
DbVisualizer 7.0

Users Guide



Copyright DbVis Software AB

<http://www.dbvis.com>

Table of Contents

Getting Started and General Overview.....	8
Introduction	8
Installing.....	8
Installation structure.....	8
Special Properties.....	8
Install license key for DbVisualizer Personal.....	9
Uninstalling the license key.....	9
Useful Resources.....	10
Starting DbVisualizer.....	10
Command line arguments.....	10
Pure command line interface.....	10
The Main Window and Common Components.....	11
Standard Components in the User Interface.....	12
Grid, Graph and Chart.....	12
Context Sensitive Components.....	13
Tooltips.....	13
Grids.....	14
Sorting.....	14
Right-click menu.....	15
Aggregation Data for Selection.....	16
Column Visibility.....	17
Auto Resize.....	17
Quick Filter.....	18
Print.....	19
Printer Setup.....	19
Grid, Chart and Plain Text.....	19
Graph.....	19
Print Preview.....	20
Checking for Updates.....	21
Problem resolution.....	21
Debugging DbVisualizer.....	22
How to satisfy the DbVisualizer support team.....	22
Load JDBC Driver and Get Connected.....	24
Introduction	24
What is a JDBC Driver?.....	24
Get the JDBC driver file(s).....	24
Connection Wizard.....	25
Driver Manager.....	30
JDBC Driver Finder.....	30
Loading and Configuring Drivers Manually.....	32
Setup a JDBC driver.....	33
JDBC drivers that requires several JAR or ZIP files.....	34
The JDBC-ODBC bridge.....	34
Loading JNDI Initial Contexts.....	34
Errors (why are some paths red?).....	35
Several versions of the same driver.....	35
Setup a database connection.....	35
Setup using JDBC driver.....	36
Setup using JNDI lookup.....	37
Connection Properties.....	37
Database Profile.....	39
Driver Properties.....	40
Driver Properties for JDBC Driver.....	40
Driver Properties for JNDI Lookup.....	41
Always ask for userid and/or password.....	42
Using variables in the Connection details.....	43
Connect to the Database.....	44
Connections Overview.....	46
Database Objects Explorer.....	47
Introduction.....	47
Create a Database Connection.....	48
Database Connection object.....	48
Alias.....	48
Default database and schema.....	48
Remove and copy database connection objects.....	49
Database Connection detailed information.....	49
Search.....	49
Organizing Database Connections in Folders.....	50
Connections overview.....	51
Database Objects Tree.....	52
Standard Actions.....	52
Object Actions.....	53

Common Object Actions.....	54
Create Table.....	54
Create Index.....	54
Import Table Data.....	54
Export Table.....	54
Script Object to SQL Editor.....	55
Script Object to New SQL Editor.....	55
Objects Tree Filtering.....	55
Show Table Row Count.....	56
Object Tree Icons.....	56
Database Profiles.....	57
Database Specific Support.....	57
Generic profile.....	57
Catalog/Database object.....	58
Schema object.....	59
Table object.....	60
Procedure object.....	61
Object Views.....	62
Grid.....	62
Form.....	63
Source.....	64
Table Row Count.....	65
Table Data.....	66
Right-click menu.....	66
Where Filter.....	67
Quick Filter.....	69
Monitor row count.....	71
Editing.....	71
DDL Viewer.....	71
References.....	72
Navigator.....	73
Procedure Editor.....	73

SQL Commander.....75

Introduction.....	75
Physical Database Connections and Transactions.....	76
Editor.....	76
Database Connection, Catalog and Schema.....	77
Limiting Result Set size (Max Rows/Chars).....	78
Load from and save to file.....	79
Load Recent.....	79
Quick File Open.....	80
Editor Preferences.....	80
Multiple editors.....	81
Permissions.....	81
Charsets and Fonts.....	81
Key Bindings.....	82
Client-Side Comments.....	83
Auto Completion.....	83
SQL Formatter.....	86
History.....	87
Bookmarks.....	87
Execution.....	87
SQL->Execute.....	88
SQL->Execute Current.....	88
SQL->Execute Buffer.....	88
SQL->Execute Explain Plan (Oracle, SQL Server and DB2).....	88
Auto Commit, Commit and Rollback.....	91
SQL Scripts.....	91
Execute Large SQL Scripts.....	92
Anonymous SQL Blocks.....	93
Stored Procedures.....	93
Client Side Commands.....	93
@run - run SQL script from file.....	94
@cd <directory> - change directory.....	94
@export - export result sets to file.....	94
Example 1: @export with minimum setup.....	96
Example 2: @export with automatic table name to file name mapping.....	96
Example 3: @export all result sets into a single file.....	96
Example 4: @export using predefined settings.....	97
@delimiter - Temporarily change the statement delimiter.....	97
@call - Execute a function or stored procedure.....	97
@echo - Echo text.....	98
>window iconify - Iconify the main window.....	98
>window restore - Raise the main window.....	98
@desc table - Describe the columns in table.....	98
@ddl - Generate DDL command.....	98

@spool log - Save log to file.....	98
@stop on error - Stop execution if any error occurs.....	99
@stop on warning - Stop execution if any warning occurs.....	99
@set autocommit - Sets the auto commit state.....	99
@commit - Commits the current transaction.....	99
@rollback - Rollback the current transaction.....	99
@set serveroutput - Enable/disable the DBMS output management for Oracle.....	99
Variables.....	99
Variable Syntax.....	99
Pre-defined Variables.....	100
Variable Substitution in SQL statements.....	100
Parameter Markers.....	103
Output View.....	104
Log.....	104
Log controls.....	105
Auto clear log.....	105
Result Set.....	105
Result set menu.....	107
Editing.....	108
Multiple result sets produced by a single SQL statement.....	108
Text.....	110
Chart.....	110
DBMS Output (Oracle).....	110
Query Builder.....	112
Introduction.....	112
Current Limitations.....	113
Creating a Query.....	113
Adding Tables.....	114
Joining Tables.....	115
Manually Joining Tables.....	115
Joining Tables Automatically.....	115
Join Properties.....	116
Remove Tables and Joins.....	117
Query Details.....	117
Columns.....	118
Conditions.....	119
Grouping.....	120
Sorting.....	120
SQL Preview.....	120
Testing the Query.....	120
Loading a Query from the SQL Editor.....	121
Properties controlling Query Builder.....	121
Express joins as JOIN clause or WHERE condition.....	121
Table and Column Name qualifiers.....	122
Delimited Identifiers.....	122
Drag style and Diagram Size.....	122
Bookmarks and History.....	123
Introduction.....	123
Bookmarks.....	123
Creating, Editing and Organizing Bookmarks.....	124
Executing Bookmarks.....	124
Adding a Bookmark as a Favorite.....	125
Sharing Bookmarks.....	125
History.....	125
Reusing a History Entry.....	126
Saving a History Entry as a Bookmark.....	126
Quick Load.....	126
Monitor and Charts.....	128
Introduction.....	128
Monitored SQL Statements.....	129
Creating, Editing and Organizing Monitored Statements.....	130
Monitor table row count.....	130
Monitor table row count difference.....	131
Monitor Window.....	132
Charts.....	133
Chart Controls.....	134
Data.....	134
Layout.....	134
Chart View.....	136
Zooming.....	136
Rotating.....	136
Export.....	137
Create and Alter Table.....	138
Introduction.....	138
Create Table.....	138

Columns tab.....	139
Primary Key tab.....	141
Foreign Keys tab.....	142
Unique Constraints tab (database-specific).....	143
Check Constraints tab (database-specific).....	143
Indexes tab (MySQL only).....	144
SQL Preview.....	145
Execute.....	145
Alter Table.....	145
Edit Table Data.....	147
Introduction.....	147
Features that support editing.....	147
Update and Delete must match one table row.....	147
Edit Multiple Rows.....	147
Data Type checking.....	148
New Line and Carriage Return.....	148
Grid Editor.....	149
Insert row.....	149
Update row.....	150
Delete row(s).....	150
Duplicate row(s).....	150
Copy/Paste.....	150
Paste data from Excel and OpenOffice.....	150
Insert pre-defined values (Set Selected Cells).....	151
Undo Edit(s).....	152
Key Column(s) Chooser.....	152
Preview Changes.....	153
Saving Changes.....	153
Transaction Control.....	153
Permissions.....	154
Errors.....	154
Form Editor/Viewer.....	154
Cell Editor/Viewer.....	155
Binary/BLOB.....	156
Image Viewers.....	157
XML Viewer.....	157
Serialized Java Objects Viewer.....	158
Hex Viewer.....	158
Large text data/CLOB.....	159
Import from File.....	160
Export to File.....	160
Table Data Navigation.....	161
Introduction.....	161
Data Navigation.....	162
Adding Context Information to the Graph.....	164
Arranging the Graph.....	165
Exporting and Printing the Graph.....	166
Procedure Editor.....	167
Introduction.....	167
Create Procedure.....	167
Edit and Compile.....	169
Execute in SQL Commander.....	171
Script CALL to Editor.....	173
Tool Properties.....	174
Customizing DbVisualizer.....	174
The user preferences (XML) files.....	174
Export Settings.....	174
Import Settings.....	175
General Settings.....	176
Appearance.....	177
Fonts.....	179
Key Bindings.....	179
Database Connection.....	181
Driver Manager.....	182
Permissions.....	182
SQL Commander Permissions.....	182
Table Data Editing Permissions.....	183
Time Zone.....	183
File Encoding.....	183
Data Formats.....	184
Date, Time and Timestamp formats.....	184
Number formats.....	185
Table Data.....	185
Transaction.....	185
Scripts.....	185

Monitor.....	185
Form Viewer.....	185
Grid.....	186
Copy.....	186
Colors.....	186
Binary/BLOB and CLOB Data.....	186
SQL Editor.....	187
Statement Delimiters.....	187
SQL Formatting.....	187
Auto Completion.....	187
Comments.....	188
Variables.....	188
Proxy Settings.....	188
Database Settings.....	189
Authentication.....	190
Delimited Identifiers.....	190
Qualifiers.....	190
Physical Connection.....	190
Transaction.....	191
SQL Statements.....	191
Connection Hooks.....	192
Objects Tree Labels.....	192
SQL Editor.....	193
Query Builder.....	194
Database Specific settings.....	194
Data Types (Oracle).....	194
Data Types (DB2 and JavaDB/Derby).....	194
Explain Plan (Oracle, SQL Server and DB2).....	194
Explain Plan (Oracle).....	195
Explain Plan (DB2).....	195
Objects Tree (Oracle).....	195
Export and Import.....	196
Introduction.....	196
Export Schema.....	196
Output Format.....	197
Output Destination.....	199
Object Types.....	199
Options.....	199
Settings.....	200
Logging.....	200
Export Table.....	200
Export Grid data.....	201
Settings page.....	201
Output Format.....	202
Encoding.....	203
Data Format.....	203
Quote Text Data.....	203
Options.....	204
CSV.....	204
HTML.....	204
SQL.....	204
XML.....	205
XLS.....	205
Settings.....	205
Data page.....	205
Generate Test Data.....	206
Test data generator example.....	207
Preview.....	209
Output Destination.....	209
Export Text data.....	210
Export Graph data.....	210
Export Chart data.....	211
Import Table Data.....	211
Source File.....	212
Settings.....	213
Data Formats.....	215
Import Destination.....	217
Import Process.....	219
Exporting and Importing Binary/BLOB and CLOB Data.....	220
Exporting Binary/BLOB and CLOB Data.....	220
Importing Binary/BLOB and CLOB Data.....	221
Using Variables and Exporting to Multiple Files.....	221
Database Profile Framework.....	222
Introduction.....	222
What features in DbVisualizer relies on the database profile?.....	222
How does DbVisualizer know what database profile to use?.....	223

XML structure.....	224
XML skeleton.....	225
<DatabaseProfile>.....	226
<InitCommands> - Initialization commands.....	227
<Commands> - The SQLs used to interact with the database.....	228
<Command>.....	228
Result set.....	228
<Input> - Setting command input.....	229
<Output> - Redefine command output.....	230
<ObjectsTreeDef> - Definition of the Database Objects Tree.....	231
<GroupNode> - Static objects used for grouping.....	232
<DataNode> - Dynamic objects created via SQL.....	232
<Command>.....	233
<Filter>.....	233
<SetVar>.....	234
<ObjectsViewDef> - Definition of the Object Views.....	235
<ObjectView>.....	236
<DataView>.....	236
Viewers.....	237
grid.....	237
text.....	239
form.....	240
node-form.....	241
table-refs.....	242
tables-refs.....	243
table-data.....	244
table-rowcount.....	245
<Command>.....	246
<Message>.....	246
Extending ObjectView.....	246
<ObjectsActionDef> - Definition of user actions.....	247
Variables.....	248
<ActionGroup>.....	250
<Action>.....	250
<Input>.....	251
text (single line).....	252
text-editor (multi line).....	253
number.....	253
password.....	253
list (large number of choices).....	253
radio (limited number of choices).....	254
check (true/false, on/off, selected/unselected).....	254
separator (visual divider between input controls).....	254
grid (configurable multi row inputs).....	255
<SetVar>.....	257
<Confirm>.....	257
<Result>.....	257
<Command>.....	258
Conditional processing.....	258
When are conditional expressions processed?.....	258
Conditional processing when database connection is established.....	258
Conditional processing during command execution.....	259
Current limitations.....	260

Getting Started and General Overview

Introduction

DbVisualizer is a feature rich, intuitive multi-database tool for developers and database administrators, providing a single powerful interface across a wide variety of operating systems. With its easy-to-use and clean interface, DbVisualizer has proven to be one of the most cost effective database tools available, yet to mention that it runs on all major operating systems and supports all major RDBMS that are available. Users only need to learn and master one application. DbVisualizer integrates transparently with the operating system being used.

This document gives a overview, installation tips and general information about the product.

The screenshots throughout the users guide are produced on Windows XP using the Windows Look and Feel, but DbVisualizer lets you choose among other Look and Feels as well.

All documents in the Users Guide are primarily focusing on the DbVisualizer Personal edition. Some of the described features are not available in the Free edition.

Installing

Installing DbVisualizer is no different then installing other modern products. The standard installation procedure is performed using a graphical application, and you just need to click through the questions that are displayed. Follow the [instructions](#) at the DbVisualizer web site if you need information on how to start the installation procedure specifically for your platform.

Installation structure

The installer and launcher for DbVisualizer is based on the **install4jTM** product (<http://www.install4j.com>). The structure of the installation directory (referred as **DBVIS-HOME** throughout the users guide) contains the following. (The exact content may differ between platforms):

```
.install4j/  
doc/  
jdbc/  
lib/  
resources/  
wrapper/  
dbvis.voptions  
dbvis.exe  
README.txt  
uninstall.exe
```

The **dbvis.exe** file is used to start DbVisualizer. The remaining files and directories are only of interest if you need to do nonstandard customization. For information on how to increase the memory for the Java process that runs DbVisualizer, and also on how to modify the Java version being used, please read the online [FAQ](#) for the latest information.

Special Properties

DbVisualizer utilizes a few special properties that you can use to modify characteristics of the application. These properties are available in the **DBVIS-HOME/resources/dbvis-custom.prefs** file.

You rarely need to modify these properties, as the default values are sufficient for most usage.

The following are the properties handled by DbVisualizer:

Property	Description
----------	-------------

dbvis.disabledataedit=false	Specifies if table data editing should be completely disabled, i.e. the form and inline editors. Note: This has an effect only when used with a licensed edition.
dbvis.driver.ignore.dir=lib:resources:.install4j	Specify directories from DBVIS-HOME that should not be listed in the Driver Manager "System Classpath" list. Directories are separated with ",". Accepted values: one or several directory names starting from DBVIS-HOME.
dbvis.grid.encode=false	Specifies if encoding of data in result set grids will be performed or not. If set to true then make sure the dbvis.grid.fromEncode and/or dbvis.grid.toEncode are also set.
dbvis.grid.fromEncode=ISO8859_1	Encoding used when translating text data that is fetched from the database
dbvis.grid.toEncode=GBK	Encoding used when translating data that will appear in the result set grid
dbvis.removepartialresultsets=false	Defines whether the result set(s) should be removed when interrupting an ongoing execution in the SQL Commander.
dbvis.savedatacolumns=false	Column layout changes such as reordering and/or visibility are saved for all grids in the Objects Views *except* for the "Data" grid. This property can be used to also include the layout in the "Data" grid. Note: This will result in DbVisualizer saving the layout for each table that is displayed in the Data grid = huge XML file...
dbvis.showactionresult=false	This defines whether the result for all actions should be displayed or only failures (default).
dbvis.sqlwarning.maxrows=5000	Defines the number of SQL Warning rows that should be processed before truncating.
dbvis.usegetobject=false	Specifies if the generic ResultSet.getObject() method in JDBC will be used in favor of the data type specific get methods or not. Default is false.
dbvis.usestandardgridfit=false	Enable this property and DbVisualizer will use an accurate but slow method to automatically resize grid columns. "Accurate" since it does a real calculation of the columns width. If leaving this property disabled then column widths are determined much faster but depending on what grid font is used some columns may be truncated with "...". This property has an effect only if Tool Properties->Grid->Auto Resize Column Widths is enabled
dbvis.-ConnectionTestTimeout=20	The timeout in seconds for the "Ping Connection" feature.
dbvis.<database>.-RemoveNewLineChars=false	Backward compatibility setting used to specify that the SQL command will be trimmed of all whitespaces, tabs and newlines just before it is executed by the DB server.
locale=en,us	Use this to specify an alternate Locale.

These properties may change in future versions of DbVisualizer. Some are also experimental and may be removed or instead introduced in the DbVisualizer GUI.

Install license key for DbVisualizer Personal

If you have a license key file for DbVisualizer Personal, then start DbVisualizer and open the **Help->License Key** window. Enter the name of the license file in the **License Key File** field, or launch the file chooser by pressing the "..." button to the right of the license file field. Once the file is loaded, press the **Install License** button.

Uninstalling the license key

If you ever need to uninstall the license key, you can do so by removing (or renaming) the following file:

Operating System	File Name
Windows	C:\Documents and Settings\ <user>\.dbvis\dbvis.license< td=""> </user>\.dbvis\dbvis.license<>
UNIX/Linux	/home/<user>/.dbvis/dbvis.license
Mac OS X	/Users/<user>/.dbvis/dbvis.license

Useful Resources

Resources related to DbVisualizer that are useful:

1. The home of [DbVisualizer](#)
2. The [FAQ](#) which is regularly updated with frequently asked questions and known problems
3. The [User Guide](#)
4. The [Databases](#) and [JDBC Drivers](#) online page. This page gives information about supported databases and JDBC drivers
5. The DbVisualizer [forums](#)
6. The online [problem report](#) form. This is the recommended channel for product support and general questions

Starting DbVisualizer

How to start DbVisualizer depends on the operating system you are using.

- **Windows**
Locate the DbVisualizer submenu in the **Start** menu. Select the **DbVisualizer** entry in that menu
- **Linux/Unix**
Open a shell and change directory to the DbVisualizer installation directory. Execute the **dbvis** program
- **Mac OS X**
Double click on the **DbVisualizer** application or the **DbVisualizer.app** application bundle.

Command line arguments

You can also start DbVisualizer from a shell on all operating systems. On Windows and Linux/Unix, change the directory to the DbVisualizer installation directory and run the **dbvisgui** command. On Mac OS X, you can use the **open** command like this:

```
open -a DbVisualizer-<Version>.app --args <Arguments>
```

The command supports a number of command line arguments. These are listed in the **Help->About** menu choice, under the **Command Line** tab, in DbVisualizer.

```
Usage: dbvisgui [<filename>] [-encoding<encoding>]
          [-prefsdir <directory>] [-help] [-version]
General Options:
<filename>          SQL script file to load into editor
-encoding <encoding> Encoding for the SQL script file
-prefsdir <directory> Use an alternate user preferences directory
-help              Display this help
-version          Show version info
```

Pure command line interface

In addition to the DbVisualizer GUI tool, there is also a pure command line tool. We recommend that you use this tool for tasks that you schedule via the operating system's scheduling tool, or when you need to include database tasks in a command script for a larger job. It is also the right tool for execution of large scripts, such as a script generated by the DbVisualizer Export Schema feature.

On Windows and Linux/Unix, you find this command as a BAT file (*dbviscmd.bat*) or a shell script (*dbviscmd.sh*) in the DbVisualizer installation directory. For Mac OS X, the shell script is located in */Application/DbVisualizer-<Version>.app/Contents/Resources/app*.

```
Usage: dbviscmd -connection <name> [-userid <userid>] [-password <password>]
        -sql <statements> | -sqlfile <filename> [-encoding <encoding>]
        [-catalog <catalog>] [-schema <schema>]
        [-maxrows <max>] [-maxchars <max>]
        [-stoponerror] [-stoponwarning]
        [-output all | none | log | result] [-outputfile <filename>]
        [-listconnections]
        [-debug [-debugfile <filename>]
        [-prefsdir <directory>] [-help] [-version]
```

Options:

-connection <name>	Database connection name (created with the GUI)
-userid <userid>	UserId to connect as
-password <password>	Password for userid
-sql <statements>	One or more delimited SQL statements
-sqlfile <filename>	SQL script file to execute
-encoding <encoding>	Encoding for the SQL script file
-catalog <catalog>	Catalog to use for unqualified identifiers
-schema <schema>	Schema to use for unqualified identifiers
-maxrows <max>	Maximum number of rows to display for a result set
-maxchars <max>	Maximum number of characters to display for a column
-stoponerror	Stop execution when getting an error
-stoponwarning	Stop execution when getting a warning
-output	"all" (default), output both log msgs and result sets "none", suppress both log messages and result sets "log", output only log messages "result", output only result sets
-outputfile <filename>	Script execution output file. Default is stdout
-listconnections	Lists all database connections
-debug	Write debug messages
-debugfile <filename>	File for debug messages. Default is stderr
-prefsdir <directory>	Use an alternate user preferences directory
-help	Display this help
-version	Show version info

Before you can use the pure command line tool, you need to create at least one database connection using the GUI tool. You need to specify the connection to use with the **-connection** option when you run **dbviscmd**. If you have forgot the connection name, use the **-listconnections** option to get a list of all connection names.

The Main Window and Common Components

As you can see in the in the screenshot below, the DbVisualizer interface has a navigation area with two tabs (Databases and Scripts) to the left and two tabs (Object View and SQL Commander) to the right.

Databases

This tab holds the **Database Object Tree**. It keeps (at the top level) all the **Database Connection** objects (or folder objects, used to organize Database Connections). Use this tree to navigate and explore the database. Clicking on an object will change the view in the **Object View** tab to show details about the selected object.

Scripts

This tab holds [Bookmarks](#) and [Monitors](#), providing easy access to SQL scripts that you use frequently for different purposes.

Object View

This tab shows detailed information about the object represented by the selected node in the Database Object Tree. The content of the Object View tab depends on the type of the selected object.

SQL Commander

The SQL Commander lets you execute any SQL statements and scripts.

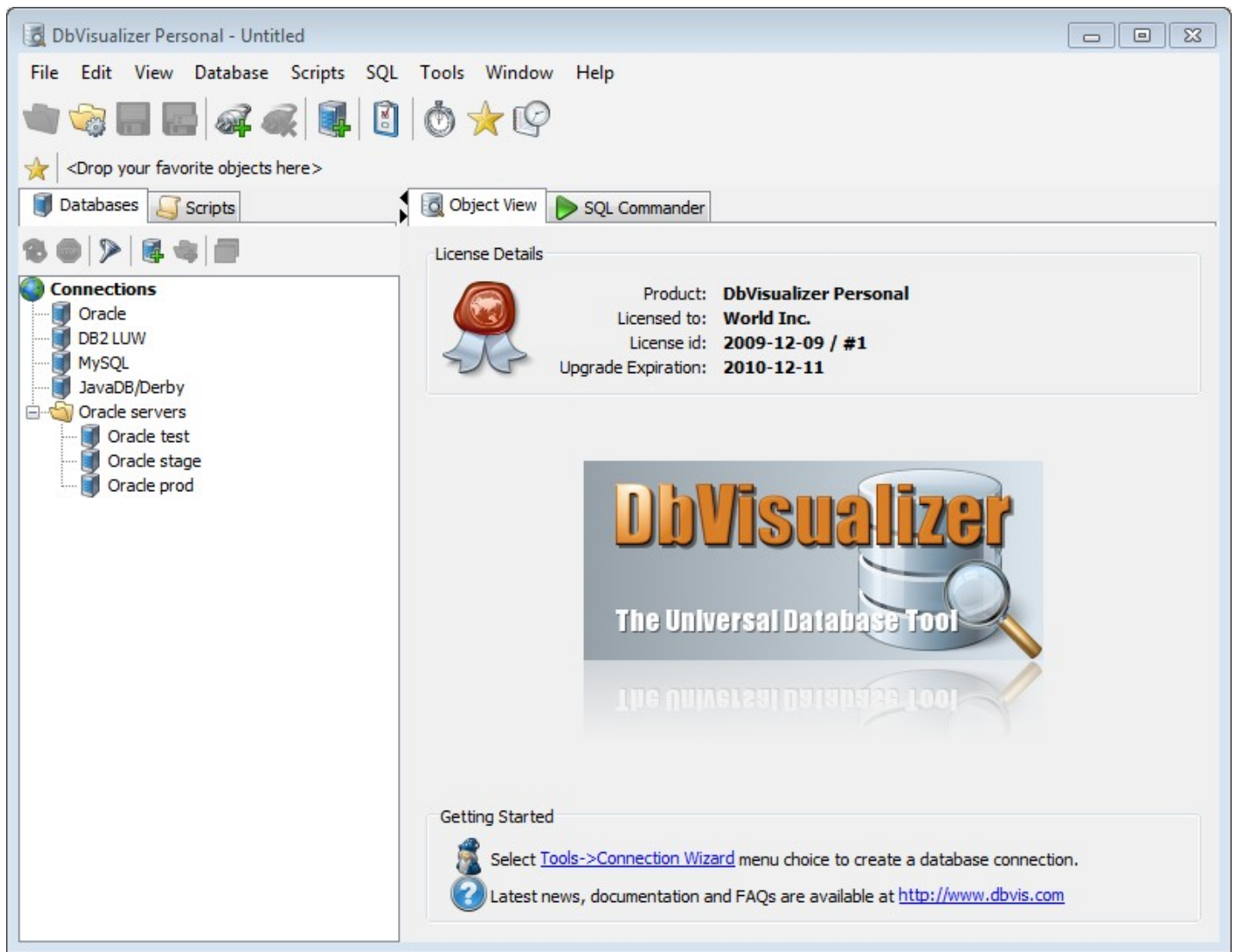


Figure: The DbVisualizer main window

Standard Components in the User Interface

The following sections introduce generic features and components that you find in many parts of DbVisualizer.

Grid, Graph and Chart

Grid, graph and chart are three terms that are often used in the application and in the documentation. The following screenshots show what they represent.

Grid

	🔑 EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
1	198	Donald	OConnell	DOCONNEL
2	199	Douglas	Grant	DGRANT
3	200	Jennifer	Whalen	JWHALEN
4	201	Michael	Hartstein	MHARTSTE
5	202	Pat	Fay	PFAY
6	203	Susan	Mavris	SMAVRIS
7	204	Hermann	Baer	HBAER
8	205	Shelley	Higgins	SHIGGINS
9	206	William	Gietz	WGIEZT
10	100	Steven	King	SKING

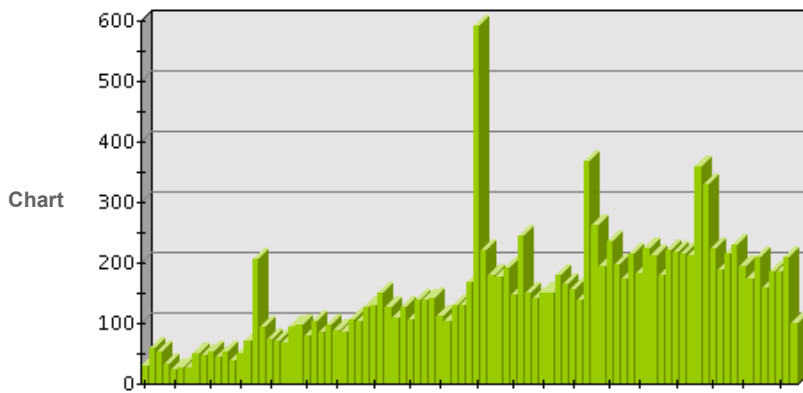
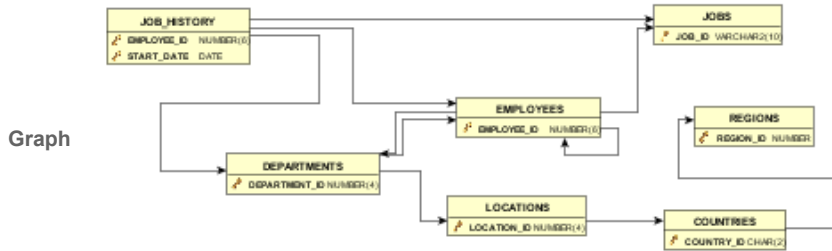


Figure: The grid, graph and chart terms

The documentation uses the term **grid** for the user interface component that represents tabular data, rather than **table**, to avoid confusion with a database table.

Context Sensitive Components

All components in the user interface (e.g., buttons and menu items) are context sensitive. They are enabled only if they can be used in the current scope.

Tooltips

Tooltips are used to provide more details about a component. They are also used to express status information. An example is the grid column header tooltip that shows information about the column. To see a tooltip, let the mouse hover over an area of the user interface, e.g., a button or grid header. If there is a tooltip for the area, it will pop up in about a second.

Related Table:	
DEPARTMENT	> EMPLOYEES (EMPLOYEE_ID)
Administrative Services	> LOCATIONS (LOCATION_ID)
Marketing	> EMPLOYEES (EMPLOYEE_ID)
Purchasing	<- JOB_HISTORY (DEPARTMENT_ID)
Human Resources	DEPARTMENTS (DEPARTMENT_ID) <- EMPLOYEES (DEPARTMENT_ID)

Figure: Tooltip example

Grids

Grids are used heavily in DbVisualizer and require a brief introduction.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	HIRE_DATE	PHOTO	RESUME
1	206	William	Gietz	1994-06-07 ...	BINARY, 5 851 Bytes	(null)
2	205	Shelley	Higgins	1994-06-07 ...	(null)	(null)
3	204	Hermann	Baer	1994-06-07 ...	BINARY, 8 719 Bytes	(null)
4	203	Susan	Mavris	1994-06-07 ...	BINARY, 8 620 Bytes	CLOB, 4 068 Bytes
5	202	Pat	Fay	1997-08-17 ...	(null)	(null)
6	200	Jennifer	Whalen	1987-09-17 ...	(null)	(null)
7	198	Donald	OConnell	1999-06-21 ...	BINARY, 10 630 Bytes	(null)
8	143	Randall	Matos	1998-03-15 ...	(null)	(null)
9	142	Curtis	Davies	1997-01-29 ...	(null)	(null)
10	141	Trenna	Rajs	1995-10-17 ...	(null)	(null)
11	140	Joshua	Patel	1998-04-06 ...	(null)	(null)
12	139	John	Seo	1998-02-12 ...	(null)	(null)
13	138	Stephen	Stiles	1997-10-26 ...	(null)	(null)

Max Rows: 50 Max Chars: 0 0.016/0.016 sec 50/13 1-13

Figure: Grid overview

The screenshot shows the grid and controls that are available for the grid in the **Data tab**, but the differences are minor compared to grids used in other places.

Sorting

You can sort the grid based on the values in one or more columns. When you click on a column header, the grid is sorted in ascending order on the values in that column, indicated by an up-arrow in the column header. If you click the same column header again, the grid is sorted in descending order, indicated by a down-arrow in the column header. If you click a third time, the data is shown in the order it was received from the database and the sort indicator disappears.

To sort on more than one column, Ctrl-click (keep the Ctrl key pressed when clicking) on additional columns. The grid is then sorted on the values in the first column you clicked on (indicated with a 1 next to the arrow), and then all rows with the same value in the first column are sorted on the values in the second sort column (indicated with a 2 next to the arrow), and so on.

Right-click menu

The generic right-click grid menu contains the following operations:

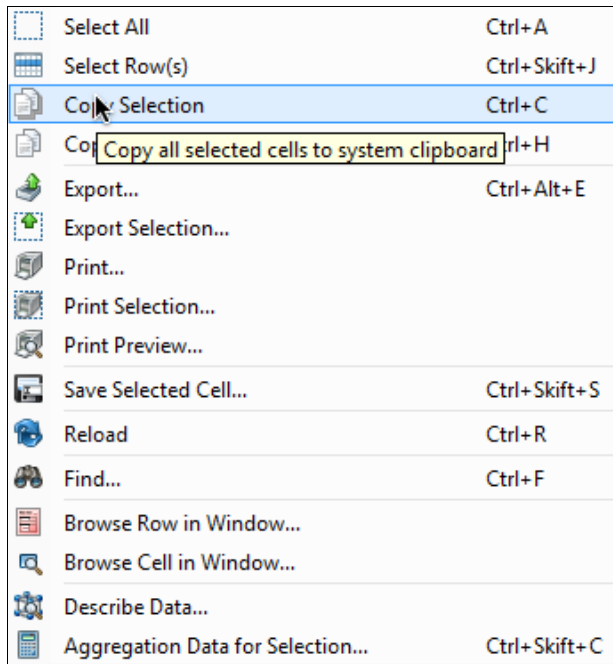


Figure: Grid right click menu

Menu Choice	Description
Select All	Select all cells (i.e., all rows and columns) in the grid
Select Row(s)	Select all cells in the selected row(s)
Copy Selection	Copy all selected cells onto the system clipboard
Copy Selection (With Column Header)	Copy all selected cells including column header onto the system clipboard
Export	Copy the export dialog
Export Selection	Export the selection using the standard export feature
Print	Open the print dialog for printing the complete grid
Print Selection	Open the print dialog for printing just the selected rows/columns
Print Preview	Open the print preview dialog
Save Selected Cell	Save the value of the selected cell to a file, selected with a file chooser dialog
Reload	Reload the grid with data from the database
Find	Open the find dialog
Browse Row in Window	Display all data for the selected row in a separate window.

Note: for a read/write grid, this entry is named Edit Row in Window.

Browse Cell in Window

Display the cell value in a separate window. This is especially useful for BLOB/CLOB data.
Note: for a read/write grid, this entry is named Edit Cell in Window.

Describe Data

Show detailed information about the columns in the grid

Aggregation Data for Selection

Displays some metrics about the current selection. Read more in [Aggregation Data for Selection](#) below.

The menu may contain additional entries based on the current scope, e.g., entries for editing cell values for a read/write grid.

Aggregation Data for Selection

The **Aggregation Data for Selection** feature performs calculations on the current selection. It provides information about cells holding numbers, text, date/time information and more. The following is an example of what it shows.

Aggregation Data for Selection	
Selected Cells Count	650
Rows	50
Columns	13
Nulls	140
Text Count	250
Shortest	2
Avg Length	7
Longest	12
Total Length	1 894
Number Count	250
Min	10
Avg	1 470,04
Median	120
Max	17 000
Sum	294 009
Std Deviation	2 997,28
Nulls	50
Timestamp Count	50
First	1987-09-17 00:00:00
Avg	1996-10-09 05:16:48
Median	1997-08-18 12:00:00
Last	2000-03-08 00:00:00
Binary/BLOB Count	50
Nulls	44
CLOB Count	50
Nulls	46

Auto Update

Update Close

Figure: The Aggregation Data for Selection dialog

With **Auto Update** checked, the data is updated automatically when you change the selection. For very large selections, you may prefer to disable this feature and instead click **Update** when you want to refresh the data. Click a link (blue underlined text) in the aggregation table to locate and highlight the actual value in the source data grid.

Column Visibility

The **Column Visibility** dialog controls which columns you want to appear in a grid. You open the column visibility dialog by clicking the button above the vertical scrollbar in the grid.

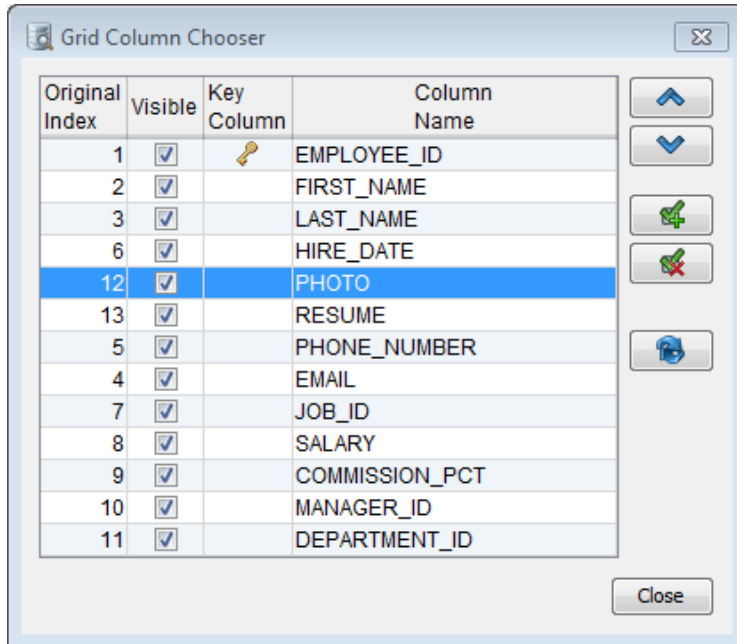


Figure: The Column Visibility dialog

The Column Visibility dialog shows all columns that are available in the grid.

The checkmark in front of a column name indicates that the column is visible in the grid, while an unchecked box indicates that it is excluded from the grid. Click the checkmark to change the visibility of a column. You can change the visibility for all columns at once using the two visibility buttons in the dialog.

The order of the columns can also be adjusted in this dialog. Just select a row and use the **Up** and **Down** buttons to move it up (left in grid) or down (right in grid).

If you want to revert your changes, you can click on the **Default Layout** button to reset the grid, i.e., making all column visible and put them in their default locations.

Note 1: Modifications of column visibility, size and order are saved between invocations of DbVisualizer for all grids in the various **Object View** tabs **except** for the Data tab.

Note 2: If you modify the column visibility in the Data tab, the changes persists throughout the session. For instance, if you remove the **Name** column in the Data tab for the table **EMPLOYEE**, the **Name** column remains excluded when you reload the table or come back to the Data tab for that table later in the same session. You must manually make it visible again to bring it back. The changes are, however, reset when you restart the application.

Auto Resize

The column header right-click menu contains a number of options for automatic resizing of column widths.

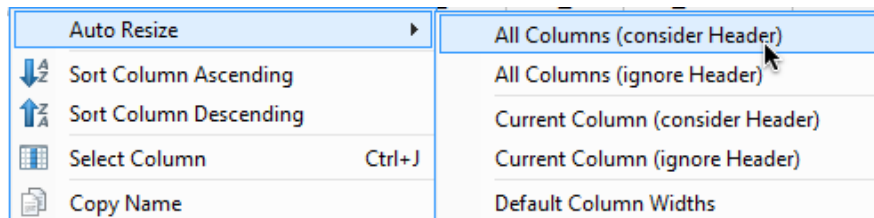


Figure: Auto Resize menu

Menu Choice	Description
All Columns (consider Header)	Resize all columns to fit the widest cell value, or the column header if it is wider than any cell value
All Columns (ignore Header)	Resize all columns to fit the widest cell value
Current Column (consider Header)	Resize the current column to fit the widest cell value, or the column header if it is wider than any cell value
Current Column (ignore Header)	Resize the current column to fit the widest cell value
Default Column Widths	Set all column widths to their default width

Quick Filter

All areas that hold a grid in DbVisualizer also provide a Quick Filter field.

The screenshot shows the DbVisualizer interface with a grid of employee data. The grid has columns for EMPLOYEE_ID, FIRST_NAME, LAST_NAME, and HIRE_DATE. A Quick Filter field is located at the top right of the grid, containing the text 'JO'. A pull-down menu is open next to the filter field, listing various columns and options. The menu items are: All, EMPLOYEE_ID, FIRST_NAME (checked), LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID, PHOTO, RESUME, Case sensitive, Case insensitive (checked), Use wild cards, Match from start, Match exactly, and Match anywhere (checked).

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	HIRE_DATE
1	110	John	Chen	1997-09-28 ... (null)
2	112	Jose Manuel	Urman	1998-03-07 ... (null)
3	139	John	Seo	1998-02-12 ... (null)
4	140	Joshua	Patel	1998-04-06 ... (null)

Figure: Grid with Quick Filter

When you type in the Quick Filter field, DbVisualizer matches the value with cell values in the grid and filters out all rows that do not have a match in at least one cell. The Quick Filter pull-down menu (click on the down arrow next to the magnifying glass) lets you choose if the filter should match cells in all columns or just one selected column, case or case insensitive matching, and where in the cell the value must match.

Print

DbVisualizer supports printing of grids, graphs, charts and plain text, such as the content of an SQL Editor. The print dialog looks somewhat different depending on what is printed. In all cases, you launch the print dialog by clicking on the Print button in the toolbar for the object you want to print, or by choosing Print from the right-click menu. The right-click menu also contains a Print Preview choice, if you want to see what the printout will look like before you actually print.

Printer Setup

If you want to set the page orientation (e.g., portrait or landscape) and paper size, you must launch the Printer Setup dialog, using the **File->Printer Setup** main menu option, before you print. Printing varies widely between platforms, so even though the Print dialog (as opposed to the Printer Setup dialog) on some platforms also lets you choose a page orientation and other options, they may be ignored if specified in that dialog. The only supported way to specify the page orientation and other options is via the Printer Setup dialog.

Grid, Chart and Plain Text

For a grid, chart and plain text, DbVisualizer launches the platform's native Print dialog, so it looks different on different platforms. The two options available on all platforms are a choice of printer and the page range. On some platforms, the dialog may offer additional options, but they may be ignored by DbVisualizer. Use the Printer Setup dialog to set other options besides which printer to use and the page range, as described above.

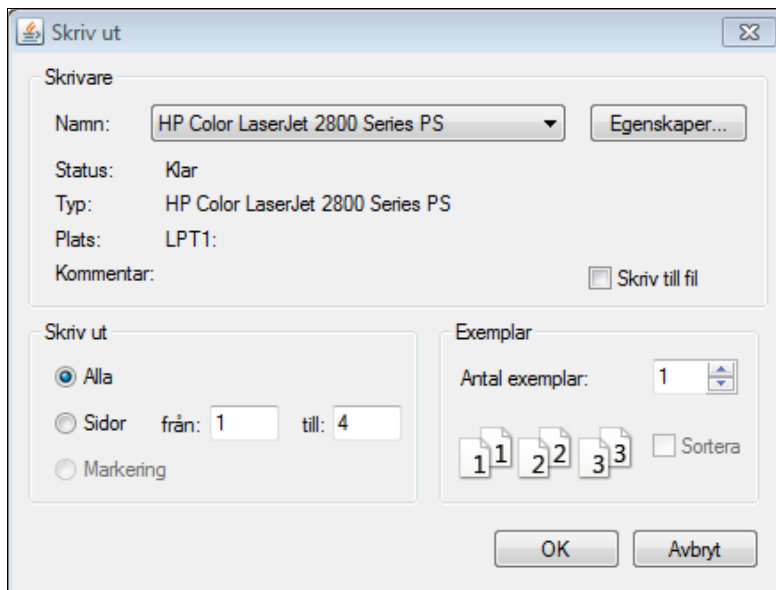


Figure: Standard print dialog

The figure above shows how the Print dialog looks on the Windows platform (the appearance and printing capabilities are platform specific so the printing dialog look different in Windows, Linux/UNIX and Mac OS X).

When you print a grid in DbVisualizer, the grid is printed as it is shown on the screen, i.e., with the table headers, sort and primary key indicator, etc. It is printed as a screenshot that may span several pages, depending on the number of rows and columns that are printed. For a grid, the right-click menu contains a Print Selection choice that you can use if you just want to print selected rows and columns.

An alternative to printing a grid as a screenshot is to export the grid to HTML and then use a web browser to print it.

Printing a chart scales the chart to the size of the paper. Plain text is printed as-is and may span multiple pages, both in height and width.

Graph

Printing a graph adds a custom dialog before the native Print dialog is displayed.

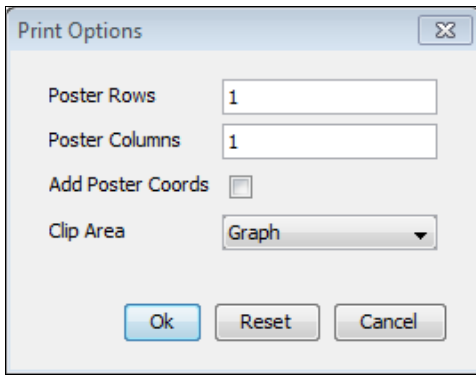


Figure: Print options when printing graphs

You can specify the number of rows (pages) and columns (pages) that the complete image will be split into. You can also select whether the view as it appears on the screen or the complete graph should be printed. When you click Ok, the native Print dialog is displayed, where you can select the printer.

Print Preview

Use the **File->Print Preview** feature to preview what the printout will look like before you actually print it.

Grid

Graph

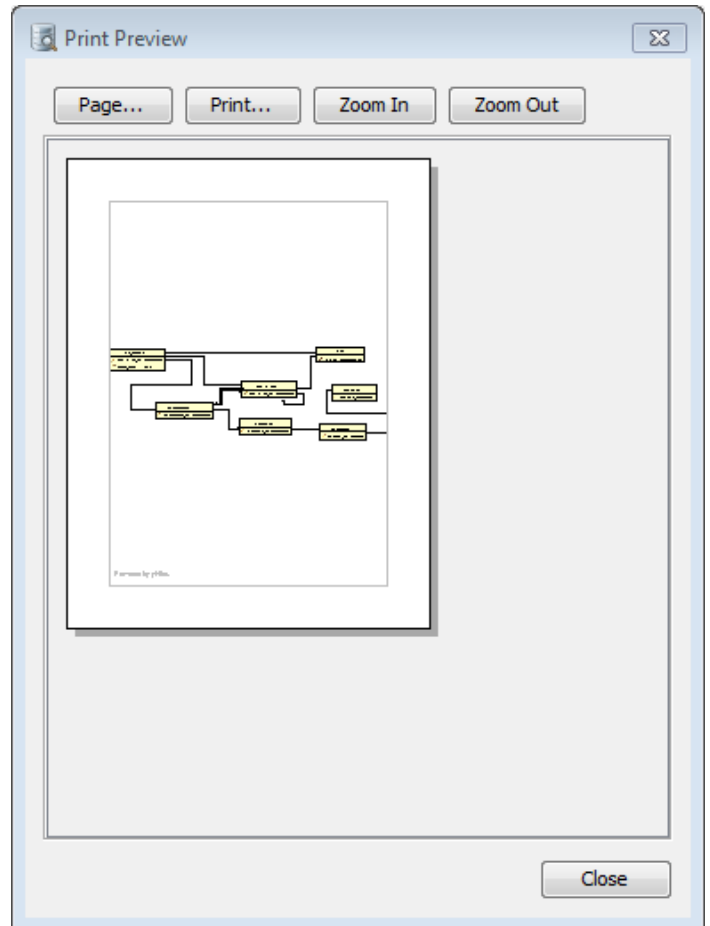
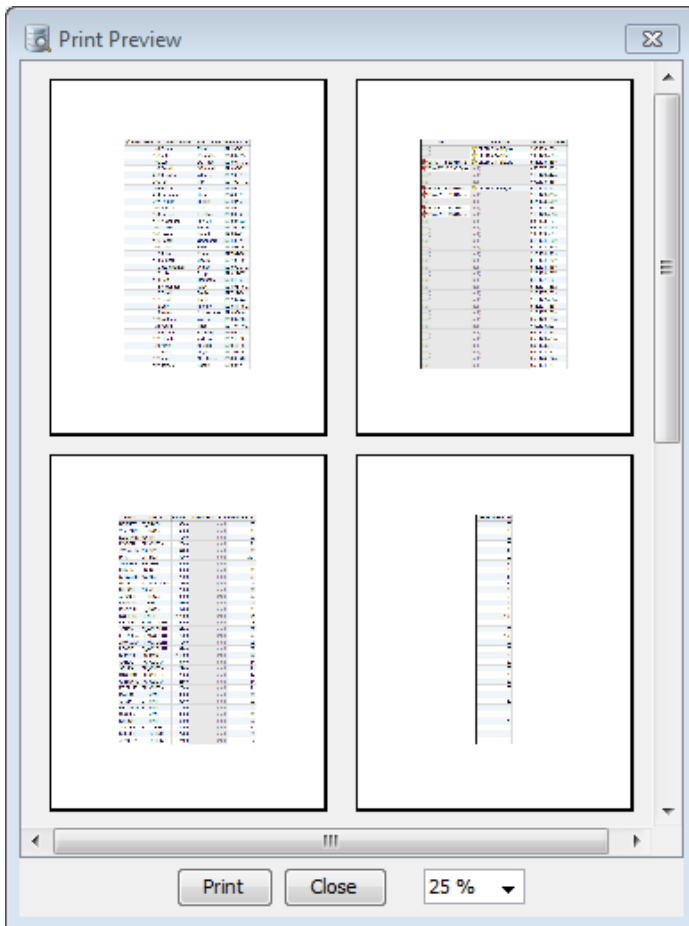


Figure: Grid and graph print previews

Checking for Updates

We frequently release new versions of DbVisualizer to introduce new features and improvements, and to fix problems. To make you aware of new versions, DbVisualizer periodically checks if a newer version than the one you are using is available when you start DbVisualizer. If there is a newer version available, you are presented with a dialog with links to pages on our site where you can read more about it and download the new version.

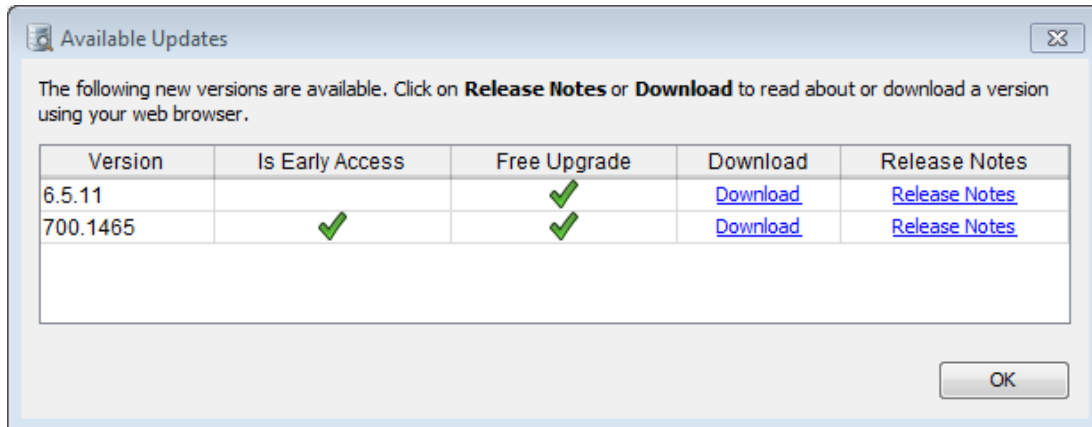


Figure: Available Updates dialog

The **Early Access** field tells you if it is an Early Access version (i.e., a preview of an upcoming major version we are currently working on) and the **Free Upgrade** field is checked if you can upgrade to this version with your current license. The list may also have a **Comments** field with more information about the version. Click on the links in the **Download** and **Release Notes** fields to open a browser with the corresponding page from our web site.

DbVisualizer checks for new versions at start-up on a weekly basis by default. You can change the interval or check manually at any time by launching the **Check for Update** dialog from the **Help** menu.

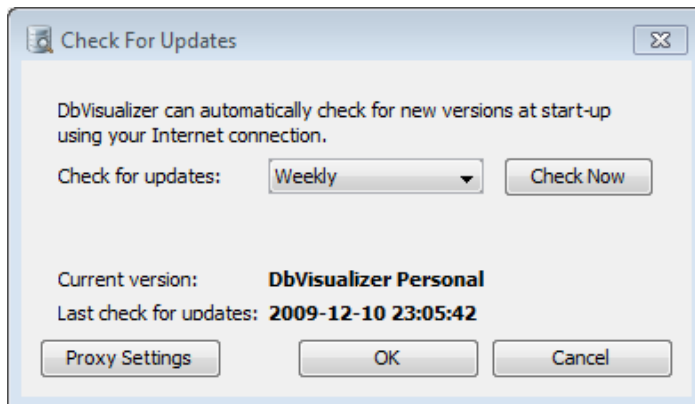


Figure: Check For Updates dialog

You can set the interval to one of **Every Start-Up**, **Weekly**, **Monthly** or **Never**, or click the **Check Now** button to see if there are any new versions available right now. If new versions are available, the same dialog as shown above appears, otherwise a message tells you that you are running the latest version.

The dialog also informs you about which version you are currently running and when the last check for updates took place.

Click **OK** to save the new interval or **Cancel** to leave it as it was.

If you are accessing the Internet through a proxy, you must enter information about the proxy in the [Tool Properties](#) dialog before you check for updates.

Problem resolution

Even though we make our very best to ensure the quality of DbVisualizer, you may run into problems of different kinds. The runtime environment for DbVisualizer is rather complicated when it comes to tracking the source of a problem, since it's not only DbVisualizer that may cause the problem but also the JDBC driver, or even the database engine.

There are a few things that you can try before reporting a problem, depending on the nature of the problem:

1. Make sure you are using the latest version of [Java](#) available for your platform (Java 6 or later)
2. Make sure you are using a [version](#) of the JDBC driver that we've tested DbVisualizer with, or a later, production quality version
3. Read the DbVisualizer [FAQ](#)
4. Check the online [Forums](#)
5. Read the DbVisualizer [Users Guide](#)
6. ... the last resort is to post a question via the [problem report form](#) or send an email to support@dbvis.com. (Note that we generally love detailed reports as well as screenshots when possible)

Debugging DbVisualizer

The **Tools->Debug Window** is useful to see what is going on in DbVisualizer and the JDBC driver(s). The checks at the top control what parts of DbVisualizer should be debugged. The **Debug JDBC Drivers** option will enable debug of the current JDBC driver. Note that the amount of output is determined by the JDBC driver.

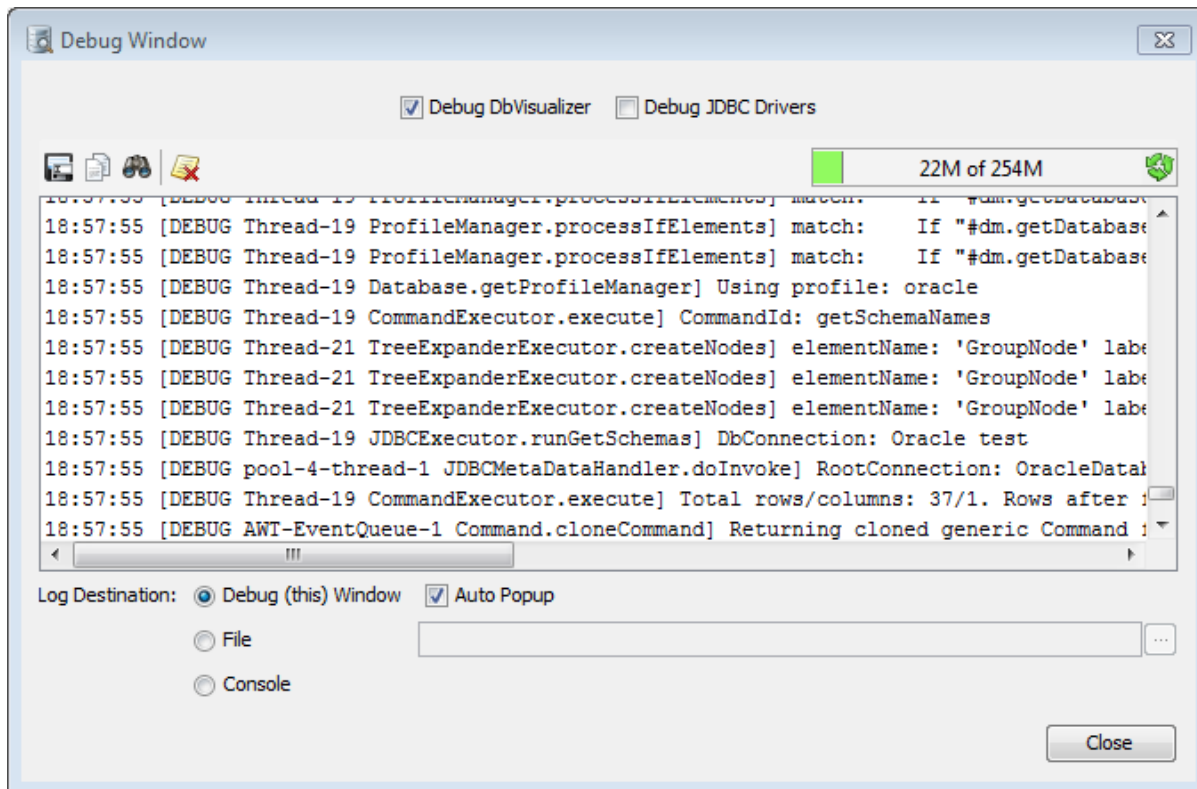


Figure: The DbVisualizer Debug Window

The **Save** and **Copy** buttons will prepare the log with information about the DbVisualizer version you are using and the connected database connections.

The log is automatically truncated to preserve memory when the log destination is set to **Debug Window**. The **Console** and **File** destinations have no such limitation.

How to satisfy the DbVisualizer support team

Quite often we get incomplete problem reports and need to follow up for additional information. If you encounter a problem, please follow these steps to include the details we need to help you:

1. Select the **Connection** tab
2. In the **Connection Message** area, select the right-click menu
3. In the menu, select **Copy**
4. This copies the system details to the clipboard. Then paste the details into an email or in the problem report form.

5. In addition, we really appreciate it if you provide us with screenshots. An image says more than ... you know.

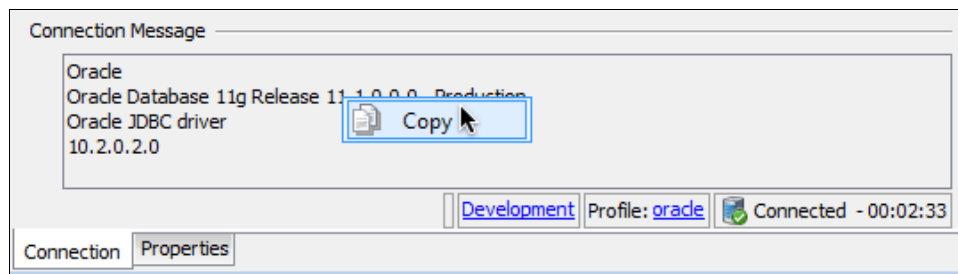


Figure: The connection message right click menu

Load JDBC Driver and Get Connected

Introduction

This document describes the way JDBC drivers are managed in DbVisualizer and all aspects about getting connected to your database(s).

If you are impatient, please go ahead and read the [Connection Wizard](#) section. It is the recommended way to create database connections in DbVisualizer.

What is a JDBC Driver?

DbVisualizer is, as you know, a generic tool for administration and exploration of databases. DbVisualizer is in fact quite simple, since it does not deal directly with how to communicate with each database type. The hard job is done by a JDBC driver, which is a set of Java classes that are either organized in a directory structure or collected into a JAR or ZIP file. The magic of these JDBC drivers is that they all match the JDBC specification and the standardized Java interfaces. This is what DbVisualizer relies on. A JDBC driver implements all details for how to communicate with a specific database and database version, and there are a range of drivers from the database vendors themselves and 3:rd party authors. To establish a connection with a database, DbVisualizer loads the driver and then get connected to the database through the driver.

It is also possible to obtain a database connection using the Java Naming and Directory Interface (JNDI). This technique is widely used in enterprise infrastructures, such as application server systems. It does not replace JDBC drivers but rather adds an alternative way to get a handle to an already established database connection. To enable database "lookup's" using JNDI, an Initial Context implementation must be loaded into the Driver Manager. This context is then used to lookup a database connection. The following sections describe the steps for getting connected using a JDBC Driver, and also how to use JNDI to obtain a database connection.

A complete JDBC driver typically consists of a number of Java classes, located in a JAR, ZIP or a folder, that need to be loaded into the DbVisualizer driver manager. DbVisualizer automatically recognizes the classes that are used to initiate the connection with the database and presents them in the **Driver Class** list. You must select the correct class in this list to make sure DbVisualizer successfully can initiate the connection. Consult the driver documentation for information of which class to select, or if the number of classes found are low, figure it out by trying each of them. More about this in the following sections.

Get the JDBC driver file(s)

DbVisualizer comes bundled with all commonly used JDBC drivers that have a license that allows for distribution with a third party product. Currently, drivers for DB2, JavaDB/Derby, Mimer, MySQL, and PostgreSQL, as well the jTDS driver for SQL Server and Sybase, are included with DbVisualizer. If you only need to connect to databases of these types, you can skip the rest of this section and jump straight to the [Connection Wizard](#) section, because by default, DbVisualizer configures all these drivers automatically the first time you start DbVisualizer.

If you need to connect to a database that is not supported by a bundled JDBC driver, you must get a JDBC driver that works with your database type and version. The following online web page contains an up-to-date listing of the database/driver combinations we have tested:

<http://www.dbvis.com/products/dbvis/drivers.html>

Information about almost all drivers that are available is maintained by Sun Microsystems on this page:

<http://industry.java.sun.com/products/jdbc/drivers>

Download the driver to an appropriate directory. Make sure to read the installation instructions provided with the driver. Some drivers are delivered in ZIP or JAR format but need to be unpacked to make the driver files visible to the Driver Manager. The [Databases and JDBC Drivers](#) web page describes where you can download each driver and also what additional steps may be needed to install and load the driver in DbVisualizer.

(Drivers are categorized into 4 types. We're not going to explain the differences here, just give you the hint that the "type 4," aka "thin," drivers are the easiest to maintain, since they are pure Java drivers and do not depend on any external DLL's or dynamic libraries. Even though DbVisualizer works with any type of driver, we recommend that you get a type 4 driver if there is one for your database).

When you have downloaded the JDBC driver into a local folder (and unpacked it, if needed), you can go ahead and create a connect with the Connection Wizard, as described in the next section. You will then asked to load the driver files when the wizard needs them. Alternatively, you can move (or copy) the JDBC driver files to the **DBVIS_HOME/jdbc** folder, where they will be picked up and loaded automatically by the JDBC Driver Finder the next time you start DbVisualizer. You can read more about this option in the [JDBC Driver Finder](#) section.

Connection Wizard

The Connection Wizard greatly simplifies the steps needed to load the JDBC driver and create a new database connection. You just enter information about the driver file(s) and the connection data on a few wizard pages, and the wizard handles all the details. Once the new database connection has been created, it appears in the database objects tree.

The wizard cannot be used to define database connections via JNDI data sources.

The first wizard screen looks like this.



Figure: Connection Wizard - Page 1

In the **connection alias** field, enter the name of the new database connection. This is the name that will be used for the connection in DbVisualizer, e.g., in the object tree.

Press **Next** to go to the next page.

On this page, select the driver you are going to use from the list. A red icon in front of the driver name indicates that the driver is not yet ready to use, while a green icon indicates that it has been properly configured (simply press Next to continue).

If the driver you select is not yet configured, a **Load Driver File(s)** button is displayed. When you click the **Load Driver File(s)** button, a file chooser is opened. You should select the JAR or ZIP file(s) that contain the driver implementation.



Figure: Connection Wizard - Page 2

In the file chooser, locate the files containing the JDBC driver files. (Select multiple files by pressing the SHIFT key while clicking).

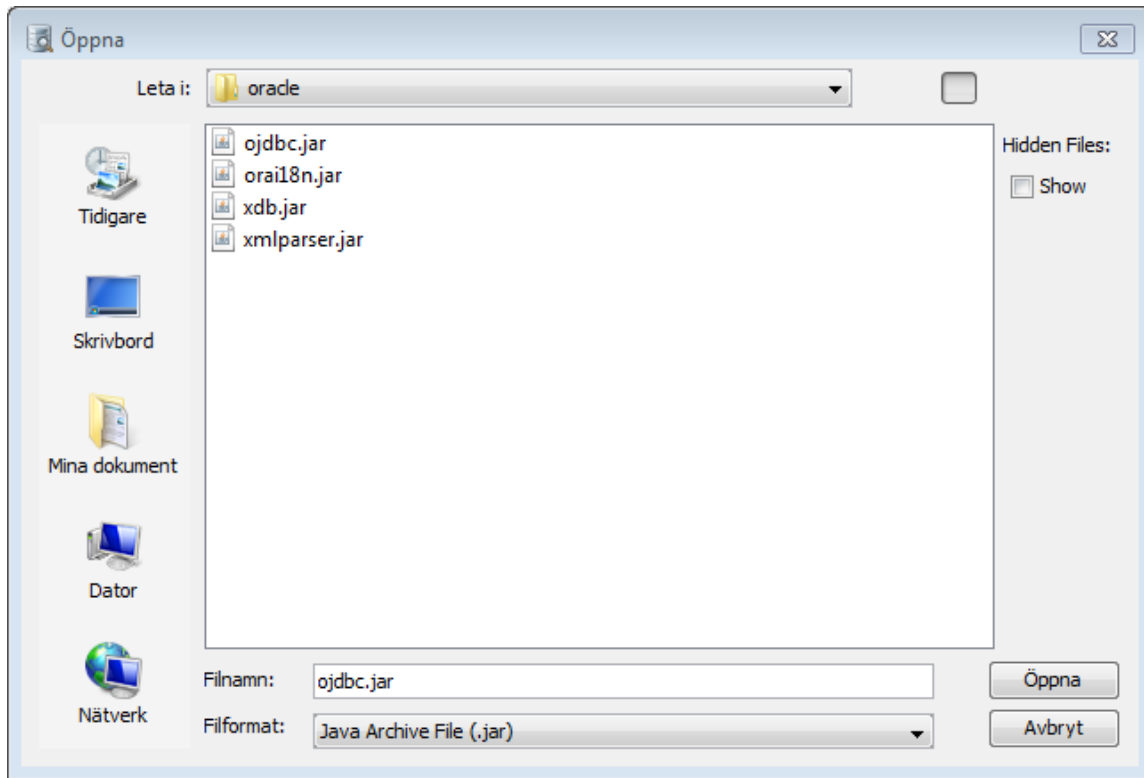


Figure: Connection Wizard - Page 3

Once the driver has been properly loaded, a green icon appears in front of the driver name. Press **Next** to continue to the last page.

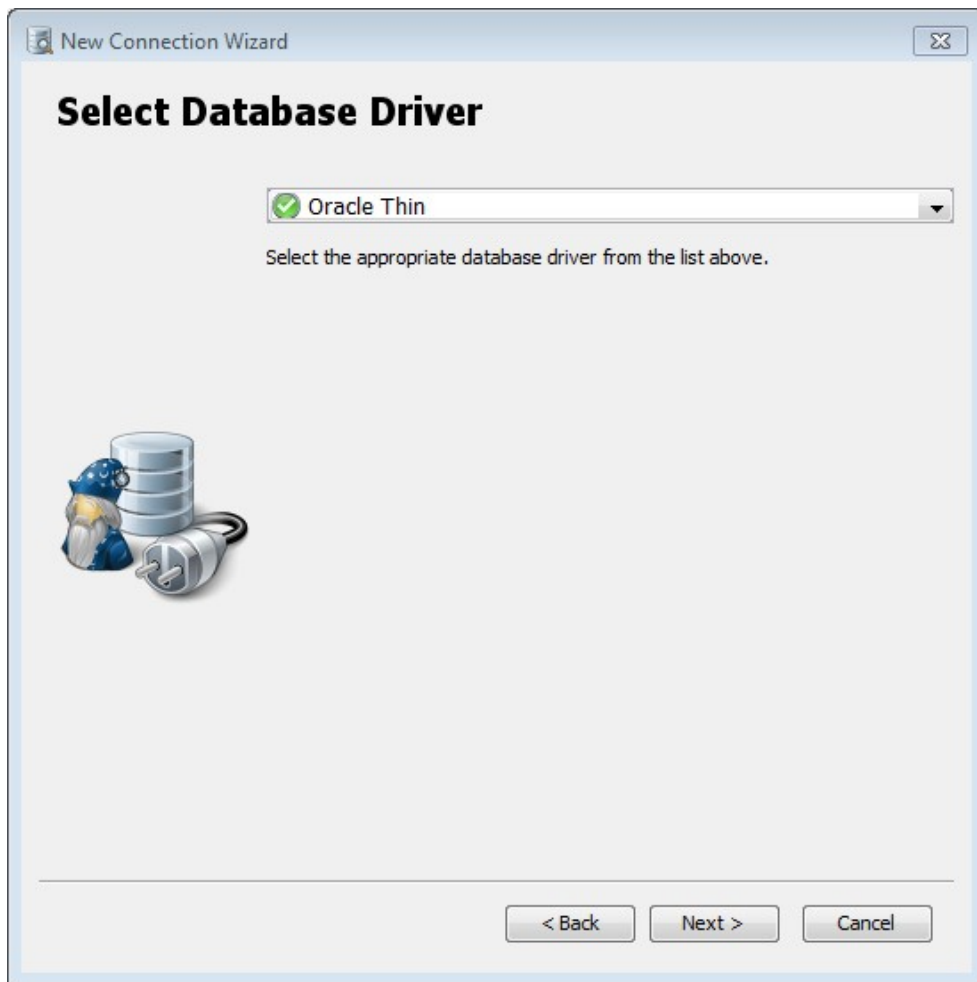


Figure: Connection Wizard - Page 4

On the last wizard pane, enter details for the new database connection. The information that must be provided varies depending on the database type. Please consult the database documentation if you are unsure about how to find the requested information.

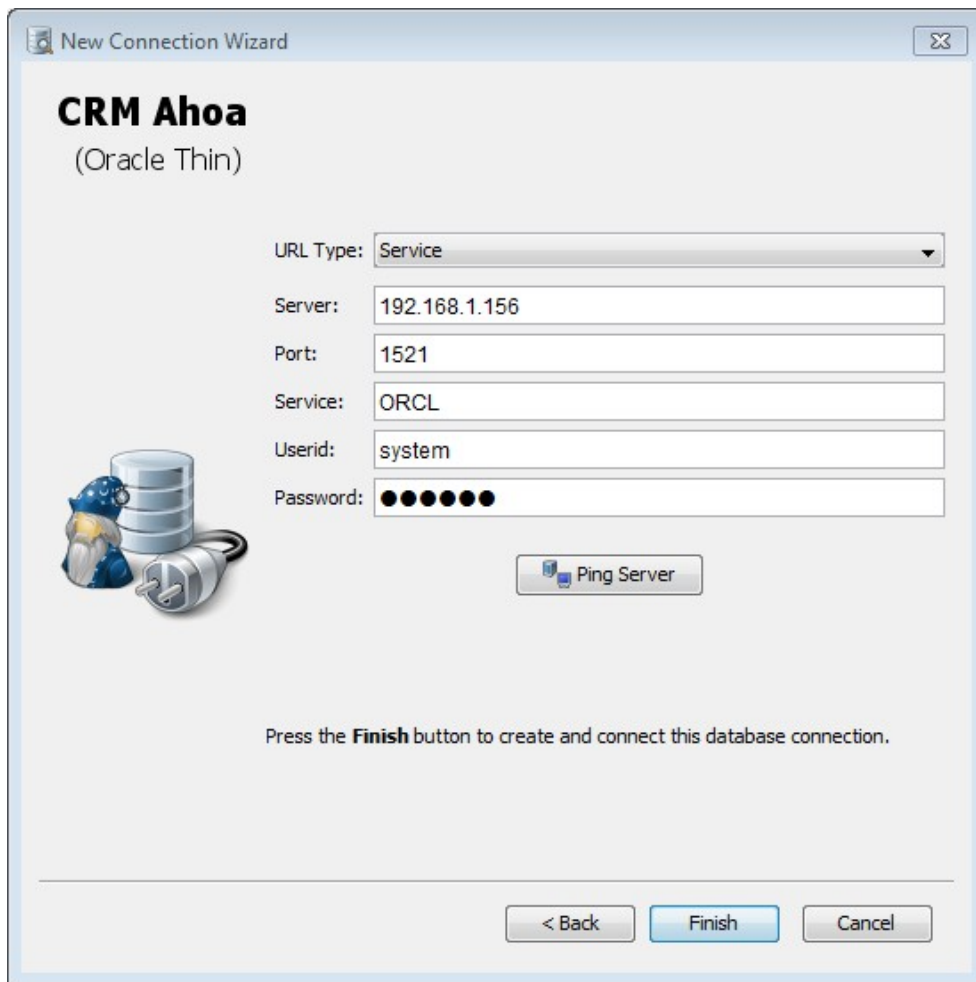


Figure: Connection Wizard - Page 5

Press **Ping Server** to verify that a network connection can be established to the specified host and port. If the test passes, press **Finish** to create the new database connection and connect to the database.

Some databases support different types of URLs. For instance, Oracle supports URLs that contain the host, port and service name or SID, but also a TNS URL type where you just include a TNS alias and get all the details from a *tnsnames.ora* file. For cases like this, the last wizard page has a URL Type list at the top where you select the type of URL to use.

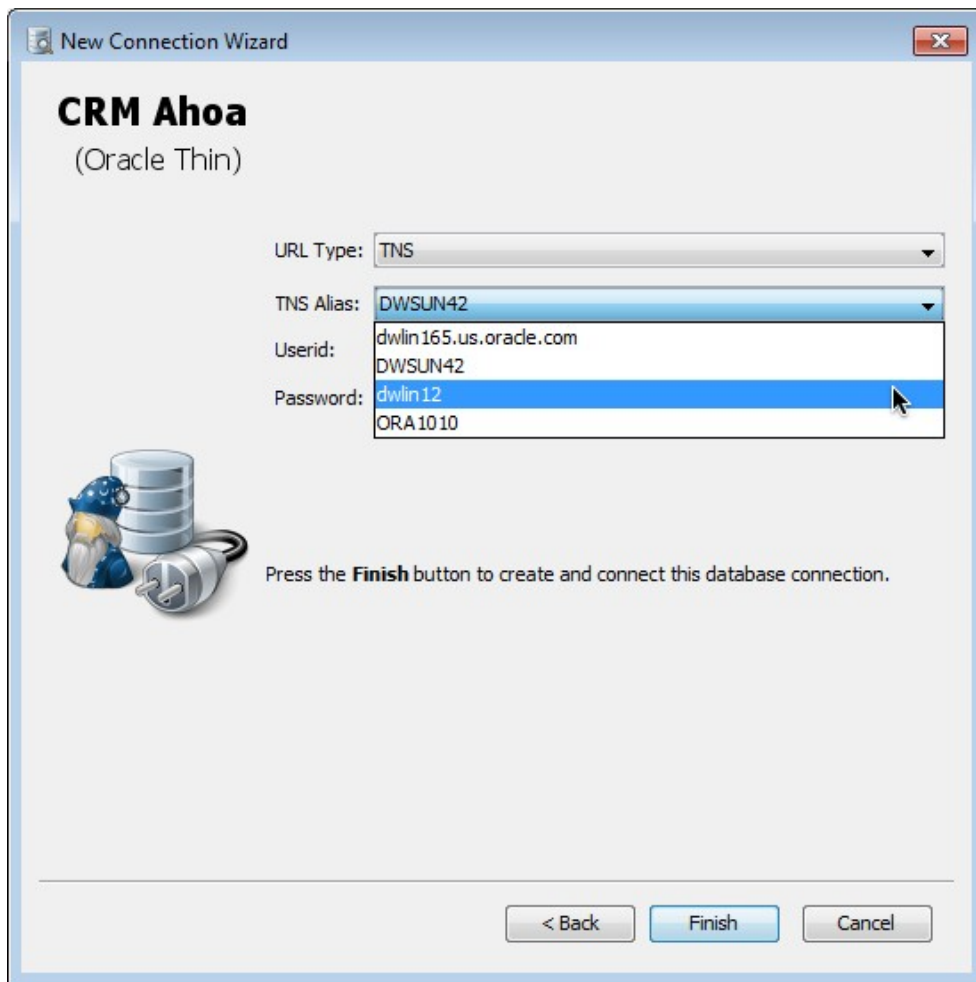


Figure: Connection Wizard - Page 5 for a database with multiple URL types

The other fields then depend on the selected URL type. For the Oracle TNS format, for example, you can pick the TNS alias from a list. If you like to use TNS URLs with an Oracle database, the tnsnames.ora file must be located either in the **ORACLE_HOME/network/admin** directory or in a directory identified by the **TNS_ADMIN** environment variable. For more about TNS, see the "Local Naming Parameters (tnsnames.ora)" chapter in Oracle Database Net Services Reference.

We recommend that you **skip the rest of this document**, unless you:

- want to learn how the driver manager in DbVisualizer works
- need to have several versions of the same JDBC driver loaded simultaneously
- need to establish a connection via the JNDI interfaces (Java Naming and Directory Interface)
- need to add a Driver that do not exist in the wizard list of drivers

Driver Manager

The **Driver Manager** in DbVisualizer is used to define the drivers that will be used to communicate with the databases. You can manually locate the JDBC driver files and configure the driver, or you can use the JDBC Driver Finder to do most of the work for you, either on demand or automatically.

JDBC Driver Finder

The **JDBC Driver Finder** is a very powerful part of the Driver Manager that automates most of the driver management work. Given the folders where JDBC drivers are located, it loads and configures new drivers (if any) every time you start DbVisualizer. You can configure the JDBC Driver Finder in Tools Properties, in the **General->Driver Manager** category.

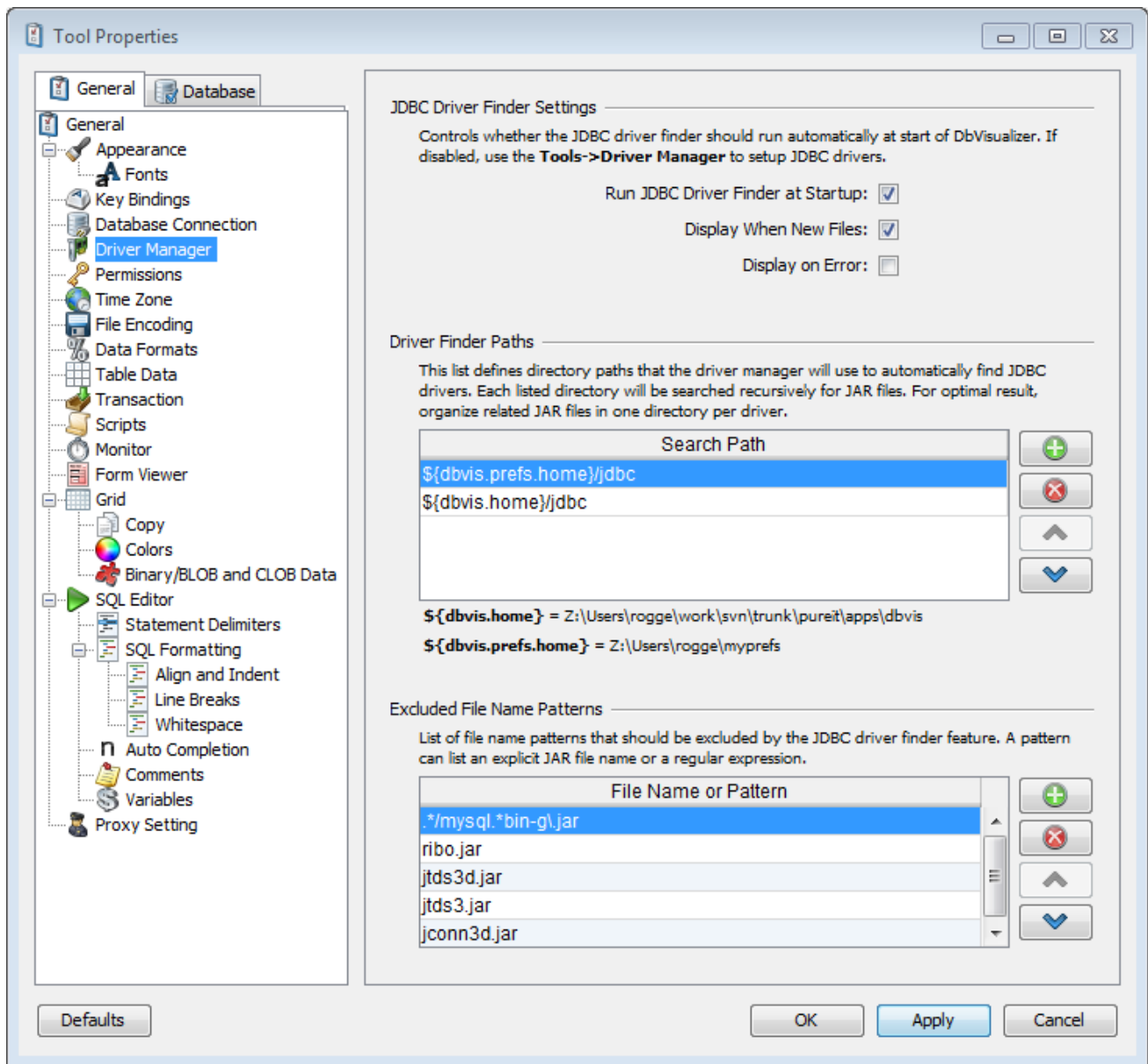


Figure: JDBC Driver Finder properties

Use the following properties to specify the finder behavior:

Property	Description
Run JDBC Driver Finder at Startup	If enabled, the finder will run automatically every time you start DbVisualizer. If it finds any new driver files, it will automatically load and configure them.
Display When New Files	If enabled, the finder window pops-up if it finds any new files when you start DbVisualizer. Otherwise, the finder runs invisibly in the background.
Display on Error	If enabled, the finder window pops-up if it encounters any errors loading and configuring new drivers. Otherwise, it is silent about errors and you have to launch the Driver Manager to see which drivers are not loaded successfully. Enabling this property is only meaningful if you have disabled Display When New Files .

You can also specify the folders the JDBC Driver Finder will search. By default, it will search folders named **jdbc** in the DbVisualizer installation directory (`${dbvis.home}`) and the DbVisualizer preferences folder (`${dbvis.prefs.home}`). These folder paths are shown under the list of Driver Finder Paths.

Finally, you can specify regular expression patterns for filenames that the finder should ignore. This can be useful if you need to store other files

besides driver files in the designated folders.

If you let the JDBC Driver Finder load all drivers for you, all you need to do to install a new driver is to put the driver files in one of the folders specified for the finder in Tool Properties and then restart DbVisualizer.

Loading and Configuring Drivers Manually

You can also load and configure JDBC drivers manually using the Driver Manager. If you use JNDI to provide access to the database, you must use this option, since the JDBC Driver Finder does not handle JNDI. Start the Driver Manager dialog using the **Tools->Driver Manager** menu choice.

The left part of the driver manager dialog contains a list of driver names with a symbol indicating whether the driver has been configured or not. The right part displays the driver configuration for the selected driver in terms of the following:

- **Name**
A **driver name** in the scope of DbVisualizer is a logical name for either a JDBC driver or an Initial Context in JNDI. This is the name shown in the **Connection** tab setup when selecting which driver to use for a **database connection**
- **URL Format**
The URL format specifies the pattern for the JDBC URL or a JNDI Lookup name. Its purpose is to assist the user in the Connection tab when entering the URL or lookup name
- **Driver Class**
Defines the main class for the JDBC driver, used for connecting to the database.
- **Web Site**
Link to the DbVisualizer web site, where you can get up-to-date information about how to download the driver.
- **Driver File Paths**
Defines all paths to search for JDBC drivers or Initial Contexts when connecting to the database. The Driver File Paths area is composed of two tabs: the paths in the **User Specified** tab are used for dynamically loaded JDBC drivers or Initial Context classes, and the **System Classpath** tab lists all paths that are part of the Java system classpath.

The System Classpath tab is only of interest for the **JNDI/ODBC** driver.

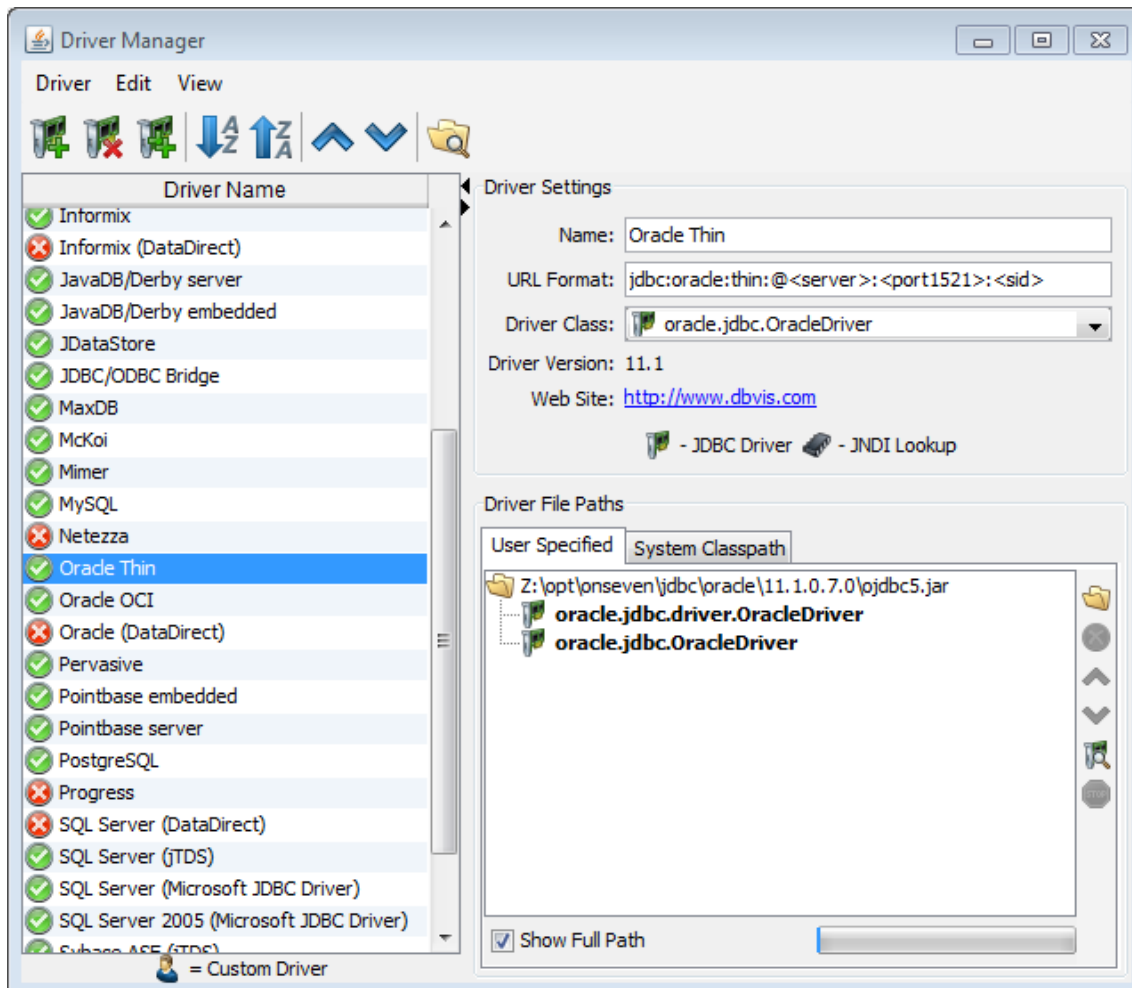


Figure: Driver Manager dialog

Initially, the driver list contains a collection of default drivers. They are not fully configured, as the paths to search for the classes need to be identified. You can edit the list, i.e., create, copy, remove and rename drivers. A driver is ready to use once a driver class has been identified, which is indicated with a green check icon in the list. Drivers that are not ready for use are indicated with a red cross icon.

Only ready (configured) drivers appear in the Connection tab driver list.

The figure shows seven drivers that are ready: **DB2 UDB, Informix, JDBC/ODBC Bridge, Mimer, MySQL, Oracle Thin and PostgreSQL.**

Setup a JDBC driver

The recommended way to setup a driver is to pick a matching driver name from the list and then simply load the JAR, ZIP or directory that keeps the driver class(es). For instances, if you are going to load the JDBC driver for **Oracle**, select the Oracle driver in the list. You can also create a new driver or copy an existing one.

Check the following online web page with the most current information about the tested databases and drivers:

<http://www.dbvis.com/products/dbvis/drivers.html>

- It lists which databases and drivers we have tested
- Download links to JDBC drivers
- Information of which files to load in the driver manager for each JDBC driver
- Information of which **Driver Class** to choose

When you have selected the driver to configure, you need to load the driver files. Click the **Load** button to the right of the **User Specified** paths tree to show the file chooser and load the driver JAR, ZIP or individual files.

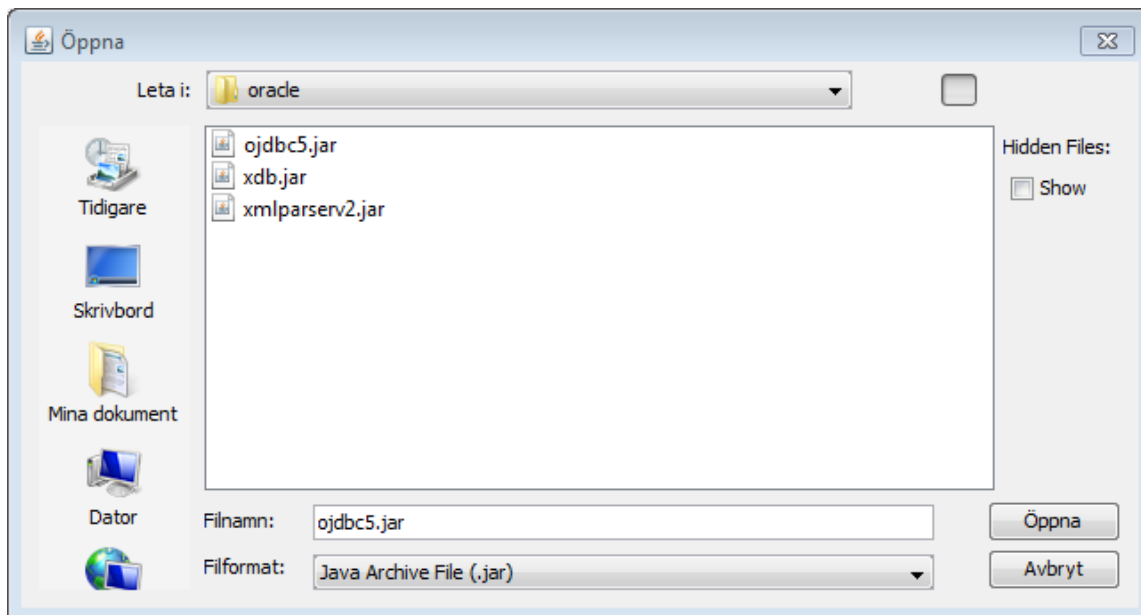


Figure: File Chooser dialog

A JDBC Driver implementation typically consists of several Java classes. If they are packaged in a JAR or a ZIP file, you don't have to worry about the details; just select and load the JAR or ZIP file. For instance, in the example above, use the **ojdbc5.jar** file.

If the driver classes are not packaged, it is important to select and load the root folder for the JDBC Driver. Java classes are typically organized using a package name structure. Example:

oracle.jdbc.driver.OracleDriver

Each package part in the name above (separated by ".") is represented by a folder in the file system. The root folder for the driver is the folder named by the first part, i.e., the **oracle** directory in this example. The class files are stored in the **oracle/jdbc/driver** sub folder. When the driver classes are located in a folder structure like this, you must select and load the root folder, so that the Driver Manager gets the complete package structure.

When a connection is established in the Connection tab, DbVisualizer searches the selected drivers path tree's in the following order:

1. User Specified
2. System Classpath

The paths are searched from the top of the tree, i.e., if there are several identical classes in, for example, the dynamic tree, the topmost class will be used. Loading several paths containing different versions of the same driver in one driver definition is not recommended, even though it works (if you do this, you must move the driver you are going to use to the top of the tree). The preferred method for handling multiple versions of a driver is to create several driver definitions.

When you load files in the User Specified paths list, DbVisualizer analyzes each file to find the classes that represent main driver classes. Each such class is listed under the path where it was found in the User Specified paths lists, and it is also added to the Driver Class list in the Driver Settings area above. If there is more than one class in the list, make sure you select the correct **Driver Class** from the list. Consult the driver documentation (or the [Databases and JDBC Drivers](#) page) for information about which class to select.

JDBC drivers that requires several JAR or ZIP files

Some drivers depend on several ZIP or JAR files, or directories. An example is if you want XML support for an Oracle database. In addition to the standard JAR file for the driver, you then also need to load two additional JAR files. These are not JDBC driver files but adds functionality the driver needs to fully support XML.

Simply select all JARs at once and press **Open** in the file chooser dialog. The Driver Manager will then automatically analyze each of the loaded files and present any JDBC driver classes or JNDI initial context classes it finds.

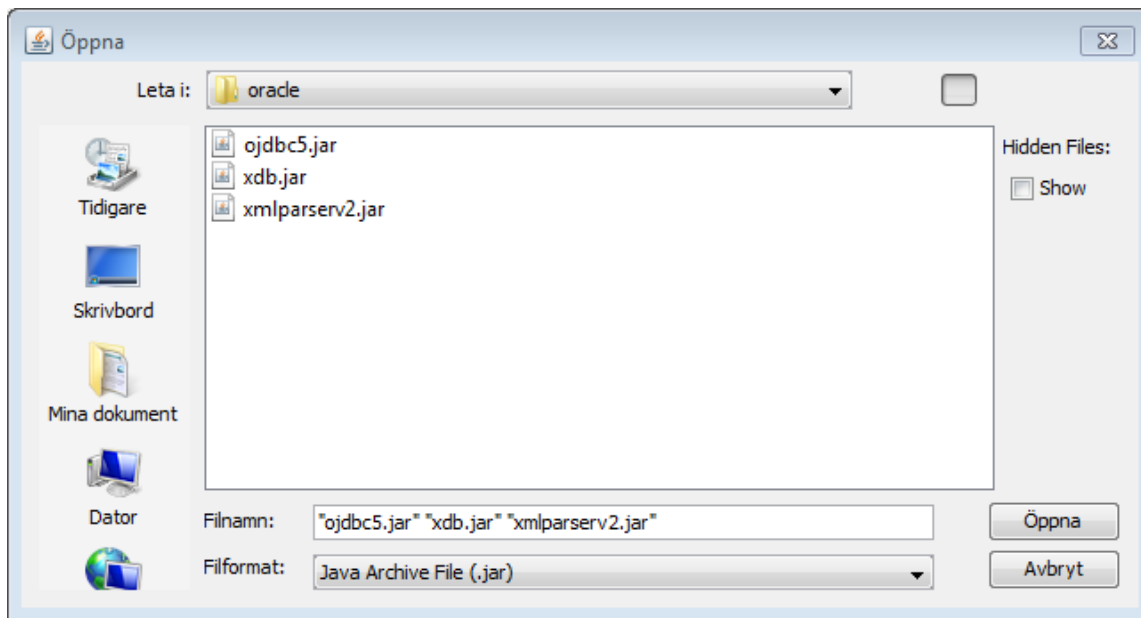


Figure: File Chooser dialog

The JDBC-ODBC bridge

The JDBC-ODBC driver is bundled with most Java installations, but not all (e.g., it is not included with Java for Max OS X). The **JdbcOdbcDriver** class is included in a JAR file that is commonly named **rt.jar**, stored somewhere in the Java directory structure. DbVisualizer automatically identifies this JAR file in the System Classpath tree. To locate the **JdbcOdbcDriver**, simply press the **Find Drivers** button to the right of the System Classpath tree. When it is found, make sure the **sun.jdbc.odbc.JdbcOdbcDriver** is selected as the **Driver Class** in the Driver Settings area.

Loading JNDI Initial Contexts

Initial Context classes are needed to get a handle to a database connection that is registered with a JNDI lookup service. In DbVisualizer, these context classes are similar to JDBC driver classes in that an Initial Context implementation for a specific environment is required.

Remember that the appropriate JDBC driver classes must be loaded into the Driver Manager even if the database connection is obtained using JNDI.

To load Initial Context classes into the Driver Manager, simply follow the steps outlined for loading JDBC drivers. The difference is that you will instead load paths containing Initial Context classes instead of JDBC drivers. When you load a path, DbVisualizer locates all Initial Context classes in the path and lists them in the User Specified paths list.

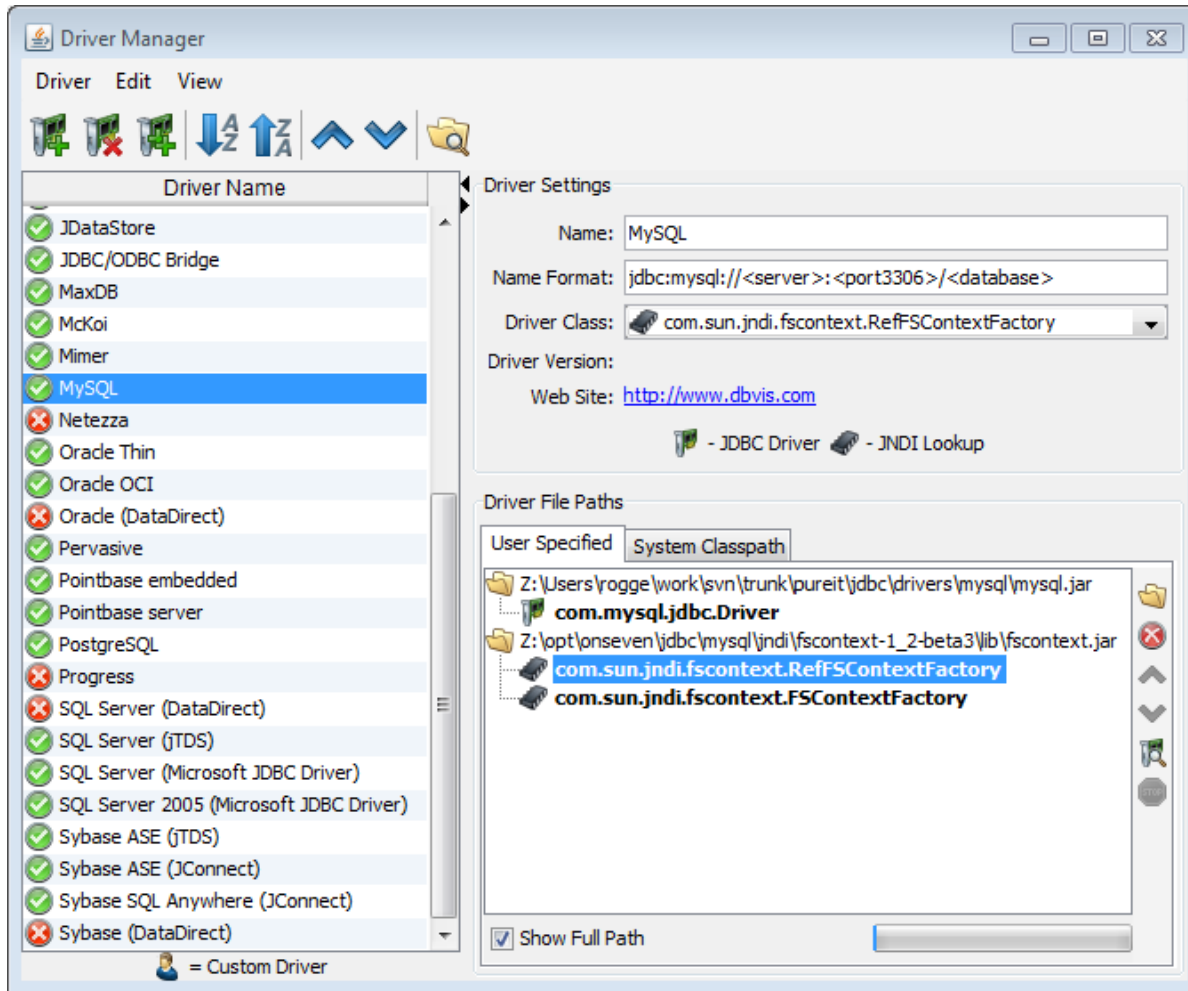


Figure: Driver Manager List with Initial Context classes

Visually, the difference between the identified JDBC drivers and Initial Context classes is the icon in the tree.

The figure shows the JAR files required to first obtain the JNDI handle, and then also the JDBC driver for the database. Check with the application server vendor or similar for more information about what files you need to load to get connected via JNDI.

Errors (why are some paths red?)

A path in red color indicates that the path is invalid. This may happen if the path has been removed or moved after it was loaded into the driver manager. Simply remove the erroneous path and locate the correct one.

Several versions of the same driver

The Driver Manager supports loading and using several versions of the same driver concurrently. We recommend that you create a unique driver definition per version of the driver and name the driver definitions properly, e.g., **Oracle 9.2.0.1**, **Oracle 10.2.1.0.1**, etc.

Setup a database connection

This section explains how to setup a Database Connection in the Connection tab.

