memory which could potentially make a system unusable.

If you retrieve large resultsets from the database, you may receive an error message indicating that the application does not have enough memory to store the data.

Please refer to Increasing the memory for details on how to increase the memory that is available to SQL Workbench/J

Some JDBC drivers also buffer the whole result in memory before returning it to the application. This can lead to high memory requirements for WbExport or WbCopy even though SQL Workbench/J does not need to load the whole result into memory. For Microsoft SQL Server and PostgreSQL this can be avoided by configuring the connection profile.

34.9. 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.

34.10. High CPU usage when displaying results

If you experience a high(er) CPU usage when displaying results, this can be caused by enabling the display of multi-line values. If that is the case, turning off that option, can reduce the CPU usage.

35. Common DBMS problems

35.1. Oracle

35.1.1. No Views or tables visible in the DbExplorer

Since Build 112 it is possible that the DbExplorer does no longer display views or tables if the selected schema (username) contains an underscore. This is caused by a bug in older Oracle JDBC drivers.

The driver calls used to display the list of tables and views in a specific schema expects a wildcard expression. To avoid listing the objects for `USERX1` when displaying the objects for `USER_1` the underscore must be escaped. The driver will create an expression similar to `AND owner LIKE 'USER_1' ESCAPE '\'` (which would return tables for `USERA1`, `USERB1` and so on, including of course `USER_1`).

The character that is used to escape the wildcards is reported by the driver. SQL Workbench/J sends e.g. the value `USER\_1` if the driver reports that a backslash is used to escape wildcards.

However some older Oracle drivers report the wrong escape character, so the value sent to the database results in `AND owner LIKE 'USER\_1' ESCAPE '/'`. The backslash in the expression is the character *reported* by the driver, whereas the forward slash in the expression is the character actually *used* by the driver.

To fix this problem, the escape character reported by the driver can be overridden by setting a property in `workbench.settings`:

```
workbench.db.oracle.searchstringescape=/
```

You can also change this property by running

```
WbSetConfig workbench.db.oracle.searchstringescape=/
```

This bug was fixed in the 11.2 drivers.

35.1.2. 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 running a `SELECT` statement that returns `LONG` or `LONG RAW` columns.

35.1.3. 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.

35.1.4. 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 WbSchemaReport command, then you have to enable the remarks reporting, otherwise the comments will not show up in the report.

35.1.5. Time for DATE columns is not displayed

A `DATE` column in Oracle always contains a time as well. If you are not seeing the time (or just 00:00:00) for a date column but you know there is a different time stored, please enable the option "Oracle DATE as Timestamp" in the "Data formatting" section of the Options dialog (Tools » Options)

35.1.6. Content of XMLTYPE columns is not displayed

The content of columns with the data type `XMLTYPE` cannot be displayed by SQL Workbench/J because the Oracle JDBC driver does not support JDBC's XMLType and returns a proprietary implementation that can only be used with Oracle's XDB extension classes.

The only way to retrieve and update XMLType columns using SQL Workbench/J is to cast the columns to a CLOB value e.g. `CAST(xml_column AS CLOB)` or `to_clob(xml_column)`

In the DbExplorer you can customize the generated SQL statement to automatically convert the XMLType to a CLOB. Please refer to the chapter Customize data retrieval in the DbExplorer for details.

Note

35.1.7. Error: "missing mandatory parameter"

When running statements that contain single line comments that are not followed by a space the following Oracle error may occur: `ORA-01009: missing mandatory parameter [SQL State=72000, DB Errorcode=1009]`.

```
--This is a comment
SELECT 42 FROM dual;
```

When adding a space after the two dashes the statement works:

```
-- This is a comment
SELECT 42 FROM dual;
```

This seems to be a problem with old Oracle JDBC drivers (such as the 8.x drivers). It is highly recommend to upgrade the driver to a more recent version (11.x or 12.x) as they not only fix this problems, but are in general much better than the old versions.

35.1.8. Connection is closed after cancelling a statement

If the Oracle server closes the connection after cancelling a statement, this can be fixed (in some installations) by adding the following extended property to the connection profile:

```
oracle.net.disableOob=true
```

35.2. MySQL

35.2.1. INFORMATION_SCHEMA tables not displayed in DbExplorer

It seems that the necessary API calls to list the tables of the `INFORMATION_SCHEMA` database (which is a database, not a schema - contrary to its name) are not implemented correctly in some versions of the MySQL driver. Currently only the version 5.1.30 is known to return the list of tables of the `INFORMATION_SCHEMA` database.

35.2.2. "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.

35.2.3. 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 behavior 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 https://dev.mysql.com/doc/connector-j/en/connector-j-reference-configuration-properties.html

To ignore errors

35.2.4. The SQL source for views is not displayed

SQL Workbench/J retrieves the view definition from `INFORMATION_SCHEMA.VIEWS`. For some unknown reason, the column `VIEW_DEFINITION` sometimes does not contain the view definition and the source is not displayed in the DbExplorer.

To make SQL Workbench/J use MySQL's `SHOW CREATE VIEW` statement instead of the `INFORMATION_SCHEMA`, you can set the property `workbench.db.mysql.use.showcreate.view` to true, e.g. by running `WbSetConfig workbench.db.mysql.use.showcreate.view=true`

35.2.5. No table comments are displayed in the DbExplorer

In order for MySQL's JDBC driver to return table comments, the connection property `useInformationSchema` must be set to true.

For details please see this bug report: https://bugs.mysql.com/bug.php?id=65213

35.3. Microsoft SQL Server

35.3.1. The value of DATE columns is not correct

It seems that the version 3.0 of the Microsoft JDBC driver returns the value of `DATE` columns with a wrong value (two days less than expected).

Version 4.0 of the Microsoft driver does not show this behavior. If you see wrong values for `DATE` columns and are using version 3.0, please upgrade your driver.

35.3.2. Column and table comments are not displayed

SQL Server does not support standard object remarks using `COMMENT ON` and the JDBC drivers (jTDS and Microsoft's driver) do not return the so called "extended attributes" through the JDBC API calls. To retrieve table and column remarks that are defined through the stored procedure `sp_addextendedproperty()`, SQL Workbench/J must run additional statements to retrieve the extended properties. As these statements can impact the performance of the DbExplorer, this is turned off by default.

To turn the retrieval of the extended properties on, please configure the necessary properties. For details, see the section Retrieving remarks for Microsoft SQL Server.

35.3.3. Using Windows authentication to connect to a SQL Server

In order to use the integrated Windows authentication (as opposed SQL Server Authentication) the Microsoft JDBC driver is **required**. It does not work with the jTDS driver.

When using Windows authentication the JDBC driver will try to load a Windows DLL named `sqljdbc_auth.dll`. This DLL either needs to be on the Windows `PATH` definition or in the directory where `SQLWorkbench.exe` is located. You need to make sure that you use the correct "bit" version of the DLL. If you are running a 32bit Java Runtime you have to use the 32bit DLL. For a 64bit Java Runtime you need to use the 64bit DLL (the architecture of the server is not relevant).

35.3.4. The Microsoft Driver throws an Exception when using SET SHOWPLAN_ALL

When displaying an execution plan using `SET SHOWPLAN_ALL ON` and the following error is thrown: `The TDS protocol stream is not valid. Unexpected token TDS_COLMETADATA (0x81).` please set "Max. Rows" to 0 for that SQL panel. Apparently the driver cannot handle showing the execution plan and having the result limited.

35.3.5. Dealing with locking problems

Microsoft SQL Server (at least up to 2000) does not support concurrent reads and writes to the database very well. Especially when using DDL statements, this can lead to database locks that can freeze the application. This affects e.g. the display of the tables in the DbExplorer. As the JDBC driver needs to issue a SELECT statement to retrieve the table information, this can be blocked by e.g. a non-committed `CREATE ...` statement as that will lock the system table(s) that store the meta information about tables and views.

Unfortunately there is no real solution to blocking transactions e.g. between a SQL tab and the DbExplorer. One (highly discouraged) solution is to run in autocommit mode, the other to have only one connection for all tabs (thus all of them share the same transaction an the DbExplorer cannot be blocked by a different SQL tab).

The Microsoft JDBC Driver supports a connection property called `lockTimeout`. It is recommended to set that to 0 (zero) (or a similar low value). If that is done, calls to the driver's API will through an error if they encounter a lock rather than waiting until the lock is released. The jTDS driver does not support such a property. If you are using the jTDS driver, you can define a post-connect script that runs `SET LOCK_TIMEOUT 0`.

35.3.6. WbExport or WbCopy using a lot of memory

The jTDS driver and the Microsoft JDBC driver read the complete result set into memory before returning it to the calling application. This means that when retrieving data, SQL Workbench/J uses (for a short amount of time) twice as much memory as really needed. This also means that WbExport or WbCopy will effectively read the entire result into memory before writing it into the output file. For large exports this us usually not wanted.

This behavior of the drivers can be changed by adding an additional parameter to the JDBC URL that is used to connect to the database. For the jTDS driver append `useCursors=true` to the URL, e.g. `jdbc:jtds:sqlserver://localhost:2068;useCursors=true`

The URL parameters for the jTDS driver are listed here: https://jtds.sourceforge.net/faq.html#urlFormat

For the Microsoft driver, use the parameter `selectMethod=cursor` to switch to a cursor based retrieval that does not buffer all rows within the driver, e.g. `jdbc:sqlserver://localhost:2068;selectMethod=cursor`

Note that since Version 3.0 of the driver

The URL parameters for the Microsoft driver are listed here: https://learn.microsoft.com/en-us/sql/connect/jdbc/setting-the-connection-properties

35.3.7. Sequences are incremented twice

When a sequence is incremented twice when running `SELECT NEXT VALUE FOR MYSEQ;` and you are using the Microsoft JDBC driver with the `selectMethod=cursor`, remove the `selectMethod=cursor` option from the JDBC URL. The sequences will then be incremented correctly.

35.4. IBM DB2

35.4.1. "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`.

35.4.2. XML columns are not displayed properly in the DbExplorer

The content of columns with the data type `XML` are not displayed in the DbExplorer (but something like `com.ibm.db2.jcc.am.ie@1cee792` instead) because the driver does not convert them to a character datatype. To customize the retrieval for those columns, please refer to the chapter Customize data retrieval in the DbExplorer.

When using a JDBC4 driver for DB2 (and Java 6), together with SQL Workbench/J build 107, XML content will be displayed directly without the need to cast the result.

