DbVisualizer 10.0

Users Guide

Table of Contents

1 DbVisualizer 10.0

2 Getting Started

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.

The screenshots throughout the users guide are produced on Windows 7 using the Windows Look and Feel, but DbVisualizer lets you choose among other Look and Feels as well.

In addition to this Users Guide, the following online resources may be be useful:

1. The home of DbVisualizer (http://www.dbvis.com/),
2. The FAQ (http://confluence.dbvis.com/display/FAQ) which is regularly updated with frequently asked questions and known problems,
3. The Databases and JDBC Drivers (http://www.dbvis.com/doc/database-drivers/) online page. This page gives information about supported databases and JDBC drivers,
4. The DbVisualizer forums (http://www.dbvis.com/forum/).

2.1 Downloading

DbVisualizer installers are available on our web site at http://www.dbvis.com/download/.

Download the installer for your operating system that fits your needs:

- Without Java VM if you already have Java installed,
- WIth Java VM if you do not have Java installed, or if you want to use the recommended Java version for DbVisualizer and another Java version for other applications,
- An Installer unless you must use an archive format for some reason.

2.2 Installing

There are two ways to install DbVisualizer: using an Installer or extracting files from an archive file.

- Installing with an Installer (see page 14)
- Installation from an archive file (see page 14)
    - Installation Notes for ZIP archives (Windows) (see page 14)
    - Installation Notes for TAR archives (Unix) (see page 14)
    - Installation Notes for RPM archives (Linux) (see page 14)
    - Installation Notes for DEB archives (Linux) (see page 14)

2.2.1 Installing with an Installer

To install DbVisualizer, just execute the Installer you have downloaded and follow the instructions in the screens
.

2.2.2 Installation from an archive file

Installation Notes for ZIP archives (Windows)

All files are contained in an enclosing folder named **DbVisualizer**.
Unpack the distribution file with the built-in zip archive extraction utility in Windows or with the **winzip** utility.

The ZIP archive installer will not add any entries to the Start menu, add desktop launchers or even register the software in the Windows registry.

Start DbVisualizer by clicking the *dbvis.exe* file in the installation directory for DbVisualizer.

To uninstall DbVisualizer installed via a ZIP archive, simply delete the complete DbVisualizer directory.

Installation Notes for TAR archives (Unix)

All files are contained in an enclosing folder named **DbVisualizer**.
Unpack the distribution file with:

```
gunzip dbvis_unix_10_0.tar.gz
tar xf dbvis_unix_10_0.tar
```

`Start DbVisualizer by executing the shell script in the installation directory, e.g`

To uninstall DbVisualizer installed via a TAR archive, simply delete the complete DbVisualizer directory.

Installation Notes for RPM archives (Linux)

Install the rpm with *rpm -i <download_filename>* or your favorite rpm tool.

Start DbVisualizer by executing the shell script in the installation directory, e.g. *DbVisualizer/dbvis*.

To uninstall DbVisualizer installed via an RPM archive, use the rpm utilities.

Installation Notes for DEB archives (Linux)

Install the package with *sudo dpkg -i <download_filename>.deb* or your favorite Debian package manager tool.

Start DbVisualizer by executing the shell script in the installation directory, e.g. *DbVisualizer/dbvis*.

To uninstall DbVisualizer installed via an DEB archive, use *dpkg --remove* .

2.3 Starting DbVisualizer

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

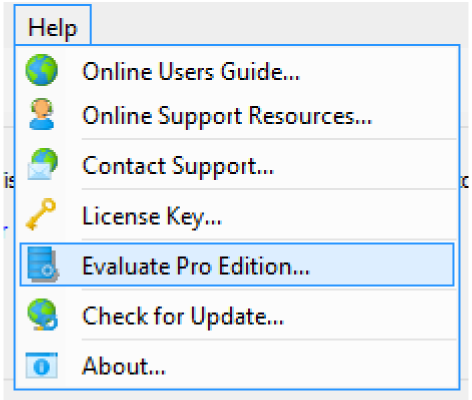
- **Windows**
  In the **Start** menu, select the **DbVisualizer** menu item.
- **Linux/Unix**
  Open a shell and change directory to the DbVisualizer installation directory. Execute the *dbvis* program
- **macOS**
  Double click on the **DbVisualizer** application or the **DbVisualizer.app** application bundle.

You can also start DbVisualizer with the bundled script files, please see the GUI Command Line Arguments (see page 429) page for details. For tasks that do not require a GUI, such as tasks scheduled via the operating system's scheduling tool, you can also use the pure command line interface (see page 319).

2.4 Evaluating the Pro Edition

The DbVisualizer Pro edition offers far more features than the Free edition. If you are using the Free edition, it is easy to activate a Pro edition evaluation to see if suits your needs:

1. Open **Help->Evaluate Pro Edition**



2. Enter your email address and click **Evaluate,**
3. Click **Restart** when prompted after the activation of the evaluation.

> ⚠ If you start DbVisualizer with one of the bundled scripts rather than with the launcher, you need to manually restart DbVisualizer after the activation.

2.5 Installing a Pro Edition License

To enable the Pro edition features, you need to install the License Key String or License Key File that you received after purchasing the license.

2.5.1 Installing a License Key String

1. Select and copy the License Key String included in the email,
2. Start DbVisualizer and select the **Help->License Key** main menu choice,
3. Select License Key String as the **License Type**,
4. Paste the key string into the text area,
5. Click **Install License**,
6. Restart DbVisualizer when prompted to do so.

The DbVisualizer main window should now say **DbVisualizer Pro** in the window title. You're ready to go.

2.5.2 Installing a License Key File

1. Save the *dbvis.license* file attached to the email to disk,
2. Start DbVisualizer and select the **Help->License Key** main menu choice,
3. Select License Key File as the **License Type**,
4. In the **License Key File** field, enter the path to the newly saved *dbvis.license* file or click the button to the right of the field to open a file browser to locate the file,
5. Click **Install License**,
6. Restart DbVisualizer when prompted to do so.

> ℹ An option to saving the *dbvis.license* file to disk is to drag it from you mail application (or elsewhere) into the **License Key File** field in the **Help->License Key** window.

The DbVisualizer main window should now say DbVisualizer Pro in the window title. You're ready to go.

2.5.3 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	Filename
Windows	*C:\Users\<user>\.dbvis\dbvis.license*
UNIX/Linux	*/home/<user>/.dbvis/dbvis.license*
macOS	*/Users/<user>/.dbvis/dbvis.license*

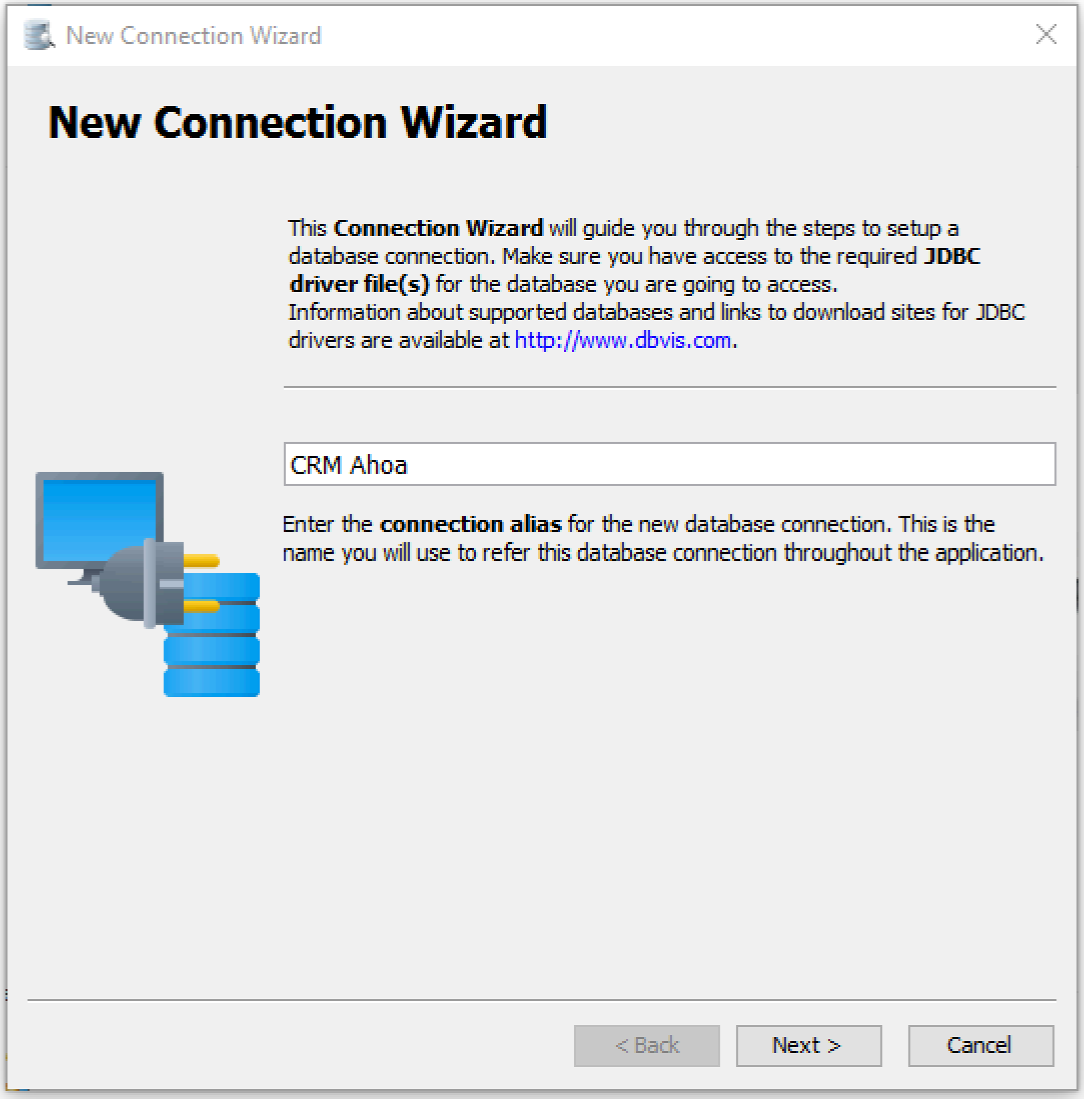
2.6 Creating a Connection - basics

To access a database with DbVisualizer, you must first create and setup a Database Connection. The easiest way to set up a connection is to use the Connection Wizard, but you can also do it manually.

- Using the Connection Wizard (see page 17)
- Setting Up a Connection Manually (see page 21)

2.6.1 Using the Connection Wizard

1. Launch the wizard from **Database->Create Database Connection** and click **Use Wizard** when prompted ,
2. Enter a name for the connection on the first Wizard page and click **Next**,



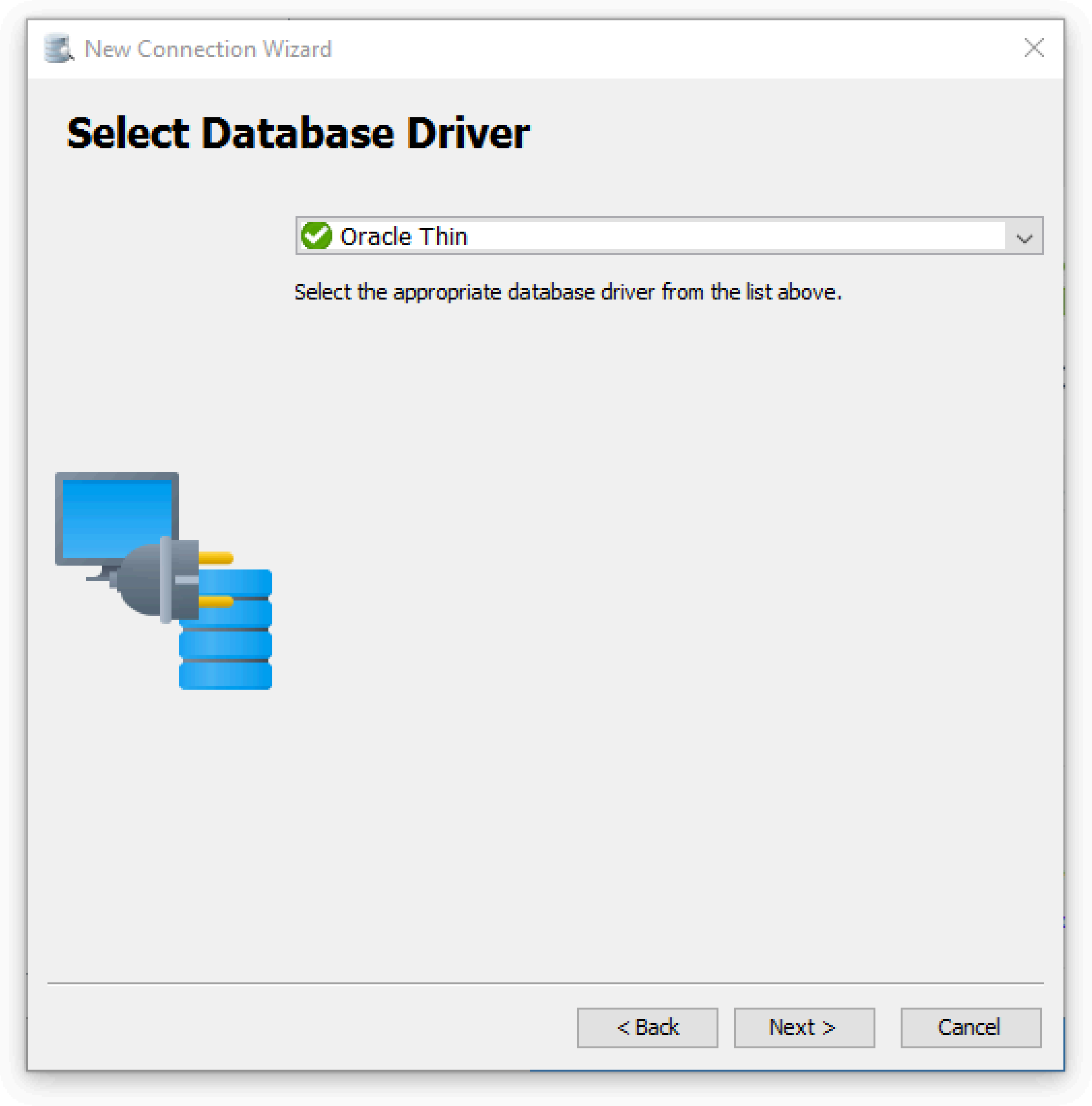
New Connection Wizard

This **Connection Wizard** will guide you through the steps to setup a database connection. Make sure you have access to the required **JDBC driver file(s)** for the database you are going to access.
Information about supported databases and links to download sites for JDBC drivers are available at http://www.dbvis.com.

CRM Ahoa

Enter the **connection alias** for the new database connection. This is the name you will use to refer this database connection throughout the application.

3. Select an installed JDBC driver (marked with a green checkmark) on the second wizard page (see Installing a JDBC Driver (see page 430) for how to install a JDBC driver manually),

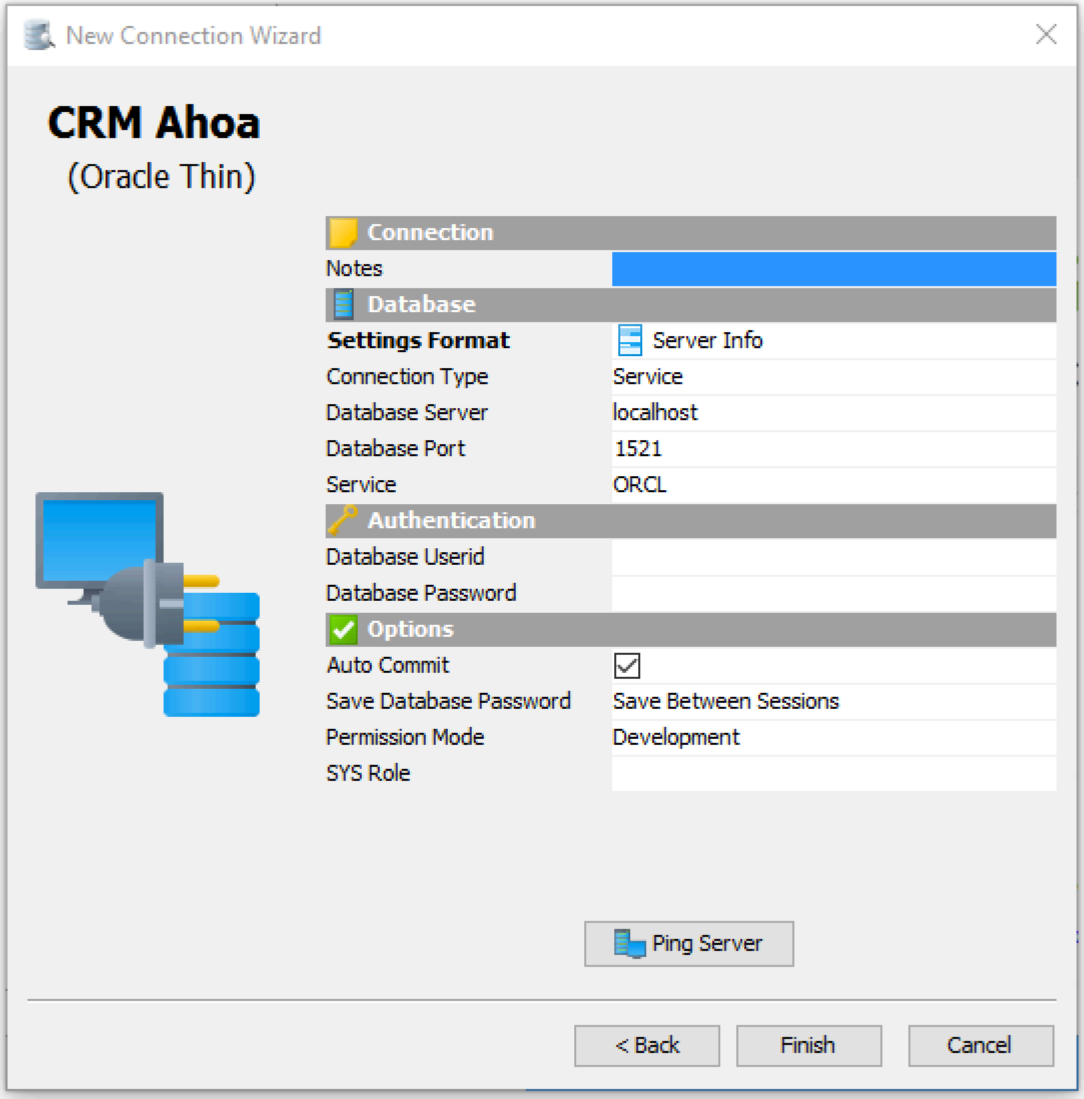


Select Database Driver

✓ Oracle Thin

Select the appropriate database driver from the list above.

4. Enter information about the database server on the third wizard page (see below for details),



Connection	
Notes	
Database	
Settings Format	Server Info
Connection Type	Service
Database Server	localhost
Database Port	1521
Service	ORCL
Authentication	
Database Userid	
Database Password	
Options	
Auto Commit	☑
Save Database Password	Save Between Sessions
Permission Mode	Development
SYS Role	

5. Verify that a network connection can be established to the specified address and port by clicking the **Ping Server** button,
6. If Ping Server shows that the server can be reached, click the **Finish** button to create the connection.
7. An Object View tab for the new connection is opened automatically. If the connection to the database can be established, the Connection Message area in the tab shows the database and driver versions, otherwise it shows a message about what is wrong.

See Fixing Connection Issues (see page 422) for some tips if you have problems connecting to the database.

The information about the database server that needs to be entered depends on the which JDBC driver you use. For most drivers, you need to specify:

Field	Description
Database Server	The IP address or DNS name for the server where the database runs.
Database Port	The TCP/IP port used by the database.
Database Userid	The database user account name. Enter `(null)` to not send an account name.
Database Password	The database user account password. Enter `(null)` to not send a password.

For some database such as Oracle, you may use a TNS name (see page 301) instead of specifying the server and port. Other drivers may add more fields that are driver specific.

You may also optionally specify SSH tunneling information (see page 298) and Options, such as:

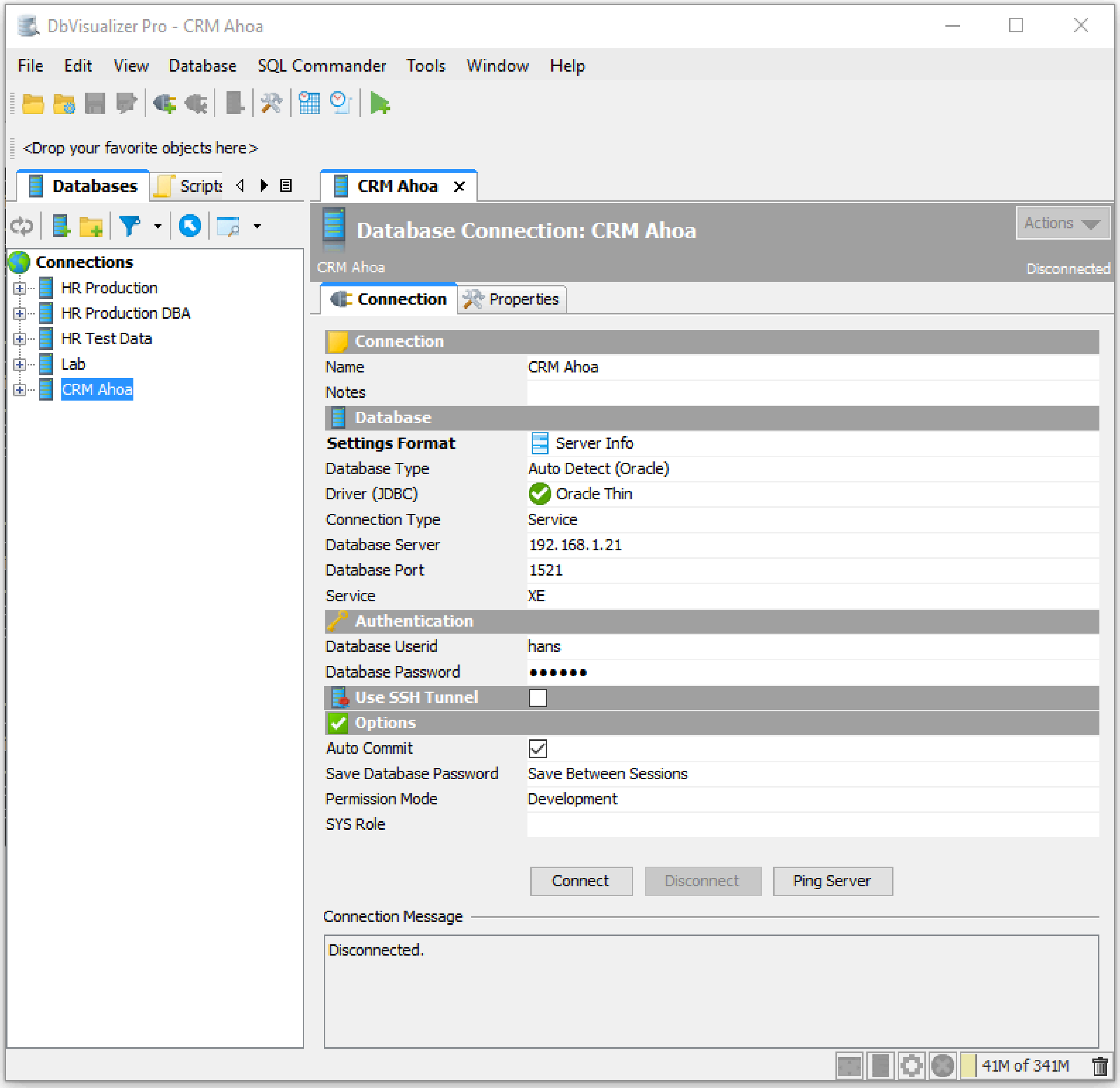
Option	Description
Auto Commit	Check if you want to enable auto commit in the SQL Commander by default for the connection.
Save Database Password	Check if you want the password to be saved (encrypted) during the session, between sessions, or cleared when you disconnect.
Permission Mode	One of **Development**, **Test** or or **Production** to select which set of Permissions (see page 285) to use.

Additional options are available for some JDBC drivers, such as Authentication Method (see page 303) for the SQL Server jTDS driver.

See the Configuring Connection Properties (see page 285) page for related topics.

2.6.2 Setting Up a Connection Manually

1. Create a new connection from **Database->Create Database Connection** and click **No Wizard** when prompted. An **Object View** tab for the new connection is opened,



Connection	
Name	CRM Ahoa
Notes	
Database	
Settings Format	Server Info
Database Type	Auto Detect (Oracle)
Driver (JDBC)	✓ Oracle Thin
Connection Type	Service
Database Server	192.168.1.21
Database Port	1521
Service	XE
Authentication	
Database Userid	hans
Database Password	••••••
Use SSH Tunnel	□
Options	
Auto Commit	☑
Save Database Password	Save Between Sessions
Permission Mode	Development
SYS Role	

Connection Message

Disconnected.

2. Enter a name for the connection in the **Name** field, and optionally enter a description of the connection in the **Notes** field,
3. Leave the **Database Type** as **Auto Detect,**
4. Select an installed JDBC driver (marked with a green checkmark) from the Driver (JDBC) list (see Installing a JDBC Driver (see page 430) for how to install a JDBC driver manually),
5. Enter information about the database server in the remaining fields (see below for details),
6. Verify that a network connection can be established to the specified address and port by clicking the **Ping Server** button,
7. If Ping Server shows that the server can be reached, click **Connect** to actually connect to the database server.

See Fixing Connection Issues (see page 422) for some tips if you have problems connecting to the database.

Alternatively, you can set the **Settings Format** to **Database URL** (this is the only choice for some custom JDBC drivers). This replaces the fields for information about the database server with a single **Database URL** field, where you can enter the JDBC URL.

The information about the database server that needs to be entered depends on the which JDBC driver you use. For most drivers, you need to specify:

Field	Description
Database Server	The IP address or DNS name for the server where the database runs.
Database Port	The TCP/IP port used by the database.
Database Userid	The database user account name. Enter `(null)` to not send an account name.
Database Password	The database user account password. Enter `(null)` to not send a password.

For some database such as Oracle, you may use a TNS name (see page 301) instead of specifying the server and port. Other drivers may add more fields that are driver specific.

You may also optionally specify SSH tunneling information (see page 298) and Options, such as:

Option	Description
Auto Commit	Check if you want to enable auto commit in the SQL Commander by default for the connection.
Save Database Password	Check if you want the password to be saved (encrypted) during the session, between sessions, or cleared when you disconnect.
Permission Mode	One of **Development**, **Test** or or **Production** to select which set of Permissions (see page 285) to use.

Additional options are available for some JDBC drivers, such as Authentication Method (see page 303) for the SQL Server jTDS driver.

See the Configuring Connection Properties (see page 285) page for related topics.

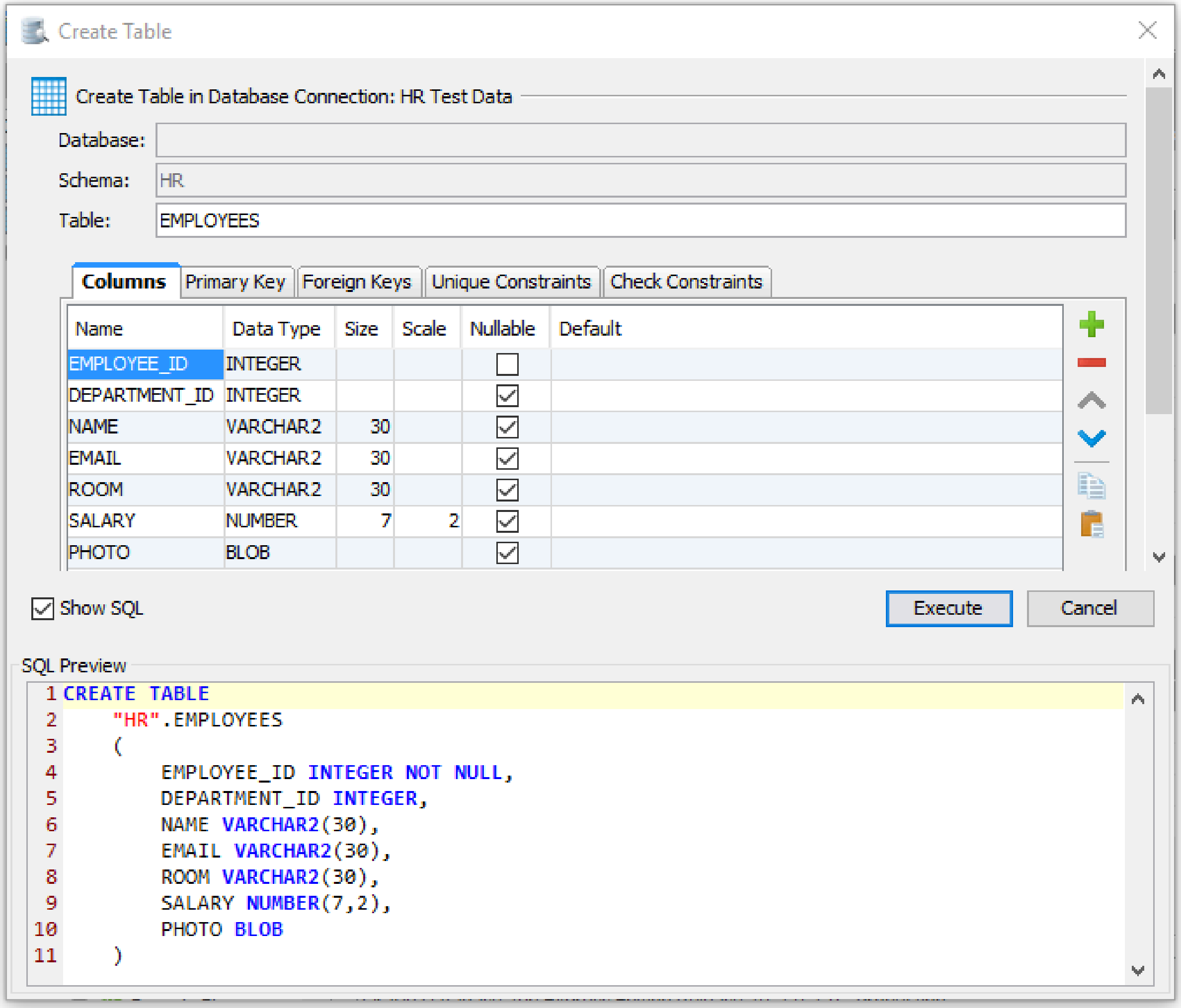
2.7 Creating a Table - basics

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

To create a new table:

1. Expand nodes in the **Databases** tab tree under the connection node until you reach the **Tables** node,
2. Select the **Tables** node and launch the **Create Table** dialog from the right-click menu:



Create Table in Database Connection: HR Test Data

Database:

Schema: HR

Table: EMPLOYEES

Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			☐	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR2	30		☑	
EMAIL	VARCHAR2	30		☑	
ROOM	VARCHAR2	30		☑	
SALARY	NUMBER	7	2	☑	
PHOTO	BLOB			☑	

☑ Show SQL

SQL Preview

```
CREATE TABLE
    "HR".EMPLOYEES
    (
        EMPLOYEE_ID INTEGER NOT NULL,
        DEPARTMENT_ID INTEGER,
        NAME VARCHAR2(30),
        EMAIL VARCHAR2(30),
        ROOM VARCHAR2(30),
        SALARY NUMBER(7,2),
        PHOTO BLOB
    )
```

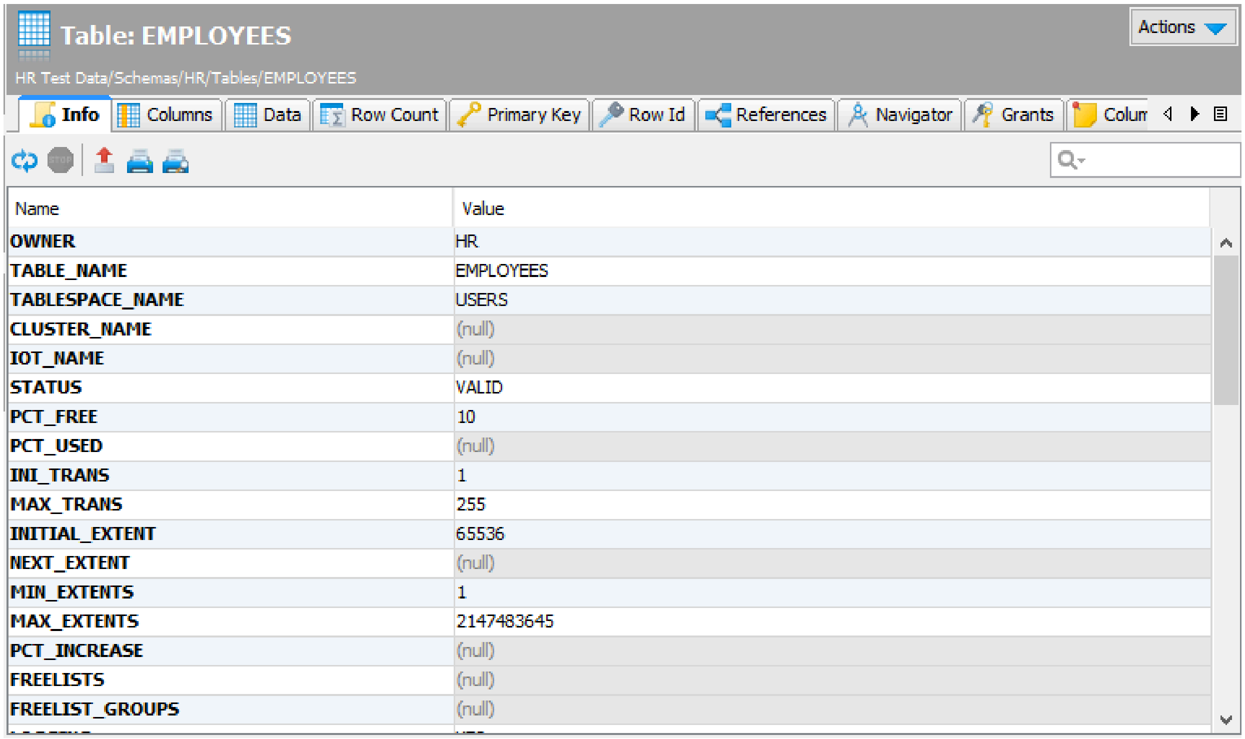
3. Add columns and constraints in the different tabs,
4. Click the **Execute** button to create the table.

You can learn more about the Create Table dialog in the Creating a Table (see page 60) page.

2.8 Viewing a Table - basics

To view details about a database table:

1. Expand nodes in the **Databases** tab tree under the connection node until you find the table,
2. Double-click on the table node to open its **Object View** tab.



Name	Value
OWNER	HR
TABLE_NAME	EMPLOYEES
TABLESPACE_NAME	USERS
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)

The Object View has a number of sub tabs. Exactly which sub tabs are available depends on the database type, but these are common:

Subtab	Description
Info	Brief information about the table.
Columns	Information about all table columns, e.g. data types and sizes.
Data	Then table data. Here you can view (see page 82) and edit (see page 94) the data.
Row Count	The number of rows in the table.
Primary Key	Information about the table's primary key columns, if any.
Indexes	Information about the table's indexes, if any.
Grants	Information about granted privileges for the table.
DDL	Shows the CREATE statement for the table.
References	Shows declared primary/foreign key relationships to other tables. Please read more in Viewing Table Relationships (see page 123).

Subtab	Description
Navigator	Navigate through the declared relationships. Please read more in Navigating Table Relationships (see page 124).

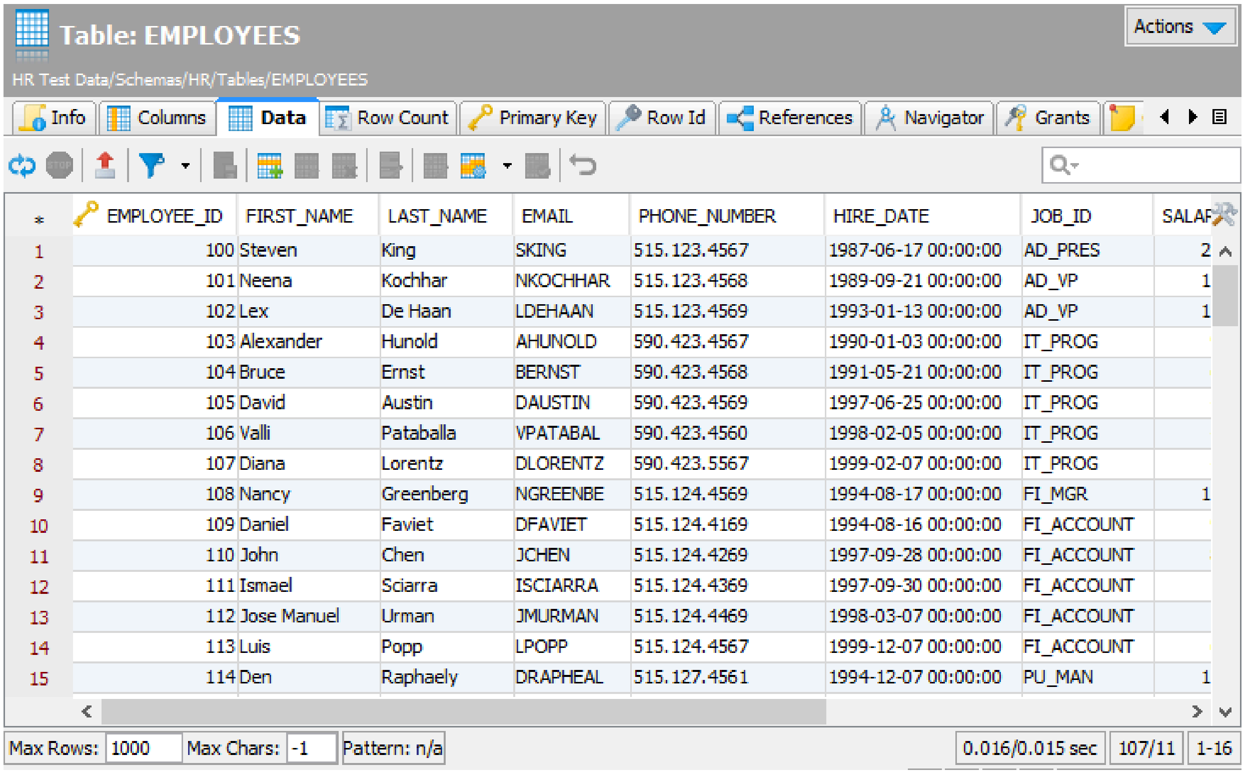
2.9 Editing a Table - basics

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

To edit table data:

1. Expand nodes in the **Databases** tab tree under the connection node until you find the table,
   Double-click on the table node to open its **Object View** tab,
   Open the **Data** sub tab



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALAR
1	100	Steven	King	SKING	515.123.4567	1987-06-17 00:00:00	AD_PRES	2
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21 00:00:00	AD_VP	1
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00	AD_VP	1
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00	IT_PROG	
5	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	IT_PROG	
6	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG	
7	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00	IT_PROG	
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02-07 00:00:00	IT_PROG	
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	1994-08-17 00:00:00	FI_MGR	1
10	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08-16 00:00:00	FI_ACCOUNT	
11	110	John	Chen	JCHEN	515.124.4269	1997-09-28 00:00:00	FI_ACCOUNT	
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	1997-09-30 00:00:00	FI_ACCOUNT	
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	1998-03-07 00:00:00	FI_ACCOUNT	
14	113	Luis	Popp	LPOPP	515.124.4567	1999-12-07 00:00:00	FI_ACCOUNT	
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00:00:00	PU_MAN	1

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.016/0.015 sec 107/11 1-16

2. Edit column values directly in the grid, add and remove rows by clicking on the buttons in the toolbar,
3. Save the edited data by clicking on the **Save** button in the toolbar.

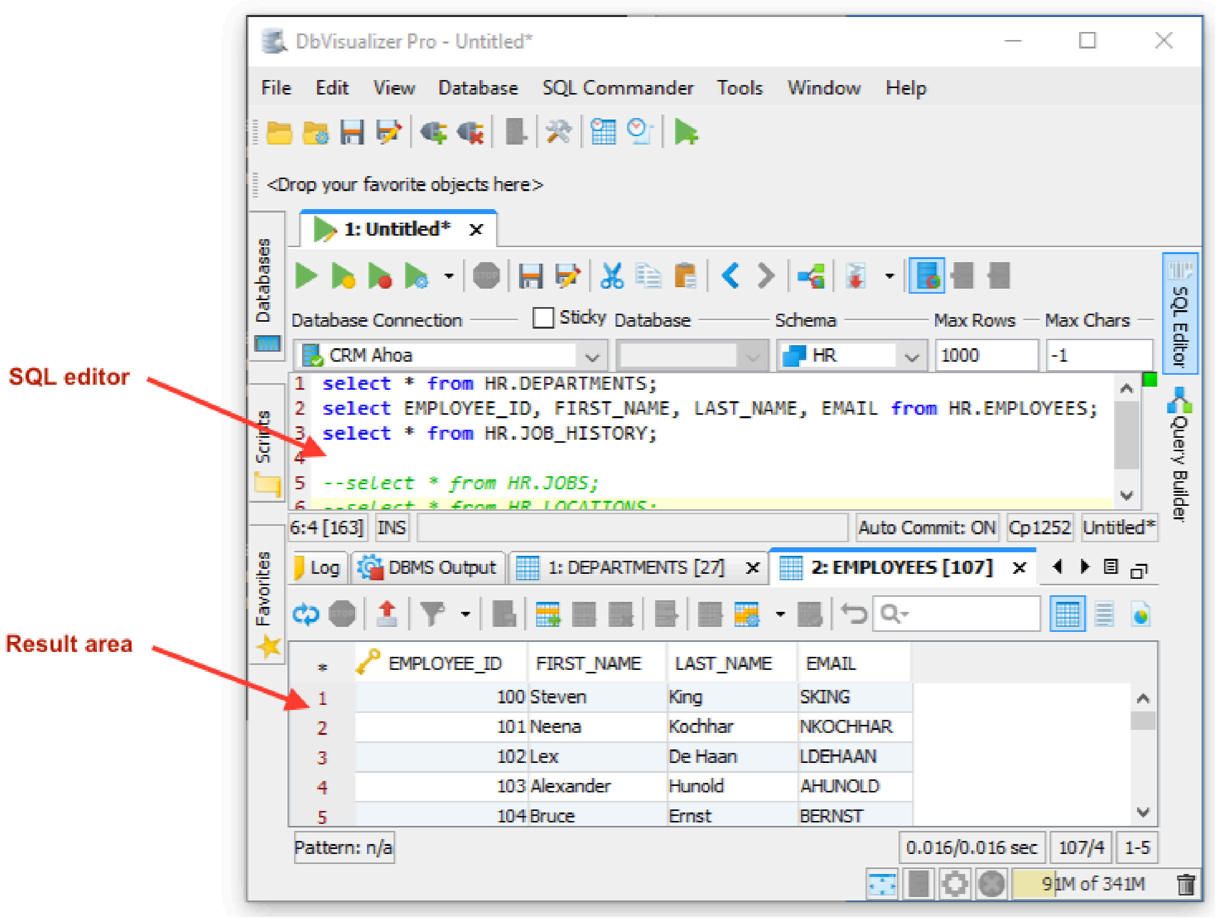
> Note that if the table does not have any declared Primary Key, you will be prompted to select the column(s) that uniquely identify a row.

You can learn more about the editing features in Editing Table Data (see page 94).

2.10 Executing SQL - basics

To execute SQL statements:

1. Open an SQL Commander window from **SQL Commander->New SQL Commander** or by clicking the **New SQL Commander** button in the main toolbar,



Database Connection: CRM Ahoa
☐ Sticky
Database:
Schema: HR
Max Rows: 1000
Max Chars: -1

```
select * from HR.DEPARTMENTS;
select EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL from HR.EMPLOYEES;
select * from HR.JOB_HISTORY;

--select * from HR.JOBS;
--select * from HR.LOCATIONS;
```

6:4 [163] INS — Auto Commit: ON — Cp1252 — Untitled*

Log | DBMS Output | 1: DEPARTMENTS [27] | 2: EMPLOYEES [107]

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
1	100	Steven	King	SKING
2	101	Neena	Kochhar	NKOCHHAR
3	102	Lex	De Haan	LDEHAAN
4	103	Alexander	Hunold	AHUNOLD
5	104	Bruce	Ernst	BERNST

Pattern: n/a — 0.016/0.016 sec — 107/4 — 1-5

91M of 341M

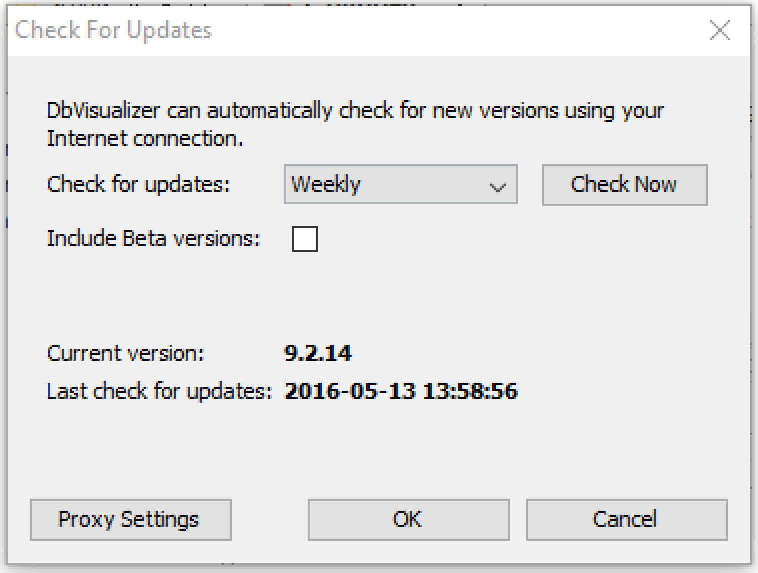
2. Select the database connection, catalog and schema to use,
3. Enter the SQL statements in the editor area,
4. Execute the statements by clicking the **Execute** button in the toolbar or choosing **SQL Commander-> Execute,**
5. The execution log and possible result sets are shown as tabs in the results area below the editor.

You can learn more about editing, saving and executing SQL statements in the Working with SQL (see page 156) section.

2.11 Checking for Updates

By default, DbVisualizer checks for new versions on a weekly basis. To change the interval or manually check for updates:

1. Open **Help->Check for Updates**,



Check For Updates

DbVisualizer can automatically check for new versions using your Internet connection.

Check for updates: Weekly

Include Beta versions: □

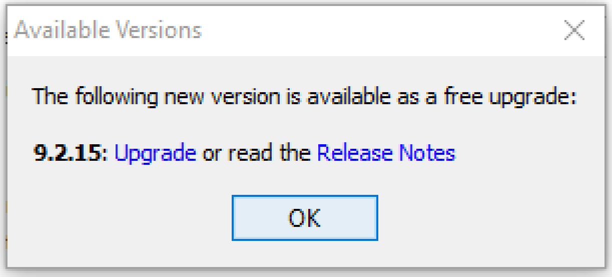
Current version: **9.2.14**

Last check for updates: **2016-05-13 13:58:56**

2. Change the interval to one of **Every Start-Up**, **Daily**, **Weekly**, **Monthly** or **Never**, or click the **Check Now** button to see if there is a new versions available right now.

> ⚠ If you are interested in getting information about **Beta** versions to help us fine tune upcoming versions, check the corresponding checkbox. By default, Beta versions are not considered when checking for updates (unless you are already running a Beta version).

If a newer version is available, a dialog is displayed from where you can install the new version, read release notes, or in case your license is not valid for the new version, open our purchase page in your web browser.

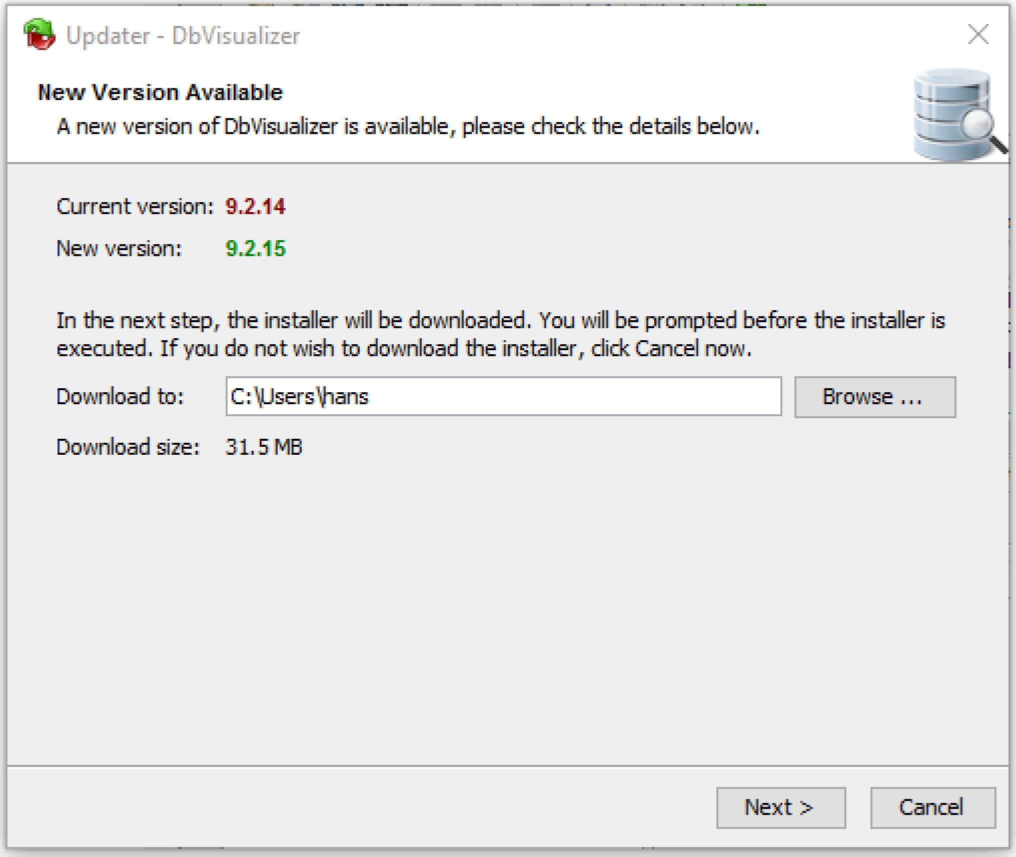


Available Versions

The following new version is available as a free upgrade:

9.2.15: Upgrade or read the Release Notes

Upgrading the currently used version is done by clicking the **Upgrade** link in the above dialog and then follow the instructions.



New Version Available

A new version of DbVisualizer is available, please check the details below.

Current version: 9.2.14

New version: 9.2.15

In the next step, the installer will be downloaded. You will be prompted before the installer is executed. If you do not wish to download the installer, click Cancel now.

Download to: C:\Users\hans

Download size: 31.5 MB

In some organizations, decisions about about software versions and installation is handled as a centralized process and individual users not allowed to upgrade or install software. The Check for Update feature and auto-update of new versions can be disabled by adding the following row in the *DBVIS-HOME/resources/dbvis-custom.prefs*:

dbvis.url.checkforupdate=

With this setting, no entry for Check for Update will appear in the **Help** menu.

2.12 Printing

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 (see page 30)
- Printing a Grid, a Chart and Plain Text (see page 30)
- Printing a Graph (see page 31)
- Print Preview (see page 31)

2.12.1 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.

2.12.2 Printing a Grid, a 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

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.

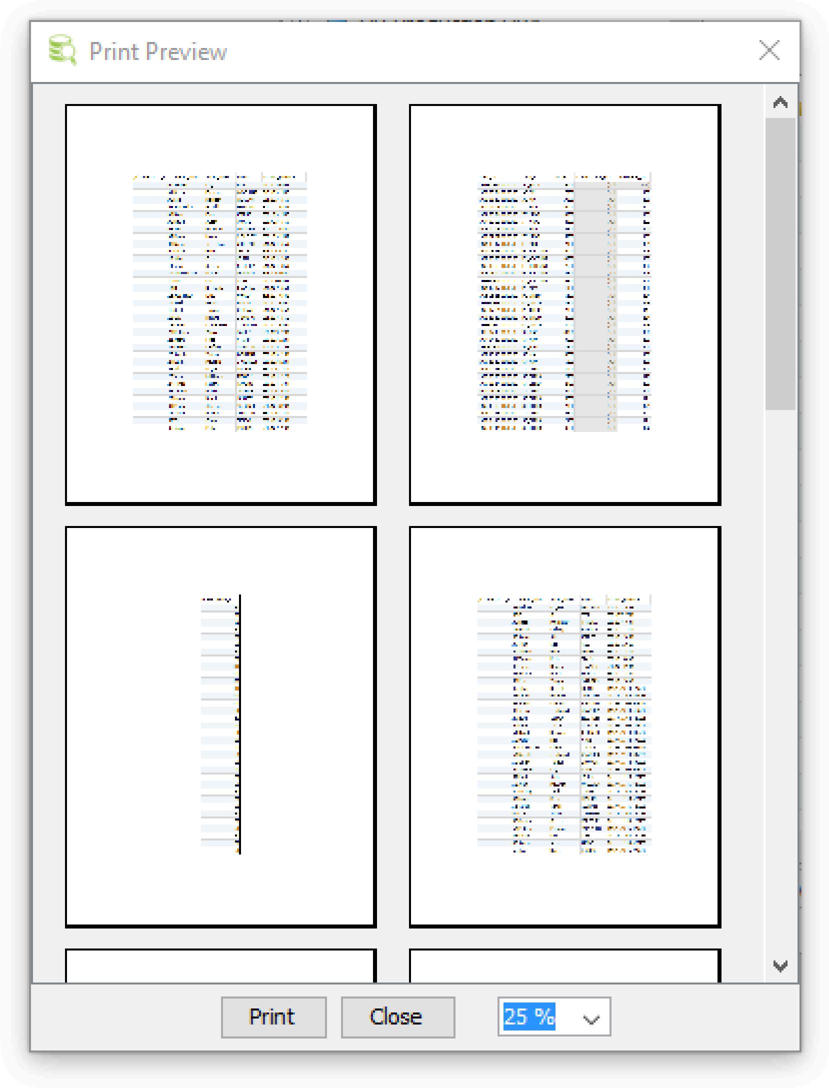
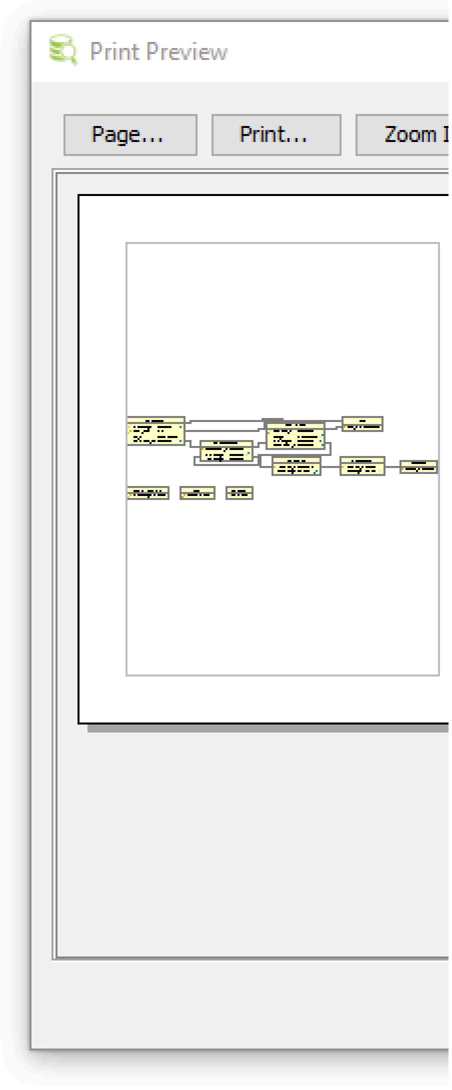
Printing a chart scales the chart to the size of the paper. Plain text is printed as-is and may span multiple pages, both in height and width.

2.12.3 Printing a Graph

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

2.12.4 Print Preview

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

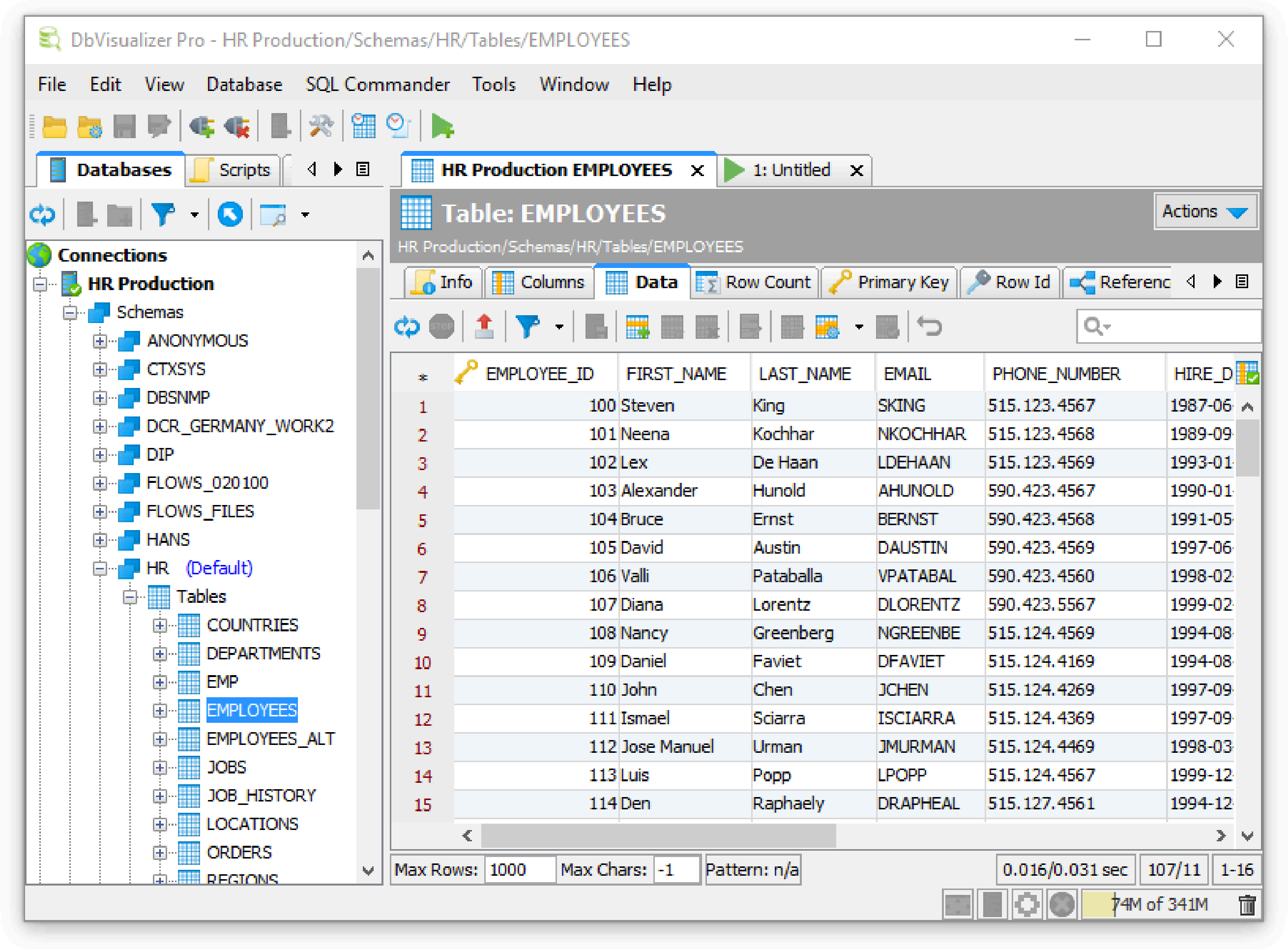
Grid	Graph
	

3 Getting the Most Out of the GUI

DbVisualizer has a tab-based user interface that gives you a lot of control over the layout and how to work with your database objects. This section describes how you can open as many tabs as you need, arrange them to focus on what is important to you, and more.

3.1 Main Window Layout

The DbVisualizer GUI main window contains a navigation area to the left and an area for working with database objects and scripts to the right.

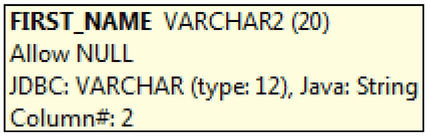


	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_D
1	100	Steven	King	SKING	515.123.4567	1987-06
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01
5	104	Bruce	Ernst	BERNST	590.423.4568	1991-05
6	105	David	Austin	DAUSTIN	590.423.4569	1997-06
7	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	1994-08
10	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08
11	110	John	Chen	JCHEN	515.124.4269	1997-09
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	1997-09
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	1998-03
14	113	Luis	Popp	LPOPP	515.124.4567	1999-12
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.016/0.031 sec 107/11 1-16

At the top of the window, you find the main menus and a toolbar.

Tooltips are used to provide more details about a component throughout the GUI. 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.



FIRST_NAME VARCHAR2 (20)
Allow NULL
JDBC: VARCHAR (type: 12), Java: String
Column#: 2

3.2 Tab Types

There are three main types of tabs in DbVisualizer:

- Navigation Tabs (see page 33)
- Object View Tabs (see page 34)
- SQL Commander Tabs (see page 35)

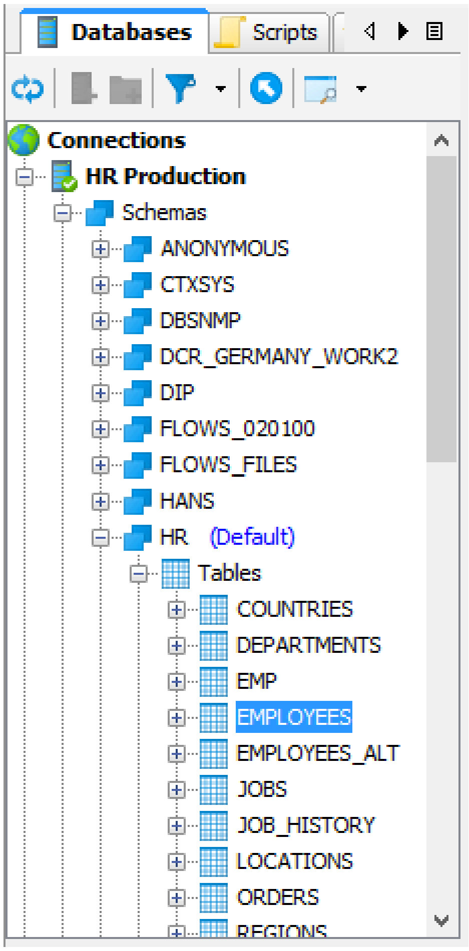
3.2.1 Navigation Tabs

The left part of the DbVisualizer window holds a navigation area with three tabs: **Databases**, **Scripts** and **Favorites**. They all contain an object tree where you can select the objects you want to work with.

The **Databases** tab tree contains you database connections at the top and, when connected, the database objects they contain. If you have many database connections, you can also create folder objects in this tree to organize them.

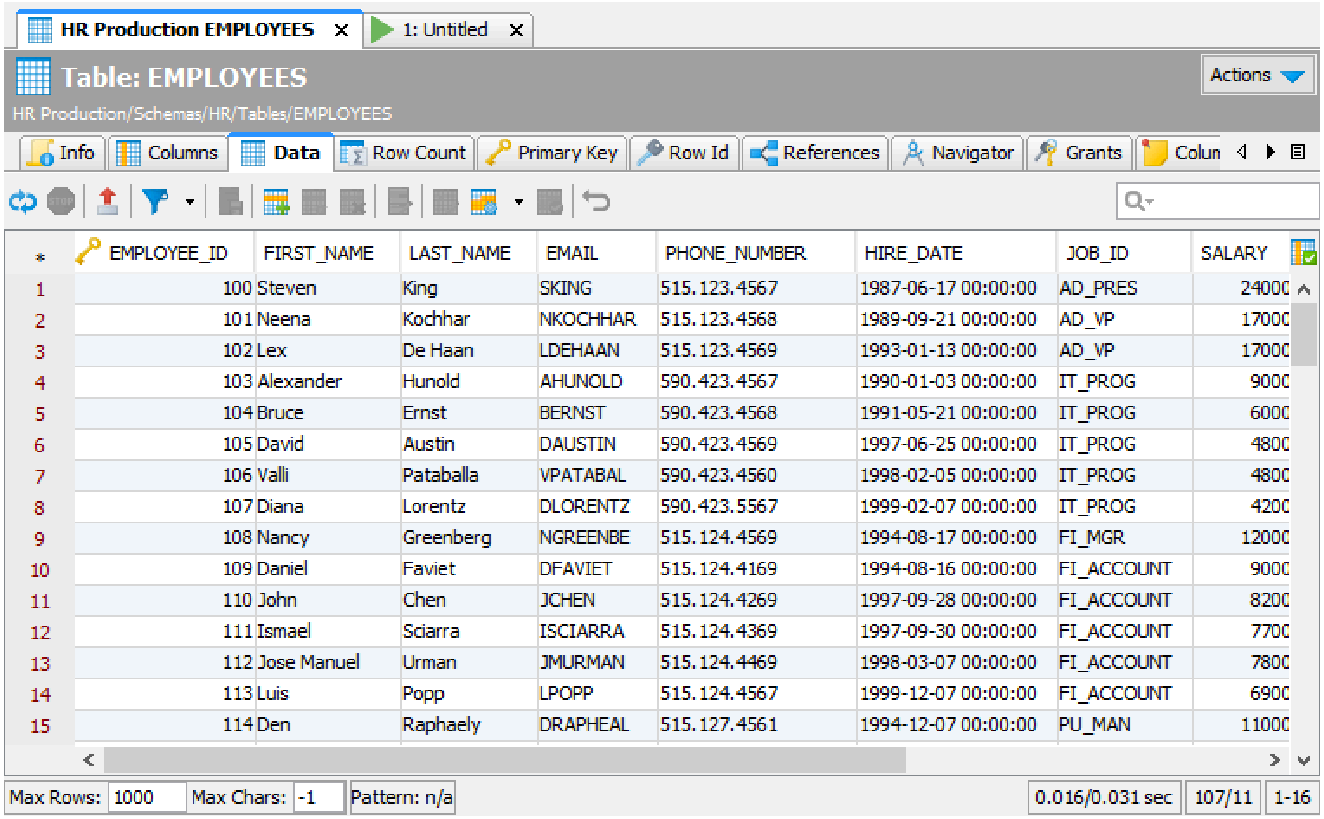
The **Scripts** tab tree contains Bookmarks and Monitors, see the Managing Frequently Used SQL (see page 187) and the Monitoring Data Changes (see page 267) pages for details.

Finally, the **Favorites** tab tree contains objects that you want to have easy access to, either database objects (such as tables, views, or procedures) or scripts. You can read more about Favorites in the Favorites (see page 275) page.



3.2.2 Object View Tabs

An **Object View** tab shows information about a database object, such as the data and DDL for a table, or the source code for a stored procedure. The different types of information are shown as sub tabs within the Object View tab.

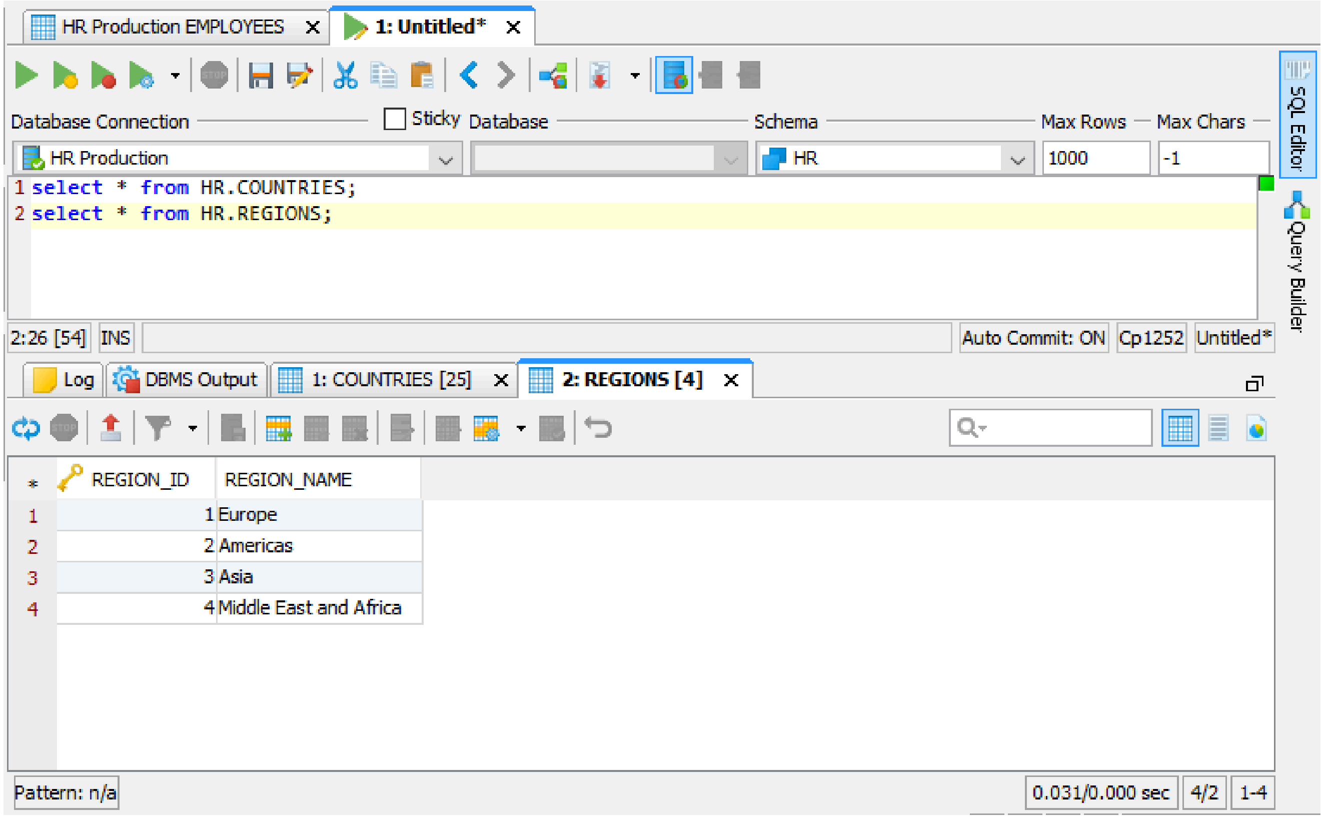


	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	1987-06-17 00:00:00	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21 00:00:00	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00	IT_PROG	9000
5	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	IT_PROG	6000
6	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG	4800
7	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00	IT_PROG	4800
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02-07 00:00:00	IT_PROG	4200
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	1994-08-17 00:00:00	FI_MGR	12000
10	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08-16 00:00:00	FI_ACCOUNT	9000
11	110	John	Chen	JCHEN	515.124.4269	1997-09-28 00:00:00	FI_ACCOUNT	8200
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	1997-09-30 00:00:00	FI_ACCOUNT	7700
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	1998-03-07 00:00:00	FI_ACCOUNT	7800
14	113	Luis	Popp	LPOPP	515.124.4567	1999-12-07 00:00:00	FI_ACCOUNT	6900
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00:00:00	PU_MAN	11000

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.016/0.031 sec 107/11 1-16

3.2.3 SQL Commander Tabs

An SQL Commander tab contains an editor for editing SQL scripts, controls for executing the script and a results area with a **Log** tab and possibly **Result Set** tabs and a **DMBS Output** tab.



Database Connection: HR Production

Schema: HR

Max Rows: 1000

Max Chars: -1

```
select * from HR.COUNTRIES;
select * from HR.REGIONS;
```

2:26 [54] INS Auto Commit: ON Cp1252 Untitled*

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa

Pattern: n/a 0.031/0.000 sec 4/2 1-4

3.3 Opening a Tab

3.3.1 Database tree objects

You can open an object by double-clicking on the object node, or by pressing **Enter** with the node selected, in all navigation tab trees. This opens either an Object View tab or an SQL Commander tab, depending on the object type: database object or script.

By default, a database object is opened in an Object View tab that is "available," meaning it is not pinned, busy running a task, or contains pending edits. The current tab is choosen if it is available, otherwise any other available tab is used. If none of the tabs is available, a new tab is created for the object. You can change this behavior in the General/Tabs category in Tools->Tool Properties, so that a new tab is always used instead of using an available tab.

If a tab is already open for the object, it is made the active tab. An alternative to double-clicking a database object is to use the **Open in Tab** choice from the node's right-click menu. **Open in Tab** is also available in the **Open Object** drop-down menu button in the **Databases** tab toolbar.

If you want to open a database object in a new tab instead of an available tab, hold down the **Alt** key when you double-click the node, use the **Open in New Tab** choice from the node's right-click menu or from the **Open**

Object drop-down menu button in the **Databases** tab toolbar. The drop-down button keeps the last choice as the default, so once you have choosed **Open in New Tab** once, you only need to click the button to do the same.

Both **Open in Tab** and **Open in New Tab** can also be used when multiple nodes are selected in the tree to open multiple tabs in one go.

To replace the content in a specific **Object View** tab with information for another object, drag the node for the new object from the object tree and drop it on the **Object View** tab header.

The behaviour when double clicking a Database object can be configured in **Tool Properties** -> **General** -> **Database Objects Tree**.

3.3.2 Scripts and Monitors

Scripts (Bookmarks and Monitors) are always opened in a new **SQL Commander** tab unless the script is already opened in a tab. If so, that tab is activated instead. An alternative to double-clicking the node is to use **Open in SQL Commander** in the right-click menu or the corresponding button in the **Scripts** tab toolbar. The righ-click menu also holds an **Open in SQL Commander and Execute** choice.

A new, empty **SQL Commander** tab is created by clicking the **Create SQL Commander** button in the main toolbar or using the corresponding **File** menu item. You can use this kind of **SQL Commander** tab for ad-hoc statement execution or to create a new script.

SQL Commander tabs can also be opened for files in the file system. Click the **Open File** button in the main toolbar or choose **File->Open** to open the file chooser window and select one of more files. Alternatively, you can drag files from your platform's file browser and drop them in the main toolbar area.

In addition see also Synchronizing object tab selection and selection in the tree (see page 313) which shows how to set up DbVisualizer to automatically select the database tree node when the corresponding object view tab is selected and vice versa.

> **Only in DbVisualizer Pro**
>
> Multiple SQL Commander tabs are only available in the DbVisualizer Pro edition.

3.4 Pinning a Tab

To prevent an **Object View** tab to be reused for another object or to prevent any tab to be removed unless you explicitly asks for it, you can "pin" it. In the right-click menu for a tab you find a **Pin Tab** toggle to accomplish this .

You can also click on the icon in the tab header as a shortcut for toggling between the "pinned" and "unpinned" states.

3.5 Closing a Tab

A top level tab and a Result Set tab in an **SQL Commander** tab can be closed by clicking the cross to the right of the tab label, or clicking the tab header with the middle mouse button.

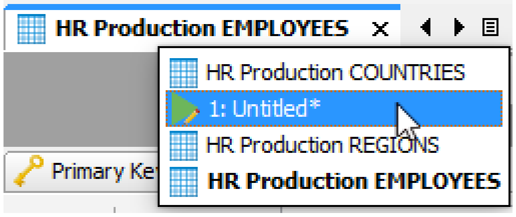
If you want to close a number of tabs at the same time, you can use the tab header menu choices:

Close Tab	Close just the current tab.
Close Other Tabs	Close all tabs except the current tab.
Close Closeable Tabs	Close all tabs that are in a state where they can be closed with no action required by the user, e.g. not pinned and no pending edits.
Close All Pinned Tabs	Close all pinned tabs.
Close All Tabs	Close all tabs, regardless of state

You can also **hide** Object View sub tabs that you are not interested in. Use the **Close Tab** right-click menu choice to do so. To see the hidden tabs again, use **Restore Hidden Tabs** in the right-click menu.

3.6 Listing Open Tabs

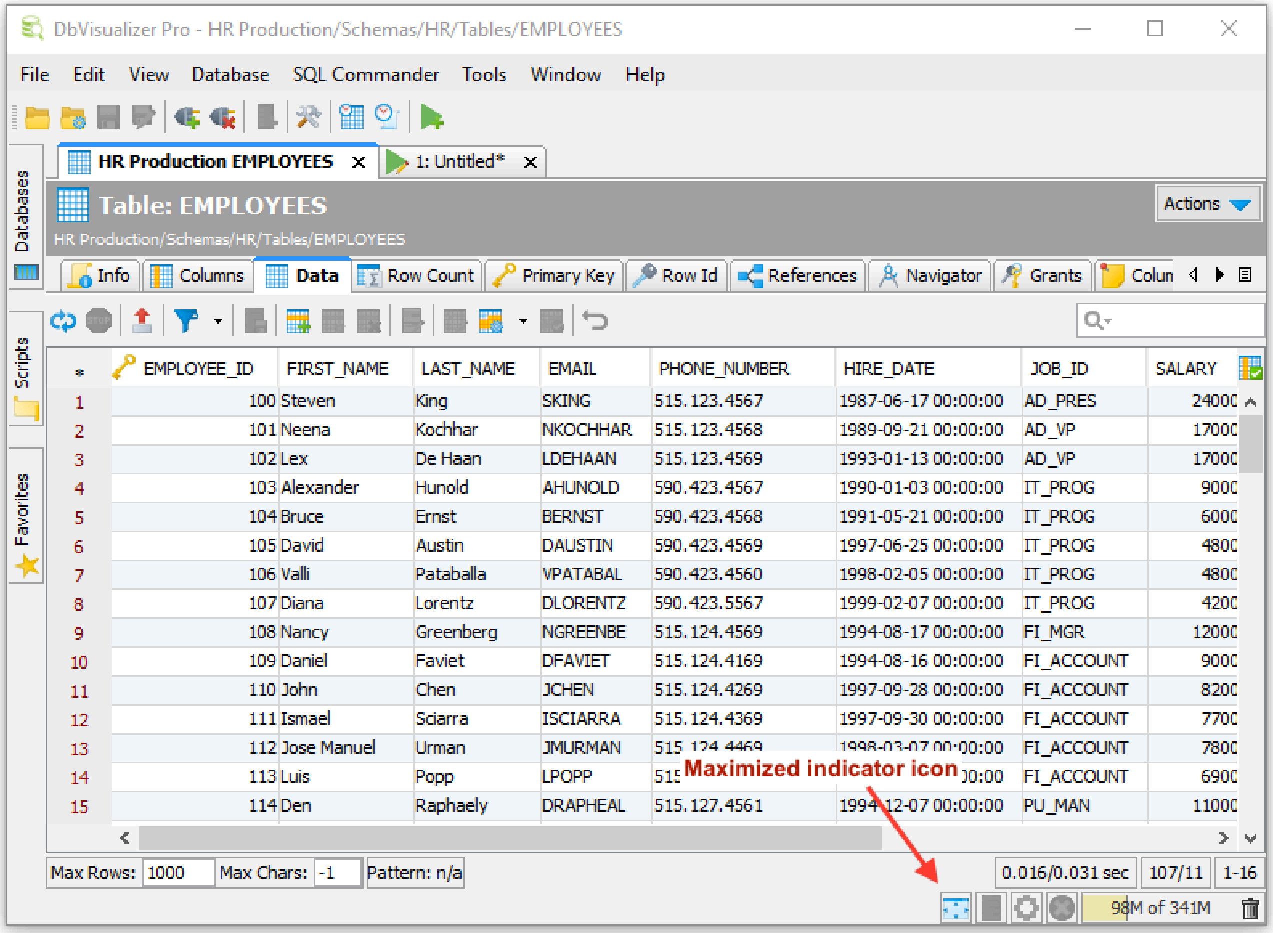
If you have many tabs opened, there may not be room enough to show them all at the same time. In this case, you can scroll through them using the arrow buttons that appear to the right of the tabs. It is, however, often faster to locate the tab you want to work with by clicking the list icon next to the arrow buttons. This brings up a list of all open tabs so you can select the one you want directly.



3.7 Maximizing and Minimizing a Tab

The three Navigation tabs can be minimized either by double-clicking on the tab header or using **Minimize Tab** in the tab's right-click menu. Clicking on a minimized tab brings it back to its regular place and size again.

All other tabs can be maximized by double-clicking on the tab header or using **Maximize Tab** in the tab's right-click menu. When you maximize a tab, the Navigation tabs are minimized to make as much room as possible available and the tab you maximized fills all available space. Double-clicking on the the tab again restores all tabs back to their original size.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	1987-06-17 00:00:00	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21 00:00:00	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00	IT_PROG	9000
5	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	IT_PROG	6000
6	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG	4800
7	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00	IT_PROG	4800
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02-07 00:00:00	IT_PROG	4200
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	1994-08-17 00:00:00	FI_MGR	12000
10	109	Daniel	Faviet	DFAVIET	515.124.4169	1994-08-16 00:00:00	FI_ACCOUNT	9000
11	110	John	Chen	JCHEN	515.124.4269	1997-09-28 00:00:00	FI_ACCOUNT	8200
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	1997-09-30 00:00:00	FI_ACCOUNT	7700
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	1998-03-07 00:00:00	FI_ACCOUNT	7800
14	113	Luis	Popp	LPOPP	515	00:00	FI_ACCOUNT	6900
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00:00:00	PU_MAN	11000

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.016/0.031 sec 107/11 1-16

98M of 341M

An icon in the main status bar indicates when a tab is maximized. An alternative for restoring the original size of all tabs is to double-click on this icon.

3.8 Floating a Tab

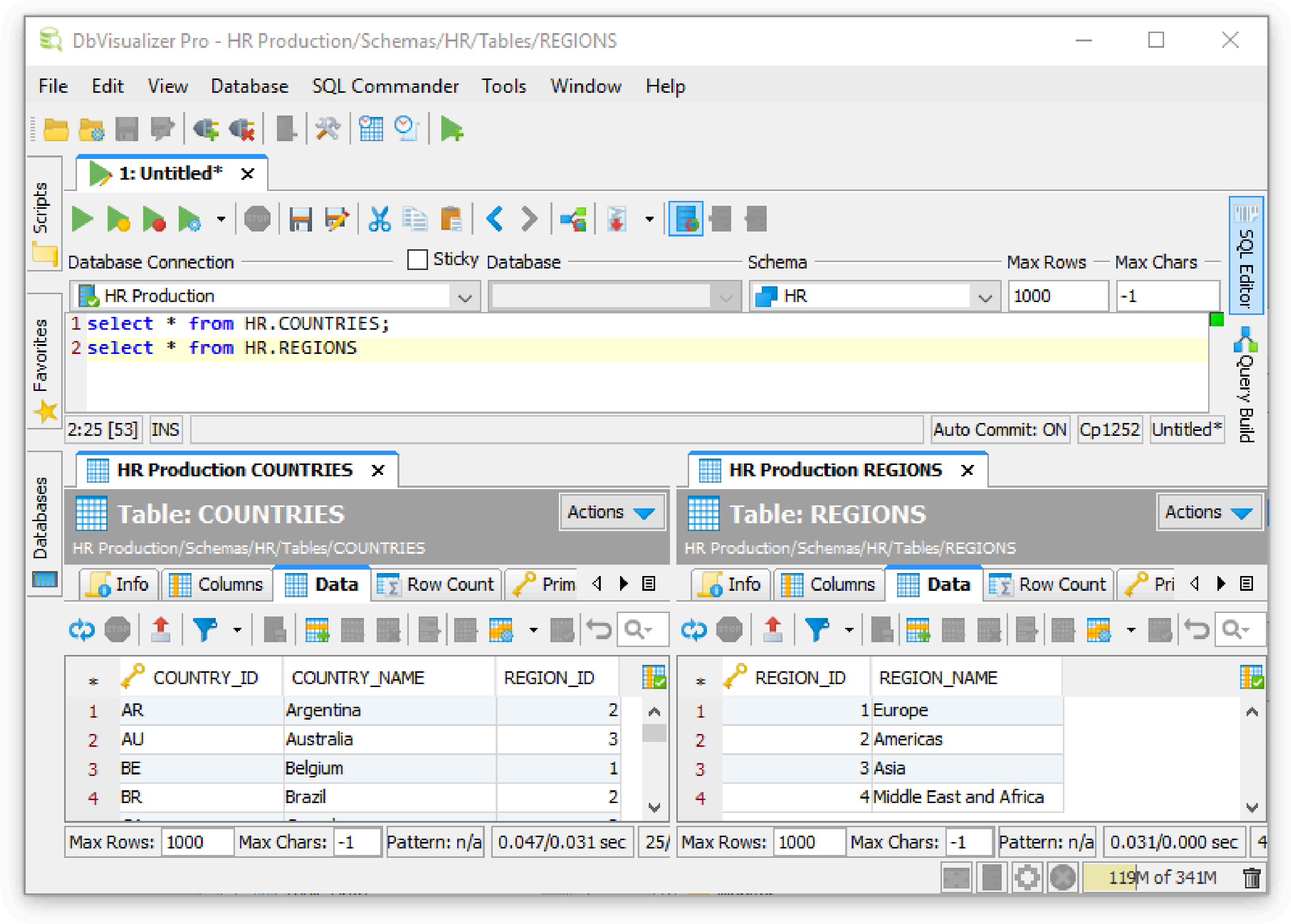
Sometimes it is handy to break up the user interface in multiple freestanding windows. Every tab in DbVisualizer can be placed in a separate window by "floating" the tab. Use the **Floating** menu choice in the tab header right-click menu to float the tab in a separate window.

When you close a floating tab, a dialog is displayed where you can choose to really close the tab or restore it back into the main window.

3.9 Rearranging Tabs

You can move the tabs around by drag and drop. This allows you to see the content of multiple tabs at the same time

When you drag a tab, and outline of the tab borders shows what will happen when you drop it: place it in above, below or next to another tab, or simply move it to another location among its siblings.



Database Connection: HR Production
Database:
Schema: HR
Max Rows: 1000
Max Chars: -1

```
select * from HR.COUNTRIES;
select * from HR.REGIONS
```

2:25 [53] INS Auto Commit: ON Cp1252 Untitled*

Table: COUNTRIES
HR Production/Schemas/HR/Tables/COUNTRIES

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AR	Argentina	2
2	AU	Australia	3
3	BE	Belgium	1
4	BR	Brazil	2

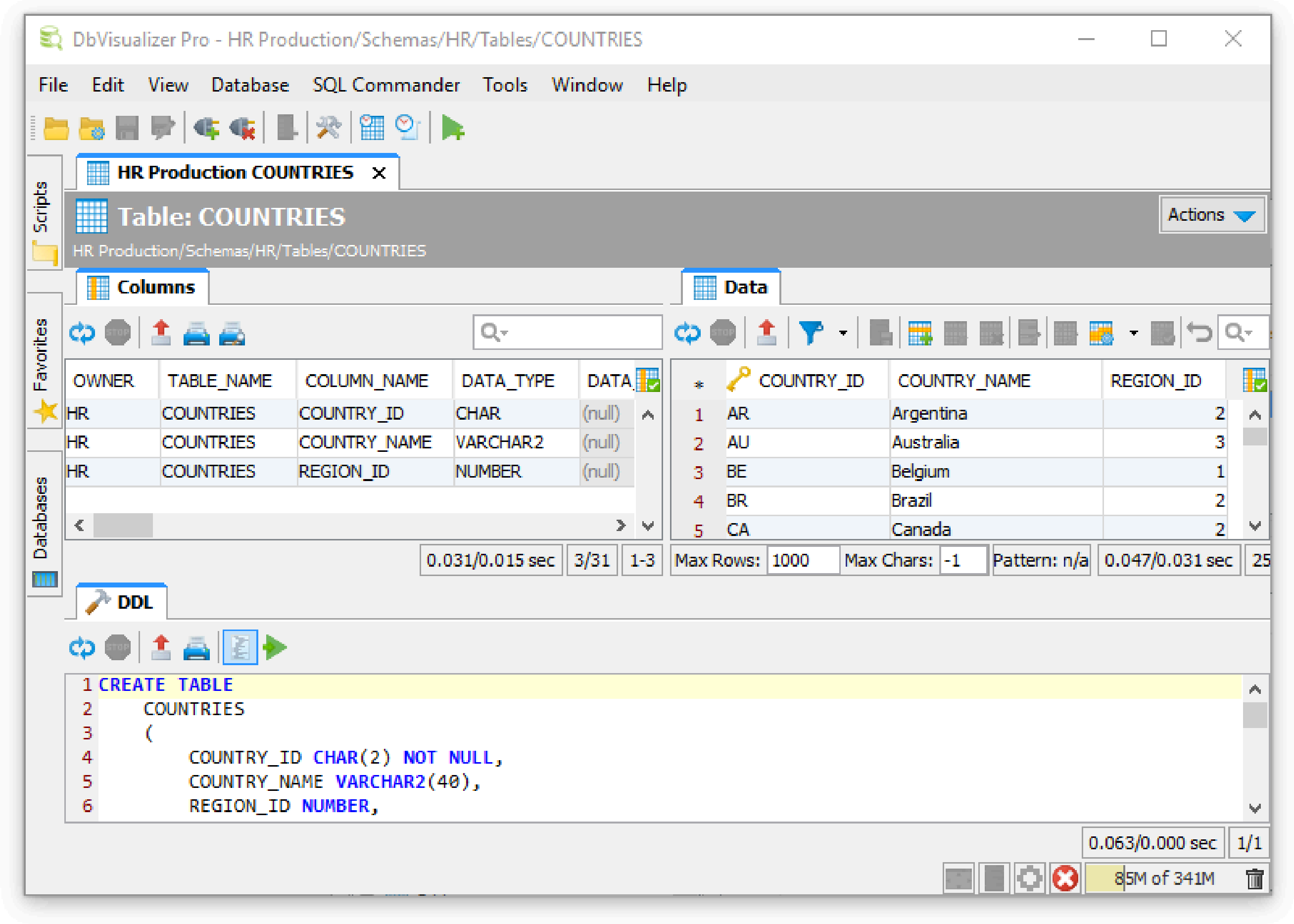
Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.047/0.031 sec 25/

Table: REGIONS
HR Production/Schemas/HR/Tables/REGIONS

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.031/0.000 sec 4

The tab header right-click menu also has a couple of choices for arranging all tabs at the same level as either " tiled" (the content of all tabs visible side-by-side) or "collapsed" (only the content of the active tab visible).



OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA
HR	COUNTRIES	COUNTRY_ID	CHAR	(null)
HR	COUNTRIES	COUNTRY_NAME	VARCHAR2	(null)
HR	COUNTRIES	REGION_ID	NUMBER	(null)

0.031/0.015 sec | 3/31 | 1-3

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AR	Argentina	2
2	AU	Australia	3
3	BE	Belgium	1
4	BR	Brazil	2
5	CA	Canada	2

Max Rows: 1000 | Max Chars: -1 | Pattern: n/a | 0.047/0.031 sec | 25

```
CREATE TABLE
    COUNTRIES
    (
        COUNTRY_ID CHAR(2) NOT NULL,
        COUNTRY_NAME VARCHAR2(40),
        REGION_ID NUMBER,
```

0.063/0.000 sec | 1/1

85M of 341M

You can save your rearranged layout of an Object View tab so that it is applied for all objects of the the same type for the same database type. In the sub tab header right-click menu, just select **Save as Default Layout**. To restore the default layout, use **Reset to Factory Layout**.

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

3.10 Changing the Tab Label

The tab labels are set based on a pattern that you can change in **Tools->Tool Properties**, in the **General/Tabs** category.

You can select one of the predefined patterns or create your own by editing the pattern. The variables available for these patterns are:

Variable	Available For	Description
${ connectionname }	All	Connection name
${index}	All	A unique index for the tab in the tab group
${userid}	All	Userid used for the connection
${filename}	SQL Commander Tabs	Script filename, or "Untitled" if no file is loaded. The **nofile** option can be used to change "Untitled" to something else, e.g. **${filename\|nofile=No File}**. Set it to blank to only show a label when a file is loaded, i.e. **${ filename|nofile=}**
${longfilename}	SQL Commander Tabs	Absolute path for the script, or "Untitled" if no file is loaded. The **nofile** option can be used to show something else, see the **${filename}** variable for details
${objectname}	Object View Tabs	Object name
${objecttype}	Object View Tabs	Object type, e.g. Table, View etc.
${catalog}	Object View Tabs	The database (if any) for the currently open object
${schema}	Object View Tabs	The schema (if any) for the currently open object
${editorindex}	Result Set Tabs	The index for the parent SQL Commander tab
${rows}	Result Set Tabs	Number of rows in the result set
${sql}	Result Set Tabs	Part of the SQL statement that produced the result set
${table}	Result Set Tabs	Name of the table (first if more than one) the result set comes from
${time}	Result Set Tabs	Time when the result set was produced

Variable	Available For	Description
${vendor}	Result Set Tabs	Database vendor name

You can also manually change the label for Object View, SQL Commander and Result Set tabs, using the **Rename** menu choice in the tab right-click menu.

3.11 Selecting a Node for a Tab

To quickly navigate to and select the node in the **Databases** tab tree that an **Object View** tab belongs to, you can click on the object path in the Object View header area. Alternatively, you can use **Select in Databases Tab** in the tab right-click menu.

Similarly, you can navigate to the object represented by a Favorite object using its **Select Target Object** right-click menu choice. It selects the object in the **Databases** or **Scripts** tab, depending on the Favorite type.

For an **SQL Commander** tab that is associated with a connection, you can use **Select in Databases Tab** in the tab right-click menu to select the connection node.

It is also possible to automatically select the Node whenever its object view tab is clicked. This is done by selecting the **Autoscroll FROM Object View tab** in the **Open selected object(s)** drop down in the toolbar of the **Databases** tab tree.

3.12 Preserving Tabs Between Sessions

If you often work with the same objects and a few scripts, you can ensure that the Object View and SQL Commander tabs for these objects remain open between DbVisualizer sessions.

1. Open **Tools->Tool Properties**,
2. Select the **Tabs** category,
3. Enable one or both of **Preserve SQL Commander tabs between Sessions** and **Preserve Object View tabs between Sessions**,
4. Click **Apply** or **OK** to apply the new settings.

This feature is enabled by default for SQL Commander tabs but not for Object View tabs.

The content of the SQL Commander tabs is saved at regular intervals so when you restart DbVisualizer, the content is the same as where you left off.

For Object View tabs, you can also enable **Preserve Object View tabs at Disconnect**. By default, Object View tabs for objects that belong to a connection are closed when it is disconnected.

3.13 Using Tab Colors and Borders

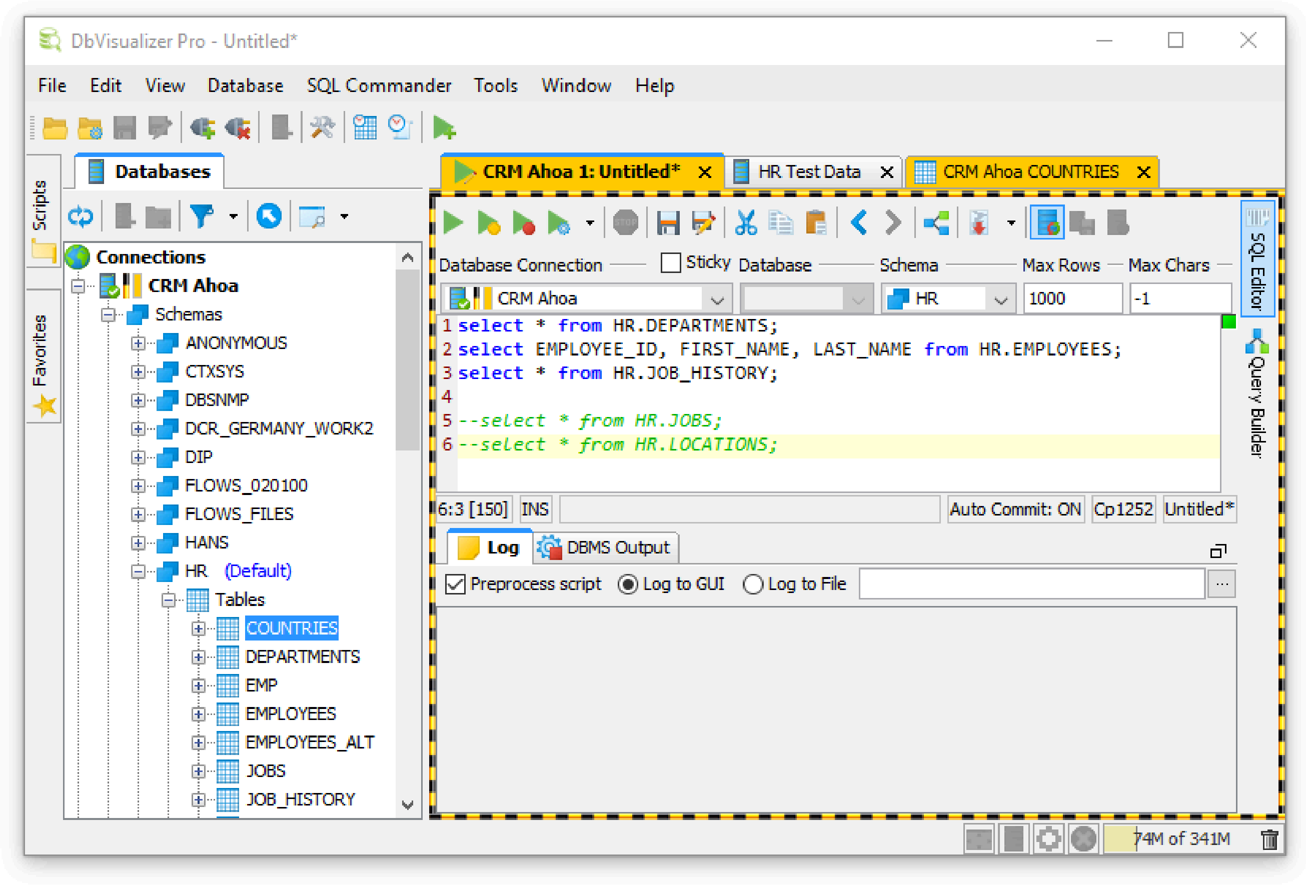
Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

If you like to distinguish tabs and other GUI components for a specific connection from those of others, you can specify a tab background color and/or border to use for a connection. If you set these in the **Tool Properties** window, they apply to all connections with the corresponding database type. Therefore these properties are typically set in the **Properties** tab for a specific connection instead.

For the border, you can either select one of the predefined styles or specify a small image file to use for the border.

The selected color and border are also shown in the connection tab node in the Databases tab and in the Database Connection list in the SQL Commander.



Database Connection: CRM Ahoa
□ Sticky
Database:
Schema: HR
Max Rows: 1000
Max Chars: -1

```
select * from HR.DEPARTMENTS;
select EMPLOYEE_ID, FIRST_NAME, LAST_NAME from HR.EMPLOYEES;
select * from HR.JOB_HISTORY;

--select * from HR.JOBS;
--select * from HR.LOCATIONS;
```

6:3 [150] INS Auto Commit: ON Cp1252 Untitled*

☑ Preprocess script ● Log to GUI ○ Log to File

3.14 Changing the GUI Appearance

To change how the DbVisualizer GUI is displayed:

1. Open **Tools->Tool Properties**,
2. Select the **Appearance** category under the **General** tab.

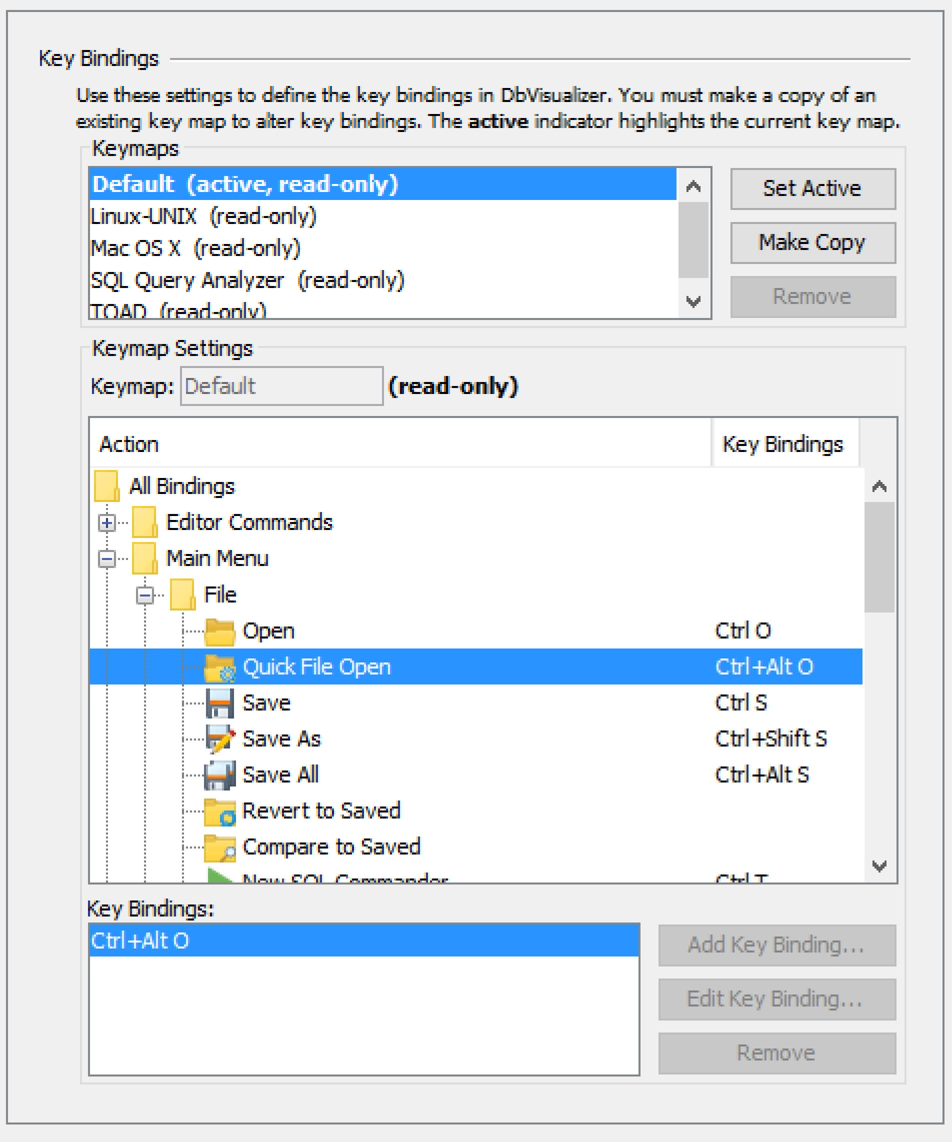
Here you can select a different Look and Feel and adjust the size and use of icons. You can also enable **Ask at Startup** to be prompted for which Look and Feel to use for each session, which may be useful if you often switch between a dark and a light Look and Feel.

In the subcategories **Fonts, Editor Styles** and **Colors** you can adjust the fonts used for different parts of the GUI and the colors used to highlight grid elements. Color choices are saved separately for dark and light Look and Feels.

The main **View** menu also contains a number of toggle controls for showing or hiding GUI elements, such as toolbars and status bars.

3.15 Changing Keyboard Shortcuts

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.



Key Bindings

Use these settings to define the key bindings in DbVisualizer. You must make a copy of an existing key map to alter key bindings. The **active** indicator highlights the current key map.

Keymaps

- **Default (active, read-only)**
- Linux-UNIX (read-only)
- Mac OS X (read-only)
- SQL Query Analyzer (read-only)
- TOAD (read-only)

Keymap Settings

Keymap: Default **(read-only)**

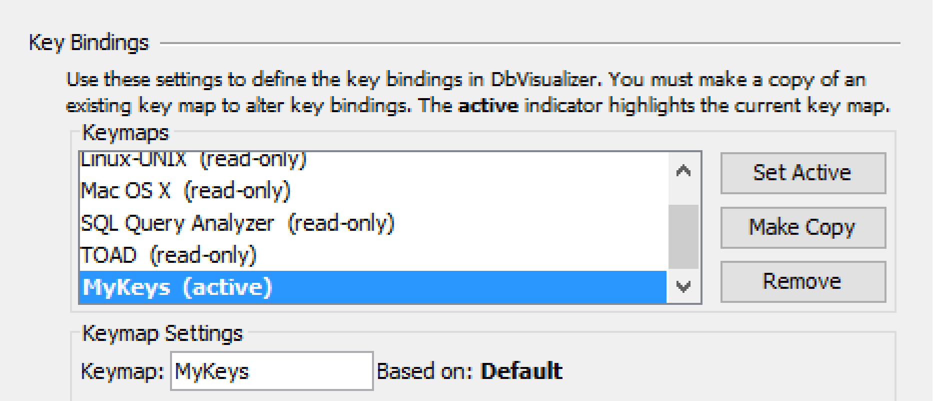
Action	Key Bindings
All Bindings	
Editor Commands	
Main Menu	
File	
Open	Ctrl O
Quick File Open	Ctrl+Alt O
Save	Ctrl S
Save As	Ctrl+Shift S
Save All	Ctrl+Alt S
Revert to Saved	
Compare to Saved	
New SQL Commander	Ctrl T

Key Bindings:

Ctrl+Alt O

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 Bindings

Use these settings to define the key bindings in DbVisualizer. You must make a copy of an existing key map to alter key bindings. The **active** indicator highlights the current key map.

Keymaps

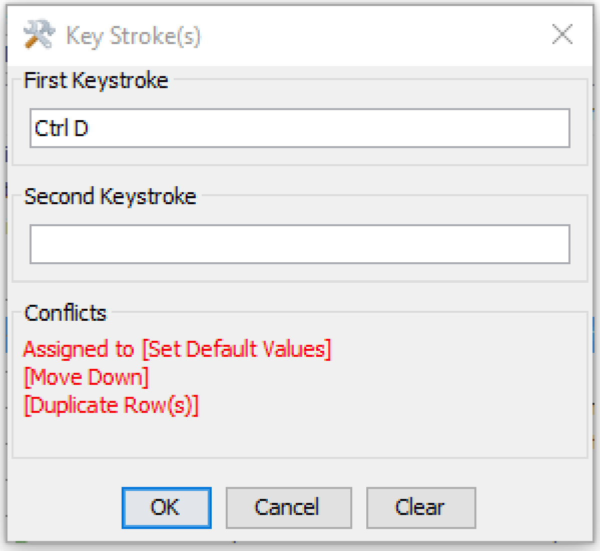
Linux-UNIX (read-only)
Mac OS X (read-only)
SQL Query Analyzer (read-only)
TOAD (read-only)
MyKeys (active)

Keymap Settings

Keymap: MyKeys Based on: **Default**

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 contains subfolders, 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.



Key Stroke(s)

First Keystroke: Ctrl D

Second Keystroke:

Conflicts

Assigned to [Set Default Values]
[Move Down]
[Duplicate Row(s)]

The keystroke dialog controls whether a key binding is already assigned somewhere else. If there is a conflict with another binding, the **Conflicts** area 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 show the first defined key binding in the list.

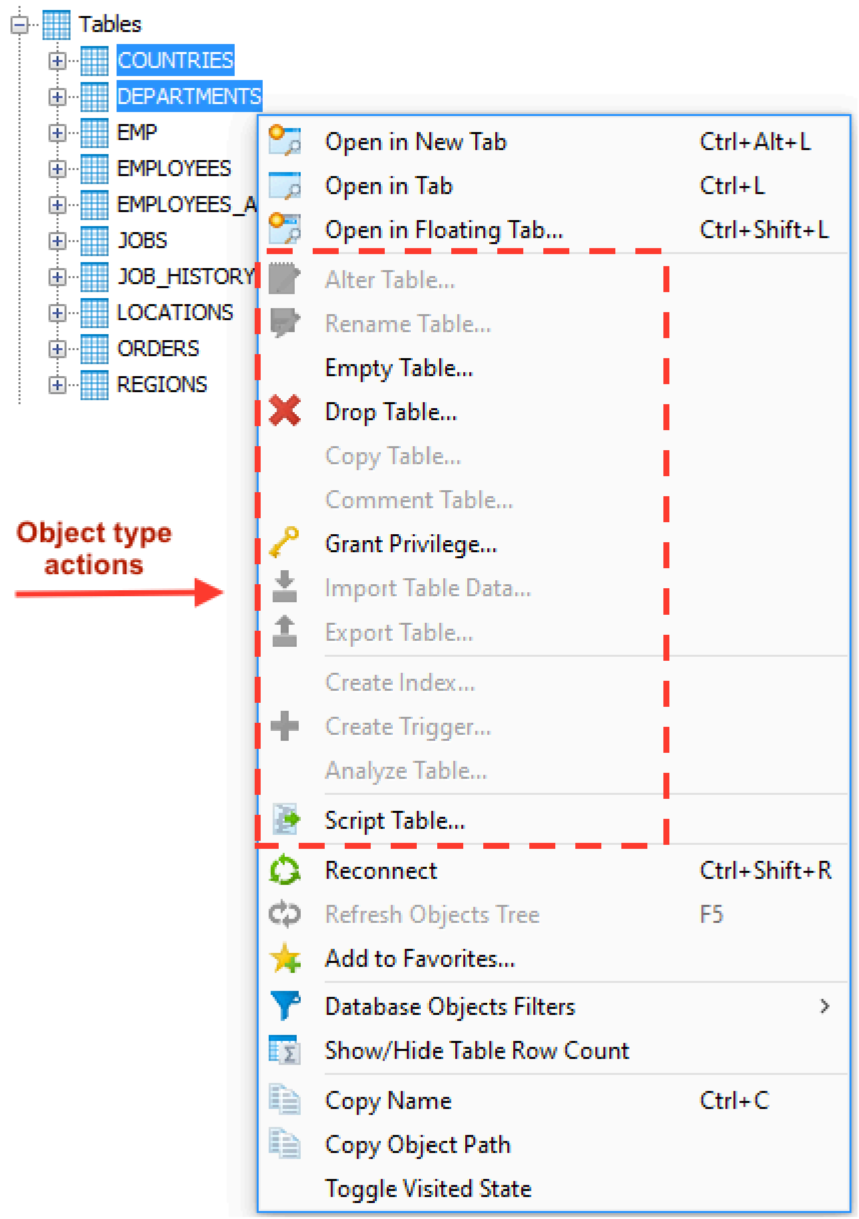
4 Managing Database Objects

4.1 Opening a Database Object

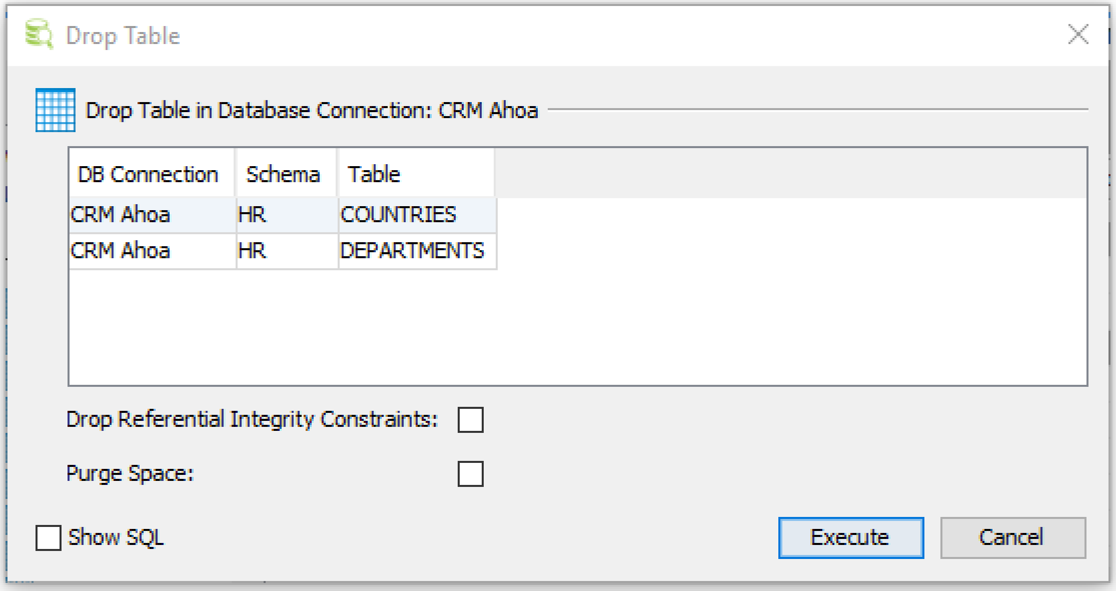
To open an Object View tab with the details for a database connection or for a specific database object, the simplest way is to double-click the object node in the Databases tab. For more information about opening tabs, check the Opening a Tab (see page 36) section.

4.2 Perform Actions on Multiple Database Objects

The actions menu for a database object in the Databases tab consists of actions that are available for the selected object type. Some of the operations are common for any currently selected object while others are only available for the current object. Sometimes it is convenient to run a single action on multiple nodes at once. To do this, select the nodes (make sure all are of the same object type) and then right-click in the object list. The menu will now highlight those actions that are valid to use.



Choosing for example the **Drop Table** action shows the following window in which the drop can then be performed on multiple tables.