Setup using JDBC driver

A Database Connection in DbVisualizer is the root of all communication with a specific database. It requires at a minimum that a driver is selected and that a **Database URL** is specified. A new Database Connection is created using the **Database->Add Database Connection** menu choice in the main window:

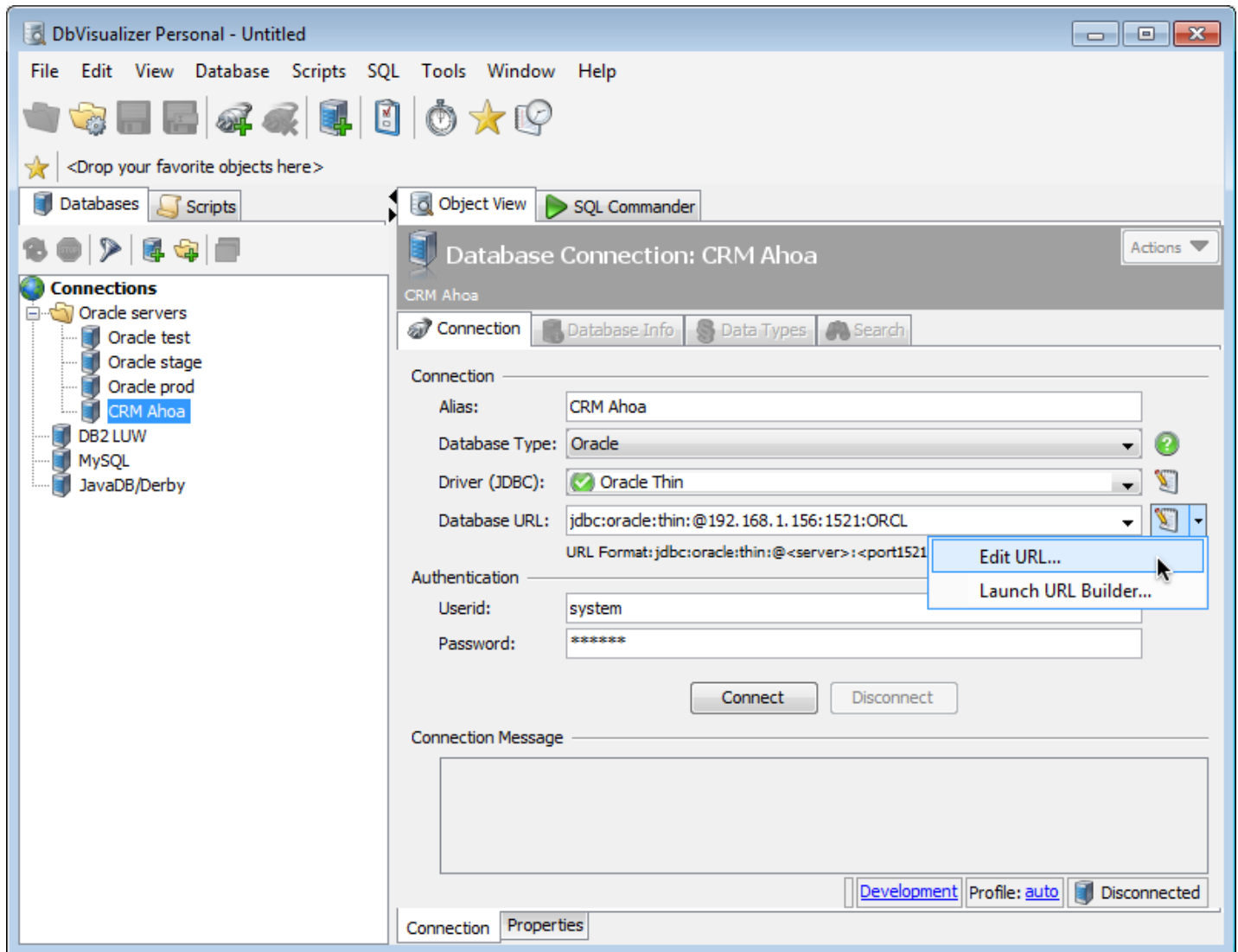


Figure: New Database Connection using JDBC driver

The **Connection** tab is the only tab that is enabled if you are not connected to the database. Database connection objects appear throughout the application and are by default listed by their **URL**. A URL can be, and often is, quite complex and long. You can use the **Database Alias** to set a more readable name for the database connection.

The **Database Type** list shows all database types that have a set of separate properties, which you can adjust in the **Tool Properties** dialog. Select the database type you are creating a connection for, or select Generic if you cannot find a matching type.

The **Driver** list shows all defined drivers that have been defined properly in the Driver Manager. Just open the list and select the appropriate driver. Clicking the button to the right of the field opens the Driver Manager dialog with the settings for the selected driver.

Enter the JDBC URL for the connection in the **Database URL** field. The drop-down menu to the right of the field provides two options for entering or editing the URL. **Edit URL** opens a multirow editor, in case your URL is extremely long. **Launch URL Builder** opens a dialog where you fill out a form with information about the connection, used to generate the URL for you when you close the dialog.

There is also a **URL Format** field under the URL field that shows the URL format that the driver supports. You can click on the format string to copy the format template into the URL field. Terms between < and > characters are placeholders that need to be replaced with appropriate values, e.g.:

```
jdbc:oracle:thin:@proddb:1521:bookstore
jdbc:sybase:Tds:localhost:2638
jdbc:db2://localhost/crm
```

```
jdbc:microsoft:sqlserver://localhost;DatabaseName=customers
```

Userid and **Password** are optional but most databases require that they are specified.

Some drivers accept additional proprietary parameters described in the [Connection Properties](#) section.

Setup using JNDI lookup

The information needed to obtain a database connection using JNDI lookup is similar to what is needed for connecting using a JDBC driver.

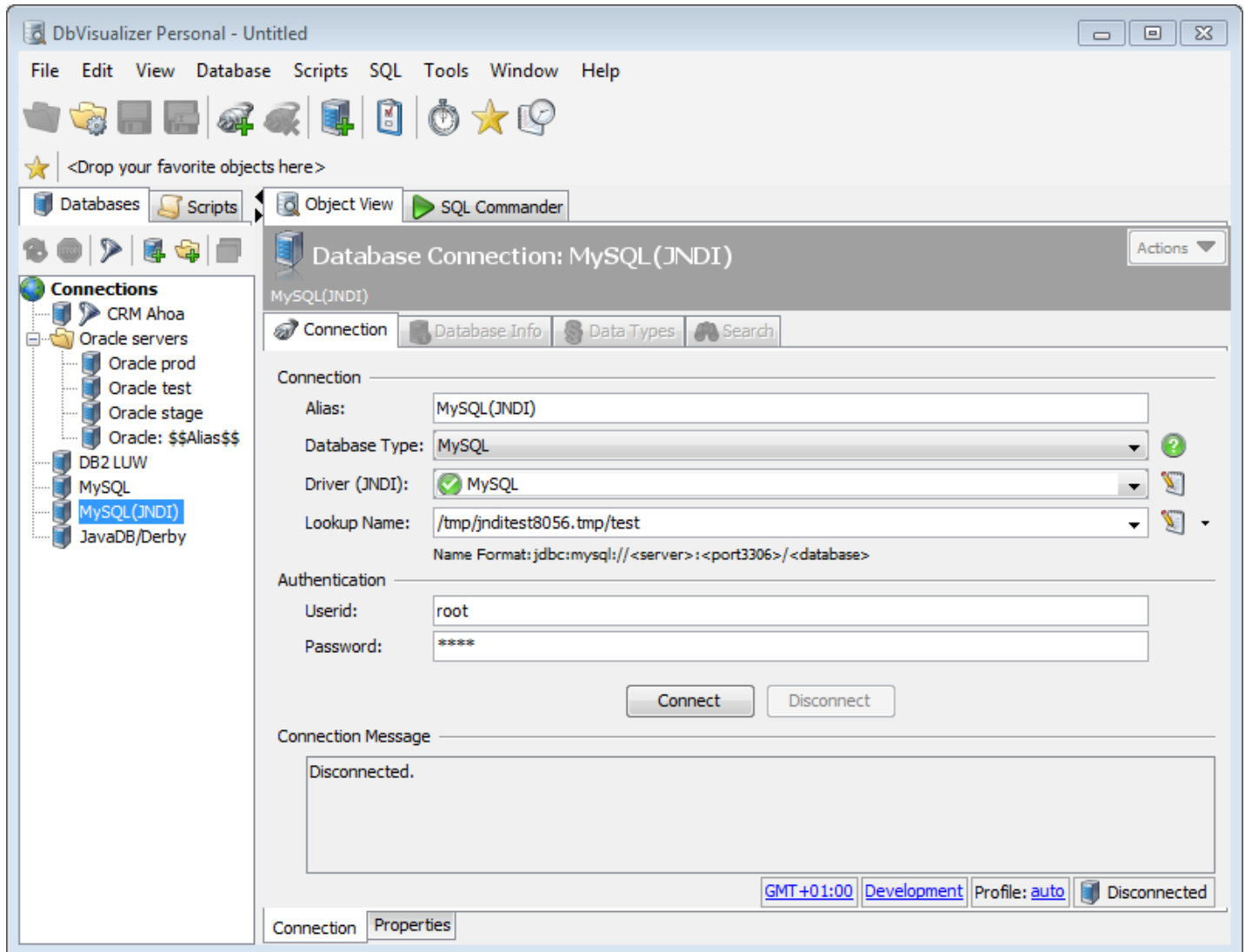


Figure: New Database Connection using JNDI lookup

The figure above shows parameters to connect with a lookup service via the MySQL RefFS driver. The `/tmp/jnditest4975.tmp/test` lookup name specifies a logical name for the database connection. This example is in its simplest form, since userid and password are not specified, nor where the database connection is finally fetched from. Any errors during the process of getting a handle to the database connection appears in the **Connection Message** area.

Connection Properties

In addition to the standard connection parameters (URL, Driver, Userid, Password, etc.), there are also a collection of connection properties. Which properties are available depends on the database type. Some database types have more properties than others. Which edition of DbVisualizer you use also affects which connection properties are available.

All supported database types (Oracle, Informix, Mimer, DB2, MySQL, etc.) are listed in the **Database** tab in the **Tool Properties** window. For each database type, there are a number of properties that are applied to any database connection of that type. This means, for instance, that a database connection defined as being a PostgreSQL database type will use the PostgreSQL properties defined in Tool Properties. The Connection Properties can then be used to override some settings specifically for one database connection. The advantage with this inheritance model is that property changes that apply to all connections can be made in one place, instead of having to apply a common setting for every database connection of a specific database type.

The following summarize the organization of the properties:

- **Tool Properties (Database)**
These apply to all database connections of the specific database type.
- **Connection Properties**
These apply for a specific database connection only.

"Okay, so there are two places to change the value of a property. Which shall I use?"

This depends on whether the change should be applied to all database connections for a specific database type or just a single one. If the majority of your database connections should use the new property, it is recommended to set it in Tool Properties. Any overridden properties in the Connection Properties tab are indicated with an icon in the **Properties** tab label.

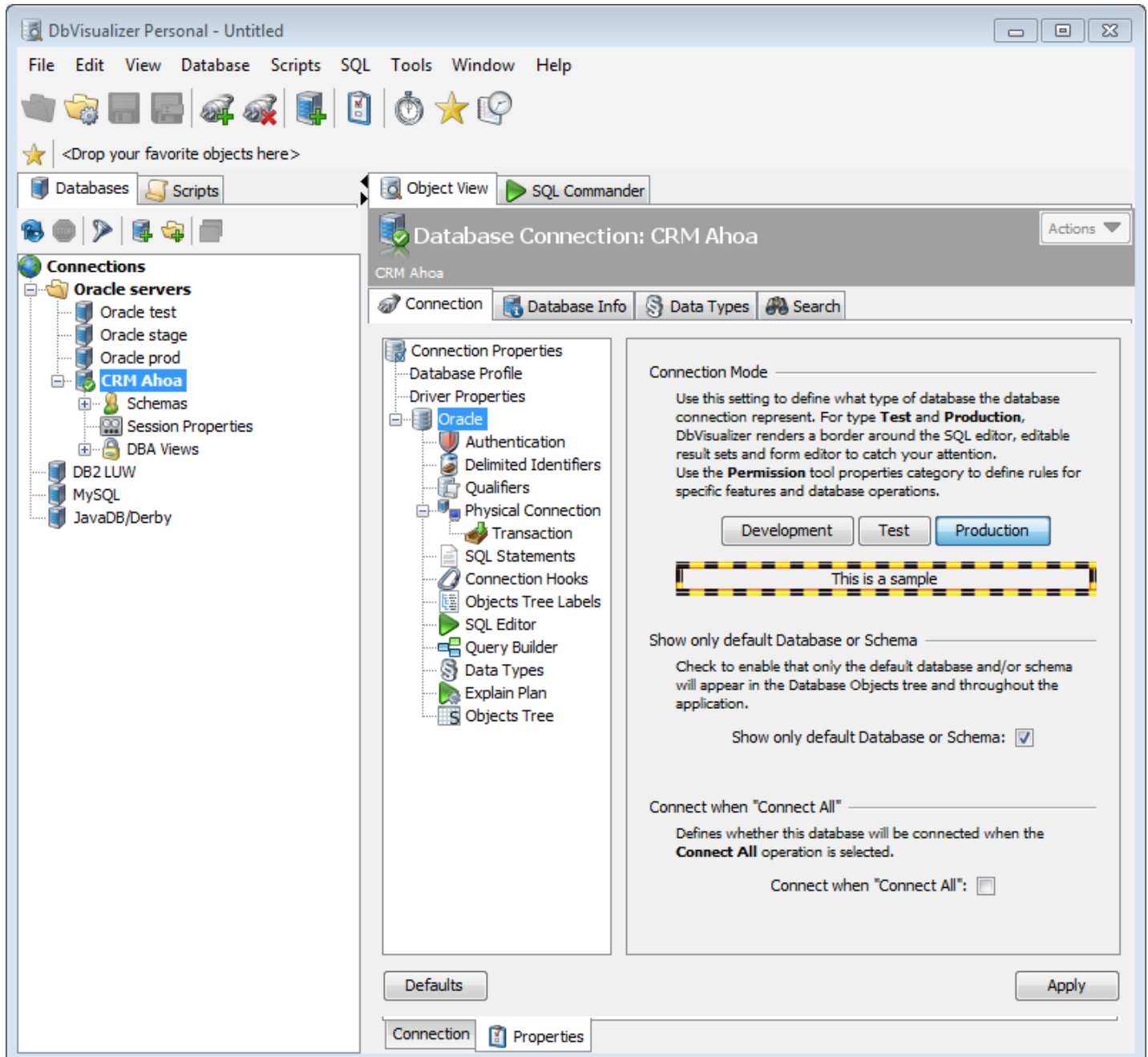


Figure: Connection Properties

The Connection Properties tab is organized in the same way as the Tool Properties window. The difference is that the list only includes the categories that are applicable for the selected database connection. Briefly, the categories are:

- Database Profile
- Driver Properties
- MySQL (The current Database Type)
 - Authentication
 - Delimited Identifiers
 - Qualifiers
 - Physical Connection
 - Transaction
 - SQL Statements
 - Connection Hooks
 - Objects Tree
 - SQL Editor
 - Query Builder

The **Database Profile** and **Driver Properties** categories are only available in the Connection Properties tab and not in Tool Properties. The next section explains the Database Profile and Driver Properties categories, while the other categories are described in the [Tool Properties](#) document.

Additional categories may appear in the connection properties depending on the type of database. An example is the category for **Explain Plan** for Oracle, DB2 and SQL Server.

Database Profile

Please read in the [Database Objects Explorer](#) document for detailed information about database profiles.

The Database Profile category is used to select whether a profile should be automatically detected and loaded by DbVisualizer, or if a specific one should be used for the database connection. The default strategy is to **Auto Detect** a database profile.

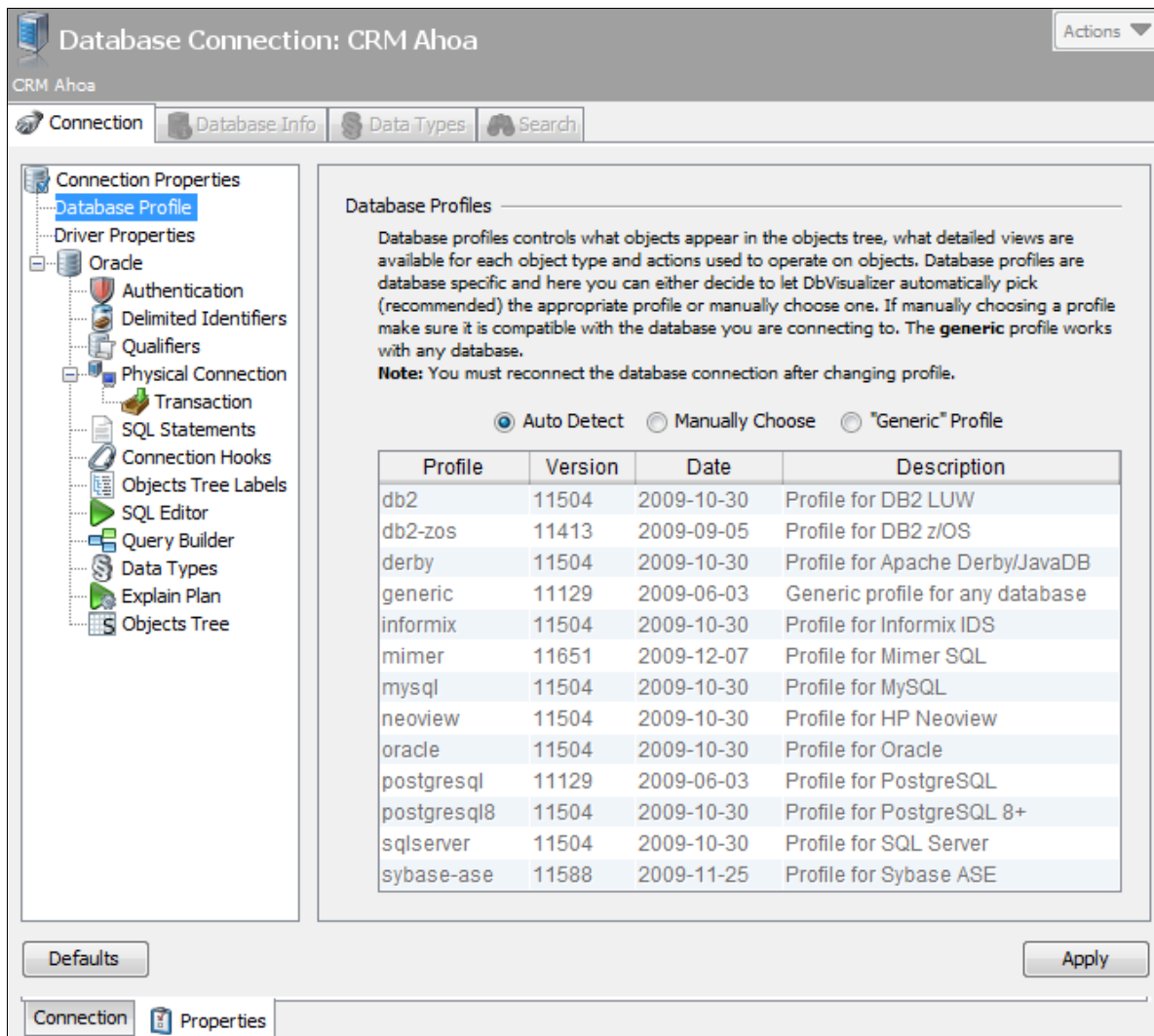


Figure: Database Profile category for a database connection

The way DbVisualizer auto detects a profile is based on the setting of **Database Type** in the connection details.

If you manually choose a database profile, this choice will be saved between invocations of DbVisualizer.

Driver Properties

The Driver Properties category is used to fine tune a driver or Initial Context before the database connection is established.

Driver Properties for JDBC Driver

Some JDBC drivers support driver specific properties that are not covered in the JDBC specification.

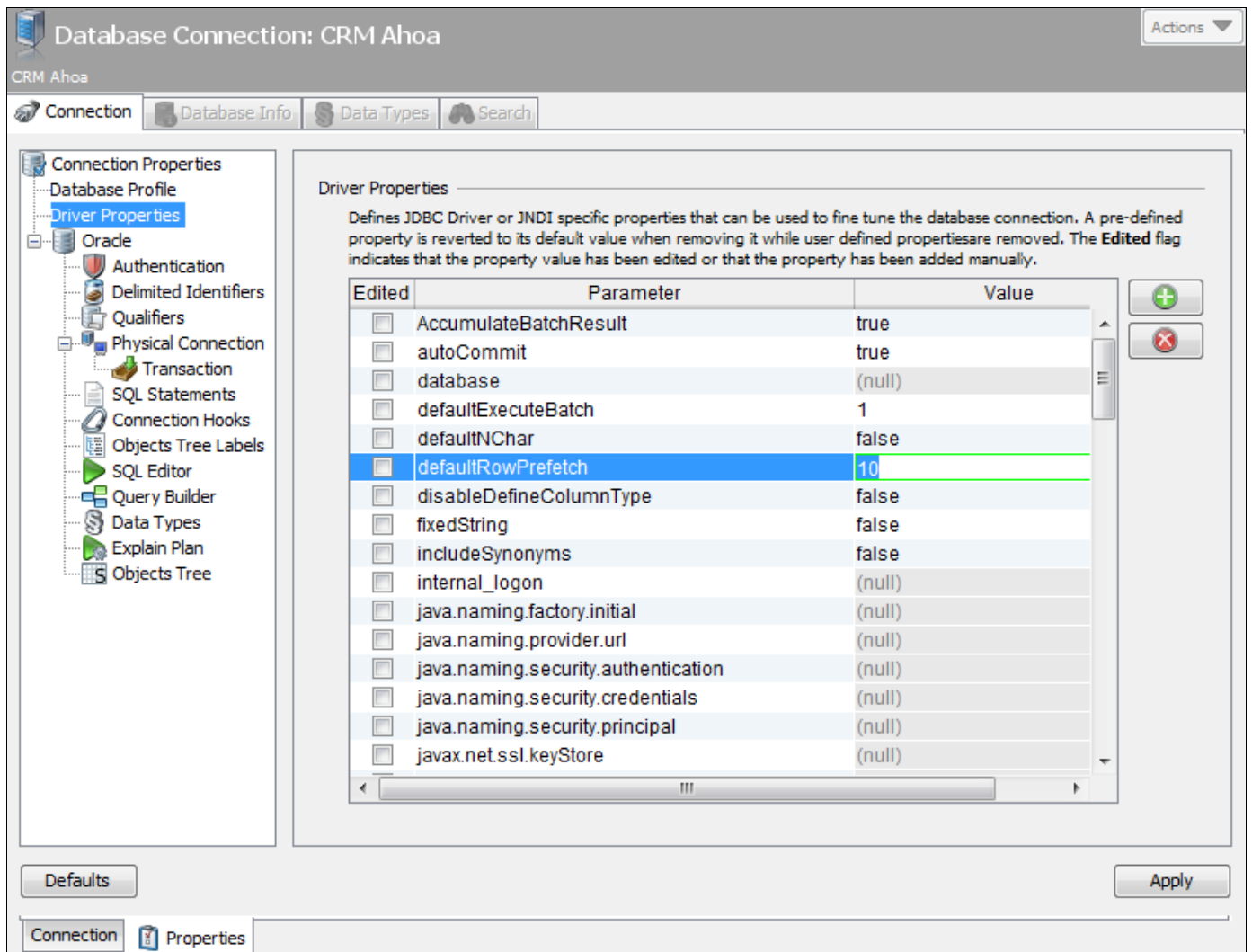


Figure: Driver Properties for JDBC Driver

The list of parameters, their default values and parameter descriptions are determined by the JDBC driver used for the connection. Not all drivers supports additional driver properties. To change a value, just modify it in the list. The first column in the list indicates whether the property has been modified or not, and so, whether DbVisualizer will pass that parameter and value onto the driver at connect time.

New parameters can be added using the buttons at the bottom of the dialog. Be aware that additional parameters do not necessarily mean that the driver will do anything with them.

Driver Properties for JNDI Lookup

The Driver Properties category for a JNDI Lookup connection always contains the same parameters.

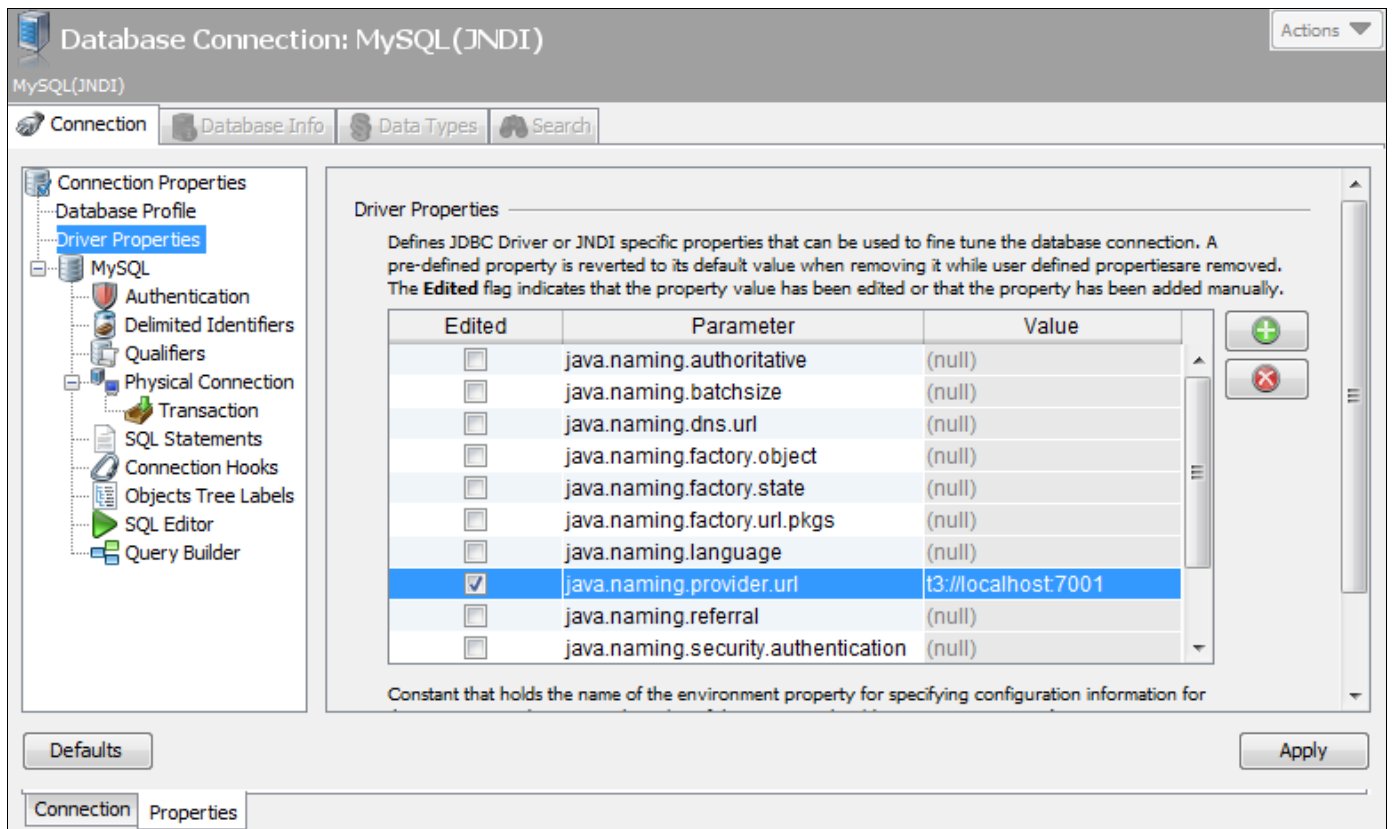


Figure: Driver Properties for JNDI lookup

The list of options for JNDI lookup is determined by the constants in the `javax.naming.Context` class. To change a value, just modify the value of the parameter. The first column in the list indicates whether the property has been modified or not, and so, whether DbVisualizer will pass that parameter and value onto the driver at connect time. New parameters can be added using the buttons at the bottom of the dialog. Be aware that additional parameters do not necessarily mean that the InitialContext class will do anything with them.

Always ask for userid and/or password

Userid and password information is generally information that should be handled with great care. By default, DbVisualizer saves both userid and password (encrypted) for each database connection. Userid is always saved while password saving can be disabled in the connection properties, in the Authentication category.

The **Require Userid** and **Require Password** connection properties in the Authentication category can be enabled to tell DbVisualizer to automatically prompt for userid and/or password when a connection is to be established. Enabling either one or both of these while leaving the **Userid** and **Password** fields blank for a database connection ensures that DbVisualizer will not keep this vital information between sessions. The following dialog is displayed if requiring both userid and password.

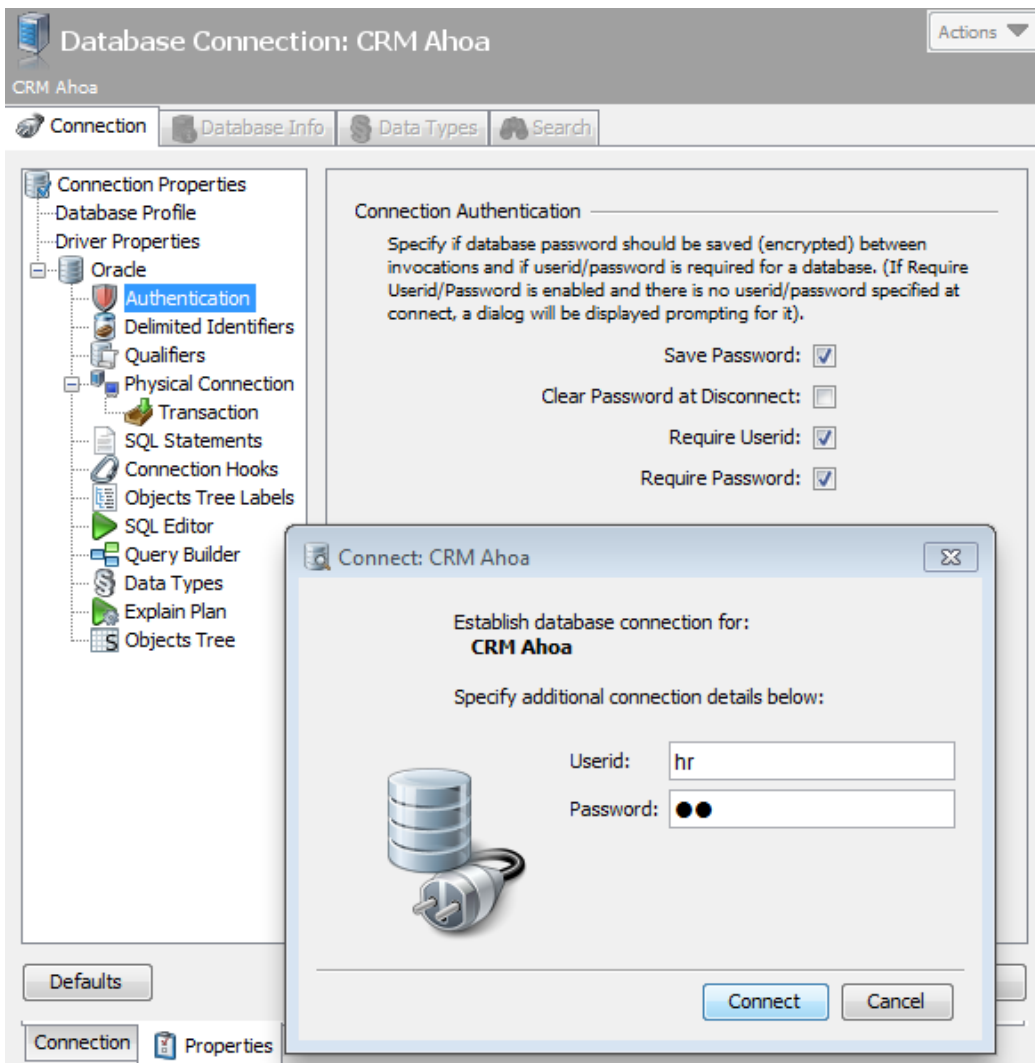


Figure: Dialog asking for Userid and Password as a result of having Require Userid and Password settings enabled

Using variables in the Connection details

Variables can be used in any of the fields in the Connection tab. This can be useful alternative to having a lot of similar database connection objects. Several variables can be in a single field, and default values can be set for each variable. The following figure shows an example with variables, i.e., variable named delimited by dollar characters, `$$...$$`.

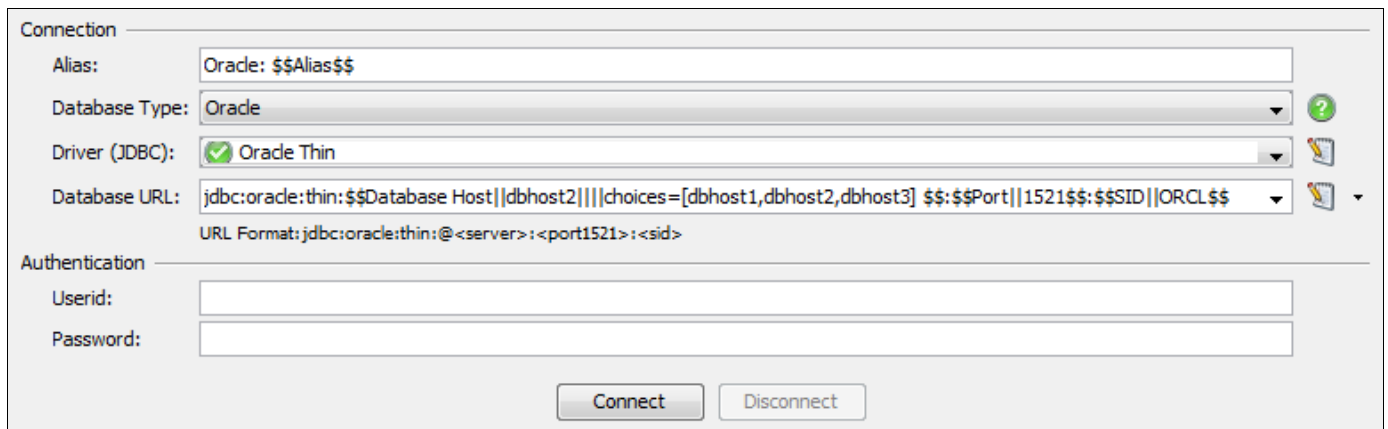


Figure: Connection tab with variables

The following variables appear in the figure:

- `$$Alias$$`
- `$$Database Host||dbhost2|||choices=[dbhost1,dbhost2,dbhost3] $$`
- `$$Port||1521$$`
- `$$SID|ORCL$$`

All of these variables define a default value after the "||" delimiter, except for the `$$Alias$$` variable, which have no default value. The default values appear in the connect dialog when you ask for a connection to be established. The `$$Database Host$$` variable includes the **choices** option, with a comma separated list of choices that should appear in a drop-down list. The drop-down list is editable, so the user is not locked into the choices from the list.

The following figure shows the connect dialog based on the connection definition shown above.

Using variables in conjunction with the **Require Userid** and/or **Require Password** settings is also supported.

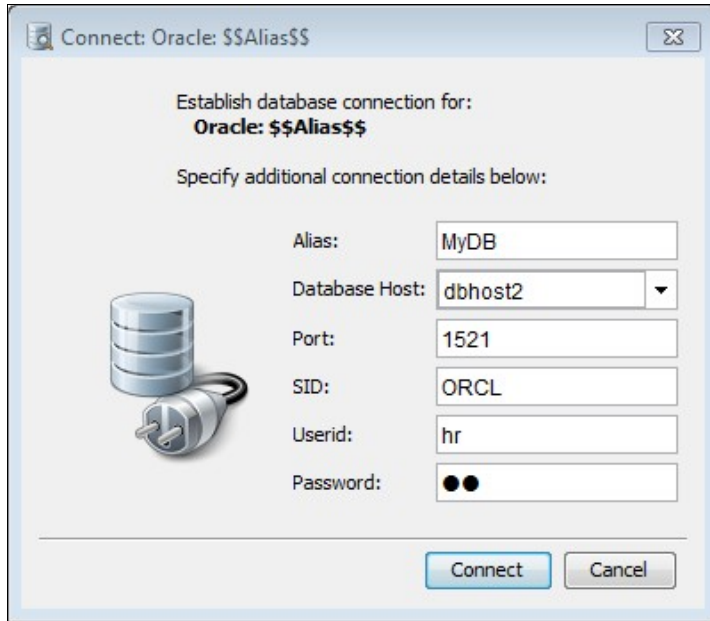


Figure: Connection tab with variables

Enter the appropriate information in the fields and then press the **Connect** button to establish the connection. When the connection is established, DbVisualizer automatically substitutes the variables in the Connection tab with the values entered in the connect dialog. At disconnect from the database, they revert back to the original variable definitions.

Connect to the Database

Press **Connect** when all information has been specified. DbVisualizer passes all information you entered on to the selected driver, and when the connection is established, the following appears.

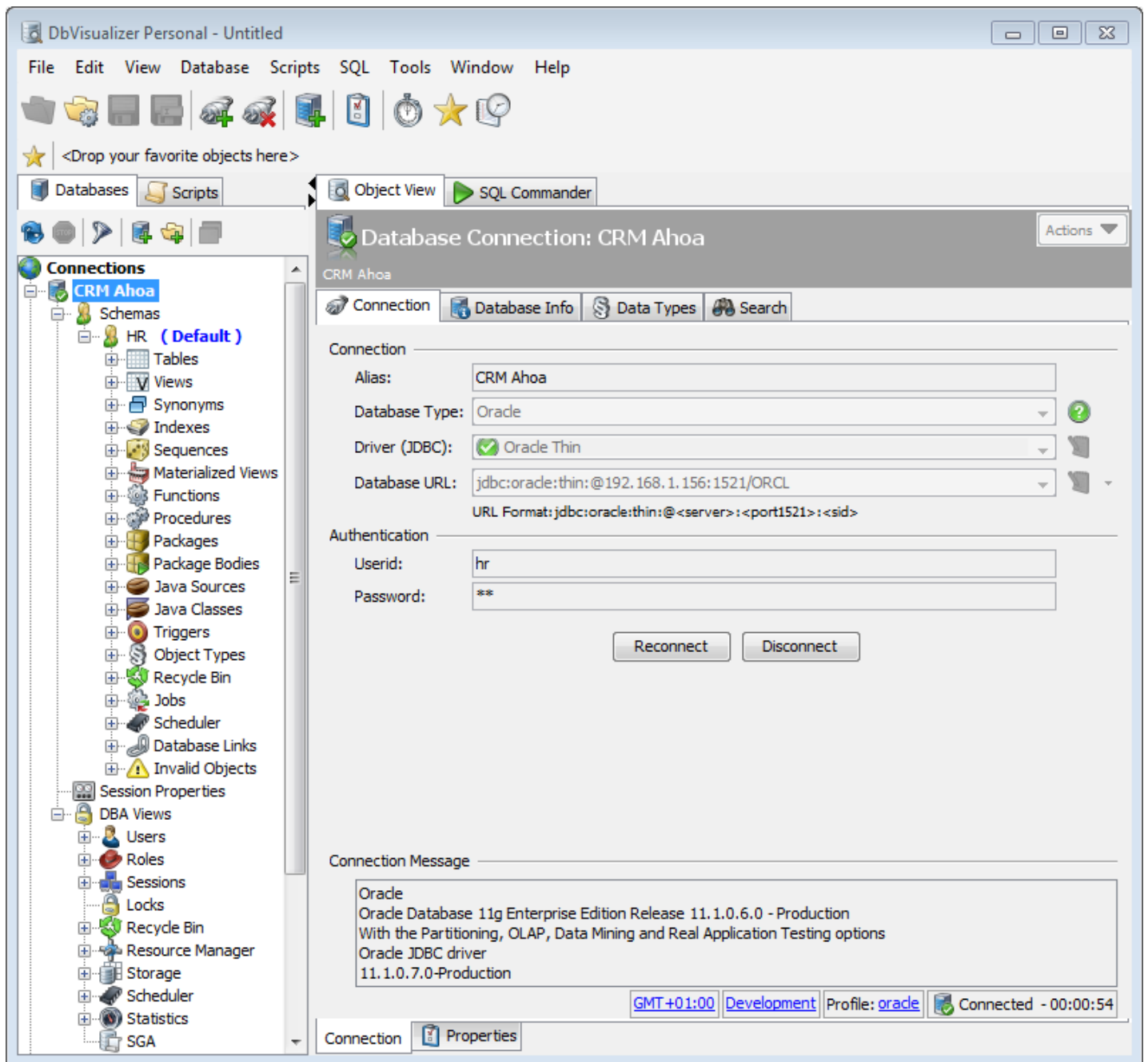


Figure: A freshly initiated database connection using JDBC driver

The **Connection Message** box now lists the name and version of the database as well as the name and version of the JDBC driver. The database connection node in the tree indicates that it is connected. The connection properties cannot be edited while a database connection is established.

The figure above also shows that the database connection node in the tree has been expanded to show its child objects.

If the connection is unsuccessful, it is indicated by an error icon in the tree. The error message as reported by the database or the driver appears in the **Connection Message** area. Use this information to track down the actual problem. Since these conditions are specific for the combination of driver and database, you should check the driver and database documentation to find out more. Below are a few common problem situations:

Error Message

Explanation

No suitable driver.
 There is no driver that can handle a connection for the specified URL. The most common reason is that the driver is not loaded in the Driver Manager. Also make sure the URL is correct spelled.

The JDBC support in Java determines what driver to load based on the database URL. If the URL is malformed then there might be no driver that is able to handle the database connection based on that URL. This error is produced when this situation occurs or when the driver is not loaded in the driver manager. The recommendation is to check the JDBC driver documentation for the correct syntax.

java.sql.SQLException: Io exception: Invalid number format for port number
 Io exception: Invalid number format for port number

The URL templates that are available in the Database URL list contains the "<" and ">" place holders. These are there to indicate that the value between them must be replaced with an appropriate value. The "<" and ">" characters must then be removed.

This example error message is produced by the Oracle driver when using the following URL: **jdbc:oracle:thin:@qinda:<1521>:<fuji>**

Simply remove the "<" and ">" characters and try again.

Connections Overview

The Connections overview is displayed by selecting the **Connections** object in the Database Objects Tree. This overview displays all database connections in a list and is handy to get a quick overview of all connections. In addition to the Alias, Profile, URL, driver, etc. there are a few symbols describing the state of each connection. Double clicking on a connection changes the display to show that specific connection.

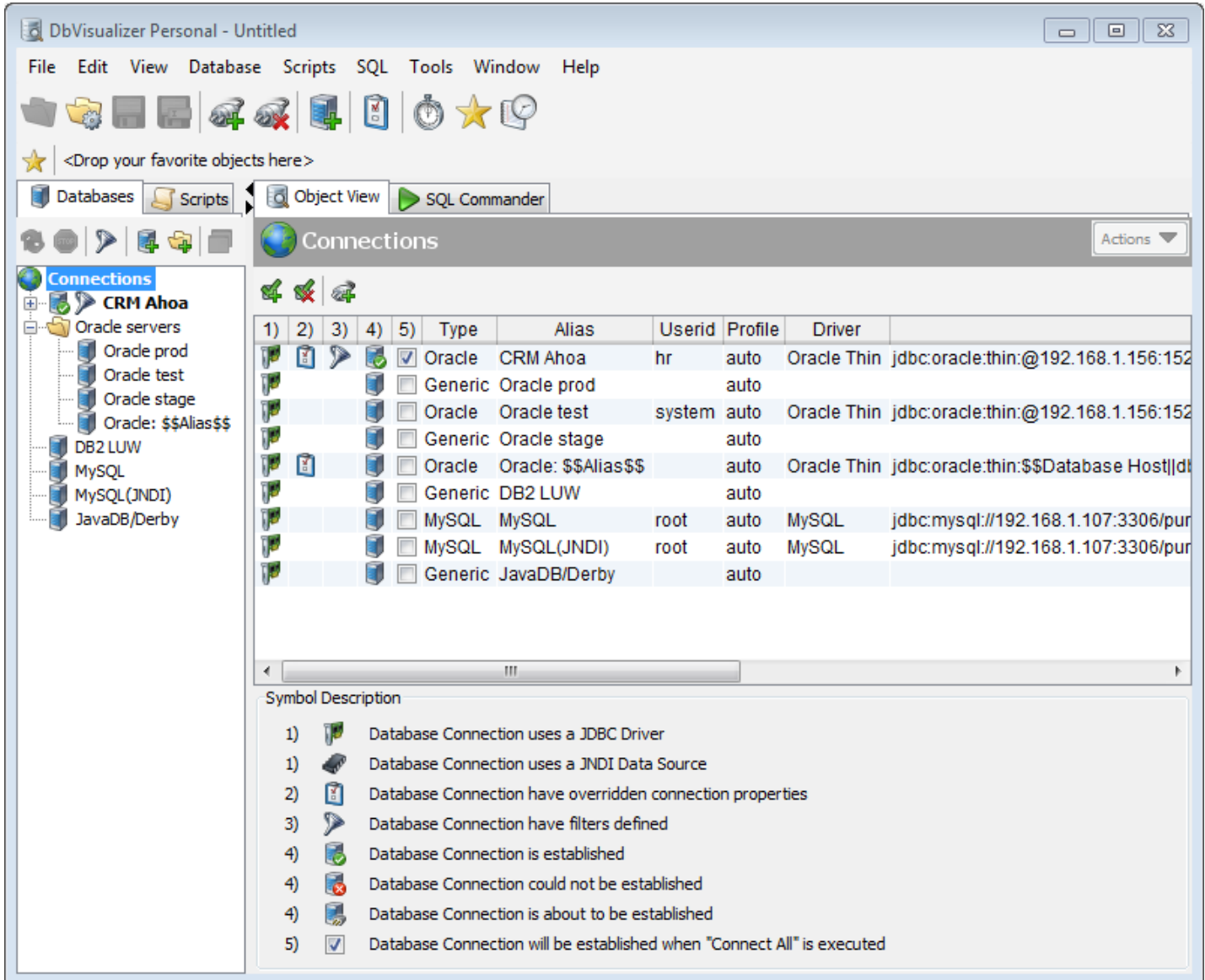


Figure: The Connections Overview

Information for each symbol is provided in the description area below the list. The fifth check symbol is the only editable symbol and is used to set the state of the **Connect when Connect All** property, i.e., whether the database connection should be connected when selecting the **Database->Connect All** menu choice.

Click the **Type** column for an entry to modify its **Database Type**.

Database Objects Explorer

Introduction

The **Database Objects Tree** is used to explore databases and browse details about objects. Which object types may be explored and which object actions exist are database dependent.

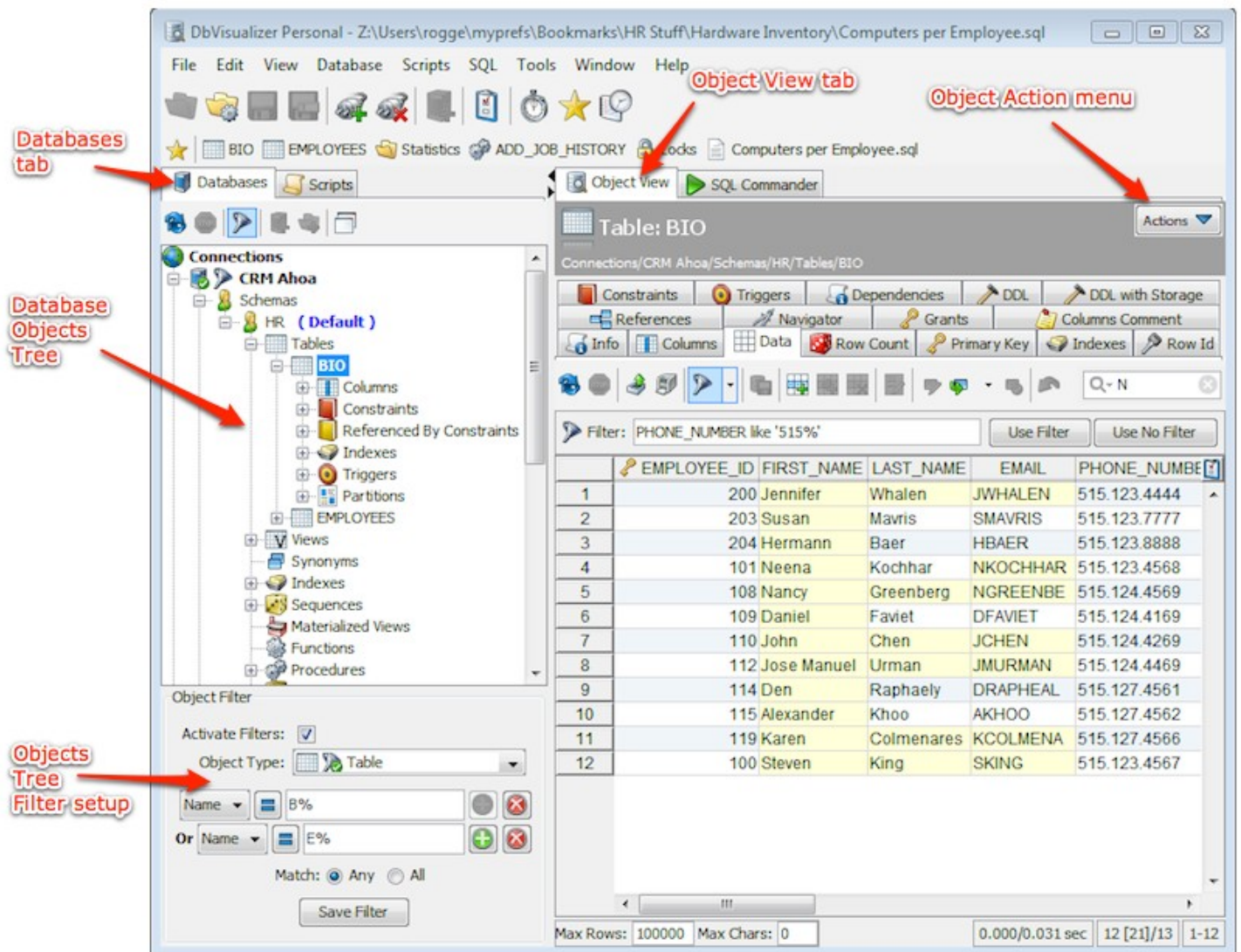


Figure: Database Objects tab

The **Databases tab** to the left is the place to setup new database connections and establish connections. Once connected, expand the database connection object and explore the objects available. The **Object View** area to the right displays detailed information about the currently selected object in the tree.

The **Filter setup** pane below the tree is used to control which objects are displayed in the tree. It comes in handy when you have many schemas or tables in your database and want to limit the number of visible objects.

For some object types, there are **actions** (small dialogs for performing a task) for common operations, such as creating, altering, and dropping database object. Which actions are available depends on the database you are connected to and the database profile used for the connection. More about this in sections below.

All object names in the tree can be dragged to any editable text fields, including to the SQL Commander editor.

Create a Database Connection

There are a few objects that always appear in the tree independent of the edition of DbVisualizer and the `database_profile` in use. The most important object is the **Database Connection**, which is used to setup and establish a database connection. The other two objects are **Folder** and **Connections Overview**. The following sections describe these objects in more detail.

Database Connection object

The **Database Connection** object is the root object for a connection. Before exploring or accessing a database, you need to establish the connection. Create a new database connection using the **Database->Add Database Connection** main menu choice and the following will appear.

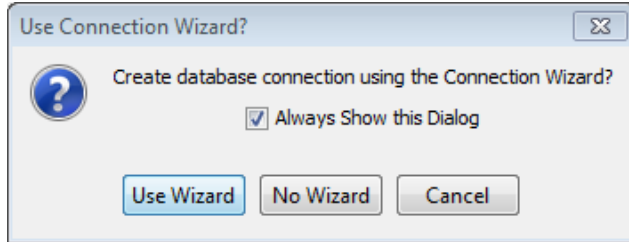


Figure: Add database connection

We recommend that you always use the connection wizard when you create a new database connection, as it hides the complexity of loading drivers and the syntax of database URLs (detailed information on how to establish a connection is provided in the [Load JDBC Driver and Get Connected](#) document).

Once a database connection has been setup properly, you just need to double-click on the database connection to establish a connection. You can use the **Database->Connect All** main menu choice to connect all enabled database connections with a single click. You make a database connection "Connect All" -aware in the **Database Properties** or in the **Connections** overview.

Alias

The name of the database connection object as it appears in the tree is by default "Database Connection". The **Connection Alias** can be used to provide a name that is more descriptive. Enter the new name in the Alias field in the Connection sub tab.

Default database and schema

The **(Default)** indicator after the name of a database or schema in the tree indicates that it is the default database or schema. The default is determined when you connect to the database.

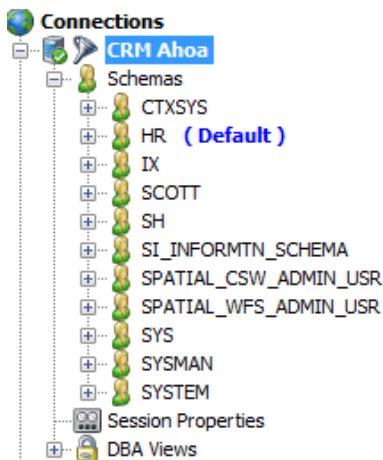


Figure: The (default) indicator for database and schema objects

Right-click while the database connection node is selected and then chose **Show Only Default Database/Schema** to limit the display to only show

your default database/schema.

Remove and copy database connection objects

To remove a database connection, select the **Database->Remove Database Connection** operation in the main menu. You can copy a database connection with **Database->Duplicate Database Connection**.

Database Connection detailed information

The following section describes the tabs in the Objects View for a database connection briefly.

Tab	Description
Connection	This tab is always enabled and is used to setup the details for a database connection. You can also connect, disconnect and reconnect using the buttons in this tab.
Database Info	When connected, the Database Info tab shows various information supplied by the driver. Much of this information is low level, even though some of it may be useful.
Data Types	The Data Types tab lists all data types supported by the database.
Search	The Search Tab is used to search among the objects in the tree. Search operates on the content in the tree. See the next section for more information about search.

Search

The Search tab is used to search among the objects in the tree by object name. Note that if you have tree filters or any other property that limits the content of the tree enabled, the search is performed only for those objects that match the filters. The types of objects that are searchable depends on the database you are connected to. For instance, columns are included in the tree for some databases but not for others.

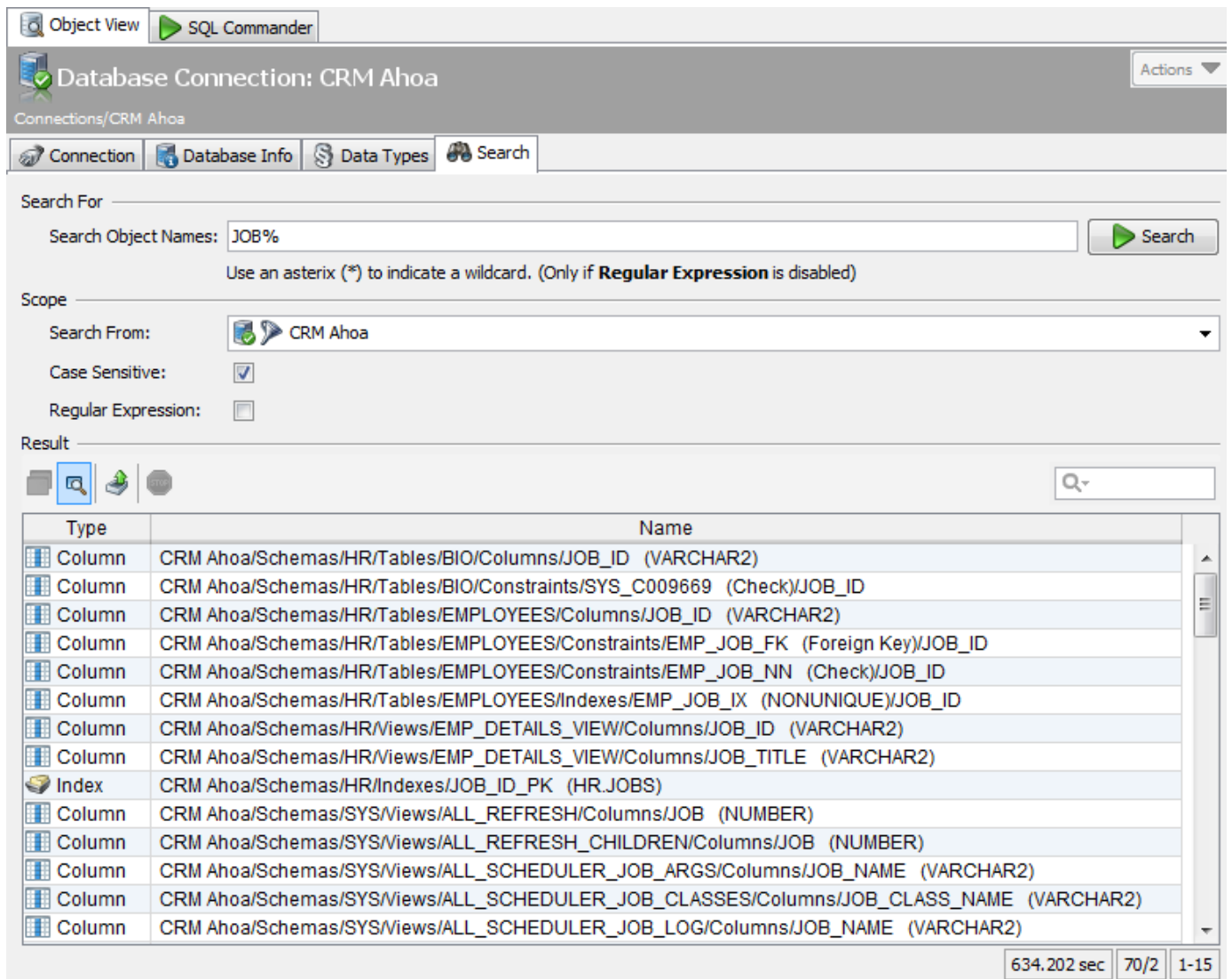


Figure: The Search tab

Search by specifying the name of the object, or name pattern, and press the **Search** button. You can use asterisk (*) as a wildcard in a pattern, or you can use a regular expression pattern if you enable it by checking the Regular Expression checkbox. You can also specify where in the tree to start the search, and whether to do a case sensitive search.

You can interrupt a search operation with the **Stop** button in the grid toolbar. Use the **Show Object Path** toolbar toggle button to include or exclude a column for the complete path for each found object in the grid. This path is the same as if navigating to each object manually in the objects tree. Other grid toolbar buttons let you export and print the search result grid.

The search may take some time to perform the first time.

Double-click on a row to see detailed information about a specific object in a separate window.

Organizing Database Connections in Folders

If you work with many database connections, you can use folder objects to organize and group them in the tree. Folder objects can have child folder objects in an unlimited hierarchy. Use the **Database->Create Folder** and **Database->Remove Folder** menu choices to create and remove folder objects. You can use the **Database->Move Up/Down** main menu choices to move the folders (and database connections) in the tree, or you can just drag and drop to the nodes to a new location.

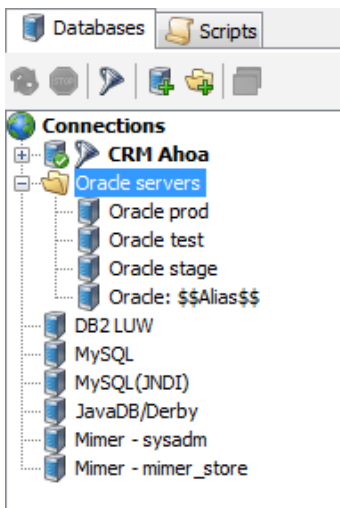


Figure: The database objects tree and the folder object type

Connections overview

The **Connections** object is the root object in the tree and acts as a holder for all database connections and folders. When selected, it displays an overview of all database connections in the Object Details view. Here you can see the basic settings and states for your database connections. For more information, see the [Load JDBC Driver and Get Connected](#) chapter.

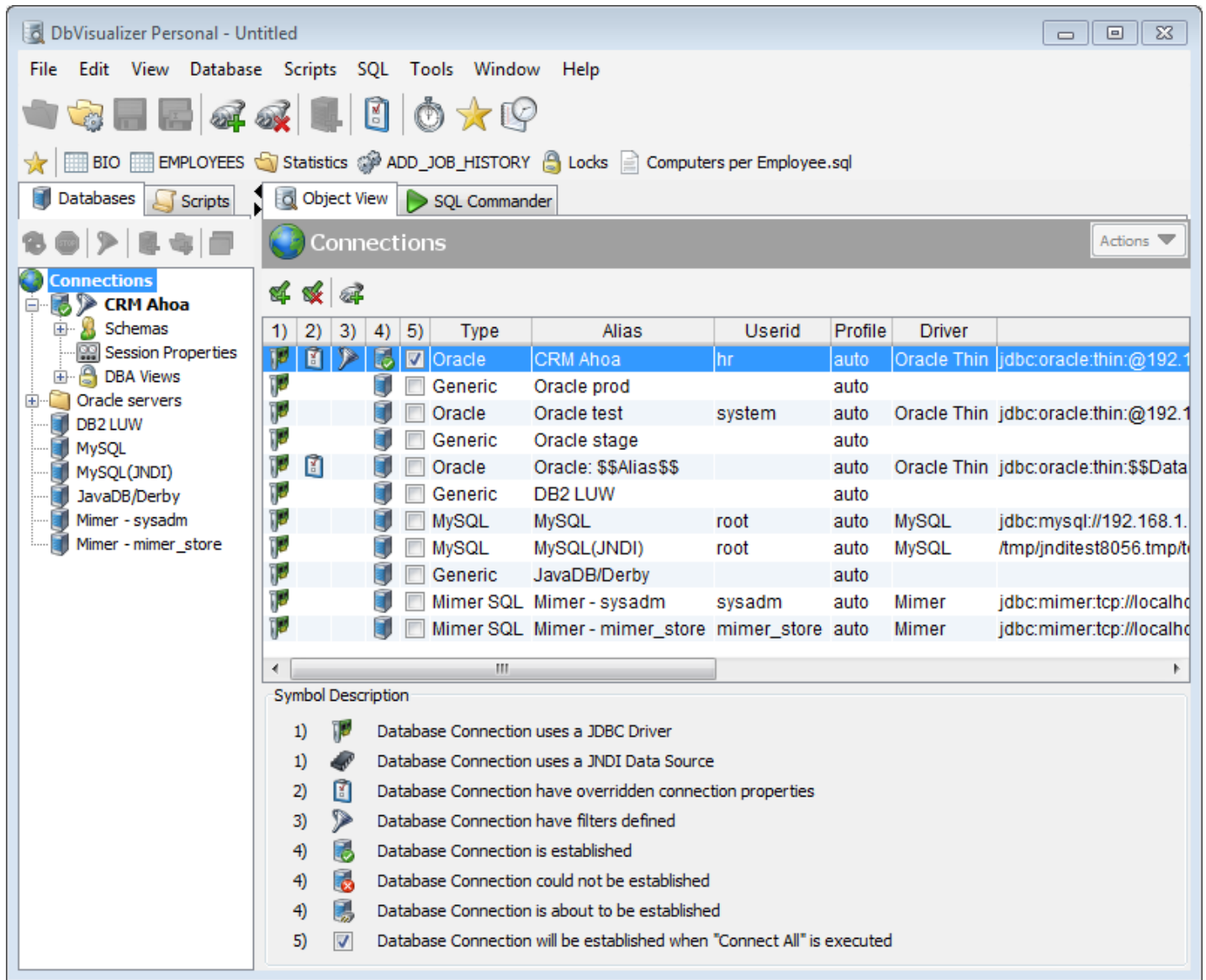


Figure: Connections object

Database Objects Tree

Standard Actions

The Database Objects toolbar buttons are used to do tree related operations. These are individually enabled or disabled based on the currently selected object.



Figure: Objects tree toolbar

Description of the buttons from the left:

Tool bar button	Description
Reload	Reloads the currently selected object by asking for new information about the object from the database. This is useful if

new objects have been created or removed.

Stop	Stops the current tree operation, for instance connecting to a database or expanding a node.
Show/Hide Tree Filter	Is a toggle button that determines whether the Filter management pane will be displayed below the tree.
Create Database Connection	Adds a new Database Connection object in the tree. The location of the new object is determined based on the current selection. If no node is selected, the new is object added at the end of the list.
Create Folder	Creates a new folder object.
Show in Window	Request to display the details view for the selected object in a separate window.

The right-click menu for an object and the **Database** main menu lists object specific actions. The following actions are always available for all objects:

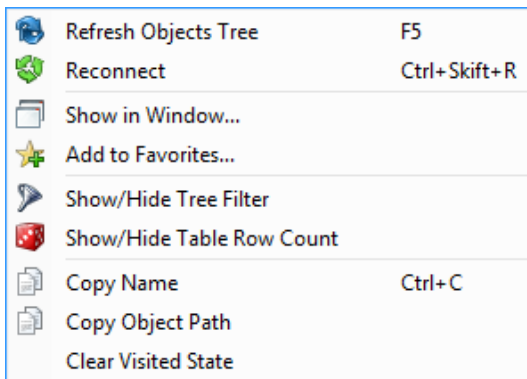


Figure: Standard right click menu actions for all objects

Object Actions

An object in the objects tree may have object specific actions attached to it. These actions are accessible via any of:

- Right-click menu in the objects tree
- Via the **Database->Selected Object** main menu
- Via the **Actions** menu button in the object view

Here is an example of the actions menu launched via the Actions menu button:

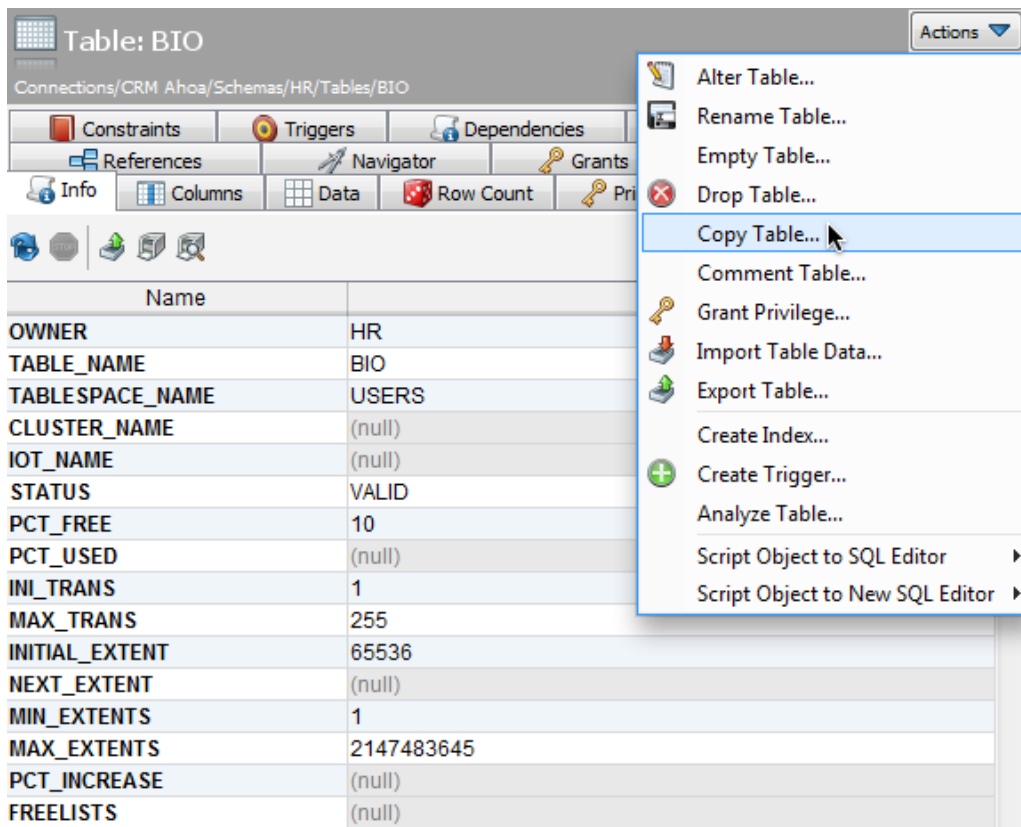


Figure: Object actions menu

Common Object Actions

There are a few actions that appear for some object types in all database profiles. These are most often valid for plain table object types and offer related functionality. Read the following sections for more information.

Create Table

The Create Table action launches the Create Table feature. You use it to create a table, optionally with a primary key, foreign keys and other constraints. Read more about this feature in [Create and Alter Table](#).

Create Index

The Create Index action launches the Create Index assistant dialog, where you can select columns to include in a new index for a table.

Import Table Data

Import Table Data launches a dialog where you can specify a CSV file to be imported into a table. Various configurations for how the source file is organized and data mapping are offered. Read more in [Export and Import](#).

Export Table

Export Table launches a dialog for exporting the DDL and/or data for the table. Read more in [Export and Import](#).

Script Object to SQL Editor

Use this action to create pre-defined SQL statements based on the source table and its columns. The created statement is copied to the current SQL editor in the SQL Commander. Here are a couple of examples:

Script Object to SQL Editor -> **Select**

```
SELECT
  COUNTRY_ID,
  COUNTRY_NAME,
  REGION_ID
FROM
  HR.COUNTRIES
```

Script Object to SQL Editor -> **Insert**

```
INSERT
INTO
  HR.COUNTRIES
(
  COUNTRY_ID,
  COUNTRY_NAME,
  REGION_ID
)
VALUES
(
  ' ',
  ' ',
  0
)
```

For databases with DbVisualizer [database specific](#) profiles, the **Script Object to SQL Editor** action menu also contains an entry for generating the **DDL** for **Table** and **View** objects.

Script Object to New SQL Editor

This is the same as **Script Object to SQL Editor**, except that the SQL is copied to a new SQL editor instead of to the current editor.

Objects Tree Filtering

The **Filtering** setup is activated via the **Database->Show/Hide Tree Filter** menu choice and the filter pane appear below the objects tree. Filtering is useful to limit the number of objects that will appear in the tree.

Tree filters are managed per database connection object. What can be filtered is defined per database profile. The generic database profile supports filtering of database (catalog), schema, table and procedure names.

The unfiltered schema objects for an Oracle connection.

The same objects but now filtered based on all schema names starting with "O" or "S".

Filter defined as all names that do not start with "O" and "S".

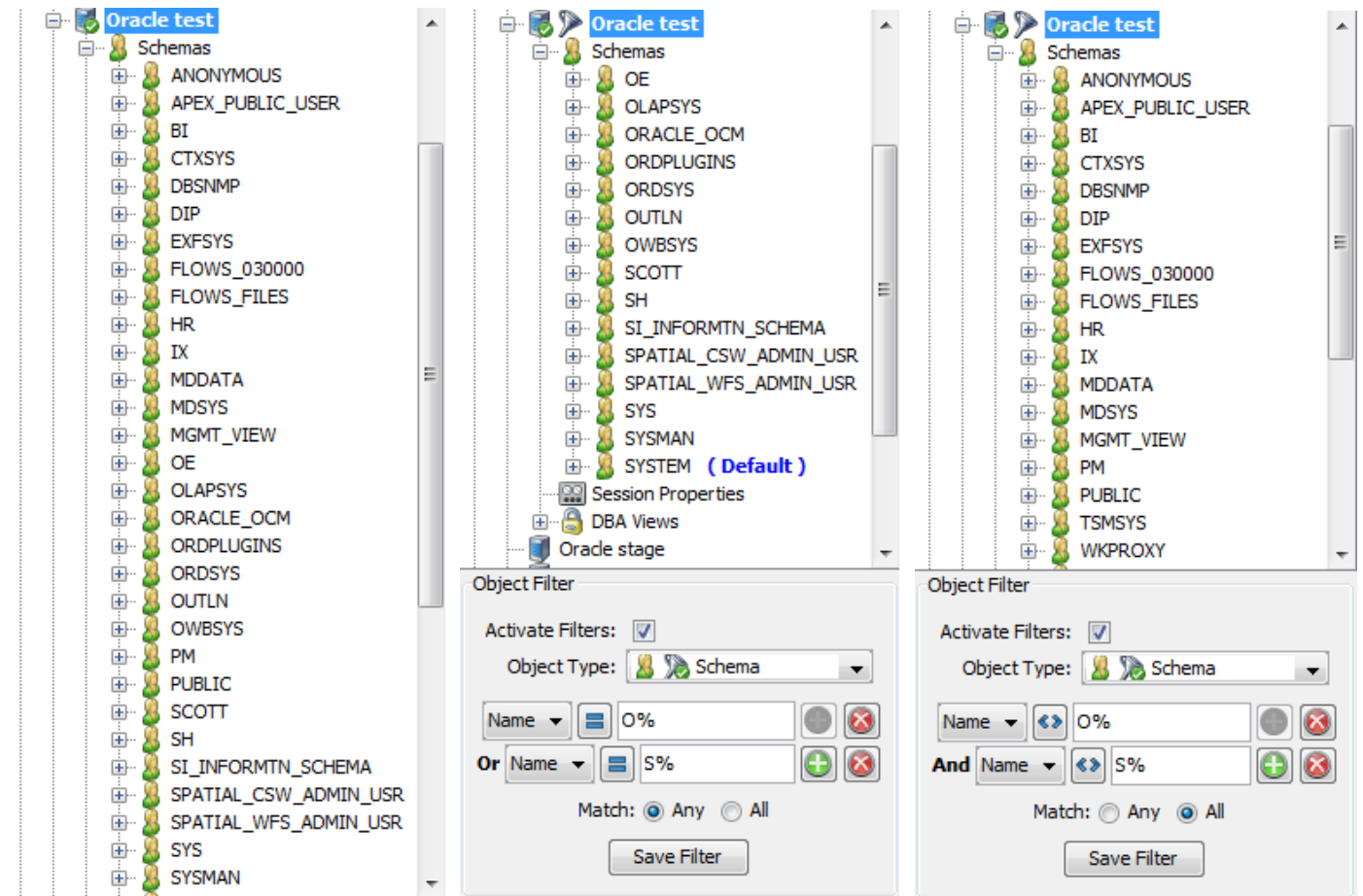


Figure: Examples of tree filter settings

An active filter for a database connection is represented by the funnel icon just before the database connection name. The active state for a filter is defined using the **Activate Filters** checkbox in the Object Filter pane. A filter can only be activated if there are any filters defined. Up to 15 filters can be defined per object type.

A common requirement is to list only the default schema or catalog (database) in the database objects tree. This can be accomplished using the filtering functionality, but the recommended way is to do this with the **Show only default Database or Schema** property in the Properties tab for the Database Connection object. You can read more about this in the [Tool Properties](#) section. A filter may be defined using regular expression syntax.

Show Table Row Count

The **Database->Show/Hide Table Row Count** menu choice decides whether the number of rows for table objects will be listed after the name of the table in the tree.

Enabling this property results in a performance degradation.

Object Tree Icons

Every known object type is associated with an icon that is displayed in the objects tree, object view tab and the actions window. A few operations may add an overlay icon in the objects tree to indicate certain conditions. These are the overlay icons that may appear on top of the object icon:

- New symbol. This indicates that the actual object is newly created since the objects tree was last loaded
- ⚠ Warning symbol indicating that the condition(s) for the actual object is in a state that may require attention
- ✖ Error symbol indicating that the object is in an erroneous or incomplete state

Database Profiles

A [Database Profile](#) is the foundation for database specific support in DbVisualizer. A database profile is, somewhat simplified, a definition of the kind of information that is presented in the database objects tree and in the various object views for a specific database engine. In addition, the profile defines the actions for the object types defined in the profile. DbVisualizer loads the matching database profile when you connect to a database. If no matching profile is found, or if you are running DbVisualizer Free, DbVisualizer uses a [Generic](#) profile with just the general database information and actions included.

Database Specific Support

DbVisualizer Personal currently offer database specific support (database profiles) for the following databases (click links for details):

- [DB2 LUW](#)
- [DB2 z/OS](#)
- [HP Neoview](#)
- [Informix](#)
- [JavaDB/Derby](#)
- [Mimer](#)
- [MySQL](#)
- [Oracle](#)
- [PostgreSQL](#)
- [SQL Server](#)
- [Sybase ASE](#)

The specialized database profiles define different object types, so the database objects tree may look different depending on which database you are connected to. The structure and organization of a database profile is also something that may impact the layout of the tree, even though the provided ones are similar in their structure. There are two root nodes in the majority of the profiles:

- User objects
- DBA objects

User objects are, for example, tables, views, triggers, and functions, while DBA objects most often are objects that require administration privileges in the database in order to access them. DbVisualizer puts all DBA objects under the **DBA Views** tree node. If you connect to a database using an account with insufficient privileges to access a DBA object, you may see error messages if you try to select nodes under the DBA Views node. The following is an example of the DBA sub tree for Oracle.

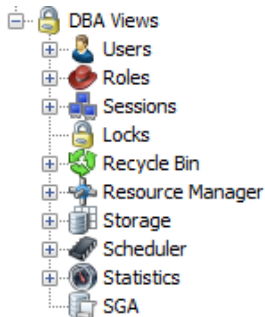


Figure: The DBA Views tree object

Database profiles are defined in XML and it is quite easy to extend and modify them. Read more in the [Database Profile Framework](#) document.

Generic profile

DbVisualizer supports a wide range of databases. The nature of the databases and what they support differ from vendor to vendor, so the appearance and structure of the tree below the Database Connection objects for different databases differ as well. The generic database profile (the only profile available in DbVisualizer Free) displays objects based on what JDBC offers in terms of database information (aka metadata information). DbVisualizer asks the JDBC driver for all schemas, databases, tables and procedures, and then builds the tree based on what the driver returns.

The advantage of using JDBC to get database metadata is that it is a standard way to access the information, independent of the database engine type; the JDBC driver layer hides the proprietary details about where and how the information is really stored. The drawback with using JDBC is that JDBC doesn't offer access to all metadata a database may hold. While the information presented by the generic profile, with its reliance on JDBC, is sufficient for many tasks, a database specific profile offers far more details as well as more features. If you use DbVisualizer Free with one of the databases supported by database specific profiles, you may want to upgrade to the DbVisualizer Personal edition.

The generic database profile when used for an Oracle connection look as follows:

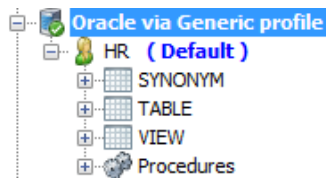


Figure: The generic database profile when applied to an Oracle database connection

The appearance of the generic database profile may include **schema** objects and/or **catalog** objects depending on whether the database supports these objects. The **Procedures** object always appear in the tree, regardless of if the database connection supports procedures or not.

The following sections describe the objects provided by the generic profile.

Catalog/Database object

Catalog is the term used in JDBC and some database engines for a logical grouping of database objects. Other database engines, e.g., Sybase, PostgreSQL, SQL Server and MySQL use the term **Database** for, more or less, the same purpose. Both terms are used interchangeably in DbVisualizer.

The Object View for a Catalog object in the generic profile is a pane with two tabs, **Tables** and **References**. The Tables tab lists all the tables that are located in the catalog while References shows the exact same list of tables but instead as a referential integrity graph.

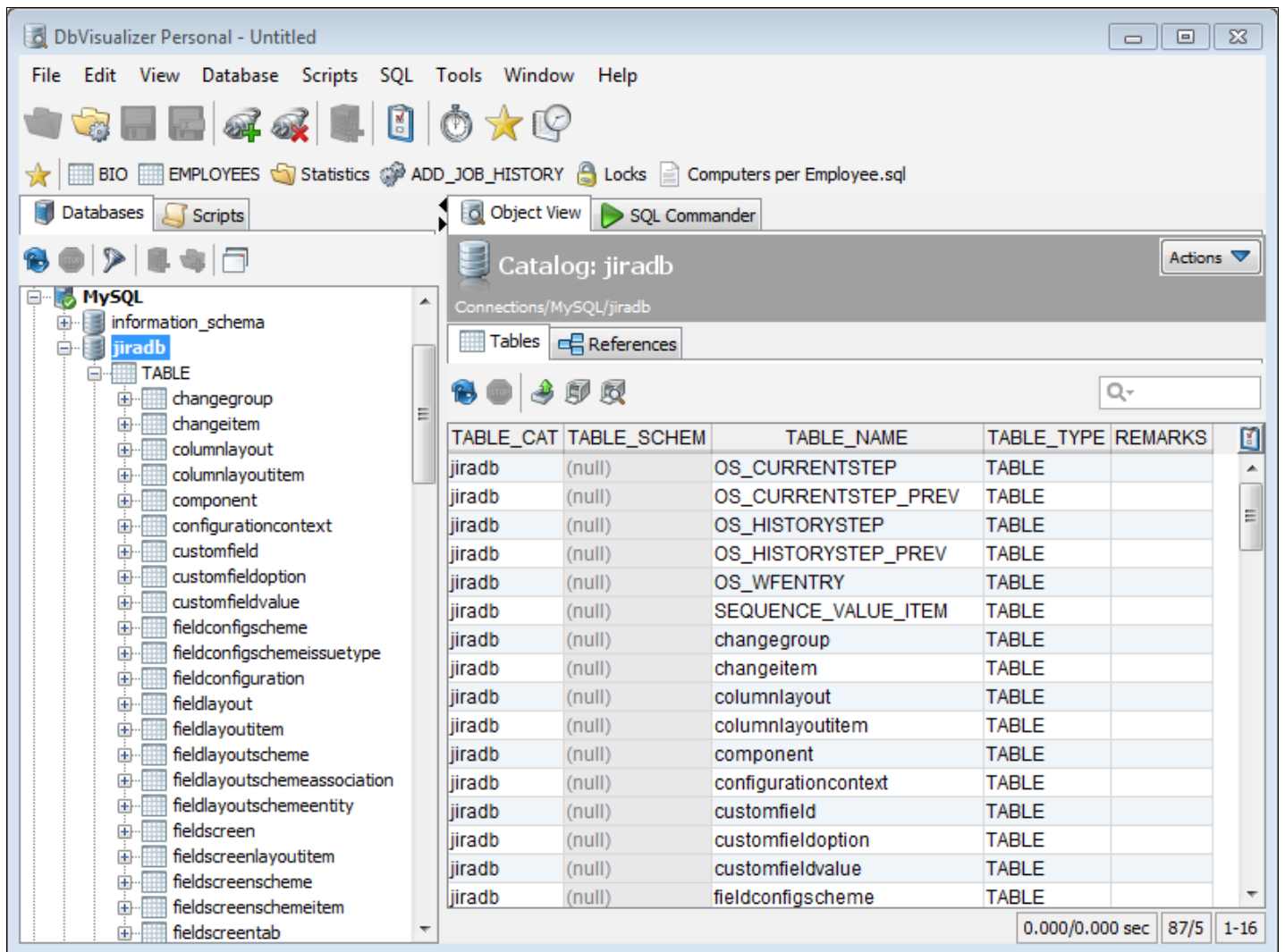


Figure: The view for Catalog objects

The child objects shown for a catalog object depend on the capabilities of the JDBC driver. Typically, a child object represents a type of table that the driver use to categorize the tables in a catalog, e.g., regular tables or system tables. For instance, the example in the figure above shows a MySQL database connection with catalog objects as its child object. The catalog child objects are **TABLE** and **LOCAL TEMPORARY**, because these are the table types that the MySQL JDBC driver supports (these table types are the same as those listed in the **Table Types** tab when selecting a database connection object). For other databases, you may see child objects representing other table types.

Select one or several rows (cells) in the tables grid and then choose **Script: SELECT ALL** to create a select script for the selected tables, copied to the current SQL Commander where it can be executed.

Schema object

The generic profile **Schema** object tree and view are organized in the same way as for the Catalog objects. There is in fact no difference except that the schema objects are in another level in the tree and is represented by a different icon.

The following screenshot shows the information for the selected schema with the **References** tab selected.

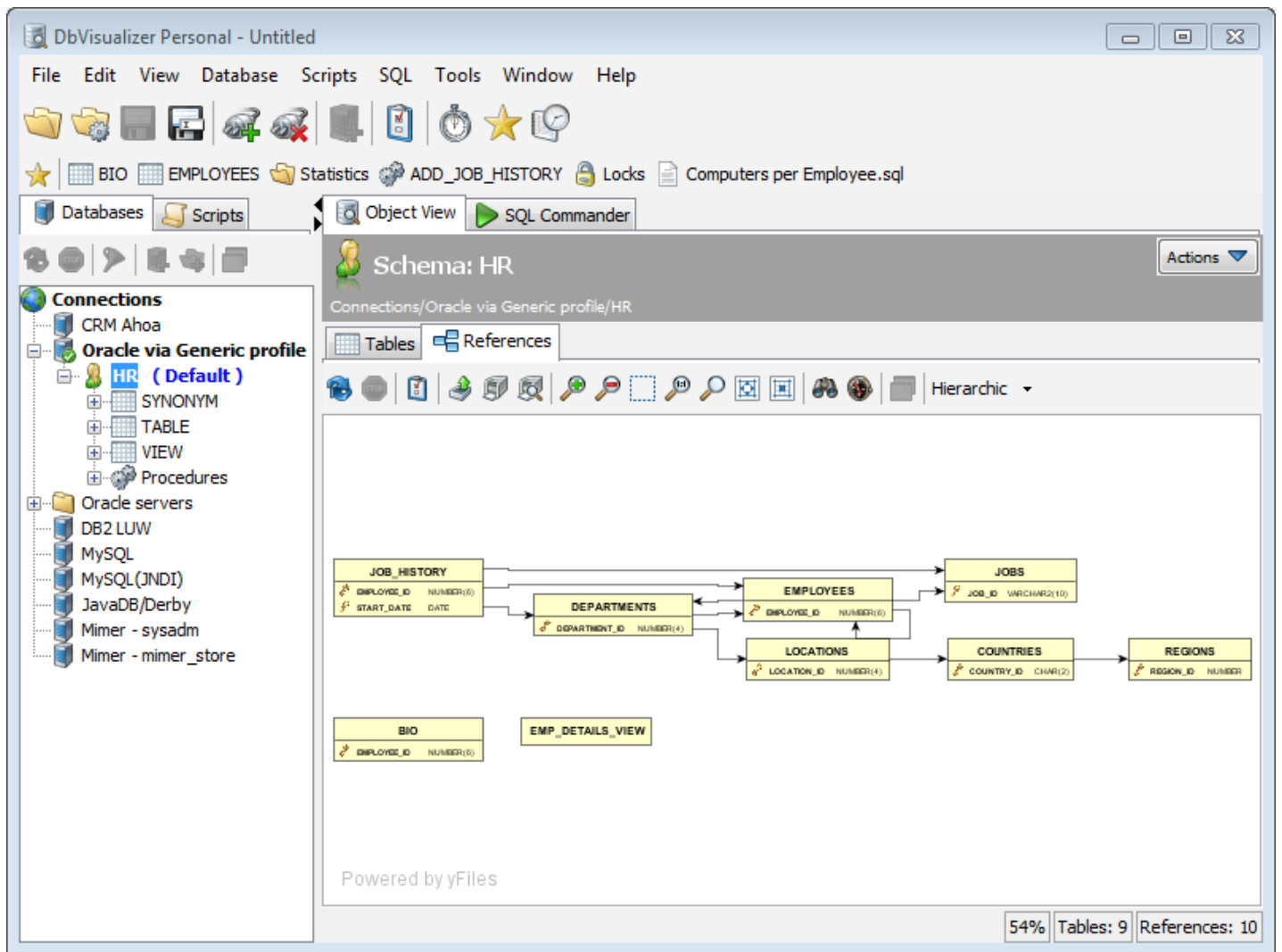
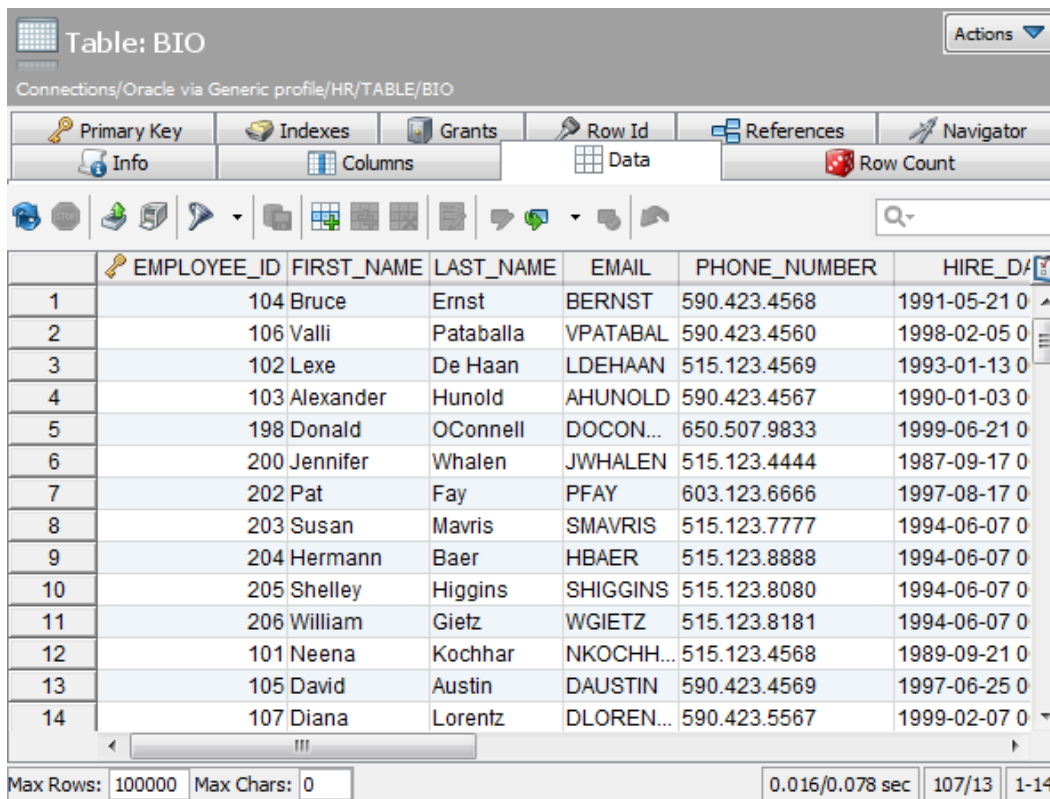


Figure: The view for Schema objects

Table object

The **Table** object is probably the most frequently accessed object in the tree, since its Object View shows not only a lot of information about the table but also the data the table holds. This is also the place where you can edit the table data (only available with the DbVisualizer Personal edition).



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 0
2	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 0
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 0
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 0
5	198	Donald	OConnell	DOCON...	650.507.9833	1999-06-21 0
6	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17 0
7	202	Pat	Fay	PFAY	603.123.6666	1997-08-17 0
8	203	Susan	Mavris	SMAVRIS	515.123.7777	1994-06-07 0
9	204	Hermann	Baer	HBAER	515.123.8888	1994-06-07 0
10	205	Shelley	Higgins	SHIGGINS	515.123.8080	1994-06-07 0
11	206	William	Gietz	WGIEZ	515.123.8181	1994-06-07 0
12	101	Neena	Kochhar	NKOCHH...	515.123.4568	1989-09-21 0
13	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 0
14	107	Diana	Lorentz	DLOREN...	590.423.5567	1999-02-07 0

Max Rows: 100000 Max Chars: 0 0.016/0.078 sec 107/13 1-14

Figure: The view for Table objects

The Object View for a table object contains the following tabs:

Tab	Description
Info	Brief information about the table object
Columns	This tab lists type information about all columns in the table
Data	Read more in Data tab
Row Count	Lists the table row count
Primary Key	Shows the primary key
Indexes	Lists all indexes for the table
Grants	Displays any privileges for the table
Row Id	Displays the optimal set of columns that uniquely identifies a row
References	Read more in References tab
Navigator	Read more in Navigator tab

Procedure object

The Procedure object shows the name of the procedure or function in the tree, and the Object View lists the parameters that are used when calling it.

Procedure: ADD_JOB_HISTORY Actions ▾

Connections/Oracle via Generic profile/HR/Procedures/ADD_JOB_HISTORY

Procedure Columns

Q

PROCEDURE_SCHEM	PROCEDURE_NAME	COLUMN_NAME	COLUMN_TYPE	DATA_TYPE	TYPE
HR	ADD_JOB_HISTORY	P_EMP_ID	1	3 NUM	▲
HR	ADD_JOB_HISTORY	P_START_DATE	1	93 DATI	
HR	ADD_JOB_HISTORY	P_END_DATE	1	93 DATI	
HR	ADD_JOB_HISTORY	P_JOB_ID	1	12 VAR	
HR	ADD_JOB_HISTORY	P_DEPARTMENT_ID	1	3 NUM	

0.203/0.000 sec 5/16 1-5

Figure: The procedure object

The Object View shows a list of column names for the selected procedure.

Object Views

The Object View tab shows detailed information about the selected tree object. The Object View may contain several sub tabs, depending on the current database profile and the type of the object selected in the tree. There may also be several representations of the same information, providing different views of the information. The following sections describe the different views, or visual presentation forms, provided by DbVisualizer.

Grid

The Grid view is the most common one as it displays the data in a standard grid style.

Table: BIO Actions ▾

Connections/Oracle via Generic profile/HR/TABLE/BIO

Primary Key Indexes Grants Row Id References Navigator

Info Columns Data Row Count

Q

TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE	BUFFER
HR	BIO	EMPLOYEE_ID	3 NUMBER		6	
HR	BIO	FIRST_NAME	12 VARCHAR2		20	
HR	BIO	LAST_NAME	12 VARCHAR2		25	
HR	BIO	EMAIL	12 VARCHAR2		25	
HR	BIO	PHONE_NUMBER	12 VARCHAR2		20	
HR	BIO	HIRE_DATE	93 DATE		7	
HR	BIO	JOB_ID	12 VARCHAR2		10	
HR	BIO	SALARY	3 NUMBER		8	
HR	BIO	COMMISSION_PCT	3 NUMBER		2	
HR	BIO	MANAGER_ID	3 NUMBER		6	
HR	BIO	DEPARTMENT_ID	3 NUMBER		4	
HR	BIO	PHOTO	2004 BLOB		4000	
HR	BIO	RESUME	2005 CLOB		4000	

0.078/0.016 sec 13/18 1-13

Figure: The Grid view

Form

The Form view extends the Grid view by adding a form below the grid. Click on a row in the grid and the information is displayed in the form.

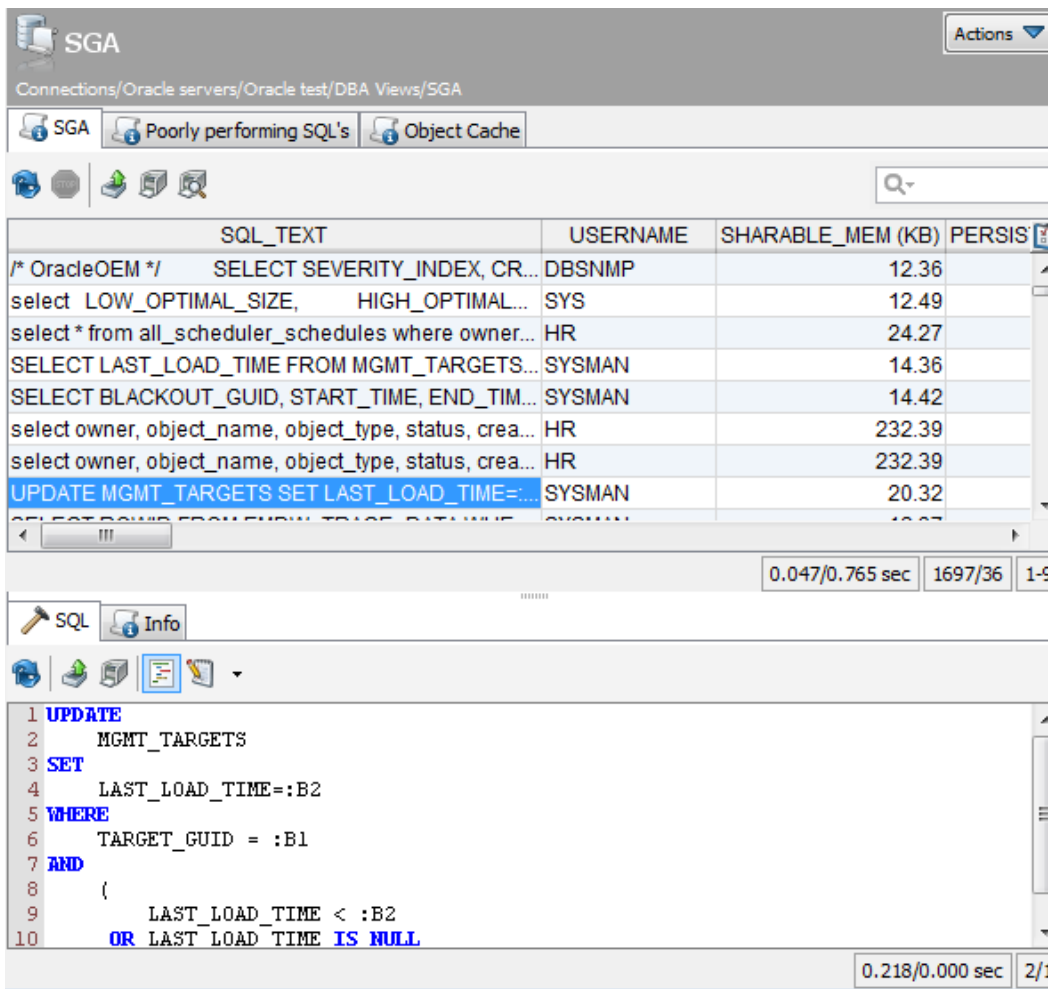


Figure: The Form view

If there is only one row of data, only the form is displayed.

Source

The Source view is typically used to show the source for functions, procedures, triggers, etc. It is based on a read only editor with SQL syntax coloring. The sub toolbar buttons from the left:

- Reload the data from the database
- Stop loading the data from the database
- Export the data to file
- Print the data
- Copy the data to SQL Commander
- Format the SQL

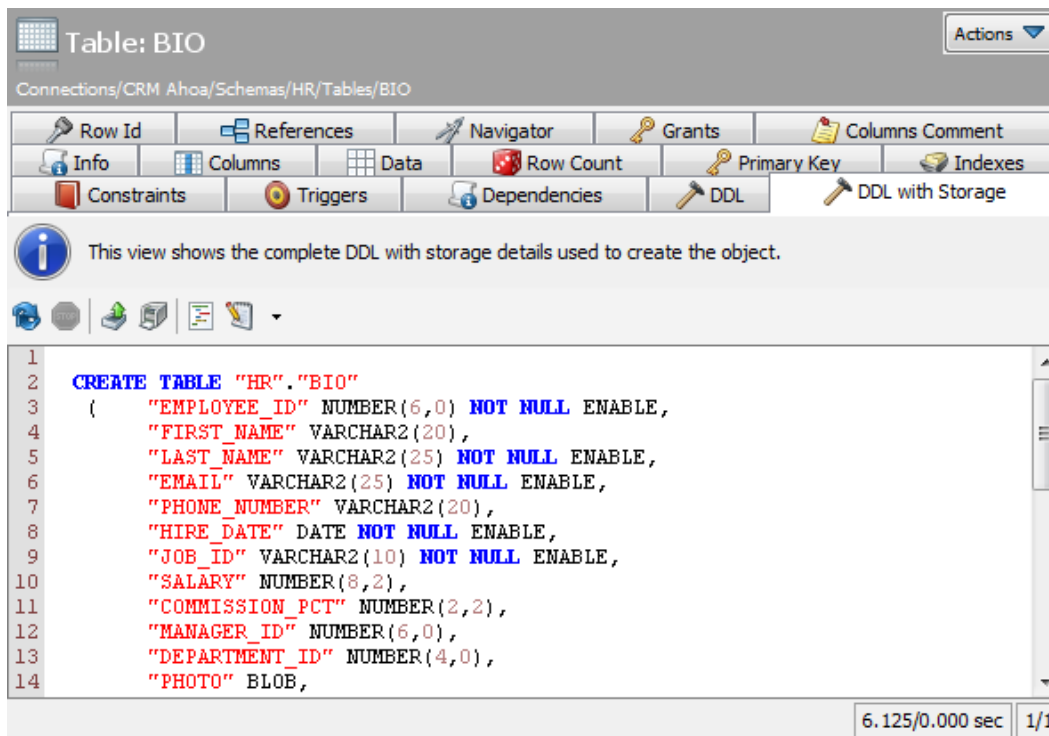


Figure: The Source view

Table Row Count

The row count view is really simple: it only shows the number of rows in the selected object.

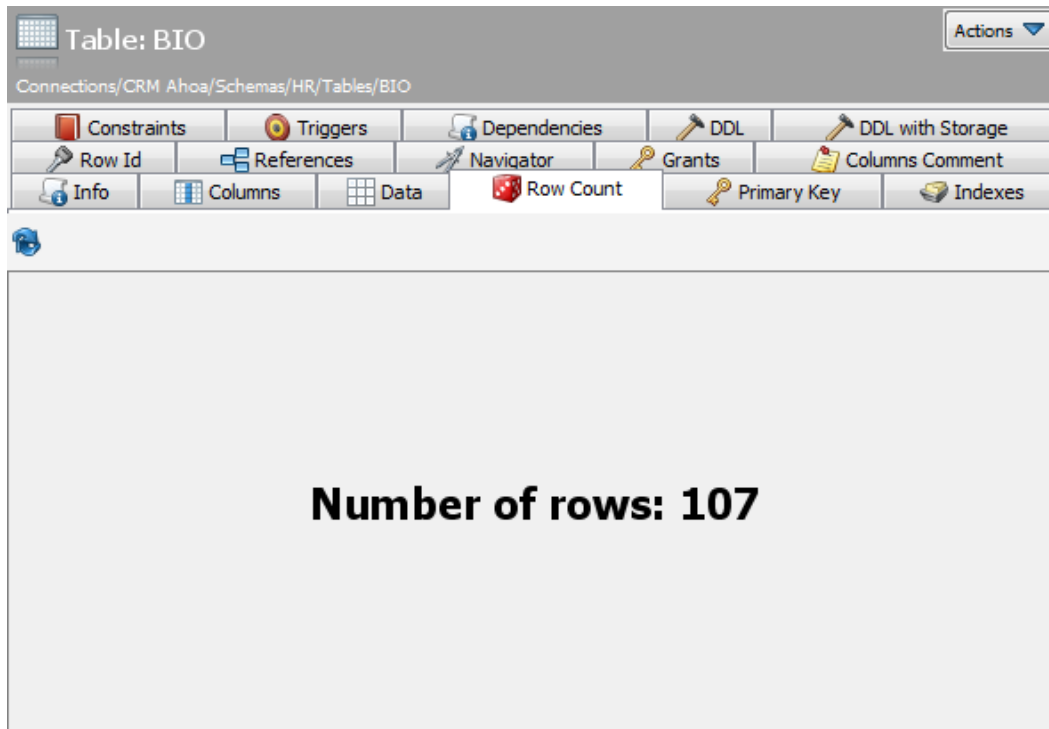


Figure: The Row Count view

Table Data

You use the **Data** tab to browse the data in the table and to do various data related operations. This view is based on the `generic_grid`, but it adds a few more visual components to limit the max number of rows, the width of text columns and the collection of data tab specific operations in the right-click menu. In addition, you can also use a filter (a SQL WHERE clause) to limit the data to the rows that match the filter. The data tab is the place to do `edits` in DbVisualizer Personal.