35.4.3. No error text is displayed

When running SQL statements in SQL Workbench/J and an error occurs, DB2 does not show a proper error message. To enable the retrieval of error messages by the driver you have to set the extended connection property `retrieveMessagesFromServerOnGetMessage` to `true`.

The connection properties for the DB2 JDBC driver are documented here: https://www.ibm.com/docs/en/db2/10.5.0?topic=information-properties-data-server-driver-jdbc-sqlj

35.4.4. Displaying column headers instead of column names in result sets.

When running SQL statements in SQL Workbench/J you might want to use the long column headings (created via `LABEL ON`) as opposed to the column name. To enable the retrieval of error messages by the driver you have to set the extended connection property `extended metadata` to `True`.

The connection properties for the DB2 JDBC driver are documented here: https://www.ibm.com/docs/en/db2/10.5.0?topic=information-properties-data-server-driver-jdbc-sqlj

35.4.5. Column or table comments are not displayed

The DB2 JDBC driver does not return the column description stored in SYSCOLUMNS.COLUMN_TEXT, or SYSTABLES.TABLE_TEXT. If you are using these descriptions, you can enable retrieving them (and overwriting the comments returned by the driver) by setting the following two configuration properties to `true` (e.g. using WbSetConfig)

`workbench.db.db2i.remarks.columns.use_columntext` for column comments
`workbench.db.db2i.remarks.tables.use_tabletext` for table comments

35.4.6. DB2 commands like REORG cannot be run

REORG, RUNSTATS and other db2 command line commands cannot be be run directly through a JDBC interface because those are not SQL statements, but DB2 commands. To run such a command within SQL Workbench/J you have to use the function `sysproc.admin_cmd()`. To run e.g. a REORG on a table you have to run the following statement:

```
call sysproc.admin_cmd('REORG TABLE my_table');
```

35.5. Amazon Redshift

35.5.1. "ERROR: type "e" does not exist" when using the DbExplorer

This error occurs when using a Postgres JDBC driver with Amazon Redshift.

For Redshift, the Redshift driver should be used: https://docs.aws.amazon.com/redshift/latest/mgmt/jdbc20-download-driver.html

35.6. PostgreSQL

35.6.1. High memory usage

The PostgreSQL JDBC driver defaults to buffer the results obtained from the database in memory before returning them to the application. This means that when retrieving data, SQL Workbench/J uses (for a short amount of time) twice as much memory as really needed. This also means that WbExport or WbCopy will effectively read the entire result into memory before writing it into the output file. For large exports this us usually not wanted.

This behavior of the driver can be changed so that the driver uses cursor based retrieval. To do this, the connection profile must disable the "Autocommit" option, and must define a default fetch size that is greater than zero. A recommended value is e.g. 100, it might be that higher numbers give a better performance. The number defined for the fetch size, defines the number of rows the driver keeps in its internal buffer before requesting more rows from the backend.

More details can be found in the driver's manual: https://jdbc.postgresql.org/documentation/query/#getting-results-based-on-a-cursor

35.7. Sybase SQL Anywhere

35.7.1. Columns with type `nvarchar` are not displayed properly

The jConnect driver seems to have a problem with `nvarchar` columns. The data type is not reported properly by the driver, so the display of the table structure in the DbExplorer will be wrong for those columns.

36. Options dialog

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

36.1. General options

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.

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 home page. If a newer version is found on the web site 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 home page.

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

When SQL Workbench/J performs an update check, it sends the following information as part of the request to the server:

- The version of SQL Workbench/J you are using
- Whether the check was an automatic check or a manual one
- The interface language selected
- The operating system as reported by your Java installation
- The Java version you are using

Show connect dialog

If this option is enabled, the connect dialog will be shown automatically when the application is started.

Master password

After providing a master password, all passwords in the connection profiles are encrypted using the master password. Once the master password is set, it is necessary to enter it, whenever SQL Workbench/J is started.



If the master password is lost or forgotten, it's impossible to restore the original database passwords.

Exit on first connect cancel

If this option is enabled, then the application is closed completely if the initial connect dialog is canceled.

This option is only valid if "Show connect dialog" is selected.

Consolidate script log

Usually SQL Workbench/J reports the success and timings for each statement that is being 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.

Auto-save connection profiles

If this option is enabled, the connection profiles are automatically saved when closing the connection dialog using the OK button.

If this option is disabled, the connection profiles are saved when closing the application.

Single page HTML help

If this option is enabled, the HTML help will be shown as a single page in the browser instead of one page per chapter.

Quick filter for profiles

If this option is enabled, an input field to filter the profiles is displayed above the list of profiles.

Set focus to quick filter

If this option is enabled, the initial focus will be put into the quick filter input field after opening the connection dialog.

Separate variables per window

Controles the visibility of variables if more than one main window is opened. See Scope of variables for more details.

Restore expanded profile groups

If this option is enabled, the state of the expanced and opened profile groups is restored when the application is started.

Confirm tab close

If this option is enabled, closing a tab needs to be confirmed, to prevent accidental closing.

Confirm closing multiple tabs

If this option is enabled, a confirmation is required when closing multiple tabs using the option "Close other tabs".

Show tab index

If this option is enabled, each editor tab will be shown with its index. You can then select the first 9 tabs by pressing **Ctrl-1**, **Ctrl-2** and so on.

Scroll tabs

This option controls the behavior of the tab display, if more tabs are opened than can be displayed in the current width of the window.

If this option is enabled, the tabs are always displayed in a single row. If too many tabs are open, the row can be scrolled to the display the tabs that are not visible.

If this option is disabled, the tabs are displayed in multiple rows, so that all tabs are always visible.

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 (negative) 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.

Configuration file information

At the bottom of the "General options" page, the full filename of the configuration file and the log file are listed.

36.2. Editor options

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.

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.

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.

Files in history

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

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.

Editor tab width

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

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).

Insert closing brackets for

To enable auto-completion of brackets, enter pairs of characters that should automatically be "closed", e.g. `()''` will automatically insert a closing bracket when an opening bracket is typed. To auto-complete quote characters enter two quotes.

To disable automatic closing of brackets enter nothing in this input field.

Right click moves cursor

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)

Store last directory in workspace

If this option is enabled, the directory from the last opened file is stored in the workspace of the current profile, not globally. If this option is unchecked, the last directory will be stored globally and will be used for all connections.

Current directory follows active file

If this option is enabled, the file open dialog will default to the directory of the current file in the editor. If no file is loaded in the editor, the directory that is defined through the "Default directory" option will be selected.

36.3. SQL Excecution options

Error prompt for scripts

This options defines what kind of dialog is shown when an error occurs during script execution. The dialog always offers the choice to ignore the error, ignore all subsequent errors or to cancel the script execution.

The following options are available:

- Simple prompt - it shows only the statement number that failed.
- Include error message - this includes the actual error message from the DBMS (this is the default)
- Show statement and allow retry - this includes the error message and the complete SQL statement that failed. It allows to edit and re-submit the statement.

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

Highlight current statement

When running several statements (e.g. by using "Execute all") this option will highlight the current statement. The editor will be scrolled to make sure the currently executed statement is visible.

Retain current statement highlight

If "Highlight current statement" is enabled and this option is turned on, the highlighting will be kept once execution has finished.

Highlight errors

If "Highlight errors" is enabled then the statement that generated an error is highlighted after execution.

Always allow "Execute Selected"

If this option is turned off, then SQL » Execute Selected will only work if text is selected in the editor. If this option is turned on and no text is selected, the complete content of the editor will be executed.

Auto advance to 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 » Settings » Auto advance to next

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

Allow editing while executing

When running a statement, the editor is set to read-only in order to allow a consistent statement highlighting. When this option is turned on, the text in the editor may be modified even if a statement is running. If the text in the editor is modified during execution, statement and error highlighting will not be done any more.

Allow empty lines as statement delimiter

When analysing statements in the editor, it is assumed that individual statements are separated with a semicolon. This property controls if an empty line delimits a statement as well. This setting will be used to detect the current statement for auto-completion and when using Execute current inside the editor.



This does not influence the behavior when running scripts in batch mode or when using the `WbInclude` command.

Use the statement in the current line

When this option is enabled, the line of the cursor position determines the "current statement" rather than the cursor position itself when using Execute current [50].

36.4. Macro options

Macro expansion key

This defines the key combination that triggers the detection of expandable macros.

Save window settings in workspace

If this option is enabled, the current position and the expaned macro groups of the macro popup window are stored in the current workspace. It this option is disabled, these settings are saved globally.

Close window with ESC

If this option is enabled the macro popup window can be closed using the **ESC** key.

Execute macro with Enter

If this option is enabled the currently selected macro can be run by hitting the **Enter** key.

Macrolist position

This option controls where the list of macros will be displayed, if Macros » Macro list is selected. The possible posititons are:

- Floating - the macro list will be shown as a popup window
- Left - the macro list will be in a panel on the left side inside the main window
- Right - the macro list will be in a panel on the right side inside the main window

Base directory

If a set of macros is specified for a connection profile with a relative filename, the macros will be loaded relative to this base directory.

36.5. Bookmark options

Also use @WbResult for bookmarks

By default only locations marked with `@WbTag` are included in the list of bookmarks. When this option is enabled, locations that are marked with `@WbResult` are also included in the bookmark list.

Use procedure/function names as bookmarks

If this is enabled SQL Workbench/J will also the names of procedures or functions for which a `CREATE` statement is present in the editor as a bookmark.

Include parameter names for procedures and functions

When including procedure and function names in the bookmarks, only the datatype of the parameters are shown in the bookmark list. If this option is enabled the parameter names (if available) are also shown.

Remember column widths

If this is enabled, the width of the columns in the bookmark window is not resized to match the displayed value, but rather the last with is remembered.

Remember sort order

If this is enabled, the sort order of the bookmark list is restored the next time the bookmark window is displayed.

36.6. Editor colors

Current line color

If you want to highlight the line in which the cursor is located, specify the color for the highlighting. To disable the highlight for the current line, simply "remove" the color selection by clicking on the remove button.

Selected text

The color that is used to highlight selected text.

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.

Syntax highlighting colors

You can change the colors for the different types of keywords in the editor.

36.7. Font settings

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.

Data font

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

Message font

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

Standard font

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

36.8. Auto-completion options

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)

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.