DB Connection	Schema	Table
CRM Ahoa	HR	COUNTRIES
CRM Ahoa	HR	DEPARTMENTS

Drop Referential Integrity Constraints: □

Purge Space: □

4.3 Filtering Database Objects

inlinefiltering.png

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

Managing what database objects are listed in the **Databases** tab is done at two levels:

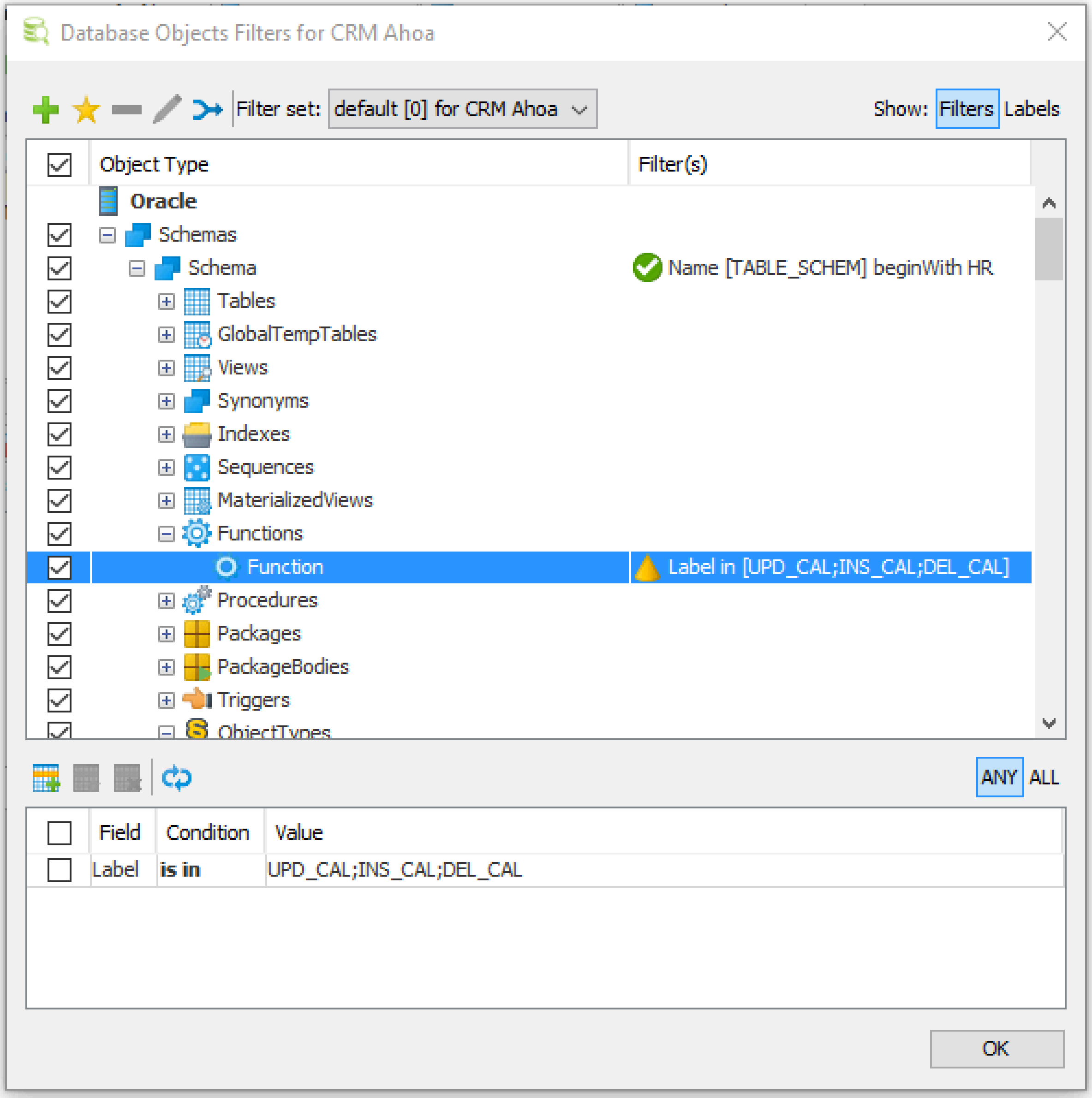
1. **Objects Filtering**
   Defines what individual object nodes are listed for a specific object type. This allows filtering on for example Table objects so that only tables matching a condition are listed.

2. **Object Type Visibility**
   Defines what object types such as views, tables, indexes, procedures, etc. are listed. Every database in DbVisualizer supports all sorts of database objects. Having the ability to hide some object types makes it easier to locate the database objects that are of primary interest.

The **Filter Editor** is used to manage both object type visibility and object filters. The Filter Editor is opened from one of:

- **Database** main menu and the **Database Objects Filters->Open Filter Editor**
- Right-click in the **Databases tab** and chose **Database Objects Filters->Open Filter Editor**
- Click the drop-down button on the funnel icon in the **Databases tab** toolbar and chose **Open Filter Editor**.

ℹ Managing filters and opening the filter editor requires that a node is selected in the Databases tab tree and that the related database connection is connected.



Database Objects Filters for CRM Ahoa

Filter set: default [0] for CRM Ahoa

Show: Filters Labels

☑	Object Type	Filter(s)
	Oracle	
☑	⊟ Schemas	
☑	⊟ Schema	✓ Name [TABLE_SCHEM] beginWith HR
☑	⊞ Tables	
☑	⊞ GlobalTempTables	
☑	⊞ Views	
☑	⊞ Synonyms	
☑	⊞ Indexes	
☑	⊞ Sequences	
☑	⊞ MaterializedViews	
☑	⊟ Functions	
☑	Function	▲ Label in [UPD_CAL;INS_CAL;DEL_CAL]
☑	⊞ Procedures	
☑	⊞ Packages	
☑	⊞ PackageBodies	
☑	⊞ Triggers	
☑	⊟ ObjectTypes	

ANY ALL

□	Field	Condition	Value
□	Label	**is in**	UPD_CAL;INS_CAL;DEL_CAL

OK

The upper list shows all available object types for the actual database connection. The leading **check mark** controls whether the object type should be visible in the Databases tab. Read more in the Object Type Visibility (see page) section. The **Filter(s)** column shows any filters defined for each object type. A **green symbol** indicate that at least one filter is active while a **yellow symbol** indicate that filters are defined but all are deactivated.

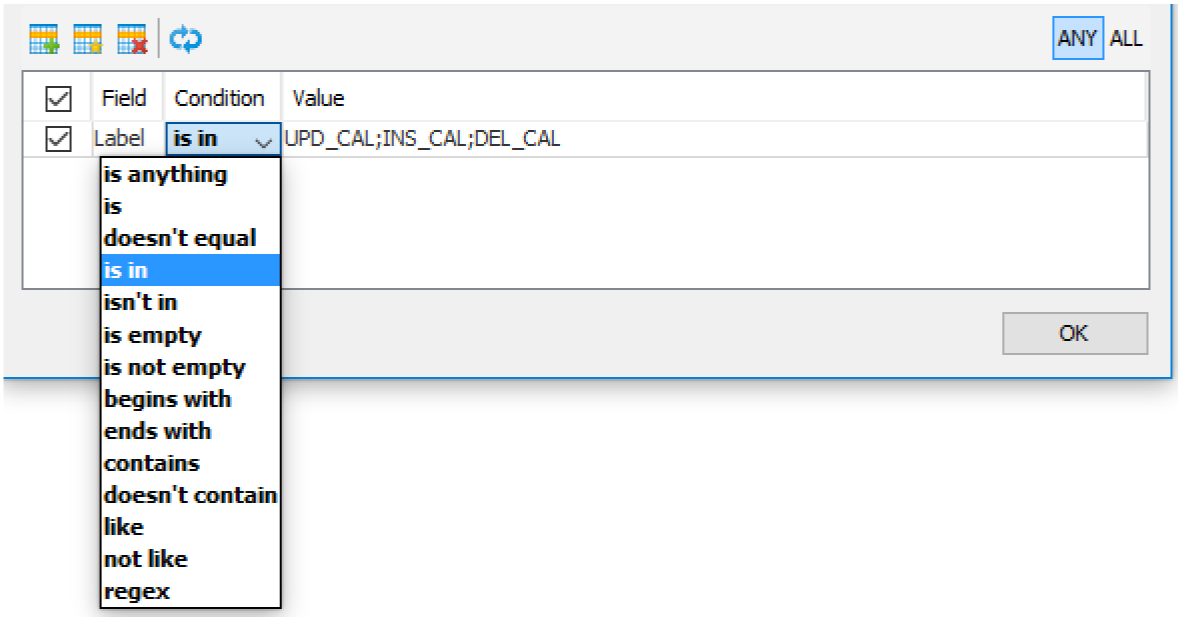
Selecting an object type enables the objects filtering area in the lower part of the window. Here individual filters are defined for an object type.

4.3.1 Object Filtering

Object filtering can be made on any database object (**Table, Function, Procedure, User**) except for grouping objects such as **Tables, Functions, Procedures, Users**. Grouping objects in DbVisualizer are often labeled with the related object type name in plural.

> Object Filters for Database and Schema objects also apply to the corresponding drop-down lists in SQL Commander tabs, but only after a reconnect.

To setup a filter, select the object type in the objects list and in the filter area, click the button with a plus sign to insert a new row.



☑	Field	Condition	Value
☑	Label	is in	UPD_CAL;INS_CAL;DEL_CAL

A filter entry consists of a field (e.g **Label),** a condition and a value to match against. Click the **condition** field to select the condition to use. In the **Value** field, enter the value that should be matched. For multi value conditions , such as **is in** and **isn't in,** the list of values are separated with a semicolon or edited in separate window.

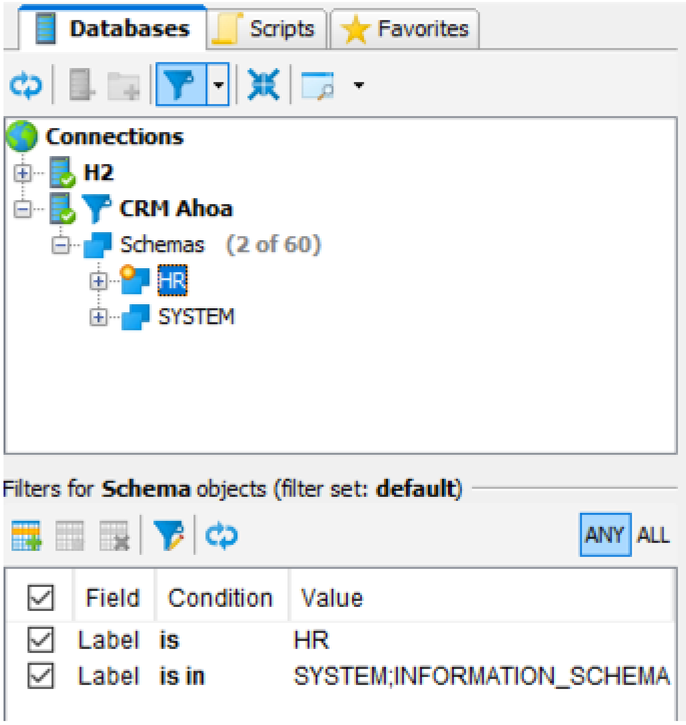
Each individual filter can be deactivated using the **check mark**. Uncheck it and the corresponding filter will not be used unless reactivated. The currently defined filters are listed in the upper object type list for each object type. The leading symbol shows either a green check mark which indicates that some of its filters are active or a yellow symbol that shows that no filter is active.

You can define more than one filter. Just click the plus or double plus (to duplicate an existing row) buttons. If you have more than one active filter, you must also select if the filter should match **Any** or **All** filter entries.

Inline Objects Filtering

The filter area can also be displayed just below the objects tree in the Databases tab. This is convenient as you can then quickly manage and verify the effect of certain filters. To toggle the display of the filtering area, either click the left part of the funnel icon in the **Databases** tab toolbar or click the right menu arrow and select **Database Objects Filter->Show/Hide Filter Area**.

> Managing filters and opening the filter editor requires that a node is selected in the Databases tab tree and that the related database connection is connected.

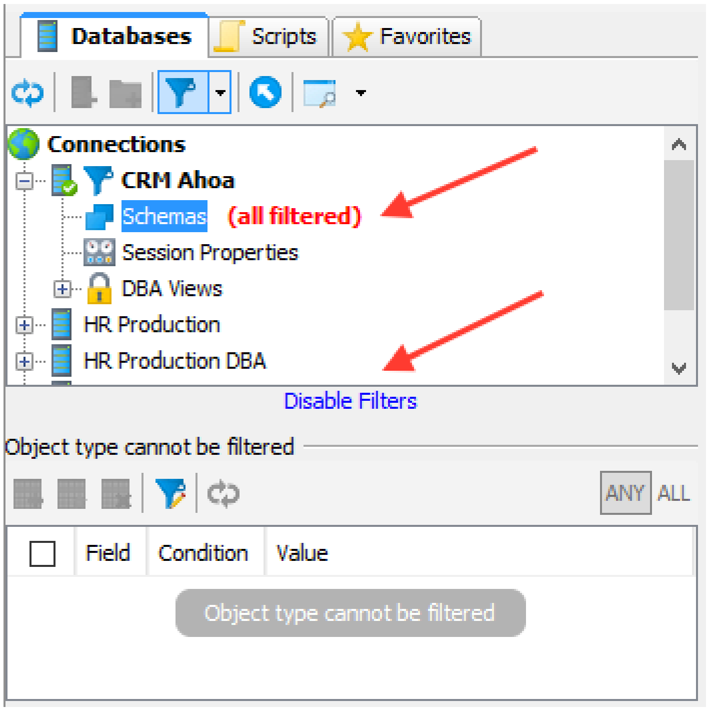


Connections
- H2
- CRM Ahoa
  - Schemas (2 of 60)
    - HR
    - SYSTEM

Filters for **Schema** objects (filter set: **default**)

☑	Field	Condition	Value
☑	Label	is	HR
☑	Label	is in	SYSTEM;INFORMATION_SCHEMA

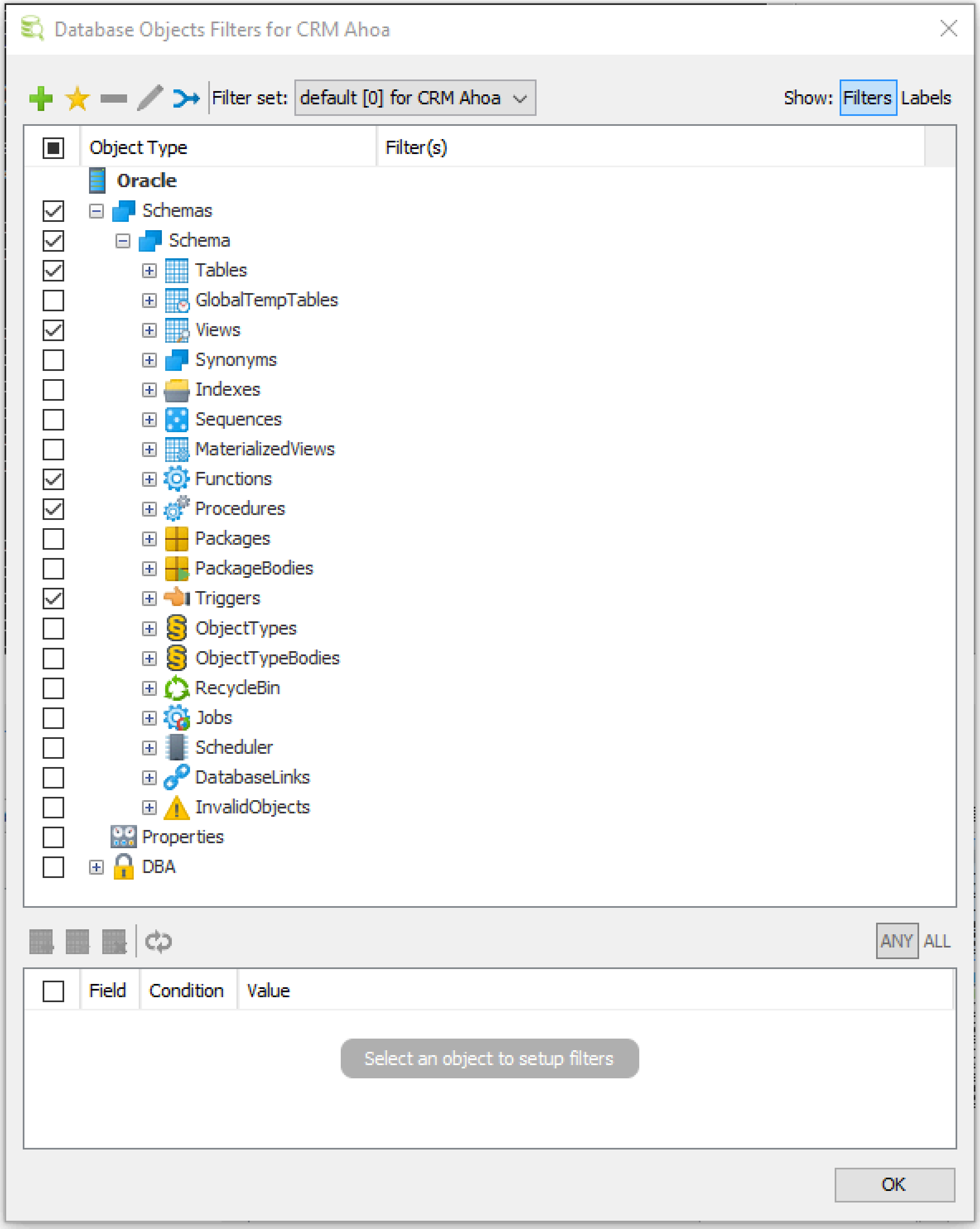
In this example, the filter area is displayed with the filters for the **Schema** object type listed. When filters have been modified, you need to manually apply the filters using one of the **reload buttons** in the filter area toolbar or in the Databases tab toolbar. If a filter results in all object nodes being filtered, " **(all filtered)** " is displayed next to the parent object node in the tree. Since the filtered object type is now invisible as a result of the current filter, you need to disable the filter to refine it. Do this by selecting the parent node with the " **(all filtered)** " label and then click the " Disable Filters " link just below the objects tree:



Clicking " Disable Filters**"** deactivates all filters for the object type and you are now able to modify the filter to get the matches you want.

4.3.2 Object Type Visibility

Object type visibility is the functionality used to define what object types should be visible in the Databases tab. For some databases, the tree of object types can be really long and many objects are rarely used or of any interest. By hiding object types, the tree is compressed to only show what you are really interested in. To control the visibility of object types, open the **Filter Editor**.



Database Objects Filters for CRM Ahoa

Filter set: default [0] for CRM Ahoa

Show: Filters Labels

■	Object Type	Filter(s)
	Oracle	
☑	Schemas	
☑	Schema	
☑	Tables	
□	GlobalTempTables	
☑	Views	
□	Synonyms	
□	Indexes	
□	Sequences	
□	MaterializedViews	
☑	Functions	
☑	Procedures	
□	Packages	
□	PackageBodies	
☑	Triggers	
□	ObjectTypes	
□	ObjectTypeBodies	
□	RecycleBin	
□	Jobs	
□	Scheduler	
□	DatabaseLinks	
□	InvalidObjects	
□	Properties	
□	DBA	

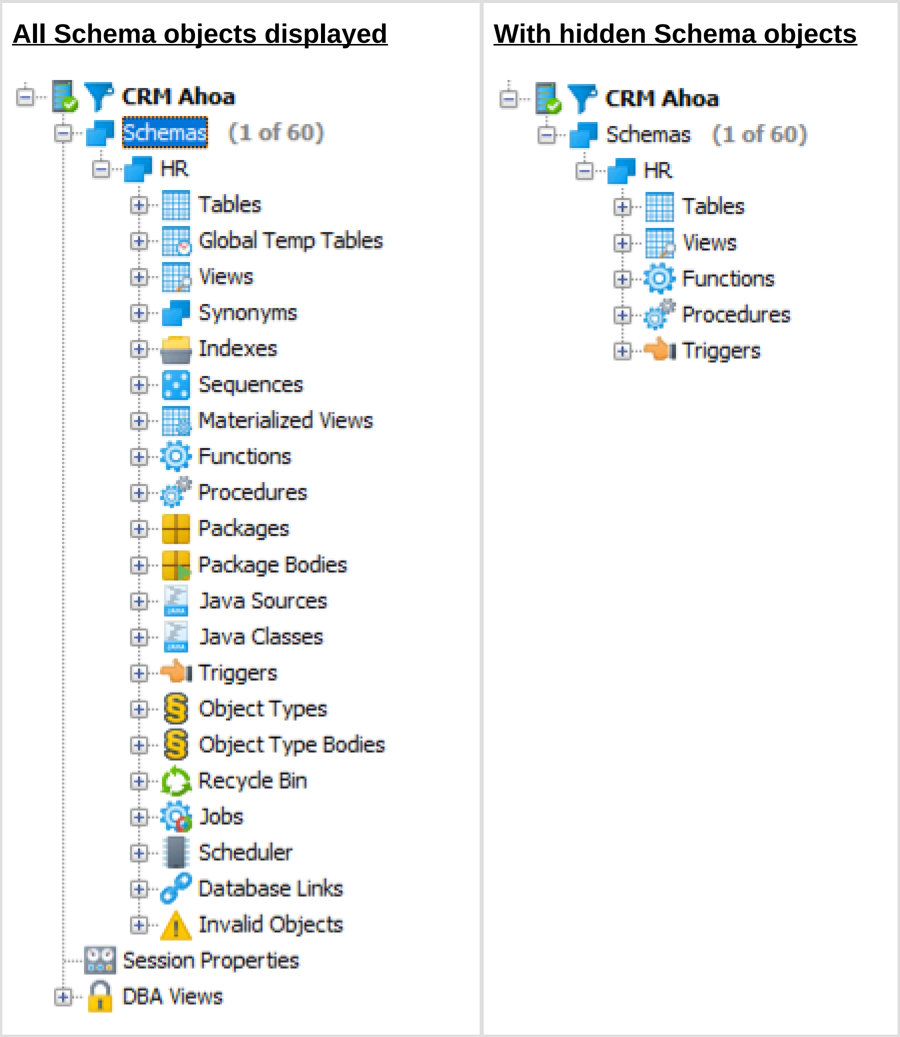
ANY ALL

□	Field	Condition	Value

Select an object to setup filters

OK

The previous image shows the object types available under a **Schema** in Oracle. As you can see, there are plenty of them. (Their sub object types are collapsed in the example for better illustration). The following example shows the database tree in the Databases tab before and after the visibility has been set based on the previous screenshot.

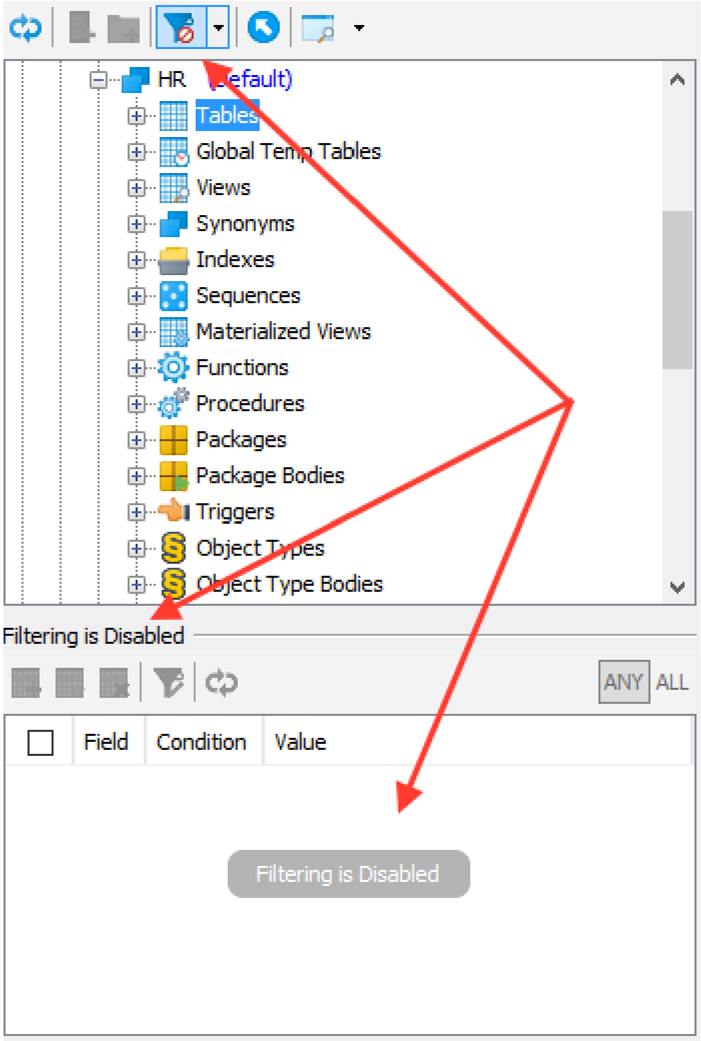


All Schema objects displayed

With hidden Schema objects

4.3.3 Temporarily Disable Filtering

While browsing the objects in the Databases tab, it may be convenient to quickly toggle between the standard non filtered view and the filtered view of objects and types. This is easily accomplished with the **Disable Filtering** action in the funnel drop-down menu. While filtering is disabled, the funnel symbol in the toolbar shows a red indicator and text in the inline filtering area shows the current status.



While filtering is disabled it is not possible to manage filters at all and the related actions are disabled.

Disabling filters is applied per database connection. This means that if you disable the current filter set on a MySQL connection it wont affect any other database connections.

4.3.4 Filter Sets

When you apply an object filter or hide an object type, that configuration is saved in a **Filter Set** that is saved between sessions. For every database connection, there is always a **default filter set**. If you are happy with the basic filtering capability you can stop reading here. If you however are interested in having multiple filter sets that can optionally be shared between multiple database connections and easily be switched between, keep on reading.

Filter sets are managed in the filter editor. Here you can create and delete filter sets and merge from another. Filter sets are either associated with a specific database connection, its name is then **default for <database**

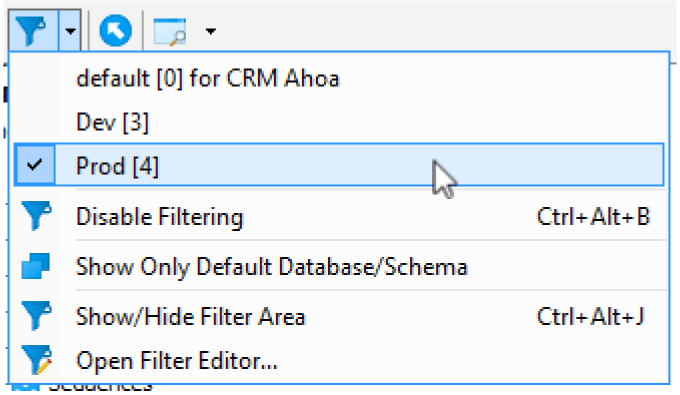
connection name> or a custom filter set that can be enabled for many connections, named as you like. A custom filter set is always associated with the database type currently being used for your database connection. The latter means that you are able to share filters sets that are all associated with the same database type. If you create a filter set for MySQL then this will never show up if you are working with an Oracle database.

The Filter Set drop-down in the filter editor shows the currently used filter set, and when clicked, the available filter sets that you may switch to.

The merge functionality can merge not only from custom filter sets but also from the default filter sets associated with other database connections with the same database type.

Switching Filter Set

In the filter editor, you switch filter set by selecting one from the Filter Set drop-down. In the Databases tab, there is the funnel symbol which when clicked is used to toggle the display of the inline filter area. Clicking on the funnel drop-down symbol opens a menu:


default [0] for CRM Ahoa
Dev [3]
✓ Prod [4]
Disable Filtering Ctrl+Alt+B
Show Only Default Database/Schema
Show/Hide Filter Area Ctrl+Alt+J
Open Filter Editor...

At the top of the menu, the default filter set for the database connection is displayed first with any custom filter sets below it. An entry that is check marked indicates that it is active.

4.3.5 Show Only Default Database/Schema filter

There is a special filter used to filter any database and schema objects to show only the default for the session. It is listed as Show Only Default Database/Schema in the filter menu. When selected, a special filter is applied on the corresponding Database and/or Schema object type and the effect is what its name implies.

5 Working with Tables

DbVisualizer provides many ways to work with tables.

5.1 Creating a Table

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

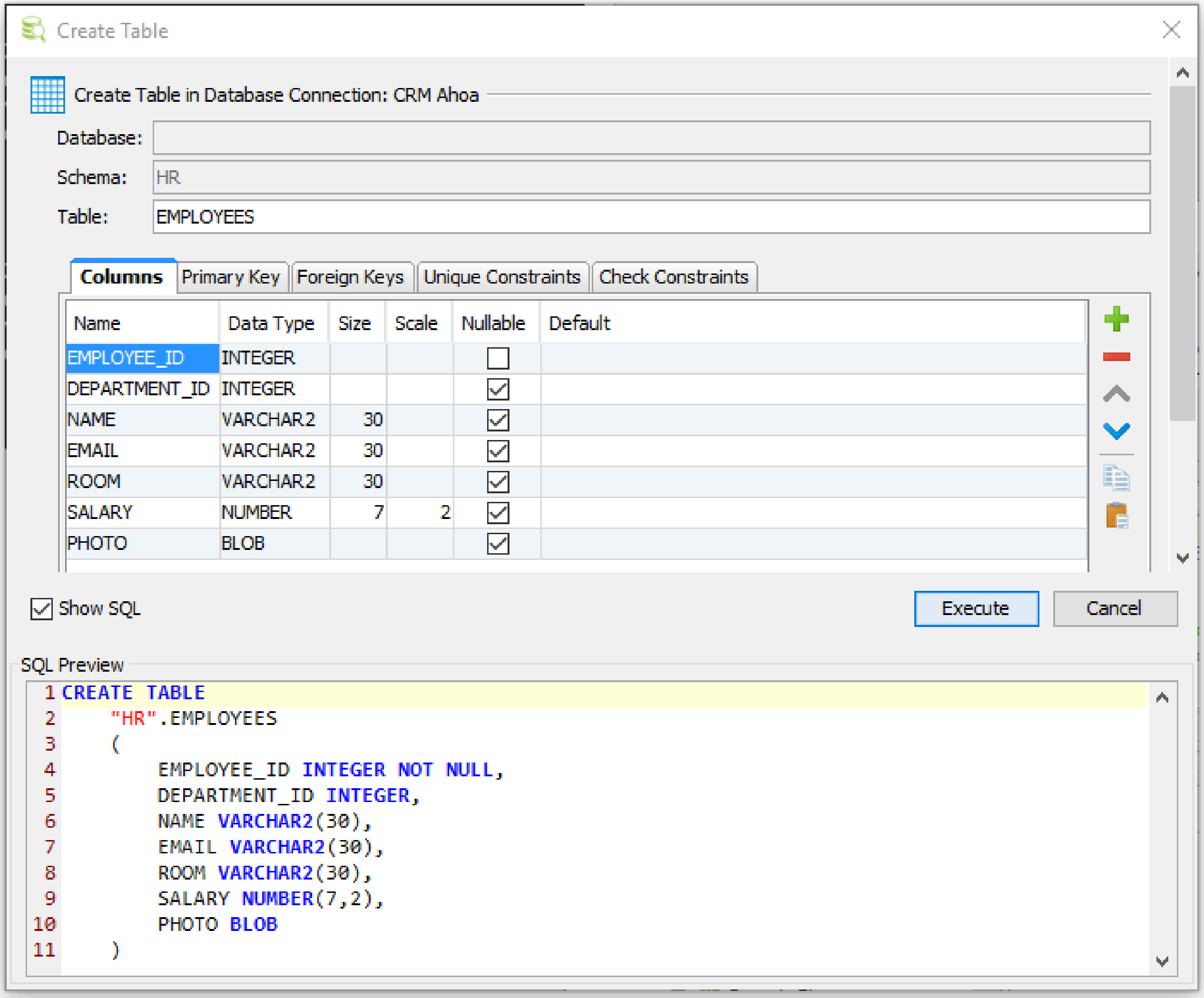
The Create Table dialog helps you create a table without writing SQL.

- Opening the Create Table Dialog (see page 60)
- Columns Tab (see page 62)
- Primary Key Tab (see page 64)
- Foreign Keys Tab (see page 64)
- Unique Constraints Tab (see page 66)
- Check Constraints Tab (see page 67)
- Indexes Tab (see page 67)
- SQL Preview (see page 68)
- Execute (see page 69)

5.1.1 Opening the Create Table Dialog

To create a new table:

1. Expand nodes in the tree under the connection node in the **Databases** tab tree until you reach the **Tables** node,
2. Select the **Tables** node and open the **Create Table** dialog from the right-click menu.



Create Table in Database Connection: CRM Ahoa

Database:

Schema: HR

Table: EMPLOYEES

Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR2	30		☑	
EMAIL	VARCHAR2	30		☑	
ROOM	VARCHAR2	30		☑	
SALARY	NUMBER	7	2	☑	
PHOTO	BLOB			☑	

☑ Show SQL

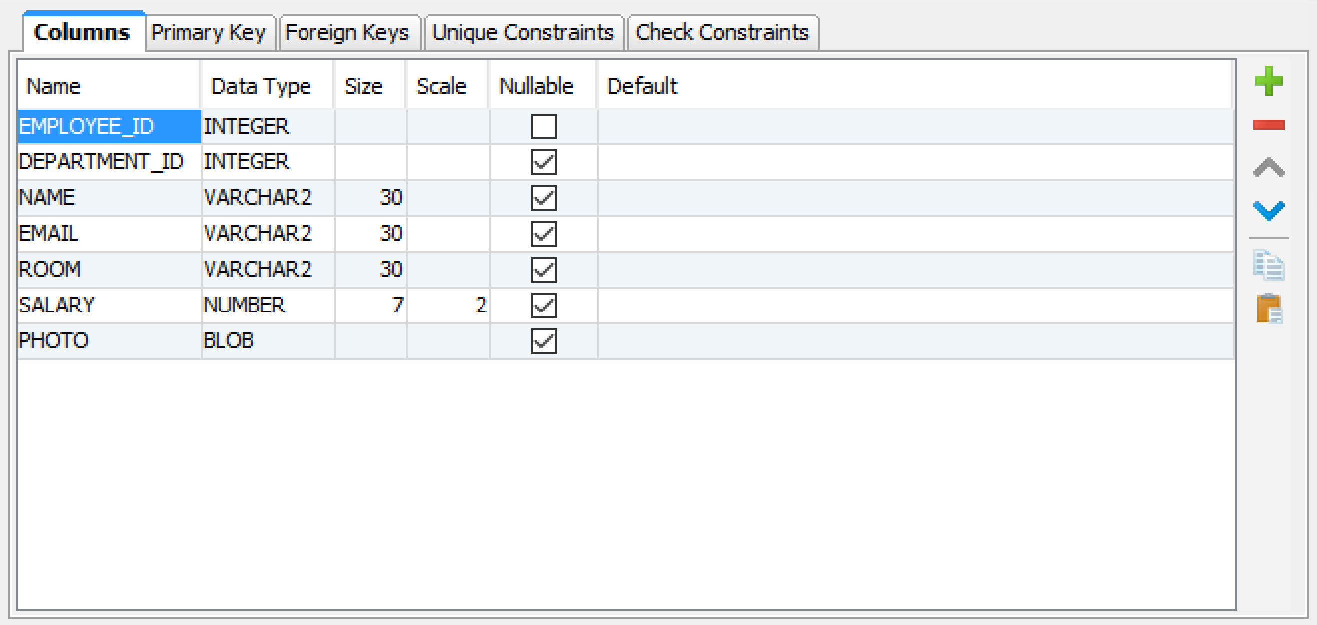
```
CREATE TABLE
    "HR".EMPLOYEES
    (
        EMPLOYEE_ID INTEGER NOT NULL,
        DEPARTMENT_ID INTEGER,
        NAME VARCHAR2(30),
        EMAIL VARCHAR2(30),
        ROOM VARCHAR2(30),
        SALARY NUMBER(7,2),
        PHOTO BLOB
    )
```

The **Create Table** dialog 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 dialog is opened. Table name is set to a default name that you should 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 shows the SQL statement for creating the table based on your input.

5.1.2 Columns Tab

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

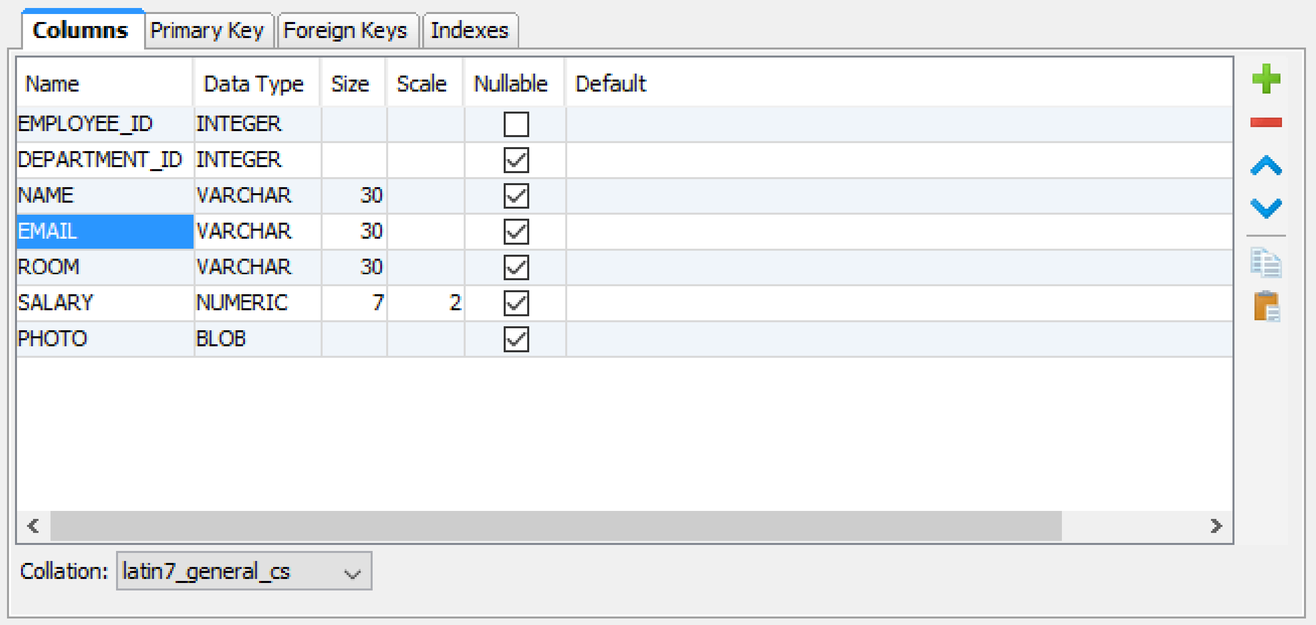


Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR2	30		☑	
EMAIL	VARCHAR2	30		☑	
ROOM	VARCHAR2	30		☑	
SALARY	NUMBER	7	2	☑	
PHOTO	BLOB			☑	

To add a column:

1. Click the **Add** button,
2. Enter the name of the column in the first field and select a data type from a drop down list in the second field, or start typing the data type name to find it and select it with the **Enter** key. The list contains the names of all data types the database supports,
3. 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),
4. In the last two fields, 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.

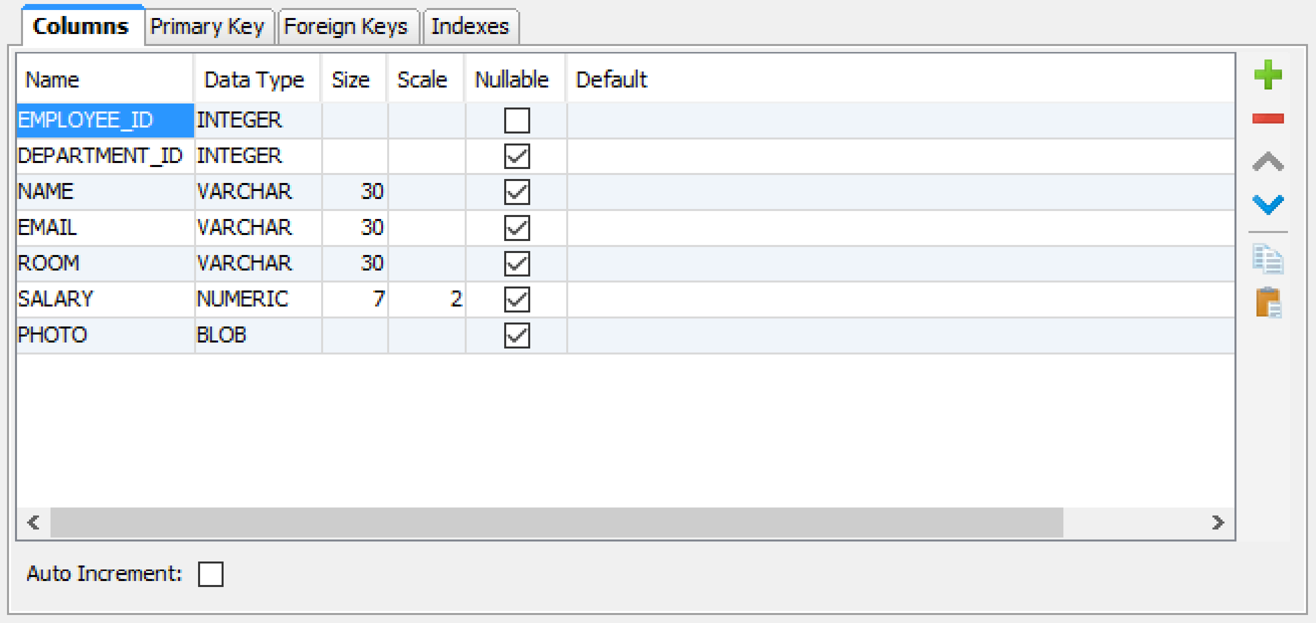
Below the column list, you may find additional fields depending on the features supported by the database you create the table for and the data type for the current column. The **Collation** field is shown for character columns if the database supports the declaration of a collation for textual data.



Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR	30		☑	
EMAIL	VARCHAR	30		☑	
ROOM	VARCHAR	30		☑	
SALARY	NUMERIC	7	2	☑	
PHOTO	BLOB			☑	

Collation: latin7_general_cs

The **Auto Increment** field, and possibly **Start With** and **Increment By** fields, are shown 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			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR	30		☑	
EMAIL	VARCHAR	30		☑	
ROOM	VARCHAR	30		☑	
SALARY	NUMERIC	7	2	☑	
PHOTO	BLOB			☑	

Auto Increment: □

The **Create Table** dialog 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.

To remove a column:

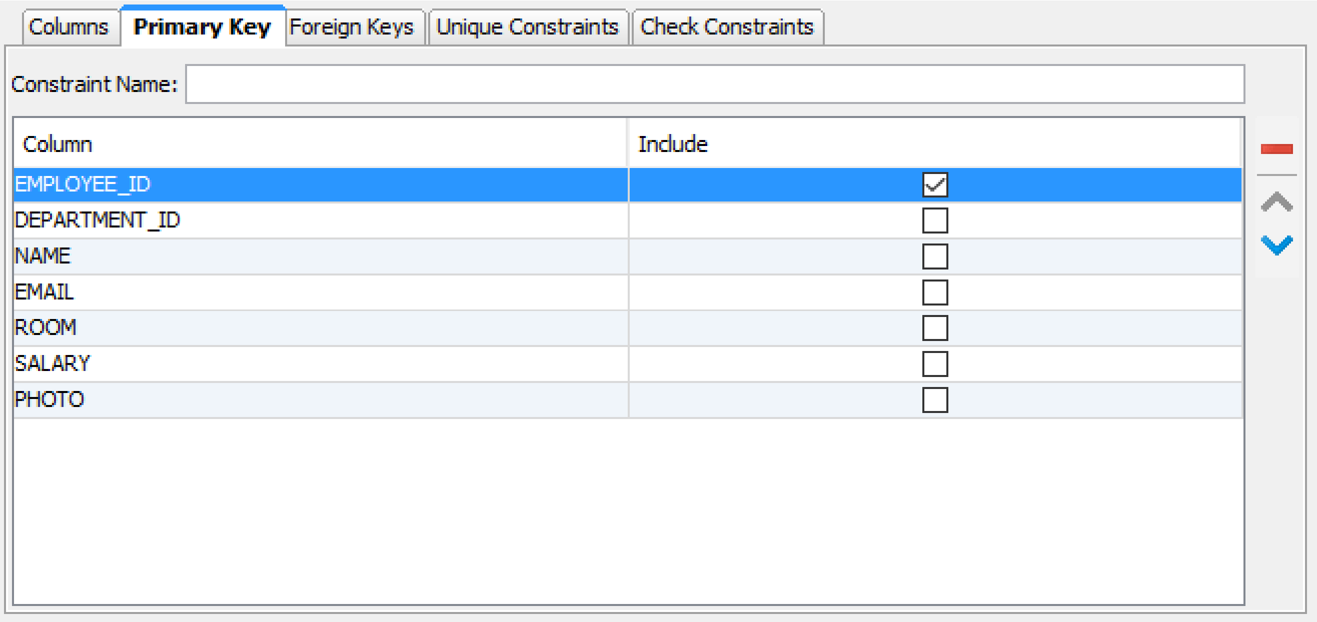
1. Select a cell in the column row,
2. Click the **Remove** button.

To move a column to another location:

1. Select a cell in the column row,
2. Click the **Up** or **Down** buttons.

5.1.3 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.



Constraint Name:

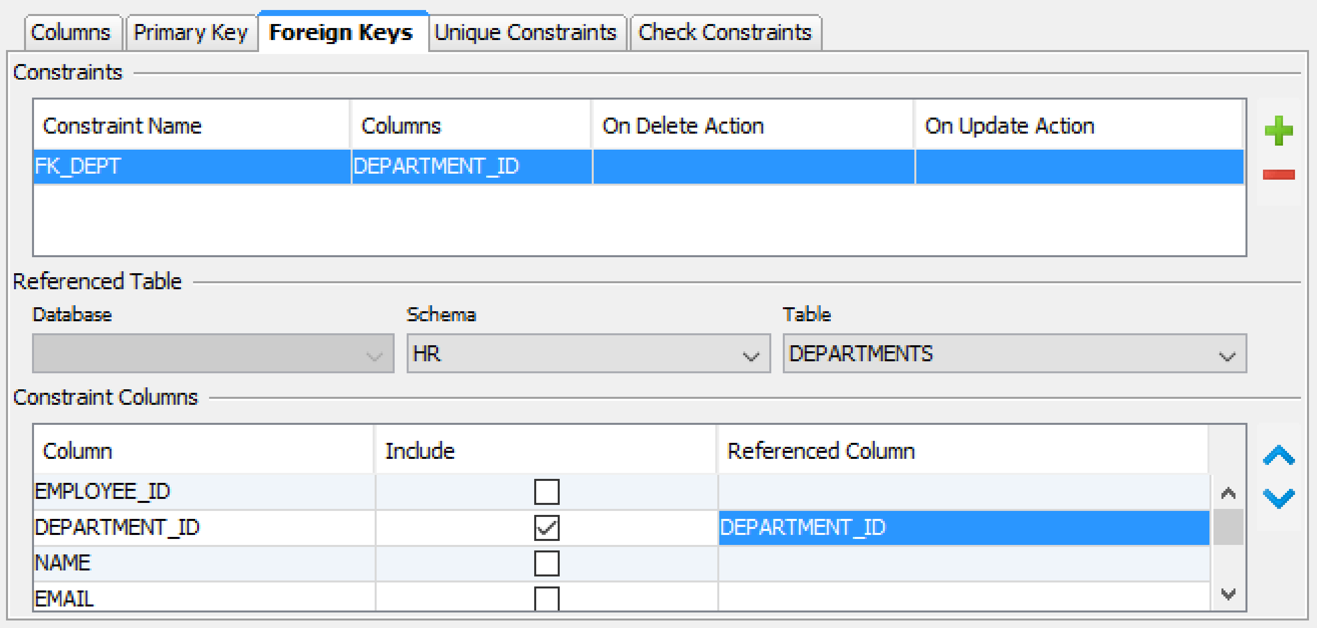
Column	Include
EMPLOYEE_ID	☑
DEPARTMENT_ID	□
NAME	□
EMAIL	□
ROOM	□
SALARY	□
PHOTO	□

To declare a Primary Key:

1. Optionally enter a constraint name for the primary key constraint in the **Constraint Name** field,
2. Select the columns to be part of the primary key by clicking the checkboxes in the **Include** field in the columns list.

5.1.4 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 you can optionally declare rules for what to do when a referenced primary key is removed or updated in the referenced table.



Constraints

Constraint Name	Columns	On Delete Action	On Update Action
FK_DEPT	DEPARTMENT_ID		

Referenced Table

Database	Schema	Table
	HR	DEPARTMENTS

Constraint Columns

Column	Include	Referenced Column
EMPLOYEE_ID	☐	
DEPARTMENT_ID	☑	DEPARTMENT_ID
NAME	☐	
EMAIL	☐	

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:

1. Click the **Add** button next to the list of foreign keys,
2. Optionally enter a name for the foreign key in the first field in the list,
3. 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,
4. Use the **Referenced Table** controls to select the table that the foreign key refers to,
5. 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.

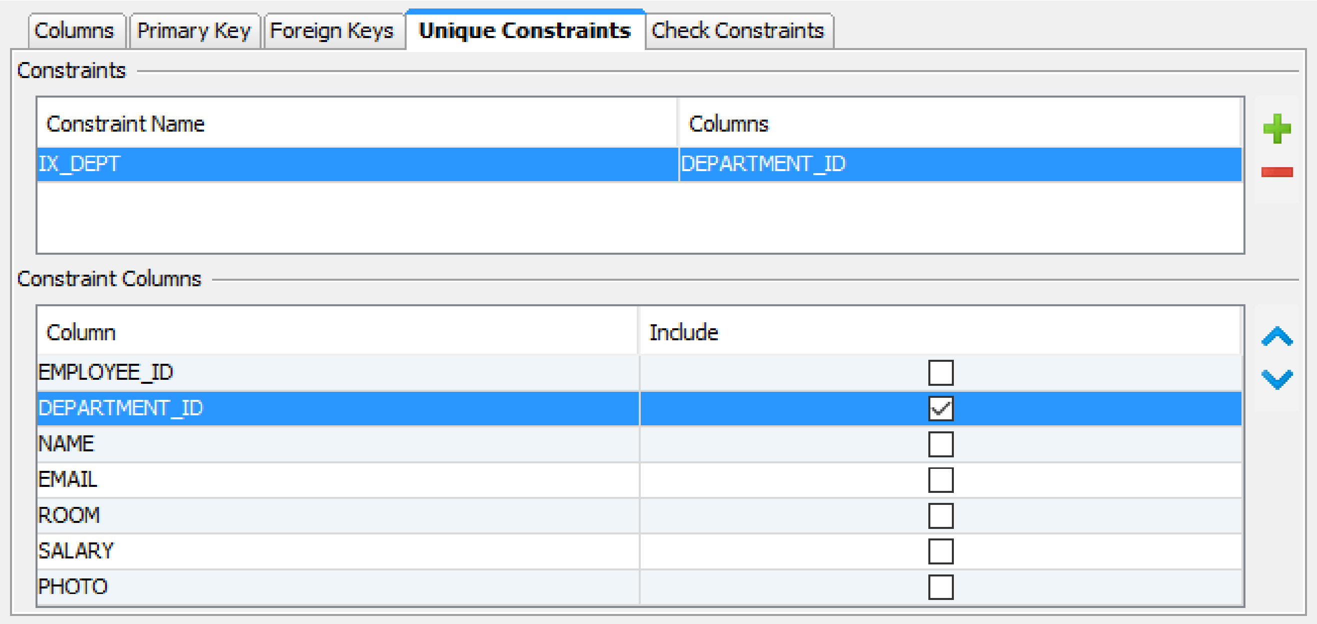
You can change the column order for the key with the **Up** and **Down** buttons.

To remove an existing foreign key:

1. Select the foreign key in the list in the top section,
2. Click the **Remove** button.

5.1.5 Unique Constraints Tab

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	□
DEPARTMENT_ID	☑
NAME	□
EMAIL	□
ROOM	□
SALARY	□
PHOTO	□

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:

1. Click the **Add** button,
2. 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 constraint,
3. Select the columns to be part of the constraint by clicking the checkboxes in the Include field in the columns list.

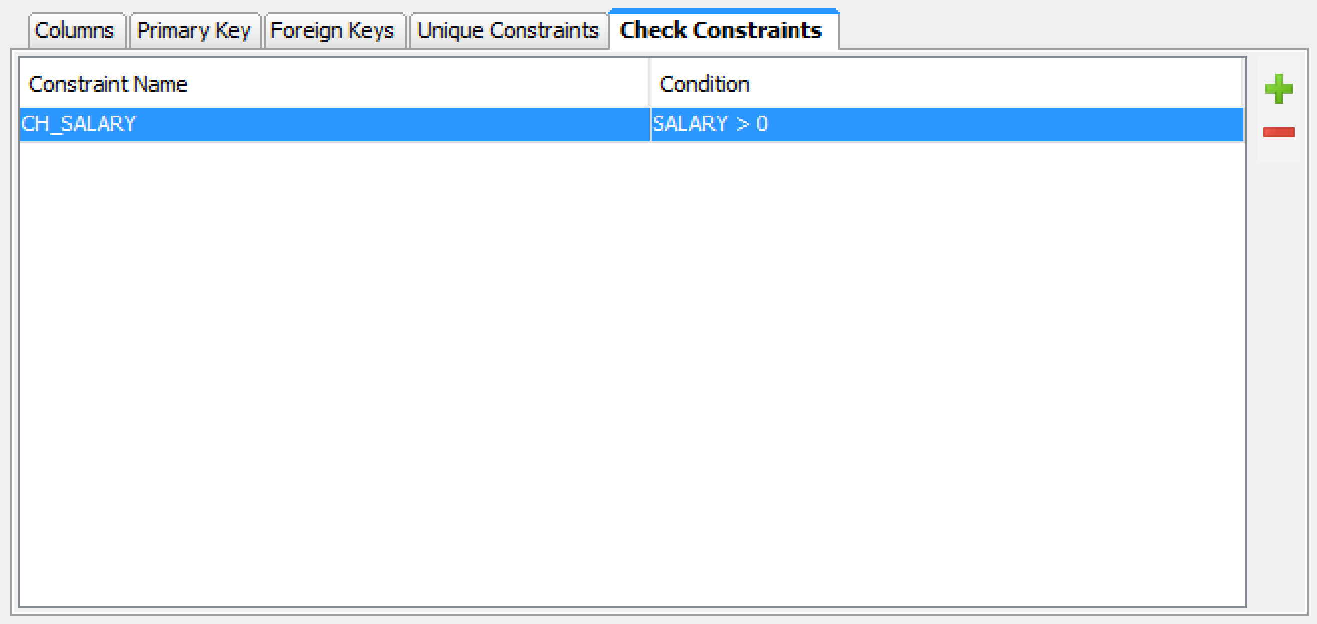
You can change the column order for the constraint with the **Up** and **Down** buttons.

To remove an existing constraint:

1. Select the constraint in the list in the top section,
2. Click the **Remove** button.

5.1.6 Check Constraints Tab

The **Check Constraints** tab is only available for databases that support this constraint type. A check constraint declares 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, you may see auto-generated check constraints for columns that you marked as not allowing null values in the **Columns** tab.



Constraint Name	Condition
CH_SALARY	SALARY > 0

To create a check constraint:

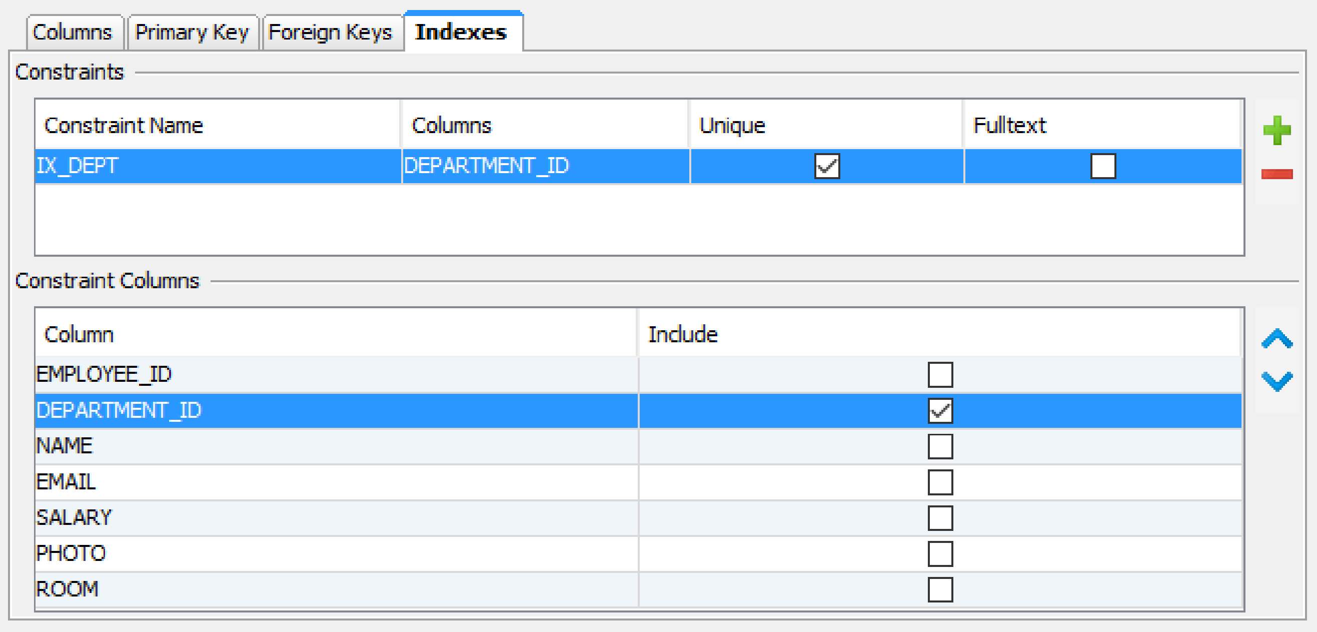
1. Click the **Add** button,
2. Optionally enter a constraint name in the **Constraint Name** field.
3. Enter the condition for the column in the **Condition** field. You can use the same type of conditions as you use in a SELECT WHERE clause.

To remove an existing constraint:

1. Select the constraint in the list,
2. Click the **Remove** button.

5.1.7 Indexes Tab

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.



Constraint Name	Columns	Unique	Fulltext
IX_DEPT	DEPARTMENT_ID	☑	□

Column	Include
EMPLOYEE_ID	□
DEPARTMENT_ID	☑
NAME	□
EMAIL	□
SALARY	□
PHOTO	□
ROOM	□

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:

1. Click the **Add** button,
2. 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 constraint,
3. If you want the index columns to have unique values for all rows in the table, click the checkbox in the **Unique** field,
4. 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 column order for the index with the **Up** and **Down** buttons.To remove an existing index:

1. Select the index in the list in the top section,
2. Click the **Remove** button.

5.1.8 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 corresponding choices in the preview area right-click menu.

5.1.9 Execute

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

5.2 Altering a Table

> **Only in DbVisualizer Pro**
>
> This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

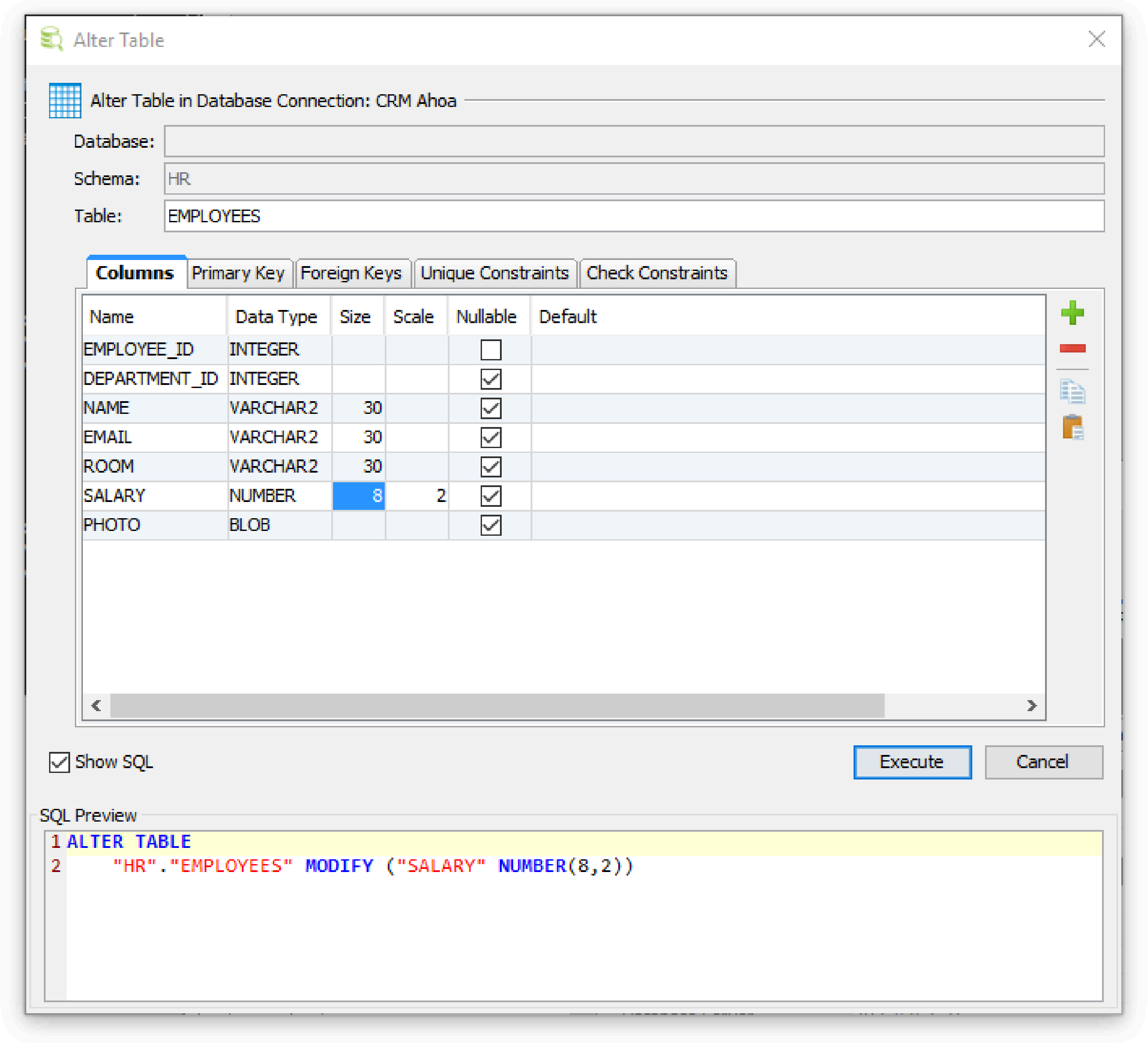
The Alter Table dialog helps you alter a table without writing SQL.

- Opening the Alter Table Dialog (see page 69)
- Columns Tab (see page 71)
- Primary Key Tab (see page 73)
- Foreign Keys Tab (see page 73)
- Unique Constraints Tab (see page 75)
- Check Constraints Tab (see page 76)
- Indexes Tab (see page 76)
- SQL Preview (see page 77)
- Execute (see page 78)

5.2.1 Opening the Alter Table Dialog

To create a new table:

1. Locate the table node in the **Databases** tab tree,
2. Select the table node and open the **Alter Table** dialog from the right-click menu.



Alter Table in Database Connection: CRM Ahoa

Database:

Schema: HR

Table: EMPLOYEES

Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR2	30		☑	
EMAIL	VARCHAR2	30		☑	
ROOM	VARCHAR2	30		☑	
SALARY	NUMBER	8	2	☑	
PHOTO	BLOB			☑	

☑ Show SQL

SQL Preview

```
ALTER TABLE
    "HR"."EMPLOYEES" MODIFY ("SALARY" NUMBER(8,2))
```

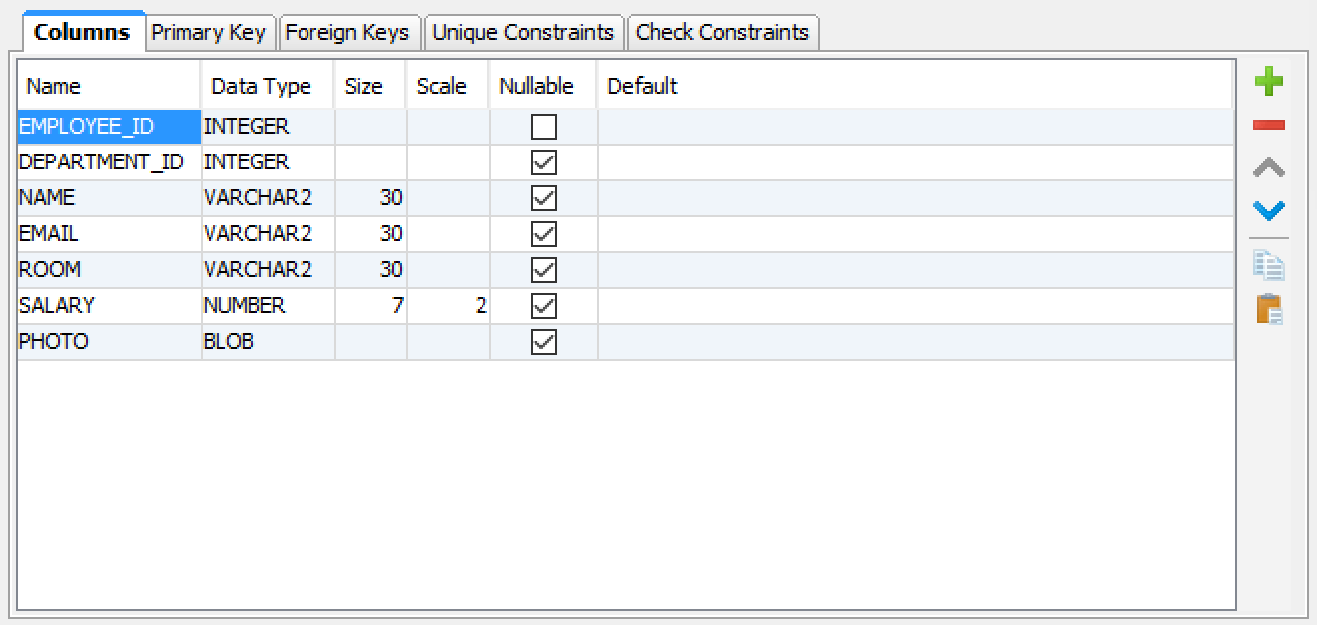
The **Alter Table** dialog is organized in three areas from the top:

- General Table Info
  Specifies the owning database connection, database and/or schema, and table name. These are picked up from the selection in the tree when the assistant is started.
- 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 shows the SQL statements for altering the table based on your input.

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.

5.2.2 Columns Tab

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

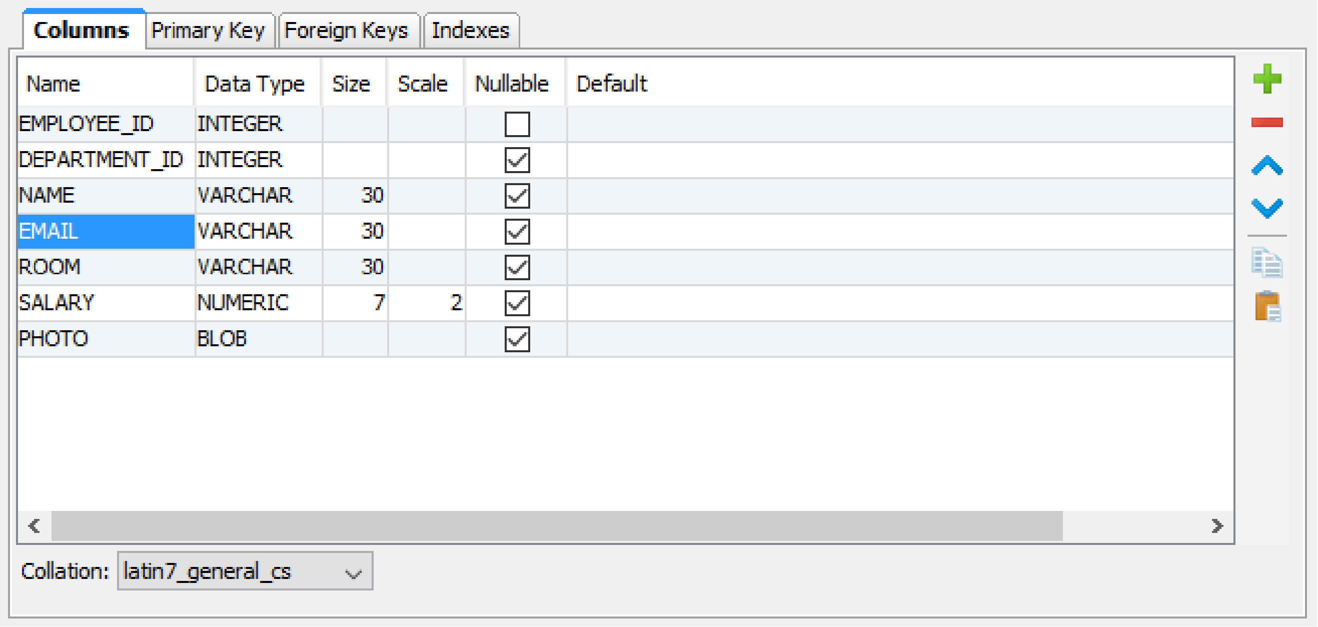


Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR2	30		☑	
EMAIL	VARCHAR2	30		☑	
ROOM	VARCHAR2	30		☑	
SALARY	NUMBER	7	2	☑	
PHOTO	BLOB			☑	

To add a column:

1. Click the **Add** button,
2. Enter the name of the column in the first field and select a data type from a drop down list in the second field, or start typing the data type name to find it and select it with the **Enter** key. The list contains the names of all data types the database supports,
3. 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),
4. In the last two fields, 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.

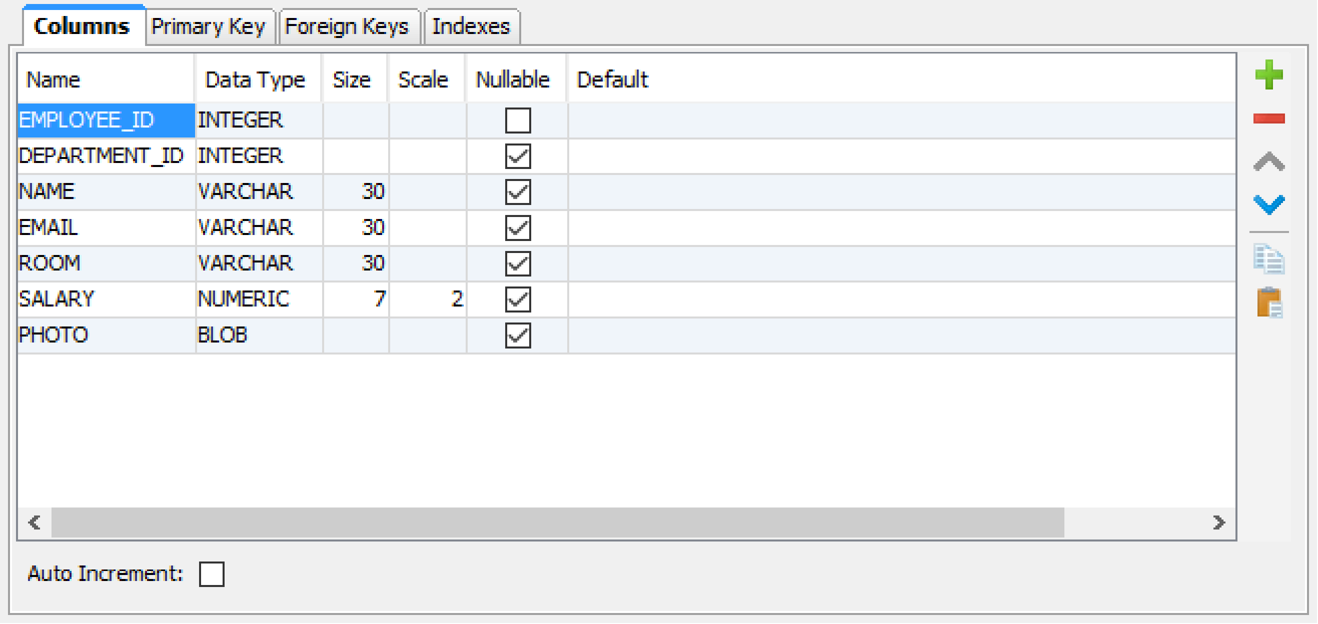
Below the column list, you may find additional fields depending on the features supported by the database you create the table for and the data type for the current column. The **Collation** field is shown for character columns if the database supports the declaration of a collation for textual data.



Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR	30		☑	
EMAIL	VARCHAR	30		☑	
ROOM	VARCHAR	30		☑	
SALARY	NUMERIC	7	2	☑	
PHOTO	BLOB			☑	

Collation: latin7_general_cs

The **Auto Increment** field, and possibly **Start With** and **Increment By** fields, are shown 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			□	
DEPARTMENT_ID	INTEGER			☑	
NAME	VARCHAR	30		☑	
EMAIL	VARCHAR	30		☑	
ROOM	VARCHAR	30		☑	
SALARY	NUMERIC	7	2	☑	
PHOTO	BLOB			☑	

Auto Increment: □

The **Create Table** dialog 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.

To remove a column:

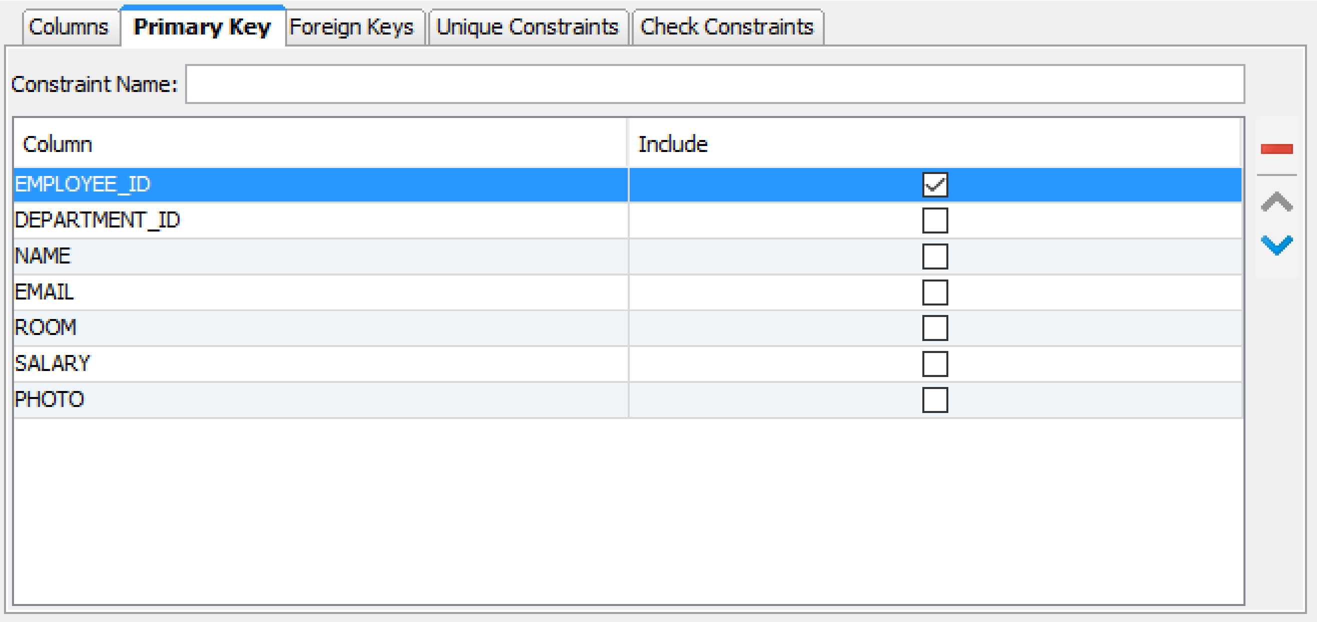
1. Select a cell in the column row,
2. Click the **Remove** button.

To move a column to another location:

1. Select a cell in the column row,
2. Click the **Up** or **Down** buttons.

5.2.3 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.



Constraint Name:

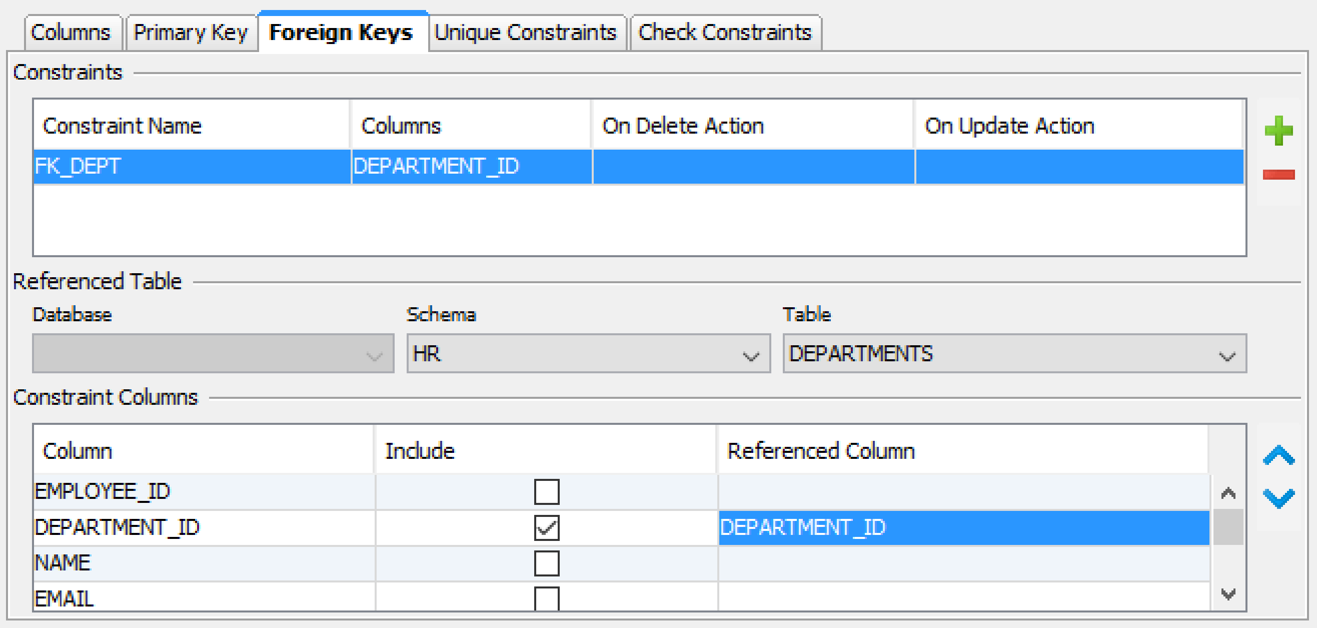
Column	Include
EMPLOYEE_ID	☑
DEPARTMENT_ID	□
NAME	□
EMAIL	□
ROOM	□
SALARY	□
PHOTO	□

To declare a Primary Key:

1. Optionally enter a constraint name for the primary key constraint in the **Constraint Name** field,
2. Select the columns to be part of the primary key by clicking the checkboxes in the **Include** field in the columns list.

5.2.4 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 you can optionally declare rules for what to do when a referenced primary key is removed or updated in the referenced table.



Constraint Name	Columns	On Delete Action	On Update Action
FK_DEPT	DEPARTMENT_ID		

Database:
Schema: HR
Table: DEPARTMENTS

Column	Include	Referenced Column
EMPLOYEE_ID	□	
DEPARTMENT_ID	☑	DEPARTMENT_ID
NAME	□	
EMAIL	□	

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:

1. Click the **Add** button next to the list of foreign keys,
2. Optionally enter a name for the foreign key in the first field in the list,
3. 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,
4. Use the **Referenced Table** controls to select the table that the foreign key refers to,
5. 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.

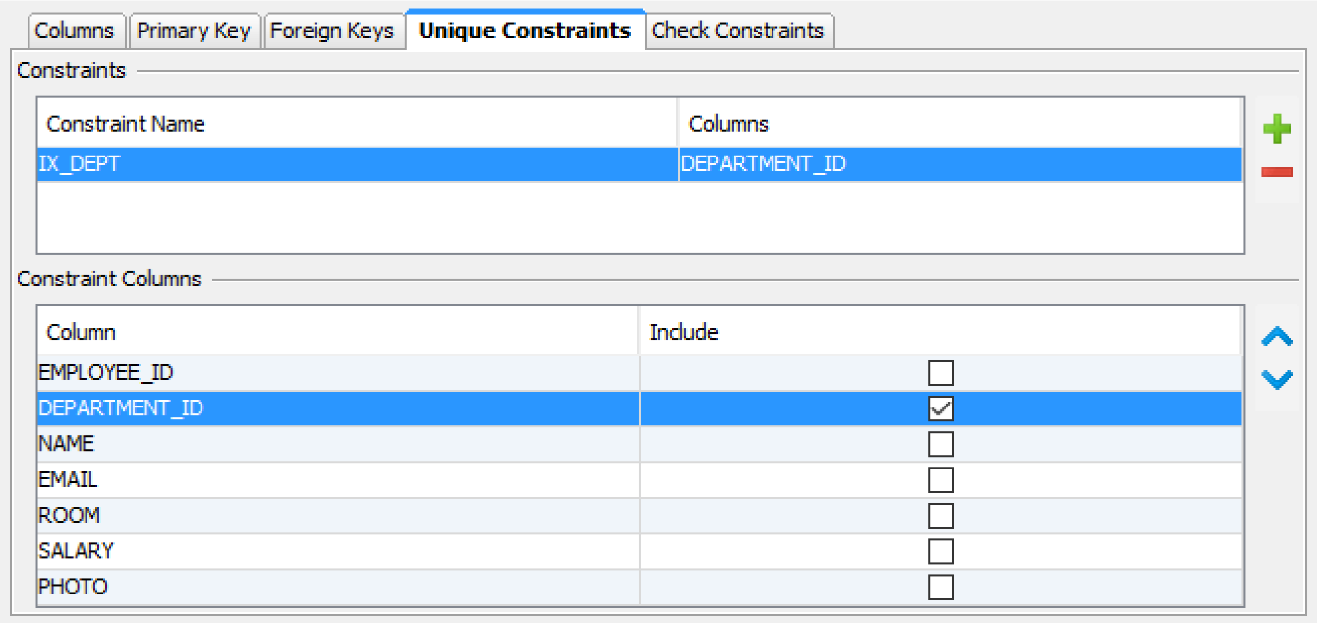
You can change the column order for the key with the **Up** and **Down** buttons.

To remove an existing foreign key:

1. Select the foreign key in the list in the top section,
2. Click the **Remove** button.

5.2.5 Unique Constraints Tab

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	□
DEPARTMENT_ID	☑
NAME	□
EMAIL	□
ROOM	□
SALARY	□
PHOTO	□

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:

1. Click the **Add** button,
2. 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 constraint,
3. Select the columns to be part of the constraint by clicking the checkboxes in the Include field in the columns list.

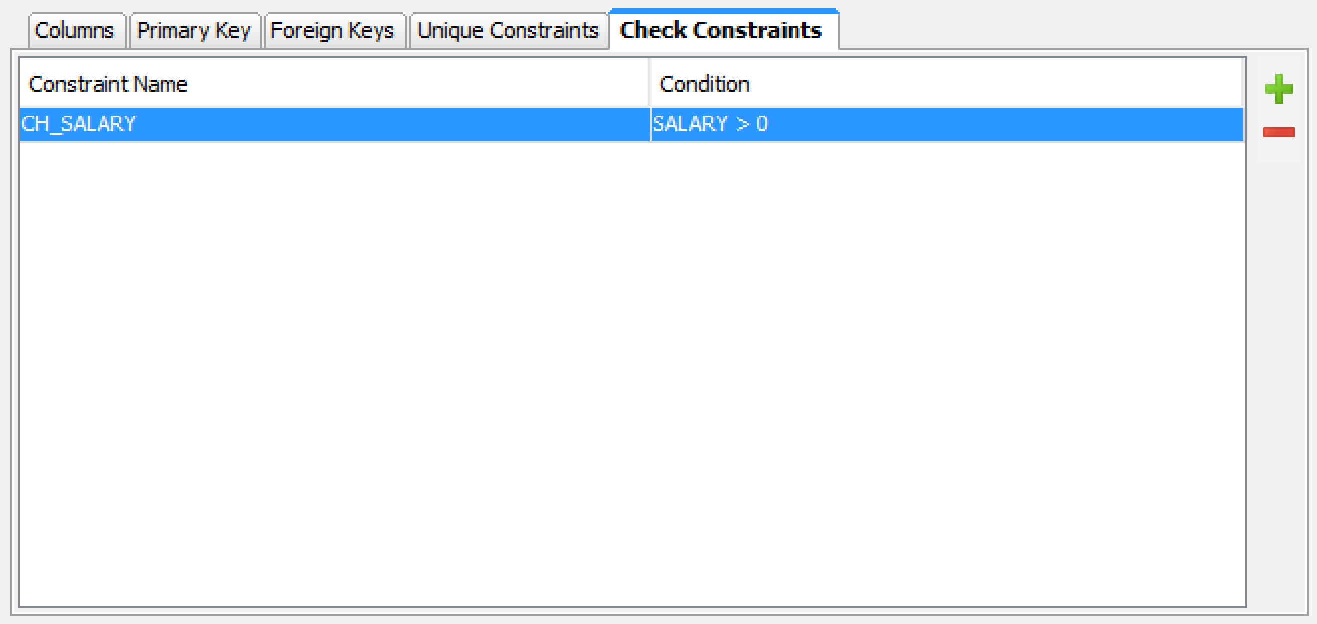
You can change the column order for the constraint with the **Up** and **Down** buttons.

To remove an existing constraint:

1. Select the constraint in the list in the top section,
2. Click the **Remove** button.

5.2.6 Check Constraints Tab

The **Check Constraints** tab is only available for databases that support this constraint type. A check constraint declares 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, you may see auto-generated check constraints for columns that you marked as not allowing null values in the **Columns** tab.



Constraint Name	Condition
CH_SALARY	SALARY > 0

To create a check constraint:

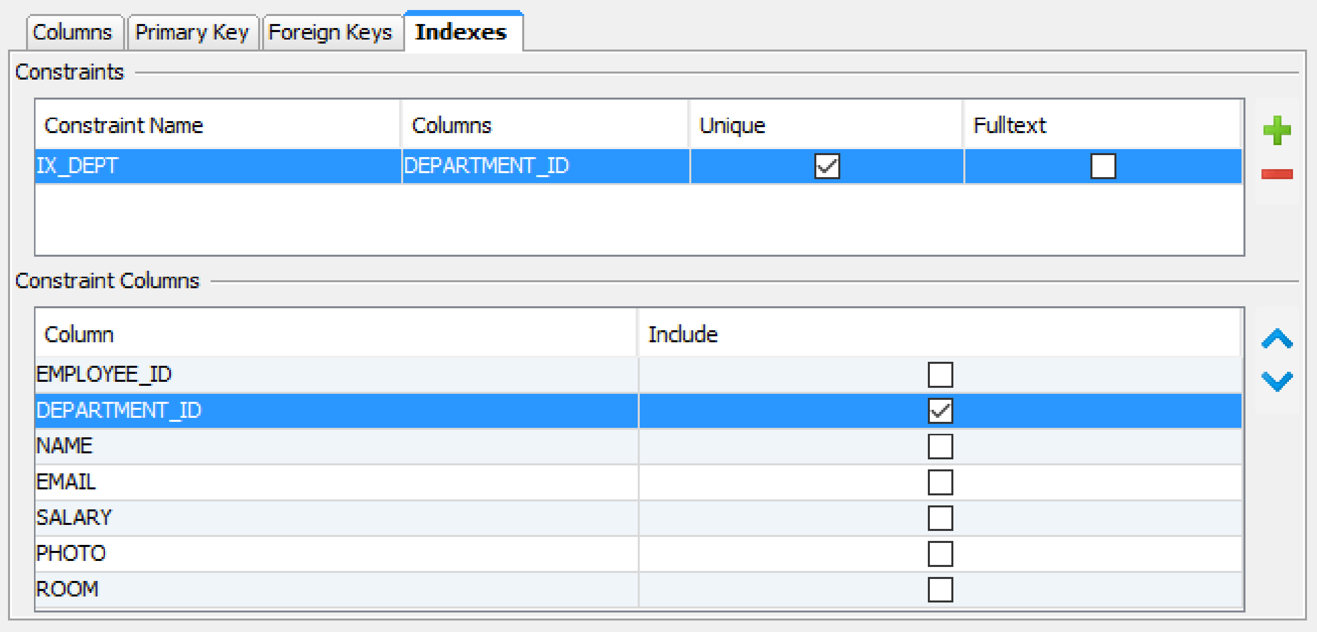
1. Click the **Add** button,
2. Optionally enter a constraint name in the **Constraint Name** field.
3. Enter the condition for the column in the **Condition** field. You can use the same type of conditions as you use in a SELECT WHERE clause.

To remove an existing constraint:

1. Select the constraint in the list,
2. Click the **Remove** button.

5.2.7 Indexes Tab

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.



Constraints

Constraint Name	Columns	Unique	Fulltext
IX_DEPT	DEPARTMENT_ID	☑	□

Constraint Columns

Column	Include
EMPLOYEE_ID	□
DEPARTMENT_ID	☑
NAME	□
EMAIL	□
SALARY	□
PHOTO	□
ROOM	□

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:

1. Click the **Add** button,
2. 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 constraint,
3. If you want the index columns to have unique values for all rows in the table, click the checkbox in the **Unique** field,
4. 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 column order for the index with the **Up** and **Down** buttons.To remove an existing index:

1. Select the index in the list in the top section,
2. Click the **Remove** button.

5.2.8 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 corresponding choices in the preview area right-click menu.

5.2.9 Execute

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

5.3 Creating a Trigger

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

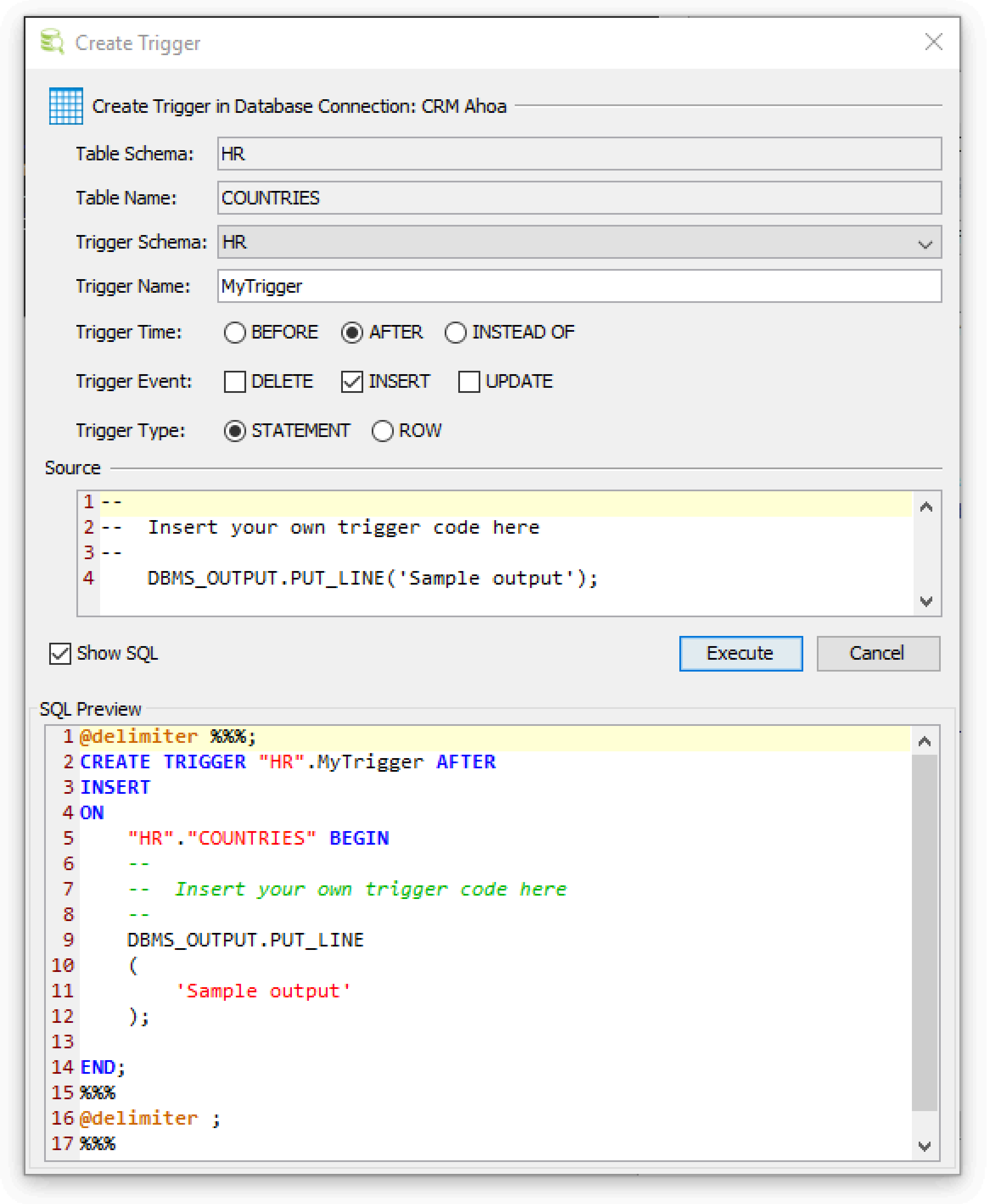
The Create Trigger dialog helps you create a trigger for a table.

- Opening the Create Trigger Dialog (see page 78)
- Trigger Editor (see page 80)

5.3.1 Opening the Create Trigger Dialog

To create a trigger for a table:

1. Locate the table in the **Databases** tab tree,
2. Select the table node and open the **Create Trigger** dialog from the right-click menu,



Create Trigger

Create Trigger in Database Connection: CRM Ahoa

Table Schema: HR

Table Name: COUNTRIES

Trigger Schema: HR

Trigger Name: MyTrigger

Trigger Time: ○ BEFORE ● AFTER ○ INSTEAD OF

Trigger Event: □ DELETE ☑ INSERT □ UPDATE

Trigger Type: ● STATEMENT ○ ROW

Source

```
--
--  Insert your own trigger code here
--
    DBMS_OUTPUT.PUT_LINE('Sample output');
```

☑ Show SQL

SQL Preview

```
@delimiter %%%;
CREATE TRIGGER "HR".MyTrigger AFTER
INSERT
ON
    "HR"."COUNTRIES" BEGIN
    --
    --  Insert your own trigger code here
    --
    DBMS_OUTPUT.PUT_LINE
    (
        'Sample output'
    );

END;
%%%
@delimiter ;
%%%
```

3. Enter the required info in the fields, e.g. trigger name. The fields are database dependent so the figure is just an example,
4. The **Source** area contains stub code that you can later edit in the Trigger Editor (see page 80). For most databases you can leave it as is, but for some databases, you must adjust the stub code to match your database objects.
5. Click the **Execute** button to create the trigger

5.3.2 Trigger Editor

To edit the trigger code:

1. Expand the **Trigger** node for the table in the Databases tab tree,
2. Double-click the trigger node to open its **Object View** tab,
3. Open the **Trigger Editor** tab and edit the code (see page 141) in the editor,
4. Click the **Save** toolbar button to save (and for some databases, compile) the trigger.

If the database reports any errors, the location of the errors are highlighted with curly red underlines in the editor for most databases. Hovering the mouse over such an underline shows the error message.

The **Log** tab in the results area also lists all errors. Clicking on the icon next to an error message selects the corresponding line and positions the caret at the error location, if the database reports error locations.

5.4 Creating an Index

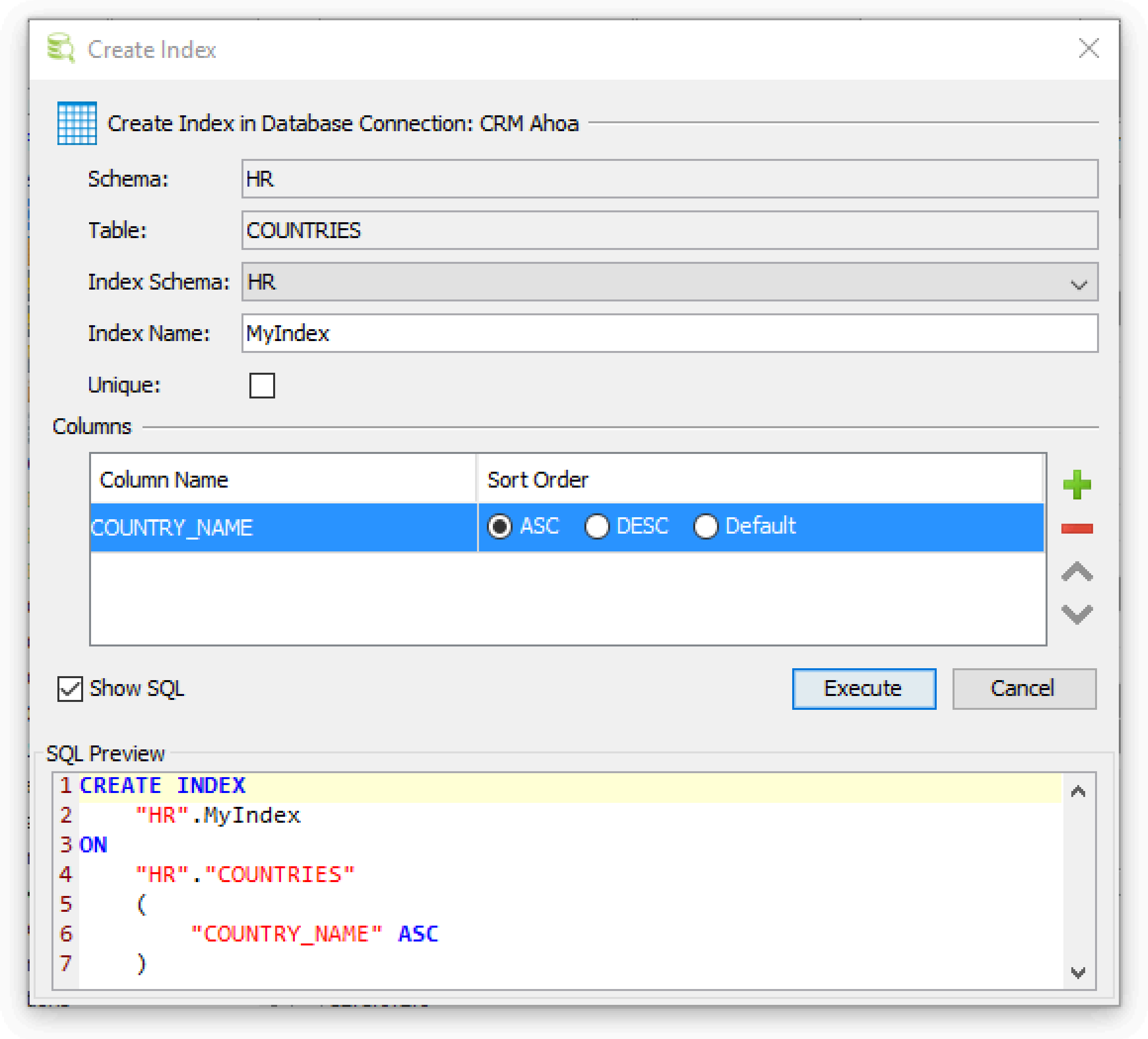
ⓘ **Only in DbVisualizer Pro**

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

The Create Index dialog helps you create an index for a table without writing SQL.

To create an index for a table:

1. Locate the table in the **Databases** tab tree,
2. Select the table node and open the **Create Index** dialog from the right-click menu,



Create Index in Database Connection: CRM Ahoa

Schema: HR

Table: COUNTRIES

Index Schema: HR

Index Name: MyIndex

Unique: □

Column Name	Sort Order
COUNTRY_NAME	● ASC ○ DESC ○ Default

☑ Show SQL

```
CREATE INDEX
    "HR".MyIndex
ON
    "HR"."COUNTRIES"
    (
        "COUNTRY_NAME" ASC
    )
```

3. Enter the required info in the fields, e.g. index name. The fields are database dependent so for some databases there are additional fields compared to the figure,
4. Click the **Add** button in the **Columns** area to add an index column,
5. Select the column to index from the **Column Name** drop down list,
6. Select the sort order for the index column from the **Sort Order** radio buttons,
7. Click the **Execute** button to create the index.

To remove an index column:

1. Select the column row,
2. Click the **Remove** button.

To move an index column

1. Select the column row,
2. Click the **Up** and **Down** button to move it.

5.5 Viewing Table Data

A table's data can be viewed in in various ways in the **Data** tab in its **Object View** tab.

5.5.1 Opening the Data tab

To open the Data tab for a table:

1. Locate the table in the **Databases** tab tree,
2. Double-click the table node to open its **Object View** tab,
3. Open the **Data** sub tab.

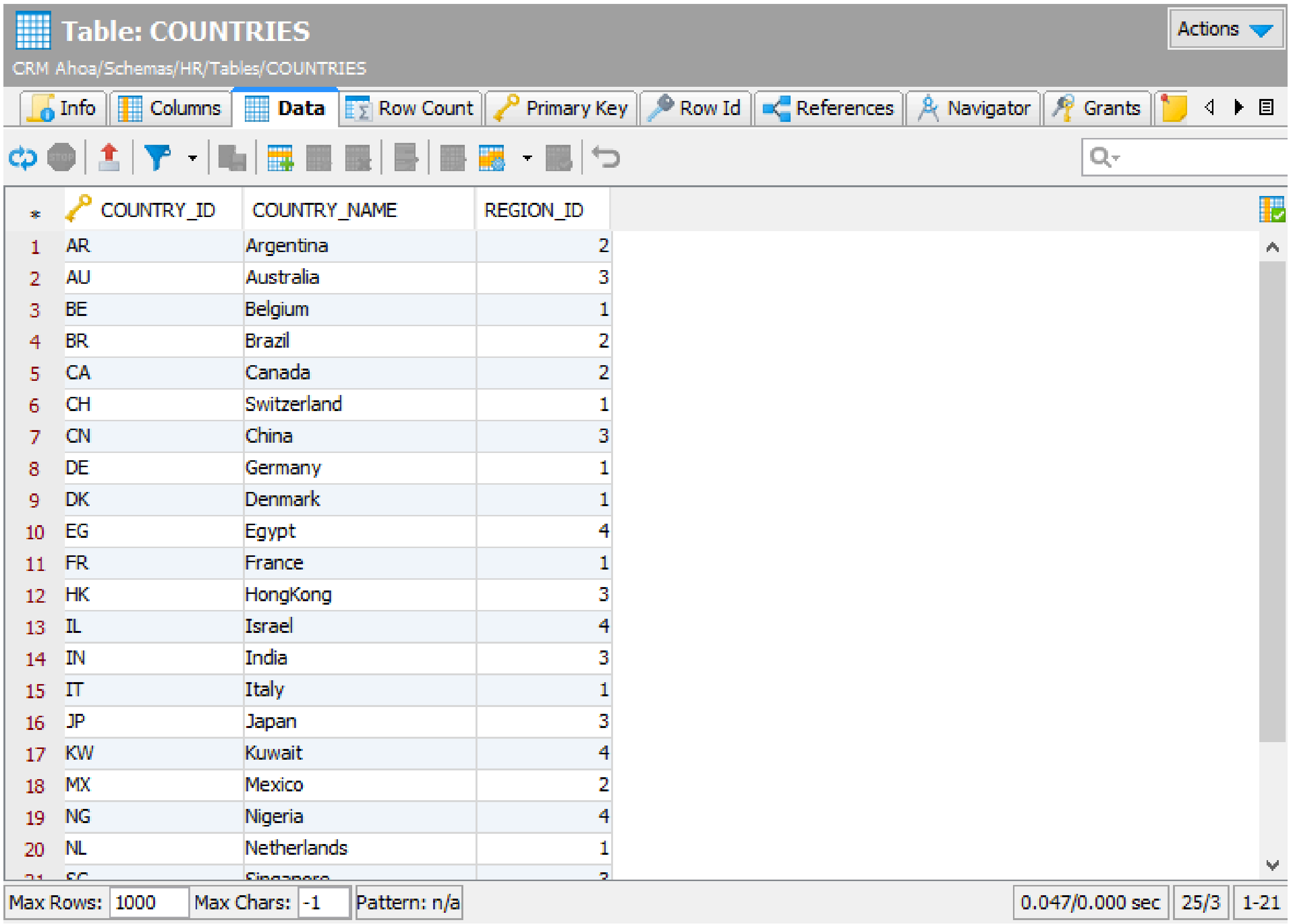


Table: COUNTRIES

CRM Ahoa/Schemas/HR/Tables/COUNTRIES

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AR	Argentina	2
2	AU	Australia	3
3	BE	Belgium	1
4	BR	Brazil	2
5	CA	Canada	2
6	CH	Switzerland	1
7	CN	China	3
8	DE	Germany	1
9	DK	Denmark	1
10	EG	Egypt	4
11	FR	France	1
12	HK	HongKong	3
13	IL	Israel	4
14	IN	India	3
15	IT	Italy	1
16	JP	Japan	3
17	KW	Kuwait	4
18	MX	Mexico	2
19	NG	Nigeria	4
20	NL	Netherlands	1
21	SG	Singapore	3

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.047/0.000 sec 25/3 1-21

Each column width is automatically resized to match the column width, including the column header, by default. You can disable this behavior in the the Tool Properties dialog, in the **Grid** category under the General tab.

If **Auto Resize Column Widths** is enabled, the **Max Column Width** setting can be used to limit the column width so that an extremely wide column does not take up all space.

In the same Tool Properties category, you can also disable **Show Grid Row Header**, i.e. the row number shown to the left of the data rows, for read-only grids such as the Data tab in the DbVisualizer Free edition and result sets from joined tables.

The column headers corresponds to the column names by default, but you can specify in the Tool Properties dialog, in the Grid category under the General tab, that you like to use the column alias instead. This is mostly useful for grids representing SQL Commander result sets, but may also be useful in the Data tab grid for some databases.

The **Data** tab contains a number of features for locating and focusing on just the data of interest as described in the following sections.

5.5.2 Sorting

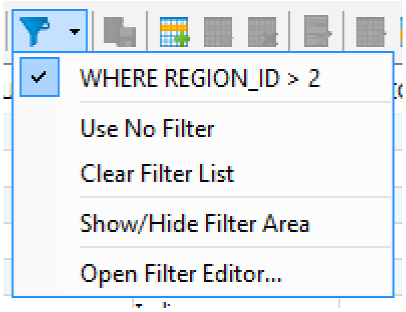
You can sort the data grid based on the values in one or more columns:

1. Click on a column header to sort the grid in ascending order on the values in that column, indicated by an up-arrow in the column header.
2. Click the same column header again to sort in descending order, indicated by a down-arrow in the column header.
3. Click a third time to show the data in the order it was received from the database. This removes the sort indicator.

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.

5.5.3 Where Filter

You can use the filter capability in the **Data** tab to limit the number of rows shown 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.

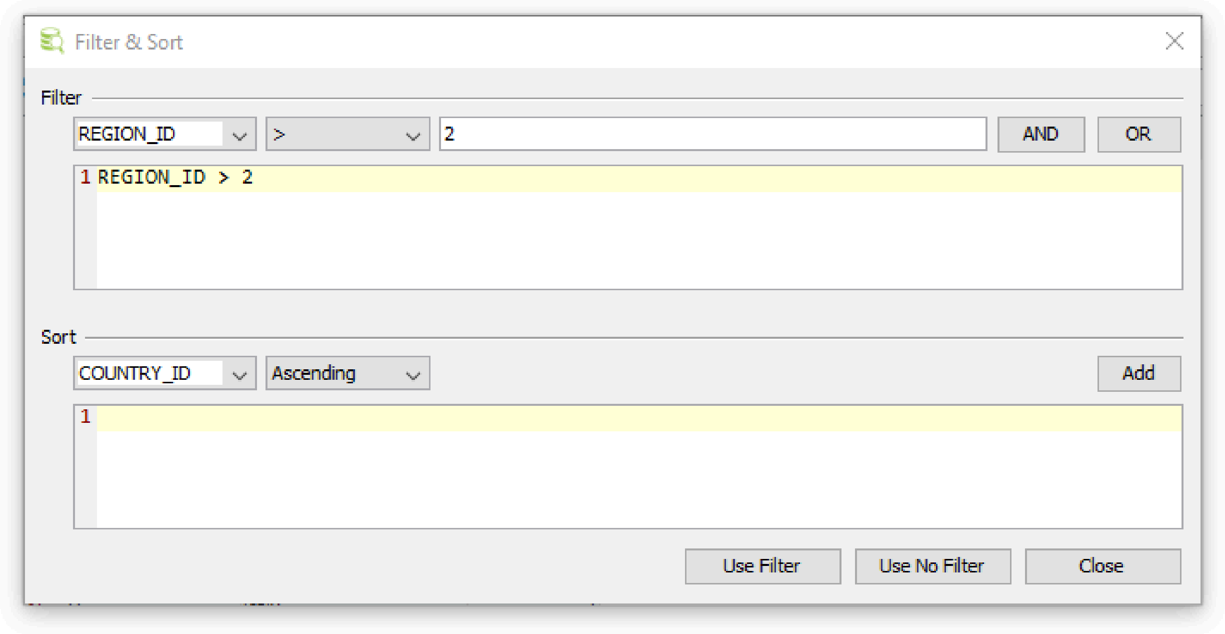


The top entries in the menu are previously used filters for the table, if any. The checkbox is selected for the filter that is currently in use. The filters are saved between DbVisualizer sessions, and you can toggle between them by selecting them from the menu. The maximum number of filters to save is specified in the Tool Properties dialog, in the **Table Data** category under the General tab.

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:

1. Select **Open Filter Editor** to launch the **Filter & Sort Configuration** dialog,

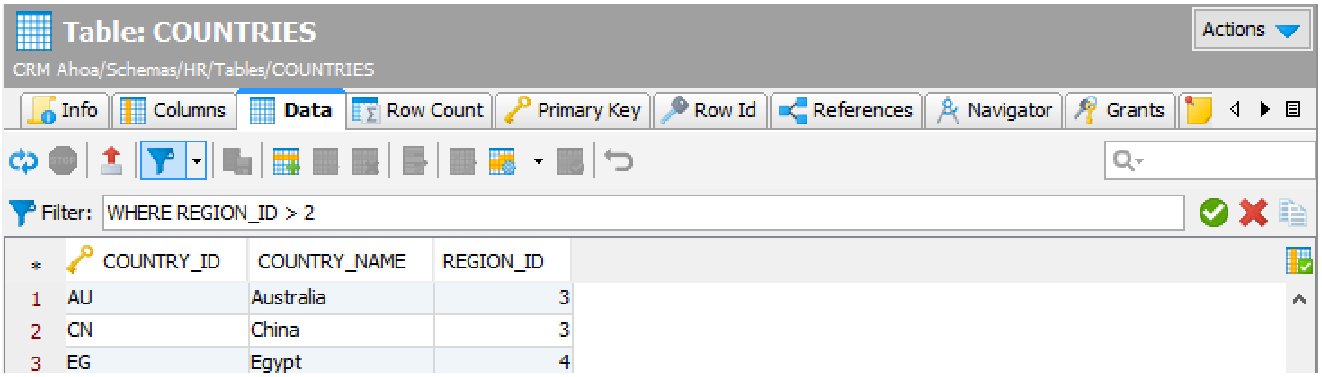


2. Select a column name, an operation and the value for the condition using the controls in the **Filter** area,
3. Add the condition to the filter by clicking the **AND** or **OR** button, and create additional conditions in the same way if needed,
4. Click the **Use Filter** button to apply the filter, save it and close the dialog, or close the dialog without applying and saving the filter by clicking the **Close** button.

You can use **Ctrl-Enter** while editing the value field to reload the grid with that single condition applied, or in the editor to reload the grid based on all filter conditions created so far.

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.

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 Filter Area** choice in the **Filter** menu to toggle the visibility of the inline filter.



Filter: WHERE REGION_ID > 2

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AU	Australia	3
2	CN	China	3
3	EG	Egypt	4

5.5.4 Column Filter

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **Column Filter** acts on the data that is already loaded in the grid, as opposed to a Where Filter (see page 84) which is used to limit the number of rows fetched from the database. With a **Column Filter**, you can easily list only those rows having a column value matching the **Column Filter.** By customizing the filter it is possible to add more complex filters such as only listing rows where the column value does not contain a certain value or substring.

A **Column Filter is added** by clicking the right part of the column header. A column having a Column Filter is indicated by the presence of a filter icon in the column header.