	EMPLOYEE_ID	FIRST_N.	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	121	Adam	Fripp	AFRIPP	650.123.2234	1997-04-10
2	196	Alana	Walsh	AWALSH	650.507.9811	1998-04-24
3	147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429...	1997-03-10
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03
5	115	Alexander	Khoo	AKHOO	515.127.4562	1995-05-18
6	185	Alexis	Bull	ABULL	650.509.2876	1997-02-20
7	158	Allan	McEwen	AMCEWEN	011.44.1345.829...	1996-08-01
8	175	Alyssa	Hutton	AHUTTON	011.44.1644.429...	1997-03-19
9	167	Amit	Banda	ABANDA	011.44.1346.729...	2000-04-21
10	187	Anthony	Cabrio	ACABRIO	650.509.4876	1999-02-07
11	193	Britney	Everett	BEVERETT	650.501.2876	1997-03-03
12	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21
13	179	Charles	Johnson	CJOHNSON	011.44.1644.429...	2000-01-04
14	153	Christopher	Olsen	COLSEN	011.44.1344.498...	1998-03-30
15	162	Clara	Vishnev	CVISHNFY	011.44.1346.129	1997-11-11

Figure: The Data tab for Table objects

Right-click menu

The Data tab grid right-click menu contains some operations in addition to those in the standard grid right-click menu. The additional operations are primarily for creating SQL statements based on the current selection. Choosing any of these creates the appropriate SQL and then switch the view to the SQL Commander tab. You must use these operations to edit table data in the DbVisualizer Free edition. With the DbVisualizer Personal edition, you can instead use inline and form based editing. (Information about the standard right click menu operations are available in the [Getting Started and General Overview](#) document).

You can generate SQL with either static values as they appear in the grid, or with DbVisualizer **variables**. A variable is essentially a placeholder for a value in an SQL statement. When the statement is executed, DbVisualizer locates all variables and presents them in a dialog where you can enter or modify values for the variables. DbVisualizer replaces the variable placeholders with the new values before executing the statement. Variables can be used in any SQL statement and DbVisualizer relies on them heavily. (Read more about variables in the [SQL Commander](#) document).

Whether to use variables in the SQL statements generated by the right-click menu SQL operations depends on the **Table Data->Include Variables in SQL** setting in **Tool Properties**, under the General tab. By default, variables are being used in the statement. If you disable the property, literal values are instead used in the generated statement.

Here is an example with the **Include Variables in SQL** setting enabled and then disabled. The SQL is generated when the **Script: SELECT ALL WHERE** operation is selected based on the selection in the previous figure.

Include Variables in SQL is enabled:

```
select *
from HR.COUNTRIES
where COUNTRY_NAME = ${COUNTRY_NAME (where)||Brazil||String||where nullable ds=40 dt=VARCHAR }$
```

Include Variables in SQL is disabled:

```
select *
from HR.COUNTRIES
where COUNTRY_NAME = 'Brazil'
```

The following lists the generated SQL for each of the operations based on the selection of **COUNTRY_NAME = Brazil**, with variables disabled.

Operation	SQL Example
Script: SELECT ALL	<pre>select * from HR.COUNTRIES</pre>
Script: SELECT ALL WHERE	<pre>select * from HR.COUNTRIES where COUNTRY_NAME = 'Brazil'</pre>
Script: SELECT ALL WITH FILTER	<pre>select * from HR.COUNTRIES where REGION_ID = 1 // If this is the filter, see below</pre>
Script: INSERT INTO TABLE	<pre>insert into HR.COUNTRIES (COUNTRY_ID, COUNTRY_NAME, REGION_ID) values ('', '',)</pre>
Script: INSERT COPY INTO TABLE	<pre>insert into HR.COUNTRIES (COUNTRY_ID, COUNTRY_NAME, REGION_ID) values ('BR', 'Brazil', 2)</pre>
Script: UPDATE WHERE	<pre>update HR.COUNTRIES set COUNTRY_ID = 'BR', COUNTRY_NAME = 'Brazil', REGION_ID = 2 where COUNTRY_NAME = 'Brazil'</pre>
Script: DELETE WHERE	<pre>delete from HR.COUNTRIES where COUNTRY_NAME = 'Brazil'</pre>

Where Filter

The filter capability in the Data tab lets you limit the number of rows in the grid, using the same syntax as for an SQL WHERE clause. The Filter menu button in the grid toolbar contains all operations related to using a filter.

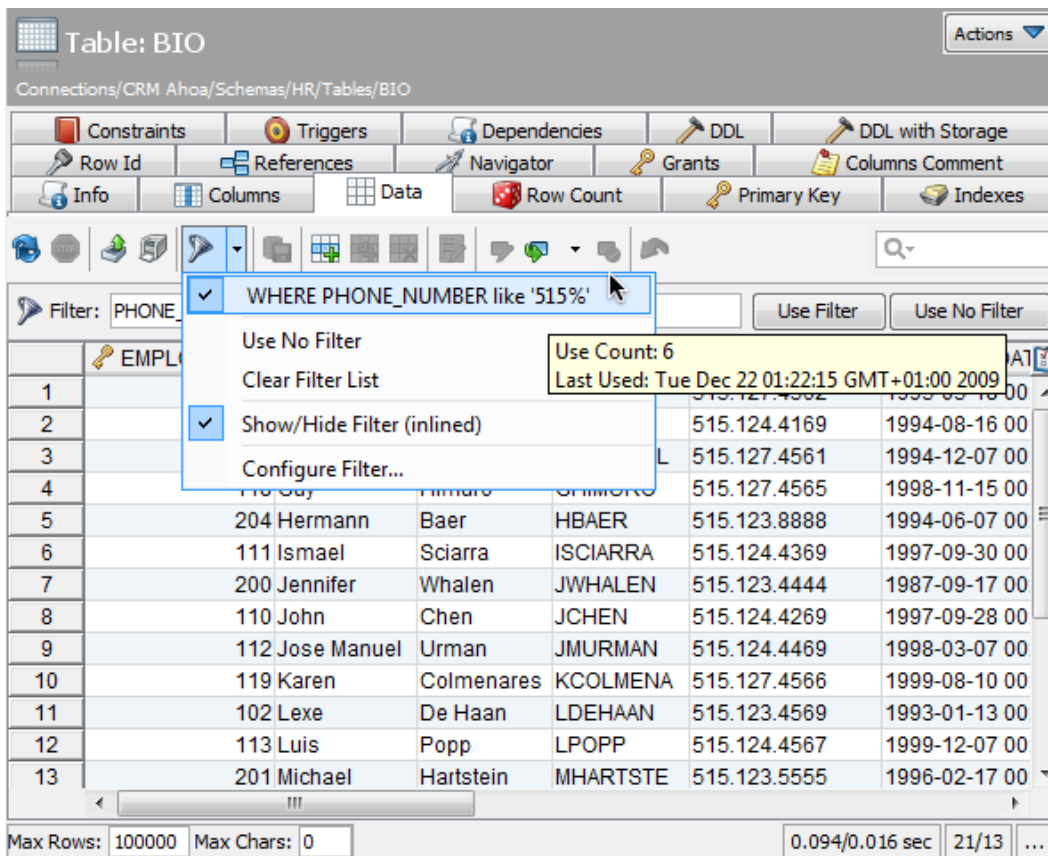


Figure: Filter menu

The top entries in the menu are previously used filters for the table, if any. The checkbox is selected for the filter that is currently in use. The filters are saved between DbVisualizer sessions, and you can toggle between them by selecting them from the menu. You use the **Use No Filter** choice to disable all filters for the table, and the **Clear Filter List** to permanently remove all filters for the table.

To create a new filter, select **Configure Filter** to launch the Filter Configuration dialog.

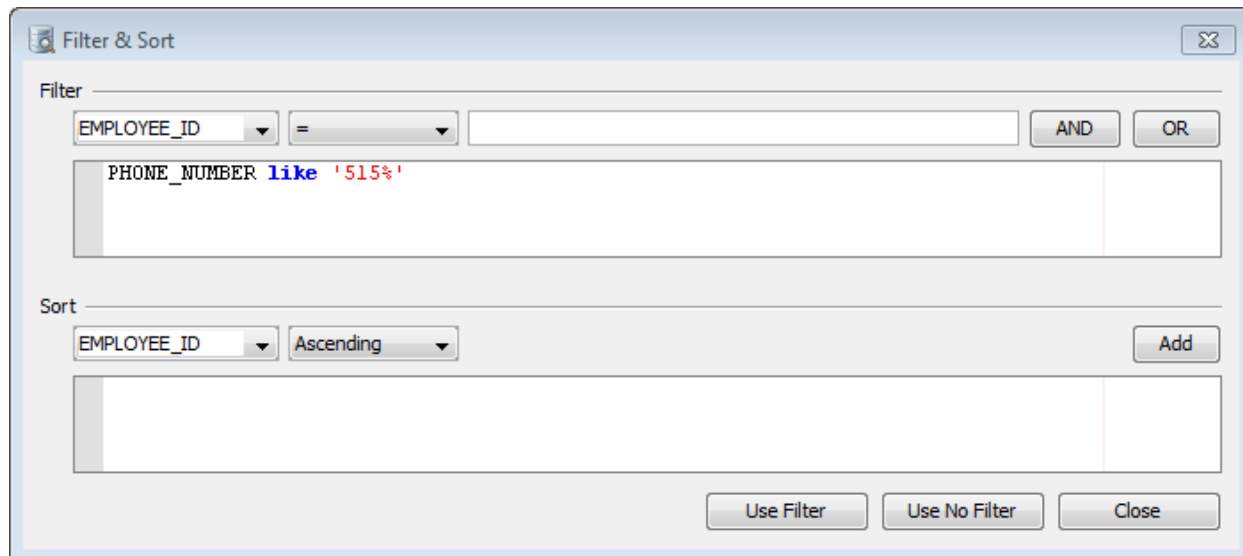


Figure: The Data tab Filter Configuration dialog

The Filter Configuration dialog contains one Filter area and a Sort area.

The Filter area is composed of two parts. The upper one is used to define a condition for a single column. You can use the two lists to select the column name and an operator, and enter the value of the column in the text field. You can use **Ctrl-Enter** while editing the value to force a reload of the grid based on that single filter. The lower part displays the complete filter and the buttons are used to control whether the newly entered filter will

be **AND**'ed or **OR**'ed with the complete filter. The buttons change appearance based on whether there is any filter or not. While in the complete filter you can use **Ctrl-Enter** to force a reload based on the complete filter.

The Sort area is similar to the Filter area. You can select column names and sort order from the two lists, and click the **Add** button to add the sort criteria for the single column to the complete criteria.

Click the **Use Filter** button to apply the filter and save it, and close the dialog by clicking the **Close** button.

If you often need to tweak the filter conditions and want a more compact user interface, you can use the inline filter view. Use the **Show/Hide Inline Filter** choice in the Filter menu to toggle the visibility of the inline filter.

The screenshot shows a database application window titled "Table: BIO". The interface includes a toolbar with various database-related icons and buttons. Below the toolbar, there is a filter input field containing the text "PHONE_NUMBER like '515%'", with "Use Filter" and "Use No Filter" buttons to its right. The main area displays a data grid with the following columns: EMPLOYEE_ID, FIRST_N., LAST_NAME, EMAIL, PHONE_NUMBER, and HIRE_DATE. The grid contains 13 rows of data, with the first row being highlighted. At the bottom of the window, there are status indicators for "Max Rows: 100000", "Max Chars: 0", "0.141/0.031 sec", "21/13", and "1-13".

	EMPLOYEE_ID	FIRST_N.	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	115	Alexander	Khoo	AKHOO	515.127.4562	1995-05-18 00
2	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08-16 00
3	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00
4	118	Guy	Himuro	GHIMURO	515.127.4565	1998-11-15 00
5	204	Hermann	Baer	HBAER	515.123.8888	1994-06-07 00
6	111	Ismael	Sciarra	ISCIARRA	515.124.4369	1997-09-30 00
7	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17 00
8	110	John	Chen	JCHEN	515.124.4269	1997-09-28 00
9	112	Jose Manuel	Urman	JMURMAN	515.124.4469	1998-03-07 00
10	119	Karen	Colmenares	KCOLMENA	515.127.4566	1999-08-10 00
11	102	Lexe	De Haan	LDEHAAN	515.123.4569	1993-01-13 00
12	113	Luis	Popp	LPOPP	515.124.4567	1999-12-07 00
13	201	Michael	Hartstein	MHARTSTE	515.123.5555	1996-02-17 00

Figure: Data tab with the Inline filter enabled

The inline filter is displayed above the grid. You can edit the condition in the text field and use **Ctrl-Enter** or click the **Use Filter** button to apply the modified condition. Instead of manually typing column names in the field use the **Ctrl-Space** key binding to show a list of available columns.

Quick Filter

The quick filter acts on the data that is already in the grid, as opposed of a **WHERE filter** which is used to limit the number of rows fetched from the database. With a Quick filter, you can easily list only those rows in the grid that match the entered search string.

The following figure shows data that matches the search string "d". Matching cells are highlighted.

Table: BIO

Connections/CRM Ahoa/Schemas/HR/Tables/BIO

Constraints Triggers Dependencies DDL DDL with Storage
 Row Id References Navigator Grants Columns Comment
 Info Columns Data Row Count Primary Key Indexes

EMPLOYEE_ID FIRST_N. LAST_NAME EMAIL PHONE_NUMBER HIRE_DATE

	EMPLOYEE_ID	FIRST_N.	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
2	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00
3	198	Donald	OConnell	DOCONNEL	650.507.9833	1999-06-21 00:00:00
4	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17 00:00:00
5	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21 00:00:00
6	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00
7	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02-07 00:00:00
8	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08-16 00:00:00
9	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00:00:00
10	115	Alexander	Khoo	AKHOO	515.127.4562	1995-05-18 00:00:00
11	116	Shelli	Baida	SBAIDA	515.127.4563	1997-12-24 00:00:00
12	121	Adam	Fripp	AFRIPP	650.123.2234	1997-04-10 00:00:00
13	127	James	Landry	JLANDRY	650.124.1334	1999-01-14 00:00:00
14	137	Renske	Ladwig	RLADWIG	650.121.1234	1995-07-14 00:00:00
15	142	Curtis	Davies	CDAVIES	650 121 2994	1997-01-29 00:00:00

Max Rows: 100000 Max Chars: 0 0.219/0.078 sec 32 [107]/13 1-15

Figure: Using the Quick Filter

Entering successive characters will narrow the result even further, as in the following figure.

Table: BIO

Connections/CRM Ahoa/Schemas/HR/Tables/BIO

Constraints Triggers Dependencies DDL DDL with Storage
 Row Id References Navigator Grants Columns Comment
 Info Columns Data Row Count Primary Key Indexes

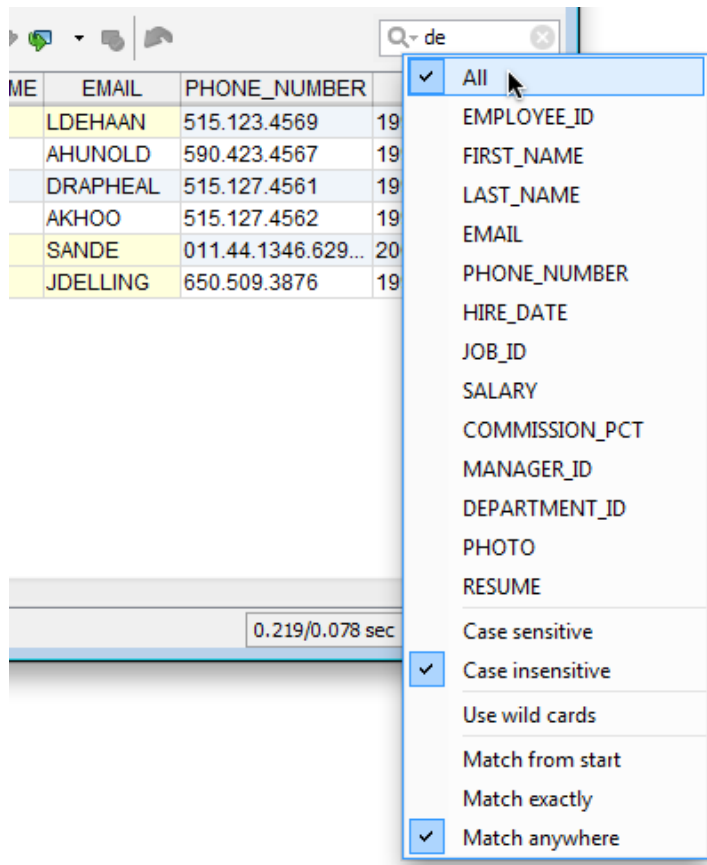
EMPLOYEE_ID FIRST_N. LAST_NAME EMAIL PHONE_NUMBER HIRE_DATE

	EMPLOYEE_ID	FIRST_N.	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
2	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00
3	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00:00:00
4	115	Alexander	Khoo	AKHOO	515.127.4562	1995-05-18 00:00:00
5	166	Sundar	Ande	SANDE	011.44.1346.629...	2000-03-24 00:00:00
6	186	Julia	Dellinger	JDELLING	650.509.3876	1998-06-24 00:00:00

Max Rows: 100000 Max Chars: 0 0.219/0.078 sec 6 [107]/13 1-6

Figure: Refining the filtering

The Quick Filter pull-down menu (click on the down arrow next to the magnifying glass) lets you choose if the filter should match cells in all columns or just one selected column, case or case insensitive matching, and where in the cell the value must match.



Monitor row count

Read more about the **Monitor Row Count** and **Monitor Row Count Difference** in [Monitor and Charts](#).

Editing

Read about data editing in [Edit Table Data](#)

DDL Viewer

The DDL Viewer tabs appear only for objects in databases that have [specialized database profiles](#).

```

1 CREATE
2   TABLE BIO
3   (
4     EMPLOYEE_ID NUMBER(6) NOT NULL,
5     FIRST_NAME VARCHAR2(20),
6     LAST_NAME VARCHAR2(25) NOT NULL,
7     EMAIL VARCHAR2(25) NOT NULL,
8     PHONE_NUMBER VARCHAR2(20),
9     HIRE_DATE DATE NOT NULL,
10    JOB_ID VARCHAR2(10) NOT NULL,
11    SALARY NUMBER(8,2),
12    COMMISSION_PCT NUMBER(2,2),
13    MANAGER_ID NUMBER(6),
14    DEPARTMENT_ID NUMBER(4),
15    PHOTO BLOB,
16    RESUME CLOB,
17    PRIMARY KEY (EMPLOYEE_ID)
18  )

```

11.179/0.000 sec 1/1

Figure: The DDL viewer for a table

References

The **References** tab for a Table object shows how the table references other tables (Imported Keys) and how other tables reference the selected table (Exported Keys), based on primary and foreign key declarations. Use the sub tabs at the bottom of the display to show either view. The following shows the references from the table.

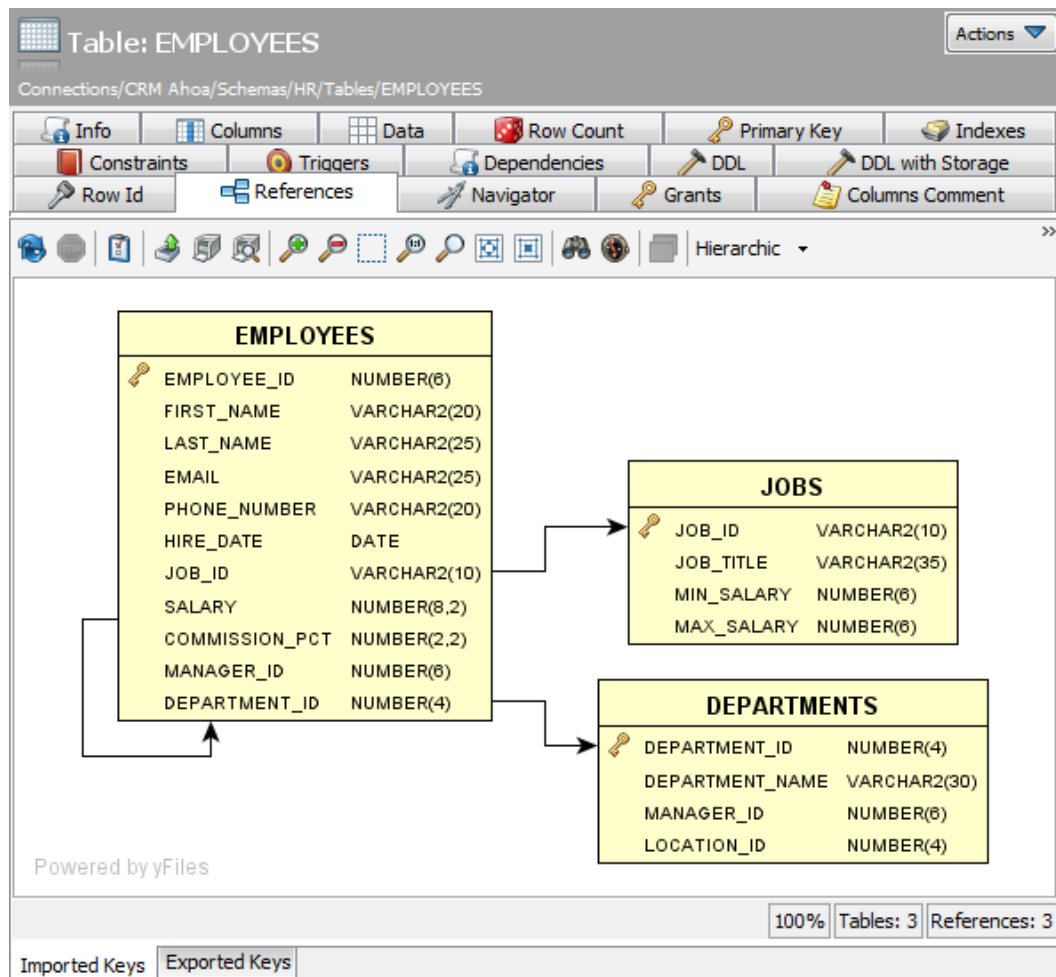


Figure: The references graph showing imported keys for a table

Navigator

The **Navigator** tab (only available in the DbVizualizer Personal edition) provides an interactive way to navigate in data by following primary key and foreign key references.

The screenshot shows the Navigator tab for the **EMPLOYEES** table. The interface includes a menu bar with options like Info, Columns, Data, Row Count, Primary Key, and Indexes. Below the menu is a toolbar with icons for various database operations. The main area displays a navigation graph with three tables: **EMPLOYEES**, **JOBS**, and **JOB_HISTORY**. Arrows indicate relationships: **EMPLOYEES** to **JOBS** (via **JOB_ID** and **EMPLOYEE_ID 198**), **EMPLOYEES** to **JOB_HISTORY** (via **EMPLOYEE_ID**), and **JOBS** to **EMPLOYEES** (via **JOB_ID**). Below the graph is a data grid for the **EMPLOYEES** table, showing columns for **EMPLOYEE_ID**, **FIRST_NAME**, **LAST_NAME**, **EMAIL**, **PHONE_NUMBER**, and **HIRE_DATE**. The grid contains six rows of data. At the bottom, there are status indicators for Max Rows, Max Chars, execution time, and page navigation.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	198	Donald	OConnell	DOCONNEL	650.507.9833	1999-06-21 00:00:00
2	199	Douglas	Grant	DGRANT	650.507.9844	2000-01-13 00:00:00
3	180	Winston	Taylor	WTAYLOR	650.507.9876	1998-01-24 00:00:00
4	181	Jean	Fleur	JFLEAUR	650.507.9877	1998-02-23 00:00:00
5	182	Martha	Sullivan	MSULLIVA	650.507.9878	1999-06-21 00:00:00
6	183	Girard	Geoni	GGEONI	650.507.9879	2000-02-03 00:00:00

Figure: The Navigator tab showing two navigation cases

The tab contains a graphic view showing navigation cases (paths through the data) at the top and a data grid showing the data for the navigation case selected in the graph. You navigate in the data by selecting the row in the grid that holds the key value you want to follow, e.g., a specific department in the example shown in the figure, and then select a primary or foreign key relationship from the Related Table list above the grid. This creates a new navigation case in the graph and updates the grid with the corresponding data.

How to use the navigator is described in more detail in the [Data Navigation](#) section.

Procedure Editor

You can use the procedure editor to browse, edit, compile and execute procedures, functions, packages, package bodies, triggers and other database objects that represent custom code that can be invoked in a database.

You can edit the source code in the editor and then click **Save** to save/compile the code. If errors are found, selecting an error message in the error list highlights the row containing the incorrect statement in the editor (in the cases when a row number is available, which is not true for all databases). To test the code, click **Execute** and a script for calling the procedure with the parameter values you provide is generated and executed in the SQL Commander.

More information can be found in the [Procedure Editor](#) document.

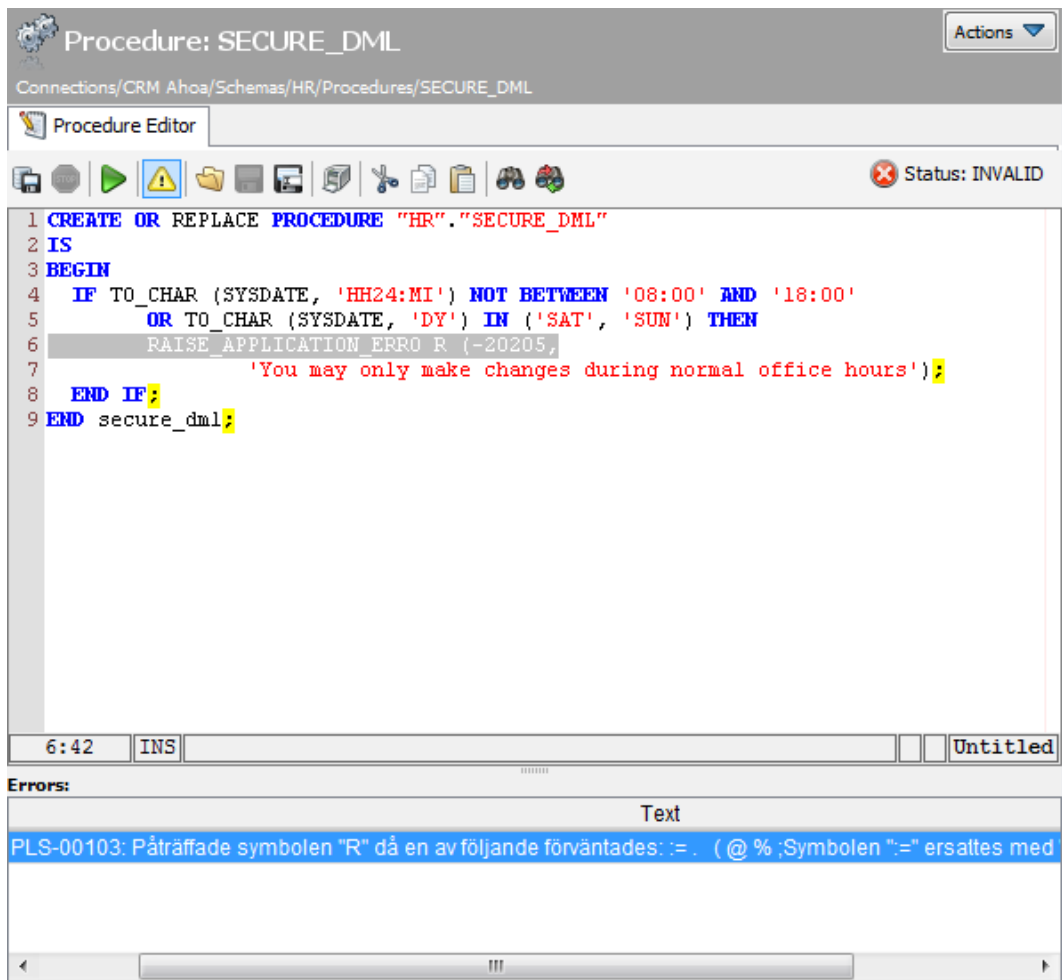


Figure: The procedure editor for functions, procedures, packages etc.

SQL Commander

Introduction

The SQL Commander is used to edit, format and execute SQL statements or SQL scripts. Multiple editors may be open at the same time, each controlling its own SQL log and result sets. Result sets can be displayed in grid, text or chart formats.

The SQL Commander supports the following features:

- Syntax coloring
- Auto completion
- Multiple SQL editors
- Multiple result sets
- SQL editors displayed as tabs or windows
- Result sets displayed as tabs or windows
- Support for stored procedures producing multiple result sets
- SQL formatter with extensive customization options
- Execution control (stop on error/warning)
- View result sets as grid, text or chart
- Editable result sets with the inline or form editors
- Support for BLOB, CLOB and binary data
- View BMP, TIFF, PNG, GIF and JPEG images
- View XML data in tree or text format
- Export result sets as CSV, HTML, Excel, XML, SQL or text
- Batch execution enabling export of unlimited sized result sets
- Execution of script files of unlimited size
- SQL history saved between sessions
- Bookmark management (save favorite SQLs)
- Sort, quick filter and basic calculations of result sets
- Parameterized queries
- Drop objects dragged from the Objects Tree
- Auto Commit on/off support with confirmation checks if uncommitted updates
- Full key binding support with predefined key maps for for Windows, Mac OS X, Linux-UNIX, SQL Query Analyzer and TOAD users

Database specific support:

- **Oracle, DB2 and SQL Server:** Explain Plan queries presented either in tree or graph format
- **Oracle:** support for TIMESTAMPLTZ, TIMESTAMPTZ and XML data types
- **Oracle:** support for DBMS Output

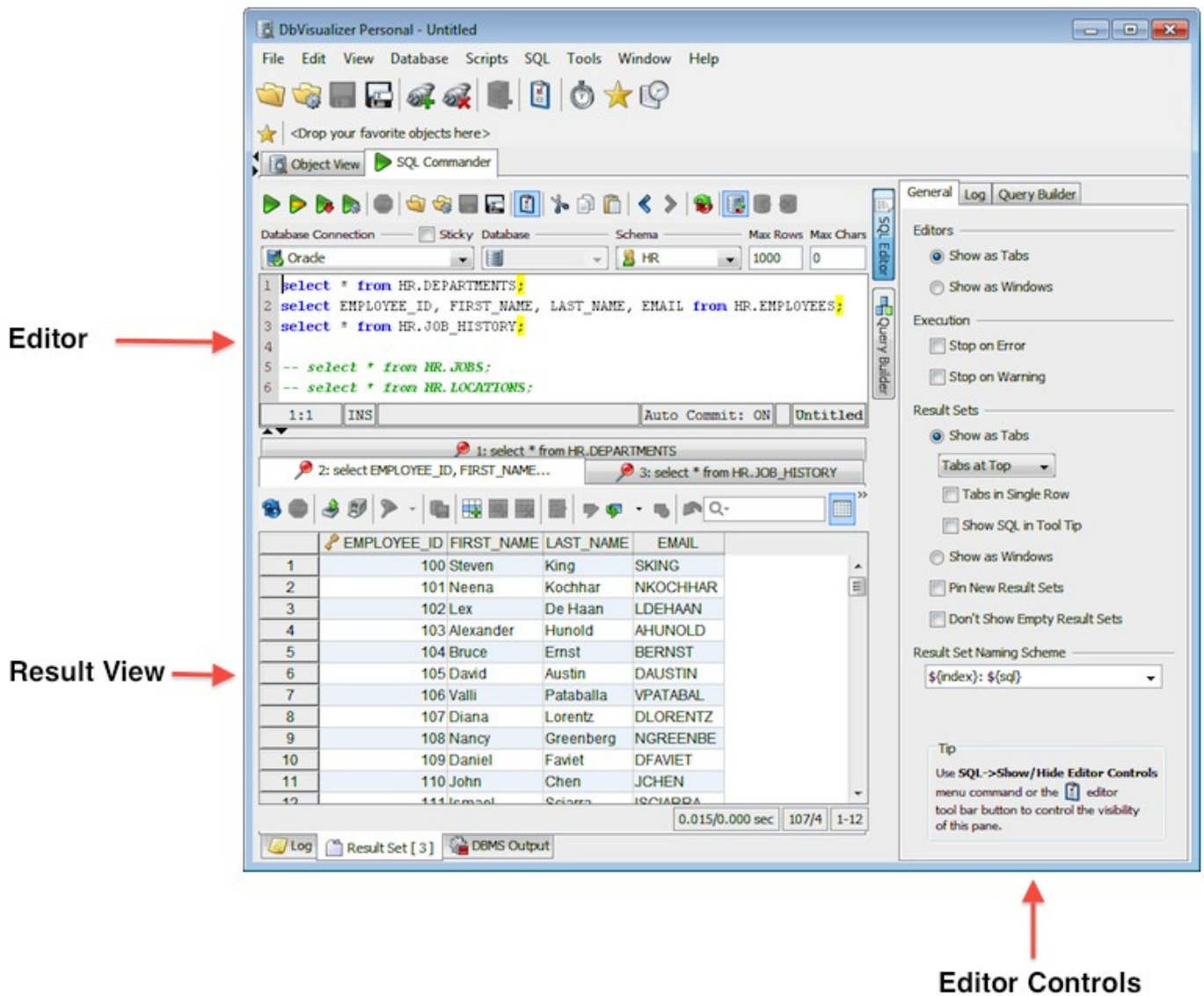


Figure: SQL Commander overview

The figure shows the editing area with its controls above and the output view in the lower part of the screen. The following sections describe all features and controls in the SQL Commander in detail.

Physical Database Connections and Transactions

The SQL Commander supports database transaction control via Auto Commit or manually using commit or rollback. The **Use Single Physical Database Connection** setting in connection properties specifies whether DbVisualizer will use one or multiple physical database connection. This setting is disabled by default and DbVisualizer will then always use at least one physical connection and one for every SQL editor that is created. Running a statement or sequence of statements in one SQL editor will not lock the rest of the user interface while it is executing. If instead a single physical database connection is used, all of the UI is locked until the execution in the SQL Editor has completed. The reason for this behavior is that it could otherwise lead to data corruption when using the same physical database connection for all DbVisualizer operations.

Another important feature is that the editor status bar shows the number of uncommitted requests if auto commit is off. Pay extra attention to this as it indicates that you should complete the current transaction with either commit or rollback.

Editor

The SQL Commander always has at least one editor. It is called the primary editor and cannot be removed. To create additional editors, use the **File->Create SQL Editor** menu choice or the appropriate key binding. To close an editor, use the right-click menu on the editor tab or the close

operations in the File menu.

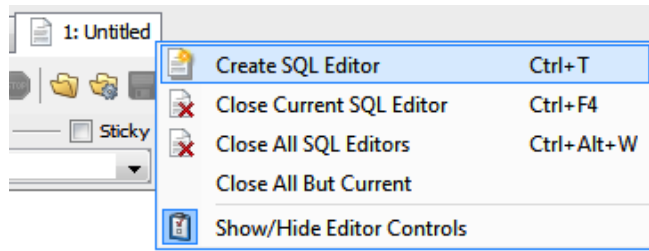


Figure: Editor tab menu

The SQL editor in DbVisualizer is based on the [NetBeans](#) editor module and supports all standard editing features. The right-click menu have the following operations:


























	Execute	Ctrl+Enter
	Execute Current	Ctrl+Period
	Execute Buffer	
	Execute Explain Plan	Ctrl+Alt+Enter
	Undo	Ctrl+Z
	Redo	Ctrl+Shift+Z
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Print...	
	Print Preview...	
	Clear All	Ctrl+Shift+Delete
	Find...	Ctrl+F
	Find Next	F3
	Find Previous	Shift+F3
	Replace...	Ctrl+H
	Goto Line...	Ctrl+G
	Lower Case	Ctrl+Shift+L
	Upper Case	Ctrl+Shift+U
	Comment Line	Ctrl+Shift+R
	Comment Block	Ctrl+Shift+B
	Format SQL	
	Show Auto Completion...	Ctrl+Space
	Select All	Ctrl+A
	Select Current Statement	Ctrl+Shift+Period

Figure: The SQL editor right click menu

The SQL editor is also used when editing CLOBs in the form editor.

Database Connection, Catalog and Schema

You use the Database Connection and Database (or Catalog) lists above the editor to specify which connection and database to use when executing the SQL in the editor. The list of connections shows all connections as they are ordered in the Database Objects tree, except that all currently active connections are listed first.

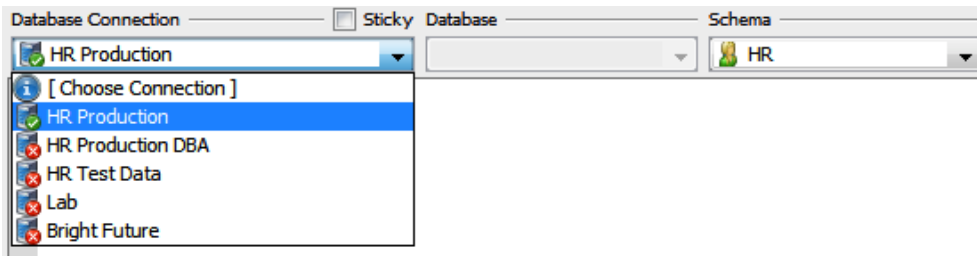


Figure: Database Connection, Database and Schema lists

If you check the **Sticky** box above the Database Connection, the current connection selection will not change automatically when passing SQL statements from other parts of DbVisualizer, for instance, when opening a Bookmark. Consider an Bookmark defined for database connection "ProdDB". If the Sticky checkbox is not checked (i.e., disabled), the database connection is automatically changed to ProdDB when you open the Bookmark in the SQL Editor. However, if the Sticky checkbox is checked (i.e., enabled), the current database connection setting is unchanged. The Sticky setting is per SQL editor instance.

The **Database list** (or Catalog) defines which catalog in the connection is the target for the execution. Since not all databases use catalogs, this list may be disabled.

For most databases, the schema selected in the **Schema list** is used only to limit the tables the auto completion feature shows in the completion pop-up; it does not define a default schema for tables referenced in the SQL, because most databases do not allow the default schema to be changed during a session. For the databases that do allow the default schema to be changed, however, the selected schema is also used as the default schema, i.e., the schema used for unqualified table names in the SQL. Currently, the databases that support setting a default schema are DB2, HP Neoview, JavaDB/Derby and Oracle. If you don't want the selected schema to be used as the default schema for these database, you can disable this behavior in the Tool Properties, under the database node's SQL Editor settings.

Limiting Result Set size (Max Rows/Chars)

The **Max Rows** field in the SQL Editor toolbar is used to control how many rows DbVisualizer will fetch for each result set. If there are more rows available than presented in the result set, you will see a warning indicator in the grid status bar.

Setting Max Rows technically means that it is the JDBC driver limiting the rows. This may for some databases also affect non result set operations such as DELETE. MS SQL Server is one example.

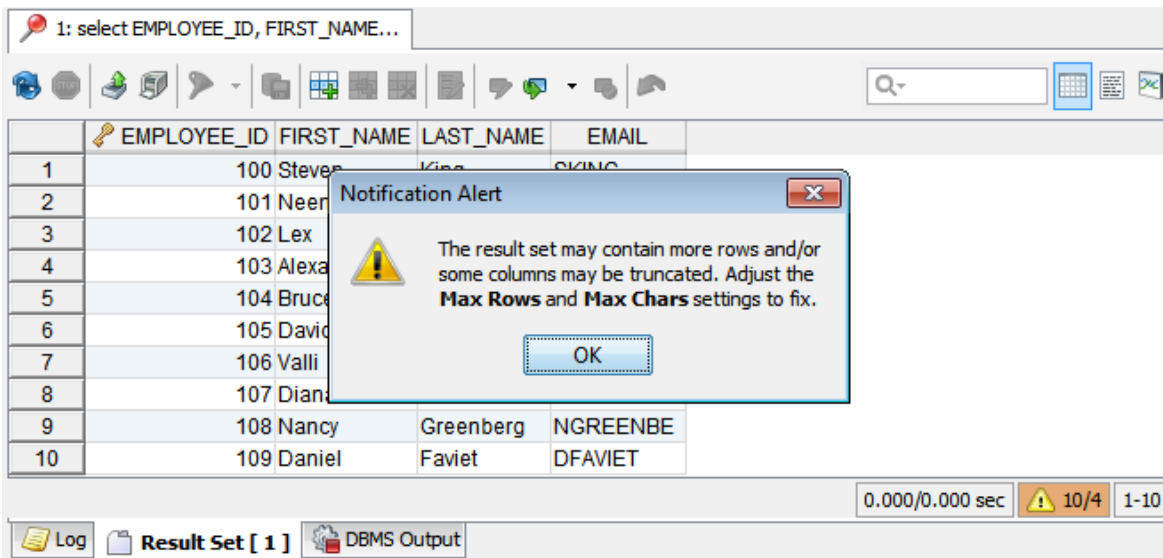


Figure: Max Rows exceeded warning

Clicking on the icon below the grid shows more information about the warning.

Setting **Max Chars** limits the number of characters that are presented for text data. A column that contains values with more characters than the specified Max Chars is shown with a different background color to highlight that it is truncated.

Load from and save to file

The SQL editor supports loading statements from a file and saving the content of the editor to a file. Use the standard file operations, **Open**, **Save** and **Save As** in the **File** main menu or the toolbar to accomplish this. Loading a file always loads it into the currently selected editor.

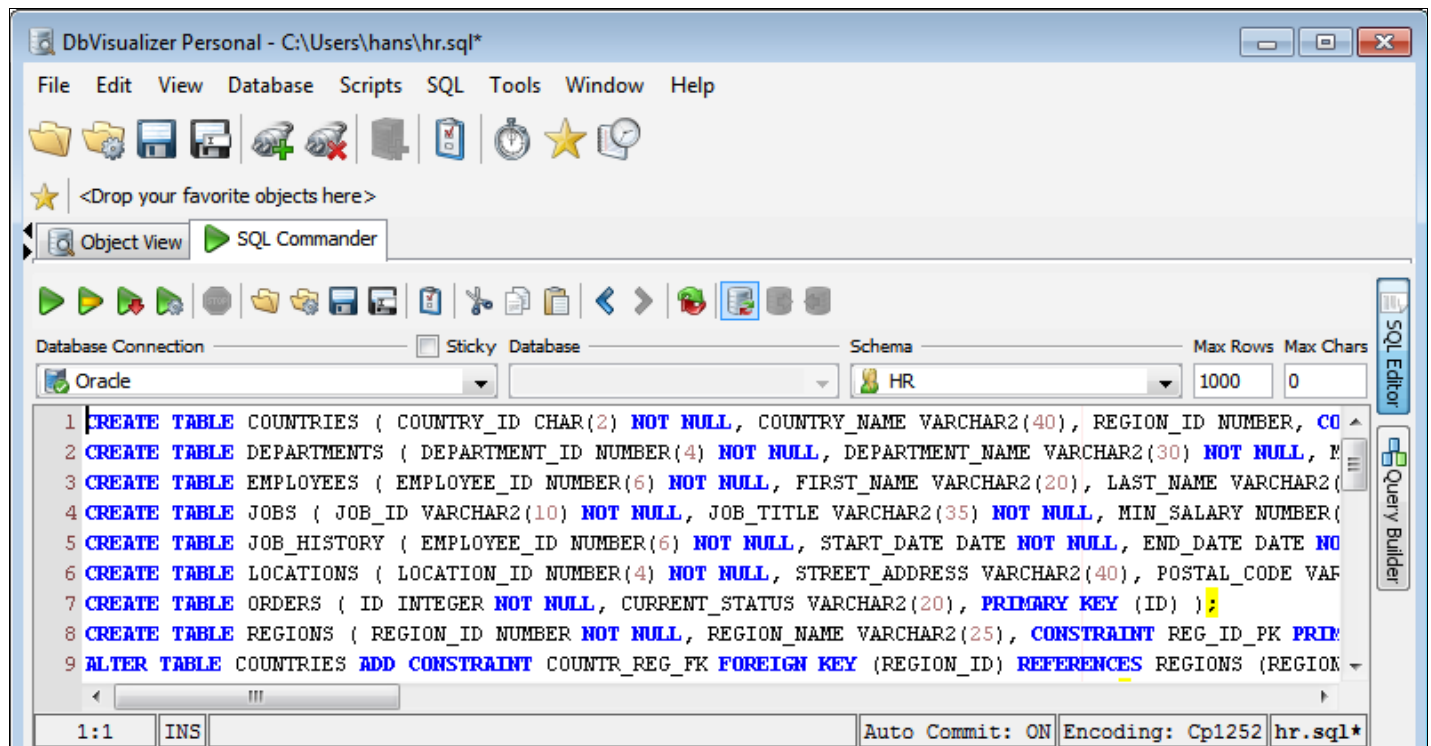


Figure: Loading a file into the SQL Commander

The name of the loaded file is listed in the status bar of the editor, with the full file path shown in the window title. The editor tracks any modifications and indicates changes with an asterisk (*) after the filename.

When you exit DbVisualizer, you are asked what to do if there are any pending edits that need to be saved.

Load Recent

The **File->Load Recent** sub menu lists the recently loaded files. When you choose an entry, the file is opened in the current SQL editor.

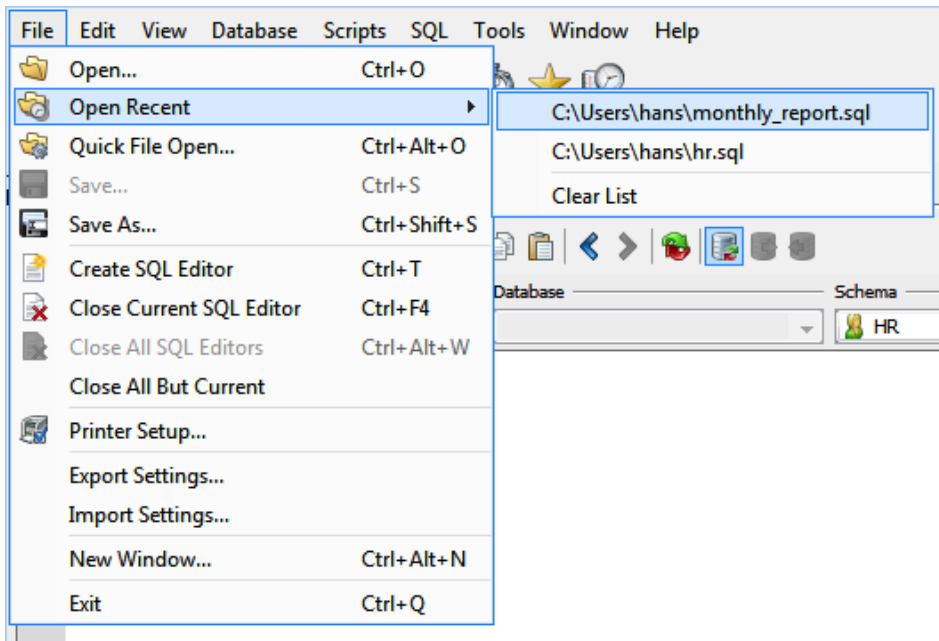


Figure: Open Recent Files menu

Quick File Open

You can also use the Quick File Open feature to open recent files as well as [Bookmarks and History entries](#). By default, it is bound to the **Ctrl+Alt+O** key combination, and is also available via a toolbar button in the SQL Editor as well as in the main **File->Quick File Open** menu.

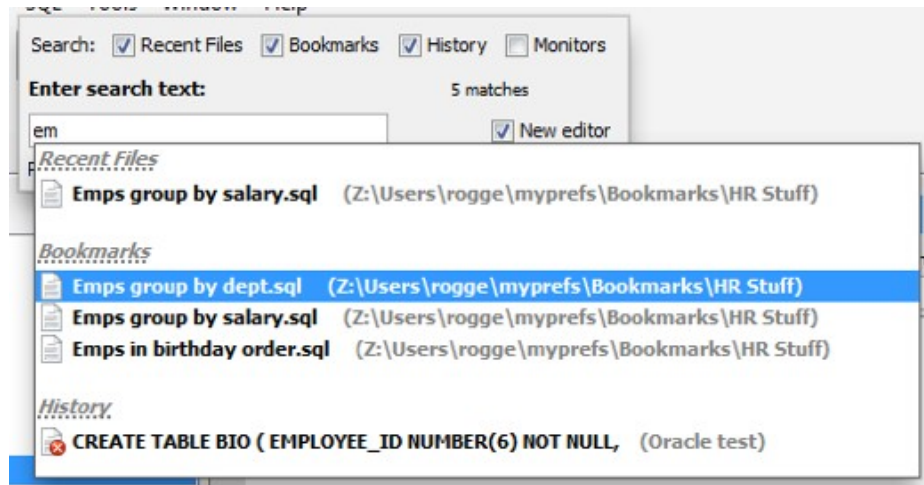


Figure: Quick File Open dialog

Editor Preferences

The Editor preferences pane is activated via the **SQL->Show/Hide Editor Controls** main menu option. It holds settings that control the appearance of the SQL editor, result sets and the log.

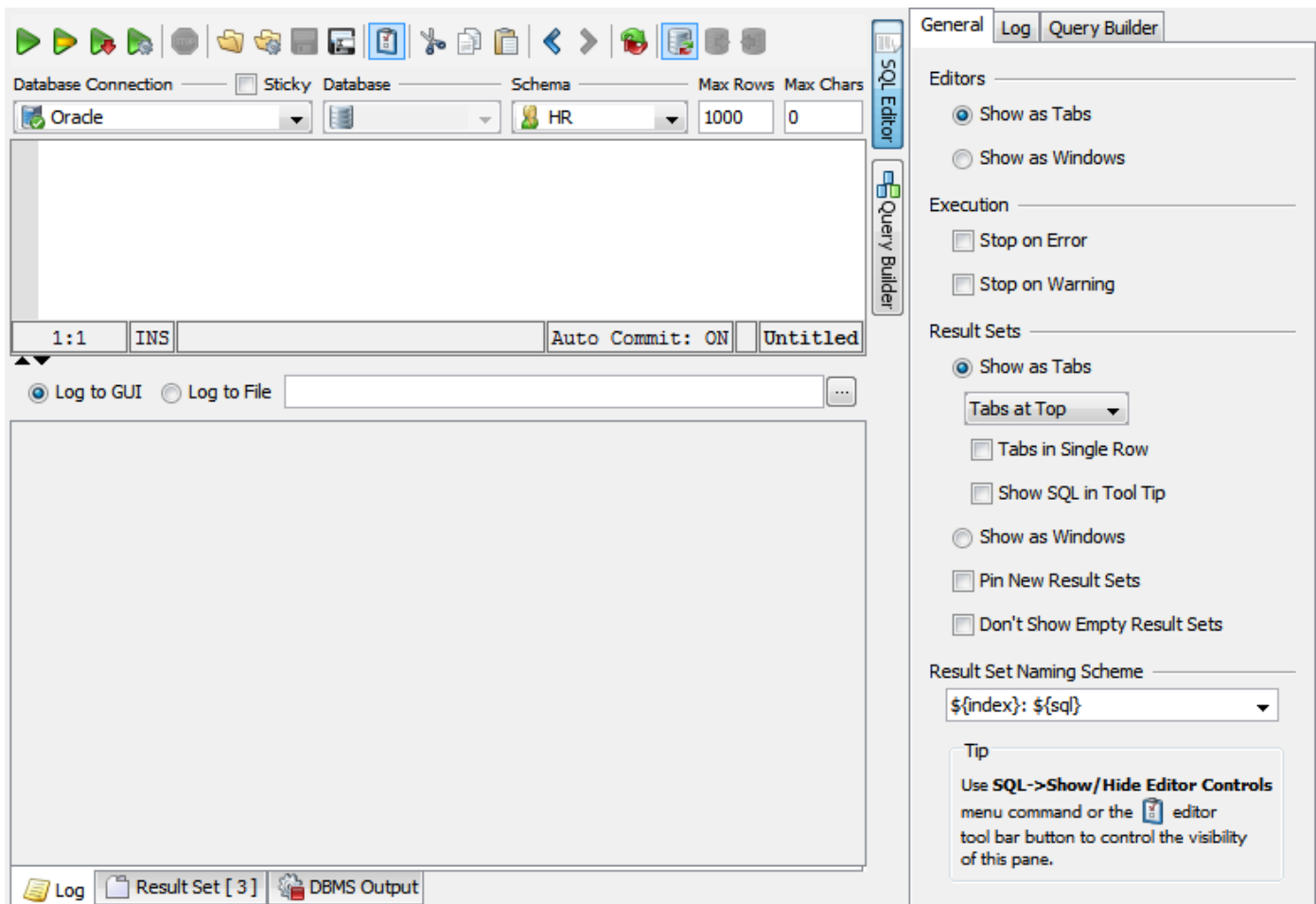


Figure: Editor preferences pane

All settings made in the editor preferences pane are saved between invocations.

Tip: The **Result Set Naming Scheme** may include HTML code, typically used to change the style of the elements.

Example: `<html>${index}: ${sql} (${rows})</html>`

Multiple editors

There is always one default editor named **Main Editor**. You can open additional SQL editors with the **File->Create SQL Editor** main menu operation. Editors can be organized as tabs or internal windows using the **View** buttons. To remove all but the Main Editor select the **File->Close all SQL Editors** menu operation.

Permissions

All SQL commands executed in the SQL Commanded are checked with the DbVisualizer Permission verifier before being executed by the database server. The permission verifier use various rules to determine if a specific SQL is allowed, denied or need confirmation before being executed. You can specify the rules for the verifier in [Tool Properties->Permissions](#).

Charsets and Fonts

You can change the SQL editor font, which is useful and necessary in order to display characters for languages like Chinese, Japanese, etc.

```
1 insert into RESTAURANTS (ID, NAME, SPECIALITY, STYLE, LOCATION) values (
2 1, 'すべてのための肉', 'ミートボール', '現代', '東京');
3
4 insert into RESTAURANTS (ID, NAME, SPECIALITY, STYLE, LOCATION) values (
5 2, 'どこでもすべてのための魚', '魚', '古典的', '北京');|
```

5:38 INS

Figure: SQL Editor with another font

Open [Tool Properties](#) and select the Font category to set the font for the SQL Editor. (It is a good idea to set the same font for both the SQL editor and the grid components).

Displaying data correctly is not just a matter of setting the font, because the character encoding on the client side (in which DbVisualizer runs) and in the database server may not be compatible. There is experimental support to set encodings to accomplish proper conversation between different encodings. Please see the [Getting Started and General Overview](#) document for more information.

Key Bindings

The editor shortcuts, or key bindings, can be redefined in the Tool Properties **Key Bindings** category. Select the **Editor Commands** folder to browse all editor actions.

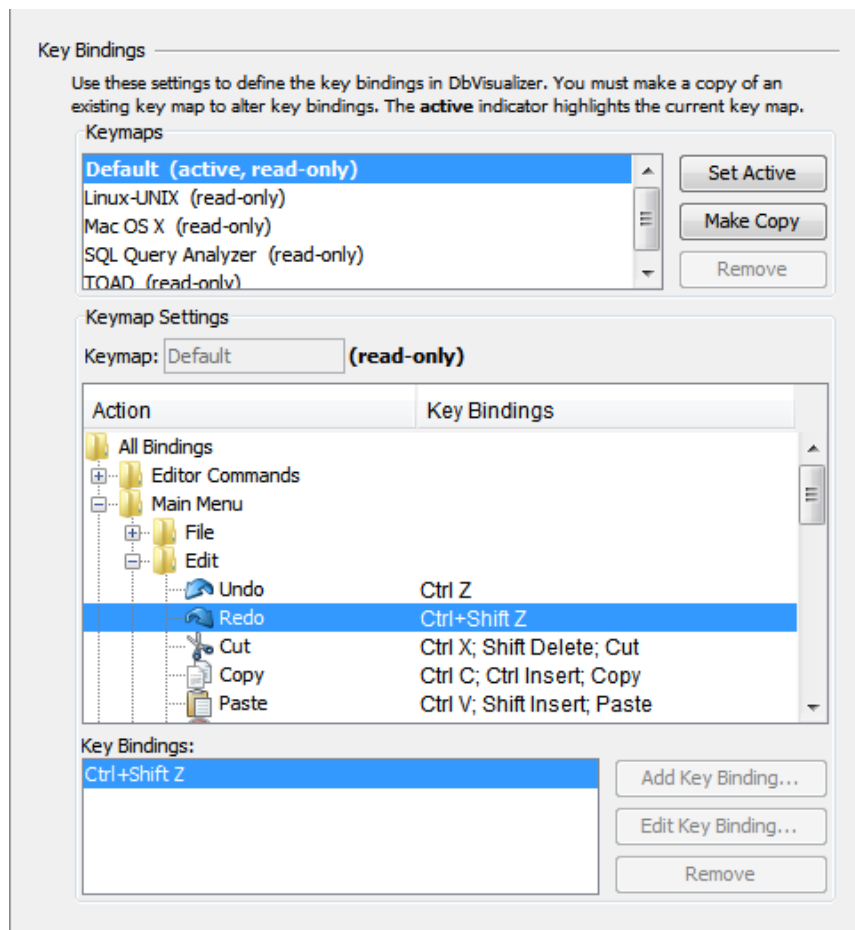


Figure: The Key Bindings editor in Tool Properties

Read more about configuring key bindings in the [Tool Properties](#) document.

Client-Side Comments

Comments in the SQL Editor are identified by the comment identifiers in Tool Properties. These are client side comments and are removed by DbVisualizer before execution.

Sometimes the comments need to be passed to the database. Oracle, for example, uses the block comment identifier to express **optimizer hints** for the database. These must be passed to the database for processing. To enable this, just change the delimiters for the block comment to something that doesn't interfere with the /*+ ... */ notation that Oracle uses. An example is that you add a space after /*.

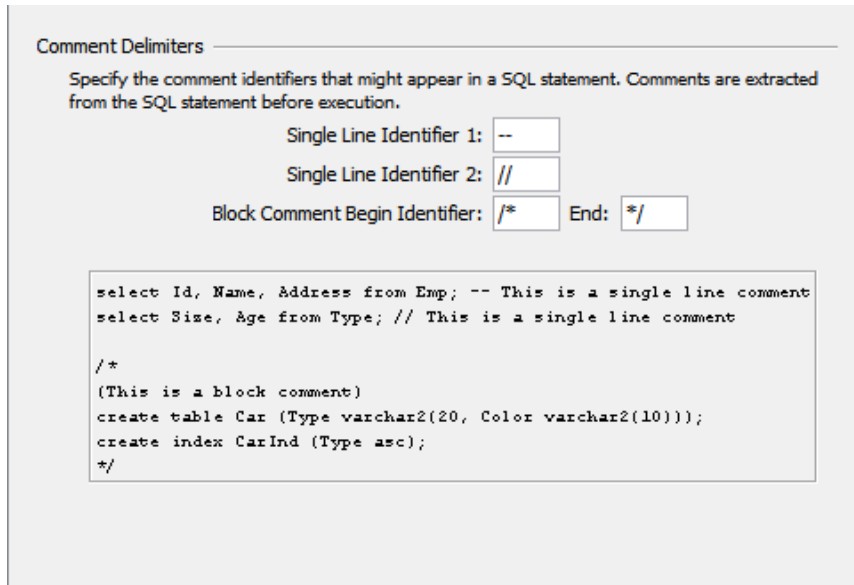


Figure: The SQL Editor -> Comments category in Tool Properties

Auto Completion

Auto completion is a convenient feature used to assist you when editing SQL statements.

The following figure shows the completion pop up with table names.



Figure: Auto completion pop up showing table names

Here is another completion pop-up showing column names.

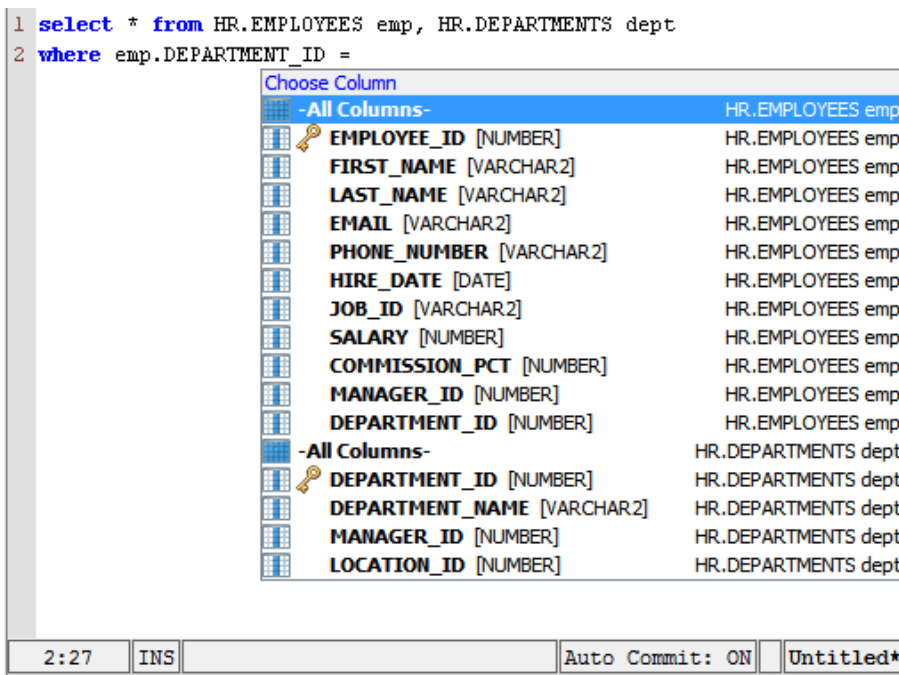


Figure: Auto completion pop up showing column names

DbVisualizer currently provides auto completion for table and columns names for the following DML commands:

- SELECT
- INSERT
- UPDATE
- DELETE

To display the completion pop-up, use the key binding Ctrl-SPACE. You select an entry in the pop-up menu with a mouse double-click, the ENTER key, or the TAB key. To cancel the pop-up, press the ESC key.

Tip: The **SPACE** key can be configured to select entries in the pop up. Do this in the Tool Properties **General->Key Bindings** category. Select the Editor Commands key bindings and add the SPACE key for the **Insert Newline** editor action.

Note 1: If there are several SQL statements in the editor then make sure to separate them using the statement delimiter character (default to ";").

Note 2: In order for the column name completion pop-up to appear, you must first make sure there are table names in the statement.

Note 3: All table names that has been listed in the completion pop-up are cached by DbVisualizer to make sure subsequent displays of the pop-up is performed quickly without asking the database. The cache is cleared only when doing a **Refresh** in the database objects tree or reconnecting the database connection.

Note 4: The **Schema** list above the editor is used to assist the auto completion feature to limit which tables to list in the pop-up.

General display settings for the auto completion feature are managed in [Tool Properties](#).

Here are some examples of how the auto completion works depending on when it is activated. The <AC> symbol indicates the position where the auto completion pop-up is requested. The currently selected catalog is empty and the selected schema is HR. (These examples are when accessing an Oracle database).

SQL	Result
<code>select * from <AC></code>	Shows all tables in the HR schema (since HR is the selected schema)
<code>select * from SYS.<AC></code>	The pop up displays all tables in the SYS schema independent of the schema list selection
<code>select * from SYS.a<AC></code>	Lists all tables in the SYS schema beginning with the A character

<code>select <AC> from SYS.all_objects</code>	Lists all column in the SYS.all_objects table
<code>select <AC> from SYS.all_objects all, EMPLOYEES</code>	Lists all columns in the SYS.all_objects and EMPLOYEES table (in the HR schema)
<code>select emp.<AC> from EMPLOYEES emp</code>	Lists all columns in the EMPLOYEES table, here identified by the alias emp
<code>select emp.N<AC> from EMPLOYEES emp</code>	Lists all columns in the EMPLOYEES table identified by alias emp starting with the N character
<code>insert into EMPLOYEES (<AC></code>	Lists all columns in the EMPLOYEES table. Selecting the -All Columns- in the pop-up results in all columns being added, comma separated.

It is possible to fine-tune how auto completion shall work in the connection properties. The following settings can be used to adjust whether table and column names should be qualified.

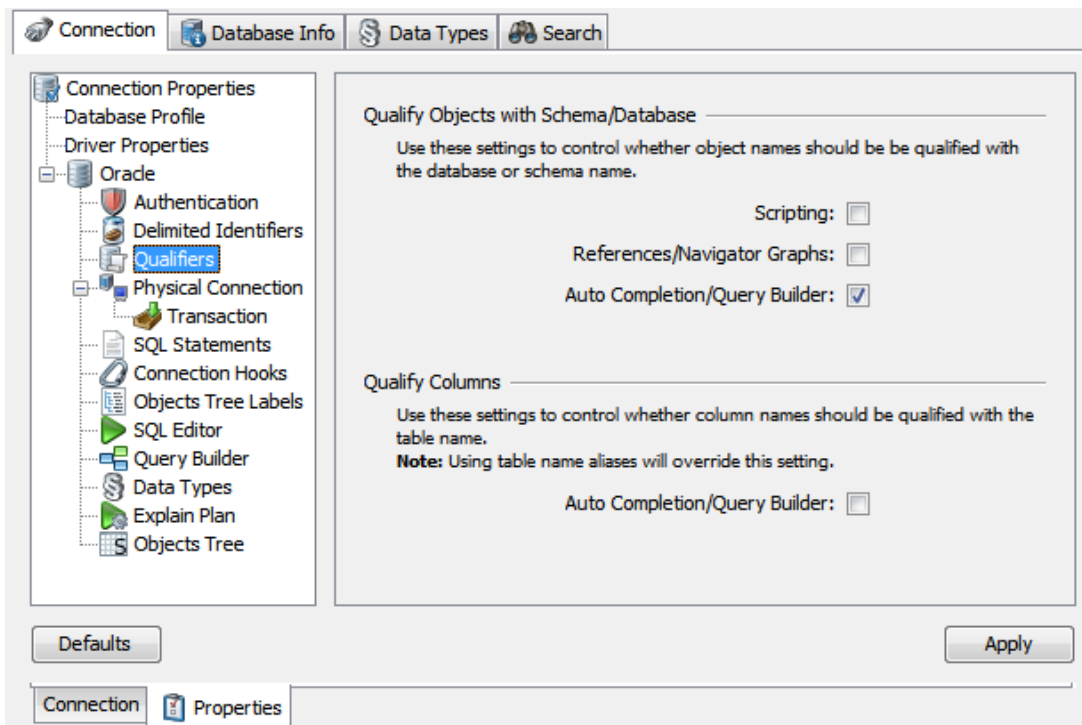


Figure: Properties controlling auto completion qualifiers

With **Qualify disabled** (for both table names and columns):

```
select Name, Address from EMPLOYEE where Id > 240
```

With **Qualify enabled**:

```
select EMPLOYEE.Name, EMPLOYEE.Address from HR.EMPLOYEE where EMPLOYEE.Id > 240
```

(The setting of Qualify Columns is ignored when an alias is used for a table name in the SQL).

The property settings in the figure below define whether delimited identifiers should be part of the completed SQL.

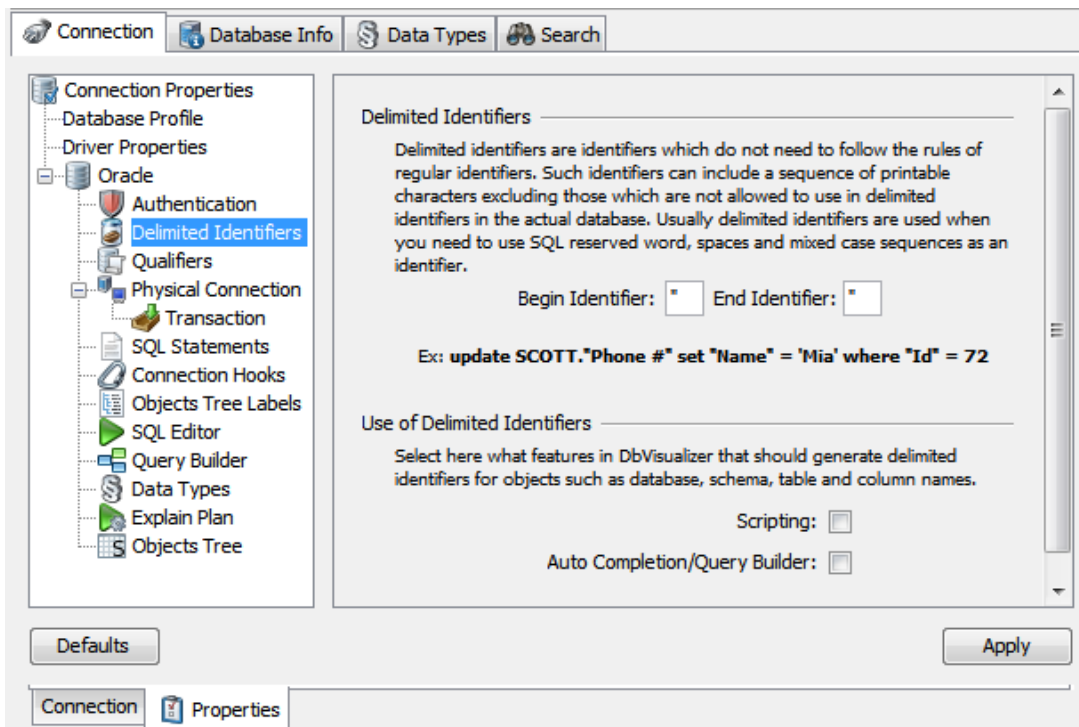


Figure: Properties controlling delimited identifiers for auto completion

With **Delimited Identifiers** disabled:

```
select Name, Address from HR.EMPLOYEE where Id > 240
```

With **Delimited Identifiers** enabled:

```
select "Name", "Address" from HR."EMPLOYEE" where "Id" > 240
```

SQL Formatter

The **SQL->Format SQL** feature is used to format the complete editor buffer or current SQL (at cursor position) according to the settings defined in the Tool Properties **SQL Editor->SQL Formatting** category. There are many things you can configure. After making some changes, press **Apply** and format again to see the result. The formatter can work with the source SQL enclosed in quotes (e.g., when copied from a program), and it can format the final SQL for inclusion in a program written in languages like Java, C#, PHP, VB, etc.

Example of the SQL before formatting:

```
select
CompanyName, ContactName, Address,
City, Country, PostalCode from
Northwind.dbo.Customers OuterC
where CustomerID in (select top 2 InnerC.CustomerId
from Northwind.dbo.[Order Details] OD
join Northwind.dbo.Orders O on OD.OrderId = O.OrderID
join Northwind.dbo.Customers InnerC
on O.CustomerID = InnerC.CustomerId
Where Region = OuterC.Region
group by Region, InnerC.CustomerId
order by sum(UnitPrice * Quantity * (1-Discunt)) desc)
order by Region
```

And after formatting has been applied:

```

SELECT
  CompanyName,
  ContactName,
  Address,
  City,
  Country,
  PostalCode
FROM
  Northwind.dbo.Customers OuterC
WHERE
  CustomerID in
  (
  SELECT
  top 2 InnerC.CustomerId
  FROM
  Northwind.dbo.[
  ORDER
  Details] OD
  JOIN
  Northwind.dbo.Orders O
  ON
  OD.OrderId = O.OrderID
  JOIN
  Northwind.dbo.Customers
  InnerC
  ON
  O.CustomerID =
  InnerC.CustomerId
  WHERE
  Region = OuterC.Region
  GROUP BY
  Region,
  InnerC.CustomerId
  ORDER BY
  sum(UnitPrice * Quantity *
  (1-Discount)) desc
  )
ORDER BY
  Region

```

History

The SQL Editor keeps track of all executed SQL statements. You can use the **Previous** and **Next** buttons in the editor toolbar to walk forward and backward through the statements. They insert the previously executed SQL, with accompanying settings for **Database Connection**, **Catalog** and **Schema** (if **Sticky** is disabled). Please see [Bookmarks and History](#) for more details.

Bookmarks

Bookmarks are used to save frequently used SQL statements between invocations of DbVisualizer. They are managed primarily in the Scripts tab but edited and executed in the SQL Commander. Please see [Bookmarks and History](#) for more details.

Execution

The execution of multiple SQL statements can be controlled using the **Stop Execution On** controls. These define whether the execution of the following SQL statements will be stopped based on two states:

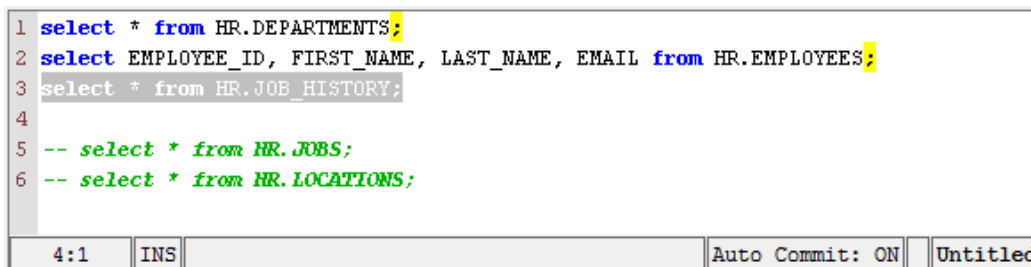
- **Errors**
Stop the execution if the SQL resulted in an error
- **Warnings**
Stop the execution if the SQL executed successfully but no rows were affected

SQL->Execute

Use the **SQL->Execute** main menu operation to execute the SQL in the current SQL editor. The SQL Commander executes the statements one by one and indicates the progress in the log area. The currently selected Database Connection is used for all statements. The SQL Commander does not support executing SQLs for multiple database connections in one batch.

The result of the execution is displayed in the output view based on what result(s) are returned. If there are several results and an error occurred in one of them, the **Log** view is automatically displayed to indicate the error.

If you select a statement in the SQL editor and choose **SQL->Execute** main menu option, only the selected statement is executed. This is a useful feature when you have several SQL statements in the SQL editor and you just want to execute one or a few of the statements.



```
1 select * from HR.DEPARTMENTS;
2 select EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL from HR.EMPLOYEES;
3 select * from HR.JOB_HISTORY;
4
5 -- select * from HR.JOBS;
6 -- select * from HR.LOCATIONS;
```

4:1 | INS | Auto Commit: ON | Untitled

Figure: Selection execute

In the above figure, only the highlighted statement is being executed.

Comments in the SQL editor are not sent to the database when you use **SQL->Execute**. If you want comments to be preserved when creating or changing a stored procedure or function, please use the [Create Procedure dialog and Procedure Editor](#) instead of the SQL Commander.

SQL->Execute Current

The **Execute Current** operation is useful when you have a script with several SQL statements. It lets you execute the statement at the cursor position without first having to select the SQL statement. The default key binding for execute current is **Ctrl-PERIOD** (Ctrl-.).

Execute Current determines the actual statement by parsing the editor buffer using the standard statement delimiters.

Tip: If you are unsure what the boundaries are for the current statement then use **Edit->Select Current Statement**. This will highlight the current statement without executing it.

SQL->Execute Buffer

Execute Buffer sends the complete editor buffer for execution as one statement. No comments are removed and no parsing of individual statements based on any delimiters is made. This operation is useful when executing anonymous SQL blocks or SQLs used to create procedures, functions, etc.

SQL->Execute Explain Plan (Oracle, SQL Server and DB2)

Explain Plan is supported for Oracle, DB2 and SQL Server. Explain Plan executes your query and records the plan that the database devises to execute it. By examining this plan, you can find out if the database is picking the right indexes and joining your tables in the most efficient manner. The explain plan feature works much the same as executing SQLs to present result sets; you may highlight statements, run a script or load from file. The explain plan results can easily be compared by using the pin feature in combination with window style presentation.

DbVisualizer presents the plan either in a tree style format or in a graph. What information is shown depends on what database it is. In the tree view put the mouse pointer on the column header for a tooltip description what that column represents. The following screenshot shows the SQL in the editor at top and the corresponding explain plan as the result.


```

1 SELECT
2   d.DEPARTMENT_NAME,
3   l.CITY,
4   c.COUNTRY_NAME,
5   r.REGION_NAME
6 FROM
7   HR.DEPARTMENTS d,
8   HR.LOCATIONS l,
9   HR.COUNTRIES c,
10  HR.REGIONS r
11 WHERE
12  d.LOCATION_ID = l.LOCATION_ID
13 AND l.COUNTRY_ID = c.COUNTRY_ID
14 AND c.REGION_ID = r.REGION_ID
15 AND d.MANAGER_ID IN
16  (
17    SELECT
18     EMPLOYEE_ID
19    FROM
20     HR.EMPLOYEES
21    WHERE FIRST_NAME LIKE 'A%'
22  )

```

21:15 INS Auto Commit: ON Untitled*

EXPLAIN 1: SELECT d.DEPARTMENT_NAME, l.CI...

Tree View Graph View

Operation	Node Cost (%)	Cost	CPU Cost	I/O Cost	Optimizer	Cardinality
SELECT STATEMENT	0.0 %	6	363492	6	ALL_ROWS	3
NESTED LOOPS	0.0 %	6	363492	6		3
NESTED LOOPS	16.7 %	5	338468	5		3
NESTED LOOPS	0.0 %	3	244292	3		11
NESTED LOOPS	0.0 %	2	150072	2		23
HR.COUNTRY_C_ID_PK INDEX (FULL SCAN)	16.7 %	1	12121	1	ANALYZED	25
HR.LOCATIONS TABLE ACCESS (BY INDEX ROWID)	16.7 %	1	8871	1	ANALYZED	1
HR.LOC_COUNTRY_IX INDEX (RANGE SCAN)	0.0 %	0	1250	0	ANALYZED	2
HR.DEPARTMENTS TABLE ACCESS (BY INDEX ROWID)	16.7 %	1	9689	1	ANALYZED	1
HR.DEPT_LOCATION_IX INDEX (RANGE SCAN)	0.0 %	0	1650	0	ANALYZED	4
HR.EMPLOYEES TABLE ACCESS (BY INDEX ROWID)	16.7 %	1	8561	1	ANALYZED	1

Log Result Set [1] DBMS Output

Figure: Explain Plan presented as a tree

The Graph View shows the plan as a graph. The graph can be exported to an image file or printed. Use the menubar buttons to export and print.

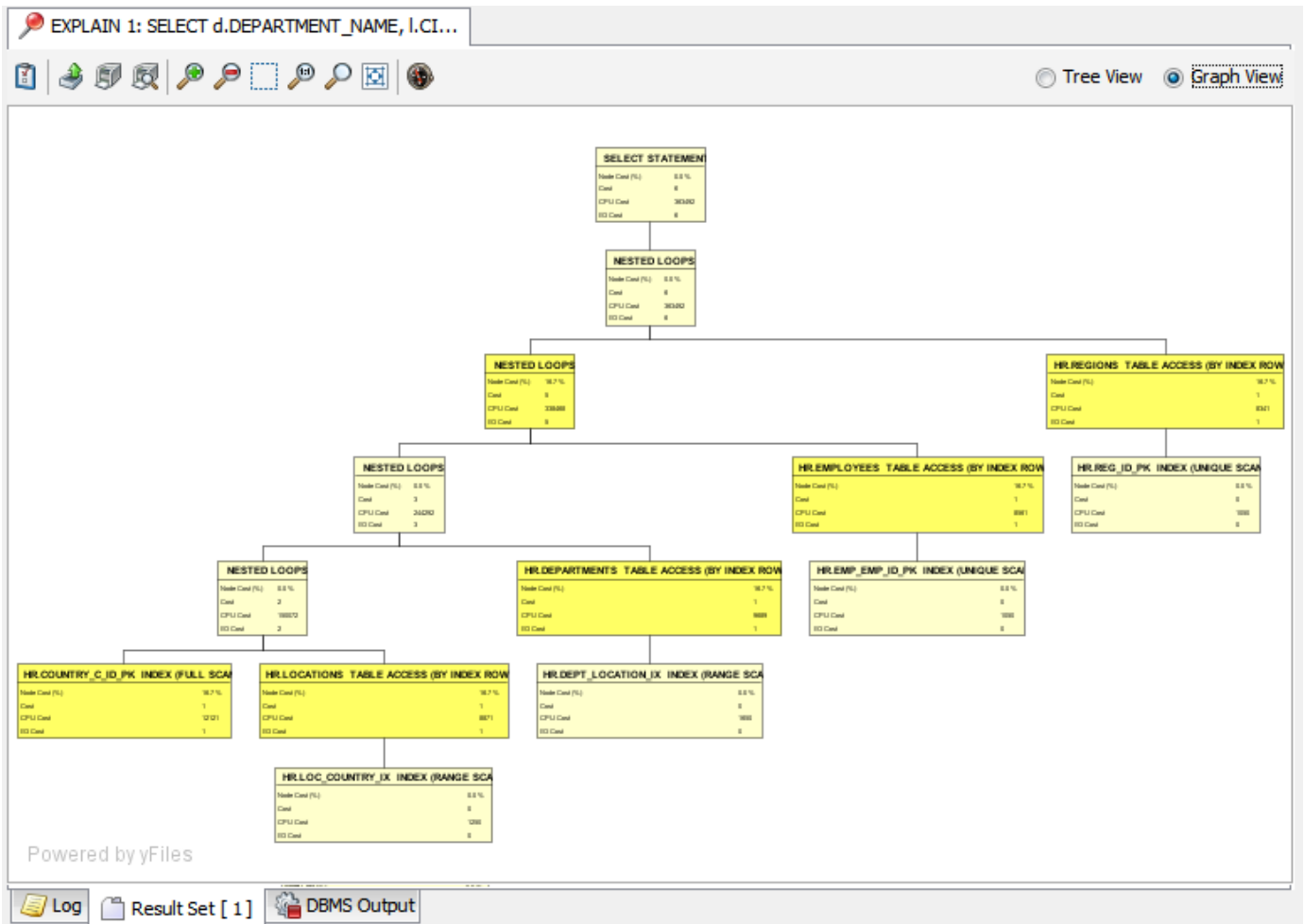


Figure: Explain Plan presented as a graph

Each of the supported databases use different techniques to manage their explain plan support. To control this, either click the **Preferences** toolbar button or go to **Connection Properties->[database]->Explain Plan**.

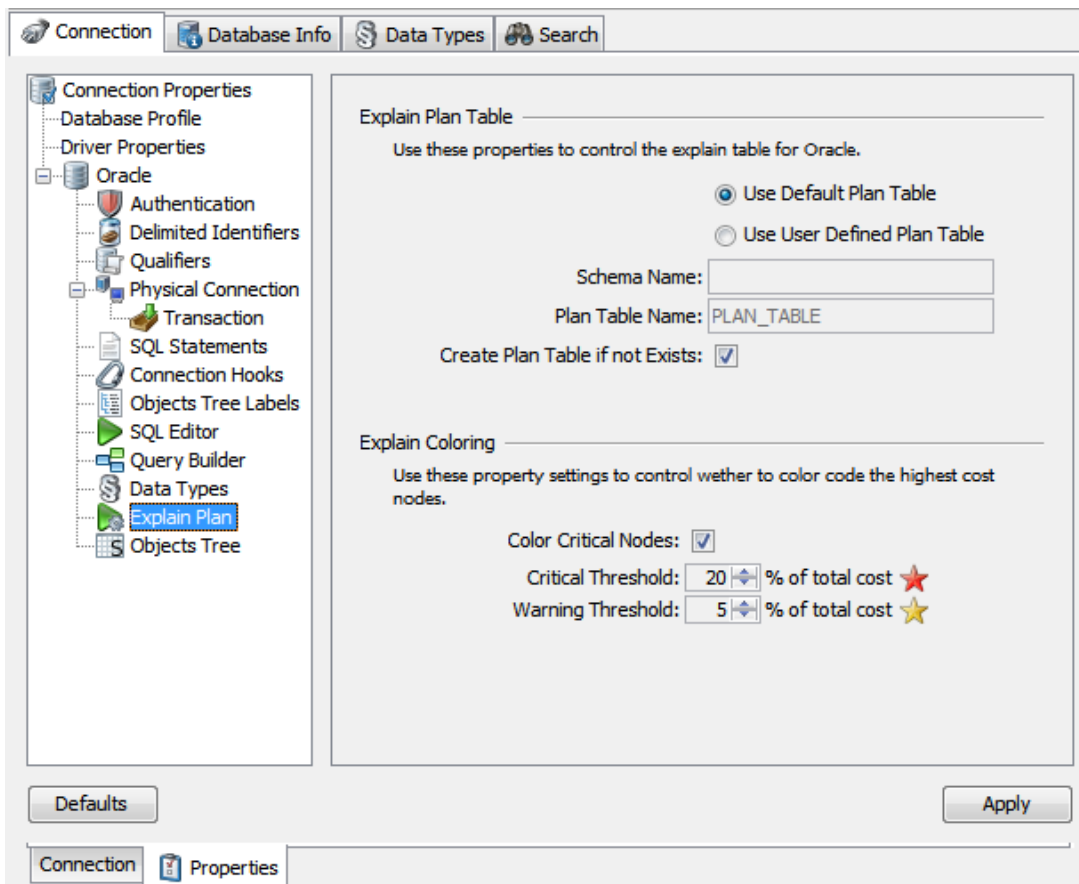


Figure: Explain Plan configuration

The configuration options are different for each of the supported databases.

Auto Commit, Commit and Rollback

The commit and rollback buttons and the accompanying operations in the **SQL** main menu are enabled if the **Auto Commit** setting is off for the current SQL editor. The default setting for Auto Commit is on, which means that the driver/database automatically commits each SQL that is executed. If Auto Commit is disabled, it is very important to manually issue the commit or rollback operations when appropriate.

The following commands can be executed in the SQL Commander for database independent commit and rollback:

```
@commit
@rollback
```

The Auto Commit setting is enabled by default and can be adjusted in the Connection Properties. You may also adjust the auto commit state for the SQL editor you are using in the SQL Commander with the following command:

```
@set autocommit true/false
```

SQL Scripts

An SQL script is composed of several SQL statements and can be executed in a batch. Each SQL statement is separated by a single character, a sequence of characters, or the word "go" on a single line. The default settings for the separator characters are defined in Tool Properties and can be modified to match your needs.

SQL Statement Delimiter _____

The character(s) that delimits one SQL statement from another during execution.

The delimiter is ignored if found within apostrophes ('), double quotes (" "), in a single line comment or in a block comment. It is also ignored if found in a Variable.

SQL Statement Delimiter 1:

SQL Statement Delimiter 2:

Allow "go" as Delimiter _____

Check to enable **go** as the only word on a line as an additional SQL statement delimiter.

Allow "go" as Delimiter:

Anonymous SQL Block Identifiers _____

The character(s) used in the SQL Commander to identify the begin and end of an anonymous SQL block.

Begin Identifier:

End Identifier:

Figure: Statement Delimiters

The following SQL script illustrates some uses of the SQL statement delimiters based on the settings in the previous figure:

```
select * from MyTable;                /* Stmt 1 */
insert into table MyTable            /* Stmt 2 */
  (Id, Name) /* This is a comment */ values (1, 'Arnold')
go
update MyTable set Name = 'George' where Id = 1;    /* Stmt 3 */
select * from                               /* Stmt 4 */
  MyTable; // This is a comment
```

You can also use the [@delimiter client side command](#) to temporarily change the delimiter in a script.

Execute Large SQL Scripts

If you have a large script (tens of MB), loading it into the SQL Commander and generating log entries in the GUI for all statements require a lot of memory.

For a script that is large but still small enough to load into the SQL Commander, you can save memory (and therefore run it faster and more efficiently) by selecting to log to a file instead of the GUI:

Log to GUI Log to File

To save even more memory, you can use the [@run client side command](#) to run the script without loading it into the SQL Commander:

```
@run my_huge_script.sql;
```

The `@run` command reads one statement at a time from the file. There are, however, still a few things that require the whole file to be read before the statements can be executed: parsing the script for variables, parameter markers, and restricted commands, as well as counting all statements in order to provide progress information. When you run a script that is large enough (more than 10 MB) for these things to potentially cause memory problems and slow down the processing, DbVisualizer gives you a chance to turn off this preprocessing and progress reporting so that the statements instead can be executed directly as they are read from the file, one at a time.

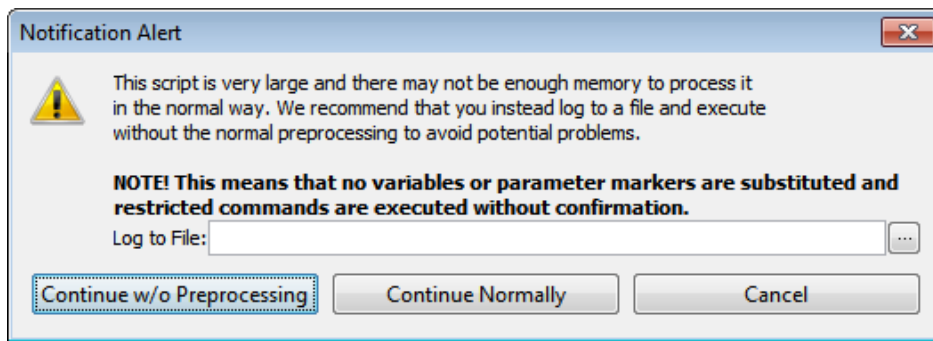


Figure: Disable preprocessing dialog

To ensure that you don't have any problems running scripts this large, you must specify a file for logging. We also strongly recommend that you click **Continue w/o Preprocessing**, thereby disabling all variable, parameter and restricted commands processing. Only click **Continue Normally** if you know for sure that you have enough memory available and have adjusted your installation so that DbVisualizer can use it. With the preprocessing disabled and all logging going to a file instead of the GUI, you should be able to execute scripts of any size (we have tested with scripts as large as 4 GB).

Another alternative for execution of large scripts is to use the DbVisualizer [command line interface](#) instead of the GUI application. This option is the absolute most efficient and fastest.

Anonymous SQL Blocks

An anonymous SQL block is a block of code which contains not only standard SQL but also proprietary code for a specific database. The anonymous SQL block support in the SQL Commander uses another technique in the JDBC driver to execute these blocks. The way you tell the SQL Commander that a SQL block is to be executed is to insert a begin identifier just before the block and an end identifier after the block. The figure in the previous section shows these settings and the default values for the **Begin Identifier** it is `--/` and for the **End Identifier** it is `/`.

Here follows an example of an anonymous SQL block for Oracle:

```
--/ script to disable foreign keys

declare cursor tabs is select table_name, constraint_name
  from user_constraints where constraint_type = 'R' and owner = user;

begin
  for j in tabs loop
    execute immediate ('alter table '||j.table_name||' disable constraint'||j.constraint_name);
  end loop;
end;
/
```

If you want to execute the complete editor buffer as an anonymous SQL block, use the SQL->Execute Buffer operation. In this case, you do not need the begin and end identifiers.

Stored Procedures

With DbVisualizer Personal, you can execute stored procedures and functions for databases with extended support using the [@call client side command](#) described below and in the [Procedure Editor](#) chapter.

For other databases it is not officially supported by DbVisualizer even though the native commands (e.g. CALL or EXEC) work for some databases. The best way to figure it out is to try. Our internal tests show that the Sybase ASE and SQL Server procedure calls work okay with literal IN parameters in the SQL Commander. DbVisualizer also presents multiple result sets from a single procedure call as of version 4.0 for these databases.

Client Side Commands

The SQL Commander supports a number of DbVisualizer specific commands that you can use in the SQL Editor. A command begins with the at sign, "@". The following sections describe the available commands.

@run - run SQL script from file

@cd <directory> - change directory

Use the following commands to locate and execute SQL scripts directly from a file without first loading the file into the SQL editor. This is useful if you are using an external editor or a development environment to edit the SQL but use DbVisualizer to execute it.

- **@run <file>**
Request to execute the file specified as parameter
- **@cd <directory>**
Change the working directory for the following @run command

Example of a script utilizing the file referencing commands:

```
select * from MyTable; -- Selects data from MyTable
@run createDB.sql;     -- Execute the content in the
                       -- createDB.sql file without loading into the SQL editor.
                       -- The location of this file is the same as the working
                       -- directory for DbVisualizer.
@cd /home/mupp;       -- Request to change directory to /home/mupp
@run loadBackup.sql;  -- Execute the content in the loadBackup.sql
                       -- file. This file will now be read from the
                       -- /home/mupp directory.
```

@export - export result sets to file

The @export commands are used to declare that any result sets from the SQL statements that follows should be written to a file instead of being presented in the DbVisualizer tool. This is really useful, since it enables dumping very large tables to a file for later processing or, for example, to perform backups. The following commands are used to control the export:

- **@export on**
Defines that the SQL statements that follows will be exported rather than being presented in DbVisualizer
- **@export set parm1="value1" parm2="value2"**
The set command is used to customize the export process. Check the table below for the complete set of parameters.
- **@export off**
Defines that SQL statements that follows will be handled the normal way, i.e., the result sets are presented in the DbVisualizer tool

These parameters are supported:

Parameter	Default	Valid Values
AppendFile	false	true, false, clear
BinaryFileDir		Directory path for data files when BinaryFormat is set to File
BinaryFormat	Don't Export	Don't Export, Size, Value, Hex, Base64, File
BooleanFalseFormat	false	false, no, 0, off
BooleanTrueFormat	true	true, yes, 1, on
CLOBFileDir		Directory path for data files when CLOBFormat is set to File
CLOBFormat	Value	Don't Export, Size, Value, File
CsvColumnDelimiter	\t (TAB)	

CsvIncludeColumnHeader	true	true, false
CsvIncludeSQLCommand	false	true, false
CvsRemoveNewlines	false	true, false
CsvRowCommentIdentifier		
CsvRowDelimiter	\n	\n (UNIX/Linux/Mac OS X), \r\n (Windows)
DateFormat	yyyy-MM-dd	See valid formats in Tool Properties document
DecimalNumberFormat	Unformatted	See valid formats in Tool Properties document
Destination	File	File
Encoding	UTF-8	
ExcelIncludeColumnHeader	true	true, false
ExcelIncludeSQLCommand	false	
ExcelIntroText		Any description
ExcelTextOnly	false	true, false
ExcelTitle	DbVisualizer export output	Any title
Filename	REQUIRED	
Format	CSV	CSV, HTML, XML, SQL, XLS
HtmlIncludeSQLCommand	false	true, false
HtmlIntroText		Any description
HtmlTitle	DbVisualizer export output	Any title
NumberFormat	Unformatted	See valid formats in Tool Properties document
QuoteDuplicateEmbedded	true	true, false (quote char is the same as QuoteTextData)
QuoteTextData	None	None, Single, Double
Settings		
ShowNullAs	(null)	
SqlIncludeCreateDDL	false	true, false
SqlIncludeSQLCommand	false	true, false
SqlRowCommentIdentifier	--	

SqlSeparator	;	
TableName		Can be set if DbVisualizer cannot determine the value for the <code>#{dbvis-object}</code> variable
TimeFormat	HH:mm:ss	See valid formats in Tool Properties document
TimeStampFormat	yyyy-MM-dd HH:mm:ss.SSSSSS	See valid formats in Tool Properties document
XmlIncludeSqlCommand	false	true, false
XmlIntroText		
XmlStyle	DbVisualizer	DbVisualizer, XmlDataSet, FlatXmlDataSet

Example 1: @export with minimum setup

The following example shows the minimum commands to export a result set. The result set produced by the `select * from Orders` will be exported to the `C:\Backups\Orders.csv` file, using the default settings.

```
@export on;
@export set filename="c:\Backups\Orders.csv";
select * from Orders;
```

Example 2: @export with automatic table name to file name mapping

This example shows how to make the filename the same as the table name in the select statement. The example also shows several select statements. Each will be exported in the SQL format. Since the filename is defined to be automatically set, this means that there will be one file per result set and each file is named by the name of its table.

There must be only one table name in a select statement in order to automatically set the filename, i.e if the select joins from several tables or pseudo tables are used, you must explicitly name the file.

```
@export on;
@export set filename="c:\Backups\#{dbvis-object}%" format="sql";
select * from Orders;
select * from Products;
select * from Transactions;
```

Example 3: @export all result sets into a single file

This example shows how all result sets can be exported to a single file. The `AppendFile` parameter supports the following values.

- **true**
The following result sets will all be exported to a single file
- **false**
Turn off the append processing
- **clear**
Same as the **true** value but this will in addition clear the file before the first result set is exported

```
@export on;
@export set filename="c:\Backups\alltables.sql" appendfile="clear" format="sql";
select * from Orders;
select * from Products;
select * from Transactions;
```


Example 4: @export using predefined settings

The [Export dialogs](#) let you save export settings to a file for later use. Such an export settings file can be referenced in the **@export set** command.

```
@export on;
@export set settings="c:\tmp\htmlsettings.xml" filename="c:\Backups\${dbvis-object}$";
select * from Orders;
select * from Products;
select * from Transactions;
```

The example shows that all settings will be read from the `c:\tmp\htmlsettings.xml` file.

@delimiter - Temporarily change the statement delimiter

When you use SQL statements to create functions and stored procedures in a script that also contains other SQL statements, the statement delimiters for statements within the code body of the CREATE statement often clash with the delimiters for the other statements. One way to handle this is with [Anonymous SQL Blocks](#), but it may be more convenient to temporarily change the statement delimiter. That is what the **@delimiter** command is for:

```
@delimiter ++;
CREATE OR REPLACE FUNCTION HELLO (p1 IN VARCHAR2) RETURN VARCHAR2
AS
BEGIN
RETURN 'Hello ' || p1;
END;
++
@delimiter ;++
@call ${returnValue||(null)||String||nshow dir=out}$ = HELLO('World');
@echo returnValue = ${returnValue}$;
```

The first **@delimiter** command sets the delimiter to "++" so that the default ";" delimiter can be used within the function body in the CREATE statement. The "++" delimiter is then used to end the CREATE statement, and another **@delimiter** command sets the delimiter back to ";" for the remaining commands in the script.

Note that current delimiter must be used to delimit the **@delimiter** command itself from the other statements: the first **@delimiter** command uses ";" and the second uses "++".

@call - Execute a function or stored procedure

You can use the **@call** command to execute a function or a stored procedure.

For a function, returning a value, use this syntax:

```
@call <OutVariable> = <FunctionName>(<ParamList>);
```

where the **<FunctionName>** may need to be fully qualified with a schema (and/or catalog/database) and the **<ParamList>** is a comma separated list of literal values or variables. Here's an example:

```
@call ${return_value||(null)||String||dir=out noshow}$ = get_some_value();
```

For a procedure, use this syntax:

```
@call <ProcedureName>(<ParamList>);
```

where the **<ProcedureName>** may need to be fully qualified with a schema (and/or catalog/database) and the **<ParamList>** is a comma separated list of literal values or variables. Here's an example:

```
@call my_process('literal input',
```

```
{var_in|(null)|String|dir=in}$,  
{var_out|(null)|String|dir=out noshow}$,  
{var_inout|'in_value'|String|dir=inout}$};
```

As shown in these examples, you must use the **dir** option to specify how the variable is to be used (in, out or inout) and you may use the **noshow** option to prevent being prompted for a value for an output variable .

You can use the `@echo` command described earlier to write the value assigned to an output variable to the log.

The [Procedure Editor](#) chapter shows a few more examples, and how you can generate a script for calling a procedure or function. The [Variables](#) section below describes the variable syntax in more detail.

@echo - Echo text

The `@echo` command simply echos the supplied text or the value of a variable in the output.

@window iconify - Iconify the main window

This command results in the main window being lowered (iconified).

@window restore - Raise the main window

This command results in the main window being raised (if iconified).

@desc table - Describe the columns in table

Use the `@desc` command to show column information for a table. For tables that are not in the current database or schema, you need to qualify the table name properly.

```
@desctable;  
@desc database.table;  
@desc schema.table;
```

@ddl - Generate DDL command

The `@ddl` command is used to generate a DDL command (CREATE statement) for a number of different database object types. The command supports this general syntax:

```
@ddl <objType>=<objId> [ drop="true | false" ] [ constrCtrl=<constrCtrl> ]
```

<objType> is one of **table**, **indexesfortable**, **view**, **procedure**, **function**, **package** (Oracle only), **packagebody** (Oracle only), **module** (Mimer only) or **trigger**, and **<objId>** is the qualified identifier for the object (case sensitive).

If **drop** is set to **true**, a DROP statement is included before the CREATE statement.

The **constrCtrl** parameter only applies to tables. It accepts two values: **noconstr** means that no constraints should be included in the statement that can potentially cause creating the table or inserting data into it to fail (FK and CHECK constraints), while **onlyconstr** means that an ALTER statement adding the remaining constraints should be generated instead of a CREATE statement.

@spool log - Save log to file

The `@spool log` command is used to save the log to file. (The log is not cleared after being saved).

```
@spool log mylog.txt
```

@stop on error - Stop execution if any error occurs

@stop on warning - Stop execution if any warning occurs

The @stop on error and warning can be used to control that the script processing should stop if any error or warning occurs. The corresponding @continue on xxx command is used to ignore any error or warning conditions.

```
@stop on error;  
@stop on warning;  
@continue on error;  
@continue on warning;
```

@set autocommit - Sets the auto commit state

Pass either **true** or **false** as a parameter and it will set the auto commit state accordingly.

@commit - Commits the current transaction

Commits the current transaction via this database independent command.

@rollback - Rollback the current transaction

Rollback the current transaction via this database independent command.

@set serveroutput - Enable/disable the DBMS output management for Oracle

Pass either true or false as a parameter to start or stop the DBMS output management for Oracle.

Variables

Variables are used to build parameterized SQL statements and let DbVisualizer prompt you for the values when the SQL is executed. This is handy if you are executing the same SQL repetitively, just wanting to pass new data in the same SQL statement.