Close completion with search

When using the quicksearch feature in the code completion this option controls the behavior 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.

Sort entries in popup

If this is enabled, columns are sorted alphabetically in the popup. If not, they are listed in the order as they are returned by the the database.

Quick search matches anywhere

If this option is enabled, the typed characters match anywhere in the object name. If this option is disabled, the object name must start with the entered search value.

Filter by quicksearch

When this option is enabled, only those entries are shown in the popup that match the entered values in the quick search.

Prefer USING operator

If this option is enabled, the JOIN completion generates a `USING` clause instead of an `ON` clause to join the tables. If there are no columns with identical names, a join with an `ON` operator is generated.

Always use parentheses

If this option is enabled, JOIN completion will generated redundant parentheses around the join condition for the `ON` operator.

36.9. Workspace options

Save workspace before running SQL statements

If this option is enabled, the current workspace is saved each time you run a SQL statement.

If this option is disabled, the workspace is only saved when the application is closed, when a a different workspace is loaded (either manually or by connecting to a different connection profile) or when the workspace is manually saved through the menu using Workspace » Save Workspace

If automatic backups of the workspace is enabled, this means that a new version of the workspace file is created **each time** a statement is executed. In that case the number of backups should be high enough so that also substantial changes to the workspace are still kept in the history.

Save interval in minutes

The workspace is automatically saved after the specified number of minutes. An empty value or a value of zero disables this.

Base directory

If workspaces are specified as relative files or directories in the connection profile they are searched relative to this base directory.

Remember open files in workspace

You can customize how external files (that have been loaded using File » Open) are remembered in the workspace. You can select three different options:

Content and filename	When this option is selected, the filename that is loaded in the editor tab will be stored in the workspace. The next time the workspace is loaded the file is opened as well. This is the default setting
Content only	When this option is selected, only the content of the editor tab is save (just like any other editor tab), but the link to the filename is removed. The next time the workspace is loaded, the file will **not** be opened.
Nothing	Neither the content, nor the filename will be saved. The next time th workspace is loaded, the editor tab will be empty.

Controlling how file references are stored

When a reference to a file is stored in the workspace, this is stored with the full path to the file by default. If workspaces are moved or shared between computers, it might be better to store these file references as relative references. It is possible to configure the base directory to which relative file references are stored.

The relative file path is always stored in addition to the absolute file path.

Never	If this is selected, file references are never stored as relative paths.
Config directory	The stored path will resolve relatively to the configuration directory
Workspace base directory	The stored paths will resolve relatively to the base directory for workspaces.
Workspace base	The stored paths will resolve relatively to the directory of the workspace itself.
Profile script dir	The stored paths will resolve relatively to the default directory of the connection profile. If this option is selected, then every profile using a workspace **must** specify a default directory, otherwise no relative path will be stored in the workspace.
Specific directory	The stored paths will resolve relatively to the directory specified. If this is chosen, then the specific directory needs to be configured for each installation that uses this feature. The selected directory is stored in the `workbench.settings` file using the property name `workbench.workspace.external.files.relative.basedir`. Configuration properties can also be provided as system property when starting SQL Workbench/J e.g. using: `-Dworkbench.workspace.external.files.relative.basedir=/path/to/base/directory`

36.10. Backup options

If an automatic backup is enabled for a file, then each time a file is saved, a new version of the file will be created in the backup directory if Number of backups is greater than 1.

A file version will be created by appending a version number to the file name (e.g. `testdata.wksp.1`, `testdata.wksp.2` and so on). The most recent version is the one with the highest number.

Create backups of workspace files

If this option is enabled the current workspace file will be backed up.

Create backups of the settings file

If this option is enabled, the settings file will be backed up.

Create backups of the macro file

If this option is enabled, the file that contains macro definitions will be backed up.

Create backups of the connection profiles

If this option is enabled, the file that contains connection profiles will be backed up.

Create backups of the driver definitions

If this option is enabled, the file that contains JDBC drivers will be backed up.

Number of backups

This option defines the maximum number of version to keep if backups are enabled.

Backup directory

By default the backups for the workspaces are stored in the same directory as the workspace file itself. If you want to keep the (versioned) backups in a separate directory, you can specify it here.

If you specify a relative directory, it will be relative to the config directory.

36.11. Options for displaying data

Show selection summary in statusbar

If this option is enabled the number of selected rows in the result will be displayed in the status bar.

If you have a single numeric column selected (by holding down the **Alt** key while selecting with the mouse), the status bar will display the sum of the selected values.

Use table name for result tabs

If this option is enabled, the name of a result tab is derived from the `SELECT` that was used to generate the result. The query is analyzed and the first table name mentioned in the `FROM` clause will be used for the name of the result.

The table name will not be used when the `@WbResult` annotation is also specified for the query.

Retrieve column remarks for queries

If this option is enabled, the remarks defined for table columns in a result set are retrieved and shown as a tool tip. As this requires additional overhead after processing a query, it can be disabled for performance reasons.

Show max. rows warning

If this option is enabled the result tab will show a warning sign if the limit defined by the max. rows setting is reached, indicating that the result might be incomplete.

Show row numbers

If this option is enabled the row numbers for result sets are shown at the left hand side of the result.

Show tooltip warning

If this option is enabled, a tooltip indicating that the maximum number of rows has been reached is shown for the result tab.

Default max. Rows

When adding a SQL panel, this number will be used as a default for the max. rows value for the new panel.

Display SQL query

If this option is enabled, the query that generated a result is shown right above the result grid.

`NULL` string

The specified value will be displayed instead of `NULL` values in the result of a SQL statement.

Append results

This option defines the default behavior for appending results [52] when a new editor tab is opened.

Number alignment

This controls the alignment of numbers in the result grid.

Result tab tooltip

This options configures the tooltip that is shown when the mouse is hovering over a result tab.

Sort Locale

When you sort the result set, characters values will be sorted case-sensitive 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. With the "Sort Locale" option you can select which language rules should be applied while sorting. Note that sorting with a locale is slower than using the "Default" setting.

36.11.1. Column header settings

Show data type in header

If this option is enabled, the data type (as returned by the JDBC driver) of each column is shown in the header below the column name.

Bold header

If this option is enabled, the name of the columns in the result is shown with a bold font, instead of the regular data font.

Show remarks in header

If this option is enabled, the column remarks are shown in the header. This requires that Retrieve column remarks for queries is enabled

Show table name in header

If this option is enabled, the table to which a column belongs to is shown in the header.



This option depends on the JDBC driver. If the JDBC driver does not return this information nothing will be displayed.

Show as column prefix

By default, the the table name for a column is displayed below the column name. When this option is enabled, it will be shown as part of the column name, e.g. `person.firstname`.

36.11.2. Column width settings

Automatically 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) after retrieving data. Note that you can manually optimize the column widths using View » Optimize width for all columns.

Adjust to 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.

This option is also used when manually optimizing the column width,

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.

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.

Displaying multi-line values

SQL Workbench/J uses a special display component for the contents of `CLOB` columns that is capable of displaying multiple lines. This component honors newlines and linefeeds in the data retrieved from the database and is capable of using word wrapping for long lines (even if no newlines are embedded).

By default only `CLOB` are considered to be able to contain multiple lines, so `VARCHAR` columns are usually not treated as multi-line columns. If your database stores text in VARCHAR columns that contains line breaks, you can define a threshold for the length of the column. Any column that is defined with a higher value will be displayed with the multiline component.

The default value of 250 means that a `VARCHAR(250)` column will be displayed with the multi line renderer. A `VARCHAR(210)` will be displayed in a single line. Note that this limit refers to the *defined* length of the column, not the actual length of the data.

Displaying data using the multi line component is slower than using the standard (single line) component.

The feature Adjust row height only works with multi-line columns.

36.11.3. Row height settings

Automatically adjust row height

If this option is enabled, the height of each column is automatically adjusted after data retrieval to display as many lines of the column values (for character columns) as possible. Note that you can manually optimize the row height using View » Optimize row height.

Not every (character) column is displayed in a manner that multiple lines will be displayed. The default setting is to always display `CLOB` columns as multi line. `VARCHAR` (and `CHAR`) columns will only be displayed in multi line mode if they can hold more than 250 characters. This limit can be changed.

This can be enabled for a single query using the `@WbOptimizeRowHeight` annotation.

Allow row height resizing

If this option is enabled, you can manually adjust the height of each row using the mouse. This option does not need to be enabled in order to (automatically) optimize the row height.

Max. number of lines

When calculating the optimal height for each row, the number of lines defined with this option will never be exceeded.

36.12. Options for formatting data

Date, timestamp and time formats

Define the format for displaying date, date/time (timestamp) and time columns in the result set. For more details on the format of this option, please refer to the documentation of the `DateTimeFormatter` 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)

Letter	Description
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)
V	Time zone ID, e.g. Asia/Saigon

Optional elements can be enclosed in square brackets, e.g. `[ Z]` to display time zone information only if available. If no format is defined through "Timestamp TZ format" for columns defined with `TIMESTAMP WITH TIME ZONE`, then the format for timestamps used. In that case the time zone information should be formatted with square brackets to only show them when needed.

Oracle `DATE` as `TIMESTAMP`

The Oracle `DATE` datatype includes the time as well. But the JDBC driver does not retrieve the time part of a DATE column, so when retrieving `DATE` values, this would remove the time stored in the database. If this option is enabled, SQL Workbench/J will treat Oracle's `DATE` columns as `TIMESTAMP` columns, thus preserving the time information.

Note that the Oracle 12.x drivers don't allow to switch this **off**. If if this parameter is unchecked the Oracle 12.x driver will return values from `DATE` columns as timestamps.

Decimal symbol

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

Group separator

The character to be used to separate groups inside a number. This is only used when a format string is specified, that includes the placeholder for the group character.

Decimal digits

Define the maximum number of digits which will be displayed for numeric columns. This only affects the display of the number, not the storage or retrieval. Internally they are still stored as the DBMS returned them. To see the internal value, leave the mouse cursor over the cell. The tool tip 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.

If this value is set to 0 (zero) values will be display with as many digits as available.

Decimal/Integer format

Define the format for displaying integers or numbers with decimals. For details on the format of this option, please refer to the documentation of the `DecimalFormat` class.

If a format string is defined, the "Decimal digits" option will not be used. The number of decimal digits has to be controlled through the format string in that case.