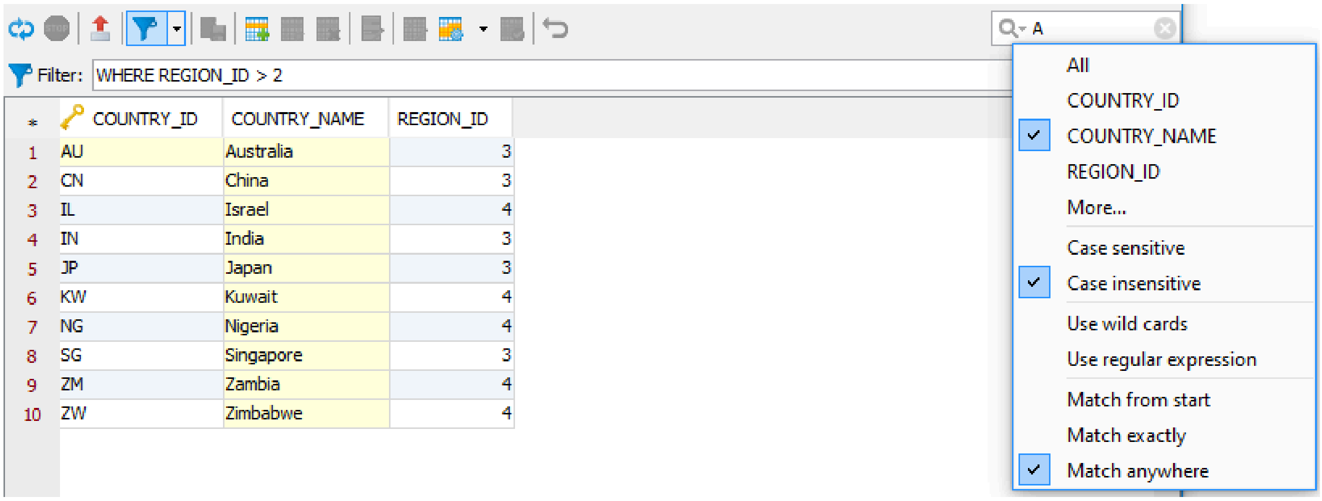
It is possible to to add **Column Filters** to multiple columns. For cases where you don't care which column has a specific value we recommend the Quick Filter (see page 86)

5.5.5 Quick Filter

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **Quick Filter** acts on the data that is already loaded in the grid, as opposed to a Where Filter (see page 84) 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.



Filter: WHERE REGION_ID > 2

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AU	Australia	3
2	CN	China	3
3	IL	Israel	4
4	IN	India	3
5	JP	Japan	3
6	KW	Kuwait	4
7	NG	Nigeria	4
8	SG	Singapore	3
9	ZM	Zambia	4
10	ZW	Zimbabwe	4

Use the Quick Filter pull-down menu (click on the down arrow next to the magnifying glass) to 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.

For the **Use wild cards** option the following characters have special meaning:

? - The question mark indicates there is zero or one of the preceding element. For example, colou?r matches both "color" and "colour".
* - The asterisk indicates there are zero or more of the preceding element. For example, ab*c matches "ac", " abc", "abbc", "abbbc", and so on.
+ - The plus sign indicates that there is one or more of the preceding element. For example, ab+c matches "abc" , "abbc", "abbbc", and so on, but not "ac".

5.5.6 Max Rows/Max Chars

DbVisualizer limits the number of rows shown in the Data tab to 1000 rows, by default. This is done to conserve memory. If this limit prevents you from seeing the data of interest, you should first consider:

1. Using a Where Filter (see page 84) to only retrieve the rows of interest instead of all rows in the table,
2. Exporting the table (see page 109) to a file

If you really need to look at more than 1000 rows, you can change the value in the **Max Rows** field in the grid status bar. Use a value of 0 or -1 to get all rows, or a specific number (e.g. 5000) to set a new limit.

Character data columns may contain very large values that use up lots of memory. If you are only interested in seeing a few characters, you can set the **Max Chars** field in the grid status bar to the number of characters you want to see.

You can define how to deal with columns that have more characters than the specified maximum in the Tool Properties dialog, in the Grid category under the General tab. You have two choices: **Truncate Values** or **Truncate Values Visually**.

- **Truncate Values** truncates the original value for the grid cell to be less then the setting of Max Chars.

⚠ 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** truncates 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.

When the grid data is limited due to either the **Max Rows** or **Max Chars** value, you get an indication about this in the rows/columns field in the grid status bar and in the corresponding limit field.

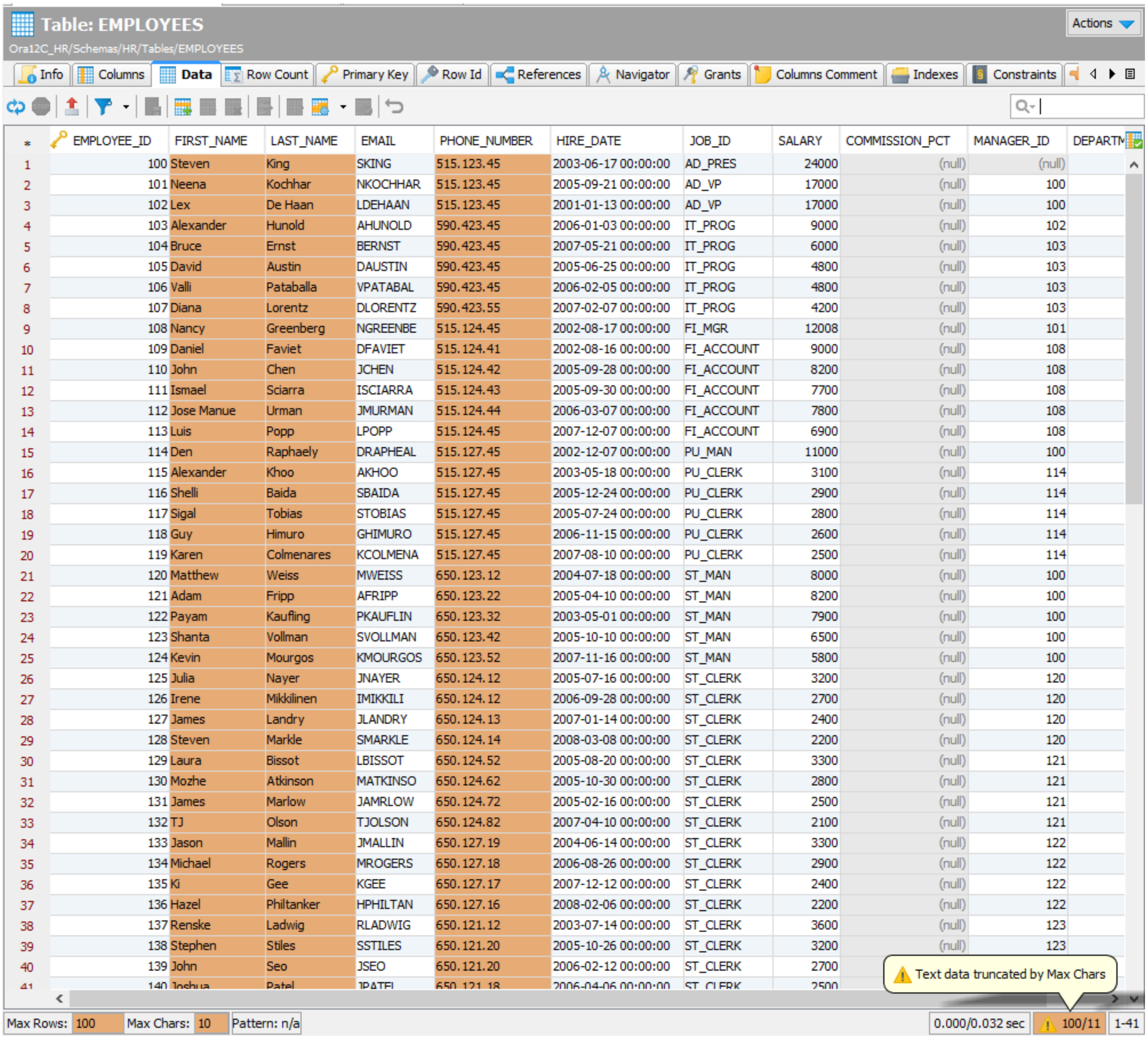


Table: EMPLOYEES

Ora12C_HR/Schemas/HR/Tables/EMPLOYEES

*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTM
1	100	Steven	King	SKING	515.123.45	2003-06-17 00:00:00	AD_PRES	24000	(null)	(null)	
2	101	Neena	Kochhar	NKOCHHAR	515.123.45	2005-09-21 00:00:00	AD_VP	17000	(null)	100	
3	102	Lex	De Haan	LDEHAAN	515.123.45	2001-01-13 00:00:00	AD_VP	17000	(null)	100	
4	103	Alexander	Hunold	AHUNOLD	590.423.45	2006-01-03 00:00:00	IT_PROG	9000	(null)	102	
5	104	Bruce	Ernst	BERNST	590.423.45	2007-05-21 00:00:00	IT_PROG	6000	(null)	103	
6	105	David	Austin	DAUSTIN	590.423.45	2005-06-25 00:00:00	IT_PROG	4800	(null)	103	
7	106	Valli	Pataballa	VPATABAL	590.423.45	2006-02-05 00:00:00	IT_PROG	4800	(null)	103	
8	107	Diana	Lorentz	DLORENTZ	590.423.55	2007-02-07 00:00:00	IT_PROG	4200	(null)	103	
9	108	Nancy	Greenberg	NGREENBE	515.124.45	2002-08-17 00:00:00	FI_MGR	12008	(null)	101	
10	109	Daniel	Faviet	DFAVIET	515.124.41	2002-08-16 00:00:00	FI_ACCOUNT	9000	(null)	108	
11	110	John	Chen	JCHEN	515.124.42	2005-09-28 00:00:00	FI_ACCOUNT	8200	(null)	108	
12	111	Ismael	Sciarra	ISCIARRA	515.124.43	2005-09-30 00:00:00	FI_ACCOUNT	7700	(null)	108	
13	112	Jose Manue	Urman	JMURMAN	515.124.44	2006-03-07 00:00:00	FI_ACCOUNT	7800	(null)	108	
14	113	Luis	Popp	LPOPP	515.124.45	2007-12-07 00:00:00	FI_ACCOUNT	6900	(null)	108	
15	114	Den	Raphaely	DRAPHEAL	515.127.45	2002-12-07 00:00:00	PU_MAN	11000	(null)	100	
16	115	Alexander	Khoo	AKHOO	515.127.45	2003-05-18 00:00:00	PU_CLERK	3100	(null)	114	
17	116	Shelli	Baida	SBAIDA	515.127.45	2005-12-24 00:00:00	PU_CLERK	2900	(null)	114	
18	117	Sigal	Tobias	STOBIAS	515.127.45	2005-07-24 00:00:00	PU_CLERK	2800	(null)	114	
19	118	Guy	Himuro	GHIMURO	515.127.45	2006-11-15 00:00:00	PU_CLERK	2600	(null)	114	
20	119	Karen	Colmenares	KCOLMENA	515.127.45	2007-08-10 00:00:00	PU_CLERK	2500	(null)	114	
21	120	Matthew	Weiss	MWEISS	650.123.12	2004-07-18 00:00:00	ST_MAN	8000	(null)	100	
22	121	Adam	Fripp	AFRIPP	650.123.22	2005-04-10 00:00:00	ST_MAN	8200	(null)	100	
23	122	Payam	Kaufling	PKAUFLIN	650.123.32	2003-05-01 00:00:00	ST_MAN	7900	(null)	100	
24	123	Shanta	Vollman	SVOLLMAN	650.123.42	2005-10-10 00:00:00	ST_MAN	6500	(null)	100	
25	124	Kevin	Mourgos	KMOURGOS	650.123.52	2007-11-16 00:00:00	ST_MAN	5800	(null)	100	
26	125	Julia	Nayer	JNAYER	650.124.12	2005-07-16 00:00:00	ST_CLERK	3200	(null)	120	
27	126	Irene	Mikkilinen	IMIKKILI	650.124.12	2006-09-28 00:00:00	ST_CLERK	2700	(null)	120	
28	127	James	Landry	JLANDRY	650.124.13	2007-01-14 00:00:00	ST_CLERK	2400	(null)	120	
29	128	Steven	Markle	SMARKLE	650.124.14	2008-03-08 00:00:00	ST_CLERK	2200	(null)	120	
30	129	Laura	Bissot	LBISSOT	650.124.52	2005-08-20 00:00:00	ST_CLERK	3300	(null)	121	
31	130	Mozhe	Atkinson	MATKINSO	650.124.62	2005-10-30 00:00:00	ST_CLERK	2800	(null)	121	
32	131	James	Marlow	JAMRLOW	650.124.72	2005-02-16 00:00:00	ST_CLERK	2500	(null)	121	
33	132	TJ	Olson	TJOLSON	650.124.82	2007-04-10 00:00:00	ST_CLERK	2100	(null)	121	
34	133	Jason	Mallin	JMALLIN	650.127.19	2004-06-14 00:00:00	ST_CLERK	3300	(null)	122	
35	134	Michael	Rogers	MROGERS	650.127.18	2006-08-26 00:00:00	ST_CLERK	2900	(null)	122	
36	135	Ki	Gee	KGEE	650.127.17	2007-12-12 00:00:00	ST_CLERK	2400	(null)	122	
37	136	Hazel	Philtanker	HPHILTAN	650.127.16	2008-02-06 00:00:00	ST_CLERK	2200	(null)	122	
38	137	Renske	Ladwig	RLADWIG	650.121.12	2003-07-14 00:00:00	ST_CLERK	3600	(null)	123	
39	138	Stephen	Stiles	SSTILES	650.121.20	2005-10-26 00:00:00	ST_CLERK	3200	(null)	123	
40	139	John	Seo	JSEO	650.121.20	2006-02-12 00:00:00	ST_CLERK	2700			
41	140	Joshua	Patel	JPATEL	650.121.18	2006-04-06 00:00:00	ST_CLERK	2500			

⚠ Text data truncated by Max Chars

Max Rows: 100 | Max Chars: 10 | Pattern: n/a | 0.000/0.032 sec | ⚠ 100/11 | 1-41

Along with the highlighted field, a warning pops up close to the field. You can disable this behavior in the Tool Properties dialog, in the **Grid** category under the General tab.

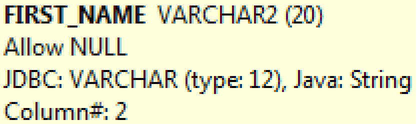
5.5.7 Max Rows at First Display

By default, opening the Data tab for a table loads all rows, unless there is a Max Rows limit. If you have very large tables and don't want to risk memory issues if you accidentally open the Data tab and have no Max Rows limit, you can specify a **Max Rows at First Display** limit. You do this in the Tool Properties dialog, in the **Table Data** category under the General tab.

The default is -1, which means no limit. If you set it to a positive number, only the specified number of rows are loaded when the Data tab is first opened for a table. To load more rows, click the **Reload** button in the Data tab toolbar.

5.5.8 Column Header Tooltips

The column header tooltip shows data type information about the column. To see the tooltip, let the mouse hover over the column header. The tooltip pops up in about a second.



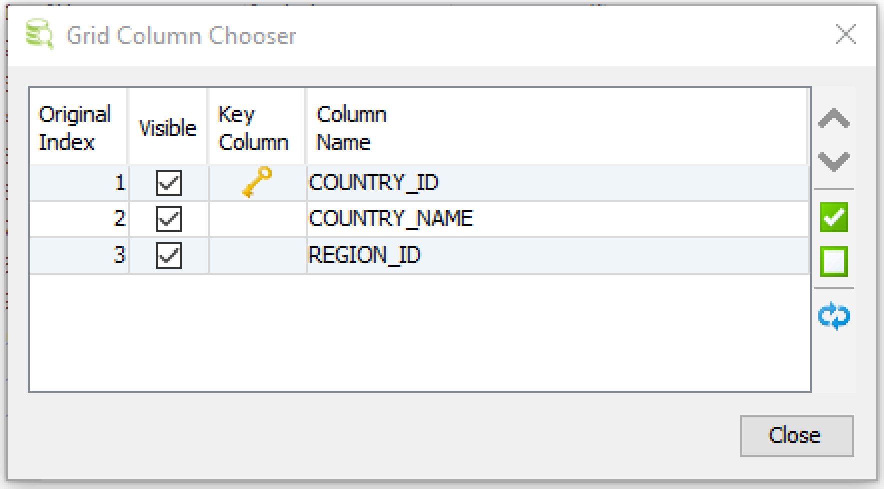
FIRST_NAME VARCHAR2 (20)
Allow NULL
JDBC: VARCHAR (type: 12), Java: String
Column#: 2

5.5.9 Highlight Primary Key Columns

By default, a Primary Key column is shown with an icon in the column header. You can disable this in the Tool Properties dialog, in the **Table Data** category under the General tab.

5.5.10 Show Only Some Columns

The **Grid Column Chooser** dialog controls which columns you want to appear in a grid. Open the dialog by clicking the button above the vertical scrollbar in the grid.



Grid Column Chooser

Original Index	Visible	Key Column	Column Name
1	☑	🔑	COUNTRY_ID
2	☑		COUNTRY_NAME
3	☑		REGION_ID

The **Grid Column Chooser** 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).

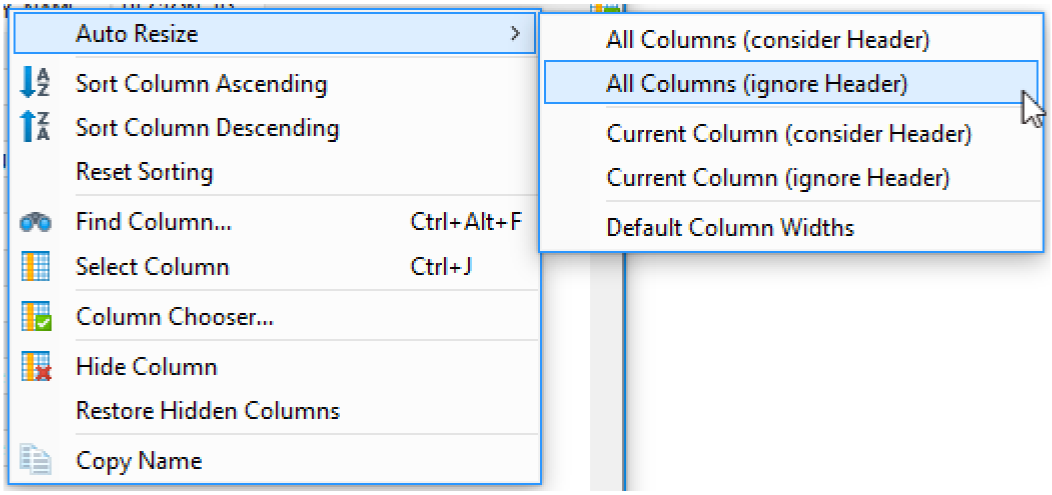
If you want to revert your changes, you can click on the **Default Layout** button to reset the grid, i.e., making all column visible and put them in their default locations.

⚠ 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.

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.

5.5.11 Auto Resize Columns

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



5.5.12 Right-Click Menu Operations

The right-click menu for the grid contains a lot of operations for working with the data without changing it. In addition to the common select, copy, and print operations, some operations that may require a bit of an explanation are:

Operation	Description
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.
Copy Selection as Formatted Text	Copy all selected cells including column header in fixed width columns onto the system clipboard.
Copy Selection as Comma List	Copy all selected cells, with cells from the same column formatted as comma separated values
Copy Selection as IN Clause	Copy all selected cells, with cells from the same column as formatted as an IN clause, e.g. `col1 IN ('val1', 'val2', 'val3')`
Save Selected Cell	Save the value of the selected cell to a file, selected with a file chooser dialog
Compare	Compare (see page 122) the data in this grid to the data in other open grids.
Compare Selected Cells	Compare the data in the two cells as text
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**.
Show in Navigator	Open the Navigator (see page 124) tab with the current selections and sorting.
Describe Data	Show detailed information about the columns in the grid.
Aggregation Data for Selection	Displays aggregation data for the current selection. Read more in Aggregation Data for Selection (see page 93) below.
Generate Filter & Sort	The operations in this submenu helps you create common Where Filters (see page 84).
Create Row Count Data Monitor	Creates a monitor (see page 93) for tracking the row count in the table over time.
Create Row Count DIff Data Monitor	Creates a monitor (see page 93) for tracking the number of added or removed rows in the table over time.

There are also a set of operations for generating SQL statements based on the current selection. Choosing any of these creates the appropriate SQL and then switches the view to a new **SQL Commander** tab. You must use these operations to edit table data in the DbVisualizer Free edition. With the DbVisualizer Pro edition, you can instead use inline and form based editing (see page 94).

Operation	SQL Example

Script: SELECT ALL	`select *`
	`from HR.COUNTRIES`

Script: SELECT ALL WHERE	`select *`
	`from HR.COUNTRIES`
	`where COUNTRY_NAME = 'Brazil'`

Script: SELECT ALL WITH FILTER	`select *`
	`from HR.COUNTRIES`
	`where REGION_ID = 1 // If this is the filter, see above`

Script: INSERT INTO TABLE	`insert into HR.COUNTRIES`
	`(COUNTRY_ID, COUNTRY_NAME, REGION_ID)`
	`values ('', '', )`

Script: INSERT COPY INTO TABLE	`insert into HR.COUNTRIES`
	`(COUNTRY_ID, COUNTRY_NAME, REGION_ID)`
	`values ('BR', 'Brazil', 2)`

Script: UPDATE WHERE	`update HR.COUNTRIES`
	`set COUNTRY_ID = 'BR',`
	`COUNTRY_NAME = 'Brazil',`
	`REGION_ID = 2`
	`where COUNTRY_NAME = 'Brazil'`

Script: DELETE WHERE	`delete from HR.COUNTRIES`
	`where COUNTRY_NAME = 'Brazil'`

You can generate SQL with either static values as they appear in the grid, or with DbVisualizer variables (see page 227). 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 are used in the generated SQL statements by default. You can disable the **Include Variables in SQL** setting in the Tool Properties dialog, in the **Table Data** category under the General tab, to use literal values are instead.

Here is an example of the SQL generated for **Script: SELECT ALL WHERE** with the **Include Variables in SQL** setting enabled, assuming the table is named HR.COUNTRIES and has a column named COUNTRY_NAME with the value 'Brazil' on the selected row:

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

And here is the same example with the **Include Variables in SQL** setting disabled:

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

5.5.13 Creating Monitors

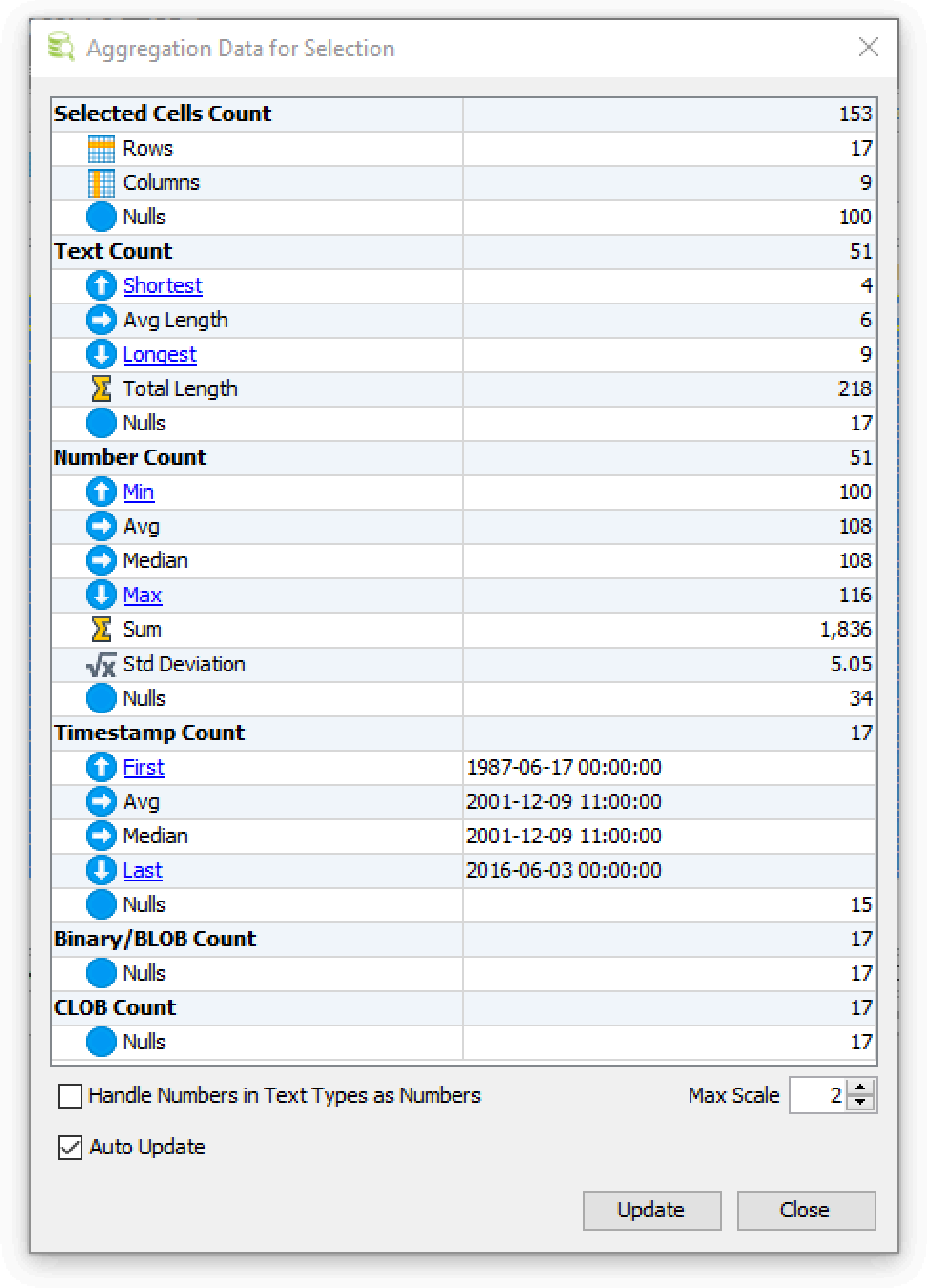
A monitor in DbVisualizer is an SQL query executed at a specified frequency so you can track changes in data over time. The result can be viewed either as a grid or a graph. The right-click menu for the grid in the Data tab contains operations for creating two common types of monitors for the table: a **Row Count Data** monitor or a **Row Count Diff Data** monitor. The first tracks the number of rows in the table over time and the second tracks the number of added or removed rows over time. Please read more about monitors in Monitoring Data Changes (see page 267).

5.5.14 Aggregation Data for Selection

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **Aggregation Data for Selection** feature presents aggregation data organized per data type on the current selection in a grid. 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	153
Rows	17
Columns	9
Nulls	100
Text Count	51
Shortest	4
Avg Length	6
Longest	9
Total Length	218
Nulls	17
Number Count	51
Min	100
Avg	108
Median	108
Max	116
Sum	1,836
Std Deviation	5.05
Nulls	34
Timestamp Count	17
First	1987-06-17 00:00:00
Avg	2001-12-09 11:00:00
Median	2001-12-09 11:00:00
Last	2016-06-03 00:00:00
Nulls	15
Binary/BLOB Count	17
Nulls	17
CLOB Count	17
Nulls	17

□ Handle Numbers in Text Types as Numbers   Max Scale 2

☑ Auto Update

With **Auto Update** checked, the data is updated automatically when you change the selection in the grid. 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. The **Handle Number Values in Text Types as Numbers** setting simply treats all valid numbers in text data types as numbers and include them in the **Number Count** summary.

5.6 Editing Table Data

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

With the DbVisualizer Pro edition, you can edit table data in the **Data** tab grid; 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, PDF, XML, HEX, etc.

- Opening the Data tab (see page 95)
- Editing Data in the Grid (see page 96)
- Copy/Paste (see page 97)
- Updates and Deletes Must Match Only One Table Row (see page 99)
- Key Column(s) Chooser (see page 100)
- Editing Multiple Rows (see page 101)
- Data Type checking (see page 101)
- New Line and Carriage Return (see page 101)
- Using the Cell Editor/Viewer (see page 102)
- Using the Form Editor/Viewer (see page 103)
- Preview Changes (see page 105)
- Editing Binary/BLOB and CLOB Data (see page 105)

5.6.1 Opening the Data tab

To open the **Data** tab for a table:

1. Locate the table in the **Databases** tab tree,
2. Double-click the table node to open its **Object View** tab,
3. Open the **Data** sub tab.

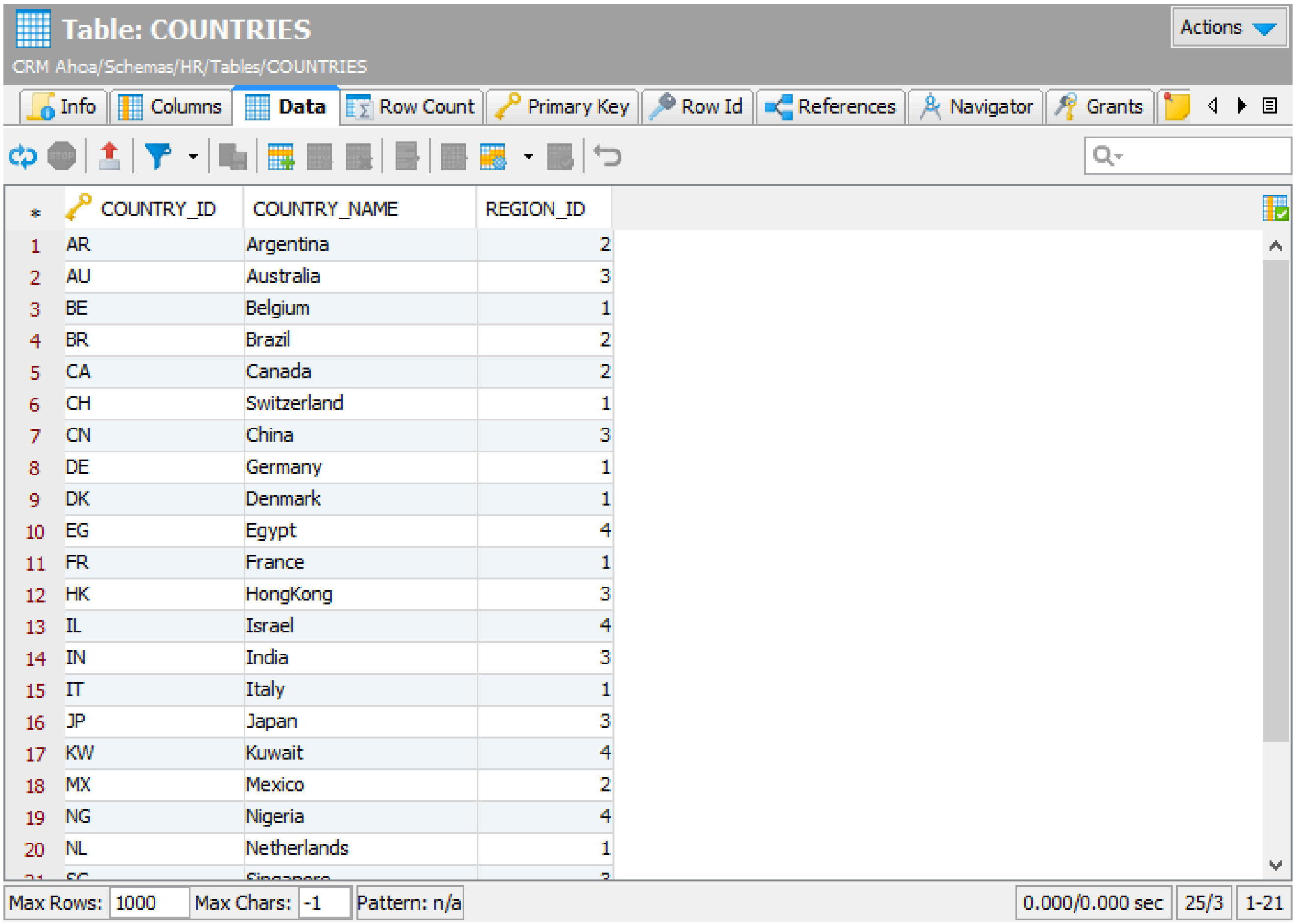


Table: COUNTRIES

CRM Ahoa/Schemas/HR/Tables/COUNTRIES

*	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AR	Argentina	2
2	AU	Australia	3
3	BE	Belgium	1
4	BR	Brazil	2
5	CA	Canada	2
6	CH	Switzerland	1
7	CN	China	3
8	DE	Germany	1
9	DK	Denmark	1
10	EG	Egypt	4
11	FR	France	1
12	HK	HongKong	3
13	IL	Israel	4
14	IN	India	3
15	IT	Italy	1
16	JP	Japan	3
17	KW	Kuwait	4
18	MX	Mexico	2
19	NG	Nigeria	4
20	NL	Netherlands	1
21	SG	Singapore	3

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.000/0.000 sec 25/3 1-21

Each column width is automatically resized to match the column width, including the column header, by default. You can disable this behavior in the the Tool Properties dialog, in the **Grid** category under the General tab.

If **Auto Resize Column Widths** is enabled, the **Max Column Width** setting can be used to limit the column width so that an extremely wide column does not take up all space.

5.6.2 Editing Data in the Grid

To edit a column value:

1. Select the column cell,
2. Type the new value, or double click to edit the current value,
3. Click the **Save** toolbar button to update the database.

You can also use the **Set Selected Cells** drop down menu to set a number of column values to things like null or the current date or time.

To add a new row:

1. Select the row above where you want to insert the new row,

2. Click the **Add Row** toolbar button,
3. Enter values for the columns,
4. Click the **Save** toolbar button to update the database.

To duplicate a row:

1. Select the row you want to duplicate,
2. Click the **Duplicate Row** toolbar button,
3. Edit at least the key column(s) value(s),
4. Click the **Save** toolbar button to update the database.

To delete one or more rows:

1. Select the rows to delete,
2. Click the **Delete Rows** toolbar button,
3. Click the **Save** toolbar button to update the database.

If you change your mind, you easily can undo edits:

1. Select the cell(s) you want to revert,
2. Click the **Undo** toolbar button.

Reverting all cells in a row that are marked as **Insert** or **Duplicate** removes 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.

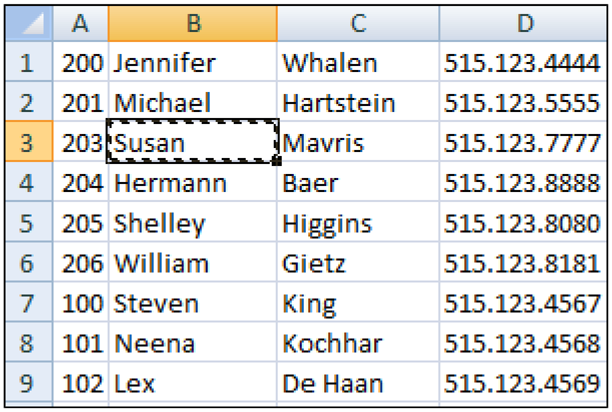
5.6.3 Copy/Paste

You can copy selected cell values with the **Copy Selection** right-click menu choice or the corresponding key binding (**Ctrl-C** or **Command-C** by default). The data on the clipboard may then be pasted either into DbVisualizer or any external application. The column and newline delimiter used for copy and paste operations in the grid editor are defined by the **Copy Grid Cells in CSV Format** settings in the **Grid** category in the Tool Properties dialog, under the General tab. The default setting are sufficient for most uses.

The grid editor supports pasting data from the major spreadsheet applications, such as Excel and OpenOffice. The grid editor supports pasting single data as well as block of data. Copy/paste of binary data is transparent between grids or in the same grid. Binary files may also be copied in an external application and pasted in a cell in DbVisualizer (target cell must be a binary type).

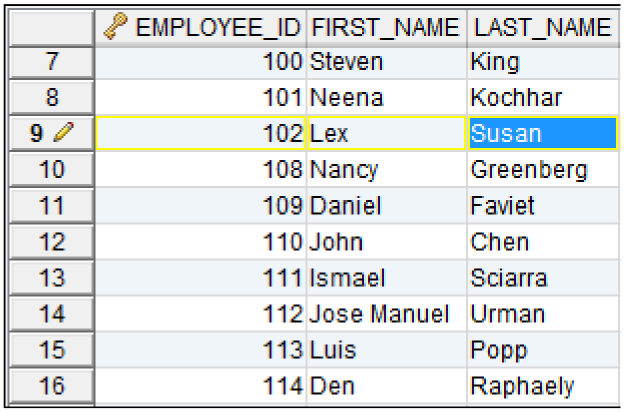
Copy from spreadsheet		Paste into DbVisualizer grid
A single cell is copied	➡	Paste into selected target cell

Copy from spreadsheet



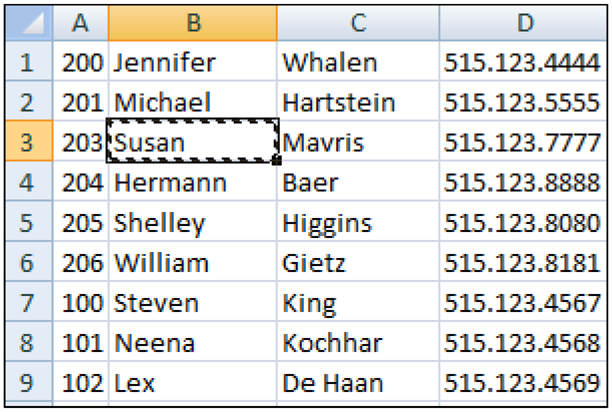
	A	B	C	D
1	200	Jennifer	Whalen	515.123.4444
2	201	Michael	Hartstein	515.123.5555
3	203	Susan	Mavris	515.123.7777
4	204	Hermann	Baer	515.123.8888
5	205	Shelley	Higgins	515.123.8080
6	206	William	Gietz	515.123.8181
7	100	Steven	King	515.123.4567
8	101	Neena	Kochhar	515.123.4568
9	102	Lex	De Haan	515.123.4569

Paste into DbVisualizer grid



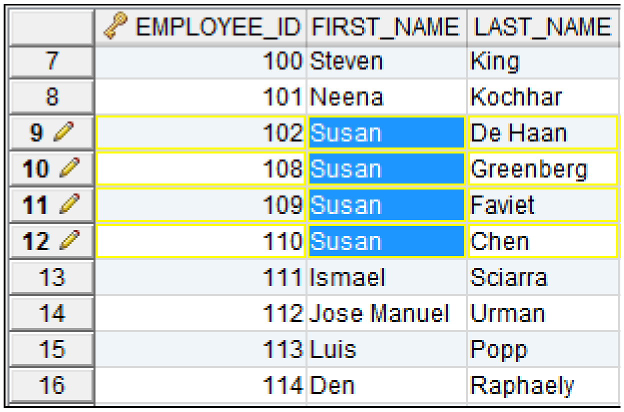
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
7	100	Steven	King
8	101	Neena	Kochhar
9	102	Lex	Susan
10	108	Nancy	Greenberg
11	109	Daniel	Faviet
12	110	John	Chen
13	111	Ismael	Sciarra
14	112	Jose Manuel	Urman
15	113	Luis	Popp
16	114	Den	Raphaely

A single cell is copied



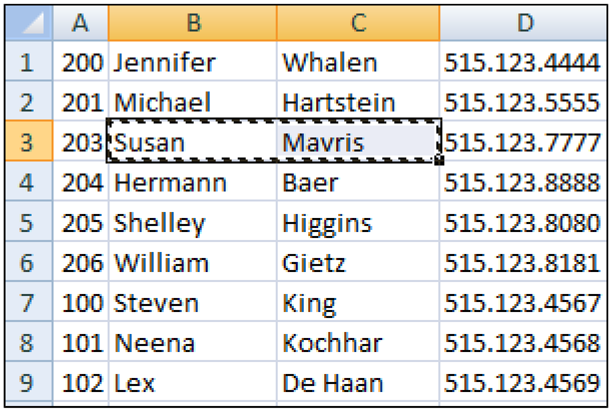
	A	B	C	D
1	200	Jennifer	Whalen	515.123.4444
2	201	Michael	Hartstein	515.123.5555
3	203	Susan	Mavris	515.123.7777
4	204	Hermann	Baer	515.123.8888
5	205	Shelley	Higgins	515.123.8080
6	206	William	Gietz	515.123.8181
7	100	Steven	King	515.123.4567
8	101	Neena	Kochhar	515.123.4568
9	102	Lex	De Haan	515.123.4569

Paste and fill the single column target selection



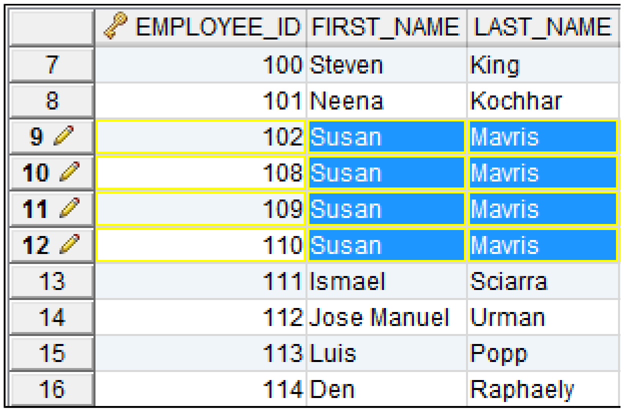
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
7	100	Steven	King
8	101	Neena	Kochhar
9	102	Susan	De Haan
10	108	Susan	Greenberg
11	109	Susan	Faviet
12	110	Susan	Chen
13	111	Ismael	Sciarra
14	112	Jose Manuel	Urman
15	113	Luis	Popp
16	114	Den	Raphaely

Multiple cells in a single row is copied



	A	B	C	D
1	200	Jennifer	Whalen	515.123.4444
2	201	Michael	Hartstein	515.123.5555
3	203	Susan	Mavris	515.123.7777
4	204	Hermann	Baer	515.123.8888
5	205	Shelley	Higgins	515.123.8080
6	206	William	Gietz	515.123.8181
7	100	Steven	King	515.123.4567
8	101	Neena	Kochhar	515.123.4568
9	102	Lex	De Haan	515.123.4569

Paste and fill the target selection



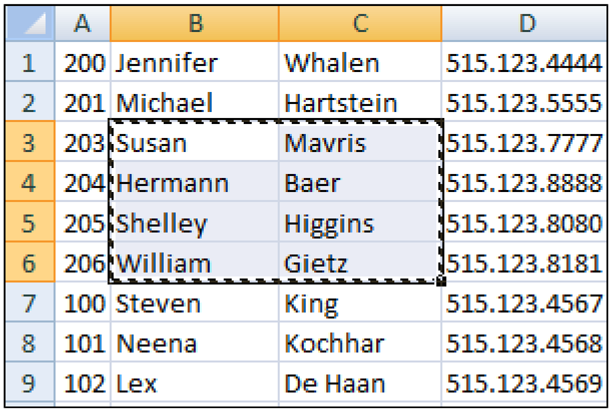
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
7	100	Steven	King
8	101	Neena	Kochhar
9	102	Susan	Mavris
10	108	Susan	Mavris
11	109	Susan	Mavris
12	110	Susan	Mavris
13	111	Ismael	Sciarra
14	112	Jose Manuel	Urman
15	113	Luis	Popp
16	114	Den	Raphaely

A block of cells is copied

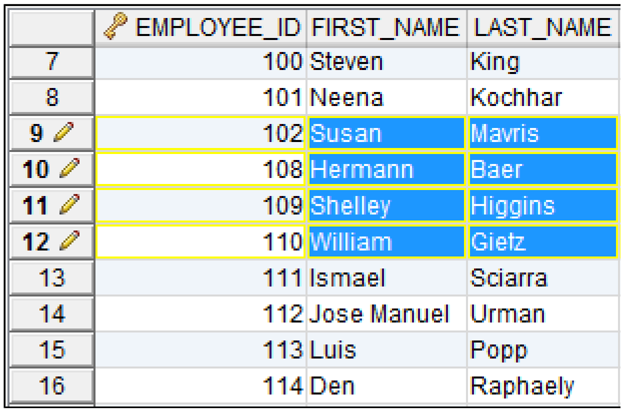
The block is pasted into the selected region

Copy from spreadsheet

Paste into DbVisualizer grid

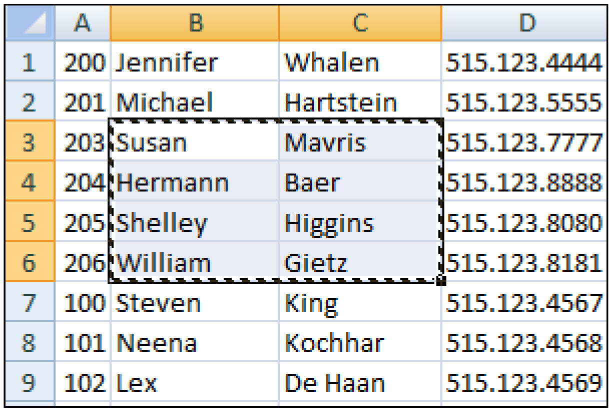


	A	B	C	D
1	200	Jennifer	Whalen	515.123.4444
2	201	Michael	Hartstein	515.123.5555
3	203	Susan	Mavris	515.123.7777
4	204	Hermann	Baer	515.123.8888
5	205	Shelley	Higgins	515.123.8080
6	206	William	Gietz	515.123.8181
7	100	Steven	King	515.123.4567
8	101	Neena	Kochhar	515.123.4568
9	102	Lex	De Haan	515.123.4569



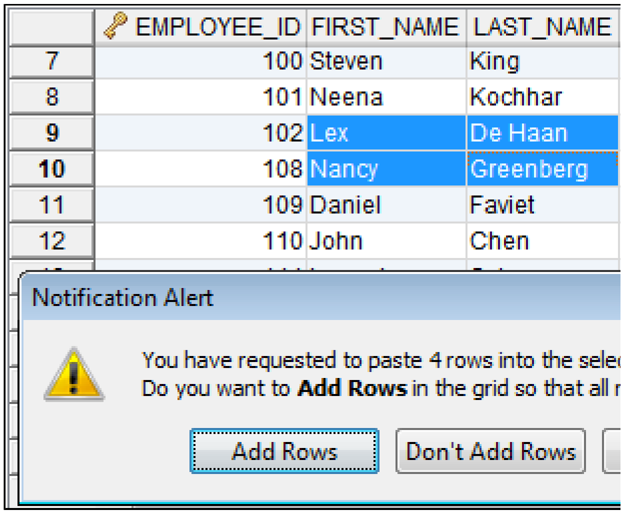
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
7	100	Steven	King
8	101	Neena	Kochhar
9	102	Susan	Mavris
10	108	Hermann	Baer
11	109	Shelley	Higgins
12	110	William	Gietz
13	111	Ismael	Sciarra
14	112	Jose Manuel	Urman
15	113	Luis	Popp
16	114	Den	Raphaely

A block of cells is copied



	A	B	C	D
1	200	Jennifer	Whalen	515.123.4444
2	201	Michael	Hartstein	515.123.5555
3	203	Susan	Mavris	515.123.7777
4	204	Hermann	Baer	515.123.8888
5	205	Shelley	Higgins	515.123.8080
6	206	William	Gietz	515.123.8181
7	100	Steven	King	515.123.4567
8	101	Neena	Kochhar	515.123.4568
9	102	Lex	De Haan	515.123.4569

The block is pasted into a non equal number of targe



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
7	100	Steven	King
8	101	Neena	Kochhar
9	102	Lex	De Haan
10	108	Nancy	Greenberg
11	109	Daniel	Faviet
12	110	John	Chen

Notification Alert

You have requested to paste 4 rows into the sele
Do you want to **Add Rows** in the grid so that all r

5.6.4 Updates and Deletes Must Match Only One Table Row

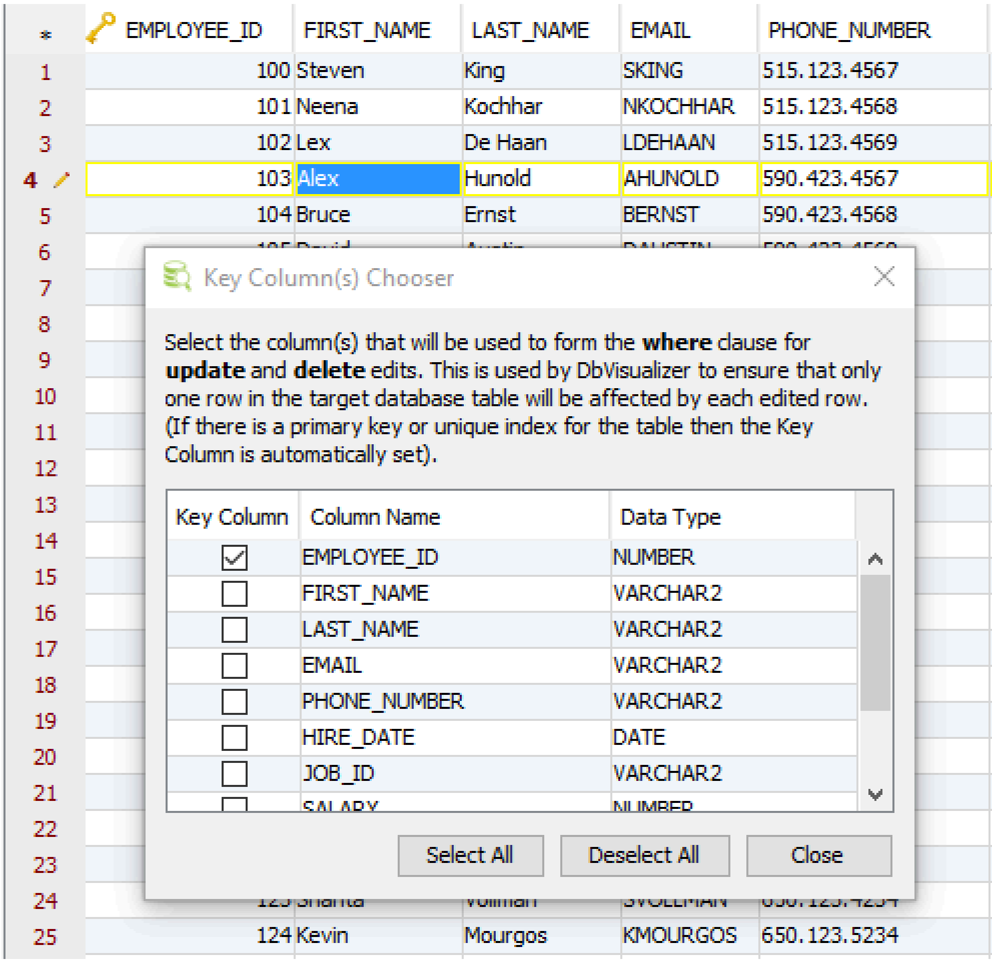
When you update or delete rows, DbVisualizer ensures that only one row in the table will be affected. This is to prevent changes in one row to silently affect data in other rows. DbVisualizer uses the following strategies to determine the uniqueness of the edited row:

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 uses 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 checks if there is a unique index. If there are several unique indexes, DbVisualizer picks 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.

5.6.5 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 checks that there is a way to identify unique rows in the table. If it cannot be accomplished, the following window is displayed.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER
1	100	Steven	King	SKING	515.123.4567
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568
3	102	Lex	De Haan	LDEHAAN	515.123.4569
4	103	Alex	Hunold	AHUNOLD	590.423.4567
5	104	Bruce	Ernst	BERNST	590.423.4568
6	105	David	Austin	DAUSTIN	590.423.4569
24	123	Shanta	Vollman	SVOLLMAN	650.123.4234
25	124	Kevin	Mourgos	KMOURGOS	650.123.5234

Key Column(s) Chooser

Select the column(s) that will be used to form the **where** clause for **update** and **delete** edits. This is used by DbVisualizer to ensure that only one row in the target database table will be affected by each edited row. (If there is a primary key or unique index for the table then the Key Column is automatically set).

Key Column	Column Name	Data Type
☑	EMPLOYEE_ID	NUMBER
□	FIRST_NAME	VARCHAR2
□	LAST_NAME	VARCHAR2
□	EMAIL	VARCHAR2
□	PHONE_NUMBER	VARCHAR2
□	HIRE_DATE	DATE
□	JOB_ID	VARCHAR2
□	SALARY	NUMBER

The key column chooser can also 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, an error dialog is displayed and the editing state is kept for that row in the grid editor.

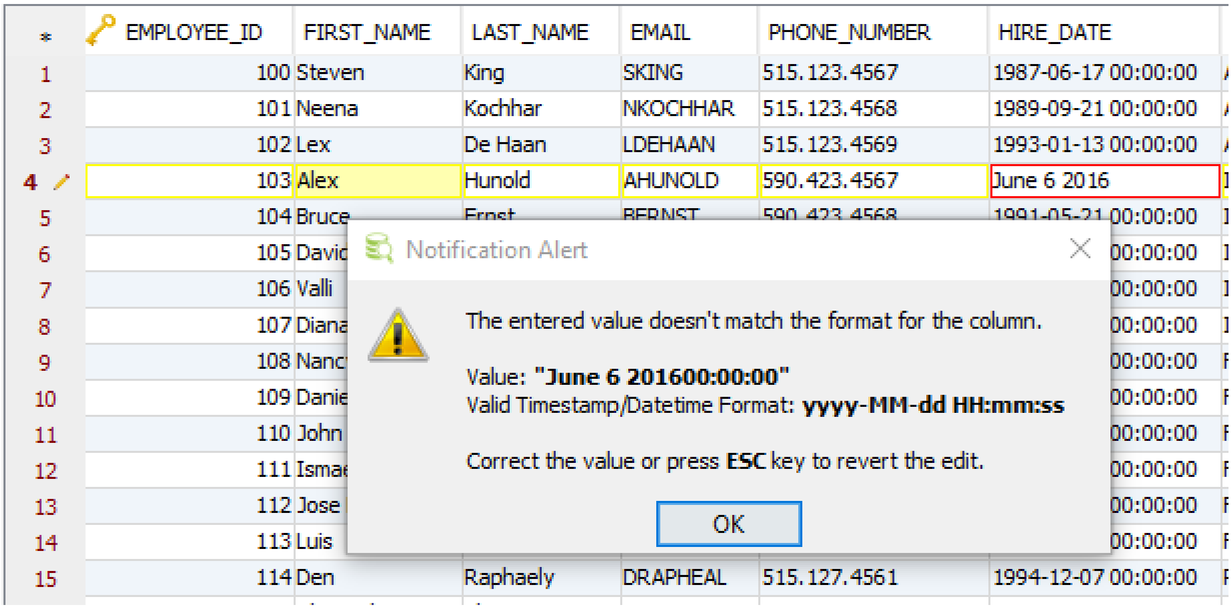
5.6.6 Editing Multiple Rows

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)

5.6.7 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.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	
1	100	Steven	King	SKING	515.123.4567	1987-06-17 00:00:00	A
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21 00:00:00	A
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00	A
4 ✎	103	Alex	Hunold	AHUNOLD	590.423.4567	June 6 2016	I
5	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	I
6	105	Davic				00:00:00	I
7	106	Valli				00:00:00	I
8	107	Diana				00:00:00	I
9	108	Nanc				00:00:00	F
10	109	Danie				00:00:00	F
11	110	John				00:00:00	F
12	111	Ismae				00:00:00	F
13	112	Jose				00:00:00	F
14	113	Luis				00:00:00	F
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07 00:00:00	F

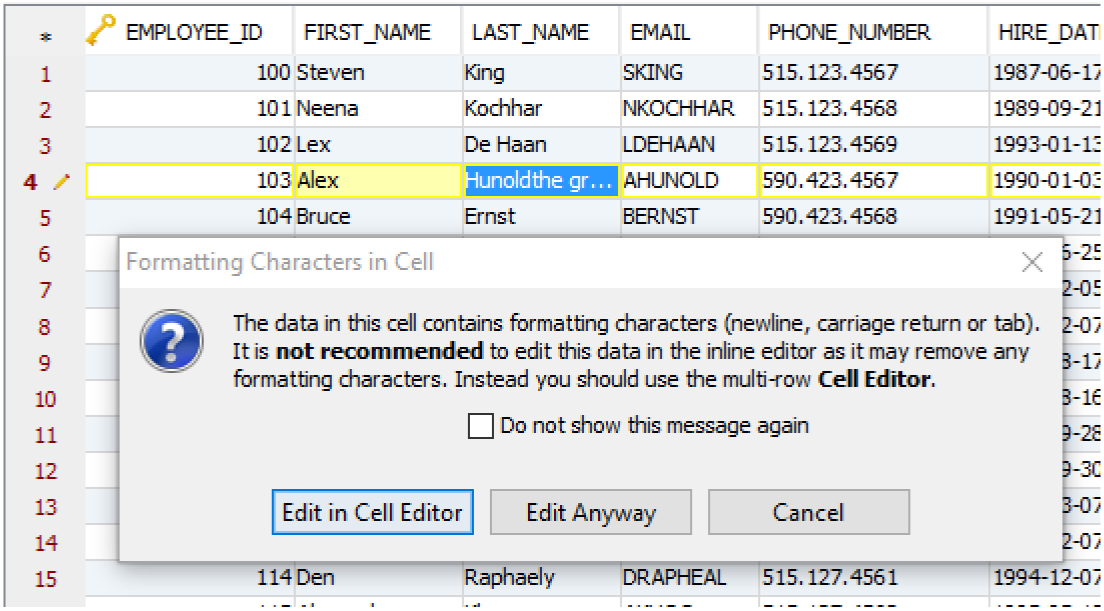
The entered value doesn't match the format for the column.

Value: **"June 6 201600:00:00"**
Valid Timestamp/Datetime Format: **yyyy-MM-dd HH:mm:ss**

Correct the value or press **ESC** key to revert the edit.

5.6.8 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:



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DAT
1	100	Steven	King	SKING	515.123.4567	1987-06-17
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13
4	103	Alex	Hunoldthe gr…	AHUNOLD	590.423.4567	1990-01-03
5	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21
6						5-25
7						2-05
8						2-07
9						3-17
10						3-16
11						9-28
12						9-30
13						3-07
14						2-07
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	1994-12-07

Formatting Characters in Cell

The data in this cell contains formatting characters (newline, carriage return or tab). It is **not recommended** to edit this data in the inline editor as it may remove any formatting characters. Instead you should use the multi-row **Cell Editor**.

☐ Do not show this message again

You may chose to edit the value in the Cell Editor (see page 102), which we recommend, as the control characters will then be preserved. Alternatively, you can edit the value in the grid anyway but you then risk loosing the control characters.

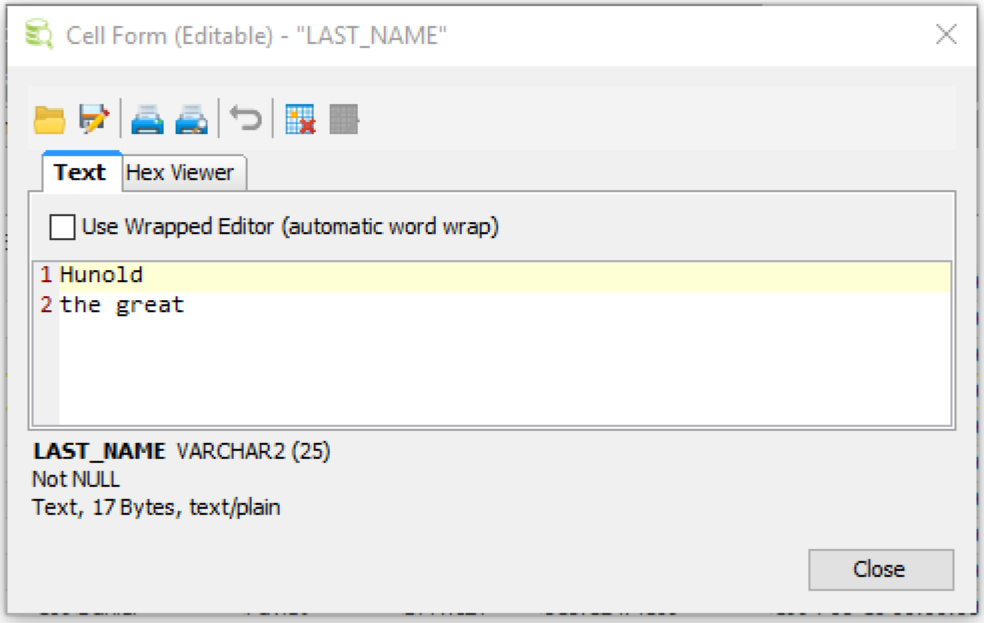
5.6.9 Using the Cell Editor/Viewer

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 also allow you to save the 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.



□ Use Wrapped Editor (automatic word wrap)

```
Hunold
the great
```

LAST_NAME VARCHAR2 (25)
Not NULL
Text, 17 Bytes, text/plain

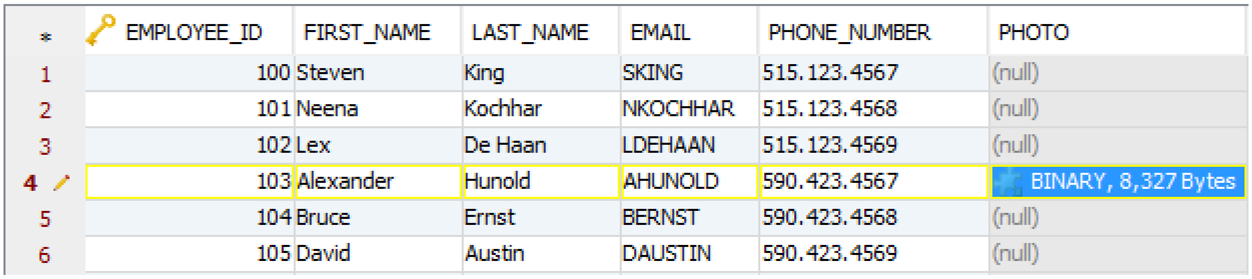
5.6.10 Using the Form Editor/Viewer

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.

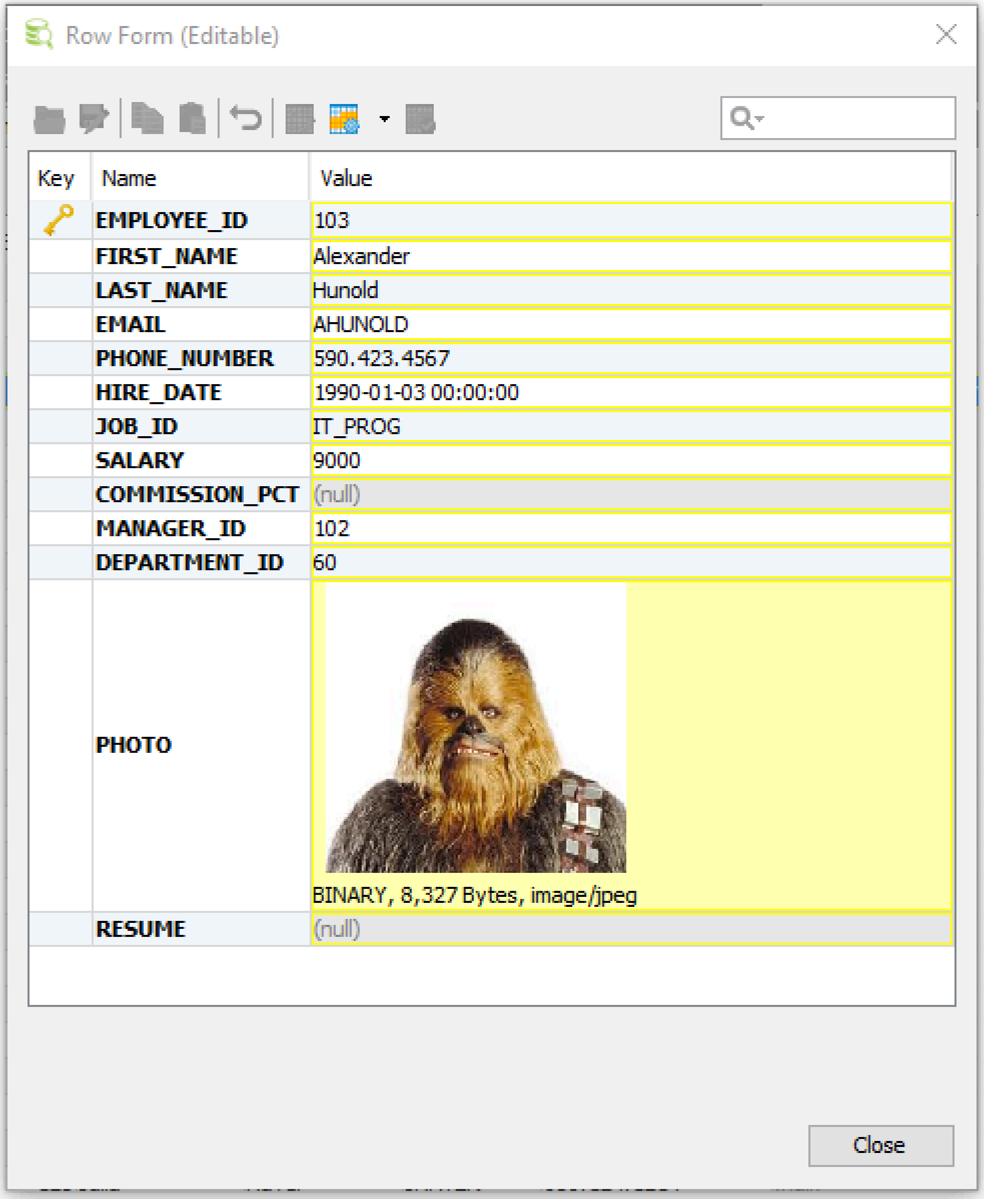
The form editor "rotate" the data in one row and presents it as a vertical form with the column name as a label. All edits made in the form editor are reflected in the grid with the edited state icon being updated along with new values. Saving edits in the database is always done with the Save button in the grid editor toolbar, just as for data edited directly in the grid.

Open the form editor via the **Edit Row in Form** right-click menu choice, via the corresponding button in the toolbar or by double-clicking the row number header.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	PHOTO
1	100	Steven	King	SKING	515.123.4567	(null)
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	(null)
3	102	Lex	De Haan	LDEHAAN	515.123.4569	(null)
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	BINARY, 8,327 Bytes
5	104	Bruce	Ernst	BERNST	590.423.4568	(null)
6	105	David	Austin	DAUSTIN	590.423.4569	(null)

The same row looks like this in the row form window:



Key	Name	Value
🔑	EMPLOYEE_ID	103
	FIRST_NAME	Alexander
	LAST_NAME	Hunold
	EMAIL	AHUNOLD
	PHONE_NUMBER	590.423.4567
	HIRE_DATE	1990-01-03 00:00:00
	JOB_ID	IT_PROG
	SALARY	9000
	COMMISSION_PCT	(null)
	MANAGER_ID	102
	DEPARTMENT_ID	60
	PHOTO	BINARY, 8,327 Bytes, image/jpeg
	RESUME	(null)

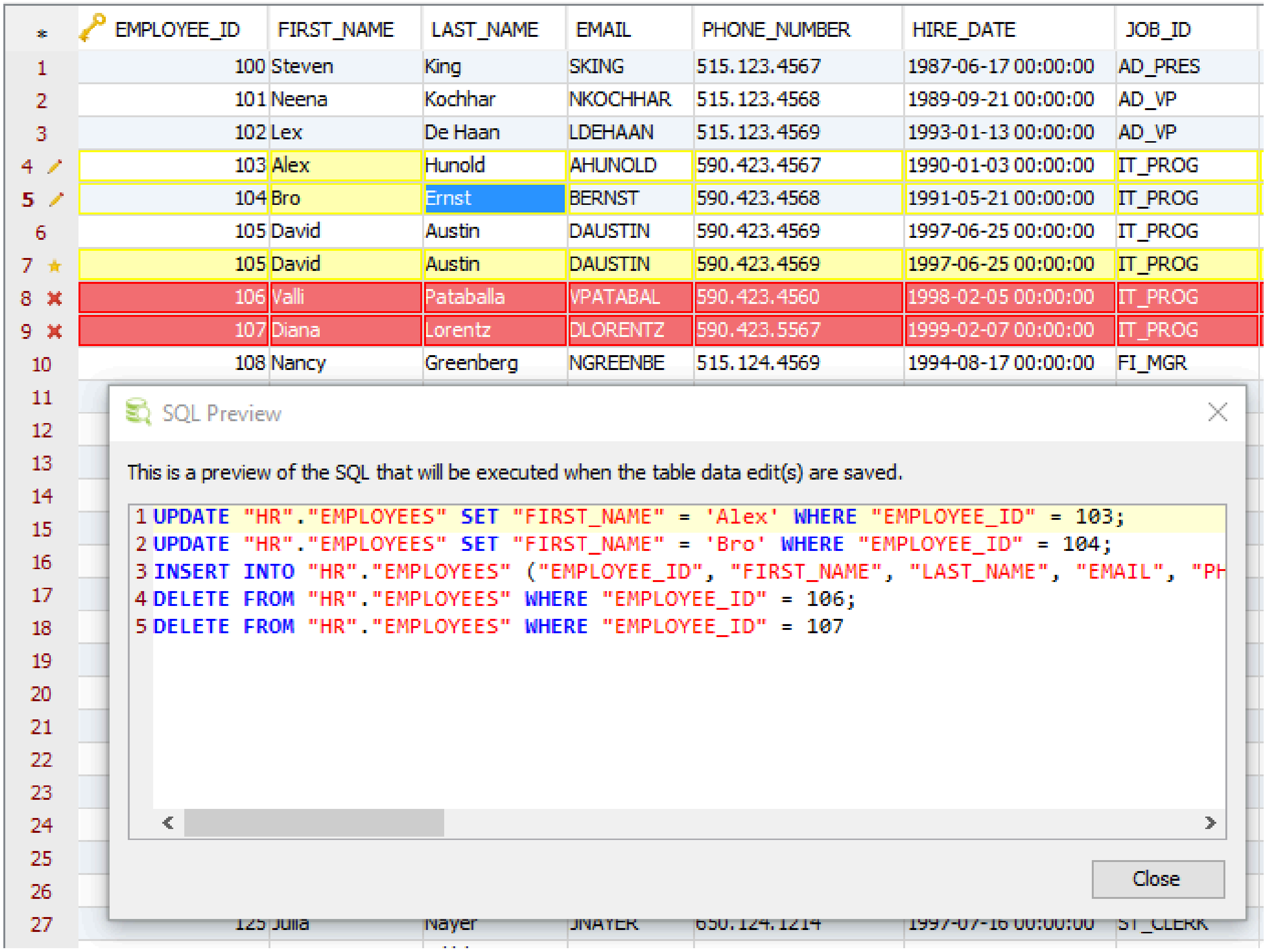
The **Key** field contains an icon for primary key columns and the **Name** field corresponds to the column name in the grid. None of **Key** or the **Name** fields can be edited. You can edit the values in the form in the same way as you edit values in the grid editor.

The form viewer presents images as thumbnails. The size of these is controlled by the **Image Thumbnail Size** setting in the Tool Properties dialog, in the **Form Viewer** category under the General tab. To see the original size of an image, open the cell in the cell viewer either by selecting **Edit in Cell Window** in the grid right-click menu, the toolbar button or by double-clicking on the image.

If you want numbers to be right-aligned in the Form Viewer, enable **Right Aligned Numbers** in the Tool Properties dialog, in the **Form Viewer** category under the General tab.

5.6.11 Preview Changes

You may preview the SQL statements that will be executed when choosing to **Save** the edits via the **Edit Table Data->SQL Preview** right-click menu choice.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID
1	100	Steven	King	SKING	515.123.4567	1987-06-17 00:00:00	AD_PRES
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21 00:00:00	AD_VP
3	102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13 00:00:00	AD_VP
4 ✎	103	Alex	Hunold	AHUNOLD	590.423.4567	1990-01-03 00:00:00	IT_PROG
5 ✎	104	Bro	Ernst	BERNST	590.423.4568	1991-05-21 00:00:00	IT_PROG
6	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG
7 ★	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25 00:00:00	IT_PROG
8 ✖	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05 00:00:00	IT_PROG
9 ✖	107	Diana	Lorentz	DLORENTZ	590.423.5567	1999-02-07 00:00:00	IT_PROG
10	108	Nancy	Greenberg	NGREENBE	515.124.4569	1994-08-17 00:00:00	FI_MGR
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27	125	Julia	Nayer	JNAYER	650.124.1214	1997-07-16 00:00:00	ST_CLERK

This is a preview of the SQL that will be executed when the table data edit(s) are saved.

```
UPDATE "HR"."EMPLOYEES" SET "FIRST_NAME" = 'Alex' WHERE "EMPLOYEE_ID" = 103;
UPDATE "HR"."EMPLOYEES" SET "FIRST_NAME" = 'Bro' WHERE "EMPLOYEE_ID" = 104;
INSERT INTO "HR"."EMPLOYEES" ("EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "EMAIL", "PH
DELETE FROM "HR"."EMPLOYEES" WHERE "EMPLOYEE_ID" = 106;
DELETE FROM "HR"."EMPLOYEES" WHERE "EMPLOYEE_ID" = 107
```

⚠ The listed SQL statements may not be 100% identical to what is sent to the database, as the save process uses variable binding to pass values to the database.

5.6.12 Editing Binary/BLOB and CLOB Data

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 (see page 102). (There is partial support in the Form Editor (see page 103) to view image data and to load from file).

In the grid, Binary/BLOB and CLOB data is by default presented by an icon and the size of the value. You can select another presentation format in the Tools Properties dialog, in the Grid / Binary/BLOB and CLOB Data category under the General tab. Selecting **By Value** results in performance penalties and the memory consumption increases dramatically.

In the same Tool Properties category, you can also specify how to handle **Copy/Paste and Drag and Drop** when pasting binary data in a target component that doesn't support binary data.

Editing binary data can be done by importing from a file or via the text editor in the Cell Editor. You can also copy the file in the operating system's file browser and paste it into a BLOB/CLOB cell.

Binary data in DbVisualizer is the generic term for several common binary database types:

- LONGVARBINARY
- BINARY
- VARBINARY
- BLOB

The Image Viewer supports displaying full size images for the following formats:

- GIF
- JPG
- PNG
- TIFF
- BMP
- PDF

5.7 Working with Binary and BLOB Data

DbVisualizer provides special support for working with Binary/BLOB data in a number of areas, such as:

- Viewing and Editing Binary/BLOB data (see page 105),
- Exporting Binary/BLOB data (see page 111),
- Importing Binary/BLOB data (see page)

5.8 Working with Large Text/CLOB Data

DbVisualizer provides special support for working with Large Text/CLOB data in a number of areas, such as:

- Viewing and Editing Large Text/CLOB data (see page 105),
- Exporting Large Text/CLOB data (see page 111),
- Importing Large Text/CLOB data (see page)

5.9 Using Max Rows and Max Chars for a Table

DbVisualizer limits the number of rows shown in the Data tab to 1000 rows, by default. This is done to conserve memory. If this limit prevents you from seeing the data of interest, you should first consider:

1. Using a Where Filter (see page 84) to only retrieve the rows of interest instead of all rows in the table,
2. Exporting the table (see page 109) to a file

If you really need to look at more than 1000 rows, you can change the value in the **Max Rows** field in the grid status bar. Use a value of 0 or -1 to get all rows, or a specific number (e.g. 5000) to set a new limit.

Character data columns may contain very large values that use up lots of memory. If you are only interested in seeing a few characters, you can set the **Max Chars** field in the grid status bar to the number of characters you want to see.

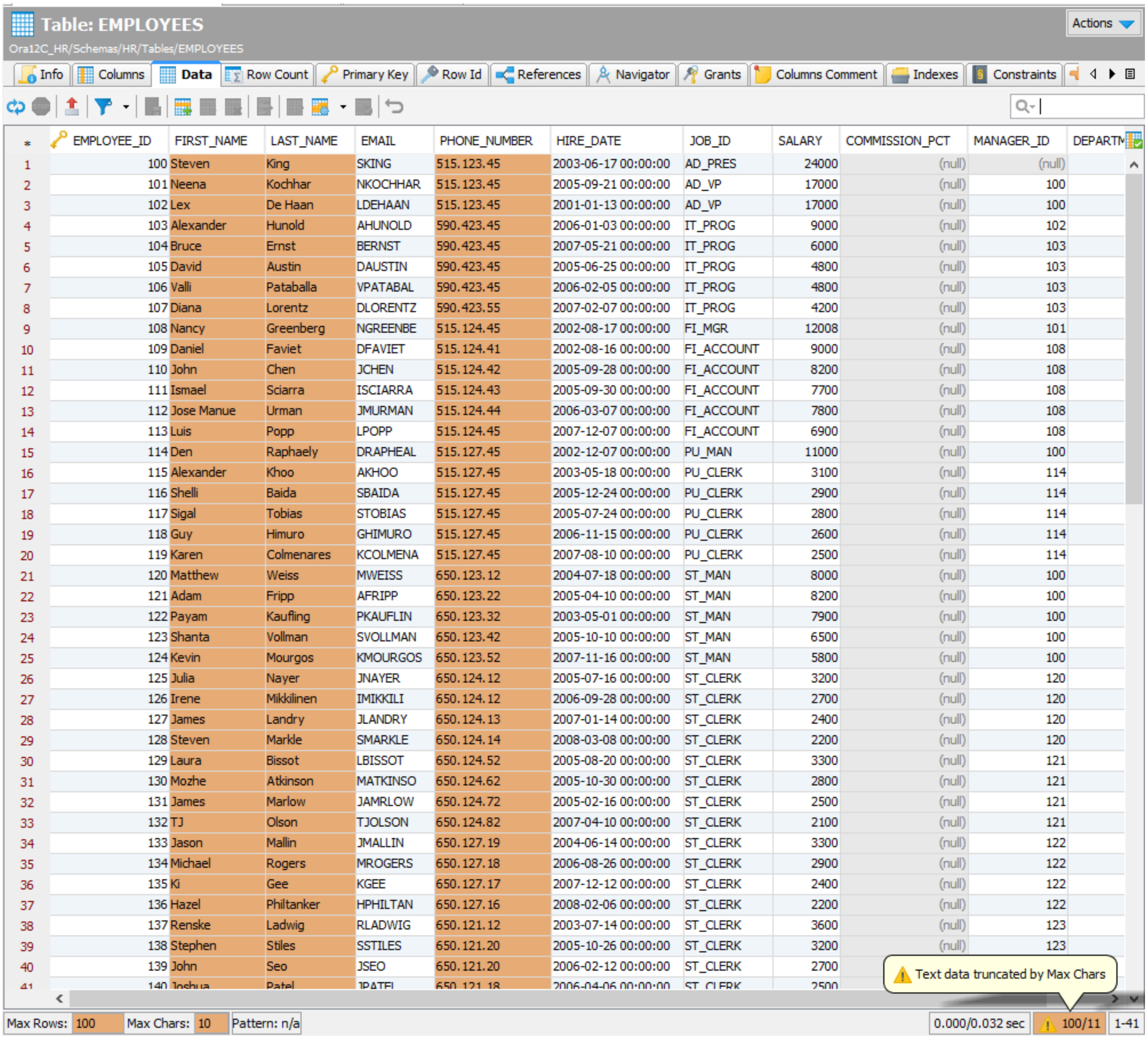
You can define how to deal with columns that have more characters than the specified maximum in the Tool Properties dialog, in the Grid category under the General tab. You have two choices: **Truncate Values** or **Truncate Values Visually**.

- **Truncate Values** truncates the original value for the grid cell to be less then the setting of Max Chars.

⚠ 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** truncates 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.

When the grid data is limited due to either the **Max Rows** or **Max Chars** value, you get an indication about this in the rows/columns field in the grid status bar and in the corresponding limit field.



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTM
1	100	Steven	King	SKING	515.123.45	2003-06-17 00:00:00	AD_PRES	24000	(null)	(null)	
2	101	Neena	Kochhar	NKOCHHAR	515.123.45	2005-09-21 00:00:00	AD_VP	17000	(null)	100	
3	102	Lex	De Haan	LDEHAAN	515.123.45	2001-01-13 00:00:00	AD_VP	17000	(null)	100	
4	103	Alexander	Hunold	AHUNOLD	590.423.45	2006-01-03 00:00:00	IT_PROG	9000	(null)	102	
5	104	Bruce	Ernst	BERNST	590.423.45	2007-05-21 00:00:00	IT_PROG	6000	(null)	103	
6	105	David	Austin	DAUSTIN	590.423.45	2005-06-25 00:00:00	IT_PROG	4800	(null)	103	
7	106	Valli	Pataballa	VPATABAL	590.423.45	2006-02-05 00:00:00	IT_PROG	4800	(null)	103	
8	107	Diana	Lorentz	DLORENTZ	590.423.55	2007-02-07 00:00:00	IT_PROG	4200	(null)	103	
9	108	Nancy	Greenberg	NGREENBE	515.124.45	2002-08-17 00:00:00	FI_MGR	12008	(null)	101	
10	109	Daniel	Faviet	DFAVIET	515.124.41	2002-08-16 00:00:00	FI_ACCOUNT	9000	(null)	108	
11	110	John	Chen	JCHEN	515.124.42	2005-09-28 00:00:00	FI_ACCOUNT	8200	(null)	108	
12	111	Ismael	Sciarra	ISCIARRA	515.124.43	2005-09-30 00:00:00	FI_ACCOUNT	7700	(null)	108	
13	112	Jose Manue	Urman	JMURMAN	515.124.44	2006-03-07 00:00:00	FI_ACCOUNT	7800	(null)	108	
14	113	Luis	Popp	LPOPP	515.124.45	2007-12-07 00:00:00	FI_ACCOUNT	6900	(null)	108	
15	114	Den	Raphaely	DRAPHEAL	515.127.45	2002-12-07 00:00:00	PU_MAN	11000	(null)	100	
16	115	Alexander	Khoo	AKHOO	515.127.45	2003-05-18 00:00:00	PU_CLERK	3100	(null)	114	
17	116	Shelli	Baida	SBAIDA	515.127.45	2005-12-24 00:00:00	PU_CLERK	2900	(null)	114	
18	117	Sigal	Tobias	STOBIAS	515.127.45	2005-07-24 00:00:00	PU_CLERK	2800	(null)	114	
19	118	Guy	Himuro	GHIMURO	515.127.45	2006-11-15 00:00:00	PU_CLERK	2600	(null)	114	
20	119	Karen	Colmenares	KCOLMENA	515.127.45	2007-08-10 00:00:00	PU_CLERK	2500	(null)	114	
21	120	Matthew	Weiss	MWEISS	650.123.12	2004-07-18 00:00:00	ST_MAN	8000	(null)	100	
22	121	Adam	Fripp	AFRIPP	650.123.22	2005-04-10 00:00:00	ST_MAN	8200	(null)	100	
23	122	Payam	Kaufling	PKAUFLIN	650.123.32	2003-05-01 00:00:00	ST_MAN	7900	(null)	100	
24	123	Shanta	Vollman	SVOLLMAN	650.123.42	2005-10-10 00:00:00	ST_MAN	6500	(null)	100	
25	124	Kevin	Mourgos	KMOURGOS	650.123.52	2007-11-16 00:00:00	ST_MAN	5800	(null)	100	
26	125	Julia	Nayer	JNAYER	650.124.12	2005-07-16 00:00:00	ST_CLERK	3200	(null)	120	
27	126	Irene	Mikkilinen	IMIKKILI	650.124.12	2006-09-28 00:00:00	ST_CLERK	2700	(null)	120	
28	127	James	Landry	JLANDRY	650.124.13	2007-01-14 00:00:00	ST_CLERK	2400	(null)	120	
29	128	Steven	Markle	SMARKLE	650.124.14	2008-03-08 00:00:00	ST_CLERK	2200	(null)	120	
30	129	Laura	Bissot	LBISSOT	650.124.52	2005-08-20 00:00:00	ST_CLERK	3300	(null)	121	
31	130	Mozhe	Atkinson	MATKINSO	650.124.62	2005-10-30 00:00:00	ST_CLERK	2800	(null)	121	
32	131	James	Marlow	JAMRLOW	650.124.72	2005-02-16 00:00:00	ST_CLERK	2500	(null)	121	
33	132	TJ	Olson	TJOLSON	650.124.82	2007-04-10 00:00:00	ST_CLERK	2100	(null)	121	
34	133	Jason	Mallin	JMALLIN	650.127.19	2004-06-14 00:00:00	ST_CLERK	3300	(null)	122	
35	134	Michael	Rogers	MROGERS	650.127.18	2006-08-26 00:00:00	ST_CLERK	2900	(null)	122	
36	135	Ki	Gee	KGEE	650.127.17	2007-12-12 00:00:00	ST_CLERK	2400	(null)	122	
37	136	Hazel	Philtanker	HPHILTAN	650.127.16	2008-02-06 00:00:00	ST_CLERK	2200	(null)	122	
38	137	Renske	Ladwig	RLADWIG	650.121.12	2003-07-14 00:00:00	ST_CLERK	3600	(null)	123	
39	138	Stephen	Stiles	SSTILES	650.121.20	2005-10-26 00:00:00	ST_CLERK	3200	(null)	123	
40	139	John	Seo	JSEO	650.121.20	2006-02-12 00:00:00	ST_CLERK	2700			
41	140	Joshua	Patel	JPATEL	650.121.18	2006-04-06 00:00:00	ST_CLERK	2500			

⚠ Text data truncated by Max Chars

Max Rows: 100 | Max Chars: 10 | Pattern: n/a | 0.000/0.032 sec | ⚠ 100/11 | 1-41

Along with the highlighted field, a warning pops up close to the field. You can disable this behavior in the Tool Properties dialog, in the **Grid** category under the General tab.

5.10 Changing the Data Display Format

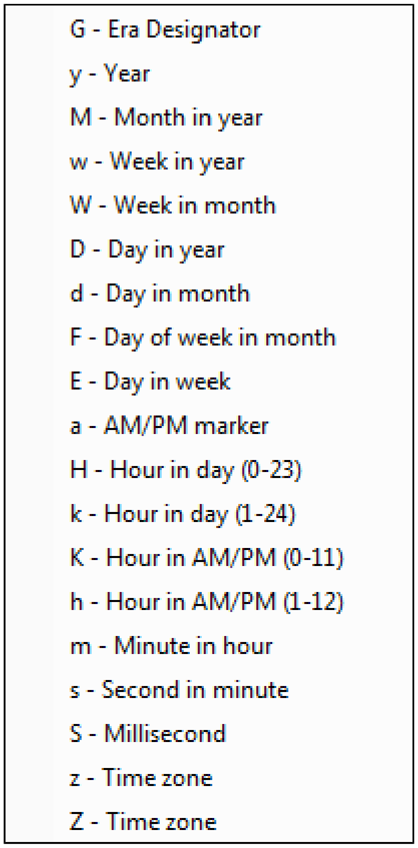
Some data, like numeric and date/time data, can be displayed in many different ways.

To define how to display and enter data in grids and forms in DbVisualizer:

1. Open **Tools->Tool** Properties,
2. Select the **Data Formats** node under the **General** tab,
3. Select or enter your preferred format for the different data types.

5.10.1 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

The complete documentation for these tokens is available at the following web page: SimpleDateFormat (http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html) .

5.10.2 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 (http://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html).

The **Unformatted** format for numbers and decimal number allows no grouping character. For Unformatted and decimal numbers a dot (".") is used as the decimal number separator.

5.11 Exporting a Table

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

You can export an individual table using the Export Table assistant.

To export a table:

1. Select the table node in the **Databases** tab tree,
2. Open the **Export Table** dialog from the right-click menu,
3. Select an **Output Format**, **Output Destination**, and **Options**,
4. Click **Export**.

5.11.1 Output Format

You can export tables in one of these formats: **CSV**, **HTML**, **SQL**, **XML**, **XLS** (Excel), or **JSON**.

For the **SQL** and **XML** formats, you can choose to export the DDL and the table data; the other formats only export table data.

You can control whether to use delimited identifiers and/or qualified names (see page 279) by default in the DDL and INSERT statements generated for the SQL format, and you can override the defaults in the Export dialog for a single export operation.

5.11.2 Output Destination

The destination can be one of:

- a file,
- an open or new SQL Commander tab, with options for where in an open SQL Commander to insert the result,
- to the system clipboard.

5.11.3 Options

The Options are contains options common to all Output Formats at the top, followed by options for the selected format.

For the SQL and XML formats, you can choose to export the DDL, the DDL for indexes for a table and the table data: as INSERT statements for the SQL statement or in one of three XML formats.

For the XLS format, you can choose to export table data as either regular Binary Excel or OOXML for Excel 2007 and later.

Most formats also let you specify other options, such as delimiters, title and descriptions. Just select an Output Format to see which options are available. All options are described in the context of the @export command (see page 214), as the Export dialog is just a GUI for the command.

You can adjust the **Data Formats** specifically for the exported table data. By default, the formats defined in **Tool Properties** are used, but sometimes you need to export dates and numbers in a different format because you intend to import the data into a different type of database.

> ℹ If you are exporting table data in the SQL format from one database type (e.g. Oracle) to import it in a database of a different type (e.g. PostgreSQL) by executing the generated script, you need to be aware of differences in the literal formats for Date, Time and Timestamp data. If you connect to the other database using a JDBC client like DbVisualizer, you can select the **JDBC escape** format for these data format. This generates literals that the JDBC driver converts into a format the target database can interpret.

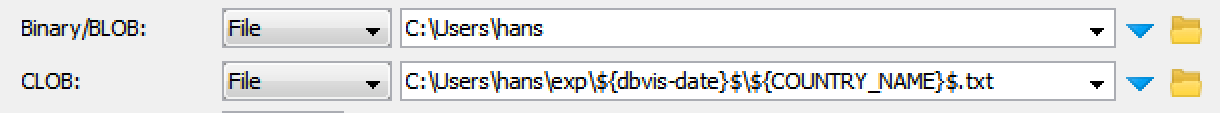
In the **Data Format Settings** dialog you can also specify how to quote text data and how to handle quotes within the text value.

Using Variables in Fields

You can use some of the pre-defined DbVisualizer variables (see page 227) (`${dbvis-date}$`, `${dbvis-time}$`, `${dbvis-timestamp}$`, `${dbvis-connection}$`, `${dbvis-database-type}$` and `${dbvis-object}$`) in all fields that hold free text (e.g. title and description fields) and as part of the file name field.

5.11.4 Exporting Binary/BLOB and CLOB Data

You can use the export assistant 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 or filename pattern for the data files. If you do not specify a directory or filename pattern, the operating system's default directory for temporary files (e.g. *C:\TEMP* or */tmp*) is used.



Binary/BLOB: File C:\Users\hans

CLOB: File C:\Users\hans\exp\${dbvis-date}$\${COUNTRY_NAME}$.txt

A pattern is a path with fixed and variable parts, where the variable parts are expressed as DbVisualizer variables, e.g. *C:\Users\hans\exp\${dbvis-date}$\${COUNTRY_NAME}$.txt*. You can select variables to insert at the current caret position in the path field from the dropdown. The variables that can be used are the predefined DbVisualizer variables plus variables for each column in the table, e.g. `${COUNTRY_NAME}$` in the example above. Another special variable that can be helpful here is `${dbvis-column-name}$`. If a table has multiple BLOB or CLOB columns, you can use it in the pattern to export the columns to separate files, e.g. *C:\Users\hans\exp\${dbvis-column-name}$.txt*. All variables that can be used are listed in the menu that is displayed when you click the blue arrow to the right of the field. You can select a variable from the menu to insert it in the pattern field at the care position.

The data for each individual value of this type is then exported to a separate file and a DbVisualizer variable referencing the file is inserted in the main export file.

5.11.5 Saving And Loading Settings

If you often use the same settings, you can save them as the default settings for this assistant. If you use a number of common settings, you can save them to individual files that you can load as needed. Use the Settings button menu to accomplish this:

- **Save as Default Settings**
  Saves all format settings as default. These are then loaded automatically when open an Export Schema dialog
- **Use Default Settings**
  Use this choice to initialize the settings with default values
- **Remove Default Settings**
  Removes the saved defaults and restores the regular defaults
- **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.

5.11.6 Other Ways to Export Table Data

- Export all or selected tables with the Export Schema (see page 152) assistant,
- Export a subset of the table data with the @export command (see page 214).

5.12 Importing Table Data

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

You can import data using the Import Table Data wizard.

You can import data from a file into an existing table or to a new table. The import source can be either a CSV file or an Excel file (*.xls* or *.xlsx*). The steps are almost identical:

1. Select the table node for the table you want to import to, or the **Tables** node if you are importing to a new table, in the **Databases** tab tree,
2. Open the **Import Table Data** wizard from the right-click menu,
3. Specify the input file on the first wizard page (CSV or Excel file),
4. [Excel only] : If the input file is an Excel file, you are asked to choose the Excel sheet to import on the next page.
5. Specify file format and other options,
6. Specify data formats and the data type per column,
7. Adjust details about the destination table,
8. Click **Import** on the last page.

Instead of choosing Import Table Data from the right-click menu, you can **drag and drop a file** from the operating system's file manager on the Tables node or a table node.

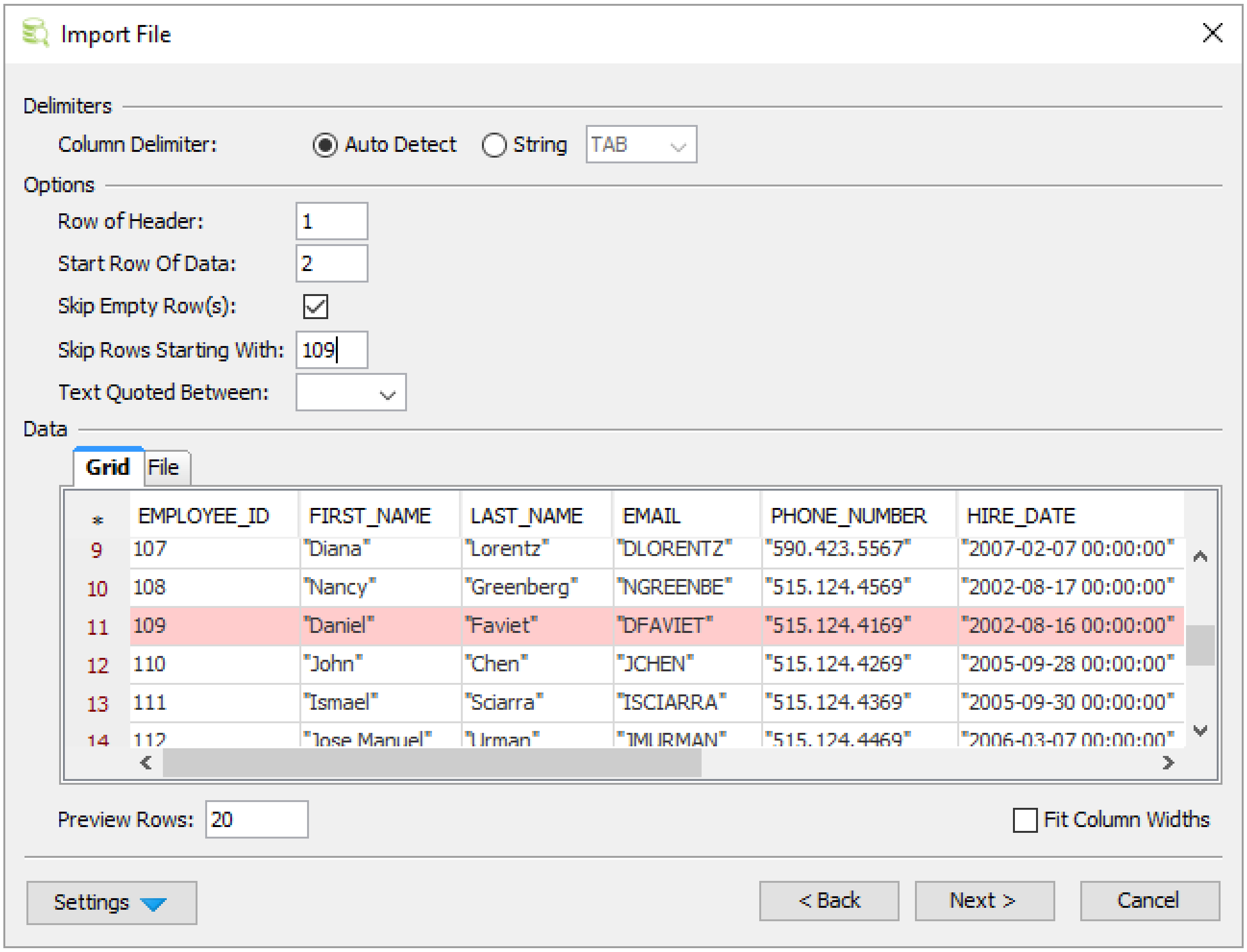
How many INSERT statements to execute during the import process before committing the changes can be specified in the **Properties** tab for the connection, in the **Transaction** category.

5.12.1 Input File Format and Other Options

On the File Format page, you specify what and how the data in the source file should be imported. This includes specifying what row to start the import from and if empty rows should be skipped.

Currently DbVisualizer supports import of CSV files and the Excel file formats ".xsl" and "xslx".

CSV format page



Import File

Delimiters

Column Delimiter: ● Auto Detect ○ String TAB

Options

Row of Header: 1

Start Row Of Data: 2

Skip Empty Row(s): ☑

Skip Rows Starting With: 109

Text Quoted Between:

Data

*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
9	107	"Diana"	"Lorentz"	"DLORENTZ"	"590.423.5567"	"2007-02-07 00:00:00"
10	108	"Nancy"	"Greenberg"	"NGREENBE"	"515.124.4569"	"2002-08-17 00:00:00"
11	109	"Daniel"	"Faviet"	"DFAVIET"	"515.124.4169"	"2002-08-16 00:00:00"
12	110	"John"	"Chen"	"JCHEN"	"515.124.4269"	"2005-09-28 00:00:00"
13	111	"Ismael"	"Sciarra"	"ISCIARRA"	"515.124.4369"	"2005-09-30 00:00:00"
14	112	"Jose Manuel"	"Urman"	"JMURMAN"	"515.124.4469"	"2006-03-07 00:00:00"

Preview Rows: 20

□ Fit Column Widths

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 "%"

⚠

You can specify any character sequence as a delimiter, but it must not contain more than four characters.

You can use the **Options** area to further specify how to read the input file, for instance if certain rows should be skipped and how text data is quoted.

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 any of the **Options** settings result in rows not being qualified.

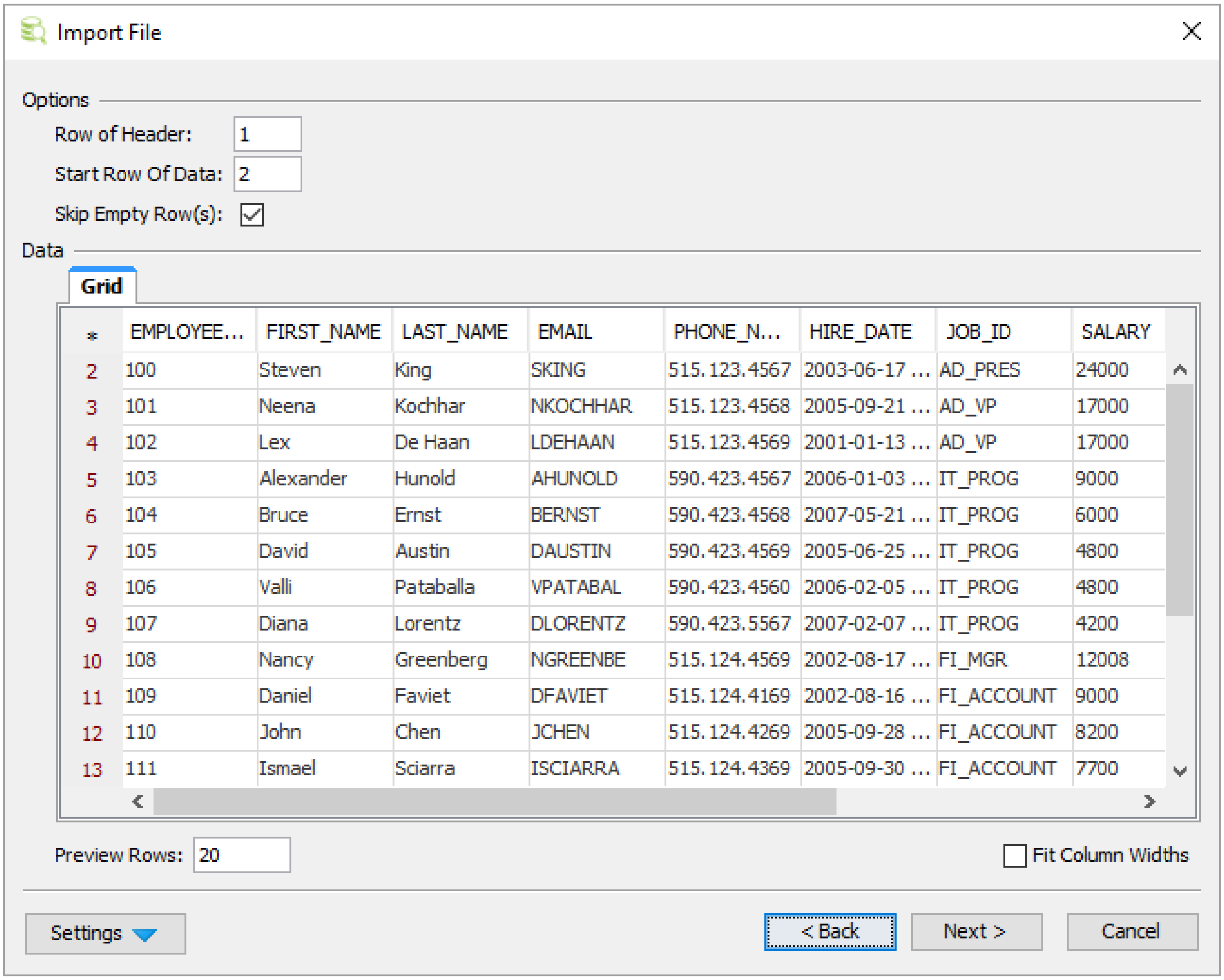
Excel format page

The Excel format page is very much like the CSV format page.

As Excel is from start organized in columns and rows the **Column delimiter** setting is not applicable to Excel files. The **Skip Rows Starting With** and the **Text Quoted Between** options are also not supported for Excel.

As shown in the snapshot below there is no **File** tab for Excel files.

The **Grid** tab shows a preview of the data, just as in the CSV case.



Import File

Options

Row of Header: 1

Start Row Of Data: 2

Skip Empty Row(s): ☑

Data

Grid

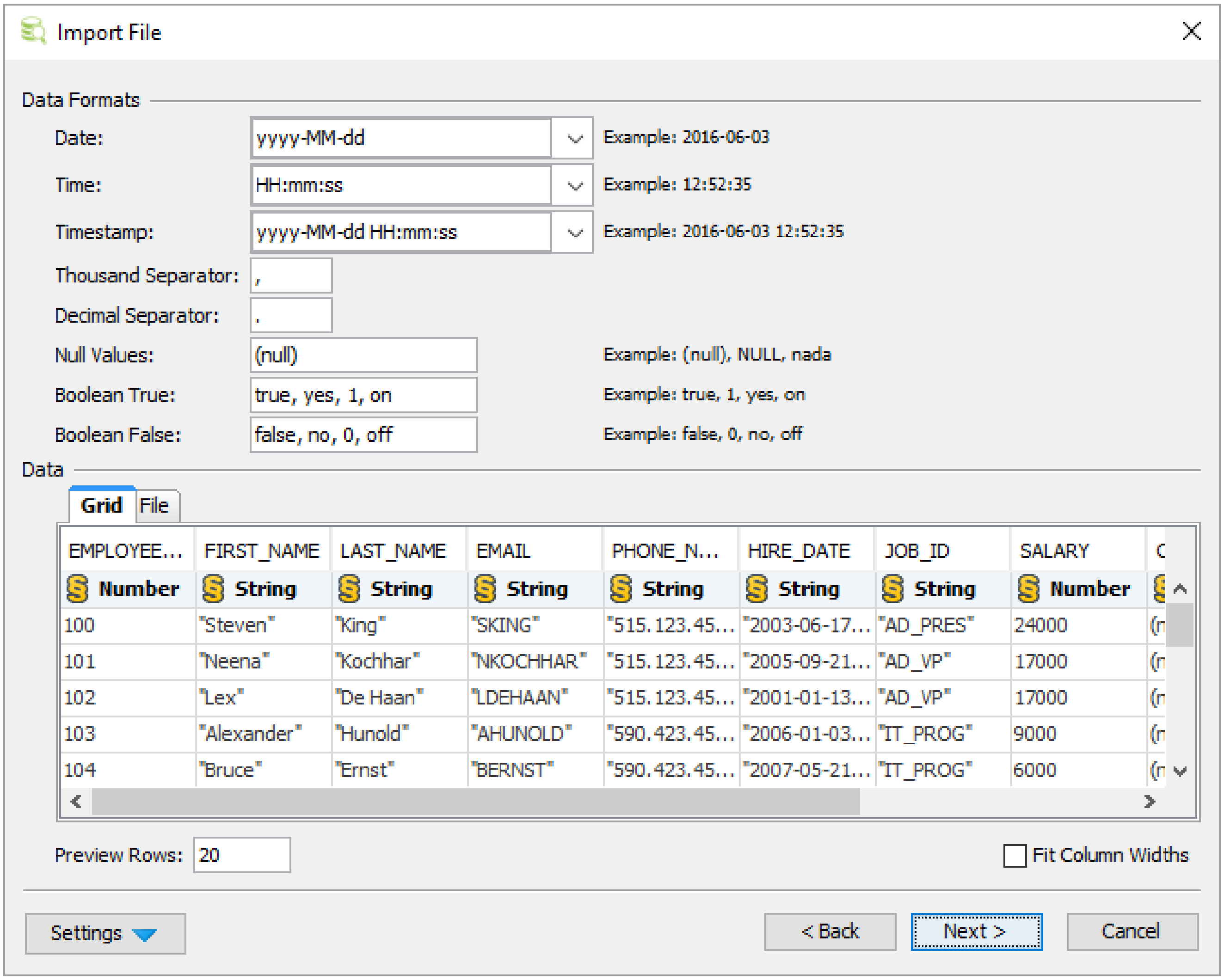
*	EMPLOYEE...	FIRST_NAME	LAST_NAME	EMAIL	PHONE_N...	HIRE_DATE	JOB_ID	SALARY
2	100	Steven	King	SKING	515.123.4567	2003-06-17 ...	AD_PRES	24000
3	101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-21 ...	AD_VP	17000
4	102	Lex	De Haan	LDEHAAN	515.123.4569	2001-01-13 ...	AD_VP	17000
5	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03 ...	IT_PROG	9000
6	104	Bruce	Ernst	BERNST	590.423.4568	2007-05-21 ...	IT_PROG	6000
7	105	David	Austin	DAUSTIN	590.423.4569	2005-06-25 ...	IT_PROG	4800
8	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-05 ...	IT_PROG	4800
9	107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-07 ...	IT_PROG	4200
10	108	Nancy	Greenberg	NGREENBE	515.124.4569	2002-08-17 ...	FI_MGR	12008
11	109	Daniel	Faviet	DFAVIET	515.124.4169	2002-08-16 ...	FI_ACCOUNT	9000
12	110	John	Chen	JCHEN	515.124.4269	2005-09-28 ...	FI_ACCOUNT	8200
13	111	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-30 ...	FI_ACCOUNT	7700

Preview Rows: 20

☐ Fit Column Widths

5.12.2 Data Formats and Data Type Per Column

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 for a column, use the drop-down lists to select the appropriate type.



Import File

Data Formats

Date:	yyyy-MM-dd	Example: 2016-06-03
Time:	HH:mm:ss	Example: 12:52:35
Timestamp:	yyyy-MM-dd HH:mm:ss	Example: 2016-06-03 12:52:35
Thousand Separator:	,	
Decimal Separator:	.	
Null Values:	(null)	Example: (null), NULL, nada
Boolean True:	true, yes, 1, on	Example: true, 1, yes, on
Boolean False:	false, no, 0, off	Example: false, 0, no, off

Data

EMPLOYEE...	FIRST_NAME	LAST_NAME	EMAIL	PHONE_N...	HIRE_DATE	JOB_ID	SALARY	C
Number	String	String	String	String	String	String	Number	
100	"Steven"	"King"	"SKING"	"515.123.45...	"2003-06-17...	"AD_PRES"	24000	(r
101	"Neena"	"Kochhar"	"NKOCHHAR"	"515.123.45...	"2005-09-21...	"AD_VP"	17000	(r
102	"Lex"	"De Haan"	"LDEHAAN"	"515.123.45...	"2001-01-13...	"AD_VP"	17000	(r
103	"Alexander"	"Hunold"	"AHUNOLD"	"590.423.45...	"2006-01-03...	"IT_PROG"	9000	(r
104	"Bruce"	"Ernst"	"BERNST"	"590.423.45...	"2007-05-21...	"IT_PROG"	6000	(r

Preview Rows: 20

□ Fit Column Widths

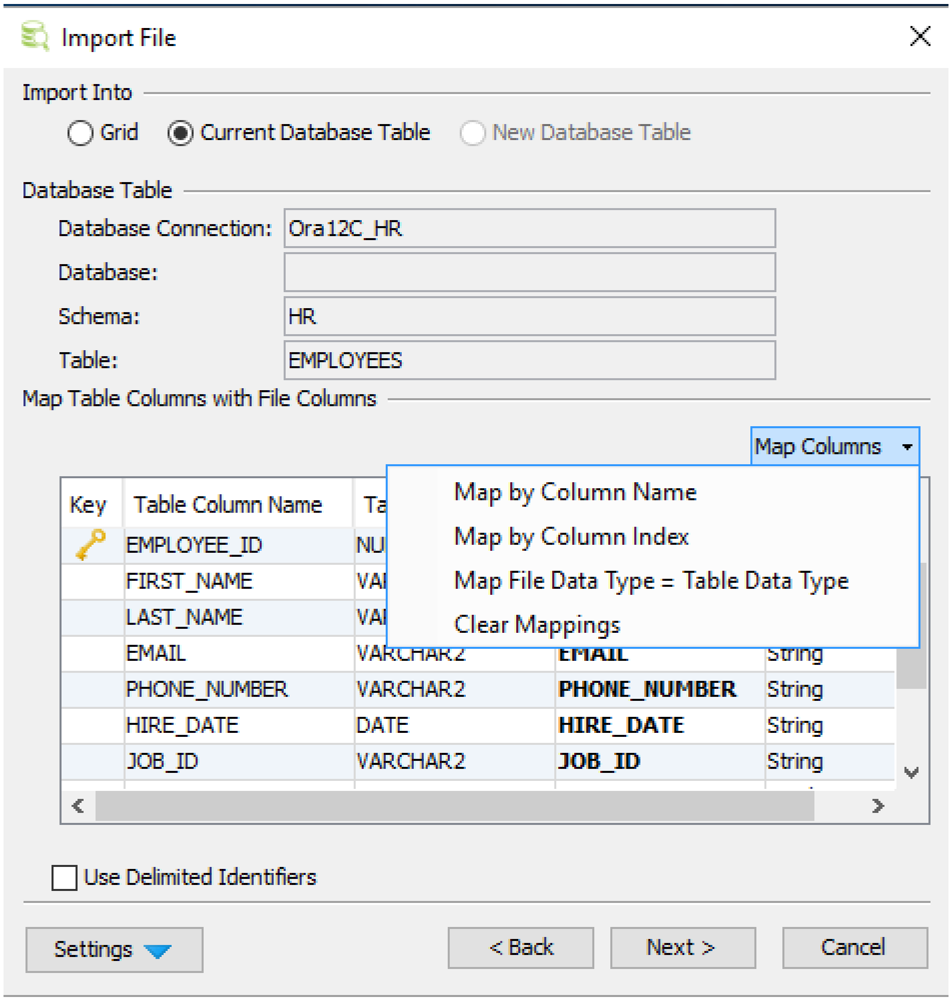
If you need to change the data type for a number of columns, e.g. set them all to String, you can Copy/Paste the data type. First change it for one of the columns using the drop-down, select and copy that new data type value and then select the data type for all other columns and use paste to change them all at once. If you make a mistake, you can change the **Preview Rows** value to let DbVisualizer determine the types again.

If you import to an existing table, there is yet another way to adjust the data types for the file columns, described in the next section.

5.12.3 Matching Columns and Data Types for an Existing Table

When you are importing to an existing table, the Import Destination page provides two options: **Grid** and **Current Database Table**. You can use the **Grid** choice to import the data into a grid that is presented in its own window in DbVisualizer if you just want to just process the data in some way without saving it in the database.

When the **Current Database Table** choice is selected, the page shows information about the table into which the data will be imported in the **Map Table Columns with File Columns** grid shows the columns in the selected database table and the columns in the source file.