Variable Syntax

The variable format supports setting a default value, data type and a few options as in the following example:

```
${FullName||Andersson||String||where pk}$
```

Here is the complete variable syntax:

```
${name || value || type || options}$
```

- **name**
Required. This is the name that appear in the substitution dialog. If multiple variables in a script have the same name, the substitution dialog shows only one and the entered value will be applied to all variables of that name.
- **default**
The default value that appears in the substitution dialog
- **type**

The type of variable - String, Integer, Float, etc. This is used to determine whether the value should be enclosed by quotes. If no type is specified, it is treated as an Integer (no quotes).

- **options**

The options part is used to express certain conditions:

- **pk**
Indicates that the variable is part of the primary key in the final SQL. Represented with a key icon
- **where**
Defines that the variable is part of the WHERE clause. The green star icon further illustrate this condition
- **noshow**
This option define that the variable should not appear in the substitution dialog. A proper default value must be set if using this option, unless it is an output variable (see **dir** below)
- **nobind**
Specifies that the value should be replaced as text in the final statement instead of being replaced as a parameter marker
- **dir=in | out | inout**
The direction for a variable used with the @call command (it is ignored for other uses). A variable assigned the return value for a function must be declared as **dir=out**, and a variable used for a procedure parameter must use a **dir** type matching the procedure parameter direction declaration. **in** is the default.

Pre-defined Variables

A few pre-defined DbVisualizer variables can be used anywhere in the SQL. These are replaced with actual values just before the SQL is sent to the DB server. The final value for these variables are self explanatory.

```
#{dbvis-date}$  
#{dbvis-time}$  
#{dbvis-timestamp}$
```

By default, the values are formatted as defined in **Tool Properties->Data Formats**, but you can also specify a custom format for a single use of the variable, e.g.

```
#{dbvis-date|||||format=[yyyyMMdd]}$
```

The following variables can be used only when monitoring a SQL statement that produce a result set and the **Allowed Row Count** for the monitor is > 0. The output format is seconds and milliseconds. Ex: 2.018

```
#{dbvis-exec-time}$  
#{dbvis-fetch-time}$
```

Note that none of the above variables will appear in the Variable Substitution window explained below.

Variable Substitution in SQL statements

For variable processing to work in the SQL Commander, make sure the **SQL->Process Variables in SQL** main menu option is enabled.

A simple variable may look like this:

```
#{FullName}$
```

A variable is identified by the start and end sequences, **#{ ... }\$**. (These can be re-defined in Tool Properties). During execution, the SQL Commander searches for variables and displays a window with the name of each variable and an input (value) field. Enter the value for each variable and then press **Execute**. This will then replace the variable with the value and finally let the database execute the statement.

Consider the following SQL statement with variables. It is the simplest use of variables as it only contains the variable names. In this case it is also necessary to enclose text values within quotes since the substitution window cannot determine the actual data type from these variable expressions.

```
INSERT  
INTO  
  "SCOTT"."EMP"  
VALUES  
(  
  #{EMPNO}$,  
  '#{ENAME}$',  
  '#{JOB}$',  
  #{MGR}$,  
  '#{HIREDATE}$',  
  #{SAL}$,
```

```
 ${COMM}$/,  
 ${DEPTNO}$/,  
 )
```

Executing the above SQL will result in the following window being displayed:

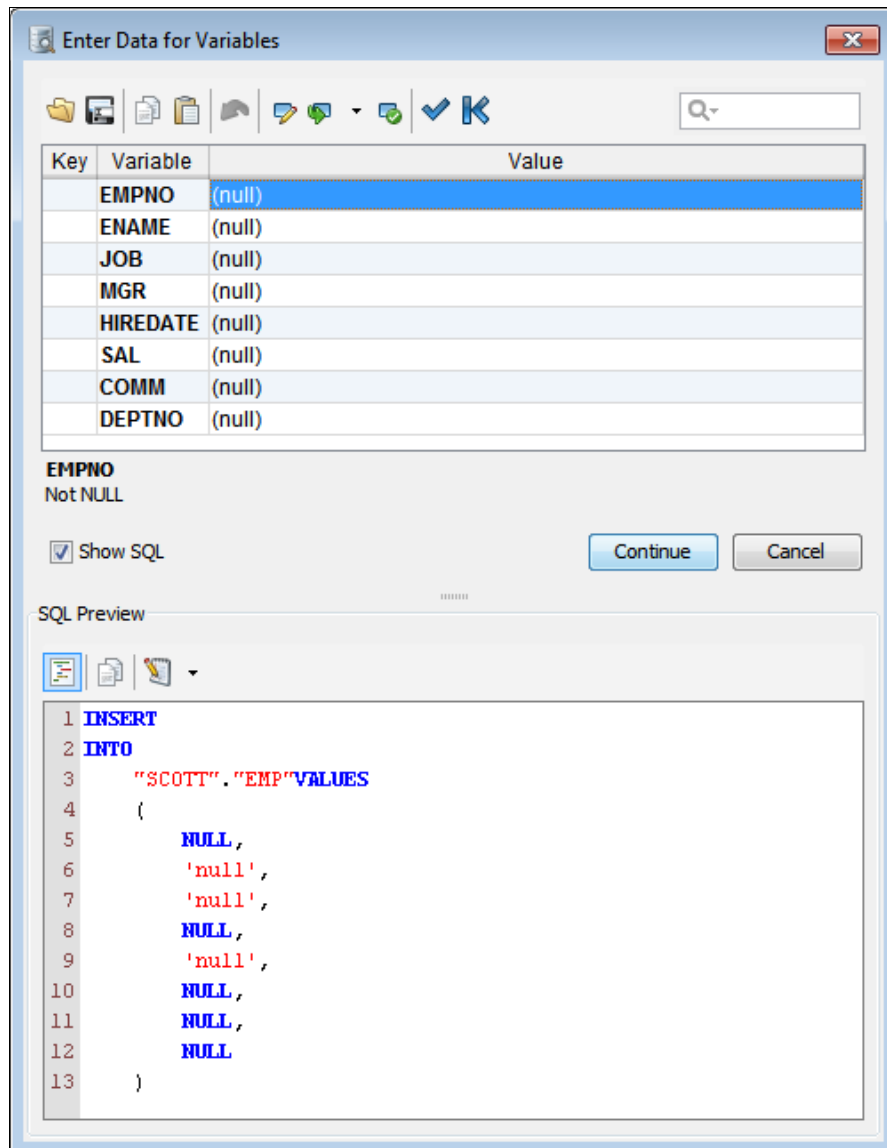


Figure: The substitute variables window

The substitution window has the same look and functionality as the **Form Data Editor** i.e. you can **sort**, **filter**, **insert pre-defined data**, **copy**, **paste** and edit cells in the **multi line editor**, plus a lot of other things. In addition the substitution window adds two new commands (leftmost in the toolbar and in the form right-click menu) specifically for the substitution window:

- **Set Default Values**
This will set the value to the default value for the variable. If a default value was not specified in the variable, **(null)** will appear
- **Set Previously Used Values**
Set the value for each variable to the values used in the previous run (if there are no values from a previous run, this button is disabled).

The **SQL Preview** area shows the statement with all variables substituted with the values.

Here is an example of a more complex use of variables.

```
update SCOTT.EMP set  
EMPNO = ${EMPNO||7698||BigDecimal||pk ds=22 dt=NUMERIC }$,  
ENAME = ${ENAME||BLAKE||String||nullable ds=10 dt=VARCHAR }$,
```

```

JOB = ${JOB||MANAGER||String||nullable ds=9 dt=VARCHAR }$,
MGR = ${MGR||7839||BigDecima||nullable ds=22 dt=NUMERIC }$,
HIREDATE = ${HIREDATE||1981-05-01 00:00:00.0||Timestamp||nullable ds=7 dt=TIMESTAMP }$,
SAL = ${SAL||2850||BigDecima||nullable ds=22 dt=NUMERIC }$,
COMM = ${COMM||(null)||BigDecima||nullable ds=22 dt=NUMERIC }$,
DEPTNO = ${DEPTNO||30||BigDecima||nullable ds=22 dt=NUMERIC }$
where EMPNO = ${EMPNO (where)||7698||BigDecima||where pk ds=22 dt=NUMERIC }$

```

This example use the full capabilities of variables. It is in fact generated by the **Script to SQL Editor->INSERT COPY INTO TABLE** right click menu choice in the Data tab grid.

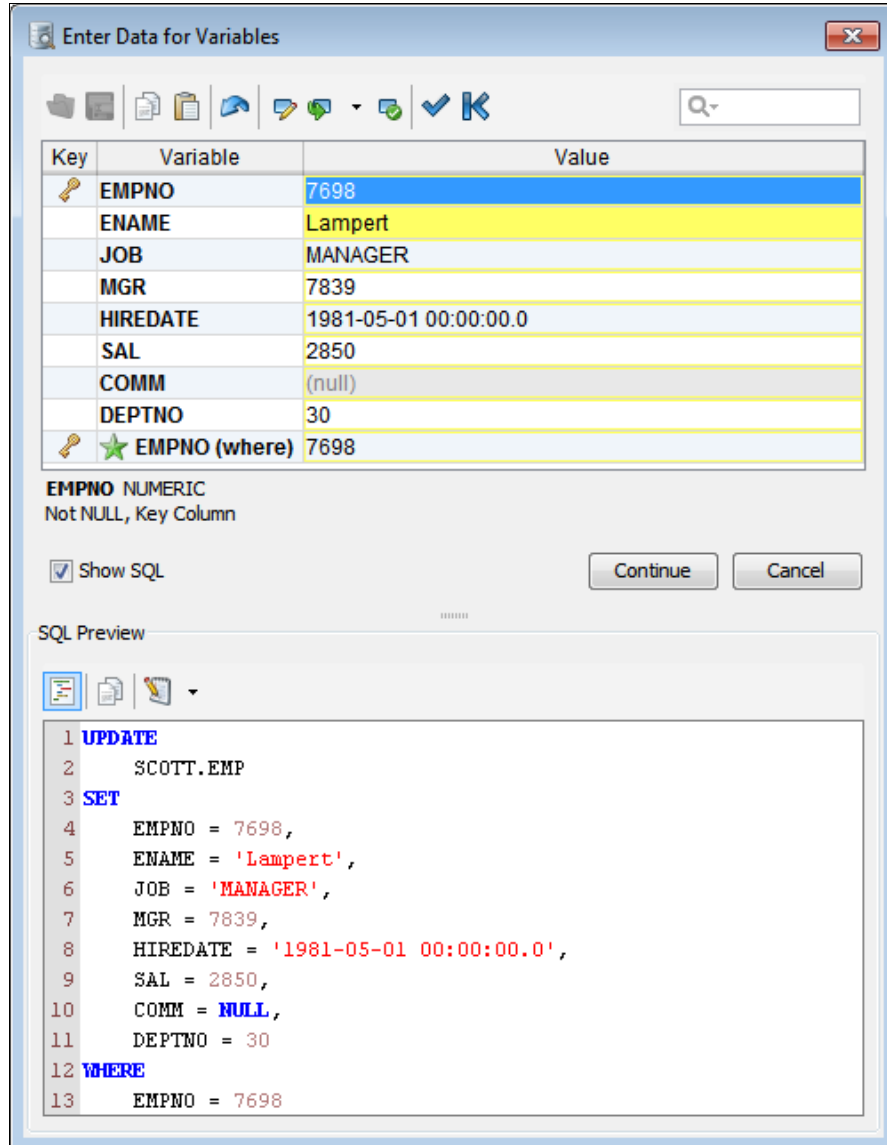


Figure: The substitute variables window

To highlight that a variable is part of the WHERE clause in the final SQL it is represented with a green icon in front of the name.

When executing a statement that consist of variables, DbVisualizer replaces each variable with either the value as inline text or as a parameter marker. Using parameter markers to pass data with a statement is more reliable and safe than inline values. It is also the recommended technique to set values as the database engine may then pre-compile these statements properly. DbVisualizer will automatically generate a parameter marker if the variable have the **type** section set and if there is no **nobind** option specified.

The following will be replaced with a parameter marker:

\${Name||rolle||String}\$

These will be replaced with the variable value:

`${Name||rolle}$`
`${Name||rolle||String||nobind}$`

Variables in DbVisualizer may be used anywhere in a statement. However, there may be problems once the final statement is passed to the database for execution and it contain parameter markers in non supported places. A simple example is Oracle that don't accept parameter markers for a table name. To solve this problem either clear the **type** part of the variable expression or add the option **nobind** (see above).

Parameter Markers

Parameter markers are are usually represented in a SQL statement with a question mark, `?` or a string prefixed with a colon, `:somenam`. Example:

```
select * from EMP where ENAME = ? or ENAME = ?
```

Parameter markers are primarily used in prepared SQL statements that will be cached by the database server. The purpose with cached statements is that the database server will analyze the execution plan once when the SQL is first executed. Subsequent invocations of the same SQL will then only replace the parameter markers with appropriate values, which results in much better response than executing SQLs with dynamic values directly in the SQL.

Parameter marker processing is managed by the JDBC driver and not all drivers supports it. One notable example is that the Oracle JDBC driver lacks support completely.

With a JDBC driver that does support parameter marker processing, the following window appears when executing the previous SQL statement.

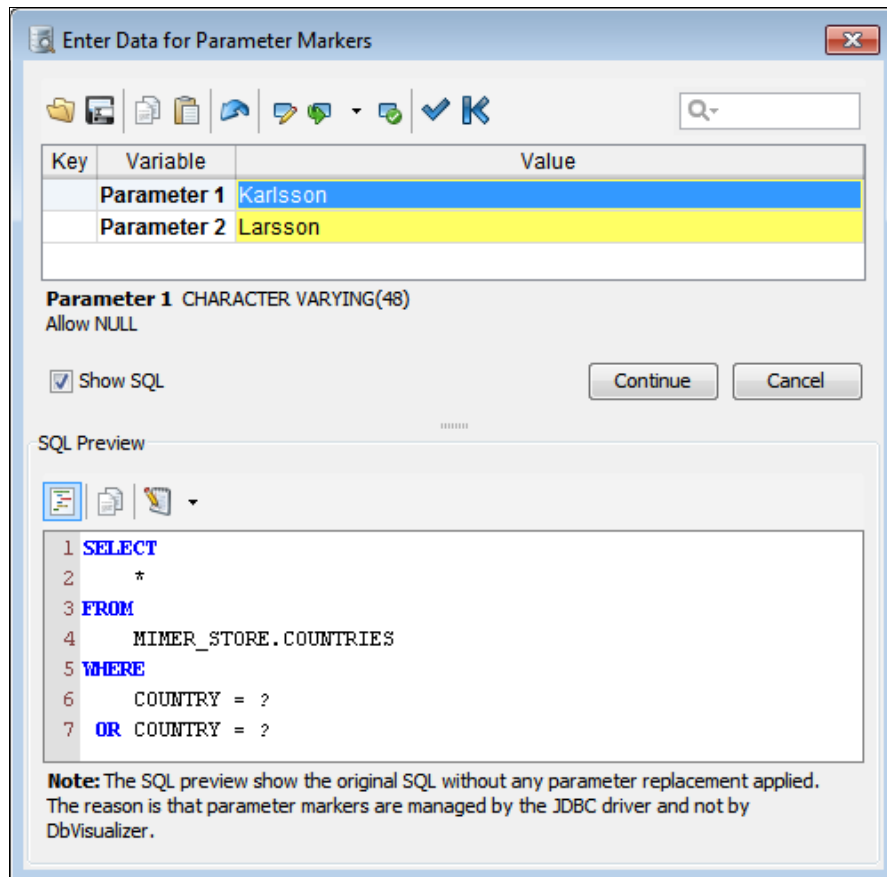


Figure: The parameter marker substitution window

(For parameter marker processing to work in the SQL Commander, make sure the **SQL->Process Parameter Markers in SQL** main menu option is enabled).

Output View

The **Output View** in the lower area of the SQL Commander is used to display the result of the SQLs being executed. How the results are presented is based on what type of result it is. A log entry is always produced in the **Log** view for each SQL statement that is executed. This entry shows at a minimum the execution time and how many rows were affected by the SQL. There may also be a result set if the SQL returned one. These result sets are presented either as tabs or windows based on your choice.

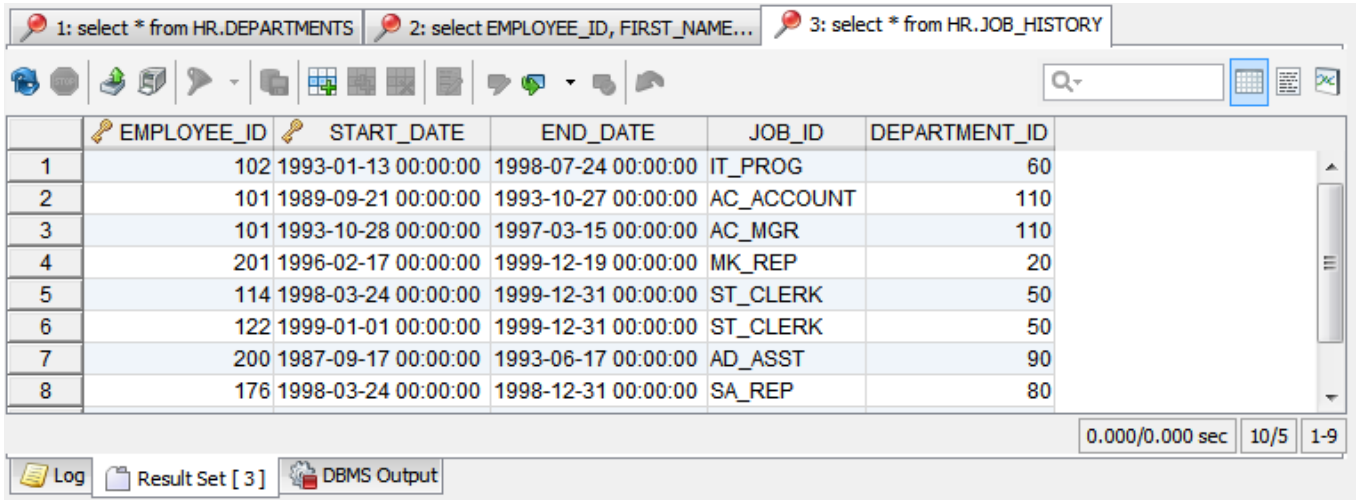


Figure: The output view

If an error occurs during execution, the SQL Commander automatically switches to the Log view so that you can further analyze the problem.

Log

At the top of the Log tab, you can choose to log information about the execution of your SQL statements to the GUI or to a file.



Figure: The Log destination controls

If you choose to log to file, you can enter the file path in the text field or click the button to the right of the field to launch a file browser. By default, the log information is written to the GUI, below the log destination controls.

The log keeps an entry for each SQL statement that has been executed. It provides generic information, such as how many rows were affected and the execution time. The important piece of information is the execution message which shows how the execution of that specific statement ended. If an error occurred, the complete log entry will be in red, indicating that something went wrong.



Figure: The Log with one failed statement

The detail level in an error message depends on the driver and database that is being used. Some databases are very good at telling what went wrong and why, while others provide less detail.

Clicking the icon to the left of each log entry selects the corresponding SQL statement in the SQL editor. The icon also has a right-click menu with two choices: **Load SQL into Editor** and **Insert SQL into Editor**. The first choice replaces the current content of the SQL Editor with the SQL statement corresponding to the log entry, while the second inserts it at the caret position in the SQL Editor.

Log controls

The Editor Control area contains a Log tab where you can control the log content. Use the **Show** controls to define which information you want to appear in the log. The **Filter** controls are used to specify which entries should be displayed.

Auto clear log

If you enable the **Auto Clear Log** control, the SQL Commander automatically clears the log between executions.

Result Set

A result set grid is created for every SQL that returns one or more result sets. These grids can be displayed in a tab or window style view, similar to how the SQL editors are displayed. Each grid shares the common layout and features as described in the [Getting Started and General Overview](#) document. The format of the result can be one:

- **Grid**
The result is presented in a grid.
- **Text**
The result is presented in a tabular format.
- **Chart**
Read more in [Monitor and Charts](#).

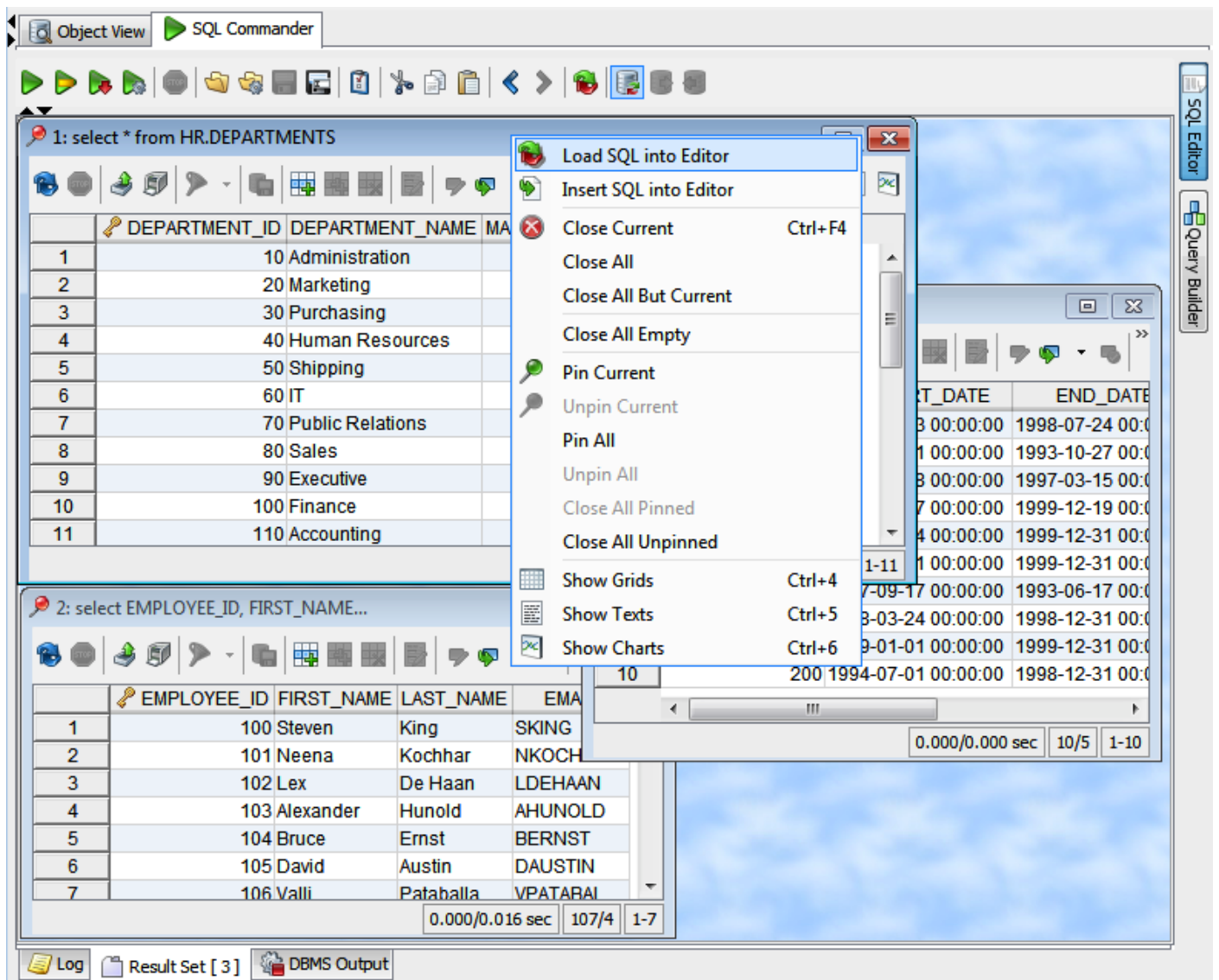


Figure: The windows output view

The figure above shows the windows output style with three result set grids. A result set grid can be closed using the red cross in the window frame header.

With the tabs style, you use the **Close** right click menu choice when the mouse pointer is in the tab header to close a result set:

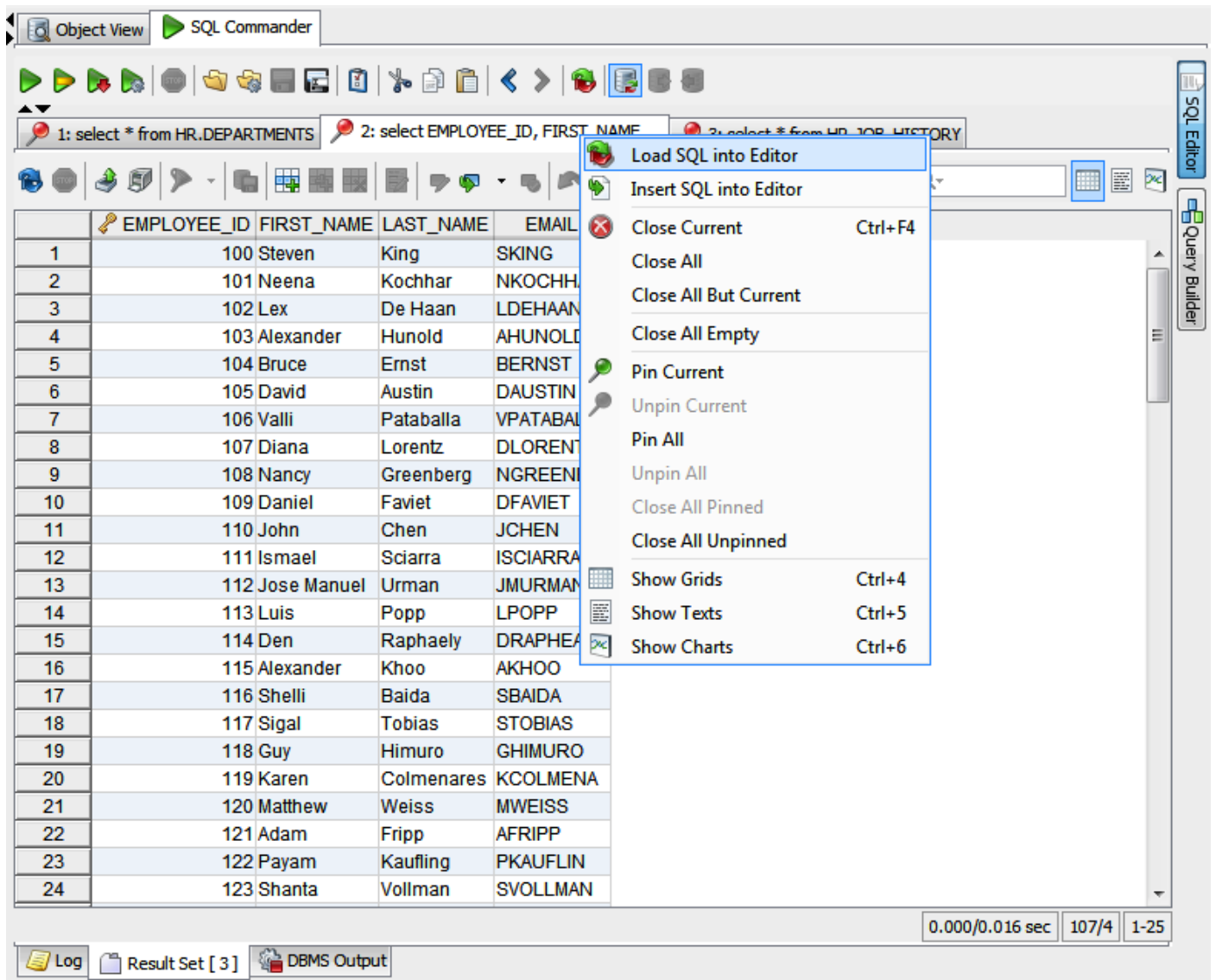


Figure: The right click menu for tabs

Result set menu

The result set menu is available by right-clicking on a tab or on the result set desktop (window style). It contains options to control the current result set and all result sets. The following actions are available:

Menu Choice	Description
Load SQL into Editor	Loads the SQL for the selected result set tab or window into the current editor.
Insert SQL into Editor	Inserts the SQL for the selected result set tab or window into the current editor at the cursor position.
Close Current	Closes the current result set
Close All	Closes all result sets
Close All But Current	Closes all but the current result set
Close All Empty	Closes all result sets that are empty (no data)

Pin Current	Pins the current result set (preventing it from being removed at the next execution).
Unpin Current	Unpins the current result set
Pin All	Pins all result sets. Pinning a result set prevents it from being removed at the next execution.
Unpin All	Unpins any pinned result sets, making them candidates for removal during the next execution.
Close All Pinned	Removes all pinned result sets directly.
Close All Unpinned	Removes all unpinned result sets directly.
Show Grids	Changes the display mode to show the grid tab for all result sets
Show Texts	Changes the display mode to show the text tab for all result sets
Show Charts	Changes the display mode to show the chart tab for all result sets

Editing

A result set grid may be enabled for editing based on the following criteria:

1. The result really is a result set
2. The SQL is a SELECT command
3. Only one table is referenced in the FROM clause
4. All columns in the result set exist in the table with exactly the same names

For some databases, you must also either qualify the table name with the schema name, or make sure that the table belongs to the schema selected in the Schema list for the SQL Editor. If all of the above is true, the standard editing tool bar appears just above the grid. Read more about editing in the [Edit Table Data](#) chapter.

Multiple result sets produced by a single SQL statement

Some SQL statements may produce multiple result sets. Examples of this are stored procedures in Sybase ASE and SQL Server. The SQL Commander checks the results as returned by the JDBC driver and add grids to the output view accordingly. The following shows the **sp_help** command which returns several result sets with various information about the **newTable** table.

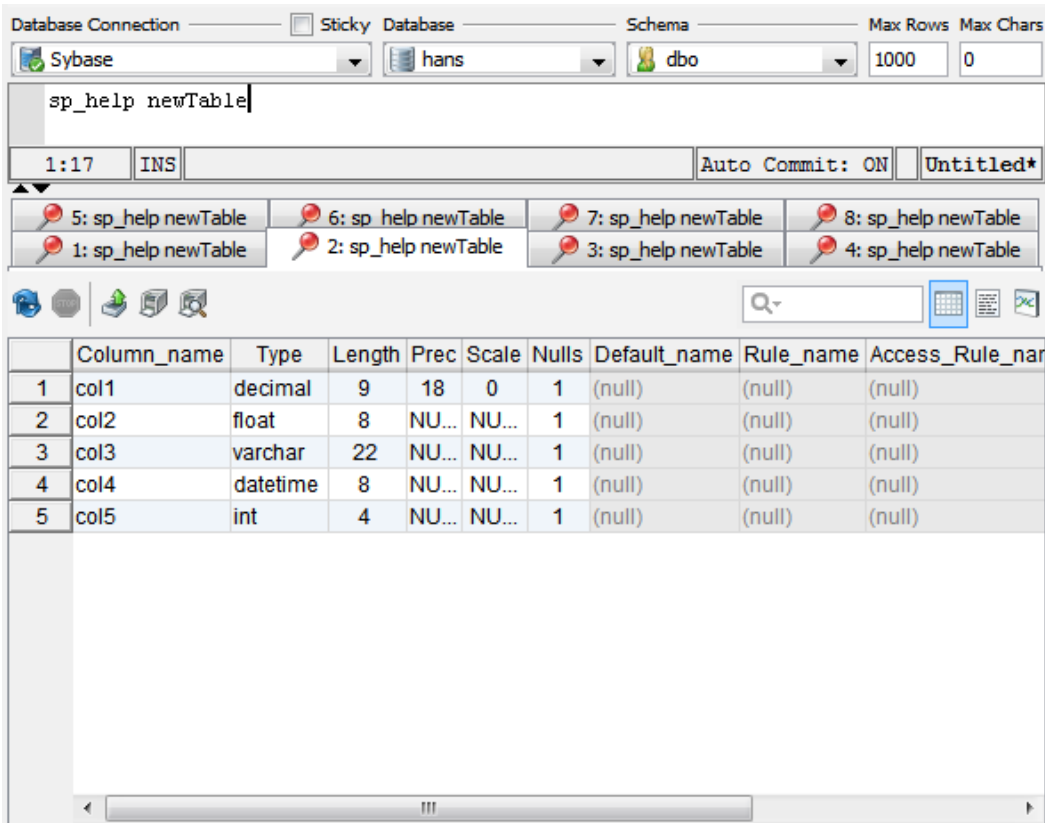


Figure: Multiple result set grids produced by a single SQL statement

The result set grids in the figure above all share the same label, `sp_help newTable`. The number after the label represents the order number for the actual result. A stored procedure can return different results, not all being result sets. The number helps you identify which entry matches which result set grid in the log. Here is the Log output view for the previous example.

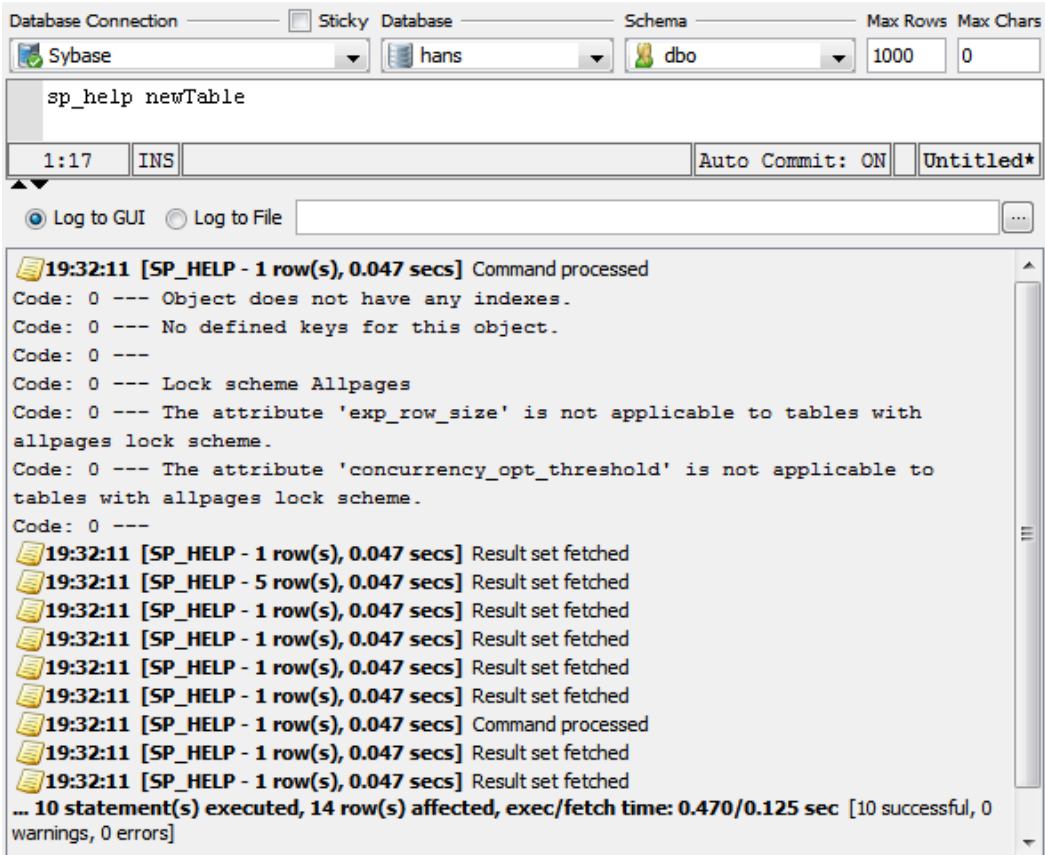


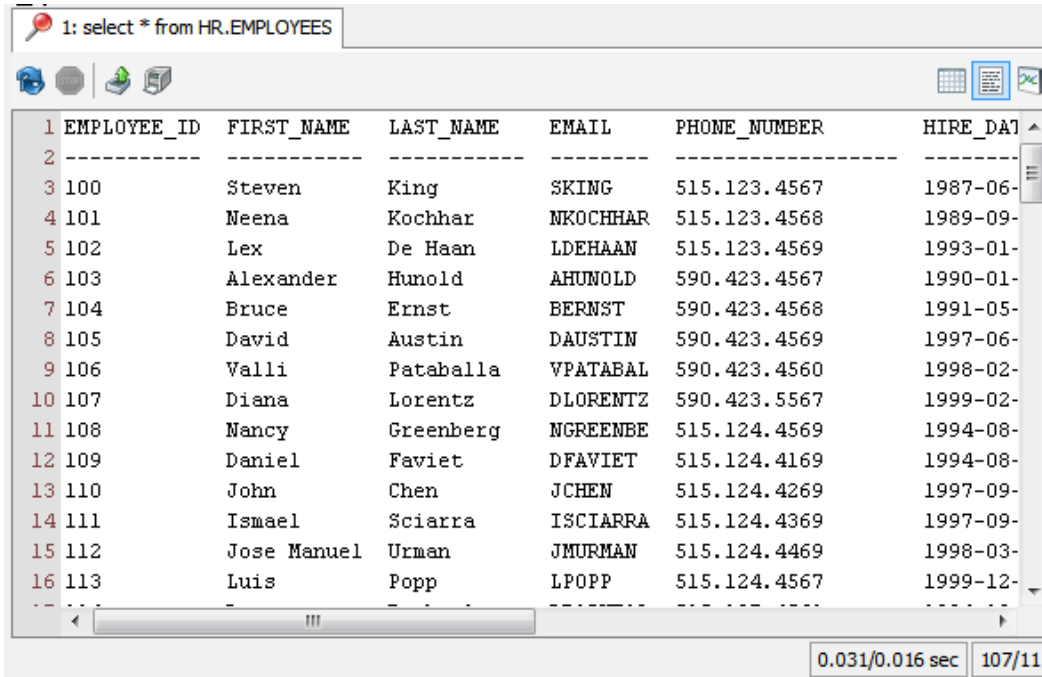
Figure: The Log after executing an SQL statement that returns multiple results

All entries with the log message "Result set fetched" are represented by a grid in the previous figure.

Text

The **Text** format for a result set presents the data in a tabular style. The column widths are calculated based on the length of each value and the length of the column label.

The column widths may vary between executions of the SQL.



The screenshot shows a SQL query window with the following text-based result set:

1	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
2	-----	-----	-----	-----	-----	-----
3	100	Steven	King	SKING	515.123.4567	1987-06-
4	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-
5	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-
6	103	Alexander	Humold	AHUNOLD	590.423.4567	1990-01-
7	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-
8	105	David	Austin	DAUSTIN	590.423.4569	1997-06-
9	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-
10	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02-
11	108	Nancy	Greenberg	NGREENBE	515.124.4569	1994-08-
12	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08-
13	110	John	Chen	JCHEN	515.124.4269	1997-09-
14	111	Ismael	Sciarra	ISCIARRA	515.124.4369	1997-09-
15	112	Jose Manuel	Urman	JMURMAN	515.124.4469	1998-03-
16	113	Luis	Popp	LPOPP	515.124.4567	1999-12-

At the bottom right of the window, the execution statistics are shown: 0.031/0.016 sec and 107/11.

Figure: The Text result set format

Chart

A result set can be charted using the **Chart** view in a grid. Please read more about it in the [Monitor and Charts](#) document.

DBMS Output (Oracle)

The **DBMS Output** tab for Oracle is used to enable and disable capturing of messages produced by stored procedures, packages, and triggers. These messages are typically inserted in the code for debugging purposes. For SQL*Plus users, the corresponding feature is enabled via the `set serveroutput on` command. To enable display of DBMS messages in DbVisualizer, select the DBMS Output tab and press the Enable button.

Once DBMS output is enabled, the icon in the tab header is changed. Invoking a stored procedure in the SQL editor will result in the following being displayed in the output tab. (Each block of output is separated with a timestamp).

Database Connection: Sticky Database: Schema: HR Max Rows: 1000 Max Chars: 0

Oracle

```
call emp_report()
```

1:1 INS Auto Commit: ON Untitled*

Buffer Size: 100000

```
1 --- 19:47:35 ---
2 Empno  Ename      Job
3 -----
4 100    King       AD_PRES
5 101    Kochhar    AD_VP
6 102    De Haan    AD_VP
7 103    Hunold     IT_PROG
8 104    Ernst      IT_PROG
9 105    Austin     IT_PROG
10 106    Pataballa IT_PROG
11 107    Lorentz    IT_PROG
12 108    Greenberg FI_MGR
13 109    Favier     FI_ACCOUNT
14 110    Chen       FI_ACCOUNT
15 111    Sciarra    FI_ACCOUNT
16 112    Urman      FI_ACCOUNT
17 113    Popp       FI_ACCOUNT
18 114    Raphaely   PU_MAN
19 115    Khoo       PU_CLERK
```

Log Result Set DBMS Output

Figure: DBMS Output tab

Query Builder

Introduction

The **Query Builder** provides an easy way to develop database queries. The Query Builder provides a point and click interface and does not require in-depth knowledge about the SQL syntax.

The Query Builder is part of the SQL Commander, alongside the SQL Editor. To open the Query Builder, make sure the SQL Commander tab is selected and then choose either the **SQL->Show Query Builder** menu choice or click the vertical **Query Builder** button to the right in the SQL Commander. When you are ready to test a query built with the Query Builder, you just load it to the SQL Editor for execution.

This document talks only about **Tables** even though the Query Builder supports both table and view objects.

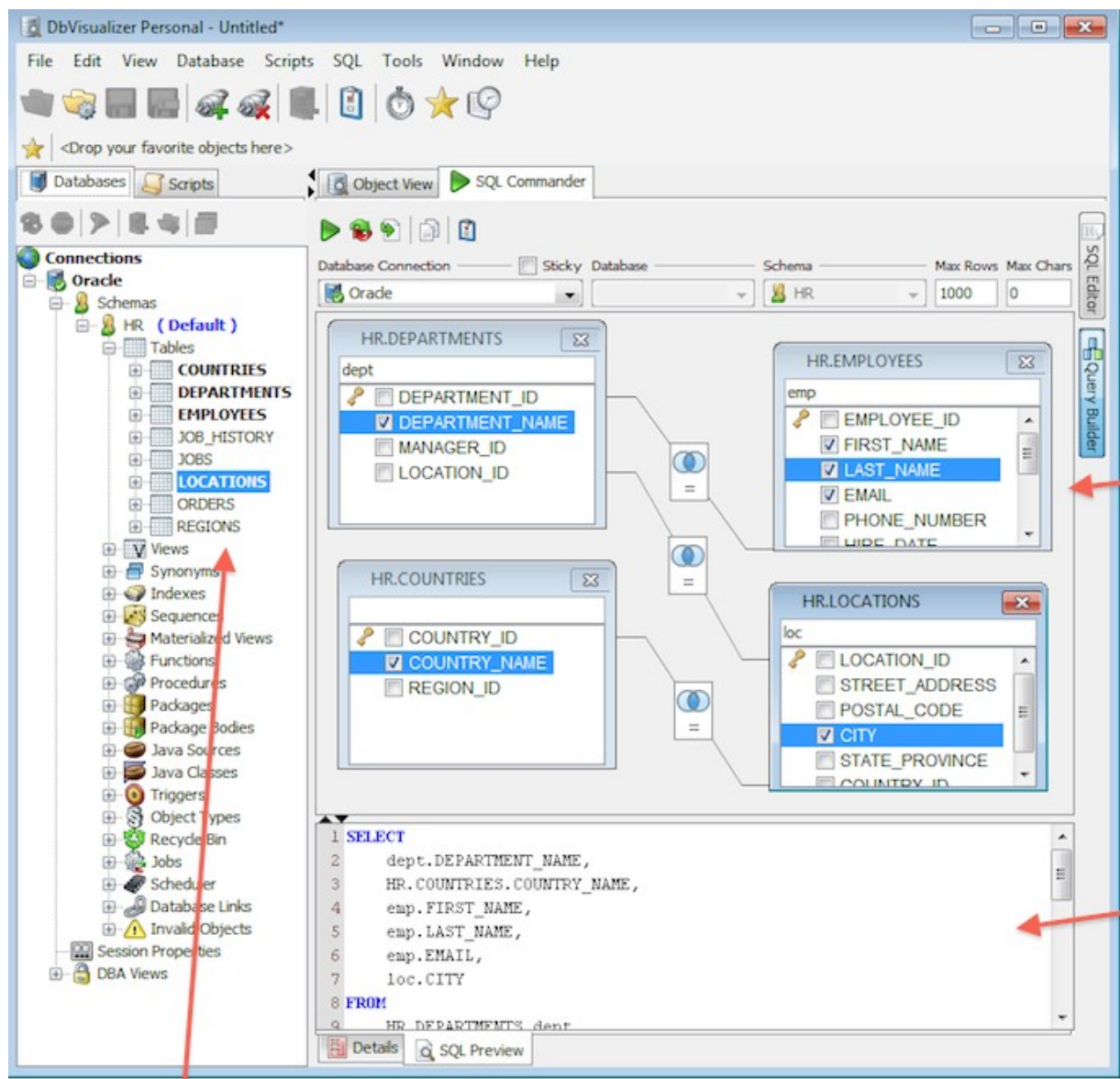


Diagram a

Query det

Drag tables/views from objects tree

Figure: The query builder

Current Limitations

These are the current limitations in the Query Builder:

- Unions and sub queries are not supported.
- Not all join types are supported when joins are expressed as WHERE clause conditions. The **Inner** join type is supported for all databases, but the **Left** and **Right** types are only supported for databases with proprietary syntax to express these types, e.g., Oracle, SQL Server and Sybase. The **Full** type is not supported for any database. If a join type is not supported, the setting in the Join Properties dialog is silently ignored.
- When importing an SQL query from the SQL Editor, unsupported keywords and statement clauses are ignored. A dialog tells you which parts of the query are being ignored when unsupported parts are found in the imported statement.

Creating a Query

To create a query, open the query builder using the **SQL->Show Query Builder** menu choice or click the Query Builder button in the SQL Commander as described earlier. Make sure that the controls in the top section of the Query Builder are set correctly, as described in [Database Connection, Catalog and Schema](#).

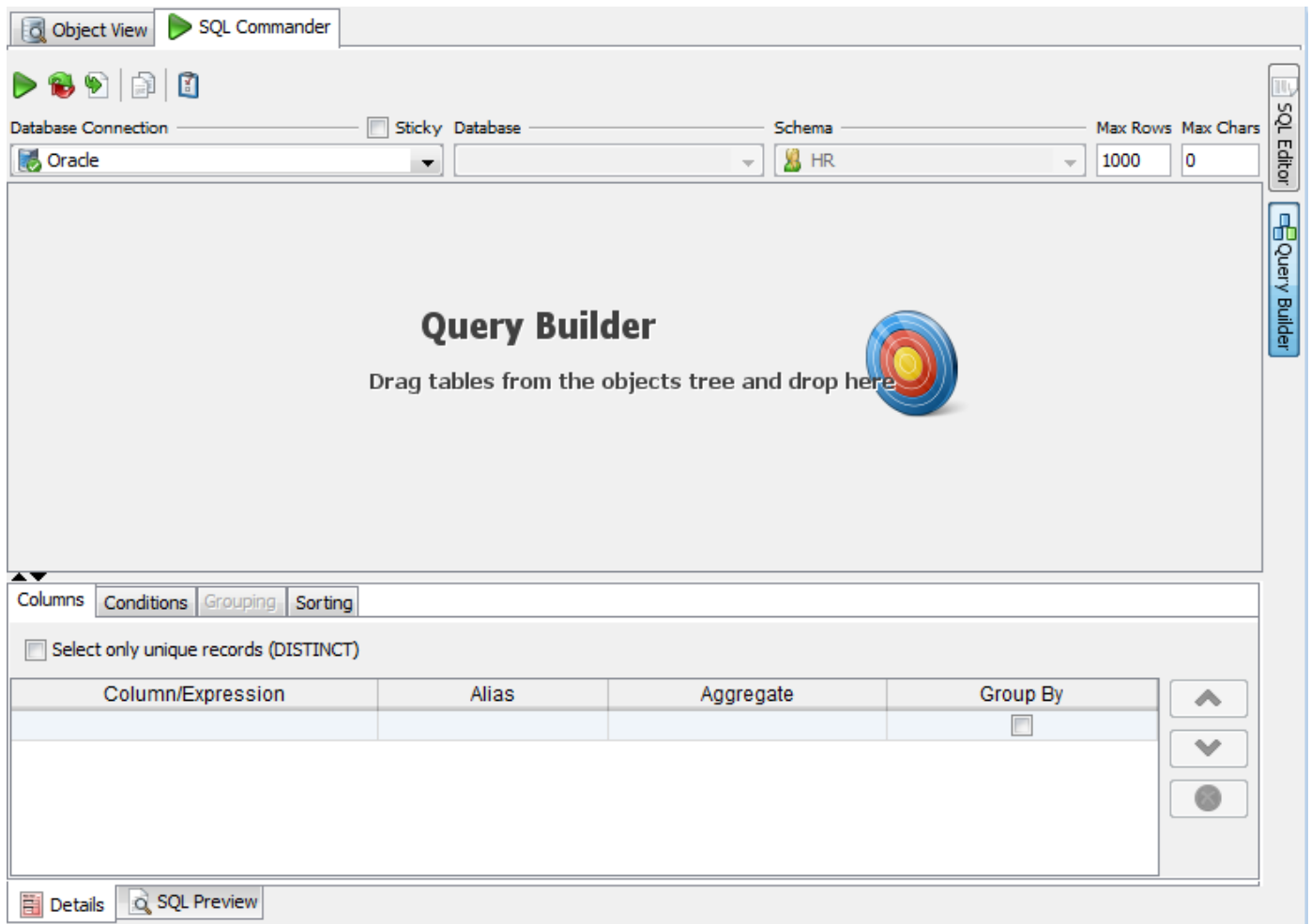


Figure: The initial appearance of the query builder

The easiest way to jump between the Query Builder and the SQL Editor is by clicking the vertical control buttons to the right in the SQL Commander. Clicking these buttons changes the display, but does not copy the query from one display to the other. To copy the current query from the Query Builder to the SQL Editor, use the toolbar buttons at the top of the Query Builder:



Figure: Query builder toolbar

1. The first button (from left) replaces the content of the SQL Editor with the query SQL and executes it
2. The second button replaces the content of the SQL Editor with the query SQL, without executing it
3. The third button adds the query last in the SQL Editor
4. The fourth button copies the query to the system clipboard
5. The fifth button opens the [editor properties](#)

The first three buttons automatically change the display to the SQL Editor.

You can also load a query from the SQL Editor into the Query Builder, as described in detail [below](#).

Adding Tables

To add tables, make sure the database objects tree and the table and/or view objects are visible. Then select and drag nodes from the tree into the diagram area.

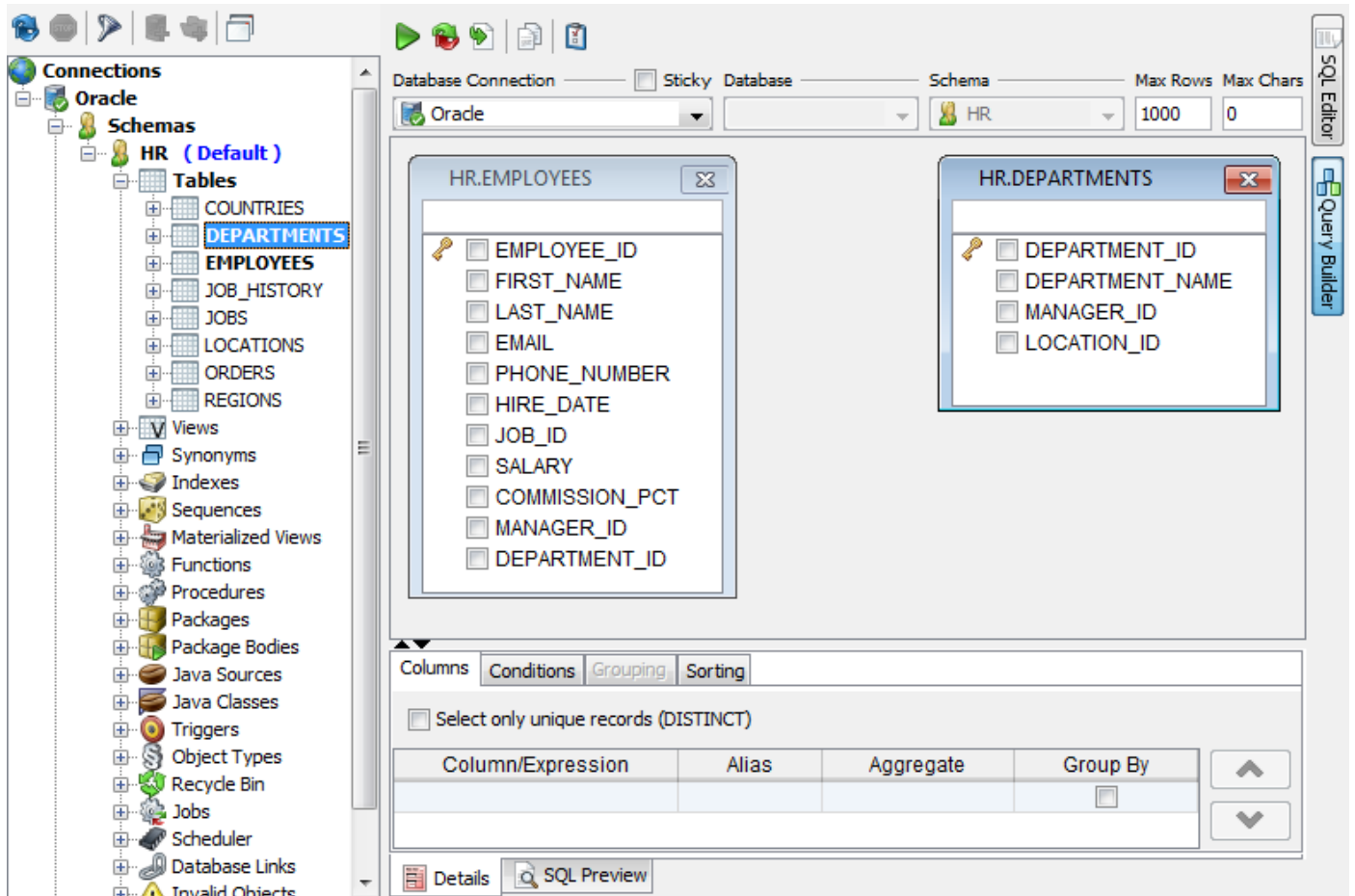


Figure: Adding tables to the query builder

To add a table, drag it from the object tree to the diagram area of the Query Builder. When the table is dropped in the diagram area, it is shown as a window with the table name as the window title.

Below the title is a text field where an optional table alias can be entered. If a table alias is specified, it is used in the Query Builder and the generated SQL statement to refer to this table.

Under the table alias field is a list of all table columns. A check box in front of each name is used to select whether the column should be included in the query result set. Columns selected for the query result set also appear in the **Columns** and **Sorting** details tabs.

Joining Tables

Manually Joining Tables

To join two tables, select the column in the source table window with the mouse, drag it to the target table column, and drop it.

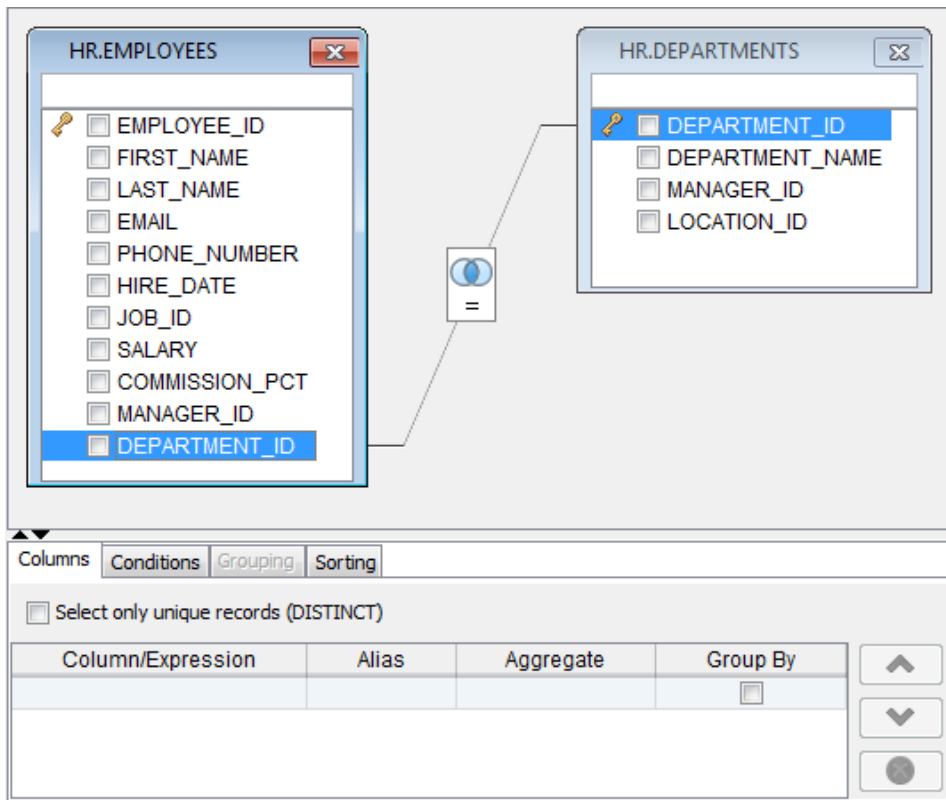


Figure: Joining two tables

The two columns now represent a join condition, shown in the graph as a link between the columns. If more than one join condition is needed, link additional columns in the two tables by dragging and dropping the columns in the same way as for the first join condition. The default join type is an Inner join and the default condition is "equal to" (=), represented as an icon with overlapping circles with the shared area shaded and an equal sign below them.

Joining Tables Automatically

Some database schemas declare how tables are related using primary and foreign keys. Other schemas use column names to indicate these relationships. For instance, in the figure above, the EMPLOYEES table has a column named DEPARTMENT_ID, which refers to the column with the same name in the DEPARTMENTS table. The Query Builder can be configured to use both kinds of rules to automatically join the tables you add to the query builder.

The auto-join feature is disabled by default. You can enable it in the tool properties for the database type (**Tools->Tool Properties**, under the **Database** tab) or for a specific connection (the **Properties** tab at the bottom of the Object View window when the connection is selected in the object tree).

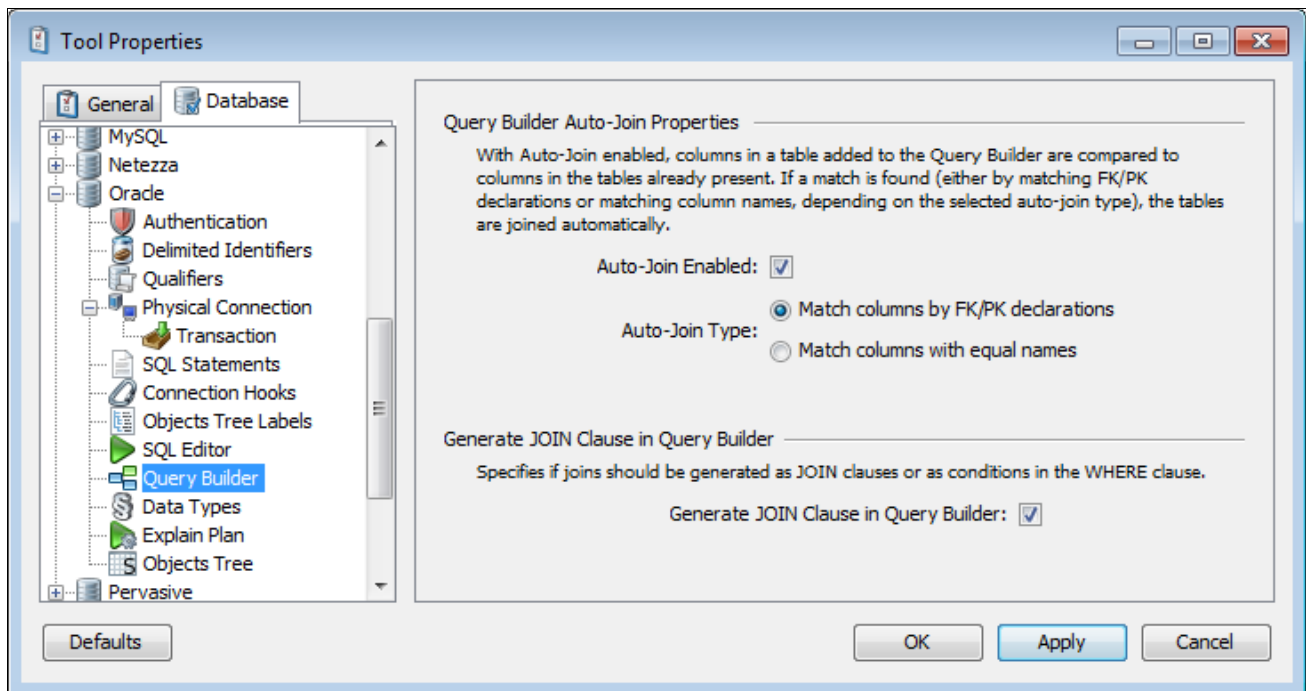


Figure: Query Builder Properties

The Query Builder node lets you enable the auto-join feature and select whether to use key declarations (FK/PK) or column names to find out how the tables are related.

When you add a new table with auto-join enabled, the Query Builder automatically joins it to the tables already in the builder if table columns match the selected matching rule.

If columns in the table you add are related to other columns in the same table, the Query Builder creates two windows for the table and joins them based on the matching rule. In this case, a table alias is also added for one of the windows so that you can tell the two windows for the same table apart.

Join Properties

A Join Properties dialog can be opened by double-clicking the icon or selecting Join Properties from the right-click menu while the mouse pointer is over the join icon. The Join Properties dialog shows the source and target table columns and the conditional operator.

You can change the join type and the conditional operator in the Join Properties dialog. The join type defines how the records from the tables should be combined:

- **Inner**
This is the most common join type as it finds the results in the intersection between the tables.
- **Left**
This join type limits the results to those in the left table leaving 0 matching records in the right table as NULL.
- **Right**
This is the same as left join but reversed
- **Full**
A full join combines the results of both left and right joins.

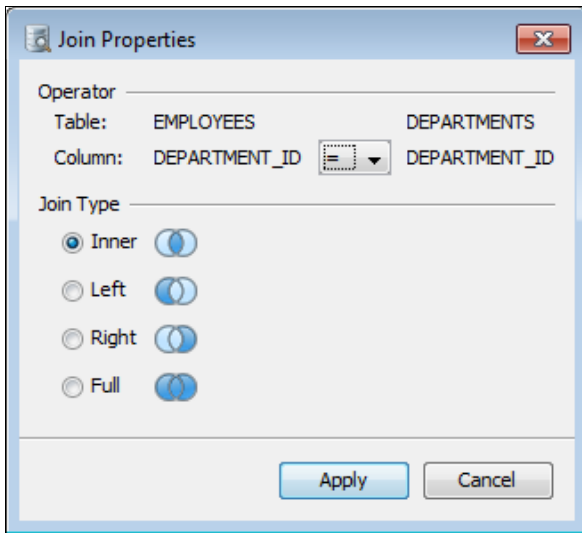


Figure: Join Properties dialog

If you have multiple join conditions (linked columns) between two tables, you can specify different conditional operators for each join condition, but the join type is shared between all join conditions; if you change it for one join condition, it is changed for all the other join conditions linking the two tables. This is not a restriction in the Query Builder but rather how SQL is defined.

Here is the sample SQL generated from the previous join definition:

```
SELECT
  *
FROM
  HR.EMPLOYEES
INNER JOIN
  HR.DEPARTMENTS
ON
  (HR.EMPLOYEES.DEPARTMENT_ID = HR.DEPARTMENTS.DEPARTMENT_ID)
```

Remove Tables and Joins

A table window is removed by clicking the close icon in the window header. A join is removed by selecting **Remove Join** in the right-click menu while the mouse pointer is over the join icon.

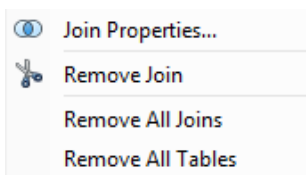


Figure: Diagram right click menu

All tables and joins may be removed via **Remove All Joins** and **Remove All Tables**.

Query Details

The Details tabs below the diagram area are used to define the various parts of the query. The tabs basically represent the following parts of the final SQL:

```
SELECT <Columns>
FROM <Tables>
WHERE <Conditions>
```

```

GROUP BY <Columns>
HAVING <Grouping>
ORDER BY <Sorting>

```

(The **Tables** clause is defined in the diagram, not by a tab).

Columns

Use the Columns tab to specify characteristics of the columns that are included in the query. The list is initially empty until a column is checked in a table window or a column expression is added manually (see below). Columns will appear in the list in the same order as they are checked but may be moved at any time with the up and down buttons. To include all columns from a table, right-click in the column list in the table window and choose **Select All**.

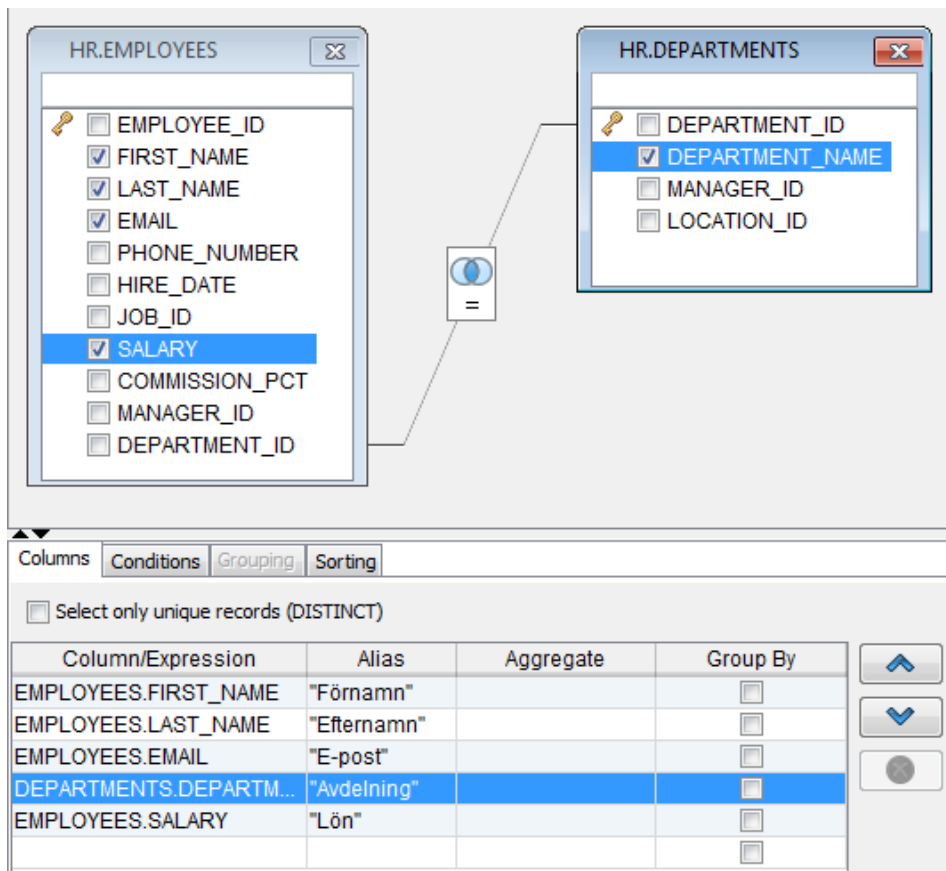


Figure: The Columns tab

The previous screenshot shows a total of 5 checked columns in the two tables. These are presented in the columns list by their full column identifier, qualified by either the table name or the table alias. To remove a column from the list, uncheck the corresponding column in the table window.

The **alias** field is used to specify an optional alias identifier for the column. The alias is used as the identifier for the column in the final query and also appears as the column name in the result set produced by the query. Check the documentation for the actual database to see if the alias must be quoted since the Query Builder does not do this for you.

The **Aggregate** and **Group by** fields are used in combination:

- The **Aggregate** field lists the available aggregation functions (AVG, COUNT, MAX, MIN, SUM) that may be used for columns
- The **Group By** field specifies whether the column should be included in the group for which aggregate columns are summarized

The Group By field is disabled unless an aggregate function is selected for at least one column, and once you select an aggregate function for one column, you must set Group By for at least one of the other columns to form a valid query. If you remove the aggregate function for all columns, Group By is automatically reset for all columns. Group By and aggregate are also mutually exclusive options for one column, so when you select one of them, the field for the other is disabled for that column.

A custom expression may be added by entering data in the empty row last in the list, e.g., **"col1 + col2"** or **"TO_CHAR(ts_col, 'DD-MON-YYYY HH24:MI:SSxFF')"**. Once entered, press enter to insert a new empty row. You can remove a custom expression by selecting it and clicking the

Remove button.

Conditions

The Conditions tab is used to manage the **WHERE** clause for the query. A WHERE clause may consist of several conditions connected by AND or OR. The evaluation order for each condition is defined by indentation in the condition list. Each level in the list will be enclosed by brackets in the final SQL.

Here is an example from the Conditions tab.

The screenshot shows the 'Conditions' tab in a query builder. It displays two tables: HR.EMPLOYEES and HR.DEPARTMENTS. The HR.EMPLOYEES table has columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, and DEPARTMENT_ID. The HR.DEPARTMENTS table has columns: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. A condition is set for emp.SALARY > 4000. Below this, a compound condition is set for 'Any' of the conditions in this branch must match, with sub-conditions: dept.DEPARTMENT_NAME = 'IT' and dept.DEPARTMENT_NAME = 'Human Resources'.

Figure: Condition settings

To create a new WHERE condition, press the indexed button in the list. In the menu that is displayed you may choose to create a new condition on the same level, a compound condition or delete the current condition. For compound conditions you may choose whether **All** (AND), **Any** (OR), **None** (NOT OR) or **Not All** (NOT AND) conditions must be met for its sub conditions. The SQL for the Conditions tab in the figure is:

```
WHERE
  emp.SALARY > 4000
AND
  (
    dept.DEPARTMENT_NAME = 'Human Resources'
  OR dept.DEPARTMENT_NAME = 'IT'
  )
```

Next to the input field for each condition, there is a drop down button. When pressed it shows all columns that are available in the tables currently being in the Query Builder. You can pick columns from the list instead of typing these manually.

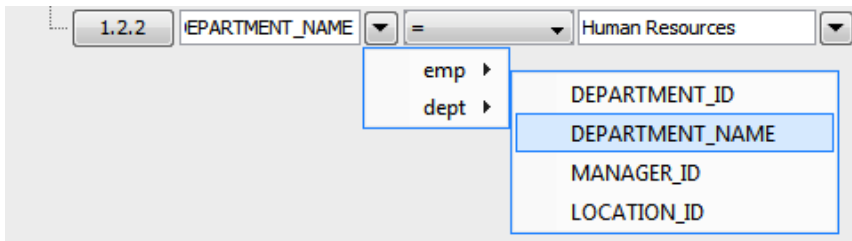


Figure: List of columns in the Conditions tab

Grouping

The Grouping tab is used to define the conditions for the **HAVING** clause that may follow a GROUP BY clause in an SQL query. This tab is only enabled when at least one of the columns in the Columns tab is marked as a Group By column.

The HAVING clause is similar to the WHERE clause, except that the HAVING clause limits what rows are included in the groups defined by the GROUP BY clause, after the WHERE clause has been used to limit the total number of rows to process.

You work with conditions in this tab in the same way as described in the [Conditions](#) section, with one exception regarding the drop-down button for the fields in a condition. In the Grouping tab, the drop-down shows all columns listed in the Columns tab, with an aggregate function expression for columns that have an aggregate function defined. This is because (according to the SQL specification) the conditions in a HAVING clause must only refer to columns that are being returned by the query.

Sorting

The sorting tab is used to specify how the final result set will be sorted. All columns for the tables in the graph, plus any custom expressions created for the selection list in the Columns tab, are listed in the Sorting tab.

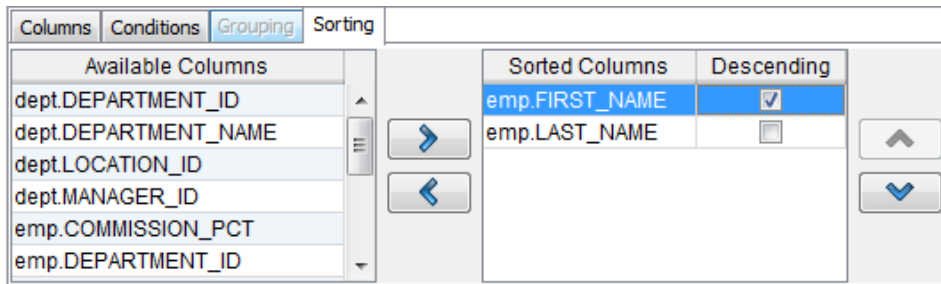


Figure: The Sorting tab

All columns listed in the Columns tab are initially listed in the **Available Columns** table. Select the ones you want to use in the sorting criteria and click the **Move Right** button to move them to the **Sorted Columns** table.

In the Sorted Columns table, you can change the default sort order (ascending) by clicking the check box in the **Descending Order** column. You can remove columns from the sorting criteria by selecting them in the Sorted Columns table and clicking the Move Left button.

SQL Preview

The SQL Preview tab at the bottom of the query builder is used to show a preview of the final SQL. This is a read-only view and cannot be modified.

Testing the Query

To test the query, simply press the appropriate toolbar buttons in the Query Builder to copy the SQL to the SQL Editor. Then execute the SQL as usual in the SQL Editor.

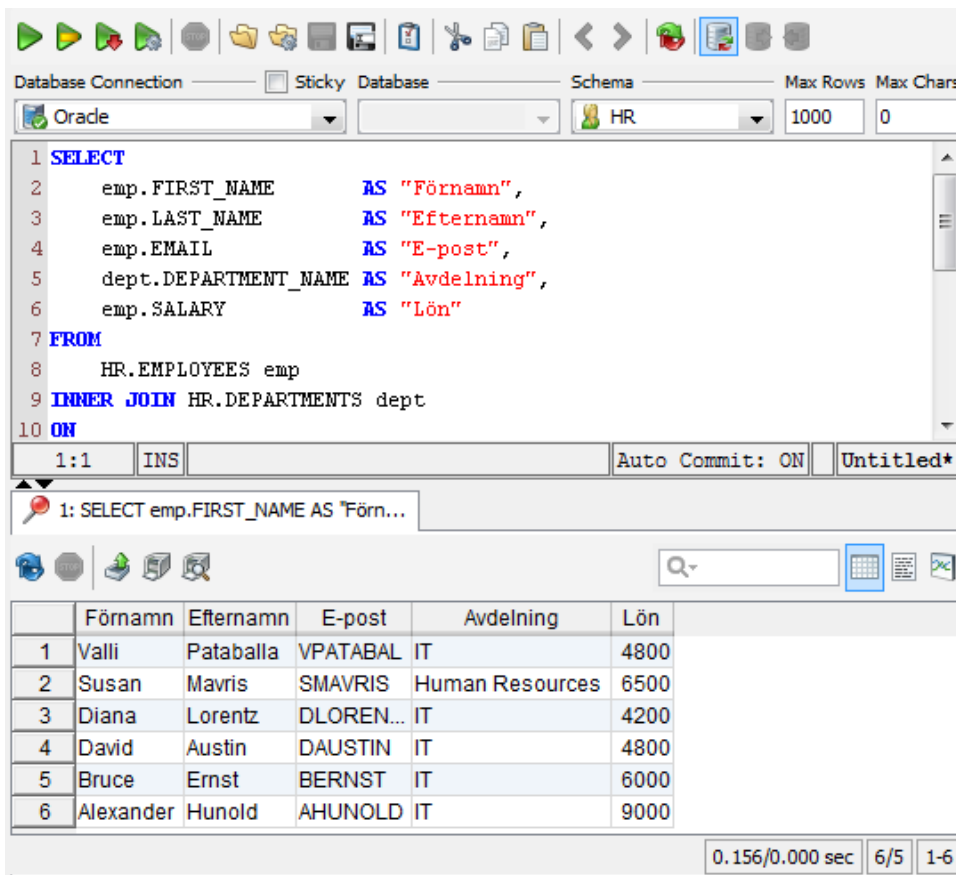



Figure: Testing the SQL

To further refine the SQL press the Query Builder button and make the necessary changes.

Loading a Query from the SQL Editor

If you have an existing SQL query that you want to modify using the Query Builder, you can load it from the SQL Editor into the Query Builder by clicking the rightmost button in the SQL Editor toolbar: 

It's important to be aware that the Query Builder does not support all features of the SQL SELECT statement, such as comments, UNION, and database-specific keywords. If you load a query into the Query Builder that contains unsupported constructs or keywords, they are ignored and a dialog pops up with a warning about this fact. You can then use the SQL Preview tab in the Query Builder to compare the SQL as it is represented in the Query Builder with the original SQL that you loaded to understand what was ignored.

Properties controlling Query Builder

There are a few properties that control how the Query Builder works and the SQL it generates. You can set these properties for the database type (**Tools->Tool Properties**, under the **Database** tab) or for a specific connection (the **Properties** tab at the bottom of the Object View window when the connection is selected in the object tree). Check the following sections for details.

Express joins as JOIN clause or WHERE condition

The **Generate JOIN Clause in SQL Builder** property is available in the **[Database Type]->Query Builder** node. Joins can be expressed either via the standardized SQL JOIN clause or a WHERE clause, using database-specific syntax for the **Left** and **Right** join types. The database-specific WHERE clause syntax is somewhat different between the supported databases and the **Full** outer join type is not supported. The default for this property is to use a JOIN clause.

A simple inner join expressed as a JOIN clause:

```
FROM HR.EMPLOYEES
INNER JOIN HR.DEPARTMENTS
ON (HR.EMPLOYEES.DEPARTMENT_ID = HR.DEPARTMENTS.DEPARTMENT_ID
```

Here is the same join expressed as a WHERE condition:

```
FROM HR.EMPLOYEES, HR.DEPARTMENTS
WHERE HR.EMPLOYEES.DEPARTMENT_ID = HR.DEPARTMENTS.DEPARTMENT_ID
```

The syntax for expressing Inner and Outer joins in WHERE conditions is different between databases. Oracle, for example, uses the "(+)" sequence to the left or right of the conditional operator to express left or right joins. SQL Server and Sybase use "*"=" or "=" for the same purpose.

DbVisualizer automatically uses the correct join notation when generating joins as WHERE conditions for databases that support left and right joins using WHERE conditions. For databases that do not provide syntax for left and right joins, the join type is ignored and the WHERE condition that is generated produces an inner join result.

Table and Column Name qualifiers

Whether to qualify table names with the schema or database name and whether to qualify column names with the table name are defined in the **[Database Type]->Qualifiers** node.

Delimited Identifiers

Identifiers that contain mixed case characters or include special characters need to be delimited. Define this in the **[Database Type]->Delimited Identifiers** node.

Drag style and Diagram Size

If you enable the editor controls from the Query Builder or SQL Editor toolbar, you can also set the style table windows in the Query Builder diagram should have when moving them, as well as the default size for newly added table windows.

Bookmarks and History

Introduction

When you work with a database, there are often some set of SQL statements that you use over and over to perform frequent tasks. You probably have them saved in script files that you can load into an SQL Editor, but DbVisualizer Bookmarks make it even easier to work with them. A Bookmark is a script visible in the Scripts tab in the tree area of the GUI.

Other times you type SQL statements directly in an SQL Editor and execute them. Later you may realize that you need to execute a statement again. You can then use the History window to locate the statement and reuse it.

In this chapter, you will learn all about how to use Bookmarks and the History.

Bookmarks

You find your Bookmarks under the **Scripts** tab in the tree area to the left in the main DbVisualizer window. The content of a Bookmark is one or more SQL statements. It may also be associated with a Connection, a Catalog and a Schema, to be used when executing the statements. This information is displayed, and can be edited, in the lower part of the Scripts tab, along with information about the file that holds the Bookmark. If you don't want to see these details, you can disable it with the **Show Details** toggle control in the right-click menu for a node.

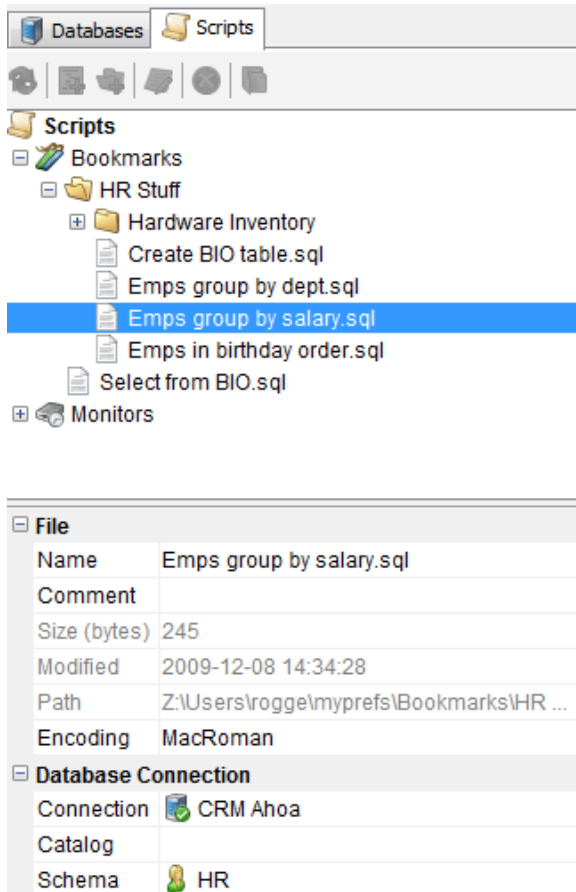


Figure: The Scripts tab with Bookmarks

Creating, Editing and Organizing Bookmarks

You can create a new Bookmark by selecting the **Bookmarks** node in the tree and clicking the **Create File** toolbar button. This adds a new node in the tree, with the default name selected so that you can replace it with the name you want to use. You can also rename the Bookmark later using the **Rename** toolbar button with the node selected.

A Bookmark can also be created from the current content in an SQL Editor. Click the **Save File As** toolbar button to open a file chooser dialog, and click the Bookmarks button in the file chooser dialog to go to the Bookmarks root directory. Enter a filename for the Bookmark and click **Save**.

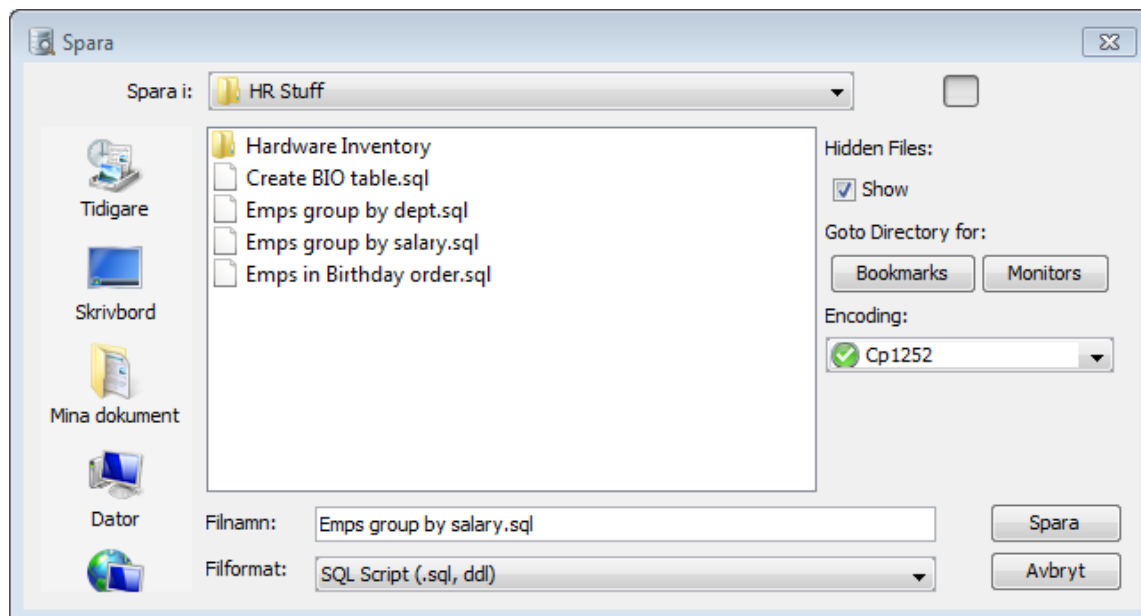


Figure: The File Chooser dialog

To put some SQL statements in a new empty Bookmark or to edit the contents of an existing Bookmark, you need to open the Bookmark in an SQL Editor. If you want to use the SQL Editor you currently have open, you can double-click the Bookmark node in the tree or click the corresponding toolbar button. If you want to edit the Bookmark in a new editor, use the right-click **Open File -> Open in New Editor** operation or press Alt and then double-click. When you are done with your edits, use the **Save** toolbar button in the SQL Editor to save them.

The default open behaviour when double-click a script file can be configured in **Tool Properties->Script** category.

You can also add the content of a Bookmark to the current content of an SQL Editor. Select the Bookmark node, drag it with the mouse key depressed to the position in the editor where you want to add it, and drop it there by releasing the mouse button.

Folders can be used to organize your Bookmarks. Click the **Create Folder** toolbar button to create a new folder and give it the name you want. You can then drag an existing Bookmark node to the folder, and create new Bookmarks and subfolders in the folder by selecting it and clicking the **Create File** and **Create Folder** buttons.

The order of the folders and the Bookmarks within a folder is determined by the filesystem and cannot be changed manually.

Executing Bookmarks

With a Bookmark opened in the SQL Editor, you can of course execute its statements by clicking the **Execute** toolbar buttons as usual, but you can also open and execute a Bookmark directly by selecting it in the tree and using the **Execute File** operations in the right-click menu.

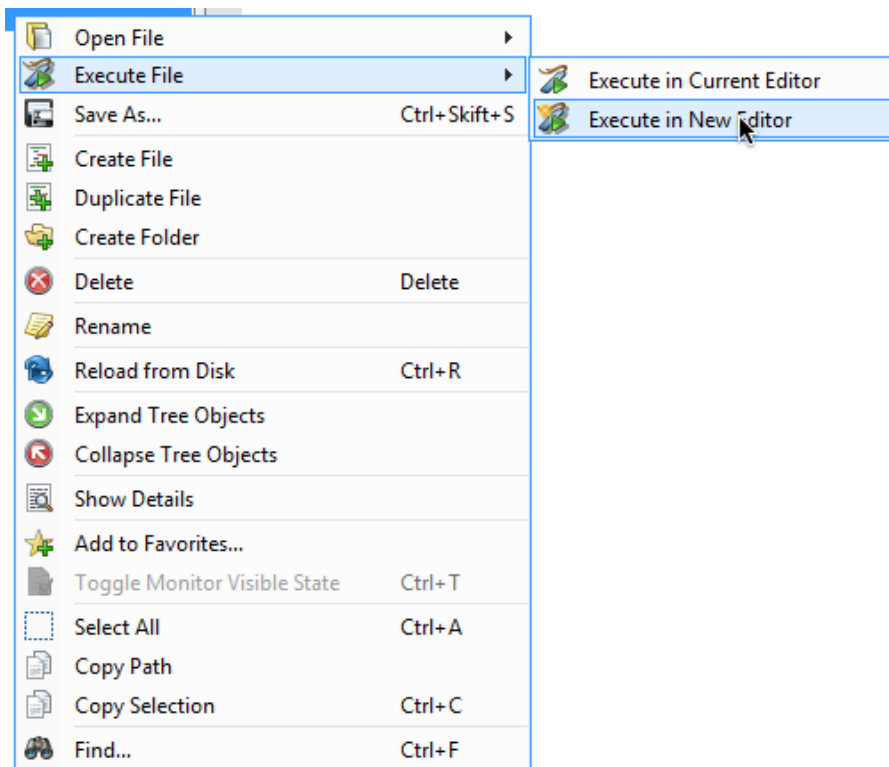


Figure: The Execute File operations in the right-click menu for a Bookmark

Adding a Bookmark as a Favorite

If you are using a Bookmark very often, you may find it more convenient to add it to the [Favorites area](#). You can drag and drop a Bookmark from the Scripts tab to the Favorites area, or launch the Favorites editor via the **Add to Favorites** right-click menu operation.

Sharing Bookmarks

It's easy to share your Bookmarks with someone else because they are stored as regular files. The files are located in a subfolder of the DbVisualizer user preferences folder named *Bookmarks*. The user preferences folder is typically a subfolder named *.dbvis* in your home folder.

The main Bookmark content is stored in a file with exactly the same name as the node in the Scripts tab. The additional data associated with the Bookmark is stored in a file with the same name plus the *.met* extension.

To share some of your Bookmarks with someone, we recommend that you use DbVisualizer to create a separate Bookmarks subfolder for the shared Bookmarks. You can then use any external tool to create a file archive (e.g. a ZIP file) of that subfolder and send it to your friend or colleague. He or she can then extract the files into the local DbVisualizer user preferences *Bookmarks* folder.

History

As you execute SQL statements in the SQL Commander, DbVisualizer saves them as History entries, along with information about the Connection, Catalog, Schema and the execution result. This makes it easy to locate statements and scripts you have executed in the past.

If you just want to go back and forth between statements you have executed recently, you can use the **Get Previous from History** and **Get Next from History** toolbar buttons in the SQL Editor.

To look through all saved statements, you can display the History entries by clicking the **Display SQL History** toolbar button in the main window or select the corresponding operation from the **Tools** menu.

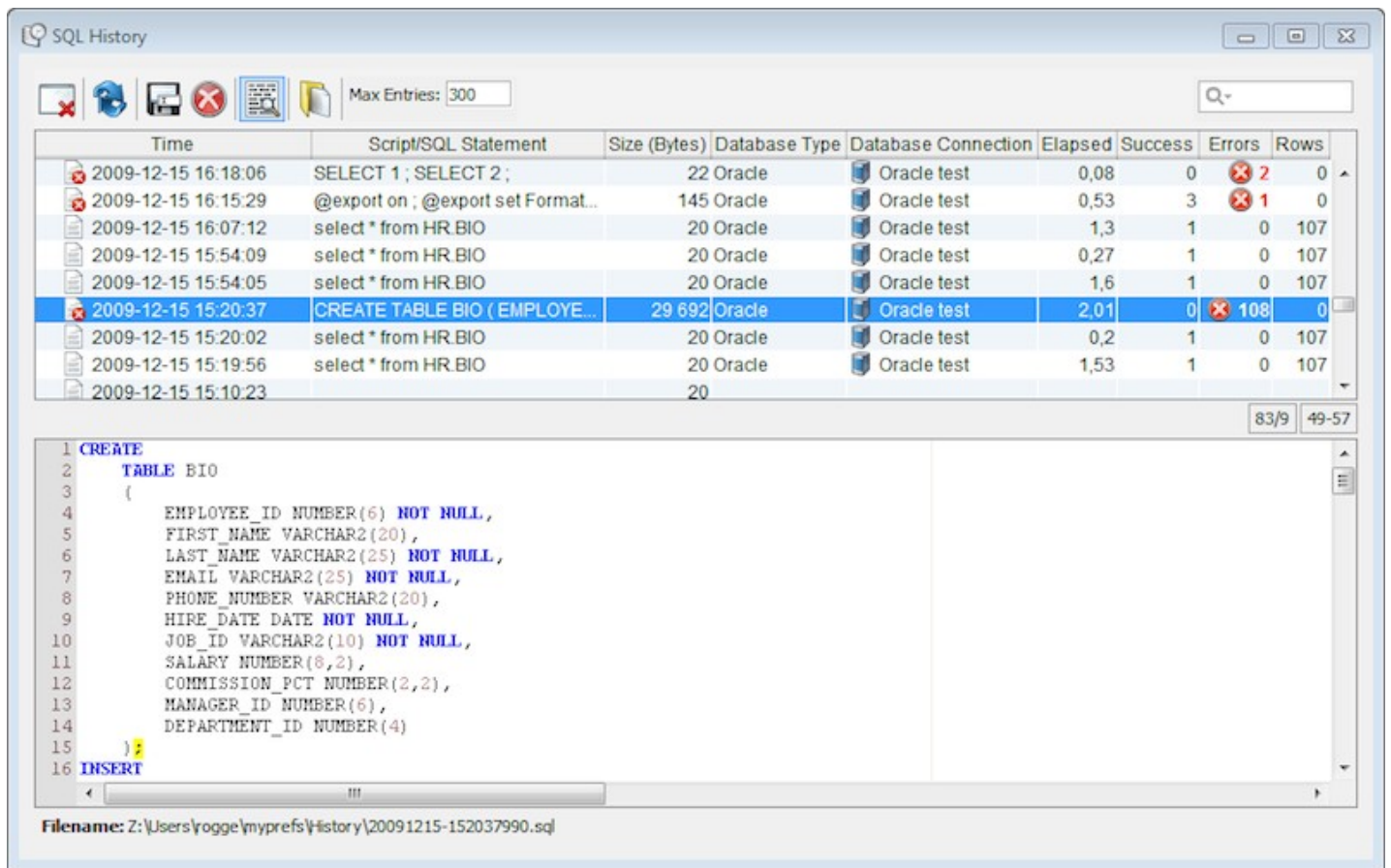


Figure: The SQL History window

The entries are ordered with the most recently executed entries at the top by default, but you can reorder them by clicking on the column headers. The complete content of the selected entry is shown below the list, unless you disable it by clicking the **Show Details** toolbar button or select the corresponding operation from the right-click menu.

Reusing a History Entry

When you have found the entry you're looking for, you can open it in the current SQL Editor by double-clicking it or clicking the corresponding toolbar button. If you want to open the entry in a new editor, use the right-click **Open File -> Open in New Editor** operation.

You can also add the content of an entry to the current content of an SQL Editor. Select the entry in the list, drag it with the mouse key depressed to the position in the editor where you want to add it, and drop it there by releasing the mouse button.

Saving a History Entry as a Bookmark

If you realize that you need easy access to a History entry in the future, you can save it as a Bookmark. Just select the entry and use the **Save as Bookmark** operation, or just drag it to the Bookmarks tree and drop it.

Quick Load

An alternative to locating Bookmarks in the Scripts tab and History entries in the History window is to use the Quick Load feature, by default bound to the **Ctrl+Alt+O** key combination. It is also available via a toolbar button in the SQL Editor as well as in the main **File->Quick File Open** menu.

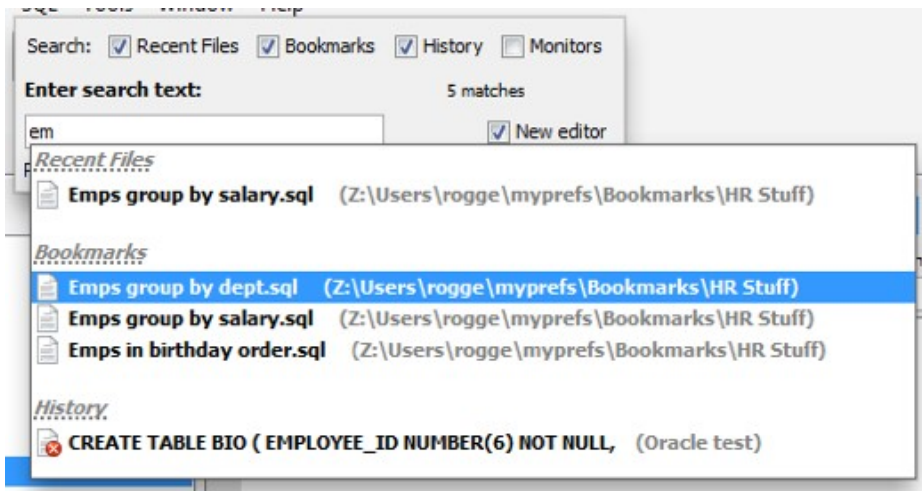


Figure: Quick Load window

The Quick Load feature locates files with partly matching names from the categories you have selected, as you type. You can use an asterisk ("*") as a wildcard in the search string.

When you see the file you're looking for, just select it and click Enter to load it into an SQL Editor. Use the New Editor checkbox to decide whether you want to open it in a new editor or in the current editor, and use the Max field to limit the number of matching files to display in the list.

Monitor and Charts

Introduction

With the monitor feature, you can track changes in data over time, viewing the results of one or many SQL statements either as grids or graphs. Typically, you configure the monitor to run the statements automatically at certain intervals.

The monitoring feature combined with the charting capability in DbVisualizer Personal is really powerful, delivering real time charts of many result sets simultaneously. For example, you can use monitoring to spot trends in a production database, surveillance, statistics, database metrics, and so on. At DbVis Software, we have a dedicated workstation that uses the monitoring feature to automatically present live chart information from our Internet servers and customer database.

Any SQL statement that produces a result set can be monitored, and when you monitor multiple statements, different statements may use different database connections concurrently.

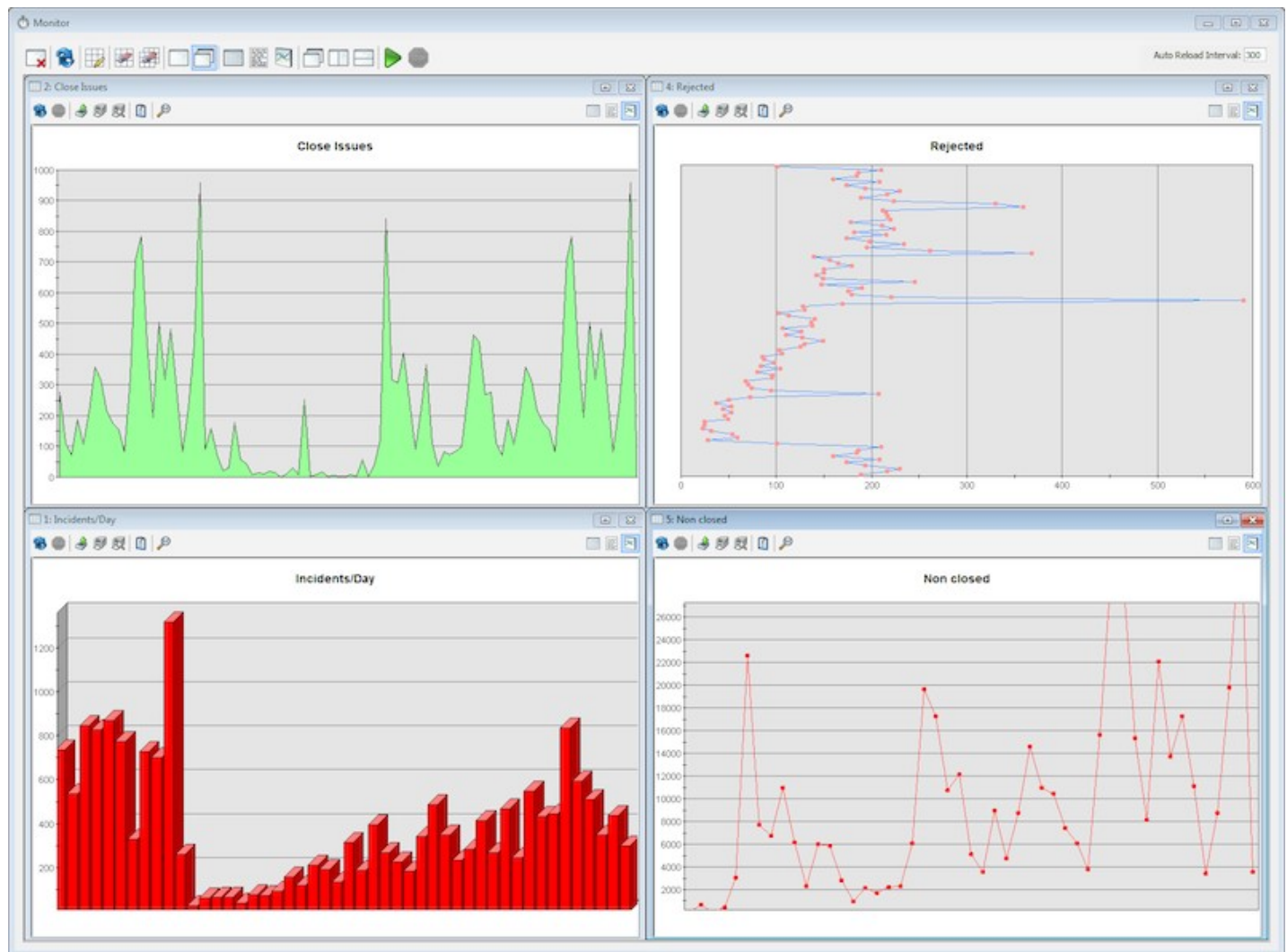


Figure: Monitor window showing result as charts

The chart customization covered in this document is also applicable to the charts for result sets in the SQL Commander (DbVisualizer Personal).

Monitored SQL Statements

Monitored SQL statements are managed under the **Monitors** node in the **Scripts** tab in the tree area to the left in the main DbVisualizer window. A monitored SQL statement is associated with information about the target database connection and (optionally) the catalog (the JDBC term which translates to a *database* for some databases, like Sybase, MySQL, SQL Server, etc) and schema. It also has a title, a maximum row count (how many results to keep track of) and a visibility status (whether the monitored statement result should be included in the Monitors windows, discussed below). This information is displayed, and can be edited, in the lower part of the Scripts tab, along with information about the file that holds the monitored statement. If you don't want to see these details, you can disable it with the **Show Details** toggle control in the right-click menu for a node.

The screenshot shows the DbVisualizer interface with the **Scripts** tab selected. The left sidebar shows a tree view under **Scripts** with a **Monitors** folder containing several SQL files, with **Incidents.sql** selected. Below the tree is a detailed view for the selected file, showing its name, size, modification date, path, encoding, database connection (MySQL, puredb), and monitor settings (Title: Incidents/Day, Visible: checked, Max Row Count: 100). The main window displays the SQL query for **Incidents.sql** in the **SQL Commander** view. The query is a SELECT statement that groups incidents by year and month, ordered by year and month ascending. The status bar at the bottom shows the current line (13:14), mode (INS), and other settings like Auto Commit: ON and Encoding: Cp1252.

```
1 SELECT
2   YEAR(TDate) AS YEAR ,
3   MONTH(TDate) AS MONTH ,
4   concat(YEAR(TDate), '-', DATE_FORMAT(TDate, '%b')) ,
5   SUM(Incidents) AS COUNT
6 FROM
7   BugHouse
8 GROUP BY
9   YEAR ,
10  MONTH
11 ORDER BY
12  YEAR ,
13  MONTH ASC
```

Figure: The Scripts tab with Monitors

The figure above shows the **Incidents/Day** monitored statement and the SQL that is associated with it.