Here is an overview of the characters that can be used to format numbers:

Character	Description
0 (zero)	Digit, a zero is shown as a zero
#	Digit, zeroes are suppressed
.	Marks the position where the decimal separator should be displayed
,	Marks the position where the grouping symbol should be displayed

36.13. Data display colors

Text color

The text color in result grids.

Background color

The background color in result grids.

Alternate row color

If this color is defined, the background of every other row will use this color instead of the default background color.

Selection color

The background for selected rows in result grids.

Selected text color

The selection color when editing values in result grids.

Color for `NULL` values

The background of cells in the result grid where its value is NULL.

Color for modified values

The background of cells in the result grid if they have been changed after retrieval.

Color for search highlight

The background of cells with matching values when searching for values in the result and enabling the option "Highligh all".

Blend with selection

Blend the row selection color with the column highlighting colors. A value of 0 means the selection color will be used, a value of 256 means the column color will be used.

Blend with alternate color

Blend the alternate row color with the column highlighting colors. A value of 0 means the alternating row color will be used, a value of 256 means the column highlight color will be used.

36.14. Options for data editing

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!

Confirm discarding changed results

When running a statement that would replace a result that has changes that are not saved to the database, you will be prompted whether you want to cancel the current operation that would discard those changes.

This applies to statements run in the editor, as well as to changes done in the Data tab of the DbExplorer.

You will not be prompted when running statements in the editor, when the option Append results [52] is enabled.

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.

Color for required fields

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

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.

Single record dialog

When displaying data in the Single record dialog you can customize the width for the input fields, and the default height for multiline columns.

36.15. DbExplorer options

36.15.1. General options

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

Automatically retrieve dependency tree

If this option is enabled, the tree display in the "References" and "Referenced by" tabs will automatically be loaded when the list of foreign keys is loaded. If this option is disabled, loading of the tree display must be started manually by clicking on the "reload" button.

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.

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.

Focus to source panel

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

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**.

Object details tabs

With this drop down you can select the position of the details tabs (Columns, Source, Data etc).

36.15.2. Object list options

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.

Allow table altering

If this option is enabled, column definitions of a table can directly be altered by editing them inside the "Columns" tab. It also allows to directly change the name of a table in the table list.

Remember object type

The list of objects can be filtered with the drop down. 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.

Default object type

If "Remember object type" is not enabled, you can define a default object type that is selected in the drop down when the DbExplorer is displayed initially.

Quick filter options

The following configuration options of the quick filter are also used for the quick filter in the Database Objects Tree.

Use RegEx in Quickfilter

If this option is enabled, the expression entered in the quick filter of the DbExplorer's table list is used as a regular expression (rather than a "SQL" Expression) to filter the list.

Partial match

If this option is enabled, then any text that is typed into the quick filter will be matched anywhere in the object name. It is equivalent to typing `*foo*` into the quick filter. If this option is enabled and a wildcard is part of the value, then only that wildcard is used. Using `foo*` for the filter while this option is enabled, shows all objects that *start* with `foo`.

This also enables filtering for multiple values, separated by comma. Entering `foo,bar` in the quick filter will show all objects that contain either `foo` or `bar`.

This option is only available when the use of regular expressions in the quick filter is disabled.

Filter while typing

If this option is enabled, the filter expression is applied while you type. In this case, the "Filter" button does not need to be clicked in order to apply the filter expression.

36.15.3. SQL Generation options

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.

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

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

Generate table grants

If this option is enabled the generated table source will contain any table grants that have been defined.

Generate DROP statement

If this option is enabled the generated table source will start with the appropriate `DROP` statement.

36.15.4. Data display options

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.

Remember column order

When you reorder the column in the data display of a table, enabling this option will automatically store the new column order and apply it the next time the table data is displayed.

Apply sort order as ORDER BY

If the table data was sorted by clicking on one of the columns, reloading the data will use an appropriate `ORDER BY` clause for the data retrieval. This is useful if not all rows were displayed in the data panel due to a max. row limit and you want the first rows displayed based on the current column sort.

36.16. DbTree options

Use the current tab's connection

By default the DbTree uses a separate connection to the database to avoid locking problems. However this could mean that changes done in a transaction in a SQL tab, might not be visible in the DbTree. E.g. when a session local temporary table is created.

If this option is enabled, the DbTree will use the connection of the currently active tab.

Apply filter while typing

If this option is enabled, the quick filter is applied while typing in the input field. If this option is disabled, the filter will be activated when pressing the **Enter** key in the filter field.

36.17. File tree options

Exclude files

Enter a comma separated list of file names that should never be shown in the file tree.

Exclude directories

Enter a comma separated list of directory names that should never be shown in the file tree.

Default directories

Select a list of directories that should be opened automatically when the file tree is displayed.

When the file tree is open, when a workspace is saved (e.g. because SQL Workbench/J is closed), then the currently displayed directories are saved in the workspace. The next time the workspace is loaded, the directories stored in the workspace are used, not the default directories from this option.

36.18. SQL Formatting

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

Max. length for sub-select

When the SQL formatter hits a sub-SELECT while parsing it will not reformat any statement which is shorter than 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.

Columns in SELECT

This property defines the number of columns the formatter puts in on line when formatting a `SELECT` statement. The default of 1 (one) will put each column into a separate line:

```
SELECT p.name,
       p.firstname,
       a.city,
       a.zip
FROM person p
  JOIN address a ON p.person_id = a.person_id;
```

If this is set to 2, this would result in the following formatted SELECT:

```
SELECT p.name, p.firstname,
       a.city, a.zip
FROM person p
  JOIN address a ON p.person_id = a.person_id;
```

The above example would list all columns in a single line, if this option is set to 4 (or a higher value):

```
SELECT p.name, p.firstname, a.city, a.zip
FROM person p
  JOIN address a ON p.person_id = a.person_id;
```

Columns in INSERT

This property defines the number of columns the formatter puts in on line when formatting an `INSERT` statement. A value of one will list each column in a separate line in the `INSERT` part and the `VALUES` part

```
INSERT INTO PERSON
```

```
(
  id,
  firstname,
  lastname
)
VALUES
(
  42,
  'Arthur',
  'Dent'
);
```

When setting this value to 2, the above example would be formatted as follows:

```
INSERT INTO PERSON
  (id, firstname,
   lastname)
VALUES
  (42, 'Arthur',
   'Dent');
```

Columns in UPDATE

This property defines the number of columns the formatter puts in on line when formatting an `UPDATE` statement. A value of 1 (one) will put each column into a separate line:

```
UPDATE person
   SET firstname = 'Arthur',
       lastname = 'Dent'
WHERE id = 42;
```

With a value of 2, the above example would be formatted as follows:

```
UPDATE person
   SET firstname = 'Arthur', lastname = 'Dent'
WHERE id = 42;
```

Keywords

This option defines if standard SQL keywords are generated in upper case, lower case or left unchanged.

Identifiers

This option defines if identifiers (table names, column names, ...) are generated in upper case, lower case or left unchanged.

Functions

This option defines if the names of SQL functions are generated in upper case, lower case or left unchanged. This does not apply to user-written functions, only standard functions available for the current DBMS.

JOIN wrapping

This option controls how conditions for `JOIN` operators are generated

Never

The JOIN condition is always kept on a single line:

```
SELECT *
FROM person p
  JOIN address a ON p.person_id = a.person_id;
```

Always

the JOIN condition is always written on a new line:

```
SELECT *
FROM person p
  JOIN address a
    ON p.person_id = a.person_id;
```

Multiple conditions

the JOIN condition is generated on multiple lines only if the join involves more than one condition:

```
SELECT *
FROM person p
  JOIN address a ON p.person_id = a.person_id;
  JOIN address_details ad
    ON ad.address_id = a.address_id
   AND ad.person_id = a.person_id;
```

Space after comma for IN lists

If this option is selected, a space is added after the comma inside an `IN` list.

Comma after line break

If this option is enabled, the commas inside the `SELECT` list are put on the start of the next line, rather than on same line as the last column.

If this option is disabled a `SELECT` statement will be formatted like this:

```
SELECT id,
       lastname,
       firstname
FROM person;
```

If this option is enabled the above statement will be formatted like this:

```
SELECT id
       ,lastname
       ,firstname
FROM person;
```

Space after a comma with line break

This option is only available is "Comma after line break" is enabled. In that case it controls if a space character is inserted after the comma.

Maximum elements per line for "Create SQL IN list

This option is used when changing the selected text into elements suitable for an `IN` list using SQL » Create SQL List. The limit can be defined separately for values that require the use of single quotes and values that don't need single quotes like numbers.

36.19. External SQL Formatter

Use of the external formatter

When formatting a SQL statement, SQL Workbench/J first looks if a formatter for the current DBMS is defined and active. If a formatter is found, that is used. If no formatter for the current DBMS is found, and the "Default" external formatter is active, that is used. If no active external formatter is found, the internal formatter is used.

Executable

This is the full path to the formatter's program

Command line

The command line configures the parameters passed to the formatter. The input file for the formatter can be specified by using the placeholder `${wbin}`. If no input file is specified on the command line, the SQL statement will be passed through stdin. If the formatter writes the output to a file, the placeholder `${wbout}` can be used in the command line. If no output file is specified the result will be read from stdout of the process.

Supports multiple statements

If this option is enabled, SQL Workbench/J send the selected text that should be formatted as a single input to the formatter. If this option is disabled, SQL Workbench/J will split up the text to be formatted and send each statement seperately to the formatter.

Enabled

This option can be used to turn of the usage of a formatter without deleting the definition.

36.20. SQL Generation

Format UPDATEs

If formatting of `UPDATE` statements is enabled, generated `UPDATE` statements are formatted using the SQL formatter before they are displayed.

Format INSERTs

If formatting of `INSERT` is enabled, the way they generated `INSERT` statements are formatted using the SQL formatter before they are displayed.

Format DELETEs

If formatting of `DELETE` is enabled, the way they generated `DELETE` statements are formatted using the SQL formatter before they are displayed.

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.

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.

Date literals for WbDataDiff

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

Include owner in export

This setting controls whether SQL Workbench/J uses the owner (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.

Include empty comments

By default the DbExplorer will not generate the SQL to create empty comments (tables, views, columns,...). If this option is enabled then a corresponding SQL statement to define a comment with an empty string will be generated. If a comment is `NULL` comments will never be generated.