Import Into

○ Grid ● Current Database Table ○ New Database Table

Database Table

Database Connection: Ora12C_HR

Database:

Schema: HR

Table: EMPLOYEES

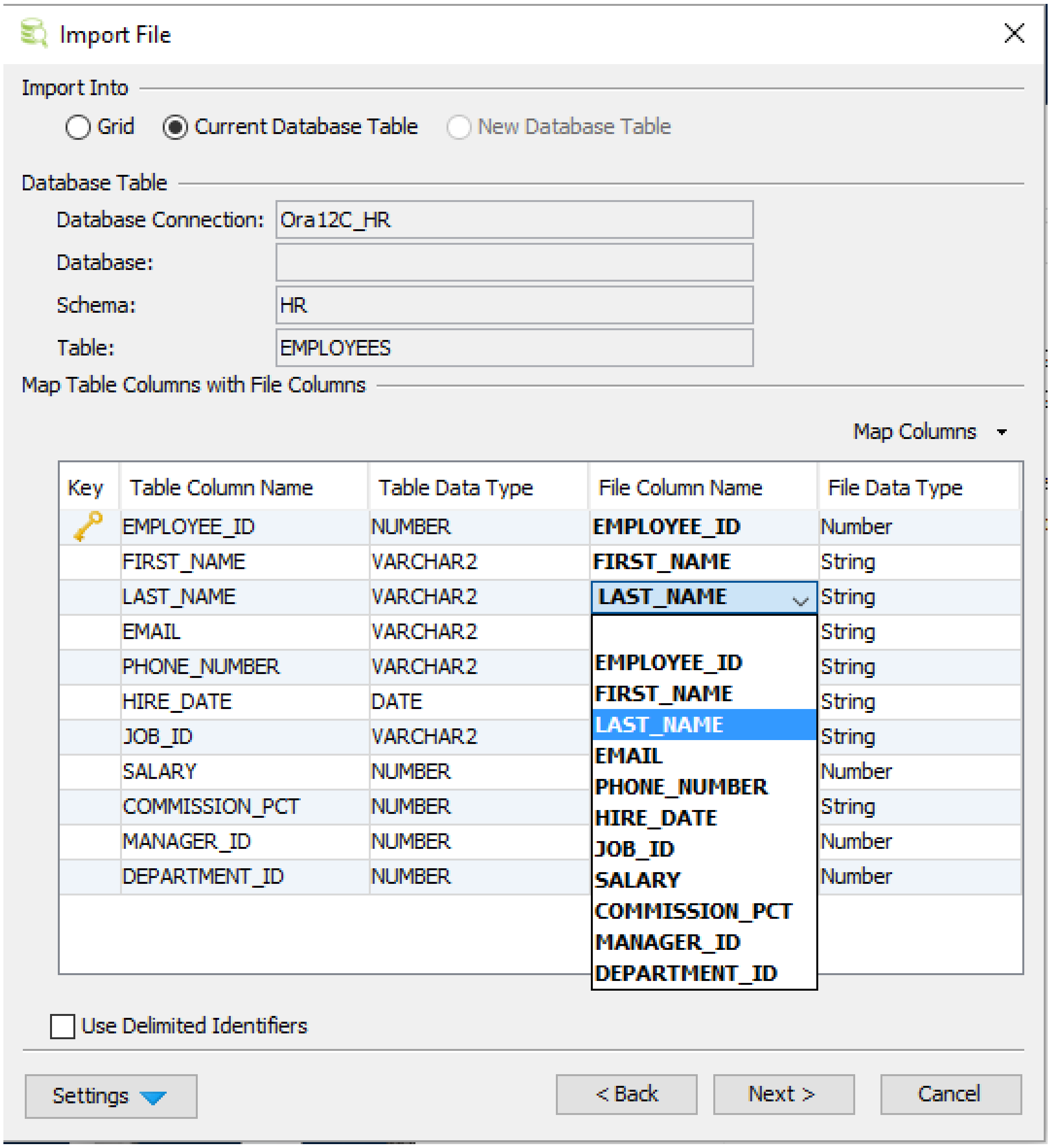
Map Table Columns with File Columns

Key	Table Column Name	Ta		
🔑	EMPLOYEE_ID	NU		
	FIRST_NAME	VA		
	LAST_NAME	VA		
	EMAIL	VARCHAR2	EMAIL	String
	PHONE_NUMBER	VARCHAR2	PHONE_NUMBER	String
	HIRE_DATE	DATE	HIRE_DATE	String
	JOB_ID	VARCHAR2	JOB_ID	String

□ Use Delimited Identifiers

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. **Map by Column Name** and **Map by Column Index** do exactly what it sounds like. **Map File Data Type = Table Data Type** sets the **File Data Type** for each column to the type of the corresponding table column.

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.



Import File

Import Into

○ Grid ● Current Database Table ○ New Database Table

Database Table

Database Connection: Ora12C_HR

Database:

Schema: HR

Table: EMPLOYEES

Map Table Columns with File Columns

Key	Table Column Name	Table Data Type	File Column Name	File Data Type
🔑	EMPLOYEE_ID	NUMBER	EMPLOYEE_ID	Number
	FIRST_NAME	VARCHAR2	FIRST_NAME	String
	LAST_NAME	VARCHAR2	LAST_NAME	String
	EMAIL	VARCHAR2		String
	PHONE_NUMBER	VARCHAR2		String
	HIRE_DATE	DATE		String
	JOB_ID	VARCHAR2		String
	SALARY	NUMBER		Number
	COMMISSION_PCT	NUMBER		String
	MANAGER_ID	NUMBER		Number
	DEPARTMENT_ID	NUMBER		Number

□ Use Delimited Identifiers

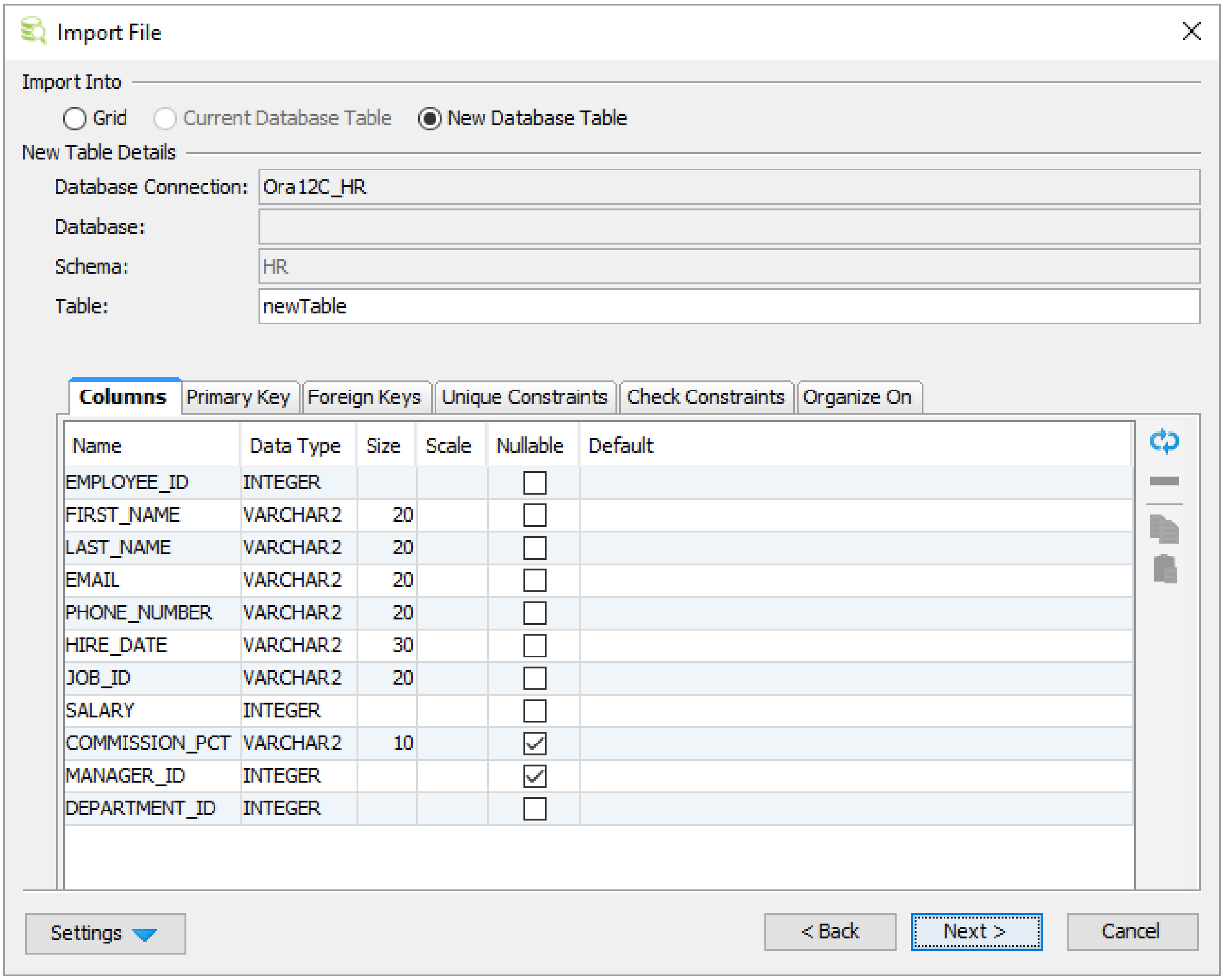
⚠ You can use copy/paste of the values in the **File Column Name** and **File Data Type** fields to quickly fill the selection of cells instead of manually selecting the correct data in the drop-downs.

There is also a **Use delimited identifiers** checkbox. Check this box if you want the SQL statements for importing the table to use delimited identifiers; in other words, if you want to use table and column names with special characters, mixed case, or anything else that requires delimited (quoted) identifiers.

5.12.4 Adjusting Table Declaration for a New Table

When you are importing to a new table, the Import Destination page provides two options: **Grid** and **New Database Table**. You can use the Grid choice to import the data into a grid that is presented in its own window in DbVisualizer if you just want to just process the data in some way without saving it in the database.

When the New Database Table choice is selected, 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 Into

○ Grid ○ Current Database Table ● New Database Table

New Table Details

Database Connection: Ora12C_HR

Database:

Schema: HR

Table: newTable

Name	Data Type	Size	Scale	Nullable	Default
EMPLOYEE_ID	INTEGER			□	
FIRST_NAME	VARCHAR2	20		□	
LAST_NAME	VARCHAR2	20		□	
EMAIL	VARCHAR2	20		□	
PHONE_NUMBER	VARCHAR2	20		□	
HIRE_DATE	VARCHAR2	30		□	
JOB_ID	VARCHAR2	20		□	
SALARY	INTEGER			□	
COMMISSION_PCT	VARCHAR2	10		☑	
MANAGER_ID	INTEGER			☑	
DEPARTMENT_ID	INTEGER			□	

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**, adjusted up to the next power of ten. You can ignore certain columns by removing them in the **Columns** tab. **Keys** and other constraints can be created using the other tabs.

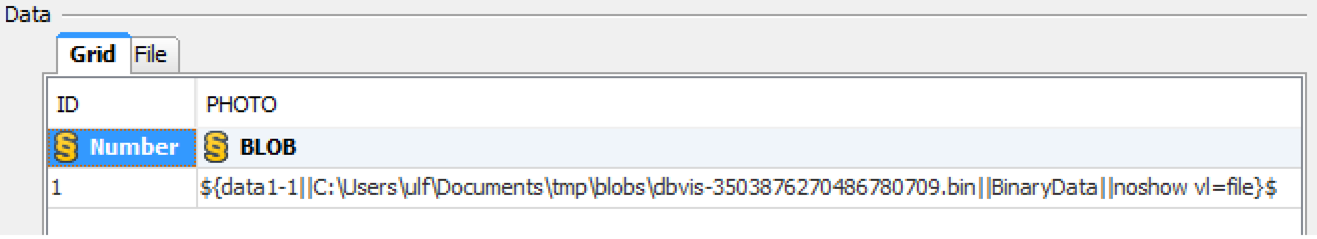
You can go back to the Data Format page and increase the **Preview Rows** value if you believe that it will help DbVisualizer to pick better defaults. If you do so, you need to click the **Reload** button when you come back to this page to rescan the source data and get new default values.

If you make a mistake, or if the import fails, so you have to go back and make adjustments before you import again, make sure you enable **Drop existing table, if any**. It is disabled by default to prevent you from accidentally dropping an existing table when you intend to import to a new table, but if the import fails, the new table may already have been created so it needs to be dropped before a new table with your adjusted input can be created.

There is also a **Use delimited identifiers** checkbox. Check this box if you want the SQL statements for importing the table to use delimited identifiers; in other words, if you want to use table and column names with special characters, mixed case, or anything else that requires delimited (quoted) identifiers.

5.12.5 Importing Binary/BLOB and CLOB Data (CSV and SQL Only)

If you have exported data to a CSV file using DbVisualizer, 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**.



ID	PHOTO
Number	BLOB
1	${data1-1\|\|C:\Users\ulf\Documents\tmp\blobs\dbvis-3503876270486780709.bin\|\|BinaryData\|\|noshow vl=file}$

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.

5.12.6 Saving And Loading Settings

If you often use the same settings, you can save them as the default settings for this assistant. If you use a number of common settings, you can save them to individual files that you can load as needed. Use the Settings button menu to accomplish this:

- **Save as Default Settings**
  Saves all format settings as default. These are then loaded automatically when open an Export Schema dialog
- **Use Default Settings**
  Use this choice to initialize the settings with default values

- **Remove Default Settings**
  Removes the saved defaults and restores the regular defaults
- **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

5.12.7 Other Ways to Import Table Data

If you have a script containing INSERT statements for all data, you can execute it in the SQL Commander (see page 174).

5.12.8 Known limitations

- Excel files cannot contain CLOB/BLOB type of data (e.g. images etc). Cells with this kind of data are imported as empty.
- There is a size limitation when importing Excel files with the `.xls` filename extension. The size limitation is roughly 20 megabytes, depending on your configuration and how much memory is used for other things. Increasing DbVisualizer max memory (http://confluence.dbvis.com/pages/viewpage.action?pageId=3146118) may allow you to import larger files.

5.13 Comparing Tables

> **Only in DbVisualizer Pro**
>
> This feature is only available in the DbVisualizer Pro edition.

You can compare different aspects of a table to other tables and/or result set grids.

For instance, to compare the DDL for a table to the DDL for another table:

1. Open the **DDL** tab for the table,
2. Open the **DDL** tab for the other table,
3. Select **Compare** from the right-click menu in one of the **DDL** tabs to compare their text content (see page 261).

To compare the table data to the data of another table or a result set:

1. Open the **Data** tab for the table,
2. Open the **Data** tab for another table or execute an SQL query to open a result set tab,

3. Select **Compare** from the right-click menu in one of the tabs to compare their grid content (see page 262) .

You can do the same for all the other Object View sub tabs containing a grid, such as the **Primary Key** or **Columns** tab.

5.14 Viewing Table Relationships

To see how a table is related to other tables through Foreign Keys:

1. Locate the table in the **Databases** tab tree,
2. Double-click the node to open its **Object View** tab,
3. Select the **References** sub tab,
4. Select **Imported Keys** from the drop-down list in the toolbar to see tables referenced by foreign keys in this table,
5. Select **Exported Keys** from the drop-down list to see tables referencing this table by foreign keys.

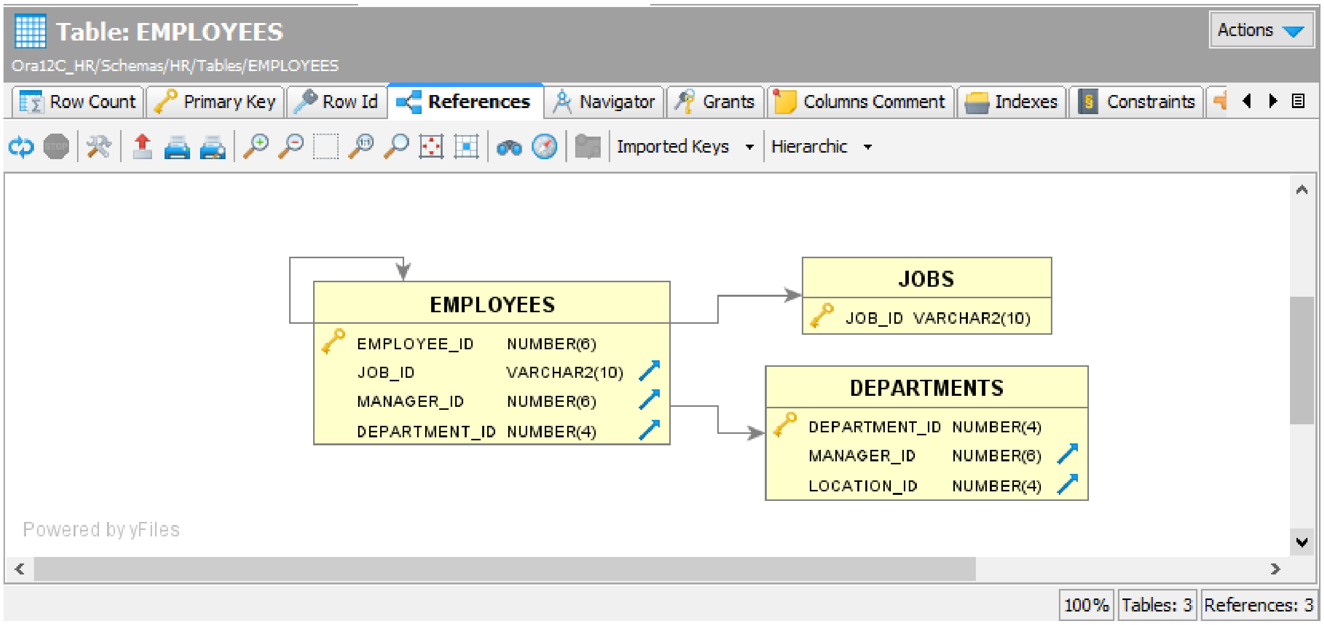


Table: EMPLOYEES

Ora12C_HR/Schemas/HR/Tables/EMPLOYEES

EMPLOYEES	
EMPLOYEE_ID	NUMBER(6)
JOB_ID	VARCHAR2(10)
MANAGER_ID	NUMBER(6)
DEPARTMENT_ID	NUMBER(4)

JOBS	
JOB_ID	VARCHAR2(10)

DEPARTMENTS	
DEPARTMENT_ID	NUMBER(4)
MANAGER_ID	NUMBER(6)
LOCATION_ID	NUMBER(4)

You can select among different graph layouts in the layout drop-down list in the toolbar: **Hierarchic**, **Organic**, **Orthogonal**, or **Circular**.
Other layout settings can be changed in the **Graph Control** area, which is shown or hidden with the settings toggle button in the toolbar. For instance, you can select how much information to include for each table in the graph: just the **Table Name**, the **Primary Key** column(s) or all **Columns**.

The graph can be **Exported** to a file in **JPG**, **GIF**, **PNG**, **SVG** or **PDF** or **Saved** as a Graph Modeling Language (**GML**) file that you can then open in the yEd (http://www.yworks.com/en/products_yed_about.html) tool from yWorks for further manipulation.

You can control whether the table names should be qualified with the schema/catalog (see page 279) in the graph.

5.15 Navigating Table Relationships

Only in DbVisualizer Pro

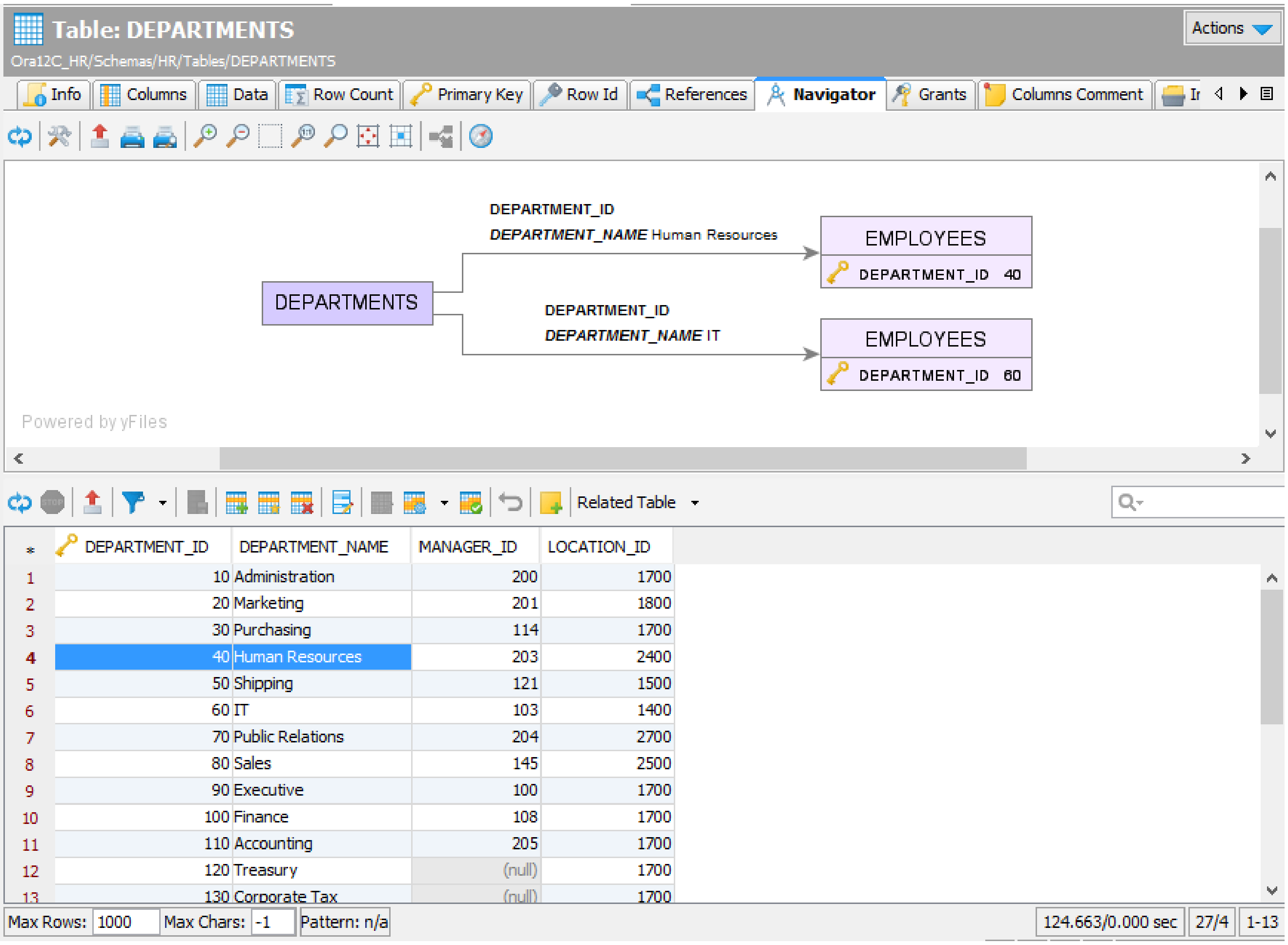
This feature is only available in the DbVisualizer Pro edition.

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.

5.15.1 Opening the Navigator

To launch the **Navigator:**

1. Locate the table you want to start the navigation from in the **Databases** tab tree,
2. Double-click the table node to open its **Object View** tab,
3. Select the **Navigator** sub tab.



*	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500
9	90	Executive	100	1700
10	100	Finance	108	1700
11	110	Accounting	205	1700
12	120	Treasury	(null)	1700
13	130	Corporate Tax	(null)	1700

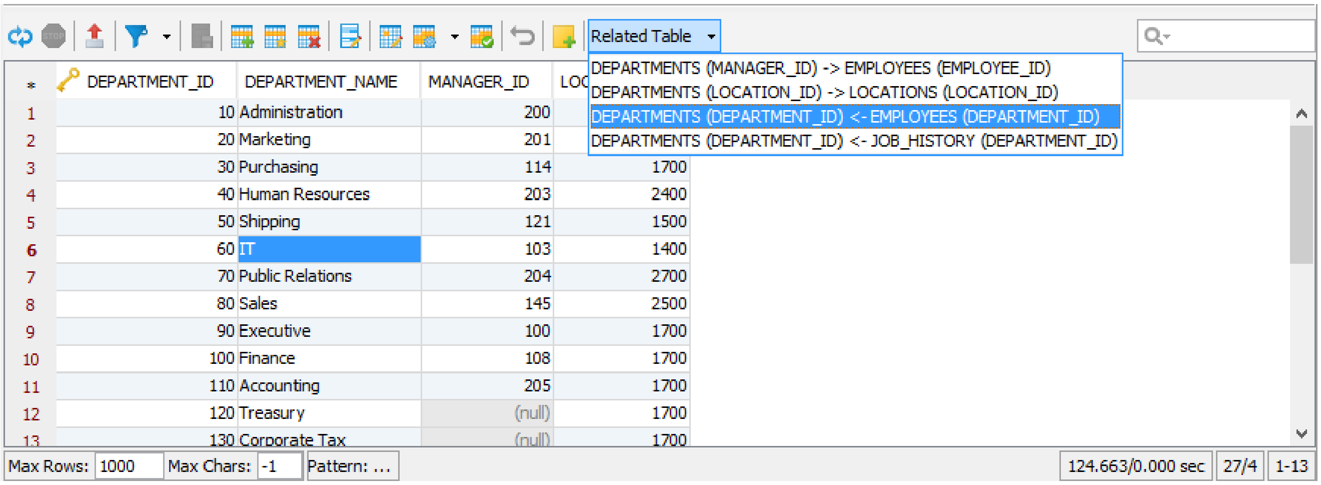
Max Rows: 1000 Max Chars: -1 Pattern: n/a 124.663/0.000 sec 27/4 1-13

The **Navigator** tab 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 of the same type as you encounter in other parts of DbVisualizer, such as in the Data tab (see page 94), but extended with a **Related Table** list and a **Tag** button.

5.15.2 Navigating Realtionships

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 screen shots 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.

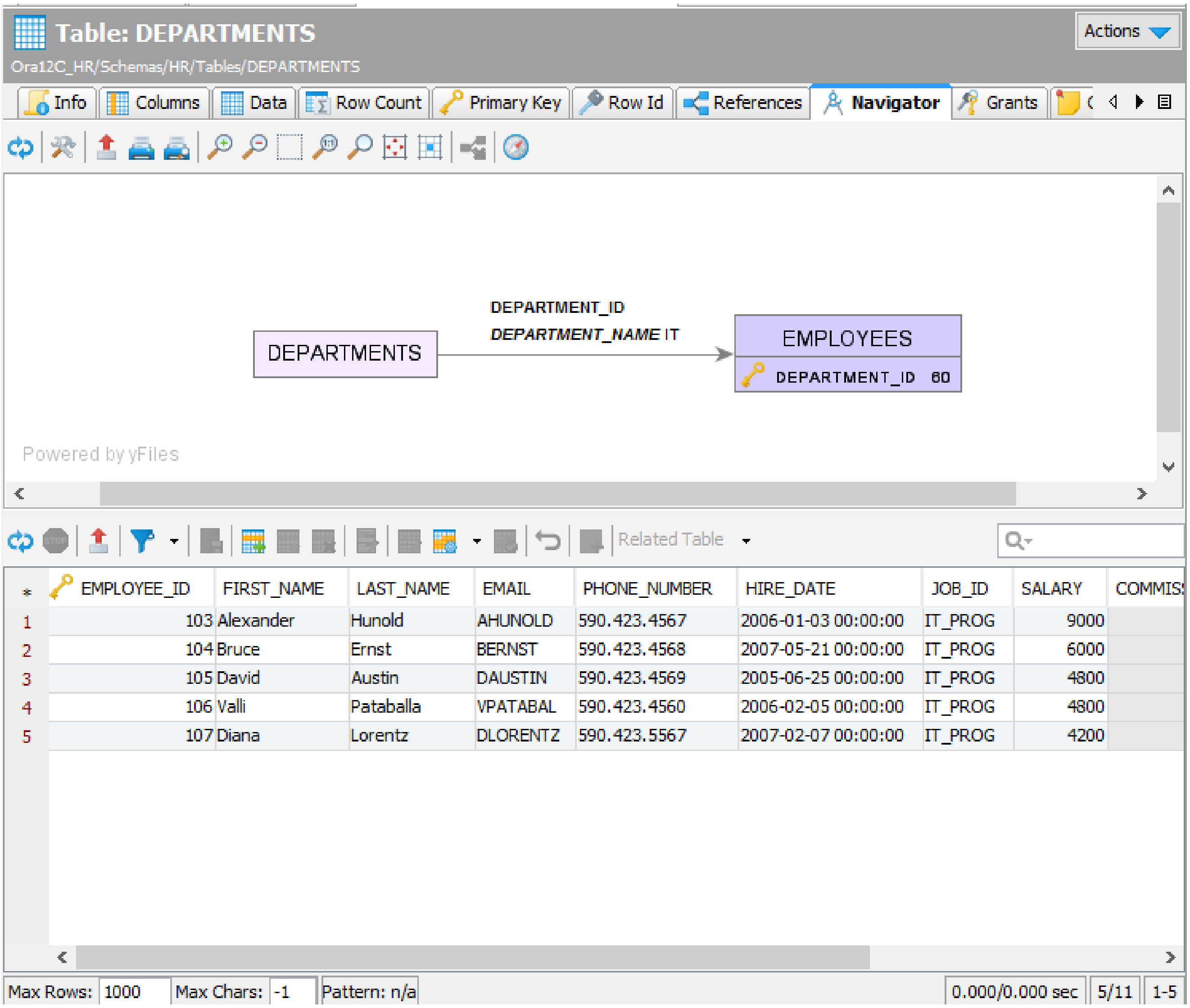

Related Table
DEPARTMENTS (MANAGER_ID) -> EMPLOYEES (EMPLOYEE_ID)
DEPARTMENTS (LOCATION_ID) -> LOCATIONS (LOCATION_ID)
DEPARTMENTS (DEPARTMENT_ID) <- EMPLOYEES (DEPARTMENT_ID)
DEPARTMENTS (DEPARTMENT_ID) <- JOB_HISTORY (DEPARTMENT_ID)

*	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOC
1	10	Administration	200	
2	20	Marketing	201	
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500
9	90	Executive	100	1700
10	100	Finance	108	1700
11	110	Accounting	205	1700
12	120	Treasury	(null)	1700
13	130	Corporate Tax	(null)	1700

Max Rows: 1000 Max Chars: -1 Pattern: ... 124.663/0.000 sec 27/4 1-13

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 (->) between the table names 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.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMIS
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03 00:00:00	IT_PROG	9000	
2	104	Bruce	Ernst	BERNST	590.423.4568	2007-05-21 00:00:00	IT_PROG	6000	
3	105	David	Austin	DAUSTIN	590.423.4569	2005-06-25 00:00:00	IT_PROG	4800	
4	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-05 00:00:00	IT_PROG	4800	
5	107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-07 00:00:00	IT_PROG	4200	

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.000/0.000 sec 5/11 1-5

When you select "DEPARTMENTS (DEPARTMENT_ID) <- 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.

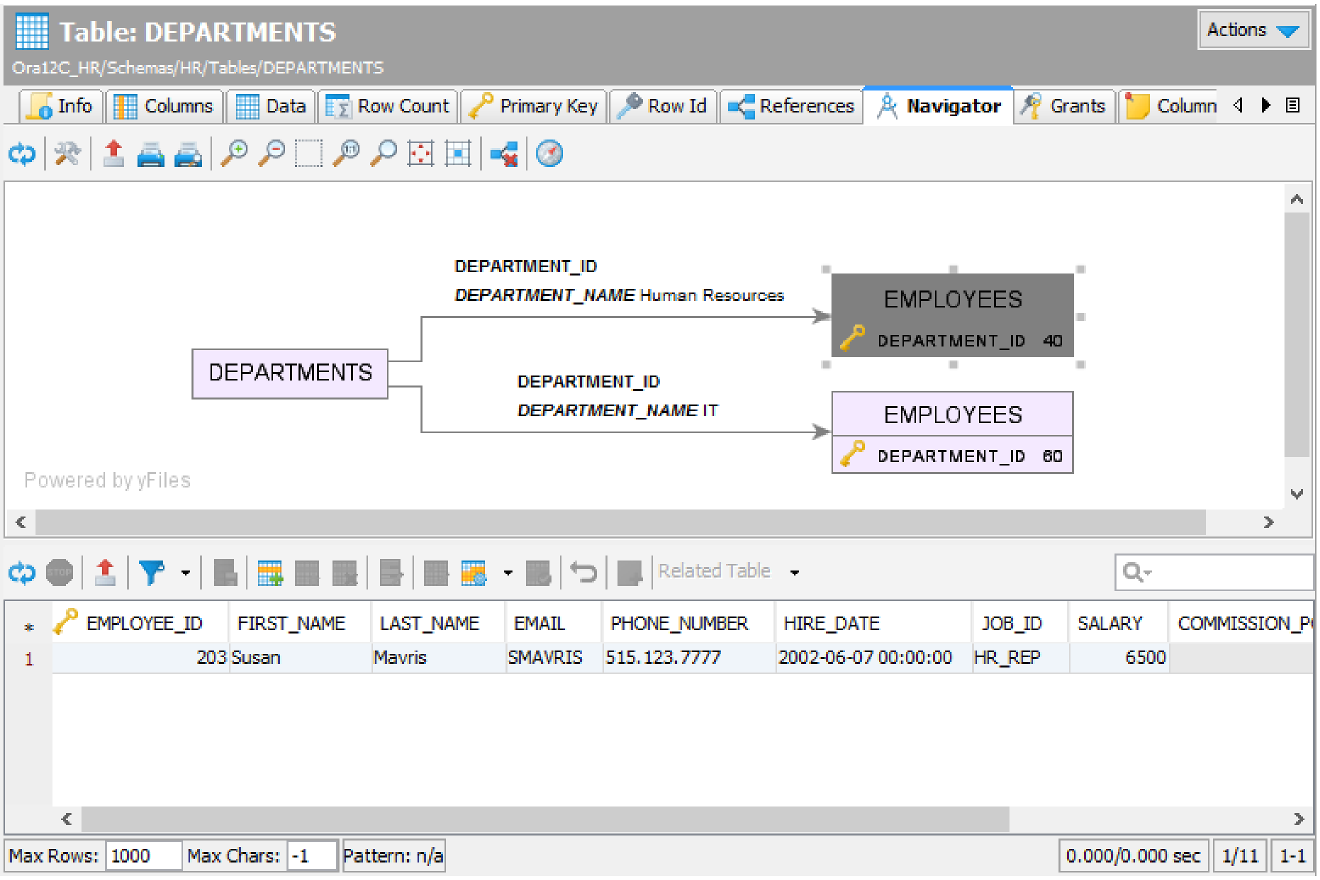
The `EMPLOYEES` node contains the key columns (just one in this example) and their values.

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 edit the grid values, but be aware that if you change the value of a key in the grid for a navigation case, the row will disappear from the grid since the grid only shows rows with keys matching the navigation case key value.

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.



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
1	203	Susan	Mavris	SMAVRIS	515.123.7777	2002-06-07 00:00:00	HR_REP	6500	

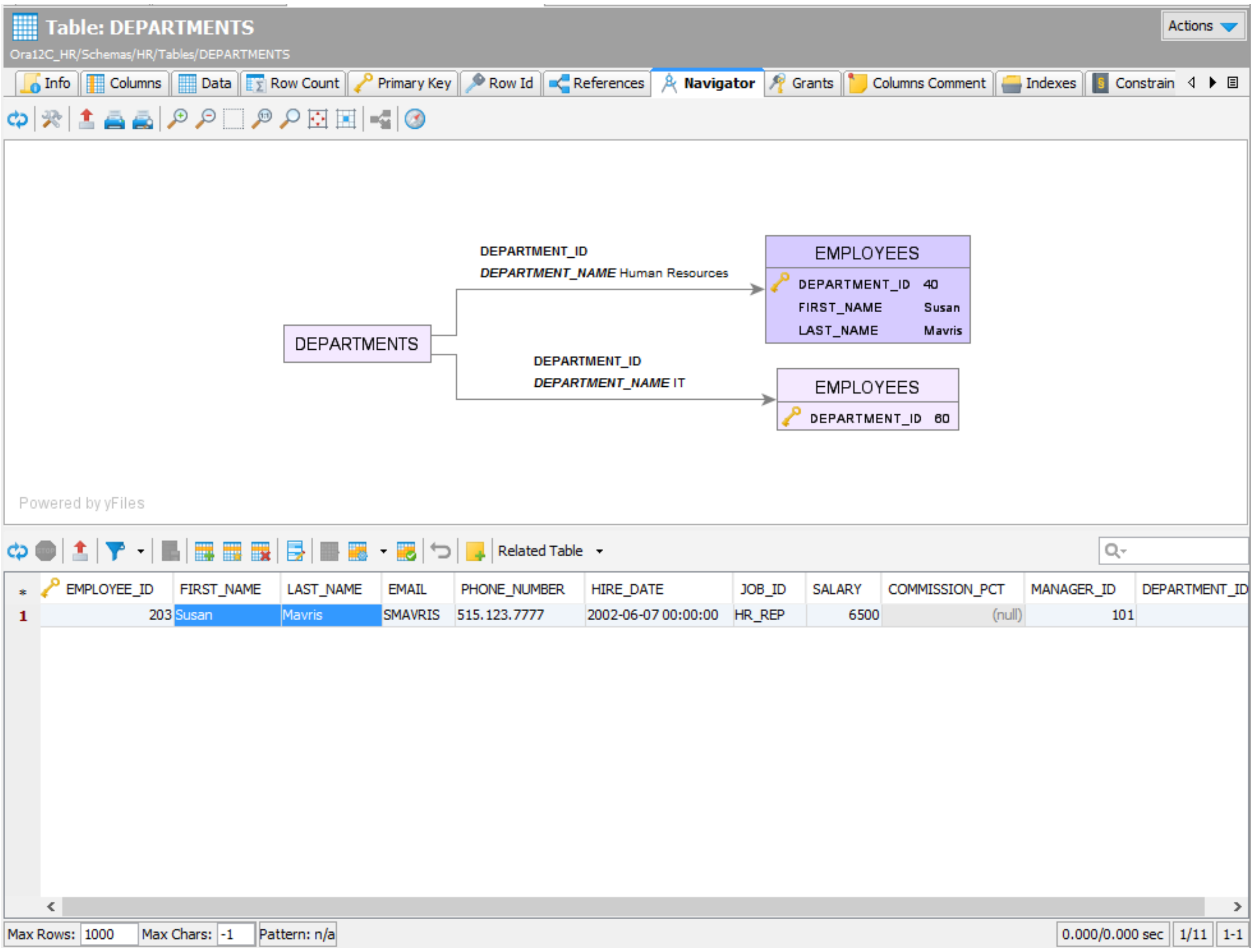
> If you want to create multiple navigation cases from one table to another using the same relationship, you can select columns in multiple rows in the first table. When you make a selection in the **Related Table** list, one navigation case per row is created.

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.

5.15.3 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.



DEPARTMENT_ID
DEPARTMENT_NAME Human Resources

EMPLOYEES
DEPARTMENT_ID 40
FIRST_NAME Susan
LAST_NAME Mavris

DEPARTMENT_ID
DEPARTMENT_NAME IT

EMPLOYEES
DEPARTMENT_ID 60

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	203	Susan	Mavris	SMAVRIS	515.123.7777	2002-06-07 00:00:00	HR_REP	6500	(null)	101	

Max Rows: 1000 Max Chars: -1 Pattern: n/a 0.000/0.000 sec 1/11 1-1

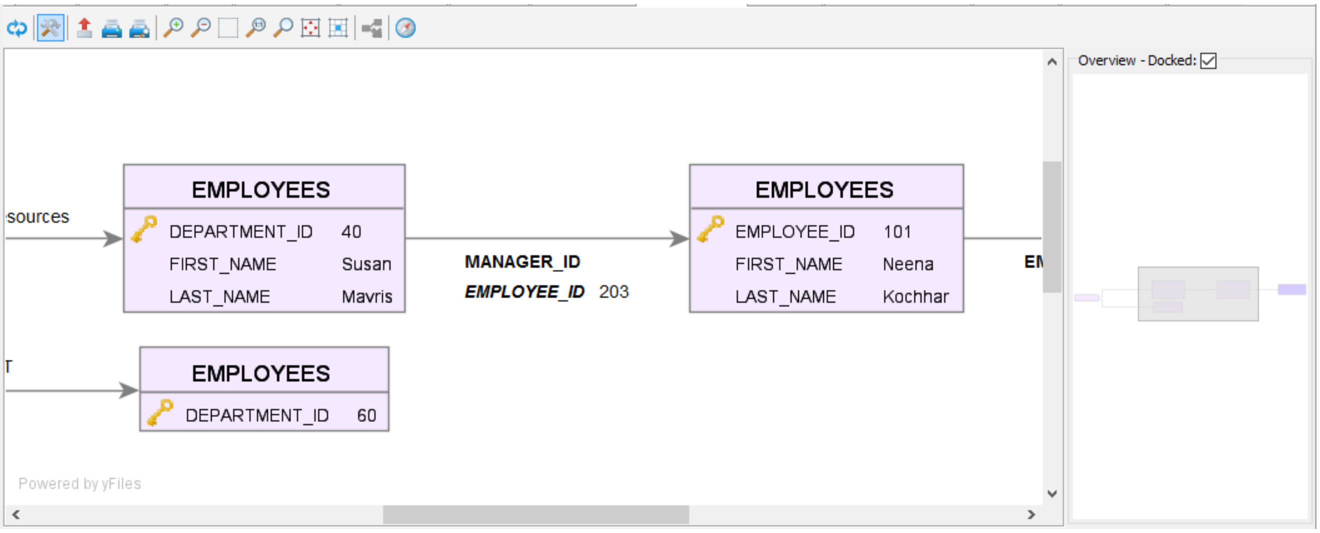
Alternatively, you can select the cells in the grid and click on the Tag button ( ) to add the cell values to the currently selected node.

5.15.4 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 move with the node.

The toolbar for the graph offers a number of tools to help you with other tasks
The **Overview** control is useful for large graphs that do not fit into the display area.



EMPLOYEES

DEPARTMENT_ID	40
FIRST_NAME	Susan
LAST_NAME	Mavris

MANAGER_ID
EMPLOYEE_ID 203

EMPLOYEES

EMPLOYEE_ID	101
FIRST_NAME	Neena
LAST_NAME	Kochhar

EMPLOYEES

DEPARTMENT_ID	60

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.

5.15.5 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
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).

5.15.6 Opening the Navigator from the Data tab

Sometimes, you may realize that you want to analyze the relationships for a table when you are working with it in the **Data** tab. If you have configured the **Data** tab to show only filtered data, sorted in a specific way, etc. opening the **Navigator** tab and making all the same configurations there may be a bit of a hassle. A more convenient way is to just pick **Show in Navigator** in the right-click menu in the **Data** tab. It opens the table in the **Navigator** tab with all the same configurations as you made in the **Data** tab.

5.16 Viewing the Table DDL

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

To see the DDL (CREATE statement) for a table:

1. Locate the table node in the **Databases** tab tree,
2. Double-click the table node to open its **Object View** tab,
3. Select the **DDL** sub tab.

The DDL shown is based on metadata retrieved from the database, but it may not include some database-specific clauses, such as storage clauses. For some databases, there is an additional sub tab named **Native DDL** (or similar) that shows the DDL as generated by the database itself, including all clauses.

5.17 Filtering Tables in the Tree

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

If you have many tables in the tree, it may be hard to find the ones of most interest. You can then define a filter so that only a few tables are shown, as described in Filtering Database Objects (see page 50).

5.18 Showing Row Count in the Tree

You can use the **Database->Show/Hide Table Row Count** menu choice to see the number of rows within parenthesis next to the table name in the Database tab tree.

Enabling this property results in a performance degradation.

5.19 Using Permissions for Table Data Editing

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **Permission** functionality is a security mechanism, where you can specify that certain database operations must be confirmed. You configure permissions in the Tool Properties dialog, in the **Permissions** category of the General tab, per *connection mode* (Development, Test and Production).

You specify which connection mode to use for a connection in the **Properties** tab of the Object View tab for the connection.

The permission feature is part of DbVisualizer and does not replace the authorization system in the actual database.

For table grid edits, you can pick the permission type from a drop-down list for each operation:

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

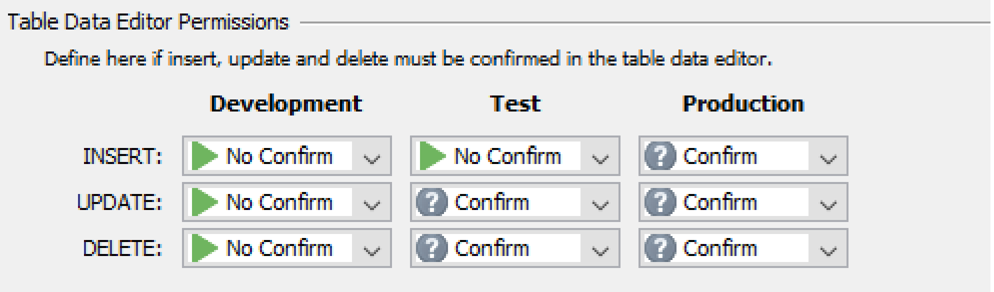


Table Data Editor Permissions

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

5.20 Scripting a Table

To open the Script Table dialog, where you can insert generated text for a table in an SQL Commander editor:

1. Select one or more table nodes in the Databases tab tree,
2. Choose **Script Table** from the right-click menu.

You can also launch the dialog by dragging and dropping one or more nodes of the same type in an SQL Commander editor.

⚠ If you just want to insert the object names in the editor, hold down the **Ctrl** key (or the **Alt** key on macOS) while dragging and dropping. This behavior can be reversed in **Tool Properties**, in the **SQL Commander** category, so that dropping without pressing a key inserts the names and pressing the key launches the dialog.

The Script dialog provides a choice of which type of statement to generate, options for formatting, use of delimited identifiers, qualified names and statement delimiters. You can also pick an open SQL Commander or a new as the destination, and where in the SQL Commander editor to insert the text.

5.21 Managing Table and Column Comments

⚠ This feature is only available for some databases. Please execute the corresponding SQL in the SQL Commander (see page 156) if it is not available for your database.

Many databases support adding comments for various database objects, such as tables and columns. How this is done is database-dependent, but for some of the databases with specific support in DbVisualizer, you can enter and edit comments using an action in the right-click menu for the object node selected in the **Databases** tab. For instance, for an Oracle database, you can manage table comments like this:

1. Select the table node in the tree,
2. Choose **Comment Table** in the right-click menu,
3. Add or edit the comment.

For Oracle, you can manage comments for other database object types, such as columns and views, in the same way. An Oracle **Tables** node also has a **Table Comments** tab in its Object Views tab, listing then comments for all tables in a schema.

6 Working with Views

DbVisualizer provides many ways to work with views.

6.1 Creating a View

There is no GUI dialog for creating a view, but you can:

1. Use the Query Builder (see page 191) to create the SELECT statement graphically,
2. Load the generated SELECT statement into the SQL Editor by clicking the corresponding button in the toolbar,
3. Add `CREATE VIEW name AS` before the SELECT statement,
4. Execute the CREATE VIEW statement.

6.2 Altering a View

Views can typically not be altered; they must be dropped and recreated. You can:

1. Select the view in the **Databases** tree,
2. Double-click the view node to open its **Object View** tab,
3. Open the **DDL** sub tab,
4. Select **Copy to New Editor** from the **DDL** tab's right-click menu, which opens an **SQL Commander** tab with the DDL,
5. Remove the CREATE VIEW part in the **SQL Commander** editor so you are left with just the SELECT statement,
6. Load the SELECT statement into the **Query Builder** and alter it graphically,
7. Launch the **Drop View** assistant from the view node's right-click menu, and click **Execute** to drop it,
8. Create the new view (see page 134) from the altered SELECT statement.

6.3 Editing a View

You can edit view data the same as you edit table data (see page 94).

6.4 Exporting a View

You can export a view the same way as you export a table (see page 109).

6.5 Viewing the View DDL

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

To see the DDL (CREATE statement) for a view:

1. Locate the view node in the **Databases** tab tree,
2. Double-click the view node to open its **Object View** tab,
3. Select the **DDL** sub tab.

6.6 Filtering Views in the Tree

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

If you have many views in the tree, it may be hard to find the ones of most interest. You can then define a filter so that only a few views are shown, as described in Filtering Database Objects (see page 50).

6.7 Scripting a View

To open the Script View dialog, where you can insert generated text for a view in an SQL Commander editor:

1. Select one or more view nodes in the Databases tab tree,
2. Choose **Script View** from the right-click menu.

You can also launch the dialog by dragging and dropping one or more nodes of the same type in an SQL Commander editor.

⚠ If you just want to insert the object names in the editor, hold down the **Ctrl** key (or the **Alt** key on macOS) while dragging and dropping. This behavior can be reversed in **Tool Properties**, in the **SQL Commander** category, so that dropping without pressing a key inserts the names and pressing the key launches the dialog.

The Script dialog provides a choice of which type of statement to generate, options for formatting, use of delimited identifiers, qualified names and statement delimiters. You can also pick an open SQL Commander or a new as the destination, and where in the SQL Commander editor to insert the text.

7 Working with Procedures, Functions and Other Code Objects

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 code 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.

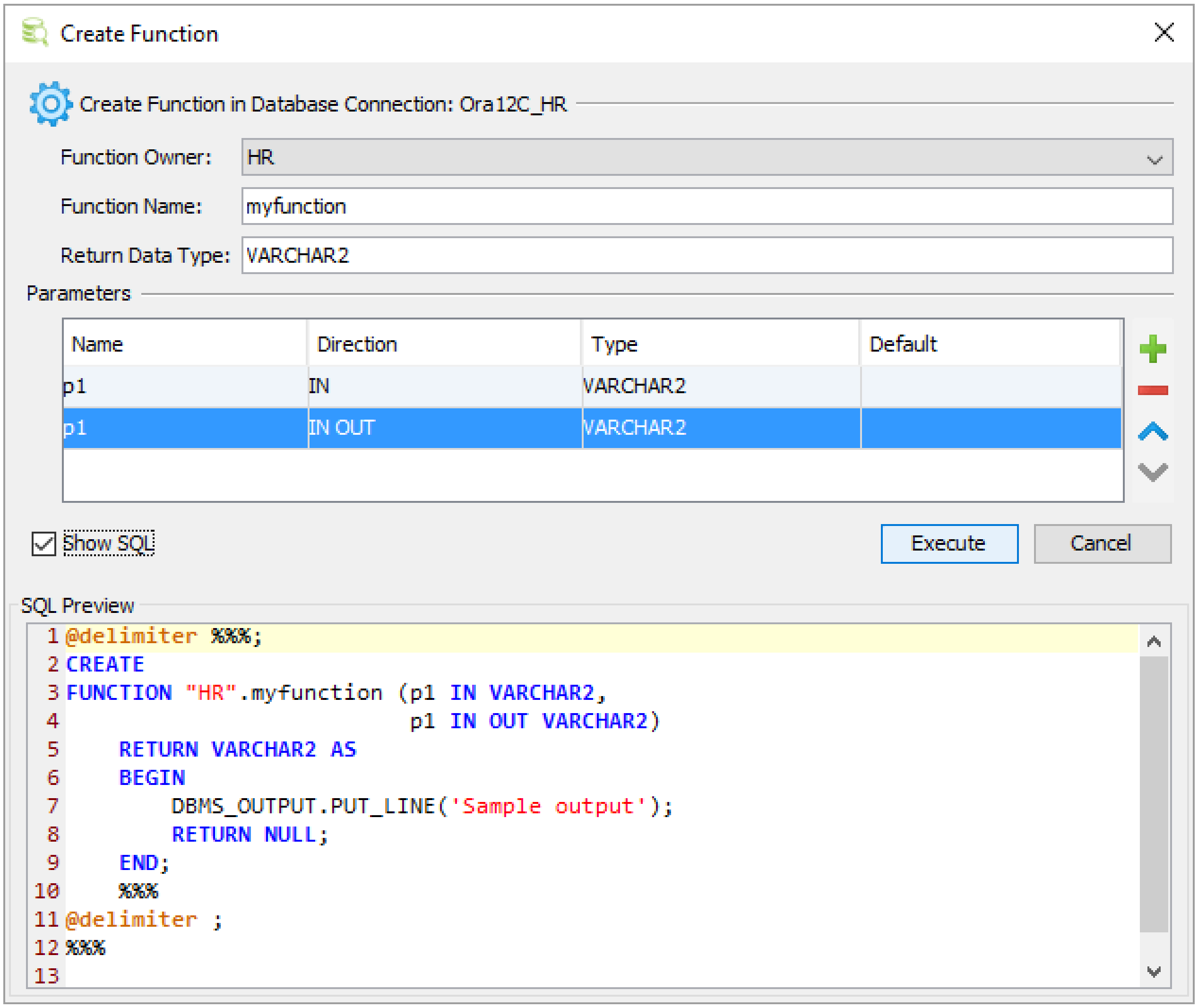
7.1 Creating a Function

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

To create a new function:

1. Expand nodes in the tree under the connection node in the **Databases** tab tree until you reach the **Functions** node,
2. Select the **Functions** node and open the **Create Function** dialog from the right-click menu.



Create Function in Database Connection: Ora12C_HR

Function Owner: HR

Function Name: myfunction

Return Data Type: VARCHAR2

Parameters

Name	Direction	Type	Default
p1	IN	VARCHAR2	
p1	IN OUT	VARCHAR2	

☑ Show SQL

```
@delimiter %%%;
CREATE
FUNCTION "HR".myfunction (p1 IN VARCHAR2,
                          p1 IN OUT VARCHAR2)
    RETURN VARCHAR2 AS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Sample output');
        RETURN NULL;
    END;
    %%%
@delimiter ;
%%%
```

The details of the dialog depends on the database, but typically you need to:

1. Enter an object name,
2. Click the **Add** button in the **Parameters** area to add parameters,
3. Enter a name and data type for each parameter. For some databases you can also enter a direction (typically IN, OUT, or INOUT) and a default value.

You can use the other buttons to the right of the parameter list to remove and move a parameter.

The dialog uses this information together with a simple sample body to compose a CREATE statement. For most databases, 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 would fit in a dialog via the Code Editor (see page 141). What you create with the assistant should be seen as a template that you then complete and work with in the editor.

For some databases the sample code is editable because there is no way to write a generic sample that compiles. You must then modify the template to something that is syntactically correct, but we still recommend that you finish the real code in the Code Editor instead.

Click **Execute** in the dialog to create the new function.

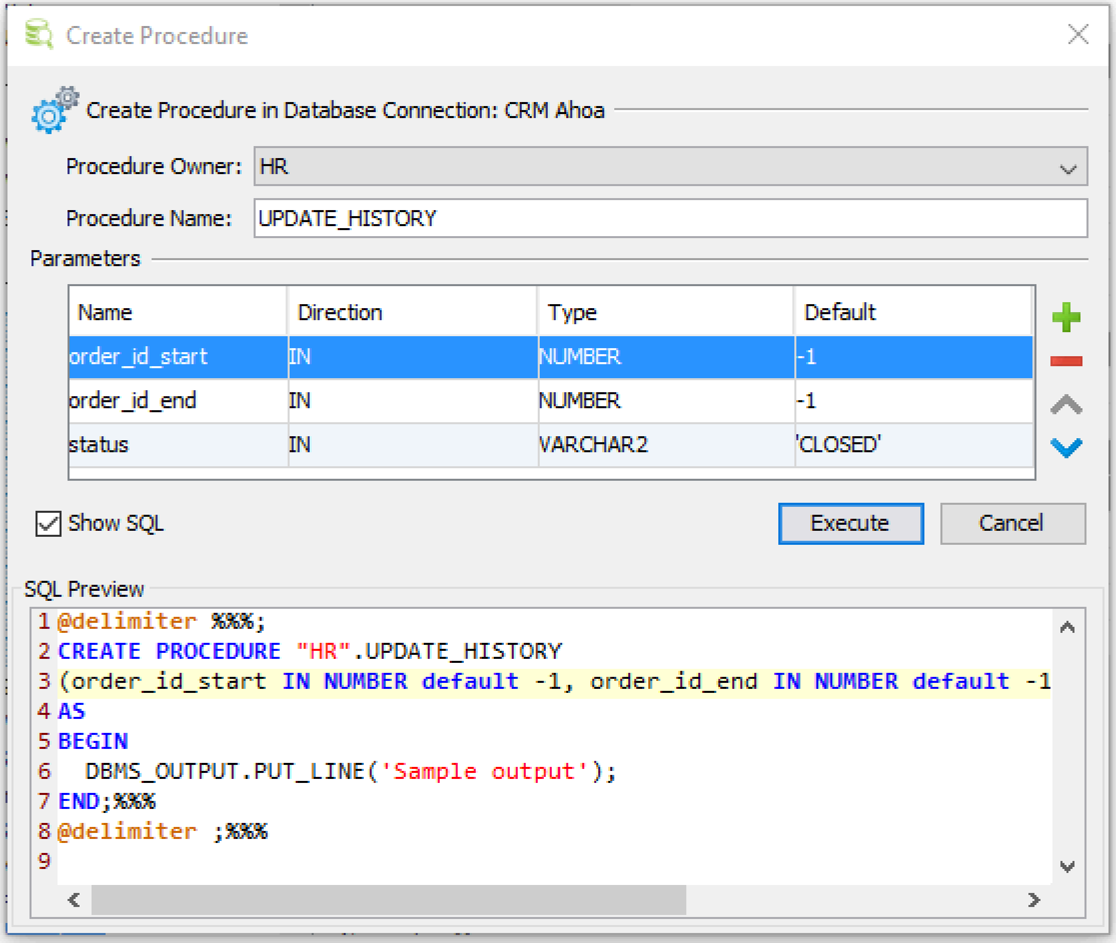
7.2 Creating a Procedure

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

To create a new procedure:

1. Expand nodes in the tree under the connection node in the **Databases** tab tree until you reach the **Procedures** node,
2. Select the **Procedures** node and open the **Create Procedure** dialog from the right-click menu.



Create Procedure in Database Connection: CRM Ahoa

Procedure Owner: HR

Procedure Name: UPDATE_HISTORY

Parameters

Name	Direction	Type	Default
order_id_start	IN	NUMBER	-1
order_id_end	IN	NUMBER	-1
status	IN	VARCHAR2	'CLOSED'

☑ Show SQL

SQL Preview

```
@delimiter %%%;
CREATE PROCEDURE "HR".UPDATE_HISTORY
(order_id_start IN NUMBER default -1, order_id_end IN NUMBER default -1
AS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Sample output');
END;%%%
@delimiter ;%%%

```

The details of the dialog depends on the database, but typically you need to:

1. Enter an object name,
2. Click the **Add** button in the **Parameters** area to add parameters,
3. Enter a name and data type for each parameter. For some databases you can also enter a direction (typically IN, OUT, or INOUT) and a default value.

You can use the other buttons to the right of the parameter list to remove and move a parameter.

The dialog uses this information together with a simple sample body to compose a CREATE statement. For most databases, 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 would fit in a dialog via the Code Editor (see page 141). What you create with the assistant should be seen as a template that you then complete and work with in the editor.

For some databases the sample code is editable because there is no way to write a generic sample that compiles. You must then modify the template to something that is syntactically correct, but we still recommend that you finish the real code in the Code Editor instead.

Click **Execute** in the dialog to create the new procedure.

7.3 Creating Other Code Objects

Only in DbVisualizer Pro

This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

Some databases support other code object types in addition to function, stored procedure and trigger, e.g. Package in Oracle and Module in Mimer.

To create a new database-specific code object:

1. Expand nodes in the tree under the connection node in the **Databases** tab tree until you reach the group node for the code type, e.g. **Packages**,
2. Select the grouping node and open the **Create** dialog from the right-click menu.

The details of the dialog depends on the database, but typically you need to:

1. Enter an object name,
2. Click the **Add** button in the **Parameters** area to add parameters,
3. Enter a name and data type for each parameter. For some databases you can also enter a direction (typically IN, OUT, or INOUT) and a default value.

You can use the other buttons to the right of the parameter list to remove and move a parameter.

The dialog uses this information together with a simple sample body to compose a CREATE statement. For most databases, 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 would fit in a dialog via the Code Editor (see page 141). What you create with the assistant should be seen as a template that you then complete and work with in the editor.

For some databases the sample code is editable because there is no way to write a generic sample that compiles. You must then modify the template to something that is syntactically correct, but we still recommend that you finish the real code in the Code Editor instead.

Click **Execute** in the dialog to create the new code object.

7.4 Editing a Code Object

Only in DbVisualizer Pro

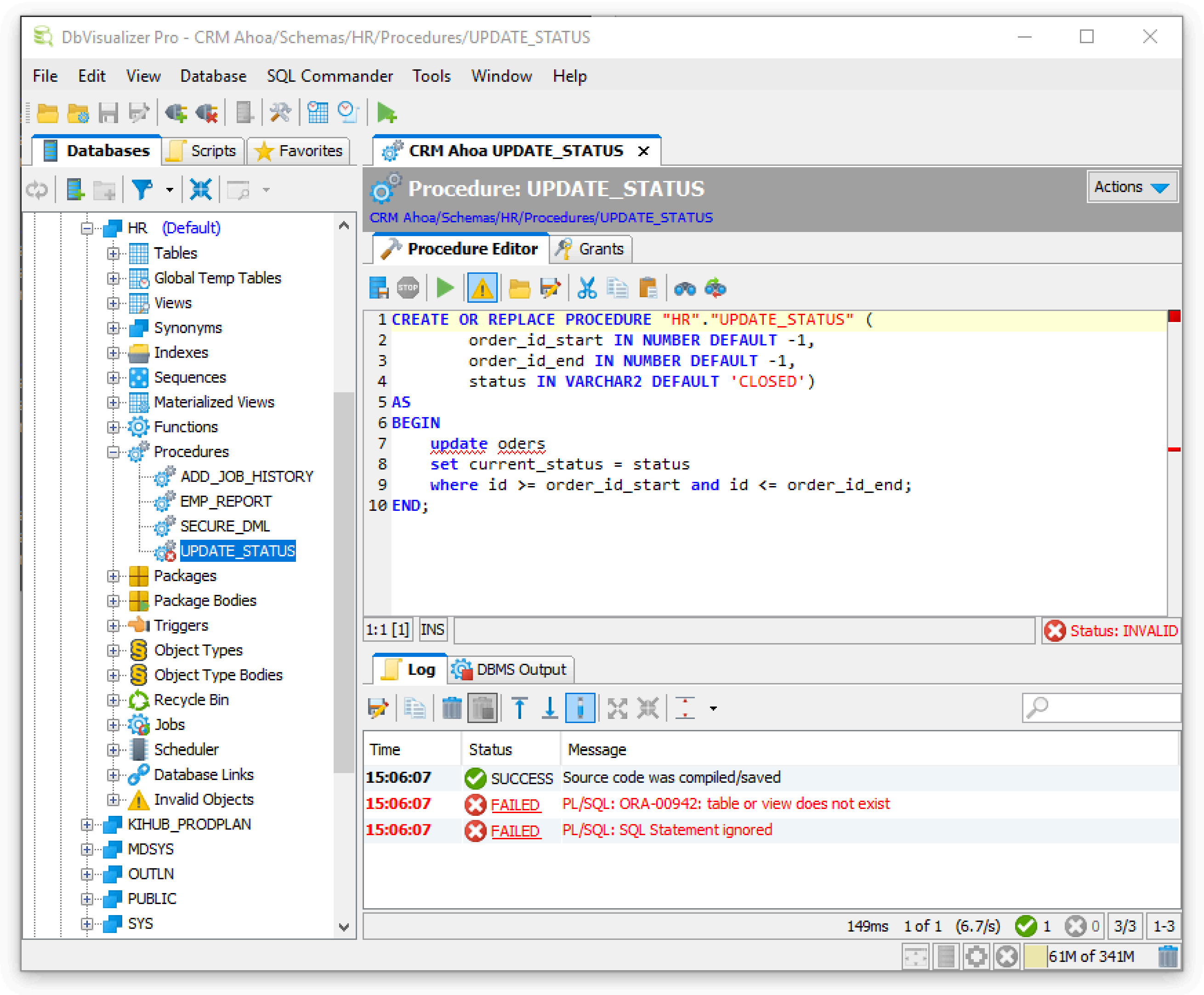
This feature is only available in the Pro edition. In the Free edition, please execute the corresponding SQL in the SQL Commander (see page 156).

To edit the code for an object, such as a function, stored procedure or database-dependent code object:

1. Expand nodes in the tree under the connection node in the **Databases** tab tree until you reach the node for the object you want to edit,
2. Double-click the node to open an **Object View** tab for it, and select the editor sub tab (e.g., the **Procedure Editor** tab).

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.



```
CREATE OR REPLACE PROCEDURE "HR"."UPDATE_STATUS" (
        order_id_start IN NUMBER DEFAULT -1,
        order_id_end IN NUMBER DEFAULT -1,
        status IN VARCHAR2 DEFAULT 'CLOSED')
AS
BEGIN
    update oders
    set current_status = status
    where id >= order_id_start and id <= order_id_end;
END;
```

Time	Status	Message
15:06:07	SUCCESS	Source code was compiled/saved
15:06:07	FAILED	PL/SQL: ORA-00942: table or view does not exist
15:06:07	FAILED	PL/SQL: SQL Statement ignored

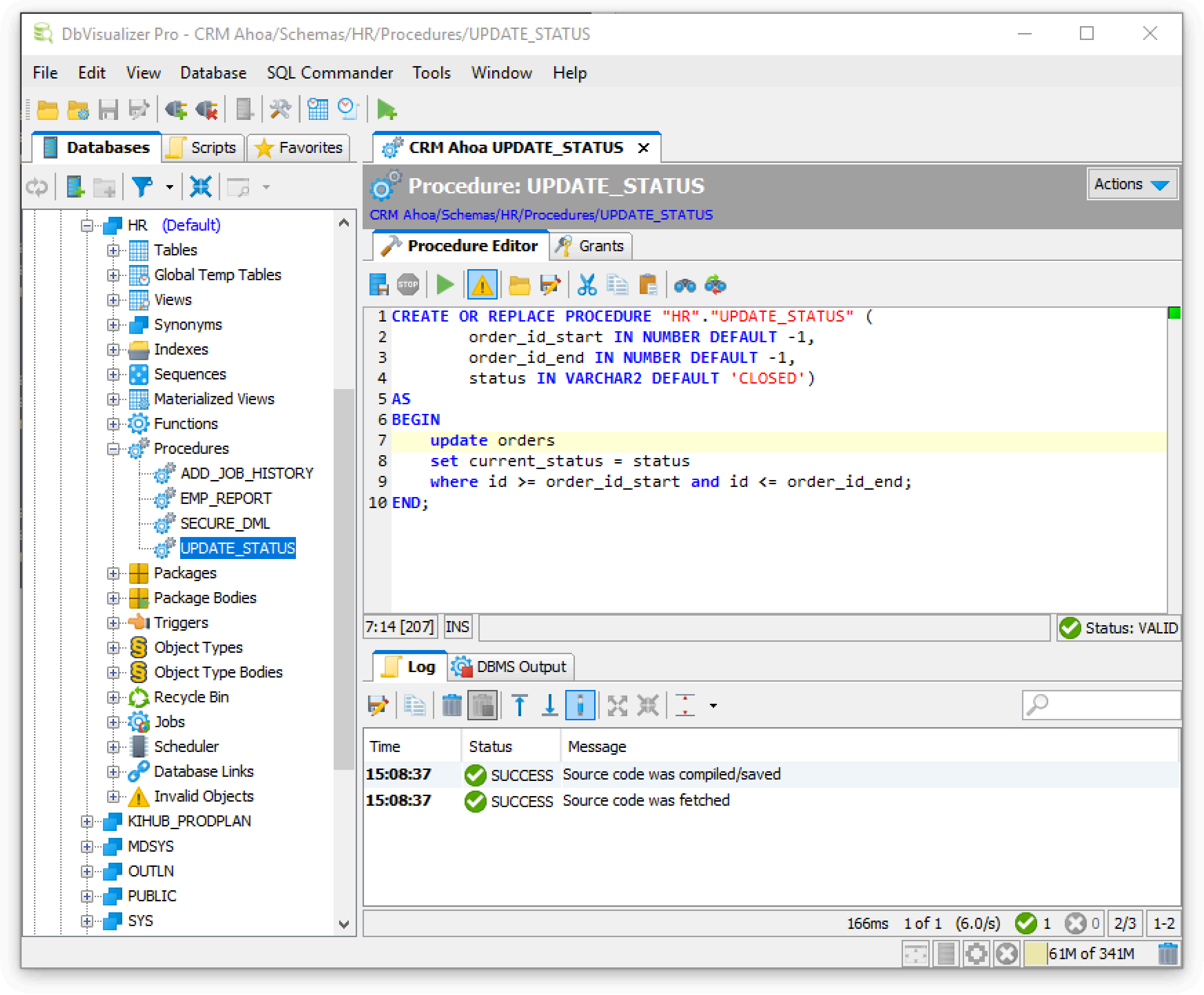
If errors occur, the corresponding text is underlined with a red wavy line. Hovering the mouse over the error indication shows the corresponding error message. The right margin contains markers for each error as well, and clicking on a marker scrolls the editor to the corresponding error. Alternatively, you can click the FAILED link in the Log tab grid to move the caret to the error location.

If you prefer to navigate between errors using the keyboard, you can use the define key bindings for the **Insertion Point to Next Marker** and **Insertion Point to Previous Marker** actions in the Tool Properties dialog, in the **Key Bindings** category, in the Editor Commands group. Alternative is to use the **Goto Next Failed** and **Goto Previous Failed** in the Log tab grid, and then click the link for the FAILED entries.

⚠ Error location information is not available for some databases. In that case the complete statement is underlined in the editor.

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 screenshot above.

The figure below shows the result after correcting the errors and recompiling the procedure:



```
CREATE OR REPLACE PROCEDURE "HR"."UPDATE_STATUS" (
        order_id_start IN NUMBER DEFAULT -1,
        order_id_end IN NUMBER DEFAULT -1,
        status IN VARCHAR2 DEFAULT 'CLOSED')
AS
BEGIN
    update orders
    set current_status = status
    where id >= order_id_start and id <= order_id_end;
END;
```

Time	Status	Message
15:08:37	SUCCESS	Source code was compiled/saved
15:08:37	SUCCESS	Source code was fetched

The status indicator now shows that the procedure is **VALID**.

7.5 Executing a Code Object

Only in DbVisualizer Pro

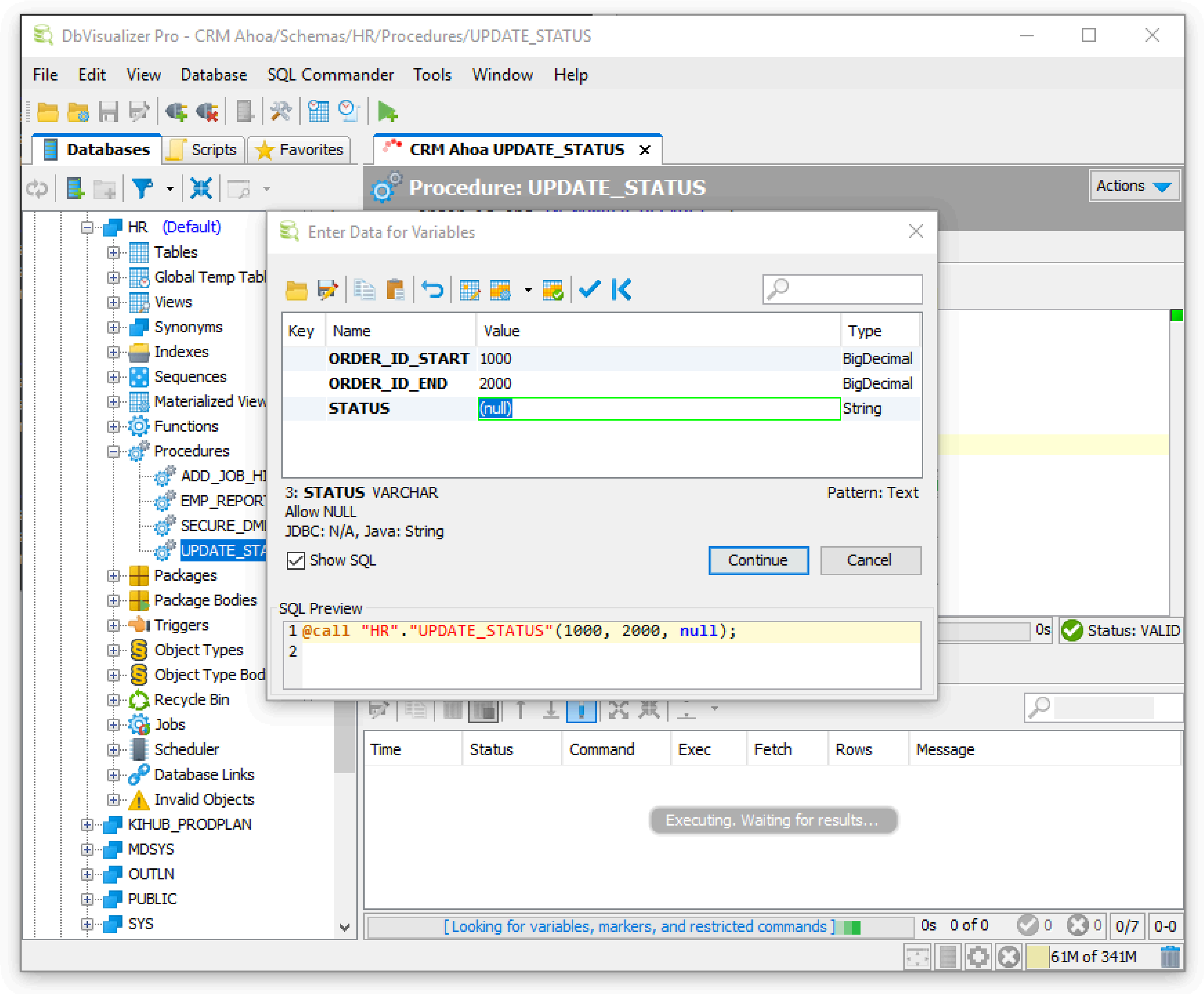
This feature is only available in the DbVisualizer Pro edition.

You can execute a code object, such as a function or stored procedure, either in the Code Editor (see page 141) or in the SQL Commander (see page 156).

- Executing in the Code Editor (see page 145)
- Executing in the SQL Commander (see page 146)
- Using the Script Object Dialog (see page 147)

7.5.1 Executing in the Code Editor

In the Code Editor, click the **Execute** button. DbVisualizer then generates a script for executing the code object, using variables for all parameters, and executes it.



Key	Name	Value	Type
	ORDER_ID_START	1000	BigDecimal
	ORDER_ID_END	2000	BigDecimal
	STATUS	(null)	String

3: **STATUS** VARCHAR
Allow NULL
JDBC: N/A, Java: String

Pattern: Text

☑ Show SQL

```
@call "HR"."UPDATE_STATUS"(1000, 2000, null);
```

0s
Status: VALID
Time
Status
Command
Exec
Fetch
Rows
Message
Executing. Waiting for results...
[Looking for variables, markers, and restricted commands]
0s 0 of 0
0
0
0/7
0-0
61M of 341M

Because the script contains variables, the **Variable Prompt** dialog pops up. Enter values for all parameters and click **Continue** to execute the procedure. The result is shown in the results area below the editor.

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. In this case, the generated script includes `@echo` statements to write the result in the **Log** tab. Please see below for more details.

7.5.2 Executing in the SQL Commander

The scripts generated and executed by the Code Editor can also be included in a script and executed in an SQL Commander. Here's an example of a script calling a function and writing the result in the **SQL Commander Log** tab:

```
@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 is 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.

More formally, the `@call` command has this syntax when calling a function:

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

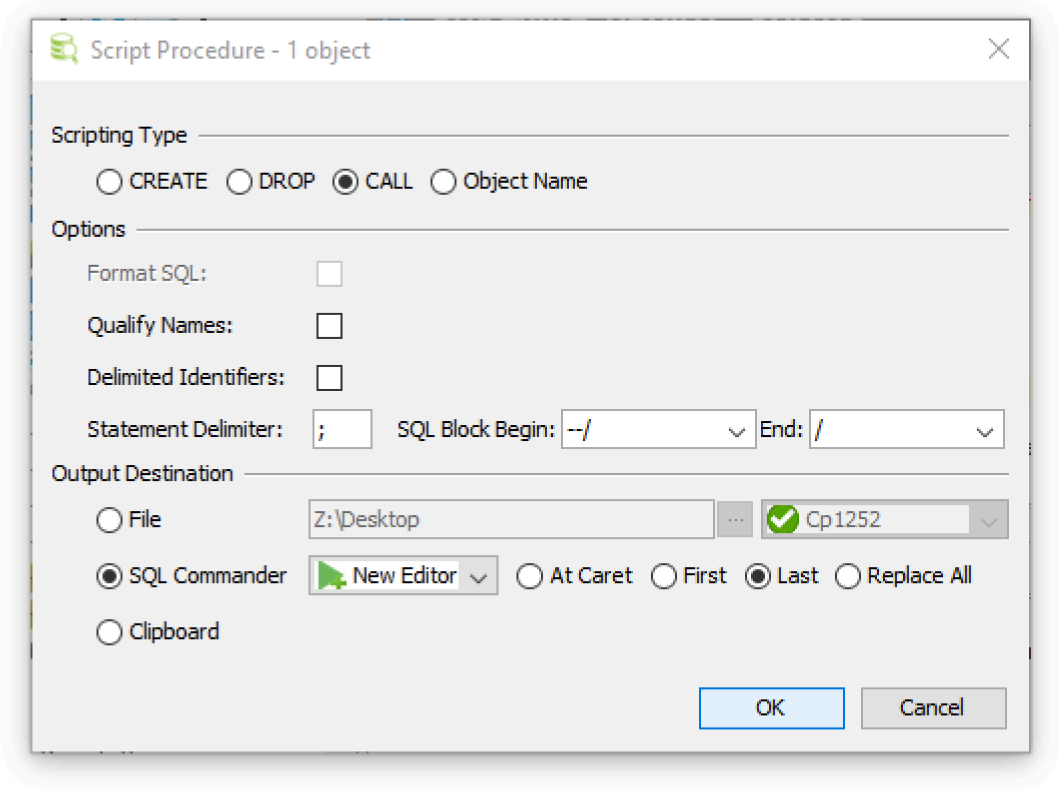
As shown in these examples, you must use the `dir` option to specify how the variable is to be used (in, out or inout) and you may use the `noshow` option to prevent being prompted for a value for an output variable.

You can use the @echo command (see page 212) write the value assigned to an output variable to the log.

7.5.3 Using the Script Object Dialog

Instead of writing a `@call` script by hand in an SQL Commander, you can use the Script Object (e.g. **Script Procedure** or **Script Function**) right-click menu choices for the object node in the tree.

This opens the Script Object dialog where you select that you want to generate a CALL script and can adjust settings for using delimiters and qualifiers, as well as the destination for the generated script.



Script Procedure - 1 object

Scripting Type

○ CREATE ○ DROP ● CALL ○ Object Name

Options

Format SQL: □

Qualify Names: □

Delimited Identifiers: □

Statement Delimiter: ; SQL Block Begin: --/ End: /

Output Destination

○ File Z:\Desktop ... Cp1252

● SQL Commander New Editor ○ At Caret ○ First ● Last ○ Replace All

○ Clipboard

7.6 Exporting a Code Object

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

To export a code object:

1. Select the object node in the **Databases** tab tree,
2. Open the Export Object dialog (e.g. **Export Procedure** or **Export Function**) from the right-click menu,
3. Select an **Output Format**, **Output Destination**, and **Options**,
4. Click **Export**.

For these object types, you can select either the **SQL** (CREATE statement) or **XML** Output Format and which delimiters to use in the Options area.

You can control whether to use delimited identifiers and/or qualified names (see page 279) in the DDL and INSERT statements generated for the SQL format.

7.7 Scripting a Code Object

To open the Script Function/Procedure dialog, where you can insert generated text for a code object in an SQL Commander editor:

1. Select one or more code object nodes in the Databases tab tree,
2. Choose **Script Function/Procedure** from the right-click menu.

You can also launch the dialog by dragging and dropping one or more nodes of the same type in an SQL Commander editor.

⚠ If you just want to insert the object names in the editor, hold down the **Ctrl** key (or the **Alt** key on macOS) while dragging and dropping. This behavior can be reversed in **Tool Properties**, in the **SQL Commander** category, so that dropping without pressing a key inserts the names and pressing the key launches the dialog.

The Script dialog provides a choice of which type of statement to generate, options for formatting, use of delimited identifiers, qualified names and statement delimiters. You can also pick an open SQL Commander or a new as the destination, and where in the SQL Commander editor to insert the text.

8 Working with Schemas

DbVisualizer provides many ways to work with schemas.

8.1 Creating a Schema

To create a new schema:

1. Locate the **Schemas** node in the Databases tab tree,
2. Open the **Create Schema** dialog from the right-click menu,
3. Enter all required information (database dependent),
4. Click **Execute** to create the schema.

⚠ This feature is only available for some databases. Please execute the corresponding SQL in the SQL Commander (see page 156) if it is not available for your database.

8.2 Comparing Schemas

ℹ **Only in DbVisualizer Pro**

This feature is only available in the DbVisualizer Pro edition.

While DbVisualizer does not provide a way to fully show the differences between two schemas in terms of objects and data, you can compare different aspects of a schema to another schema one by one.

For instance, to compare the list of tables in one schema with the list of tables in another schema:

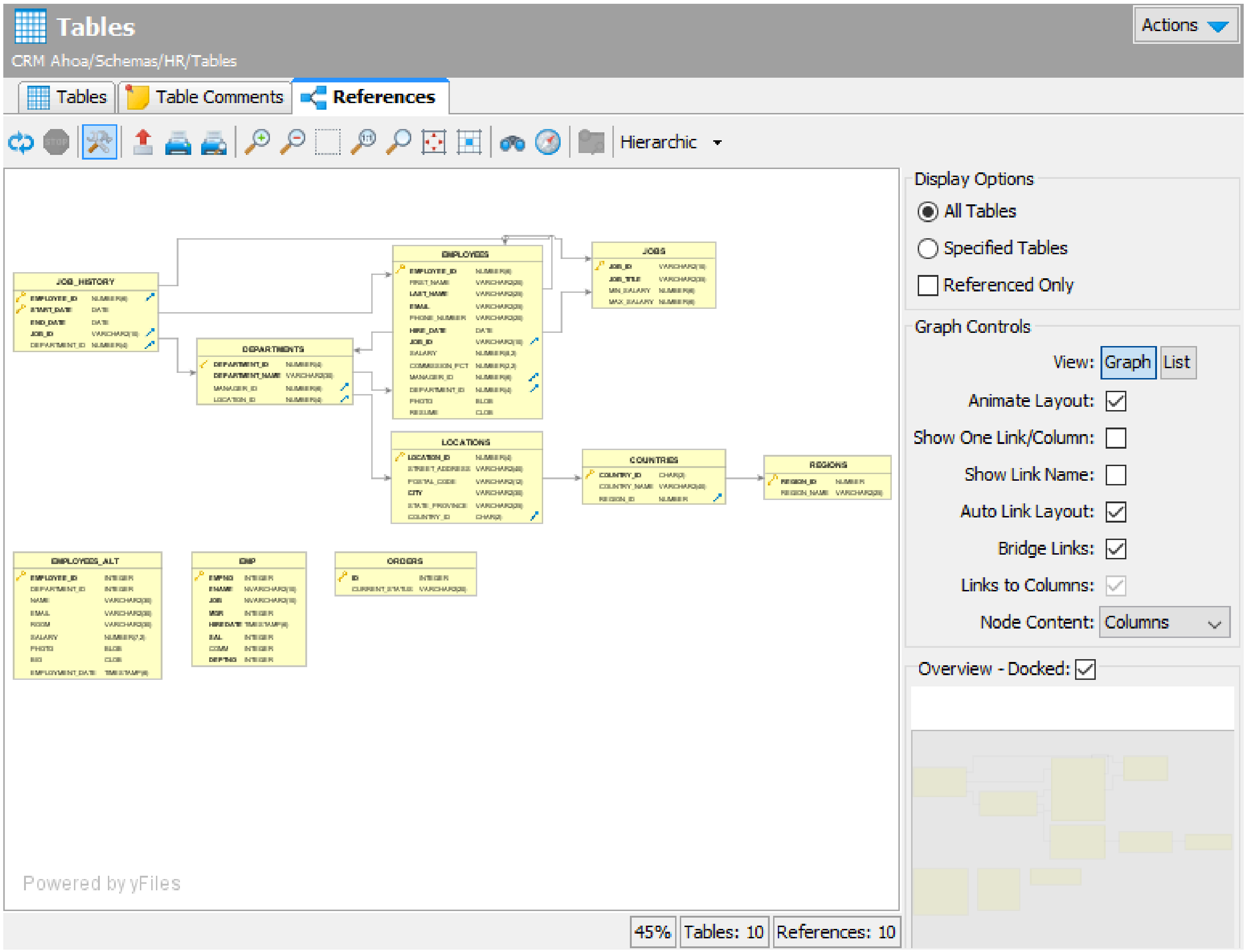
1. Double-click the schema node for the first schema to open its Object View tab and select the **Tables** tab,
2. Do the same for the second schema,
3. Select **Compare** from the right-click menu in one of the Tables tabs to compare their grid content (see page 262).

You can do the same for all the other schema object types, such as views and stored procedures. You can also dig deeper and compare the individual objects, such as comparing individual tables (see page 122).

8.3 Viewing Entity Relationships

To see how a table is related to other tables through Foreign Keys:

1. Locate the **Tables** node in the **Databases** tab tree,
2. Double-click the node to open its **Object View** tab,
3. Select the **References** sub tab.



Tables

CRM Ahoa/Schemas/HR/Tables

Display Options

- ● All Tables
- ○ Specified Tables
- □ Referenced Only

Graph Controls

- View: Graph | List
- Animate Layout: ☑
- Show One Link/Column: □
- Show Link Name: □
- Auto Link Layout: ☑
- Bridge Links: ☑
- Links to Columns: ☑
- Node Content: Columns

Overview - Docked: ☑

45% | Tables: 10 | References: 10

You can select among different graph layouts in the layout drop-down list in the toolbar: **Hierarchic**, **Organic**, **Orthogonal**, or **Circular**.

Other layout settings can be changed in the **Graph Control** area, which is shown or hidden with the settings toggle button in the toolbar. For instance, you can select how much information to include for each table in the graph: just the **Table Name**, the **Primary Key** column(s) or all **Columns**.

The graph can be **Exported** to a file in **JPG**, **GIF**, **PNG**, **SVG** or **PDF** or **Saved** as a Graph Modeling Language (**GML**) file that you can then open in the yEd (http://www.yworks.com/en/products_yed_about.html) tool from yWorks for further manipulation.

You can control whether the table names should be qualified with the schema/catalog (see page 279) in the graph.

8.4 Exporting a Schema

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

You can export all or selected objects in a schema using the Export Schema assistant.

- Output Format (see page 152)
- Output Destination (see page 153)
- Object Types (see page 153)
- Options (see page 153)
- Saving And Loading Settings (see page 154)

To export a schema:

1. Select the schema node in the **Databases** tab tree,
2. Launch the **Export Schema** assistant from the right-click menu,
3. Select an **Output Format**, **Output Destination**, **Objects** to export and **Options**,
4. Click **Export**.

8.4.1 Output Format

You can export objects in one of these formats: **CSV**, **HTML**, **SQL**, **XML**, **XLS** (Excel), or **JSON**.

The **CSV**, **HTML**, **XSL** and **JSON** formats are specifically for table data and are not supported for any other type of objects.

The **SQL** and **XML** formats can be used for all objects to export the DDL, and for tables you can also choose to include the table data in these formats.

You can control whether to use delimited identifiers and/or qualified names (see page 279) in the DDL and INSERT statements generated for the SQL format, using the controls in the Options area.

8.4.2 Output Destination

The destination can be one of:

- a file,
- an open or new SQL Commander tab, with options for where in an open SQL Commander to insert the result,
- to the system clipboard.

8.4.3 Object Types

In the **Object Types** area you select what to export. You can check the checkbox for an object type to export all objects of that type, or expand a type node and select individual objects. To select all objects, just check the checkbox for the schema itself at the top of the tree.

8.4.4 Options

The Options are contains options common to all Output Formats at the top, followed by options for the selected format.

For the SQL and XML formats, you can choose to export the DDL, the DDL for indexes for a table and the table data: as INSERT statements for the SQL statement or in one of three XML formats.

For the XLS format, you can choose to export table data as either regular Binary Excel or OOXML for Excel 2007 and later.

Most formats also let you specify other options, such as delimiters, title and descriptions. Just select an Output Format to see which options are available. All options are described in the context of the @export command (see page 214), as the Export dialog is just a GUI for the command.

You can adjust the **Data Formats** specifically for the exported table data. By default, the formats defined in **Tool Properties** are used, but sometimes you need to export dates and numbers in a different format because you intend to import the data into a different type of database.

> ⓘ If you are exporting table data in the SQL format from one database type (e.g. Oracle) to import it in a database of a different type (e.g. PostgreSQL) by executing the generated script, you need to be aware of differences in the literal formats for Date, Time and Timestamp data. If you connect to the other database using a JDBC client like DbVisualizer, you can select the **JDBC escape** format for these data format. This generates literals that the JDBC driver converts into a format the target database can interpret.

In the **Data Format Settings** dialog you can also specify how to quote text data and how to handle quotes within the text value.

Using Variables in Fields

You can use some of the pre-defined DbVisualizer variables (see page 227) (`${dbvis-date}$`, `${ dbvis-time}$`, `${dbvis-timestamp}$`, `${dbvis-connection}$`, `${dbvis-database-type}$` and `${ dbvis-object}$`) in all fields that hold free text (e.g. title and description fields) and as part of the file name field.

Use the **${dbvis-object}$** variable as part of the filename 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.

8.4.5 Saving And Loading Settings

If you often use the same settings, you can save them as the default settings for this assistant. If you use a number of common settings, you can save them to individual files that you can load as needed. Use the Settings button menu to accomplish this:

- **Save as Default Settings**
  Saves all format settings as default. These are then loaded automatically when open an Export Schema dialog
- **Use Default Settings**
  Use this choice to initialize the settings with default values
- **Remove Default Settings**
  Removes the saved defaults and restores the regular defaults
- **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.

8.5 Filtering Schemas in the Tree

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

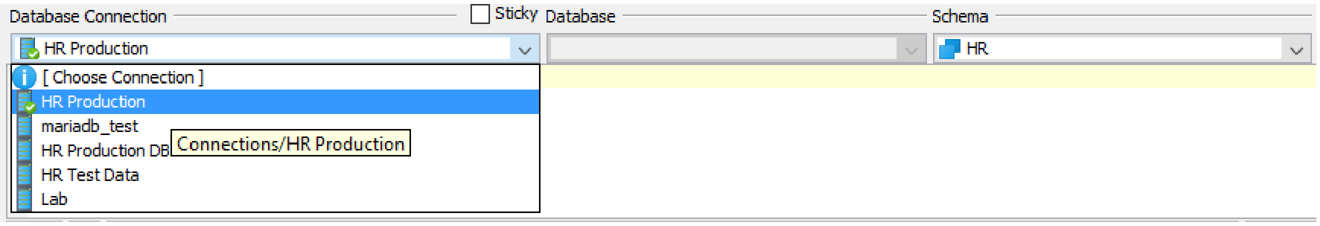
If you have many schema in the tree, it may be hard to find the ones of most interest. You can then define a filter so that only a few schemas are shown, as described in Filtering Database Objects (see page 50).

9 Working with SQL

With DbVisualizer, you can use a powerful SQL editor or a graphical Query Builder to create and edit your scripts, save them as Bookmarks for easy access, and execute all or just a few of the statements, and lots more.

9.1 Selecting 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 SQL Commander. The list of connections shows all connections as they are ordered in the Databases tab tree, except that all currently active connections are listed first.



Database Connection: HR Production
□ Sticky
Database:
Schema: HR

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 (see page 187). 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. You can specify if you want to have Sticky enabled by default in the Tool Properties dialog, in the **SQL Commander** category under the General tab.

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 LUW, DB2 z/OS, DB2 iSeries, H2, JavaDB/Derby, Oracle, NuoDB, PostgreSQL, and Vertica. If the Database Type is set to **Generic** for a connection, DbVisualizer tries to set the default schema (if **Use Schema** is chosen) but it depends on the JDBC driver if this works or not.

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 **Properties** tab for the connection, in the **SQL Commander** category.

9.1.1 Configuring the Initial Values

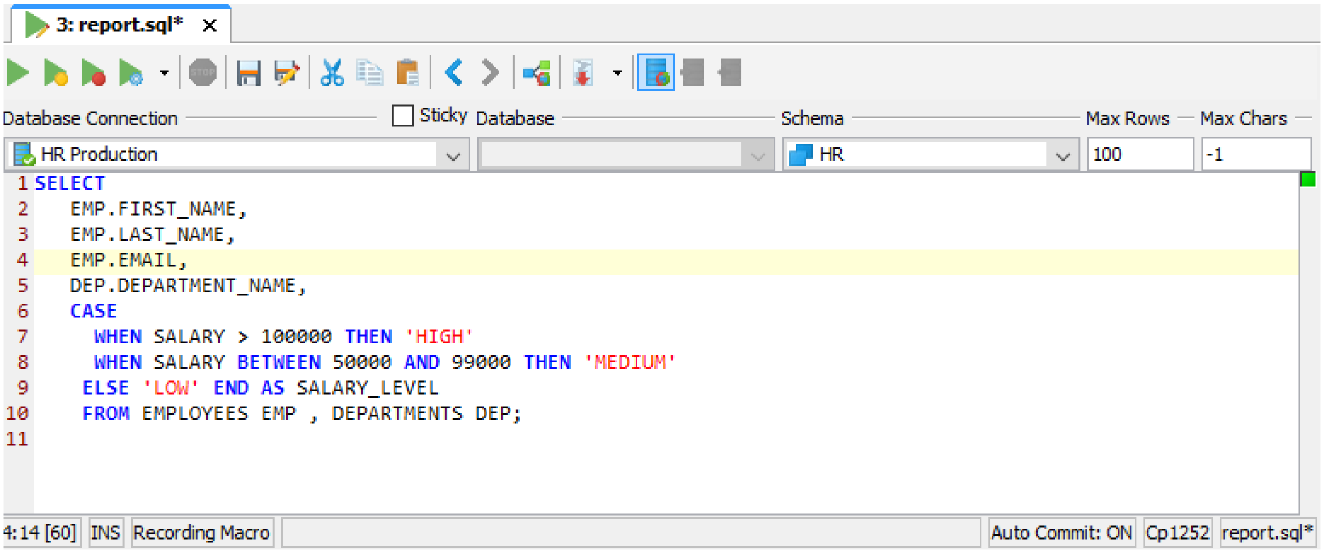
When a new Database Connection is selected, the initial values for the **Database** and **Schema** lists are by default based on the default values for the connection. You can change this behavior in the **Properties** tab for the connection, in the **SQL Commander** category. The initial value for each list that is applicable for the database type can be set to one of **The Connection Default**, **None** or **The Most Recently Used**.

9.2 Editing SQL Scripts

The SQL Commander contains an SQL editor, used to edit SQL scripts.

- Syntax Color Coding (see page 159)
- Charsets and Fonts (see page 161)
- Loading and Saving Scripts (see page 161)
- Drag and Drop a File (see page 162)
- Drag and Drop Database Objects (see page 162)
- Loading and Saving Bookmarks and Monitors (see page 163)
- Navigating Between History Entries (see page 164)
- Confirming Overwriting Unsaved Changes (see page 164)
- SQL Formatting (see page 164)
- Auto Completion (see page 166)
- Recording and Playing Edit Macros (see page 169)
- Folding Selected Text (see page 170)
- Selecting a Rectangular Area (see page 171)
- Tab Key Treatment (see page 171)
- Key Bindings (see page 172)

The editor area looks like this:



□ Sticky

HR Production | | HR | 100 | -1

```
SELECT
   EMP.FIRST_NAME,
   EMP.LAST_NAME,
   EMP.EMAIL,
   DEP.DEPARTMENT_NAME,
   CASE
     WHEN SALARY > 100000 THEN 'HIGH'
     WHEN SALARY BETWEEN 50000 AND 99000 THEN 'MEDIUM'
    ELSE 'LOW' END AS SALARY_LEVEL
    FROM EMPLOYEES EMP , DEPARTMENTS DEP;
```

4:14 [60] | INS | Recording Macro | Auto Commit: ON | Cp1252 | report.sql*

Above the editor is a toolbar with buttons related both to execution of scripts and to editing. The editing related buttons are covered below.

The left margin shows the line numbers.

Below the editor, you see a Status Bar. The first field shows the current caret position in the format:

<line>:<column> [<position from top>]

The last figure, within square brackets, is the caret position from the top. This can be useful when you get an error message executing a script that contains this information rather than a line/column location.

The next field in the Status Bar shows INS if characters you type will be inserted at the caret position or OVR if they will overwrite the current text at the caret position. You can toggle this mode using the **Toggle Typing Mode** keyboard shortcut, by default bound to the **Insert** key.

The next field shown in the screenshot is only visible when working with macros, described in the Recording and Playing Edit Macros (see page 169) section.

Next comes the Auto Commit Status field, showing whether Auto Commit (see page 185) is enabled.

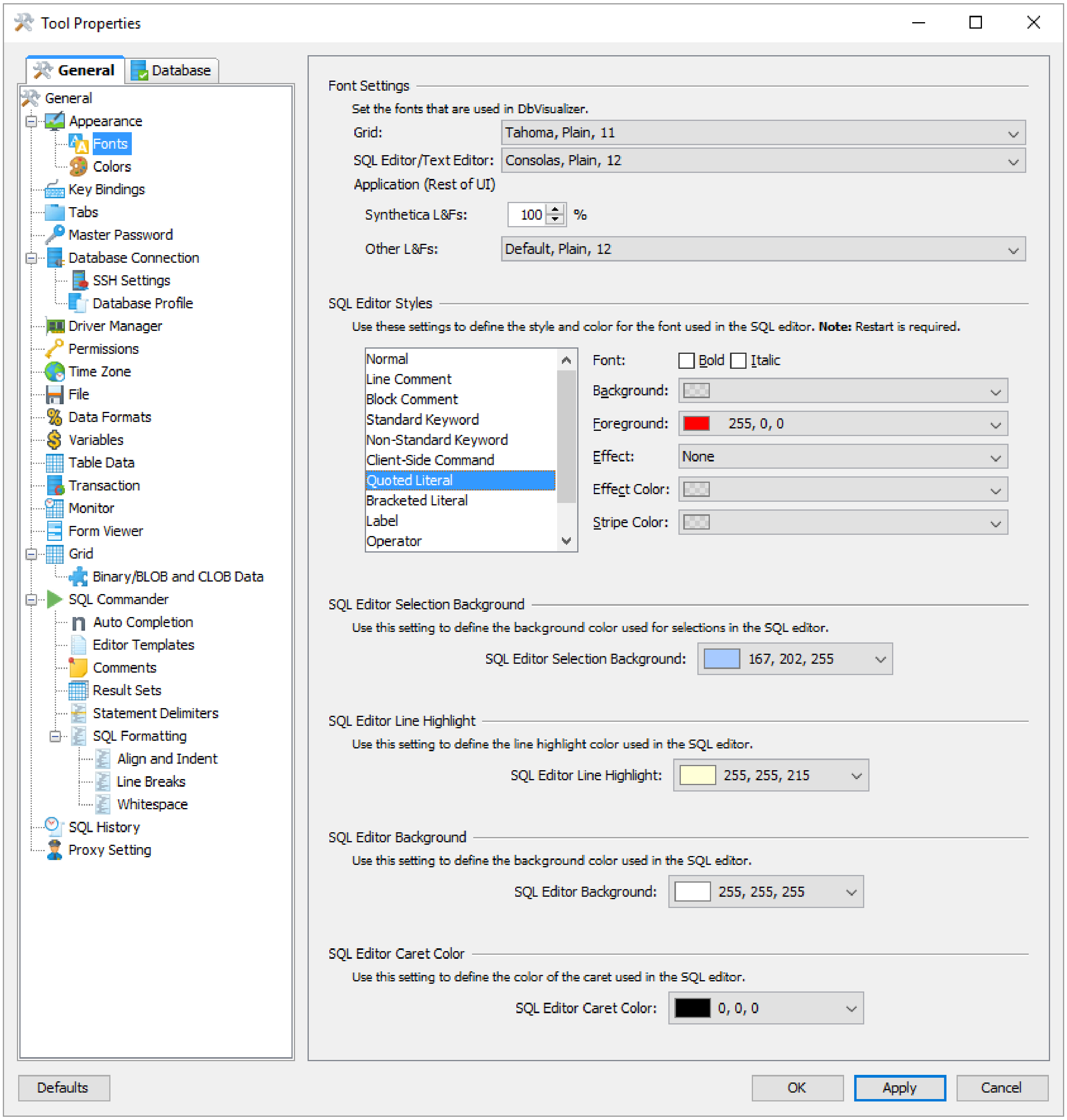
The last two fields show information about the file loaded in the editor, if any. First the character encoding and then the filename. You can click on the filename to copy the file path or open the OS file chooser for the directory holding the file. If you just type into the editor without loading a file, the filename "Untitled" is shown instead. An asterisk after the filename indicates that there are unsaved edits.

The SQL Editor is like any editor you're used to when it comes to typing, scrolling etc. But it also offers additional features to help you specifically with editing SQL scripts. These are described in the following sections .

9.2.1 Syntax Color Coding

An SQL script consists of keywords, operators, object identifiers, quoted text, etc. It may also contain comments. To make it easier to see at a glance what is what, the SQL Editor displays words using different font styles depending on their classification. For instance, keywords are displayed with a bold blue font, while quoted text is displayed with a regular type red font.

You can change how to display the different kinds of words, as well as the editor selection background color, the current line highlight color and the editor background color, in the **Tools->Tool Properties** dialog, in the **Appearance/Fonts** category.



Tool Properties

Font Settings

Set the fonts that are used in DbVisualizer.

Grid: Tahoma, Plain, 11

SQL Editor/Text Editor: Consolas, Plain, 12

Application (Rest of UI)

Synthetica L&Fs: 100 %

Other L&Fs: Default, Plain, 12

SQL Editor Styles

Use these settings to define the style and color for the font used in the SQL editor. **Note:** Restart is required.

Font: □ Bold □ Italic

Background:

Foreground: 🟥 255, 0, 0

Effect: None

Effect Color:

Stripe Color:

SQL Editor Selection Background

Use this setting to define the background color used for selections in the SQL editor.

SQL Editor Selection Background: 167, 202, 255

SQL Editor Line Highlight

Use this setting to define the line highlight color used in the SQL editor.

SQL Editor Line Highlight: 255, 255, 215

SQL Editor Background

Use this setting to define the background color used in the SQL editor.

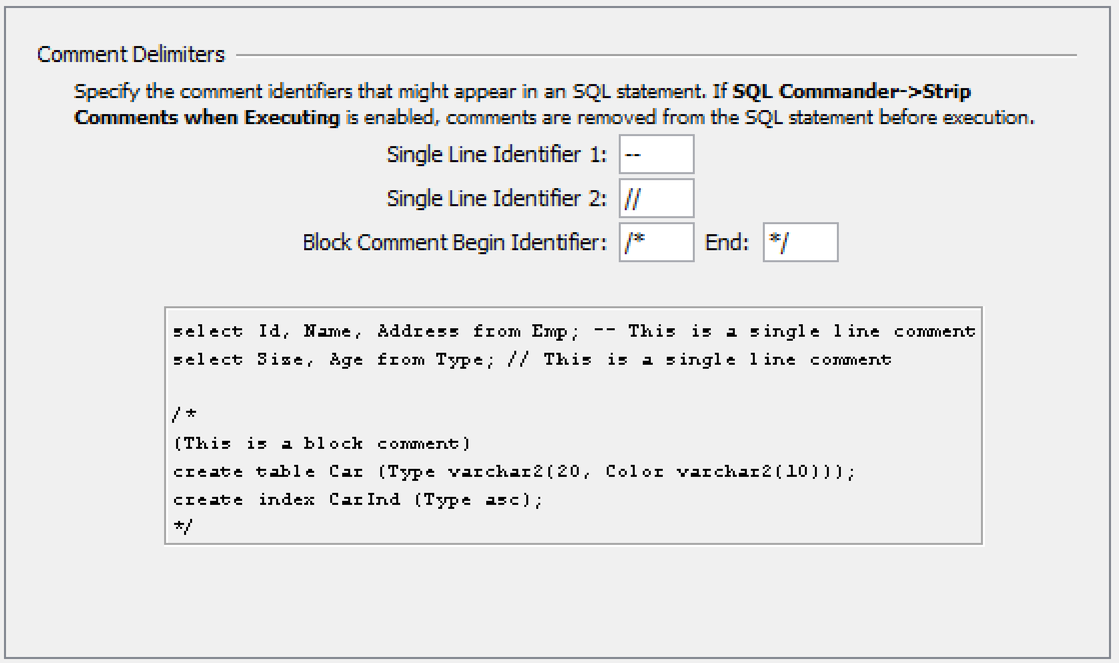
SQL Editor Background: 255, 255, 255

SQL Editor Caret Color

Use this setting to define the color of the caret used in the SQL editor.

SQL Editor Caret Color: ■ 0, 0, 0

The editor uses the Tool Properties settings from the **SQL Commander/Comments** category under the General tab to detect comments.



Comment Delimiters

Specify the comment identifiers that might appear in an SQL statement. If **SQL Commander->Strip Comments when Executing** is enabled, comments are removed from the SQL statement before execution.

Single Line Identifier 1: --

Single Line Identifier 2: //

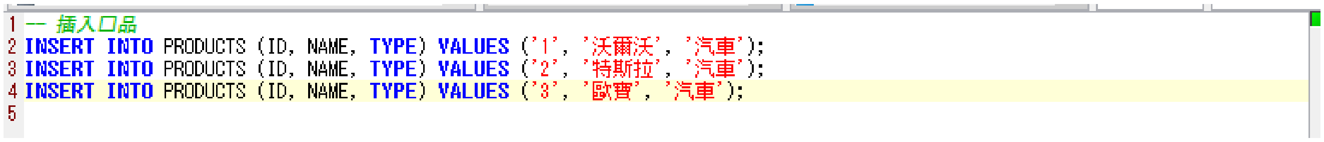
Block Comment Begin Identifier: /* End: */

```
select Id, Name, Address from Emp; -- This is a single line comment
select Size, Age from Type; // This is a single line comment

/*
(This is a block comment)
create table Car (Type varchar2(20, Color varchar2(10)));
create index CarInd (Type asc);
*/
```

9.2.2 Charsets and Fonts

You can also change the SQL Editor font family, which is useful and necessary in order to display characters for languages like Chinese, Japanese, etc., in **Tool Properties** in the **Appearance/Fonts** category to set the font for the SQL Editor.



```
-- 插入口品
INSERT INTO PRODUCTS (ID, NAME, TYPE) VALUES ('1', '沃爾沃', '汽車');
INSERT INTO PRODUCTS (ID, NAME, TYPE) VALUES ('2', '特斯拉', '汽車');
INSERT INTO PRODUCTS (ID, NAME, TYPE) VALUES ('3', '歐寶', '汽車');
```

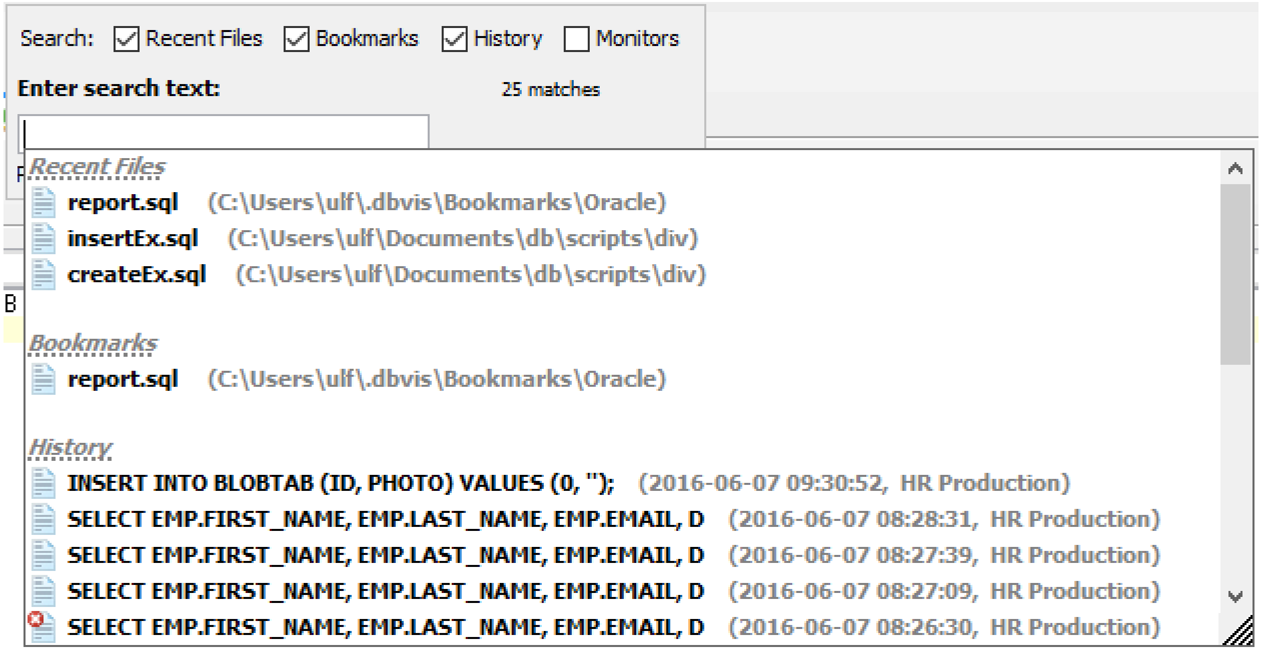
9.2.3 Loading and Saving Scripts

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 main toolbar to accomplish this. Loading a file loads it into a new **SQL Commander** tab or activates the tab that already holds it.

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 close the SQL Commander tab or exit DbVisualizer, you are asked what to do if there are any pending edits that need to be saved.

The **File->Open Recent** submenu lists the recently loaded files. How many recent files to keep track of can be specified in the Tool Properties dialog, in the **SQL Commander** category under then General tab.

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 main toolbar button as well as in the main **File->Quick File Open** menu.



Search: ☑ Recent Files ☑ Bookmarks ☑ History □ Monitors

Enter search text: 25 matches

Recent Files
- report.sql (C:\Users\ulf\.dbvis\Bookmarks\Oracle)
- insertEx.sql (C:\Users\ulf\Documents\db\scripts\div)
- createEx.sql (C:\Users\ulf\Documents\db\scripts\div)

Bookmarks
- report.sql (C:\Users\ulf\.dbvis\Bookmarks\Oracle)

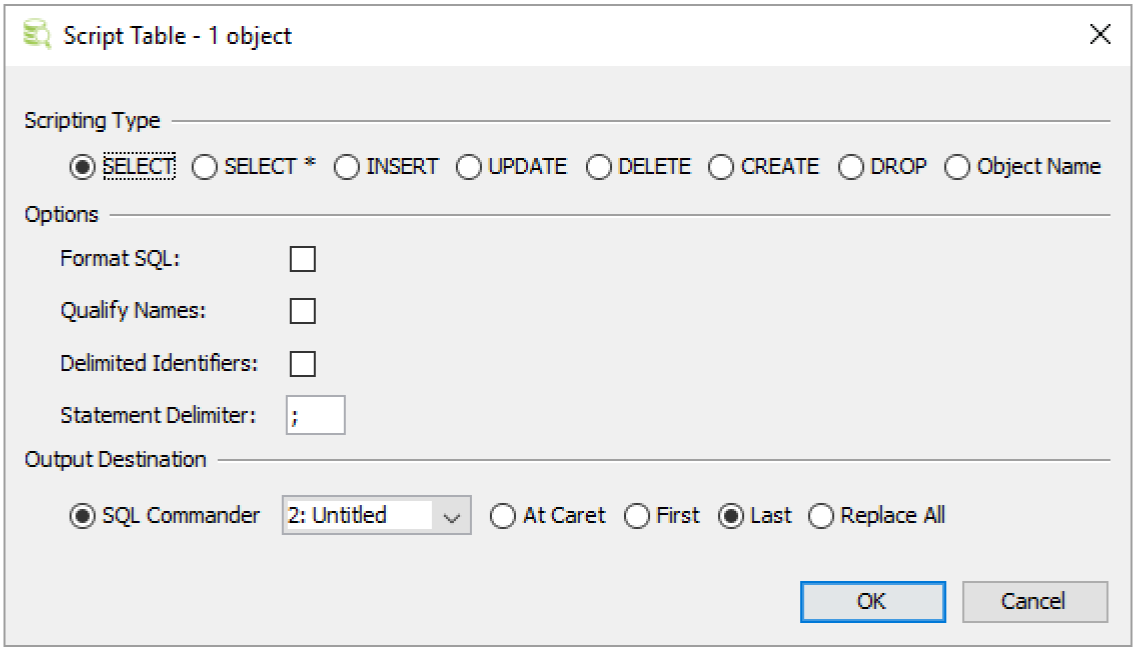
History
- INSERT INTO BLOBTAB (ID, PHOTO) VALUES (0, ''); (2016-06-07 09:30:52, HR Production)
- SELECT EMP.FIRST_NAME, EMP.LAST_NAME, EMP.EMAIL, D (2016-06-07 08:28:31, HR Production)
- SELECT EMP.FIRST_NAME, EMP.LAST_NAME, EMP.EMAIL, D (2016-06-07 08:27:39, HR Production)
- SELECT EMP.FIRST_NAME, EMP.LAST_NAME, EMP.EMAIL, D (2016-06-07 08:27:09, HR Production)
- SELECT EMP.FIRST_NAME, EMP.LAST_NAME, EMP.EMAIL, D (2016-06-07 08:26:30, HR Production)

9.2.4 Drag and Drop a File

You can also select a file in the platform's file browser and drop it somewhere in the DbVisualizer window. If you drop it in an editor, the file content is inserted at the caret position in the editor. If you instead drop it in the toolbar area, the file is opened in a new **SQL Commander** tab.

9.2.5 Drag and Drop Database Objects

If you want to include an object shown in the database objects tree, you can select the node and drop it in the editor where you want it inserted. The **Script Object** dialog is shown where you can select exactly what you want to insert in the editor.



Script Table - 1 object

Scripting Type

● SELECT ○ SELECT * ○ INSERT ○ UPDATE ○ DELETE ○ CREATE ○ DROP ○ Object Name

Options

Format SQL: □

Qualify Names: □

Delimited Identifiers: □

Statement Delimiter: ;

Output Destination

● SQL Commander 2: Untitled ○ At Caret ○ First ● Last ○ Replace All

First of all, you can select to insert an SQL statement based on the dropped object, e.g. a SELECT statement or a CREATE statement. You can also choose to just insert the object name. The choices available depends on the type of object you drop.

In the **Options** area, you can opt to format the SQL before it is inserted and use qualifiers and quoted identifiers, and even change which statement delimiter to use.

The **Output Destination** is set to the SQL Commander tab you dropped the object on by default, but you can change your mind and pick another destination. If you stick with an **SQL Commander** as the destination, you can tell where in the editor to insert the text.

You can also open this dialog from the **Databases** tab, from the object's right-click menu.

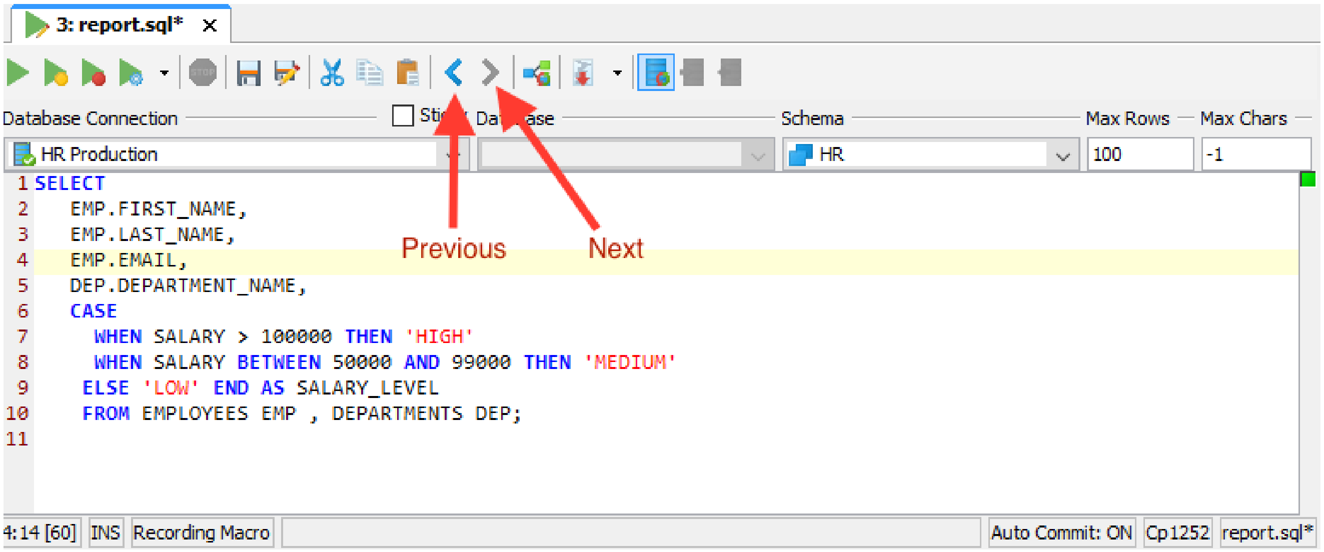
⚠ If you just want to insert the object names in the editor, hold down the **Ctrl** key (or the **Alt** key on macOS) while dragging and dropping. This behavior can be reversed in **Tool Properties**, in the **SQL Commander** category, so that dropping without pressing a key inserts the names and pressing the key launches the dialog.

9.2.6 Loading and Saving Bookmarks and Monitors

Bookmarks and Monitors are also files, but with special meaning. See the Managing Frequently Used SQL (see page 187) for how to create and edit them in the SQL Editor.

9.2.7 Navigating Between History Entries

When you execute a script, DbVisualizer saves it as a history entry, see the Re-Executing SQL Statements (see page 176) section for details. You can use the **Previous** and **Next** buttons in the editor toolbar to navigate between (load) these entries.



Database Connection: HR Production
Schema: HR
Max Rows: 100
Max Chars: -1

```
SELECT
   EMP.FIRST_NAME,
   EMP.LAST_NAME,
   EMP.EMAIL,
   DEP.DEPARTMENT_NAME,
   CASE
     WHEN SALARY > 100000 THEN 'HIGH'
     WHEN SALARY BETWEEN 50000 AND 99000 THEN 'MEDIUM'
    ELSE 'LOW' END AS SALARY_LEVEL
    FROM EMPLOYEES EMP , DEPARTMENTS DEP;
```

4:14 [60] INS Recording Macro Auto Commit: ON Cp1252 report.sql*

9.2.8 Confirming Overwriting Unsaved Changes

By default, you have to confirm overwriting unsaved changes in an editor, e.g. when navigating between history entries, and when closing an SQL Commander tab with unsaved edits. You can disable these confirmation popups in the Tool Properties dialog, under the **SQL Commander** category under the General tab.

9.2.9 SQL Formatting

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **SQL->Format SQL** menu contains four operations for formatting SQL statements.

Format Buffer formats the complete editor content and **Format Current** formats the current SQL (at cursor position).

The formatting is done according to the settings defined in the Tool Properties dialog, in the **SQL Editor/SQL Formatting** category under the General tab. There are many things you can configure. After making some changes, press **Apply** and format again to see the result.

Here is an example of an SQL statement 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-Discount)) desc)
order by Region
```

And here is the same statement after formatting has been applied with default settings:

```
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
```

Copy Formatted and **Paste Formatted** are powerful tools for copying SQL statements between programs written in languages like Java, C#, PHP, VB, etc. and the SQL Editor. Both operations display a dialog where you can adjust some of the formatting options, most importantly the **Target SQL** option and the **SQL is Between** option. **Target SQL** can be set to a number of common programming language formats.