The following is an example of the result set produced by the statement:

	YEAR	MONTHNUM	MONTH	COUNT
26	2004	8	2004 Aug	269
27	2004	9	2004 Sep	467
28	2004	10	2004 Oct	244
29	2004	11	2004 Nov	549
30	2004	12	2004 Dec	433
31	2005	1	2005 Jan	443
32	2005	2	2005 Feb	837
33	2005	3	2005 Mar	593
34	2005	4	2005 Apr	511
35	2005	5	2005 May	349
36	2005	6	2005 Jun	437
37	2005	7	2005 Jul	300
38	2005	8	2005 Aug	417
39	2005	9	2005 Sep	581
40	2005	10	2005 Oct	549

0.032/0.000 sec 89/4 26-40

Figure: Monitor window showing the result in Grid format

The interesting columns in the result are the **Month** and **Count**. The **Year** and **MonthNum** are there just to get the correct ascending order of the result.

Creating, Editing and Organizing Monitored Statements

You can create and work with monitored statements in the same way as with a Bookmark. The main difference is how they are used and a couple of additional ways monitored statements can be created. For information about how to manually create, manage and share monitored statements, please see the [Bookmarks and History](#). The following sections describe how you can get help creating the bookmarks for a couple of cases that are commonly used for monitoring.

Monitor table row count

It is very common to want to keep track of how the number of rows in a table varies over time. The right-click menu in the Data tab grid for a table therefore has a **Create Row Count Monitor** operation that creates a monitored statement for you automatically.

It creates a monitor with SQL for returning a single row with the timestamp for when the monitor was executed and the total number of rows in the table at that time. Every time the monitor is executed, a new row is added to the grid, up to a specified maximum number of rows. When the maximum row limit is reached, the oldest row is removed when a new row is added. Example:

PollTime	RowCount
2003-01-23 12:19:10	43123
2003-01-23 12:11:40	43139
2003-01-23 12:21:10	43143
2003-01-23 12:22:40	43184
...	...

Figure: Example of the result from a Table Row Count monitor

The SQL for this monitor uses two variables, **DbVis-Date** and **DbVis-Time**. These variables are substituted with the current date and time, formatted

according to the corresponding Tool Properties settings. The reason for using these variables instead of using SQL functions to retrieve the values is simply that it is almost impossible to get the values in a database-independent way. Another reason is that we want to see the client machine time rather than the database server time. You can, of course, modify the SQL any way you see fit, as long as the **PollTime** and **RowCount** labels are not changed.

```
select '$$DbVis-Date$$ $$DbVis-Time$$' as PollTime,
       count(*) as RowCount
from Computers
```

Figure: Sample of the SQL for the Table Row Count monitor

DbVisualizer keeps the result for previous executions, up to the specified maximum number of rows, so that you can see how the result changes over time. You define the maximum number of rows in the **Max Row Count** field in the details area at the bottom of the Scripts tab. This property is initially set to 100 when you use Create Row Count Monitor to create the monitor.

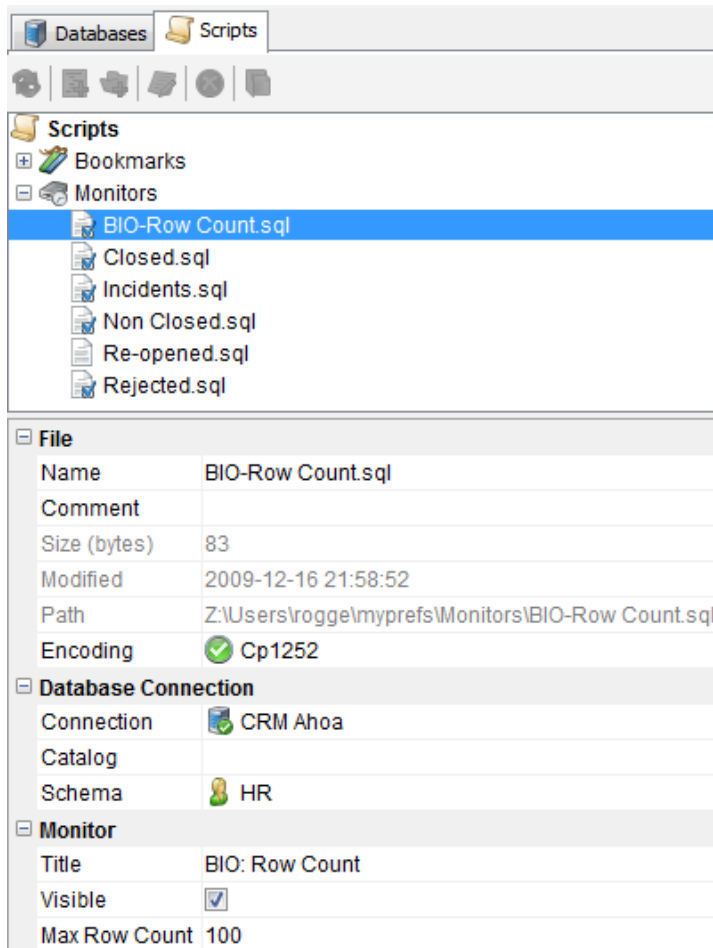


Figure: Details area with the Max Row Count field for a monitored statement

You can change the value to limit or extend the number of rows that DbVisualizer should keep. Setting it to 0 or a negative number tells DbVisualizer to always clear the grid between executions of monitors.

Monitor table row count difference

In addition to tracking the number of rows in a table over time, you may want to see by how many rows the value changes. You can create a monitor for this purpose with the **Create Row Count Diff Monitor** operation, available in the right-click menu for the grid in the Data tab.

In addition to the Row Count Monitor, the Row Count Diff Monitor reports the difference between the number of rows in the last two executions:

```
PollTime      RowCount RowCountChange
```

```

2003-01-23 12:19:10 43123    0
2003-01-23 12:11:40 43139    16
2003-01-23 12:21:10 43143    4
2003-01-23 12:22:40 43184    41
...

```

Figure: Example of the result from a Table Row Count Difference monitor

The SQL for this monitor adds a third column, named **RowCountChange**. It utilizes the fact that DbVisualizer automatically creates variables for the columns in a monitor result set, holding the values from the previous execution. The RowCountChange column is set to the value returned by the count(*) aggregate function for the current execution minus the value from the previous execution, held by the RowCount variable. All columns in a monitor result set can be used like this to reference values from the previous execution of the monitor.

```

select '$$DbVis-Date$$ $$DbVis-Time$$' as PollTime,
       count(*) as RowCount,
       count(*) - $$RowCount||count(*)$$ as RowCountChange
from Computers

```

Figure: Sample of the SQL for the Table Row Count Difference monitor

Monitor Window

The Monitor window, launched via the **Tools->Monitor** menu option, is where you active monitors and look at the results. The monitors can be organized either as tabs or internal windows. The monitor results can be viewed only as grids in DbVisualizer Free, while DbVisualizer Personal adds the capability to view them as charts or text.

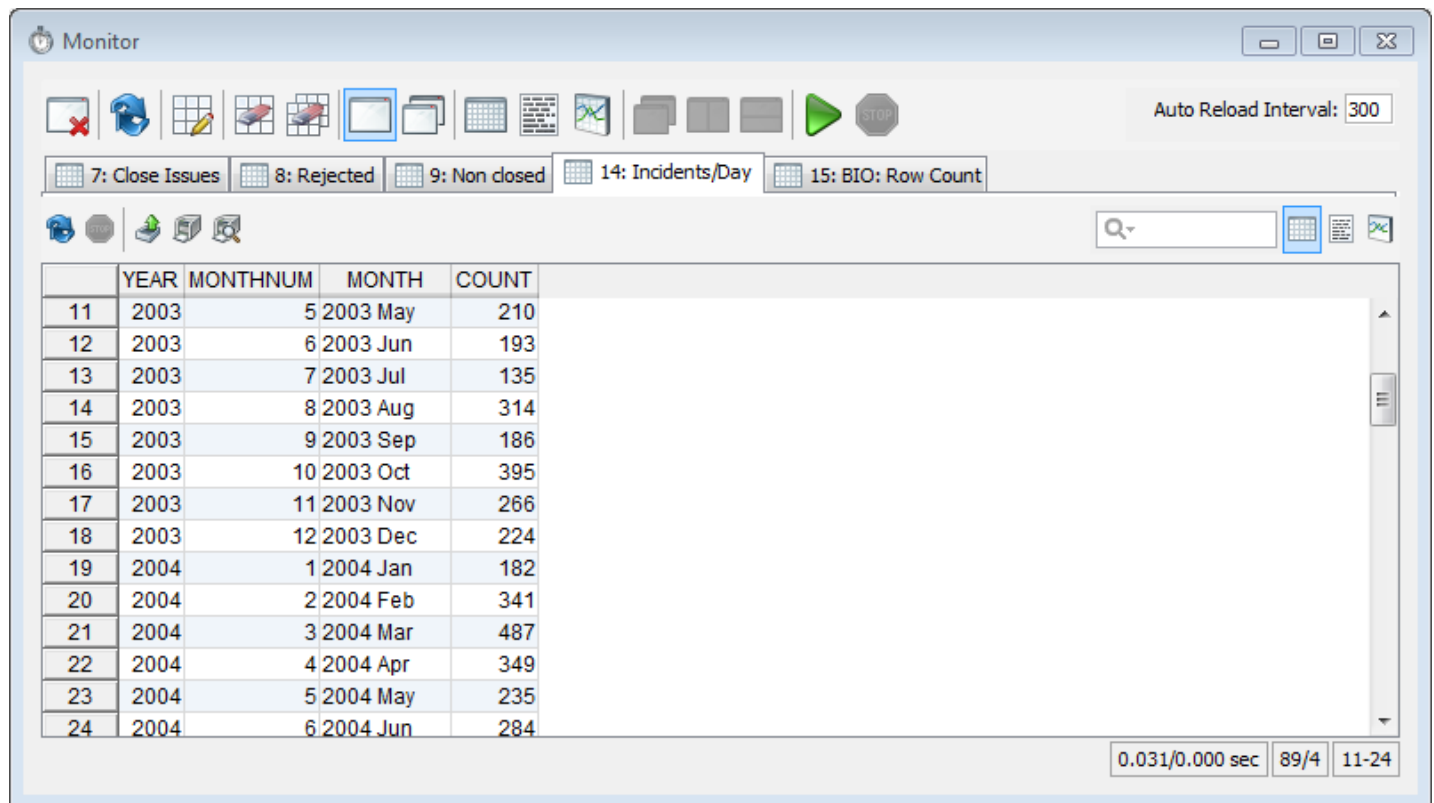


Figure: The Monitor window with all monitors organized as tabs

The Monitor window has toolbar at the top with an **Auto Reload Interval** field. The rest of the window holds result areas for each monitored statement with the **Visible** attribute enabled. Each individual monitor result tab or window also has a toolbar with controls that apply just to that result. The screenshot is from DbVisualizer Personal, with **View** buttons in the toolbar for the selected monitor; these buttons are not included in DbVisualizer Free.

The main toolbar buttons have the following functions:

Toolbar Button	Description
Close	Closes the Monitor window
Reload	Reloads all results (i.e., executes all monitors and updates the result sets)
Locate Current	Locates and select the monitor node in the Scripts tab corresponding to the currently selected result
Clear Current	Clears the currently selected result
Clear All	Clears all results
Show as Tabs	Shows the results as tabs
Show as Windows	Shows the results as internal windows
Show Grids	Shows all results as grids
Show Graphs	Shows all results as graphs
Show Text	Shows all results as text
Cascade Windows	Arrange the result windows so the overlap each other
Tile Vertically	Arrange the result windows side-by-side vertically
Tile Horizontally	Arrange the result windows side-by-side horizontally
Start Monitors	Starts auto-update of all monitors, repeatedly executing all statements at the intervals specified by the Auto Reload Interval field
Stop Monitors	Stops the auto-update

The **Auto Reload Interval** field is used to control how often to execute the monitors when auto update is running. Use the field to specify how many seconds to wait between auto-reloads. The specified number of seconds may be increased automatically by DbVisualizer if the total execution time for all monitors is longer than the specified value.

Charts

This section is only applicable to DbVisualizer Personal.

Charts in conjunction with the Monitor feature is really powerful, since monitored data is very often a good candidate to be charted. The charting capability in DbVisualizer Personal is also available in the SQL Commander; everything described here also applies to the grids for the result sets in the SQL Commander.

The basic setup of a chart is really easy. It is just a matter of selecting one or more columns that should appear as series in the chart. The basic requirement is that the monitor has been executed, so that there are columns to choose the series from. The appearance of the charts can be thoroughly customized using the advanced customization editor.

The chart view is controlled by the buttons in toolbar shown in the monitor result area when the graph mode is selected:



Figure: Chart control buttons

The following sections explain the features and how to setup the chart.

Chart Controls

The chart controls are used to customize the **Data** that shall be displayed in the chart, optional axis labels, titles, etc. It is also used to control the **Layout** of the chart in terms of chart type, legend type, etc.

Data

Use the controls in the Data tab to customize which data shall appear in the chart.

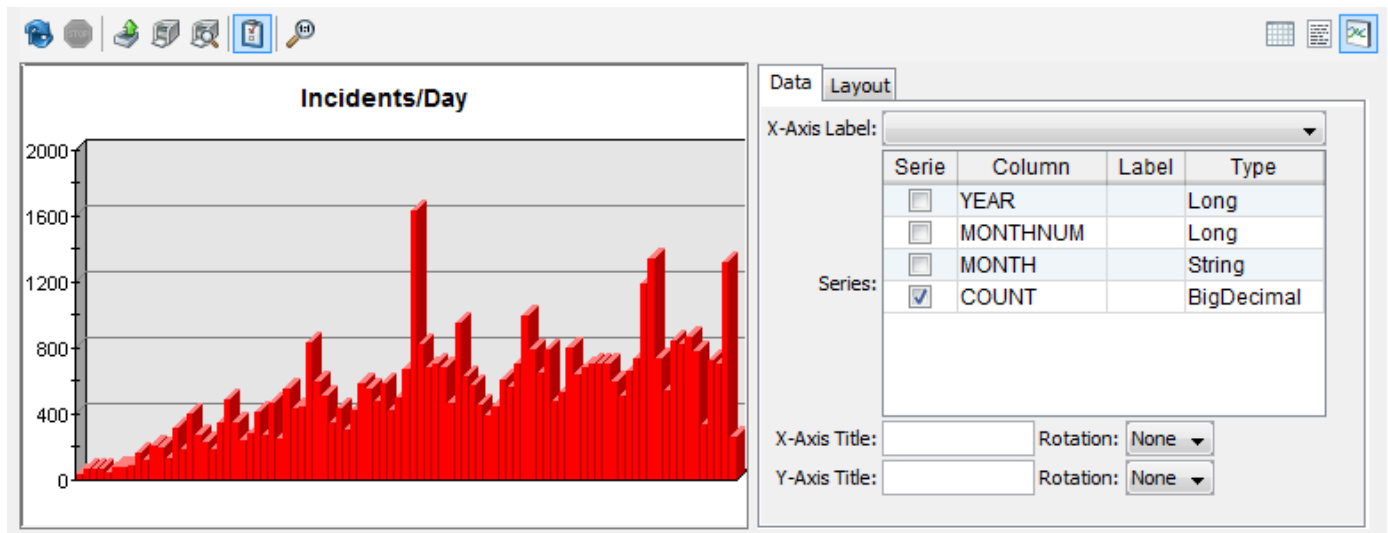


Figure: Data customizer

Select at least one **Series** from the list of columns. As soon as you select a series, it is immediately added to the graph. The **Label** field can be used to specify an optional label for the series as it should appear in a legend. The column name is used if no label is specified.

The **X-Axis Label** box is used to specify the column in the result that should be used to render the labels of the X-axis. **X-Axis Title** and **Y-Axis Title** specifies the titles for the X and Y axis. You can use the **Rotation** settings if you want the X and Y axis text rotated.

The **title** for the monitored statement, as defined in the details area in the Scripts tab, is used as the title for the chart. The script file name is used if no title has been specified.

Layout

The layout tab is used to configure the appearance of the chart, primarily the type of chart want to use. Note that all settings are per monitor. The following screen shots show some of the most commonly used chart types.

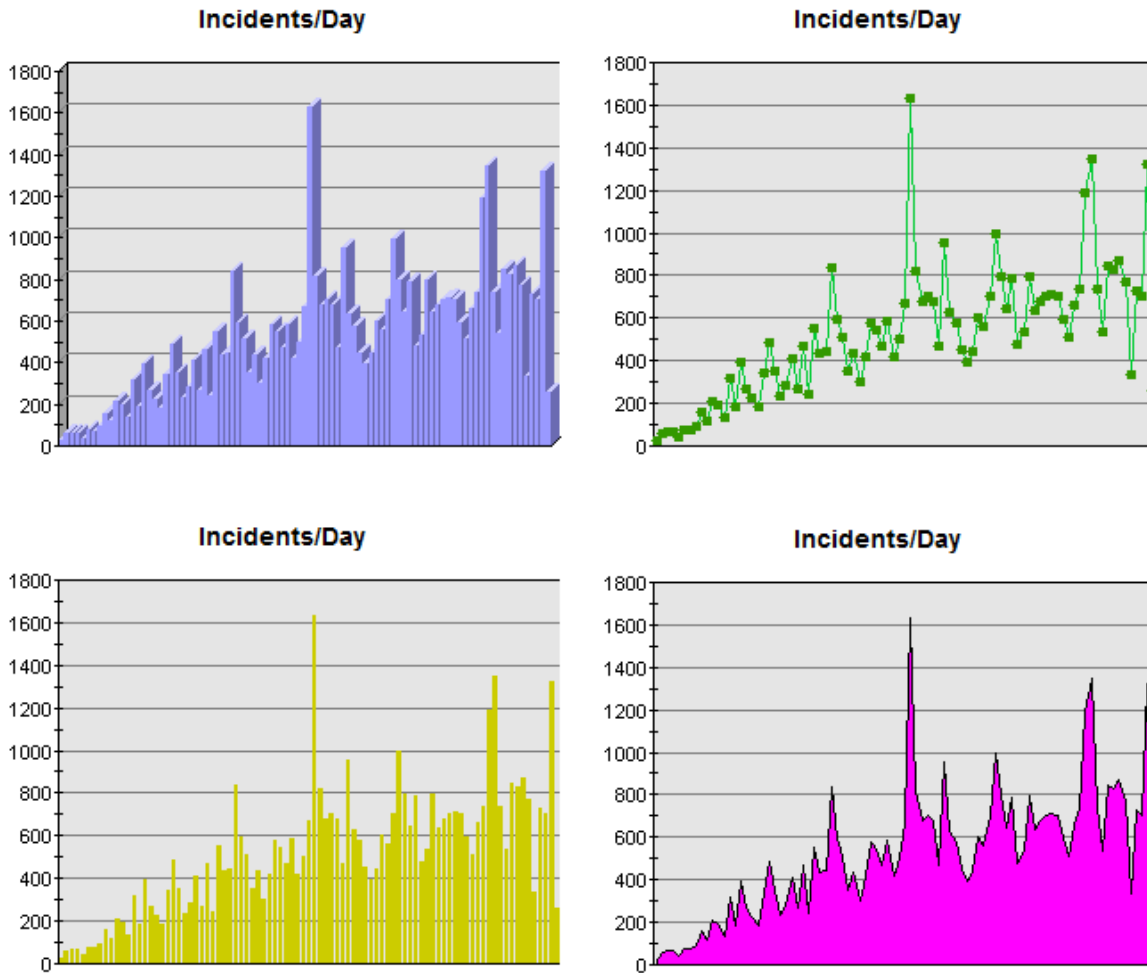


Figure: Chart type examples

The advanced layout editor can be used to customize every aspect of the layout. The basic layout settings, however, are the following:

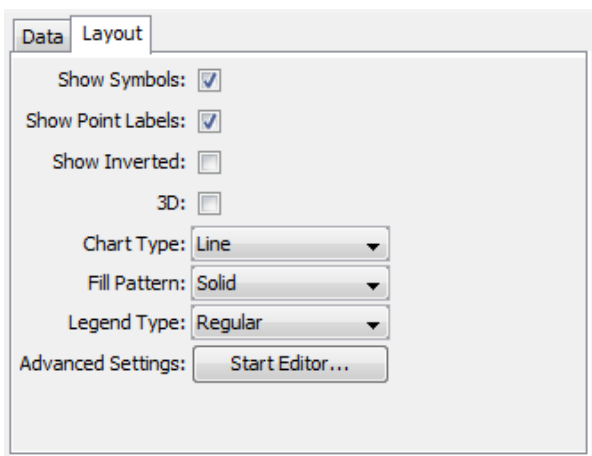


Figure: Layout customizer

Show symbols specifies whether each value in a line chart should be represented by a symbol. **Show Inverted** defines whether the X and Y axis will be switched. **3D** specifies if a bar chart will be displayed in 3D. The **Chart Type** lists all the available chart types. Fill Pattern defines how a bar, area and pie chart shall be filled. **Legend Type** specifies whether a legend will be displayed or not.

You can use the **Advanced Settings** editor to customize all the bits and pieces of the chart. This document does not explain all the configurations that can be done using this editor since that would result in a 100 page book. Play around with the different settings and see how the graph changes.

Settings that are made in the Advanced Editor are not saved between invocations of DbVisualizer.

Chart View

Zooming

Charts support zooming by selecting a rectangle in the chart area. Selecting another rectangle in that zoomed area will zoom the chart even further, and so on. To reset the zoom, click the **Reset Zoom** button or press the "r" keyboard button while the mouse pointer is in the chart area.

Rotating

All 3D chart types support rotating and changing the depth of the chart. Use the following to change the appearance:

- **Shift+Left Mouse button**
Changes the depth of the chart
- **Ctrl+Left Mouse button**
Changes the rotation of the chart

Examples:

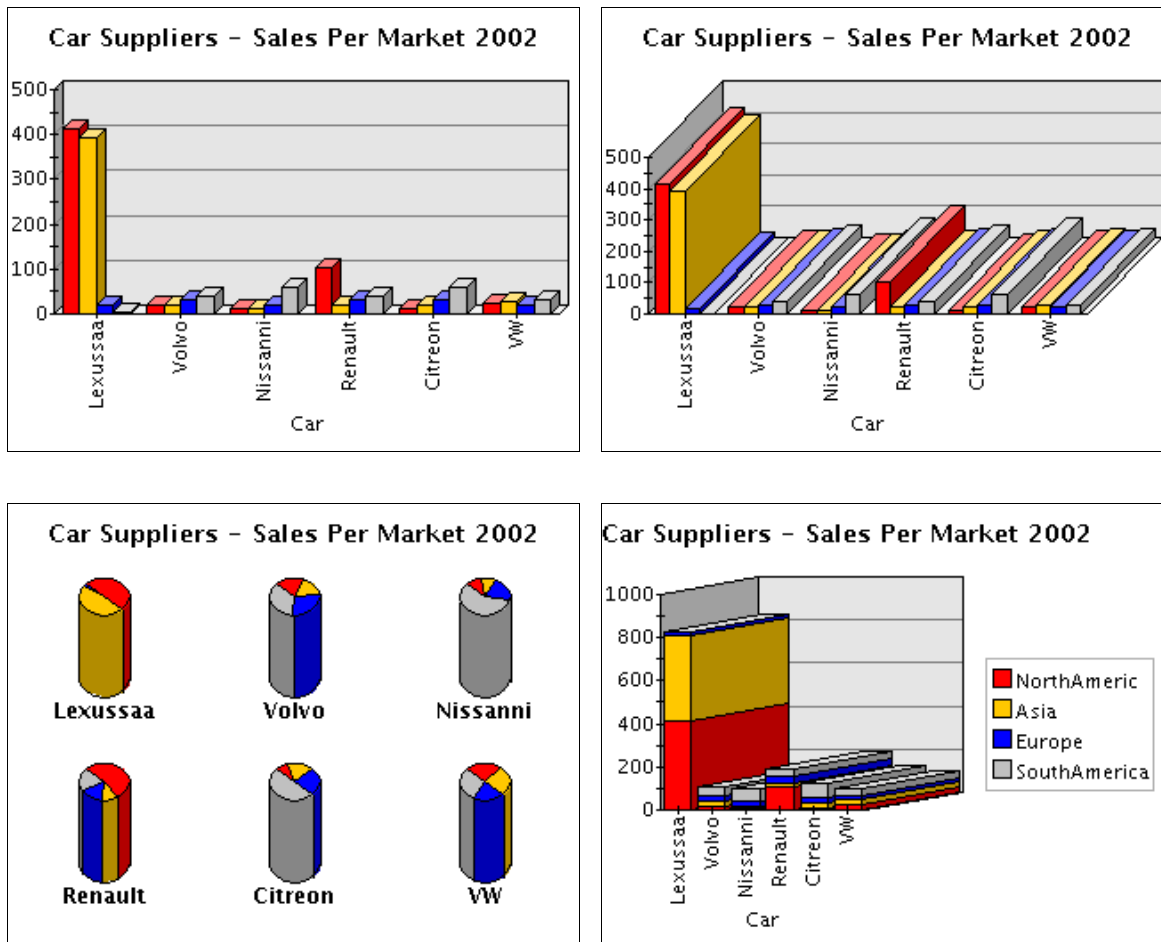


Figure: Example of 3D charts

The above screen shots are just a few examples of the 3D chart types and how depth and rotation settings are used to change the appearance.

Export

The export operation is context sensitive and works on the currently selected chart, graph or grid. The controls in the export dialog also adapt to the currently selected object. If a chart is the current object the following export dialog will appear:

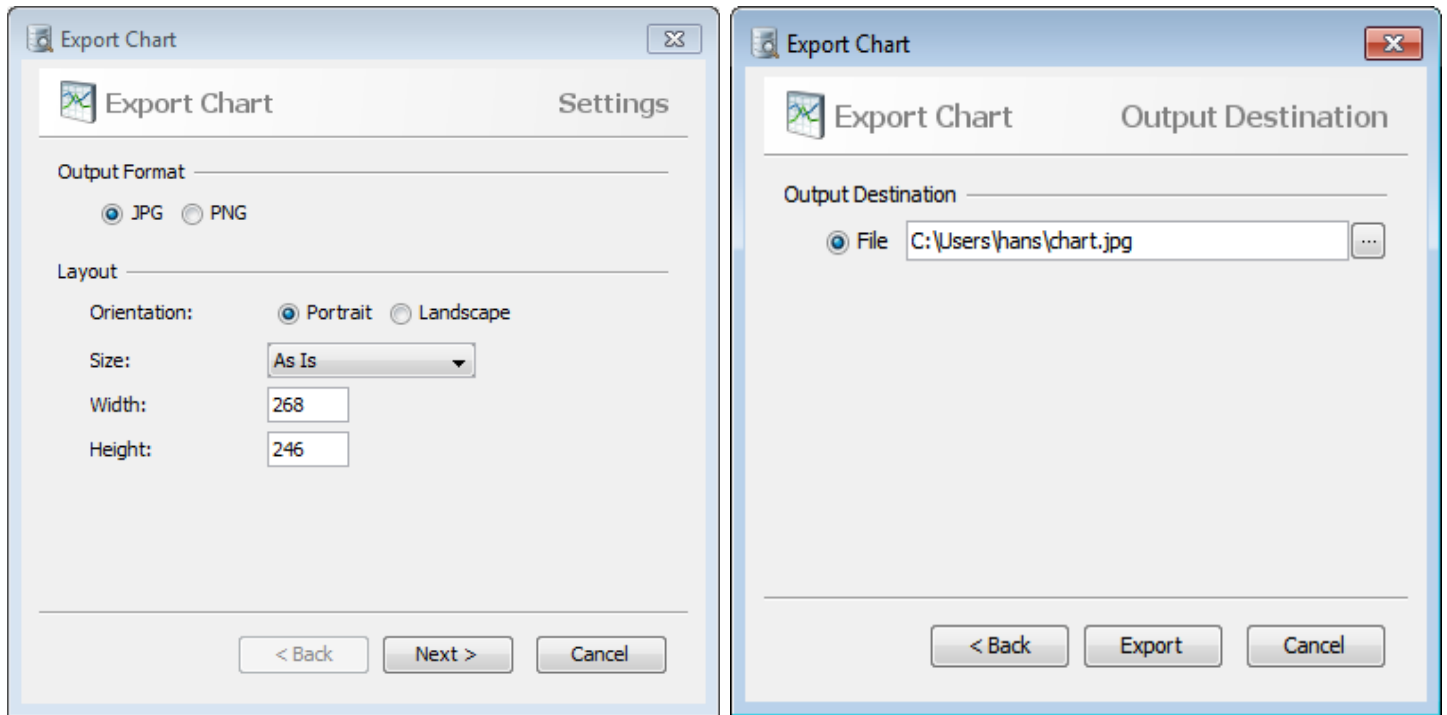


Figure: Export dialog for charts

The default size of the exported image is the same as it appears on the screen. To change the size, either select a pre-defined paper size in the **Size** list or enter a size in pixels.

Create and Alter Table

Introduction

The Create Table, Create Index and Alter Table Assistants are used to create new tables and indexes and to alter existing tables. The assistants are quite simple to use since they examine various metadata in the database (depending on which assistant is used) and then let you point and click to define the table or index.

The assistants are launched from the **Database->Selected Object** main menu, from the **Database Objects** tree right-click menu, or from the **Actions** menu button in the object view. The menu choices are enabled only if a table or index can be created for the selected node in the Database Objects tree.

Create Table

To create a table, select an appropriate node in the objects tree, typically a Tables node, and launch the Create Table assistant from one of the menus as described earlier.

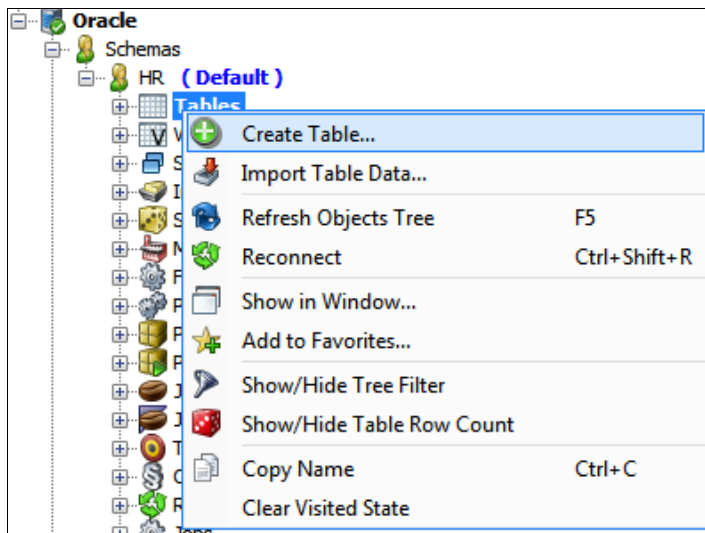


Figure: The right-click menu in the Database Objects tree

The Create Table assistant is organized in three areas from the top:

- **General Table Info**
Specifies the owning database connection, database and/or schema. These are picked up from the selection in the tree when the assistant is started. Table name is set to a default name that you can change to the real table name.
- **Table Details**
A number of tabs where you specify information about the columns and, optionally, various constraints. The Columns, Primary Key and Foreign Key tabs are available for all databases. The remaining tabs are database-specific and depends on the features supported by the database engine.
- **SQL Preview**
The SQL previewer instantly shows the SQL statement for creating the table.

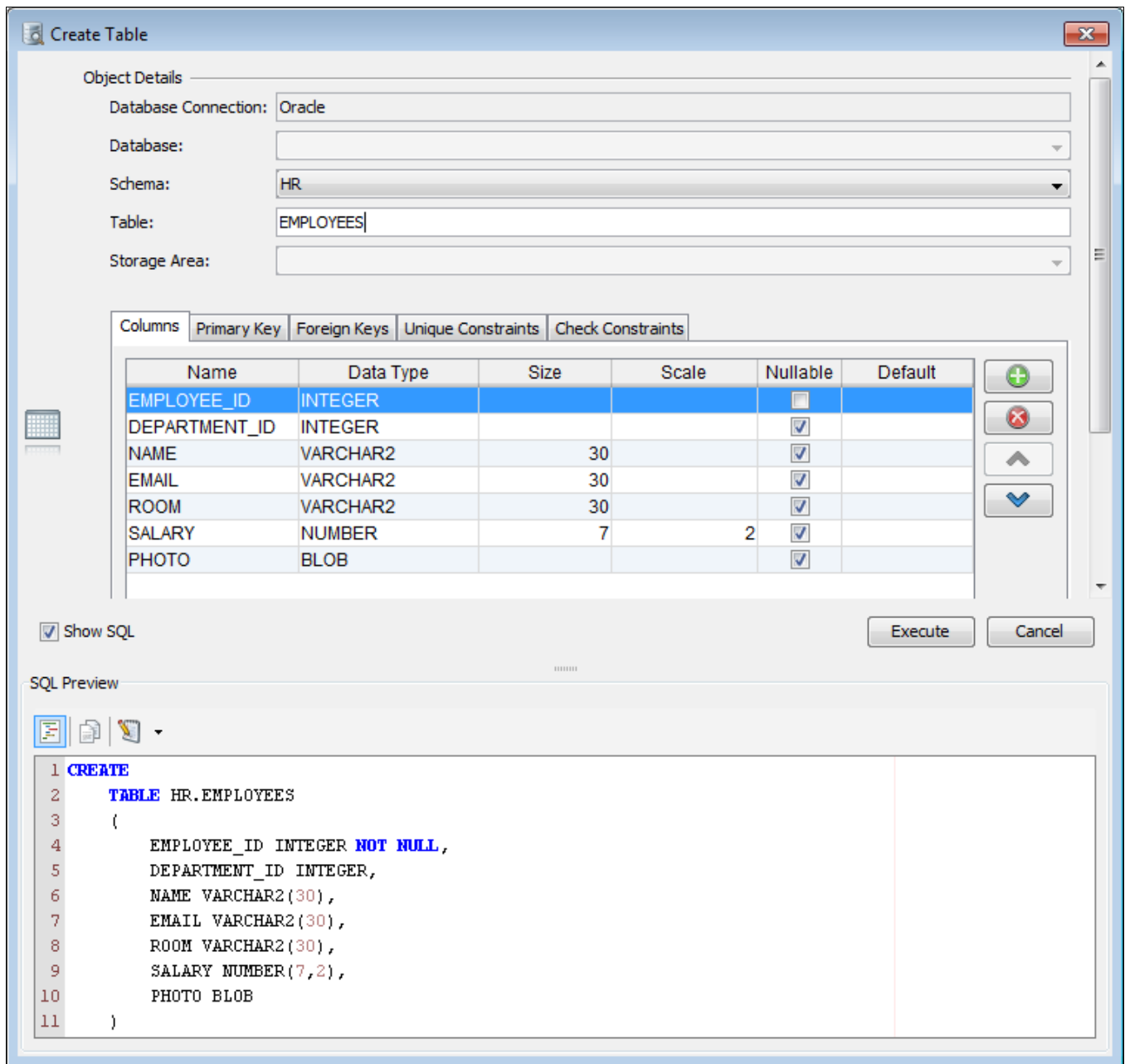


Figure: The Create Table assistant

Just enter as much information as is needed to describe the table and click **Execute** to create the table.

Columns tab

The Columns tab lists all table columns along with their attributes.

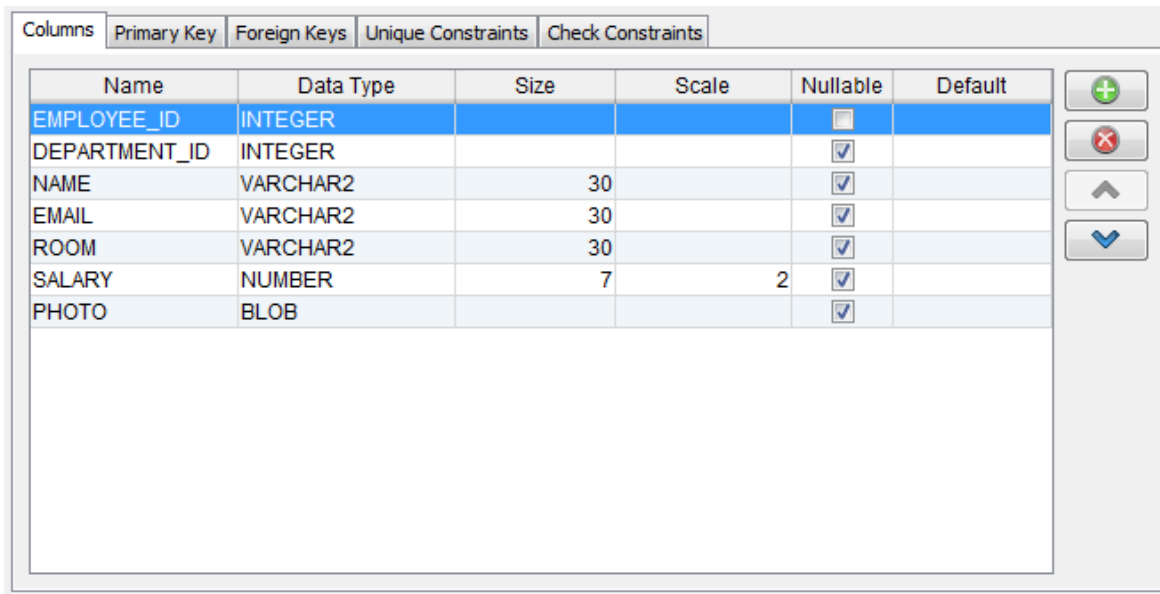


Figure: The Columns tab (for Oracle)

Add columns by clicking on the **Add** button, and remove the currently selected column by clicking on the **Remove** button. You can reorganize the columns using the **Up** and **Down** buttons.

Enter the name of the column in the first field and select a data type from a drop down list in the second field. The list contains the names of all data types the database supports.

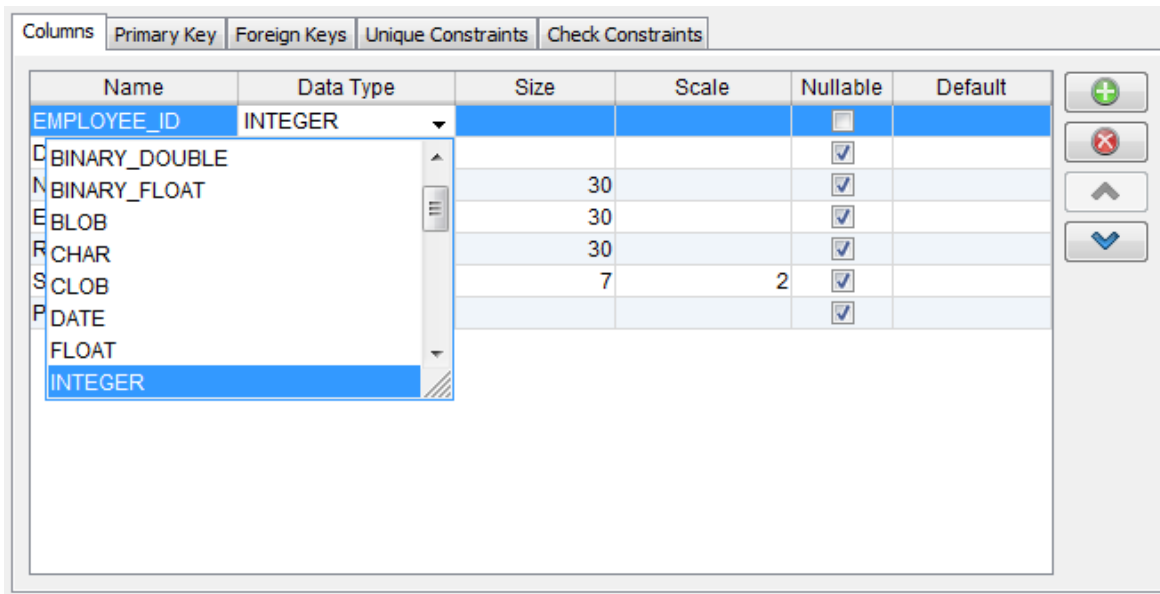


Figure: Data Type list (for Oracle)

For some data types, such as character types, you may also specify a size, i.e., the maximal length of the value. For others, like the decimal types, you can may specify both a size and a scale (the maximal number of decimals).

ROOM	VARCHAR2	30		<input checked="" type="checkbox"/>	
SALARY	NUMBER	7	2	<input checked="" type="checkbox"/>	
PHOTO	BLOB			<input checked="" type="checkbox"/>	

Figure: Size and scale for a decimal data type

The above example will allow a total length (including the decimal places) of 7. Examples:

```
9871.1
8172.0
 12.112 Error!
1921211.11 Error!
```

The last two fields let you specify if the table is nullable and a default value to use for rows inserted into the table without specifying a value for the column.

Below the column list, you may see one or two additional fields, depending on the features supported by the database you create the table for. The **Collation** field is enabled for character columns if the database supports the declaration of a collation for textual data. The **Auto Increment** field is enabled for numeric fields if the database supports automatically inserting the next available sequence number in a numeric column.

Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			<input type="checkbox"/>	
DEPARTMEN...	INTEGER			<input checked="" type="checkbox"/>	
NAME	VARCHAR	30		<input checked="" type="checkbox"/>	
EMAIL	VARCHAR	30		<input checked="" type="checkbox"/>	
ROOM	VARCHAR	30		<input checked="" type="checkbox"/>	
SALARY	DECIMAL	7	2	<input checked="" type="checkbox"/>	
PHOTO	BLOB			<input checked="" type="checkbox"/>	

Collation: Auto Increment:

Figure: Collation and Auto Increment controls

The Create Table assistant uses database metadata to try to enable only the fields that apply to the selected data type, but please note that it is not always possible. For instance, there is no metadata available to tell if a data type requires, or allows, a size. If you don't enter a required attribute or enter an attribute that is unsupported for a data type, you will get an error message when you click Execute to create the table.

Primary Key tab

The Primary Key tab contains information about an optional primary key for the table. A primary key is a column, or a combination of columns, that uniquely identifies a row in a table.

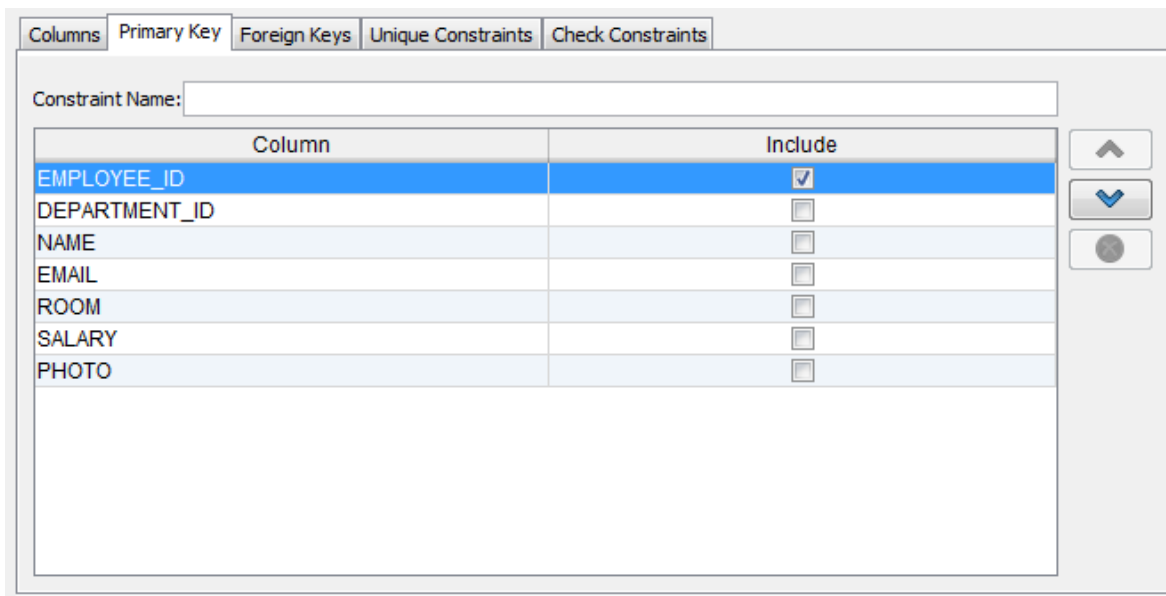


Figure: Primary Key tab

You can, optionally, enter a constraint name for the primary key constraint in the Constraint Name field. Select the columns to be part of the primary key by clicking the checkboxes in the Include field in the columns list.

You can change the order of the columns in the key by selecting a column and move it using the **Up** and **Down** buttons.

Foreign Keys tab

In the Foreign Keys tab, you can declare one or more foreign keys for the table. A foreign key is a column, or a combination of columns, that refer to the primary key of another table. Foreign keys are used by the database to enforce integrity, i.e., that there is a row in the referenced table with a primary key that matches the foreign key value when a new row is inserted or updated, and can optionally declare rules for what to do when a referenced primary key is removed or updated in the referenced table.

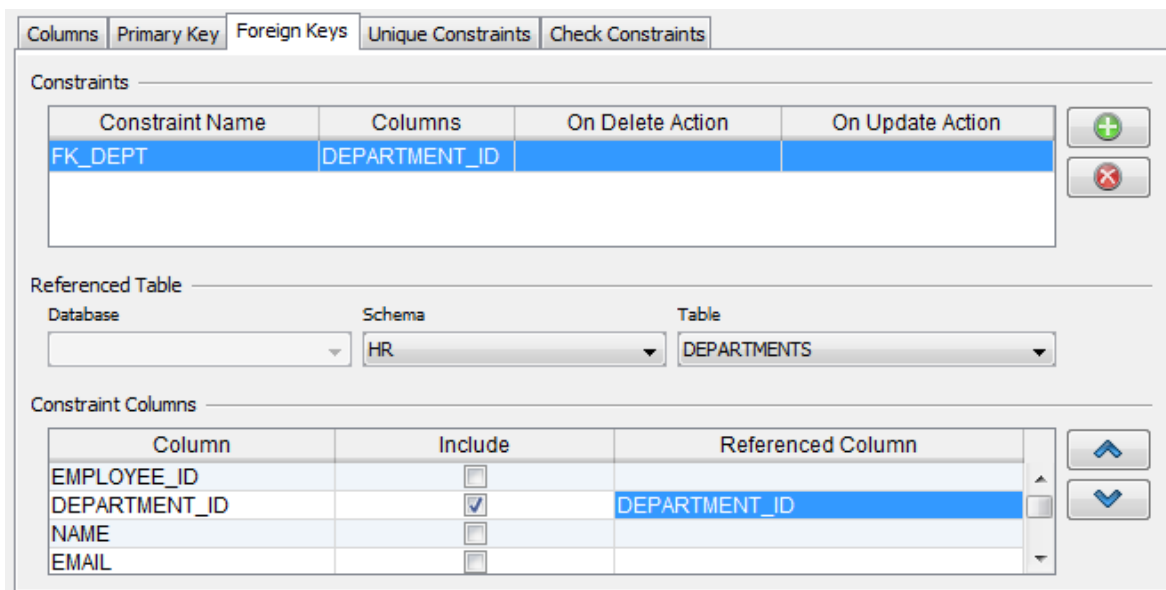


Figure: Foreign Keys tab

The tab has the following sections:

- A list of foreign keys.
- Controls for selecting the table the currently selected foreign key refers to, including the database (catalog) and/or schema for the table.
- A list of all columns for the table being created.

To declare a new foreign key constraint, click the **Add** button next to the list of foreign keys. You can then enter a name for the foreign key in the first field in the list, and select On Delete and On Update actions from the pull-down menus. The pull-down lists include all actions that the database support, typically CASCADE, RESTRICT, NO ACTION and SET NULL. The Columns field is read-only and gets its value automatically when you select which columns to include in the key later.

Next, use the Referenced Table controls to select the table that the foreign key refers to.

Finally, check the Include checkbox for all columns in the column list that should be part of the foreign key and then select the corresponding column in the referenced table from the pull-down menu in the Referenced Column field. You can change the column order for the key with the **Up** and **Down** buttons.

To remove an existing foreign key, select it in the list in the top section and click the **Remove** button.

Unique Constraints tab (database-specific)

The Unique Constraints tab is only available for databases that support this constraint type. A unique constraint declares that the columns in the constraint must have unique values in the table.

Constraint Name	Columns
IX_DEPT	DEPARTMENT_ID

Column	Include
EMPLOYEE_ID	<input type="checkbox"/>
DEPARTMENT_ID	<input checked="" type="checkbox"/>
NAME	<input type="checkbox"/>
EMAIL	<input type="checkbox"/>
ROOM	<input type="checkbox"/>
SALARY	<input type="checkbox"/>
PHOTO	<input type="checkbox"/>

Figure: Unique Constrains tab

The top portion of the tab holds a list of all unique constraints, and the lower portion holds a list of all table columns.

To create a constraint, click the **Add** button and optionally enter a constraint name in the Constraint Name field. The Columns field in the constraints list is read-only, filled automatically as you include columns in the constrain. Select the columns to be part of the constraint by clicking the checkboxes in the Include field in the columns list.

You can change the order of the columns in the constraint by selecting a column and move it using the **Up** and **Down** buttons.

To remove an existing constraint, select it in the list in the top section and click the **Remove** button.

Check Constraints tab (database-specific)

The Check Constraints tab is only available for databases that support this constraint type. Check constraints declare that a column value fulfills a certain condition when a row is inserted or updated. Some databases uses check constraints to enforce nullability rules, so when you alter a table (as described later), you may see auto-generated check constraints for columns that you marked as not allowing null values in the Columns tab.

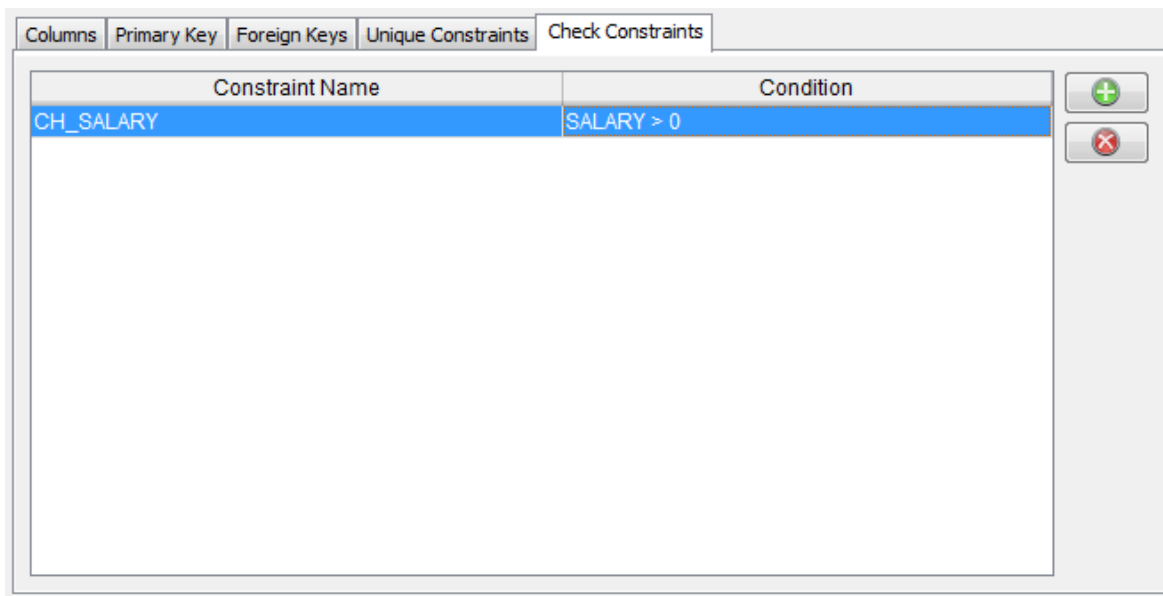


Figure: Check Constrains tab

To create a check constraint, click the **Add** button and optionally enter a constraint name in the Constraint Name field. Enter the condition for the column in the Condition field. You can use the same type of conditions as you use in a SELECT WHERE clause.

To remove an existing constraint, select it in the list and click the **Remove** button.

Indexes tab (MySQL only)

The Indexes tab is only used for the MySQL database, as a replacement for the Unique Constraints tab. The reason is that for MySQL, the CREATE TABLE statement can be used to declare both unique and non-unique indexes. MySQL also does not make a clear distinction between a unique constraint (a rule, most often enforced and implemented as an index by the database) and a unique index (primarily a database structure for speeding up queries, with the side-effect of ensuring unique column values), as most other databases do.

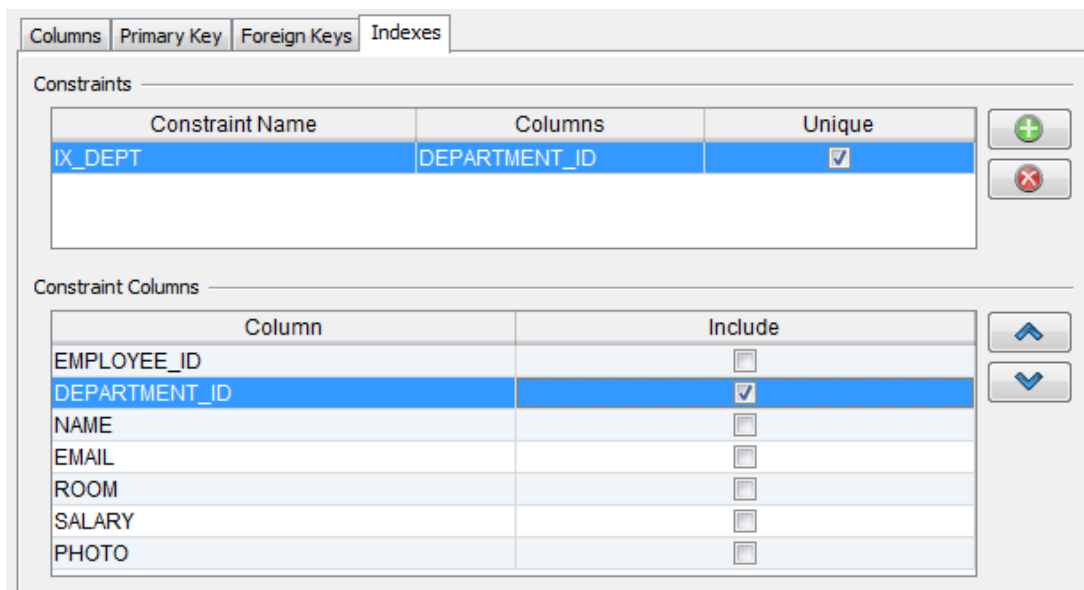


Figure: Indexes tab

The top portion of the tab holds a list of all indexes, and the lower portion holds a list of all table columns.

To create an index, click the **Add** button and optionally enter a name in the Constraint Name field. The Columns field in the constraints list is read-only, filled automatically as you include columns in the constrain. If you want the index columns to have unique values for all rows in the table, click the checkbox in the Unique field.

Select the columns to be part of the index by clicking the checkboxes in the Include field in the columns list. You can change the order of the columns in the constraint by selecting a column and move it using the **Up** and **Down** buttons.

To remove an existing constraint, select it in the list in the top section and click the **Remove** button.

SQL Preview

The SQL Preview area is updated automatically to match the edits made in the assistant. The preview is read only, but you can copy the SQL to the SQL Commander and flip between formatted and unformatted views using the two buttons in the toolbar above the preview area.

Execute

When you are satisfied with the table declaration, click the **Execute** button to create it.

Alter Table

To alter a table, select the table node in the objects tree and launch the Alter Table assistant from the **Database->Selected Object** main menu, the **Database Objects** tree right-click menu, or from the **Actions** menu button in the object view.

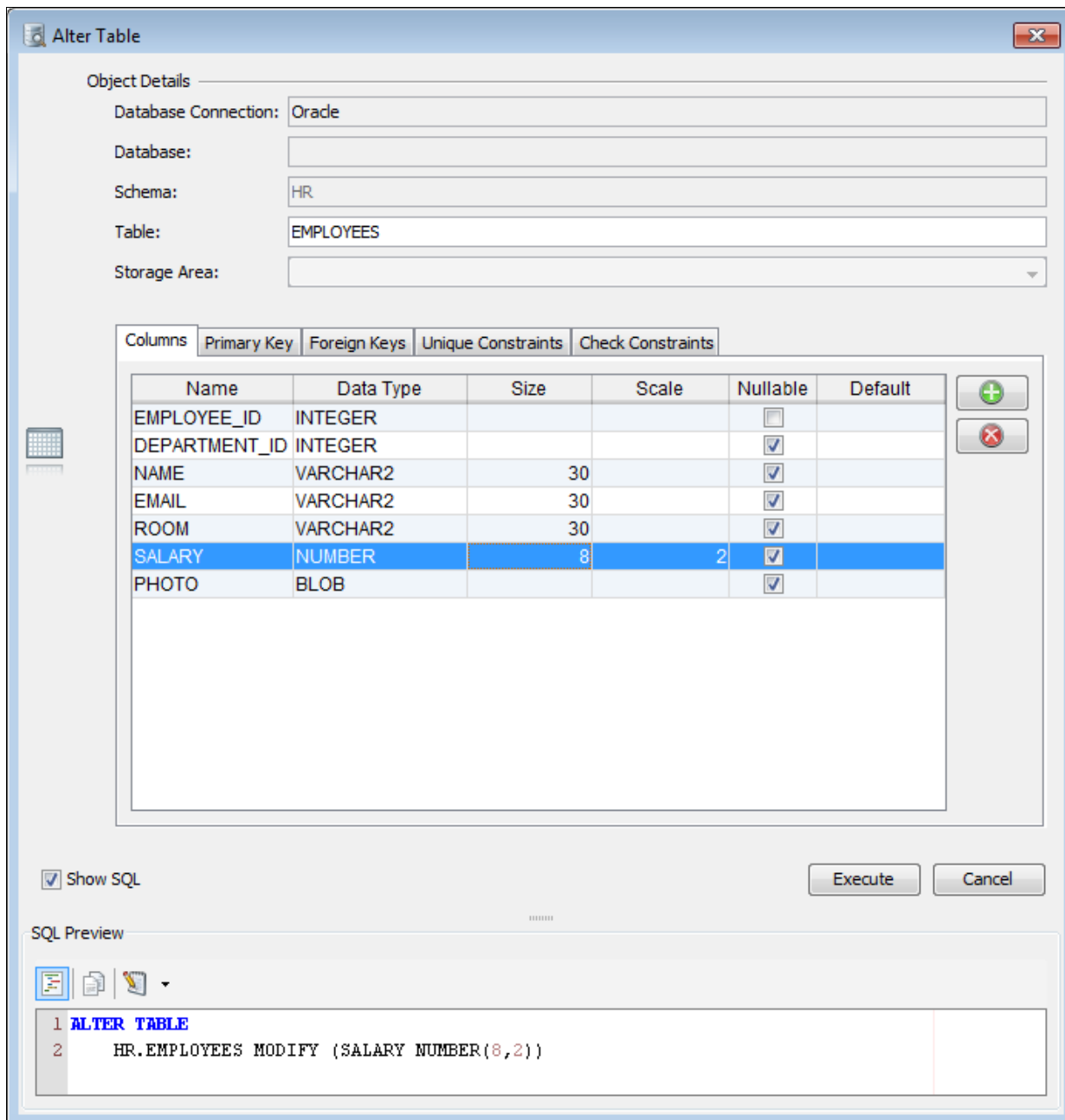


Figure: The Alter Table assistant

The Alter Table assistant has exactly the same layout as the Create Table assistant, with all information about the table you wish to alter shown when you launch it. As you make changes, such as adding a column, the SQL Preview area shows the corresponding ALTER TABLE statements. See the [Create Table section](#) for descriptions of all parts of the assistant.

The controls, such as the fields, pull-down menus and buttons, in the assistant are only enabled if the ALTER TABLE statement for the database holding the table provides a way to alter the corresponding table attribute. For instance, for a database that only allows the size of a VARCHAR column to be altered, the Size field in the Columns tab is disabled for all columns with other data types. If you find that you can not make the change you want, it is because the ALTER TABLE statement does not allow that change to be made.

Edit Table Data

Introduction

The table data editing feature mimics how editing is performed in standard spreadsheet applications; just click a cell value and edit. Edits are saved in a single database transaction which ensures that all or no changes are committed. The editing feature supports saving binary and large text data and it even presents common data formats in their respective viewers, such as image viewer, XML, HEX, etc.

A block of data can easily be interchanged via standard copy and paste operations between the grid editor and other applications, such as Microsoft Excel, StartOffice and OpenOffice.

Editing is primarily performed in the grid editor. For some data, such as binary or large formatted text data, editing in the grid editor is not optimal, so for these situations, we recommend that you to use the form or cell data editors. The form editor presents a single row of data in a separate window, organized as a form with the column name in the first column and the data in the second column. All editing capabilities in the grid editor are also available in the form editor. The cell editor is used to edit a single cell value in a separate window. This is useful when editing formatted text data or to browse binary data.

Most of the editing functions have a key binding assigned. Check the right-click menu in the data grid to find out what they are.

Features that support editing

Editing of table data can be performed in the **Database Objects->Data** tab and in a result set generated by a SELECT statement in the SQL Commander.

There are a few conditions that must be fulfilled for editing to be enabled in the SQL Commander:

1. It is a result set
2. The SQL is a SELECT command
3. Only one table is referenced in the FROM clause
4. All current columns exist by name (case sensitive) in the identified table

The editing tool bar is hidden if these rules are not met.

Update and Delete must match one table row




The editing features in DbVisualizer ensure that only one row in the table will be affected by update and delete edits. This prevents the user from doing changes in one row that might also silently affect data in other rows. DbVisualizer uses the following strategies to determine the uniqueness of the edited row:

1. Primary Key
2. Unique Index
3. Manually Selected Columns

The Primary Key concept is widely used in databases to uniquely identify the key columns in tables. If the table has a primary key, DbVisualizer will use it. There are situations when primary keys are not supported by a database or when primary keys are supported but not used. If no primary key is defined, DbVisualizer will check if there is a unique index. If there are several unique indexes, DbVisualizer will pick one of them. If there is no primary key or any unique indexes defined for the table, you need to manually choose what columns to use. The key column chooser is automatically displayed if the key columns can't be determined automatically.

Edit Multiple Rows

The grid editor supports editing multiple rows and saving all changes in one database transaction. Edited rows are indicated with an icon in the row header:

-  Cell(s) in the row has been edited
-  Row is new
-  Row is duplicated from another row

✘ Row is marked for deletion (edit is not allowed)

Edits are saved when explicitly saving changes via the **Edit Table Data->Save Edit(s)** right click menu choice or via the **Save** button in the tool bar.

Data Type checking

When leaving an edited cell the new value is validated with the data type for the column. If there is an error, the following dialog is displayed.

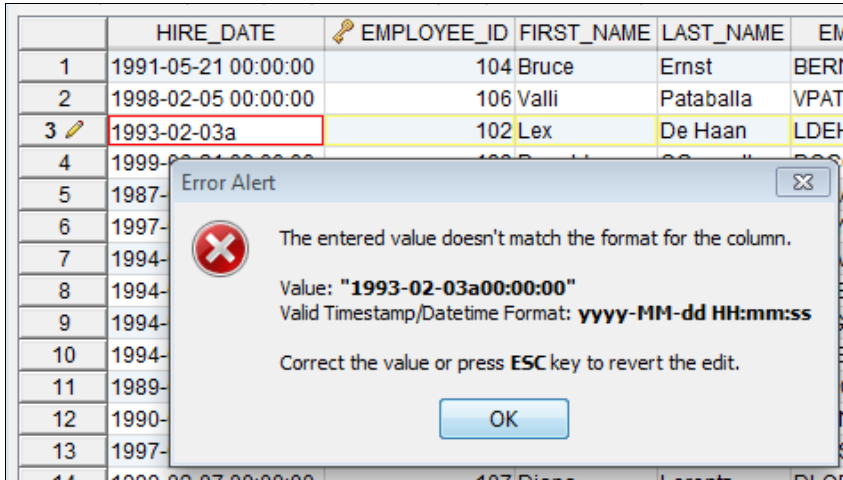


Figure: Data type error

New Line and Carriage Return

If a cell in the grid editor or form editor contains new line, carriage return or tab characters, these are not visually represented in the grid. Instead a warning will be displayed whenever you try to edit such value:

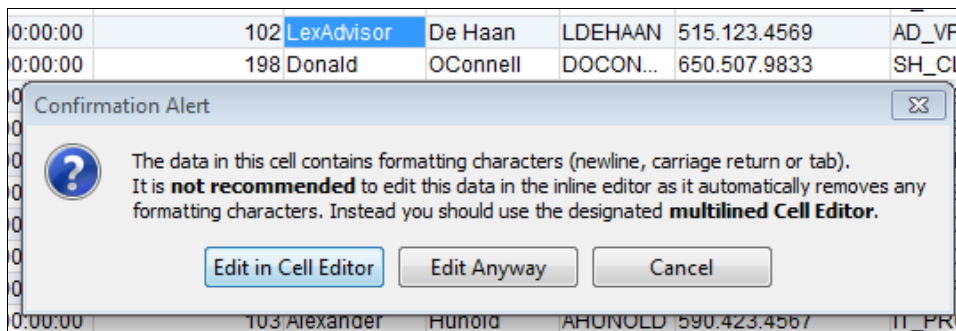


Figure: Warning when editing a value containing a carriage return, line feed and tab characters

You may chose to edit the value in the **Cell Editor**, which we recommend, as the control characters will then be preserved. The other choice is to edit the value anyway and risk loosing the control characters. This is **not recommended**.

The **Cell Editor** is a designated multi-line text editor suitable for editing large chunks of text:

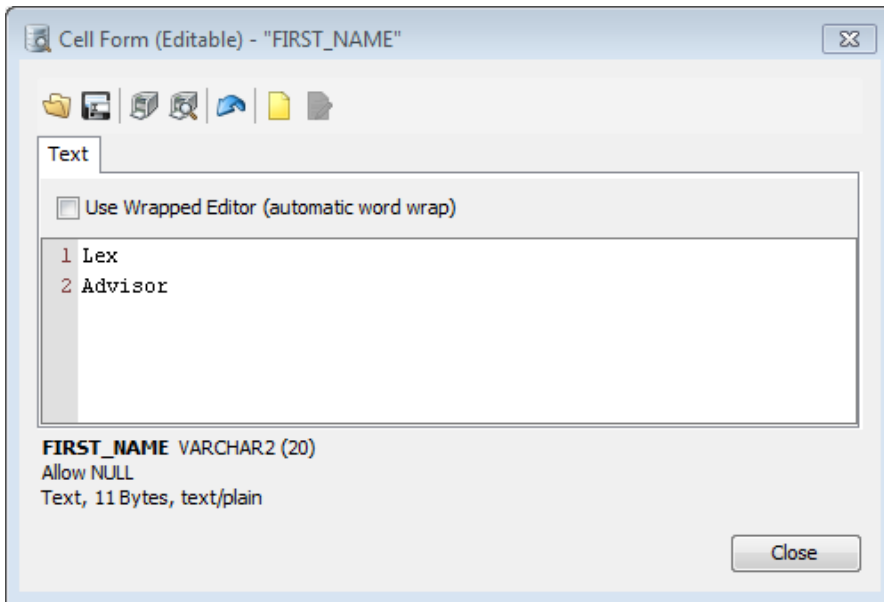


Figure: Editing multi lined text data in the cell editor

Grid Editor

The grid editor tool bar is decorated with buttons for editing and the right-click menu contains all related operations.

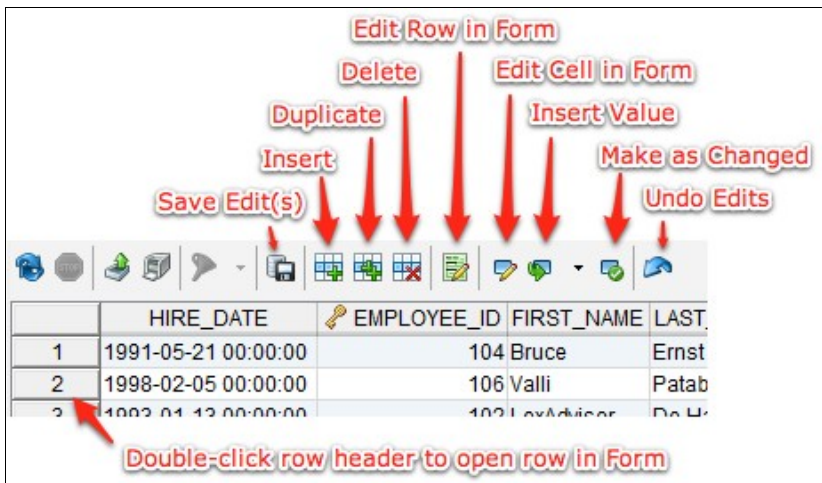


Figure: Toolbar buttons to control the grid editor

Cells that have been edited are indicated by a yellow background color. Only these columns will be updated when the changes are saved in the database. All cells in the edited row are highlighted with a yellow border to indicate that some cell(s) in the row has been edited.

Insert row

To insert a new row, choose the **Edit Table Data->Insert New Row** right-click menu choice or press the **Insert** toolbar button. The new row will be inserted after the selected row or at the top if no row is selected. You can now start editing the cells in the grid or open the form editor to insert values.

Update row

To update a row, just double-click in the target cell and modify the value.

Delete row(s)

One or multiple selected rows are marked for deletion via the **Edit Table Data->Delete Row(s)** right-click menu choice or by pressing the **Delete** toolbar button. Each deleted row will be highlighted with a red background color and no further editing of the content is allowed.

Deleting a row that has been updated will automatically undo all edits and show the original values. This is done so that it is obvious which data will be deleted. Deleting a row that has been inserted (or via duplication) will be removed from the grid.

Duplicate row(s)

Duplicate a row or several rows by selecting the cells in the rows that should be duplicated, then choose **Edit Table Data->Duplicate Row(s)** or press the **Duplicate** tool bar button. All cells in the new row will be marked as being edited (yellow background color). The exception is any **Auto Increment** field, which should be assigned a value by the database.

Copy/Paste

Copying selected cell values is accomplished via the **Copy Selection** right-click menu choice. The data on the clipboard may then be pasted either into DbVisualizer or any external application. The copy and paste operations in the grid editor are defined by the **Grid->Copy** category in Tool Properties. The default setting for column and newline delimiters are sufficient for most uses.

Paste data from Excel and OpenOffice

The grid editor supports pasting data from the major spreadsheet applications. The grid editor support pasting single data as well as block of data.

Copy from Excel

A single cell is copied in Excel

	A	B	C	D
1	98	Roger	Wruck	jsewell@hotmail.co
2	104	Steeve	Heer	gertveters@hotmail
3	105	Marc	Meller	mintegen@hotmail.
4	102	Bernhard	Kessler	brian@yahoo.com
5	193	Luci	Young	stephen@osp.edu
6	185	Robert	Austad	kathleen@bto.com
7	183	Pawan	Yucel	michael@ware.de
8	181	Hong	Graff	mail1235@yahoo.d
9	174	Rick	Stewart	jay@tury.com
10	171	Michael	Palm	mike@gmail.com
11	154	Peter	Pullabhotla	dsimmons@at123.i

Paste into DbVisualizer Grid

The selected cell is pasted into one selected target cell

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	104	Bruce	Ernst
2	106	Valli	Pataballa
3	102	LexAdvisor	De Haan
4	198	Donald	Kessler
5	200	Jennifer	Whalen
6	202	Pat	Fay
7	203	Susan	Mavris
8	204	Hermann	Baer
9	205	Shelley	Higgins
10	206	William	Gietz
11	101	Neena	Kochhar

A single cell is copied in Excel

The selected cell is pasted into multi selected target cells

=
=

	A	B	C	D
1	98	Roger	Wruck	jsewell@hotmail.co
2	104	Steeve	Heer	gertveters@hotmail
3	105	Marc	Meller	mintegen@hotmail.
4	102	Bernhard	Kessler	brian@yahoo.com
5	193	Luci	Young	stephen@osp.edu
6	185	Robert	Austad	kathleen@bto.com
7	183	Pawan	Yucel	michael@ware.de
8	181	Hong	Graff	mail1235@yahoo.d
9	174	Rick	Stewart	jay@tury.com
10	171	Michael	Palm	mike@gmail.com
11	154	Peter	Pullabhotla	dsimmons@at123.i

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	104	Bruce	Ernst
2	106	Valli	Pataballa
3	102	LexAdvisor	De Haan
4	198	Donald	Kessler
5	200	Jennifer	Kessler
6	202	Pat	Kessler
7	203	Susan	Kessler
8	204	Hermann	Kessler
9	205	Shelley	Higgins
10	206	William	Gietz
11	101	Neena	Kochhar

A block of cells is copied from Excel

	A	B	C	D
1	98	Roger	Wruck	jsewell@hotmail.co
2	104	Steeve	Heer	gertveters@hotmail
3	105	Marc	Meller	mintegen@hotmail.
4	102	Bernhard	Kessler	brian@yahoo.com
5	193	Luci	Young	stephen@osp.edu
6	185	Robert	Austad	kathleen@bto.com
7	183	Pawan	Yucel	michael@ware.de
8	181	Hong	Graff	mail1235@yahoo.d
9	174	Rick	Stewart	jay@tury.com
10	171	Michael	Palm	mike@gmail.com
11	154	Peter	Pullabhotla	dsimmons@at123.i

The block is pasted into the selected region

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	104	Bruce	Ernst
2	106	Valli	Pataballa
3	102	LexAdvisor	De Haan
4	198	Marc	Meller
5	200	Bernhard	Kessler
6	202	Luci	Young
7	203	Robert	Austad
8	204	Hermann	Baer
9	205	Shelley	Higgins
10	206	William	Gietz
11	101	Neena	Kochhar

A block of cells copied from Excel

	A	B	C	D
1	98	Roger	Wruck	jsewell@hotmail.co
2	104	Steeve	Heer	gertveters@hotmail
3	105	Marc	Meller	mintegen@hotmail.
4	102	Bernhard	Kessler	brian@yahoo.com
5	193	Luci	Young	stephen@osp.edu
6	185	Robert	Austad	kathleen@bto.com
7	183	Pawan	Yucel	michael@ware.de
8	181	Hong	Graff	mail1235@yahoo.d
9	174	Rick	Stewart	jay@tury.com
10	171	Michael	Palm	mike@gmail.com
11	154	Peter	Pullabhotla	dsimmons@at123.i

The block is pasted into a non equal number of target cells

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NU
1	104	Bruce	Ernst	BERNST	590.423.4568
2	106	Valli	Pataballa	VPATABAL	590.423.4560
3	102	LexAdvisor	De Haan	LDEHAAN	515.123.4569
4	198	Donald	OConnell	DOCONNEL	650.507.9833

Notification Alert

You have requested to paste 4 rows into the selection of 2 target rows. Do you want to **Add Rows** in the grid so that all rows in the clipboard are pasted?

Insert pre-defined values (Set Selected Cells)

The **Edit Table Data->Set Selected Cells** right-click menu choice or the **Set Selected Cells** tool bar button lists a few pre-defined functions that will fill the selected cells with data.

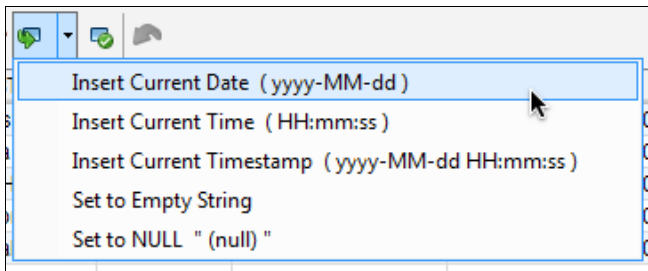


Figure: Set Selected cells functions

Use these to insert data into the selected cells. Note that the target column type must accept the selected value type; nothing will happen if you choose, for example, "Insert Current Time" for a DATE data type cell.

Undo Edit(s)

The **Edit Table Data->Undo Edit(s)** operation is used to revert all edits in the selected cell(s). Reverting all cells in a row that are marked as **Insert** or **Duplicate** will remove the complete row from the grid while a **Delete** marked row is cleared from its delete state. Undoing updated cells simply reverts the changes to the original values.

Key Column(s) Chooser

Normally database tables have a primary key or at least one unique index. If this is the case, editing is no problem. If there is no way to uniquely identify rows in the table, you need to manually define what columns DbVisualizer should use. While saving the changes DbVisualizer will check that there is a way to identify unique rows in the table. If it cannot be accomplished, the following window is displayed.

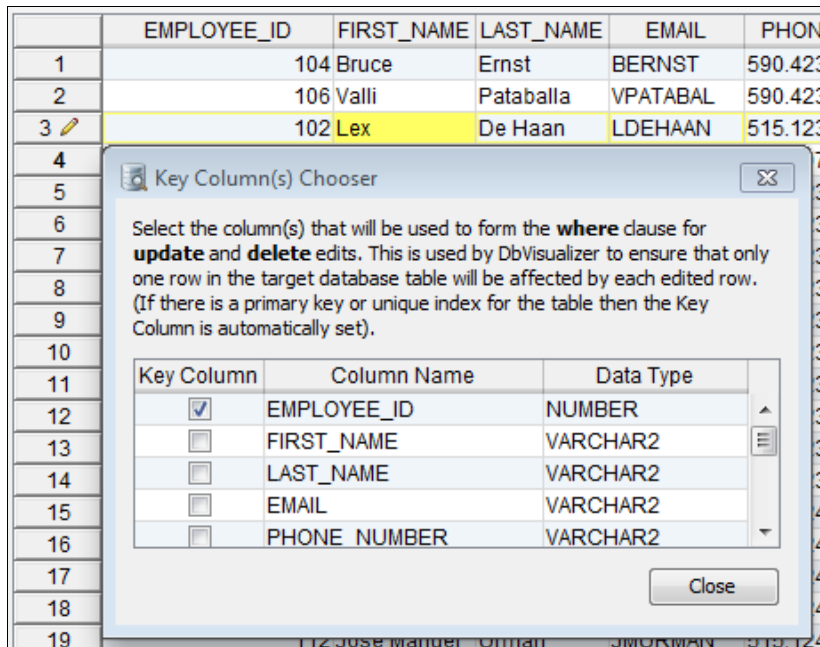


Figure: Key Column(s) Chooser

The key column chooser can be manually opened via the **Edit Table Data->Key Column Chooser** right-click menu choice.

If the database request to save the edits cannot uniquely identify the single row that should be changed, the error dialog is displayed and the editing state is kept for that row in the grid editor.

Preview Changes

You may preview the SQL statements that will be executed when choosing to **Save** the edits. It is displayed via the **Edit Table Data->SQL Preview** right-click menu choice.

2		100	Van	Patricia	VATPAAL	590.423.4300	1998-02-03 00:00:00	AD_VP	
3		102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00	AD_VP	
4		198	Ronald	OConnell	DOCONNEL	650.507.9833	1999-06-21 00:00:00	SH_CLERK	
5		200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17 00:00:00	AD_ASST	
6	✗	202	Pat	Fay	PFAY	603.123.6666	1997-08-17 00:00:00	MK_REP	
7	+	45	Pat	Benatar	pat.benatar	N/A	2009-12-16 16:18:42	(null)	
8		203	Susan	Mavris	SMAVRIS	515.123.7777	1994-06-07 00:00:00	HR_REP	
9	✗	204	Hermann	Baer	HBAER	515.123.8888	1994-06-07 00:00:00	PR_REP	
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									

SQL Preview

This is a preview of the SQL that will be executed when the table data edit(s) are saved.

```
1 update "HR"."BIO" set "FIRST_NAME" = 'Lex' where "EMPLOYEE_ID" = 102;
2 update "HR"."BIO" set "FIRST_NAME" = 'Ronald' where "EMPLOYEE_ID" = 198;
3 delete from "HR"."BIO" where "EMPLOYEE_ID" = 202;
4 insert into "HR"."BIO" ("EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "EMAIL", "PHONE_NUMBER", "HIRE
5 delete from "HR"."BIO" where "EMPLOYEE_ID" = 204
```

Close

Figure: SQL Preview

(The listed SQL statements may not be 100% compliant as the save process use variable binding to pass values to the database).

Saving Changes

To save table data edits, select the **Edit Table Data->Save Edit(s)** right-click menu choice or click the **Save** toolbar button. If there are rows that have been edited or deleted, these are first checked so that there is only one database table row affected by each edited row. If this pass is successful, DbVisualizer will save the changes in the table. The progress is displayed in the status bar and Save may be interrupted by pressing the **Stop** button in the toolbar. While save is in progress, no other operation may be performed in DbVisualizer, i.e., the rest of the application is locked.

Transaction Control

DbVisualizer use the physical root connection for the actual database connection when saving table data. Once save is requested, DbVisualizer will implicitly set the auto commit state to off and reset it to what it was prior to requesting save when saving is completed. If the **Use Single Shared Physical Database Connection** is enabled in connection properties, DbVisualizer will check whether there are any uncommitted updates in the database when save is requested. If there are uncommitted changes you must first commit or rollback these changes before save is started.

Saving table data edits are batched in a single transaction. There is no DbVisualizer restriction on the number of edits that may be saved in a single save operation, but the database server may put either explicit or implicit restrictions. The connection property **Physical Connection->Transaction->Commit Batch Size** specifies how many edits should be performed in the database table until commit is automatically initiated. If you, for example, are saving 150 edited rows and an error occurs while saving the 121:st row, then the first 100 rows will have been committed and the rest are left unchanged. The visual indication in the grid after a incomplete save operation is that rows that weren't saved keep their original editing state indicator. Rows that were saved properly are indicated with a green checked cylinder icon.

	🔑	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	🔑	104	Bruce1	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00
2	🔑	106	Valli1	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00
3	🔑	102	Lexe1	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
4	+	102	Lexe	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
5	+	102	Lexe	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
6	+	102	Lexe	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
7	+	102	Lexe	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00
8	🔑	198	Donald1	OConnell	DOCONNEL	650.507.9833	1999-06-21 00:00:00
9		200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17 00:00:00
10	+	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17 00:00:00
11		202	Pat	Fay	PFAY	603.123.6666	1997-08-17 00:00:00
12		203	Susan	Mavris	SMAVRIS	515.123.7777	1994-06-07 00:00:00
13		204	Hermann	Baer	HBAER	515.123.8888	1994-06-07 00:00:00

Figure: Saved rows state

The cylinder icon with the green check mark indicates that the row was saved in the database table. Normally the grid is reloaded after a successful save operation and there is no cylinder icon displayed. It only appears if the save operation was partly successful. Rows that weren't saved are still represented with the original editing state icon and you may request save one more time.

Rows that have been properly saved (indicated with the cylinder icon) cannot be edited until all rows are saved properly or the grid is reloaded.

Permissions

All of the insert, update and delete requests performed by the grid editor may require confirmation before being executed by the database server. Specify in **Tool Properties->Permissions** which operations should require confirmation. The default behavior is that delete operations must be confirmed while insert and update need no confirmation.

Errors

If a database error occurs while saving changes, details about the errors are displayed in a window along with the actual SQL that was executed.

Form Editor/Viewer

The **Form Viewer** is available in the right-click menu (**Browse Row in Form**) for all grids in DbVisualizer. It is used to browse information and to present binary data in viewers.

The **Form Editor** adds editing capability to the form viewer. This editor is useful when inserting new rows and when it is important to get a more balanced overview of all the data.

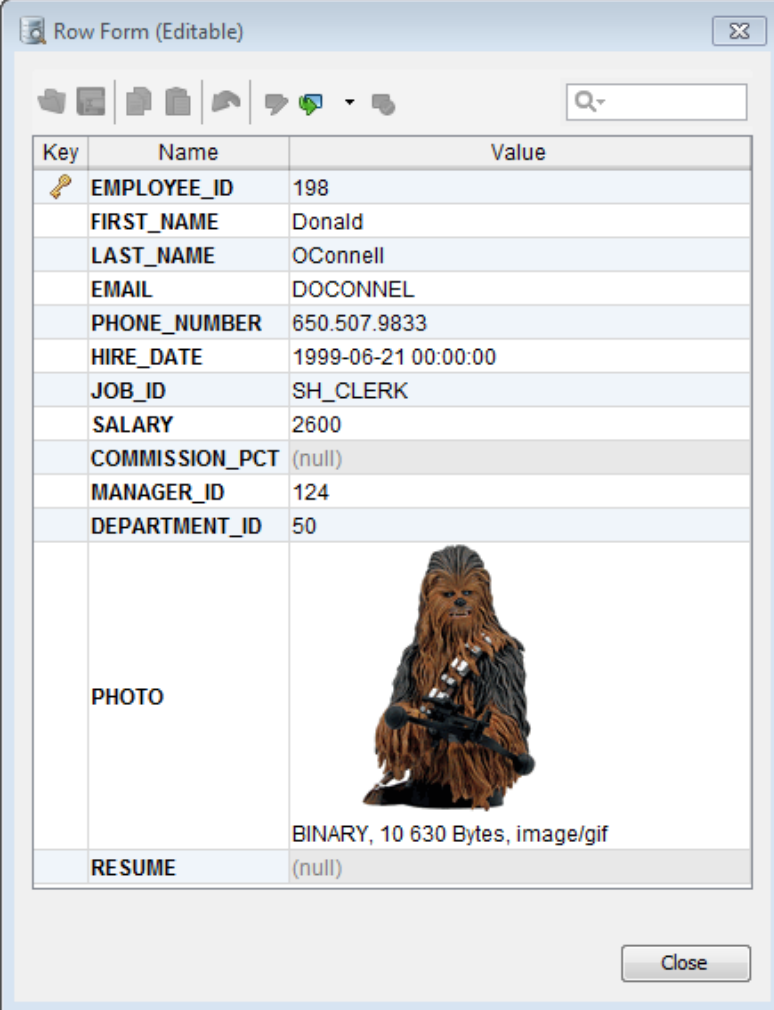
The form editor "rotate" the data in one row and presents it as a vertical form with the column name as a label. All edits made in the form editor are reflected in the grid with the edited state icon being updated along with new values. Saving edits in the database is always done with the **Save Edit(s)** control in the original grid editor.

Open the form editor via the **Edit Row in Form** right-click menu choice, via the button in the toolbar or by double-clicking the row number header.

	🔑	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	PHOTO
1		104	Bruce	Ernst	BERNST	590.423.4568	(null)
2		106	Valli	Pataballa	VPATABAL	590.423.4560	(null)
3		102	Lexe	De Haan	LDEHAAN	515.123.4569	🔑 BINARY, 5 888 Bytes
4		198	Donald	OConnell	DOCON...	650.507.9833	🔑 BINARY, 10 630 Bytes
5		200	Jennifer	Whalen	JWHALEN	515.123.4444	(null)
6		202	Pat	Fay	PFAY	603.123.6666	(null)
7		203	Susan	Mavris	SMAVRIS	515.123.7777	🔑 BINARY, 8 620 Bytes
8		204	Hermann	Baer	HBAER	515.123.8888	🔑 BINARY, 8 719 Bytes
9		205	Shelley	Higgins	SHIGGINS	515.123.8080	(null)

Figure: A row in the grid

Here is the same row as selected in the previous screen shot displayed in the row form window.





Key	Name	Value
	EMPLOYEE_ID	198
	FIRST_NAME	Donald
	LAST_NAME	OConnell
	EMAIL	DOCONNEL
	PHONE_NUMBER	650.507.9833
	HIRE_DATE	1999-06-21 00:00:00
	JOB_ID	SH_CLERK
	SALARY	2600
	COMMISSION_PCT	(null)
	MANAGER_ID	124
	DEPARTMENT_ID	50
	PHOTO	 BINARY, 10 630 Bytes, image/gif
	RESUME	(null)

Figure: Form viewer

The **Key** field optionally contains an icon for primary key columns and the **Name** field corresponds to the column name in the grid. None of Key or the Name fields can be edited. You can edit the values in the form in the same way as you edit values in the grid editor.

The form viewer presents images as thumbnails. The size of these is controlled in **Tool Properties->Form Viewer->Image Thumbnail Size**. To see the original size of an image, open the cell in the cell viewer either by selecting **Edit in Cell Window** in the right-click menu, the toolbar button or by double-clicking on the image.

Cell Editor/Viewer

The **Cell Viewer** is available in the right-click menu for all grids in DbVisualizer. It presents the data for a single cell (column in a row) in a window. If the data is of a recognized type, it is presented by a corresponding viewer:

- Image viewer
- XML viewer
- Serialized Java object viewer
- Hex viewer
- Text viewer

The cell viewer allows saving data to a file and to print it.

The **Cell Editor** adds editing capability to the cell viewer. You may import data from a file or manually change the text in a text editor.

Opening an image in the cell editor will display the following window.

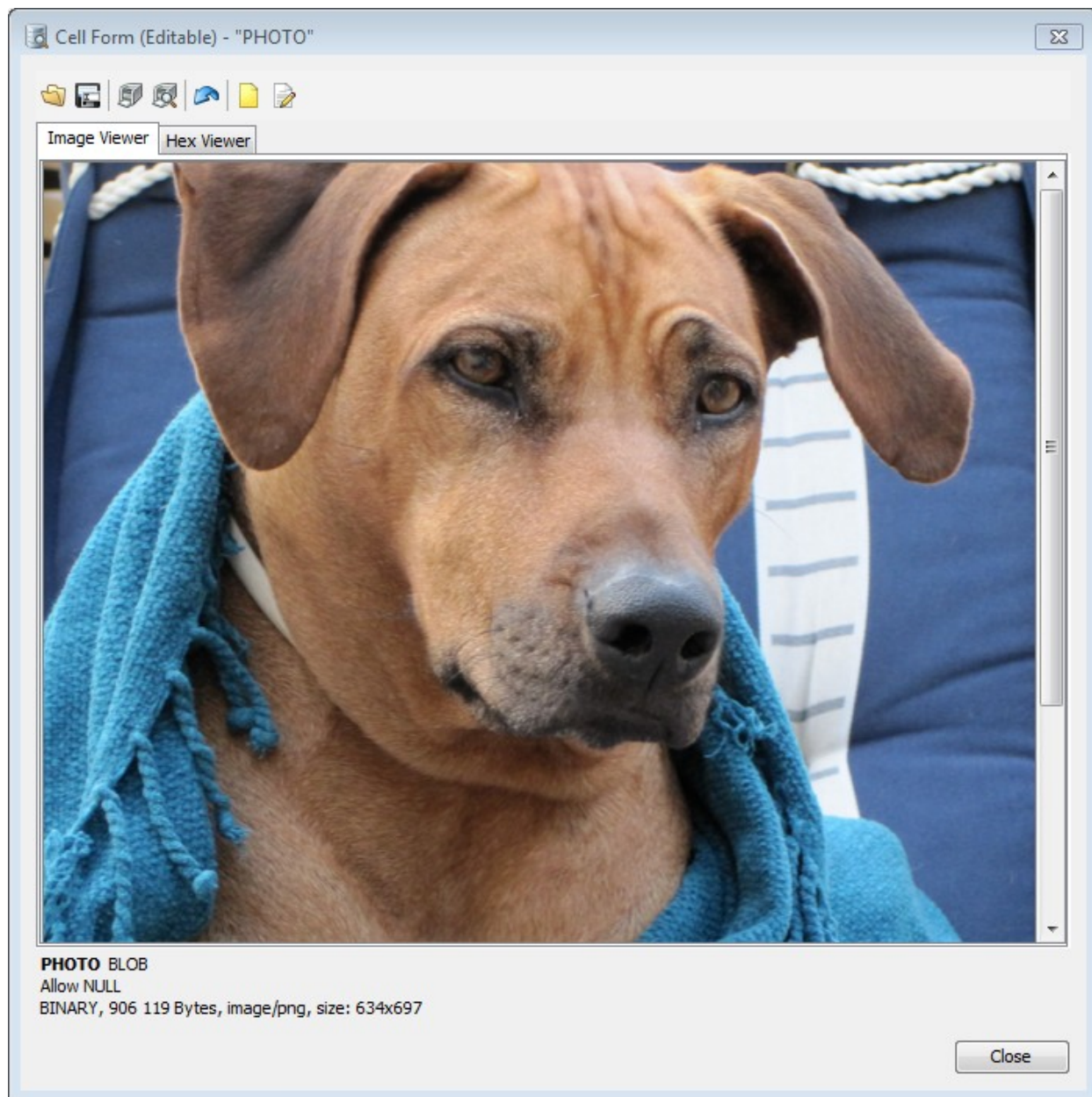


Figure: Cell editor

The tabs at the top shows the available viewers for the current data. When you load a file into the cell editor, the tabs may change to reflect the newly loaded data. To nullify the cell value, press the **Set Value to NULL** toolbar button.

Read more about binary and formatted text data in the following chapters.

Binary/BLOB

Due to the nature of binary/BLOB and CLOB data, cells of these types can only be fully modified and viewed in the cell editor. (There is partial support in the form editor to view image data and to load from file). Editing binary data can be done by importing from a file or via the text editor.

Binary data in DbVisualizer is the generic term for several common binary database types:

- LONGVARBINARY
- BINARY
- VARBINARY

- BLOB

Image Viewers

The image viewer supports displaying the full size images for the following formats:

- GIF
- JPG
- PNG
- TIFF
- BMP

XML Viewer

The XML Viewer shows the content of an XML document in a tree with color highlighting. You can switch to an editable text view by pressing the **Edit value in text editor** toolbar button.

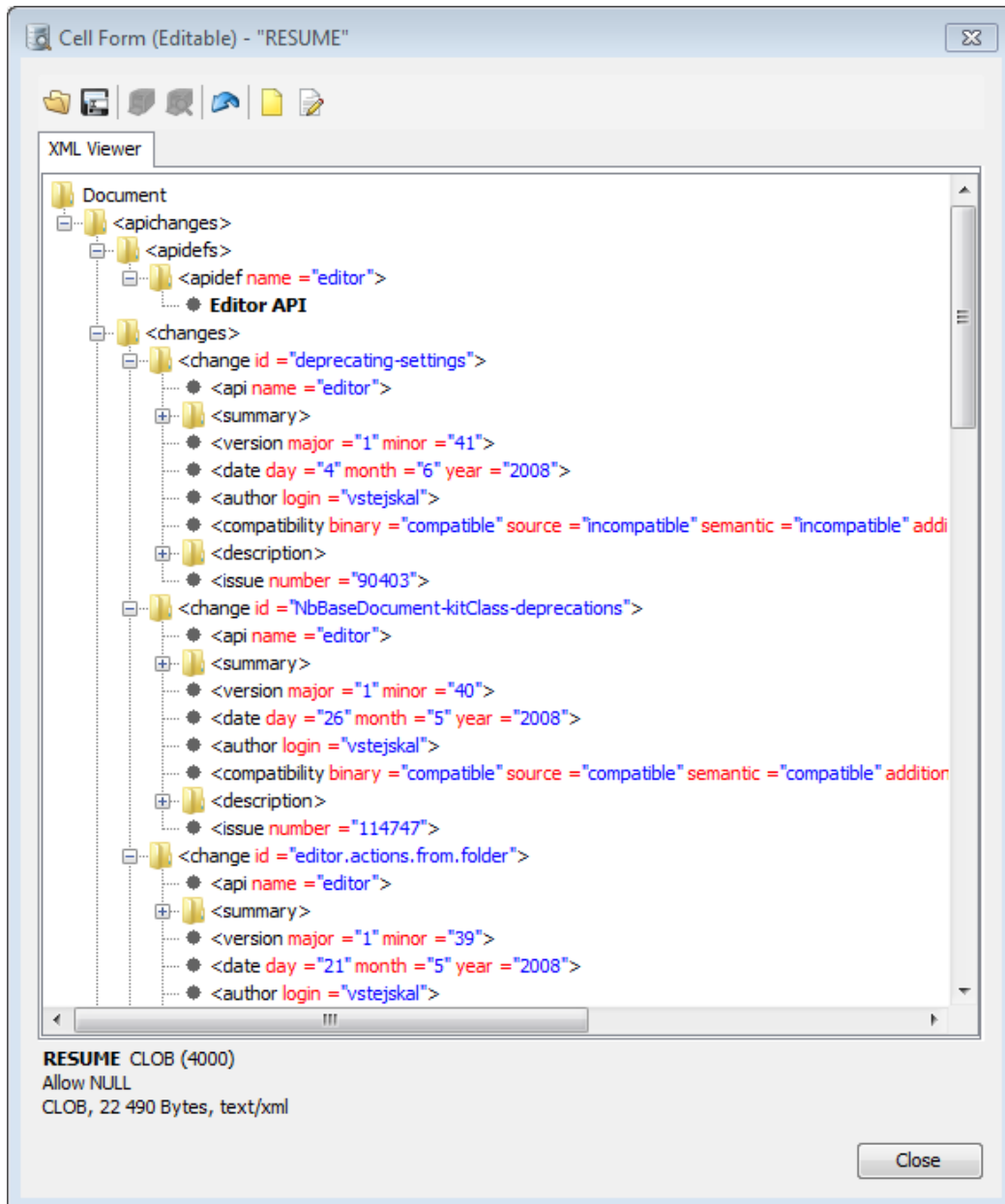


Figure: XML viewer

Serialized Java Objects Viewer

The serialized Java object viewer renders a java object in a tree style. All aspects about the object may be browsed.

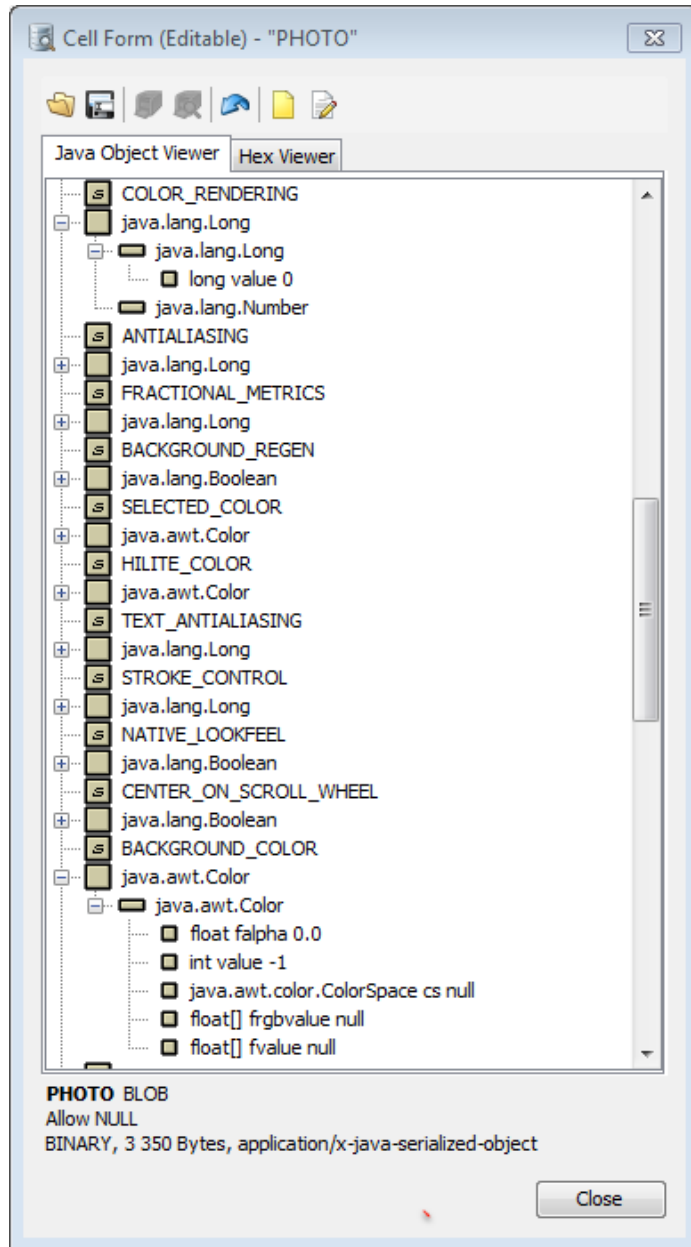


Figure: Binary data viewer for serialized Java objects

Hex Viewer

The generic Hex/ASCII viewer shows the hexadecimal representation of every byte in the data and its text representation. This is the default viewer for unknown data.