Ignore identity and autoincrement columns for INSERTs

If this option is enabled, generated INSERT statements (e.g. when editing data) will not contain identity or autoincrement columns. When using `WbExport` to create a SQL script, this can be controlled independently from the global option.

36.21. 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.

Application name at end

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

Show Workspace name

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

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.

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.

Separator

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

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.

36.22. SSH Hosts

By defining global SSH hosts, it's possible to edit the login information for the SSH host centrally instead of editing the SSH settings for each connection profile.

If the details of such a SSH host change (e.g. the password), they only need to be changed once, and all connection profiles using the host automatically use the new definition.

Option	Description
Configuration	The name for this global SSH host.
SSH Hostname	This is the remote computer that is used to establish the SSH connection and from which the remote database server is reachable

Option	Description
SSH Port	The port for the SSH connection. If this is empty the default port 22 is used.
Username	The username on the SSH host to be used for the login.
Private key file	If the SSH host is configured to use authentication through a SSH key file, the path to the key file can be entered here
Password	This is either the password for the remote user login, or, if a private key file is specified, the passphrase for that key file. If no password is provided, you will be prompted to enter one when connecting. If a password is entered, it is **always** stored in the profile. If no password is entered, you will be prompted for one when the connection is established. If the password is used as a passphrase for a private key file, the entered password will be remembered until the application is closed. It will be (re)used for other tunnels using the same private key file.
Authenticate through SSH agent	If this option is enabled SQL Workbench/J will try to use a running SSH agent to authenticate against the SSH host. Supported agents are `pageant.exe` for Windows® and `ssh-agent` for Linux systems. In this case supplying a private key file or a password is not necessary, but the user name is still required. In order to be able to communicate with the SSH agent, additional libraries are needed, which can be downloaded from https://www.sql-workbench.eu/jna.zip. Please unzip the archive and put the two jar files into the `ext` sub-directory where `sqlworkbench.jar` is located.

36.23. 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.

If the tool needs additional parameters (e.g. to select a hex editing mode for a text editor), they have to be entered in the "Parameters" field. Do not add parameters to the definition of the executable.

36.24. 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.



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.

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.

36.25. Customize the FlatLaf Look & Feel

The FlatLaf Look and Feel is bundled with SQL Workbench/J

FlatLaf can be customized through properties files. This file needs to be put into the configuration directory or `ext` directory where the `flatlaf.jar` is located.

FlatLaf will search multiple files to find the one to be used. A file named `FlatLaf.properties` will always be loaded. After that, a property file specific to the selected FlatLaf variant is loaded. The variant specific properties file needs to have the name of the variant's class, e.g. `FlatLightLaf.properties` or `FlatDarkLaf.properties`.

36.26. Logging

Log all SQL statements as INFO messages

If this option is enabled, all manually entered SQL statements that logged into the log file as INFO messages. The statements will be logged without changs, which means potentially sensitive information could be written to the logfile.

Log internal SQL statements as INFO messages

If this option is enabled, all statements that SQL Workbench/J runs internally are written to the log file as INFO messages.

Obfuscate information in logfile

If this option is enabled, some information is obfuscated before written to the logfile. This includes details about the connection (JDBC URL, server, database and user names) as well as paths to local files.

If SQL statements are logged, those are **not** changed in any way to hide sensitive information

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.

When using Log4J as the logger, this will change the log level of the root logger.

37. 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.

37.1. 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

37.2. 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).

37.3. 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.

38. Advanced configuration options

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, create a new line with the property name and the value as described here. The position where you add this entry does not matter.

You can also change the values for these properties while the application is running by using the command WbSetConfig.



Every property can also be specified on the command line when starting SQL Workbench/J by setting a system property with that name using the `-Dworkbench.property=value` switch when starting the application using the `java` command directly.

You can edit the file using a text editor. In that case you must close the application before editing the file, otherwise your changes will be overwritten when the application is closed.

You can also change any property using the SQL Workbench/J command WbSetConfig. For most of the parameters the change will be in effect immediately. For some you will still need to restart the application or at least re-connect to the database.

38.1. DBID

DBMS specific settings are controlled through properties that contain a DBMS specific value, called the the DBID.

This DBID is displayed in the connection info dialog that is displayed, when you right click on the connection URL in the main window, and then choose **"Connection Info"**

Additionally, the DBID is also reported in the log file:

```
INFO  15.08.2014 10:24:42 Using DBID=postgresql
```

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 a property key this will be referred to as `[dbid]` in this chapter.

When using `WbSetConfig` you can use the value `[dbid]` inside the property name and it will get replaced with the **current** DBID automatically. The following command changes the property named `workbench.db.postgresql.ddlneedscommit` if the current connection is against a PostgreSQL database:

```
WbSetConfig workbench.db.[dbid].ddlneedscommit=true;
```

Alternatively the command `WbSetDbConfig` can be used, which only requires to specify the property name after the DBID, e.g. `WbSetDbConfig ddlneedscommit=true;`

38.2. GUI related settings

Showing accelerator in menu

Property: `workbench.gui.showmnemonics`

Possible values: `true`, `false`

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`

Controlling the type of print dialog

Property: `workbench.print.nativepagedialog`

Possible values: `true`, `false`

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`

38.3. Editor related settings

Define the default name for new tabs

Property: `workbench.gui.tabs.defaultlabel`

When adding a new editor tab, the value of this property will be used to set the new tab's title.

Include Oracle public synonyms in auto-completion of tables

Property: `workbench.editor.autocompletion.oracle.public_synonyms`

Possible values: `true`, `false`

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`

Set the modifier key for rectangular selections in the editor

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`

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

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)

38.4. Controlling code generation for code snippets

Controlling newlines in code snippets

Property: `workbench.clipcreate.includenewline`

Possible values: `true`, `false`

When creating a Copy 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 to not put any `\n` characters into the Java string.

Default: `true`

Controlling the concatenation character for code snippets

Property: `workbench.clipcreate.concat`

When creating a Copy 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: +

Controlling the prefix for code snippets

Property: `workbench.clipcreate.codeprefix`

When creating a Copy code snippet, this is prefixed with `String sql =` . With this property you can adjust this prefix.

Default: `String sql =`

Controlling termination character code snippets

Property: `workbench.clipcreate.codeend`

When creating a Copy code snippet, this character will be appended to the end of the generated code.

Default: `;`

Controlling escaping of embedded quotes

Property: `workbench.clipcreate.escape`

Possible values: `escape`, `duplicate`

When set to `escape`, embedded quotes will be escaped using a backslash, e.g. `"select * from \"some_table \" where id = 42"`.

When set to `duplicate`, embedded quotes will be escaped doubling them, e.g. `"select * from ""some_table"" where id = 42"`.

Default: `escape`

38.5. DbExplorer Settings

Switching the current database in the 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`

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:

```
.default=view,table,system view,system table
.postgresql=view,table,system view,system table,sequence
.rdb=view,table,system,system view
```

The values in this property are not case-sensitive (`TABLE` is the same as `table`)

Customizing the SELECT to be used for the data tab

You can customize the generated SELECT that is used to display the table data depending on the column type. Please refer to the DbExplorer chapter for details.

Customizing columns that can be searched

Property: `workbench.db.[dbid].datatypes.searchable`

DbExplorer's "Search table data" feature only includes columns with the datatypes `CHAR` and `VARCHAR` into the WHERE clause for searching.

Some database systems allow CLOB columns to be searched using a `LIKE` expression as well. This property can be used to list all datatypes that can be used in a `LIKE` condition.

Default values:

For PostgreSQL: `text`
For MySQL: `longtext,tinytext,mediumtext`

Changing the isolation level for the table list retrieval

Property: `workbench.db.[dbid].dbexplorer.use.read_uncommitted`

To avoid blocking of the table list retrieval, the isolation level used in the DbExplorer can be switched to `READ_UNCOMMITTED` for DBMS that support this. This is e.g. necessary for Microsoft SQL Server as an uncommitted DDL statement from a different connection can block the `SELECT` statement that retrieves the table information.

The isolation level will **only** be changed if Separate connection per tab is enabled.

For Microsoft SQL Server the timeout waiting for such a lock can be configured as an alternative.

Default values:

For Microsoft SQL Server: `true`

38.6. General configuration settings

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 command line.

No default

38.7. Database related settings

Include dependency information for "Object info"

Property: `workbench.db.objectinfo.includedeps`
`workbench.db.[dbid].objectinfo.includedeps`

If Object info is invoked, this setting controls if dependent objects (indexes, triggers) are also displayed for tables. This setting serves as a default for all DBMS. Displaying dependent objects can also be controlled on per DBMS by adding the DBID to the property key. The value without the DBID serves as a default setting for all DBMS.

Default: `false`

Include foreign key constraint information for "Object info"

Property: `workbench.db.objectinfo.includefk`
`workbench.db.[dbid].objectinfo.includefk`

If Object info is invoked, this setting controls if foreign key constraints are also displayed when dependent objects are displayed for tables. This setting serves as a default for all DBMS. When adding the DBID to the property key this is controlled on a per DBMS level.

Default: `false`

Automatically connect the DataPumper

Property: `workbench.datapumper.autoconnect`

When opening the DataPumper as a separate window 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`

Controlling COMMIT for DDL statements

Property `workbench.db.[dbid].ddlneedscommit`

Possible values: `true`, `false`

Defines if the DBMS supports transactional DDL (CREATE TABLE, DROP TABLE, ...)

Default: `false`

COMMIT/ROLLBACK behaviour

Property: `workbench.db.[dbid].usejdbccommit`

Possible values: `true`, `false`

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.

Default: `false`

Generating constraints for SQL source

Property: `workbench.db.[dbid].inlineconstraints`

Possible values: `true`, `false`

This setting controls the generation of the `CREATE TABLE` source in the DbExplorer. If a DBMS only supports defining primary and foreign keys inside the `CREATE TABLE` statement, then this property should be set to `true`.

Case sensitivity when comparing values

Property `workbench.db.[dbid].casesensitive`

Possible values: `true`, `false`

The search panel of the DbExplorer highlights matching values in the result tables. When using the "Server Side Search", the highlighter needs to know whether string comparisons in the database are case sensitive in order to highlight the correct values.

Default: `false`

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

Databases supporting the check for uncommitted changes

Property: `workbench.db.drivers.opentransaction.check`

A list of JDBC driver class names that map to databases that support checking for uncommitted changes. If one of these drivers is selected in a connection profile, the option Check for uncomitted changes will be visible in the connection dialog.