For instance, to copy an SQL statement and paste it as Java code for adding it to a Java StringBuffer:

1. Select the statement,
2. Choose **SQL->Format SQL->Copy Formatted**,
3. Set **Target SQL** to Java StringBuffer,
4. Click **Format** to place the formatted statement on the system clipboard,
5. Paste it into your Java code.

To copy a statement wrapped in code from a program:

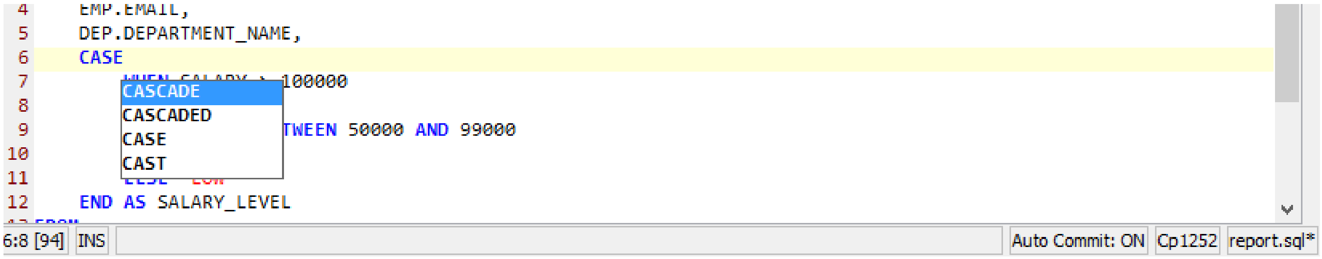
1. Select the code containing an SQL statement in your program,
2. Copy it to the system clipboard,
3. Choose **SQL->Format SQL->Paste Formatted**,
4. Check **SQL is Between** and enter the character enclosing the SQL statement in the code,
5. Click **Format** to extract the SQL statement and paste the formatted SQL in the editor.

9.2.10 Auto Completion

> **Only in DbVisualizer Pro**
>
> This feature is only available in the DbVisualizer Pro edition.

Auto completion is a convenient feature used to assist you when editing SQL statements and DbVisualizer commands. By default, you activate auto completion with the key binding **Ctrl-SPACE**, but you can also configure it to activate as you type (in the Tool Properties dialog, in the **SQL Editor/Auto Completion** category under the General tab).

With the caret in any place in a statement where you can type something other than a table name or a column name, and at least one character just before the caret, activating auto completion displays a list of keywords that starts with the letters you have typed so far. As you continue to type, the list narrows.



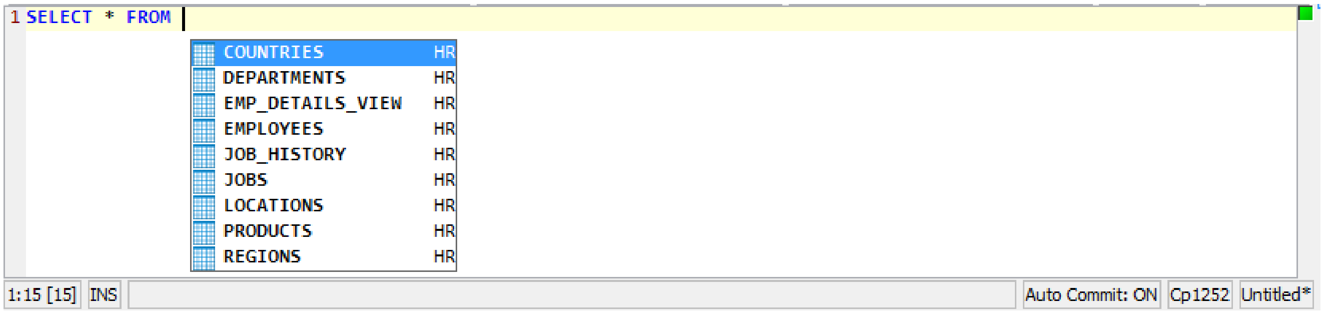
```
    EMP.EMAIL,
    DEP.DEPARTMENT_NAME,
    CASE
        WHEN SALARY > 100000

                        TWEEN 50000 AND 99000

    END AS SALARY_LEVEL
```

The list of keywords is database specific, selected based on the database type for the connection currently selected in the **Database Connection** list above the editor.

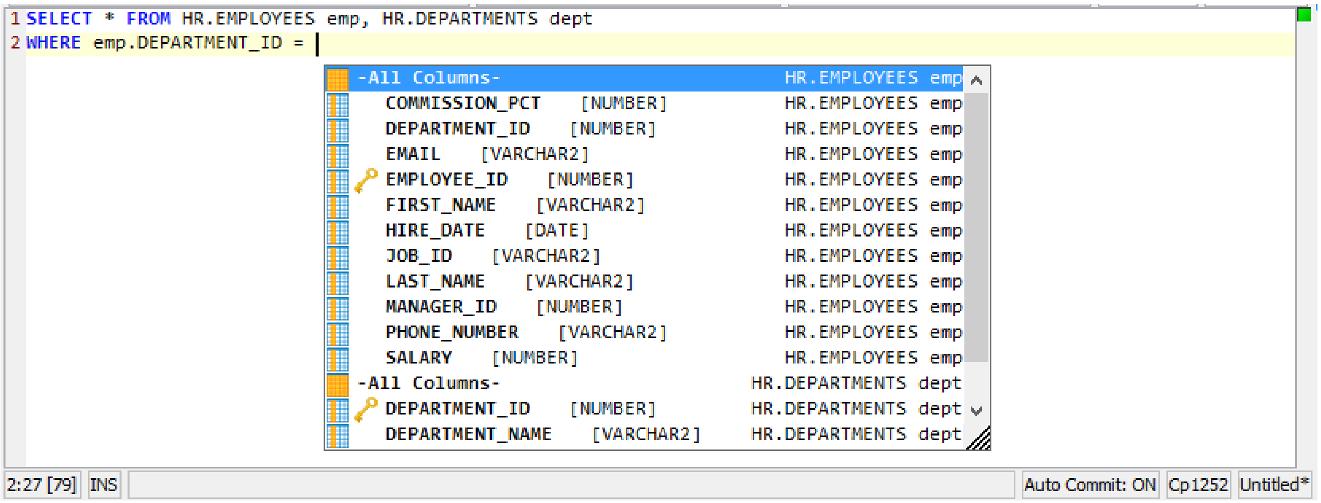
With the caret placed where a table or view name may be typed in a supported SQL statement type, the auto completion list shows a list of tables and views from the currently selected database connection, assuming you are actually connected to the database. The following figure shows the completion pop up with table names.



```
SELECT * FROM
```

COUNTRIES	HR
DEPARTMENTS	HR
EMP_DETAILS_VIEW	HR
EMPLOYEES	HR
JOB_HISTORY	HR
JOBS	HR
LOCATIONS	HR
PRODUCTS	HR
REGIONS	HR

A completion pop-up showing column names is shown when the caret is placed where a column name may be typed.



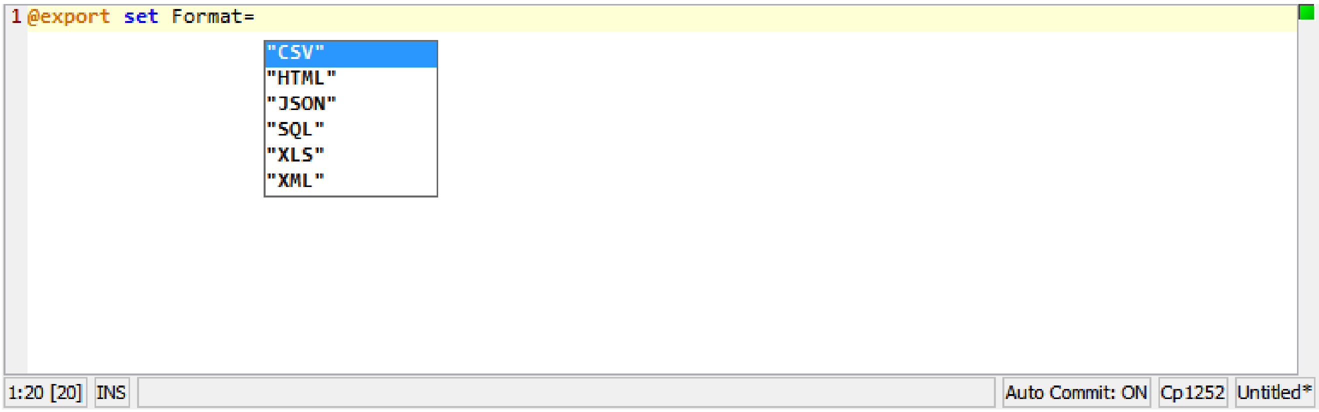
```
SELECT * FROM HR.EMPLOYEES emp, HR.DEPARTMENTS dept
WHERE emp.DEPARTMENT_ID =
```

-All Columns-		HR.EMPLOYEES emp
COMMISSION_PCT	[NUMBER]	HR.EMPLOYEES emp
DEPARTMENT_ID	[NUMBER]	HR.EMPLOYEES emp
EMAIL	[VARCHAR2]	HR.EMPLOYEES emp
EMPLOYEE_ID	[NUMBER]	HR.EMPLOYEES emp
FIRST_NAME	[VARCHAR2]	HR.EMPLOYEES emp
HIRE_DATE	[DATE]	HR.EMPLOYEES emp
JOB_ID	[VARCHAR2]	HR.EMPLOYEES emp
LAST_NAME	[VARCHAR2]	HR.EMPLOYEES emp
MANAGER_ID	[NUMBER]	HR.EMPLOYEES emp
PHONE_NUMBER	[VARCHAR2]	HR.EMPLOYEES emp
SALARY	[NUMBER]	HR.EMPLOYEES emp
-All Columns-		HR.DEPARTMENTS dept
DEPARTMENT_ID	[NUMBER]	HR.DEPARTMENTS dept
DEPARTMENT_NAME	[VARCHAR2]	HR.DEPARTMENTS dept

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

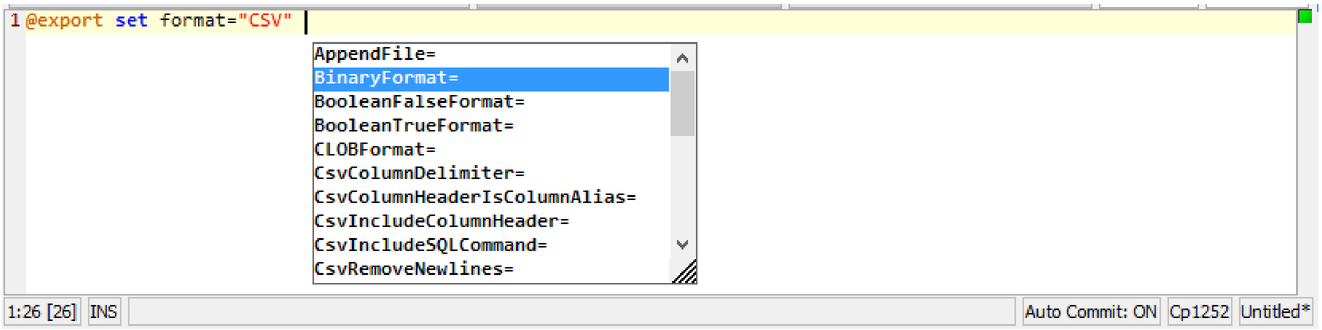
- SELECT
- INSERT
- UPDATE
- DELETE

Auto completion for DbVisualizer commands is very similar. Activating it after a partial command name lists all matching commands. If you activate it after a complete command name, you get a list of all valid parameters for the command. After a parameter name, you can select from a list of valid values.



```
1 @export set Format=
```

For the `@export set` command, the parameter list is adapted for the specified output format after you have entered the `Format` parameter setting, for instance only showing parameters that are valid for the CSV format.



```
1 @export set format="CSV" 
```

To display the completion pop-up, use the key binding **Ctrl-SPACE** (by default). 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.

⚠ If there are several SQL statements in the editor, make sure to separate them using the statement delimiter character (the default is ";").

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

All table names that have 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.

The **Database** and **Schema** lists above the editor are used to limit the list of tables in the auto complete pop-up to those in the selected database and/or schema. To include all tables, select the blank entries in these lists. The default selections for the lists can be set as connection properties, in the **SQL Commander** category.

It is possible to fine-tune how auto completion works in the connection properties.

- Enable or disable the use of identifier qualifiers (i.e. qualifying table names with the schema name) in the **[Database Type]/Qualifiers** category,
- Enable or disable the use of delimited identifiers (e.g. quotes around a table name) in the **[Database Type]/Delimited Identifiers** category.

Sorting, when to show the popup, etc. can be configured in the Tool Properties dialog, in the **SQL Editor/Auto Completion** category under the General tab.

9.2.11 Recording and Playing Edit Macros

If you repeatedly need to run a sequence of edit operation, you can record them as a macro and play it as many times as needed during an editing session. The editor status bar indicates when a recording is in progress and when a macro is available to play.

As an example, suppose you have some plain text that you need to convert into INSERT statements:

```
12345 123456
89012 890123
45678 456789
```

Place the caret at the beginning of the first line and start the macro recording, using the right-click menu or the corresponding key binding, and then type text and use key bindings (or menu items) to perform the following operations:

1. Type `insert into mytable values('`
2. **Insertion Point to End of Word**
3. Type `',`
4. **Insertion Point to Next Word**
5. Type `'`
6. **Insertion Point to End of Word**
7. Type `');`
8. **Insertion Point Down**
9. **Insertion Point to Beginning of Line**

Then stop the recording. You now have a macro for converting a single line to an INSERT statement. To convert the remaining lines, just use Play Macro for each line. The result will look like this:

```
insert into mytable values('12345', '123456');
insert into mytable values('89012', '890123');
insert into mytable values('45678', '456789');
```

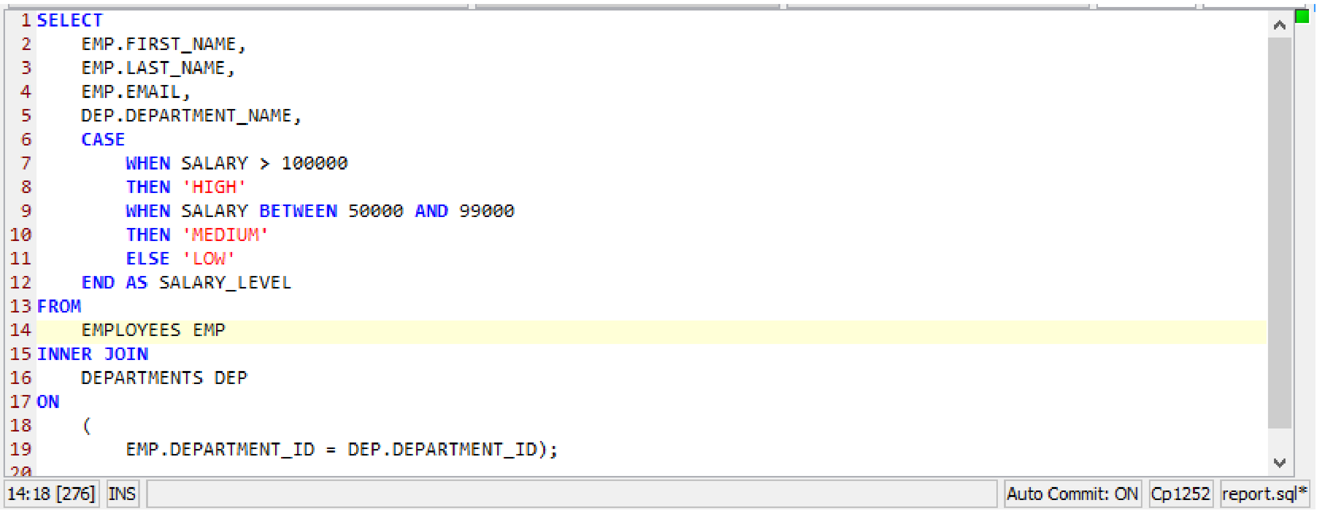
The Find operation, by default mapped to the **Find** key and **Ctrl-F** key stroke, can not be recorded. You must instead use **Find Selection**, **Find with Dialog**, **Find Next** and **Find Previous**. Mouse gestures are also not recorded, only key strokes and menu selections.

9.2.12 Folding Selected Text

If you work with a large script, it can sometimes be helpful to hide parts of it. You can do so using the Code Folding feature.

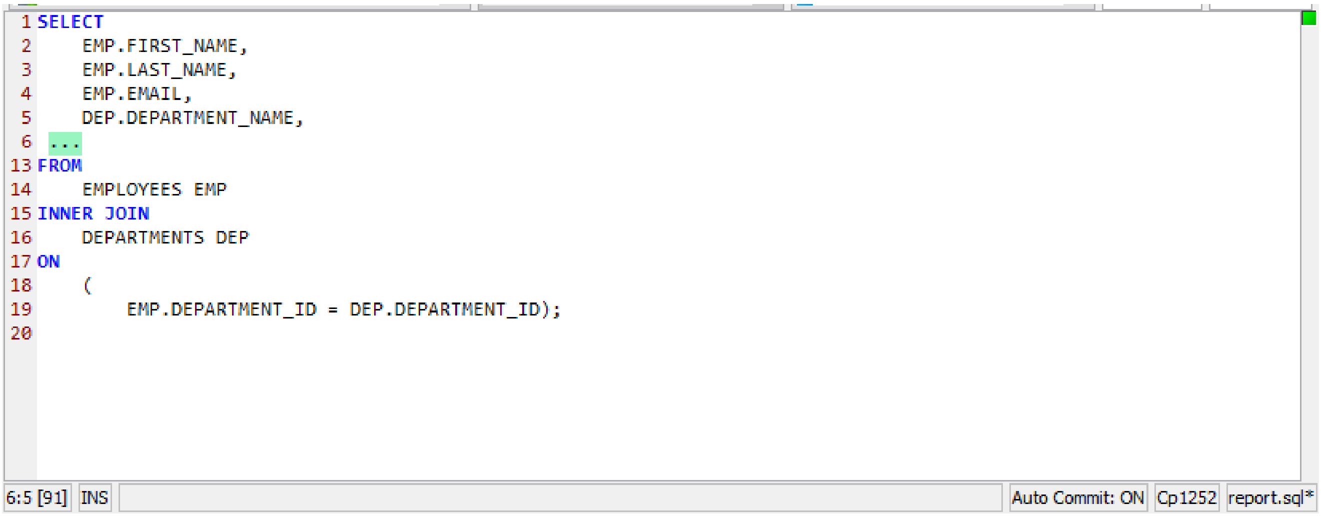
Select the text you want to hide and then choose **Toggle Fold Selection** in the right-click menu. The selected text is then replaced (visually only) with a folding marker.

Here's an unfolded script:



```
SELECT
    EMP.FIRST_NAME,
    EMP.LAST_NAME,
    EMP.EMAIL,
    DEP.DEPARTMENT_NAME,
    CASE
        WHEN SALARY > 100000
        THEN 'HIGH'
        WHEN SALARY BETWEEN 50000 AND 99000
        THEN 'MEDIUM'
        ELSE 'LOW'
    END AS SALARY_LEVEL
FROM
    EMPLOYEES EMP
INNER JOIN
    DEPARTMENTS DEP
ON
    (
        EMP.DEPARTMENT_ID = DEP.DEPARTMENT_ID);
```

And here is the same script with the CASE expression folded:



```
SELECT
    EMP.FIRST_NAME,
    EMP.LAST_NAME,
    EMP.EMAIL,
    DEP.DEPARTMENT_NAME,
 ...
FROM
    EMPLOYEES EMP
INNER JOIN
    DEPARTMENTS DEP
ON
    (
        EMP.DEPARTMENT_ID = DEP.DEPARTMENT_ID);
```

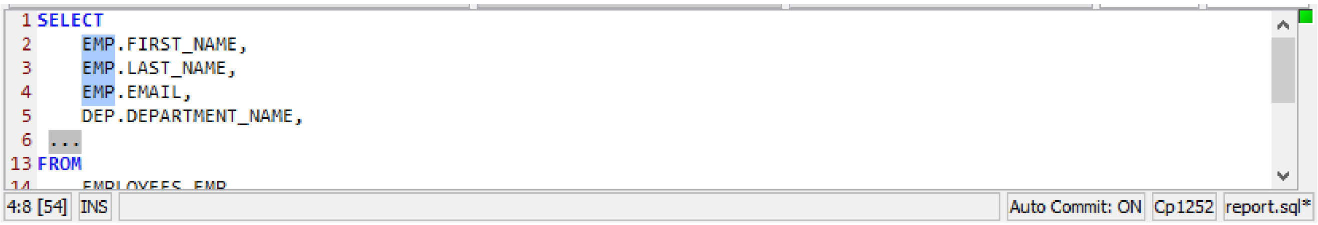
You can fold more than one part of a script using the same procedure.

To unfold just one part, select the folding marker (be careful to select all of it) and then choose **Toggle Fold Selection** from the menu again. To unfold all folded parts, use **Expand All Foldings**.

9.2.13 Selecting a Rectangular Area

In some cases, it is handy to be able to select a rectangular area in the middle of a script. Say, for instance, that you need to copy just the first part of a few lines and paste it at the beginning of some other lines.

To do this in the SQL editor, click the mouse where you want to start the selection and then press the **Alt** key (by default) while you extend the selection by dragging the mouse. If you prefer to use the **Ctrl** key as the modifier, you can change the default in Tool Properties in the SQL Commander category under the General tab.



```
SELECT
    EMP.FIRST_NAME,
    EMP.LAST_NAME,
    EMP.EMAIL,
    DEP.DEPARTMENT_NAME,
 ...
FROM
    EMPLOYEES EMP
```

9.2.14 Tab Key Treatment

Pressing the TAB key in the editor inserts four space characters by default. If you instead want a TAB character to be inserted, or want to insert another number of space characters, you can specify this in the Tool Properties dialog, in the **SQL Commander** category under the General tab.

9.2.15 Key Bindings

The editor shortcuts, or key bindings, can be redefined in the **Tool Properties** dialog, in the **Key Bindings** category under the General tab. Expand the **Editor Commands** node to manage all editor actions and the **Main Menu/Edit** node to manage the key bindings for the edit operations in the right-click editor menu and the main window **Edit** menu.

9.3 Using Editor Templates

Editor Templates can be used to easily insert text that you often use, such as code snippets, current date and time, or anything you like.

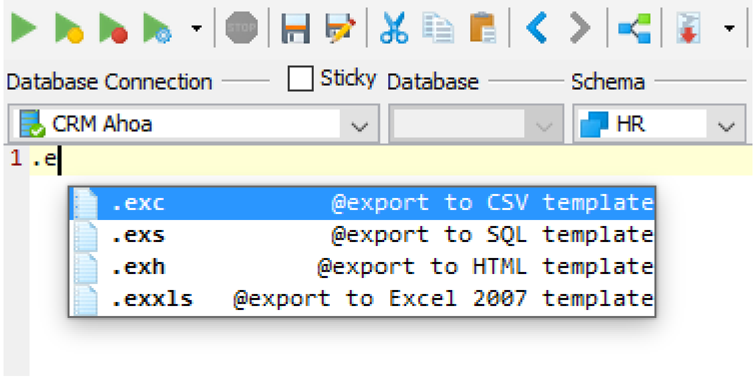
Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

9.3.1 Using a Template

A template is an **Abbreviation** that can easily be replaced by its **Expanded Text**, and it may optionally have a **Description**.

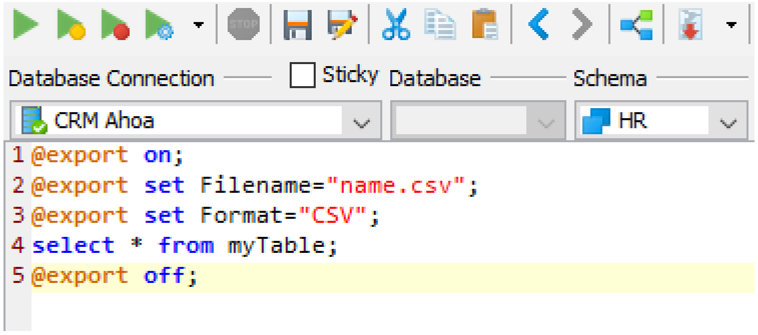
To expand a template, just type a few characters that at least partially match one or more template abbreviation and press the TAB key. This displays a list of matching templates that you can pick from, along with the description or a part of the corresponding expanded text for each.



```
.e
```

```
.exc        @export to CSV template
.exs        @export to SQL template
.exh        @export to HTML template
.exxls  @export to Excel 2007 template
```

When you select a template by pressing the TAB or ENTER key, its expanded text replaces the abbreviation you typed in the editor.



```
@export on;
@export set Filename="name.csv";
@export set Format="CSV";
select * from myTable;
@export off;
```

You can also enable **Immediate Substitution** to immediately expand the text if you have typed enough of the abbreviation to match a single template when you press the TAB key:

1. Open **Tools->Tool Properties**,
2. Select the **General/SQL Commander/Editor Templates** category,
3. Click the **Immediate Substitution** checkbox in the toolbar,
4. Click the **Apply** or **OK** button to save the updated setting.

Instead of pressing the TAB key to list matching templates, you can use the **Show Editor Templates** entry in the main Edit menu or the editor right-click menu.

9.3.2 Creating a new Template

DbVisualizer comes with some default templates, but you can create additional templates:

1. Open **Tools->Tool Properties**,
2. Select the **General/SQL Commander/Editor Templates** category,
3. Click the **Insert** button in the toolbar to add a new template, and enter an Abbreviation, Expanded Text, and optionally a Description,
4. Check off the checkbox in the **Format** column if you wan the Expanded Text to be formatted by the SQL Formatter when it is inserted in the editor,
5. Click the **Apply** or **OK** button to save the new templates.

If you want to use an existing template as a starting point, you can select it and then click the **Duplicate** button instead of the Insert button. The Edit in Window button opens the selected cell in a separate window where it is easier to work with larger templates.

⚠ Always use a special character, such as dot or hashmark, as the first character in the Abbreviation. The default templates use a dot but you can pick any special character you like. If you use a regular character as the first character, unexpected matches may be found if one abbreviation starts with the same characters as another abbreviation ends with.

9.3.3 Editing or Deleting a Template

You can edit any piece of a template or delete templates you no longer need:

1. Open **Tools->Tool Properties**,
2. Select the **General/SQL Commander/Editor Templates** category,
3. To edit, double-click any cell you want to change and edit its value,
4. To delete a template, select any of its cells and click the Delete button,
5. Click the **Apply** or **OK** button to save the changes.

9.3.4 Changing the Expand Keybinding

The keyboard shortcut used to expand a template, or bring up the list of matching templates if more than one, is the TAB key. You can add other shortcuts or change the default, as described in Changing Keyboard Shortcuts (see page 45). The shortcut is named **Show Editor Templates** and you find it in the **Main Menu/Edit** category.

9.4 Executing SQL Statements

In the SQL Commander, you can execute one or multiple statements. You can also control if the execution should stop or continue when the execution of a statement results in a warning or error.

- Execute Multiple Statements (see page 174)
- Execute Only the Current Statement (see page 175)
- Control Execution after a Warning or an Error (see page 175)

9.4.1 Execute Multiple Statements

Use the **SQL->Execute** main menu operation to execute the SQL in the SQL Commander 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.

DbVisualizer uses the delimiters specified in the Tool Properties dialog, in the **SQL Commander/Statement Delimiters** category under the General tab, to separate one statement from the next.

The result of the execution is displayed in the results area based on the type of results result(s) that are returned . If there are several results and an error occurred in one of them, the Log tab 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.

⚠ Comments in the SQL editor are sent to the database when you use SQL->Execute, unless you have enabled **Preprocess Script->Strip Comments when Executing** in the **SQL Commander** menu.

9.4.2 Execute Only the Current Statement

ⓘ **Only in DbVisualizer Pro**

This feature is only available in the DbVisualizer Pro edition.

The **SQL->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.The current statement is the statement containing the caret or that ends on the line with the caret. This means that the caret may be after the statement delimiter as long as there is no other statement on the same line.

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.

9.4.3 Control Execution after a Warning or an Error

You can control whether subsequent statements should be executed when a statement results in an error, a warning or returns or affects no rows.

Open **Tools->Tool Properties** and select the **SQL Commander** category under the **General** tab. There you find **Stop on Error, Stop on SQL Warning** and **Stop on No Rows** check boxes for enabling these features in all **SQL Commander** tabs.

Alternatively, you can use DbVisualizer client side commands to enable or disable these features in a script.

```
@stop on error;
@stop on sqlwarning;
```

```
@stop on norows;
@continue on error;
@continue on sqlwarning;
@continue on norows;
```

9.5 Re-Executing SQL Statements

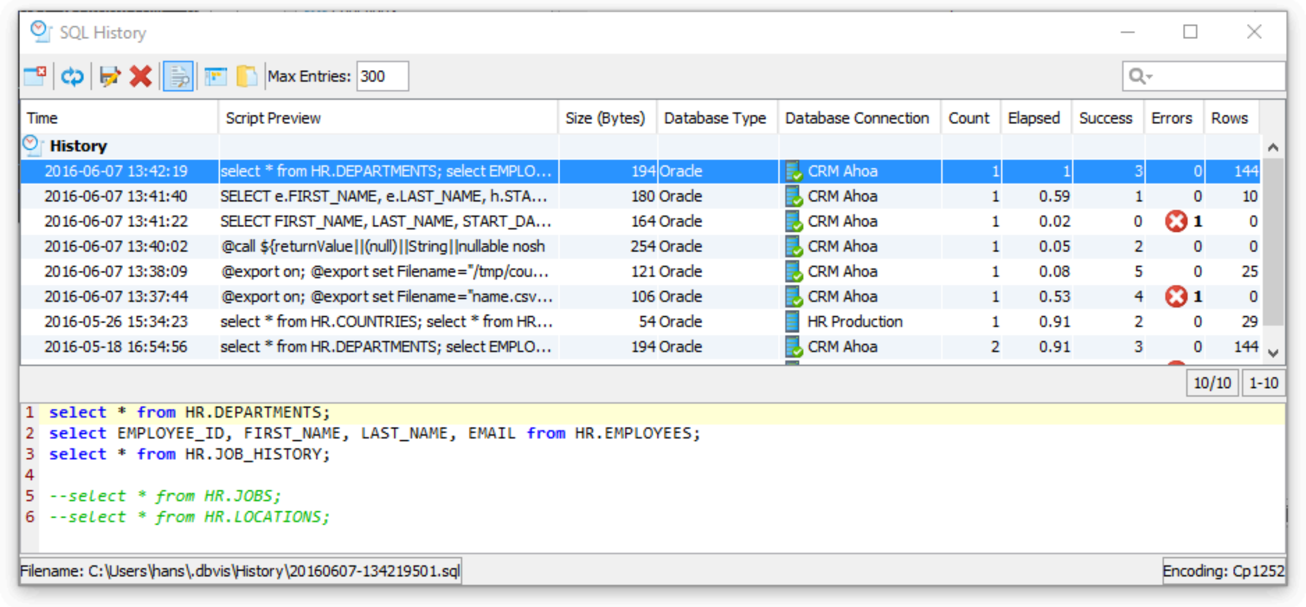
As you execute SQL statements in the SQL Commander, DbVisualizer saves them as History entries, along with information about the Connection, Catalog, Schema and the execution result. This makes it easy to locate statements and scripts you have executed in the past.

9.5.1 Using Previous and Next in the SQL Commander

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 Commander.

9.5.2 Using the SQL History Window

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



Time	Script Preview	Size (Bytes)	Database Type	Database Connection	Count	Elapsed	Success	Errors	Rows
2016-06-07 13:42:19	select * from HR.DEPARTMENTS; select EMPLO…	194	Oracle	CRM Ahoa	1	1	3	0	144
2016-06-07 13:41:40	SELECT e.FIRST_NAME, e.LAST_NAME, h.STA…	180	Oracle	CRM Ahoa	1	0.59	1	0	10
2016-06-07 13:41:22	SELECT FIRST_NAME, LAST_NAME, START_DA…	164	Oracle	CRM Ahoa	1	0.02	0	☒ 1	0
2016-06-07 13:40:02	@call ${returnValue||(null)||String||nullable nosh	254	Oracle	CRM Ahoa	1	0.05	2	0	0
2016-06-07 13:38:09	@export on; @export set Filename="/tmp/cou…	121	Oracle	CRM Ahoa	1	0.08	5	0	25
2016-06-07 13:37:44	@export on; @export set Filename="name.csv…	106	Oracle	CRM Ahoa	1	0.53	4	☒ 1	0
2016-05-26 15:34:23	select * from HR.COUNTRIES; select * from HR…	54	Oracle	HR Production	1	0.91	2	0	29
2016-05-18 16:54:56	select * from HR.DEPARTMENTS; select EMPLO…	194	Oracle	CRM Ahoa	2	0.91	3	0	144

```
select * from HR.DEPARTMENTS;
select EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL from HR.EMPLOYEES;
select * from HR.JOB_HISTORY;

--select * from HR.JOBS;
--select * from HR.LOCATIONS;
```

Filename: C:\Users\hans\.dbvis\History\20160607-134219501.sql

Encoding: Cp1252

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.

The columns show when the entry was executed, a part of the script/statement, the size of the complete statement/script, and then the database type and connection it was executed for, and how long it took to execute. The **Success** and **Errors** columns show how many of the statements in a script were executed successfully or that caused an error. Finally, the **Rows** column show the number of rows retrieved or affected by the script.

You can use the field at the top right corner in the dialog to search for entries matching a critera. Clicking on the magnifying glass reveals a configuration menu where you can, among other things, specify which columns to search in and if you want to search through the complete script rather than just the part of the script shown in the Script/SQL Statement column.

In the Tool Properties dialog, in the **SQL History** category under the General tab, you can specify that sequential executions of the same SQL statement/script should be collected into a single history entry. When this feature is enabled, the **Count** field number is increased for each execution. In the same Tool Properties category you can also specify rules for when not to create a history entry.

Reusing a History Entry

When you have found the entry you're looking for, you can open it in an SQL Commander by double-clicking it or clicking the corresponding toolbar button.

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.

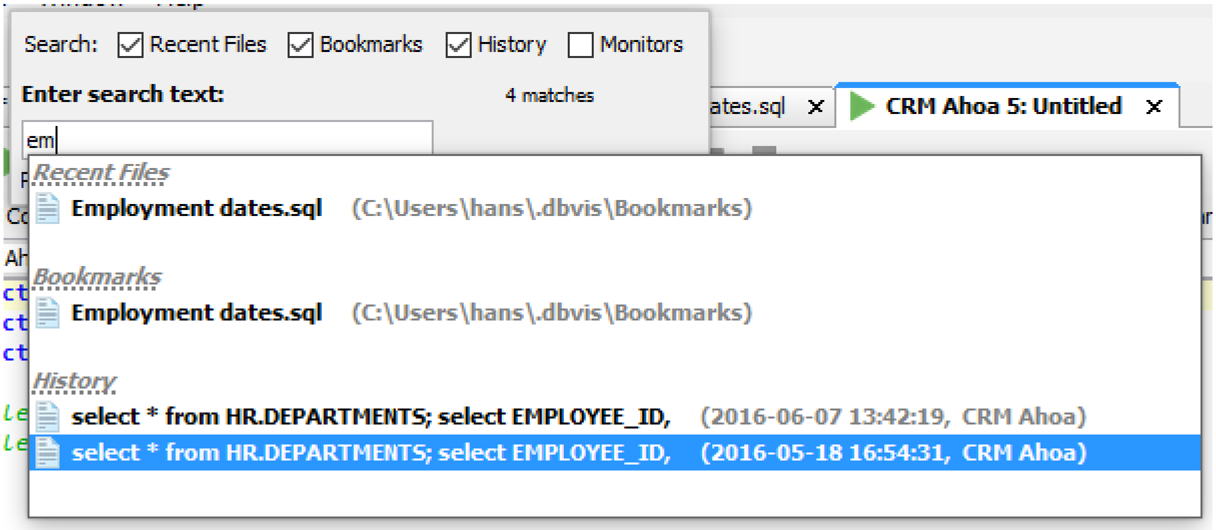
Saving a History Entry as a Bookmark or Other File

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

You can also locate the file holding the history entry in the file system using the the **Open Enclosing Directory** right-mouse menu entry or toolbar button. This opens a file chooser for the directory holding the file.

9.5.3 Using 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 main toolbar button as well as in the **File->Quick File Open** menu.



Search: ☑ Recent Files ☑ Bookmarks ☑ History □ Monitors

Enter search text: em

4 matches

Recent Files

Employment dates.sql (C:\Users\hans\.dbvis\Bookmarks)

Bookmarks

Employment dates.sql (C:\Users\hans\.dbvis\Bookmarks)

History

select * from HR.DEPARTMENTS; select EMPLOYEE_ID, (2016-06-07 13:42:19, CRM Ahoa)

select * from HR.DEPARTMENTS; select EMPLOYEE_ID, (2016-05-18 16:54:31, CRM Ahoa)

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. Use the **Max** field to limit the number of matching files to display in the list.

When you see the file you're looking for, just select it and click **Enter** to load it into an SQL Commander editor. If the file is already loaded in an editor, that tab is made visible instead.

9.6 Executing Complex Statements

If you need to execute a complex statement that itself contains other statements in the SQL Commander, such as a CREATE PROCEDURE statement, you need to help DbVisualizer figure out where the complex statement starts and ends. The reason is that DbVisualizer needs to send statements to the database for execution one by one.

We recommend that you use the Create Procedure (see page 139) dialog and Procedure Editor (see page 141) to work with procedures and similar objects, so only use the SQL Commander to run statements like this if these features do not work for you for some reason.

- Using Execute Buffer (see page 179)
- Using an SQL Block (see page 179)
- Using the @delimiter command (see page 179)

9.6.1 Using Execute Buffer

The **SQL->Execute Buffer** operation 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. You can use this feature if the complex statement is the only statement in the SQL Commander editor.

9.6.2 Using an SQL Block

To tell DbVisualizer that a part of a script should be handled as a single statement, you can insert an SQL block begin identifier just before the block and an end identifier after the block. The delimiter must be the only text on the line. The default value for the **Begin Identifier** is `--/` and for the **End Identifier** it is `/`.

Here is an example of an 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;
/
```

9.6.3 Using the @delimiter command

With the **@delimiter** command you can temporarily change the statement delimiter DbVisualizer uses to separate the statements and send them one by one to the database. Use it before the complex statement, and after the statement if the script contains additional statements. Here's an example:

```
@delimiter ++;
CREATE OR REPLACE FUNCTION HELLO (p1 IN VARCHAR2) RETURN VARCHAR2
```

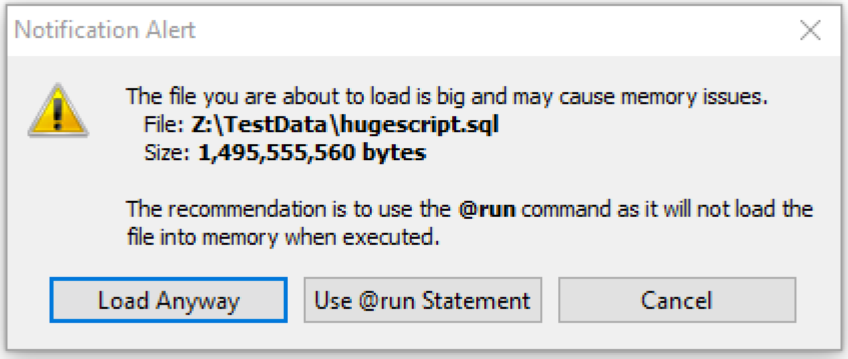
```
AS
BEGIN
 RETURN 'Hello ' || p1;
END;
++
@delimiter ;++
@call ${returnValue||(null)||String||noshow dir=out}$ = HELLO('World');
@echo returnValue = ${returnValue}$;
```

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.

9.7 Executing an External Script

If you have a very large script to execute, you may not have enough memory available to be allocated for DbVisualizer to load it into an SQL Commander editor.

To save memory, you can use the **@run** command. If you try to load a very large file, DbVisualizer suggests using the @run command automatically:



The file you are about to load is big and may cause memory issues.
File: **Z:\TestData\hugescript.sql**
Size: **1,495,555,560 bytes**

The recommendation is to use the **@run** command as it will not load the file into memory when executed.

The **@run** command executes a script file by only loading one statement at a time, minimizing the memory requirements. A related command is the **@cd** command for changing the current directory.

- **@run <file> [<variables>]**
  Request to execute the file specified as parameter, optionally with a list of variables
- **@cd <directory>**
  Change the working directory for the following **@run** command

Here's an example of a script using these commands:

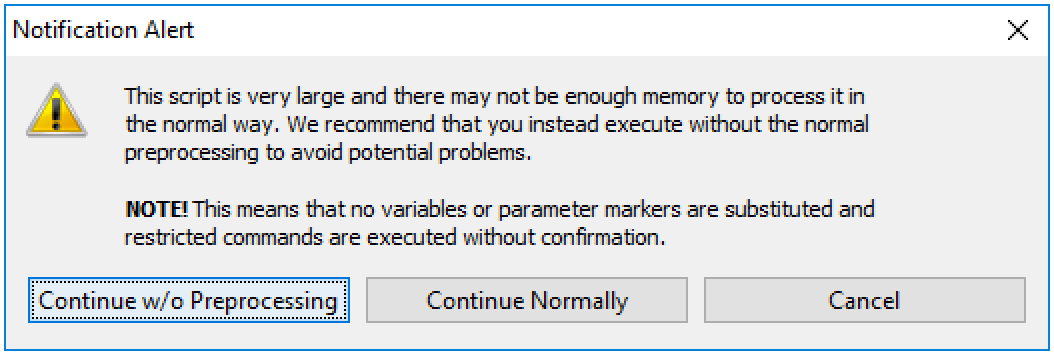
```
@run createDB.sql;     -- Execute the content in the
                       -- createDB.sql file without loading into the SQL editor.
```

```
                              -- The location of this file is the same as the working
                              -- directory for DbVisualizer (when not using an absolut path).
@cd /home/mupp;               -- Request to change directory to /home/mupp
@cd myscripts;                -- Request to change directory relative to current, i.e. to /home/mupp/
myscripts
@run loadBackup.sql;          -- Execute the content in the loadBackup.sql
                              -- file relative to current directory. This file will now be read from the
                              -- /home/mupp/myscripts directory.
```

You can also include DbVisualizer variables (see page 227) as parameters to the **@run** command, with values to be used for the corresponding variables in the script:

```
@run monthlyReport ${month||2010-05-05||Date||noshow}$ ${dept||HR||String||noshow}$
```

Even though the @run command reads one statement at a time from the file, there are other parts of the execution process 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 the are read from the file, one at a time.



This script is very large and there may not be enough memory to process it in the normal way. We recommend that you instead execute without the normal preprocessing to avoid potential problems.

NOTE! This means that no variables or parameter markers are substituted and restricted commands are executed without confirmation.

To ensure that you don't have any problems running scripts this large we 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, 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 (see page 319) instead of the GUI application. This option is the most efficient and fastest.

Running without preprocessing is always more efficient, so if your script does not use any variables or parameter markers and you do not use the Permissions feature, you can disable preprocessing even for scripts smaller than 10 MB by unchecking the **Preprocess Script** checkbox in the SQL Commander menu.

9.8 Locating SQL Errors

If errors occur, the corresponding text is underlined with a red wavy line. Hovering the mouse over the error indication shows the corresponding error message. The right margin contains markers for each error as well, and clicking on a marker scrolls the editor to the corresponding error. Alternatively, you can click the FAILED link in the Log tab grid to move the caret to the error location.

If you prefer to navigate between errors using the keyboard, you can use the define key bindings for the **Insertion Point to Next Marker** and **Insertion Point to Previous Marker** actions in the Tool Properties dialog, in the **Key Bindings** category, in the Editor Commands group. Alternative is to use the **Goto Next Failed** and **Goto Previous Failed** in the Log tab grid, and then click the link for the FAILED entries.

> ⚠ Error location information is not available for some databases. In that case the complete statement is underlined in the editor.

9.9 Analyzing (explain) Query Performance

> ℹ **Only in DbVisualizer Pro**
>
> This feature is only available in the DbVisualizer Pro edition.

You can analyze how a query is executed by the database, e.g. whether indexes are used or if the database has to do an expensive full scan. To analyze a query:

1. Enter the query in the **SQL Commander** editor,
2. Click **Execute Explain Plan** button in the toolbar,
3. Look at the result in the results area.

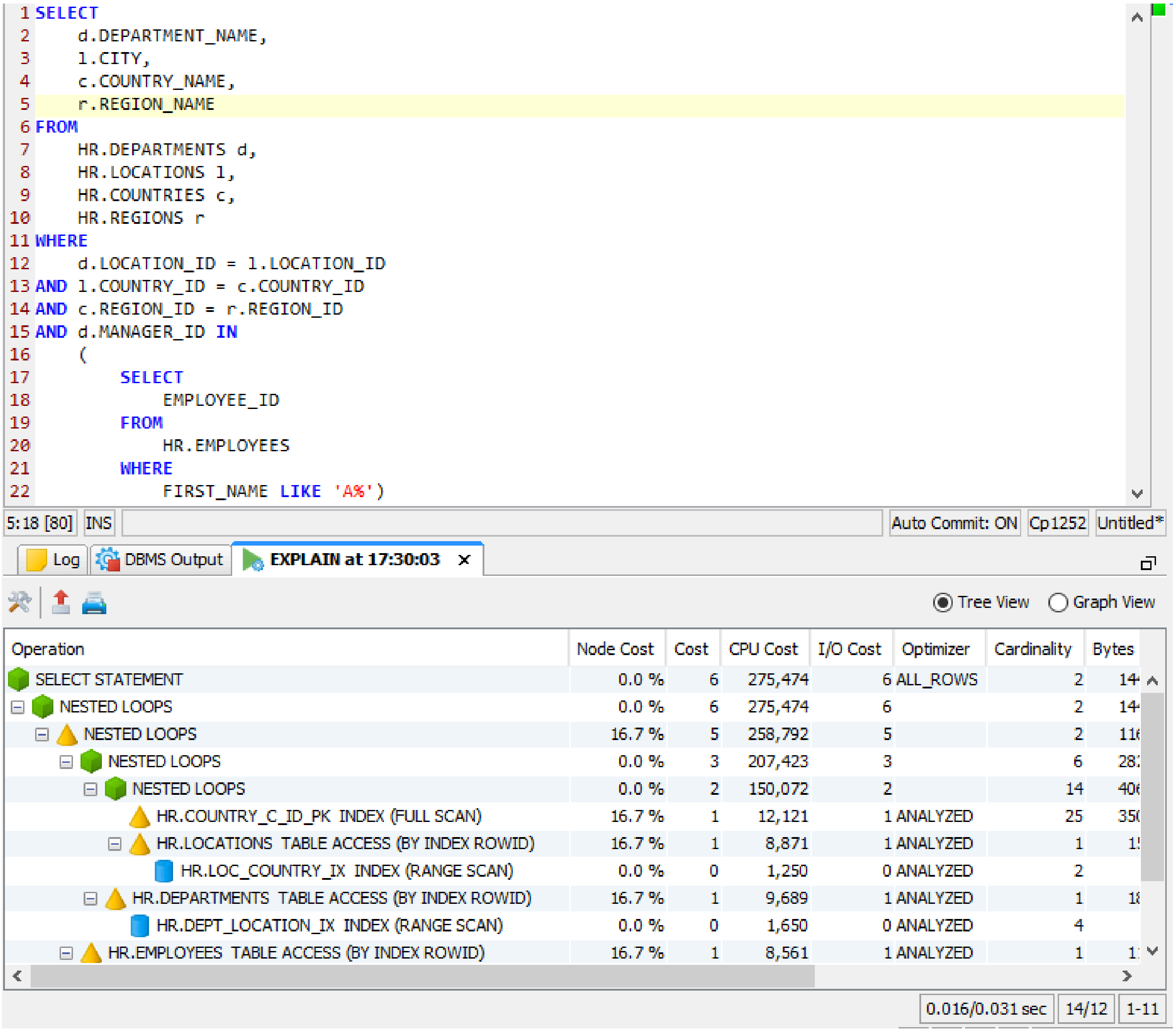
Explain Plan is supported for DB2 LUW, H2, JavaDB/Derby, Mimer SQL, MySQL, Netezza, NuoDB, Oracle, PostgreSQL, Amazon Redshift, SQLite, Microsoft SQL Server and Vertica. The available presentations options vary per database as shown in this table.

Database	Graph Format	Tree Format	Text Format	Node Cost Coloring (Graph and Tree only)
DB2 LUW	✓	✓		✓
H2			✓	

Database	Graph Format	Tree Format	Text Format	Node Cost Coloring (Graph and Tree only)
JavaDB/Derby			✓	
Mimer SQL	✓	✓		✓
MySQL	✓	✓	✓	✓
Netezza			✓	
NuoDB			✓	
Oracle	✓	✓	✓	✓
PostgreSQL	✓	✓	✓	✓
Amazon Redshift			✓	
SQLite			✓	
Microsoft SQL Server	✓	✓		✓
Vertica			✓	

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 pinning the tabs for different runs.

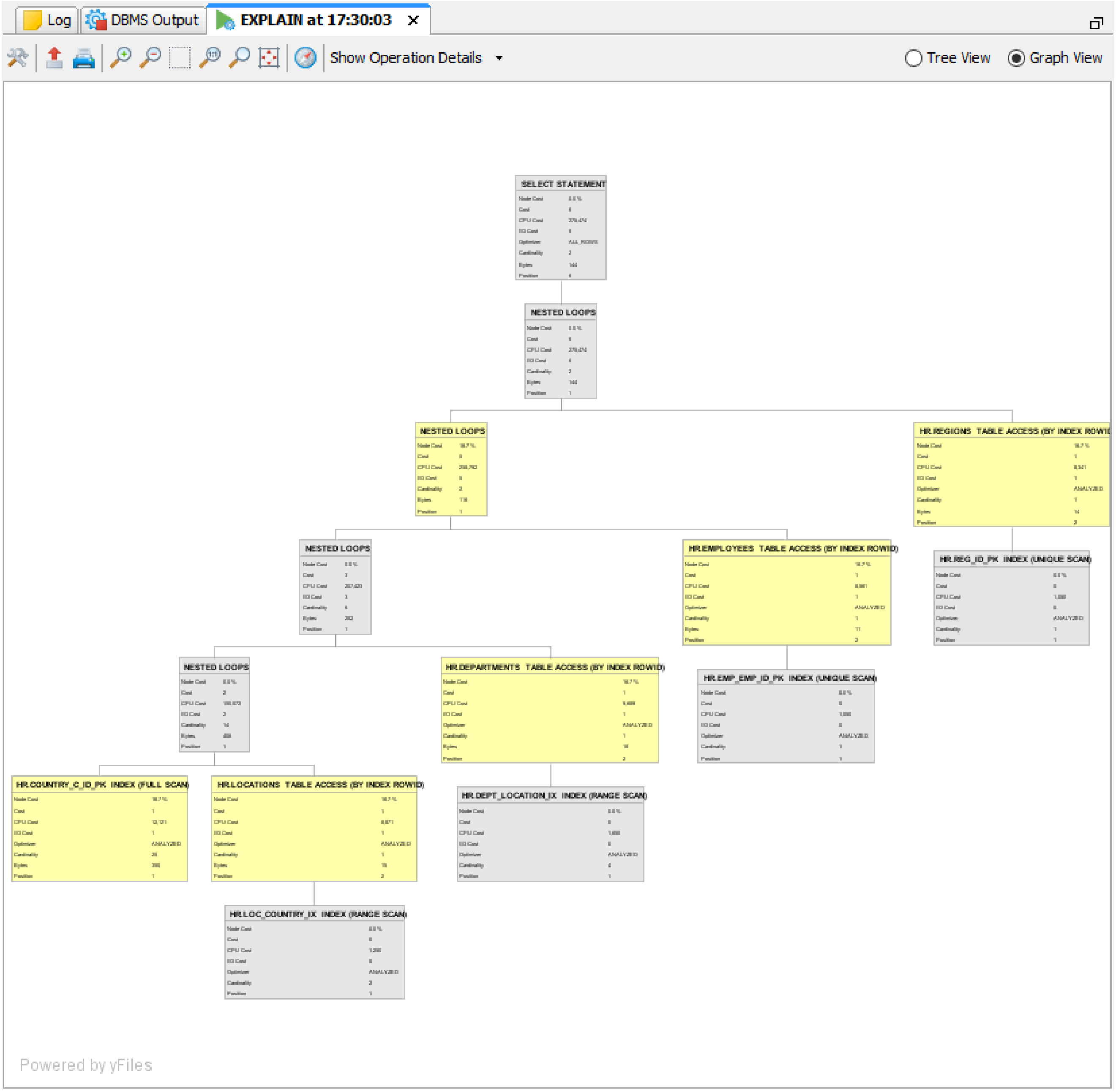
DbVisualizer presents the plan either in a tree style format or in a graph, or in a simple text format. What information is shown depends on the database you use. In the tree view, put the mouse pointer on the column header for a tooltip description what that column represents. The following screenshot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



```
SELECT
    d.DEPARTMENT_NAME,
    l.CITY,
    c.COUNTRY_NAME,
    r.REGION_NAME
FROM
    HR.DEPARTMENTS d,
    HR.LOCATIONS l,
    HR.COUNTRIES c,
    HR.REGIONS r
WHERE
    d.LOCATION_ID = l.LOCATION_ID
AND l.COUNTRY_ID = c.COUNTRY_ID
AND c.REGION_ID = r.REGION_ID
AND d.MANAGER_ID IN
    (
        SELECT
            EMPLOYEE_ID
        FROM
            HR.EMPLOYEES
        WHERE
            FIRST_NAME LIKE 'A%')
```

Operation	Node Cost	Cost	CPU Cost	I/O Cost	Optimizer	Cardinality	Bytes
SELECT STATEMENT	0.0 %	6	275,474	6	ALL_ROWS	2	14
⊟ NESTED LOOPS	0.0 %	6	275,474	6		2	14
⊟ NESTED LOOPS	16.7 %	5	258,792	5		2	11
⊟ NESTED LOOPS	0.0 %	3	207,423	3		6	28
⊟ NESTED LOOPS	0.0 %	2	150,072	2		14	40
HR.COUNTRY_C_ID_PK INDEX (FULL SCAN)	16.7 %	1	12,121	1	ANALYZED	25	35
⊟ HR.LOCATIONS TABLE ACCESS (BY INDEX ROWID)	16.7 %	1	8,871	1	ANALYZED	1	1
HR.LOC_COUNTRY_IX INDEX (RANGE SCAN)	0.0 %	0	1,250	0	ANALYZED	2	
⊟ HR.DEPARTMENTS TABLE ACCESS (BY INDEX ROWID)	16.7 %	1	9,689	1	ANALYZED	1	1
HR.DEPT_LOCATION_IX INDEX (RANGE SCAN)	0.0 %	0	1,650	0	ANALYZED	4	
⊟ HR.EMPLOYEES TABLE ACCESS (BY INDEX ROWID)	16.7 %	1	8,561	1	ANALYZED	1	1

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.


Log
DBMS Output
EXPLAIN at 17:30:03
Show Operation Details
○ Tree View
● Graph View
SELECT STATEMENT
NESTED LOOPS
NESTED LOOPS
HR.REGIONS TABLE ACCESS (BY INDEX ROWID)
NESTED LOOPS
HR.EMPLOYEES TABLE ACCESS (BY INDEX ROWID)
HR.REG_ID_PK INDEX (UNIQUE SCAN)
NESTED LOOPS
HR.DEPARTMENTS TABLE ACCESS (BY INDEX ROWID)
HR.EMP_EMP_ID_PK INDEX (UNIQUE SCAN)
HR.COUNTRY_C_ID_PK INDEX (FULL SCAN)
HR.LOCATIONS TABLE ACCESS (BY INDEX ROWID)
HR.DEPT_LOCATION_IX INDEX (RANGE SCAN)
HR.LOC_COUNTRY_IX INDEX (RANGE SCAN)
Node Cost
Cost
CPU Cost
IO Cost
Optimizer
Cardinality
Bytes
Position
Powered by yFiles

The databases use different techniques to manage their explain plan support. You can make database-specific configurations in the **Properties** tab for a connection, in the **Explain Plan** category.

9.10 Auto Commit, Commit and Rollback

With Auto Commit enabled, all changes you make to the database data is automatically committed after the successful execution of an SQL statement. Auto Commit is enabled for a connection by default. You can change the default in the **Options** area of the **Object View** tab for the connection.

You can toggle the Auto Commit setting for an open SQL Commander tab using the **SQL Commander** main menu item of the corresponding button in the SQL Commander toolbar. Alternatively, you can use this command in a script to set it:

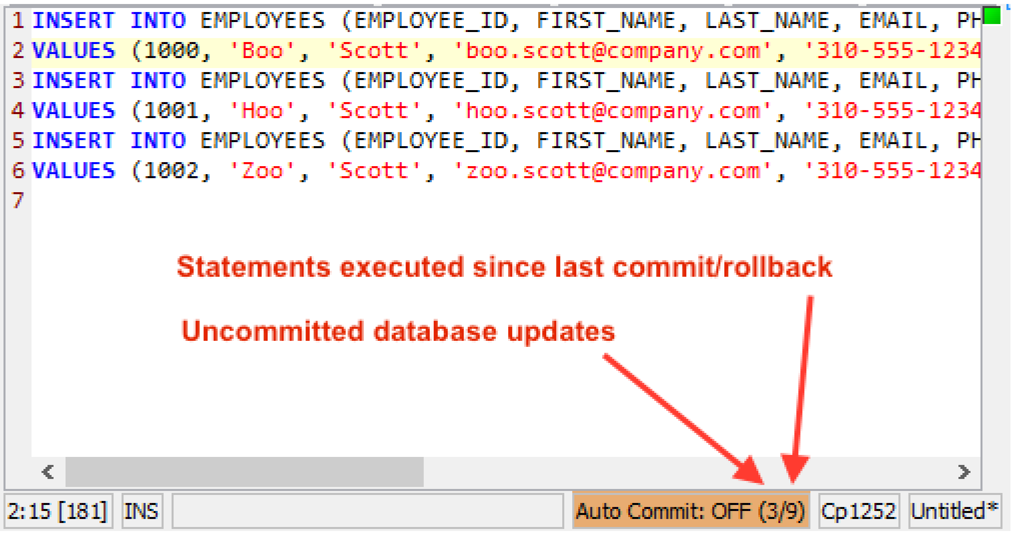
```
@set autocommit on/off;
```

If Auto Commit is disabled, it is very important to manually issue the commit or rollback operations when appropriate. Use the **Commit** and **Rollback** buttons in the SQL Commander toolbar or the corresponding operations in the **SQL Commander** main menu to commit and rollback transactions.

Alternatively, you can use the following commands in a script executed in the SQL Commander:

```
@commit;
@rollback;
```

There is an **Auto-Commit: ON/OFF** indicator in the editor status bar:



```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PH
VALUES (1000, 'Boo', 'Scott', 'boo.scott@company.com', '310-555-1234
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PH
VALUES (1001, 'Hoo', 'Scott', 'hoo.scott@company.com', '310-555-1234
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PH
VALUES (1002, 'Zoo', 'Scott', 'zoo.scott@company.com', '310-555-1234
```

Statements executed since last commit/rollback
Uncommitted database updates
2:15 [181]
INS
Auto Commit: OFF (3/9)
Cp1252
Untitled*

The first number represents the number of records updated in the database since the last commit/rollback. The second number shows the number of statements (except SELECTs) that has been executed since last commit/ rollback.

Having Auto Commit off for a connection should be handled with great care since transactions may lock parts of the database (this is database dependent). To minimize the risk of forgetting uncommitted transactions, there is an **Ask when Auto Commit is OFF** settings in the connection **Properties** tab, in the **Transactions** category, that can be set to warn you when there are changes that hasn't been committed. You can set it to **Always** or **When Uncommitted Updates**. When set to **When Uncommitted Updates**, you are warned when there is at least one updated record reported by the database. For database that do not accurately report updated records, you can set it to **Always** to be warned if at least one statement (other than SELECT) has been executed since the last commit or rollback.

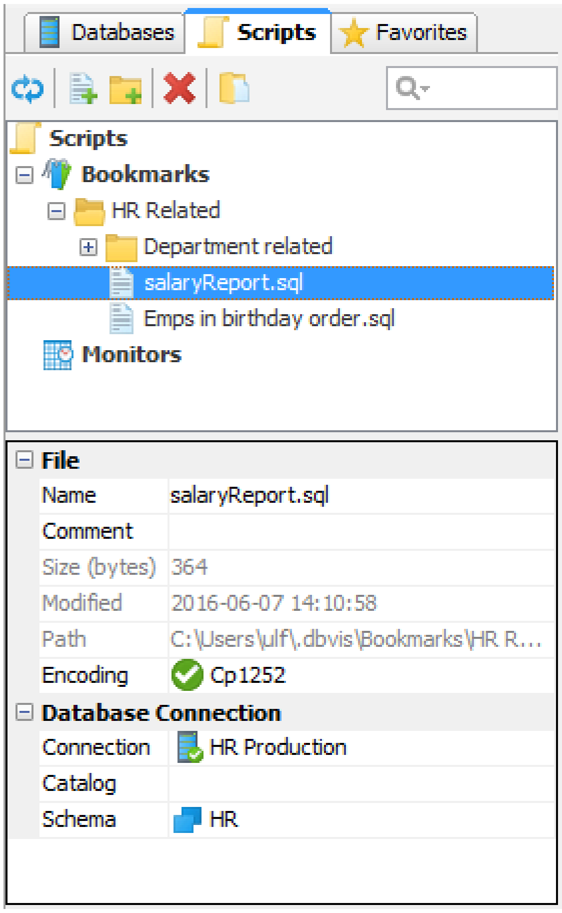
There is also a **Pending Transactions at Disconnect** setting in the Tool Properties dialog, in the Transaction category under the General tab. It specifies what DbVisualizer should do when you disconnect a connection that has pending changes, and you can set it to **Commit**, **Rollback** or **Ask**.

9.11 Managing Frequently Used SQL

quickload.pngYou may have a 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 Commander, but DbVisualizer **Bookmarks** make it even easier to work with them. A Bookmark is a script visible in the Scripts tab in the navigation area.

- Creating, Editing and Organizing Bookmarks (see page 188)
- Executing Bookmarks (see page 189)
- Adding a Bookmark as a Favorite (see page 190)
- Sharing Bookmarks (see page 190)
- Using Quick Load (see page 190)

You find your Bookmarks under the Scripts tab in the navigation 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.



- Scripts
  - Bookmarks
    - HR Related
      - Department related
      - salaryReport.sql
      - Emps in birthday order.sql
  - Monitors

File

Name	salaryReport.sql
Comment	
Size (bytes)	364
Modified	2016-06-07 14:10:58
Path	C:\Users\ulf\.dbvis\Bookmarks\HR R...
Encoding	✓ Cp1252

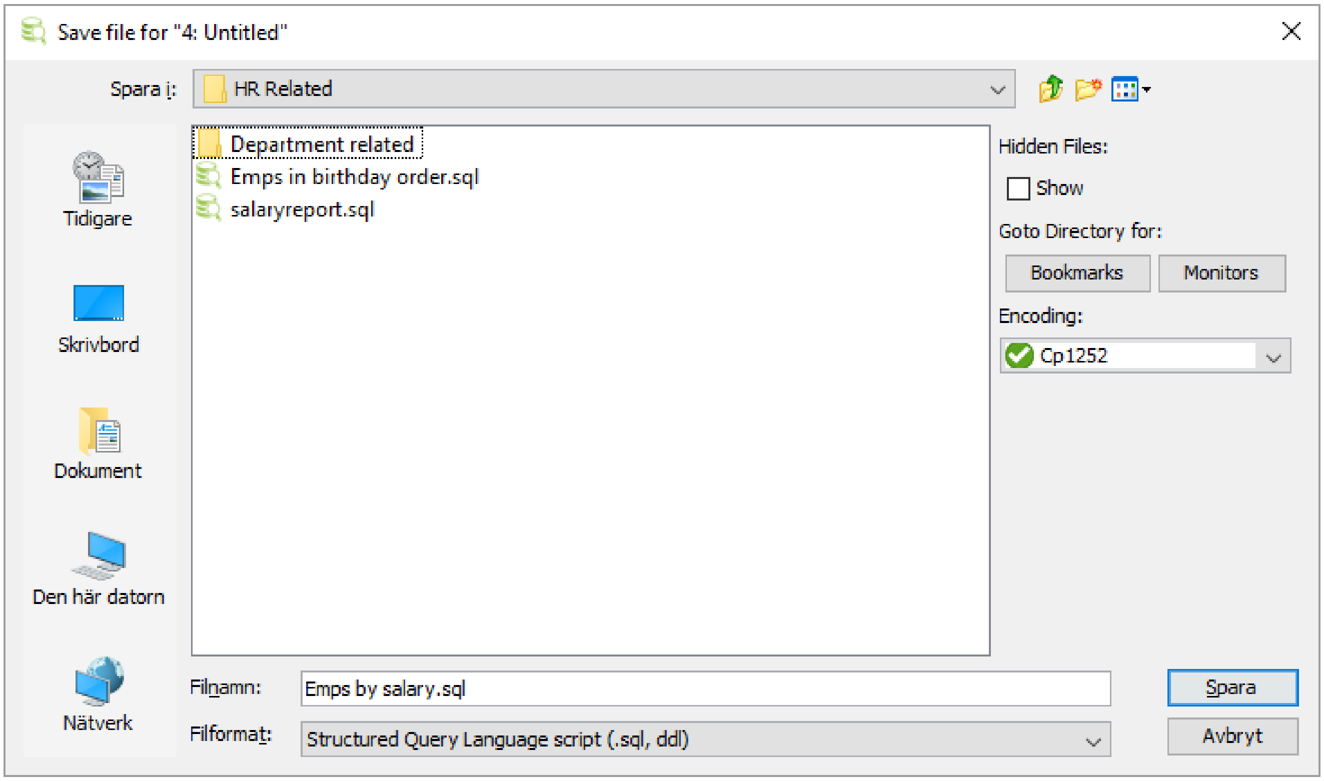
Database Connection

Connection	HR Production
Catalog	
Schema	HR

9.11.1 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** right-click menu item.

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**.



Spara i: HR Related

□ Show

Encoding: Cp1252

Filnamn: Emps by salary.sql

Filformat: Structured Query Language script (.sql, ddl)

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 Commander. Double-click the Bookmark node in the tree or click the corresponding toolbar button to open a new SQL Commander tab for the Bookmark, or activate the SQL Commander that already holds the Bookmark. When you are done with your edits, use the **Save** toolbar button in the SQL Editor to save them.

You can also add the content of a Bookmark to the current content of an SQL Commander 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 folders and the Bookmarks within a folder are ordered alphabetically and cannot be changed manually.

9.11.2 Executing Bookmarks

With a Bookmark opened in an SQL Commander tab, 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 **Open in SQL Commander and Execute** operations in the right-click menu.

9.11.3 Adding a Bookmark as a Favorite

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

If you are using a Bookmark very often, you may find it more convenient to add it to the Favorites area (see page 275). You can drag and drop a Bookmark from the Scripts tab to the Favorites area, or via the **Add to Favorites** right-click menu operation.

9.11.4 Sharing Bookmarks

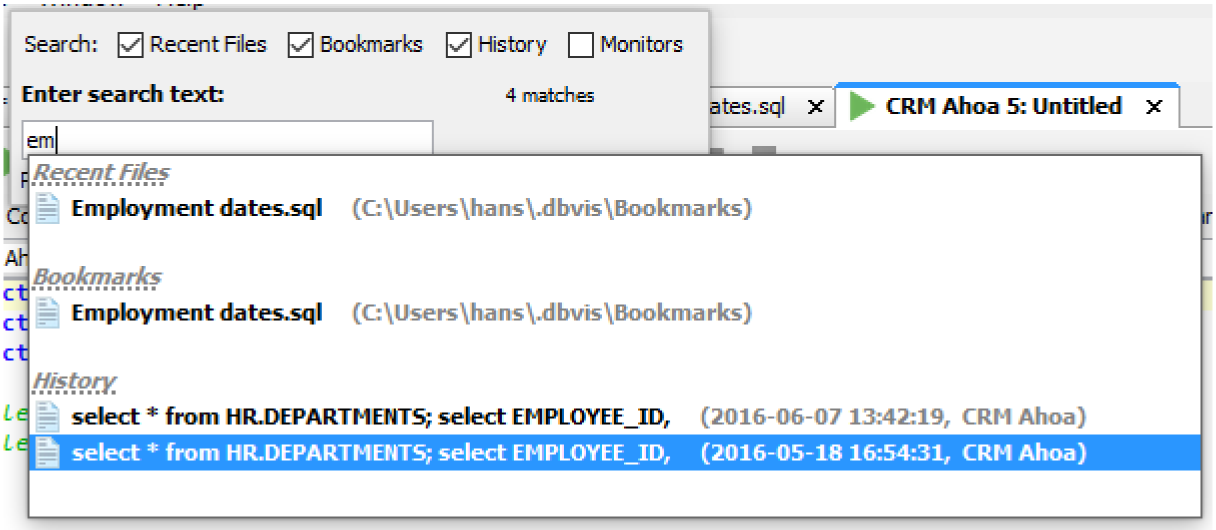
It's easy to share your Bookmarks with someone else because they are stored as regular files. The files are located in a subfolder of the DbVisualizer user preferences folder named *Bookmarks*. The user preferences folder is typically a subfolder named *.dbvis* in your home folder.

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.

9.11.5 Using 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 main toolbar button as well as in the **File->Quick File Open** menu.


Search: ☑ Recent Files ☑ Bookmarks ☑ History □ Monitors
Enter search text:
4 matches
em
ates.sql ✕
CRM Ahoa 5: Untitled ✕
Recent Files
Employment dates.sql (C:\Users\hans\.dbvis\Bookmarks)
Bookmarks
Employment dates.sql (C:\Users\hans\.dbvis\Bookmarks)
History
select * from HR.DEPARTMENTS; select EMPLOYEE_ID, (2016-06-07 13:42:19, CRM Ahoa)
select * from HR.DEPARTMENTS; select EMPLOYEE_ID, (2016-05-18 16:54:31, CRM Ahoa)

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. Use the **Max** field to limit the number of matching files to display in the list.

When you see the file you're looking for, just select it and click **Enter** to load it into an SQL Commander editor. If the file is already loaded in an editor, that tab is made visible instead.

9.12 Creating Queries Graphically

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

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.

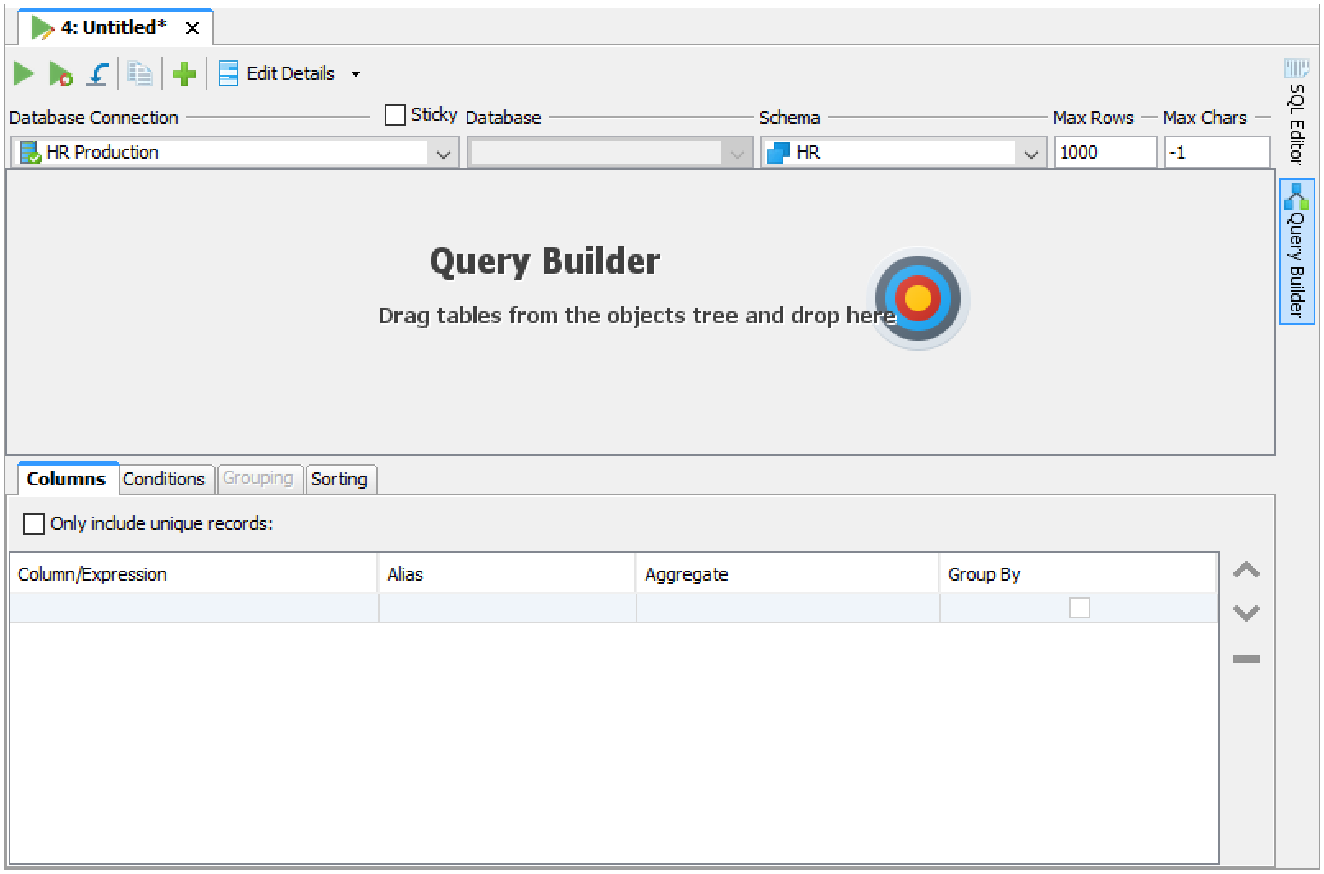
  - Delimited Identifiers (see page 204)
- Current Limitations (see page 204)

The Query Builder is part of the SQL Commander, alongside the SQL Editor. 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.

9.12.1 Creating a Query

To create a query, open the query builder using the **SQL Commander->Show Query Builder** menu choice or click the vertical **Query Builder** button in the SQL Commander. Make sure that the controls in the top section of the Query Builder are set correctly.



Database Connection: HR Production

□ Sticky

Database:

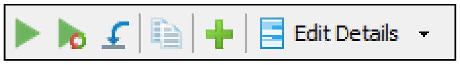
Schema: HR

Max Rows: 1000

Max Chars: -1

□ Only include unique records:

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:



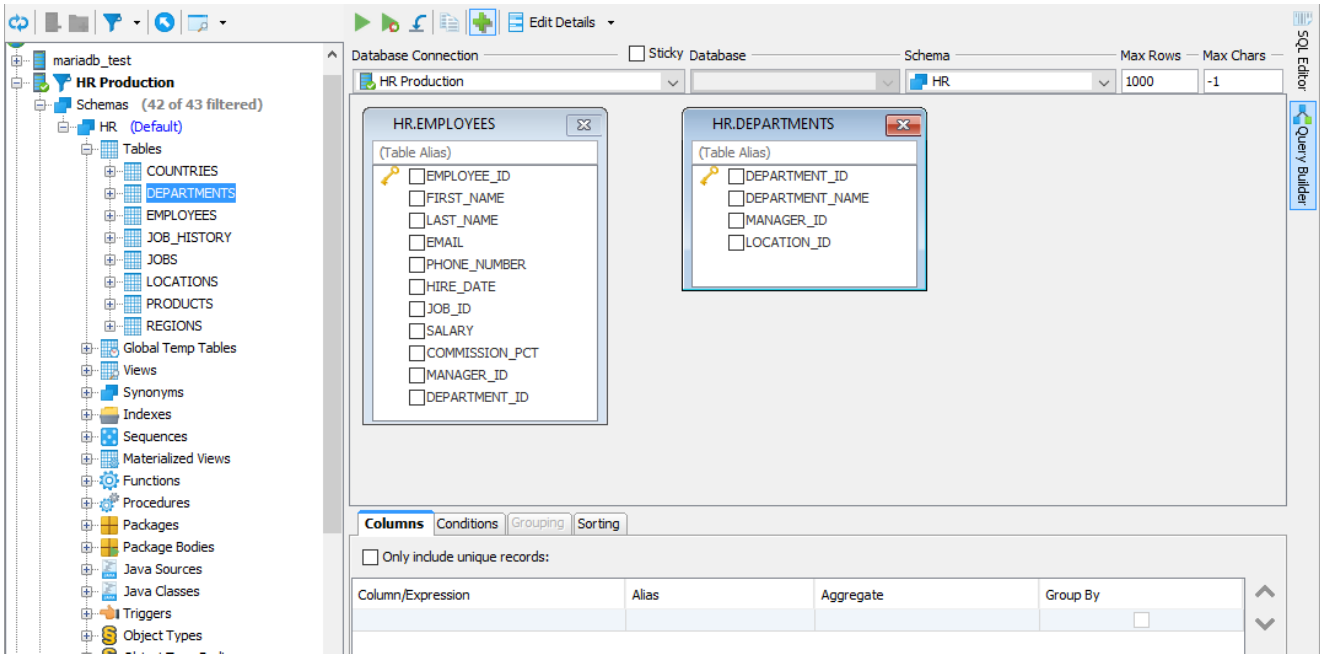
1. The first button (from left) replaces the content of the SQL Editor with the query SQL and executes it,
2. The second button replaces the content of the SQL Editor with the query SQL, without executing it,
3. The third button adds the query last in the SQL Editor,
4. The fourth button copies the query to the system clipboard,
5. The fifth button opens a dialog that lets you add tables matching a search criteria,
6. The sixth button is a drop-down menu for selecting what to show below the diagram area: query details or the SQL preview.

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 (see page 202).

Adding tables

To add tables using drag and drop, 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.



Database Connection: HR Production
□ Sticky
Database:
Schema: HR
Max Rows: 1000
Max Chars: -1

□ Only include unique records:

Column/Expression	Alias	Aggregate	Group By
			□

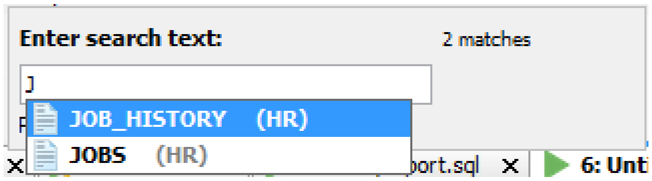
Drag and Drop

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. You can select multiple tables and/or views and drag and drop them together.

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. Use the check box in front of each name 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.

An alternative to dragging and dropping tables into the Query Builder is to use the Quick Table Add dialog.



Enter search text: J

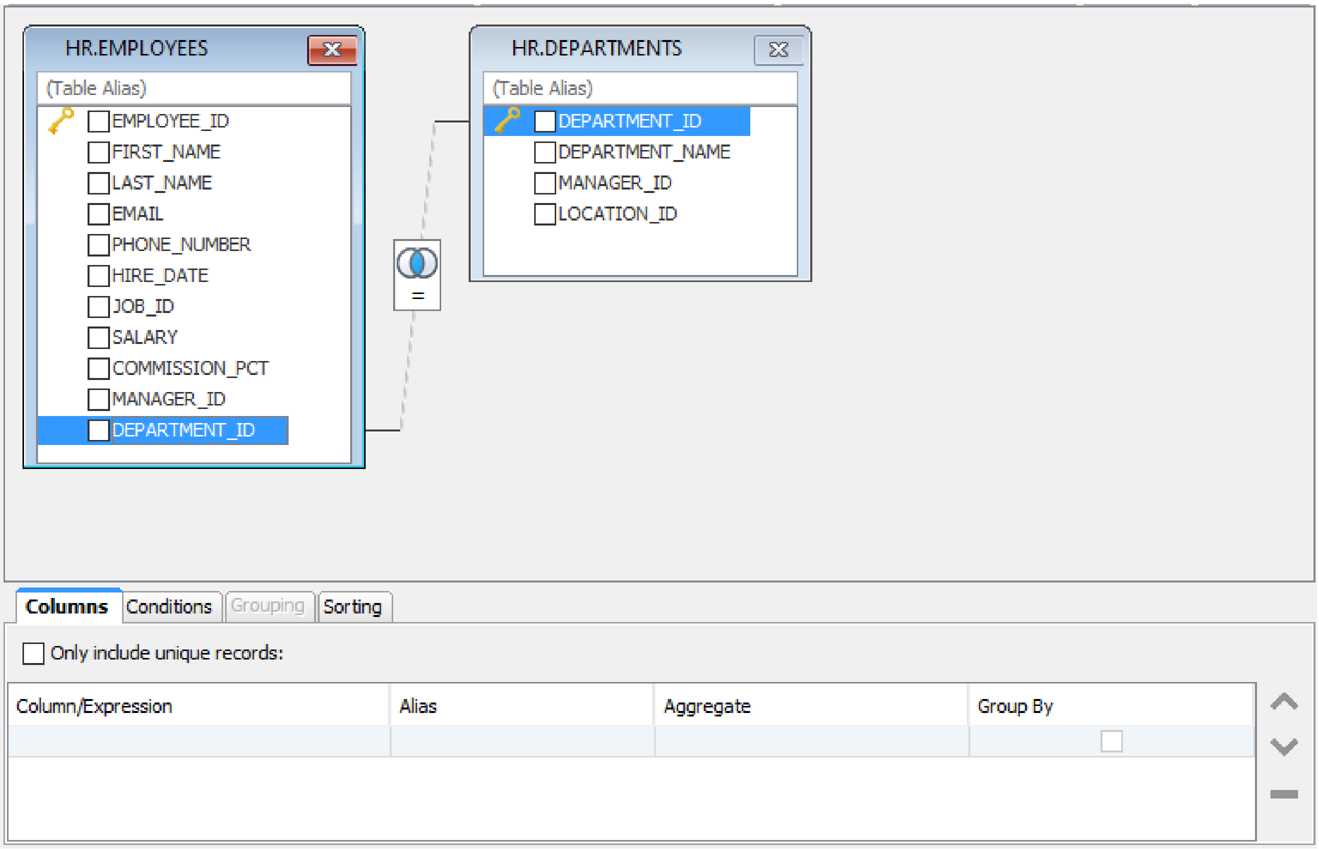
2 matches

- JOB_HISTORY (HR)
- JOBS (HR)

It lists tables matching the search criteria as you type it in the search text field. An asterisk ("*") can be used as a wildcard for any characters.

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.



HR.EMPLOYEES

(Table Alias)

- □ EMPLOYEE_ID
- □ FIRST_NAME
- □ LAST_NAME
- □ EMAIL
- □ PHONE_NUMBER
- □ HIRE_DATE
- □ JOB_ID
- □ SALARY
- □ COMMISSION_PCT
- □ MANAGER_ID
- □ DEPARTMENT_ID

HR.DEPARTMENTS

(Table Alias)

- □ DEPARTMENT_ID
- □ DEPARTMENT_NAME
- □ MANAGER_ID
- □ LOCATION_ID

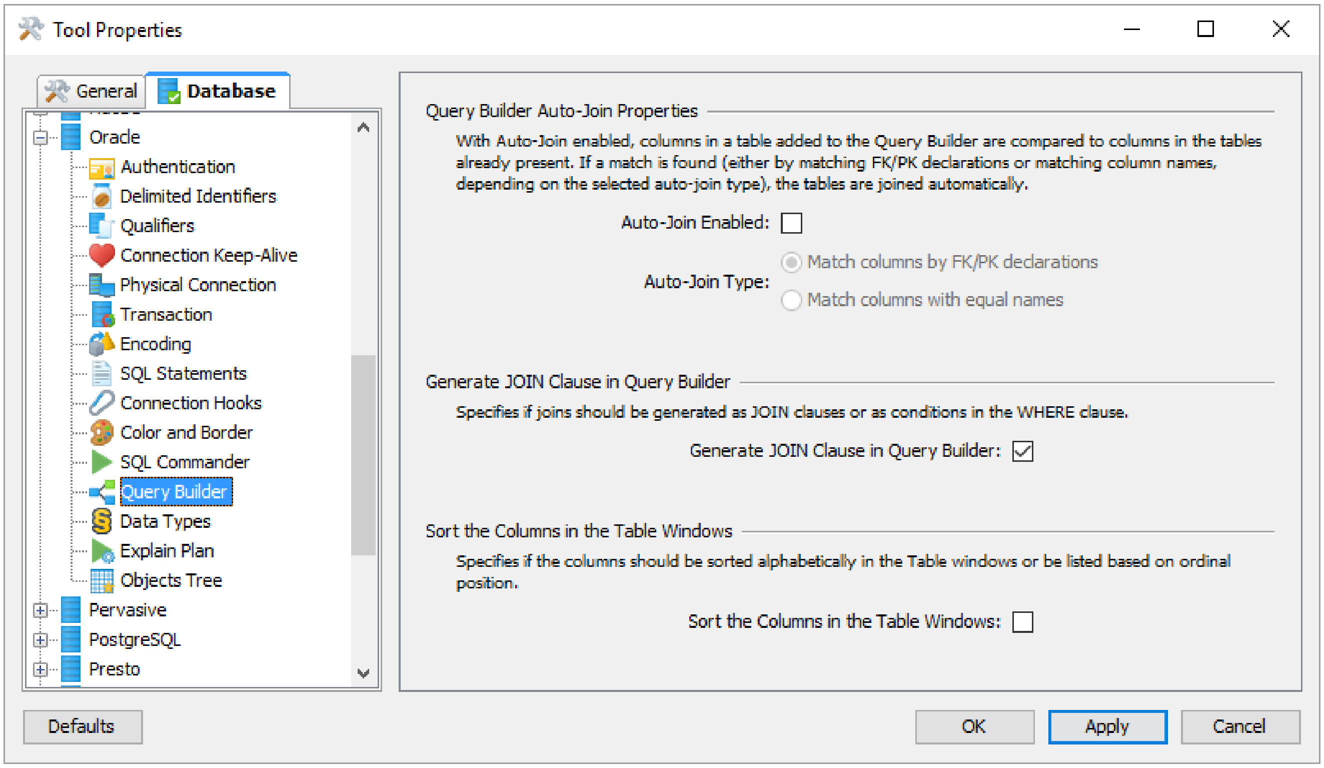
□ Only include unique records:

Column/Expression	Alias	Aggregate	Group By
			□

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.

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 in the **Object View** tab for the connection).



Query Builder Auto-Join Properties

With Auto-Join enabled, columns in a table added to the Query Builder are compared to columns in the tables already present. If a match is found (either by matching FK/PK declarations or matching column names, depending on the selected auto-join type), the tables are joined automatically.

Auto-Join Enabled: □

Auto-Join Type: ● Match columns by FK/PK declarations ○ Match columns with equal names

Generate JOIN Clause in Query Builder

Specifies if joins should be generated as JOIN clauses or as conditions in the WHERE clause.

Generate JOIN Clause in Query Builder: ☑

Sort the Columns in the Table Windows

Specifies if the columns should be sorted alphabetically in the Table windows or be listed based on ordinal position.

Sort the Columns in the Table Windows: □

In the Query Builder category, you can 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.

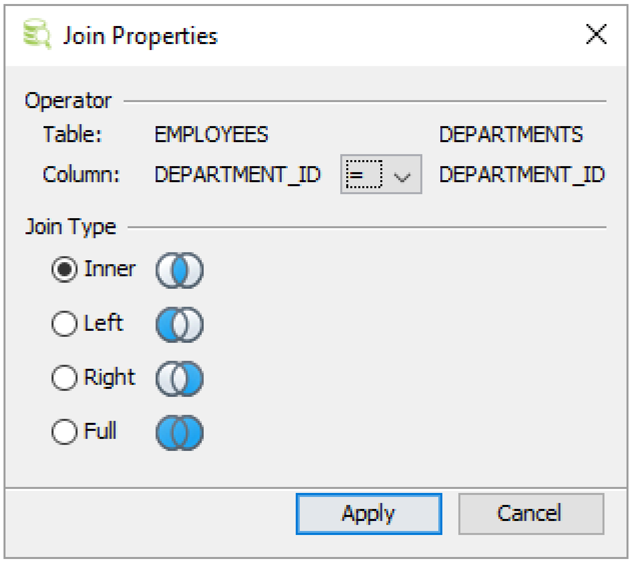
If columns in the table you add are related to other columns in the same table, the Query Builder creates two windows for the table and joins them based on the matching rule. In this case, a table alias is also added for one of the windows so that you can tell the two windows for the same table apart.

Setting 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.



Operator

Table:	EMPLOYEES		DEPARTMENTS
Column:	DEPARTMENT_ID	=	DEPARTMENT_ID

Join Type

● Inner
○ Left
○ Right
○ Full

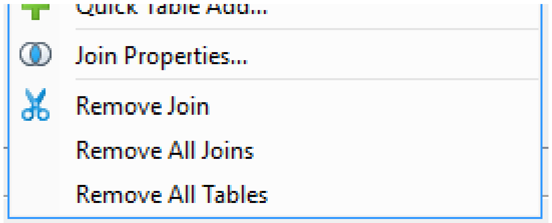
If you have multiple join conditions (linked columns) between two tables, you can specify different conditional operators for each join condition, but the join type is shared between all join conditions; if you change it for one join condition, it is changed for all the other join conditions linking 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)
```

Removing 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.



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

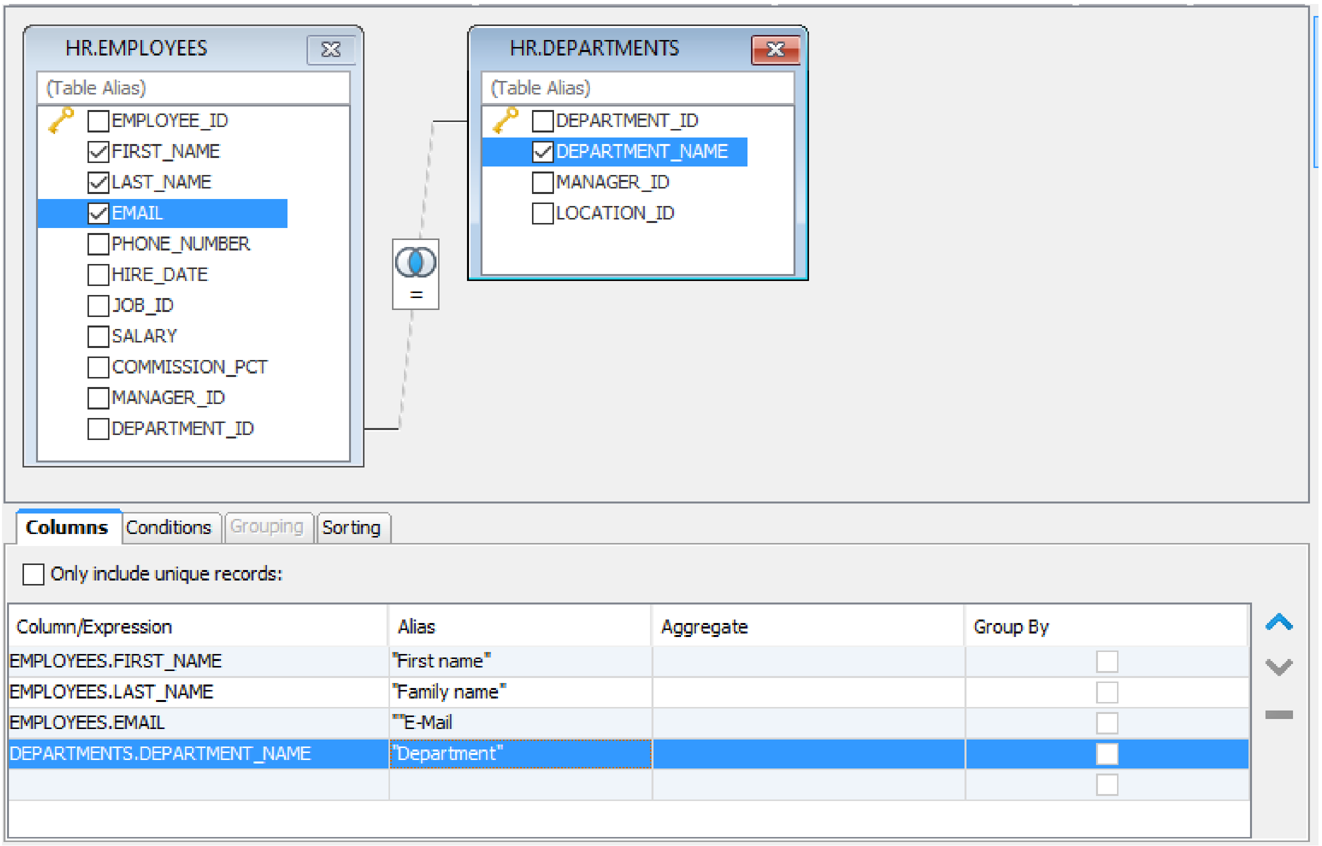
Specifying Query Details

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).

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**.



HR.EMPLOYEES

- ☐ EMPLOYEE_ID
- ☑ FIRST_NAME
- ☑ LAST_NAME
- ☑ EMAIL
- ☐ PHONE_NUMBER
- ☐ HIRE_DATE
- ☐ JOB_ID
- ☐ SALARY
- ☐ COMMISSION_PCT
- ☐ MANAGER_ID
- ☐ DEPARTMENT_ID

HR.DEPARTMENTS

- ☐ DEPARTMENT_ID
- ☑ DEPARTMENT_NAME
- ☐ MANAGER_ID
- ☐ LOCATION_ID

☐ Only include unique records:

Column/Expression	Alias	Aggregate	Group By
EMPLOYEES.FIRST_NAME	"First name"		☐
EMPLOYEES.LAST_NAME	"Family name"		☐
EMPLOYEES.EMAIL	""E-Mail		☐
DEPARTMENTS.DEPARTMENT_NAME	"Department"		☐
			☐

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.

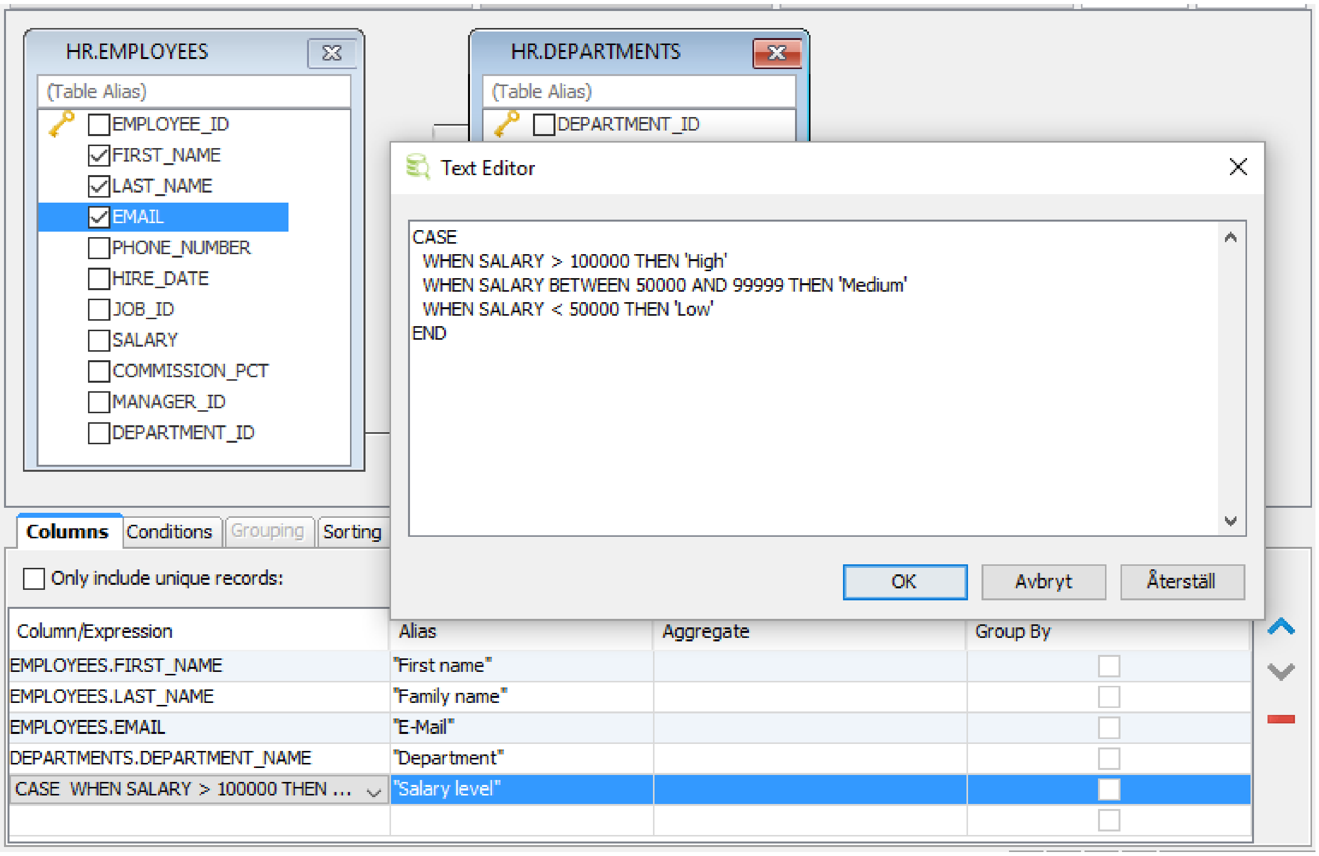
The **Aggregate** and **Group** by fields are used in combination:

- 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.

You can also launch a multi-line text editor for a custom expression, to make it easier to edit a large expression such as a CASE clause. Just double-click the expression cell, and then click on the editor launch button.

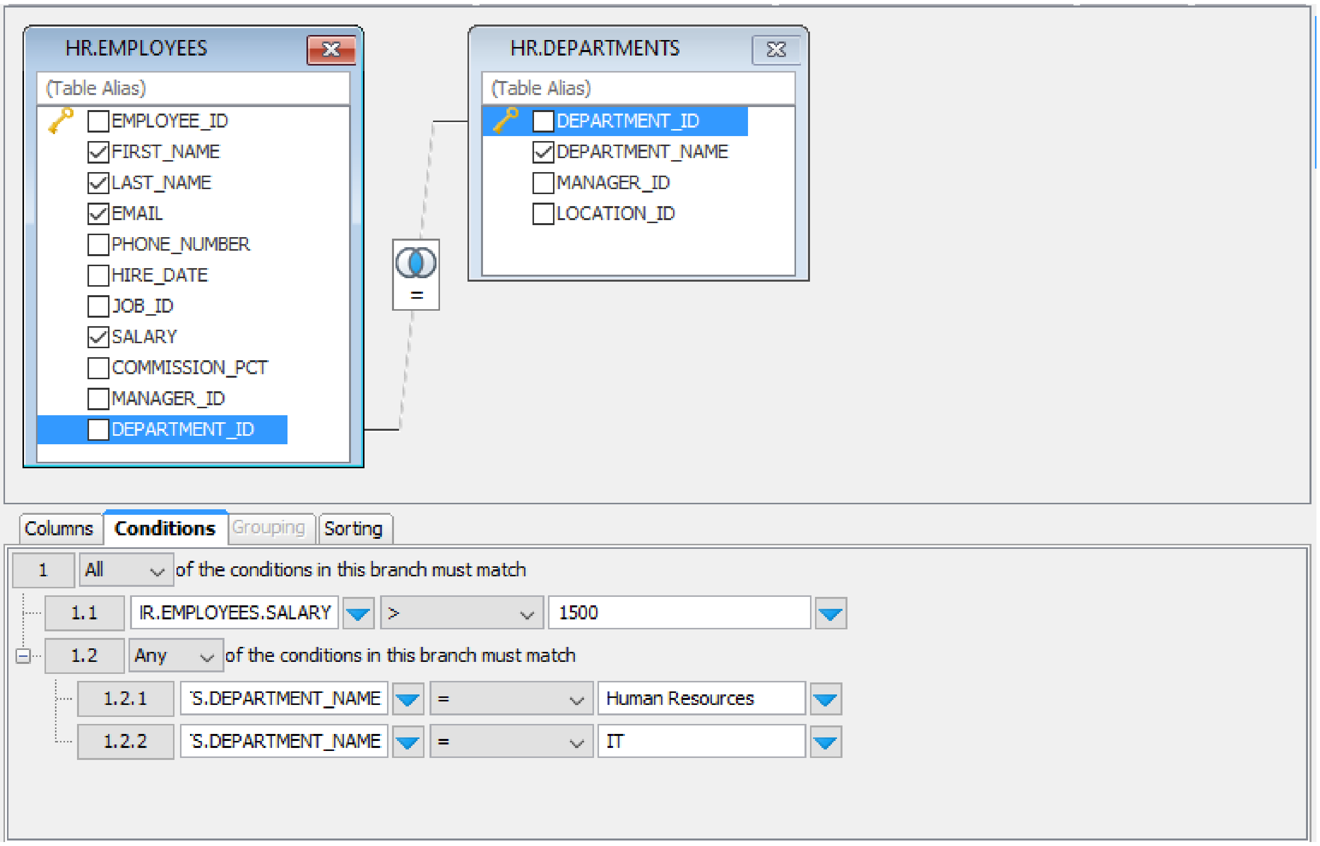


```
CASE
  WHEN SALARY > 100000 THEN 'High'
  WHEN SALARY BETWEEN 50000 AND 99999 THEN 'Medium'
  WHEN SALARY < 50000 THEN 'Low'
END
```

Column/Expression	Alias	Aggregate	Group By
EMPLOYEES.FIRST_NAME	"First name"		☐
EMPLOYEES.LAST_NAME	"Family name"		☐
EMPLOYEES.EMAIL	"E-Mail"		☐
DEPARTMENTS.DEPARTMENT_NAME	"Department"		☐
CASE WHEN SALARY > 100000 THEN ...	"Salary level"		☐
			☐

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.



HR.EMPLOYEES

(Table Alias)

- □ EMPLOYEE_ID
- ☑ FIRST_NAME
- ☑ LAST_NAME
- ☑ EMAIL
- □ PHONE_NUMBER
- □ HIRE_DATE
- □ JOB_ID
- ☑ SALARY
- □ COMMISSION_PCT
- □ MANAGER_ID
- □ DEPARTMENT_ID

HR.DEPARTMENTS

(Table Alias)

- □ DEPARTMENT_ID
- ☑ DEPARTMENT_NAME
- □ MANAGER_ID
- □ LOCATION_ID

Columns | **Conditions** | Grouping | Sorting

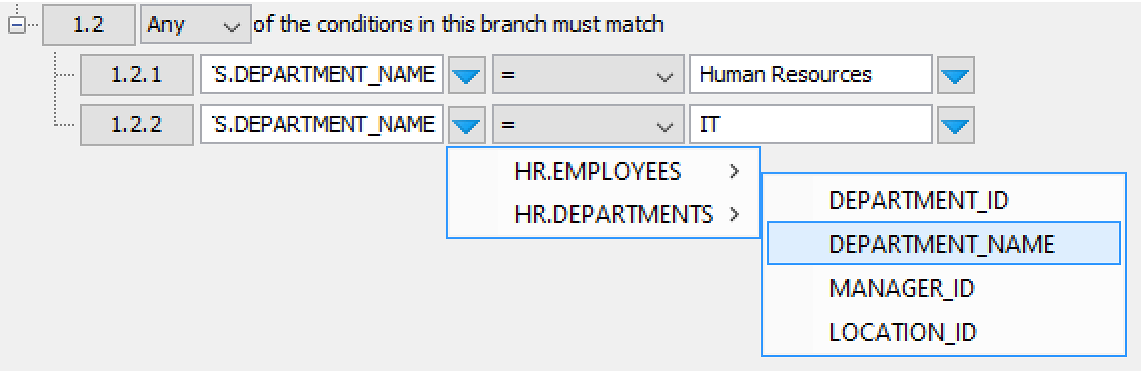
```
1      All  of the conditions in this branch must match
  1.1    IR.EMPLOYEES.SALARY   >   1500
  1.2    Any  of the conditions in this branch must match
    1.2.1  'S.DEPARTMENT_NAME  =   Human Resources
    1.2.2  'S.DEPARTMENT_NAME  =   IT
```

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.



1.2 Any of the conditions in this branch must match

1.2.1 'S.DEPARTMENT_NAME = Human Resources

1.2.2 'S.DEPARTMENT_NAME = IT

A condition field may also contain a custom expression, and just as for a custom expression in the columns list, you can launch a multi-line editor for the expression by selecting the field and click the editor launch button.

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 in the **Conditions** tab, 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.

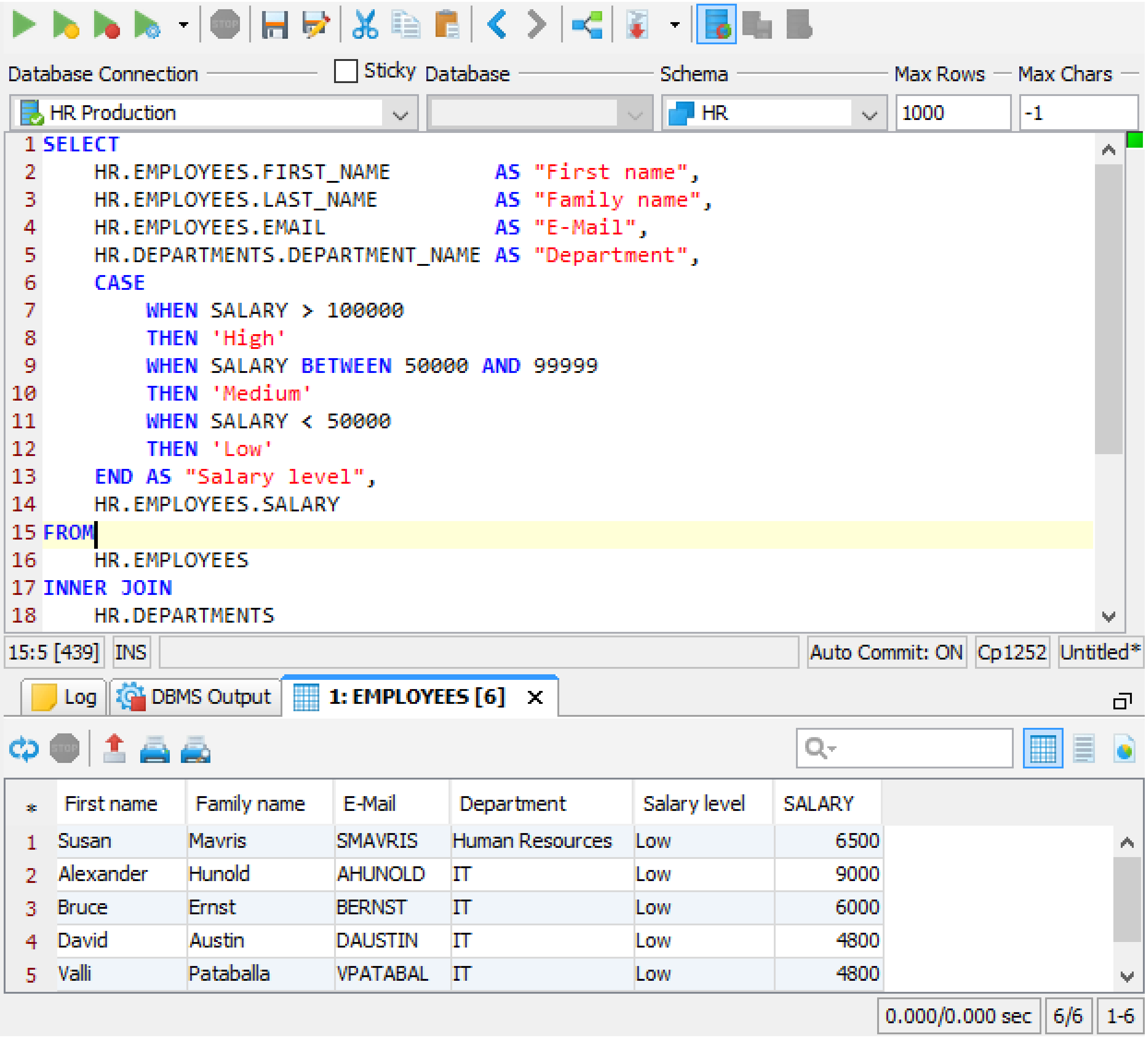
Finally, the **Sorting** tab is used to specify how the final result set will be sorted. All columns for the tables in the graph, plus any custom expressions created for the selection list in the **Columns** tab, are listed in the **Sorting** tab.