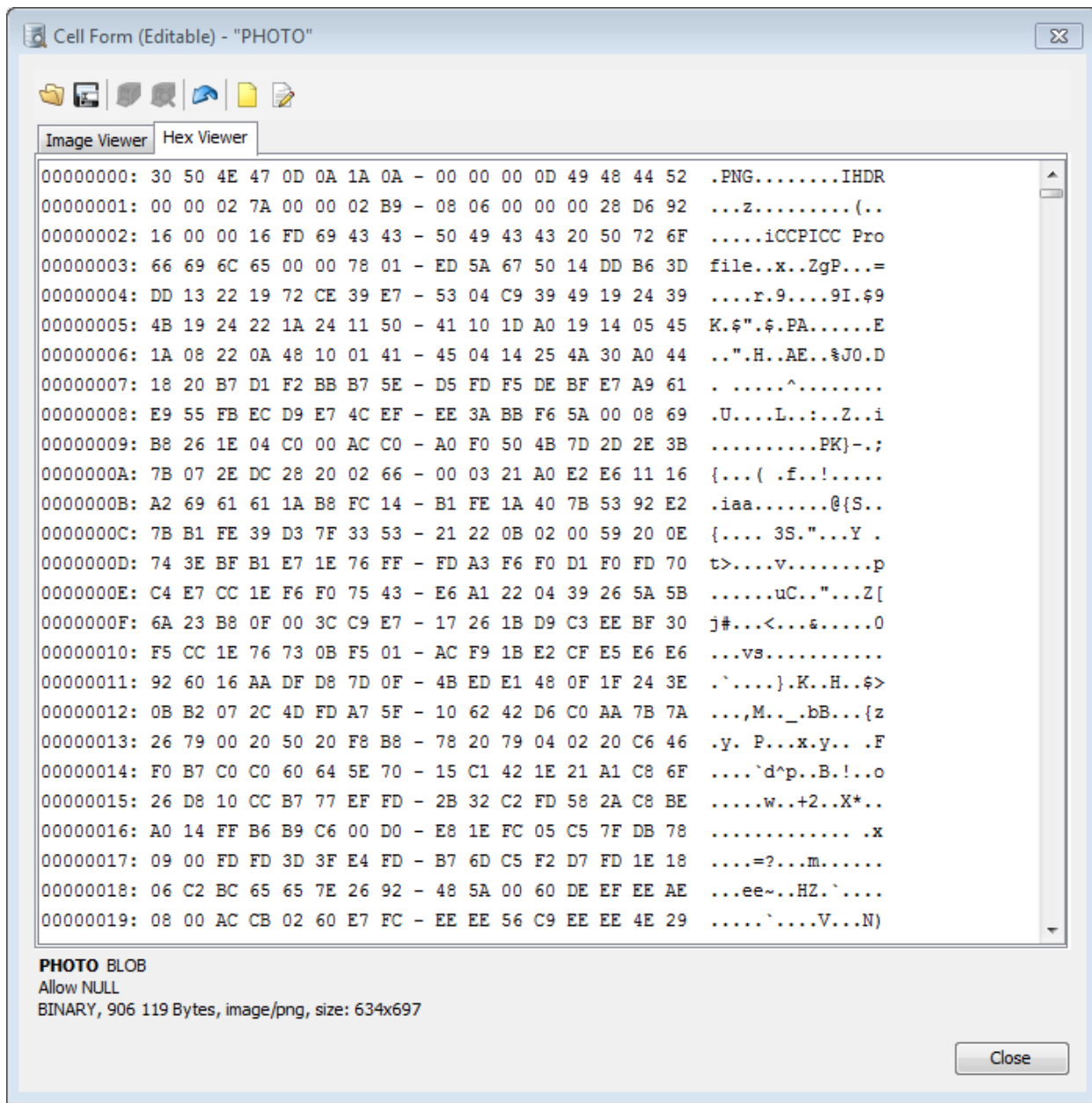


Figure: Hex/ASCII viewer

Large text data/CLOB

Large text data and CLOB data types are typically edited in the multi line text editor. For formatted data (that includes new lines), the default editor is useful. If editing a large chunk of non-formatted data, enable the **Use Wrapped Editor** setting and DbVisualizer will then automatically wrap the text for easy editing.

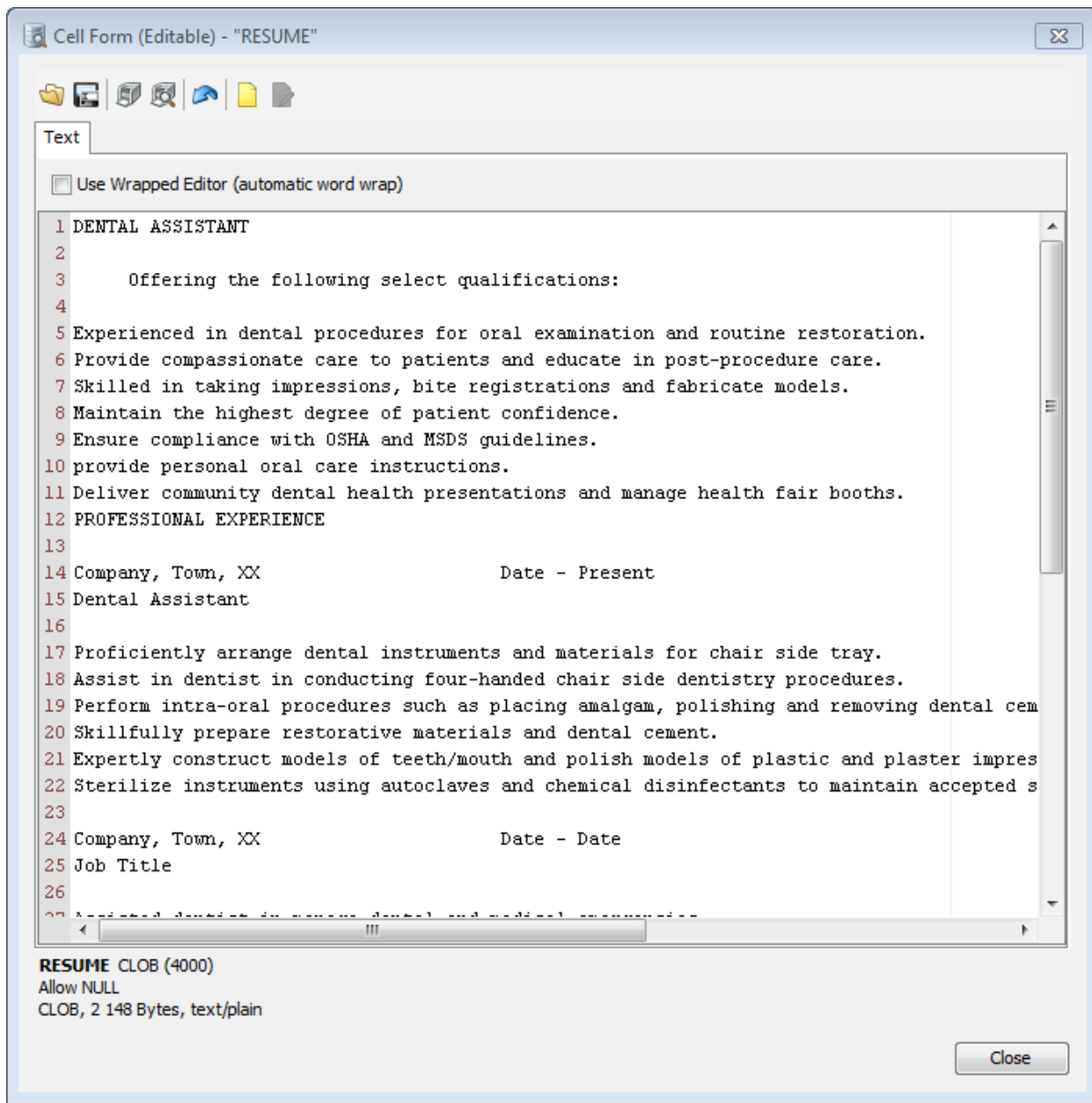


Figure: The text viewer and editor

Import from File

Importing data from a file can be done in the form and cell editors. Imported binary data of a recognized type is displayed by the corresponding binary viewer. Import is supported for both binary and text data.

Export to File

Export can be made in the grid, form and cell editors for binary and text data.

Table Data Navigation

Introduction

A powerful way to study database data is to navigate between the tables in a schema by following table relationships declared by primary and foreign keys. DbVisualizer includes a Navigator feature for this purpose, visualizing the relationships graphically while making the data for each navigation case easily accessible in a data grid.

To launch the **Navigator**, select the table you want to start the navigation from in the Database Objects Tree, and then open the Navigator tab in the Object View.

Table: DEPARTMENTS

Oracle | Schemas | HR | Tables | DEPARTMENTS

Grants | Columns Comment | Constraints | Triggers | Dependencies | DDL | DDL with Storage

Info | Columns | Data | Row Count | Primary Key | Indexes | Row Id | References | Navigator

DEPARTMENT_ID
DEPARTMENT_NAME Human Resources

HR.DEPARTMENTS

DEPARTMENT_ID
DEPARTMENT_NAME IT

HR.EMPLOYEES
DEPARTMENT_ID 40

HR.EMPLOYEES
DEPARTMENT_ID 60

Powered by yFiles

Related Table:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COM
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00	IT_PROG	9000	
2	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	IT_PROG	6000	
3	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG	4800	
4	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00	IT_PROG	4800	
5	107	Diana	Lorentz	DLOREN...	590.423.5567	1999-02-07 00:00:00	IT_PROG	4200	

Max Rows: 1000 | Max Chars: 0 | 0.000/0.016 sec | 5/11 | 1-5

Figure: The Navigator tab showing the initial table

The Navigator has two parts: a graphical view and a data grid. Initially, the graphical view shows just the selected start table, and the data grid shows the data for the start table.

The data grid is a read-only grid of the same type as you encounter in other parts of DbVisualizer, but extended with a Related Table list and a Tag button. You can learn more about the general data grid in the [Data Grid](#) section of the [Getting Started](#) chapter. The Navigator specific extensions are described in detail in the following section.

Data Navigation

Data navigation in DbVisualizer means following table relationships declared by primary and foreign keys, using a unique key value. In the example schema shown in the screenshots in this section, there is a table named DEPARTMENTS with a primary key named DEPARTMENT_ID. Another table named EMPLOYEES has a foreign key constraint, declaring that values in its DEPARTMENT_ID column refer to primary key values in the column with the same name in the DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	EMPLOYEE_COUNT	MANAGER_COUNT
1	10 Administrative Services	203	2400
2	20 Marketing	121	1500
3	30 Purchasing	103	1400
4	40 Human Resources	204	2700
5	50 Shipping	121	1500
6	60 IT	103	1400
7	70 Public Relations	204	2700

Figure: The Related Table list

If you use DEPARTMENTS as you start table, you can easily navigate to the EMPLOYEES table for different DEPARTMENT_ID values. In the data grid, select one or more columns in the row that holds the DEPARTMENT_ID you want to use for navigation. In the figure above, the DEPARTMENT_NAME column in the row for DEPARTMENT_ID = 60 is selected.

Next, bring up the Related Table list. It lists all tables the DEPARTMENTS table is related to through primary and foreign keys, with the key columns within parenthesis. A forward arrow (->) in front of the table name means that the DEPARTMENTS table has a foreign key relation to the named table. A backward arrow (<-) means that the named table has a foreign key relation to the DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	EMPLOYEE_COUNT	MANAGER_COUNT
1	10 Administrative Services	203	2400
2	20 Marketing	121	1500
3	30 Purchasing	103	1400
4	40 Human Resources	204	2700
5	50 Shipping	121	1500
6	60 IT	103	1400
7	70 Public Relations	204	2700

Figure: Tooltip for a Related Table list entry

The Related Table list shows only the table name and columns of the related table, because there is not room for more when a key contains many columns with long names. Sometimes this information is not enough to understand what the relation really means. To make it easier to figure out, you can let the mouse hover over a list entry. A tooltip then shows you the other end of the relation as well, e.g., in the figure above, the tooltip shows that "<- EMPLOYEES (DEPARTMENT_ID)" represents a foreign key from the EMPLOYEES DEPARTMENT_ID column to the DEPARTMENTS DEPARTMENT_ID column.

Table: DEPARTMENTS

Oracle | Schemas | HR | Tables | DEPARTMENTS

Grants Columns Comment Constraints Triggers Dependencies DDL DDL with Storage
 Info Columns Data Row Count Primary Key Indexes Row Id References Navigator

HR.DEPARTMENTS → DEPARTMENT_ID DEPARTMENT_NAME IT → HR.EMPLOYEES
 DEPARTMENT_ID 60

Powered by yFiles

Related Table:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COM
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00	IT_PROG	9000	
2	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	IT_PROG	6000	
3	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG	4800	
4	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00	IT_PROG	4800	
5	107	Diana	Lorentz	DLOREN...	590.423.5567	1999-02-07 00:00:00	IT_PROG	4200	

Max Rows: 1000 Max Chars: 0 0.000/0.016 sec 5/11 1-5

Figure: Navigation from DEPARTMENTS to EMPLOYEES for DEPARTMENT_ID = 60

When you select "<- EMPLOYEES (DEPARTMENT_ID)" in the Related Table list, a node is added to the graph for the EMPLOYEES table, with an arrow from the DEPARTMENTS table node to show the navigation direction. We call this a *navigation case*.

The EMPLOYEES node contains the key columns (just one in this example) and their values.

The arrow between the nodes is labeled with the key column name. In addition, the arrow label also shows the name and value of the column that you selected in the DEPARTMENTS table when you created this navigation case, i.e., the DEPARTMENT_NAME column. If you select multiple columns when you create a navigation case, all non-key column names and values are included in the arrow label. This can make it easier to see at a glance what a navigation case represents.

The grid is also updated when you create a navigation case, to show all rows in the table you navigated to that has a key value corresponding to the selected key value in the table you navigated from. In this case, it shows all rows in the EMPLOYEES table with DEPARTMENT_ID equal to 60.

You can continue to create more navigation cases from any node in the graph. For instance, if the schema contains a table with job history information for employees, you can navigate to the history for an employee from the EMPLOYEES node. Or, you can select the DEPARTMENTS node in the graph to navigate to the EMPLOYEES table for a different department. Just click on the DEPARTMENTS node, select another row in the data grid and then the same Related Table list entry.

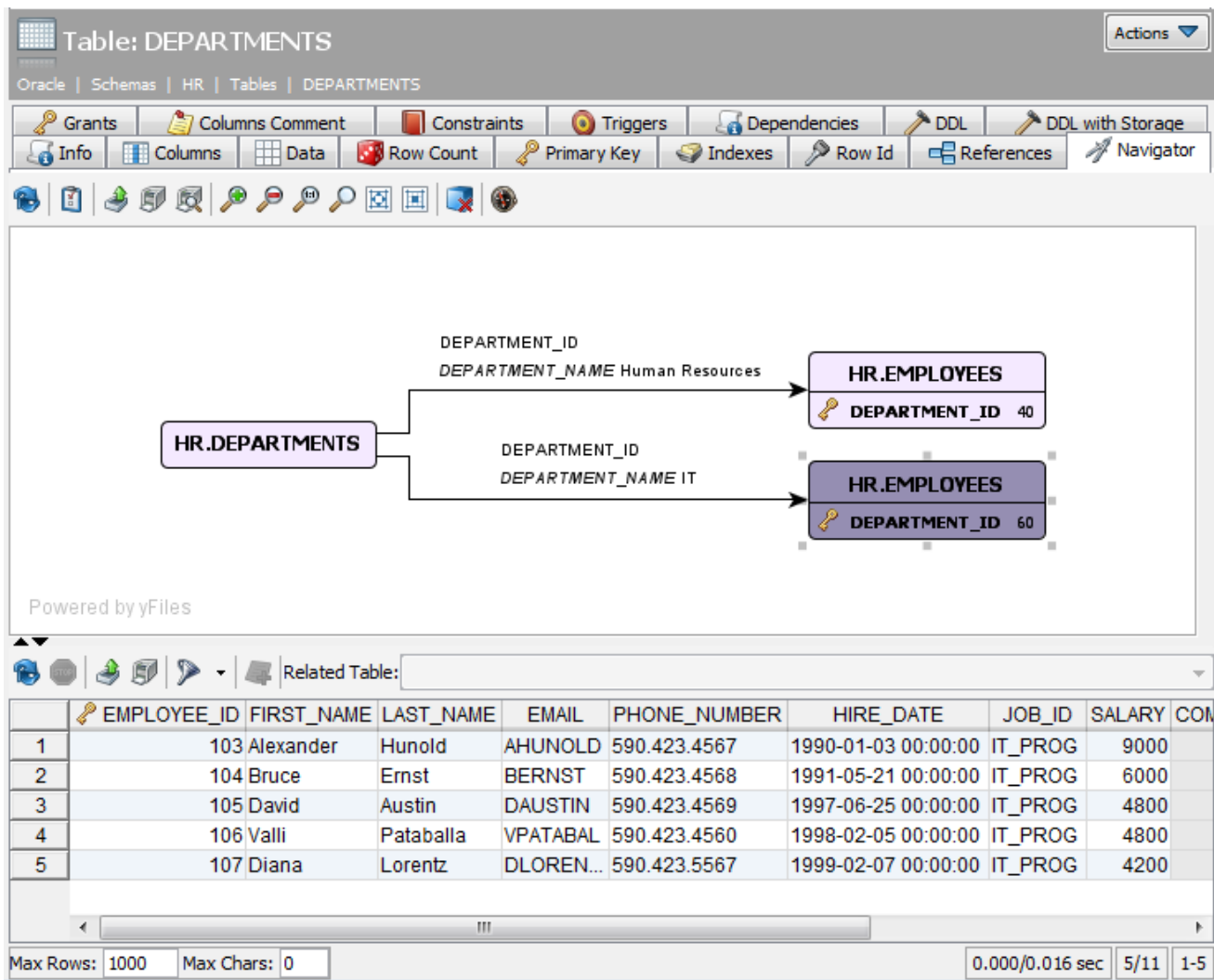


Figure: Two navigation cases

If you want to create multiple navigation cases from one table to another using the same relationship, you can select columns in multiple rows in the first table. When you make a selection in the Related Table list, one navigation case per row is created.

Every time you select a node in the graph, the data grid is updated to show the corresponding data. The grid settings for one node are independent of the settings for another node. For instance, if you define a filter for one node, the filter is only associated with the grid for that node.

Adding Context Information to the Graph

The navigation node always shows the key columns and their values, but sometimes you may want to add other columns to the node to better describe what it represents. This is called *tagging* the node. There are two ways to do so: drag and drop cells from the grid to any node, or use the **Tag** button in the grid toolbar to tag the currently selected node with the currently selected cells in the grid.

To drag and drop cells to a node, select one or more cells in the grid. With the left mouse button pressed and the mouse positioned over one of the selected cells, drag the cells over a node in the graph and release the mouse button. The cells are added to the node.

Table: DEPARTMENTS

Oracle | Schemas | HR | Tables | DEPARTMENTS

Grants Columns Comment Constraints Triggers Dependencies DDL DDL with Storage
 Info Columns Data Row Count Primary Key Indexes Row Id References Navigator

Powered by yFiles

Related Table:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMM
1	203	Susan	Mavris	SMAVRIS	515.123.7777	1994-06-07 00:00:00	HR_REP	6500	

Max Rows: 1000 Max Chars: 0 0.000/0.000 sec 1/11 1-1

Figure: A node tagged with additional column values

Alternatively, you can select the cells in the grid and click on the **Tag** button (📌) to add the cell values to the currently selected node.

Arranging the Graph

As you add navigation cases, you may find that you need to move nodes around, remove selected nodes, zoom and move around in the graph, etc.

You can rearrange the layout of the graph by selecting a node and, with the left mouse button pressed, drag it around. The arrow and its label moves with the node.

The toolbar for the graph offers a number of tools to help you with other tasks.










Figure: The graph toolbar

All these tasks can also be accessed through the graph popup menu.

🔄 Clicking the **Reload** button removes all navigation cases, leaving just the node for the table you started with.

📄 You use the **Show/Hide Controls** button to control the display of an Overview control, see below.

🔍 The **Zoom In** button lets you zoom into the graph, one step per click.

-  The **Zoom Out** button zooms the graph out one step with each click.
-  Clicking the **Zoom 100%** button zooms the graph so that all items are shown with their standard size.
-  Toggle the **Magnifying Mode**. When enabled, the content around the mouse pointer is magnified
-  Use the **Fit** button to make all graph items fit in the graph display area.
-  The **Relayout** button lays out all graph item with standard positions, distances between items, etc. This can be useful after making manual changes, such as removing nodes or tagging nodes.
-  The **Remove Node** button removes the selected node. It is only enabled when a navigation case node is selected.
-  Toggle between **Navigation** and **Edit Modes**. With Navigation Mode enabled, you can move the graph content with the left mouse button depressed.

The Overview control is useful for large graphs that do not fit into the display area.

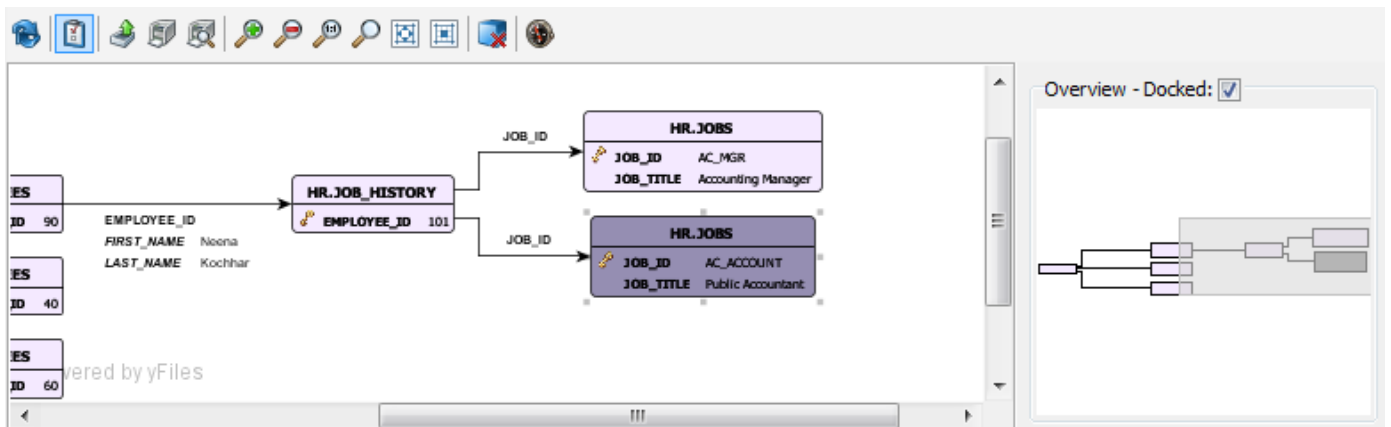


Figure: Graph with the Overview Window displayed


The gray area in the Overview control indicates the portion of the graph that is currently shown in the display area. You can drag the gray area around to study other portions of the graph.


To get a larger graph display area, you can put the Overview control in a separate window. Just uncheck the Docked checkbox.

Exporting and Printing the Graph

You can also export the graph to an image file or print it. Use the corresponding toolbar buttons to do this.

 Export the graph to a file in JPG, GIF, PNG, SVG or PDF format.

 Print the graph

 Show a preview of how the graph will be printed

When you print the graph, you are prompted for information about what to print (the Graph or the View, i.e., just the portion visible in the display area) and how many rows and columns to split the printing over (one page is used for each row/column). See the [Export and Import](#) chapter and the Print section in the [Getting Started](#) chapter for details.

Procedure Editor

Introduction

Many databases offer the capability to store custom code in the database, primarily as functions and procedures, where a function has a return value but a procedure does not (a procedure may instead have output parameters). In addition, some databases offer a package concept, which means that a collection of functions and/or procedures are grouped together in one unit. A package is the interface describing the functions and procedures, while the package body contains the implementation. Many databases also support triggers: code that is executed when triggered by an event such as deleting a row in a table. You can use DbVisualizer actions to create and drop procedural object of these types, and use the procedure editor to browse, edit and compile these object types. Procedures and functions can also be executed in the SQL Commander, with return values and parameters bound to DbVisualizer variables.

The examples throughout this document refer to the procedure object type, but all described features can also be applied to the other types of custom code objects. The screenshots show the interface for the Oracle profile, but it is very similar for other profiles.

Create Procedure

To create a new procedure, simply select the **Procedures** node in the objects tree and choose **Create Procedure** from its action menu.

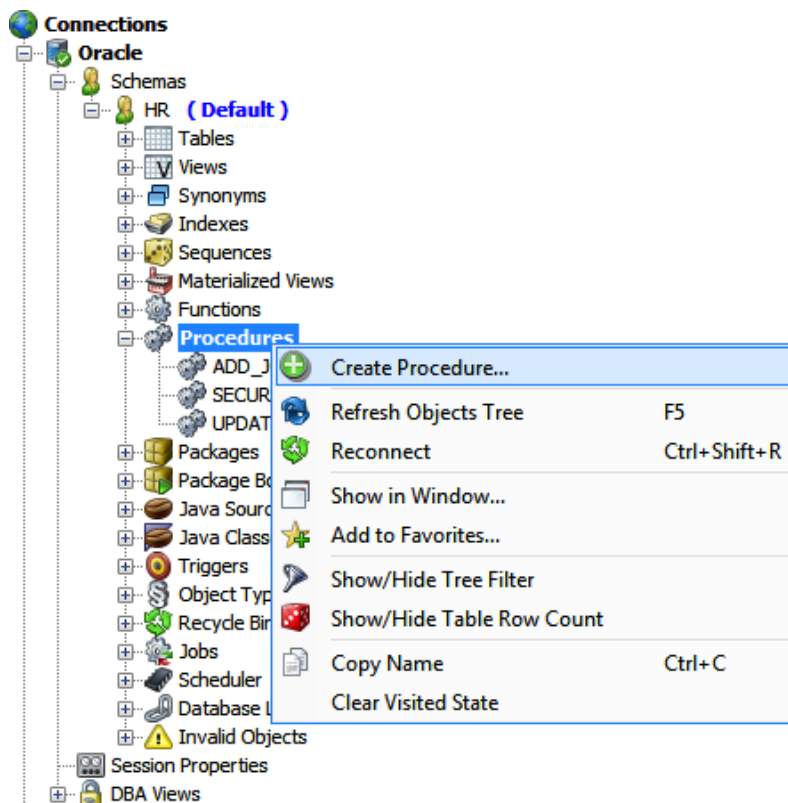


Figure: The actions menu for the Procedures node

Next, a dialog is displayed in which you enter the procedure name and the parameters for the new procedure. This data forms the interface for the procedure.

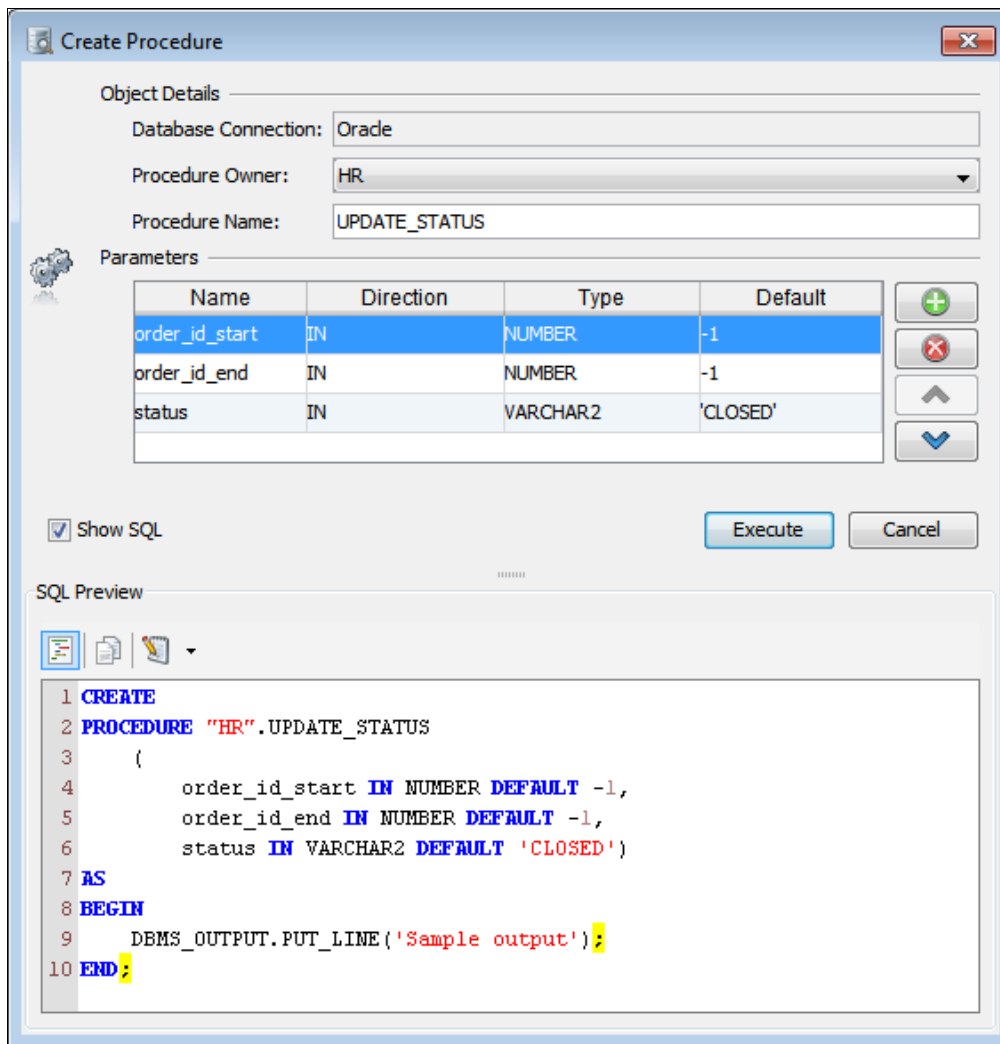


Figure: The create procedure dialog

Use the buttons to the right of the parameter list to insert, remove and move the parameters. For every parameter, you must supply its **Name**, the **Data Type** and the **Direction** (typically one of IN, OUT or INOUT).

The action uses this information together with a simple sample body to compose a CREATE statement. You can not enter the real code in the action dialog. The real code is often complex and large, so DbVisualizer provides a more powerful editing environment than what would fit in an action dialog via the Procedure Editor, described below. What you create with the action should be seen as a template that you then complete and work with in the editor.

Click **Execute** in the dialog to create the new procedure.

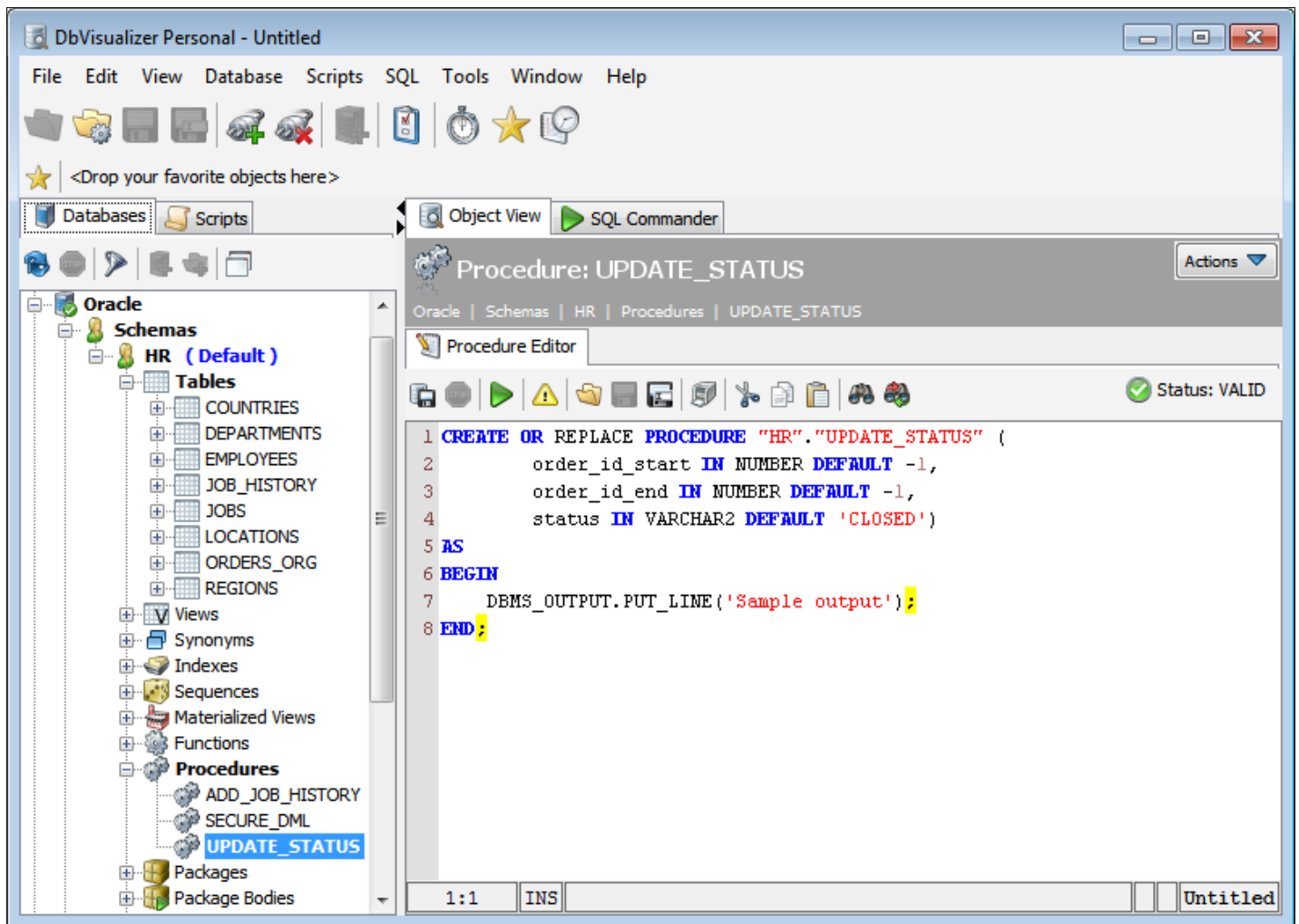


Figure: The newly created procedure

Selecting the newly created procedure in the tree will show the source for it in the Procedure Editor.

Edit and Compile

The editor has a toolbar with various actions to save/compile the procedure, save and load the source to/from file and perform common editing operations. The **Status** indicator shows whether the procedure is valid or invalid based on last compilation (not available for all databases).

Edit the source code and save/compile the procedure when you are happy with the code, using the **Save** toolbar button.

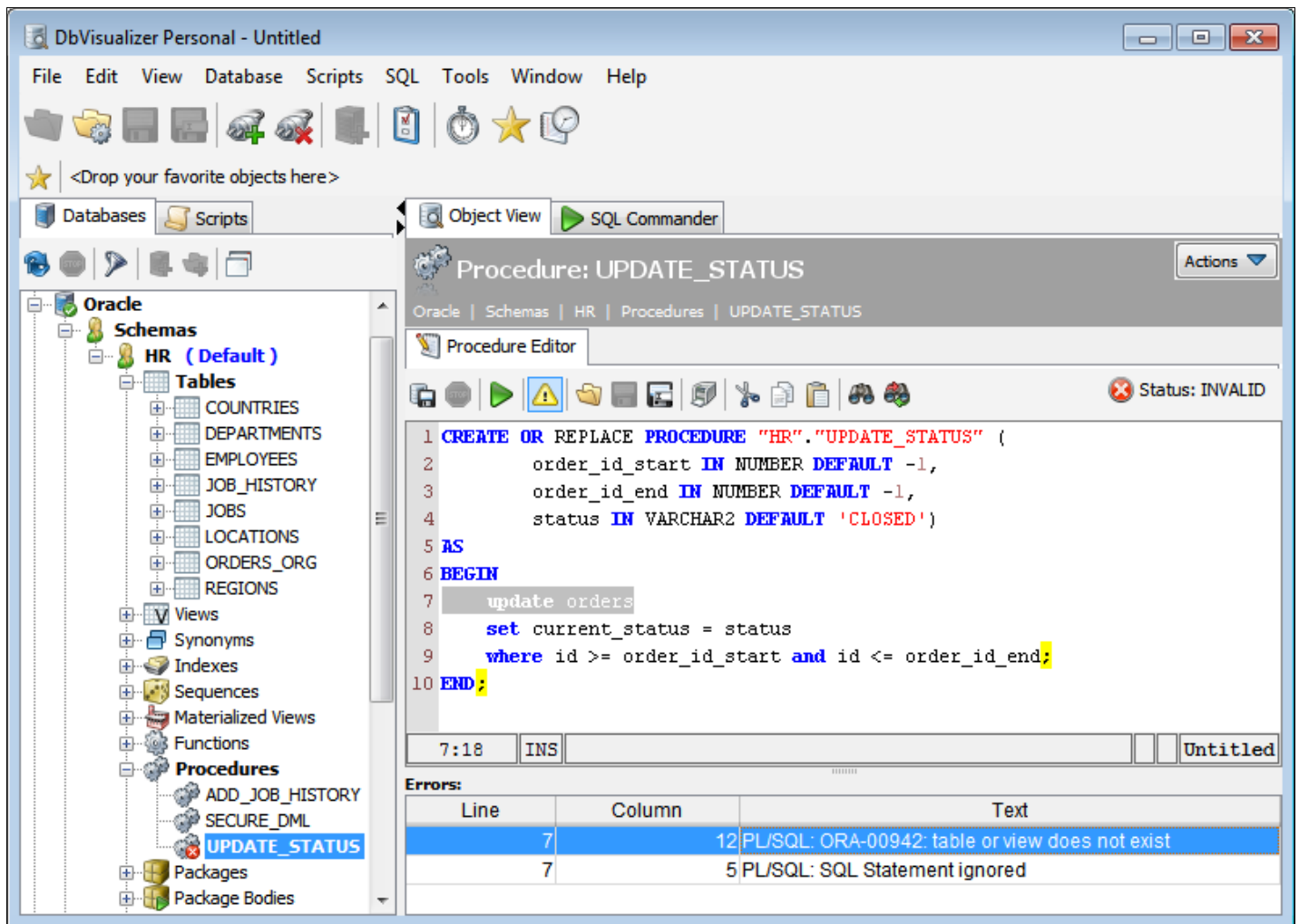


Figure: Compiling procedure with errors

If errors occur during compilation, the error list appears below the editor. It shows the row/column number for the error in the source editor and an error message. When you click the error in the list, the corresponding row is highlighted in the editor. Note, however, that some databases do not provide row/column information, only an error message. You then have to locate the incorrect statement yourself based on the description of the error.

In addition to the **Status** indicator in the editor, the object icon in the tree shows a little red cross for invalid procedures, for databases that provide this information. You can see this for the UPDATE_STATUS procedure node in the figure in the previous section.

The figure below shows the result after correcting the errors and recompiling the procedure:

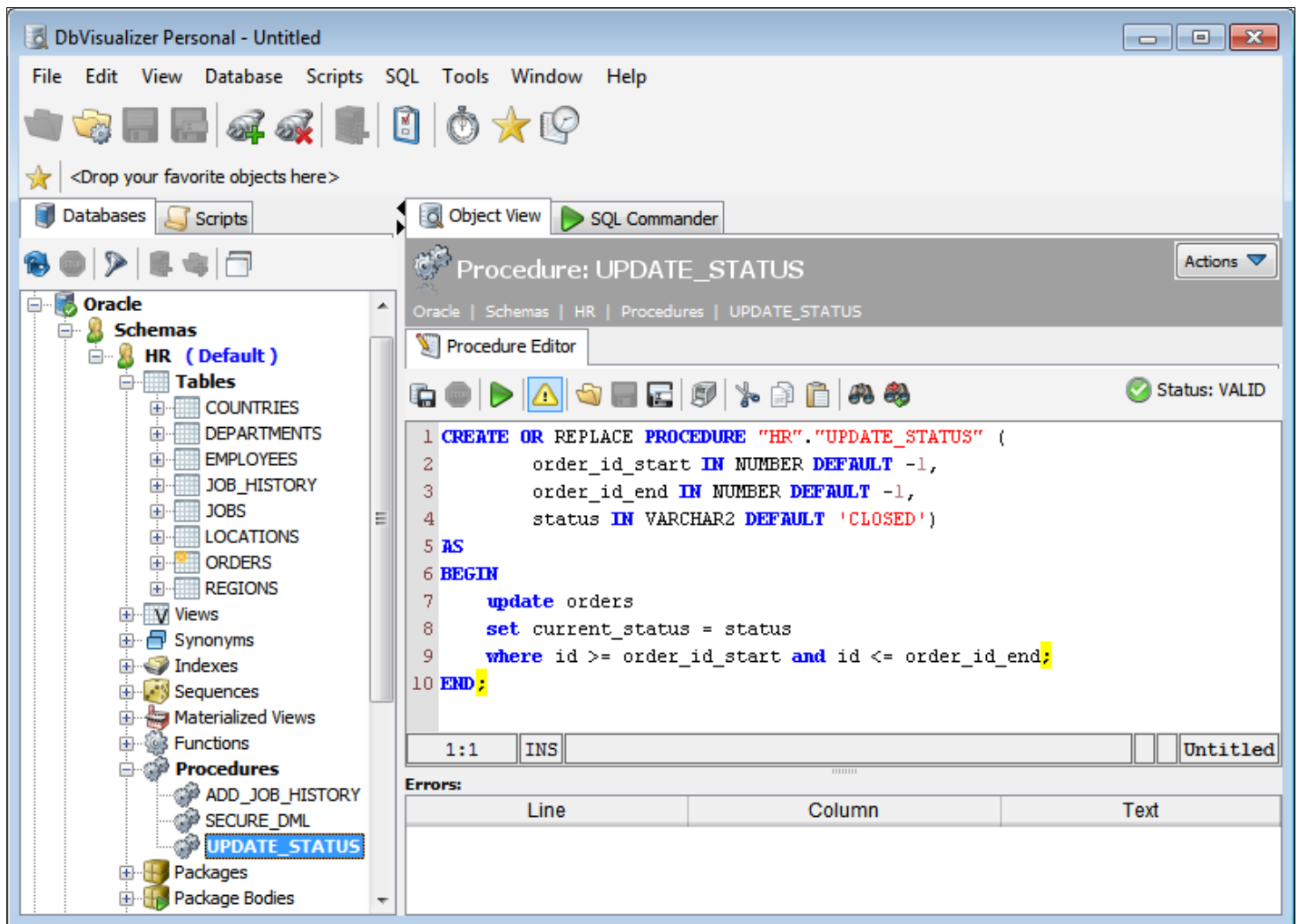


Figure: Compiling procedure with successful result

The status indicator now shows that the procedure is **VALID**.

Execute in SQL Commander

You can also test the procedure. First, click the **Execute** button in the Procedure Editor. DbVisualizer then generates a script for executing the procedure, using variables for all parameters, and executes it in the SQL Commander as shown in the next screenshot.

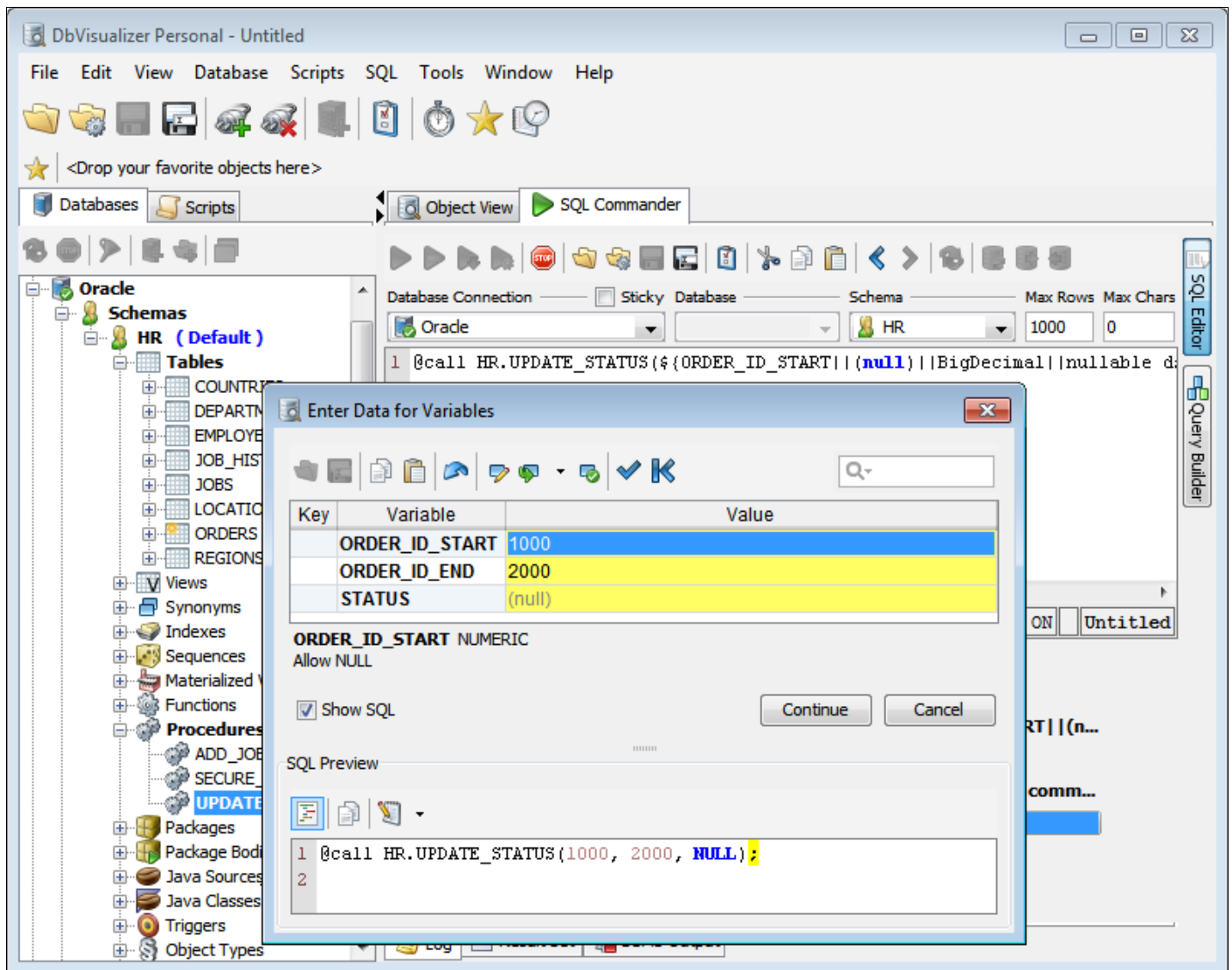


Figure: Running the procedure in SQL Commander

Because the script contains variables, the Variable Prompt dialog pops up. Enter values for all parameters and click **Continue** to execute the procedure.

In the example shown in the figure, all parameters are input parameters but DbVisualizer also support execution of procedures with output parameters and functions returning a value:

```
@call ${STATUS||(null)||String||noshow dir=out}$ = "HR"."GET_STATUS"(1002);
@echo STATUS: ${STATUS}$;
```

In this example, the result value from the GET_STATUS function is assigned to a variable named STATUS. Note that it has an option **dir=out**. This is a requirement for a variable that is assigned a value at runtime, whether it is used for a return value from a function call or for an output parameter in a procedure call. It also has the **noshow** option, to avoid getting prompted for a value for the variable. The value of the STATUS variable is then written to the log using the **@echo** command.

You can also use the output from one function or procedure as input to another, or even as a value in a SELECT or other SQL statement:

```
@call ${STATUS||(null)||String||noshow dir=out}$ = "HR"."GET_STATUS"(1002);
@call "HR"."UPDATE_STATUS"(1000, 2000, ${STATUS||||String||noshow dir=in}$);
```

Note that **dir=in** is specified for the STATUS variable when it is used in the UPDATE_STATUS procedure call. When you use a variable first for output and then as input with another **@call** command, you must change the direction option like this.

The **@call** command is described more formally in the [Client Side Commands](#) section of the [SQL Commander](#) chapter, and the full variable syntax is described in the [Variables](#) section of the same chapter.

Script CALL to Editor

As an alternative to using the **Execute** button in the Procedure Editor to generate a @call script, you can use the **Script Object to SQL Editor** right-click menu choices for a procedure or function object in the Objects Tree.

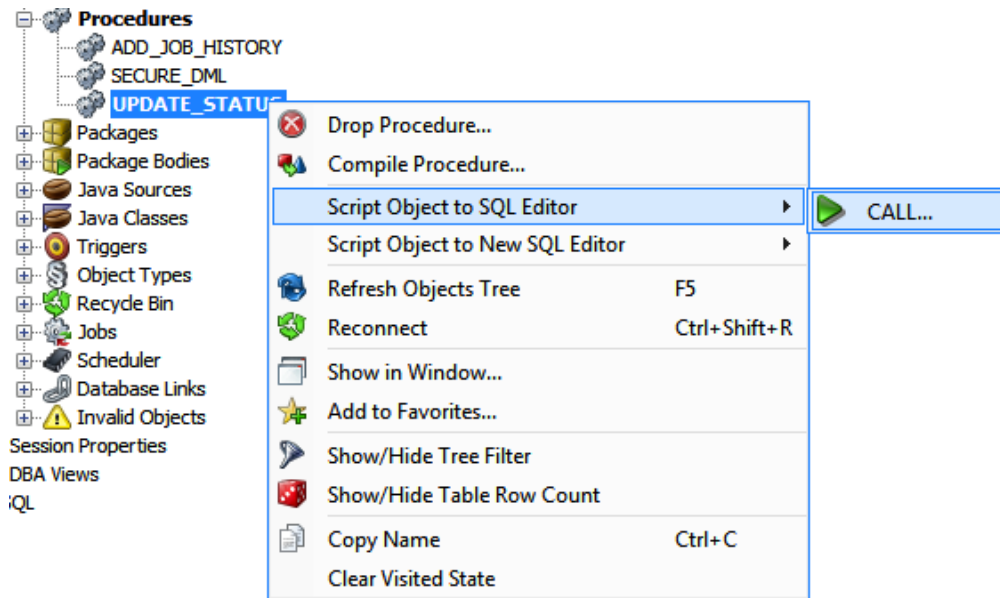


Figure: Creating a @call script for a procedure

Tool Properties

Customizing DbVisualizer

DbVisualizer is highly customizable. You can control formatting, layout and the way DbVisualizer interacts with databases. The default settings are good enough for normal use, but sometimes it is necessary to modify these properties. This chapter guides you through all the properties.

The Tool Properties window divides properties into two groups:

- **General Settings**
These settings controls DbVisualizer in general, such as fonts, colors, data formats, etc.
- **Database Settings**
These settings are per supported database type and defines properties that are used in database specific operations. When you set a database property in Tool Properties, it applies to all database connections defined for that database type. To set a property for one specific connection, use the Connection Properties, available in the Object Details area when you select a connection.

The user preferences (XML) files

All properties are saved in XML files. The exact location of these files is platform dependent. The location on your system is listed in the first, **General** category, in the Tool Properties window. These files contains, in addition to all properties, also the information about drivers, database connections, bookmarks, etc. We recommend that you do **not edit** these files manually; even though it is quite easy to do so, even a simple typo of an element name may cause problems. It's safer to edit all properties from the DbVisualizer GUI.

DbVisualizer automatically creates a backup copy of the XML files when the application is started. The location of these files is the same as for the standard XML files, but a **.bak** suffix is appended to the filename. The standard XML file might get broken for various reasons not in control by DbVisualizer. If you see a warning message that the XML file can not be read when you launch DbVisualizer, simply copy the backup file to the standard location and restart the application. If you move the XML file from its standard location, or if you remove it, DbVisualizer will automatically create a new one.

Tip: the `-prefsdir` command line argument is used to identify an alternative directory for your user settings.

Export Settings

Sometimes it may be necessary to migrate all your settings for DbVisualizer and import them in second setup of DbVisualizer. This is very handy if you are migrating from one machine to another, or if you want to setup an exact copy on your home computer, etc. Another key reason is for backup purposes. Loosing all database connection due to various reasons can be really frustrating. The Export Settings feature is available from the **File->Export Settings** main window menu choice.

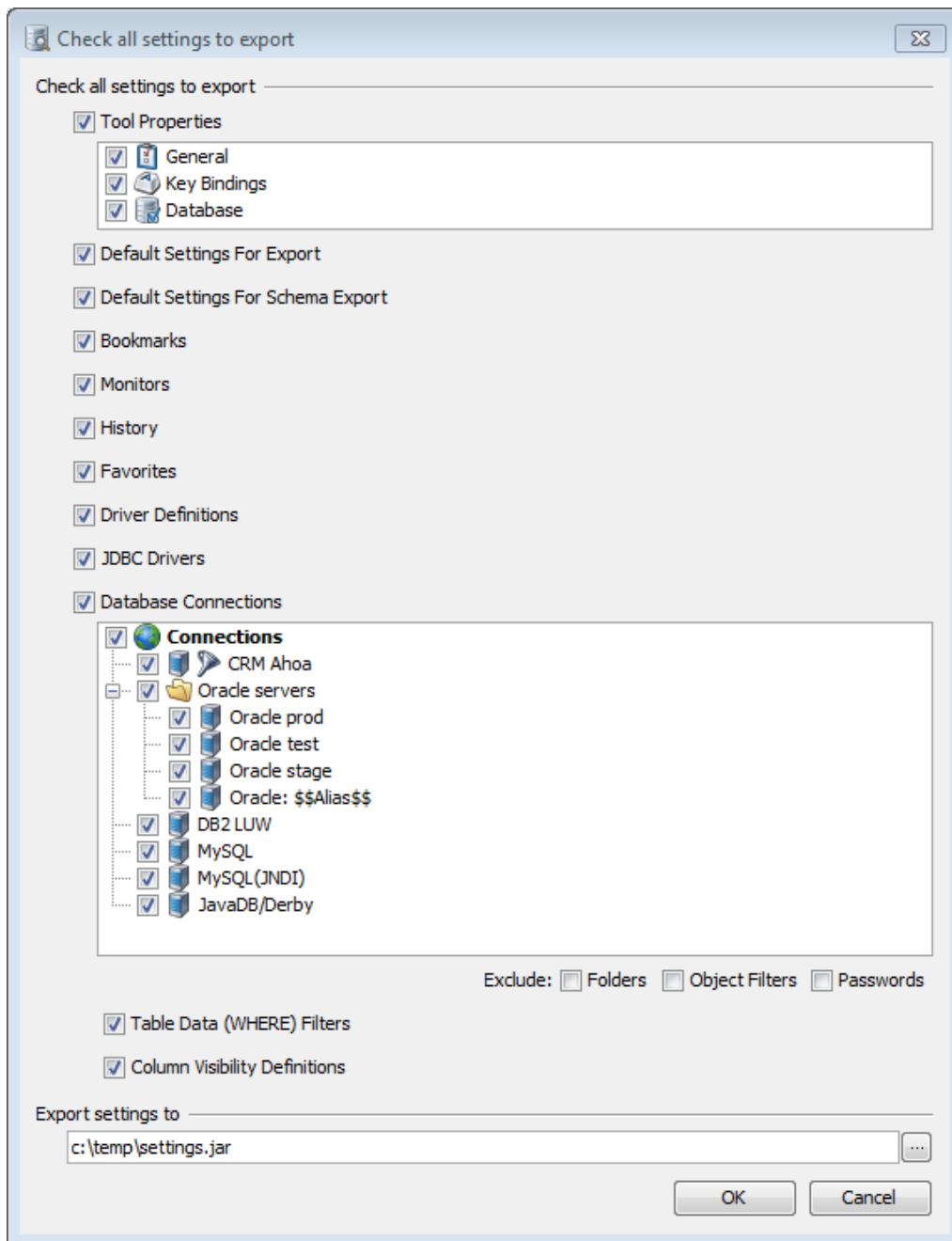


Figure: Export User Settings window

The default layout of the Export Settings window is that all settings will be exported. Once you're done press OK and all settings will be saved in the specified file. The structure of this JAR file is the same as the content in your DbVisualizer settings directory.

All settings can be exported except JDBC driver files.

Import Settings

The Import Settings feature is used to import settings as previously exported via the Export Settings feature. Import will examine the content of the specified file and present the choices available. Consider the previous screenshot and that we export the settings for the Database Connections only. Here is how the **Import Settings** window will look:

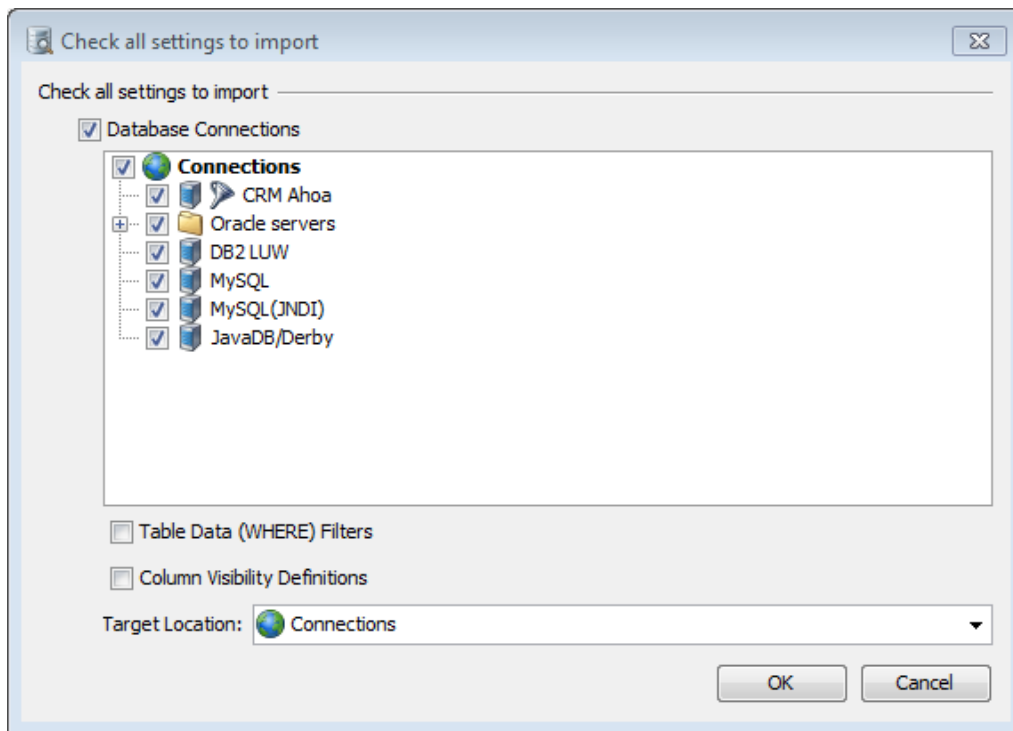


Figure: Import User Settings window

Use the **Target Location** button to set where the imported database connections will appear in the objects tree.

General Settings

The General settings tab collects all categories that are used to control the general aspects of DbVisualizer.

Use the buttons at the bottom of the window when you have made some changes: Click **OK** to save the changes and close the window, the **Apply** button to save the changes but keep the window open, and the **Cancel** button to revert all changes. To reset the properties to the factory defaults, use the **Defaults** button.

Changes are tracked on a per category basis. If you have made changes and click on another category, you are asked whether the changes should be applied or not. When you click **Defaults** (for both the **General** and the **Database** properties), you can reset either all properties or just the properties for currently selected category.

This is a screenshot of the **General** category tree.

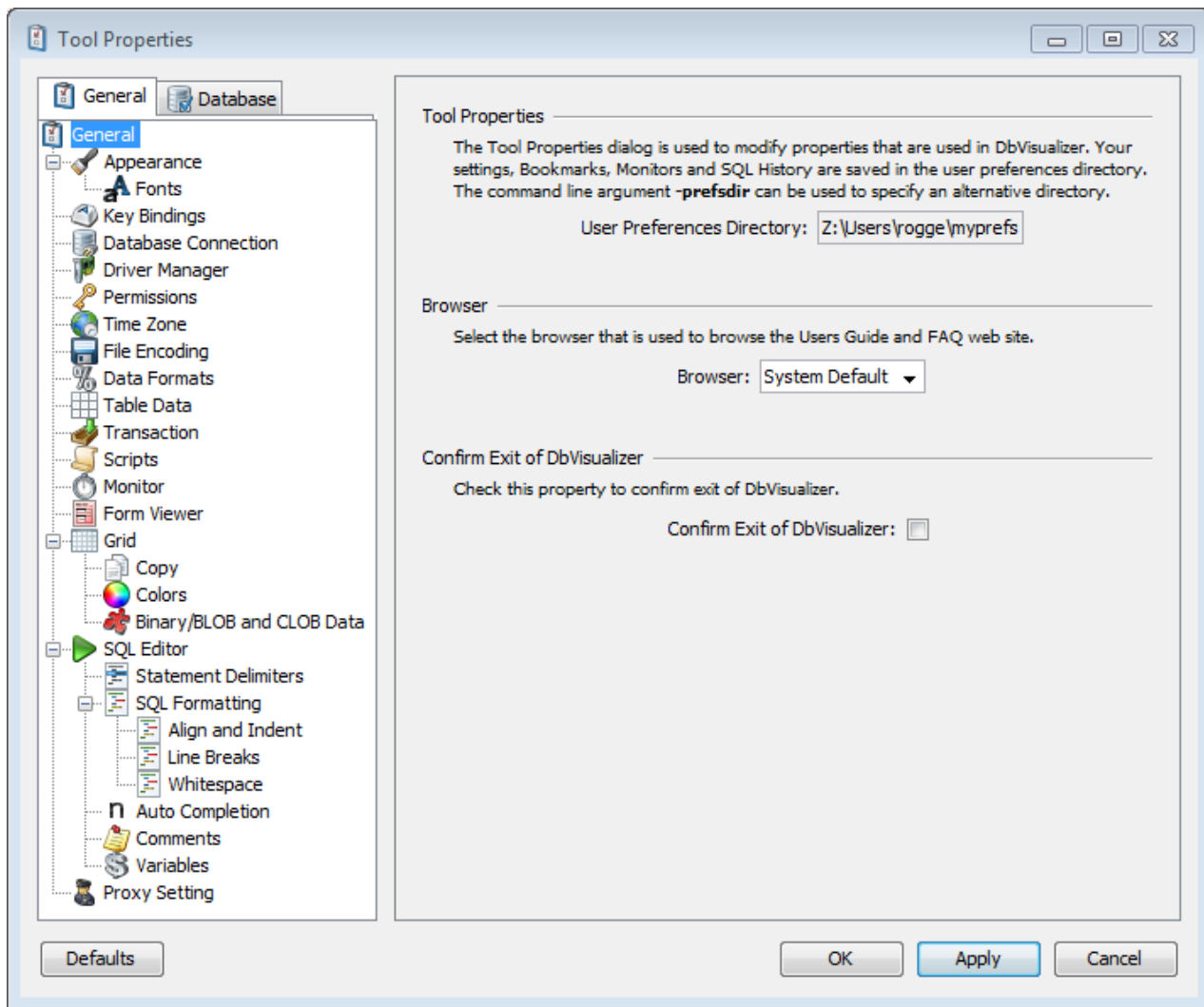
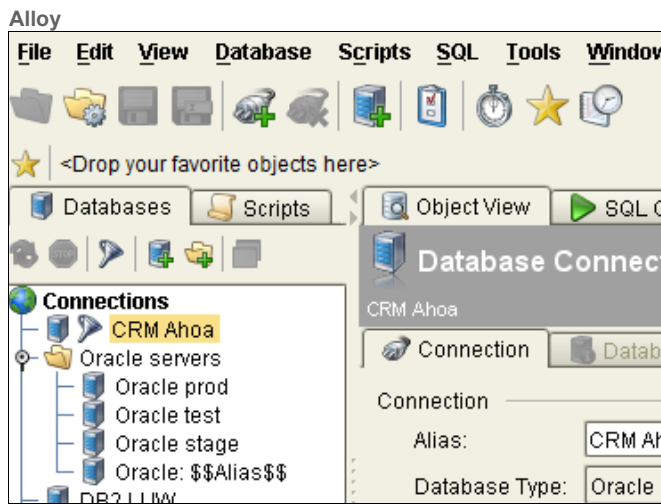
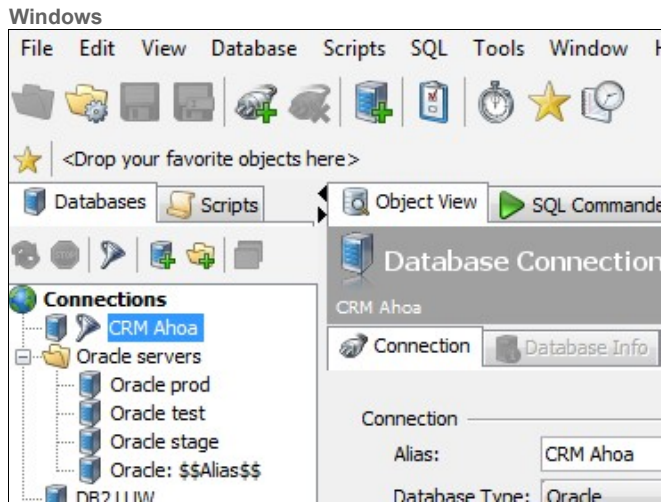
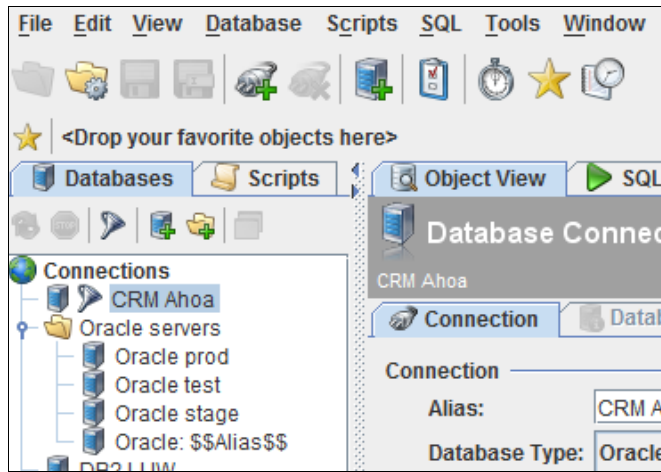


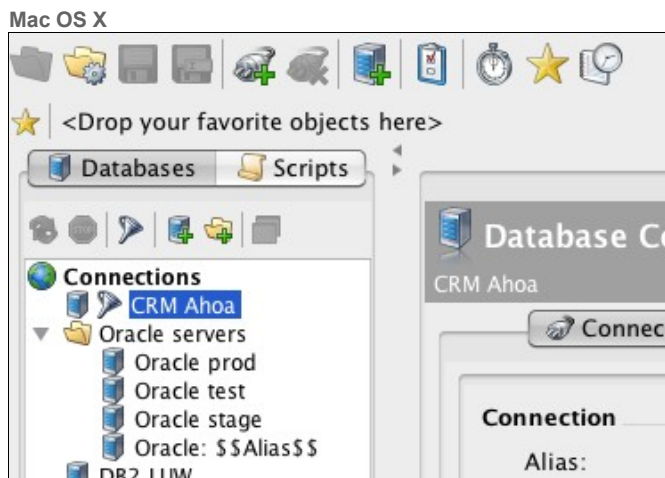
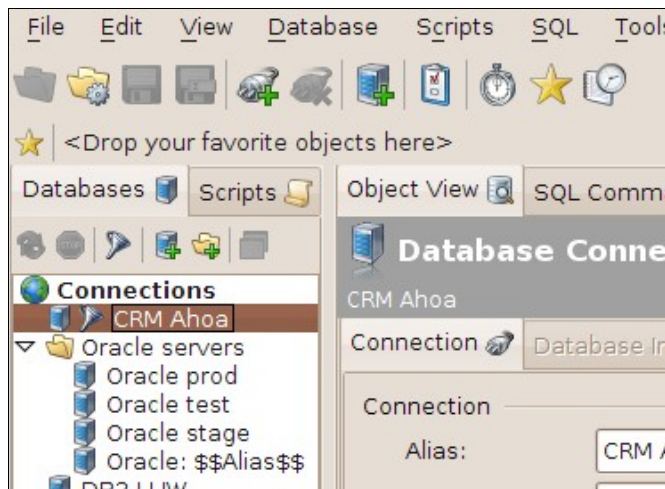
Figure: The Tool Properties window showing the tree with General categories

Appearance

Property	Description
Look and Feel	Controls which look and feel to use. Note 1: You must restart DbVisualizer after you have selected a new look and feel. Note 2: Some look and feels are platform specific and do not appear on all OS'es Metal (Ocean)



GTK+



Icon Sizes The **Menus**, **Main Tool Bars**, **Sub Tool Bars** settings are used to control the size of the icons.

Show Tab Icons Specifies whether an icon will appear in the header of all object view tabs.

Fonts

Individual fonts can be defined for **SQL Editors**, **Grids** and **Text** output data. The **Application Font** settings is used to control the font for all other components in the user interface, such as labels. Increasing the application font size is useful at demos or presentations. Anti-Aliased Fonts is supported by some look and feels and when enabled it gives a much smoother appearance of text in the application. Anti-Aliased font is not supported by the SQL editor.

Key Bindings

You can define key bindings for almost all operations and editor commands in DbVisualizer. Key bindings are grouped in **Key Maps**. DbVisualizer includes a set of predefined key maps targeted for the supported operating systems. These key maps cannot be deleted or modified. To customize key bindings, copy an existing key map and make your changes.

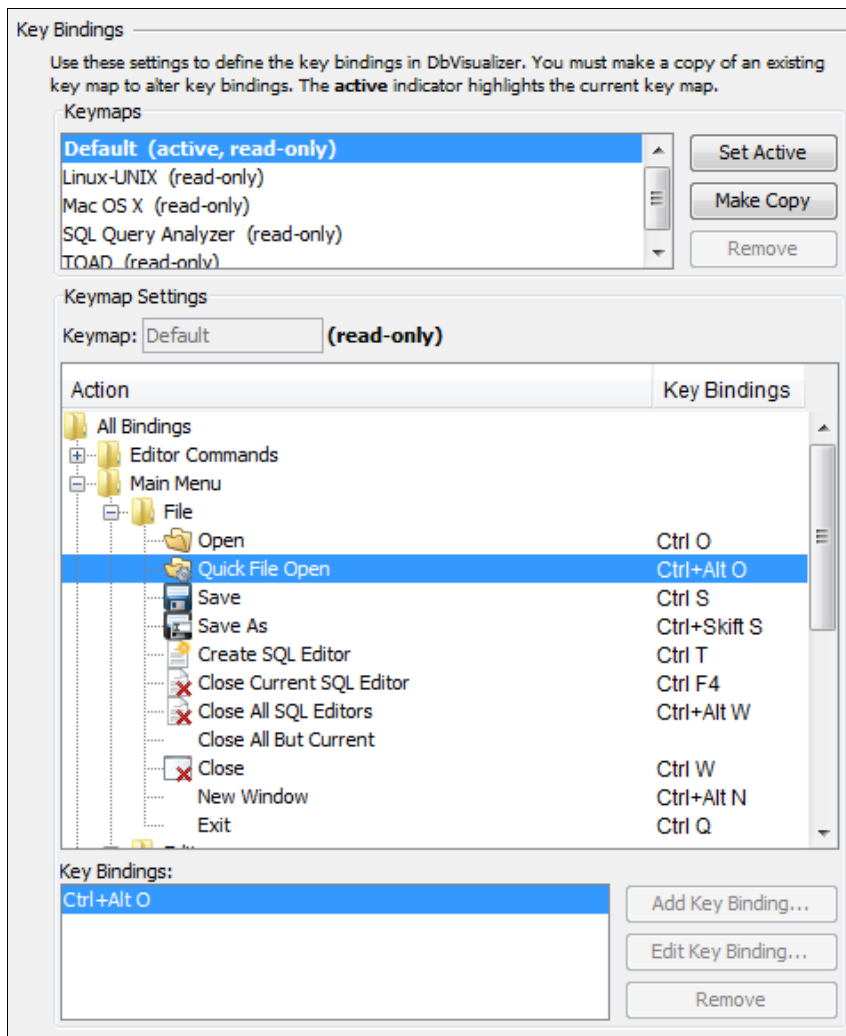


Figure: The key binding editor

All user defined key maps are stored in your `$HOME/.dbvis/config70/keymaps` directory. A key map file contain only the differences between the copied key map and the current.

To create a new key map, select the map you want to copy and click the **Make Copy** button. Set a name on the new key map and activate it with the **Set Active** button. The newly created key map now has the exact same key bindings as the parent key map.

Key maps must be uniquely named.

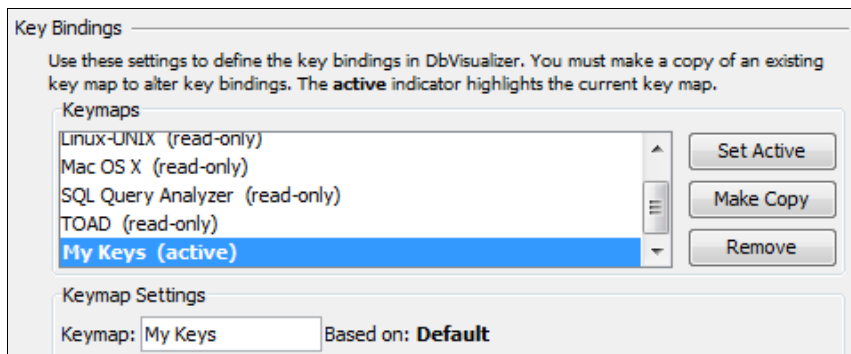


Figure: User defined key map

The action list is organized in folders. The **Editor Commands** folder lists all actions available in the SQL Commander editor and their current key bindings. The **Main Menu** folder contain sub folders, each representing a main window menu. The other folders group feature specific actions, such as actions to control the references graph, form editor, etc.

To modify the key bindings for an action, select the action from the action list. The current key bindings are listed in the **Key Bindings** list.

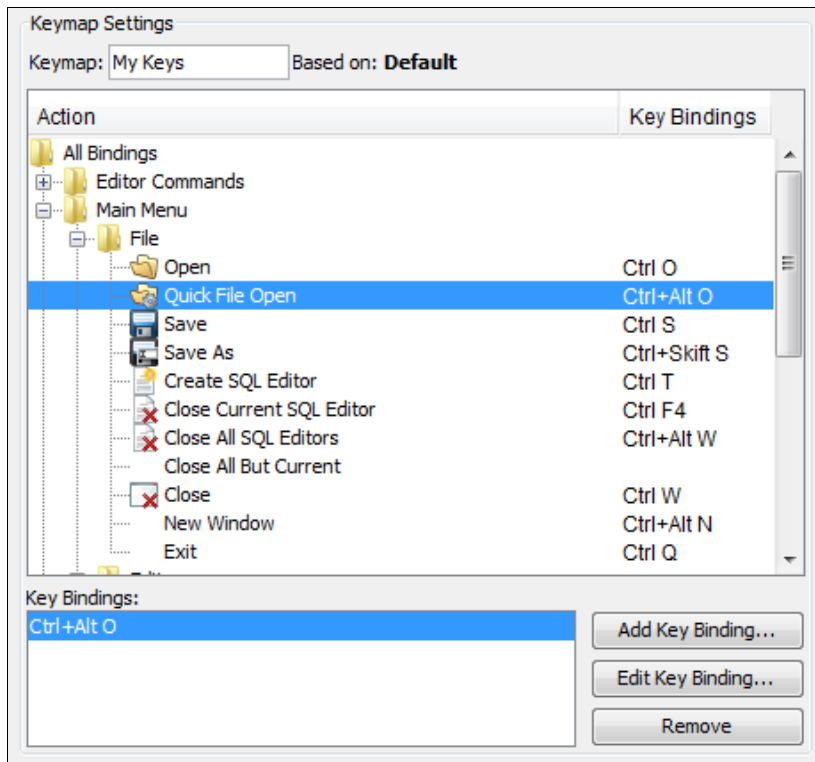


Figure: User defined key map

To add an additional key binding, press **Add Key Binding** or press **Edit Key Binding** to edit the selection.

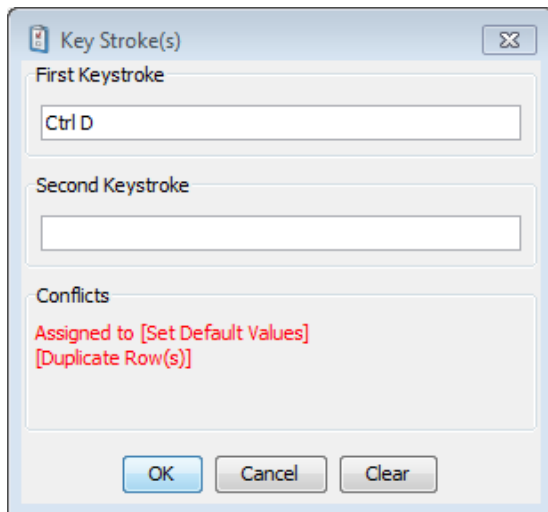


Figure: Key stroke dialog

The key stroke dialog controls whether a key binding is already assigned somewhere else. If there is a conflict with another binding, the Conflicts are shows the names of the actions that are conflicting. The modifier keys Shift, Alt, Ctrl and Command can be used to form the final key binding.

Menu items and tooltips shows the first defined key binding in the list.

Database Connection

Property

Description

Ask When Creating Database Connection	If enabled, you will be asked if you want to use the Connection Wizard to create new connections.
Run "Connect All" at Startup	If enabled, the Connect All operation is automatically run when you launch DbVisualizer, connecting all Database Connections marked as being included in the Connect All operation (see the Database properties further down for more on this).
Confirm "Disconnect All"	If enabled, a dialog to be displayed before disconnecting all current database connections when using the Disconnect All operation.
Execution Timeout Value (seconds)	Specify here how many seconds after which a database call that locks the GUI will time out

Driver Manager

The Driver Manager searches specified folders for JDBC drivers and helps you make them available for use by DbVisualizer, see the [Load JDBC Driver and Get Connected](#) section for details. In the Driver Manager properties category, you can specify if you want the Driver Manager to run automatically at start-up, when new files are discovered in the specified driver folders, or when driver related errors are encountered. You can also specify the folders to search and files to exclude, if any.

Permissions

The Permission functionality is a security mechanism, where you can specify that certain database operations must be confirmed. You configure permissions per connection mode (Development, Test and Production) for feature areas described in the following sections.

Note: The permission feature is part of DbVisualizer and does not replace the authorization system in the actual database.

SQL Commander Permissions

For the SQL Commander, you can pick the permission type from a drop-down list for each SQL command:

Permission Type	Description
Allow	This permission type means that you can run the SQL statement without any confirmation
Deny	This permission type means that the SQL statement is not executed at all.
Ask	This permission type means that when executing an SQL statement, or a script of statements, the SQL Commander asks you whether the actual SQL command(s) should be executed.

SQL Commander Permissions

Define here whether the SQL Commander should **allow (execute)**, **deny (not execute)** or **ask** before executing the following SQLs organized per connection mode.
(The connection mode for a database connection is set either in the **Tool Properties->Database** tab or in **Connection Properties**).

	Development	Test	Production
SELECT:	Allow	Allow	Allow
INSERT:	Allow	Allow	Ask
UPDATE:	Allow	Allow	Ask
DELETE:	Allow	Allow	Ask
TRUNCATE:	Allow	Ask	Ask
CREATE:	Allow	Allow	Ask
ALTER:	Allow	Allow	Ask
DROP:	Allow	Ask	Ask
COMMIT/ROLLBACK:	Allow	Allow	Ask
Other:	Allow	Ask	Ask

Figure: SQL Commander Permissions

Table Data Editing Permissions

The permission types for the inline editor are:

Permission Type	Description
Confirm	A confirmation window is displayed, and you can accept the operation or cancel it
No Confirm	The SQL operation is performed without any confirmation

Table Data Editor Permissions

Define here if insert, update and delete must be confirmed in the table data editor.

	Development	Test	Production
INSERT:	No Confirm	No Confirm	Confirm
UPDATE:	No Confirm	Confirm	Confirm
DELETE:	No Confirm	Confirm	Confirm

Figure: Table Data Editor Permissions

Time Zone

In the Time Zone properties category, you can change the time zone for the DbVisualizer process, and thereby how date and time data is interpreted. DbVisualizer uses the OS time zone by default, which is usually what you want to use.

Changing the time zone is only of interest if you work with a database running with a different time zone than the time zone set on the client where you run DbVisualizer. One example is when working with a database that uses the UTC/GMT time zone to normalize all date/time data.

File Encoding

In the File Encoding category, you can set which file encoding DbVisualizer uses by default when reading and writing files, e.g., SQL scripts loaded into the SQL Commander or files with exported data. By default, DbVisualizer uses the default encoding for your operating system, and this is

typically what you want. You only need to change this setting if you often work with files in another encoding, or if DbVisualizer can not find the default encoding for your operating system.

Data Formats

Property	Description
Date Format	Specifies the date format to use throughout the application (i.e., in grids, forms and during editing). More information below.
Time Format	Specifies the time format to use throughout the application (i.e., in grids, forms and during editing). More information below.
Timestamp Format	Specifies the timestamp format to use throughout the application (i.e., in grids, forms and during editing). More information below.
Numbers Format	Specifies how numbers will be formatted.
Decimal Number Format	Specifies how decimal numbers will be formatted.
Boolean/BIT Format	Specifies the textual representation of boolean values (true/false).
Null String	Specifies the string representation of the null value. This string is the readable form of null and appears in grids, forms, exports and during editing.

Date, Time and Timestamp formats

The lists for date, time and timestamp format contain collections of standard formats. If these formats are not suitable, you can enter your own format in the appropriate field. The tokens used to define the format are listed in the right-click menu when the field has focus.

G - Era Designator
y - Year
M - Month in year
w - Week in year
W - Week in month
D - Day in year
d - Day in month
F - Day of week in month
E - Day in week
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 - Millisecond
z - Time zone
Z - Time zone

Figure: The date and time right click menu

The complete documentation for these tokens is available at the following web page: [SimpleDateFormat](#).

Number formats

The lists for number and decimal number contain collections of standard formats. If these formats are not suitable, you can enter your own format in the appropriate field. The tokens used to define the format are listed in the right-click menu when the field has focus, and complete documentation for these tokens is available at the following web page: [DecimalFormat](#).

Table Data

Property	Description
Show Table Row Count	Specifies if the number of rows in a table will be displayed in the header of the table in the Database Objects->Data tab . Enabling this property will cause an extra round trip to the database (i.e., a minor performance penalty)
Highlight Primary Key Columns	Specifies if Primary Key columns will be indicated in the Database Objects->Data tab , Variable Substitution window, SQL Commander result grids and in the References graph.
Include Variables in SQL	Specifies if the right-click menu operations in the Data tab will create SQL statements that include DbVisualizer variables or if the generated statements are plain SQL. Letting DbVisualizer generate statements with variables results in the Variable Substitution window being displayed when these statements are executed in the SQL Commander.
Max Rows at First Display	Set the number of rows that will be fetched for a table in the Data tab when a table is first displayed.

Transaction

Property	Description
Pending Transactions at Disconnect	Specifies what DbVisualizer does on exit from the application, when the auto commit setting is disabled.

Scripts

Property	Description
Default Editor when Double-click	Define what target editor should load the file when double-click Bookmark, Monitor or History script files.
Default Editor when Alt+Double-click	Define what target editor should load the file when Alt+double-click Bookmark, Monitor or History script files.

Monitor

Property	Description
Start Monitors Automatically	Check to enable start of monitors automatically when database connections are established.

Form Viewer

Property	Description
----------	-------------

Right Aligned Numbers If enabled, numbers are displayed as right-aligned in the Form Editor/Viewer.

Image Thumbnail Size The number of pixels for the widest side of an image (represented by binary data) when shown in the Data Form Viewer. The value is used to scale the image proportionally. The default is 150.

Grid

Property	Description
Auto Resize Column Widths	If Auto Resize is enabled, DbVisualizer automatically sizes each grid column based on the widest cell value. If Consider Column Header is also enabled, the header widths are also considered when calculating the column widths.
Show Grid Row Header	If enabled, a row header is shown also for read-only result set grids, such as monitoring result set grids.
Max Column Separator Width	This setting is used only when Auto Resize Column Widths is enabled and specifies a maximum visual column width for grids. The Max Chars property in the Database Objects Data tab and in the SQL Commander is used to control the max number of characters for text values. If the number of characters for a text column is more than this setting, the column is colored in a light red color and the value is truncated as specified by this property:
Meaning of setting Max Chars	<ul style="list-style-type: none">• Truncate Values Truncate the original value to be less then the setting of Max Chars. Note: This affects any subsequent edits and SQL operations that use the value since it's truncated. This setting is only useful to save memory when viewing very large text columns.• Truncate Values Visually Truncate the visible value only and leave the original value intact. This is the preferred setting since it will not harm the original value. The disadvantage is that more memory is needed when dealing with large text columns.

Copy

The Copy category groups properties that control the result of using **Copy Selection** and **Copy Selection (With Column Header)** via the grid right-click menu, the corresponding key bindings, and drag and drop.

Property	Description
Column Delimiter	Specifies the delimiter between columns in a multi column copy
End of Line Delimiter	Specifies the new line control characters for multi row copy requests

Colors

The Colors category is used to define alternative background, foreground and grid colors for grid components.

Binary/BLOB and CLOB Data

Property	Description
BLOB	Specifies how BLOB and binary data values are represented in grids. Setting this property to By Value results in performance penalties and the memory consumption increases dramatically.

CLOB Specifies how CLOB data values are represented in grids. Setting this property to **By Value** results in performance penalties and the memory consumption increases dramatically.

SQL Editor

The editor category controls various settings specific for the SQL Commander editor.

Property	Description
Tabs	Specifies settings for the tab keyboard key: Tab Size (the number of characters a tab character corresponds to), Whitespace(s) per Tab (by how many characters to indent when the tab key is pressed), and Expand Tab to Whitespace (if enabled, always insert space characters when the tab key is pressed). If Expand Tab to Whitespace is disabled, a tab character is inserted when the tab key has been clicked as many times as it takes to indent to the value specified by Tab Size, i.e., if Whitespace(s) per Tab is set to 4 and Tab Size is set to 8, clicking the tab key twice results in a tab character.
Recent Files Limit	Specifies the max number of files listed in the File->Open Recent sub menu.
Confirm Close of Unsaved Editors	If enabled, DbVisualizer asks you whether to save the text in an SQL editor with modified content (any editor; not only editors loaded from file) when you close the editor.
Set "Sticky" for SQL Editor(s)	If enabled, the Sticky flag is automatically set for all new SQL Editors, which means that the database connection details only can be changed manually.

Statement Delimiters

Statement delimiters define how a script should be divided into specific SQL statements in the pre-processing phase.

Property	Description
SQL Statement Delimiter 1	Defines the character(s) used to delimit one SQL statement from another in a SQL script
SQL Statement Delimiter 2	Defines the additional character(s) used to delimit one SQL statement from another in a SQL script. If there is no need for more than one SQL statement delimiter, set this one to the same as delimiter 1.
Allow "go" as Delimiter	Specifies whether go as the first word on a single line should be interpreted as a statement delimiter.
Begin Identifier	Defines the character(s) that identifies the start of an anonymous SQL block.
End Identifier	Defines the character(s) that identifies the end of an anonymous SQL block

SQL Formatting

The SQL formatting category groups properties to control the SQL formatting feature in the SQL Commander. To see the effect of each property, modify it, press **Apply** and format the SQL in the SQL Commander.

Auto Completion

This category is used to define the visual appearance of the auto completion popup in SQL Editors.

Property	Description
Sort Tables List	Enable this to always present tables sorted in the auto completion popup

Sort Columns List	Enable this to always present column names sorted in the auto completion popup
Display Automatically	Enable this and the auto completion popup is automatically displayed whenever possible
Instant Substitution	Enable this and the auto completion feature substitutes directly if there is only one matching entry
Display Delay	Specifies the time in milliseconds until the auto completion popup is displayed automatically

Comments

Property	Description
Single Line Identifier 1	Specifies the character(s) that identifies the beginning of a one line comment
Single Line Identifier 2	Specifies the additional character(s) that identifies the beginning of a one line comment
Block Comment Begin Identifier	Specifies the character(s) that identifies the start of a multi line comment block
Block Comment End Identifier	Specifies the character(s) that identifies the end of a multi line comment block

Variables

Variables can be used in the SQL executed in the SQL Commander. Before executing an SQL statement or connecting a database connection, a dialog is displayed, asking for replacement values.

These settings define a character sequence that identifies a variable and another sequence that delimits different parts of a variable. Example: `${variable}$`.

Property	Description
Variable Identifier Prefix	The start identifier for a variable. Default is <code>\${</code> .
Variable Identifier Suffix	The end identifier for a variable. Default is <code>}\$</code> .
Variable Delimiter	The delimiter used to identify the parts of a variable. Default is <code> </code> .

Proxy Settings

The **Check for Updates** feature requires HTTP access to the internet. If you access the internet through a proxy, you must specify the proxy settings in order to use this feature.

Property	Description
Proxy Type	Specifies the type of proxy you use: HTTP or SOCKS
Proxy Host	Specifies the name or the IP address for the proxy host
Proxy Port	Specifies the proxy port number
Proxy User	If the proxy requires authentication, specifies the proxy user account name. Leave blank for a non-authenticating proxy

Proxy Password If the proxy requires authentication, specifies the password for the proxy user account name. Leave blank for a non-authenticating proxy

Database Settings

Database settings extends the General settings with properties that may have different values per supported database type. You specify the database type for a connection by choosing the appropriate type from the **Database Type** list in the Connection tab. If there is no matching entry, use the **Generic** database type.

The database type specific properties in the Tool Properties apply to all connections of the specific database type. You can also override these properties in the Connection Properties tab for a specific connection, in case you need to use different values for connections of the same database type.

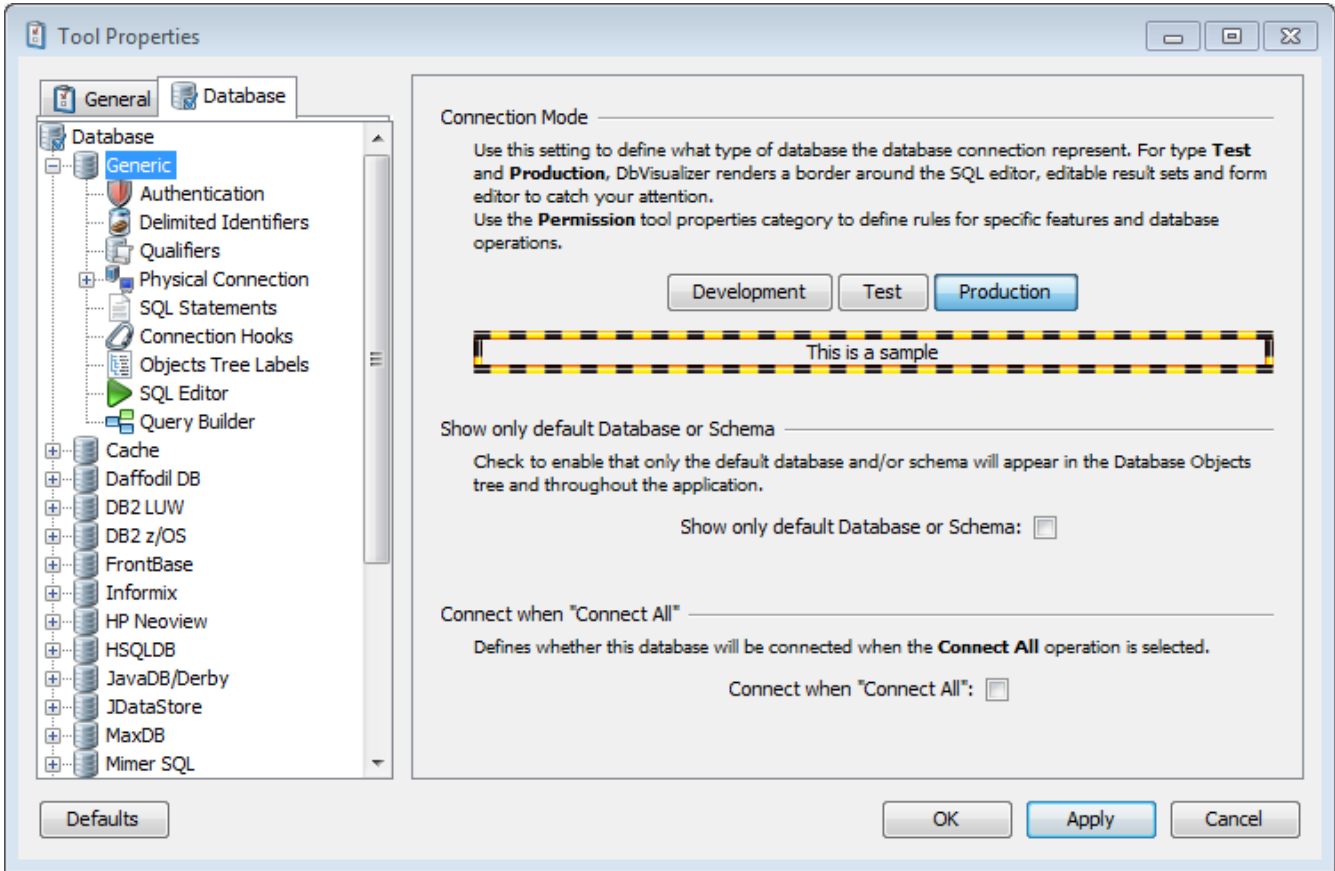


Figure: The Tool Properties window showing the tree with Database categories

The following properties are displayed when selecting a database type in the tree.

Property	Description
Connection Mode	Specifies the connection mode for the database connection: Development, Test or Production. Permissions are based on connection mode. For the Test and Production modes, DbVisualizer displays a border around areas where database content can be edited, to bring your attention to the fact that you are connected to a database where others may be affected by your changes.
Show only default Database or Schema	Enable this if you only want the default database or schema listed in the database objects tree.
Connect when "Connect All"	The Connect All feature allows you to connect to multiple database connections with a single click. Enable this property to include database connections of this type when using the Connect All feature.

Authentication

Property	Description
Save Password	If enabled, DbVisualizer saves the password for the database connection between invocations. (The password is saved encrypted)
Clear Password at Disconnect	If enabled, the password is cleared at disconnect
Require Userid	If enabled, you are asked to enter a userid whenever the database connection is established
Require Password	If enabled, you are asked to enter a password whenever the database connection is established

Delimited Identifiers

Delimited identifiers are identifiers which do not need to follow the rules of regular database object identifiers. Usually, delimited identifiers are used when you need to use SQL reserved words, spaces and mixed case sequences in an identifier.

Property	Description
Begin Identifier	Defines the start character for a delimited identifier. Normally, this is a double quote as in "...".
End Identifier	Defines the end character for a delimited identifier. Normally, this is a double quote as in "...".
Scripting	Enable this to use delimited identifiers in the Scripting features
Auto Completion/Query Builder	Enable this to use delimited identifiers in the auto completion and query builder features

Qualifiers

These properties control whether table and column names should be qualified when DbVisualizer generates SQL statement.

Property	Description
Qualify with Schema/Database: Scripting	Enable this to qualify object names with the schema/database in the Scripting features
Qualify with Schema/Database: Auto Completion/Query Builder	Enable this to qualify object names with the schema/database in the auto completion and query builder features.
Qualify with Schema/Database: References/Navigator Graphs	Enable this to qualify object names with the schema/database in the graphs shown in the References and Navigator tabs.
Qualify Columns: Auto Completion/Query Builder	Enable this to qualify column names with the table name in the auto completion and query builder features. Note: When you specify a table name alias, it is always used as a column name qualifier, regardless of this property setting.

Physical Connection

The Physical Connection category controls whether DbVisualizer should use only one physical connection with the database server or if physical connections will be acquired when needed. The **Use Single Shared Physical Database Connection** is disabled by default. If enabled then briefly it means that whenever establishing a connection DbVisualizer will assign one physical database connection for the objects tree and one per every SQL editor in the SQL Commander. The physical connection for a SQL editor is not acquired directly when the editor is created but rather when

doing the first execute in it.

If enabling **Use Single Shared Physical Database Connection** then only one physical connection will be used for that database. DbVisualizer will then share the physical connection among all features communicating with the database. If using a single physical connection and auto commit is off then a confirmation dialog may appear when launching features that require transaction control and if there are uncommitted changes in the database.

Transaction

Property	Description
Auto Commit	Defines if each executed SQL statement will be auto committed or not. This setting applies for all SQL statements that are executed in the SQL Commander.
Ask when Auto Commit is Off	If auto commit is off then this setting when enabled will show a confirmation dialog if there are uncommitted changes produced by the last execution in the SQL Commander.
Transaction Isolation	Attempts to change the transaction isolation level for all database connections. Note: If this property is changed during a transaction, the result is JDBC driver specific.
Commit Batch Size	Specifies after how many rows DbVisualizer commits the transaction when saving a batch of changes in the table data editor and when inserting rows in table data import .

SQL Statements

This category controls the SQL templates that DbVisualizer uses internally throughout the application. Each SQL template is composed of the standard SQL and variables. Variables are identified with **\${...}\$**. DbVisualizer relies on a number of predefined variables, listed in the **SQL Templates** area right-click menu:

catalog
catalogseparator
schema
schemaseparator
table
table-name
where-columns
quoted-where-columns
columns
values
column-values
create-columns
index-type
index
unique
index-columns
create-primary-key
DbVis-Date
DbVis-Time
DbVis-Exec-Time
DbVis-Fetch-Time

Figure: All predefined variables

A specific predefined variable can be used in one or more of the SQL templates. Using a variable that is not valid for a specific SQL statement will

result in the variable appearing as-is when the statement is executed.

There is normally no reason to modify the SQL templates, nor the variable identifier or delimiter settings. There might however be circumstances when edits are needed, for instance to modify the appearance of the where clause or the list of columns.

Property	Description
SELECT ALL	Command used when selecting all rows for a table
SELECT ALL WHERE	Command used when selecting some rows for a table
SELECT COUNT	Command used to get the number of rows in a table
INSERT INTO	Command used to insert a new row into a table
UPDATE WHERE	Command used to update an existing row in a table
DELETE WHERE	Command used to delete a specific row in a table
DROP TABLE	Command used to drop a specific table
CREATE TABLE	Command used to create a new table with an optional primary key
CREATE INDEX	Command used to create an index for a specific table
Monitor Row Count	Command used to get the number of rows in a table and the current time stamp
Monitor Row Count Change	Command used to get the row count difference in a table compared to the previous execution. The calculated row count and the current time stamp is returned

Connection Hooks

Connection hooks defines optional SQL commands that are sent to the database at connect and just before disconnect. They are typically used to initialize the database session with custom settings and to clean up various resources at disconnect.

Property	Description
Run SQL at Connect	Defines the SQL to be executed just after the connection has been established
Run SQL at Disconnect	Defines the SQL to be executed just before the connection will be disconnected

Objects Tree Labels

Property	Description
Custom Object Tree Labels	Here you can define custom tree labels for the data nodes in the database objects tree. The Object Type must match the corresponding type in the actual database profile, see more below.

The label for a **data node** (e.g., a table or view node, as opposed to a node that just groups nodes, such as the Tables node) is typically the name of the database object the node represents, e.g., the table or view name. In some cases, you may want to extend the label to include other information, such as the name of the schema that the object belongs to. To do this, you can use a custom tree label, defined in the Objects Tree properties category.

You need two pieces of information to define a custom label: the Object Type name for the data node, and the names of the variables that hold the information you want to use in the label. You find this information in the **<ObjectsTreeDef>** element in the database profile XML file (described in

detail in the [Database Profile Framework](#) section) for the database type you want to modify. Using the database profile for the JavaDB/Derby database type as an example, a stripped down version of the `<ObjectsTreeDef>` element looks like this:

```
<ObjectsTreeDef id="derby">
  <GroupNode type="Schemas" label="Schemas">
    <DataNode type="Schema" label="${derby.getSchemas.Schema}">
      <SetVar name="schema" value="${derby.getSchemas.Schema}"/>
      <SetVar name="schemaId" value="${derby.getSchemas.Schema Id}"/>
      [...]
    <GroupNode type="Tables" label="Tables">
      <DataNode type="Table" label="${derby.getTables.Table Name}" isLeaf="true">
        <SetVar name="objectname" value="${derby.getTables.Table Name}"/>
        <SetVar name="rowcount" value="true"/>
        <SetVar name="acceptInQB" value="true"/>
        [...]
      </DataNode>
    </GroupNode>
    [...]
  </DataNode>
  [...]
</ObjectsTreeDef>
```

In this example, there is one `<DataNode>` element with a `type` attribute set to `Schema`, with a nested `<DataNode>` element with a `type` attribute set to `Table`. These two elements represent data nodes, for the schema and table node, respectively, and the `type` attribute value is the Object Type name you need to bind the custom label to an object type.

Each `<DataNode>` element also has a number of nested `<SetVar>` elements, declaring the variables you can use in the custom label value. All variables declared for the object type node and those declared for a parent `<DataNode>` element can be used in the label. So, if you want the label for table nodes in the tree to show both the schema name and the table name, you add a custom label declaration like this:

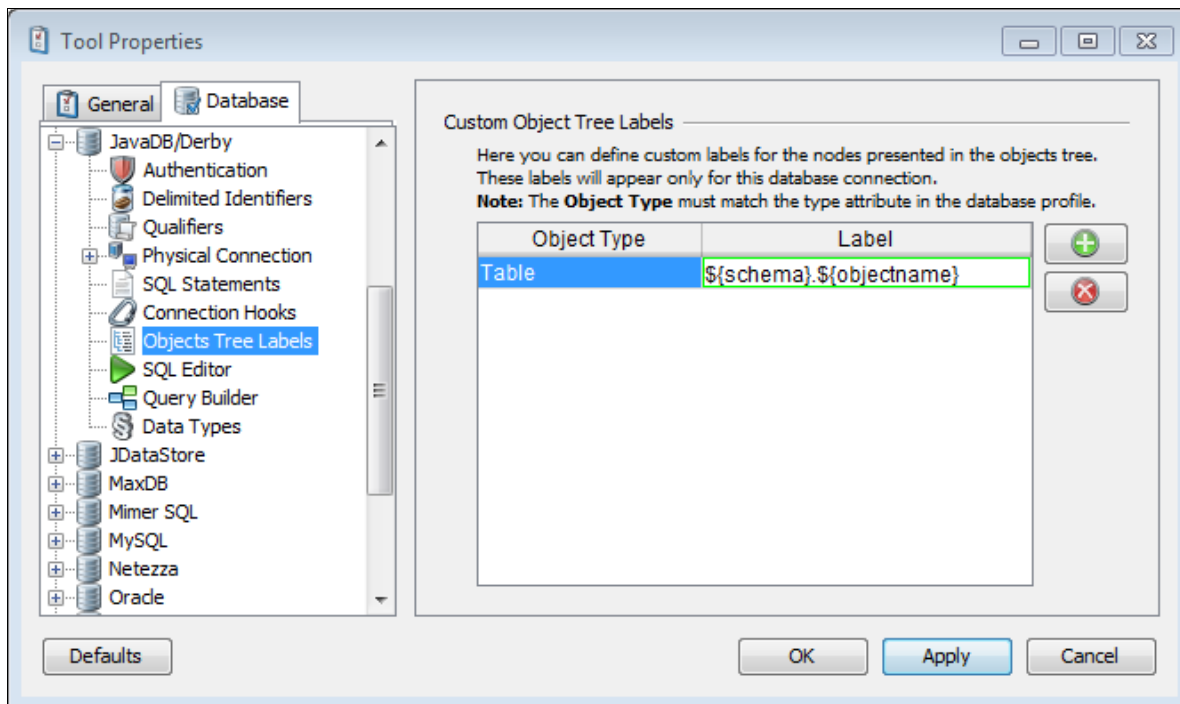


Figure: Custom label declaration

SQL Editor

Property

Description

Set Current Schema If enabled, changing the schema in the SQL Commander also changes the default schema for the database connection, so that unqualified table names in any SQL statement are associated with the selected schema. If this property is not enabled, changing the schema only affects the schemas used for auto-completion.
Note: Only a few databases supports setting the default schema for an opened connection. This property is only shown for database types that support it.