To make this option work, a query that counts the number of uncommitted changes needs to be configured as well.

Default:
`oracle.jdbc.driver.OracleDriver,oracle.jdbc.OracleDriver,org.postgresql.Driver,org.hsqldb.`

Query to check for uncommitted changes

Property: `workbench.db.[dbid].opentransaction.query`

A query that can be used to check if the current connection has any uncommitted transactions. The query is expected to return a single row with a single numeric column. If the value is zero, no uncommitted changes are detected. Any number greater than zero means that there is an uncommitted change.

Default: For Oracle, PostgreSQL and HSQLDB, the corresponding queries are configured

URL for online manual

Property: `workbench.db.[dbid].manual`

This defines the URL of the online manual for that DBMS. This URL is shown in the browser when using the menu item: Help » DBMS Manual will display the

You can append a version number after the DBID in the property key, to define different URLs for different DBMS versions. The key `workbench.db.microsoft_sql_server.8.manual` defines the URL for SQL Server 2000, whereas `workbench.db.microsoft_sql_server.10.5.manual` defines the URL for SQL Server 2008 R2. The numbers have to be `majorversion.minorversion` as shown in the "Connection Info" dialog

If the online manuals always have the version information at the same place of the URL, placeholders can be used, and only a single URL is necessary. For PostgreSQL, the following URL is used: `workbench.db.postgresql.manual=https://www.postgresql.org/docs/{0}.{1}/static/index.html`

Where `{0}` is replaced with the major version number and the `{0}` is replaced with the minor version number.

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.

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`

Use Savepoints to guard DML statement execution

Property: `workbench.db.[dbid].sql.usesavepoint`

Possible values: `true`, `false`

Some DBMS (such as PostgreSQL) cannot continue inside a transaction when an error occurs. A script with multiple DML 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 DML statement (`SELECT, INSERT`. In case of an error the savepoint will be rolled back and the transaction can continue.

Default value: `false`

Default value for PostgreSQL: `true`

Use Savepoints to guard DDL statement execution

Property: `workbench.db.[dbid].ddl.usesavepoint`

Possible values: `true`, `false`

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 rolled back and the transaction can continue.

Default value: `false`

Default value for PostgreSQL: `true`

Use Savepoints for `update/insert` mode for WbImport

Property: `workbench.db.[dbid].import.usesavepoint`

Possible values: `true`, `false`

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.

Note that enabling savepoints can drastically reduce the performance of the import.

Default value: `false`

Ignore errors during data retrieval

Property: `workbench.db.ignore.readerror`

Possible values: `true`, `false`

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`

Check read only columns

Property: `workbench.db.[dbid].resultset.columns.check.readonly`

Possible values: `true`, `false`

If this property is enabled, columns in result sets will be checked whether they are marked as read only by the JDBC driver. Read-only columns will not be included in generated DML statements when editing data. If the driver incorrectly reports columns that can be changed as read-only, setting this property to `false` will enable editing those columns.

Default value: `true`

Definining SQL commands that are not intercepted by SQL Workbench/J

Property: `workbench.db.[dbid].passthrough`

This is a list of SQL commands that are should not be handled by SQL Workbench/J

Default values:

```
workbench.db.mysql.passthrough=desc,describe
workbench.db.redshift.passthrough=desc,describe
```

Customizing data type mapping

Property: `workbench.db.[dbid].typemap`

When using the `-createTarget` parameter for WbCopy, the type mapping from the JDBC driver might not be sufficient or correct. With this setting you can define your own type mapping for a specific dbms. The entry is a list of mappings that map the numeric value of a JDBC datatype (as defined in java.sql.Types) to a real data type name for the *target* DBMS. The numeric JDBC datatype value and the DBMS specific datatype name are separated with a colon. Each pair is separated by a semicolon.

The following entry maps the JDBC datatype with the value 3 (DECIMAL) to the target datatype `double` and the value 2 (BIGINT) to the target type NUMBER. The NUMBER datatypes needs uses two parameter placeholders `$size` and `$digits`. The last mapping maps the JDBC value -1 (LONGVARCHAR) to the DBMS type `VARCHAR` using only the `$size` parameter

```
workbench.db.some_dbid.typemap=3:DOUBLE;2:NUMBER($size,$digits);-1:VARCHAR($size)
```

JDBC 4.0 defines the following constants:

- BIGINT = -5
- BINARY = -2
- BIT = -7
- BLOB = 2004
- BOOLEAN = 16
- CHAR = 1
- NCHAR = -15
- CLOB = 2005
- NCLOB = 2011
- DATE = 91
- DECIMAL = 3
- DOUBLE = 8
- FLOAT = 6
- INTEGER = 4
- LONGVARBINARY = -4
- LONGVARCHAR = -1
- LONGNVARCHAR = -16
- NUMERIC = 2
- REAL = 7
- SMALLINT = 5
- TIME = 92
- TIMESTAMP = 93
- TINYINT = -6
- VARBINARY = -3
- VARCHAR = 12
- NVARCHAR = -9
- ROWID = -8
- SQLXML = 2009

38.8. Configuring JDBC's clientInfo values

By default, SQL Workbench/J does not set values for the driver's clientInfo.

The properties to be passed to the driver, can be configured in `workbench.settings`. The property names depend on the JDBC driver.

Each property must be defined with a prefix based on the JDBC URL's "product" part which is second part of the URL. If the driver's URL is `jdbc:postgresql://localhost/workbench_db`, the product is `postgresql`.

The structure of the properties in `workbench.settings` is: `workbench.db.clientinfo.[jdbc-product].[driver property name]`.

The following definition sets a client info property named `ApplicationName` for PostgreSQL.

```
workbench.db.clientinfo.postgresql.ApplicationName=psql
```

The following placeholders are supported and will be replaced before the properties are passed to the driver:

- `${application}` - will be replaced with SQL Workbench/J
- `${version}` - will be replaced with the current build number
- `${osuser}` - will be replaced with the current oeprating system user (that is running SQL Workbench/J)
- `${connectionid}` - the SQL Workbench/J specific connection ID that can be used to identify the window or tab
- `${jdbcuser}` - the original database user as specified in the connection profile
- `${hostname}` - the hostname of the computer running SQL Workbench/J (or its IP address)

The following properties are predefined (but can be overwritten in `workbench.settings`

```
# client info for SAP HANA
workbench.db.clientinfo.sap.APPLICATION=${application}
workbench.db.clientinfo.sap.APPLICATIONSOURCE=${connectionid}
workbench.db.clientinfo.sap.APPLICATIONVERSION=${version}
workbench.db.clientinfo.sap.APPLICATIONUSER=${osuser}

# client info for DB2 for i (AS/400)
workbench.db.clientinfo.as400.ApplicationName=${application}
workbench.db.clientinfo.as400.ClientUser=${osuser}
workbench.db.clientinfo.as400.ClientHostname=${clienthost}
workbench.db.clientinfo.as400.ClientProgramID=${connectionid}
```

38.9. Configuring the check for the update table

Configuring the check process

Property: `workbench.db.updatetable.check.pkonly` (for all DBMS)

Property: `workbench.db.[dbid].updatetable.check.pkonly` (will overwrite the DBMS independent configuration)

Possible values: `true`, `false`

When changing values directly in the result set, SQL Workbench/J needs to find out which table is being edited. As this process requires multiple requests to the database server in order to support different features during editing this can be time consuming depending on the DBMS being used and the size of the database.

If this property is set to `true`, only the PK definition will be retrieved, otherwise the full definition of all columns of the table.

When this is enabled, editing results based on statements with multiple tables might not work properly. The option Highlight required fields will also have no effect as no column information will be retrieved for the table. It is also recommended to enable the option Highlight required fields to make sure the correct SQL statements are generated when only the PK information is checked.

Using unique indexes when no primary key is available

Property: `workbench.db.pk.retrieval.checkunique` (for all DBMS)

Property: `workbench.db.[dbid].pk.retrieval.checkunique` (will overwrite the DBMS independent configuration)

Possible values: `true`, `false`

This property controls the behaviour when no primary key is found when checking the update table. If this is set to true, SQL Workbench/J will use a unique index instead if available. Note that the check for the PK is still done during hte detection of the update table. Using a unique key is only a fallback.

Using the completion cache when checking the update table

Property: `workbench.db.updatetable.check.use.cache` (for all DBMS)

Property: `workbench.db.[dbid].updatetable.check.use.cache` (will overwrite the DBMS independent configuration)

Possible values: `true`, `false`

If this is set to `true`, retrieval of the table's columns, primary key (or unique index) information will be done using the completion cache. This can speed up repeated lookups for the same table(s).

The disadvantage is that when the table definitions are changed this would not be reflected in the cache and thus the PK information used or the generated SQL statements to save the changes might be wrong. It is recommend to enable Confirm result set updates to make sure the generated SQL statements are correct.

38.10. DBMS specific settings

38.10.1. Oracle specific settings

Support for Oracle materialized views

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`

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`

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`

Include tablespace information in the generated SQL source

Property: `workbench.db.oracle.retrieve_tablespace`

Possible values: `true`, `false`

If this is enabled, the generated SQL source for tables and indexes will contain the corresponding `TABLESPACE xxx` option to reflect the way the table was created. If this option should not be included in the SQL, set this parameter to `false`.

Default value: `true`

Check for the user's default tablespace

Property: `workbench.db.oracle.check_default_tablespace`

Possible values: `true`, `false`

When including the tablespace for an index or table, and this option is enabled, the tablespace for tables and indexes owned by the current user is only displayed if it is different from the default tablespace. For tables and indexes owned by other users, the tablespace will still be displayed even if it's the default tablespace of the owner.

Default value: `false`

38.10.2. Microsoft SQL Server specific settings

Define a lock timeout for the DbExplorer

Property: `workbench.db.microsoft_sql_server.dbexplorer.locktimeout`

Possible values: positive integer value. (Timeout in milliseconds)

This defines timeout that limits the time the driver should wait when hitting a read lock during retrieval of the table information. The timeout will be changed by running `SET LOCK_TIMEOUT ...` after the DbExplorer is opened.

The timeout will only be changed if Separate connection per tab is enabled.

As an alternative, the DbExplorer can be configured to change the isolation level to `READ UNCOMMITTED` to avoid the locks alltogether (but display potentially wrong information).