SQL Preview

Select **SQL Previe**w in the drop-down menu in the toolbar to show a preview of the final SQL. This is a read-only view and cannot be modified.

9.12.2 Testing the Query

To test the query, simply press the appropriate toolbar button in the Query Builder to copy the SQL to the SQL Editor. Then execute the SQL as usual in the SQL Editor. Or click the **Execute** button in the Query Builder toolbar to copy and execute the SQL in one step.



Database Connection: HR Production

☐ Sticky

Database:

Schema: HR

Max Rows: 1000

Max Chars: -1

```
SELECT
    HR.EMPLOYEES.FIRST_NAME         AS "First name",
    HR.EMPLOYEES.LAST_NAME          AS "Family name",
    HR.EMPLOYEES.EMAIL              AS "E-Mail",
    HR.DEPARTMENTS.DEPARTMENT_NAME AS "Department",
    CASE
        WHEN SALARY > 100000
        THEN 'High'
        WHEN SALARY BETWEEN 50000 AND 99999
        THEN 'Medium'
        WHEN SALARY < 50000
        THEN 'Low'
    END AS "Salary level",
    HR.EMPLOYEES.SALARY
FROM
    HR.EMPLOYEES
INNER JOIN
    HR.DEPARTMENTS
```

15:5 [439] INS Auto Commit: ON Cp1252 Untitled*

	First name	Family name	E-Mail	Department	Salary level	SALARY
1	Susan	Mavris	SMAVRIS	Human Resources	Low	6500
2	Alexander	Hunold	AHUNOLD	IT	Low	9000
3	Bruce	Ernst	BERNST	IT	Low	6000
4	David	Austin	DAUSTIN	IT	Low	4800
5	Valli	Pataballa	VPATABAL	IT	Low	4800

0.000/0.000 sec 6/6 1-6

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

9.12.3 Loading a Query From the 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 **Load in Query Builder** 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.

9.12.4 Properties for the Query Builder

In addition to the **Auto Join** setting discussed above, there are a few other 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 in the **Object View** tab for the connection).

Express joins as JOIN clause or WHERE condition

The **Generate JOIN Clause in SQL Builder** property is available in the **[Database Type]->Query Builder** category. 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** category.

Delimited Identifiers

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

9.12.5 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, comments, 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.
- There is only limited support for the CASE clause, in that everything between CASE and END is treated as uninterpreted text. This means that, as opposed to plain object references in the select list or conditions, column names and other identifiers within a CASE clause are not affected by changes to the Query Builder property settings, such as Delimited Identifiers and Qualifiers.

9.13 Formatting SQL

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **SQL->Format SQL** menu contains four operations for formatting SQL statements.

Format Buffer formats the complete editor content and **Format Current** formats the current SQL (at cursor position).

The formatting is done according to the settings defined in the Tool Properties dialog, in the **SQL Editor/SQL Formatting** category under the General tab. There are many things you can configure. After making some changes, press **Apply** and format again to see the result.

Here is an example of an SQL statement 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-Discount)) desc)
order by Region
```

And here is the same statement after formatting has been applied with default settings:

```
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
```

Copy Formatted and **Paste Formatted** are powerful tools for copying SQL statements between programs written in languages like Java, C#, PHP, VB, etc. and the SQL Editor. Both operations display a dialog where you can adjust some of the formatting options, most importantly the **Target SQL** option and the **SQL is Between** option. **Target SQL** can be set to a number of common programming language formats.

For instance, to copy an SQL statement and paste it as Java code for adding it to a Java StringBuffer:

1. Select the statement,
2. Choose **SQL->Format SQL->Copy Formatted**,
3. Set **Target SQL** to Java StringBuffer,
4. Click **Format** to place the formatted statement on the system clipboard,
5. Paste it into your Java code.

To copy a statement wrapped in code from a program:

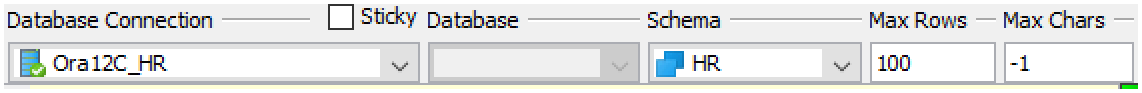
1. Select the code containing an SQL statement in your program,
2. Copy it to the system clipboard,
3. Choose **SQL->Format SQL->Paste Formatted**,
4. Check **SQL is Between** and enter the character enclosing the SQL statement in the code,
5. Click **Format** to extract the SQL statement and paste the formatted SQL in the editor.

9.14 Using Max Rows and Max Chars for Queries

DbVisualizer limits the number of rows shown in the result set tab to 1000 rows, by default. This is done to conserve memory. If this limit prevents you from seeing the data of interest, you should first consider:

1. Using a WHERE clause in the query to only retrieve the rows of interest instead of all rows in the table,
2. Exporting the table (see page 109) to a file

If you really need to look at more than 1000 rows, you can change the value in the **Max Rows** field in the SQL Commander toolbar. Use a value of 0 or -1 to get all rows, or a specific number (e.g. 5000) to set a new limit.



Database Connection	□ Sticky Database	Schema	Max Rows	Max Chars
Ora12C_HR		HR	100	-1

Character data columns may contain very large values that use up lots of memory. If you are only interested in seeing a few characters, you can set the **Max Chars** field in the SQL Commander toolbar to the number of characters you want to see.

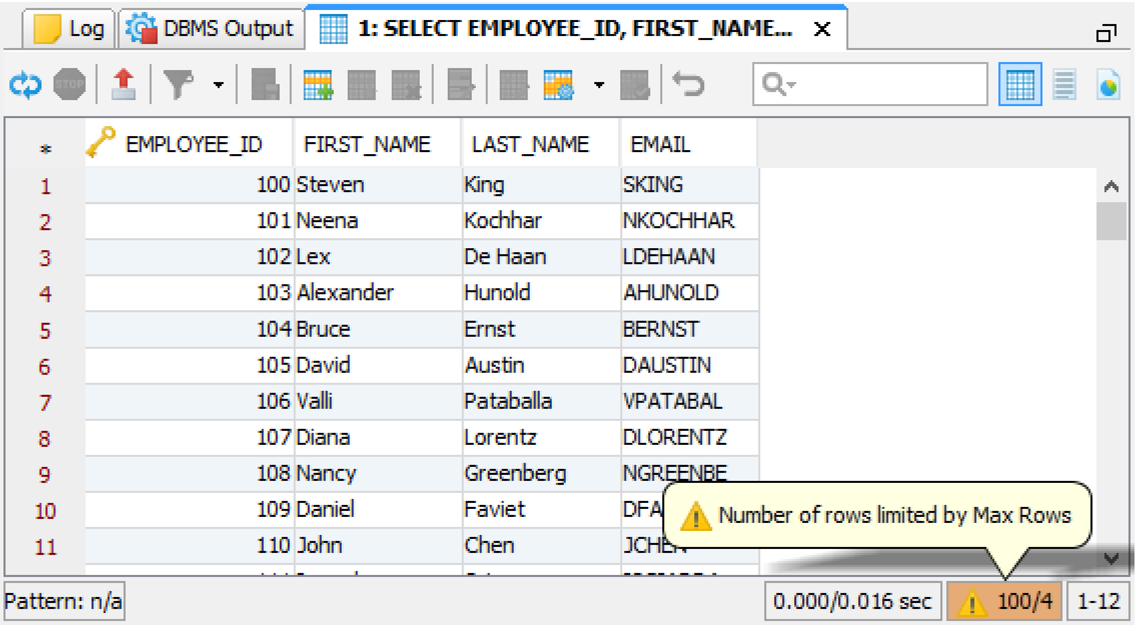
You can define how to deal with columns that have more characters than the specified maximum in the Tool Properties dialog, in the Grid category under the General tab. You have two choices: **Truncate Values** or **Truncate Values Visually**.

- **Truncate Values** truncates the original value for the grid cell to be less then the setting of Max Chars.

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** truncates 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.

When the grid data is limited due to either the **Max Rows** or **Max Chars** value, you get an indication about this in the rows/columns field in the grid status bar and in the corresponding limit field.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
1	100	Steven	King	SKING
2	101	Neena	Kochhar	NKOCHHAR
3	102	Lex	De Haan	LDEHAAN
4	103	Alexander	Hunold	AHUNOLD
5	104	Bruce	Ernst	BERNST
6	105	David	Austin	DAUSTIN
7	106	Valli	Pataballa	VPATABAL
8	107	Diana	Lorentz	DLORENTZ
9	108	Nancy	Greenberg	NGREENBE
10	109	Daniel	Faviet	DFA
11	110	John	Chen	JCHEN

Number of rows limited by Max Rows

Pattern: n/a 0.000/0.016 sec 100/4 1-12

Along with the highlighted field, a warning pops up close to the filed. You can disable this behavior in the Tool Properties dialog, in the **Grid** category under the General tab.

9.15 Getting the DDL for an Object

You can use the `@dll` command in a script to get the DDL for a number of different database object types. The command supports this general syntax:

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

where <`objType`> is one of `table`, `indexesfortable`, `view`, `procedure`, `function`, `package` (Oracle only), `packagebody` (Oracle only), `objecttype` (Oracle only), `objecttypebody` (Oracle only), `sequence` (Oracle only), `synonym` (Oracle only), `module` (Mimer SQL only) or `trigger`, and <`objId`> is the qualified identifier for the object (case sensitive). Here's an example:

```
@ddl table="HR.EMPLOYEES";
```

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.

9.16 Using the Log Tab

The Log reports progress and results when executing in the **SQL Commander**, **Import**, **Export Schema/ Database**, **Procedure Editor**, and when running **Database Object Actions**. The appearance of the Log may differ slightly depending on which of these function is used. The Log was completely redesigned in 10.0 and now saves unlimited number of log entries to file. Due to memory constraints, the Log grid in the DbVisualizer tool keeps up to 10 000 entries until it starts truncating the oldest entries.

Note that the detail level in a log entry message depends on the driver and database that is being used. Some databases are very good at telling you what went wrong and why, while others provide less detail.

9.16.1 Preprocessing Script

The preprocessing step is performed just before a script is executed. It locates any variables, parameters, and restricted commands (defined in the Permissions category in Tools->Tool Properties) in the script. Depending how large the script is, preprocessing may take some time to complete and DbVisualizer may even advise you to run without preprocessing if the script is too large.

Check Executing an External Script (see page 180) page for more information.

9.16.2 Executing

Disabled functionality during execution

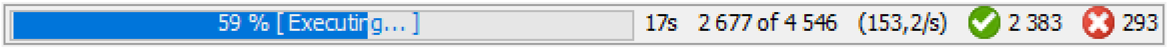
When a script is executed some functions in the Log are disabled:

- Quick Search
- Sorting
- Filtering
- Expand row heights (double-click)
- Hyperlinks to highlight errors and statements
- Save Log to Text File

Progress information

While executing there is progress information in the Log status bar showing overall metrics for the execution. It helps getting a quick overview of number of failed and successful statements, elapsed time, and number of statements executed per time unit.

The progress bar shows how much of the script has been executed. The progress bar switch to an indeterminate mode when **Preprocess Script** is switched off or when running a script using the **@run** client-side command. The reason is that the Log then doesn't count how many statements to execute.



59 % [Executing...] 17s 2 677 of 4 546 (153,2/s) ✓ 2 383 ✗ 293

Auto scroll

When new log entries are added the Log grid is automatically scrolled to the bottom. This can be turned off in the toolbar or by right-click in the grid, and select **Auto Scroll**.

Auto Clear log

Having auto clear checked, the log is cleared when execution is started. Disabling auto clear will keep all entries until number of entries exceeds 10 000 when truncation is triggered.

Log truncation

When the number of entries in the grid exceeds 10 000 the log is truncated by removing the oldest entries. When the log is truncated an icon and "Truncated" text is displayed in the grid status bar. If having auto clear log disabled, you will still see the STARTED and FINISHED entries for a previous executions in the grid along with an entry telling how many log entries has been truncated for that execution.

9.16.3 Auto resize row heights

The **Message** and **SQL/Command** columns may include multi-lined content. During execution only a single line of content is displayed in the grid. Once the execution has completed, and there are any FAILED statements these rows automatically expands to show all of its content. In the right-click menu under **Auto Resize Row Heights**, the auto resizing may be configured to also expand SUCCESS entries.

A cell having multi-lined content ends with three dots, "...". To manually expand/collapse a row with more content, double-click the row.

9.16.4 Navigating to next/previous failed log entry

To navigate to next/previous failed statement, use the **Goto Next Failed** and **Goto Previous Failed** actions in the right-click menu. An alternative is to filter out all but FAILED log entries in the grid, see Filter and search (see page 211) section below for more information.

9.16.5 Highlight statement or error in the SQL editor

If there is an editor associated with the log grid, such as the SQL Commander, the **Command** column for all log entries is represented by a hyperlink. Clicking the link will navigate to, and highlight the corresponding SQL/ command in the SQL editor.

The same applies for any FAILED statements whereas the **Status** column is then a hyperlink, when clicked it will show the statement that failed. Depending on if the database reports error positions, clicking the link will set the caret at the position where the error started.

If a single statement produces multiple errors, clicking on the FAILED status link, will then show a menu with all errors. Selecting an error highlights it in the SQL editor.

For **@run** commands, clicking the @run link in the **Command** column shows a menu in which you can select to load the related file in a new SQL Commander tab.

The matching actions are in the right-click menu, **Highlight Statement** and **Highlight Failed Statement**.

Saving all Log entries to text file

All log entries are written to file, even those that has been truncated in the Log grid. The **Save Log to Text File** right-click menu action saves this log to text. Note that the log file is cleared between executions if auto clear log is enabled and when manually clearing the log. It is also removed when closing down the SQL Commander tab.

The internal log file is stored in json format under the $HOME/.dbvis/sqllogs folder with the .dson extension. A log file may be loaded in the Log grid using **Load Log File** in the right-click menu.

9.16.6 Copy Log Entries to clipboard

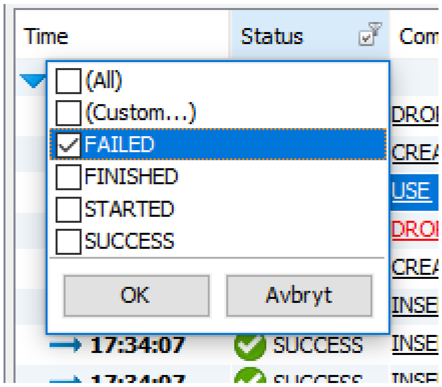
The copy log entries to clipboard actions transforms the selected rows to text format and puts these on the system clipboard. This is useful if you want to share all or some log entries with others. The difference with the standard **Copy Selection** is that the latter copies only the selected cells in standard grid copy format typically separated with tabs.

9.16.7 Copy executed SQLs to the SQL Commander

The SQLs displayed in the log may be copied as a script to an existing or a new SQL Commander. Just select the rows you want to copy and then right-click and **Load Selected Commands into Editor**.

9.16.8 Filter and search

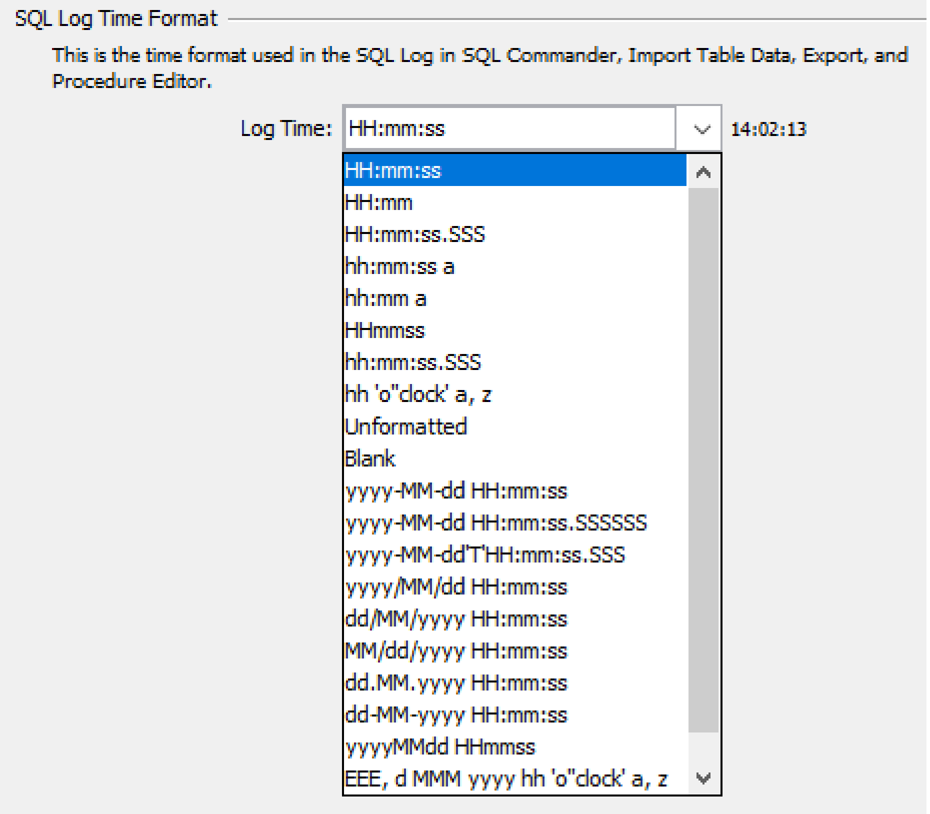
The filtering capability is useful to quickly limit what entries are displayed in the grid. A good example is to click the filtering icon in the **Status** column header and then de-select all statuses than **FAILED** (or just double-click FAILED to de-select the others). The result is that only FAILED statements are displayed in the log grid. That type of filtering can be applied to any on the columns (and in fact in most grids in DbVisualizer). The quick search field can be used to further refine to do full text search on all entries in the grid.



□ (All)
□ (Custom...)
☑ FAILED
□ FINISHED
□ STARTED
□ SUCCESS

9.16.9 Configure the Log

The **Time** column in the log shows by default **hour:minute:second** (HH:mm:ss) but can be configured to show date, milliseconds, and more. Change this in **Tools->Tool Properties** and the **General / SQL Commander / SQL Log** category.



SQL Log Time Format

This is the time format used in the SQL Log in SQL Commander, Import Table Data, Export, and Procedure Editor.

Log Time: HH:mm:ss 14:02:13

```
HH:mm:ss
HH:mm
HH:mm:ss.SSS
hh:mm:ss a
hh:mm a
HHmmss
hh:mm:ss.SSS
hh 'o''clock' a, z
Unformatted
Blank
yyyy-MM-dd HH:mm:ss
yyyy-MM-dd HH:mm:ss.SSSSSS
yyyy-MM-dd'T'HH:mm:ss.SSS
yyyy/MM/dd HH:mm:ss
dd/MM/yyyy HH:mm:ss
MM/dd/yyyy HH:mm:ss
dd.MM.yyyy HH:mm:ss
dd-MM-yyyy HH:mm:ss
yyyyMMdd HHmmss
EEE, d MMM yyyy hh 'o''clock' a, z
```

9.17 Writing to the Log Tab

You can use the @echo client side command to write information to the Log tab.

```
@echo <message>
```

The message may contain DbVisualizer variables (see page 227), e.g. one of the predefined variables.

```
@echo Today is ${dbvis-date}$ and the time is ${dbvis-time}$
```

Variables can also be used to display values produced by executing a function or stored procedure (see page 144):

```
@call ${STATUS||(null)||String||noshow dir=out}$ = "HR"."GET_STATUS"(1002);
@echo STATUS: ${STATUS}$;
```

9.18 Using the DBMS Output Tab

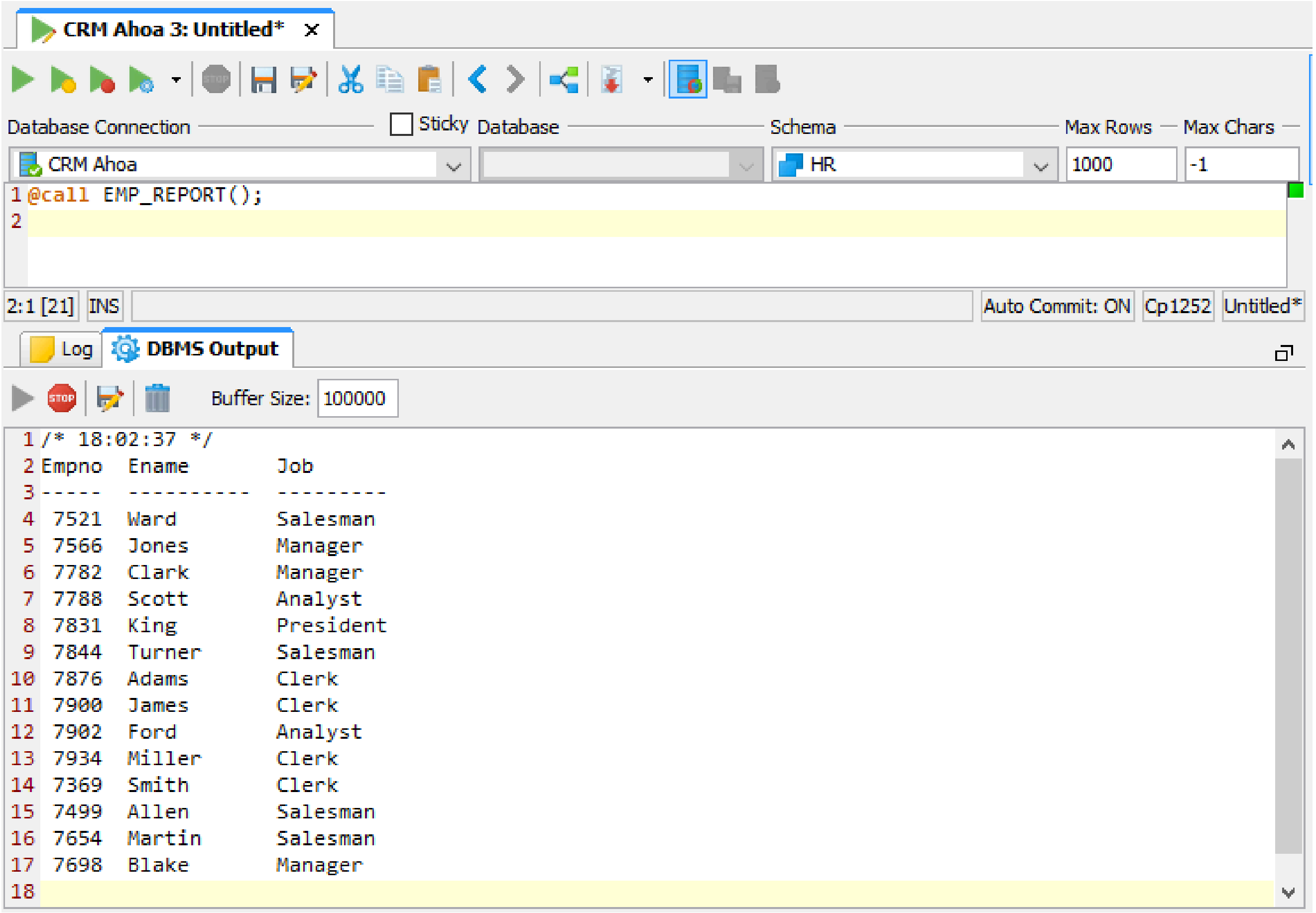
Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

The **DBMS Output** tab is only available for Oracle and DB2 LUW databases. It is used to capture messages produced by stored procedures, packages, and triggers using the DBMS_OUTPUT utility. 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,

1. Select the **DBMS Output** tab
2. Press the **Enable** button.

Once DBMS output is enabled, the icon in the tab header is changed. Invoking a stored procedure that produces messages in the SQL Commander results in content similar to this in the output tab. (Each block of output is separated with a timestamp).



Database Connection: CRM Ahoa
Database:
Schema: HR
Max Rows: 1000
Max Chars: -1

```
@call EMP_REPORT();
```

2:1 [21] INS   Auto Commit: ON   Cp1252   Untitled*

Buffer Size: 100000

```
/* 18:02:37 */
Empno  Ename       Job
-----  ----------  ---------
 7521  Ward        Salesman
 7566  Jones       Manager
 7782  Clark       Manager
 7788  Scott       Analyst
 7831  King        President
 7844  Turner      Salesman
 7876  Adams       Clerk
 7900  James       Clerk
 7902  Ford        Analyst
 7934  Miller      Clerk
 7369  Smith       Clerk
 7499  Allen       Salesman
 7654  Martin      Salesman
 7698  Blake       Manager
```

If you never use the DBMS Output tab and want to preserve screen real estate, you can select to not show it in the connection properties, in the SQL Commander category.

9.19 Comparing SQL Scripts

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

You can compare a script in one SQL Commander editor to a scripts in another SQL Commander editor, or to the original file loaded into the editor.

- To compare the script to another script, select **Compare** from the right-click menu in one editor and pick the SQL Commander holding the other script in **Select Object** dialog to compare their text content (see page 261).
- To compare the script to the original file, select **Compare to Saved** from the right-click menu.

9.20 Exporting Query Results

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

Instead of viewing query results in Result Set grids, you can export the result of one or more queries to a file. For very large results, this may be the preferred choice due to memory constraints.

To export a query result, create a script with

1. an **@export on** command,
2. an **@export set** command,
3. one or more queries,
4. an **@export off** command.

Here is a basic example:

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

The @export set command takes a parameter name followed by an equal sign and a value. You can use the following parameters, where only filename is required and all names are case-insensitive:

Parameter	Default	Valid Values
AppendFile	false	true, false, clear (i.e. start with a new file for the first result and then append to it)
BinaryFileDir		Path for data files when BinaryFormat is set to **File**, see Exporting a Table (see page 109) for details. Note! Variables for columns must include the scope option when entered manually into a script, e.g. `/Users/hans/exp/${dbvis-date}$/${COUNTRIES\|\|\|\|\|\|scope=post}$`
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		Path for data files when CLOBFormat is set to **File**, see Exporting a Table (see page 109) for details. Note! Variables for columns must include the scope option when entered manually into a script, e.g. `/Users/hans/exp/${dbvis-date}$/${COUNTRIES\|\|\|\|\|\|scope=post}$`
CLOBFormat	Value	Don't Export, Size, Value, File
CsvColumnDelimiter	\t (TAB)	
CsvIncludeColumnHeader	true	true, false
CsvIncludeColumnHeaderPerResult	false	
CsvColumnHeaderIsColumnAlias	true	true, false
CsvIncludeSQLCommand	Don't Include	Don't Include, Top, Bottom
CsvSplitFileSize	-1	Split the result over multiple files if it is larger than the specified size, or -1 to never split. The size must be specified as *`size`*`[ g | G | m | M | k | K ]`
CsvRemoveNewlines	false	true, false

Parameter	Default	Valid Values
CsvRowCommentIdentifier		
CsvRowDelimiter	\n	\n (UNIX/Linux/macOS), \r\n (Windows)
DateFormat	yyyy-MM-dd	See valid formats in Changing the Data Display Format (see page 108) document
DecimalNumberFormat	Unformatted	See valid formats in Changing the Data Display Format (see page 108) document
Destination	File	File
Encoding	UTF-8	
ExcelColumnHeaderIsColumnAlias	true	true, false
ExcelFileFormat	Binary	Two formats are supported • Binary Excel (xls) supporting export of max 65 535 rows. Specified as "***Binary Excel (xls)***", "***Binary***" or "***xls***" • OOXML, Excel 2007 (xlsx). Specified as "***OOXML, Excel 2007 (xlsx)***", "***OOXML***" or "***xlsx***"
ExcelIncludeColumnHeader	true	true, false
ExcelIncludeSQLCommand	false	true, false
ExcelIntroText		Any description
ExcelSheetName		Used when exporting to excel. Sets the name of exported excel sheet.
ExcelTextOnly	false	true, false
ExcelTitle		Any title
Filename	**REQUIRED**	
Format	Based on file extension, or CSV if none	CSV, HTML, XML, SQL, XLS, JSON. If Format is not specified, the file extension

Parameter	Default	Valid Values
		is used to determine the format. If there is no recognized file extension, CSV is used as the default.
HtmlColumnHeaderIsColumnAlias	true	true, false
HtmlConvertChars	true	true, false. Set to false if you have HTML code in the exported data, so that e.g. < and > characters are not converted to `&lt;` and `&gt;`
HtmlFooter	[Generated by: DbVisualizer *version*]	Any text to use in the document footer. Can be set to blank to remove the footer.
HtmlIncludeSQLCommand	false	true, false
HtmlIntroText		Any description
HtmlPerTableHeader	E.g. **Date:** **2017-05-31 16:48:23** **Columns:** **4** **Table:** **COUNTRIES**	HTML code that describes the table. To fit into the rest of the HTML code, it must start with `<tr>` and end with `</tr>`. The pre-defined DbVisualizer variables can be used, e.g. `${dbvis-object}$` to include the table name.
HtmlTitle		Any title
JSONColumnHeaderIsColumnAlias	true	true, false
JSONStyle	Array	Array, Rows
NumberFormat	Unformatted	See valid formats in Changing the Data Display Format (see page 108) document
QuoteDuplicateEmbedded	true	true, false (quote char is the same as QuoteTextData)
QuoteTextData	None	None, Single, Double
Settings		The path to an XML file containing all settings

Parameter	Default	Valid Values
ShowNullAs	(null)	
SqlAfterExportStmts		Any statements to include in the script after the SQL statements for the exported objects, e.g. `set foreign_key_checks = 1;`
SqlBeforeExportStmts		Any statements to include in the script before the SQL statements for the exported objects, e.g. `set foreign_key_checks = 0;`
SqlBeginIdentifier		Character to use to begin a quoted identifier. Note! To specify a double-quote, you must duplicate it since double-quote is also used to enclose the parameter value.
SqlBlockBeginDelim		String to use to begin an SQL block when exporting complex DDL statements using the @ddl command.
SqlBlockEndDelim		String to use to end an SQL block
SqlDelimitedIdentifiers	true	true, false
SqlEndIdentifier		Character to use to end a quoted identifier . Note! To specify a double-quote, you must duplicate it since double-quote is also used to enclose the parameter value.
SqlGroupBy	Object	Object, Statement. Set to Object to generate DROP, CREATE, INSERT, and ALTER statements (where applicable) for each exported object in turn. Set to Statement to group all statements of the same type together, e.g. first DROP statements for all exported objects, then CREATE statements for all exported objects, etc.
SqlIncludeCreateDDL	false	true, false
SqlIncludeSQLCommand	Don't Include	Don't Include, Top, Bottom

Parameter	Default	Valid Values
SqlQualifier		Qualifier to use when qualifying table names. If not set, DbVisualizer tries to determine the schema and use it as the qualifier.
SqlQualifyColumnName	false	true, false
SqlQualifyObjectName	true	true, false
SqlRowCommentIdentifier	--	
SqlSeparator	;	Statement separator character.
SqlSplitFileSize	-1	Split the result over multiple files if it is larger than the specified size, or -1 to never split. The size must be specified as *`size`* [`g` | `G` | `m` | `M` | `k` | `K`]
TableName		Can be set if DbVisualizer cannot determine the value for the **${ dbvis-object}$** variable
TimeFormat	HH:mm:ss	See valid formats in Changing the Data Display Format (see page 108) document
TimeStampFormat	yyyy-MM-dd HH:mm:ss.SSSSSS	See valid formats in Changing the Data Display Format (see page 108) document
XmlColumnHeaderIsColumnAlias	true	true, false
XmlIncludeSQLCommand	false	true, false
XmlIntroText		Any description
XmlStyle	DbVisualizer	DbVisualizer, XmlDataSet, FlatXmlDataSet

Here are a few examples using some of these settings.

9.20.1 Automatic table name to file 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.

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

> There must be only one table name in a select statement in order to automatically set the filename with the **${dbvis-object}$** variable, i.e if the select joins from several tables or pseudo tables are used, you must explicitly name the file.
>
> The **${dbvis-object}$** variable is not substituted with a table name if using the AppendFile="true/clear" parameter.

9.20.2 Multiple results to 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;
@export off;
```

9.20.3 Using predefined settings

If you save settings when exporting a table (see page 109) or a schema (see page 152), you can use the **Settings** parameter to reference the settings file.

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

9.20.4 Limit the number of exported rows

You can use the `@set maxrows` command in combination with the `@export` command to override the Max Rows (see page 206) field value in the SQL Commander tab toolbar.

```
@set maxrows 10;
@export on;
@export set filename="c:\Backups\alltables.sql" format="sql";
select * from Orders;
select * from Products;
select * from Transactions;
@export off;
```

If Max Rows is set to a positive number, you can use the `@set maxrows` command to set it to `-1` to export all rows.

9.21 Using Permissions in the SQL Commander

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

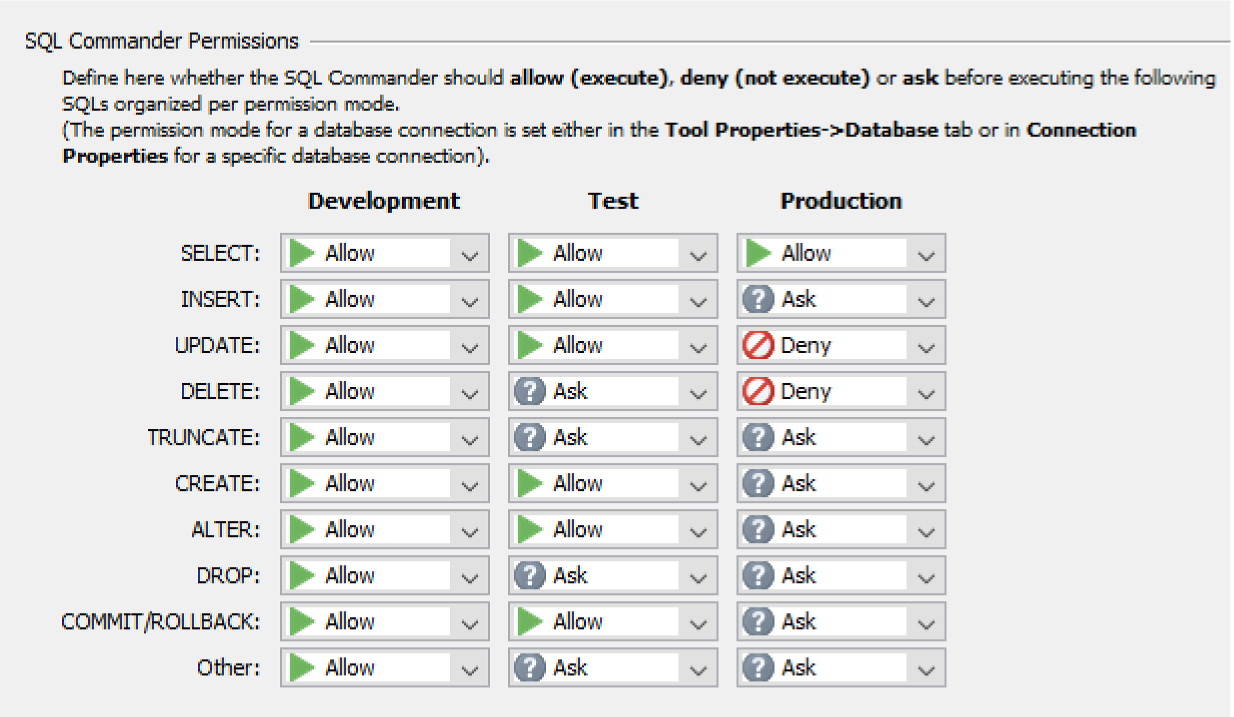
The **Permission** functionality is a security mechanism, where you can specify that certain database operations must be confirmed. You configure permissions in the Tool Properties dialog, in the **Permissions** category of the General tab, per *connection mode* (Development, Test and Production).

You specify which connection mode to use for a connection in the **Properties** tab of the Object View tab for the connection.

> The permission feature is part of DbVisualizer and does not replace the authorization system in the actual database.

For the SQL Commander, you can pick the permission mode 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.



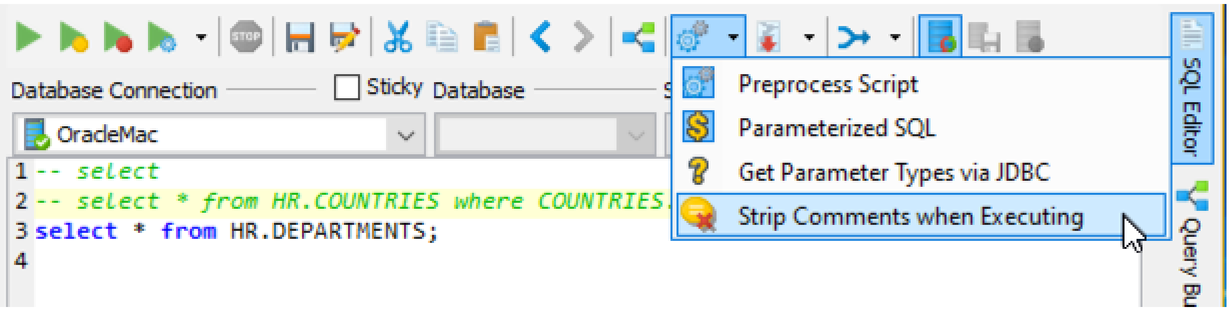
SQL Commander Permissions

Define here whether the SQL Commander should **allow (execute)**, **deny (not execute)** or **ask** before executing the following SQLs organized per permission mode.
(The permission mode for a database connection is set either in the **Tool Properties->Database** tab or in **Connection Properties** for a specific database connection).

	Development	Test	Production
SELECT:	Allow	Allow	Allow
INSERT:	Allow	Allow	Ask
UPDATE:	Allow	Allow	Deny
DELETE:	Allow	Ask	Deny
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

9.22 Sending Comments to the Database with Statements

Comments in an SQL script, identified by the delimiters defined in Tool Properties dialog in the **SQL Commander/Comments** category under the General tab, are sent to the database by default. In some cases, you may not want to include the comments with the statement, for instance if your database does not handle them. You can then enable the stripping of comments using the **Strip Comments when Executing** toggle item in the **Processing Controls** menu button in the SQL Commander toolbar .



```
-- select
-- select * from HR.COUNTRIES where COUNTRIES
select * from HR.DEPARTMENTS;
```

9.23 Using Client-Side Commands

You can use DbVisualizer these client-side command in your scripts.

Command	Description
`@cd <directory>@run <file> [ <variables> ]`	Use these command to execute an external script (see page 180).
`@export`	Use this command to export the result of a query (see page 214) or procedure call to a file.
`@delimiter <new_delimiter>`	Use this command to temporarily change the statement delimiter for a complex statement (see page 178).
`@call <function_or_proc>`	Use this command to execute a function or procedure (see page 144).
`@beep`	Use this command to emit a beep sound, e.g. to indicate a significant point in a script.
`@sleep <milliseconds>`	Use this command to halt the processing the specified number of milliseconds.
`@echo <text>`	Use this command to write to the Log tab (see page 212).

`@window iconify`
`@window restore`
	Use these commands to lower (iconify) or raise (restore) the DbVisualizer main window.

`@desc <table>`	Use this command to show column information for a table. The table name may be qualified with a schema and/or database name.
`@ddl <params>`	

Command	Description
	Use this command to get the DDL for a database object (see page 207).

`@log <file_name> [spool|close]`	Use this command to write log messages to a named file.
	Examples
	`@log /tmp/myLog.txt;`
	`@log /tmp/myLog.txt close;`
	`@log /tmp/allLog.txt spool;`
	1. Logs subsequent commands to the file /tmp/myLog.txt. If we are already logging to another file as the result of a previous @log command that log is closed
	2. Stops logging to the file
	3. Writes all log messages up to this @log command to the named file

`@spool log <file_name>`	This command is not supported any more. Please use the @log command instead.
`@stop on error` `@stop on sqlwarning@stop on norows`@continue on error @continue on sqlwarning @continue on norows	Use these commands to control what to do when a statement results in a warning or an error (see page 175).

`@set autocommit on`
`@set autocommit off`
	Use these commands to control the Auto Commit (see page 185) state.

`@commit` `@rollback`	Use these commands to explicitly commit or rollback (see page 185) updates.

`@set serveroutput on`
`@set serveroutput off`
	Use these commands to enable or disable output to the DBMS Output tab (see page 212) for Oracle databases.

`@set maxrows <number>`
`@set maxchars <number>`
	Use these commands to adjust the Max Rows and Max Chars (see page 206) limits for specific queries.

`@set resultset name <name>`	Use this command to name any following result set. When executed with a following SQL statement that produces a result set the tab showing this result set will be named using the <name> supplied as a parameter to the command.

Command	Description

	Example:
	`@set resultset name MyEmployees;`
	`select * from EMPLOYEE;`
	DbVisualizer will for the example above show the result of the select in a tab named **MyEmployees**

9.24 Parameterized SQL - Variables and Parameter Markers

A useful feature in the SQL Commander is to use variables or parameter markers in SQL scripts. Variables are used to express that certain parts of the SQL should be replaced with values when the SQL is executed. If you use a script to perform repetitive tasks, such as creating a user and granting permissions, just insert variables for the user name and permissions to grant in the script and DbVisualizer will prompt for the values at execution.

In addition to DbVisualizer's own variable syntax, two parameter marker syntaxes supported natively by some databases/drivers can also be used. This makes it easier to use SQL statements from other tools or code as-is in DbVisualizer, but you need to be aware of the limitations in how they are used compared to DbVisualizer variables.

The following gives an overview of the different formats and how they can be used.

> Even if DbVisualizer supports several variable formats it doesn't mean you can always copy/paste the SQLs including the parameter markers to another application and successfully execute it. You need to check the compatibility for the actual connector/driver/framework and even that the database itself supports the used syntax.

The following variable syntaxes are supported by DbVisualizer:

DbVisualizer Variables	
`${variable\|\|value\|\|type\|\|options}$`	This is the most flexible syntax as it supports setting a name, default value, data type, and other options. Check the DbVisualizer Variables (see page 227) section for details. A DbVisualizer variable can be used anywhere in the SQL as the specified value replaces the variable definition as a literal (unless a data type is specified; with a data type, its behavior is exactly the same as for **Named Parameter Markers**).

DbVisualizer Variables

Example

```
select *
from   EMPLOYEE
where  FIRST_NAME like '${First Name||Phil}$'
and    AGE > ${Age||20}$
```

The variable identifiers, **${...}$** can be modified in **Tools->Tool Properties** and in the **General / Variables** category.

Named Parameter Markers

&name
:name
`:{name}:`
`'name'`

These syntaxes are supported natively by a few databases. This format allows only a name for the parameter and no other settings, such as type or default value. The parameter name is the name DbVisualizer shows in the prompt window.

Named parameter values are bound at runtime with the markers in the SQL. Some JDBC drivers/databases requires that the proper data type is set while some are more relaxed. For named (and unnamed) parameter markers, you may chose data type in the prompt window.

Named parameter markers should only be used in contexts supported by the actual database, usually for column values. For example, as opposed to a DbVisualizer variable, a parameter marker cannot be used for a table or column name.

The only difference between **&name**, **:name**, **:{name}** and **:'name'** is that the latter two, **:{name }** and **:'name'**, allow white spaces in the name.

Example

```
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, &FirstName, &LastName, &Address, &Age);
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, :FirstName, :LastName, :Address, :Age);
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, :{FirstName}, :{LastName}, :{Address}, :{Age});
```

Read more about named parameter markers (see page 235).

Unnamed Parameter Markers

? The question marker symbol is probably the most supported parameter marker among the supported databases. It is also the most unintuitive marker since the user has to remember the order of question marks and the corresponding values.

Unnamed Parameter Markers

Since there is no name associated with it, DbVisualizer shows these as **Parameter 1**, **Parameter 2** and so on in the prompt window.

There is no technical difference between how unnamed and named parameter markers are handled internally in DbVisualizer or when processed by the database. All are bound with a prepared SQL statement.

Example

```
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, ?, ?, ?, ?)
```

Use named in favor of unnamed parameter markers if there is support in the target database based on the easier reading of named markers. Read more about unnamed parameter markers (see page 235).

It is not possible to mix DbVisualizer variables and parameter markers, or named and unnamed parameter markers, in the same script. If you do, you will only be prompted for values for one type and the execution will fail.

For more information about the different syntaxes check Using DbVisualizer Variables (see page 227)and Using Parameter markers (see page 235).

9.24.1 Using DbVisualizer Variables

DbVisualizer 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.

A DbVisualizer variable that **doesn't specify a data type** will always be replaced with the **value as a literal**. This allows use of variables anywhere in an SQL statement. If a **data type is specified,** the prompted value will be bound with the SQL and variables can in this context **only be used where supported** by the target database.

DbVisualizer Variables	

`${variable\|\|value\|\|type\|\|options}$`	This is the most flexible syntax as it supports setting a name, default value, data type, and other options. Check the DbVisualizer Variables (see page 227) section for details.
	A DbVisualizer variable can be used anywhere in the SQL as the specified value replaces the variable definition as a literal (unless a data type is specified; with a data type, its behavior is exactly the same as for **Named Parameter Markers**).
	Example
	`select *`
	`from   EMPLOYEE`
	`where  FIRST_NAME like '${First Name\|\|Phil}$'`
	`and    AGE > ${Age\|\|20}$`
	The variable identifiers, **${...}$** can be modified in **Tools->Tool Properties** and in the **General / Variables** category.

Variable Syntax

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

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

This is the complete syntax for a DbVisualizer variable:

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

Part	Default	Description
name	**Required**	Required. This is the name that appear in the prompt window. 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 with that name
value	null	The default value for the variable
type	none (= literal)	The type of variable: String, Boolean, Integer, Float, Long, Double, BigDecimal, Date, Time and Timestamp. In addition DbVisualizer defines: BinaryData and TextData (for CLOB). This is used to determine how the data should be passed between DbVisualizer and the database server. If no type is specified, it is treated as a literal
options	none	The options part is used to express certain conditions. Separate these with a whitespace

Part	Default	Description
		• **pk** Indicates that the variable is part of the primary key in the final SQL. Represented with a symbol in the prompt window • **where** Defines that the variable is part of the WHERE clause. A symbol indicate this condition in the prompt window • **noshow** This option define that the variable should not appear in the prompt window. A value must be set when using this option, unless it is an output variable (see **dir** below) • **nobind** Used in combination with when a **type** is set and defines that the variable should be replaced as a literal in the SQL rather than being bound 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 • **scope=post** The scope must be specified as **post** when using variables representing columns when exporting BLOB/CLOB values to separate files named based on column values.

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.

> ⚠ Note that none of the pre-defined variables below will show in the prompt window.

```
${dbvis-date}$
${dbvis-time}$
${dbvis-timestamp}$${dbvis-connection}$${dbvis-database-type}$
```

By default, date/time variable 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

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

The following variable holds the absolute path to the current directory, e.g. set by the `@cd` command:

```
${dbvis-pwd}$
```

In an sql script, the name on the result set produced by the next SELECT statement can be set with the `@set resultset name` command (see Using Client-Side Commands (see page 223)). This result set name is accessible through the variable

```
${dbvis-resultset-name}$
```

Variables for Java System Properties and OS Environment Variables

You can use DbVisualizer variables to access the value of a Java system property. Just use a name of the format `java.<property>`, e.g.

```
@echo ${java.user.home}$
```

To access the value of an operating system environment variable, use a name of the format `env.<envVariable>`, e.g.

```
@echo ${env.USER}$
```

Using "now" in values for Time, Date, and Timestamp

For variables with the type set to `Time`, `Date`, and `Timestamp`, the value may be set to the literal `now`. The value is then converted to the specified type with the format defined in **Tool Properties** and **Data Formats** category.

Variable	Format	Sample
`${myDate\|\|now\|\|Date}$`	yyyy-MM-dd	2017-07-17
`${myDate\|\|now\|\|Time}$`	HH:mm:ss	09:02:50
`${myDate\|\|now\|\|Timestamp}$`	yyyy-MM-dd HH:mm:ss	2017-07-17 09:03:11

For these types it is also possible to specify the value `now` in the variable prompting window.

Variable Substitution in SQL statements

> ⚠ For variable processing to work in the SQL Commander, make sure the **SQL->Enable Parameterized SQL** main menu option is checked.

A simple variable may look like this:

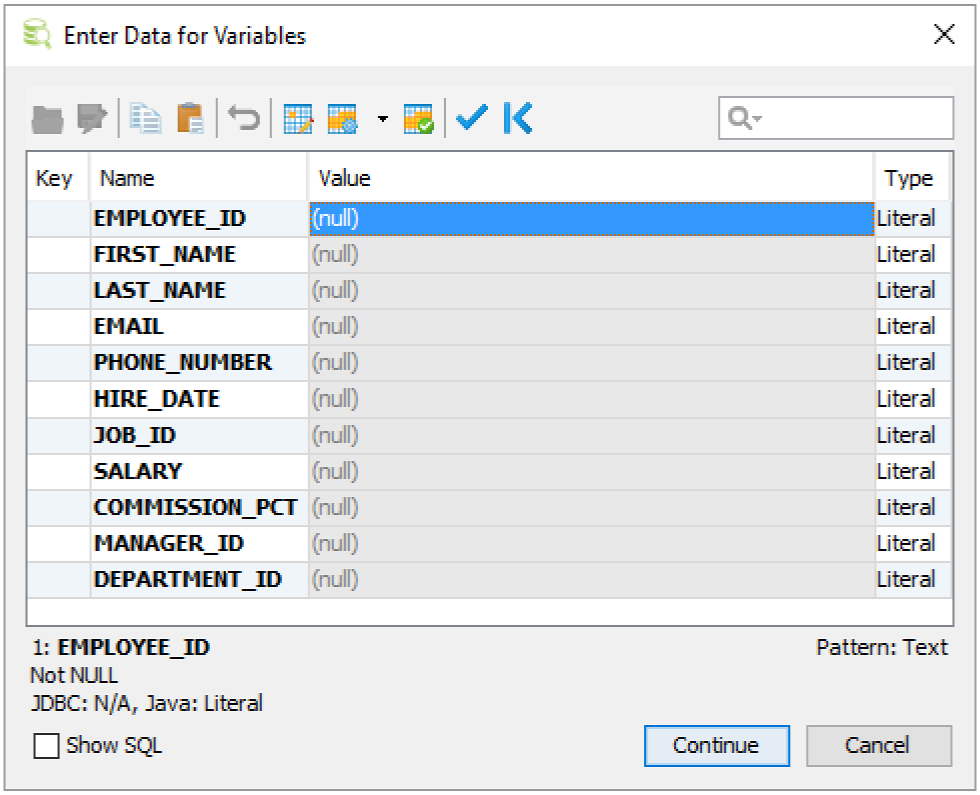
`${FullName}$`

A variable is identified by the start and end sequences, `${...}$`. (These can be re-defined (http://confluence.dbvis.com/display/UG91/Using+DbVisualizer+Variables#UsingDbVisualizerVariables-ChangingtheDelimiterCharacters) in **Tool Properties**). During execution, the SQL Commander searches for variables and displays the prompt 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 as a literal and finally let the database execute the statement.

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

```
INSERT
INTO
    EMPLOYEES
    (
        EMPLOYEE_ID,
        FIRST_NAME,
        LAST_NAME,
        EMAIL,
        PHONE_NUMBER,
        HIRE_DATE,
        JOB_ID,
        SALARY,
        COMMISSION_PCT,
        MANAGER_ID,
        DEPARTMENT_ID
    )
    VALUES
    (
        ${EMPLOYEE_ID}$,
        ${FIRST_NAME}$,
        ${LAST_NAME}$,
        ${EMAIL}$,
        ${PHONE_NUMBER}$,
        ${HIRE_DATE}$,
        ${JOB_ID}$,
        ${SALARY}$,
        ${COMMISSION_PCT}$,
        ${MANAGER_ID}$,
        ${DEPARTMENT_ID}$
    )
```

Executing the above SQL will result in the following prompt window:



Enter Data for Variables

Key	Name	Value	Type
	EMPLOYEE_ID	(null)	Literal
	FIRST_NAME	(null)	Literal
	LAST_NAME	(null)	Literal
	EMAIL	(null)	Literal
	PHONE_NUMBER	(null)	Literal
	HIRE_DATE	(null)	Literal
	JOB_ID	(null)	Literal
	SALARY	(null)	Literal
	COMMISSION_PCT	(null)	Literal
	MANAGER_ID	(null)	Literal
	DEPARTMENT_ID	(null)	Literal

1: EMPLOYEE_ID
Not NULL
JDBC: N/A, Java: Literal

Pattern: Text

Using variables with no data type defined shows these as **Literal**. This means that the specified value will replace the variable as-is in the SQL statement.

The prompt 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 prompt window adds two new commands (leftmost in the toolbar and in the form right-click menu).

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

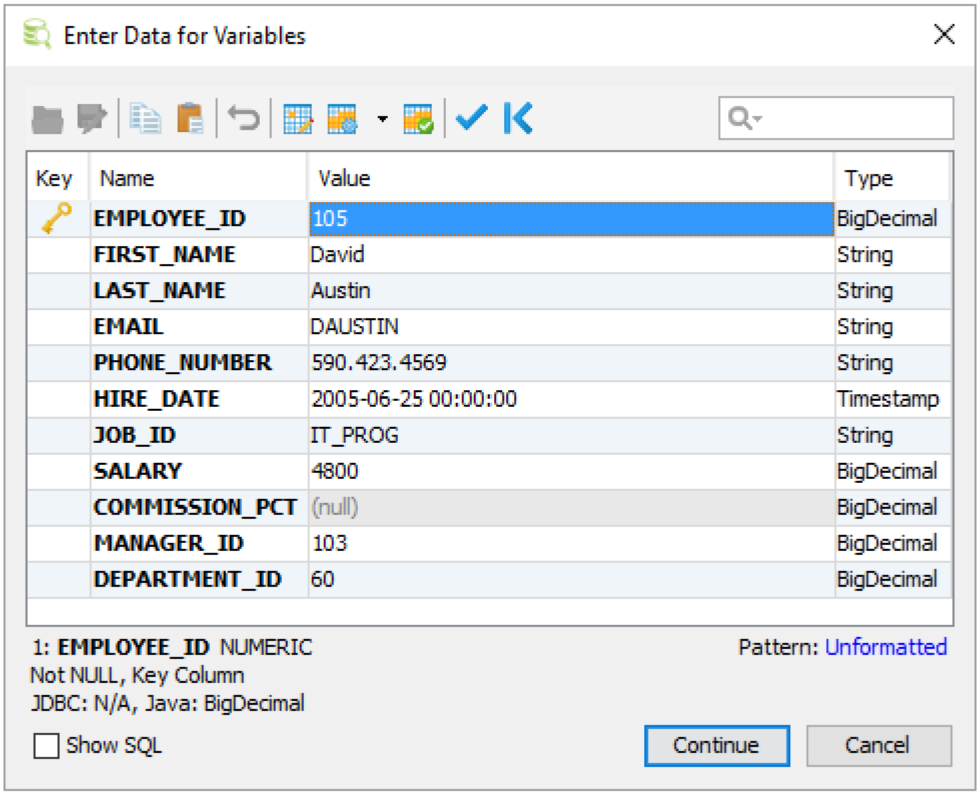
The **SQL Preview** area shows the statement with all variables replaced with the values.

Here is an example of a more complex use of variables utilizing default value, data type and options:

```
INSERT
INTO
```

```
    EMPLOYEES
    (
        EMPLOYEE_ID,
        FIRST_NAME,
        LAST_NAME,
        EMAIL,
        PHONE_NUMBER,
        HIRE_DATE,
        JOB_ID,
        SALARY,
        COMMISSION_PCT,
        MANAGER_ID,
        DEPARTMENT_ID
    )
    VALUES
    (
        ${EMPLOYEE_ID||105||BigDecimal||pk ds=7 dt=NUMERIC}$,
        ${FIRST_NAME||David||String||nullable ds=20 dt=VARCHAR}$,
        ${LAST_NAME||Austin||String||ds=25 dt=VARCHAR}$,
        ${EMAIL||DAUSTIN||String||ds=25 dt=VARCHAR}$,
        ${PHONE_NUMBER||590.423.4569||String||nullable ds=20 dt=VARCHAR}$,
        ${HIRE_DATE||2005-06-25 00:00:00||Timestamp||ds=7 dt=TIMESTAMP}$,
        ${JOB_ID||IT_PROG||String||ds=10 dt=VARCHAR}$,
        ${SALARY||4800||BigDecimal||nullable ds=10 dt=NUMERIC}$,
        ${COMMISSION_PCT||(null)||BigDecimal||nullable ds=4 dt=NUMERIC}$,
        ${MANAGER_ID||103||BigDecimal||nullable ds=7 dt=NUMERIC}$,
        ${DEPARTMENT_ID||60||BigDecimal||nullable ds=5 dt=NUMERIC}$
    )
```

This example use the full capabilities of variables. This example is generated by the **Script to SQL Commander->INSERT COPY INTO TABLE** right click menu choice in the **Data** tab grid. By default it generates variables representing the actual values and the characteristics of the columns.



Enter Data for Variables

Key	Name	Value	Type
🔑	EMPLOYEE_ID	105	BigDecimal
	FIRST_NAME	David	String
	LAST_NAME	Austin	String
	EMAIL	DAUSTIN	String
	PHONE_NUMBER	590.423.4569	String
	HIRE_DATE	2005-06-25 00:00:00	Timestamp
	JOB_ID	IT_PROG	String
	SALARY	4800	BigDecimal
	COMMISSION_PCT	(null)	BigDecimal
	MANAGER_ID	103	BigDecimal
	DEPARTMENT_ID	60	BigDecimal

1: EMPLOYEE_ID NUMERIC
Not NULL, Key Column
JDBC: N/A, Java: BigDecimal

Pattern: Unformatted

□ Show SQL

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

When executing an SQL statement that consist of variables, DbVisualizer replaces each variable with either the value as a literal or as a parameter marker. Using parameter markers to pass data with a statement is more reliable than literals. DbVisualizer will automatically generate a parameter marker if the variable has the data type 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 value as a literal in the final SQL:

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

Variables in DbVisualizer may be used anywhere in a statement as long as there is no data type specified.

Changing the Delimiter Characters

You can change which identifiers should be used as the prefix, suffix and part delimiter in a variable expression in **Tools->Tool Properties**, in the **General / Variables** category.

9.24.2 Using Parameter Markers

Parameter markers (also referred as **bind/host variables** or **place holders**) are commonly used in database applications where the SQL is composed of static text combined with values represented as markers instead of actual values. These markers are processed during the preparation of the SQL statement and values are then bound with the markers. Each database has its recommendations for how and when to use parameter markers so this is not further discussed here.

DbVisualizer supports the most common syntaxes for parameter markers to comply with the supported databases. Parameter markers are categorized as either **named** or **unnamed** markers. The following sections explains their respective syntaxes.

> It is not possible to mix DbVisualizer variables and parameter markers, or named and unnamed parameter markers, in the same script. If you do, you will only be prompted for values for one type and the execution will fail.

Named Parameter Markers

Named Parameter Markers	
&name **:name** :{name}: 'name'	These syntaxes are supported natively by a few databases. This format allows only a name for the parameter and no other settings, such as type or default value. The parameter name is the name DbVisualizer shows in the prompt window. Named parameter values are bound at runtime with the markers in the SQL. Some JDBC drivers/databases requires that the proper data type is set while some are more relaxed. For named (and unnamed) parameter markers, you may chose data type in the prompt window. Named parameter markers should only be used in contexts supported by the actual database, usually for column values. For example, as opposed to a DbVisualizer variable, a parameter marker cannot be used for a table or column name. The only difference between **&name**, **:name**, **:{name}** and **:'name'** is that the latter two, **:{name }** and **:'name'**, allow white spaces in the name. **Example** `insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)`

Named Parameter Markers

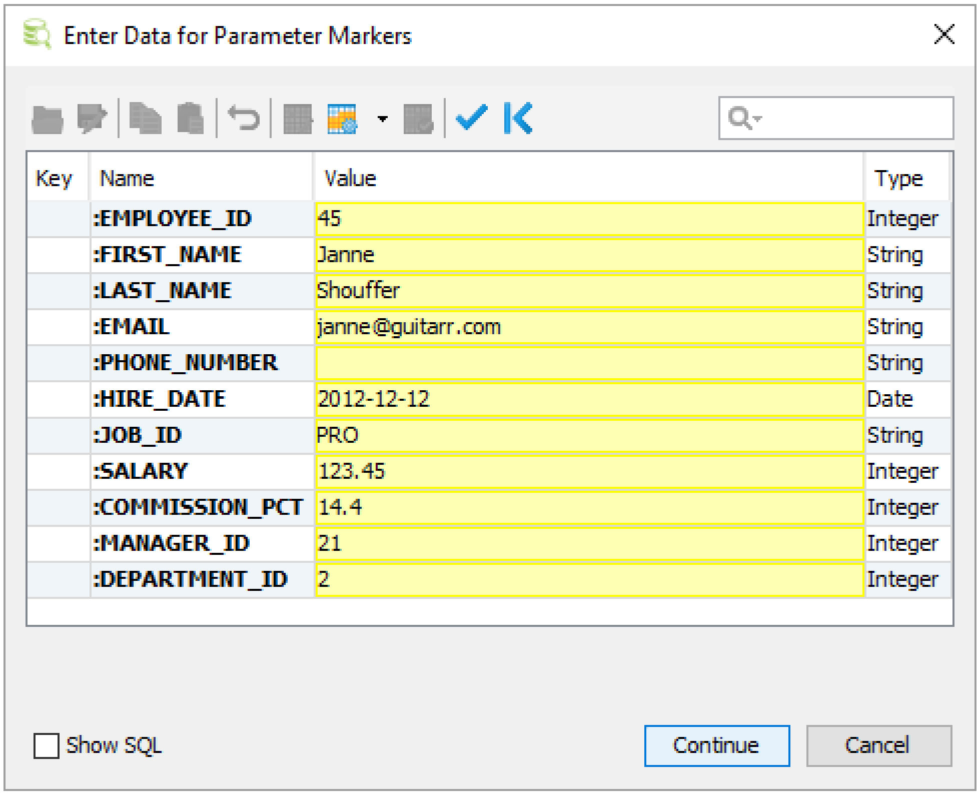
```
values (null, &FirstName, &LastName, &Address, &Age);
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, :FirstName, :LastName, :Address, :Age);
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, :{FirstName}, :{LastName}, :{Address}, :{Age});
```

Read more about named parameter markers (see page 235).

The following is a sample SQL executed in the SQL Commander:

```
INSERT INTO EMPLOYEES
  (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE,
   JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID)
VALUES
  (:EMPLOYEE_ID, :FIRST_NAME, :LAST_NAME, :EMAIL, :PHONE_NUMBER,
   :HIRE_DATE, :JOB_ID, :SALARY, :COMMISSION_PCT, :MANAGER_ID, :DEPARTMENT_ID);
```

The prompt window will show the markers with their respective names:



Key	Name	Value	Type
	:EMPLOYEE_ID	45	Integer
	:FIRST_NAME	Janne	String
	:LAST_NAME	Shouffer	String
	:EMAIL	janne@guitarr.com	String
	:PHONE_NUMBER		String
	:HIRE_DATE	2012-12-12	Date
	:JOB_ID	PRO	String
	:SALARY	123.45	Integer
	:COMMISSION_PCT	14.4	Integer
	:MANAGER_ID	21	Integer
	:DEPARTMENT_ID	2	Integer

For parameter marker processing to work in the SQL Commander, make sure the **SQL Commander-> Parameterized SQL** main menu option is checked.

To apply the values, close the window and continue with the execution, use key binding **Ctrl+Enter** (**Command+Enter** on macOS).

Unnamed Parameter Markers

Unnamed Parameter Markers

?

The question marker symbol is probably the most supported parameter marker among the supported databases. It is also the most unintuitive marker since the user has to remember the order of question marks and the corresponding values.

Since there is no name associated with it, DbVisualizer shows these as **Parameter 1**, **Parameter 2** and so on in the prompt window.

There is no technical difference between how unnamed and named parameter markers are handled internally in DbVisualizer or when processed by the database. All are bound with a prepared SQL statement.

Example

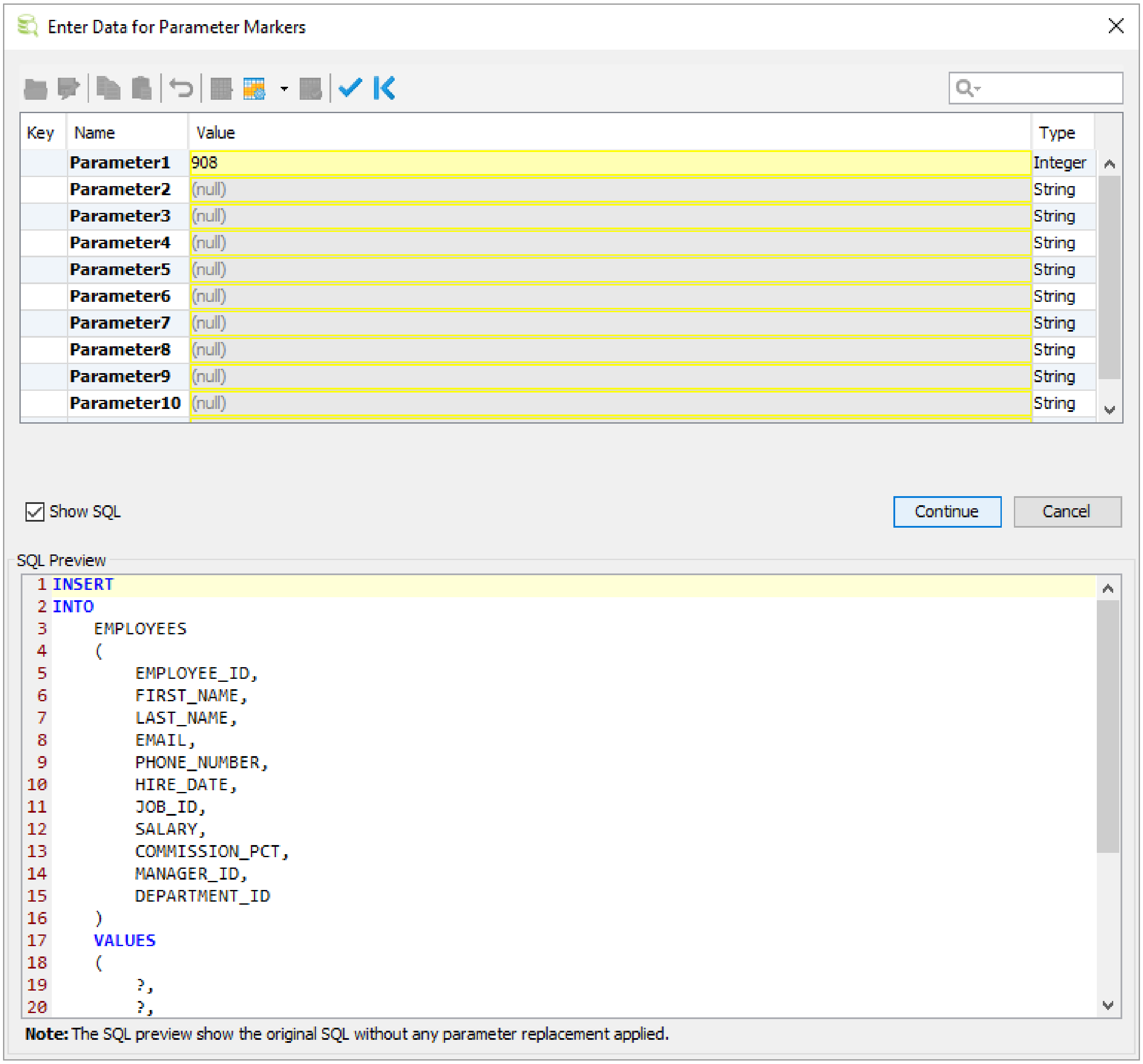
```
insert into EMPLOYEE (ID, FIRST_NAME, LAST_NAME, ADDRESS, AGE)
values (null, ?, ?, ?, ?)
```

Use named in favor of unnamed parameter markers if there is support in the target database based on the easier reading of named markers. Read more about unnamed parameter markers (see page 235).

This is the same SQL as used in the Named Parameter Marker section but here question marks are used as markers:

```
INSERT INTO EMPLOYEES
  (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE,
   JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID)
VALUES
  (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

Since we're using the unnamed marker, **?**, the name of each parameter is displayed as **Parameter 1**, **Parameter 2** and so on:



Key	Name	Value	Type
	Parameter1	908	Integer
	Parameter2	(null)	String
	Parameter3	(null)	String
	Parameter4	(null)	String
	Parameter5	(null)	String
	Parameter6	(null)	String
	Parameter7	(null)	String
	Parameter8	(null)	String
	Parameter9	(null)	String
	Parameter10	(null)	String

```
INSERT
INTO
    EMPLOYEES
    (
        EMPLOYEE_ID,
        FIRST_NAME,
        LAST_NAME,
        EMAIL,
        PHONE_NUMBER,
        HIRE_DATE,
        JOB_ID,
        SALARY,
        COMMISSION_PCT,
        MANAGER_ID,
        DEPARTMENT_ID
    )
    VALUES
    (
        ?,
        ?,
```

Note: The SQL preview show the original SQL without any parameter replacement applied.

Due to the use of unnamed markers it is not very intuitive what parameter correspond to which part in the statement. The **SQL Preview** may be handy to get an idea. The **Type** field is automatically adjusted based on what data is entered for a value. The data type may be manually set by left-click on the type and choose another type from the drop-down.

To apply the values, close the window and continue with the execution, use key binding **Ctrl+Enter** (**Command+Enter** on macOS).

Get Parameter Types via JDBC

The processing of named and unnamed parameter markers is managed by DbVisualizer. By default there is no data type detection of the target columns identified by the markers and DbVisualizer will initially present these as **String** in the prompt window. When changing the value for a parameter in the prompt window, a data analyzer is triggered which will automatically detect the type and update the **Type** field accordingly.

Some drivers (far from all) have the capability to detect the real data type for the referenced columns in the SQL statement. To enable this processing, select the **Get Parameter Types via JDBC** action in the **SQL Commander** menu. DbVisualizer will then show the correct types in the prompt window.

> ⊖ Having **Get Parameter Types via JDBC** enabled while executing may decrease performance substantially as each SQL statement in the script is then pre-processed with the database before the prompt window is displayed.

Database Support/Driver Support

The support for parameter markers may differ between databases. Please consult the documentation for the database to see what syntax it supports.

In some situations the database and DbVisualizer support for named parameters might be incompatible. An example is when using the same parameter name in multiple places in the SQL. When preparing a statement towards such a database, the database may report that the parameter is only used once. In these cases, DBVisualizer will trust the driver and revert to generating the names visible in the form as Parameter 1 , Parameter 2 and so on.

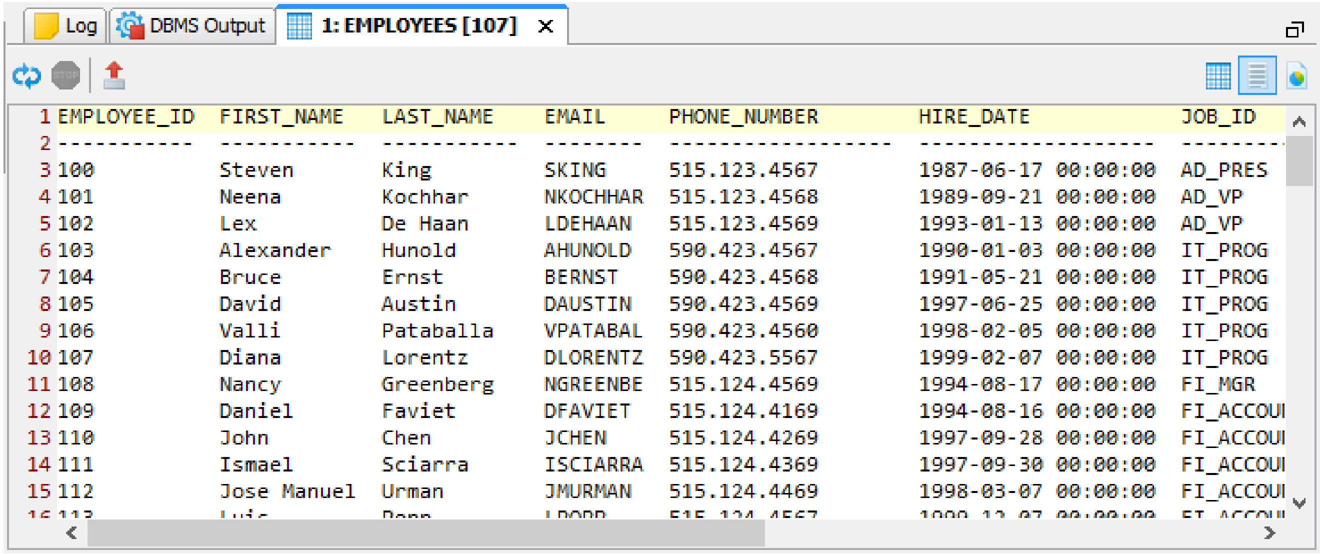
10 Working with Result Sets

You can view result sets in different ways, edit simple result sets, and export or compare them.

10.1 Viewing a Result Set

You can view a result set as a grid, as text or as a chart. Which format to use by default can be specified in the Tool Properties dialog, in the **SQL Commander/Result Sets** category under the General tab. Here you can also specify other things, like if empty result sets should be shown at all, if the SQL that produced the result set should be shown as a tab header tooltip.

Use the buttons to the right in the grid toolbar to select the format. This is an example of the text format.



```
EMPLOYEE_ID  FIRST_NAME   LAST_NAME    EMAIL     PHONE_NUMBER          HIRE_DATE            JOB_ID
-----------  -----------  -----------  --------  --------------------  -------------------  --------
100          Steven       King         SKING     515.123.4567          1987-06-17 00:00:00  AD_PRES
101          Neena        Kochhar      NKOCHHAR  515.123.4568          1989-09-21 00:00:00  AD_VP
102          Lex          De Haan      LDEHAAN   515.123.4569          1993-01-13 00:00:00  AD_VP
103          Alexander    Hunold       AHUNOLD   590.423.4567          1990-01-03 00:00:00  IT_PROG
104          Bruce        Ernst        BERNST    590.423.4568          1991-05-21 00:00:00  IT_PROG
105          David        Austin       DAUSTIN   590.423.4569          1997-06-25 00:00:00  IT_PROG
106          Valli        Pataballa    VPATABAL  590.423.4560          1998-02-05 00:00:00  IT_PROG
107          Diana        Lorentz      DLORENTZ  590.423.5567          1999-02-07 00:00:00  IT_PROG
108          Nancy        Greenberg    NGREENBE  515.124.4569          1994-08-17 00:00:00  FI_MGR
109          Daniel       Faviet       DFAVIET   515.124.4169          1994-08-16 00:00:00  FI_ACCOU
110          John         Chen         JCHEN     515.124.4269          1997-09-28 00:00:00  FI_ACCOU
111          Ismael       Sciarra      ISCIARRA  515.124.4369          1997-09-30 00:00:00  FI_ACCOU
112          Jose Manuel  Urman        JMURMAN   515.124.4469          1998-03-07 00:00:00  FI_ACCOU
113          Luis         Popp         LPOPP     515.124.4567          1999-12-07 00:00:00  FI_ACCOU
```

10.1.1 Viewing as a Grid

When you view the result set as a grid, you have access to the same features as when viewing table data (see page 82).

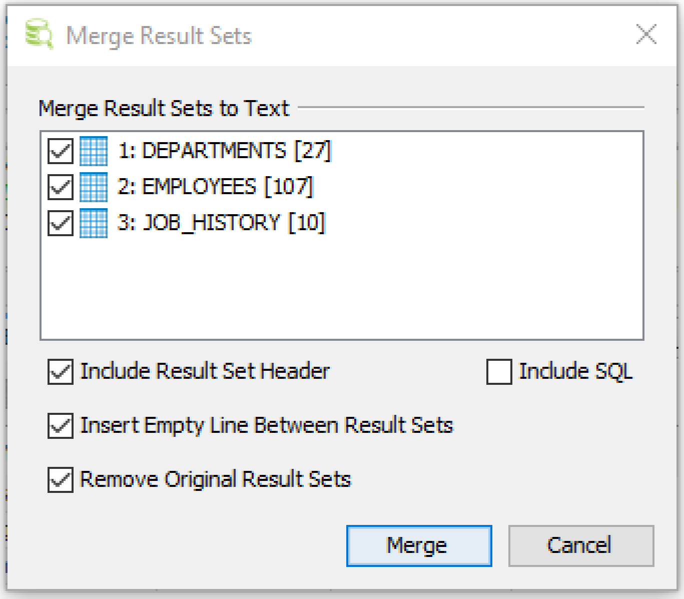
10.1.2 Viewing as Text

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

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.

If you want to combine the text view of a number of result sets into one, select **Merge Result Sets** from the result set tab right-click menu. A dialog lets you select the result sets to merge:



Merge Result Sets

Merge Result Sets to Text

☑ 1: DEPARTMENTS [27]
☑ 2: EMPLOYEES [107]
☑ 3: JOB_HISTORY [10]

☑ Include Result Set Header ☐ Include SQL

☑ Insert Empty Line Between Result Sets

☑ Remove Original Result Sets

Auto merging

Using the **Merge** drop-down menu in the SQL Commander toolbar, you can enable **Auto Merge after Execution**. You can also select **Configure Merging** from the menu to adjust various merge options.

10.1.3 Viewing as a Graph

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

To view the result set as a chart, use the rightmost button in the grid toolbar. Please see the Working with Charts (see page 244) page for how you can arrange the chart.

10.2 Editing a Result Set

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

A result set from a query that fulfills these requirements is editable:

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

A result set like this can be edited in the same way as you can edit table data (see page 94).

⚠ If all of the above requirements are fulfilled but the edit controls are still not shown, please try qualifying the table name with the schema and/or database name.

If you want results to always be read-only, you can enable the **Make Result Sets Read-Only** setting in the Tool Properties dialog, in the **SQL Commander/Result Sets** category under the General tab.

10.3 Exporting a Result Set

You can export a result set as described in Exporting a Grid (see page 253) or using the `@export` client side command (see page 223).

10.4 Comparing Result Sets

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

You can compare a result set grid to tables and/or other result set grids.

To compare the grid data to the data of a table or a another result set:

1. Open the **Data** tab for another table or execute an SQL query to open a result set tab,
2. Select **Compare** from the right-click menu in one of the tabs to compare their grid content (see page 262)
.

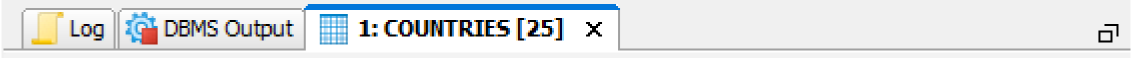
10.5 Pinning Result Sets

Existing Result Set tabs are removed when you execute a script again. If you want to save a Result Set tab between executions, you can "pin" it using the **Pin Tab** right-click menu choice for the tab header, or by simply clicking on the tab icon. There is also a **Pin All** choice if you have multiple tabs you want to pin, and **Unpin All** to make them all be replaced at the next execution.

Whether tabs should be pinned by default can be controlled in the Tool Properties dialog, in the **SQL Commander/Result Sets** category under the General tab.

10.6 Show Result Sets in a Separate Window

Results Sets and the Log tabs are located just under editor in a SQL Commander tab. Sometimes you may need to get the full screen height available for the editor and detach the result area tabs in its own window. To accomplish this click the symbol at the right-most position in the tab row.



Clicking the symbol while result area tabs are detached will bring them back into the original location.

11 Working with Charts

Only in DbVisualizer Pro

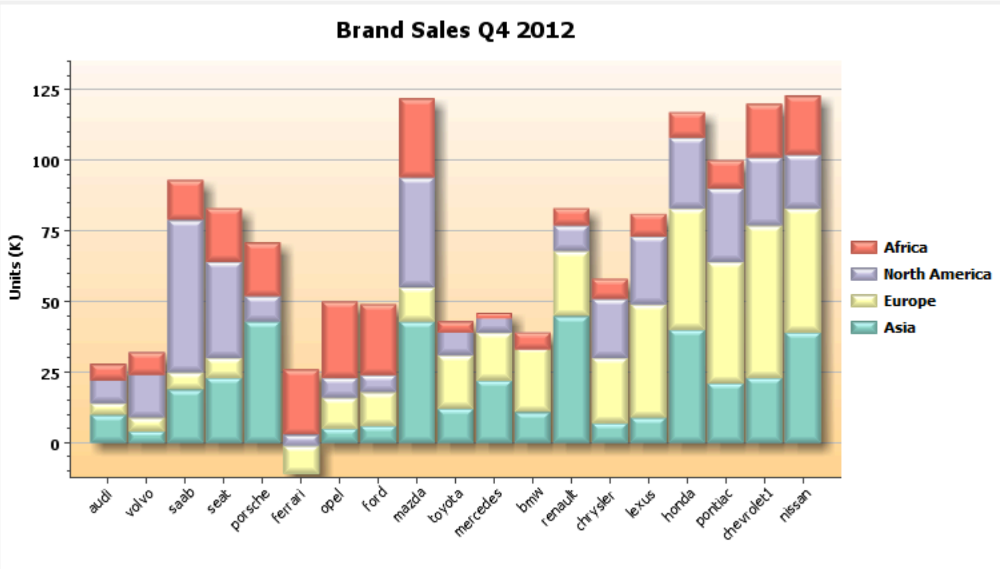
This feature is only available in the DbVisualizer Pro edition.

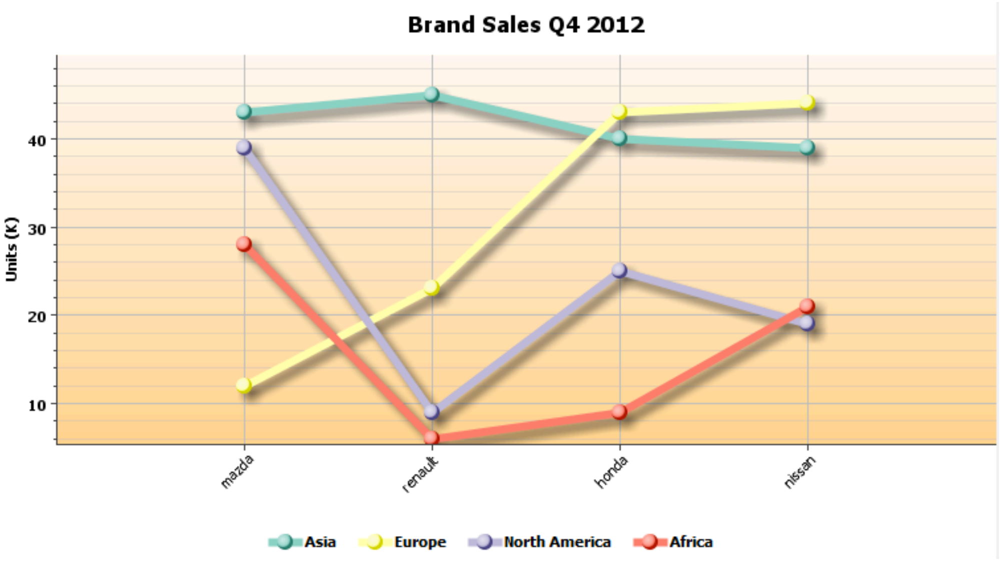
Result sets in the SQL Commander and in the Monitor tools can be viewed as charts.

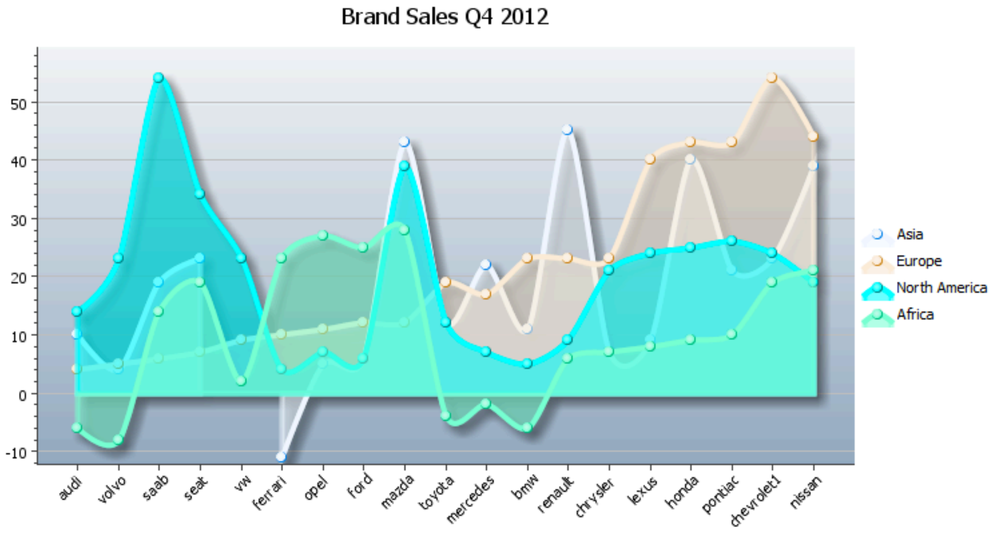
- Charting a Result Set (see page 246)
  - Selecting Category Column (see page 247)
  - Selecting Series (see page 248)
  - Chart Type (see page 249)
- Chart Preferences (see page 249)
  - Appearance Preferences (see page 249)
  - Series Preferences (see page 251)
- Zooming (see page 251)
- Export (see page 251)

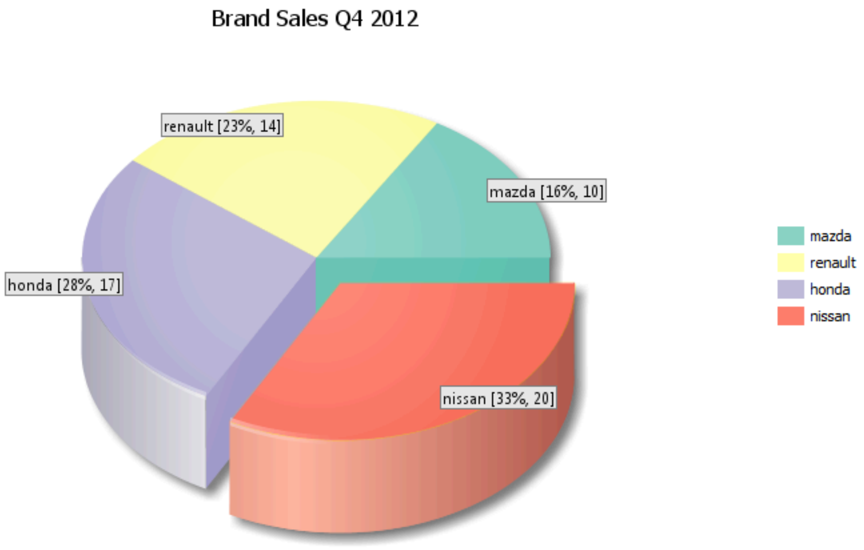
The chart support in DbVisualizer presents data from any result set in a configurable chart displayed in a line, bar, area or pie style. It offers much of the charting support you find in MS Excel and other specialized charting tools. Charts may be exported as an image to file, printed and copied to system clipboard for easy sharing with other tools. Charts are configured and viewed in the SQL Commander (see page 240) and in the Monitor (see page 267) tool which is really powerful, delivering real time charts of many result sets simultaneously.

Here are some sample charts:




Brand Sales Q4 2012
Units (K)
40
30
20
10
mazda
renault
honda
nissan
Asia
Europe
North America
Africa

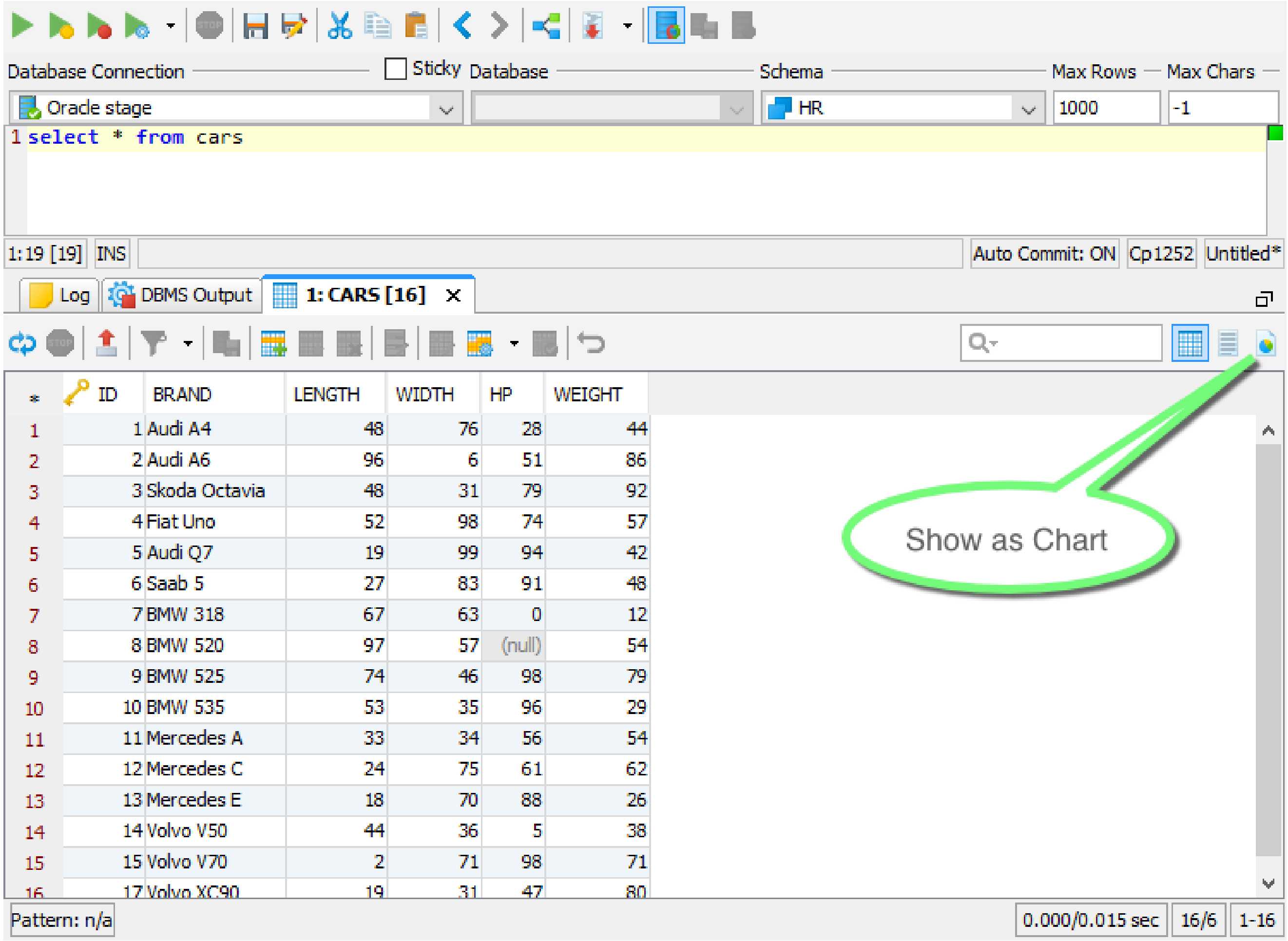

Brand Sales Q4 2012
50
40
30
20
10
0
-10
audi
volvo
saab
seat
vw
ferrari
opel
ford
mazda
toyota
mercedes
bmw
renault
chrysler
lexus
honda
pontiac
chevrolet1
nissan
Asia
Europe
North America
Africa


Brand Sales Q4 2012
renault [23%, 14]
mazda [16%, 10]
honda [28%, 17]
nissan [33%, 20]
mazda
renault
honda
nissan

11.1 Charting a Result Set

The basic setup of a chart is really easy. Just select one or more columns that should appear as series in the chart and what column to use as the category (X-axis). Further refinement of the chart can be made in the chart preferences window.

The normal output view of a result set in the SQL Commander or Monitor window is in a grid style as shown in the following screenshot. To activate the chart view click the rightmost button in the result tab toolbar:



Database Connection: Oracle stage — □ Sticky — Database: — Schema: HR — Max Rows: 1000 — Max Chars: -1

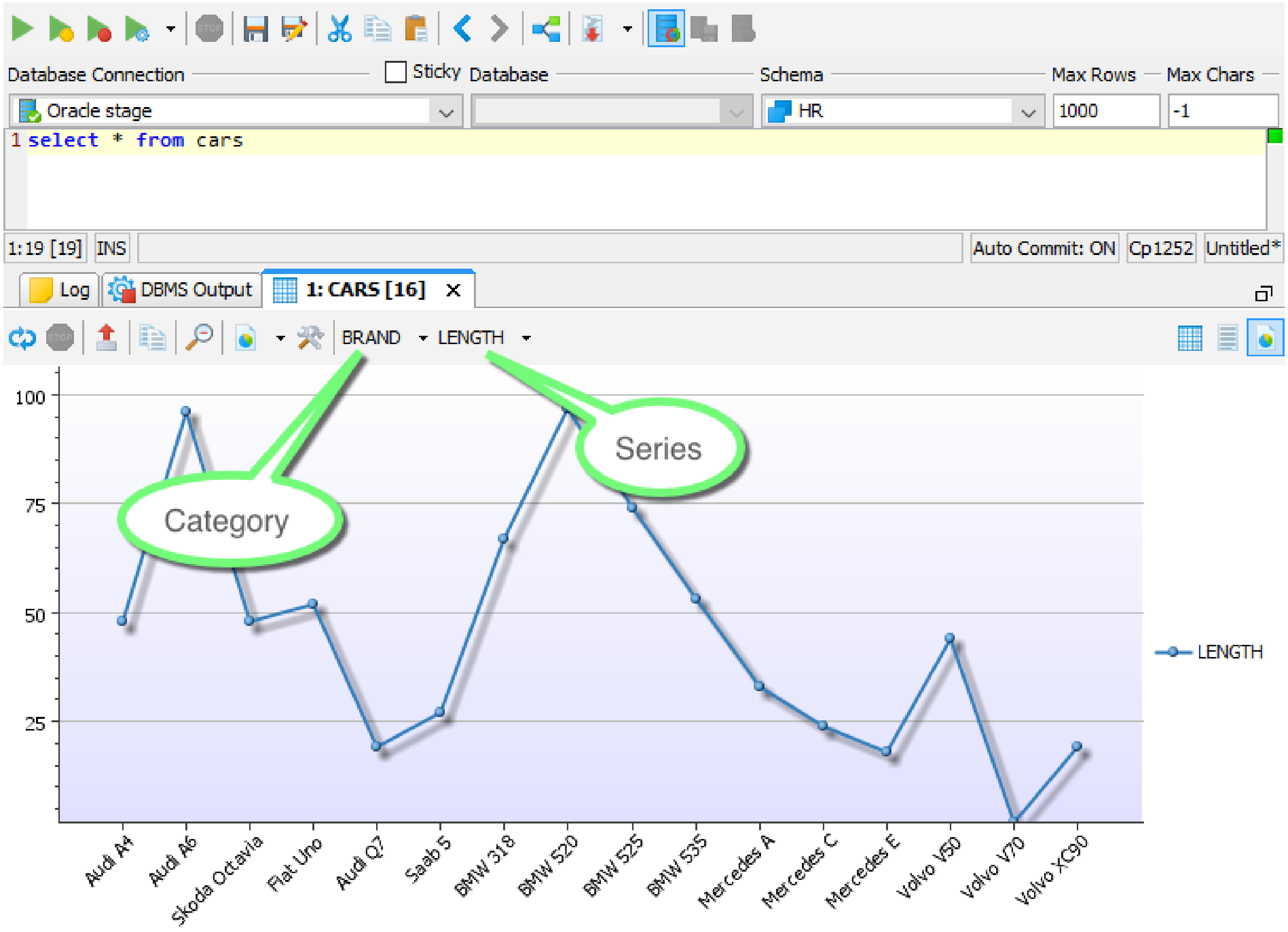
```
select * from cars
```

1:19 [19] INS — Auto Commit: ON — Cp1252 — Untitled*

	ID	BRAND	LENGTH	WIDTH	HP	WEIGHT
1	1	Audi A4	48	76	28	44
2	2	Audi A6	96	6	51	86
3	3	Skoda Octavia	48	31	79	92
4	4	Fiat Uno	52	98	74	57
5	5	Audi Q7	19	99	94	42
6	6	Saab 5	27	83	91	48
7	7	BMW 318	67	63	0	12
8	8	BMW 520	97	57	(null)	54
9	9	BMW 525	74	46	98	79
10	10	BMW 535	53	35	96	29
11	11	Mercedes A	33	34	56	54
12	12	Mercedes C	24	75	61	62
13	13	Mercedes E	18	70	88	26
14	14	Volvo V50	44	36	5	38
15	15	Volvo V70	2	71	98	71
16	17	Volvo XC90	19	31	47	80

Pattern: n/a — 0.000/0.015 sec — 16/6 — 1-16

When switching to the chart view DbVisualizer automatically pick the first date or text column as category and the first numeric column as serie. In the following example it is the **datelabel** and **id** columns for this specific result set.



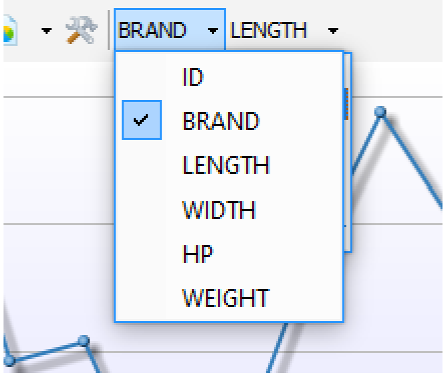
Database Connection: Oracle stage
☐ Sticky
Database:
Schema: HR
Max Rows: 1000
Max Chars: -1