Query Builder

Property	Description
Query Builder Auto-Join Properties	<p>With auto-join enabled, the Query Builder automatically joins tables as they are included in the query, based on the specified column matching rule: FK/PK declarations or columns with matching names in different tables.</p> <p>Specifies whether the Query Builder generates joins as JOIN clauses or WHERE conditions. JOIN clause:</p> <pre>SELECT * FROM HR.EMPLOYEES emp INNER JOIN HR.DEPARTMENTS dept ON (emp.DEPARTMENT_ID = dept.DEPARTMENT_ID)</pre>
Generate JOIN clauses in Query Builder	<p>WHERE condition:</p> <pre>SELECT * FROM HR.EMPLOYEES emp, HR.DEPARTMENTS dept WHERE (emp.DEPARTMENT_ID = dept.DEPARTMENT_ID)</pre>

Database Specific settings

DbVisualizer provides more support for some databases than for others, and so requires extended configuration capabilities for these databases.

Data Types (Oracle)

With Oracle, the DATE data type should sometimes be handled as TIMESTAMP. Enable **Handle DATE as TIMESTAMP** and DbVisualizer will convert DATE into TIMESTAMP objects.

Data Types (DB2 and JavaDB/Derby)

DB2 and JavaDB/Derby supports a data type named **CHAR FOR BIT DATA**. If you want to see values of this type as text, enable this property.

Explain Plan (Oracle, SQL Server and DB2)

The explain plan feature supported for Oracle, SQL Server and DB2 can be configured to highlight certain threshold levels.

Property	Description
Color Critical Nodes	If enabled, critical nodes in the explain plan are highlighted.

Critical Threshold Specifies the threshold for when a node should be handled as critical

Warning Threshold Specifies the threshold for when a node should be handled as a warning

Explain Plan (Oracle)

The explain plan feature for Oracle can be configured to define the management of the underlying plan table in which the explain plan result is stored.

Explain Plan (DB2)

The explain plan feature for DB2 can be configured to define the management of the underlying plan tables in which the explain plan result is stored.

Objects Tree (Oracle)

Property	Description
Show Empty Schemas	If disabled, only schemas that contain database objects are shown in the tree. Note: Only disable this if you have DBA permissions, otherwise no schemas as listed,
System View Prefix	Select here whether the database profile for Oracle should retrieve database information from the DBA or ALL system tables. Note: If choosing DBA, make sure the appropriate privileges are granted for the user you are connecting as.

Export and Import

Introduction

You can export both schema objects and data from DbVisualizer to a file. With the Export Schema feature, you can export the DDL and/or data for all or selected objects in a database schema. The Export Table feature offers the same options as Export Schema but for a single table.

The Export Data feature writes different types of data presented in DbVisualizer, such as text and graphs, to a file. The Export Data Wizard dialog looks different depending on whether [grid](#), [graph](#) or [chart](#) data is being exported.

The following sections describe the options available for each of these cases. There are major differences between DbVisualizer Free and DbVisualizer Personal when exporting objects and data. This document explains the complete functionality in the Personal edition, some of which is not available in DbVisualizer Free.

Exporting very large result sets using the standard Export Data feature may fail due to running out of memory, since all data must first be presented in DbVisualizer. The [@export](#) client side command in the SQL Commander solves this problem, since it exports the data on the fly while it is fetched from the database.

The Import Table Data feature reads data stored in **CSV** (Character Separated Values) format from files.

Export Schema

Sometimes you may need to copy a schema from one database to another, or compare two similar schema to see how they differ. The Export Schema feature can help you with tasks like these. This feature writes the DDL and/or the table data for all or selected database objects in a schema to a file or another destination.

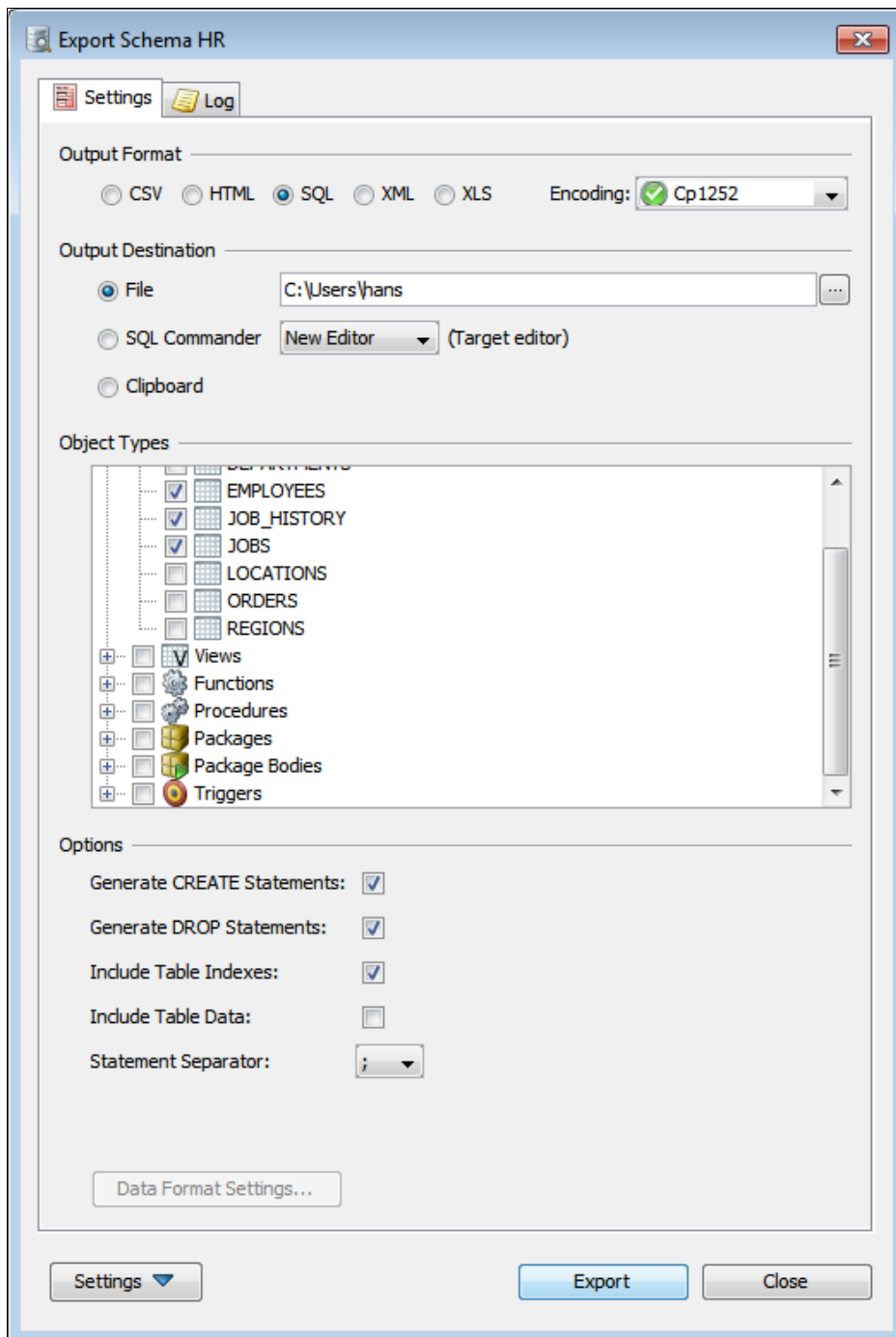


Figure: The Export Schema dialog

You launch the Export Schema dialog by selecting the schema you want to export in the object tree and choosing **Export Schema** either from the right-click menu or from the **Actions** menu.

The following sections describe the different options you can use. When you are happy with all the settings, click **Export** to start the process. Log messages are displayed during the export process.

Output Format

You can export the schema objects in a number of formats. With the SQL or XML formats you can export the DDL for all supported object types as well as the data for tables, while the other formats only apply to table data.

If you choose SQL, the objects are exported as DDL statements (CREATE TABLE, CREATE VIEW, etc.) and, if you choose to include table data, as INSERT statements. This is the format to use if you want to recreate the schema somewhere else.

If you want to compare one schema to another, you may want to pick the XML format instead. The object declarations are then exported as XML documents, like this example:

```
<?xml version="1.0" encoding="MacRoman"?>
<TABLE>
  <SCHEMA>
    HR
  </SCHEMA>
  <NAME>
    JOBS
  </NAME>
  <COLUMNS>
    <COLUMN>
      <NAME>
        JOB_ID
      </NAME>
      <DATA_TYPE>
        VARCHAR(10)
      </DATA_TYPE>
    </COLUMN>
    <COLUMN>
      <NAME>
        JOB_TITLE
      </NAME>
      <DATA_TYPE>
        VARCHAR(35)
      </DATA_TYPE>
    </COLUMN>
    <COLUMN>
      <NAME>
        MIN_SALARY
      </NAME>
      <DATA_TYPE>
        INTEGER
      </DATA_TYPE>
      <NULLABLE/>
    </COLUMN>
    <COLUMN>
      <NAME>
        MAX_SALARY
      </NAME>
      <DATA_TYPE>
        INTEGER
      </DATA_TYPE>
      <NULLABLE/>
    </COLUMN>
  </COLUMNS>
  <CONSTRAINTS>
    <CONSTRAINT>
      <NAME>
        JOB_ID_PK
      </NAME>
      <TYPE>
        PRIMARY KEY
      </TYPE>
      <COLUMNS>
        <COLUMN>
          <NAME>
            JOB_ID
          </NAME>
        </COLUMN>
      </COLUMNS>
    </CONSTRAINT>
    <CONSTRAINT>
      <NAME>
```

```
JOB_TITLE_NN
</NAME>
<TYPE>
CHECK
</TYPE>
<EXPRESSION>
"JOB_TITLE" IS NOT NULL
</EXPRESSION>
</CONSTRAINT>
</CONSTRAINTS>
</TABLE>
```

The encoding choice specifies which character encoding to use for the data when you export to a file, and it is also used as the encoding in XML header when you use the XML format. The default choice is based on your systems default encoding.

The table data is exported in the same format as described for [Export Grid Data](#) below.

Output Destination

Destination	Description
File	This option outputs the data to the named file.
SQL Commander	This option writes the export data to an SQL Commander editor. It is primarily useful when exporting with the SQL output format.
Clipboard	Exporting to the (system) clipboard is convenient if you want to use the exported data in another application without the extra step of exporting to file first.

Object Types

In the Object Types area, you select the object types or individual objects you want to export. Checking the check box for a type, e.g. Tables, selects all objects of that type. Expand the type node to select individual objects instead, e.g. just a few tables.

Options

The Options area contains different options depending on the selected Output Format. Most options are the same as for [Export Grid Data](#), but for the SQL format you can also choose to Generate CREATE and DROP Statements and to Include Table Data and Table Indexes for the exported tables. Similarly, with XML format you can choose to include the DDL, table indexes and table data.

If you choose to include table data, you can also change how the values for different data types are formatted in the output by clicking the **Data Format Settings** button.

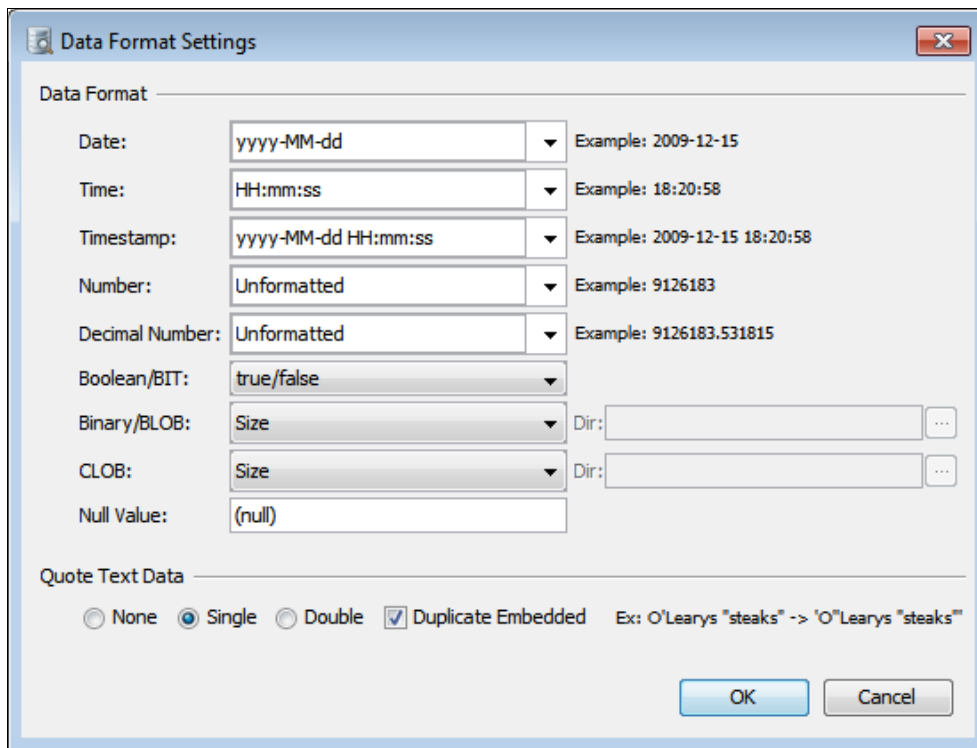


Figure: The Data Format Settings dialog

Settings

Clicking the Settings button reveals a menu with options for saving and loading settings to and from a file.

- **Save as Default Settings**
Saves all format settings as default. These are then loaded automatically when DbVisualizer is started
- **Use Default Settings**
Use this choice to initialize the settings with default values
- **Load**
Use this choice to open the file choose dialog, in which you can select a settings file
- **Save As**
Use this choice to save the settings to a file
- **Copy Settings to Clipboard**
Use this choice to copy all settings to the system clipboard. These can then be pasted into the SQL Commander to define the settings for the `@export editor commands`.

Logging

By default, log messages about the export process are shown in the Log tab. If you instead want to write the messages to a file, open the **Log** tab and specify the file before clicking **Export**.

Export Table

When you select a table node in the objects tree, you can open the Export Table dialog from the right-click menu or the **Actions** menu. It has exactly the same options as [Export Schema](#), except that it only exports the selected table.

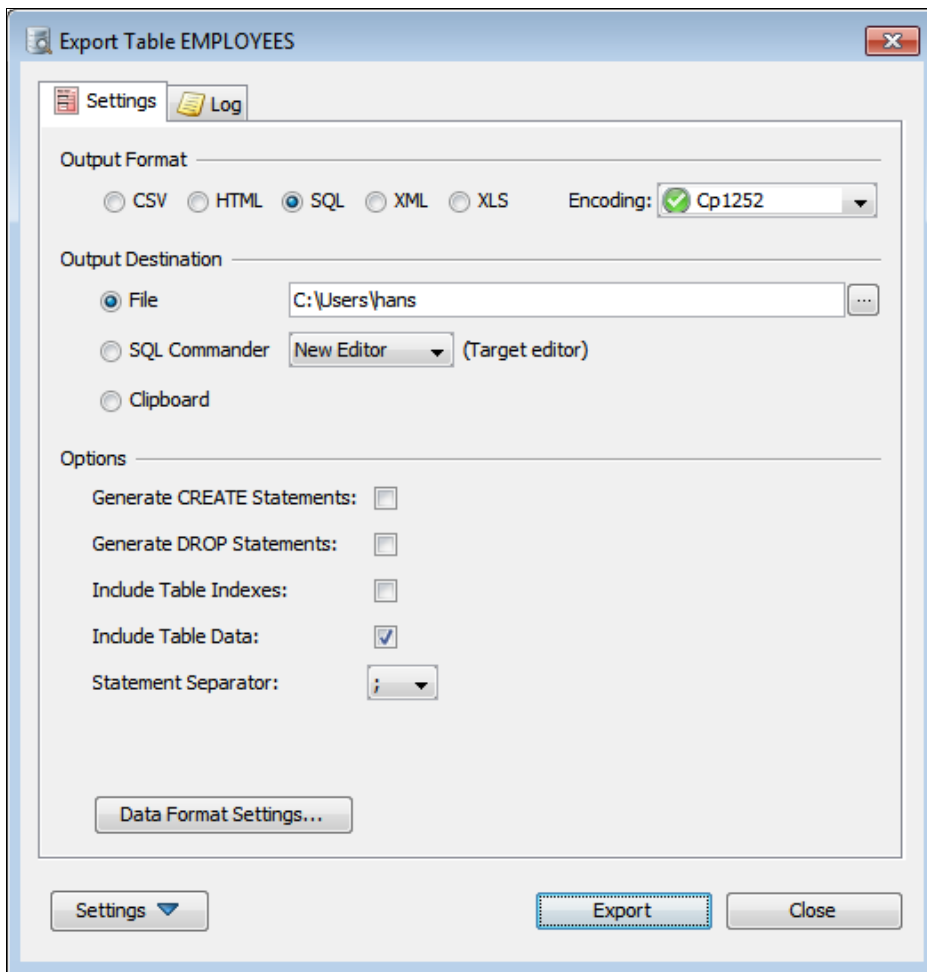



Figure: The Export Table dialog

Export Grid data

The Export wizard is launched using the **Export** button in the grid toolbar () or from the grid's right-click menu. If you want to export just some of the grid rows and columns instead of all data in the grid, select the data to export and launch the wizard with the **Export Selection** right-click menu choice.

Settings page

The first wizard page is the Settings page, containing general properties for how the exported data should be formatted. All settings in the settings page can be saved to a file for later use in the export wizard or in the SQL Commander when exporting result sets using the [@export editor command](#).

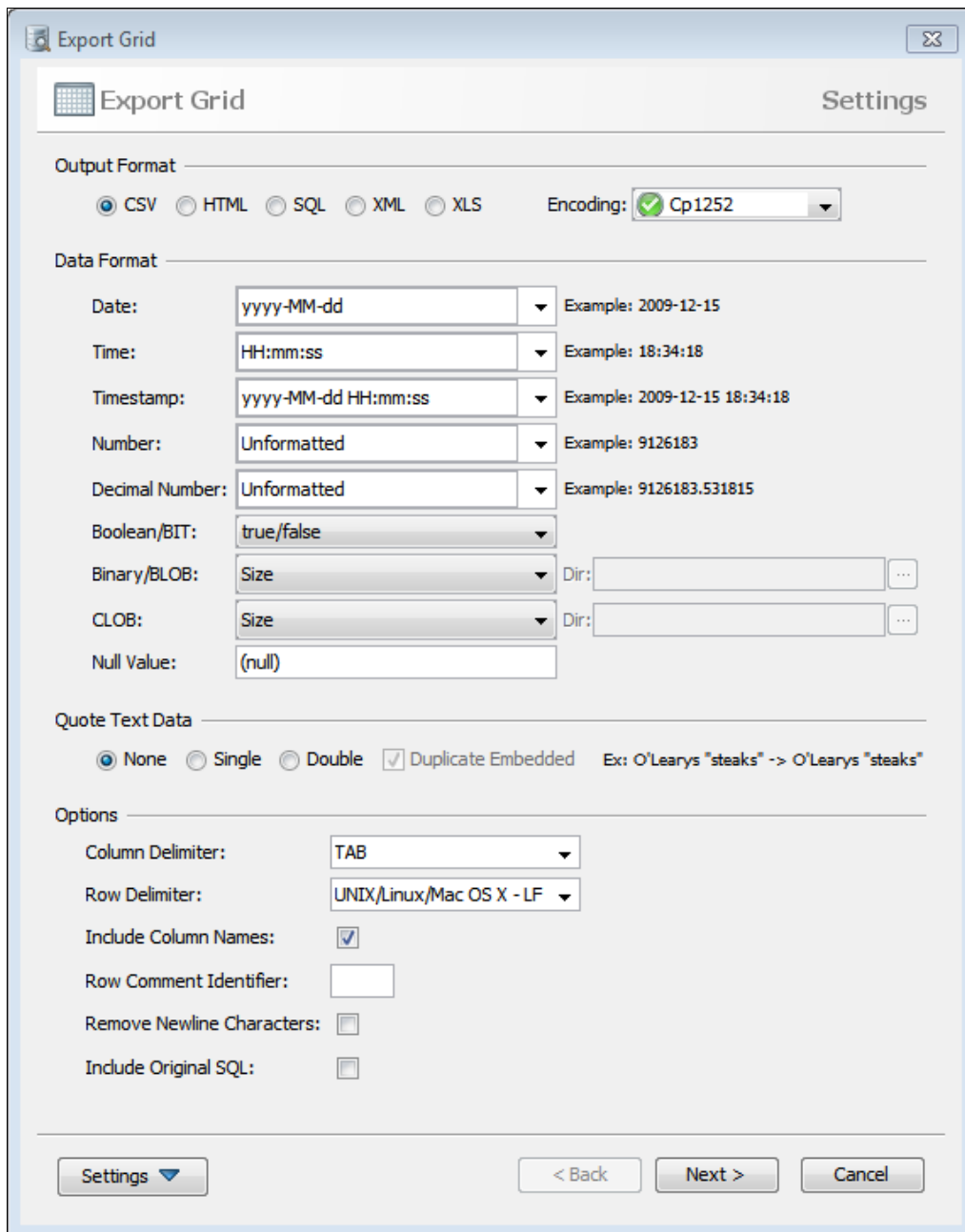


Figure: The grid export wizard

Read the sections below for detailed information about each field and the settings you can use.

Output Format

Grid data can be exported in the following formats.

Format	Description
CSV	The CSV format (Character Separated Values) is used to export the grid of data to a file in which each column is separated with one or more characters. You may also specify the row delimiter (aka newline sequence of characters).

```
5,Hepp,59248
15,Hopp,41993
```

```
16,Hupp,44115
```

The above example use a "," as the column delimiter and a "\n" sequence as the row delimiter (invisible above).

HTML The data is exported in HTML format using the <TABLE> and associated tags.

The SQL format simply creates an SQL INSERT statement for each row in the grid. It also uses the column names from the grid to define the column list in the SQL statement.

SQL

```
insert into table1 (Column1, Column2, Column3) values (5, 'Hepp', 59248);
insert into table1 (Column1, Column2, Column3) values (15, 'Hopp', 41993);
insert into table1 (Column1, Column2, Column3) values (16, 'Hupp', 44115);
```

The XML format is handy when importing or using the exported data in an XML enabled application. The default structure of the XML format is:

XML

```
<ROWSET>
  <ROW>
    <Column1>5</Column1>
    <Column2>Hepp</Column2>
    <Column3>59248</Column3>
  </ROW>
  <ROW>
    <Column1>15</Column1>
    <Column2>Hopp</Column2>
    <Column3>41993</Column3>
  </ROW>
  <ROW>
    <Column1>16</Column1>
    <Column2>Hupp</Column2>
    <Column3>44115</Column3>
  </ROW>
</ROWSET>
```

Alternatively, you can choose between the commonly used XmlDataSet and FlatXmlDataSet formats.

XLS Use the XLS format if you want to work with the exported data in Microsoft Excel or a compatible spreadsheet application, such as Open Office.

Encoding

The encoding choice specifies which character encoding to use for the data. It is also used to set the encoding in the HTML and XML headers. The default choice is based on your systems default encoding.

Data Format

The data format settings define how the data for each of the data types will be formatted.

Quote Text Data

Defines whether text data should appear between quotes. Use the Duplicate Embedded option to properly deal with text that contains the quote character when you export as SQL or CSV.

Options

The options section is used to define settings that are specific for the selected output format.

CSV

Options

Column Delimiter:

Row Delimiter:

Include Column Names:

Row Comment Identifier:

Remove Newline Characters:

Include Original SQL:

Figure: CSV specific export options

HTML

Options

Title:

Description:

Include Original SQL:

Figure: HTML specific export options

SQL

Options

Table Name:

Statement Separator:

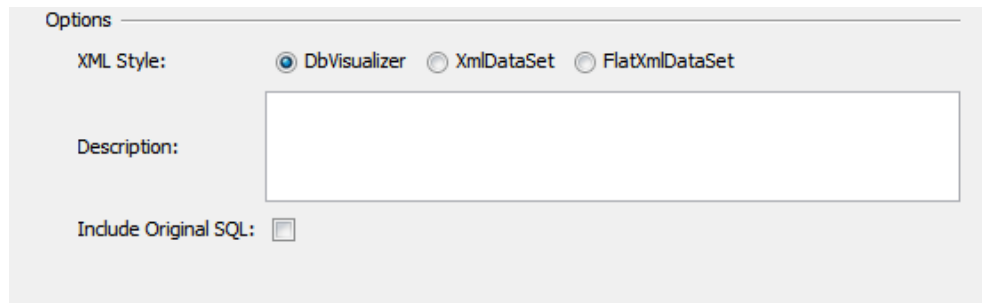
Row Comment Identifier:

Include Basic DDL:

Include Original SQL:

Figure: SQL specific export options

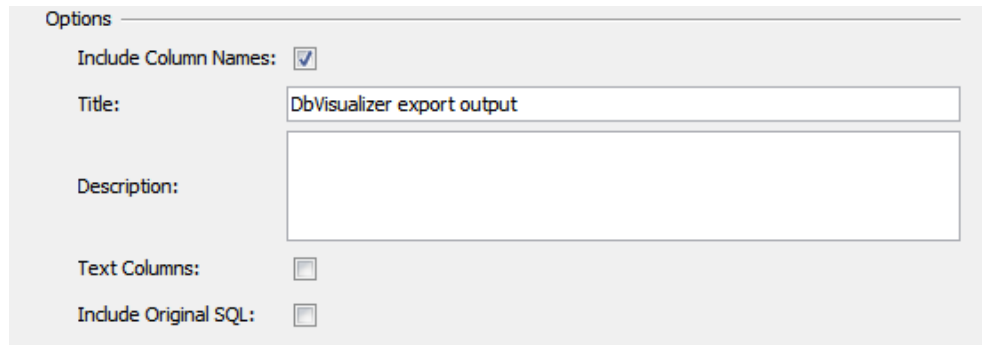
XML



The screenshot shows a dialog box titled "Options" with three radio buttons for "XML Style": "DbVisualizer" (selected), "XmlDataSet", and "FlatXmlDataSet". Below this is a "Description:" label followed by a large empty text area. At the bottom, there is a checkbox for "Include Original SQL:" which is currently unchecked.

Figure: XML specific export options

XLS



The screenshot shows a dialog box titled "Options" with a checked checkbox for "Include Column Names:". Below this is a "Title:" label followed by a text box containing "DbVisualizer export output". Underneath is a "Description:" label followed by a large empty text area. At the bottom, there are two checkboxes: "Text Columns:" (unchecked) and "Include Original SQL:" (unchecked).

Settings

Clicking the Settings button reveals a menu with options for saving and loading settings to and from a file:

- **Save as Default Settings**
Saves all settings as default. These are then loaded automatically when DbVisualizer is started
- **Use Default Settings**
Use this choice to initialize the settings with default values. Some of the settings will be fetched from the general tool properties dialog.
- **Load**
Use this choice to open the file chooser dialog, in which you can select a settings file
- **Save As**
Use this choice to save the settings to a file
- **Copy Settings to Clipboard**
Use this choice to copy all settings to the system clipboard. These can then be pasted into the SQL Commander to define the settings for the `@export` editor commands.

Data page

Clicking the Next button in the wizards moves you to the Data page. Use the columns list to control which columns to export and how to format the data for each columns. The list is exactly the same as the column headers in the original grid, i.e., if a column was manually removed from the grid before launching the Export wizard, then it will not appear in this list.

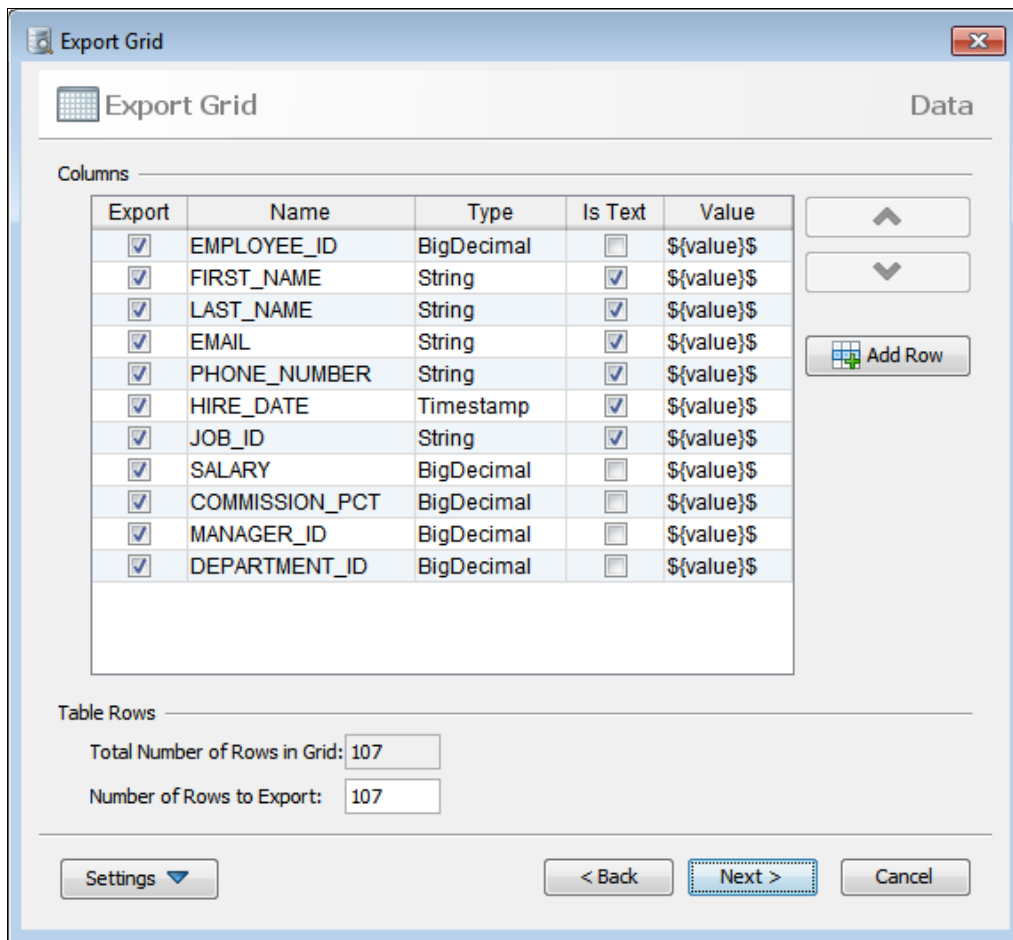


Figure: The grid export wizard

The **Table Rows** fields show you how many rows are available and let you specify the number of rows to export. This setting along with the **Add Row** button is especially useful when you use the test data generation feature described in the next section.

Here follows information about the columns in the list.

Field	Description
Export	Defines whether the column will be exported or not. Uncheck it to ignore the column in the exported output.
Name	The name of the column. This is used if exporting in HTML, XML, XLS or SQL format. Column headers are optional in the CSV output format.
Type	The internal DbVisualizer type for the column. This type is used to determine if the column is a text column (i.e., if the data should be enclosed by quotes or not).
Text	Specifies if the column is considered to be a text column (this is determined based on the type) and so if the value should be enclosed in quotes.
Value	The default `\${value}` variable is simply be substituted with the column value in the exported output. You can enter additional static text in the value field. This is also the place where any test data generators are defined.

Generate Test Data

The test data generator is useful when you need to add random column data to the exported output.

The Value field specifies the data to be in the exported output. By default, it contains the `#{value}#` variable, which is simply replaced with the real column value during the export process. You can also add static values before and after the `#{value}#` field, to be exported as entered.

Alternatively, you can use test data generator variables in the Value field. The choices are available in the right-click menu when you edit the Value field.

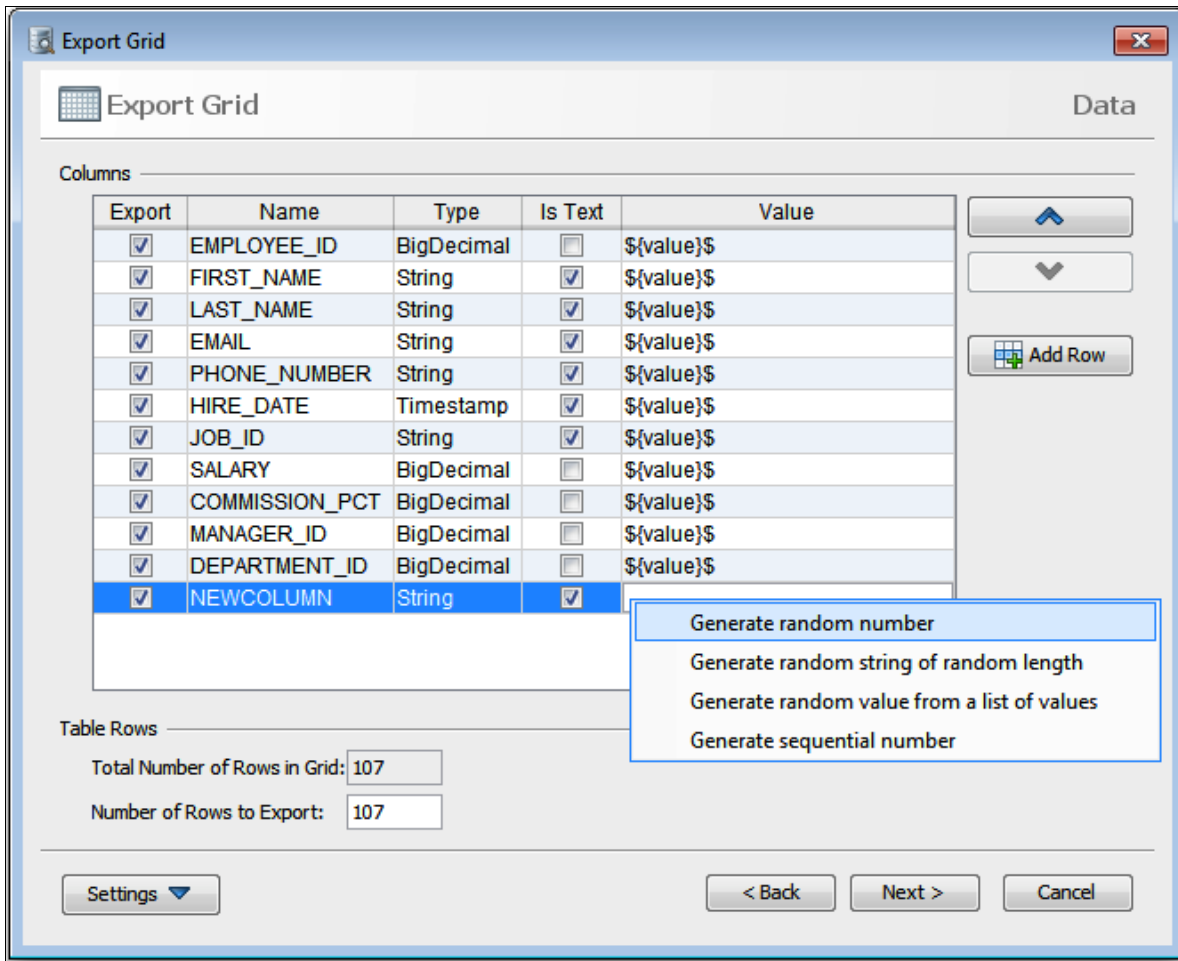


Figure: Right-click menu with the test data generator functions

Function Name	Function Call	Example
Generate random number	<code>#{var randomnumber(1, 2147483647)}#</code>	Generates a random number between 1 and 2147483647
Generate random string of random size	<code>#{var randomtext(1, 10)}#</code>	Generates random text with a length between 1 and 10 characters
Generate random value from a list of values	<code>#{var randomenum(v1, v2, v3, v4, v5)}#</code>	Picks one of the listed values in random order
Generate sequential number	<code>#{var number(1, 2147483647, 1)}#</code>	Generates a sequential number starting from 1. The generator re-starts at 1 when 2147483647 is reached. The number is increased with 1 every time a new value is generated.

Test data generator example

Here is an example of how to use the test data generators to try out planned changes to the data. Consider this initial data:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	Smith	Clerk	7902	1980-12-17 00:00:00	800	(null)	20
2	7499	Allen	Salesman	7698	1981-02-20 00:00:00	1600	300	30
3	7521	Ward	Salesman	7698	1981-02-22 00:00:00	1250	500	30
4	7566	Jones	Manager	7839	1981-04-02 00:00:00	2975	(null)	20
5	7654	Martin	Salesman	7698	1981-09-28 00:00:00	1250	1400	30
6	7698	Blake	Manager	7839	1981-05-01 00:00:00	2850	(null)	30
7	7782	Clark	Manager	7839	1981-06-09 00:00:00	2450	(null)	10
8	7788	Scott	Analyst	7566	1987-04-19 00:00:00	3000	(null)	20
9	7831	King	President	23	1981-11-17 00:00:00	5000	23	10
10	7844	Turner	Salesman	7698	1981-09-08 00:00:00	1500	0	30
11	7876	Adams	Clerk	7788	1987-05-23 00:00:00	1100	(null)	20
12	7900	James	Clerk	7698	1081-12-03 00:00:00	950	(null)	30
13	7902	Ford	Analyst	7566	1981-12-03 00:00:00	3000	(null)	20
14	7934	Miller	Clerk	7782	1982-01-23 00:00:00	1300	(null)	10

Figure: Sample of grid data

After the changes, the **JOB** column should not appear in the output and the new **JOB_FUNCTION** should contain abbreviated job function codes. To test this, we simply uncheck the **Export** field for **JOB** entry and set the Value for the **JOB_FUNCTION** to use the **Generate random value from a list of values** function.

The screenshot shows the 'Export Grid' dialog box with the following configuration:

Export	Name	Type	Is Text	Value
<input checked="" type="checkbox"/>	EMPNO	BigDecimal	<input type="checkbox"/>	`\${value}`\$
<input checked="" type="checkbox"/>	ENAME	String	<input checked="" type="checkbox"/>	`\${value}`\$
<input type="checkbox"/>	JOB	String	<input checked="" type="checkbox"/>	`\${value}`\$
<input checked="" type="checkbox"/>	JOB_CODE	String	<input checked="" type="checkbox"/>	`\${var randomenum(eng,adm,fin)}\$`
<input checked="" type="checkbox"/>	MGR	BigDecimal	<input type="checkbox"/>	`\${value}`\$
<input checked="" type="checkbox"/>	HIREDATE	Timestamp	<input checked="" type="checkbox"/>	`\${value}`\$
<input checked="" type="checkbox"/>	SAL	BigDecimal	<input type="checkbox"/>	`\${value}`\$
<input checked="" type="checkbox"/>	COMM	BigDecimal	<input type="checkbox"/>	`\${value}`\$
<input checked="" type="checkbox"/>	DEPTNO	BigDecimal	<input type="checkbox"/>	`\${value}`\$

Table Rows configuration:

- Total Number of Rows in Grid: 14
- Number of Rows to Export: 14

Figure: Customized columns list with a generator function

Previewing the data (or exporting it) in CSV format results in this:

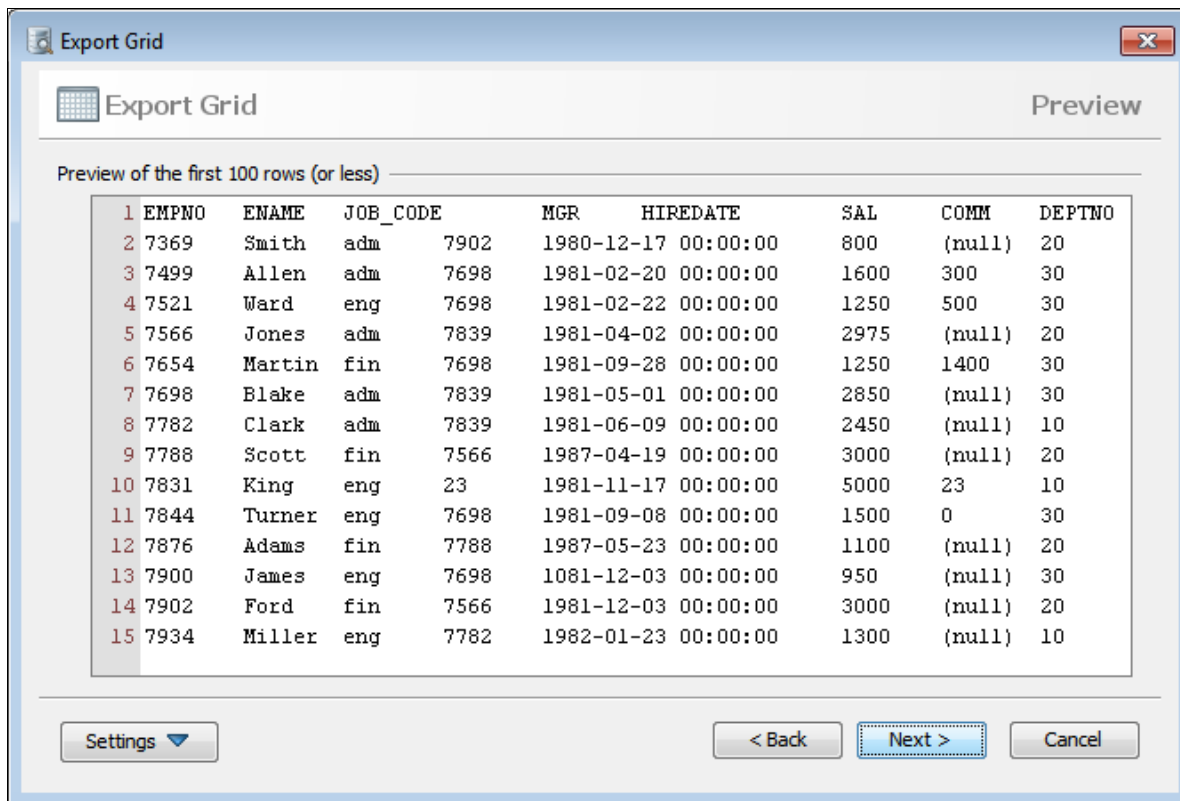


Figure: Result of generated test data

Preview

The third wizard page is the Preview page, showing the first 100 rows of the data as it will appear when it is finally exported. This is useful to verify the data before performing the export process. If the previewed data is not what you expected, just use the back button to modify the settings.

Output Destination

The final wizard page is the Output Destination page. The destination field specifies the target destination for the exported data.

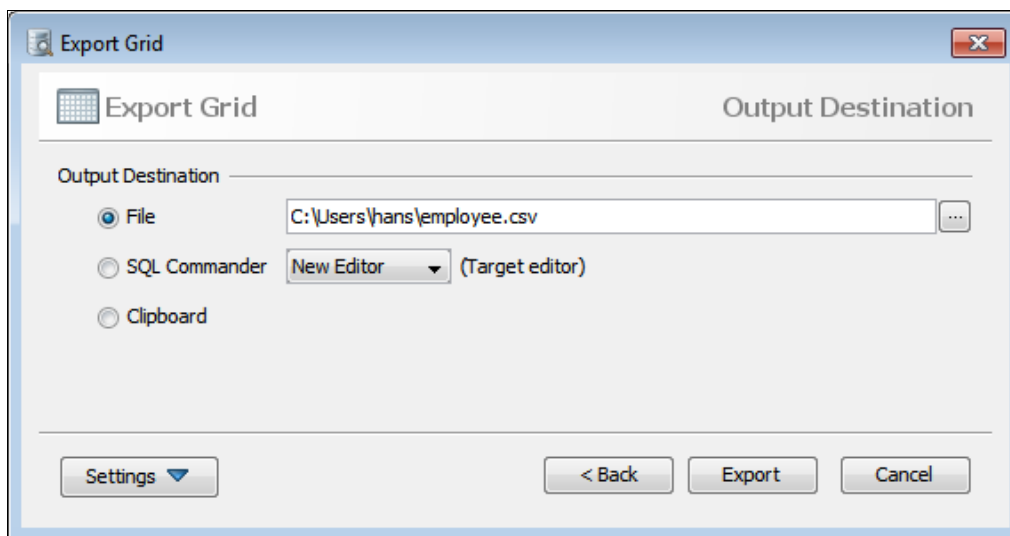


Figure: The output destination and final page for grid export

Destination	Description
File	This option outputs the data to a named file.
SQL Commander	This option writes the export data to an SQL Commander editor. It is primarily useful when exporting with the SQL output format.
Clipboard	Exporting to the (system) clipboard is convenient if you want to use the exported data in another application without the extra step of exporting to file first. CSV formatted data can even be pasted into a spreadsheet application such as Excel or StarOffice, and the cells in the grid will appear as cells in the spreadsheet. Read more about the CSV format in the Format section.

Export Text data

The wizard for exporting result sets in **Text** format is very simple, as it is only possible to specify a file for the exported output. The dialog looks slightly different on different platforms.

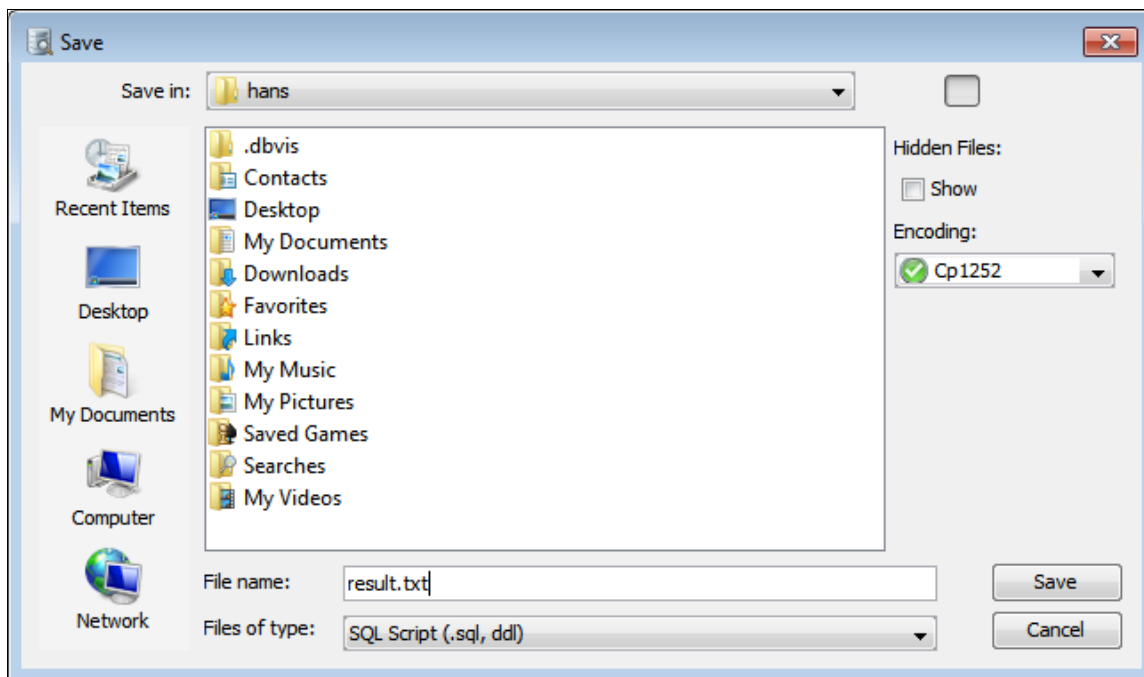


Figure: Export window for text format result sets

Export Graph data

When you export a References or Navigator graph, it is exported with the same zoom level as it appears on the screen. The Export wizard pages when exporting a graph looks like this:

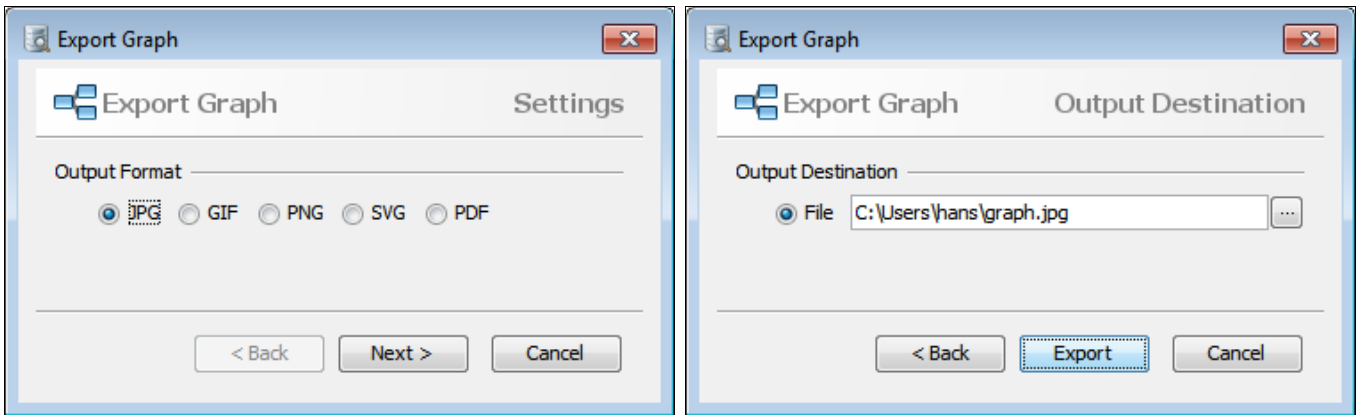


Figure: Export window for graphs

The graph can be exported to a **File** in the **JPEG**, **GIF**, **PNG**, **SVG** or **PDF** formats.

Export Chart data

The options when exporting charts are similar to those for graphs, but in addition you can set the size and orientation to use for the chart in the file.

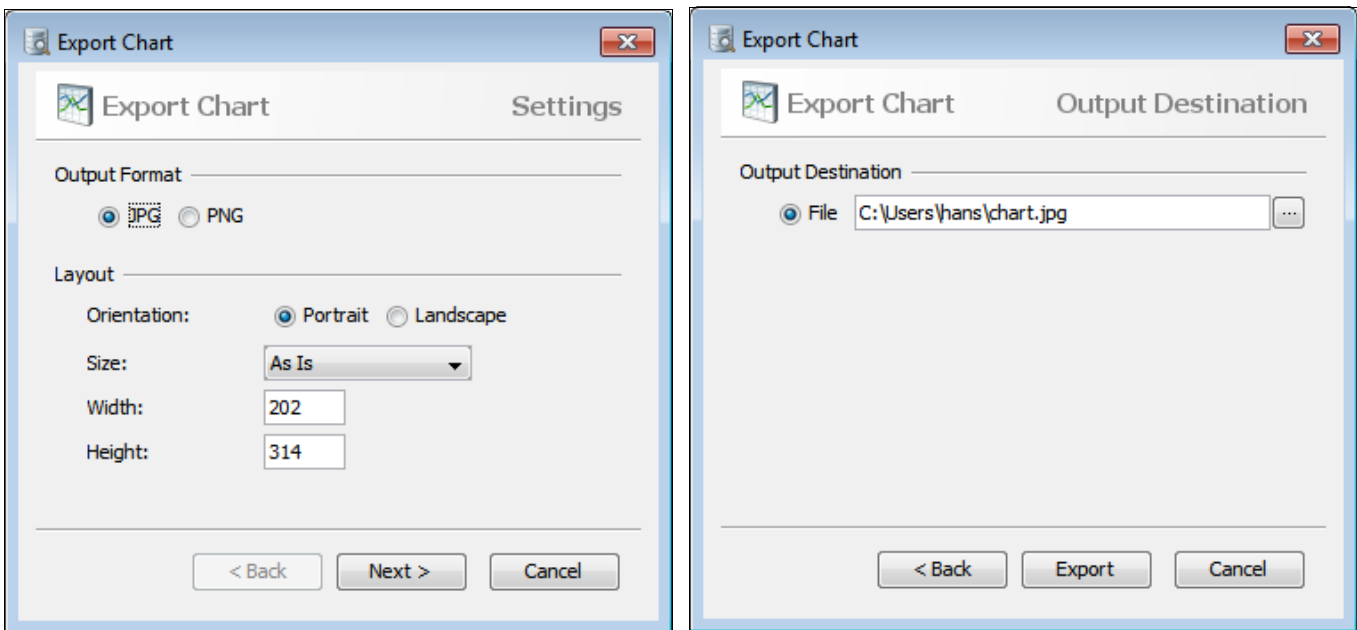


Figure: Export window for charts

A chart can be exported to a **File** in the **JPEG** and **PNG** formats. The optional **Layout** settings are used to control orientation and size of the image. The default width and height are the same as the size of the chart as it appear on the screen. The **Size** list when clicked shows a list of well known paper formats. The **Width** and **Height** are changed to match the selected size. Whether setting the width and height manually or selecting a predefined size, the exported image is scaled accordingly.

Import Table Data

The Import Table Data feature is used to import files containing data organized as rows of columns with separator characters between them, such as CSV files.

Note: The first row in the source file can be used to name the columns.

The destination for the imported data can be a database table or a grid in DbVisualizer. The grid option is convenient for smaller files, as the features available for a DbVisualizer grid can then be used to do various things with the data. An example is that a CSV file can easily be converted into an XML file or a HTML document by importing the data to a grid and then use the Export Wizard in the grid to output the grid data in the desired format.

The Import Wizard can be used in two ways. To import data into an existing table, select the table node in the objects tree and launch the wizard via the right-click menu or via the **Actions** menu.

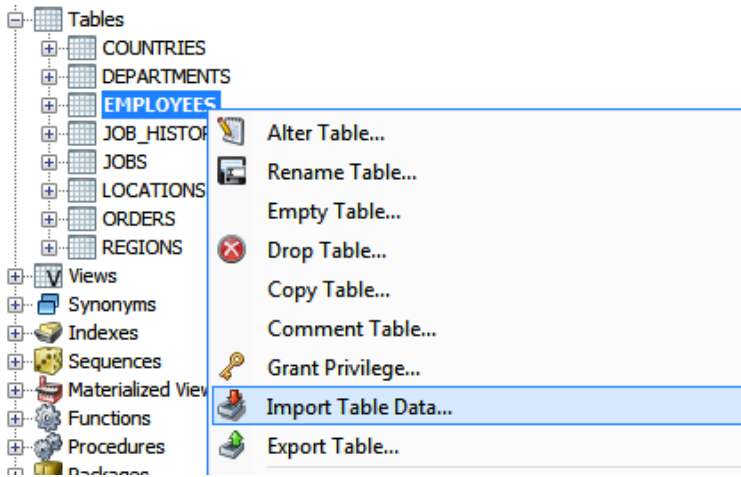


Figure: Import Table Data action in the right-click menu for a table object

If you instead want to create a new table for the imported data, select the Tables node in the objects tree and then launch the wizard.

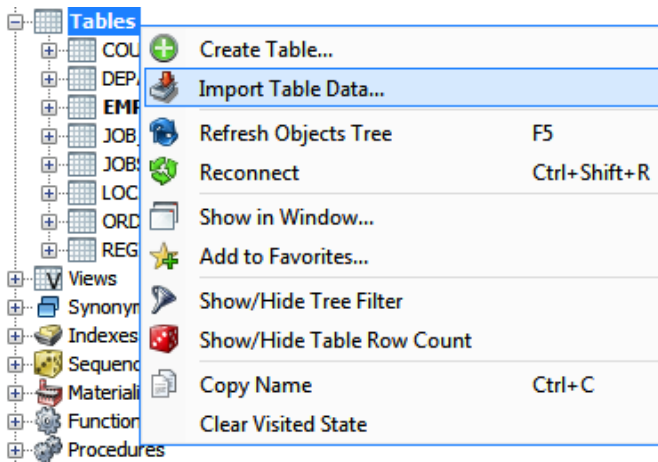


Figure: Import Table Data action in the right-click menu for the Tables object

Source File

In the first wizard page, select the source file to import and then click the Next button.

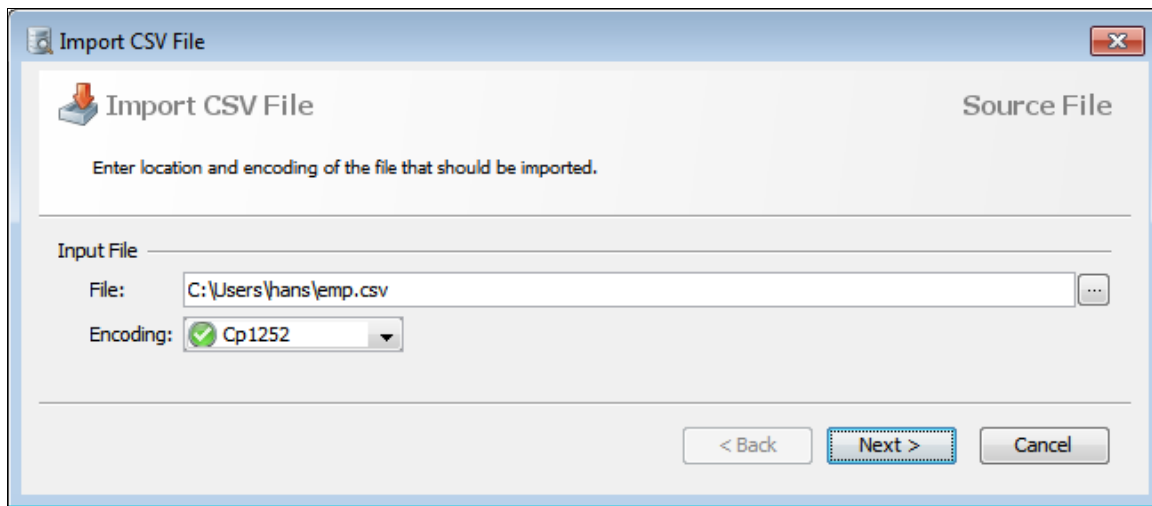


Figure: The Source File import wizard page

Settings

In the Settings page, you specify how the data in the file is organized. The **Data** section at the bottom of the page shows a preview of the parsed data in the **Grid** tab and the original source file in the **File** tab. If a row in the Grid tab is red, it indicates that the row will be ignored during the import process. This happens if setting any of the **Options** settings results in rows not being qualified.

In the Delimiters section, define the character that separates the columns in the file. If you enable **Auto Detect**, DbVisualizer tries the following characters:

- comma ","
- tab "TAB"
- semicolon ";"
- percent "%"

Use the Options section to further define how the data should be read.

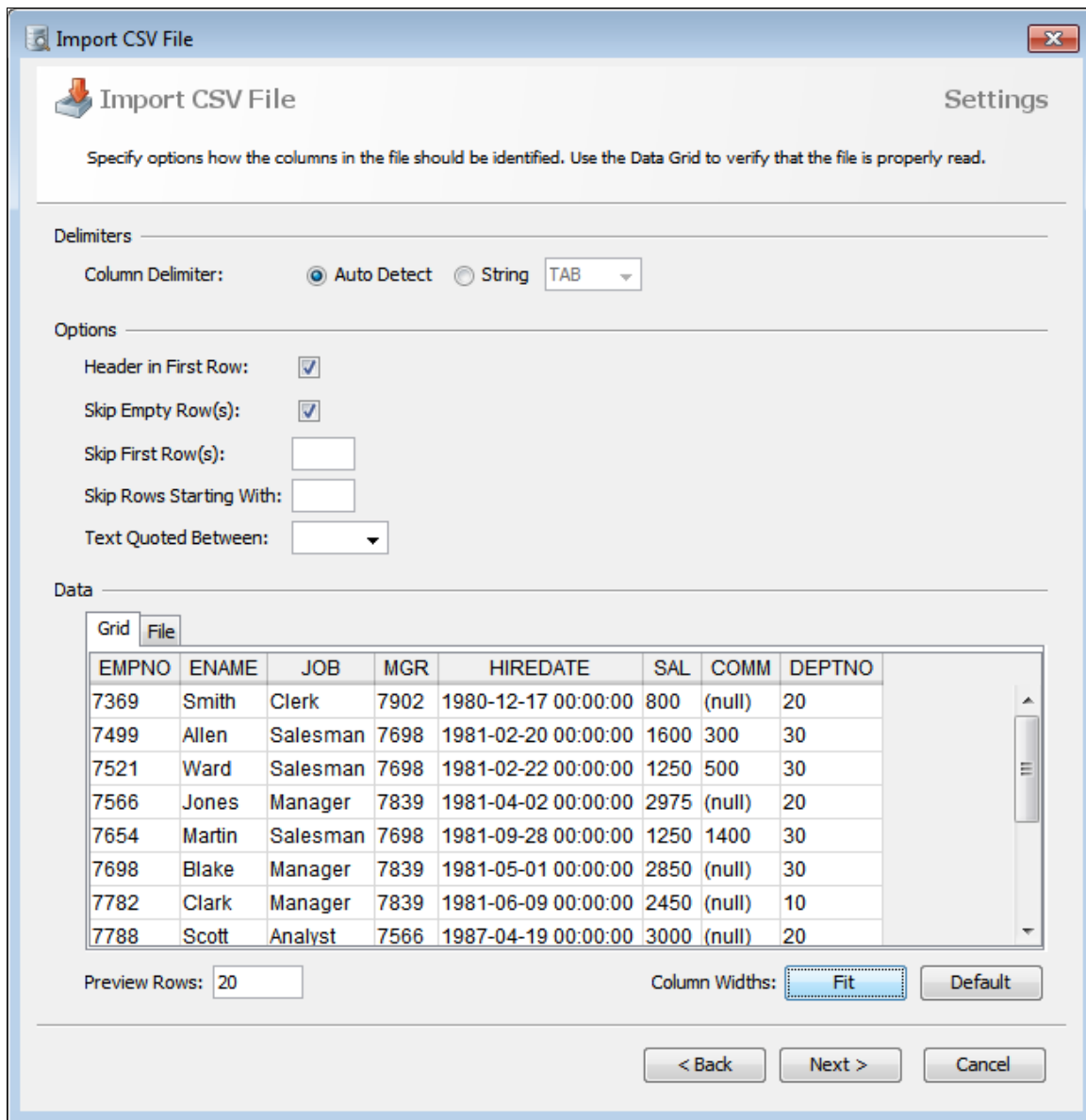


Figure: The Settings wizard page

The following shows the preview grid with some rows in red. The reason is that the **Skip First Row(s)** and **Skip Rows Starting With** are set, i.e., the first two rows and the rows starting with 103 will not be imported.

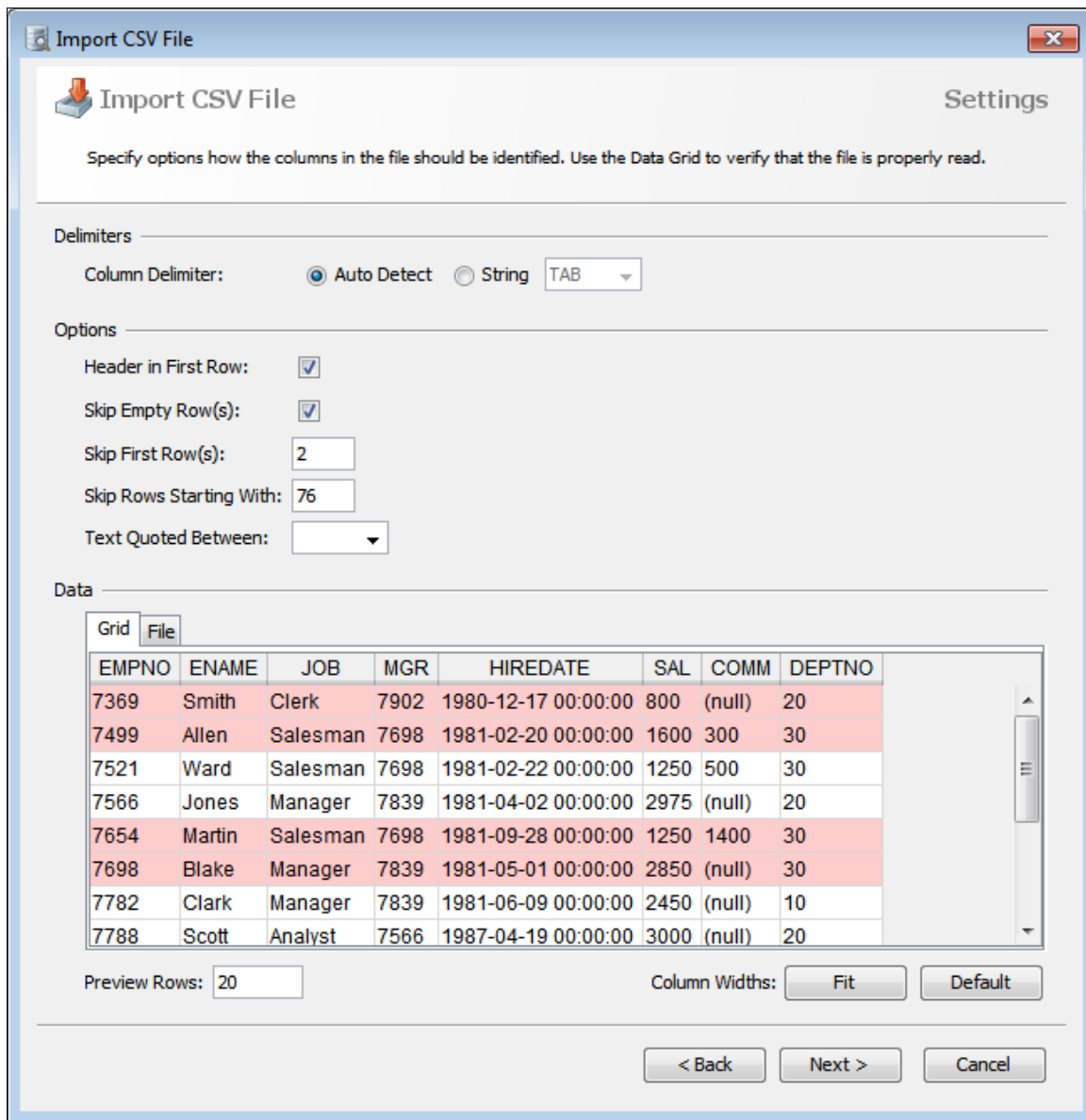


Figure: The Settings wizard page

Data Formats

The Data Formats page is used to define formats for some data types. The first row in the preview grid contains a data type drop-down lists. DbVisualizer tries to determine the data type for each column by looking at the value for the number of rows specified as Preview Rows. If this data type is incorrect, use the drop-down lists to select the appropriate type.

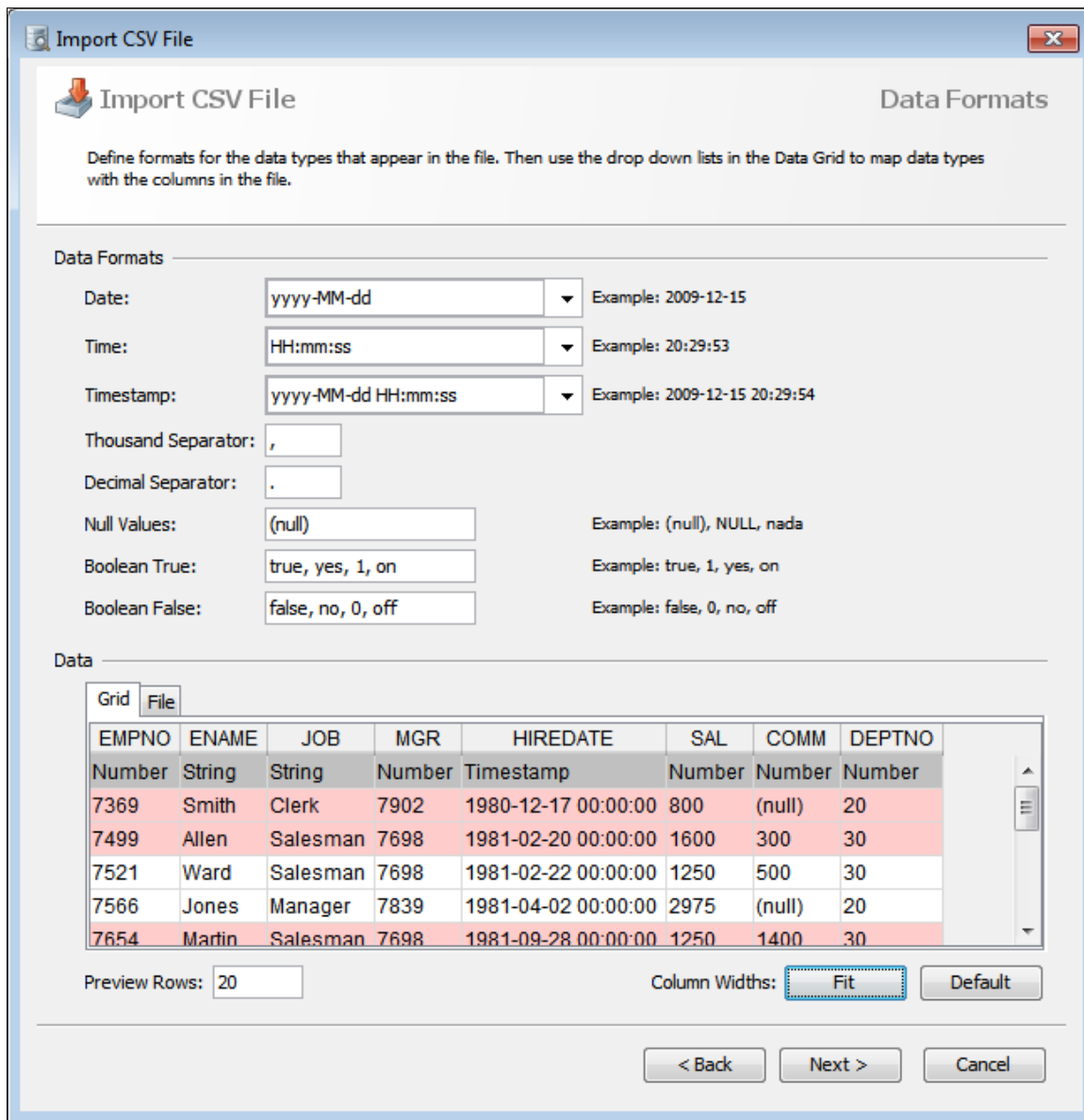


Figure: The Data Formats wizard page

The following is displayed when selecting the drop-down box in the preview grid.

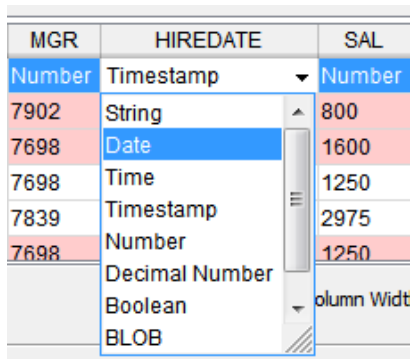


Figure: The data type drop down

Import Destination

The Import Destination page provides two options: **Grid** and **Database Table**. The Grid choice is used to import the data into a grid that will be presented in its own window in DbVisualizer.

When the Database Table choice is selected, the page shows information about the table in which the data will be imported.

If you are importing into an existing table, the Map Table Columns with File Columns grid shows the columns in the selected database table and the columns in the source file.

DbVisualizer automatically associates the columns in the source file with the columns in the target table in the order they appear. If the columns appear in a different order in the file than in the table, but they are named the same, you can use the auto-mapping menu in the upper right corner of the Map Table Columns with File Columns grid to automatically map the columns by name.

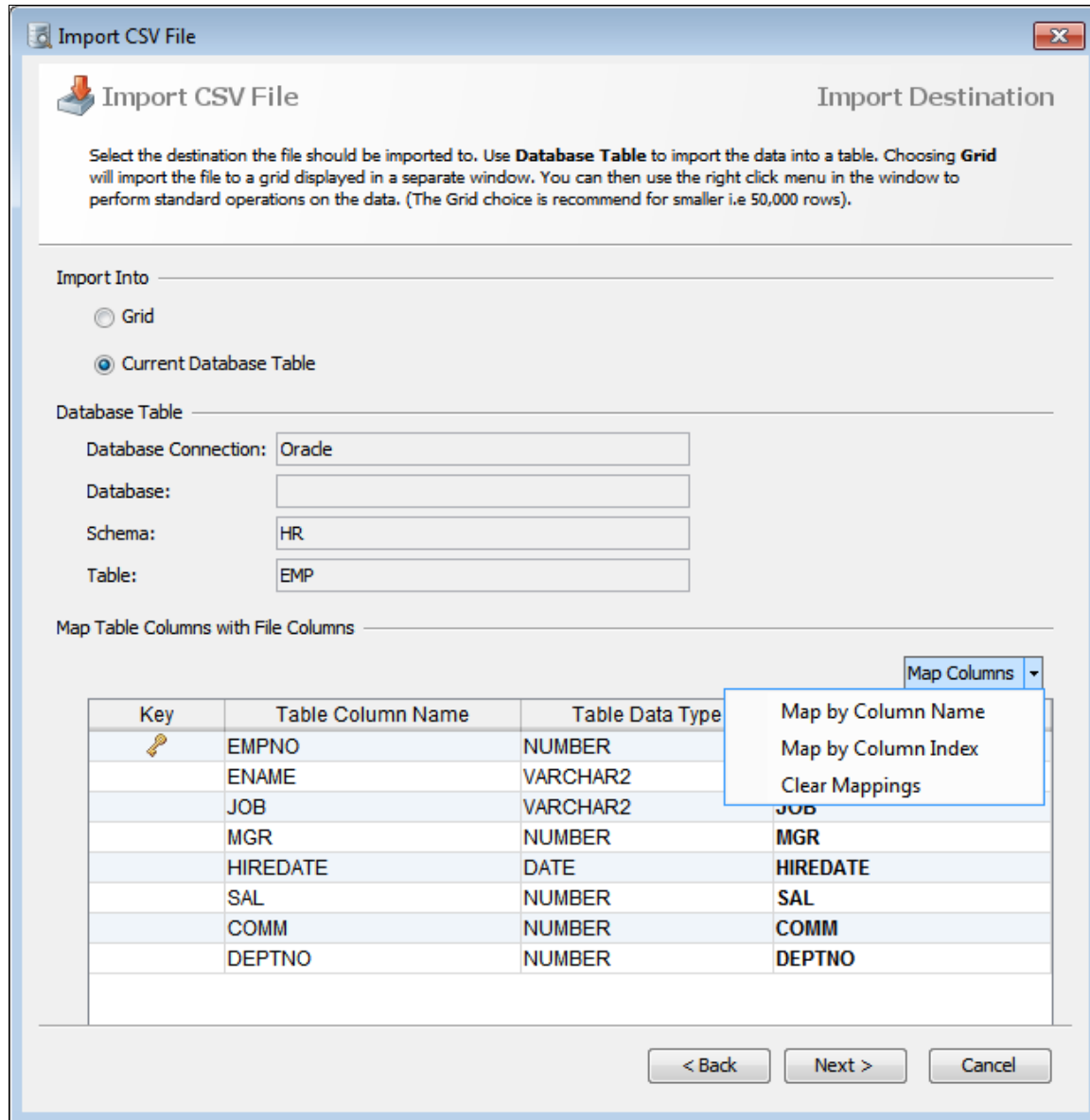


Figure: The auto-mapping menu for import into an existing table

If the column names are different between the file and the table and also appear in different order, you can manually map them using the drop-down lists in the File Column Name field. Choose the empty choice in the columns drop-down to ignore the column during import.

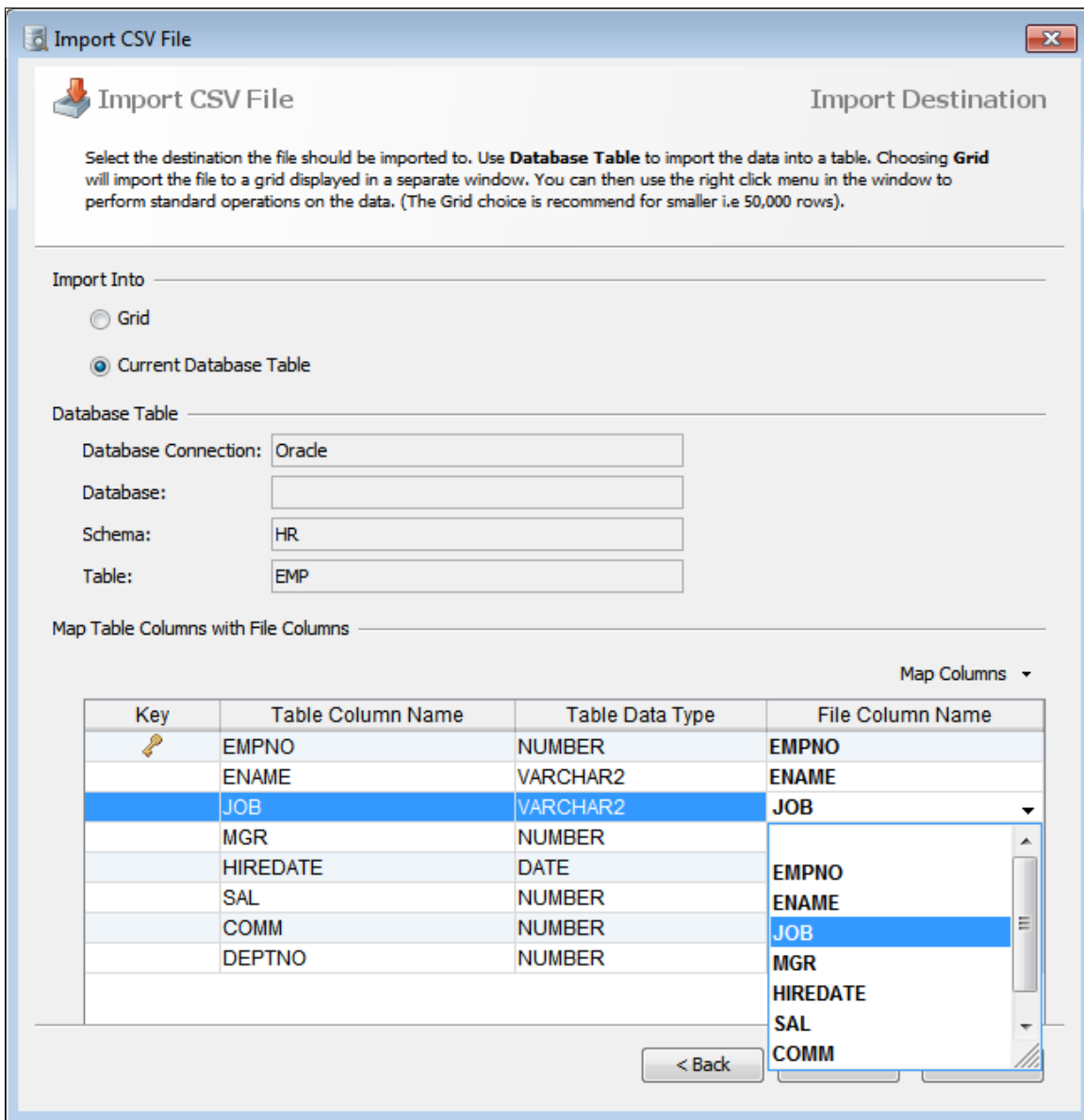


Figure: The column mapping drop down

When you import into a new table, you are presented with a field for the table name and a number of tabs for column and constraint declarations. The Columns tab is filled out based on the source data and the data types from the Data Formats page.

Import CSV File X

Import CSV File Import Destination

Select the destination the file should be imported to. Use **Database Table** to import the data into a table. Choosing **Grid** will import the file to a grid displayed in a separate window. You can then use the right click menu in the window to perform standard operations on the data. (The Grid choice is recommend for smaller i.e 50,000 rows).

Import Into

Grid

New Database Table

New Table Details

Database Connection:

Database:

Schema:

Table:

Columns | Primary Key | Foreign Keys | Unique Constraints | Check Constraints

Name	Data Type	Size	Scale	Nullable	Default
EMPNO	INTEGER			<input type="checkbox"/>	
ENAME	VARCHAR2	6		<input type="checkbox"/>	
JOB	VARCHAR2	9		<input type="checkbox"/>	
MGR	INTEGER			<input type="checkbox"/>	
HIREDATE	DATE			<input type="checkbox"/>	
SAL	INTEGER			<input type="checkbox"/>	
COMM	INTEGER			<input checked="" type="checkbox"/>	
DEPTNO	INTEGER			<input type="checkbox"/>	

< Back Next > Cancel

Figure: The table declaration form for importing into a new table

Note that it is not always possible to find a database specific type for the data format specified on the Data Format page. You must then pick the correct type from the Data Type drop-down menu. The size for string column types may also need to be adjusted. By default, the size is set to the maximum number of characters found for the column in the number of rows specified as Preview Rows. You can ignore certain columns by removing them in the Columns tab. Keys and other constraints can be created using the other tabs.

You can go back to the Data Format page and increase the Preview Rows value if you believe that it will help DbVisualizer to pick better defaults. If you do so, you need to click the Reload button when you come back to this page to rescan the source data and get new default values.

Import Process

The last wizard page is used to start and monitor the import process. Here you can select whether all rows in the source file should be imported or only a portion. You can also specify that you want to log to the GUI or to a file, and that you want keep the window open when the import is completed, so that you can see the log messages when logging to the GUI. If you want to stop the processing on the first error, check the Stop on Error check box.

If any errors occur during the import process, error messages are presented in the log and the window stays open regardless of the Keep Window after Import setting.

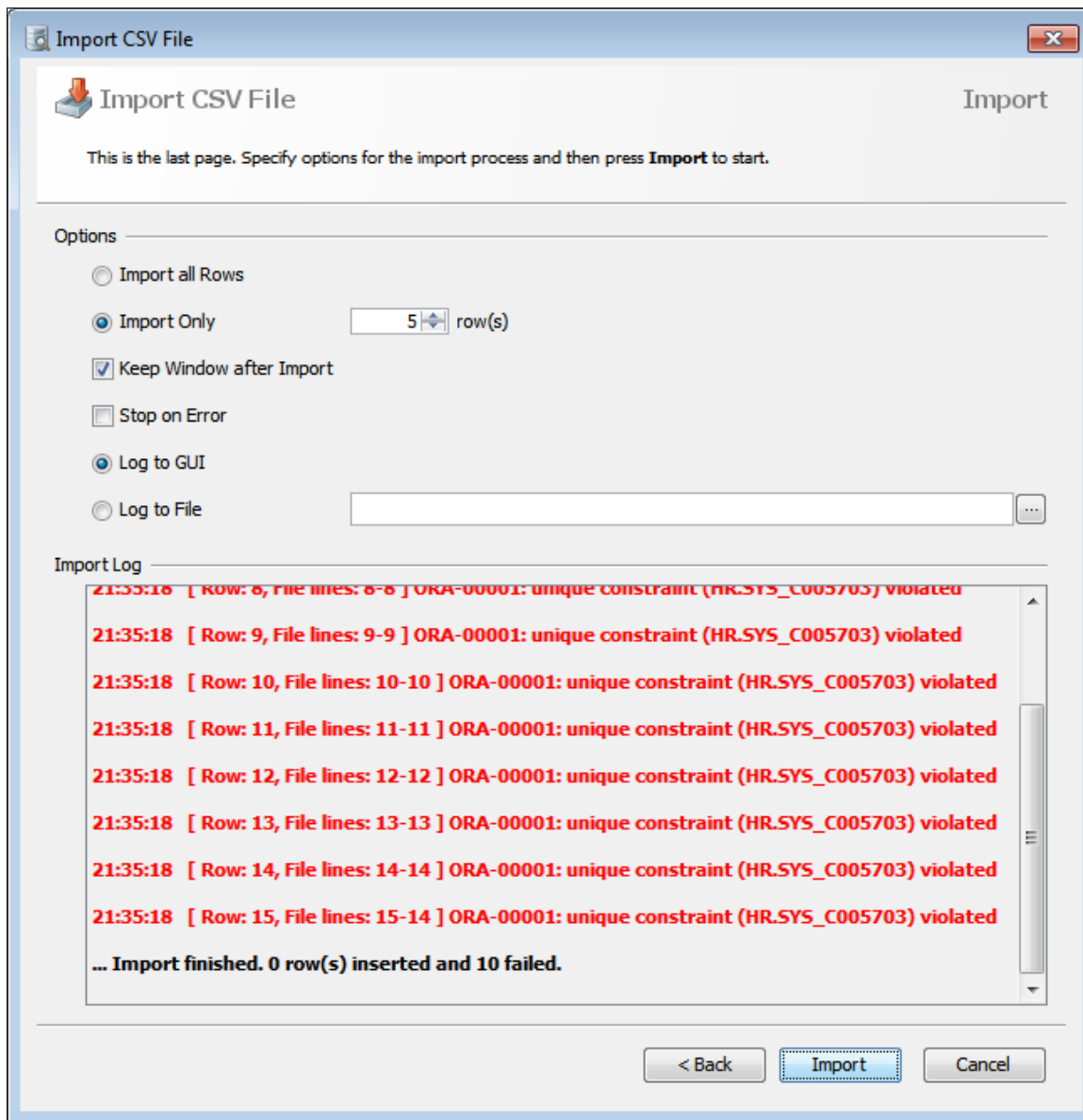


Figure: The import process page

Exporting and Importing Binary/BLOB and CLOB Data

Columns declared as Binary/BLOB and CLOB can be exported and imported using DbVisualizer as SQL or CSV files. The data for each such cell is stored in a separate file, referenced from the SQL or CSV file as a DbVisualizer variable. Here's an example of an SQL INSERT statement with a Binary/BLOB variable:

```
insert into "BLOB_TEST" ("COL1") values (${data1-0||||BinaryData|noshow vl=file}$);
```

Exporting Binary/BLOB and CLOB Data

All of the export dialogs described earlier in this section (Export Schema, Export Table, and Export Grid) can be used to export Binary/BLOB and CLOB data. You enable this by choosing **File** as the data format for Binary/BLOB and/or CLOB data. Optionally, you can specify the directory for the data files. If you do not specify a directory, the operating system's default directory for temporary files (e.g. C:\TEMP or /tmp) is used.

Binary/BLOB:	File	Dir: C:\Users\hans\blobs
CLOB:	File	Dir: C:\Users\hans\blobs

Figure: Data format File for export of Binary/BLOB and CLOB data

Importing Binary/BLOB and CLOB Data

If you have exported Binary/BLOB and CLOB data as an SQL script, you just run the script in the SQL Commander to import it. When the SQL Commander encounters a variable that refers to a file, it reads the file and inserts the content as the column value.

If you exported to a CSV file, use the Import Table Data feature to import it. On the Data Format page, ensure that the format for the source file column is set to BLOB or CLOB.

Data	
id	photo
Number	BLOB
1	<code> \${data1-1} C:\Users\hans\blobs\dbvis5202228372370417084.bin BinaryData no show vl=file}\$ </code>

Figure: Data format BLOB for import of Binary/BLOB data

Using Variables and Exporting to Multiple Files

You can use some of the [pre-defined DbVisualizer variables](#) (`${dbvis-date}$`, `${dbvis-time}$`, `${dbvis-timestamp}$` and `${dbvis-object}$`) in all fields that holds free text (e.g. title and description fields) and as part of the file name in all export dialogs.

Use the `${dbvis-object}$` variable as part of the file name in Export Schema if you want to export the DDL and/or data to a separate file for each object. The variable is replaced with the object type and object name, e.g. `${dbvis_object}.sql` becomes `table_COUNTRIES.sql` for a table named COUNTRIES.

Database Profile Framework

Introduction

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Personal edition.

This document explains the database profile framework which is the base for how DbVisualizer presents information in the **Database Objects tree** and in the **Object View**. In addition, it is also used to define object actions, such as drop, rename, compile, create, comment, alter, etc.

What features in DbVisualizer relies on the database profile?

One of the most important and central features in DbVisualizer is the database objects tree, used to navigate databases, and the object view, showing details about specific objects. The general problem exploring any database is that they are all different with respect to the information describing what's in the database (also called **system tables** or **database meta data**). This basically means that it's rather complex to implement a multi-database support product, such as DbVisualizer, since each database must be handled specifically. All databases also support different object types, apart from the most common ones, such as table, view, index, etc.

The database profile framework is used to simplify the process of defining what information DbVisualizer will display and operate on for a specific database. Technically, a database profile is an XML document with all of the logic, structure and actions easily mapped to the visual components in DbVisualizer. Another great benefit of separating the database specific logic from the implementation of DbVisualizer is that anyone with some degree of domain knowledge can create a database profile. All that is needed is a text editor (preferably with XML support) and some ideas of what should be the final result.

A great source for inspiration (except for this document) is all the existing database profiles that comes with DbVisualizer. All database profiles are (and must be) stored in the **DBVIS-HOME/resources/profiles** directory (this path is OS dependent).

The following figure illustrates which features in DbVisualizer are controlled by the database profile.

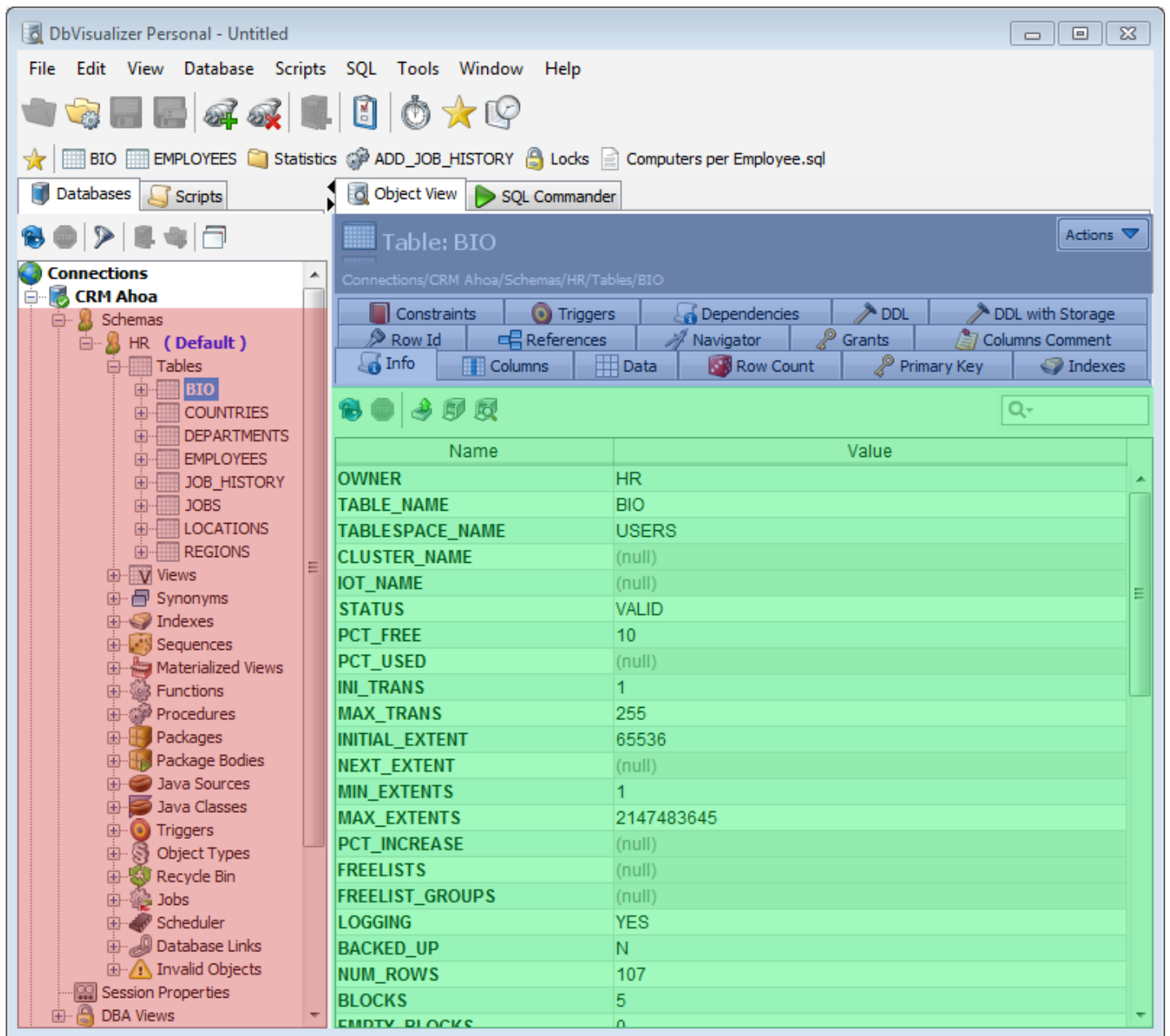


Figure: What the database profile controls in DbVisualizer

The red box at the left shows the **database objects tree**. This tree is used to navigate the objects in the database. Selecting an object in the tree shows the **object view** (blue box) for the selected object type. An object view may have several **data views** (green), showing object information. DbVisualizer shows these as labeled **tabs**. The green box in the screenshot shows the content of the data view labeled **Columns**. The type of viewer that is presenting the data in the screenshot is the **grid** viewer. Read more about all data viewers in the [Viewers](#) section.

Common to both the database objects tree and the object view are the **SQL commands** that are used to fetch the information from the database. The associated SQL is executed by DbVisualizer whenever a node in the tree is expanded (to expose any child objects) or when a node is selected (to fill the object data views).

Right-clicking the mouse on an object in the tree or clicking the **Actions** button in the object view shows a menu with all valid actions for the selected object. These are also defined per database profile and object type. Read more about the capabilities of actions in the [Definition of user actions](#) section.

How does DbVisualizer know what database profile to use?

DbVisualizer automatically load the appropriate database profile (XML file) based on the following:

1. The **Database Type** for the database connection is matched with the information in the **DBVIS-HOME/resources/database-mappings.xml** file to find out if there is a database profile available. If it finds one, it is used.
2. If there is no matching profile, the **generic** profile is used. This is very basic profile and shows only rudimentary information about the objects in the database. This is also the profile used in the DbVisualizer Free edition for all databases.

A specific database profile can be selected manually for a database connection. This is done in the database connection properties. Manually choosing a profile requires that the profile supports the actual database. If it doesn't, various errors will be reported once the database objects tree is explored. (Whenever the profile is changed, you must reconnect the database connection).

The name of the loaded profile is listed in the **Connection** tab status bar when the connection has been established. You can click the profile link to display the **Database Profile** list.

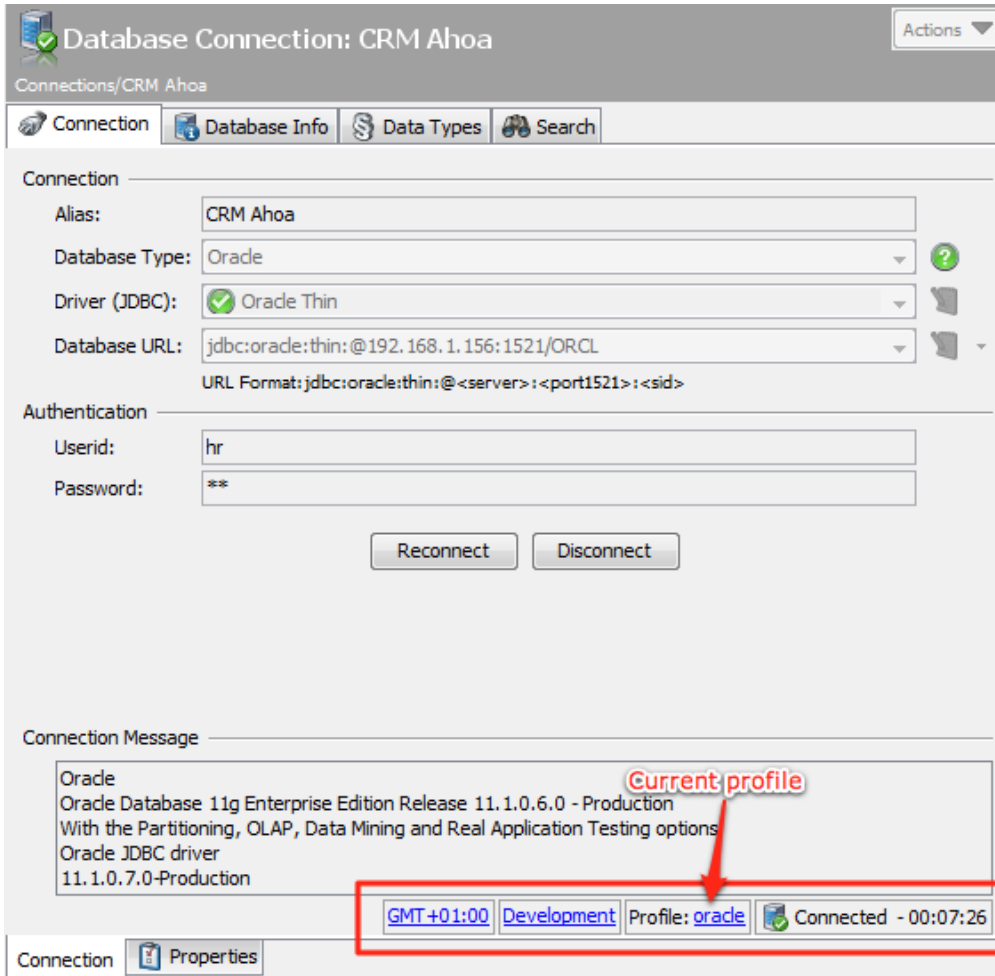


Figure: The status bar in the Connection tab when connected

XML structure

The mapping from the visual components in the user interface described earlier and the element definitions in the XML file is, briefly, as follows:

- The database objects tree (green box) is described by the **ObjectsTreeDef** root element. (The Database Connections node is mandatory and its appearance cannot be controlled by the profile).
- The object views (green and blue boxes) are described by the **ObjectsViewDef** root element.
- The commands used to execute the SQL to get the information for **ObjectsTreeDef**, **ObjectsViewDef** and optionally **ObjectsActionDef** definitions are defined by the **Commands** root element.
- All **Actions** for an object are defined in the **ObjectsActionDef** root element. (Actions are optional).

The XML for a database profile is quite simple, but there are a few things that need to be highlighted. All database connections loads a database profile from an XML file. If there is no matching database profile, the **generic** profile is used. This profile uses the standard JDBC metadata calls in order to obtain information about the structure and objects in the database. The generic profile is not one XML file, as the database specific profiles are, but instead four files:

- **generic-commands.xml**
- **generic-actions.xml**
- **generic-tree.xml**
- **generic-view.xml**

All these files are referred to in the **generic.xml** file via include statements, i.e., each of the above files are included in the generic.xml file when it is loaded. The reason for this file organization is that the four files above can also be included and extended in a specialized profile. See later for more information.

The XML structure used to represent the database profile is as follows (click on the link to read more about each specific section):

- **Commands**
Defines the SQLs for the **ObjectsTreeDef**, **ObjectsViewDef** and optionally **ObjectsActionDef**.
- **ObjectsActionDef** (optional)
Defines actions for object types.
- **ObjectsTreeDef**
Defines the structure and what objects should be visible in the objects tree.
- **ObjectsViewDef**
Defines the object views for a specific object type.

XML skeleton

The following is a minimal database profile XML file, showing its structure.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE DatabaseProfile SYSTEM "dbvis-defs.dtd" [
    <!ENTITY generic-commands SYSTEM "generic-commands.xml">
    <!ENTITY generic-view SYSTEM "generic-view.xml">
  ]>
<DatabaseProfile desc="Profile for Sybase ASE"
  version="$Revision: 11720 $"
  date="$Date: 2009-12-22 15:54:30 +0100 (Tis, 22 Dec 2009) $"
  minver="6.5.7">

<!-- ===== -->
<!-- Definition of the commands -->
<!-- ===== -->
  <Commands>
    &generic-commands;
    ...
  </Commands>
<!-- ===== -->
<!-- Definition of the object actions that are used by the tree -->
<!-- ===== -->
  <ObjectsActionDef>
    ...
  </ObjectsActionDef>
<!-- ===== -->
<!-- Definition of the database objects tree structure -->
<!-- ===== -->
  <ObjectsTreeDef id="sybase-ase">
    ...
  </ObjectsTreeDef>
<!-- ===== -->
<!-- Definition of the database objects views -->
<!-- ===== -->
  <!-- Include the generic-view -->
  &generic-view;
  <ObjectsViewDef id="sybase-ase" extends="generic">
    ...
  </ObjectsViewDef>
</DatabaseProfile>
```

The name of the XML file (sybase-ase) and the values for the **id** attribute for the **ObjectsTreeDef** and **ObjectsViewDef** elements must be the same.

The first rows in the XML defines external dependencies and their URIs. The **DOCTYPE** identifier defines the DTD that is used to validate the XML. The **ENTITY** identifiers lists URIs for external references. In this case they identify the **generic-commands.xml** and **generic-view.xml** files. They can then be referenced in the XML as **&generic-commands;** and **&generic-view;**, which simply means that the related XML files are included in the final document when the profile is loaded.

The root of the database profile is the **DatabaseProfile** element. Continue to the next sections for information about the elements forming the database profile.

Tip: If you are using an XML editor to edit the profile it is very convenient to load the DTD in the editor, as you will then get color and error highlighting.

<DatabaseProfile>

The **DatabaseProfile** is the root element in the XML file. It is required and have the following attributes.