Default value: 2500

Microsoft SQL Server extended property for remarks

Property: `workbench.db.microsoft_sql_server.remarks.propertyname`

Defines the name of the extended property that is queried in order to retrieve table or column remarks for SQL Server.

SQL Workbench/J will use the table function fn_listextendedproperty to retrieve the extended property defined by this configuration setting to retrieve remarks.

Default value: `MS_DESCRIPTION`

Retrieving remarks for Microsoft SQL Server

Property:

```
workbench.db.microsoft_sql_server.remarks.object.retrieve
workbench.db.microsoft_sql_server.remarks.column.retrieve
```

Enables/disables the retrieval of extended properties as a replacement for the standard SQL `COMMENT ON ...` capability.

SQL Workbench/J will use SQL Server's fn_listextendedproperty table function to retrieve table or column remarks. As this can have a performance impact on the retrieval of tables or columns, this retrieval can be disabled using this configuration setting.

The name of the extended property can be configured using
`workbench.db.microsoft_sql_server.remarks.propertyname`

Enabling these options is also necessary in order to get comments in a WbSchemaReport output

Default value: `true` for both properties

38.11. SQL Execution related settings

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`

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.

Default:
`workbench.db.ignore.oracle=quit,exit,whenever,spool,rem,clear,break,btitle,column,change,r`

Controlling the format for execution time messages

Property: `workbench.log.timing.format`

Possible values:

- `millis` - always show the runtime in milliseconds, e.g. 1042ms
- `seconds` - always show the runtime in seconds including fractional values, e.g. 4.2s
- `dynamic` - display the values with hours, minutes, seconds but only displays parts that are relevant e.g. 10min 5s or 3h 5min 42s

Default: `dynamic`

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`

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.

Default: `oracle`

38.12. 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.

Controlling header lines in text exports

Property: `workbench.export.text.default.header`

Possible values: `true`, `false`

This property controls whether default value for the `-header` parameter of the WbExport command.

Default: `false`

Controlling XML export format

Property: `workbench.export.xml.default.verbose`

Possible values: `true`, `false`

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`

Setting default for WbImport's -continueOnError parameter

Property: `workbench.import.default.continue`

Possible values: `true`, `false`

This property controls the default value for the parameter `-continueOnError` of the WbImport command.

Default: `false`

Setting a default for WbImport's -header parameter

Property: `workbench.import.default.header`

Possible values: `true`, `false`

This property controls the default value for the parameter `-header` of the WbImport command.

Default: `true`

Setting a default for WbImport's -multiLine parameter

Property: `workbench.import.default.multilinerecord`

Possible values: `true`, `false`

This property controls the default value for the parameter `-multiLine` of the WbImport command.

Default: `false`

Setting a default for WbImport's -trimValues parameter

Property: `workbench.import.default.trimvalues`

Possible values: `true`, `false`

This property controls the default value for the parameter `-trimValues` of the WbImport command.

Default: `false`

38.13. Controlling the log file

When SQL Workbench/J initializes the logging environment, it also adds two system property that can be used to define the logfile relative to the configuration or the installation directory:

- `workbench.config.dir` contains the full path to the configuration directory
- `workbench.install.dir` contains the full path to the directory where sqlworkbench.jar is located

These properties can be used to put the logfile into the directory relative to the config or installation directory without the need to hardcode the directory name.

38.13.1. Configure internal logging

Log file location

Property: `workbench.log.filename`

Defines the location of the logfile. By default, the file will be named `workbench.log` and will be written into the configuration directory.

Log level

Property: `workbench.log.level`

Set the log level for the log file. Valid values are

- DEBUG
- INFO
- WARN
- ERROR

Default: `INFO`

Log format

Property: `workbench.log.format`

Define the elements that are included in log messages. The following placeholders are supported:

- {type}
- {timestamp}
- {message}
- {error}
- {source}
- {stacktrace}

This property does not define the layout of the message, only the elements that are logged.

If the log level is set to DEBUG, the stacktrace will always be displayed even if it is not included in the format string.

If you want more control over the log file and the format of the message, please switch the logging to use Log4J.

Default: `{type} {timestamp} {message} {error}`

Logging to the console

Property: `workbench.log.console`

Defines whether SQL Workbench/J logs messages additionally to the standard error output

Default: `false`

Maximum logfile size

Property: `workbench.log.maxfilesize`

Defines the maximum size of the logfile in bytes. If the size is exceeded a new logfile is created during the next startup.

Default: `10485760` (1MB)

Maximum number of logfiles to keep

Property: `workbench.log.backup.count`

Defines the maximum number of logfiles to be kept after a new logfile is created. The old logfiles will be renamed with a number (`workbench.log.1` being the oldest logfile)

Default: `5`

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) pre-configured.

Default: `false`

Logging of variable substitution

Property: `workbench.sql.parameter.log.substitution`

If this property is set to `true` the result of replacing them in the SQL statements will be written to the logfile if the loglevel is DEBUG.

Default: `false`

38.14. Configure Log4j logging

38.14.1. Turn on Log4j logging

Property: `workbench.log.log4j`

If you need more control over the logfile (e.g. for batch processing) you can delegate logging to Log4j. You can turn on Log4j logging in two different ways:

- The value of the property is `true`
- The value of the property points to an existing file

If you just pass `true` as the value for this property, the Log4j configuration file must be accessible to Log4J in the classpath. If you specify a configuration file, this will be "passed" to Log4j by setting the system property `log4j.configurationFile` to contain the correct "file URL" needed by Log4j.

When passing a configuration file through this property, you can use a system property as part of the filename (e.g. `${user.home}/sqlworkbench.log`). If the filename denotes a relative filename (e.g. `log4j.xml` without any path information), then it is assumed to be relative to the configuration directory.

When you turn on Log4j logging, you must copy copy the Log4j 2.x libraries as `log4j-api.jar` and `log4j-core.jar` into the `ext` directory. Do not include the version number in the filename. These must be jar files for Log4J 2.x

If the Log4j classes are not found, the built-in logging will be used (see above).

When Log4j logging is enabled, none of the logging properties described in the previous section will be used. You have to configure everything through `log4j.xml`.

When using Help » Show log file with Log4J enabled, and you have configured Log4J to write to multiple files, only the first file will be shown.

When SQL Workbench/J initializes the logging environment, it also adds two system property that can be used to define the logfile relative to the configuration or the installation directory:

- `workbench.config.dir` contains the full path to the configuration directory
- `workbench.install.dir` contains the full path to the directory where sqlworkbench.jar is located

These properties can be used to put the logfile into the directory relative to the config or installation directory without the need to hardcode the directory name in `log4j.xml`

A sample `log4j.xml` can be found in the `scripts` directory of the SQL Workbench/J distribution.

The system properties that are set by SQL Workbench/J to point to the configuration and installation directory (see above) can also be used in the `log4j.xml` file using the `sys` prefix e.g. `${sys:workbench.config.dir}`

38.15. Configuring the logfile viewer

Property: `workbench.logfile.viewer.program`

This property controls which application is used to display the logfile when using Help » Show log file.

The possible values for this property are:

- `internal` - this is the default and uses the built-in logviewer
- `system` - this will use the tool registered in the operating system to open files with the extension `.log`
- a path to an existing application - if the value denotes an existing filename, it is assumed that this is an application and accepts a filename as a command line parameter

38.16. Configuring SQL keywords

The built-in list of keywords, reserved keywords and functions that are used for the syntax highlighting in the editor and for detecting if a word needs to be quoted can be extended. The keywords are configured in text files stored in the configuration directory.

The keywords can be configured through the following files:

- Reserved keywords (which require quoting) - `reserved_words.wb`
- Keywords - `keywords.wb`
- Data types - `datatypes.wb`

- Functions - `functions.wb`
- Operators - `operators.wb`

The editor will highlight reserved keywords and non-reserved keywords the same way. The distinction is necessary to properly quote reserved keywords when generating SQL statements.

Files without a prefix in their name (as shown above) are used for all DBMS. To configure the keywords for a specific DBMS, prefix the filename with the DBID, e.g. to extend the list of keywords for PostgreSQL, they have to be stored in a file named `postgresql.keywords.wb`

38.17. Settings related to SQL statement generation

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 sensitive.

Note that for Oracle, tables that are owned by the current user will never be prefixed with the owner.

Default values:

```
.oracle=PUBLIC
.postgresql=public
.rdb=*
```

Defining CREATE TABLE templates for WbCopy

Property: `workbench.db.[dbid].create.table.[typename]`

This defines a complete `CREATE TABLE` statement that is used by `WbCopy` to create the target table. The `typename` value is the value that has to be used for the `-tableType` parameter of the `WbCopy` command.

The following placeholders are supported in the template

`%fq_table_name%` replaced with the fully qualified table name
`%table_name%` replaced with the specified table name (without schema or catalog)
`%columnlist%` replaced with the column definitions (for all columns)
`%pk_definition%` replaced with the primary key definition.

The placeholder `%pk_definition%` can be used if the DBMS does not support defining a primary key using an ALTER TABLE on the created table. If this placeholder is present in the template and the table has a primary key, the placeholder will replaced with an appropriate `PRIMARY KEY (col1, ...)` expression. Note that the template must **not** contain the needed comma for the `PRIMARY KEY`. The comma will be added by SQL Workbench/J if a primary key is defined. If the table has no primary key, the placeholder will automatically be removed.

Default values:

```
.postgresql.create.table.temp=CREATE LOCAL TEMPORARY TABLE %fq_table_name%
( %columnlist% ) ON COMMIT DROP
```

```
.oracle.create.table.globaltemp=CREATE GLOBAL TEMPORARY TABLE %fq_table_name%
( %columnlist% ) ON COMMIT DELETE ROWS
.h2.create.table.temp=CREATE LOCAL TEMPORARY TABLE %fq_table_name% ( %columnlist
% )
.informix_dynamic_server.create.table.temp_nolog=CREATE TEMP TABLE
%fq_table_name% ( %columnlist% %pk_definition% ) WITH NO LOG
```

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 being 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`

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`

38.18. Customize table source retrieval

SQL Workbench/J re-generates the source of a table based on the information about the table's metadata returned by the driver. In some cases the driver might not return the correct information, or not all the information that is necessary to build the correct syntax for the DBMS. In those cases, a SQL query can be configured that can use the built-in functionality of the DBMS to return a DDL statement to re-create the table.

This DBMS specific retrieval of the table source is defined by two properties in `workbench.settings`.