```
select * from cars
```

1:19 [19] INS Auto Commit: ON Cp1252 Untitled*

11.1.1 Selecting Category Column

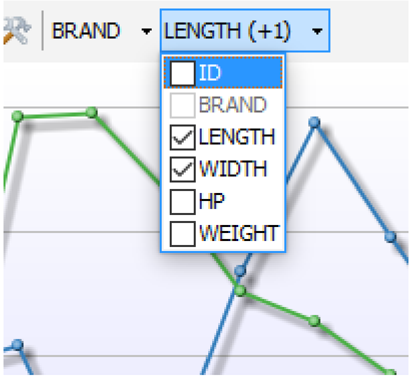
To change category column click the category drop-down and pick what column to use.



Let the mouse pointer stay on a column name for a second and tip will show what data type it is.

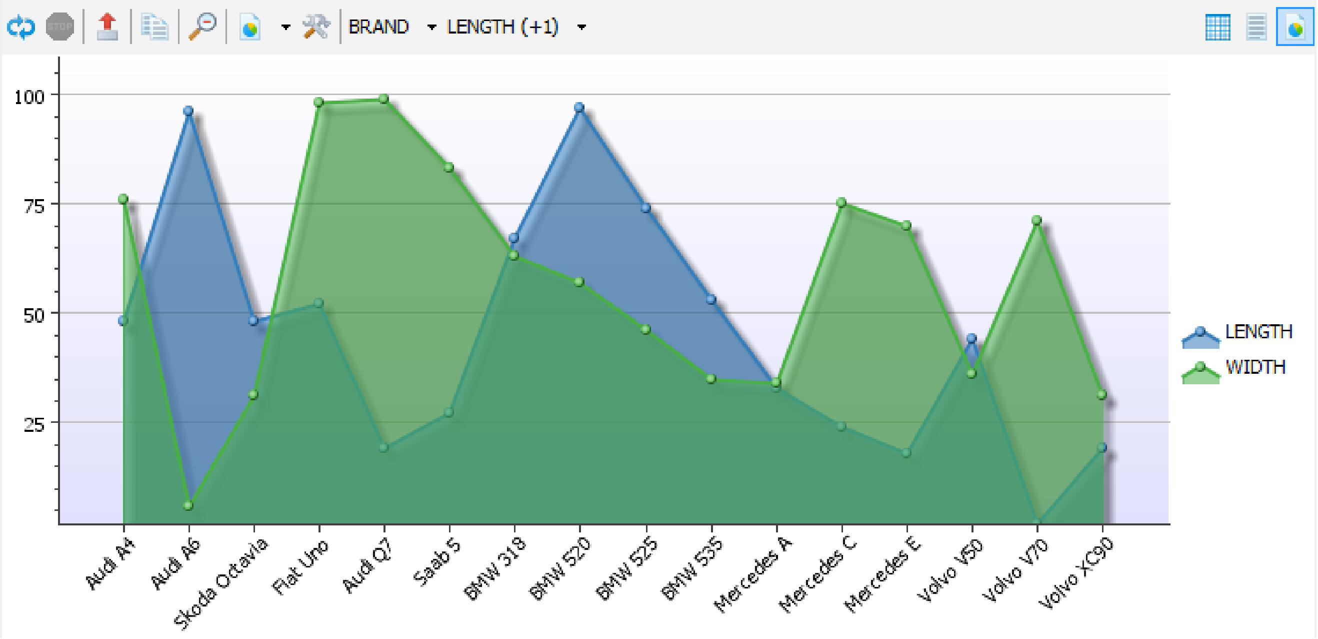
11.1.2 Selecting Series

Click the serie button to change what series to display in the chart. This drop-down stay on screen while de-selecting and selecting series and the changes are directly propagated in the chart. To close the list either press the **ESC** button or click outside the list. If only one serie is selected then its name is listed in the button label. If additional series are selected then the number of selected series are listed in parentheses.



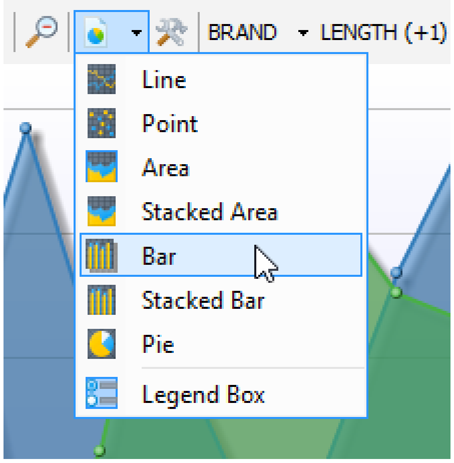
Tip: Press the ALT key while selecting a serie and all currently selected series will be de-selected.

This is the chart after applying category and 2 series:



11.1.3 Chart Type

A chart can be displayed as Line, Point, Area, Stacked Area, Bar, Stacked Bar and the Pie type. Select what type to use in the Chart Type right-click menu or in the chart toolbar:



For the Pie chart, only one Serie can be selected.

11.2 Chart Preferences

The chart preferences are used to customize the appearance of the chart such as titles, colors, legend position, etc. It can also be used to set alternative names for the series in the chart. All appearance settings are automatically re-used when running subsequent queries in the SQL Commander during the same DbVisualizer session. If you save a query as a bookmark script then all appearance settings are saved with the chart. Loading the script at a later time will also load the chart settings.

11.2.1 Appearance Preferences

Use the controls in the **General** tab to customize the layout and style of the chart.

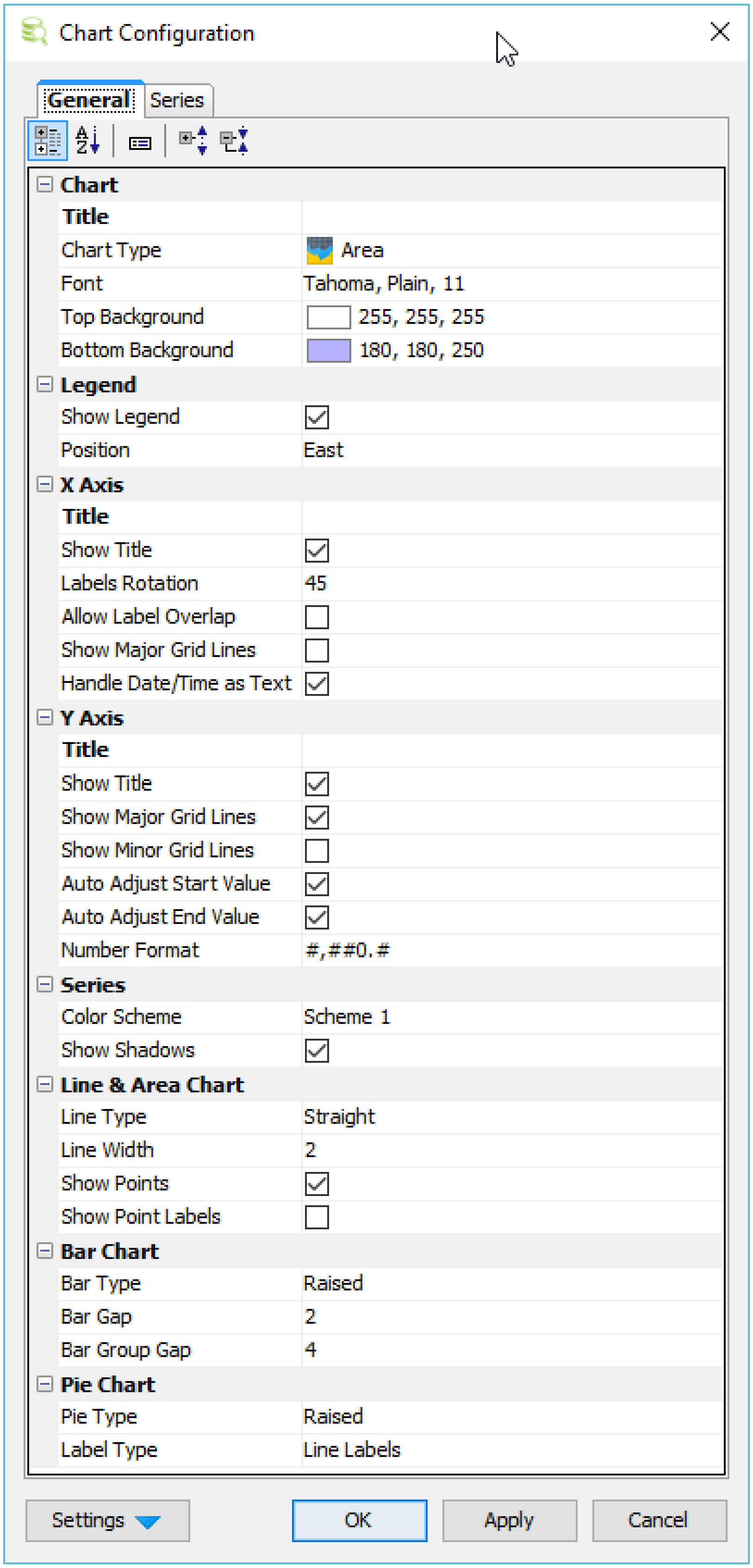
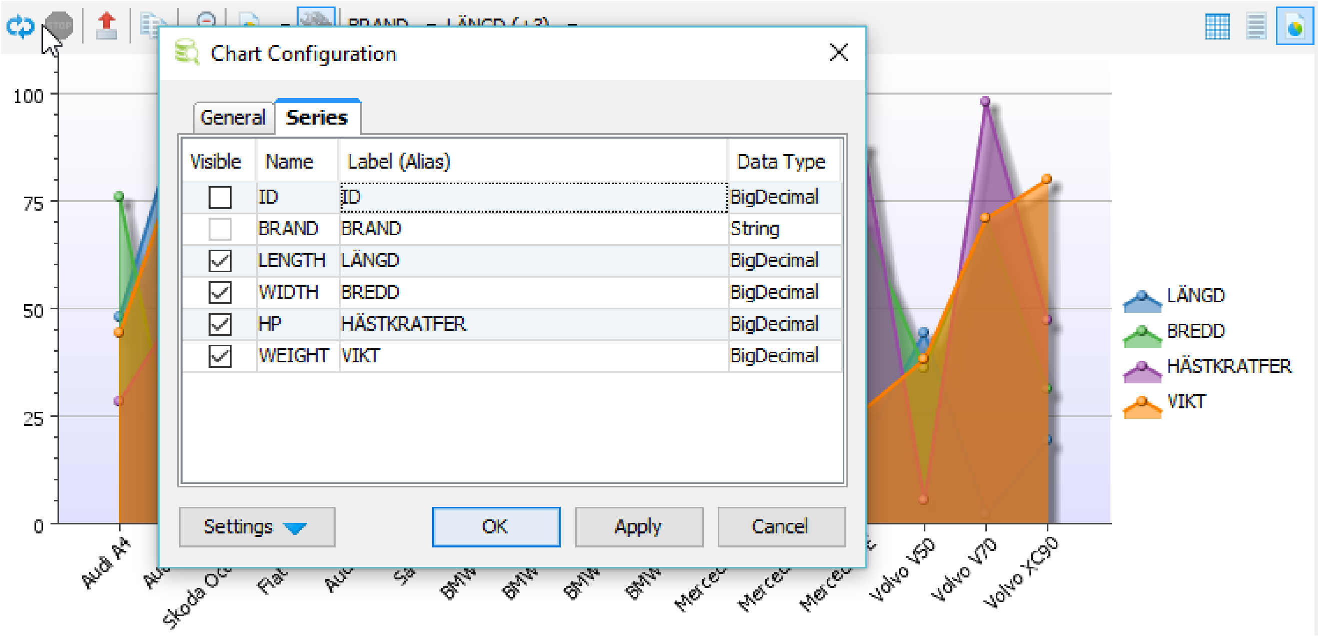


Chart	
Title	
Chart Type	Area
Font	Tahoma, Plain, 11
Top Background	□ 255, 255, 255
Bottom Background	▨ 180, 180, 250
Legend	
Show Legend	☑
Position	East
X Axis	
Title	
Show Title	☑
Labels Rotation	45
Allow Label Overlap	☐
Show Major Grid Lines	☐
Handle Date/Time as Text	☑
Y Axis	
Title	
Show Title	☑
Show Major Grid Lines	☑
Show Minor Grid Lines	☐
Auto Adjust Start Value	☑
Auto Adjust End Value	☑
Number Format	#,##0.#
Series	
Color Scheme	Scheme 1
Show Shadows	☑
Line & Area Chart	
Line Type	Straight
Line Width	2
Show Points	☑
Show Point Labels	☐
Bar Chart	
Bar Type	Raised
Bar Gap	2
Bar Group Gap	4
Pie Chart	
Pie Type	Raised
Label Type	Line Labels

Information about the selected setting in the list is displayed at the bottom of the window.

11.2.2 Series Preferences

The default serie name is the column name in the result set. In the **Series** tab you can set an alternative label name and also control what series should be visible.



Visible	Name	Label (Alias)	Data Type
☐	ID	ID	BigDecimal
☐	BRAND	BRAND	String
☑	LENGTH	LÄNGD	BigDecimal
☑	WIDTH	BREDD	BigDecimal
☑	HP	HÄSTKRATFER	BigDecimal
☑	WEIGHT	VIKT	BigDecimal

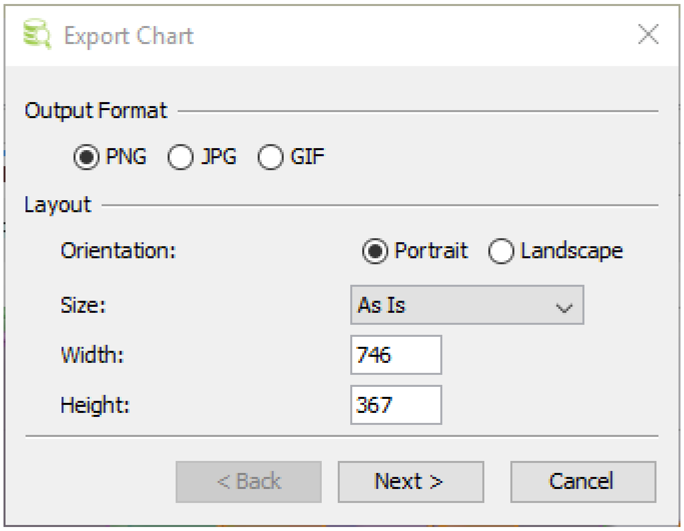
Changes in the Series tab are propagated directly in the chart.

11.3 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 unzoom one level, click the Zoom Out button.

11.4 Export

Charts can be exported in PNG, GIF or JPG formats.


Export Chart

Output Format

● PNG ○ JPG ○ GIF

Layout

Orientation: ● Portrait ○ Landscape

Size: As Is

Width: 746

Height: 367

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.

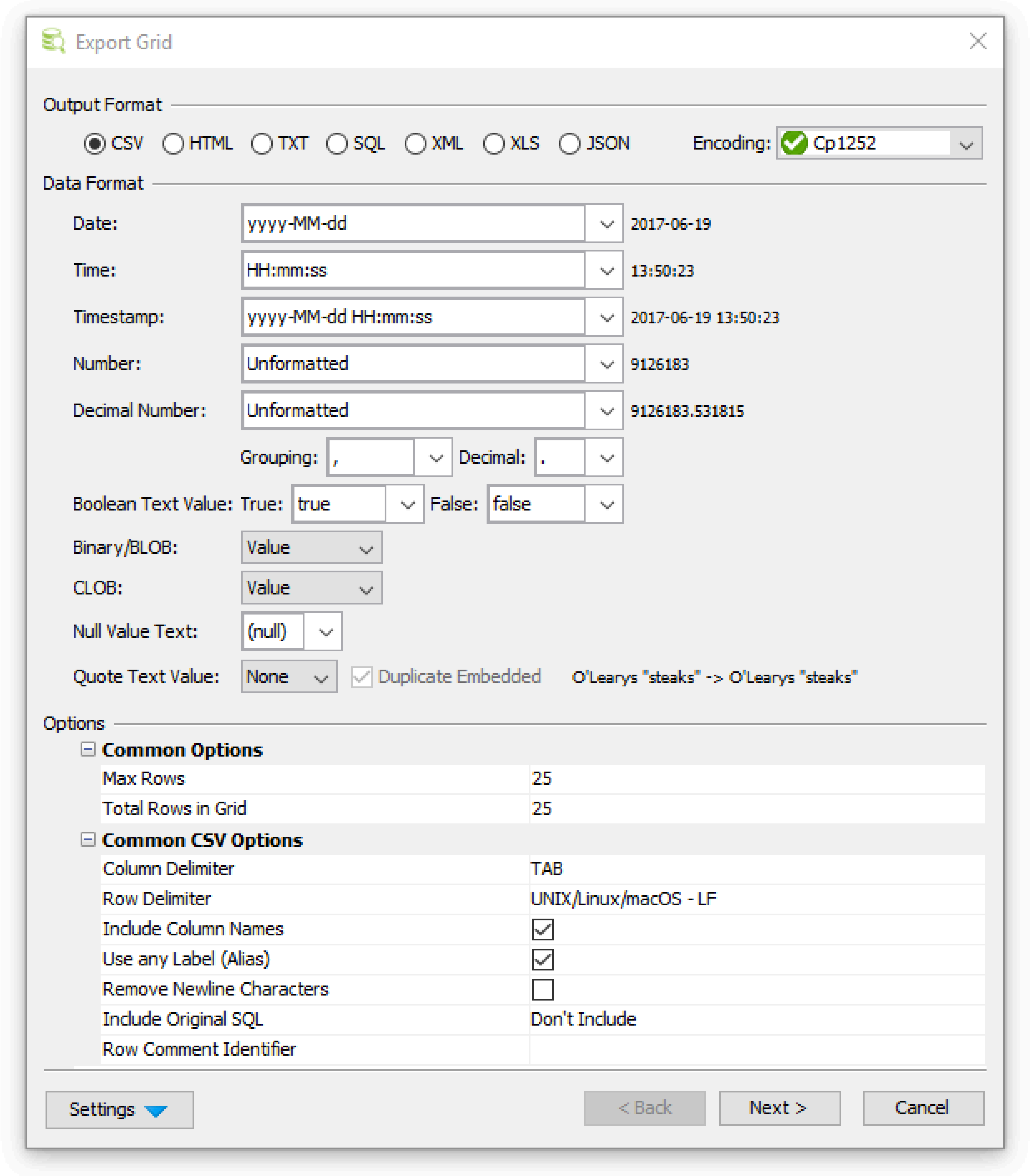
12 Exporting a Grid

You can export any grid using the Export Wizard, e.g. a result set grid, a grid showing tables in a schema, or the Data tab for a table or view.

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.

12.1 Settings page

The first wizard page is the **Settings** page, containing general properties for how the exported data should be formatted.



Output Format

● CSV ○ HTML ○ TXT ○ SQL ○ XML ○ XLS ○ JSON   Encoding: Cp1252

Data Format

Date:	yyyy-MM-dd	2017-06-19
Time:	HH:mm:ss	13:50:23
Timestamp:	yyyy-MM-dd HH:mm:ss	2017-06-19 13:50:23
Number:	Unformatted	9126183
Decimal Number:	Unformatted	9126183.531815
	Grouping: , Decimal: .	
Boolean Text Value:	True: true False: false	
Binary/BLOB:	Value	
CLOB:	Value	
Null Value Text:	(null)	
Quote Text Value:	None ☑ Duplicate Embedded	O'Learys "steaks" -> O'Learys "steaks"

Options

Common Options	
Max Rows	25
Total Rows in Grid	25
Common CSV Options	
Column Delimiter	TAB
Row Delimiter	UNIX/Linux/macOS - LF
Include Column Names	☑
Use any Label (Alias)	☑
Remove Newline Characters	□
Include Original SQL	Don't Include
Row Comment Identifier	

Select an output format, file encoding (it is also used to set the encoding in the HTML and XML headers, if you select one of those formats), and how to quote text data.

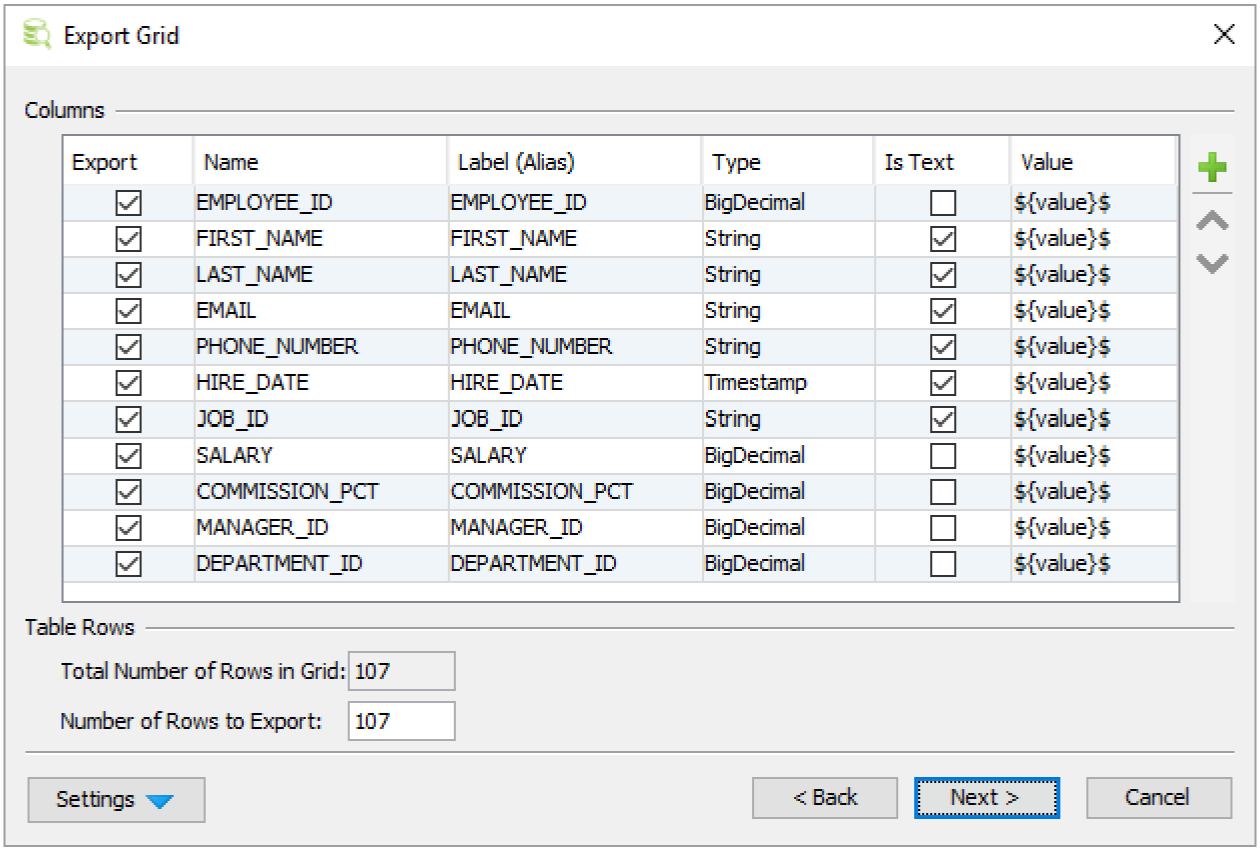
Only in DbVisualizer Pro

With the DbVisualizer Free edition, only the CSV and HTML formats are supported.

The **Options** section is used to define settings that are specific for the selected output format, for instance the column and row delimiters for the CSV format, or the Excel or Excel 2007 format for XSL.

12.2 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.


Export Grid

Columns

Export	Name	Label (Alias)	Type	Is Text	Value
☑	EMPLOYEE_ID	EMPLOYEE_ID	BigDecimal	□	${value}$
☑	FIRST_NAME	FIRST_NAME	String	☑	${value}$
☑	LAST_NAME	LAST_NAME	String	☑	${value}$
☑	EMAIL	EMAIL	String	☑	${value}$
☑	PHONE_NUMBER	PHONE_NUMBER	String	☑	${value}$
☑	HIRE_DATE	HIRE_DATE	Timestamp	☑	${value}$
☑	JOB_ID	JOB_ID	String	☑	${value}$
☑	SALARY	SALARY	BigDecimal	□	${value}$
☑	COMMISSION_PCT	COMMISSION_PCT	BigDecimal	□	${value}$
☑	MANAGER_ID	MANAGER_ID	BigDecimal	□	${value}$
☑	DEPARTMENT_ID	DEPARTMENT_ID	BigDecimal	□	${value}$

Table Rows

Total Number of Rows in Grid: 107

Number of Rows to Export: 107

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.

The columns in this page's grid can be used like this.

Column	Descriptions
Export	Defines whether the column will be exported or not. Uncheck it to ignore the column in the exported output.

Column	Descriptions
Name	The name of the column. This is used if exporting in HTML, XML, XLS, JSON or SQL format. Column headers are optional in the CSV output format.
Label (Alias)	When you export a result set grid for a SELECT statement that uses column aliases, this column holds the alias. If you have also enabled **Use any Label (Alias)** in the **Options** section, this value is used in place of the name.
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 (see page 256) are defined.

12.2.1 Generating Test Data

The test data generator is useful when you need to add random column data to the exported output.

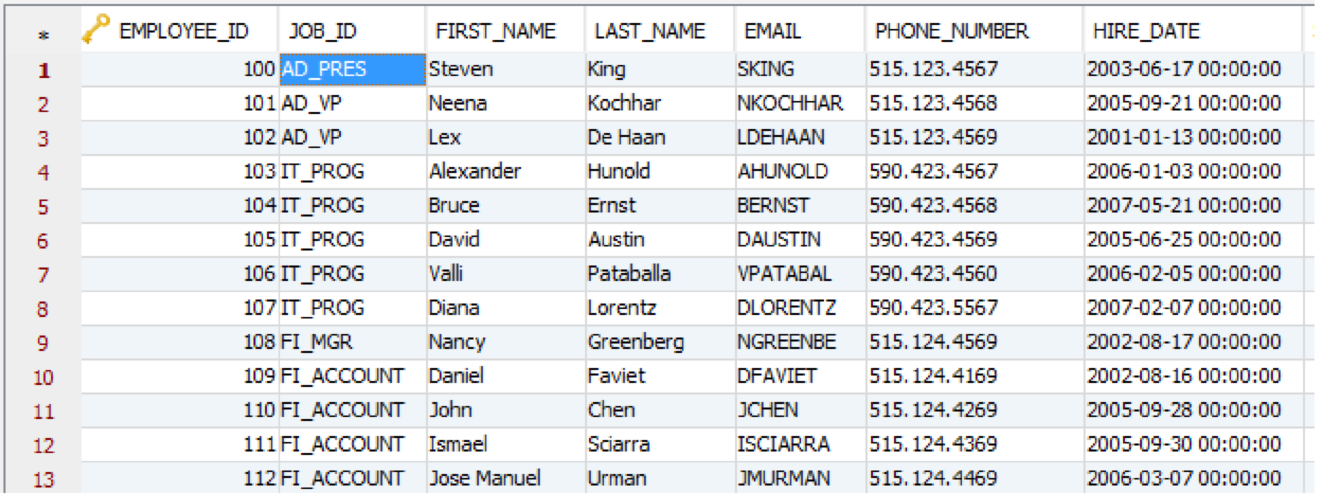
The **Value** column in the **Data** page grid specifies the data to be in the exported output. By default, it contains the `${value}$` variable, which is simply replaced with the real column value during the export process. You can also add static values before and after the `${value}$` variable, to be exported as entered.

Alternatively, you can use test data generator variables in the **Value** column. The choices are available in the right-click menu when you edit the **Value** column.

Function Name	Function Call	Example
Generate random number	`${var\|\|randomnumber(1, 2147483647)}$`	Generates a random number between 1 and 2147483647
Generate random string of random size	`${var\|\|randomtext(1, 10)}$`	Generates random text with a length between 1 an 10 characters
Generate random value from a list of values	`${var\|\|randomenum(v1, v2, v3, v4, v5)}$`	Picks one of the listed values in random order

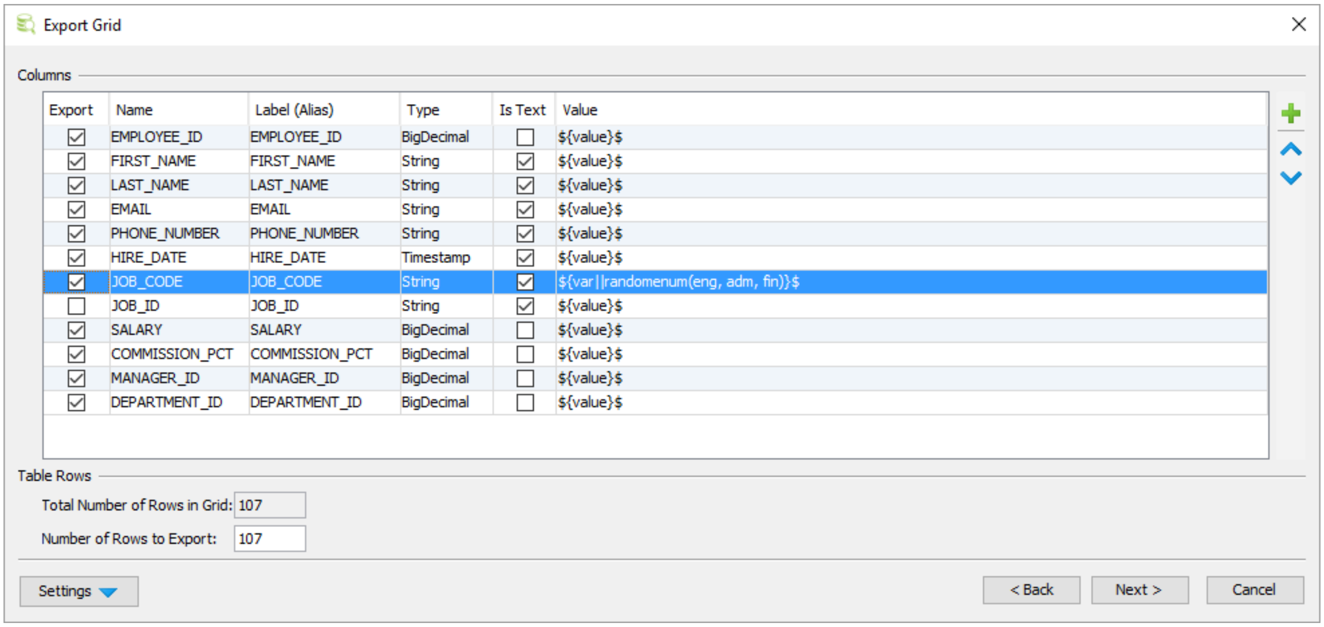
Function Name	Function Call	Example
Generate sequential number	`${var\|\|number(1, 2147483647, 1)}$`	Generates a sequential number starting from 1. The generator re-starts at 1 when 2147483647 is reached. The number is increased with 1 every time a new value is generated.

Here is an example of how to use the test data generators to try out planned changes to the data. Consider this initial data:



*	EMPLOYEE_ID	JOB_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	100	AD_PRES	Steven	King	SKING	515.123.4567	2003-06-17 00:00:00
2	101	AD_VP	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-21 00:00:00
3	102	AD_VP	Lex	De Haan	LDEHAAN	515.123.4569	2001-01-13 00:00:00
4	103	IT_PROG	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03 00:00:00
5	104	IT_PROG	Bruce	Ernst	BERNST	590.423.4568	2007-05-21 00:00:00
6	105	IT_PROG	David	Austin	DAUSTIN	590.423.4569	2005-06-25 00:00:00
7	106	IT_PROG	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-05 00:00:00
8	107	IT_PROG	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-07 00:00:00
9	108	FI_MGR	Nancy	Greenberg	NGREENBE	515.124.4569	2002-08-17 00:00:00
10	109	FI_ACCOUNT	Daniel	Faviet	DFAVIET	515.124.4169	2002-08-16 00:00:00
11	110	FI_ACCOUNT	John	Chen	JCHEN	515.124.4269	2005-09-28 00:00:00
12	111	FI_ACCOUNT	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-30 00:00:00
13	112	FI_ACCOUNT	Jose Manuel	Urman	JMURMAN	515.124.4469	2006-03-07 00:00:00

After the changes, the **JOB_ID** column should not appear in the output and the new **JOB_CODE** should contain abbreviated job codes. To test this, we simply uncheck the **Export** checkbox for **JOB** entry and set the **Value** for the **JOB_CODE** to use the **Generate random value from a list of values** function. By unchecking the "*Export*" checkbox for the column **JOB_ID** this column will not be included in the export.

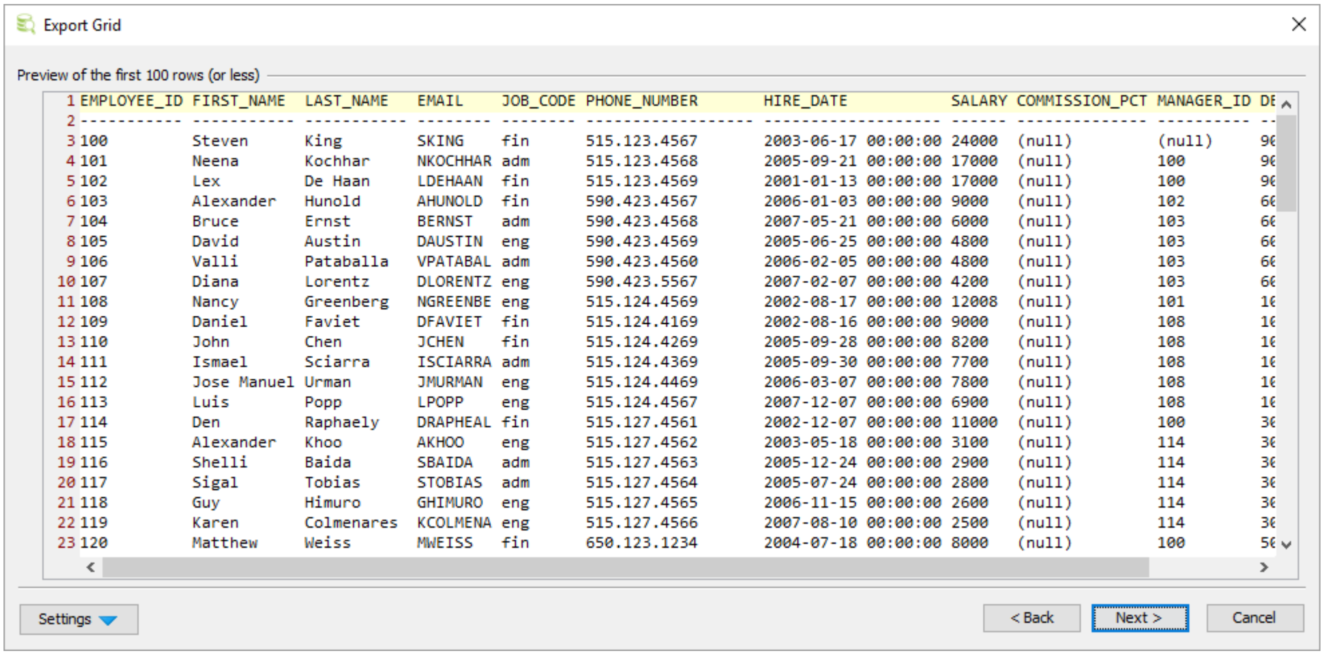


Export	Name	Label (Alias)	Type	Is Text	Value
☑	EMPLOYEE_ID	EMPLOYEE_ID	BigDecimal	☐	${value}$
☑	FIRST_NAME	FIRST_NAME	String	☑	${value}$
☑	LAST_NAME	LAST_NAME	String	☑	${value}$
☑	EMAIL	EMAIL	String	☑	${value}$
☑	PHONE_NUMBER	PHONE_NUMBER	String	☑	${value}$
☑	HIRE_DATE	HIRE_DATE	Timestamp	☑	${value}$
☑	JOB_CODE	JOB_CODE	String	☑	${var\|\|randomenum(eng, adm, fin)}$
☐	JOB_ID	JOB_ID	String	☑	${value}$
☑	SALARY	SALARY	BigDecimal	☐	${value}$
☑	COMMISSION_PCT	COMMISSION_PCT	BigDecimal	☐	${value}$
☑	MANAGER_ID	MANAGER_ID	BigDecimal	☐	${value}$
☑	DEPARTMENT_ID	DEPARTMENT_ID	BigDecimal	☐	${value}$

Total Number of Rows in Grid: 107

Number of Rows to Export: 107

Previewing the data (or exporting it) in CSV format results in this:



Export Grid

Preview of the first 100 rows (or less)

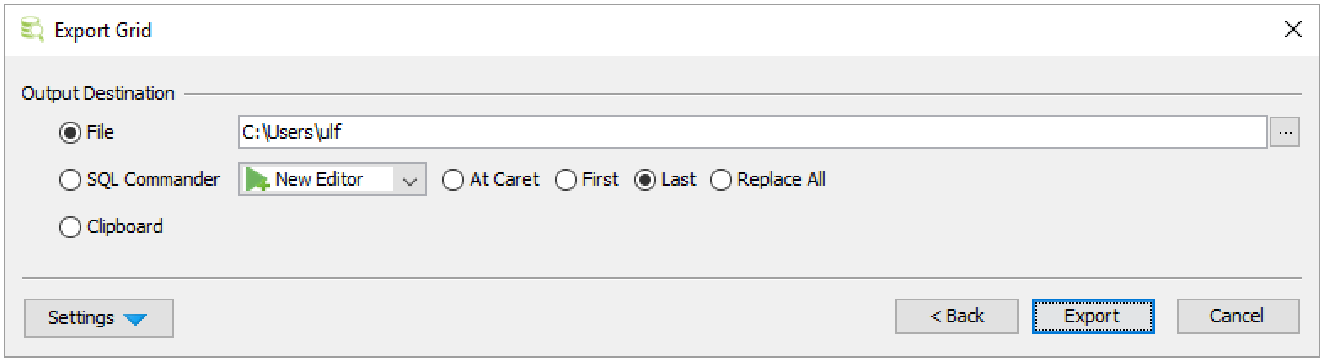
```
EMPLOYEE_ID FIRST_NAME  LAST_NAME   EMAIL    JOB_CODE PHONE_NUMBER       HIRE_DATE           SALARY COMMISSION_PCT MANAGER_ID DE
----------- ----------- ----------- -------- -------- ------------------ ------------------- ------ -------------- ---------- --
100         Steven      King        SKING    fin      515.123.4567       2003-06-17 00:00:00 24000  (null)         (null)     90
101         Neena       Kochhar     NKOCHHAR adm      515.123.4568       2005-09-21 00:00:00 17000  (null)         100        90
102         Lex         De Haan     LDEHAAN  fin      515.123.4569       2001-01-13 00:00:00 17000  (null)         100        90
103         Alexander   Hunold      AHUNOLD  fin      590.423.4567       2006-01-03 00:00:00 9000   (null)         102        60
104         Bruce       Ernst       BERNST   adm      590.423.4568       2007-05-21 00:00:00 6000   (null)         103        60
105         David       Austin      DAUSTIN  eng      590.423.4569       2005-06-25 00:00:00 4800   (null)         103        60
106         Valli       Pataballa   VPATABAL adm      590.423.4560       2006-02-05 00:00:00 4800   (null)         103        60
107         Diana       Lorentz     DLORENTZ eng      590.423.5567       2007-02-07 00:00:00 4200   (null)         103        60
108         Nancy       Greenberg   NGREENBE eng      515.124.4569       2002-08-17 00:00:00 12008  (null)         101        10
109         Daniel      Faviet      DFAVIET  fin      515.124.4169       2002-08-16 00:00:00 9000   (null)         108        10
110         John        Chen        JCHEN    fin      515.124.4269       2005-09-28 00:00:00 8200   (null)         108        10
111         Ismael      Sciarra     ISCIARRA adm      515.124.4369       2005-09-30 00:00:00 7700   (null)         108        10
112         Jose Manuel Urman       JMURMAN  eng      515.124.4469       2006-03-07 00:00:00 7800   (null)         108        10
113         Luis        Popp        LPOPP    eng      515.124.4567       2007-12-07 00:00:00 6900   (null)         108        10
114         Den         Raphaely    DRAPHEAL fin      515.127.4561       2002-12-07 00:00:00 11000  (null)         100        30
115         Alexander   Khoo        AKHOO    eng      515.127.4562       2003-05-18 00:00:00 3100   (null)         114        30
116         Shelli      Baida       SBAIDA   adm      515.127.4563       2005-12-24 00:00:00 2900   (null)         114        30
117         Sigal       Tobias      STOBIAS  adm      515.127.4564       2005-07-24 00:00:00 2800   (null)         114        30
118         Guy         Himuro      GHIMURO  eng      515.127.4565       2006-11-15 00:00:00 2600   (null)         114        30
119         Karen       Colmenares  KCOLMENA eng      515.127.4566       2007-08-10 00:00:00 2500   (null)         114        30
120         Matthew     Weiss       MWEISS   fin      650.123.1234       2004-07-18 00:00:00 8000   (null)         100        50
```

12.3 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.

12.4 Output Destination

The final wizard page is the **Output Destination** page. The destination field specifies the target destination for the exported data, one of File, SQL Commander or Clipboard.



Export Grid

Output Destination

● File: C:\Users\ulf

○ SQL Commander: New Editor — ○ At Caret ○ First ● Last ○ Replace All

○ Clipboard

Click **Export** on this page to export the grid data to the selected destination.

12.5 Settings Menu

If you often use the same settings, you can save them as the default settings for this assistant. If you use a number of common settings, you can save them to individual files that you can load as needed. Use the Settings button menu to accomplish this:

- **Save as Default Settings**
  Saves all format settings as default. These are then loaded automatically when open an Export Schema dialog
- **Use Default Settings**
  Use this choice to initialize the settings with default values
- **Remove Default Settings**
  Removes the saved defaults and restores the regular defaults
- **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

You can also use settings saved here with the @export client side command (see page 214).

13 Comparing Data

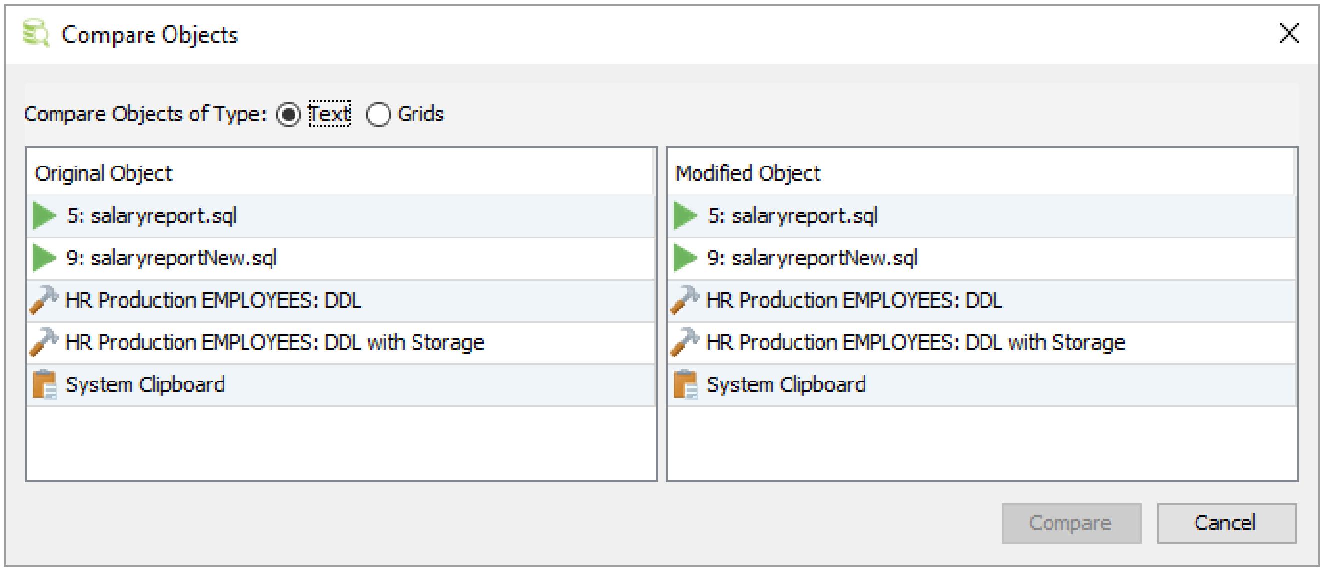
Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

With DbVisualizer, you can compare grids and text data, such as scripts or the DDL for two tables or procedures .

13.1 Selecting the Objects to Compare

You can open the **Compare Objects** object chooser via **Tools->Compare** and select two objects available for comparison.



Compare Objects

Compare Objects of Type: ● Text ○ Grids

Original Object	Modified Object
5: salaryreport.sql	5: salaryreport.sql
9: salaryreportNew.sql	9: salaryreportNew.sql
HR Production EMPLOYEES: DDL	HR Production EMPLOYEES: DDL
HR Production EMPLOYEES: DDL with Storage	HR Production EMPLOYEES: DDL with Storage
System Clipboard	System Clipboard

If you select **Text**, all **SQL Commander** editors and all **Object View** sub tabs that you have opened that contain text, such as **DDL** and **Procedure Editor** tabs, are listed. There is also an entry for the **System Clipboard**, holding the last text you copied.

Selecting **Grids** lists all **SQL Commander** Result Set tabs and all **Object View** sub tabs that you have opened that contain a grid, for instance the **Data** and **Columns** tabs for a table, or the **Tables** tab for a schema.

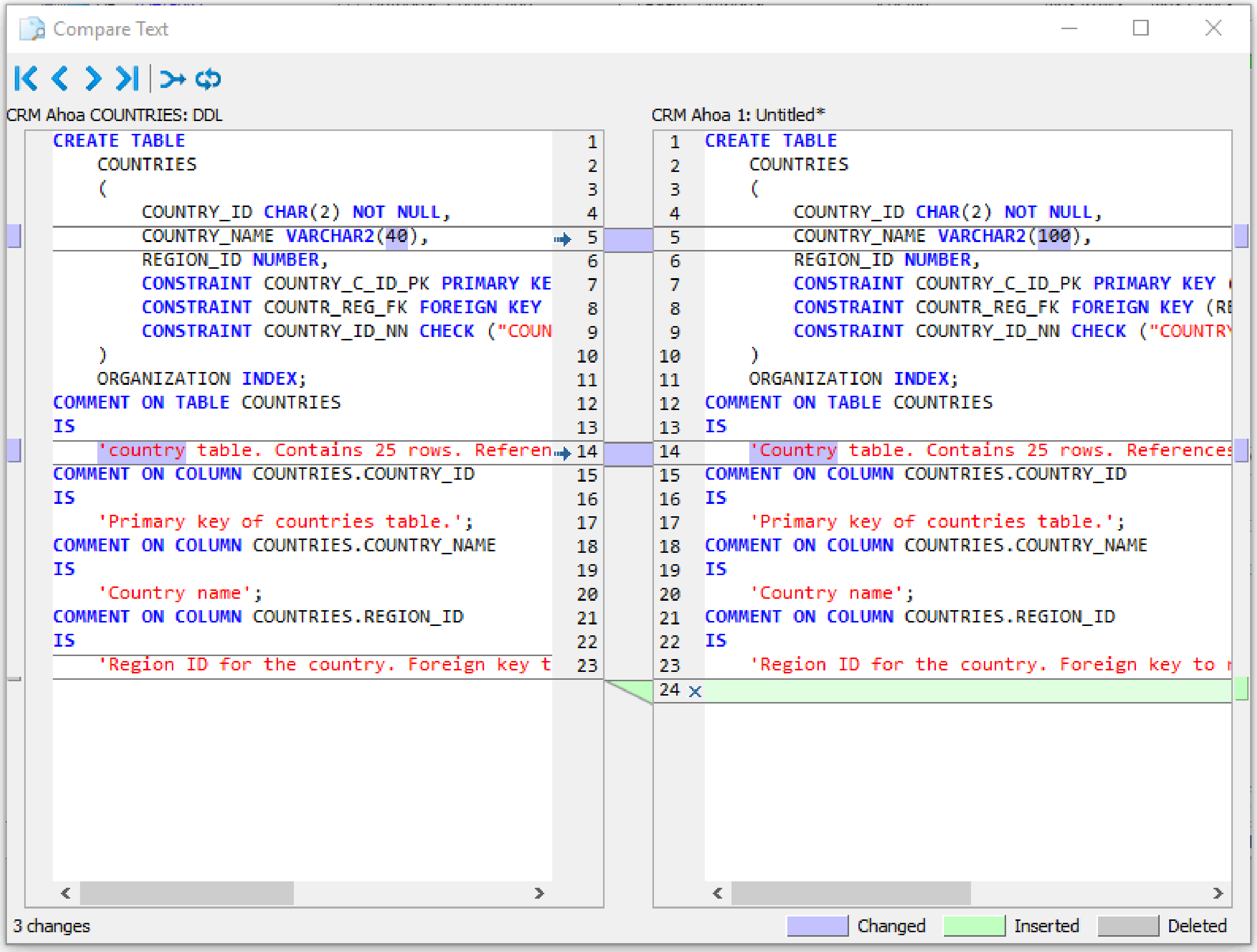
To compare two objects, select one each in the **Original Object** and **Modified Object** columns and click **Compare**.

You can also open the object selection dialog from the right-click menu inside a tab that holds an object that can be compared. The object shown in that tab is then preselected as the **Modified Object** so you only need to select the **Original Object** in the dialog.

The right-click menu for an **SQL Commander** editor also contains a **Compare to Saved** entry. This bypasses the object selection dialog and opens the **Compare** window directly, showing you how you have changed the script since loading it into the editor.

13.2 Comparing Text Data

To compare text data, either select both text object from the **Tools->Compare** dialog or choose **Compare** from the right-click menu in one of them and select the one to compare to. The text compare window shows you how they differ.



```
CREATE TABLE
    COUNTRIES
    (
        COUNTRY_ID CHAR(2) NOT NULL,
        COUNTRY_NAME VARCHAR2(40),
        REGION_ID NUMBER,
        CONSTRAINT COUNTRY_C_ID_PK PRIMARY KE
        CONSTRAINT COUNTR_REG_FK FOREIGN KEY
        CONSTRAINT COUNTRY_ID_NN CHECK ("COUN
    )
    ORGANIZATION INDEX;
COMMENT ON TABLE COUNTRIES
IS
    'country table. Contains 25 rows. Referen
COMMENT ON COLUMN COUNTRIES.COUNTRY_ID
IS
    'Primary key of countries table.';
COMMENT ON COLUMN COUNTRIES.COUNTRY_NAME
IS
    'Country name';
COMMENT ON COLUMN COUNTRIES.REGION_ID
IS
    'Region ID for the country. Foreign key t
```

```
CREATE TABLE
    COUNTRIES
    (
        COUNTRY_ID CHAR(2) NOT NULL,
        COUNTRY_NAME VARCHAR2(100),
        REGION_ID NUMBER,
        CONSTRAINT COUNTRY_C_ID_PK PRIMARY KEY (
        CONSTRAINT COUNTR_REG_FK FOREIGN KEY (RE
        CONSTRAINT COUNTRY_ID_NN CHECK ("COUNTRY
    )
    ORGANIZATION INDEX;
COMMENT ON TABLE COUNTRIES
IS
    'Country table. Contains 25 rows. References
COMMENT ON COLUMN COUNTRIES.COUNTRY_ID
IS
    'Primary key of countries table.';
COMMENT ON COLUMN COUNTRIES.COUNTRY_NAME
IS
    'Country name';
COMMENT ON COLUMN COUNTRIES.REGION_ID
IS
    'Region ID for the country. Foreign key to r

```

3 changes

▭ Changed 🟩 Inserted ▨ Deleted

The objects are shown side-by-side, with indications about how they differ in the divider between them. The areas to the left of the original object and to the right of the modified object have markings that also show sections of differences. You can navigate between the differences using the arrow buttons in the toolbar, or by clicking the markings to the left and right.

Comparing two texts is pretty straightforward. You can see what has been changed, inserted and deleted, with row level indications in the divider and details for changed text highlighted in the text panes. A difference may start on one line and span over multiple lines until a match is found again.
If the modified object (right pane) is editable, you can apply the changes needed to remove the differences, one by one or all at once.

Along with the difference indications in the divider there are small icons: arrows and cross marks. Clicking on such an icon updates the modified object to match the original object, by inserting or deleting one or more rows or updating the text or column values.

If you want to apply all changes needed to make the modified object match the original, you can click on the **Sync** button in the toolbar (the second to rightmost button).

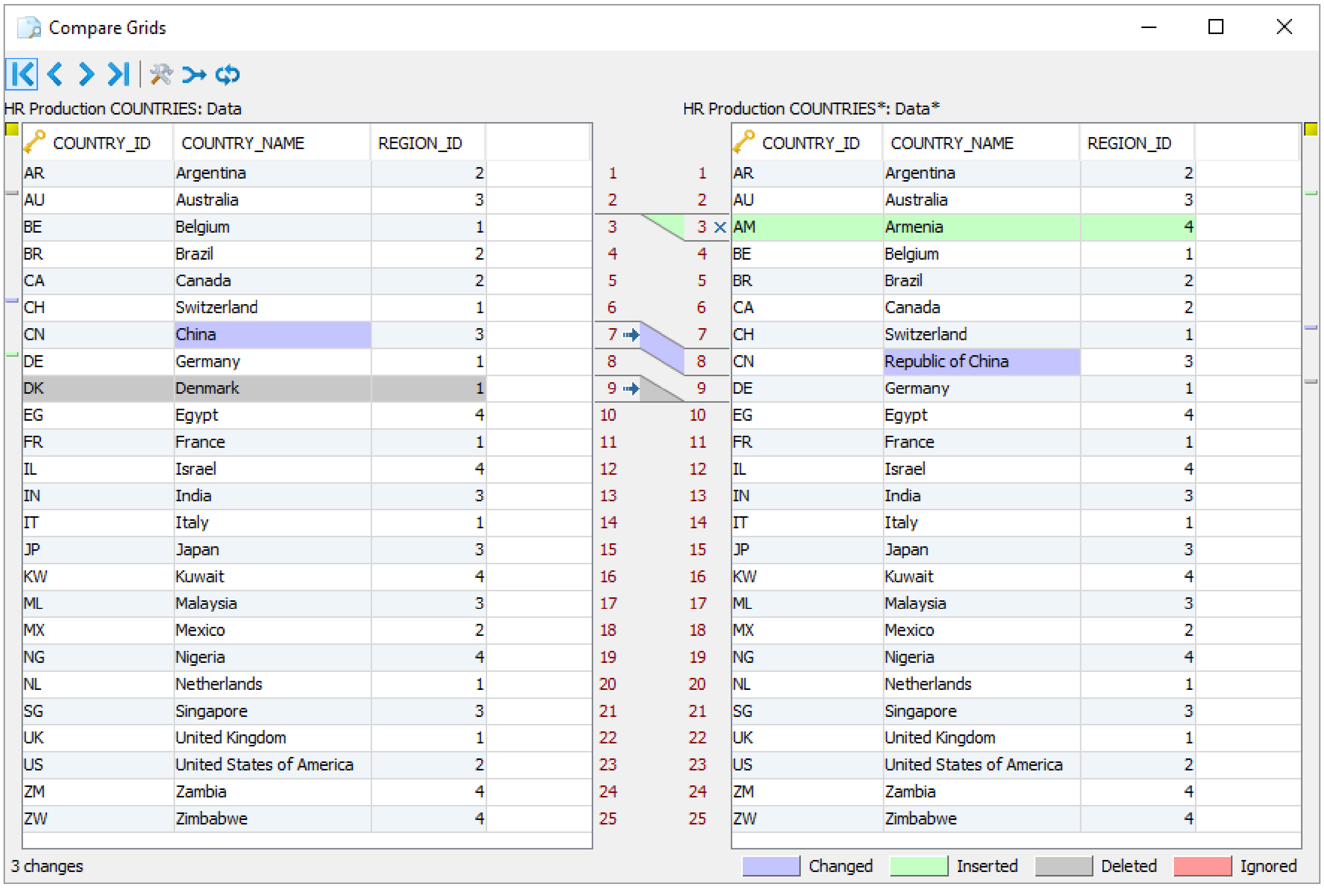
You can also edit the modified object directly in the **Compare** window.

> ⚠ No matter how you update the modified object, all changes are also applied to the tab where the object was originally opened. To permanently save the changes, you need to use the **Save** button in that tab.

If you want to change which object should be considered the original and the modified when comparing them, you can click the **Flip** button in the toolbar (the rightmost button). One reason for doing this may be that you want to update the object you originally used as the original instead of the modified object.

13.3 Comparing Grids

To compare grids, either select both grid object from the **Tools->Compare** dialog or choose **Compare** from the right-click menu in one of them and select the one to compare to. The grid compare window shows you how they differ.



HR Production COUNTRIES: Data | HR Production COUNTRIES*: Data*

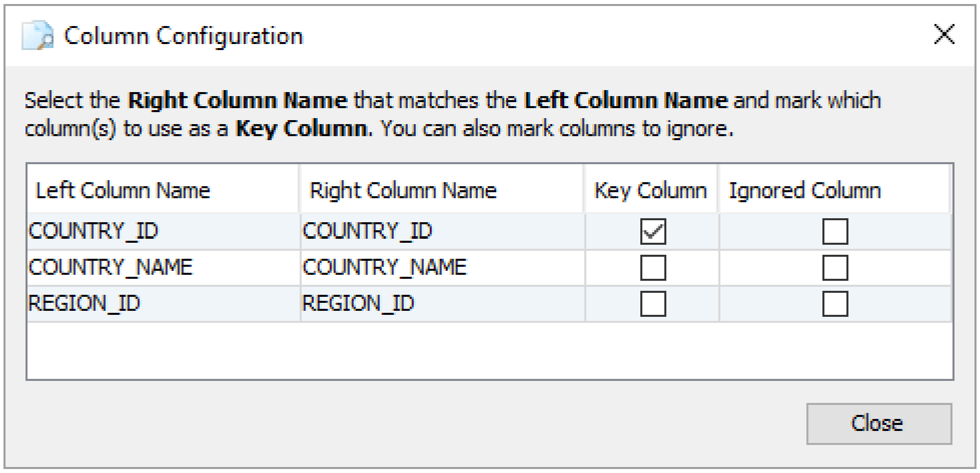
COUNTRY_ID	COUNTRY_NAME	REGION_ID			COUNTRY_ID	COUNTRY_NAME	REGION_ID
AR	Argentina	2	1	1	AR	Argentina	2
AU	Australia	3	2	2	AU	Australia	3
BE	Belgium	1	3	3 ✕	AM	Armenia	4
BR	Brazil	2	4	4	BE	Belgium	1
CA	Canada	2	5	5	BR	Brazil	2
CH	Switzerland	1	6	6	CA	Canada	2
CN	China	3	7 ➜	7	CH	Switzerland	1
DE	Germany	1	8	8	CN	Republic of China	3
DK	Denmark	1	9 ➜	9	DE	Germany	1
EG	Egypt	4	10	10	EG	Egypt	4
FR	France	1	11	11	FR	France	1
IL	Israel	4	12	12	IL	Israel	4
IN	India	3	13	13	IN	India	3
IT	Italy	1	14	14	IT	Italy	1
JP	Japan	3	15	15	JP	Japan	3
KW	Kuwait	4	16	16	KW	Kuwait	4
ML	Malaysia	3	17	17	ML	Malaysia	3
MX	Mexico	2	18	18	MX	Mexico	2
NG	Nigeria	4	19	19	NG	Nigeria	4
NL	Netherlands	1	20	20	NL	Netherlands	1
SG	Singapore	3	21	21	SG	Singapore	3
UK	United Kingdom	1	22	22	UK	United Kingdom	1
US	United States of America	2	23	23	US	United States of America	2
ZM	Zambia	4	24	24	ZM	Zambia	4
ZW	Zimbabwe	4	25	25	ZW	Zimbabwe	4

3 changes

🟪 Changed 🟩 Inserted ▨ Deleted 🟥 Ignored

Comparing grids is a bit complicated, since each row is a unit in itself, with a unique identifier in the form of a Key. A difference can therefore not span rows. It is also important that both grids are sorted the same way and that the column in one grid is compared to the corresponding column in the other grid, otherwise the result is indeterminably. By default, columns are matched by name, or index if the names differ.

If the default matching is not correct, click the **Column Configuration** button in the toolbar to manually match the columns, and optionally select key columns and ignore some columns.



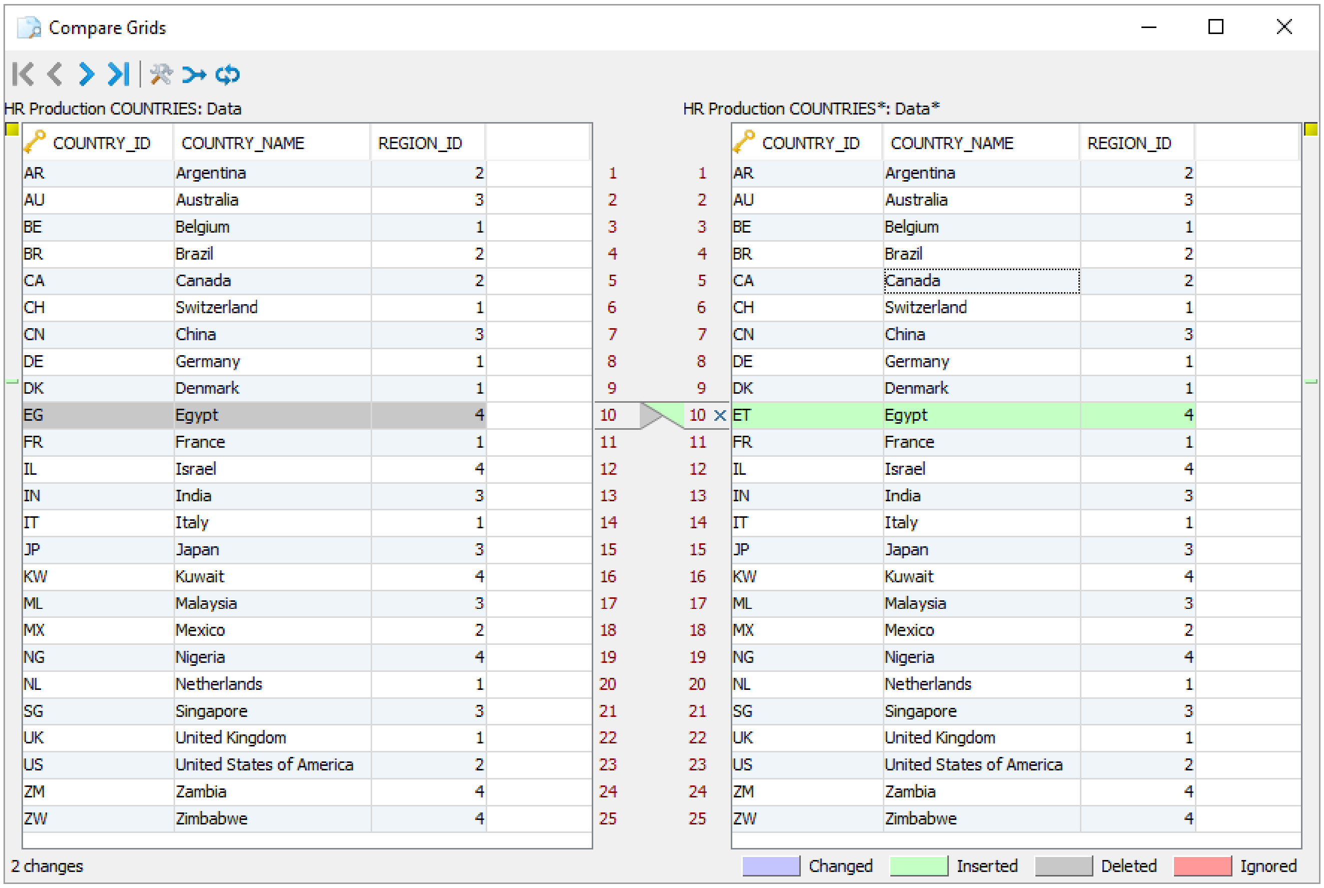
Select the **Right Column Name** that matches the **Left Column Name** and mark which column(s) to use as a **Key Column**. You can also mark columns to ignore.

Left Column Name	Right Column Name	Key Column	Ignored Column
COUNTRY_ID	COUNTRY_ID	☑	□
COUNTRY_NAME	COUNTRY_NAME	□	□
REGION_ID	REGION_ID	□	□

Use the **Right Column Name** drop down lists to select the column matching the **Left Column Name**. If a column does not match another, just leave it blank to exclude it from the comparison.

Use the **Key Column** check boxes to pick the columns that should be used as the key when comparing the grids. If you don't care about the value in some columns, e.g. a column that contains a timestamp that may vary between the grids without being an important difference, you can check the **Ignore** check box for that column to exclude it for the comparison.

It gets a bit more complicated if a key column value is changed.



Compare Grids

HR Production COUNTRIES: Data

HR Production COUNTRIES*: Data*

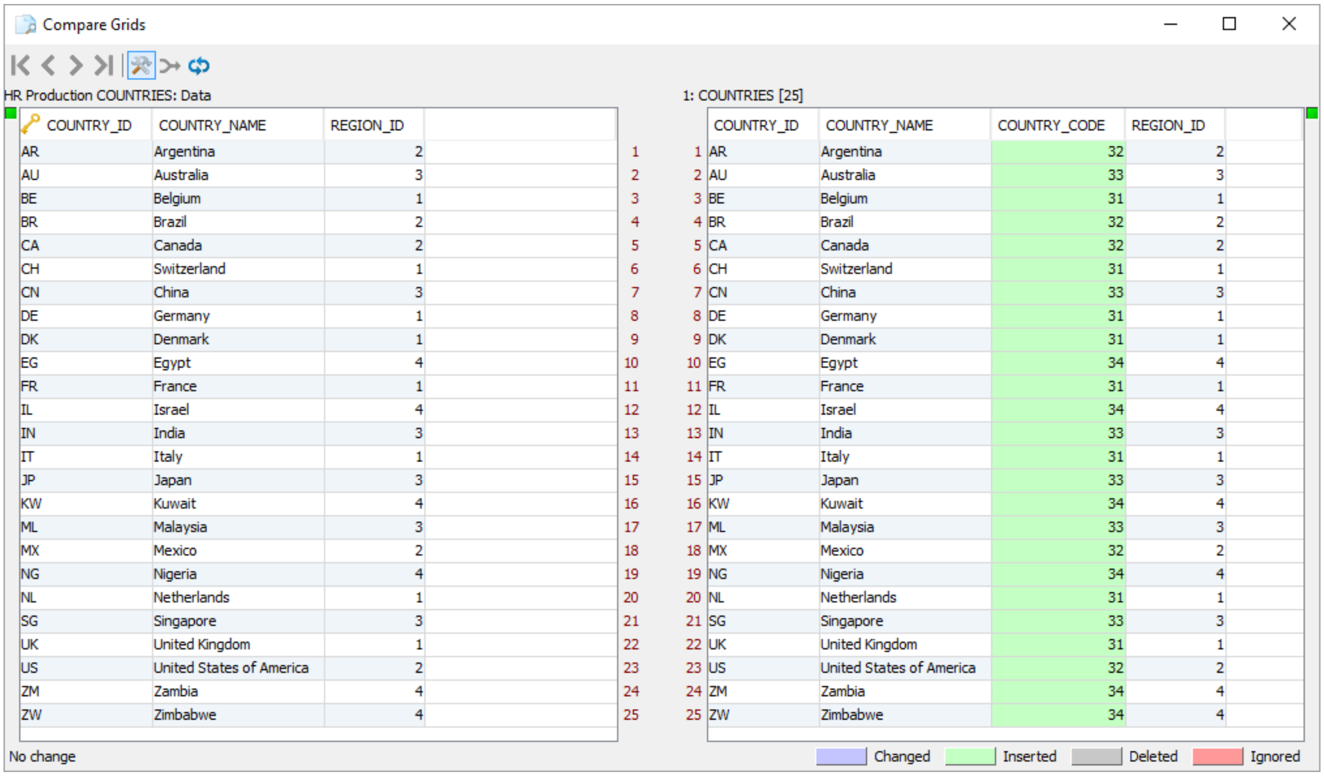
COUNTRY_ID	COUNTRY_NAME	REGION_ID			COUNTRY_ID	COUNTRY_NAME	REGION_ID
AR	Argentina	2	1	1	AR	Argentina	2
AU	Australia	3	2	2	AU	Australia	3
BE	Belgium	1	3	3	BE	Belgium	1
BR	Brazil	2	4	4	BR	Brazil	2
CA	Canada	2	5	5	CA	Canada	2
CH	Switzerland	1	6	6	CH	Switzerland	1
CN	China	3	7	7	CN	China	3
DE	Germany	1	8	8	DE	Germany	1
DK	Denmark	1	9	9	DK	Denmark	1
EG	Egypt	4	10	10 ×	ET	Egypt	4
FR	France	1	11	11	FR	France	1
IL	Israel	4	12	12	IL	Israel	4
IN	India	3	13	13	IN	India	3
IT	Italy	1	14	14	IT	Italy	1
JP	Japan	3	15	15	JP	Japan	3
KW	Kuwait	4	16	16	KW	Kuwait	4
ML	Malaysia	3	17	17	ML	Malaysia	3
MX	Mexico	2	18	18	MX	Mexico	2
NG	Nigeria	4	19	19	NG	Nigeria	4
NL	Netherlands	1	20	20	NL	Netherlands	1
SG	Singapore	3	21	21	SG	Singapore	3
UK	United Kingdom	1	22	22	UK	United Kingdom	1
US	United States of America	2	23	23	US	United States of America	2
ZM	Zambia	4	24	24	ZM	Zambia	4
ZW	Zimbabwe	4	25	25	ZW	Zimbabwe	4

2 changes

🟪 Changed 🟩 Inserted ▨ Deleted 🟥 Ignored

DbVisualizer considers a changed key value as one inserted row and one deleted row, as shown in the figure above. If the grids do not have any declared key columns, all columns are considered to be regular, non-key columns.

Two grids may also differ in the number of columns they contain. DbVisualizer finds columns that only exist in one of the grids and excludes their values when comparing the grids. If the column names do not match between the two grids, open the Column Configuration dialog to manually map the columns.



Compare Grids

HR Production COUNTRIES: Data

COUNTRY_ID	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4
FR	France	1
IL	Israel	4
IN	India	3
IT	Italy	1
JP	Japan	3
KW	Kuwait	4
ML	Malaysia	3
MX	Mexico	2
NG	Nigeria	4
NL	Netherlands	1
SG	Singapore	3
UK	United Kingdom	1
US	United States of America	2
ZM	Zambia	4
ZW	Zimbabwe	4

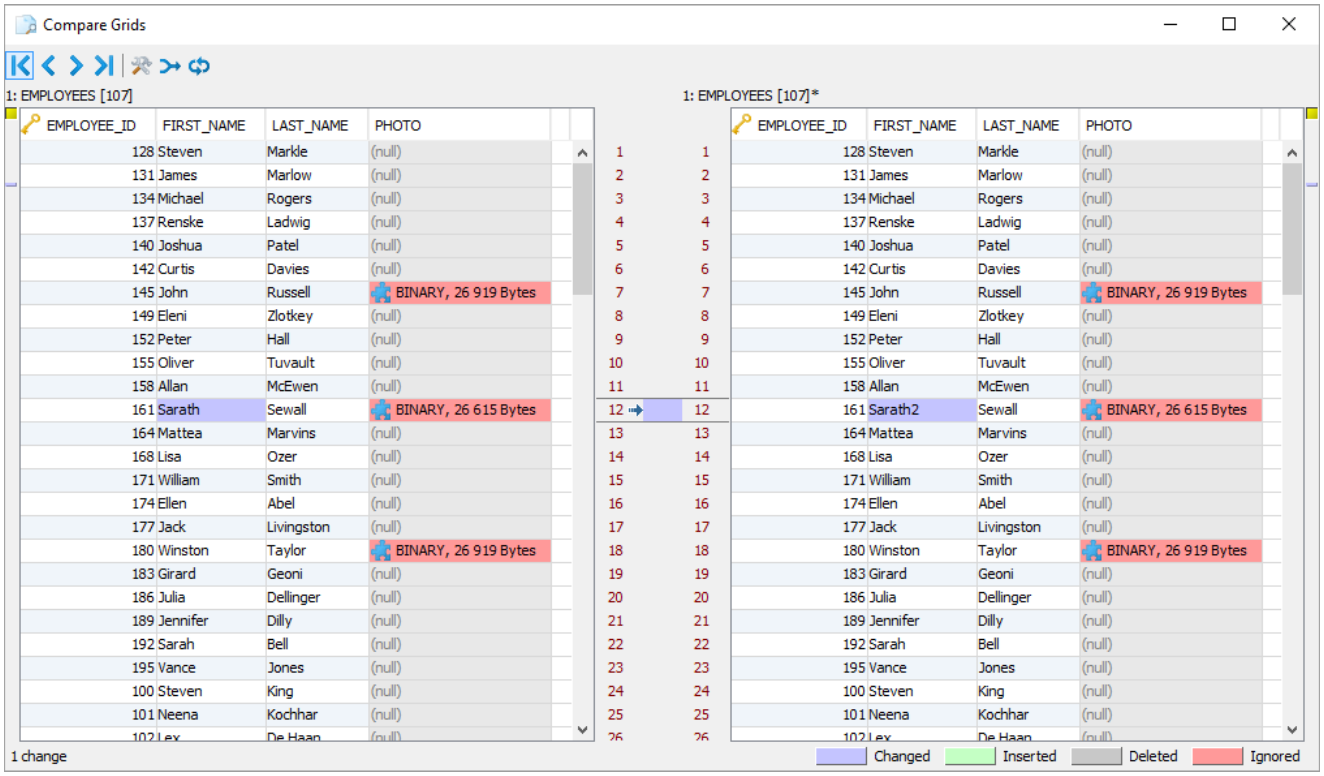
1: COUNTRIES [25]

		COUNTRY_ID	COUNTRY_NAME	COUNTRY_CODE	REGION_ID
1	1	AR	Argentina	32	2
2	2	AU	Australia	33	3
3	3	BE	Belgium	31	1
4	4	BR	Brazil	32	2
5	5	CA	Canada	32	2
6	6	CH	Switzerland	31	1
7	7	CN	China	33	3
8	8	DE	Germany	31	1
9	9	DK	Denmark	31	1
10	10	EG	Egypt	34	4
11	11	FR	France	31	1
12	12	IL	Israel	34	4
13	13	IN	India	33	3
14	14	IT	Italy	31	1
15	15	JP	Japan	33	3
16	16	KW	Kuwait	34	4
17	17	ML	Malaysia	33	3
18	18	MX	Mexico	32	2
19	19	NG	Nigeria	34	4
20	20	NL	Netherlands	31	1
21	21	SG	Singapore	33	3
22	22	UK	United Kingdom	31	1
23	23	US	United States of America	32	2
24	24	ZM	Zambia	34	4
25	25	ZW	Zimbabwe	34	4

No change

🟪 Changed 🟩 Inserted ⬜ Deleted 🟥 Ignored

Similarly, DbVisualizer does not consider Binary/BLOB and CLOB columns when comparing, and marks them as ignored. You can manually specify that CLOB columns should be compared in the Column Configuration dialog.



Compare Grids

1: EMPLOYEES [107]

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHOTO
128	Steven	Markle	(null)
131	James	Marlow	(null)
134	Michael	Rogers	(null)
137	Renske	Ladwig	(null)
140	Joshua	Patel	(null)
142	Curtis	Davies	(null)
145	John	Russell	BINARY, 26 919 Bytes
149	Eleni	Zlotkey	(null)
152	Peter	Hall	(null)
155	Oliver	Tuvault	(null)
158	Allan	McEwen	(null)
161	Sarath	Sewall	BINARY, 26 615 Bytes
164	Mattea	Marvins	(null)
168	Lisa	Ozer	(null)
171	William	Smith	(null)
174	Ellen	Abel	(null)
177	Jack	Livingston	(null)
180	Winston	Taylor	BINARY, 26 919 Bytes
183	Girard	Geoni	(null)
186	Julia	Dellinger	(null)
189	Jennifer	Dilly	(null)
192	Sarah	Bell	(null)
195	Vance	Jones	(null)
100	Steven	King	(null)
101	Neena	Kochhar	(null)
102	Lex	De Haan	(null)

1: EMPLOYEES [107]*

		EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHOTO
1	1	128	Steven	Markle	(null)
2	2	131	James	Marlow	(null)
3	3	134	Michael	Rogers	(null)
4	4	137	Renske	Ladwig	(null)
5	5	140	Joshua	Patel	(null)
6	6	142	Curtis	Davies	(null)
7	7	145	John	Russell	BINARY, 26 919 Bytes
8	8	149	Eleni	Zlotkey	(null)
9	9	152	Peter	Hall	(null)
10	10	155	Oliver	Tuvault	(null)
11	11	158	Allan	McEwen	(null)
12 →	12	161	Sarath2	Sewall	BINARY, 26 615 Bytes
13	13	164	Mattea	Marvins	(null)
14	14	168	Lisa	Ozer	(null)
15	15	171	William	Smith	(null)
16	16	174	Ellen	Abel	(null)
17	17	177	Jack	Livingston	(null)
18	18	180	Winston	Taylor	BINARY, 26 919 Bytes
19	19	183	Girard	Geoni	(null)
20	20	186	Julia	Dellinger	(null)
21	21	189	Jennifer	Dilly	(null)
22	22	192	Sarah	Bell	(null)
23	23	195	Vance	Jones	(null)
24	24	100	Steven	King	(null)
25	25	101	Neena	Kochhar	(null)
26	26	102	Lex	De Haan	(null)

1 change

🟪 Changed 🟩 Inserted ⬜ Deleted 🟥 Ignored

If the modified object (right pane) is editable, you can apply the changes needed to remove the differences, one by one or all at once.

Along with the difference indications in the divider there are small icons: arrows and cross marks. Clicking on such an icon updates the modified object to match the original object, by inserting or deleting one or more rows or updating the text or column values.

If you want to apply all changes needed to make the modified object match the original, you can click on the **Sync** button in the toolbar (the second to rightmost button).

You can also edit the modified object directly in the **Compare** window.

⚠ No matter how you update the modified object, all changes are also applied to the tab where the object was originally opened. To permanently save the changes, you need to use the **Save** button in that tab.

If you want to change which object should be considered the original and the modified when comparing them, you can click the **Flip** button in the toolbar (the rightmost button). One reason for doing this may be that you want to update the object you originally used as the original instead of the modified object.

13.4 Comparing Cell Values

In a Data tab or a result set tab, you can also compare the values of two selected cells.

1. Select the two cells to compare,
2. Choose **Compare Selected Cells** from the right-click menu.

The values are compared and shown the same way as when comparing text data, with the exception that editing is disabled.

14 Monitoring Data Changes

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 (see page 244) capability in DbVisualizer Pro 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.

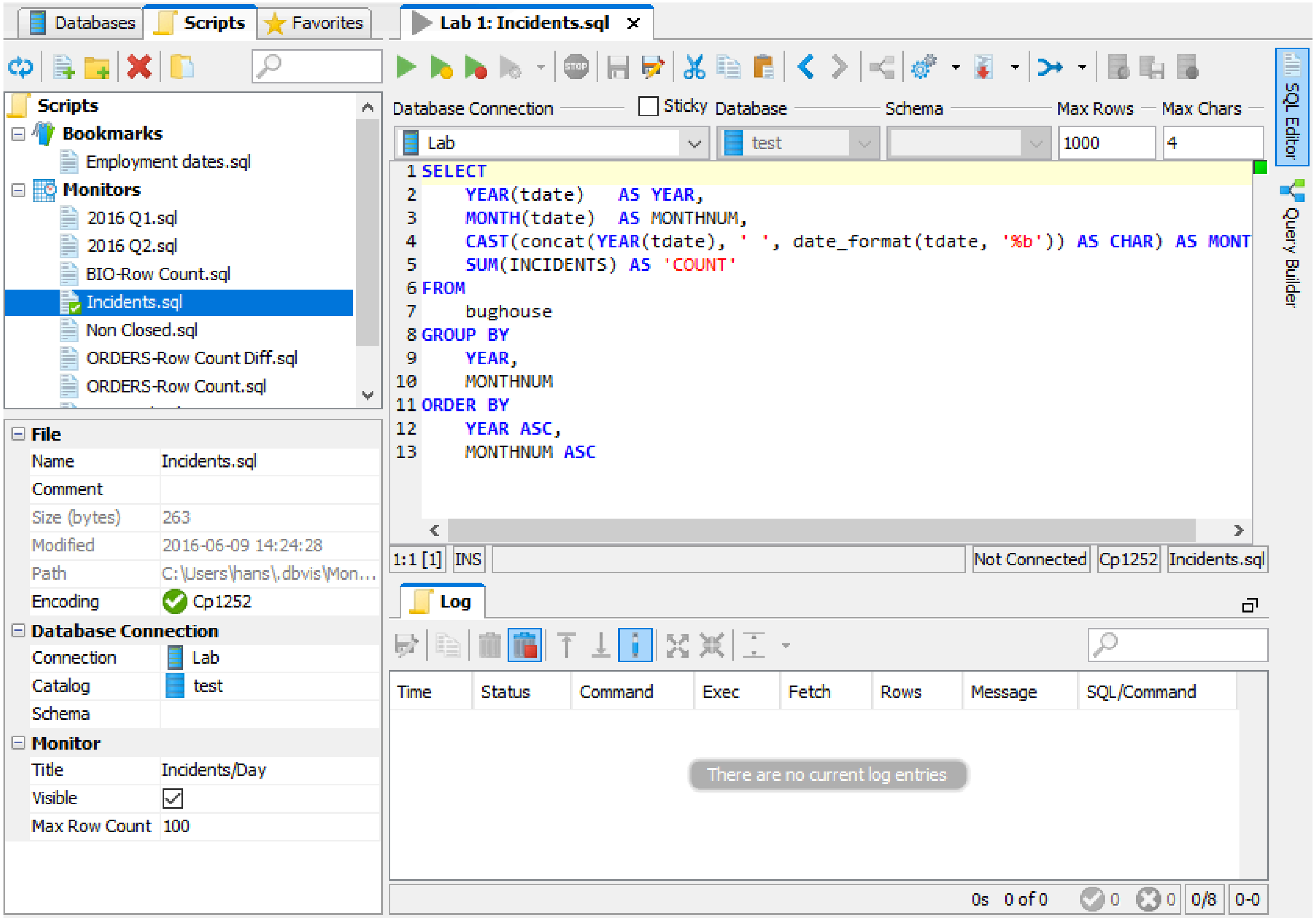
Any SQL statement that produces a result set can be monitored, and when you monitor multiple statements, different statements may use different database connections concurrently.

14.1 Creating a Monitored Query

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.

- Monitor table row count (see page 269)
- Monitor table row count difference (see page 271)

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.



Database Connection: Lab □ Sticky Database: test Schema: Max Rows: 1000 Max Chars: 4

```
SELECT
    YEAR(tdate)   AS YEAR,
    MONTH(tdate)  AS MONTHNUM,
    CAST(concat(YEAR(tdate), ' ', date_format(tdate, '%b')) AS CHAR) AS MONT
    SUM(INCIDENTS) AS 'COUNT'
FROM
    bughouse
GROUP BY
    YEAR,
    MONTHNUM
ORDER BY
    YEAR ASC,
    MONTHNUM ASC
```

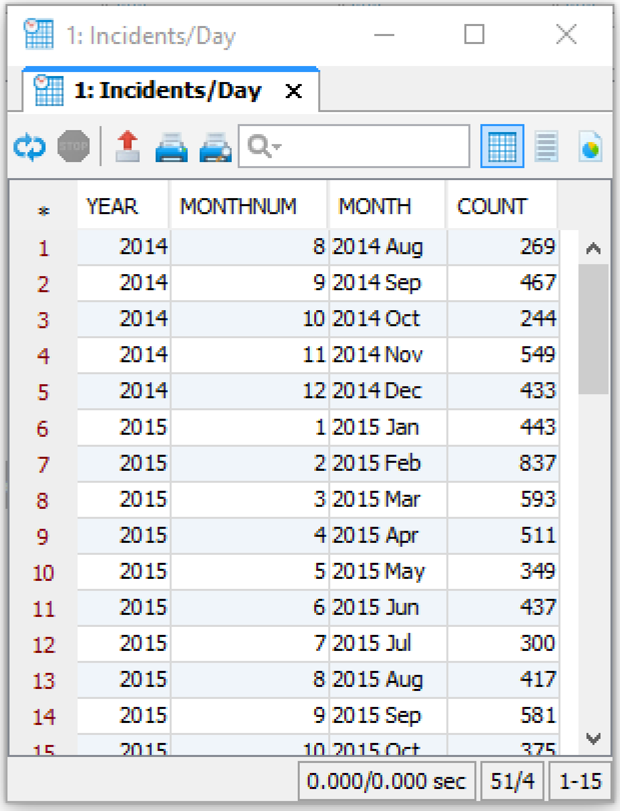
File	
Name	Incidents.sql
Comment	
Size (bytes)	263
Modified	2016-06-09 14:24:28
Path	C:\Users\hans\.dbvis\Mon...
Encoding	✓ Cp1252
Database Connection	
Connection	Lab
Catalog	test
Schema	
Monitor	
Title	Incidents/Day
Visible	☑
Max Row Count	100

Time	Status	Command	Exec	Fetch	Rows	Message	SQL/Command

There are no current log entries

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
1	2014	8	2014 Aug	269
2	2014	9	2014 Sep	467
3	2014	10	2014 Oct	244
4	2014	11	2014 Nov	549
5	2014	12	2014 Dec	433
6	2015	1	2015 Jan	443
7	2015	2	2015 Feb	837
8	2015	3	2015 Mar	593
9	2015	4	2015 Apr	511
10	2015	5	2015 May	349
11	2015	6	2015 Jun	437
12	2015	7	2015 Jul	300
13	2015	8	2015 Aug	417
14	2015	9	2015 Sep	581
15	2015	10	2015 Oct	375

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.

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 Managing Frequently Used SQL (see page 187) page. The following sections describe how you can get help creating the bookmarks for a couple of cases that are commonly used for monitoring.

14.1.1 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 grid for a table or result set therefore has a **Create Row Count Monitor** operation that creates a monitored statement for you automatically.

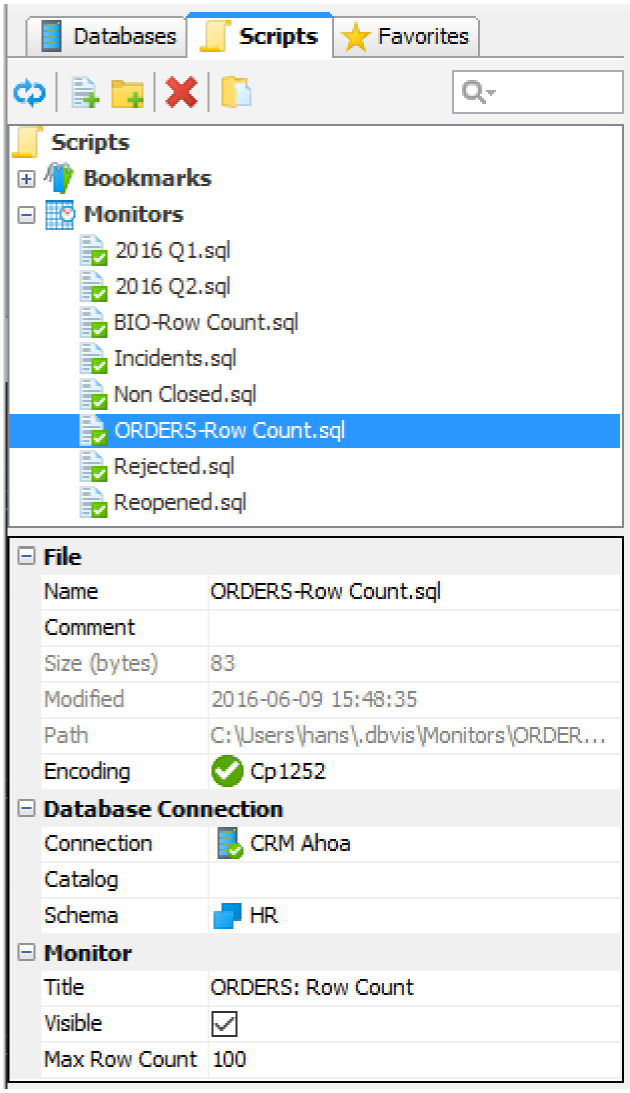
It creates a monitor with SQL for returning a single row with the timestamp for when the monitor was executed and the total number of rows in the table at that time. Every time the monitor is executed, a new row is added to the grid, up to a specified maximum number of rows. When the maximum row limit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

PollTime	NumRows
2016-01-23 12:19:10	43123
2016-01-23 12:11:40	43139
2016-01-23 12:21:10	43143
2016-01-23 12:22:40	43184
…	…

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 **NumRows** labels are not changed.

```
SELECT '${dbvis-date}$ ${dbvis-time}$' AS PollTime,
       COUNT(*) AS NumRows
FROM ORDERS
```

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.



- Scripts
  - Bookmarks
  - Monitors
    - 2016 Q1.sql
    - 2016 Q2.sql
    - BIO-Row Count.sql
    - Incidents.sql
    - Non Closed.sql
    - ORDERS-Row Count.sql
    - Rejected.sql
    - Reopened.sql

File	
Name	ORDERS-Row Count.sql
Comment	
Size (bytes)	83
Modified	2016-06-09 15:48:35
Path	C:\Users\hans\.dbvis\Monitors\ORDER...
Encoding	✓ Cp1252
Database Connection	
Connection	CRM Ahoa
Catalog	
Schema	HR
Monitor	
Title	ORDERS: Row Count
Visible	☑
Max Row Count	100

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.

14.1.2 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 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	NumRows	NumRowsChange
2016-01-23 12:19:10	43123	0

PollTime	NumRows	NumRowsChange
2016-01-23 12:11:40	43139	16
2016-01-23 12:21:10	43143	4
2016-01-23 12:22:40	43184	41
…	…	…

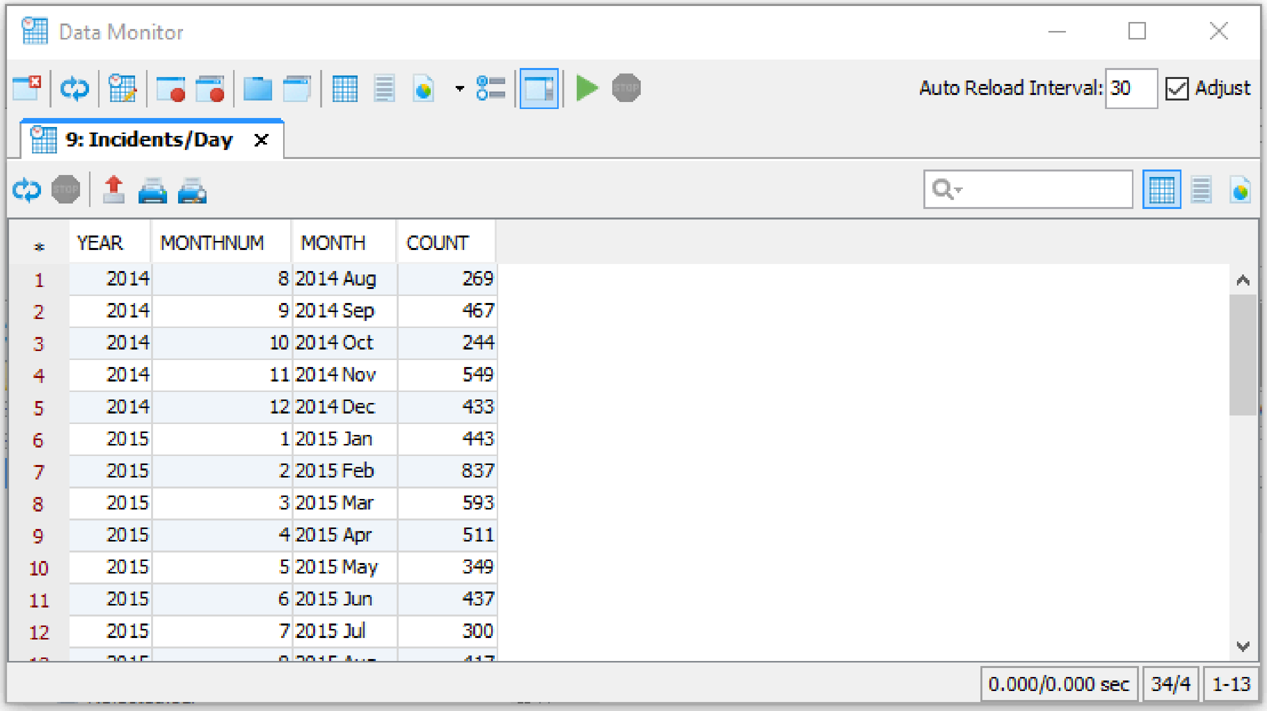
The SQL for this monitor adds a third column, named **NumRowsChange**. It utilizes the fact that DbVisualizer automatically creates variables for the columns in a monitor result set, holding the values from the previous execution. The **NumRowsChange** column is set to the value returned by the count(*) aggregate function for the current execution minus the value from the previous execution, held by the **NumRows** 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 NumRows,
       COUNT(*) - ${NumRows||count(*)}$ AS NumRowsChange
from ORDERS
```

14.2 Running a Monitored Query

The Monitor window, launched via the **Tools->Monitor** menu option, is where you active monitors and look at the results. The monitor tabs can be rearranged in the same way as all other tabs, pretty much any way you like. Please see Getting the Most Out of the GUI (see page 32) for details.

The monitor results can be viewed only as grids in DbVisualizer Free, while DbVisualizer Pro adds the capability to view them as charts or text.



Auto Reload Interval: 30 ☑ Adjust

	YEAR	MONTHNUM	MONTH	COUNT
1	2014	8	2014 Aug	269
2	2014	9	2014 Sep	467
3	2014	10	2014 Oct	244
4	2014	11	2014 Nov	549
5	2014	12	2014 Dec	433
6	2015	1	2015 Jan	443
7	2015	2	2015 Feb	837
8	2015	3	2015 Mar	593
9	2015	4	2015 Apr	511
10	2015	5	2015 May	349
11	2015	6	2015 Jun	437
12	2015	7	2015 Jul	300

0.000/0.000 sec 34/4 1-13

The Monitor window has toolbar at the top with an **Auto Reload Interval** field and a **Adjust** box. The **Auto Reload Interva**l field is used to control how often, in seconds, to execute the monitors when auto update is running. 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. Check the **Adjust** box and the Monitor feature will automatically increase the number of seconds so that all monitors will complete before next auto-update.

The rest of the window holds result areas for each monitored statement with the **Visible** attribute enabled. Each individual monitor result tab or window may also have a toolbar with controls that apply just to that result. The screenshot is from DbVisualizer Pro, 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

Toolbar Button	Description
Show as Tabs	Shows the results as collapsed tabs
Show as Windows	Shows the results as tiled tabs
Show Grids	Shows all results as grids
Show Text	Shows all results as text
Show Chart	Shows all results as graph in the selected chart type
Show/Hide Chart Legends	Toggle this to show/hide chart legends
Show/Hide Monitor Toolbars	Toggle this to show/hide toolbars for each monitor
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

In the Tool Properties dialog, you can enable **Show Monitor Window at Startup** and **Start Monitors Automatically**, in the **Monitor** category under the General tab.

15 Accessing Frequently Used Objects

When you work on many different tasks, it is important to easily find and use the data and scripts you need.

DbVisualizer helps you by keeping the tabs you use open between sessions and letting you organize references to objects and scripts.

15.1 Keeping Tabs Open Between Sessions

If you often work with the same objects and a few scripts, you can ensure that the Object View and SQL Commander tabs for these objects remain open between DbVisualizer sessions.

1. Open **Tools->Tool Properties**,
2. Select the **Tabs** category,
3. Enable one or both of **Preserve SQL Commander tabs between Sessions** and **Preserve Object View tabs between Sessions**,
4. Click **Apply** or **OK** to apply the new settings.

This feature is enabled by default for SQL Commander tabs but not for Object View tabs.

The content of the SQL Commander tabs is saved at regular intervals so when you restart DbVisualizer, the content is the same as where you left off.

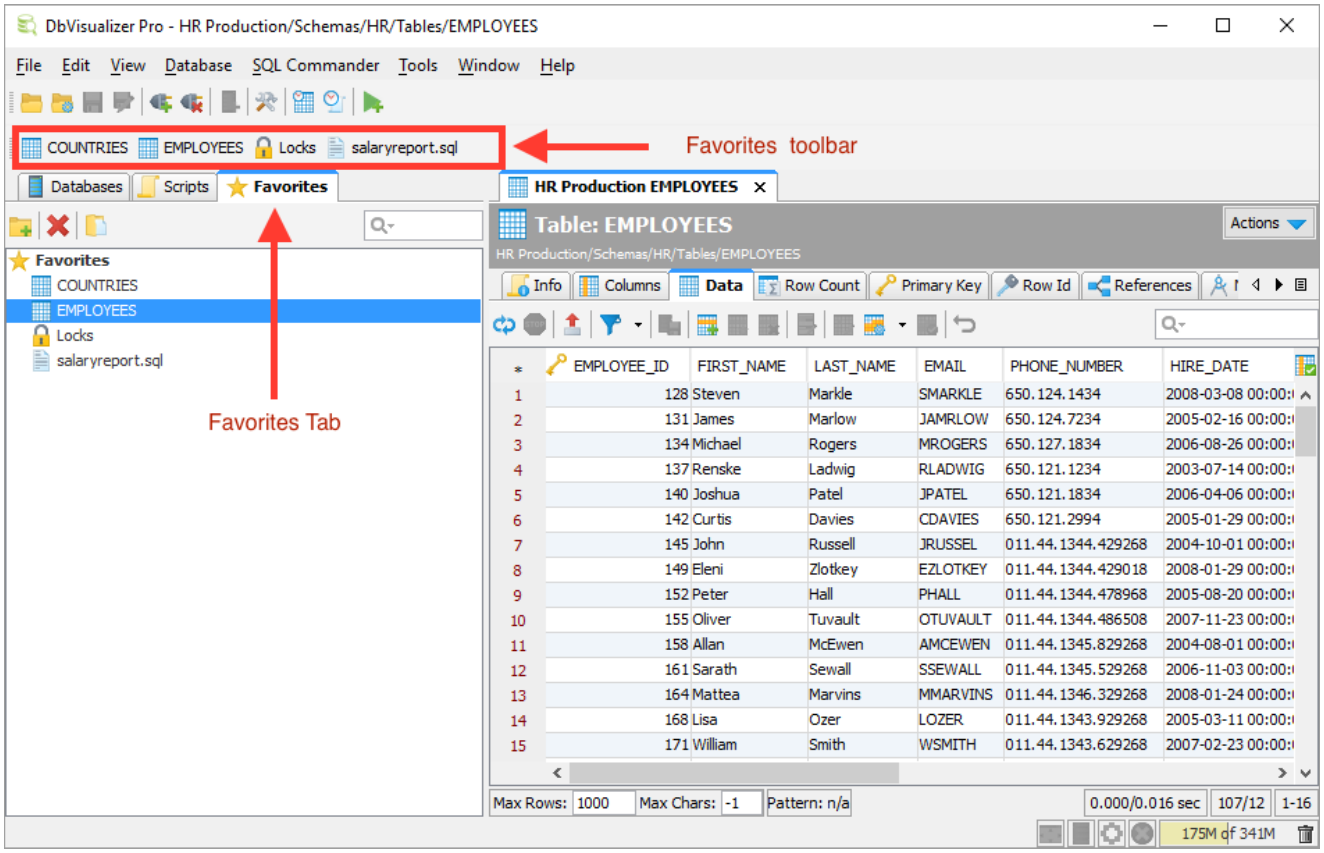
For Object View tabs, you can also enable **Preserve Object View tabs at Disconnect**. By default, Object View tabs for objects that belong to a connection are closed when it is disconnected.

15.2 Using Favorites

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

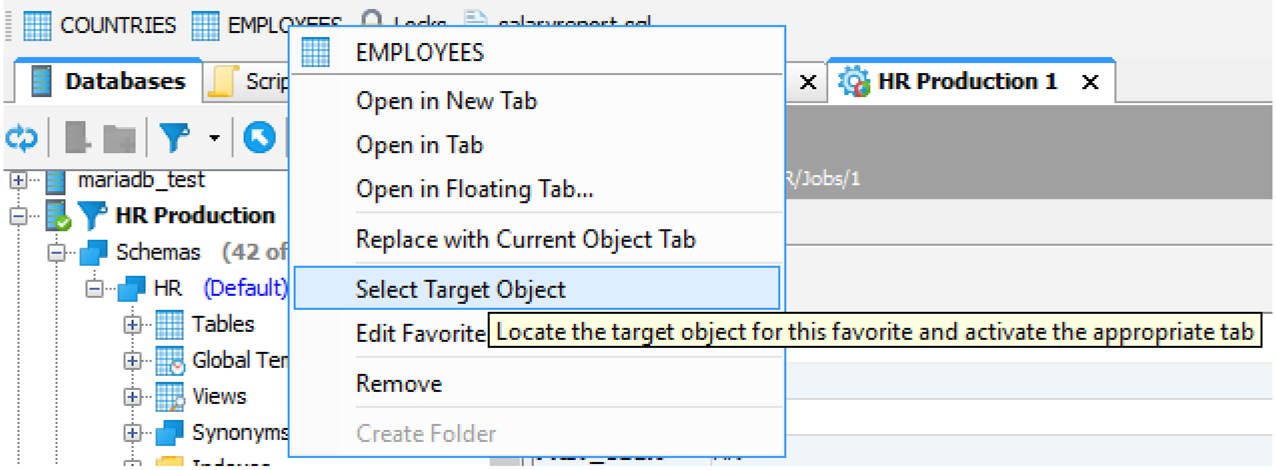
Navigating the **Databases** tab tree down to the object can be quite time consuming for database objects that you work with often. By adding these objects to the Favorites toolbar, you have one-click access to them instead . In addition to database objects, you can also add Bookmark (see page 187) script files to the Favorites toolbar.


DbVisualizer Pro - HR Production/Schemas/HR/Tables/EMPLOYEES
File Edit View Database SQL Commander Tools Window Help
COUNTRIES EMPLOYEES Locks salaryreport.sql
Favorites toolbar
Databases Scripts Favorites
Favorites
COUNTRIES
EMPLOYEES
Locks
salaryreport.sql
Favorites Tab
HR Production EMPLOYEES
Table: EMPLOYEES
Actions
HR Production/Schemas/HR/Tables/EMPLOYEES
Info Columns Data Row Count Primary Key Row Id References
Max Rows: 1000 Max Chars: -1 Pattern: n/a
0.000/0.016 sec 107/12 1-16
175M of 341M

*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	128	Steven	Markle	SMARKLE	650.124.1434	2008-03-08 00:00:
2	131	James	Marlow	JAMRLOW	650.124.7234	2005-02-16 00:00:
3	134	Michael	Rogers	MROGERS	650.127.1834	2006-08-26 00:00:
4	137	Renske	Ladwig	RLADWIG	650.121.1234	2003-07-14 00:00:
5	140	Joshua	Patel	JPATEL	650.121.1834	2006-04-06 00:00:
6	142	Curtis	Davies	CDAVIES	650.121.2994	2005-01-29 00:00:
7	145	John	Russell	JRUSSEL	011.44.1344.429268	2004-10-01 00:00:
8	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	2008-01-29 00:00:
9	152	Peter	Hall	PHALL	011.44.1344.478968	2005-08-20 00:00:
10	155	Oliver	Tuvault	OTUVAULT	011.44.1344.486508	2007-11-23 00:00:
11	158	Allan	McEwen	AMCEWEN	011.44.1345.829268	2004-08-01 00:00:
12	161	Sarath	Sewall	SSEWALL	011.44.1345.529268	2006-11-03 00:00:
13	164	Mattea	Marvins	MMARVINS	011.44.1346.329268	2008-01-24 00:00:
14	168	Lisa	Ozer	LOZER	011.44.1343.929268	2005-03-11 00:00:
15	171	William	Smith	WSMITH	011.44.1343.629268	2007-02-23 00:00:

When you click on a database object in the Favorites toolbar, the corresponding object is opened in an Object View tab. If you're not connected to the database the object belongs to, a connection is automatically established. Clicking on a Bookmark in the Favorites toolbar opens it in an SQL Commander tab.

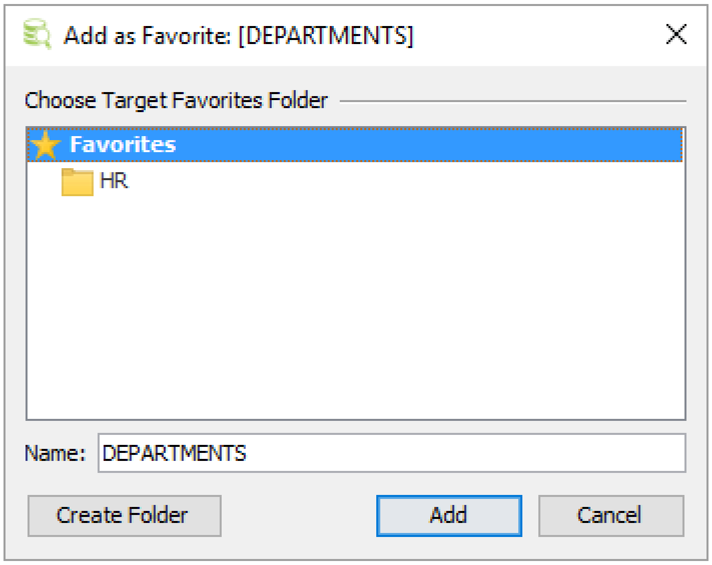
You can also easily find the corresponding object in the **Databases** tab or **Scripts** tab tree using the **Select Target Object** right-click menu item.



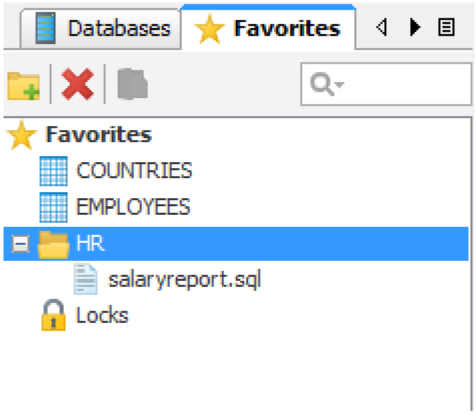
Locate the target object for this favorite and activate the appropriate tab

The easiest way to add an item to the Favorites toolbar is to select the item in the **Databases** tab tree or the **Script** tab tree, drag it with the mouse key depressed and drop it in the Favorites toolbar by releasing the mouse button. If you have created Favorite folders, you can also drop the item on a folder.

You can also use the **Add to Favorite** right-click menu operation for the database object or Bookmark. This opens a dialog where you can add the item, at the top level or in an existing or new Favorite folder.


Add as Favorite: [DEPARTMENTS]
Choose Target Favorites Folder
Favorites
HR
Name:
DEPARTMENTS
Create Folder
Add
Cancel

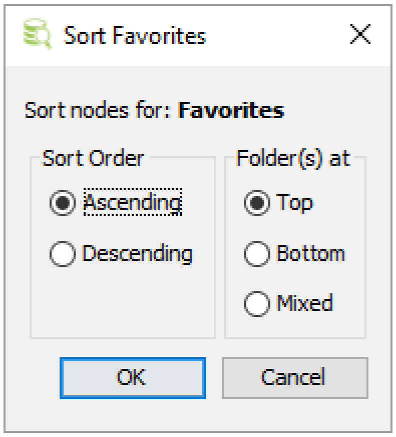
Use the **Favorites** tab in the navigation area to organize your favorites.



Here you can add folders and drag and drop entries between them. A favorite folder works as a drop-down menu in the Favorites toolbar. Double-click a favorite to open the target script file in the SQL Commander or database object in the Object View.

You can also delete and rename entries here. The right-click menu for an entry also contains entry-type dependent operations, such as executing a Bookmark and open a database object in a separate window.

To sort the favorites, select **Sort** from the right-click menu for any favorite. The sorting critera can be defined in the dialog that pops up.



Sort nodes for: **Favorites**

Sort Order
● Ascending
○ Descending

Folder(s) at
● Top
○ Bottom
○ Mixed

15.3 Using Scripts

A script is a file with one or more SQL statements that you can edit and execute in an SQL Commander tab. You can keep a script as an ordinary file anywhere in the file system and load it into an SQL Commander tab (see page 161) when needed, but managing it as a Bookmark (see page 187) under the Scripts tab is more convenient as it also keeps information about which connection you used it with last, among other things.

16 Delimited Identifiers and Qualifiers

Delimited Identifiers must generally be used for database object with names that contain special characters, reserved words or mixed case. Qualifiers are needed when referring to an object in a different schema/catalog than the current one. DbVisualizer uses delimited, qualified identifiers in all SQL it automatically executes in response to user interaction, such as when loading the Data tab for a table, dropping a stored procedure or getting metadata for a DDL statement.

For the features where DbVisualizer generates SQL that you can then execute, you can control if you want to use delimited identifiers and/or qualified object names. In the Properties tab for a connection, or in the Tool Properties dialog for a database type under the Database tab, you find the **Delimited Identifiers** and **Qualifiers** categories.

In the Delimited Identifiers category you can specify which delimiter characters to use, e.q. double-quotes or square brackets, and for which features to use them: **Scripting** (all SQL generating features), **Auto Completion /Query Builder**, **Export**, and **Actions.**

In the Qualifiers category you can specify for which features to use them, and also if column names should be qualified with the table name for the Scripting and the Auto Completion/Query Builder features. For database types that supports fully qualified named (both database and schema qualifiers), you can enable fully qualified names in the References/Navigator Graphs and for the Auto Completion/Query Builder features.

17 Handling Transactions

Database transactions are intended to make sure operations performed simultaneously do not cause data integrity problems.

By default, DbVisualizer commits all changes immediately but you can disable this to have full control over the transactions. You can also set an appropriate transaction isolation level for your connections.

17.1 Changing the Auto Commit Setting

Auto Commit means that each SQL statement successfully executed in an SQL Commander is committed automatically, permanently changing the database. This is the default for a connection, but you can change it at different levels.

- Changing Auto-Commit for a Database Type (see page 280)
- Changing Auto-Commit for a Connection (see page 280)
- Changing Auto-Commit for an SQL Commander tab (see page 280)
- Changing Auto-Commit for a Statement Block (see page 281)

17.1.1 Changing Auto-Commit for a Database Type

1. Open **Tools->Tool Properties**,
2. Select the **Database** tab,
3. Expand the node for the database type, e.g. **Oracle,**
4. Select the **Transaction** category,
5. Uncheck the **Auto Commit** checkbox.

17.1.2 Changing Auto-Commit for a Connection

1. Double-click the connection node in the **Databases** tab tree to open an **Object View** tab,
2. Select the **Properties** tab,
3. Select the **Transaction** category under the node for the database type, e.g. **Oracle**,
4. Uncheck the **Auto Commit** checkbox.

17.1.3 Changing Auto-Commit for an SQL Commander tab

- Use the **SQL Commander->Turn On/Off Auto Commit** toggle item, or,
- Use the corresponding toggle button to the right in the SQL Commander toolbar.

17.1.4 Changing Auto-Commit for a Statement Block

You can use the `@set autocommit` command in a script to enable or disable auto commit for different blocks:

```
@set autocommit off;
INSERT INTO SCOTT.EMP (EMPNO, ENAME, JOB, MGR, HIREDATE) VALUES(61, 'Boo', 2, 1, '2013-06-20');
INSERT INTO SCOTT.EMP (EMPNO, ENAME, JOB, MGR, HIREDATE) VALUES(62, 'Hoo', 2, 1, '2013-07-01');
INSERT INTO SCOTT.EMP (EMPNO, ENAME, JOB, MGR, HIREDATE) VALUES(63, 'Zoo', 2, 1, '2013-07-04');
@set autocommit on;
```

17.2 Setting Transaction Isolation

When you connect to a database that is concurrently modified by other users and processes, the Transaction Isolation Level specifies how changes made by others will affect you and how your changes will affect others.

To set the Transaction Isolation Level for a connection,

1. Double-click the connection node in the **Databases** tree to open an **Object View** tab for it,
2. Select the **Properties** tab,
3. Select the **Transaction** category,
4. Pick an appropriate **Transaction Isolation Level** from the drop-down list.

The levels that are supported depends on the database you are connecting to. Many databases though support the levels as defined in the JDBC specification. The JDBC levels are defined below.

Level	Transactions	Dirty Reads	Non-Repeatable Reads	Phantom Reads
`TRANSACTION_NONE`	Not supported	N/A	N/A	N/A
`TRANSACTION_READ_COMMITTED`	Supported	Prevented	Allowed	Allowed
`TRANSACTION_READ_UNCOMMITTED`	Supported	Allowed	Allowed	Allowed
`TRANSACTION_REPEATABLE_READ`	Supported	Prevented	Prevented	Allowed
`TRANSACTION_SERIALIZABLE`	Supported	Prevented	Prevented	Prevented

A dirty read occurs when transaction A reading a value before transaction B has made permanent, i.e. before it has been committed.

A non-repeatable read occurs when transaction A retrieves a row, transaction B subsequently updates the row, and transaction A later retrieves the same row again. Transaction A retrieves the same row twice but sees different data.

A phantom read occurs when transaction A retrieves a set of rows satisfying a given condition, transaction B subsequently inserts or updates a row such that the row now meets the condition in transaction A, and transaction A later repeats the conditional retrieval. Transaction A now sees an additional row. This row is referred to as a phantom.

Please see the database documentation for the description of transaction isolation values supported by your database.

18 Database Connection Options

The database connection is a central concept in DbVisualizer.

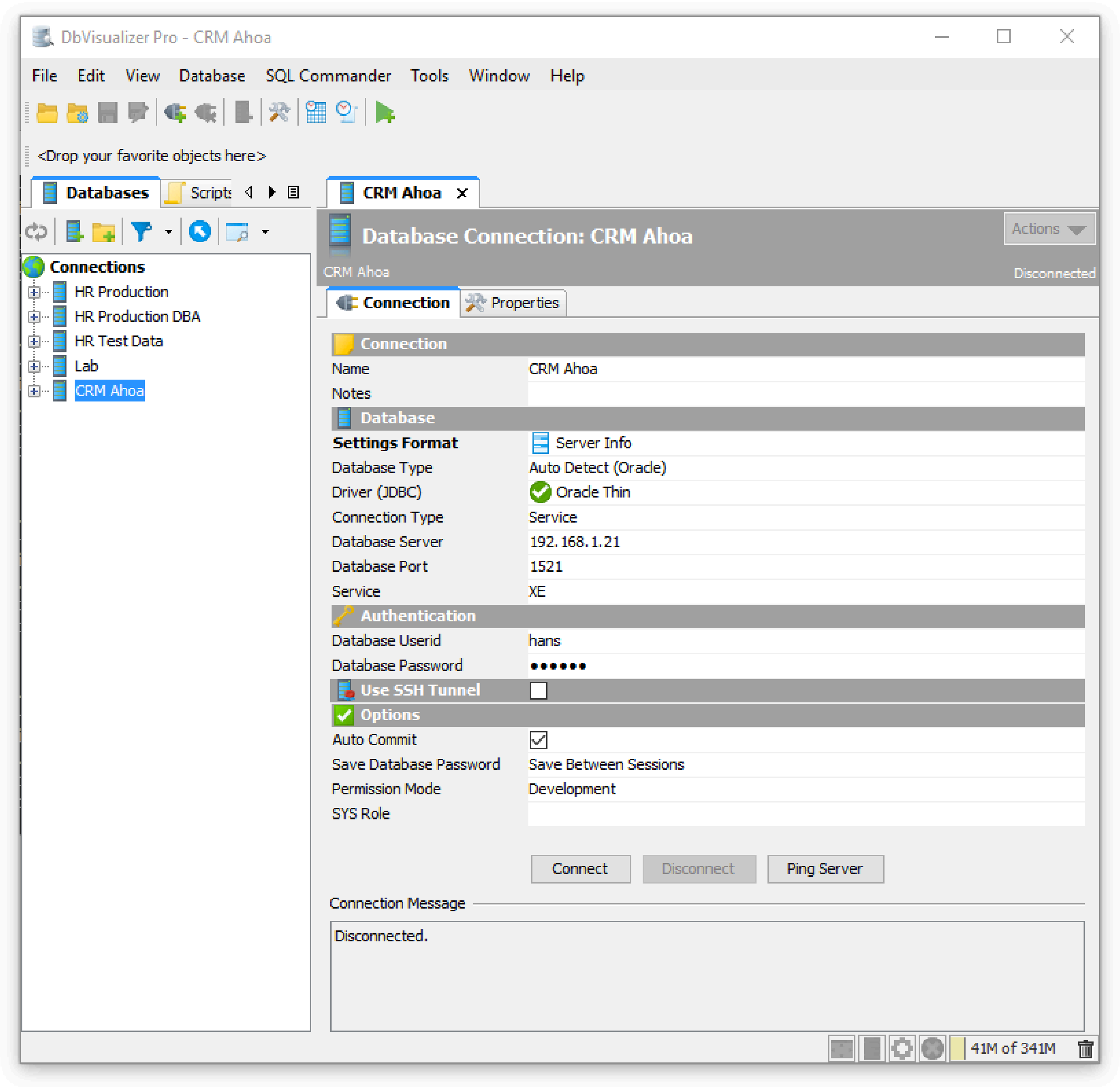
Learn how to configure it to your needs, how to use special features like connecting through an SSH tunnel, using Single-Sign-On, organizing the connections, and much more.

18.1 Setting Up a Connection Manually

To access a database with DbVisualizer, you must first create and setup a Database Connection. The easiest way to set up a connection is to use the Connection Wizard (see page 17), but you can also do it manually.

18.1.1 Setting Up a Connection Manually

1. Create a new connection from **Database->Create Database Connection** and click **No Wizard** when prompted. An **Object View** tab for the new connection is opened,



Database Connection: CRM Ahoa

CRM Ahoa — Disconnected

Connection	
Name	CRM Ahoa
Notes	
Database	
Settings Format	Server Info
Database Type	Auto Detect (Oracle)
Driver (JDBC)	✓ Oracle Thin
Connection Type	Service
Database Server	192.168.1.21
Database Port	1521
Service	XE
Authentication	
Database Userid	hans
Database Password	••••••
Use SSH Tunnel	□
Options	
Auto Commit	☑
Save Database Password	Save Between Sessions
Permission Mode	Development
SYS Role	

Connection Message

Disconnected.

2. Enter a name for the connection in the **Name** field, and optionally enter a description of the connection in the **Notes** field,
3. Leave the **Database Type** as **Auto Detect,**
4. Select an installed JDBC driver (marked with a green checkmark) from the Driver (JDBC) list (see Installing a JDBC Driver (see page 430) for how to install a JDBC driver manually),
5. Enter information about the database server in the remaining fields (see below for details),
6. Verify that a network connection can be established to the specified address and port by clicking the **Ping Server** button,
7. If Ping Server shows that the server can be reached, click **Connect** to actually connect to the database server.

⚠

See Fixing Connection Issues (see page 422) for some tips if you have problems connecting to the database.

Alternatively, you can set the **Settings Format** to **Database URL** (this is the only choice for some custom JDBC drivers). This replaces the fields for information about the database server with a single **Database URL** field, where you can enter the JDBC URL.

The information about the database server that needs to be entered depends on the which JDBC driver you use. For most drivers, you need to specify:

Field	Description
Database Server	The IP address or DNS name for the server where the database runs.
Database Port	The TCP/IP port used by the database.
Database Userid	The database user account name. Enter `(null)` to not send an account name.
Database Password	The database user account password. Enter `(null)` to not send a password.

For some database such as Oracle, you may use a TNS name (see page 301) instead of specifying the server and port. Other drivers may add more fields that are driver specific.

You may also optionally specify SSH tunneling information (see page 298) and Options, such as:

Option	Description
Auto Commit	Check if you want to enable auto commit in the SQL Commander by default for the connection.
Save Database Password	Check if you want the password to be saved (encrypted) during the session, between sessions, or cleared when you disconnect.
Permission Mode	One of **Development**, **Test** or or **Production** to select which set of Permissions (see page 285) to use.

Additional options are available for some JDBC drivers, such as Authentication Method (see page 303) for the SQL Server jTDS driver.

See the Configuring Connection Properties (see page 285) page for related topics.

18.2 Configuring Connection Properties

- Database Profile (see page 287)
- Driver Properties (see page 288)

- Invoke Java methods in the JDBC driver (see page 289)

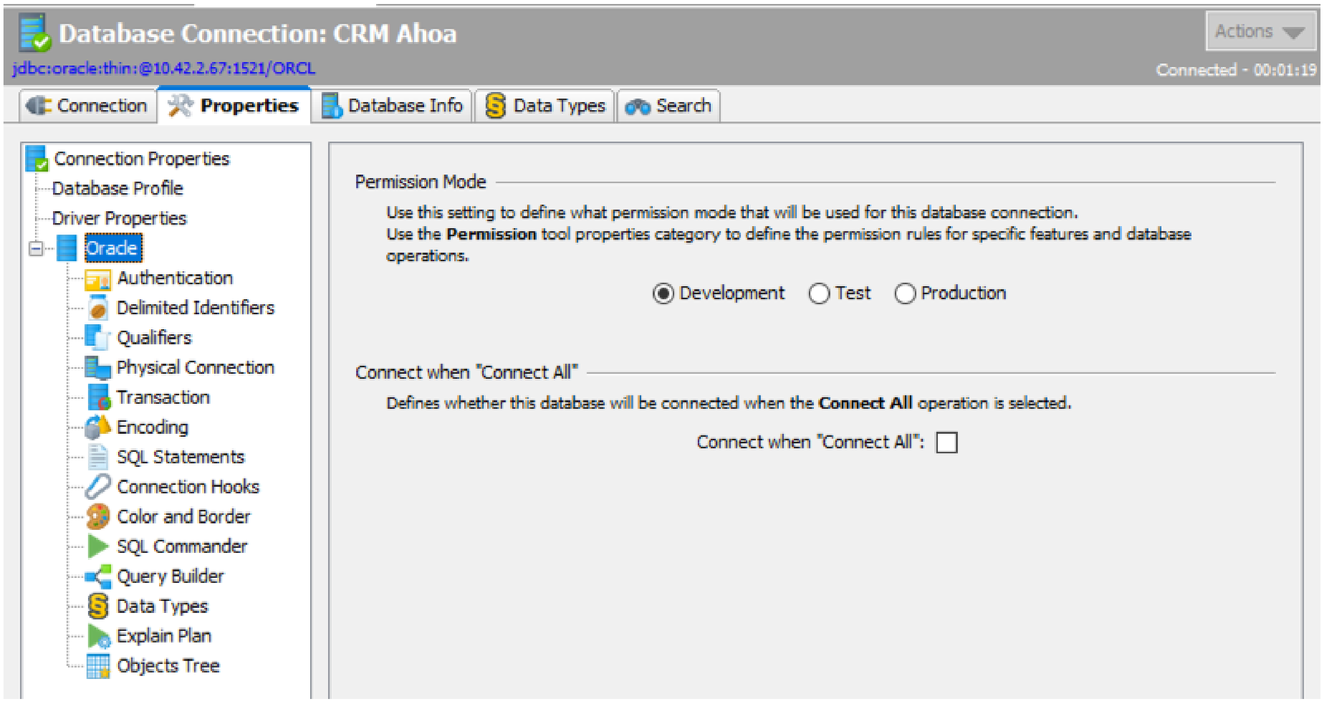
In addition to the basic connection information (see page 17) in the **Connection** tab, there is also a collection of connection properties. Which properties are available depends on the **Database Type** selected for the database connection in the Connection tab. Some database types have more properties than others. Which edition of DbVisualizer you use also affects which connection properties are available.

Properties for a connection can be defined at two different levels:

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

All supported database types (Oracle, Informix, SQL Server, 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 Connection Properties are available in the **Properties** sub tab in the the database connection's **Object View** tab.



Permission Mode

Use this setting to define what permission mode that will be used for this database connection.
Use the **Permission** tool properties category to define the permission rules for specific features and database operations.

● Development ○ Test ○ Production

Connect when "Connect All"

Defines whether this database will be connected when the **Connect All** operation is selected.

Connect when "Connect All": □

The **Properties** tab is organized basically the same way as the **Tool Properties** window. The main difference is that the list contains only the categories that are applicable to this database connection. Briefly, the categories are:

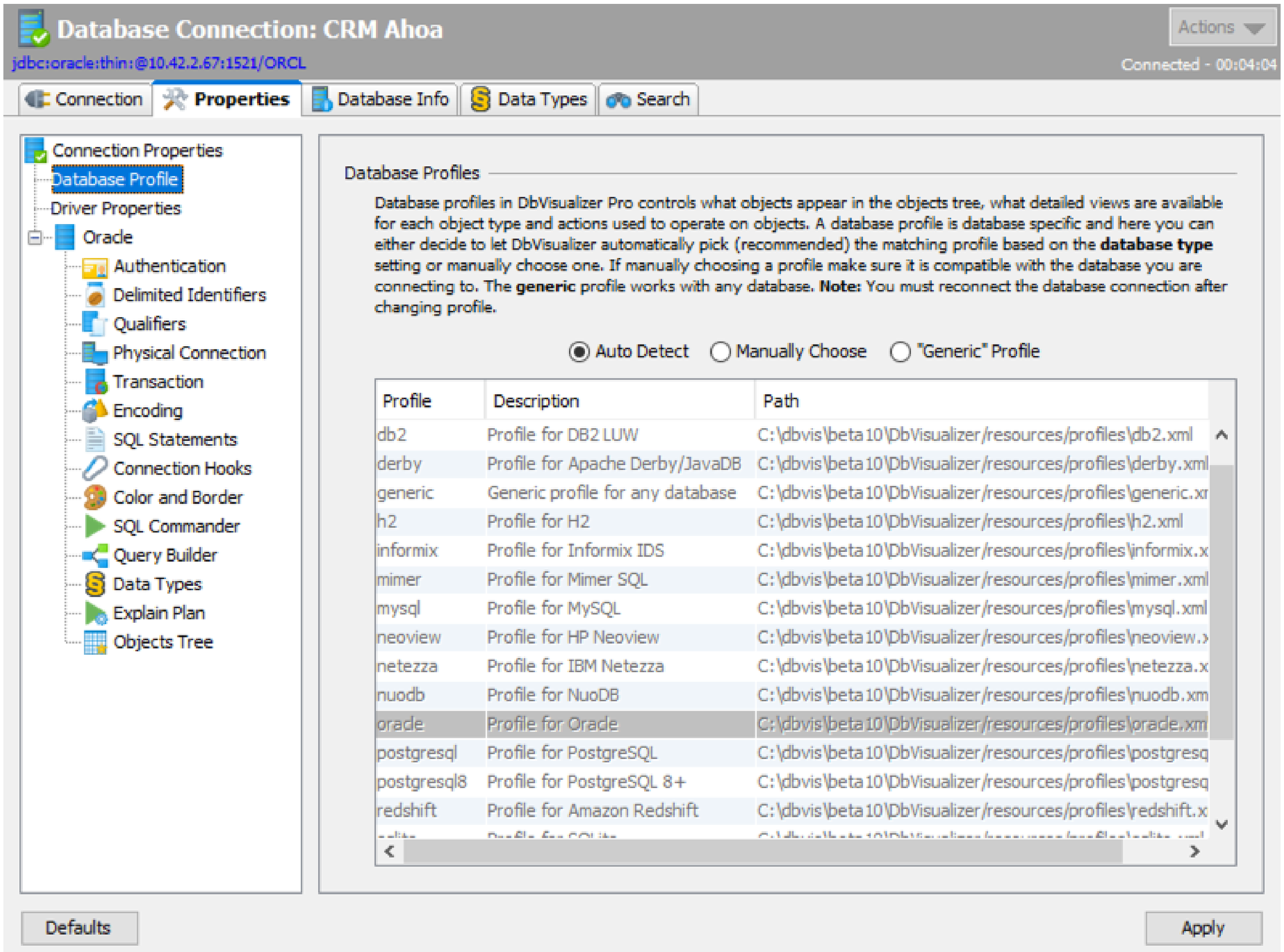
- Database Profile
- Driver Properties
- Oracle (The current Database Type)
    - Authentication
    - Delimited Identifiers
    - Qualifiers
    - Physical Connection
    - Transaction
    - Encoding
    - SQL Statements
    - Connection Hooks
    - Color and Border
    - SQL Commander
    - Query Builder
    - Data Types
    - Explain Plan
    - Objects Tree

The **Database Profile** and **Driver Properties** categories are available only in the **Properties** tab and not in **Tool Properties**. The page explains the **Database Profile** and **Driver Properties** categories, while the other categories are described in pages that describe feature the property applies to.

Additional categories may appear in the connection properties depending on the type of database. An example is the category for Explain Plan for the databases where this feature is supported.

18.2.1 Database Profile

The Database Profile (see page 328) 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.