```
<DatabaseProfile desc="Profile for Sybase ASE"
  version="$Revision: 11720 $"
  date="$Date: 2009-12-22 15:54:30 +0100 (Tis, 22 Dec 2009) $"
  minver="6.5.7">
  ...
</DatabaseProfile>
```

The attributes specified for the **DatabaseProfile** element appear in the **Database Profile** list when selecting the connection properties for a database connection:

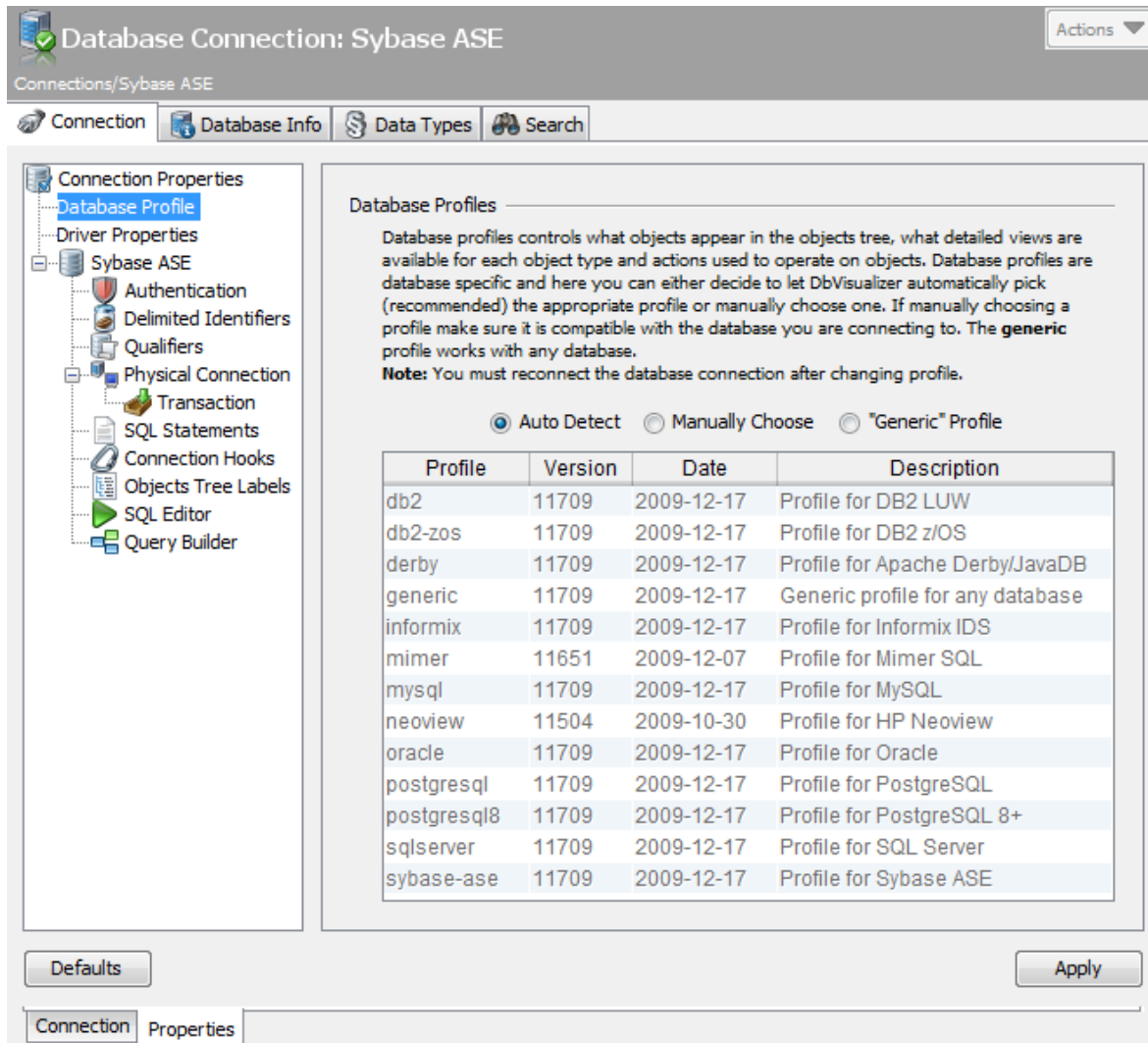


Figure: The list of available database profiles

<InitCommands> - Initialization commands

The **InitCommands** section define commands that are executed when the database profile is first loaded. These commands are typically used to determine characteristics of the target database. The result is stored in variables that can be used in conditions that are evaluated when the rest of the profile is loaded. A common use case is to find out the authorization level of the current user as defined by the database. If the user have limited privileges then make sure the supported object types, views and available actions in DbVisualizer matches the authorization.

Multiple commands may be defined in the InitCommands element and these are executed in serie from top. Conditional processing is also supported.

The following sample is from the HP Neoview database profile. The main purpose with its InitCommands is to first determine the database version by quering a system table (this information is not properly available by the Neoview JDBC driver). Based on the database version a condition controls which of two queries will be executed to find out a property from the database. The result of the executed query is stored in a the **METACAT** variable.

```
<InitCommands>
  <Command id="neoview.getDbVersion" method="runBeforeConditionsEval">
    <SQL>
      <![CDATA[
SELECT SUBSTRING(SYSTEM_VERSION FROM 10)
FROM (GET VERSION OF SYSTEM) V(SYSTEM_VERSION)
      ]]>
    </SQL>
    <Output id="DBVERSION" index="1"/>
  </Command>
  <Command id="neoview.getMaster">
    <If test="#DBVERSION gte 2400">
      <SQL>
        <![CDATA[
SELECT MIN(SYSTEM_CATALOGS) AS MASTER_CAT
FROM (GET SYSTEM_CATALOGS) V(SYSTEM_CATALOGS)
WHERE SYSTEM_CATALOGS LIKE _IS088591'NONSTOP_SQLMX_%'
        ]]>
      </SQL>
    </If>
    <Else>
      <SQL>
        <![CDATA[
SELECT 'NONSTOP_SQLMX_#{@dp.METACAT}'
FROM (VALUES(1)) AS T1
        ]]>
      </SQL>
    </Else>
    <Output id="METACAT" index="1"/>
  </Command>
</InitCommands>
```

Commands in InitCommands are processed in two phases, the first phase execute commands that have the **method="runBeforeConditionsEval"** attribute set. After the first execution phase are all conditions evaluated and processed, the last execution phase will take care of all commands with no **method="runBeforeConditionsEval"** attribute set.

Here is an example how the **METACAT** variable is used in the rest of the database profile:

```
<Command id="neoview.getCatalogs">
  <SQL>
    <![CDATA[
SELECT
  TRIM(CAT_NAME) AS CATALOG_NAME
FROM
  ${METACAT}.SYSTEM_SCHEMA.CATSYS C
WHERE
  CAT_NAME NOT LIKE _IS088591'NONSTOP_SQLMX_%'
  AND CAT_NAME NOT IN (_IS088591'NSMWEB', _IS088591'NVSCRIPT', _IS088591'METRIC',
    _IS088591'MATRIX', _IS088591'GENUSCAT', _IS088591'MANAGEABILITY')
ORDER BY
  CATALOG_NAME
FOR READ UNCOMMITTED ACCESS
```

```
]]>
</SQL>
</Command>
```

<Commands> - The SQLs used to interact with the database

This element contains all **Command** elements with SQL sub element. A **Command** element is identified by a unique **id** attribute, which is then referred in **ObjectsTreeDef**, **ObjectsViewDef** and (optionally) **ObjectsActionDef** definitions.

```
<Commands>
  &generic-commands;
  <Command>
    . . .
  </Command>
</Commands>
```

The first statement in the **<Commands>** element is:

```
&generic-commands;
```

This means that the **generic-commands** entity defined at the top of the XML file is included in the XML i.e., all its definitions are accessible from the **ObjectsTreeDef**, **ObjectsViewDef** and **ObjectsActionDef**. If you don't plan to use any of the generic command, simply ignore this include statement.

<Command>

The **Command** element specifies the SQL associated with the command. In most cases, the SQL should return a result set with 0 or several rows. (The exception is actions which not necessarily need to return a result set, e.g., a "drop" action). The following command queries for login information in Sybase ASE.

```
<Command id="sybase-ase.getLogins">
  <SQL>
    <![CDATA[
select name "Name", suid "SUID", dbname "Default Database", fullname "Full Name",
language "Default Language", totcpu "CPU Time", totio "I/O Time", pwdate "Password Set"
from master.dbo.syslogins order by 1
    ]]>
  </SQL>
</Command>
```

The **id** for this command is **sybase-ase.getLogins**. The reason for prefixing the id with the name of the profile is for maintainability. Since the **generic-commands.xml** file is included in most profiles, it is good to use unique prefixes for all commands so that they do not conflict with the commands in the **generic-commands.xml** file.

Result set

The result set for the previous query looks as follows:

name	suid	dbname	fullname	language	totcpu	totio	pwdate
jstask	3	master	(null)	(null)	0	10	2009-12-22 09:53:50
probe	2	subsystemdb	(null)	(null)	0	0	2009-12-22 08:37:35
sa	1	master	(null)	(null)	182	168723	2009-12-22 08:36:54

The way DbVisualizer handles the result set depends on whether the command is executed as a request in the database objects tree (**ObjectsTreeDef**) or in the object view (**ObjectsViewDef**). If executed in the database objects tree, each row in the result set will be represented by a new node in the tree. If executed in the object view, it is the viewer component that decides how the result will be presented. For more information on how a result set is used in the **ObjectsTreeDef** or **ObjectsViewDef**, read the specific sections.

Another important difference between the database objects tree and the object view is that the tree is a hierarchical structure of objects while the object view presents information about a specific object. An object that is inserted in the database objects tree is a 1..1 mapping to a row from the actual result set. The end user will see these objects (nodes) by some descriptive label, as defined in the **ObjectsTreeDef**. However, all data for the row from the original result set is stored with the object in the tree and may be used in the label, variables, conditions, etc. This is not the case in the **ObjectViewDef**.

The following example put some light on this. Consider the previous result set and that it's used to create objects in the database objects tree. The end user will see the following in DbVisualizer. The visible name for each row is the **name** column in the result set.

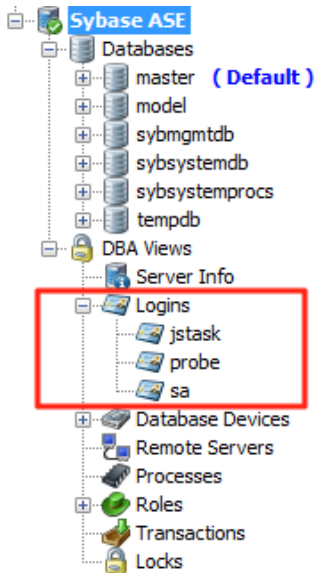


Figure: Sample of the Logins node having two child nodes

Each of the **sa** and **probe** nodes have all their respective data from the result set associated with the nodes. The data is referenced as **commandId.columnName**, i.e., **sybase-ase.getLogins.name**, **sybase-ase.getLogins.dbname**, etc. All associated data for the **sa** node in the example is listed next:

```
sybase-ase.getLogins.Name = sa
sybase-ase.getLogins.suid = 1
sybase-ase.getLogins.Default Database = master
sybase-ase.getLogins.Full Name = (null)
sybase-ase.getLogins.Default Language = (null)
sybase-ase.getLogins.CPU Time = 182
sybase-ase.getLogins.I/O Time = 168716
sybase-ase.getLogins.Password Set = 2009-12-22 08:36:54.576
```

The DataNode definition presenting **sa** and **probe** in the previous screenshot example use the associated data for the label as follows:

```
label="{sybase-ase.getLogins.name}"
```

<Input> - Setting command input

There are two types of Commands: with and without dynamic input. The difference is that dynamic input Commands accepts input data that is typically used to form the **WHERE** clause in SELECT SQLs. The previous example illustrates a static SQL (without dynamic data).

To allow for dynamic input, just add variables at the positions in the statement that should get dynamic values. The following is an extension of the previous example that allows for dynamic input.

```
<Command id="sybase-ase.getLogins">
  <SQL>
    <![CDATA[
```

```

select name "Name", suid "SUID", dbname "Default Database", fullname "Full Name",
language "Default Language", totcpu "CPU Time", totio "I/O Time", pwdate "Password Set"
from master.dbo.syslogins where name = '${name}' and suid = '${suid}' order by 1
]]>
</SQL>
</Command>

```

The example above adds two input variables: **`\${name}`** and **`\${suid}`**. Values for these variables should then be supplied wherever the command is referred for execution via the **Input** element.

The following is an example from the **ObjectsTreeDef** and its use of the **sybase-ase.getLogins** command:

```

<GroupNode type="Logins" label="Logins">
  <DataNode type="Login" label="${sybase-ase.getLogins.Name} isLeaf="true">
    <SetVar name="objectname" value="${sybase-ase.getLogins.Name}">
      <Command idref="sybase-ase.getLogins">
        <Input name="name" value="sa">
          <Input name="suid" value="${sybase-ase.getProcesses.suid}">
        </Command>
      </DataNode>
    </GroupNode>

```

(Note that the **Command** element refers the command via the **idref** attribute which will be matched with the corresponding **id** for the Command).

There is no magic with this definition, since the **`\${name}`** variable in the final SQL will be replaced with string **"sa"**.

The value for the **`\${suid}`** definition will in this case get the value of the **sybase-ase.getProcesses.suid** when the SQL is executed. So where is this variable defined? As explained in the [Result Set](#) section, all the data for a row in the result set is associated with the objects in the database objects tree. In addition, it is possible to use all the data kept by the current object and all its parent objects (as presented in the objects tree) in the input to commands. So to evaluate the **`\${sybase-ase.getProcesses.suid}`** variable, DbVisualizer first looks for the variable in the current object. If it doesn't exist, it continues to look through the parent objects until it reaches the root, which is the **Connections** object in the objects tree. If the variable is not found, it will be set to the string representation for null, which is **(null)** by default. Whenever a matching variable is found, DbVisualizer uses its value and stops searching.

<Output> - Redefine command output

As mentioned earlier, a specific column value in a result set row is referenced by the name of the column prefixed by the command id. Sometimes this is not desirable and the **Output** definition can be used to change this behavior. The following identifies a column in the result set by its index number, starting from 1, and then force its name to be set to the value of the **id** attribute.

```

<Output id="sybase-ase.getLogins.Name" index="1">
<Output id="sybase-ase.getLogins.suid" index="2">

```

The **Output** element can also be used to alter the structure of columns in the result set by adding, renaming or removing columns.

```

<Output modelaction="add" index="THIS_IS_A_NEW_COLUMN" value="Rattle and Hum">
<Output modelaction="rename" index="ADDR" name="ADDRESS">
<Output modelaction="rename" index="2" name="PHONE">
<Output modelaction="drop" index="MOBILE_PHONE">
<Output modelaction="drop" index="4">

```

(The rename and drop actions accepts either the name of the column or index number starting from the left at index 1).

- The **add** operation adds a new column to all rows. The value attribute accepts variables using the **`\${...}`** syntax.
- The **rename** operation simply renames a column.
- The **drop** operation drops the specified column.

The **rename** operation is primarily used when building a custom command that is supposed to be used by a viewer that requires predefined input by specific column names. Read more in the [ObjectsViewDef](#) section.

<ObjectsTreeDef> - Definition of the Database Objects Tree

The **ObjectsTreeDef** element section controls how the database objects tree should be presented and which commands should be executed to form its content (nodes). The mapping between the graphical representation in DbVisualizer and its **ObjectsTreeDef** XML is as straight forward as it can be:

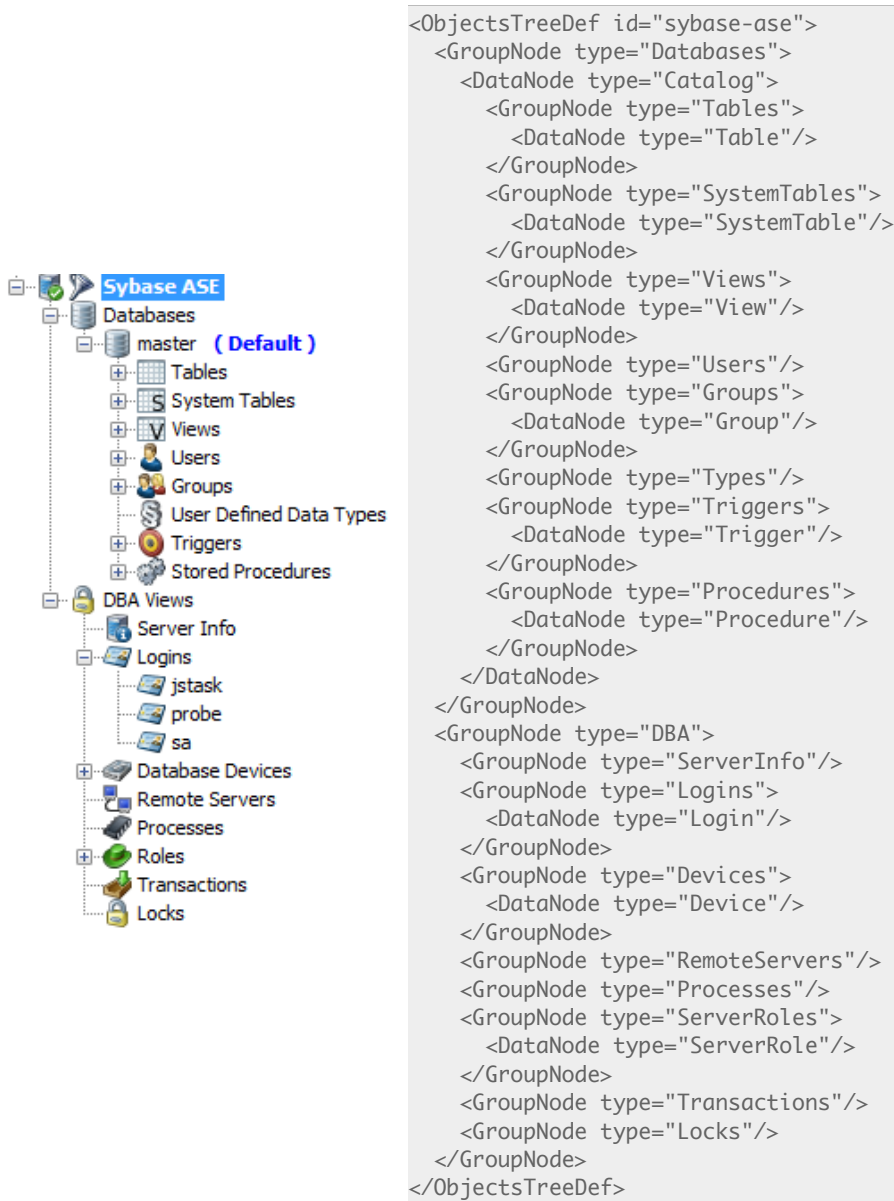


Figure: The visual database objects tree and its XML definition

The screenshot shows all nodes representing the **GroupNode** definitions in the **ObjectsTreeDef**. One exception is the **Logins** object, which has been expanded (**jstask**, **probe** and **sa** child objects) to illustrate what **DataNode** objects look like. The **ObjectsTreeDef** in the example has been simplified to show only the **type** attribute. (The label of the nodes as they appear in the visual tree is not listed in the **ObjectsTreeDef** example). The type attribute is primarily used internally in the profile as an identifier between the **ObjectsTreeDef** and the **ObjectsViewDef**. The type is also visible in the DbVisualizer GUI, in the tooltip for a tree node and in the object view header. The type is also used to identify the icon used to represent the object type.

There are no limitation on the number of levels in the **ObjectsTreeDef**. A good rule of thumb is, however, to keep it simple, clean and intuitive.

The **DataNode** definitions are the most important objects in the **ObjectTreeDef**. They also define which object tree filters are available for each object type, if overlaid icons should appear (and the criteria), etc. Read the next sections for details.

<GroupNode> - Static objects used for grouping

The **GroupNode** element represents a static object in the tree. These don't have any associated SQL and appear only once where they are defined. A **GroupNode** is primarily used for structural and grouping purposes. The **GroupNode** element has the following attributes.

```
<GroupNode type="SystemTables" label="System Tables" isLeaf="false">
  ...
</GroupNode>
```

The **isLeaf** attribute is optional and controls whether the **GroupNode** may have any child objects or not. It can always be set to true, but the effect in the visual database objects tree is then that the expand icon to the left of the group node icon will always be displayed, even if it can never have any child objects. The default setting for **isLeaf** is false.

If **isLeaf** is set to false and there are child Group and/or Data -nodes, these will not appear. The result may cause some frustration during the design...

<DataNode> - Dynamic objects created via SQL

The **DataNode** element feeds the tree with nodes produced by a **Command**. The example in the [Command](#) section querying for all logins in Sybase ASE look as follows in the **ObjectsTreeDef**:

```
<GroupNode type="Logins" label="Logins">
  <DataNode type="Login" label="${sybase-ase.getLogins.Name}" isLeaf="true">
    <Command idref="sybase-ase.getLogins"/>
  </DataNode>
</GroupNode>
```

First, there is a **GroupNode** element with the purpose to group all child objects in a **Logins** node. The **DataNode** has, in this example, the same attributes as the **GroupNode**, the type is however "**Login**" instead of "**Logins**" (as it is for the **GroupNode**). This difference is important when the user selects one of the objects, since the Object View shows the appropriate views based on the object type.

The **DataNode** definition can be seen as a template, as the associated command fetches rows of data from the database and DbVisualizer uses the **DataNode** definition to create one node per row in the result set.

The **label** attribute for the data node is somewhat different, as it introduces the use of a variable (or several). The real value for the label will, in this example, be the value in the **Name** column produced by the **sybase-ase.getLogins** command, as you can see in the **Command** definition (variable names are automatically prefixed with the command id).

The **Command** element uses the **idref** attribute to identify the command that should be executed. The command in this case and in the [Result set](#) section produces a result set with 2 rows and 8 columns. The result will be two nodes each, with the label of the **Name** column in the result set.

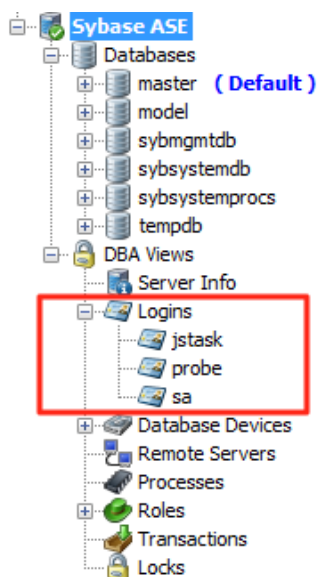


Figure: Sample of the Logins node having two child nodes

The label can be changed by setting it to any other valid variable or a combination of several variables. (It's even possible to specify static text in the label):

```
Label="{{sybase-ase.getLogins.Name}} ({{sybase-ase.getLogins.Default Database}})"
```

The example above results in the following labels:

```
jstask (master)
probe (subsystemdb)
sa (master)
```

The complete set of attributes for the **DataNode** element is:

type="value"	- The type of node (required)
actiontype="value"	- Object type used for object actions (optional)
label="value"	- The visual label (required)
isLeaf="true/false"	- Specifies if the node can have child objects (default true)
sort="col1,col2"	- A comma separated list of names/variables used for sorting
drop-label-not-equal="value"	- Do not add the node if the label is not equal to this value or variable
warnstate="condition"	- If condition is true, show an overlay icon for the node
errorstate="condition"	- If condition is true, show an overlay icon for the node
stop-label-hot-equal="value"	- The node will be a leaf if the label doesn't match this value or variable
is-empty-output="continue/stop"	- If result set is empty, use this to control whether child GroupNode/DataNodes should be added anyway or ignored

The **Command** definition in the example above is simple, since it doesn't use any variables in the SQL. Continue reading the next section for details about passing input data to commands.

<Command>

Commands are referenced in the **DataNode** definition by the **idref** attribute. Sometimes it is required that a specific **DataNode** must supply input to a command. This is done by adding **Input** elements as children to the **Command**.

```
<DataNode type="Login" label="{{sybase-ase.getLogins.Name}}" isLeaf="true">
  <Command idref="sybase-ase.getLogins">
    <Input name="name" value="sa">
      <Input name="suid" value="{{sybase-ase.getProcesses.suid}}">
    </Command>
  </DataNode>
```

The value for a variable specified in an **Input** element is evaluated using the strategy outlined in the [Result set](#) section.

<Filter>

The **Filter** element is specific for **Command** elements that appear in the **ObjectsTreeDef** section. A filter define which data for a **DataNode** that is allowed to use in filters. This filter functionality is commonly referred as the **Database Objects Tree Filtering** in DbVisualizer. The filtering setup appears below the database objects tree, and the following example shows that filtering may be specified for these object types:

- Catalog
- Table
- System Table
- View
- User
- Group
- Trigger
- Procedure

For each of the **Filter** definitions, one or several columns can be as part of the filtering criteria.

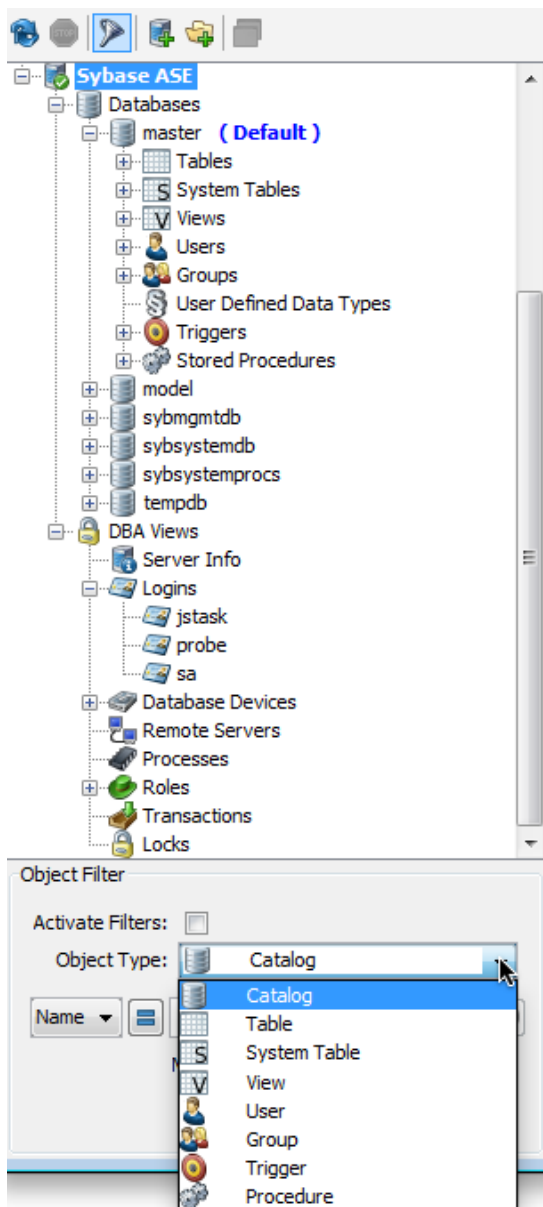


Figure: Screen shot showing the filter pane

```

<DataNode type="Views" label="{{sybase-ase.getViews.Name}}" isLeaf="true">
  <Command idref="sybase-ase.getViews">
    <Filter type="View" name="View Table">
      <Column index="TABLE_NAME" name="Name"/>
    </Filter>
  </Command>
</DataNode>
  
```

The previous filter definition specifies a filter for the **View** object type. The **name** specifies the name of the filter as it appears in the object type drop-down list. The nested **Column** element defines the **index**, which should be either a column name in the result set or an index number for the column. The **name** attribute specifies the name of the column as it appears in the filter pane.

Several **Column** elements may be specified for a **Filter** element.

<SetVar>

The **SetVar** element is needed in the **ObjectsTreeDef** for **DataNode**'s. Some object types have special meaning in DbVisualizer. Two examples are the **Catalog** and **Schema** object types. For **DataNode** objects, you must use **SetVar** elements to identify them, with **name** attributes set to **"catalog"**

and "schema", respectively.

```
<DataNode type="Catalog" label="{getCatalogs.TABLE_CAT}" isLeaf="false">
  <SetVar name="catalog" value="{getCatalogs.TABLE_CAT}">
</DataNode>
```

All non **Catalog** or **Schema DataNode**'s must use **SetVar** to set the "objectname" variable:

```
<DataNode type="Views" label="{sybase-ase.getViews.Name}" isLeaf="true">
  <SetVar name="objectname" value="{sybase-ase.getViews.Name}">
  <SetVar name="rowcount" value="true/false">
</DataNode>
```

The **objectname** variable is used to identify the object represented by the data node, so that it can be uniformly referenced in object views and object actions. Its value should be the identifier for the object as it is identified in the database, e.g., a table name or view name.

The **rowcount** variable is optional and controls whether the object supports showing row count information when **Show/Hide Table Row Count** right-click menu choice is enabled for the database connection.

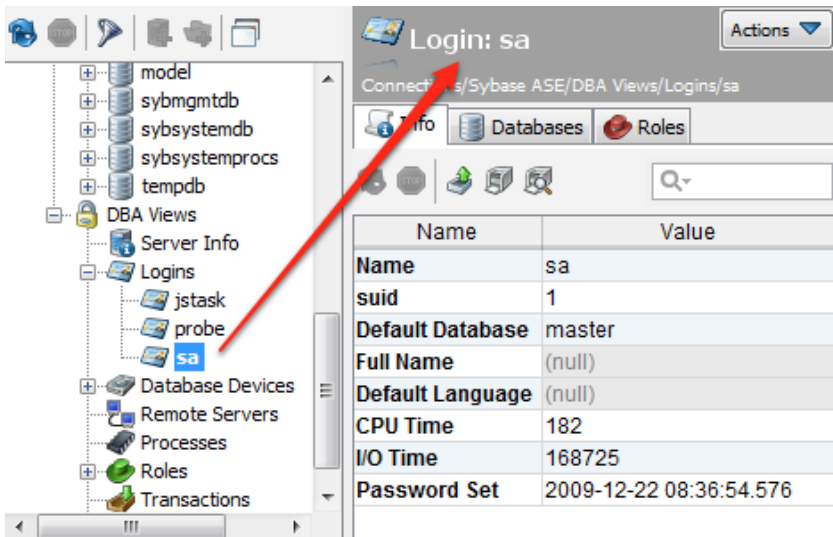
Another optional variable (not shown in the example above) is named **acceptInQB**. If set to true, nodes of this type can be used in the **Query Builder**. It should only be set to true for object types representing tabular data that can be queried with an SQL SELECT statement, such as tables, views, materialized views, etc.

<SetVar> variables are by default invisible in for example the NodeFormViewer. If you want to override this behavior then add the **action** attribute and set its value to **show**. If you want to drop a variable completely from the node simply set the **action** attribute to **drop**.

<ObjectsViewDef> - Definition of the Object Views

The **ObjectsViewDef** element defines all views for the object types in the objects tree. These views are displayed in the **Object View** area for the selected object. Which views should appear when selecting a node in the tree is based on the object type for the tree node and the corresponding object view definition.

When an object is selected in the tree (**sa** in the screenshot below), its complete information is passed to the object view handler (right in the sample). This handler determines, based on the object type, which object view should be used to present the information. When the object view is found, all data views are created as tabs in the user interface. The selected object and its information is passed to each of the data views for processing and presentation. The following shows how the Object View look in DbVisualizer and its accompanying **ObjectView** definitions.



The screenshot shows the DbVisualizer interface. On the left is the 'Objects Tree' with nodes like 'model', 'sybmgmtdb', 'sybmgmtdb', 'sybmgmtproc', 'tempdb', 'DBA Views', 'Server Info', 'Logins', 'jstask', 'probe', 'sa', 'Database Devices', 'Remote Servers', 'Processes', 'Roles', and 'Transactions'. A red arrow points from the 'sa' node in the tree to the 'Object View' area. The 'Object View' area shows a table with columns 'Name' and 'Value' containing details for the 'sa' user. To the right of the table is the XML definition for the 'Logins' and 'Login' object views.

Name	Value
Name	sa
suid	1
Default Database	master
Full Name	(null)
Default Language	(null)
CPU Time	182
I/O Time	168725
Password Set	2009-12-22 08:36:54.576

```
<ObjectView type="Logins">
  <DataView type="Logins" label="Logins"
    viewer="grid">
    <Command idref="sybase-ase.getLogins"/>
  </DataView>
</ObjectView>
<ObjectView type="Login">
  <DataView type="Info" label="Info"
    viewer="node-form"/>
  <DataView type="Databases" label="Databases"
    viewer="grid">
    <Command idref="sybase-ase.getLoginDatabases"/>
  </DataView>
  <DataView type="Roles" label="Roles"
    viewer="grid">
    <Command idref="sybase-ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```

Figure: The visual database objects tree, object view and the XML definition

The screenshot shows both the **Logins** node and its child nodes, **jstask**, **probe** and **sa**. From the **GroupNode** and **DataNode** declaration examples in the previous sections, we know that these nodes are instances of the object types **Logins** (the Login node) and **Login** (the two sub nodes, sa and probe).

The **ObjectView** XML definitions shows the data views for these two types, **Logins** and **Login**. Clicking on the node labeled **Logins** in the tree will

show the object view for the `<ObjectView type="Logins">` definition while clicking on the node labeled `jstask`, `probe` or `sa` will show the object view for the `<ObjectView type="Login">` .

The example shows `sa` being selected. Its **DataView** definitions are (by label):

- Info
- Databases
- Roles

These views are presented in DbVisualizer as tabs. The label of each tab is the label defined in the **DataView** and the icons are defined by the respective object type.

The **ObjectsViewDef** root element has the following attributes:

```
<!-- Include the generic-view -->
&generic-view;
<ObjectsViewDef id="sybase-ase" extends="generic" >
  ...
</ObjectsViewDef>
```

The first statement for the **ObjectsViewDef** elements is:

```
&generic-view;
```

This simply means that the generic-view entity defined at the top of the XML file is included in the XML, i.e., all its definitions are accessible as is. One example is the **ObjectView** definition in the generic-view.xml file for the **Table** object type. It contains a lot of **DataView** elements that identify all viewers for the **Table**. If you now want to use the generic Table **DataView**'s but add a new **Abbreviations** data view, then simply extend the generic Table **DataView**. This is done by adding a `extends="generic"` attribute in the **ObjectsViewDef** element. By using the exact same object type in the extended **ObjectView**, you will then get this behavior. Read more about extending **ObjectView**'s in the [Extending ObjectView](#) section.

<ObjectView>

The **ObjectView** element is associated with an object type and groups all **DataView** elements that appear when the object type is selected in the database objects tree. Here follows the **ObjectView** definition for the **Login** object type.

```
<ObjectView type="Login">
  ...
</ObjectView>
```

This element is simple as its only attribute is the `type` attribute. The `type` attribute value is used when a node is clicked in the database objects tree to map the object of the type clicked and its **ObjectView**.

<DataView>

The **DataView** element is as important as the **DataNode** is in the **ObjectsTreeDef**. It defines how the viewer should be labeled in DbVisualizer, which viewer (presentation form) it should use, commands and other things. The following is the **DataView** definitions for the **Login** object type. (The **ObjectView** element is part of the sample just for clarification).

```
<ObjectView type="Login">
  <DataView type="Info" label="Info" viewer="node-form"/>
  <DataView type="Databases" label="Databases" viewer="grid">
    <Command idref="sybase-ase.getLoginDatabases"/>
  </DataView>
  <DataView type="Roles" label="Roles" viewer="grid">
    <Command idref="sybase-ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```

The elements are used to define how the object is presented in DbVisualizer, as described in the [introduction](#) of the **ObjectsViewDef** section. All three data view elements have a `viewer` attribute, which identifies how the data in the view should be presented, e.g., as a grid or a form. See the next section for a list of viewers.

Viewers

The **viewer** attribute for a **DataView** specifies how the data for the view should be presented. The following sections walk through the supported viewers.

The following sample illustrates the **viewer** attribute.

```
<ObjectView type="Login">
  <DataView type="Info" label="Info" viewer="node-form"/>
</ObjectView>
```

DataView definitions may be nested and the viewers are then presented with the nested DataView in the lower part of the screen.

grid

The **grid** viewer presents a result set in a grid, with standard grid features such as search, copy, fit, export, etc. The result set is presented exactly as it is produced by the **Command** and any optional **Output** processing.

Here is a sample of the XML for the grid viewer:

```
<DataView type="Columns" label="Columns" viewer="grid">
  <Command idref="oracle.getColumns">
    <Input name="owner" value="${schema}"/>
    <Input name="table" value="${objectname}"/>
  </Command>
</DataView>
```

And here is a screenshot of the standard grid viewer created from the previous definition.

COLUMN_ID	OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRE
1	HR	BIO	EMPLOYEE_ID	NUMBER	22	
2	HR	BIO	FIRST_NAME	VARCHAR2	20	
3	HR	BIO	LAST_NAME	VARCHAR2	25	
4	HR	BIO	EMAIL	VARCHAR2	25	
5	HR	BIO	PHONE_NUMBER	VARCHAR2	20	
6	HR	BIO	HIRE_DATE	DATE	7	
7	HR	BIO	JOB_ID	VARCHAR2	10	
8	HR	BIO	SALARY	NUMBER	22	
9	HR	BIO	COMMISSION_PCT	NUMBER	22	
10	HR	BIO	MANAGER_ID	NUMBER	22	
11	HR	BIO	DEPARTMENT_ID	NUMBER	22	
12	HR	BIO	PHOTO	BLOB	4000	
13	HR	BIO	RESUME	CLOB	4000	

Figure: The grid viewer

The nesting capability for grid viewers is really powerful, as it can be used to create a drill-down view of the data. Consider the scenario with a **grid** viewer showing all Trigger objects. Wouldn't it be nice to offer the user the capability to display the trigger source when selecting a row in the list? This is easily accomplished with the following:

```
<DataView type="Trigger" label="Triggers" viewer="grid">
  <Command idref="oracle.getTriggers">
    <Input name="owner" value="${schema}"/>
    <Input name="table" value="${objectname}"/>
  </Command>
</DataView>
```

```

<DataView type="Source" label="Source" viewer="text">
  <Input name="dataColumn" value="text"/>
  <Input name="formatSQL" value="true"/>
  <Command idref="oracle.getTriggerSource">
    <Input name="owner" value="{OWNER}"/>
    <Input name="name" value="{TRIGGER_NAME}"/>
  </Command>
</DataView>
<DataView type="Info" label="Info" viewer="node-form"/>
</DataView>

```

- The first **DataView** definition defines the top grid viewer and the command to get the result set for it.
- The next **DataView** is the nested text viewer, specifying various input parameter for the viewer along with the command to get the source for the trigger. The difference here is that the input parameters for this command reference column names in the top grid. Since this viewer is nested, it will automatically be notified whenever an entry in the top grid is selected.
- The third nested **DataView** is presented as a tab next to the **Source** viewer, and presents additional information about the selected trigger.

The following screenshot illustrates the above sample:

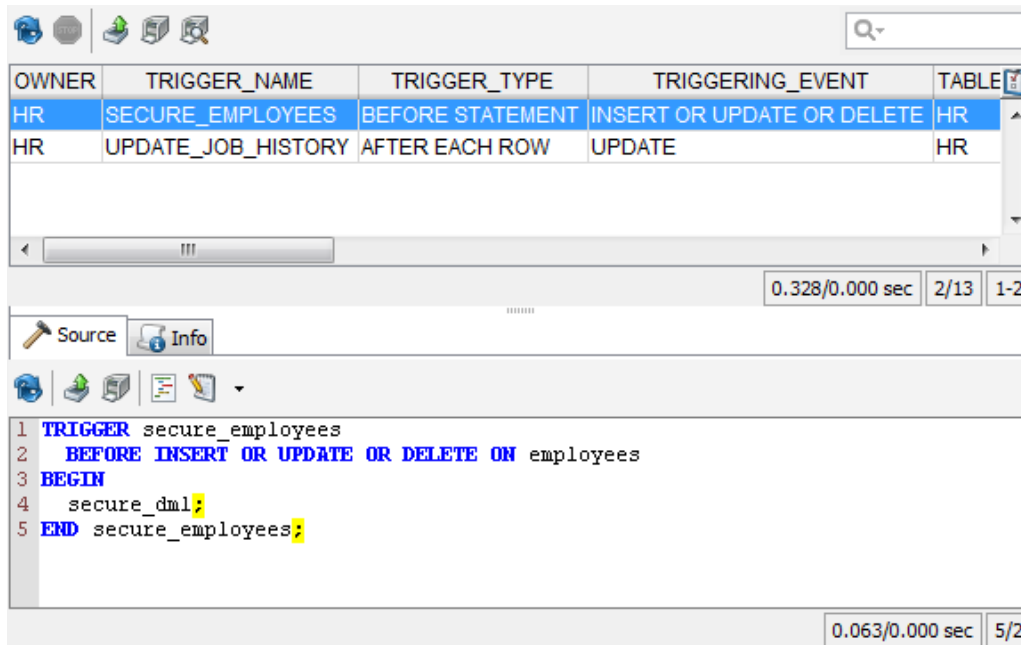


Figure: Example use of nested DataViews

Adding custom menu items in the grid

The **menutem** parameter specifies entries that should appear in the right-click menu in the grid. The value for the **menutem** is the label for the item, while the child **Input** element specifies the SQL command that should be produced for all selected rows when the menu item is selected. The result of a custom menu item is that the grid viewer creates a statement that it copies to the SQL Commander; it will never execute the produced SQL in the scope of the viewer.

The following is an example with two menu items:

- **Script: SELECT ALL**
- **Script: DROP TABLE**

The variables in the SQL statement should identify column names in the result set. The user may select any columns in the visual grid and choose a custom menu item. It is only the actual rows that are picked from the selection as the columns are predefined by the **menutem** declaration.

The variables specified in these examples starts with **\${schema=...}** and **\${object=...}**. These defines that the first variable represents a **schema** variable while the second defines an **object**. This is needed for DbVisualizer to determine whether **delimited identifiers** should be used and if identifiers should be **qualified**, as defined in the connection properties for the database.

```

<Input name="menuItem" value="Script: SELECT ALL">
  <Input name="command" value="select * from ${schema=OWNER}${object=TABLE_NAME}"/>
</Input>

```

```
<Input name="menuItem" value="Script: DROP TABLE">
  <Input name="command" value="drop table ${schema=OWNER}${object=TABLE_NAME}"/>
</Input>
```

Here is a sample:

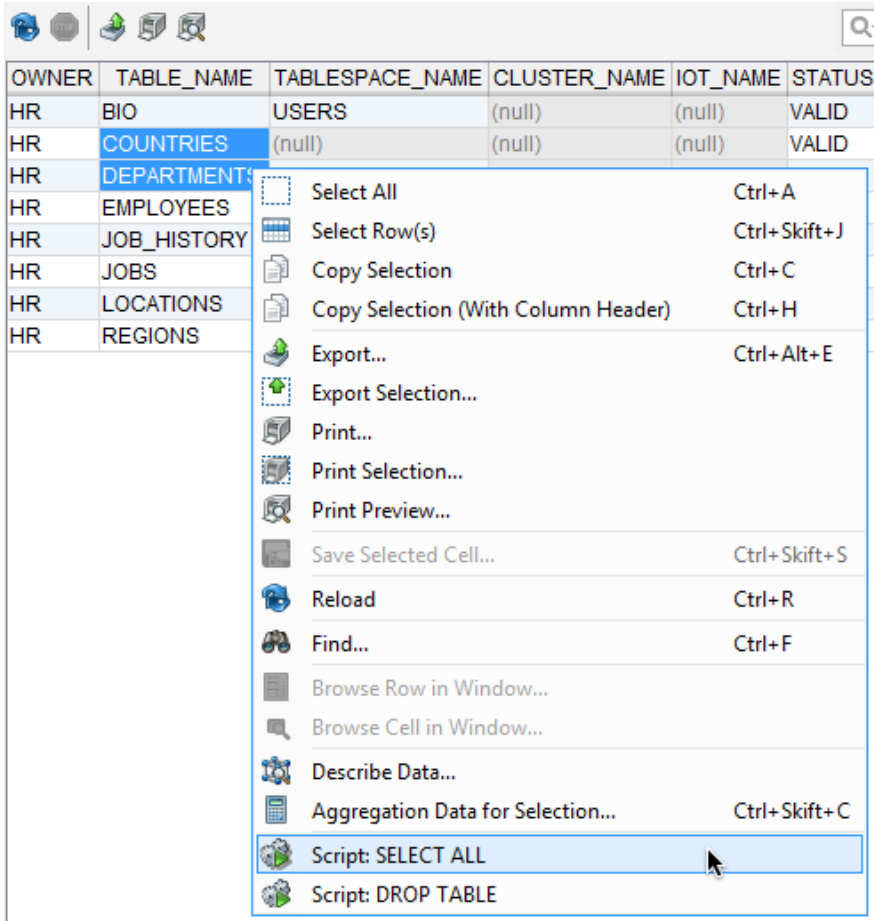


Figure: Custom menu items in grid viewer

The result of selecting a menu item defined as a **menuItem** input parameter is that the specified command is copied to the current SQL editor.

Setting initial max column width

Some result sets may contain columns with very wide data. The following parameter sets an initial maximum column width for all columns in the grid.

```
<Input name="columnWidth" value="" />
```

text

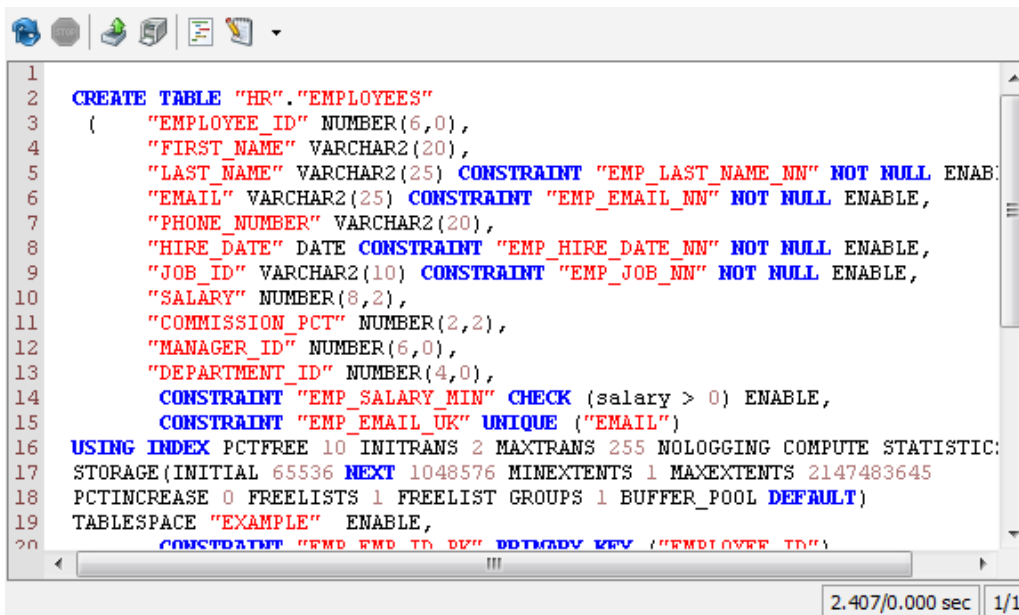
The **text** viewer presents data from one column in a result set in a text browser (read only editor). This viewer is typically used to present large chunks of data, such as source code, SQL statements, etc. If the result set contains several rows, the text viewer reads the data in the column for each row and present the combined data.

Here is a sample of the XML for the text viewer:

```
<DataView type="Source" label="Source" viewer="text">
  <Input name="dataColumn" value="text"/>
  <Input name="formatSQL" value="true"/>
  <Command idref="oracle.getTriggerSource">
    <Input name="owner" value="${schema}"/>
    <Input name="name" value="${objectname}"/>
  </Command>
</DataView>
```

```
</Command>
</DataView>
```

And here is a screenshot of the Source tab based on the previous definition.



```
1 CREATE TABLE "HR"."EMPLOYEES"
2 {
3   "EMPLOYEE_ID" NUMBER(6,0),
4   "FIRST_NAME" VARCHAR2(20),
5   "LAST_NAME" VARCHAR2(25) CONSTRAINT "EMP_LAST_NAME_NN" NOT NULL ENABLE,
6   "EMAIL" VARCHAR2(25) CONSTRAINT "EMP_EMAIL_NN" NOT NULL ENABLE,
7   "PHONE_NUMBER" VARCHAR2(20),
8   "HIRE_DATE" DATE CONSTRAINT "EMP_HIRE_DATE_NN" NOT NULL ENABLE,
9   "JOB_ID" VARCHAR2(10) CONSTRAINT "EMP_JOB_NN" NOT NULL ENABLE,
10  "SALARY" NUMBER(8,2),
11  "COMMISSION_PCT" NUMBER(2,2),
12  "MANAGER_ID" NUMBER(6,0),
13  "DEPARTMENT_ID" NUMBER(4,0),
14  CONSTRAINT "EMP_SALARY_MIN" CHECK (salary > 0) ENABLE,
15  CONSTRAINT "EMP_EMAIL_UK" UNIQUE ("EMAIL")
16 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 NOLOGGING COMPUTE STATISTIC:
17 STORAGE(INITIAL 65536 NEXT 1048576 MIMEXTENTS 1 MAXEXTENTS 2147483645
18 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
19 TABLESPACE "EXAMPLE" ENABLE,
20 CONSTRAINT "EMP_EMP_ID_PK" PRIMARY KEY ("EMPLOYEE_ID")
```

Figure: The text viewer

Specify what column to browse

By default, the text viewer uses the data in first column. This behavior can be controlled by using the `dataColumn` input parameter. Simply specify the name of the column in the result set or its index (starting at 1 from the left).

```
<Input name="dataColumn" value=""/>
```

Enable SQL formatting of the data

The text viewer includes the **SQL Formatting** toolbar button, which when pressed formats the content in the viewer. The `formatSQL` input parameter is used to control whether formatting should be enabled by default. If `formatSQL` is not specified, no initial formatting is made.

```
<Input name="formatSQL" value=""/>
```

form

The form viewer presents row(s) from a result set in a form. If several rows are in the result, they are presented in a list. Selecting one row from the list presents all columns and data for that row in a form.

Here is a sample of the XML for the form viewer:

```
<DataView type="Info" label="Info" viewer="form">
  <Command idref="oracle.getTable">
    <Input name="owner" value="{{schema}}"/>
    <Input name="table" value="{{objectname}}"/>
  </Command>
</DataView>
```

And here is a screenshot of the Info tab based on the previous definition.

Name	Value
OWNER	HR
TABLE_NAME	EMPLOYEES
TABLESPACE_NAME	EXAMPLE
CLUSTER_NAME	(null)
IOT_NAME	(null)
STATUS	VALID
PCT_FREE	10
PCT_USED	(null)
INI_TRANS	1
MAX_TRANS	255
INITIAL_EXTENT	65536
NEXT_EXTENT	(null)
MIN_EXTENTS	1
MAX_EXTENTS	2147483645
PCT_INCREASE	(null)
FREELISTS	(null)
FREELIST_GROUPS	(null)
LOGGING	NO

Figure: The form viewer

node-form

The node-form viewer presents all data associated with the selected object (variables).

Here is a sample of the XML for the node-form viewer:

```
<DataView type="Constraint" label="Constraint" viewer="node-form">
  <Input name="hidecolumn" value="oracle.getKeys.TABLE_OWNER"/>
</DataView>
```

And here is a screenshot of the Constraint tab based on the previous definition.

Name	Value
OWNER	HR
TABLE_NAME	EMPLOYEES
CONSTRAINT_NAME	EMP_EMAIL_NN
CONSTRAINT_TYPE	Check
DDL_TYPE	CONSTRAINT
SEARCH_CONDITION	"EMAIL" IS NOT NULL
R_OWNER	(null)
R_CONSTRAINT_NAME	(null)
DELETE_RULE	(null)
STATUS	ENABLED
DEFERRABLE	NOT DEFERRABLE
DEFERRED	IMMEDIATE
VALIDATED	VALIDATED
GENERATED	USER NAME
BAD	(null)
RELY	(null)
LAST_CHANGE	2009-11-26 11:45:31
INDEX_OWNER	(null)
INDEX_NAME	(null)
INVALID	(null)

Figure: The node-form viewer

Hiding columns

There may be data associated with the object that you don't want to present in the node form for the user. The **hidecolumn** input parameter control what data for the object that should be invisible and you may repeat the this option as many times you like to handle multiple hidden variables.

```
<Input name="hidecolumn" value="oracle.getKeys.TABLE_OWNER"/>
```

table-refs

The table-refs viewer shows the references graph for the current object (this must be an object supporting referential integrity constraints, such as a Table),

Here is a sample of the XML for the table-refs viewer:

```
<DataView type="References" label="References" viewer="table-refs"/>
```

And here is a screenshot of the References tab based on the previous definition.

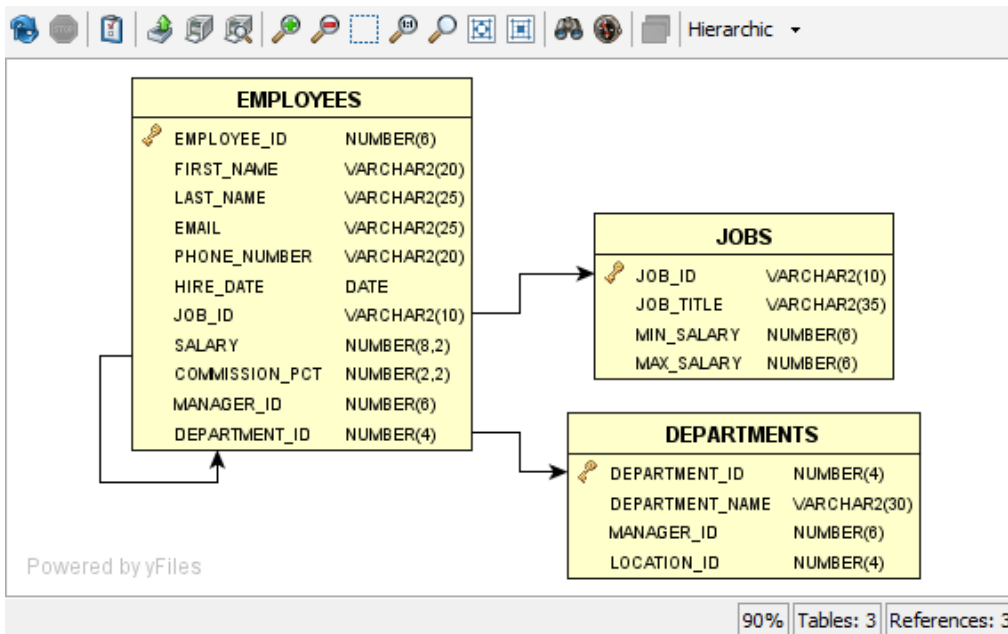


Figure: The table-refs viewer

tables-refs

The tables-refs viewer shows the references graph for several tables in the result set (the result set must contain objects supporting referential integrity constraints, such as a Table).

Here is a sample of the XML for the tables-refs viewer:

```
<DataView type="References" label="References" viewer="tables-refs">
  <Command idref="getTables">
    <Input name="catalog" value="{catalog}"/>
    <Input name="schema" value="{schema}"/>
    <Input name="table" value="{objectname}"/>
    <Input name="type" value="{tableType}"/>
  </Command>
</DataView>
```

And here is a screenshot of the References tab based on the previous definition.

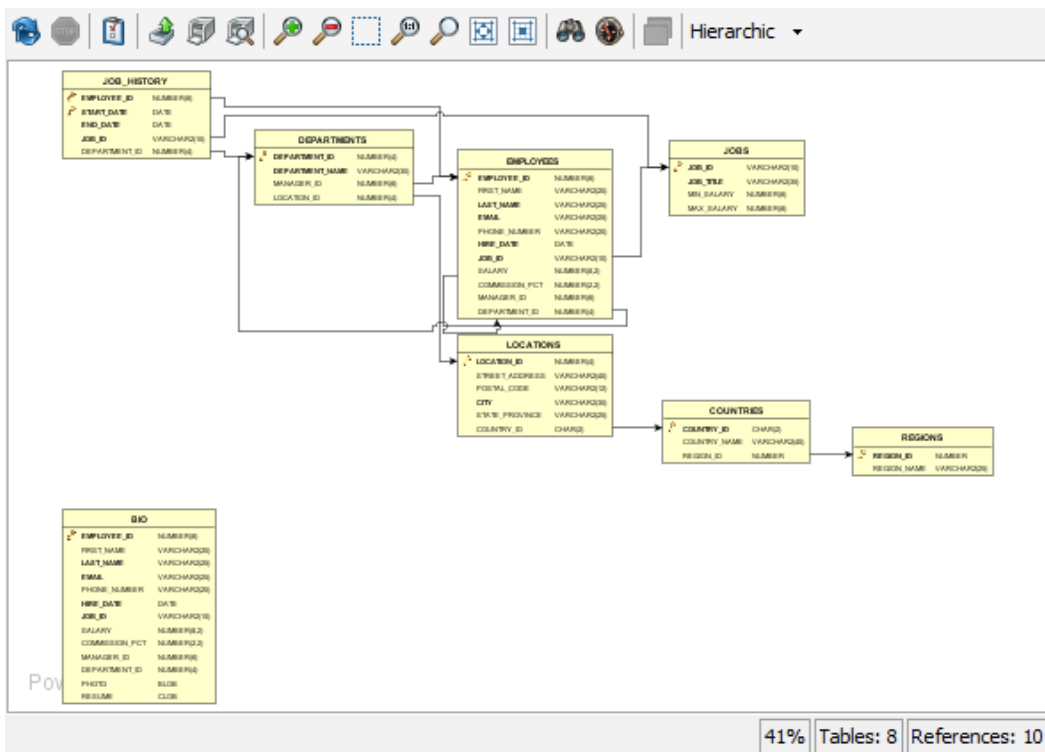


Figure: The tables-refs viewer

table-data

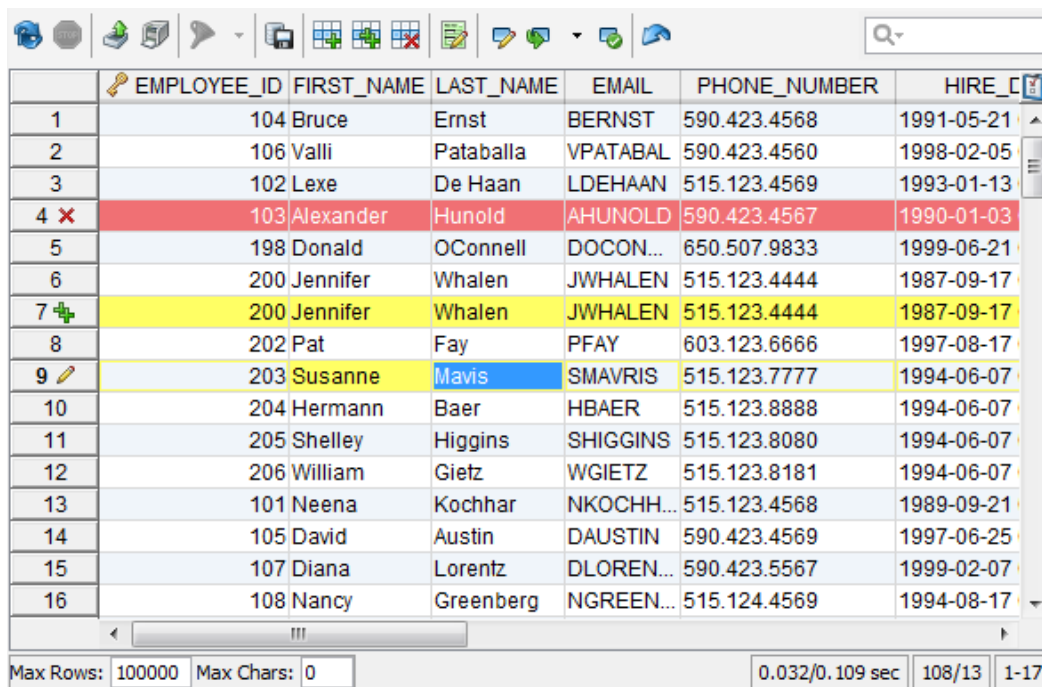
The table-data viewer shows the data for a table in a grid with editing features.

Information presented in the grid is obtained automatically by the viewer via a traditional `SELECT * FROM table` statement, i.e., the object type having this viewer defined must be able to support getting a result set via this SQL statement.

Here is a sample of the XML for the table-data viewer:

```
<DataView type="Data" label="Data" viewer="table-data">
  <Input name="disableEdit" value="<true/false>"/>
</DataView>
```

And here is a screenshot of the Data tab based on the previous definition.



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21
2	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03
5	198	Donald	OConnell	DOCON...	650.507.9833	1999-06-21
6	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17
7	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17
8	202	Pat	Fay	PFAY	603.123.6666	1997-08-17
9	203	Susanne	Mavis	SMAVRIS	515.123.7777	1994-06-07
10	204	Hermann	Baer	HBAER	515.123.8888	1994-06-07
11	205	Shelley	Higgins	SHIGGINS	515.123.8080	1994-06-07
12	206	William	Gietz	WGIETZ	515.123.8181	1994-06-07
13	101	Neena	Kochhar	NKOCHH...	515.123.4568	1989-09-21
14	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25
15	107	Diana	Lorentz	DLOREN...	590.423.5567	1999-02-07
16	108	Nancy	Greenberg	NGREEN...	515.124.4569	1994-08-17

Max Rows: 100000 Max Chars: 0 0.032/0.109 sec 108/13 1-17

Figure: The table-data viewer

Disable data editing

The default strategy for the table-data viewer is to automatically check whether the data can be edited or not. If editing is allowed a few related buttons will appear in the toolbar. However, sometimes you may want to disable editing completely for the table-data viewer. Do this with the following input element:

```
<Input name="disableEdit" value="" />
```

table-rowcount

The table-rowcount viewer shows the row count for a (table) object.

The row count is obtained automatically by the viewer via a traditional `SELECT COUNT(*) FROM table` statement, i.e., the object type having this viewer defined must be able to support getting a result set via this SQL statement.

Here is a sample of the XML for the table-rowcount viewer:

```
<DataView type="RowCount" label="Row Count" viewer="table-rowcount" />
```

And here is a screenshot of the Row Count tab based on the previous definition.

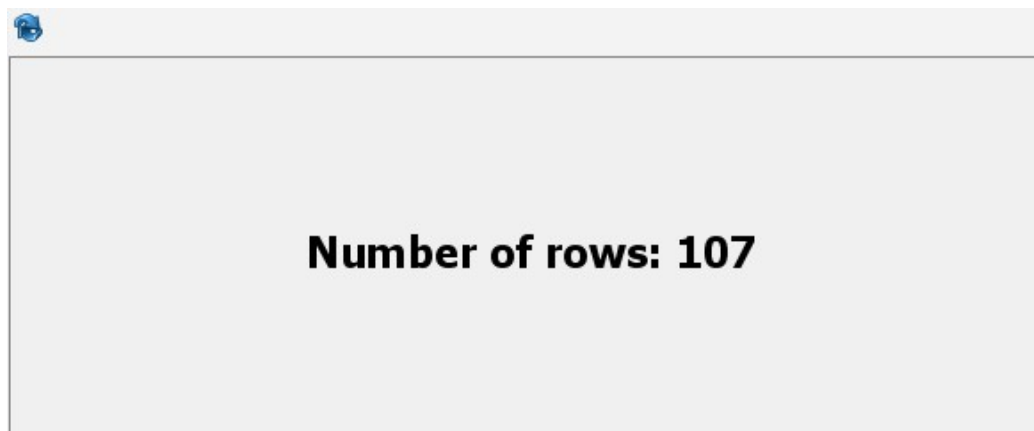


Figure: The table-rowcount viewer

<Command>

Please read the [Command](#) section above, as the capabilities of this element are the same when used with a data view.

<Message>

The **Message** element is very simple: it defines a message that should appear at the top of the viewer. The Message element is used to define the text for a description of the data presented in the viewer. The text in the message may contain common HTML tags such as **** (bold), *<i>* (italic), **
** (line break), etc.

Here is a sample of the XML for using the Message element in a grid viewer:

```
<ObjectView type="RecycleBin">
  <DataView type="RecycleBin" label="Recycle Bin" viewer="grid">
    <Command idref="oracle.getRecycleBin">
      <Input name="schema" value="{{schema}}"/>
      <Input name="login_schema" value="{{dbvis-defaultCatalogOrSchema}}"/>
    </Command>
    <Message>
      <![CDATA[
<html>
These are the tables currently in the recycle bin for this schema. Right click on a bin
table in objects tree to restore or permanently purge it.<br>
<b>Note: The recycle bin is always empty if not looking at the bin for your
login schema (default).</b>
</html>
      ]]>
    </Message>
  </DataView>
</ObjectView>
```

And here is a screenshot of the Recycle Bin tab based on the previous definition.

These are the tables currently in the recycle bin for this schema. Right click on a bin table in objects tree to restore or permanently purge it.
Note: The recycle bin is always empty if not looking at the bin for your login schema (default).

OBJECT_NAME	ORIGINAL_NAME	OPERATION	TYPE	TS_NAME	CREATE
BIN\$3AlucxwhTbOqHcaFoBJ+5w==\$0	EMP	DROP	TABLE	USERS	2007-10-15

0.266/0.000 sec 1/15 1-1

Figure: The appearance of a Message in a viewer

Extending ObjectView

An existing **ObjectView** definition in, for example, the generic-view.xml file can be extended in a database profile by using a few action attributes for each of the **DataView** elements. To extend a definition, the object type specified in the **ObjectView** type attribute must match the type in the parent profile. You have the following options when extending a definition:

- **Adding a DataView**
Simply add the **DataView** definition and it will be added to the current list of DataView definitions
- **Dropping an existing DataView**
Add the `<DataView type="xxx" action="drop">` to drop the dataview type named "xxx"
- **Replacing a DataView**
Just add the DataView with the exact same type as in the parent DataView. All the settings of the new DataView will replace the old one

<ObjectsActionDef> - Definition of user actions

The previous sections describe how to define which objects should appear in the objects tree, and which views should be displayed when selecting an object in the tree. The **ObjectsActionDef** section in the profile defines which operations are available for the object types defined in the **ObjectTreeDef**. Object actions are very powerful, as they offer an extensive number of features to define actions for almost any type of object operation.

In DbVisualizer, the object type actions menu is accessed via the right-click menu in the objects tree or via the **Actions** button in the object view:

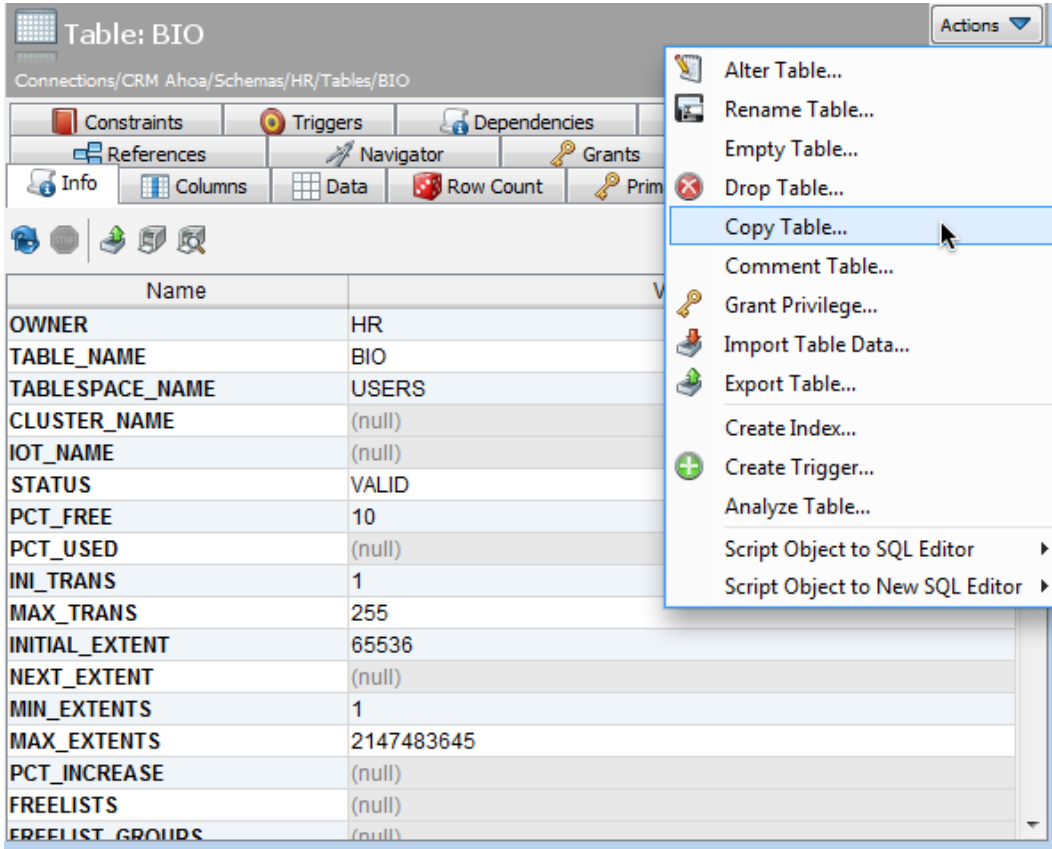


Figure: The Actions menu for the selected object

All of the operations for the selected **Table** object in the figure above are expressed in the **ObjectsActionDef** section. The implementation for these actions are either declared completely with XML elements via standard object actions, or via specialized action handlers. (The API for action handlers is not yet documented). The following screenshot shows the dialog appearing when executing an action via the default action handler:

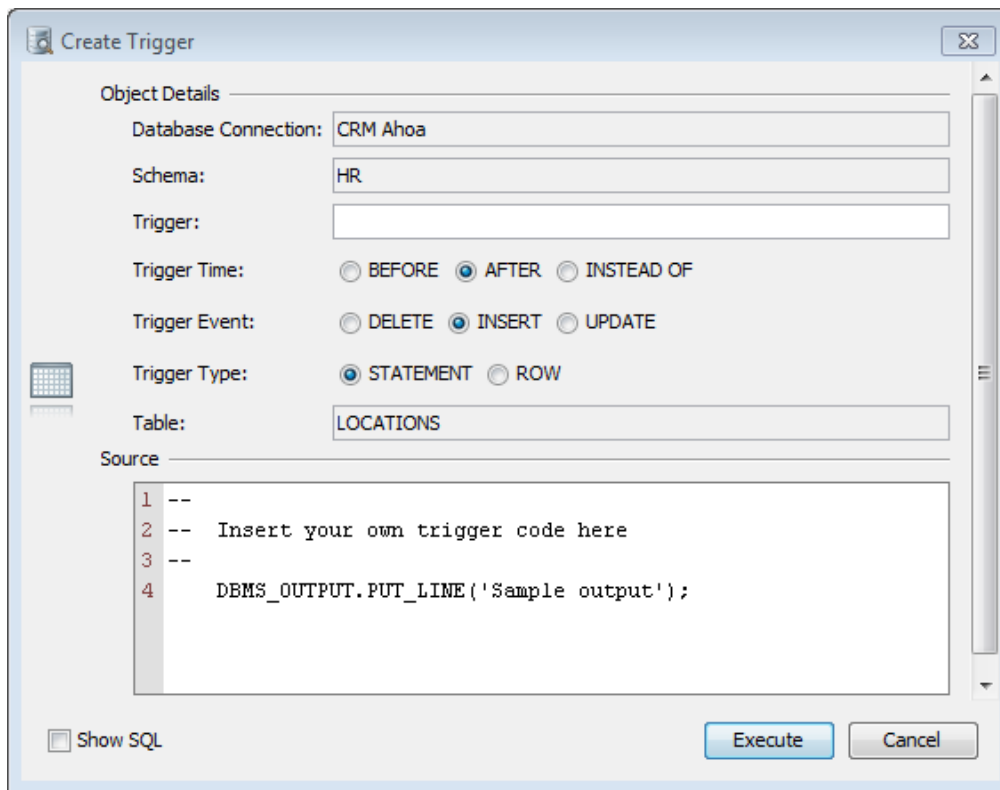


Figure: The default action handler

The first field in the dialog, **Database Connection**, is always present and shows the alias of the database connection the selected object is associated with. At the bottom, there is a **Show SQL** control that, when enabled, displays the final SQL for the action. The bottom right buttons are used to run the action (the label of the button may be **Execute** or **Script** based on the action mode), or to **Cancel** the action completely.

Variables

Variables are used to reference data for the object for which the action was launched, and the data for all its parent objects in the objects tree. Variables are also used to reference input data specified by the user in the actions dialog. Variables are typically used in the **Command**, **Confirm**, **Result** and **SetVar** elements.

Variables are specified in the following format:

`${variableName}`

The following is an example for a **Rename Table** action. It first shows the name of the database connection (which is always present) along with information about the table being renamed. The last two input fields should be entered by the user and identify the new name of the table. The **New Database** control is a list from which the user should select the name of the new database. The new table name should be entered in the **New Table Name** field.

If the **Show SQL** control is enabled, you will see any edits in the dialog being reflected directly in the final SQL Preview.

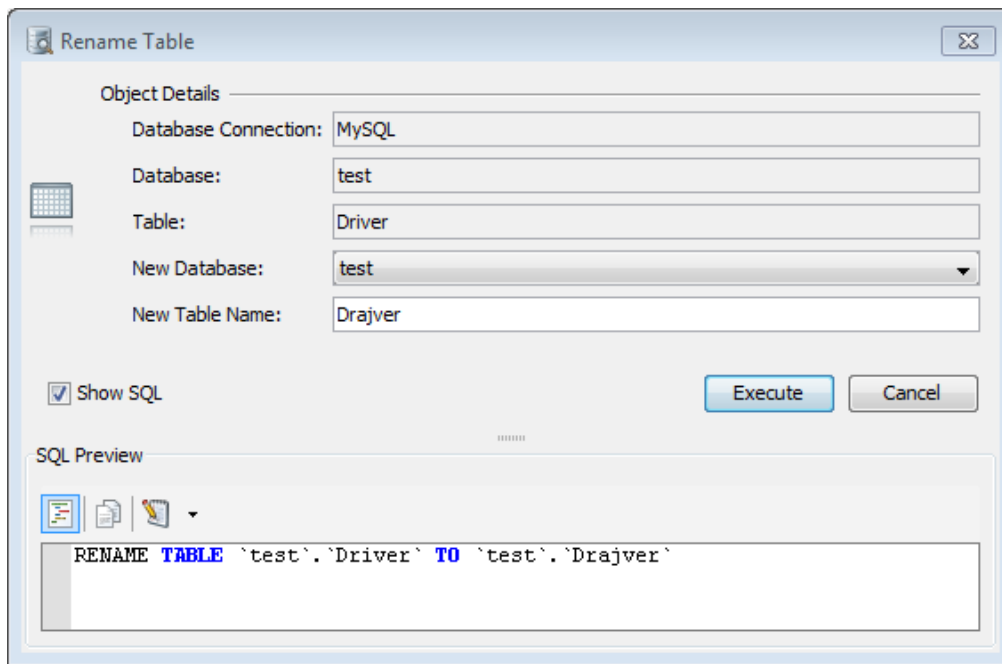


Figure: The default action handler

The complete action definition for the previous **Rename Table** action is as follows:

```
<Action id="mysql-table-rename" label="Rename Table" reload="true" icon="rename">
  <Input label="Database" style="text" editable="false">
    <Default>${catalog}</Default>
  </Input>
  <Input label="Table" style="text" editable="false">
    <Default>${objectname}</Default>
  </Input>
  <Input label="New Database" name="newCatalog" style="list">
    <Values>
      <Command><SQL><![CDATA[show databases]]></SQL></Command>
    </Values>
    <Default>${catalog}</Default>
  </Input>
  <Input label="New Table Name" name="newTable" style="text"/>
  <Command>
    <SQL>
      <![CDATA[
rename table `${catalog}`.`${objectname}`
to `${newCatalog}`.`${newTable}`
]]>
    </SQL>
  </Command>
  <Confirm>
    <![CDATA[
Confirm rename of ${catalog}.${objectname} to ${newCatalog}.${newTable}?
]]>
  </Confirm>
  <Result>
    <![CDATA[
Table ${catalog}.${objectname} renamed to ${newCatalog}.${newTable}!
]]>
  </Result>
</Action>
```

First, there is the **Action** element with some attributes specifying the label of the action, icon and whether the objects tree (and the current object view) should be reloaded after the action has been executed.

The next block of elements are **Input** fields defining the data for the action. As you can see in the example, there is a **`\${catalog}`** variable in the **Default** element for the **Database** input and an **`\${objectname}`** variable in the **Default** element for the **Table** input. The values for these variables are fetched from the selected object in the objects tree. Variables are evaluated by first checking if the variable is in the scope of the action dialog (i.e., another input field), then if the variable is defined for the object for which the action was launched, and then if it is defined for any of the parent objects until the root object in the tree (Connections node) is reached. If a variable is not found, its value is set to **(null)**.

In the previous XML sample, the value of the **`\${catalog}`** variable is the name of the database in which the table object is stored. The **`\${objectname}`** is the current name of the table (these variables are described in the ObjectsTreeDef section).

The **New Database** input field is a list component which shows a list of databases based on the result set of the specified SQL command. The **Default** setting for the database will be the database in which the table is currently stored based on the **`\${catalog}`** variable.

The **New Table Name** input field is a simple text field in which the user may enter any text.

Both the **New Database** and **New Table Name** fields are editable and should be specified by the user. This data is then accessible via the variables specified in the **name** attribute, i.e., **newCatalog** and **newTable**.

The **Command** element declares the SQL statement that should be executed by the action. In this example, the SQL combines static text with variables.

<ActionGroup>

The **ActionGroup** element is a container and groups **ActionGroup**, **Action** and **Separator** elements. It is used to define what actions should be present for a particular object type. It also defines in what order the actions should appear in the menu and where any separators should be. **ActionGroup** elements can be nested to create sub menus.

```
<ActionGroup type="Table">
```

The attributes for an **ActionGroup** are:

- **type**
this defines what object type the ActionGroup represents. This attribute is valid only for top level action groups. An example is the object of type **Table**, the corresponding **ActionGroup** will only be displayed when the selected object is a **Table**.
- **label**
this attribute is required for nested action groups. This label is displayed as the sub menu label for the nested action group. (The label attribute have no effect on top level action groups).

<Action>

The action element defines the action.

```
<Action      id = "oracle-table-drop"  
            icon = "remove"  
            label = "Drop Table..."  
            reload = "false"  
            mode = "execute"  
processmarkers = "false"  
            resulttype = "resultset"  
            resultaction = "ask"  
hideif      = "<condition>">
```

The attributes for an action are:

- **id**
the id for the action. The recommended syntax for the id is "**profileName-objectType-someGoodActionName**"
- **icon**
specifies an optional icon that should be displayed next to the label in the menu
- **label**
the label for the action as it should appear in the menu in the action dialog
- **reload**
specifies if the parent node (in the objects tree) should be reloaded after successful execution. This is recommended for actions that change the visual appearance of the object, such as remove, add or name change
- **mode** attribute, can be set to any of these:
 - **execute** (default)
show the action dialog, process user input and execute the final SQL within the scope of the action window

- **script**
show the action dialog, process user input and send the final SQL to the SQL Commander
- **script-immediate**
will not show the action dialog but instead pass the final SQL directly to the SQL Commander
- **processmarkers**
 - **true**
IN parameter markers in the SQL are processed with the JDBC driver. Not all drivers supports this
 - **false** (default)
parameter markers are not be processed
- **resulttype** specifies what kind of result is produced by the action.
 - **resultset** (default)
this is the default and indicates that the result is a standard result set produced by a SQL SELECT statement or stored procedure
 - **dbmsoutput**
this is specific for Oracle databases only and specifies that the output is produced by the DBMS_OUTPUT stored procedure
- **resultaction** attribute, is only valid in combination with **mode="execute"**. It can be set to any of:
 - **ask** (default)
if the action produced a result according to the setting of **resulttype**, ask the user whether the result should be displayed in a window or copied as text to the SQL Commander
 - **show**
if the action produced a result according to the setting of **resulttype**, show it in a window
 - **script**
if the action produced a result according to the **resulttype**, copy it to the SQL Commander.
- **hideif**
there may be situations when an action should not appear in the list of actions. The hideif attribute is used to express a condition which is evaluated when the list of object actions is created. Example: **hideif="#dataMap.get('actionlevel') neq 'toplevel'"**

<Input>

An **Input** element specifies the characteristics of a visible field component for the actions dialog. The label attribute is recommended and is presented to the left of input field. If a label is not specified, the input field will occupy the complete width of the action dialog. All input fields are editable by default. The **name** attribute is required for editable fields and should specify the identity of the variable in which the user input is stored.

This is a minimal definition of an input field. It will show a read-only text field control labeled **Size**.

```
<Input label="Size" editable="false"/>
```

If the input field is changed to be editable, the **name** attribute must be used to specify the identifier for the variable name.

```
<Input label="Size" editable="true" name="theSize"/>
```

Any input element may contain the tip attribute. It is used to briefly document the purpose of the input field and is displayed as a tooltip when the user hovers the mouse pointer over it.

```
<Input label="Size" editable="true" name="theSize" tip="Please enter the size of the new xxx"/>
```

The **hideif** attribute is useful to limit what <Input> fields should appear for an action. The condition specified in the **hideif** attribute have the same syntax as described in the [<SetVar>](#) section. Example:

```
<Input label="Unit" hideif="#dataMap.get('actionlevel') neq 'toplevel'">
```

Input fields can be aligned on a single row with the **linebreak** attribute. The default behavior is that every input field is displayed on a single row. Use the **linebreak="false"** attribute to define that the **next** input field will be arranged on the same line. To re-start the automatic line breaking feature you must use the **linebreak="true"** attribute.

```
<Input name="size" label="Size" style="number" linebreak="false">
  <Default>10</Default>
</Input>
<Input name="unit" style="list" linebreak="true">
  <Labels>KBIMB</Labels>
  <Values>KIM</Values>
  <Default>M</Default>
</Input>
```

The previous example shows the use of the linebreak attribute. The size number field and the unit list will appear on the same line.

Specifying the default value as a result from an SQL statement is a trivial task:

```
<Input label=Size" editable="true" name="theSize">
  <Default>
    <Command>
      <SQL>
select size from systables where tablename = '${objectname}'
      </SQL>
    </Command>
  </Default>
</Input>
```

Since **Default** here will execute an SQL statement, it will automatically pick the value in the first row's first column and present it as the default. SQL may be specified in the **Default** element when used for all styles while SQL in **Values** and the **Labels** elements are valid only for **list** and **radio** styles). In some rare situations it may not be possible to express a SQL statement that will return a single column that will be displayed for **Values**, **Labels** and **Default**. An example is when data is collected via a stored procedure. To solve this problem specify the **column** attribute that takes the value either by the actual column name or column index:

```
<Input label=Size" editable="true" name="theSize">
  <Default column="2">
    <Command idref="getSize">
      <Input name="objectname" value="${objectname}"/>
    </Command>
  </Default>
</Input>
```

or by column name:

```
<Input label=Size" editable="true" name="theSize">
  <Default column="THE_SIZE">
    <Command idref="getSize">
      <Input name="objectname" value="${objectname}"/>
    </Command>
  </Default>
</Input>
```

An alternative to embedding the SQL in the element body, as in the previous example, is to refer to a command via the standard **idref** attribute:

```
<Input label=Size" editable="true" name="theSize">
  <Default>
    <Command idref="getSize">
      <Input name="objectname" value="${objectname}"/>
    </Command>
  </Default>
</Input>
```

Instead of having duplicated SQLs in multiple actions, consider replacing these with **Command** elements referred via the **idref** attribute.

Referring commands in actions via the **idref** attribute is recommended when the same SQL is used in several actions. Use Input elements to pass parameters to the command.

The following sections presents the supported **styles** that can be used in the **Input** element.

text (single line)

The **text** style is used to present single-line data in a text field.

```
<Input label="Enter your userid" name="userid" style="text">
  <Default>agneta</Default>
</Input>
```

- The optional **Default** element is used to define a default value for the field. Variables, static text and **Command** elements can be used to define the default value.
- A text input is editable by default. To make it read only just specify **editable="false"**

text-editor (multi line)

A **text-editor** field is the same as the **text** style except that it presents a multi-line field.

```
<Input label="Description" name="desc" style="text-editor" editable="true" args="height=50"/>
```

The **args="height=50"** attribute defines the height (in DLU) for the text-editor. The default height is 30 DLU's.

number

A **number** style is the same as **text** except that it only accept number values.

```
<Input label="Size" name="size" style="number" editable="true"/>
```

password

A **password** field is the same as **text** except that it masks the value as "****".

```
<Input label="Password" name="pw" style="password" editable="true"/>
```

Note that the password is visible in plain text in the SQL Preview.

list (large number of choices)

The list style displays a list of choices in a drop-down component. The **list** can be editable, meaning that the field showing the selection may be editable by the user. Here is a sample XML for the list style.

```
<Input label="Select index type" name="type" style="list">
  <Values>Pizza|Pasta|Burger</Values>
  <Default>Pasta</Default>
</Input>
```

The **Values** element should, for static entries, list all choices separated by a vertical bar (|) character. A **Default** value can either list the name of the default choice or the index number (first choice starts at 0). In the example above, setting **Default** to **{2}** would set **Burger** to the default selection.

It is also possible to use the **Labels** element. If present, this should list all choices as they will appear in the actions dialog. Consider these as being the labels shown to the user, while **Values** in this case should list the choices that will go into the final SQL via the variable. Here is an example:

```
<Input label="Select index type" name="type" style="list">
  <Values>Pizza|Pasta|Burger</Values>
  <Labels>Pizza the French style|Pasta Bolognese|Texas Burger</Labels>
  <Default>Pasta</Default>
</Input>
```

If the users selects **Texas Burger** then the value for variable **type** will be **Burger**.

The following shows how to use SQL to feed the list of values:

```
<Input label="New Database" name="newCatalog" style="list">
  <Values>
    <Command>
      <SQL>
```

```

      <![CDATA[
show databases
      ]]>
    </SQL>
  </Command>
</Values>
<Default>${catalog}</Default>
</Input>

```

Here a **Command** element is specified as a sub element to **Values**. The result of the **show databases** SQL will be presented in the list component.

To make the list editable, specify the attribute **editable="true"**.

radio (limited number of choices)

The **radio** style displays a list of choices organized as button components. The only difference between the radio and list styles are:

- All choices for a radio style are displayed on the screen (better overview of choices but suitable only for a limited number of choices)
- The `args="vertical"` attribute can be specified for radio style to present the radio choices vertically

See the list style for complete capabilities of the radio style.

check (true/false, on/off, selected/unselected)

The **check** style is suitable for yes/no, true/false, here/there types of input. Its enabled state indicates that the **Value** for the input will be set in the final variable. If the check box is disabled, the variable value is blank

```

<Input label="Cascade Constraints" name="cascade" style="check">
  <Values>compact</Values>
</Input>

```

- This will create a check component with the label **Cascade Constraints**
- Enabling the check box will set the value of the variable identified by **name** (cascade) to the value of **Value**, which is **compact**.
- If the check box is unchecked, the variable value will be blank

separator (visual divider between input controls)

The **separator** style is not really an input element but is instead used to visually divide the fields in the in the actions dialog. If the **label** attribute is specified, it will be presented to the left of the separator line. If no label is specified, only the separator is displayed.

```

<Input label="Parameters" style="separator"/>

```

The separator is a useful substitute for the standard label presented to the left of every input field. Here is a sample:

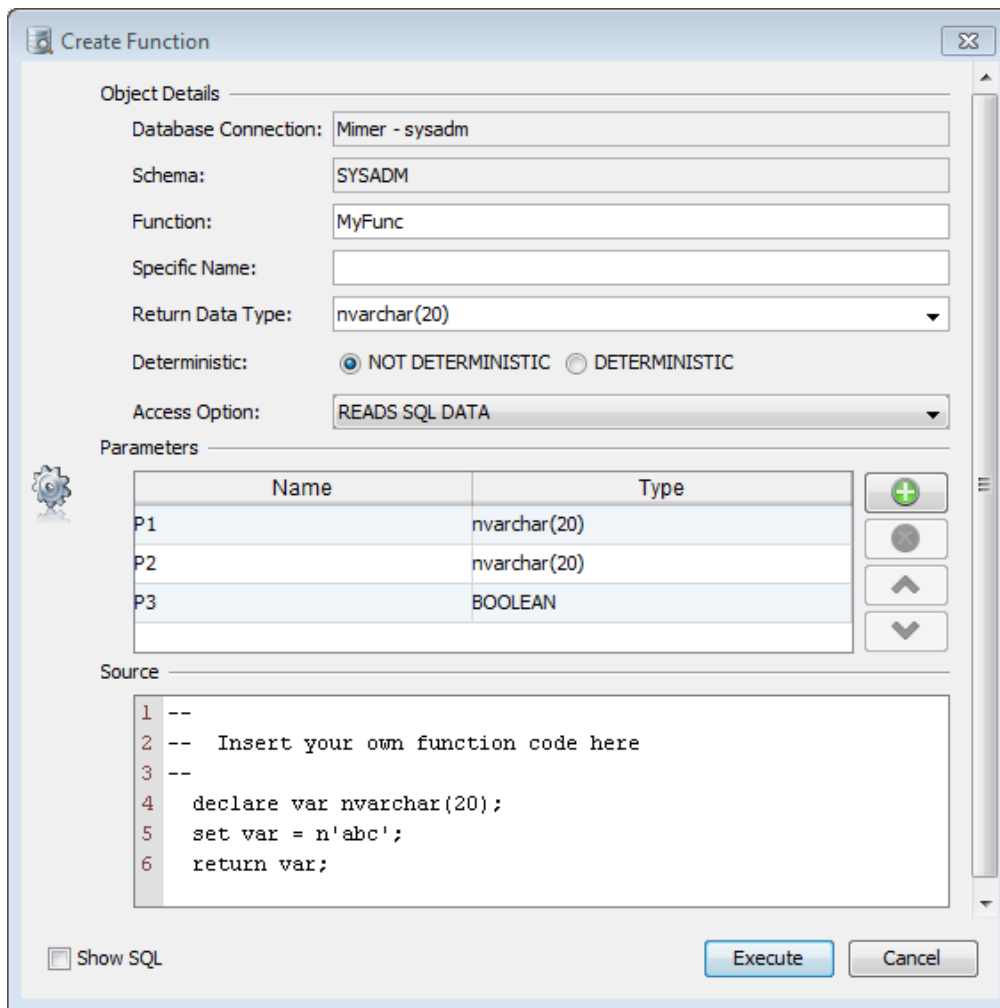


Figure: Sample showing separators and wide fields

The previous figure shows the use of separators and two fields that extend to the full width of the action dialog. The separators for **Parameters** and **Source** are here used as alternatives to labels for the fields below them.

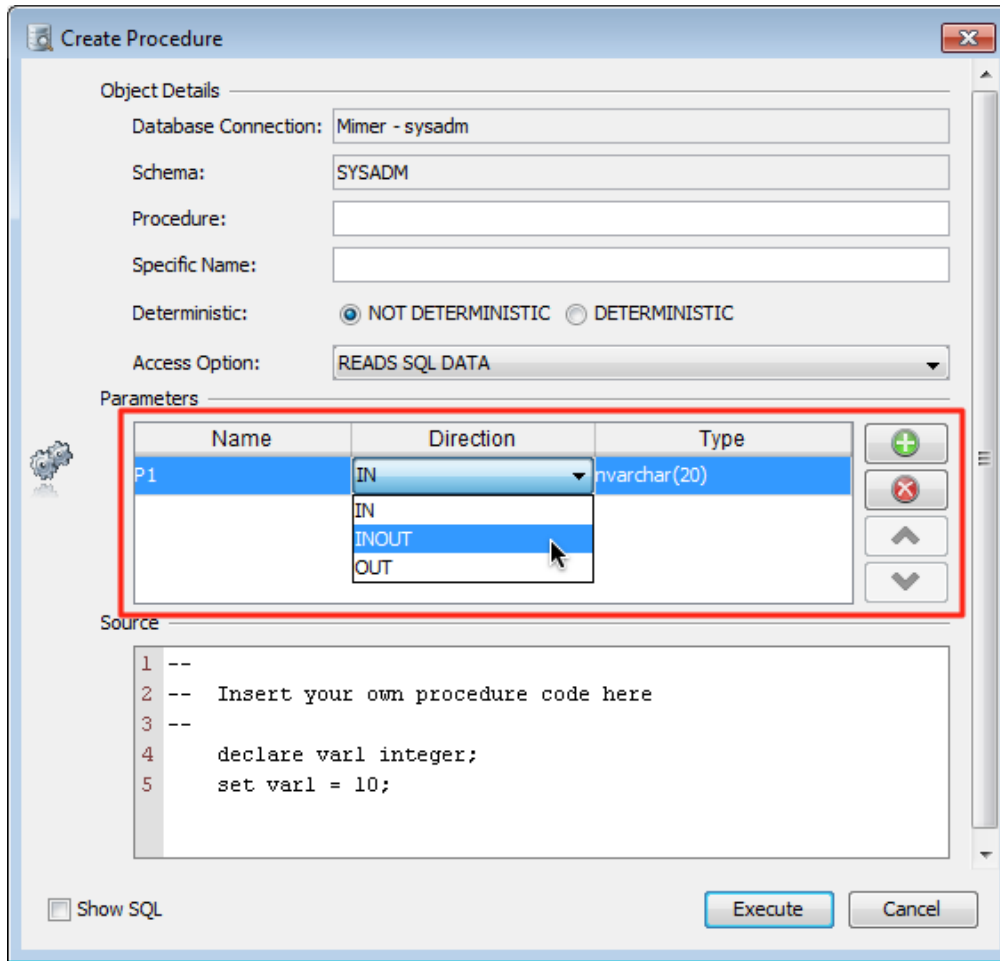
grid (configurable multi row inputs)

The **grid** input style is presented as a grid with user controls to add, remove and move rows. The columns that should appear in the grid are defined by using any of the primitive styles: **text**, **number**, **password**, **check**, **list** and **radio**. The grid style is useful for data that allows the user to define multiple entries. Examples are, defining columns that should appear in a table index, setup data files for a tablespace or databank.

This example shows a grid style definition that will ask the user for parameters that will be part of a **create procedure** action.

```
<Input name="parameters" style="grid">
  <Arg name="output" value="\${direction} \${name} \${type}\${_default}"/>
  <Arg name="newline" value=", "/>
  <Input name="name" label="Name" style="text">
    <Default>parm</Default>
  </Input>
  <Input name="direction" label="Direction" style="list">
    <Values>IN|INOUT|OUT</Values>
    <Default>IN</Default>
  </Input>
  <Input name="type" label="Type" style="text">
    <Default>nvarchar(20)</Default>
  </Input>
</Input>
```

Here is how it looks:



The sub elements for the grid style is different from the other input styles as it accepts sub **<Input>** elements. These input styles defines what columns should appear in the grid and the first input style will appear to the leftmost and the last in the rightmost column.

This example doesn't specify the label attribute as we want the grid to extend the full width of the actions dialog. The **grid** style use the **<Arg>** elements to customize the appearance and function of the field. The following arguments are handled by the grid style:

- **output**
Defines the output format for each row in the grid. The value may contain variables and static text. To create conditional output check the **<SetVar>** element below
- **newline**
Defines the static text that should separate every row in the grid. A "n" somewhere in the value will be converted to a true newline in the final output
- **rowprefix**
Specifies any prefix for every row in the grid
- **rowsuffix**
Specifies any suffix for every row in the grid

The resulting parameter list is created automatically by the control and is available in the variable name specified in the example to be **parameters**.

The **<SetVar>** element in the context of a grid style is used to process the data that will appear as defined by the **<Arg name="output">** element. It is used to process the data for every row in the grid. Let's say that the output must contain the word "default" if the value in a column named "Default" is entered. **<SetVar>** is used to handle this:

```
<SetVar name="_default" value='#default.equals("") ? "" : " default " + #default'/>
```

The #default input value is here evaluated and if it is not empty the " default " text s prefixed to the value of the #default value. The result is stored in the "_default" variable which is also refered in the output argument above.

<SetVar>

The **SetVar** element is very powerful, as it is used to do conditional processing and create new variables based on the content of other variables.

Consider an SQL statement for creating new users in the database:

```
create user 'user' identified by 'password'
```

In this case it is quite easy to map the **user** field to an **Input** element for the action since it is a required field. The question arise for **password** which is optional. The **identified by** clause should only be part of the final SQL if the password is entered by the user. The solution for this scenario is to use the **SetVar** element. Here is the complete action definition:

```
<Action id="mydb-user-create" label="Create User" reload="true" icon="add">
  <Input label="Userid" name="userid" style="text"/>
  <Input label="Password" name="password" style="password"/>
  <SetVar name="_password" value='#password.equals("") ? "" : " identified by \" + #password + "\"' />
  <Command>
    <SQL>
      <![CDATA[
create user ${userid} ${_password}
      ]]>
    </SQL>
  </Command>
</Action>
```

The **SetVar** element accepts two attributes:

- **name**
should specify the name of the new variable
- **value**
this should contain the expression that will be evaluated. The expression is based on the OGNL toolkit provided by www.ognl.org. This is an expression library that mimics most of what is being supported by Java. Variables are referenced as **#variableName**.

The expression in the example above checks whether the **password** variable is empty. If it is empty, a blank value is being assigned to the **_password** variable. If it is not empty, the value for **_password** will be set to **identified by "theEnteredPassword"**.

The SQL in the **Command** element now refer the new **\$_password** variable instead of the original **#{password}**.

It is recommended that variables produced via **SetVar** elements are prefixed with an underline () to highlight were they come from.

<Confirm>

The **Confirm** element is displayed to the user when a request to **Execute** the action is made. If there are only read-only input fields in the action, this message is displayed in the body of the action dialog. The message is displayed in a confirmation dialog if there are editable fields.

```
<Confirm>Really drop table ${table}?</Confirm>
```

Note that the message text can be composed of HTML tags such as ****, **<i>**, **
**, etc.

<Result>

The **Result** element is optional and if specified, it is shown in a dialog after successful execution.

NOTE: Result elements are currently not displayed in DbVisualizer. It is however recommend that you specify these as they will most likely appear in some way or another in a future version. If you want to test the appearance of Result elements then open the **DBVIS-HOME/resources/dbvis-custom.xml** file in a text editor and make sure **dbvis.showactionresult** is set to **true**.

```
<Result>Table ${table} has been dropped!</Result>
```

- The **Result** message will be displayed in a dialog after successful execution.
- If the execution fails, a generic error dialog is displayed and the **Result** is not displayed.

<Command>

The **Command** element specifies the SQL code that is executed by the action.

```
<Command>
  <SQL>
    <![CDATA[
drop table ${table} mode ${mode} including constraints ${includeconstraints}
    ]]>
  </SQL>
</Command>
```

Conditional processing

Conditional processing means that a profile can adjust its content based on certain conditions. A few examples:

- Which version of the database it is
- The format of the database URL
- The client environment i.e Java versions, vendor, etc.
- User properties
- Database connection properties

Conditional processing is especially useful for adapting the profile for different versions of the database (and/or JDBC driver). Another use for the conditional processing is to replace generic error messages with more user friendly messages.

Programmers familiar with **if**, **else if** and **else** will easily learn the conditional elements.

Depending on in which of the two phases the conditions should be processed, some restrictions and rules apply. Please read the following sections for more information.

When are conditional expressions processed?

There are two phases when conditions are processed:

1. Conditional processing when database connection is established

<If>, <Elseif> and <Else> elements can be specified almost everywhere in the profile.

2. Conditional processing during command execution

The <OnError> element is used to define a message that will appear in DbVisualizer if a command fails. Conditions are used to control what message should appear.

DbVisualizer uses the **type** attribute to determine which **If** elements should be executed in which phase. If this attribute has the value **runtime**, it will be processed in the second phase. If it is not specified or set to **load**, it will be processed in the first phase.

Conditional processing when database connection is established

The following example shows the use of conditions that are processed during connect of the database connection.

```
<Command id="sybase-ase.getLogins">
  <If test="#DatabaseMetaData.getDatabaseMajorVersion() lte 8">
    <SQL>
      <![CDATA[
select name from master.dbo.syslogins
      ]]>
    </SQL>
```

```

</If>
<ElseIf test="#DatabaseMetaData.getDatabaseMajorVersion() eq 9">
  <SQL>
    <![CDATA[
select name, suid from master.dbo.syslogins
    ]]>
  </SQL>
</ElseIf>
<Else>
  <SQL>
    <![CDATA[
select name, suid, dbname from master.dbo.syslogins
    ]]>
  </SQL>
</Else>
</Command>

```

The above means that if the major version of the database being accessed is less than or equal to 8, the first SQL is used. If the version is equal to 9, the second SQL is used, and the last SQL is used for all other version. The test attribute may contain conditions that are ANDed or ORed. Conditions can contain multiple evaluations, combined using parenthesis. The **If**, **Elseif** and **Else** elements may be placed anywhere in the XML file.

Here is another example that controls whether certain nodes will appear in the database objects tree or not.

```

<!-- Getting Table Engines was added in MySQL 4.1 -->
<If test="( #dm.getDatabaseMajorVersion() eq 4 and #dm.getDatabaseMinorVersion() gte 1)
or #dm.getDatabaseMajorVersion() gte 5">
  <GroupNode type="TableEngines" label="Table Engines" isLeaf="true"/>
  <!-- "Errors" was added in MySQL 5 -->
  <If test="#dm.getDatabaseMajorVersion() gte 5">
    <GroupNode type="Errors" label="Errors" isLeaf="true"/>
  </If>
</If>

```

As you can see, this example contains nested uses of If.

Conditional processing during command execution

Using conditional processing to evaluate any errors from a **Command** may be useful to rephrase error messages to be more user friendly.

```

<Commands>
  <OnError>
    <!-- The ORA-942 error means "the table or view doesn't exist" -->
    <!-- It is caught here since these errors typically indicates -->
    <!-- that the user don't have privileges to access the SYS and/or -->
    <!-- V$ tables. -->
    <If test="#result.getErrorCode() eq 942" context="runtime">
      <Message>
        <![CDATA[
You don't have the required privileges to view this object.
        ]]>
      </Message>
    </If>
    <ElseIf test="#result.getErrorCode() eq 17008" context="runtime">
      <Message>
        <![CDATA[
Your connection with the database server has been interrupted!
Please <a href="connect" action="connect">reconnect</a> to re-establish the connection.
        ]]>
      </Message>
    </ElseIf>
  </OnError>
  ...
</Commands>

```

The **OnError** element can be used in **Commands** and **Command** elements. If used in **Commands** element, its conditions are processed for all commands. If it is part of a specific **Command**, it is processed only for that command.

Current limitations

- The **SQL** statements in the profile must be statements that DbVisualizer can execute with JDBC. It can not contain any executables, scripts or OS specific calls
- It is not possible to specify conditions or compound commands, i.e., everything needed to execute a command must be expressed in a single SQL statement.