Defining the SQL statement

Property: `workbench.db.[dbid].retrieve.create.table.query`

This property defines the SQL query that retrieves the DDL for the table. It must be a statement that returns a result set. The statement may contain the following placeholders:

`%catalog%` the catalog in which the table is defined
`%schema%` the schema in which the table is defined
`%table_name%` the name of table
`%fq_index_name%` the fully qualified name of the table (including catalog and schema)

If the SQL returned by the DBMS includes the indexes defined for the table, the property:
`workbench.db.[dbid].retrieve.create.table.index_included` has to be set to `true`. Otherwise SQL Workbench/J will generate the DDL statement to re-create the indexes.

If the SQL returned by the DBMS includes table and column comments, the property: `workbench.db.[dbid].retrieve.create.table.comments_included` has to be set to `true`. Otherwise SQL Workbench/J will generate table and column comments.

Defining the result column

Property: `workbench.db.[dbid].retrieve.create.table.sourcecol`

By default the source code is assumed to be in the first column of the result. If that is not the case this property can be used to define the column index of the result in which the table's source is available. The first column has the index 1.

The following example configures a SQL statement to retrieve the table source using MySQL's `SHOW CREATE TABLE`:

```
workbench.db.mysql.retrieve.create.table.query=show create table %fq_table_name%
workbench.db.mysql.retrieve.create.table.sourcecol=2
workbench.db.mysql.retrieve.create.table.index_included=true
```

Using use Oracle's `DBMS_METADATA` to retrieve the table source, is controlled through an Oracle specific configuration property.

38.19. Customize index source retrieval

SQL Workbench/J re-generates the source of an index based on the information about the table's metadata returned by the driver. In some cases the driver might not return the correct information, or not all the information that is necessary to build the correct syntax for the DBMS. In those cases, a SQL query can be configured that can use the built-in functionality of the DBMS to return the DDL to recreate the index.

This DBMS specific retrieval of the index source is defined by two properties in `workbench.settings`.

Defining the SQL statement

Property: `workbench.db.[dbid].retrieve.create.index.query`

This property defines the SQL query that should be executed to retrieve the DDL to re-create the index. It must be a statement that returns a result set. The statement may contain the following placeholders:

`%catalog%` the catalog in which the index is defined
`%schema%` the schema in which the index is defined
`%indexname%` the name of the index
`%fq_index_name%` the fully qualified name of the index (including catalog and schema)
`%table_name%` the name of table on which the index is defined, including the catalog or schema if necessary
`%simple_table_name%` the name of table on which the index is defined without the catalog or schema.

Defining the result column

Property: `workbench.db.[dbid].retrieve.create.index.sourcecol`

By default the source code is assumed to be in the first column of the result. If that is not the case this property can be used to define the column index of the result in which the table's source is available. The first column has the index 1.

If an error occurs during retrieval, SQL Workbench/J will revert to the built-in table source generation.

The following example configures the use of the function `pg_get_indexdef()` to be used

```
workbench.db.postgresql.retrieve.create.index.query=select pg_get_indexdef('%fq_index_name
```

Using Oracle's `DBMS_METADATA` to retrieve the index source, is controlled through an Oracle specific configuration property.

38.20. Filter settings

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 drop down.

Default value: `15`

39. Simple properties based profile storage

The default file format for saving connection profiles is XML, however when using SQL Workbench/J in batch mode or as a console application editing the XML format is tedious. Therefor it is possible to store the profiles in a "plain" properties file.



The properties file **must** have the extension `.properties`, otherwise it will not treated as a properties file by SQL Workbench/J

The properties file can contain multiple profiles, each property key has to start with the prefix `profile`. The second element of the key is a unique identifier for the profile that is used to combine the keys for one profile together. This identifier can be any combination of digits and characters. The identifier is **case sensitive**.

The last element of the key is the actual profile property.

A minimal definition of a profile in a properties file, could look like this:

```
profile.042.name=Local Postgres
profile.042.driverclass=org.postgresql.Driver
profile.042.url=jdbc:postgresql://localhost/postgres
profile.042.username=arthur
profile.042.password=dent
profile.042.driverjar=postgresql-9.4-1203.jdbc41.jar
```

In the above example the identifier `042` is used. The actual value is irrelevant. It is only important that all properties for one profile have the same identifier. You can also use any other combination of digits and characters.

For each profile the following properties can be defined. The property name listed in the following table is the last element for each key in the properties file.

Key	Value
`name`	This defines the name of the connection profile which can e.g. be used with the `-profile` command line parameter. This parameter is mandatory.
`url`	This defines the JDBC URL for the connection This parameter is mandatory.
`username`	This defines the username that should be used to connect to the database This parameter is mandatory.
`password`	This defines the password that should be used to connect to the database This parameter is mandatory.
`drivername`	This defines the named JDBC driver as registered within SQL Workbench/J. If this is specified the corresponding driver needs to be defined and available in the `WbDrivers.xml` file. Either this parameter or `driverjar` is mandatory.
`driverjar`	This specifies the jar file that contains the JDBC driver. If `driverjar` is specified `drivername` is ignored.

Key	Value
	If the filename is not specified as an absolute file, it is assumed to be relative to the location of the properties file. Either this parameter or `drivername` is mandatory. If `driverjar` is specified, `driverclass` must also be specified. Defining the driver jar in this way is not supported when running in GUI mode. Drivers managed through the GUI will always be saved in `WbDrivers.xml`
`driverclass`	This specifies JDBC driver's class name. This parameter is required if `driverjar` is specified.
`autocommit`	Defines the autocommit behaviour of the connection. This defaults to `false`
`fetchsize`	Defines the fetchsize attribute of the connection.

Index

A

Amazon Redshift
- ERROR: type "e" does not exist, 210

B

Batch files
- connecting, 85
- defining variables, 88
- setting SQL Workbench/J configuration properties, 89
- specify SQL script, 85
- starting SQL Workbench/J, 85

C

Clipboard
- export result to, 60
- import data from, 62

Command line
- connection profile, 21
- JDBC connection, 21
- parameters, 19

Configuration
- advanced configuration properties, 241
- change advanced configuration properties, 188
- configure SQL keywords, 259
- JDBC driver, 24

Connection profile, 26
- autocommit, 27
- Configure SSH tunnel, 34
- connection URL, 27
- create, 26
- default fetch size, 27
- delete, 26
- extended properties, 28
- separate connection, 29
- separate session, 29
- timeout, 28

Customize
- DbExplorer DDL generation, 261

D

DB2
- Column comments not displayed, 210
- Connection closed, 209
- Problems, 209
- Table comments not displayed, 210

DbExplorer
- customize DDL generation, 261
- prevent locking, 245
- show all triggers, 229

DbTree
- macros, 71

DDL
Execute DDL statements, 46
DML
select values for foreign key columns, 74

E

Editing data
deleting rows, 58
deleting rows which are referenced through a foreign key, 75
select values for foreign key columns, 74
Editor
expanding text clips, 71
Excel export
corrupt file, 204
installation, 96, 204
Export
clipboard, 60
compress, 110
Excel, 108
HTML, 110
JSON, 110
memory problems, 96
OpenOffice, 108
parameters, 97
result set, 60
Spreadsheet, 108
SQL INSERT script, 106
SQL query result, 96
SQL UPDATE script, 106
table, 96
text files, 103
XML files, 105

F

FlatLaf
Configure, 239
Theme, 239
Foreign keys
editing values of foreign key columns, 74
Update foreign key columns, 74

I

Import
clipboard, 62, 62
csv, 114
Excel, 114
flat files, 114
OpenOffice, 114
parameters, 114
result set, 61
tab separated, 114
XML, 114
XSLT, 114

J

Java runtime
  Java not found on Windows, 15
JDBC Connection
  connection properties, 28
JDBC driver
  class name, 24
  configure client info, 251
  jar file, 24
  library, 24
  license file, 24
  sample URL, 24

L

Liquibase
  Run SQL from Liquibase file, 178
Look and Feel
  FlatLaf Theme, 239

M

Memory problems
  WbCopy or WbExport using a lot of memory with PostgreSQL, 210
  WbCopy or WbExport using a lot of memory with SQL Server, 208
Microsoft SQL Server
  Incorrect value for DATE columns, 207
  JDBC URL properties, 209
  Locking problems, 208
  lock timeout for DbExplorer, 253
  prevent locking in DbExplorer, 245
  Problems, 207
  Problem when running SHOWPLAN_ALL, 208
  Sequence increments twice, 209
  WbCopy memory problem, 208
  WbExport memory problem, 208
  Windows authentication, 208
MySQL
  display table comments in DbExplorer, 207
  problems, 207

O

Options dialog
  dialog too small, 204
Oracle
  autotrace, 78
  check for pending transactions, 77
  connection closed, 206
  database comments, 205
  DATE datatype, 226
  DBMS_METADATA, 79
  dbms_output, 79
  missing mandatory parameter, 206
  No views displayed in DbExplorer, 205
  Problems, 205
  show system information, 78
  tablespace information, 253, 253

Tables with underscores not treated correctly, 205

P

PostgreSQL
.pgpass, 76
check for pending transactions, 76
COPY, 76
libpq, 28
pgpass, 28
pgpass.conf, 76
Problems, 210
WbCopy memory problem, 210
WbExport memory problem, 210
Problems
Amazon Redshift, 210
Cipher not available, 35
create stored procedure, 203
create trigger, 203
dialog too small, 204
driver not found, 203
Excel export not possible, 204
Excel files can't be opened, 204
IBM DB2, 209
memory usage during export, 96
Microsoft SQL Server, 207
MySQL, 207
Oracle, 205
out of memory, 204
PostgreSQL, 210
Sybase SQL Anywhere, 210
timestamp with timezone, 203
timezone, 203
wrong DDL, 203
wrong index source, 203
wrong table source, 203
Profile dialog
dialog too small, 204

S

SQL
change the statement delimiter, 46
configure SQL keywords, 259
SSH Host
Setup, 237
SSH Tunnel
Setup, 34
Starting
Java runtime not found on Windows, 15
Statement delimiter
change the statement delimiter, 46
Stored procedures
create stored procedure, 46

T

Triggers

create trigger, 46
show all triggers in DbExplorer, 229

V

Variables
define on command line, 20
define through SQL query, 81
definition, 81
editing, 82
load from file, 82
prompting, 83
use in batch files, 88
using, 82

W

Windows
Java not found, 15
using the launcher, 15