Database Profiles

Database profiles in DbVisualizer Pro controls what objects appear in the objects tree, what detailed views are available for each object type and actions used to operate on objects. A database profile is database specific and here you can either decide to let DbVisualizer automatically pick (recommended) the matching profile based on the **database type** setting or manually choose one. If manually choosing a profile make sure it is compatible with the database you are connecting to. The **generic** profile works with any database. **Note:** You must reconnect the database connection after changing profile.

● Auto Detect ○ Manually Choose ○ "Generic" Profile

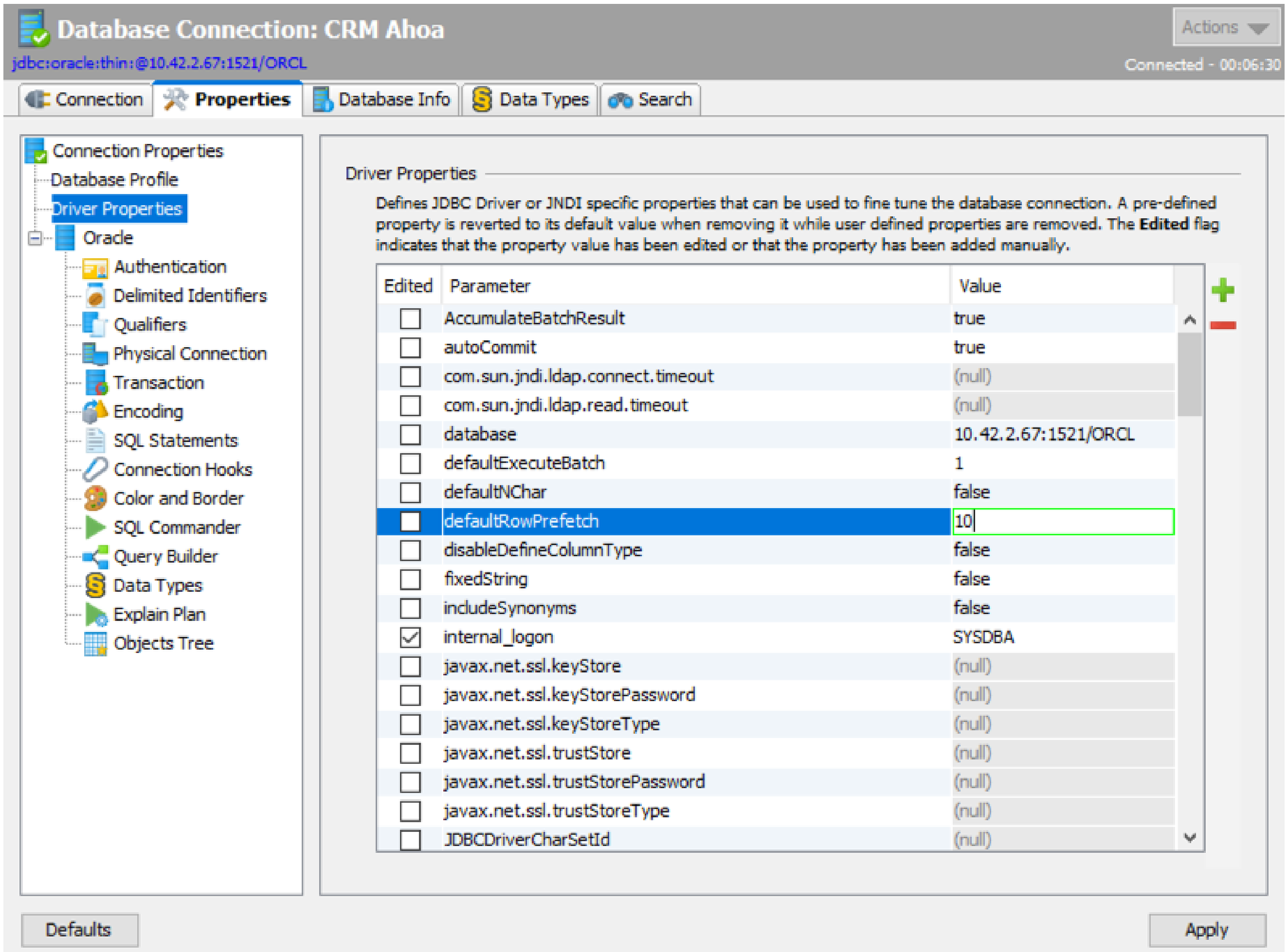
Profile	Description	Path
db2	Profile for DB2 LUW	C:\dbvis\beta10\DbVisualizer/resources/profiles\db2.xml
derby	Profile for Apache Derby/JavaDB	C:\dbvis\beta10\DbVisualizer/resources/profiles\derby.xml
generic	Generic profile for any database	C:\dbvis\beta10\DbVisualizer/resources/profiles\generic.xr
h2	Profile for H2	C:\dbvis\beta10\DbVisualizer/resources/profiles\h2.xml
informix	Profile for Informix IDS	C:\dbvis\beta10\DbVisualizer/resources/profiles\informix.x
mimer	Profile for Mimer SQL	C:\dbvis\beta10\DbVisualizer/resources/profiles\mimer.xml
mysql	Profile for MySQL	C:\dbvis\beta10\DbVisualizer/resources/profiles\mysql.xml
neoview	Profile for HP Neoview	C:\dbvis\beta10\DbVisualizer/resources/profiles\neoview.x
netezza	Profile for IBM Netezza	C:\dbvis\beta10\DbVisualizer/resources/profiles\netezza.x
nuodb	Profile for NuoDB	C:\dbvis\beta10\DbVisualizer/resources/profiles\nuodb.xm
oracle	Profile for Oracle	C:\dbvis\beta10\DbVisualizer/resources/profiles\oracle.xm
postgresql	Profile for PostgreSQL	C:\dbvis\beta10\DbVisualizer/resources/profiles\postgresq
postgresql8	Profile for PostgreSQL 8+	C:\dbvis\beta10\DbVisualizer/resources/profiles\postgresq
redshift	Profile for Amazon Redshift	C:\dbvis\beta10\DbVisualizer/resources/profiles\redshift.x
sqlite	Profile for SQLite	C:\dbvis\beta10\DbVisualizer/resources/profiles\sqlite.xml

The way DbVisualizer auto detects a profile is based on the setting of **Database Type** in the **Connection** tab. If the **Database Type** is also set to **Auto Detect**, DbVisualizer first detects the database type based on the JDBC information, and then detects the profile based on the database type.

There is rarely a reason to use another setting than **Auto Detect**, but if you manually choose a database profile, this choice will be saved between invocations of DbVisualizer.

18.2.2 Driver Properties

The **Driver Properties** category is used to fine tune a JDBC driver before the database connection is established.



Defines JDBC Driver or JNDI specific properties that can be used to fine tune the database connection. A pre-defined property is reverted to its default value when removing it while user defined properties are removed. The **Edited** flag indicates that the property value has been edited or that the property has been added manually.

Edited	Parameter	Value
□	AccumulateBatchResult	true
□	autoCommit	true
□	com.sun.jndi.ldap.connect.timeout	(null)
□	com.sun.jndi.ldap.read.timeout	(null)
□	database	10.42.2.67:1521/ORCL
□	defaultExecuteBatch	1
□	defaultNChar	false
□	defaultRowPrefetch	10
□	disableDefineColumnType	false
□	fixedString	false
□	includeSynonyms	false
☑	internal_logon	SYSDBA
□	javax.net.ssl.keyStore	(null)
□	javax.net.ssl.keyStorePassword	(null)
□	javax.net.ssl.keyStoreType	(null)
□	javax.net.ssl.trustStore	(null)
□	javax.net.ssl.trustStorePassword	(null)
□	javax.net.ssl.trustStoreType	(null)
□	JDBCDriverCharSetId	(null)

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 to the right of the list.

18.2.3 Invoke Java methods in the JDBC driver

In addition to driver properties it is also possible to invoke low level Java methods in the JDBC driver classes, java.sql.Connection (http://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html) and java.sql.Statement (http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html). These are edited in the driver properties (see page 288) list.

You may run single argument methods taking one of **String.class**, **Integer.class**, or **Boolean.class** Java types as argument. The method name should be specified as **Parameter** which must be fully qualified with the all lowercase class name. The argument is specified in the **Value** field in the driver properties list.

Here are a few examples:

Parameter	Value
`java.sql.connection.setReadOnly`	`true`
`java.sql.statement.setFetchSize`	`1000`

java.sql.connection methods are invoked just after a physical connection with the database has been established. **java.sql.statement** methods are invoked just after a statement has been created and ready for execution.

18.3 Copying an Existing Connection

To copy an existing connection and use as the basis for a new:

1. Select the original connection node in the **Databases** tab tree,
2. Use the **Database->Duplicate Database Connection** to create a copy,

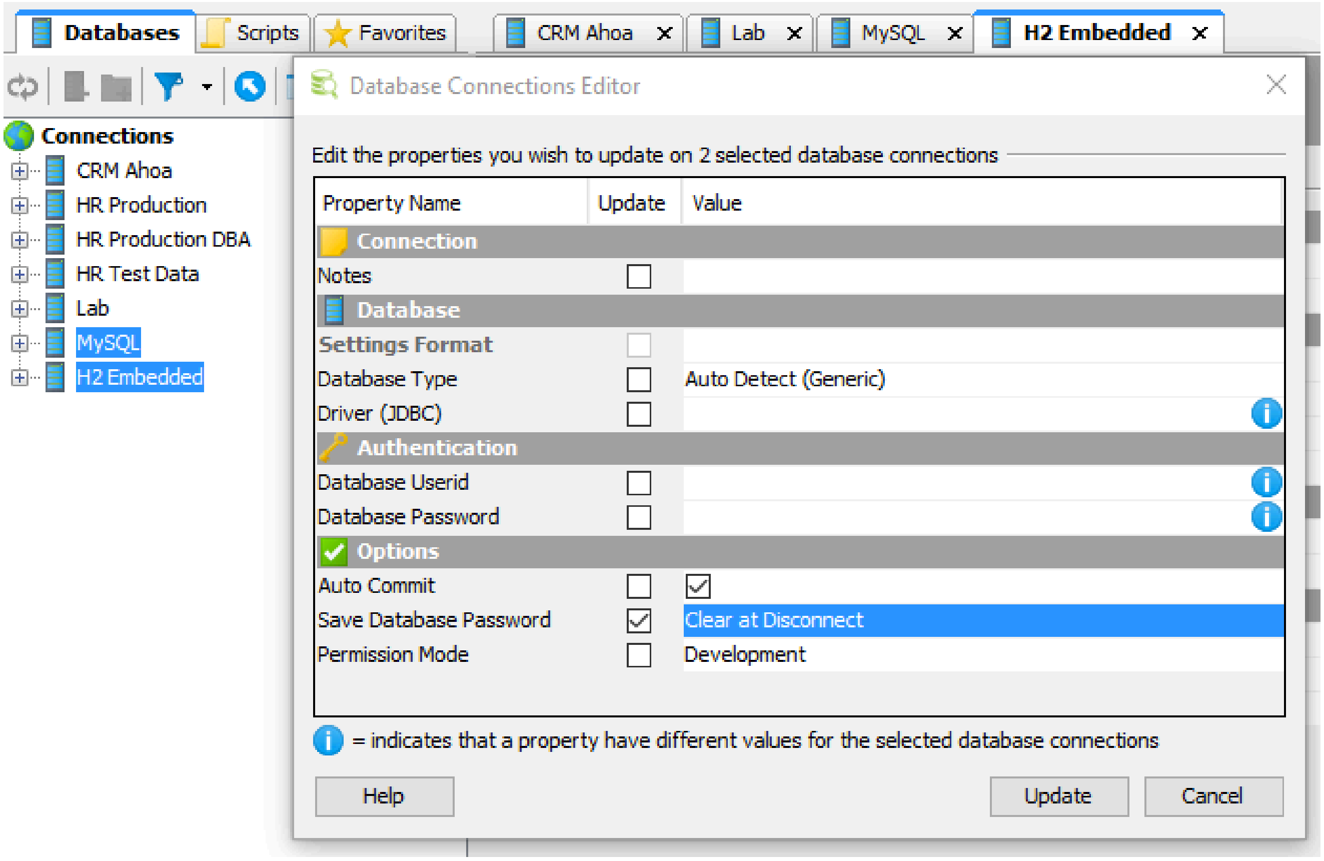
The Object View tab for the copied connection is opened where you can make adjustments.

18.4 Edit Multiple Database Connections

> ℹ **Only in DbVisualizer Pro**
>
> This feature is only available in the DbVisualizer Pro edition.

There are situation when you may need to change one or several connection settings for multiple database connections at the same time. Instead of doing this one connection at a time, the connections editor come in handy.

To open the connections editor select multiple database connections in the Databases tab, right-click and chose **Edit Multiple Connections**.



Database Connections Editor

Edit the properties you wish to update on 2 selected database connections

Property Name	Update	Value
Connection		
Notes	□	
Database		
Settings Format	□	
Database Type	□	Auto Detect (Generic)
Driver (JDBC)	□	ⓘ
Authentication		
Database Userid	□	ⓘ
Database Password	□	ⓘ
Options		
Auto Commit	□	☑
Save Database Password	☑	Clear at Disconnect
Permission Mode	□	Development

ⓘ = indicates that a property have different values for the selected database connections

The connections editor will show only those fields that are commonly available for the database connections being edited.

The window has the following columns:

- **Update**: Indicates if the value is edited. You may manually uncheck this to indicate the setting should not be updated
- **Value**: Enter the new value here. Once the field is being edited its background will change and the Update box is checked.

18.4.1 Changing the database driver

The **Driver (JDBC)** setting is the main setting that controls what properties are available for a database connection. Once Driver (JDBC) has been edited its **Update** checkbox cannot be unchecked.

18.5 Removing a Connection

To remove a connection,

1. Select the connection node in the Database tab tree,
2. Use the **Database->Remove Database Connection** menu choice to remove the connection.

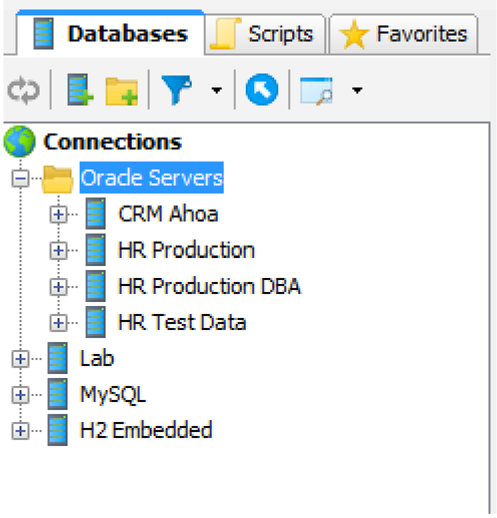
18.6 Organizing Connections in Folders

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

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 drag and drop folders and other objects to move them to a new location in the **Databases** tab tree.



18.7 Rearranging Connections and Folders

You can sort the nodes under the **Connections** node or a folder node in the Databases tab:

1. Select the Connections node or a folder node,
2. Open the Sort dialog from the right-click menu,
3. Select the sort order and where to place nestled folder,
4. Click **OK**.

To move an individual connection node or folder node:

1. Select one or more nodes,
2. Drag the node(s) to the new location,

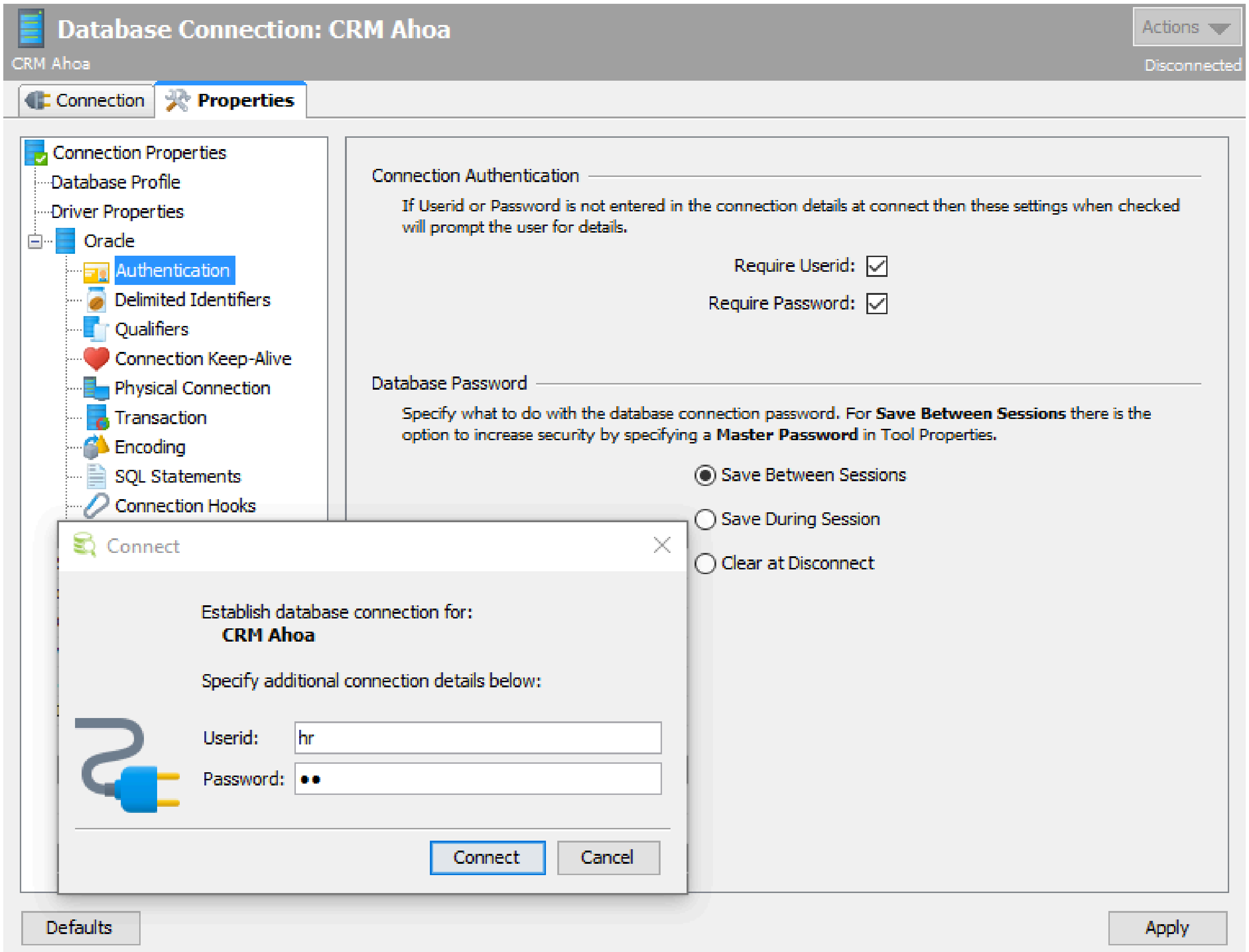
3. Drop the node.

18.8 Setting Common Authentication Options

DbVisualizer lets you handle authentication in a manner that suits your balance between security and convenience.

Userid and password information is generally information that should be handled with great care. By default, DbVisualizer saves both the Database Userid and Database Password (encrypted) for each database connection. The default for SSH (see page 298) is to save the SSH Userid but not the SSH Password (or Key Pass-phrase). You can change this behavior to fit your preferences. You specify how to handle the Database Userid and Password in the **Authentication** category of the **Properties** tab. The same options are available for the SSH Userid and Password in the **Database Connection/SSH Settings** category in the **General** tab of the **Tool Properties** window.

The **Require Userid** and **Require Password** properties can be enabled to tell DbVisualizer to automatically prompt for userid and/or password when a connection is to be established if they are not specified for the connection. The following dialog is displayed if requiring both userid and password.



Connection Authentication

If Userid or Password is not entered in the connection details at connect then these settings when checked will prompt the user for details.

Require Userid: ☑

Require Password: ☑

Database Password

Specify what to do with the database connection password. For **Save Between Sessions** there is the option to increase security by specifying a **Master Password** in Tool Properties.

● Save Between Sessions

○ Save During Session

○ Clear at Disconnect

Establish database connection for:
CRM Ahoa

Specify additional connection details below:

Userid: hr

Password: ••

Connect Cancel

Since a password may need to be handled with great care, you can also specify for how long it should be saved, if at all. If **Clear at Disconnect** is selected, DbVisualizer ensures that the Password field is cleared as soon as the connection is terminated. With **Save During Session**, it is cleared when you shut down DbVisualizer. To keep the password between sessions, select **Save Between Sessions**. If you use Save Between Sessions, we recommend that you also set a Master Password (see page 294).

18.9 Setting a Master Password

When you use **Save Between Sessions** for database passwords and SSH passwords or pass-phrases, they are by default encrypted using a static key. The same is true for the Proxy password if you have specified one. For better security, you can specify a Master Password that is then used instead of the static key to encrypt all passwords and pass-phrases. This way, only you know the information needed to decrypt the data. The algorithms used for encryption with a Master Password are also more advanced, minimizing the risk that the data can be decrypted by brute force.

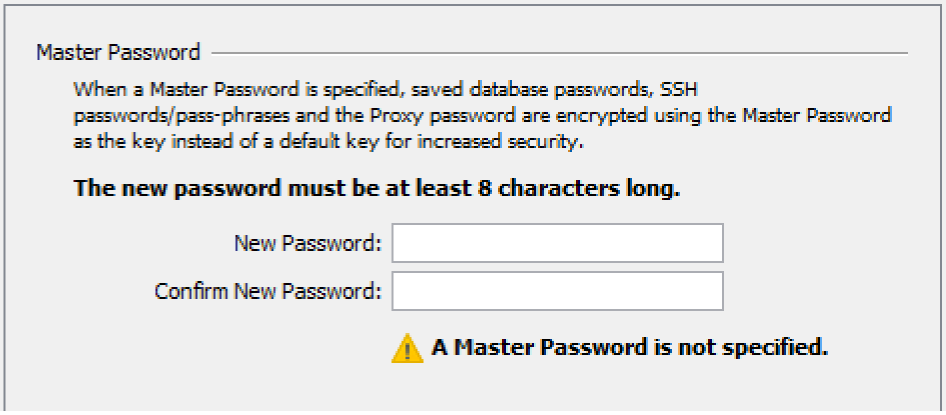
Using a Master Password does, however, mean that if you forget it, there is no way to retrieve it and therefore no way to decrypt the saved passwords. It also means that the encrypted passwords cannot be read by a DbVisualizer version earlier than 9.2.

> ⊖ If you forget the Master Password, it cannot be recovered. The only way forward is to reset the Master Password, which also clears all passwords encrypted with it.
>
> Passwords encrypted with a Master Passwords cannot be used in DbVisualizer version earlier than 9.2 . If you set a Master Password in 9.2 and then use an earlier version, you will get "invalid password" errors when trying to connect with a saved password. You must enter the database or SSH password again in the earlier version, or go back to using DbVisualizer version 9.2 or later.

18.9.1 Specifying a Master Password

To use a Master Password for encoding of passwords saved between sessions:

1. Open **Tools->Tool Properties** and select the **General/Master Passwor**d category,
2. Enter a password matching the described rules in both the **New Password** and **Confirm New Password** fields,
3. Click **Apply** and then confirm that you want to do this after reading the warning about what it implies.



Master Password

When a Master Password is specified, saved database passwords, SSH passwords/pass-phrases and the Proxy password are encrypted using the Master Password as the key instead of a default key for increased security.

The new password must be at least 8 characters long.

New Password:	
Confirm New Password:	

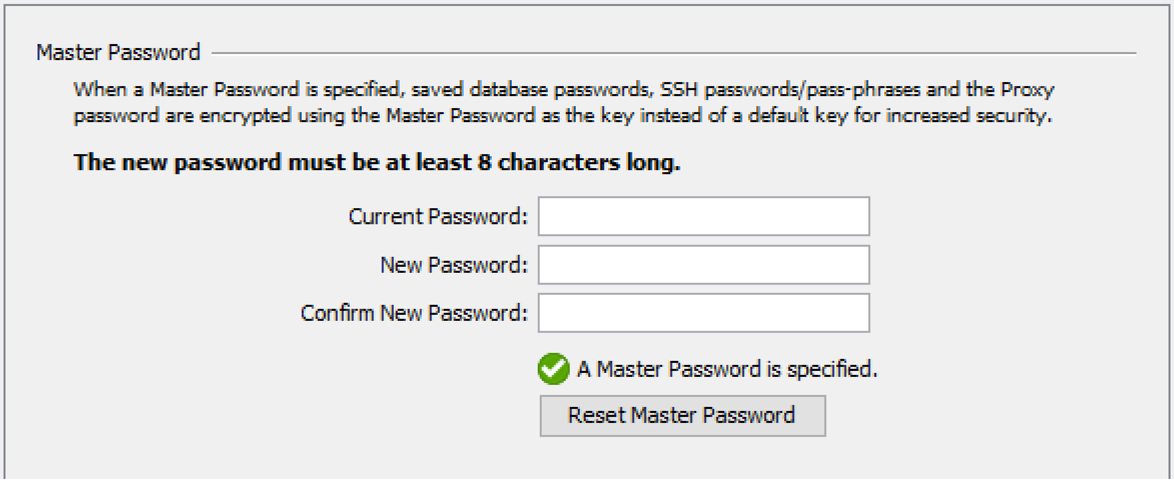
⚠ **A Master Password is not specified.**

The passwords for all connections with **Save Between Sessions** chosen for the password are now encrypted with the Master Password. The same goes for the SSH passwords/pass-phrases if you have selected to have them saved between sessions, as well as the proxy password, if any.

18.9.2 Changing a Master Password

If you want to change the Master Password:

1. Open **Tools->Tool Properties** and select the **General/Master Password** category,
2. Enter the current password in the **Current Password** field and the new password in both the **New Password** and **Confirm New Password** fields,
3. Click Apply.



Master Password

When a Master Password is specified, saved database passwords, SSH passwords/pass-phrases and the Proxy password are encrypted using the Master Password as the key instead of a default key for increased security.

The new password must be at least 8 characters long.

Current Password:

New Password:

Confirm New Password:

✓ A Master Password is specified.

The saved passwords are then decrypted with the current Master Password and re-encrypted with the new.

18.9.3 Resetting the Master Password

If you have forgotten the Master Password, or simply no longer want to use one, you can reset it:

1. Open **Tools->Tool Properties** and select the **General/Master Password** category,
2. Click the **Reset Master Password** button and confirm that you want to do this.

> Note that all passwords encoded with the Master Password are then immediately cleared and there is no way to recover them.

18.9.4 Connecting with a Master Password specified

When you have a Master Password specified, you will be prompted to enter it the first time within a DbVisualizer session that you need to connect with a saved password. From then on, you can make other connections with saved passwords without being prompted until you restart DbVisualizer.

18.9.5 Manually Requesting the Master Password for New Connections

You have two options to manually require being prompted for the Master Password again after entering it once within a DbVisualizer session:

1. Select **Database->Require Master Password at Next Connect**,
2. Open **Tools->Tool Properties**, select the **General/Database Connection** category and enable **Require Master Password after All Connections Closed**.

18.9.6 Showing the Encrypted Password in Cleartext

When you have specified a Master Password, you can view the saved database password or SSH password/ pass-phrase in cleartext.

1. Right-click on the password field label and select **Show Password**,
2. Enter the Master Password when prompted.

18.9.7 Declaring a Master Password Rule

A Master Password must have at least eight characters of any kind by default, but you can declare your own rule using a regular expression in an installation configuration file:

1. Open the *DBVIS-HOME/resources/dbvis-custom.prefs* file,
2. Enter a regular expression as the value of the `dbvis.-MasterPasswordRule` property,
3. Enter a description of the rule for showing the user in Tool Properties as the value of the `dbvis.-MasterPasswordRuleDescr` property.

The regular expression for the default rule is `.{8,}`. It is easy to change the number in this expression to any number you want. There are regular expressions that can describe pretty much any rule you can come up with. For instance, this rule requires at least nine characters, with at least one symbol, one digit, one uppercase character, and one lowercase character:

```
(?=.{9,})(?=.*?[^\\w\\s])(?=.*?[0-9])(?=.*?[A-Z]).*?[a-z].*
```

If you cannot adopt these examples to your own policy, you can search the Internet for other examples of " regular expressions for password validations".

18.10 Using Connection Keep-Alive

Databases can be configured to terminate sessions that have been idle for some time, and networks often does the same with TCP/IP connection. The **Connection Keep-Alive** feature helps preventing connections to be closed due to time-outs of this kind by periodically executing a simple SELECT statement.

> Network connections may be terminated for other reasons than a time-out in the database or at the network layer, e.g due to a restart of the database or a network element. The Connection Keep-Alive feature does not help in those cases. Also note that connections that are busy, e.g. actively used to run a script, are not "pinged". If a SELECT statement or stored procedure takes a very long time to complete, it is therefore possible that a time-out happens at the network level. In this case, the network configuration must be tuned to handle long running statements without timing out.

To enable Keep-Alive for a connection:

1. Open the **Object View** tab for the connection,
2. Open its **Properties** tab,
3. Select the **Physical Connection** category,
4. Modify or enter a simple SQL statement in the **Validation and Keep-Alive SQL** field, if needed (see note below),
5. Enable **Connection Keep-Alive** and optionally change the idle time interval,
6. Click the **Apply** button.

> The SELECT statement used for Connection Keep-Alive can also be specified in the properties pane. For supported databases, it is set to a SELECT statement that has been verified to work for the database type but for connections that use the Generic profile, you must specify a valid SELECT statement in order for this feature to work. For many databases, `SELECT 1` or `SELECT 1 FROM aSmallTable WHERE 1 = 0` should work.

18.11 Using an SSH Tunnel

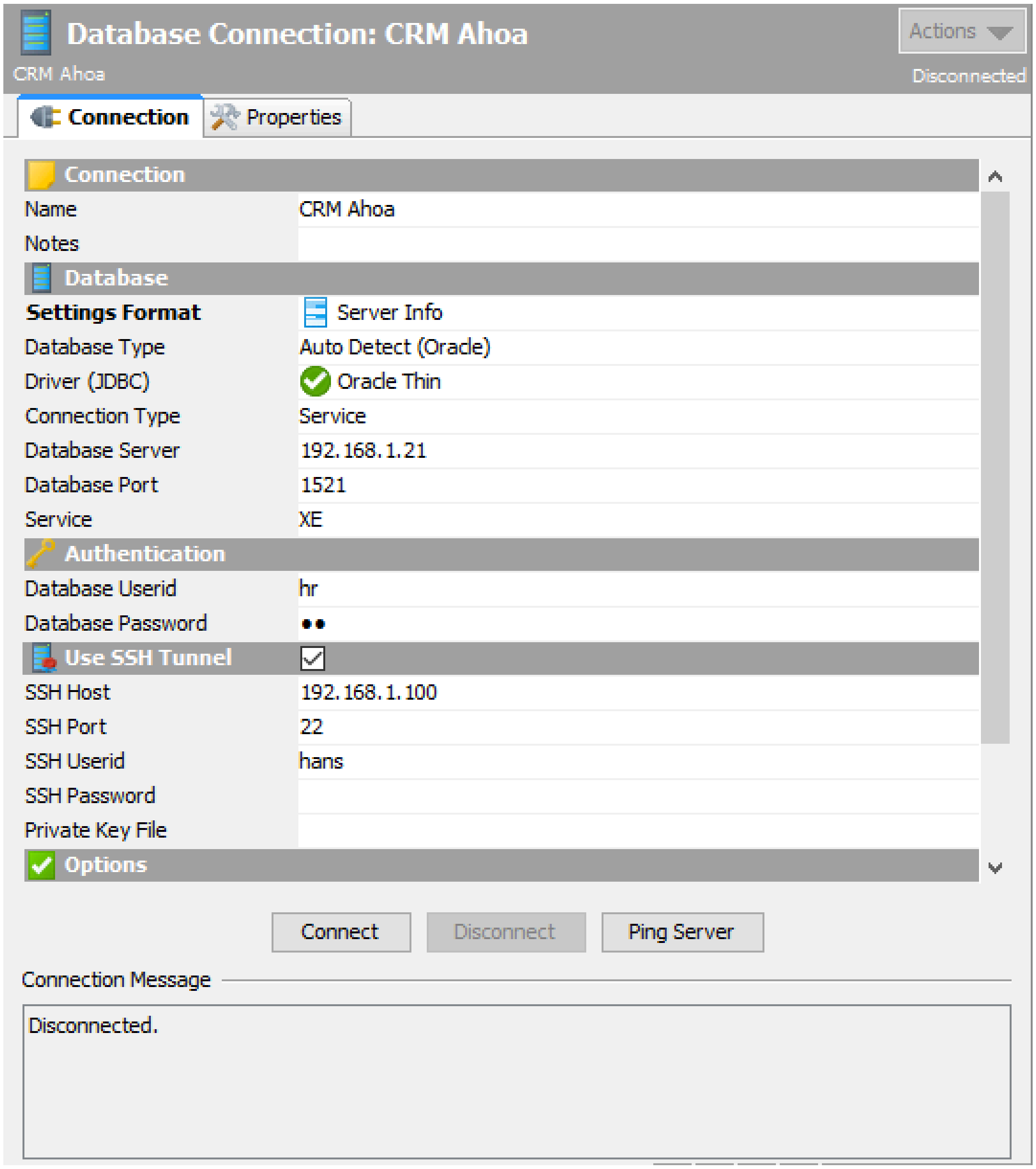
> **Only in DbVisualizer Pro**
>
> This feature is only available in the DbVisualizer Pro edition.

A database that sits behind a firewall cannot be accessed directly from a client on the other side of the firewall, but it can often be accessed through an SSH tunnel. The firewall must be configured to accept SSH connections and you also need to have an account on the SSH host for this to work.

If you need to access a database that can only be accessed vi an SSH tunnel, you need to specify additional information in the Use SSH Tunnel area of the Connection tab.

> ⚠ This area is only shown when the **Server Info** settings format is selected, and only for databases identified by at least a **Database Server** and a **Database Port** (i.e. not for embedded databases or when using the TNS Connections Type for an Oracle database, or similar).

Enable SSH tunneling by clicking on the checkbox. When it is enabled, five additional fields are shown.



Database Connection: CRM Ahoa

CRM Ahoa

Disconnected

Connection	
Name	CRM Ahoa
Notes	
Database	
Settings Format	Server Info
Database Type	Auto Detect (Oracle)
Driver (JDBC)	✓ Oracle Thin
Connection Type	Service
Database Server	192.168.1.21
Database Port	1521
Service	XE
Authentication	
Database Userid	hr
Database Password	●●
Use SSH Tunnel	☑
SSH Host	192.168.1.100
SSH Port	22
SSH Userid	hans
SSH Password	
Private Key File	
Options	

Connection Message

Disconnected.

The **SSH Host** is the name or IP address for the host accepting SSH connections. The SSH Host is typically the same as the Database Server. Enter the port for SSH connections in the **SSH Port** field. The default value is 22.

You may also enter the userid and password for your SSH host account in the **SSH Userid** and **SSH Password** fields, but see Setting Common Authentication Options (see page 293) for other options. Alternatively, you can enter the path to a private key file (using either the RSA or DSA algorithms) in the Private Key File field. The SSH Password field is then replaced by a Key Passphrase field where you can enter the passphrase if the private key is protected with one.

When SSH tunneling is enabled, a tunnel is established when you connect to the database and the connection is then made through the tunnel by constructing a JDBC URL that uses information from both the Connection and Use SSH Tunnel sections.

If you're familiar with using the `ssh` command to set up a tunnel manually, you may be interested in more details . The tunnel corresponds to the tunnel you would set up with the ssh command like this:

```
ssh -p <SSHPort> -L<LocalPort>:<DatabaseServer>:<DatabasePort> <SSHUserid>@<
SSHHost>
```

where the placeholders correspond to the fields in the Connect and Use SSH Tunnel sections, except for < LocalPort> which is any available port, determined at connect time.

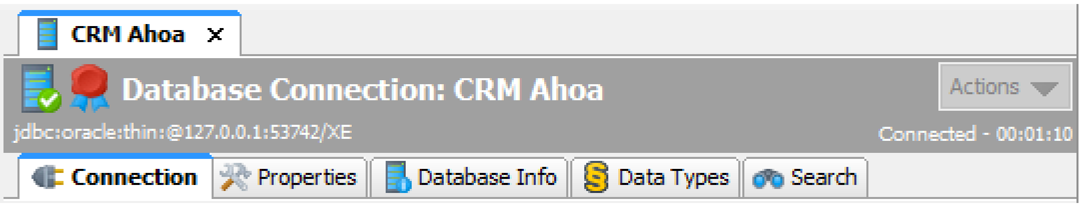
⚠ Note that when using an SSH tunnel, the **Database Server** is evaluated on the SSH host. If the database server is running on the SSH host, you can therefore set **Database Server** to `localhost` in case the database only accepts local connections.

The JDBC URL is constructed using 127.0.0.1 as the Database Server portion and <LocalPort> as the Database Port portion, e.g. like this for the Oracle Thin driver when <LocalPort> is 50538:

```
jdbc:oracle:thin@127.0.0.1:50538/XE
```

In other words, the JDBC driver connects to the SSH tunnel's local port, which then forwards all communication to the database server.

The URL that is used for the connection is shown at the top of the **Object View** tab for the database connection when a connection is established, along with a certificate icon if the connection is made through an SSH tunnel.



Database Connection: CRM Ahoa

jdbc:oracle:thin:@127.0.0.1:53742/XE

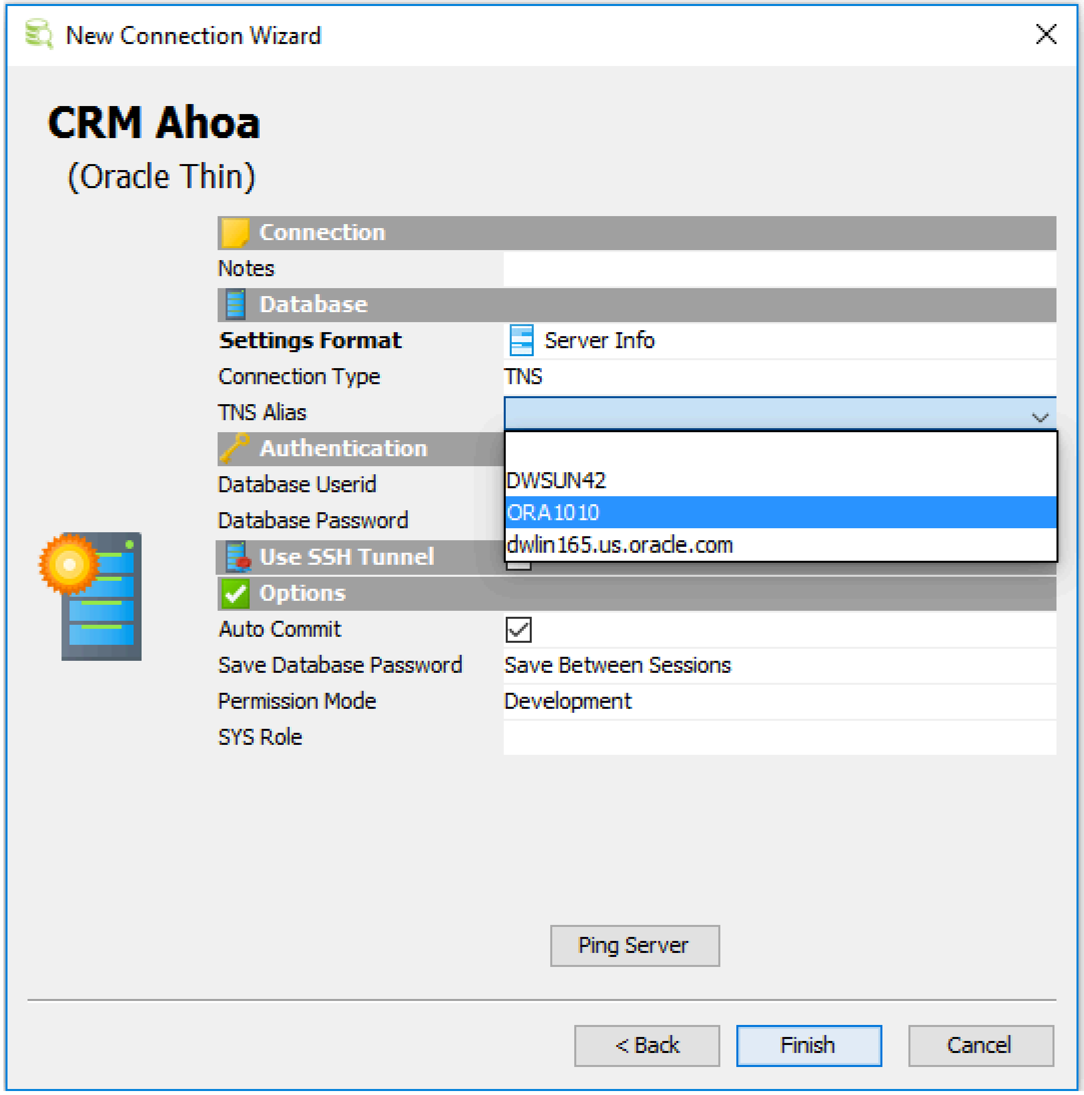
Connected - 00:01:10

If you use the SSH Tunnel feature, you may also want to configure a few things in **Tools->Tool Properties**. In the **Database Connection/SSH Settings** category under the General tab, you can specify an SSH Keep-Alive Interval to minimize the risk that the tunnel is disconnected due to inactivity, and an SSH Known Hosts File so you don't have to accept connections to known SSH hosts every time you connect.

18.12 Using Oracle TNS Names

All information for connecting to an Oracle database may be stored in a *tnsnames.ora* file, with each database instance defined by a TNS alias. If you want to create a connection in DbVisualizer that uses the information from this file, you must first tell DbVisualizer where it is stored by setting either the `TNS_ADMIN` environment variable to the path of the folder holding the file, or making sure it is located in the *ORACLE_HOME/network/admin* folder and that the `ORACLE_HOME` environment variable is set.

When this configuration is done, you can select **TNS** from the **Connection Type** list for the Oracle connection and then pick the TNS alias from a list of all aliases found in the file.



New Connection Wizard

CRM Ahoa

(Oracle Thin)

Connection	
Notes	
Database	
Settings Format	Server Info
Connection Type	TNS
TNS Alias	
Authentication	
Database Userid	
Database Password	
Use SSH Tunnel	
Options	
Auto Commit	☑
Save Database Password	Save Between Sessions
Permission Mode	Development
SYS Role	

DWSUN42
ORA1010
dwlin165.us.oracle.com

On macOS, environment variables set in `.bash_profile` or similar are not available to applications started via Spotlight or by double-clicking an app icon. The `launchctl` command can be used to set these environment variables instead, for instance from the `.bash_profile` script. This thread discusses an alternative solutions for Yosemite and El Capitan: http://stackoverflow.com/questions/25385934/setting-environment-variables-via-launchd-conf-no-longer-works-in-os-x-yosemite

18.13 Changing an Oracle Password

If your Oracle password has expired, you get a message about this in the Connection Message area for the database connection when you try to connect. For Oracle 12c or later, you can change the password from within DbVisualizer if you use the Oracle Thin JDBC driver:

1. Open the Object View tab for the connection if it is not already open,
2. Check the **Change Password** checkbox in the Authentication area,
3. Enter the new password in the **New Password** field,
4. Click Connect.

The connection is then established and the password is changed. The **Database Password** field is updated with the new password automatically.

You can also change the password as described at any time, even if it hasn't expired.

18.14 Using SQL Server Single-Sign-On or Windows Authentication

With Microsoft SQL Server, you can either let the database server or a Windows domain server handle the authentication. The database server handles it by default using the database user and password you enter for the connection.

To let a Windows domain server handle the authentication instead, you must use the **SQL Server (jTDS)** JDBC driver (bundled with DbVisualizer).

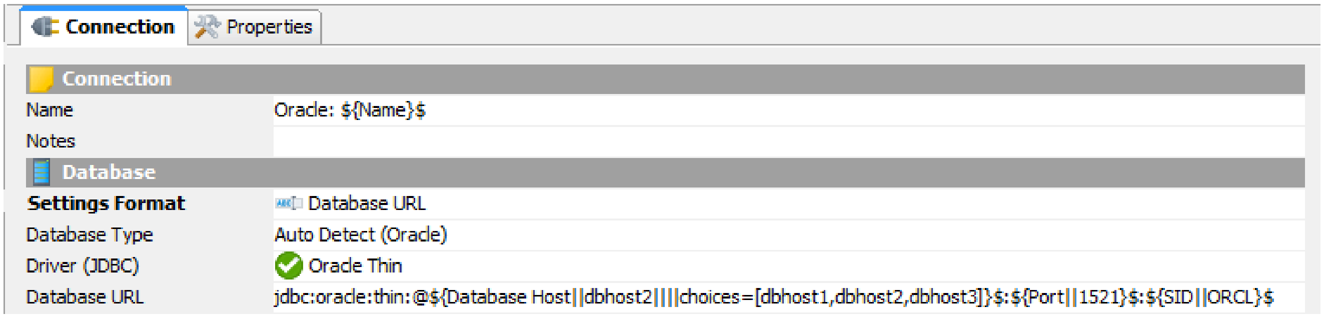
If you run DbVisualizer on a Windows OS client in the same domain as the SQL Server database, leave the **Database User** and **Database Password** fields in the **Connection** tab empty. This is also known as Single-Sign On (SSO).

> ⚠ SSO only works if you installed DbVisualizer using an installer, not if you used an installation archive, because the installer also installs the DLL files needed for SSO.

If you run DbVisualizer on another OS in a network with a Windows domain server, enter the **Domain** name in the **Options** area in the **Connection** tab. You must also enter the domain user and and password in the **Database User** and **Database Password** fields. The driver then authenticates with the domain server and then uses those credentials to log in to the database server.

18.15 Using Variables in Connection Fields

Variables can be used in some of the **Connection** tab fields. You can use variables in the **Name**, **Userid** and **Password** (both Database and SSH) fields with the **Server Info** settings format, or in the **Database URL** field when using this settings format. This can be a 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, described in more detail in the Using DbVisualizer Variables (see page 227) page.



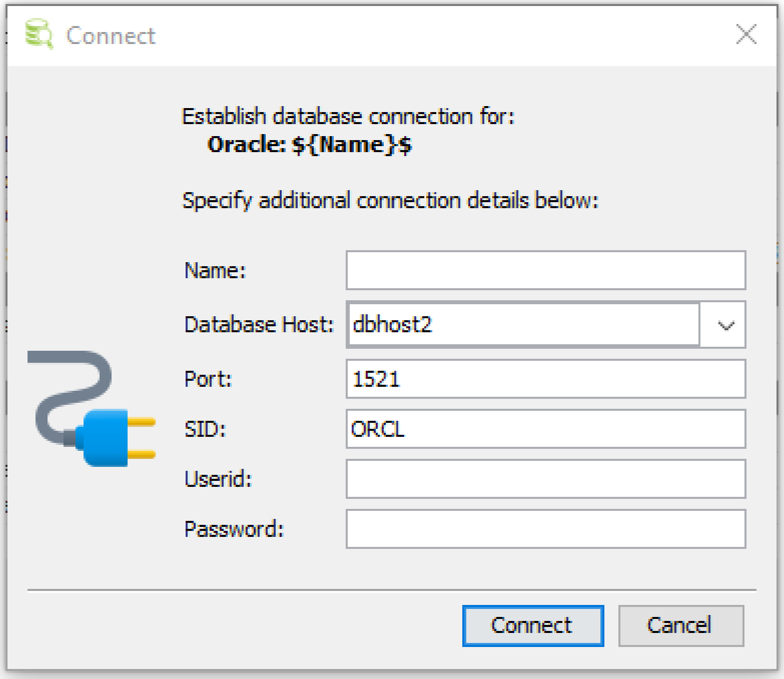
Connection	
Name	Oracle: ${Name}$
Notes	
Database	
Settings Format	Database URL
Database Type	Auto Detect (Oracle)
Driver (JDBC)	✓ Oracle Thin
Database URL	jdbc:oracle:thin:@${Database Host\|\|dbhost2\|\|\|\|choices=[dbhost1,dbhost2,dbhost3]}$:${Port\|\|1521}$:${SID\|\|ORCL}$

The following variables appear in the figure:

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

All of these variables define a default value after the "||" delimiter, except for the `${Name}$` variable, which has 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.



Establish database connection for:
Oracle: ${Name}$

Specify additional connection details below:

Name:
Database Host: dbhost2
Port: 1521
SID: ORCL
Userid:
Password:

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.

18.16 Automatically Connecting at Startup

If you want to automatically connect to a database when you start DbVisualizer:

1. Open the **Properties** sub tab in the connections Object View tab,
2. Select the database type category in the tree, e.g. the node named Oracle for an Oracle connection,
3. Enable **Connect when "Connect All"**,
4. Open **Tools->Tool Properties** and select the Database Connection category under the General tab,
5. Enable **Run "Connect All" at Startup**.

If you enable Connect when "Connect All" but do not enable Run "Connect All" at Startup, you can instead use the **Database->Connect All** main menu choice to manually connect all connections marked this way.

18.17 Executing SQL at Connect and Disconnect

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.

You can enter the SQL you want to execute in the Properties tab for the connection, in the **Connection Hooks** category.

18.18 Using a Single Shared Physical Connection

By default, DbVisualizer uses multiple physical connections to a database. Each SQL Commander tab is allocated its own connection. Other processes that update the database, such as saving grid edits or importing data to a table, also use their own connections. Finally most read-only operations, such as navigating the database objects tree, use a separate shared connection. This is normally the most efficient way to access the database, but in certain circumstances it is important to instead use one single shared physical connection for all operations. Some examples are:

- Only one session per account is allowed in the target database,
- Locking issues when modifying the same table in the Data tab and in an SQL Commander (when a pending transaction locks the whole table)
- When using one-time passwords, new physical connections cannot be established without prompting for a new password.

For situations like these, you can force DbVisualizer to use a single shared physical connection.

18.18.1 Selecting the Single Shared Physical Connection Mode

To use a single shared physical connection:

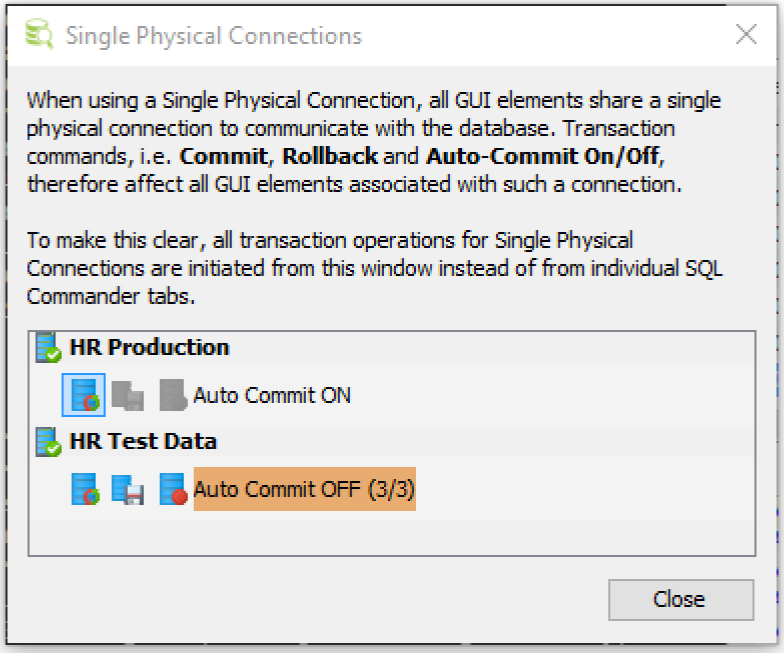
1. Open the Object View tab for the connection node,
2. Select the **Properties** tab,
3. Select the **Physical Connection** category and enable **Use a Single Shared Physical Connection**.

18.18.2 Data Manipulation with a Single Shared Physical Connection

Executing a script in an SQL Commander tab, using an Action, editing a table in a grid and importing data to a table are all operations that (potentially) modify data in the database. When a single shared physical connection is used, only one such operation may be performed at a time. If you try to start an operation like this while another one is already being processed, a dialog will pop up asking you to try again later.

18.18.3 Transaction Handling with a Single Shared Physical Connection

If you have **Auto-Commit** disabled with Single Shared Physical Connection enabled, commits or rollbacks done in one part of the GUI affect changes done in any other part of the GUI. For instance, if you have executed UPDATE or INSERT statements in an SQL Commander tab and then edit a table in its Data tab and commit those changes, you are also committing the changes made by the UPDATE or INSERT statements. To make this clear, all GUI controls for transaction handling for shared physical connections are shown in a separate Single Physical Connections window.



When using a Single Physical Connection, all GUI elements share a single physical connection to communicate with the database. Transaction commands, i.e. **Commit**, **Rollback** and **Auto-Commit On/Off**, therefore affect all GUI elements associated with such a connection.

To make this clear, all transaction operations for Single Physical Connections are initiated from this window instead of from individual SQL Commander tabs.

HR Production

Auto Commit ON

HR Test Data

Auto Commit OFF (3/3)

This window pops up when you connect to a database with Single Shared Physical Connection enabled, or when clicking any of the transaction control buttons in an SQL Commander tab for such a database. You can also click the corresponding button in the DbVisualizer status bar to bring it up. From this window, you can enable or disable Auto-Commit and manually commit or rollback a pending transaction.

You also get prompted to commit, rollback or continue working within the same transaction every time an operation results in data changes. Before potentially making lots of changes, you get prompted to enable Auto-Commit, since making lots of changes (e.g. importing lots of data) may fill up redo logs if running with Auto-Commit disabled.

18.19 JDBC-ODBC Bridge Driver Alternatives

Java has included a JDBC/ODBC Bridge driver as a transitional solution for accessing ODBC data sources, but it has always been considered a very limited driver and the recommendation has always been to use a pure JDBC driver instead. Starting with Java 8, the bridge driver is no longer provided.

For most databases, you can find JDBC drivers from the database vendor or a third party. Try searching the net for the name of your database plus "JDBC driver".

If you cannot find a JDBC driver for a database that can be accessed via ODBC, you find a few alternatives below. Note that we have have no relationship with any of the organizations behind these driver and have not thoroughly tested any of the drivers with DbVisualizer. In other words, please make sure that the driver works for you before committing to one.

> ℹ For any technical assistance setting up or using these drivers, please contact respective author.

18.19.1 The UCanAccess Driver for MS Access

This is an Open Source driver specifically for Microsoft Access databases, not for ODBC data sources in general. You can download it here:

http://ucanaccess.sourceforge.net/site.html

To use it, you also need to download the following Open Source JAR files:

jackcess-2.0.8.jar from
http://sourceforge.net/projects/jackcess/files/

commons-lang-2.6-bin.zip (unzip to find JAR file) from
http://commons.apache.org/proper/commons-lang/download_lang.cgi

commons-logging-1.2-bin.zip (unzip to find JAR file) from
http://commons.apache.org/proper/commons-logging/download_logging.cgi

hsqldb-2.3.2.zip (unzip to find JAR file) from
http://sourceforge.net/projects/hsqldb/files/

apache-log4j-2.1-bin.zip (unzip to find JAR file; you only need log4j-1.2-api-2.1.jar) from
http://logging.apache.org/log4j/2.x/download.html

poi-3.11-20141221.zip (unzip to find JAR file) from
http://poi.apache.org/download.html

Then start DbVisualizer and open **Tools->Driver Manager** and

1. Create a new Driver and name it UCanAccess,
2. Load all the JAR files in the User Specified tab in the order listed above,
3. Close the Driver Manager,
4. Open the Object View tab for your MS Access connection and set the Driver to UCanAccess,
5. Enter the JDBC URL in this format:
   `jdbc:ucanaccess://<absolute_path_to_the_ms_access_file>`
6. Open the Properties tab for the connection and select the **Delimited Identifiers** category,
7. Change the **Begin Identifier** to `[` and the **End Identifier** to `]` and click Apply,
8. Switch back to the Connection tab and click Connect.

18.19.2 Easysoft JDBC-ODBC Bridge Driver

This is a commercial JDBC-ODBC Bridge Driver, available here:

http://www.easysoft.com/products/data_access/jdbc_odbc_bridge/ (http://www.easysoft.com/products/data_access/jdbc_odbc_bridge/#section%3Dtab-1)

The "Working with ODBC data in DbVisualizer (http://www.easysoft.com/blog/dbvisualizer.html)" page at the Easysoft web site explains all details how to use it with DbVisualizer.

19 Finding Database Objects and Data

DbVisualizer provides ways to find all kinds of things, from parts of a script and data in a grid to objects in a connection tree.

19.1 Finding and Replacing Text in the Editor

The **Edit** main menu and the editor right-click menu contain two choices for finding text: **Find** and **Find with Dialog**.

Find displays a Quick Find field where you can type text to look for, and use the **Up** and **Down** keys (and **F3** and **Shift-F3**, by default) to find the next or previous occurrence. Use the **Escape** key to close the field.

Find with Dialog shows a dialog where you can enter what to look for, either as text or as a regular expression. You can also limit the search to the current selection and use other options for a more precise search. You can use Find Next and Find Previous to navigate to other matches, by default mapped to the **F3** and **Shift-F3** keys.

Use the **Replace** menu choice to show a dialog identical to **Find with Dialog** but with an additional field for entering then replacement text. When you use a regular expression with group expressions in the **Find what** field, you can use back references to the captured text in the **Replace with** field. The back reference is written as the dollar sign plus the group number. For instance, to replace all occurrences of the words `COL1` and `COL2` with the same word plus the character A (`COL1A` and `COL2A`) in a text, you specify the regular expression as a group, i.e. `(COL1|COL2)`, and use a back reference for the first group in the replacement text plus the additional character A, i.e. `$1A`.

19.2 Finding Data in a Grid

The right-click menu for a grid contains the **Find Data** and **Find Column** choices.

Find Data shows a Quick Find field where you can type text to look for, and use the **Up** and **Down** keys to find the next or previous occurrence. Use the **Escape** key to close the field.

Find Column works the same, except it locates a column with the name you type.

19.3 Locating an Object in an SQL Statement

To open an **Object View** tab for an object named in an SQL statement (e.g. a table in a SELECT statement) in the SQL Commander tab, place the caret in or next to the name and choose **Show Object at Cursor** from the right-click menu.

19.4 Locating an Object in the Databases tab

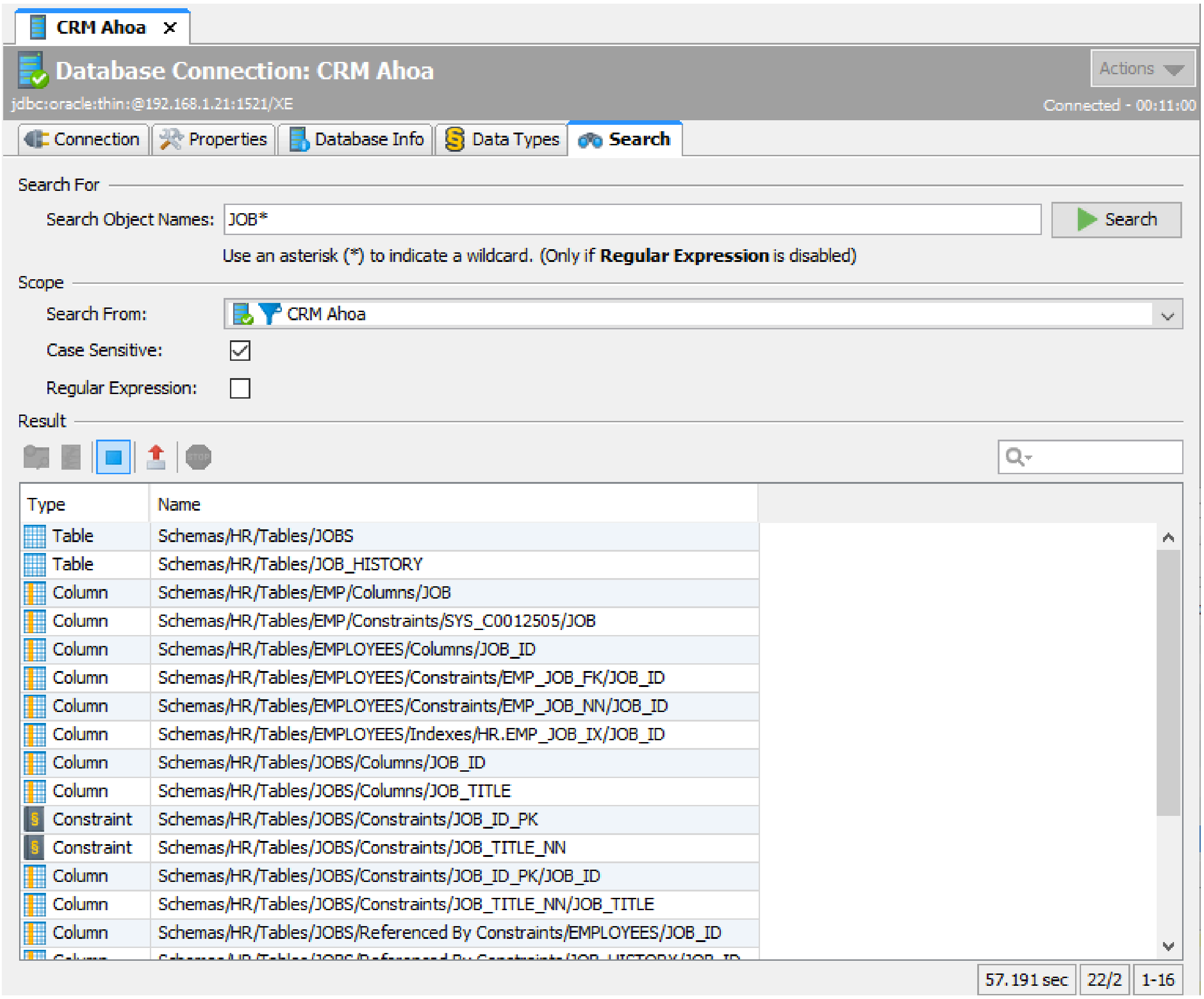
With a node selected in the Databases tab, typing any character shows a Quick Find field where you can type the name of an object you want to locate. Use the **Escape** key to close the field.

> ⚠ Note that only the visible, expanded, nodes are searched. To search among all nodes for a connection, see Searching a Connection (see page 311).

19.5 Searching a Connection

> **Only in DbVisualizer Pro**
>
> This feature is only available in the DbVisualizer Pro edition.

The **Search** tab in the **Object View** tab for a connection is used to search among the objects in the tree by object name. 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.



Search For

Search Object Names: JOB* [Search]

Use an asterisk (*) to indicate a wildcard. (Only if **Regular Expression** is disabled)

Scope

Search From: CRM Ahoa

Case Sensitive: ☑

Regular Expression: ☐

Result

Type	Name
Table	Schemas/HR/Tables/JOBS
Table	Schemas/HR/Tables/JOB_HISTORY
Column	Schemas/HR/Tables/EMP/Columns/JOB
Column	Schemas/HR/Tables/EMP/Constraints/SYS_C0012505/JOB
Column	Schemas/HR/Tables/EMPLOYEES/Columns/JOB_ID
Column	Schemas/HR/Tables/EMPLOYEES/Constraints/EMP_JOB_FK/JOB_ID
Column	Schemas/HR/Tables/EMPLOYEES/Constraints/EMP_JOB_NN/JOB_ID
Column	Schemas/HR/Tables/EMPLOYEES/Indexes/HR.EMP_JOB_IX/JOB_ID
Column	Schemas/HR/Tables/JOBS/Columns/JOB_ID
Column	Schemas/HR/Tables/JOBS/Columns/JOB_TITLE
Constraint	Schemas/HR/Tables/JOBS/Constraints/JOB_ID_PK
Constraint	Schemas/HR/Tables/JOBS/Constraints/JOB_TITLE_NN
Column	Schemas/HR/Tables/JOBS/Constraints/JOB_ID_PK/JOB_ID
Column	Schemas/HR/Tables/JOBS/Constraints/JOB_TITLE_NN/JOB_TITLE
Column	Schemas/HR/Tables/JOBS/Referenced By Constraints/EMPLOYEES/JOB_ID
Column	Schemas/HR/Tables/JOBS/Referenced By Constraints/JOB_HISTORY/JOB_ID

57.191 sec | 22/2 | 1-16

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.

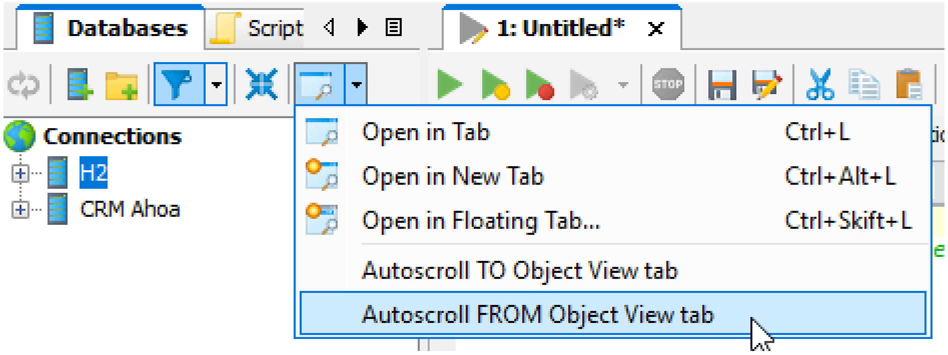
Shift+Click on a row to switch to the Object View to see detailed information about a specific object. **Shift+ Double-click** on a row to see detailed information about a specific object in a separate window.

⚠ 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.

19.6 Synchronizing object tab selection and selection in the tree

By use of this functionality it is possible to set up DbVisualizer to automatically select the database tree node when the corresponding object view tab is selected and vice versa.

This is done through the **Open selected object(s)** dropdown in the toolbar of the Databases tab tree.


Databases
Script
1: Untitled*
Connections
H2
CRM Ahoa
Open in Tab Ctrl+L
Open in New Tab Ctrl+Alt+L
Open in Floating Tab... Ctrl+Skift+L
Autoscroll TO Object View tab
Autoscroll FROM Object View tab

The two options controlling this are

- **Autoscroll TO Object View tab** Automatically open the object in an object view tab when the tree node is selected
- **Autoscroll FROM Object View tab** Automatically select the tree node whenever its object view tab is clicked

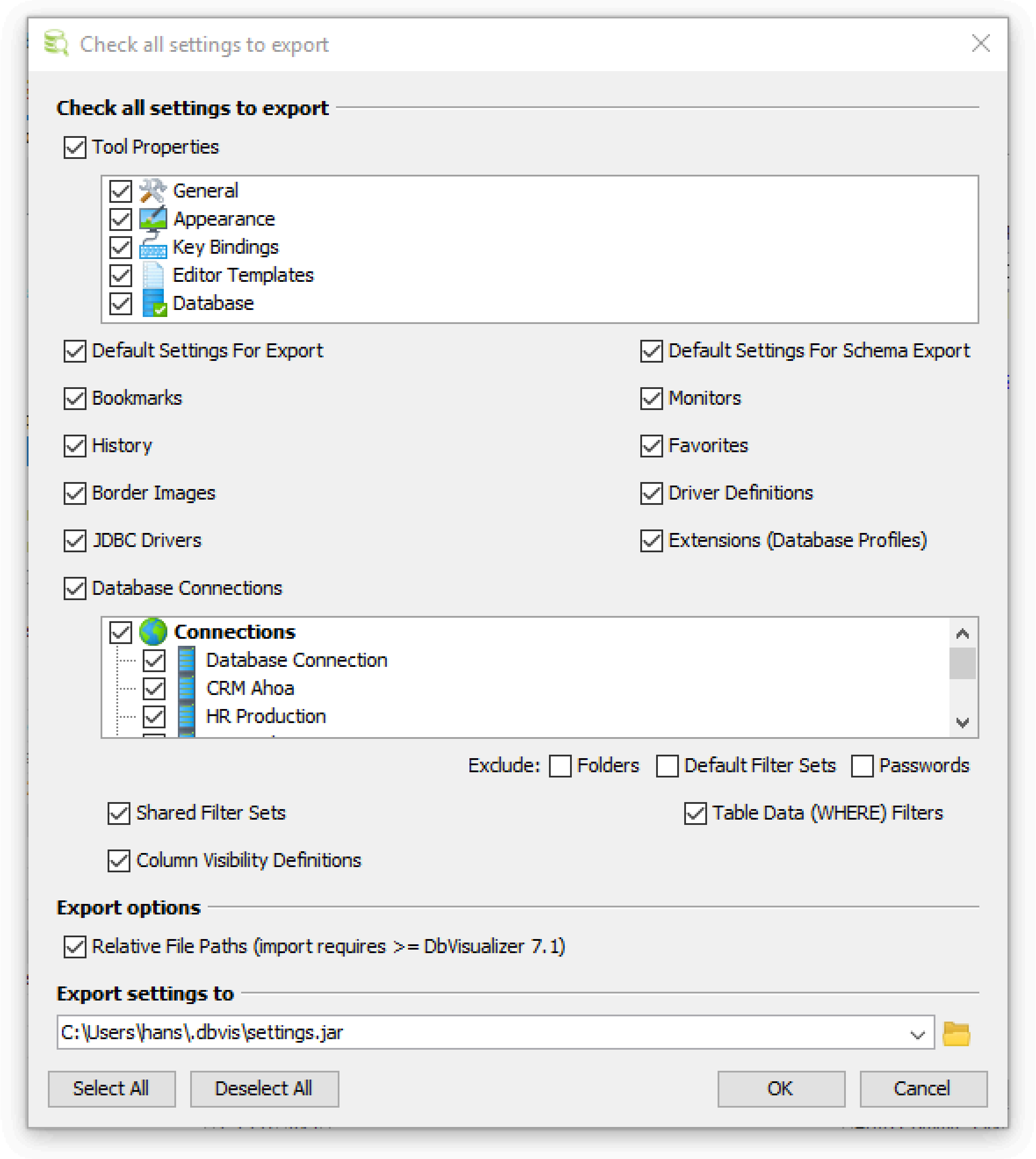
The default values for **Autoscroll** can be configured in **Tool Properties** -> **General** -> **Database Objects Tree**.

20 Exporting and Importing Settings

Sometimes it may be necessary to migrate all your settings for DbVisualizer and import them in second DbVisualizer installation. 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, or if you would like to share you settings with other users. Another key reason is for backup purposes. Loosing all database connection configurations can be really frustrating.

20.1 Export Settings

The Export Settings feature is available from the **File->Export Settings** main window menu choice.



Check all settings to export

Check all settings to export

☑ Tool Properties

- ☑ General
- ☑ Appearance
- ☑ Key Bindings
- ☑ Editor Templates
- ☑ Database

☑ Default Settings For Export ☑ Default Settings For Schema Export

☑ Bookmarks ☑ Monitors

☑ History ☑ Favorites

☑ Border Images ☑ Driver Definitions

☑ JDBC Drivers ☑ Extensions (Database Profiles)

☑ Database Connections

- ☑ **Connections**
  - ☑ Database Connection
  - ☑ CRM Ahoa
  - ☑ HR Production

Exclude: □ Folders □ Default Filter Sets □ Passwords

☑ Shared Filter Sets ☑ Table Data (WHERE) Filters

☑ Column Visibility Definitions

Export options

☑ Relative File Paths (import requires >= DbVisualizer 7.1)

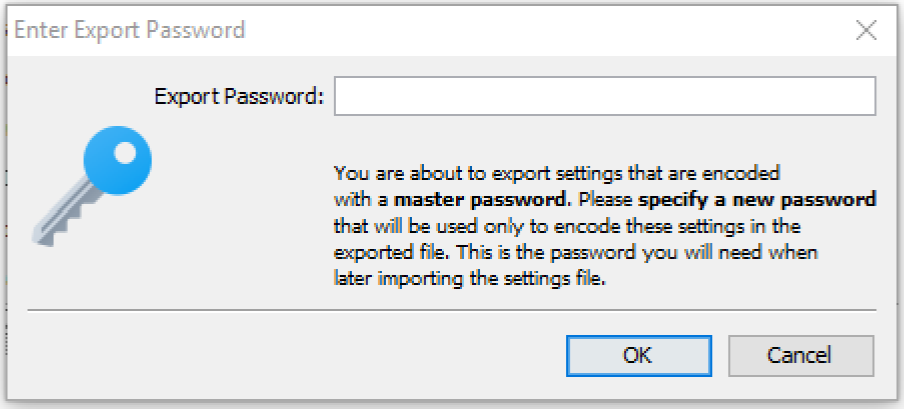
Export settings to

C:\Users\hans\.dbvis\settings.jar

The default settings ensures that all settings are exported, but you can selectively exclude certain items. Once you've made the adjustments you want, press OK and the settings will be saved in the specified file. The structure of this JAR file is the same as the content in your DbVisualizer preferences directory.

⚠ The **Relative File Paths** option will transform all path definitions in the exported file to be relative to the DbVisualizer installation directory and your personal settings directory. This is useful if you will import the settings on another machine or share it with other users. Note that the DbVisualizer version importing relative file paths must be 7.1 or later to work properly (importing in earlier versions than 7.1 will not fail but path information will be erroneous for things such as drivers, favorites, etc.)

If Master Password (see page 294) is enabled for DbVisualizer and passwords are not excluded in the export you must specify an export password. The export password is used to protect the passwords that are exported. This export password is used later when importing the settings.



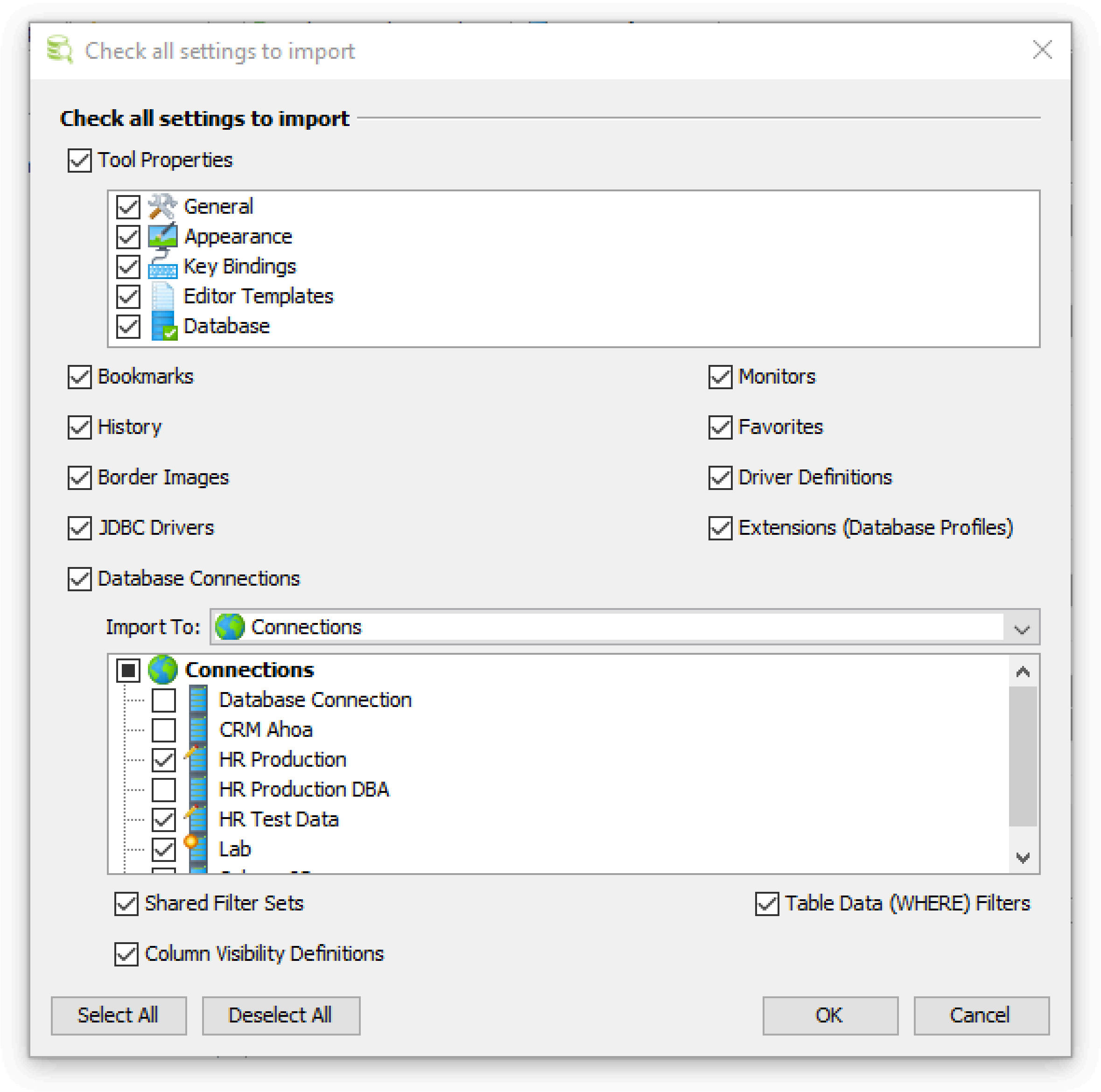
You are about to export settings that are encoded with a **master password.** Please **specify a new password** that will be used only to encode these settings in the exported file. This is the password you will need when later importing the settings file.

You should **not** use your master password as the export password.

20.2 Import Settings

The Import Settings dialog is launched from the **File->Import Settings** main window menu choice.

Import Settings is used to import settings as previously exported with Export Settings. Import examines the content of the specified file and present the choices available.



Check all settings to import

Check all settings to import

☑ Tool Properties

- ☑ General
- ☑ Appearance
- ☑ Key Bindings
- ☑ Editor Templates
- ☑ Database

☑ Bookmarks ☑ Monitors

☑ History ☑ Favorites

☑ Border Images ☑ Driver Definitions

☑ JDBC Drivers ☑ Extensions (Database Profiles)

☑ Database Connections

Import To: Connections

■ **Connections**

- ☐ Database Connection
- ☐ CRM Ahoa
- ☑ HR Production
- ☐ HR Production DBA
- ☑ HR Test Data
- ☑ Lab

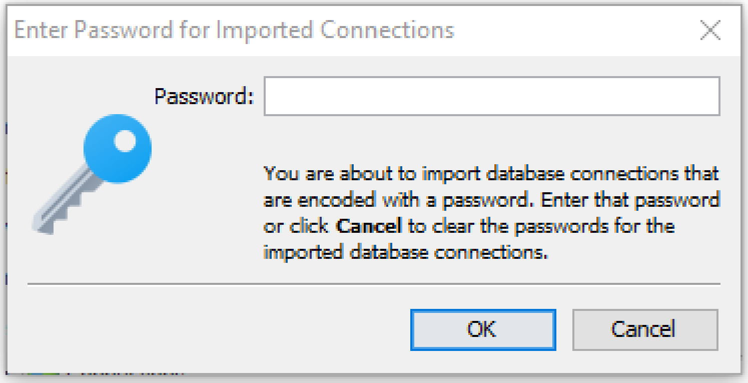
☑ Shared Filter Sets ☑ Table Data (WHERE) Filters

☑ Column Visibility Definitions

Use the **Import To** list to set where the imported database connections will appear in the objects tree.

Note how some of the databases and folders are pre-selected. DBVisualizer will automatically pre-select databases and folders depending on differences between the current connection settings and the imported connection settings. In this example, the icon decorations for the pre-selected connections show that the **HR Production** and **HR Test Data** connection already exist but the imported settings are different, and that the **Lab** connection exists only in the imported settings. All the other connections exists with the same settings as in the imported settings.

If the imported data has been exported with an export password you will be prompted to enter it. You can optionally click Cancel to clear the passwords for all the connections.



Enter Password for Imported Connections

Password:

You are about to import database connections that are encoded with a password. Enter that password or click **Cancel** to clear the passwords for the imported database connections.

21 Command Line Interface

In addition to the DbVisualizer GUI tool, there is also a pure command line interface for running scripts. We recommend that you use this interface 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 macOS, the shell script is located in */Application/DbVisualizer-<Version>.app/Contents/Resources/app*.

21.1 Command Line Options

The command line interface supports the following options:

```
 Usage: dbviscmd (-connection <name> |
                -url -drivername | -url -driverclass -driverpath))
                [-userid <userid>] [-password <password>]
                -sql <statements> | -sqlfile <filename> [-encoding <encoding>]
                [-catalog <catalog>] [-schema <schema>]
                [-maxrows <max>] [-maxchars <max>]
                [-stoponerror] [-stoponwarning]
                [-stripcomments true | false]
                [-output all | none | log | result] [-outputfile <filename>]
                [-listconnections]
                [-debug [-debugfile <filename>]
                [-prefsdir <directory>]
                [-masterpw <password>] [-help] [-version]
Options:
 -connection <name>     Database connection name (created with the GUI)
 -url <URL>             Database URL
 -drivername <name>     Database driver name (created with the GUI)
 -driverclass <name>    Full name of the JDBC Driver class name
 -driverpath <p1:p2..>  Paths to the jar files constituting the JDBC driver. Each path separated by
a ":"
 -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
 -stoponsqlwarning      Stop execution when getting an SQL warning
 -stoponnorows          Stop execution when empty result set or no affected rows
 -stripcomments         Strip comments before sending to database. Default is
                        the setting made in the GUI
 -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
 -masterpw <password>   Master Password for encrypted database passwords
 -help                  Display this help
 -version               Show version info
```

There are two options to to specify which database to connect to

- Using a connection already defined by the GUI tool. This is done by using the -connection parameter. If you have forgot the connection name, use the `-listconnections` option to get a list of all connection names.
- Specifying the connection properties by using the parameter -url. The -url parameter specifies the JDBC URL for the database to connect to. For information about the JDBC url see Setting Up a Connection Manually (see page 283). There are also some examples below showing how to specify connection properties using the -url parameter

21.2 Examples

21.2.1 Executing single statements

You can use the command line interface to execute a single SQL statement:

```
> dbviscmd.bat -connection "Oracle" -sql "select * from hr.countries"
14:34:48  START Executing Command Line, Database Connection: Oracle Database Type: ORACLE Catalog:
null Schema: SYSTEM
14:34:48  INFO  Physical database connection acquired for: Oracle
```

```
COUNTRY_ID  COUNTRY_NAME              REGION_ID
----------  ------------------------  ---------
AR          Argentina                 2
AU          Australia                 3
BE          Belgium                   1
BR          Brazil                    2
CA          Canada                    2
CH          Switzerland               1
CN          China                     3
DE          Germany                   1
DK          Denmark                   1
EG          Egypt                     4
FR          France                    1
HK          HongKong                  3
IL          Israel                    4
IN          India                     3
IT          Italy                     1
JP          Japan                     3
KW          Kuwait                    4
MX          Mexico                    2
NG          Nigeria                   4
NL          Netherlands               1
SG          Singapore                 3
UK          United Kingdom            1
US          United States of America  2
ZM          Zambia                    4
ZW          Zimbabwe                  4
14:34:48  SUCCESS  [SELECT - 25 rows, 0.007 secs]  Result set fetched
select * from hr.countries;
14:34:48  END Execution 1 statement(s) executed, 25 row(s) affected, exec/fetch time: 0.007/0.005
secs   [1 successful, 0 errors]
```

If you like to execute just a few statements, you can pass in a list of statements:

```
> dbviscmd.bat -connection "Oracle" -sql "select * from hr.countries; select * from hr.regions"
14:42:21  START Executing Command Line, Database Connection: Oracle Database Type: ORACLE Catalog:
null Schema: SYSTEM
14:42:21  INFO  Physical database connection acquired for: Oracle
COUNTRY_ID  COUNTRY_NAME              REGION_ID
----------  ------------------------  ---------
AR          Argentina                 2
AU          Australia                 3
BE          Belgium                   1
BR          Brazil                    2
CA          Canada                    2
CH          Switzerland               1
CN          China                     3
DE          Germany                   1
DK          Denmark                   1
EG          Egypt                     4
FR          France                    1
HK          HongKong                  3
```

```
IL          Israel                    4
IN          India                     3
IT          Italy                     1
JP          Japan                     3
KW          Kuwait                    4
MX          Mexico                    2
NG          Nigeria                   4
NL          Netherlands               1
SG          Singapore                 3
UK          United Kingdom            1
US          United States of America  2
ZM          Zambia                    4
ZW          Zimbabwe                  4
14:42:21  SUCCESS  [SELECT - 25 rows, 0.004 secs]  Result set fetched
select * from hr.countries;
REGION_ID  REGION_NAME
---------  ----------------------
5          Australia
6          South America
1          Europe
2          Americas
3          Asia
4          Middle East and Africa
14:42:21  SUCCESS  [SELECT - 6 rows, 0.003 secs]  Result set fetched
select * from hr.regions;
14:42:21  END Execution 2 statement(s) executed, 31 row(s) affected, exec/fetch time: 0.007/0.002
secs   [2 successful, 0 errors]
```

21.2.2 Executing scripts

If you frequently want to execute a number of statements, it's best to put them into a script file. Here's how to execute a script that contains the two statements from the example above:

```
> dbviscmd.bat -connection "Oracle" -sqlfile "myscript.sql"
14:45:11  START Executing Command Line, Database Connection: Oracle Database Type: ORACLE Catalog:
null Schema: SYSTEM
14:45:11  INFO  Physical database connection acquired for: Oracle
COUNTRY_ID  COUNTRY_NAME              REGION_ID
----------  ------------------------  ---------
AR          Argentina                 2
AU          Australia                 3
BE          Belgium                   1
BR          Brazil                    2
CA          Canada                    2
CH          Switzerland               1
CN          China                     3
DE          Germany                   1
DK          Denmark                   1
EG          Egypt                     4
FR          France                    1
```

```
HK          HongKong                  3
IL          Israel                    4
IN          India                     3
IT          Italy                     1
JP          Japan                     3
KW          Kuwait                    4
MX          Mexico                    2
NG          Nigeria                   4
NL          Netherlands               1
SG          Singapore                 3
UK          United Kingdom            1
US          United States of America  2
ZM          Zambia                    4
ZW          Zimbabwe                  4
14:45:11  SUCCESS  [SELECT - 25 rows, 0.004 secs]  Result set fetched
select * from hr.countries;
REGION_ID  REGION_NAME
---------  ----------------------
5          Australia
6          South America
1          Europe
2          Americas
3          Asia
4          Middle East and Africa
14:45:11  SUCCESS  [SELECT - 6 rows, 0.003 secs]  Result set fetched
select * from hr.regions;
14:45:11  END Execution 2 statement(s) executed, 31 row(s) affected, exec/fetch time: 0.007/0.002
secs   [2 successful, 0 errors]
```

21.2.3 Controlling the output

You can use options to control how much output to generate. If you only want to see the results, use the `-output` option with the result keyword:

```
> dbviscmd.bat -connection "Oracle" -sqlfile "myscript.sql" -output result
COUNTRY_ID  COUNTRY_NAME              REGION_ID
----------  ------------------------  ---------
AR          Argentina                 2
AU          Australia                 3
BE          Belgium                   1
BR          Brazil                    2
CA          Canada                    2
CH          Switzerland               1
CN          China                     3
DE          Germany                   1
DK          Denmark                   1
EG          Egypt                     4
FR          France                    1
HK          HongKong                  3
IL          Israel                    4
```

```
IN          India                     3
IT          Italy                     1
JP          Japan                     3
KW          Kuwait                    4
MX          Mexico                    2
NG          Nigeria                   4
NL          Netherlands               1
SG          Singapore                 3
UK          United Kingdom            1
US          United States of America  2
ZM          Zambia                    4
ZW          Zimbabwe                  4
REGION_ID  REGION_NAME
---------  ----------------------
1          Europe
2          Americas
3          Asia
4          Middle East and Africa
```

For other scripts, for instance a script containing INSERT statements, you may only want to see the log messages:

```
> dbviscmd.bat -connection "Oracle" -sqlfile "myscript.sql" -output log
14:25:29  START Executing Command Line, Database Connection: Oracle Database Type: ORACLE Catalog:
null Schema: SYSTEM
14:25:30  INFO  Physical database connection acquired for: Oracle
14:25:30  SUCCESS  [SELECT - 25 rows, 0.012 secs]  Result set fetched
select * from hr.countries;
14:25:30  SUCCESS  [SELECT - 4 rows, 0.009 secs]  Result set fetched
select * from hr.regions;
14:25:30  END Execution 2 statement(s) executed, 29 row(s) affected, exec/fetch time: 0.021/0.004
secs   [2 successful, 0 errors]
```

21.2.4 Combining OS scripts, the command line interface and DbVisualizer variables

For more complex tasks, you can call the command line interface from a shell script, for instance a Bourne shell script on Unix or a BAT file on Windows. You can also use DbVisualizer variables to pass information between the shell script and the SQL script. In this example, we have a simple SQL script (*cmdtest.sql*) that contains a SELECT statement with a variable in place for the table name:

cmdtest.sql

```
select * from ${table}$
```

A text file (*tables.txt*) contains the table names we want to execute the SQL script with:

tables.txt

```
hr.countries
hr.regions
```

In a command shell (Bourne or Bash), we can then execute the script using the table names from the text file:

```
for name in `cat tables.txt`;
  do ./dbviscmd.sh -connection "oracle" -sql "@run cmdtest.sql \${table||$name|||nobind}\$; ";
done
15:01:19  START Executing Command Line, Database Connection: oracle Database Type: ORACLE Catalog:
null Schema: SYSTEM
15:01:20  INFO  Physical database connection acquired for: oracle
15:01:20  RUNNING  [@run ...ntries|||nobind}$ - - secs]
@run cmdtest.sql ${table||hr.countries|||nobind}$;
COUNTRY_ID  COUNTRY_NAME              REGION_ID
----------  ------------------------  ---------
AR          Argentina                 2
AU          Australia                 3
BE          Belgium                   1
BR          Brazil                    2
CA          Canada                    2
CH          Switzerland               1
CN          China                     3
DE          Germany                   1
DK          Denmark                   1
EG          Egypt                     4
FR          France                    1
HK          HongKong                  3
IL          Israel                    4
IN          India                     3
IT          Italy                     1
JP          Japan                     3
KW          Kuwait                    4
MX          Mexico                    2
NG          Nigeria                   4
NL          Netherlands               1
SG          Singapore                 3
UK          United Kingdom            1
US          United States of America  2
ZM          Zambia                    4
ZW          Zimbabwe                  4
15:01:20  SUCCESS  [SELECT - 25 rows, 0.016 secs]  Result set fetched
select * from hr.countries;
15:01:20  SUCCESS  [@run ...ntries|||nobind}$ - 0.016 secs]  Script processed
@run cmdtest.sql ${table||hr.countries|||nobind}$;
15:01:20  END Execution 1 statement(s) executed, 25 row(s) affected, exec/fetch time: 0.016/0.012
secs   [1 successful, 0 errors]
java -cp /Users/ulf/work/github/dbvis/trunk/pureit/apps/dbvis/classes:/Users/ulf/work/github/dbvis/
trunk/pureit/apps/dbvis/resources:/Users/ulf/work/github/dbvis/trunk/pureit/apps/dbvis/external/* -
```

```
Xmx512M -Djava.awt.headless=true -Ddbvis.home=/Users/ulf/work/github/dbvis/trunk/pureit/apps/dbvis
com.onseven.dbvis.DbVisualizerCmd -masterpw stairway -connection oracle -sql @run cmdtest.sql ${
table||hr.regions||||nobind}$;
15:01:23  START Executing Command Line, Database Connection: oracle Database Type: ORACLE Catalog:
null Schema: SYSTEM
15:01:23  INFO  Physical database connection acquired for: oracle
15:01:23  RUNNING  [@run ...egions||||nobind}$ - - secs]
@run cmdtest.sql ${table||hr.regions||||nobind}$;
REGION_ID  REGION_NAME
---------  ----------------------
5          Australia
6          South America
1          Europe
2          Americas
3          Asia
4          Middle East and Africa
15:01:23  SUCCESS  [SELECT - 6 rows, 0.001 secs]  Result set fetched
select * from hr.regions;
15:01:23  SUCCESS  [@run ...egions||||nobind}$ - 0.001 secs]  Script processed
@run cmdtest.sql ${table||hr.regions||||nobind}$;
15:01:23  END Execution 1 statement(s) executed, 6 row(s) affected, exec/fetch time: 0.001/0.000
secs   [1 successful, 0 errors]
```

The command line interface is called with the `-sql` option, specifying the client-side command `@run` (see page 180). A DbVisualizer variable (see page 227) is passed to the `@run` command with the value taken from the shell variable. This DbVisualizer variable value is then available to the script executed by the `@run` command.

Note that you may need to escape certain characters that the shell would otherwise interpret, like the dollar signs that are part of the DbVisualizer variable delimiters.

21.3 Setting up the connection properties on the command line

As an alternative to using a connection already set-up through GUI tool you may use the -url parameter. In combination with the parameters -drivername, -driverclass and -driverpath this parameter enables you to get connected without prior specification using the GUI.

Following are some examples.

Executing SQL towards a MYSQL instance running on localhost port 3306. The parameter "-drivername MYSQL " specifies that we are using a drive specified in the GUI named MYSQL. For listing of the existing drivers use the GUI Driver Manager. See Installing a JDBC Driver (see page 430).

Using url and drivername

```
./dbviscmd.sh -url jdbc:mysql://localhost:3306/ -drivername MYSQL -sql "select * from sakila.actor"
  -userid root
```

An alternative to use the -drivername you may use the parameters -driverclass and -driverpath to specify the JDBC driver.

Using url and drivername

```
./dbviscmd.sh -url jdbc:oracle:thin:@localhost:1521/ORCL -driverclass oracle.jdbc.OracleDriver -
driverpath "/tmp/my_oracle_driver/ojdbc6.jar:/tmp/my_oracle_driver/orai18n.jar:/tmp/
my_oracle_driver/xdb.jar:/tmp/my_oracle_driver/xmlparserv2.jar" -sql "select * from HR.COUNTRIES"
-userid system -password oracle
```

The above example connects to an ORACLE instance on localhost and port 1521. Note that the separator character ':' between the different jar files is platform dependant. On Windows-based desktop platforms, the value of this field is the semicolon (;) .

22 Database Profiles

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

A Database Profile is the foundation for database specific support in DbVisualizer. Technically the database profile is a single XML file specifying what object types, actions and viewers/editors should be available in the DbVisualizer user interface for a specific database.

22.1 Understanding Database Profiles

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

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.

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

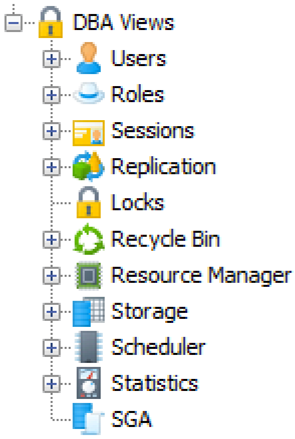
- DB2 LUW (http://www.dbvis.com/doc/db2-luw-database-support/)
- H2 (http://www.dbvis.com/doc/h2-database-support/)
- Informix (http://www.dbvis.com/doc/informix-database-support/)
- JavaDB/Derby (http://www.dbvis.com/doc/javadb-derby-database-support/)
- Mimer SQL (http://www.dbvis.com/doc/mimer-database-support/)
- MySQL (http://www.dbvis.com/doc/mysql-database-support/)
- Netezza (http://www.dbvis.com/doc/netezza-database-support/)
- NuoDB (http://www.dbvis.com/doc/nuodb-database-support/)
- Oracle (http://www.dbvis.com/doc/oracle-database-support/)
- PostgreSQL (http://www.dbvis.com/doc/postgresql-database-support/)
- Redshift (http://www.dbvis.com/doc/redshift-database-support/)

- SQL Server (http://www.dbvis.com/doc/sqlserver-database-support/)
- SQLite (http://www.dbvis.com/doc/sqlite-database-support/)
- Sybase ASE (http://www.dbvis.com/doc/sybase-ase-database-support/)
- Vertica (http://www.dbvis.com/doc/vertica-database-support/)

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 objects tree, even though the provided ones are similar in their structure. There are two root nodes in the majority of the provided profiles:

- **Schema Objects**
- **DBA objects**

Schema 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. The convention in DbVisualizer is to put 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:

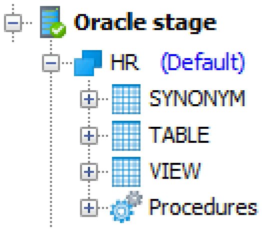


For databases that have no specific profile, DbVisualizer uses the **Generic** profile. DbVisualizer supports a wide range of databases. The nature of the databases and what they support differ from vendor to vendor, so the appearance and structure of the tree below the database connection objects for 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 Pro edition.

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



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. There is no DBA Views node in the generic profile.

22.1.1 Affected DbVisualizer features

Only in DbVisualizer Pro

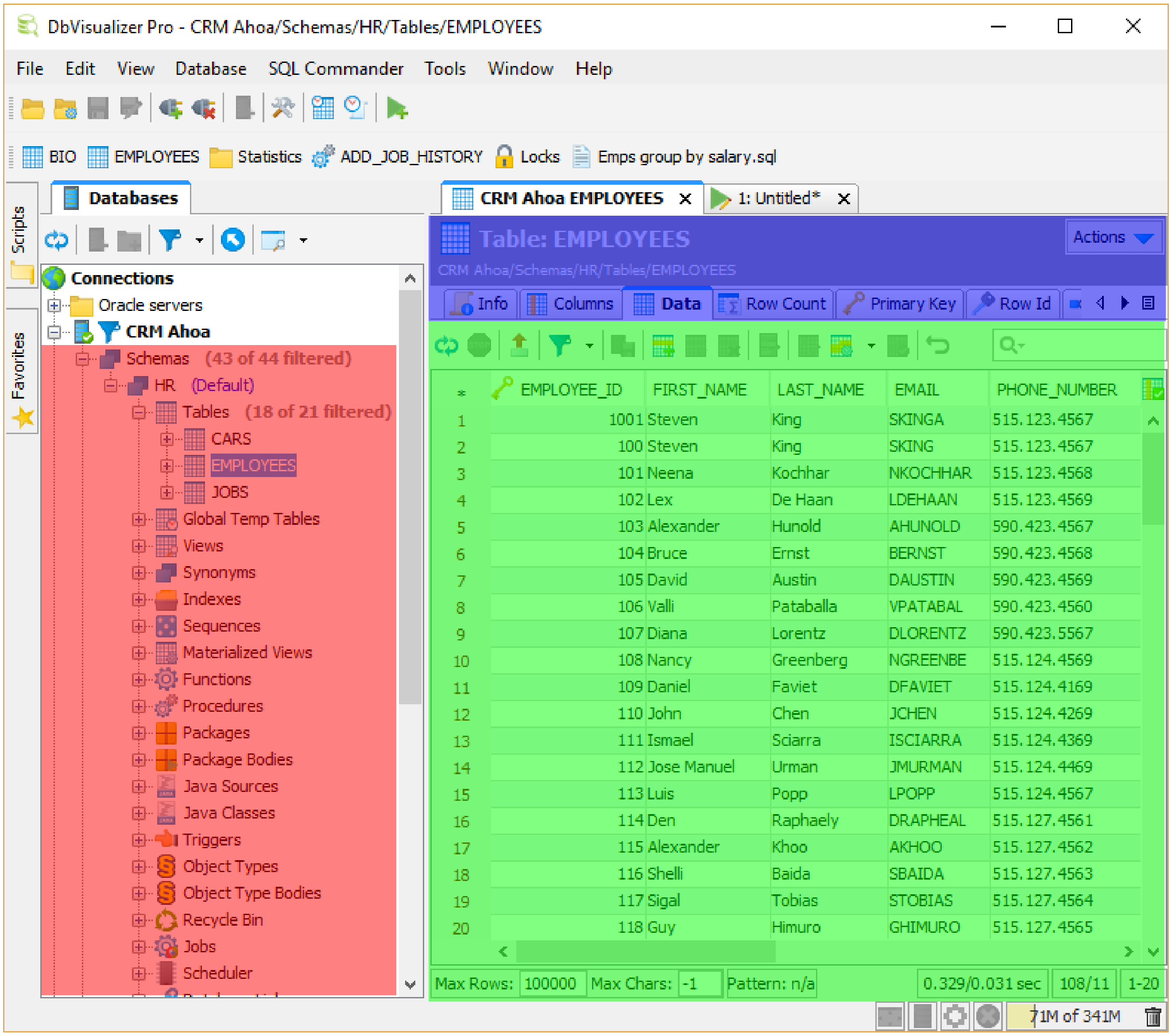
This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

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** file with all of the logic, structure and actions 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 related sections in the users guide) is all the existing database profiles that comes with DbVisualizer. All database profiles that comes with DbVisualizer are stored in the `DBVIS-HOME /resources/profiles` directory (exact path is OS dependent).

The following figure illustrates which features in DbVisualizer are controlled by the database profile.



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER
1	1001	Steven	King	SKINGA	515.123.4567
2	100	Steven	King	SKING	515.123.4567
3	101	Neena	Kochhar	NKOCHHAR	515.123.4568
4	102	Lex	De Haan	LDEHAAN	515.123.4569
5	103	Alexander	Hunold	AHUNOLD	590.423.4567
6	104	Bruce	Ernst	BERNST	590.423.4568
7	105	David	Austin	DAUSTIN	590.423.4569
8	106	Valli	Pataballa	VPATABAL	590.423.4560
9	107	Diana	Lorentz	DLORENTZ	590.423.5567
10	108	Nancy	Greenberg	NGREENBE	515.124.4569
11	109	Daniel	Faviet	DFAVIET	515.124.4169
12	110	John	Chen	JCHEN	515.124.4269
13	111	Ismael	Sciarra	ISCIARRA	515.124.4369
14	112	Jose Manuel	Urman	JMURMAN	515.124.4469
15	113	Luis	Popp	LPOPP	515.124.4567
16	114	Den	Raphaely	DRAPHEAL	515.127.4561
17	115	Alexander	Khoo	AKHOO	515.127.4562
18	116	Shelli	Baida	SBAIDA	515.127.4563
19	117	Sigal	Tobias	STOBIAS	515.127.4564
20	118	Guy	Himuro	GHIMURO	515.127.4565

Max Rows: 100000 Max Chars: -1 Pattern: n/a 0.329/0.031 sec 108/11 1-20

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 (see page 369) 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 (see page 386) section.

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 (see page 359) root element,
- The object views (green and blue boxes) are described by the ObjectsViewDef (see page 369) root element,
  The commands used to execute the SQL to get the information for ObjectsTreeDef, ObjectsViewDef and ObjectsActionDef definitions are defined by the Commands (see page 349) root element,
- All Actions for an object are defined by the ObjectsActionDef (see page 386) root element.

22.1.2 How a Database Profile is loaded

> **Only in DbVisualizer Pro**
>
> This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

DbVisualizer automatically detect what database profile to use based on the **Database Type** setting for a database connection. A database profile can also be manually specified in the connection properties.

A database profile is located using the search paths defined in **Connection Properties** tab and the **Database Profile** category. The standard directories in the search path are:

1. *PREFSDIR\ext\profile*
2. *DBVIS-HOME\resources\profiles*

PREFSDIR is the *.dbvis* directory located in the users home directory and it keeps all user settings for DbVisualizer. The list of paths may be reorganized and directories can be added. Profile files are searched in the specified order.

If the actual database profile is found in the search path it is loaded and any parent profile(s) it extends are also loaded and finally merged.

If there is no matching profile or if using the **DbVisualizer Free** edition then the **generic** profile is automatically used. This is very basic profile and shows only rudimentary information about the objects in the database and should support most databases with a JDBC driver.

A database profile other than the generic is built for a specific database. Manually selecting for example a database profile for Oracle while connecting to DB2 will result in all sorts of errors.

22.2 Creating a Database Profile

ⓘ **Only in DbVisualizer Pro**

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

At a first glance creating and developing a database profile in XML may seem difficult. However, all definitions forming the functionality for a specific database are expressed in a single file and the XML elements are well formed. The general recommendation is to use one of the existing database profiles as base (copy/paste) and then step-by-step modify it for the actual database. You may use an external text editor (preferably with XML support) or the SQL editor in DbVisualizer to edit the file. Once new changes has been made, save the file, reconnect the database connection in DbVisualizer to test the layout and functionality changes.

Creating a new database profile should only be made for databases with no database profile available. If you are looking into changing one of the existing database profiles by adding or modifying existing functionality, it should be extended.

ⓘ For information where custom database profiles should be saved check the How a Database Profile is loaded (see page 332) is section.

For more information how to **create a new database profile** or **extending an existing database profile** check the Extending a Database Profile (see page 333) section.

22.3 Extending a Database Profile

ⓘ **Only in DbVisualizer Pro**

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

- Extending Commands (see page 334)
- Extending Database Objects Tree (see page 335)
- Extending Actions (see page 339)
- Extending Object Views (see page 340)
- Remove an Element (see page 340)

- Complete sample Database Profile (see page 341)

All database profiles must extend the **generic** database profile. The generic profile handles the very basic object types in a relational database such as **Tables**, **Columns**, **Indexes** and **Procedures**. Its implementation is based entirely on what the **JDBC** driver provide in terms of database meta data. Due to the tight connection between the generic profile and the JDBC driver, the generic profile can be used to access almost any database with a JDBC driver.

The selection on what database profile should be used is determined with the **Database Type** setting for the database connection. Some of the database types that can be picked have a dedicated database profile with extended support while others have not. For databases with no database profile available, the generic one is used. It is also possible to manually chose the generic profile in the **Connection Properties / Database Profiles** settings.

The most important area in the database profile as seen from the DbVisualizer user interface is the section describing the **database objects tree**. This is the browser or navigator showing database objects. This is also the place that connects **actions** used to operate on database objects and **views** (not database views) used to display detailed information.

This section of the users guide is mainly focused on extending an existing database profile (not the generic profile) rather than creating a completely new profile (which should extend the generic database profile).

> Extending a database profile is not only about adding functionality to an existing profile but also the process of changing and removing existing definitions in any of the profiles that are extended.

22.3.1 Extending Commands

Extending the **Commands** and **InitCommands** elements is simple as every command should be uniquely identified. To add a **Command** just insert the new command.

```
   <Commands extends="true">
      <Command id="sample.getLoginSchema">
         <SQL>
            <![CDATA[
select '${schema}' as schema from dual
            ]]>
         </SQL>
      </Command>
   </Commands>
```

To override the definition of an existing command in the parent profile, just make sure the **id** of the new command match the id of the parent profile command. It will then be replaced.

22.3.2 Extending Database Objects Tree

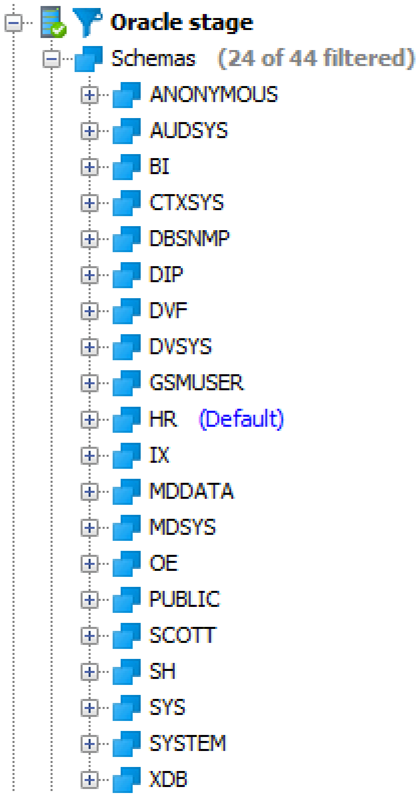
Extending or modifying the database objects tree (ObjectsTreeDef) require some attention since the modifications must match the exact object paths as defined in the parent profile. The object path is determined by the **GroupNode** and **DataNode** structure in the ObjectsTreeDef with the addition of the **type** attribute. The following is an example of the object path to the **Columns** sub node for a **Table** node:

```
GroupNode[@type='Schemas']/DataNode[@type='Schema']/GroupNode[@type='Tables']/DataNode[@type='Table
']/GroupNode[@type='Columns']
```

(The analyze database profile (see page 413) utility will report object paths in the above xpath (https://www.w3schools.com/xml/xpath_intro.asp) format).

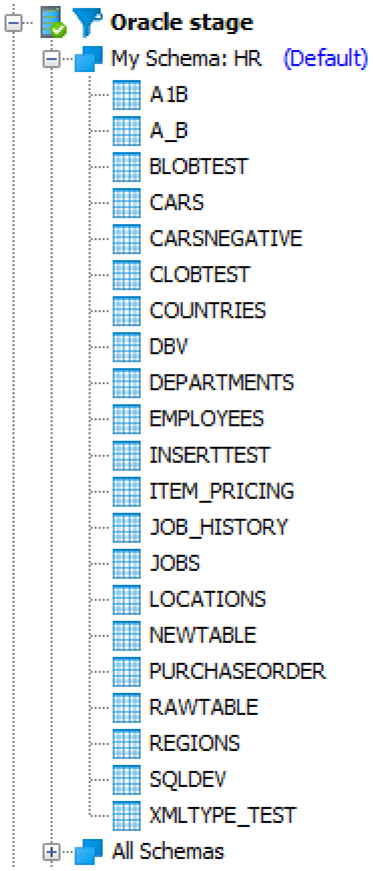
> The hierarchy of GroupNode and DataNode is important when extending the database objects tree since the exact same hierarchy must be implemented in the extended profile. This also involve any conditional elements such as If/Else/ElseIf that are used in the parent profile.

Consider the following example showing the objects tree (for Oracle) with the **Schemas** node being expanded to show all schemas in the database:



Oracle stage
Schemas (24 of 44 filtered)
ANONYMOUS
AUDSYS
BI
CTXSYS
DBSNMP
DIP
DVF
DVSYS
GSMUSER
HR (Default)
IX
MDDATA
MDSYS
OE
PUBLIC
SCOTT
SH
SYS
SYSTEM
XDB

Instead of showing all schema objects in the database we want to adjust so that only the default schema is displayed at the top level (below the Oracle database connection node). The default schema node should in addition only show **table** objects rather than all 20 (or so) object types being displayed in the standard Oracle database profile.



Oracle stage
My Schema: HR (Default)
A1B
A_B
BLOBTEST
CARS
CARSNEGATIVE
CLOBTEST
COUNTRIES
DBV
DEPARTMENTS
EMPLOYEES
INSERTTEST
ITEM_PRICING
JOB_HISTORY
JOBS
LOCATIONS
NEWTABLE
PURCHASEORDER
RAWTABLE
REGIONS
SQLDEV
XMLTYPE_TEST
All Schemas

The previous screenshot shows the new **My Schema: HR** node at the top while the **Schemas** node has been renamed **All Schemas**. To accomplish the above a custom database profile has been created in the *${ dbvis.prefsdir}/ext/profiles/sample-ext-oracle.xml* file with the following content required for the **ObjectsTreeDef** definition:

```
<!--Commands used in this profile-->
<Commands extends="true">
   <Command id="sample.getLoginSchema">
      <SQL>
         <![CDATA[
select '${schema}' as schema from dual
         ]]>
      </SQL>
   </Command>
</Commands>
<ObjectsTreeDef extends="true">
   <!--The following "Schema" definition shows the login schema directly below-->
   <!--the Database Connection for faster access. It is limited to only show-->
   <!--tables (by setting the "Table" DataNode to isLeaf="true")-->
   <DataNode type="Schema" label="My Schema: ${sample.getLoginSchema.SCHEMA}" order-before="0">
      <SetVar name="schema" value="${sample.getLoginSchema.SCHEMA}"/>
      <Command idref="sample.getLoginSchema">
         <Input name="schema" value="${#db.loginSchema}"/>
      </Command>
      <DataNode type="Table" label="${oracle.getTables.TABLE_NAME}" isLeaf="true">
         <SetVar name="objectname" value="${oracle.getTables.TABLE_NAME}"/>
         <SetVar name="rowcount" value="true"/>
         <SetVar name="acceptInQB" value="true"/>
         <Command idref="oracle.getTables">
            <Input name="owner" value="${schema}"/>
            <Input name="temporary" value="N"/>
            <ProcessDataSet action="sortcolumn" index="getTables.TABLE_NAME"/>
            <Filter index="getTables.TABLE_NAME" label="Table"/>
         </Command>
         <!--These are needed for the viewers defined in the parent profile-->
         <!--associated with the "Table" type-->
         <SetVar name="theTableName" value="${objectname}"/>
         <SetVar name="theParentName" value="${objectname}"/>
         <SetVar name="triggersCondition" value="and table_name = '${theTableName}'"/>
      </DataNode>
   </DataNode>
   <!--Renaming the standard Schemas node to "All Schemas"-->
   <GroupNode type="Schemas" label="All Schemas"/>
</ObjectsTreeDef>
```

In the **Commands** section there is a new **Command** that run a dummy SQL SELECT only to create a result set containing a single row/column with the value of the **${schema}** variable. The value for the **${schema}** variable is provided in the Command element for **DataNode type="Schema"** using the **${#db.loginSchema}** variable value as input. This variable is maintained by DbVisualizer and contain the login schema as specified in the connection setup. For Oracle this is the **userid**.

```
<Command idref="sample.getLoginSchema">
   <Input name="schema" value="${#db.loginSchema}"/>
</Command>
```

The command above is used to present the default schema as in the following **DataNode** declaration.

```
<DataNode type="Schema" label="My Schema: ${sample.getLoginSchema.SCHEMA}" order-before="0">
   <SetVar name="schema" value="${sample.getLoginSchema.SCHEMA}"/>
   <Command idref="sample.getLoginSchema">
      <Input name="schema" value="${#db.loginSchema}"/>
   </Command>
   <DataNode type="Table">
      ...
   </DataNode>
</DataNode>
```

The label for this Schema type is **label="My Schema: ${sample.getLoginSchema.SCHEMA}"**. The **${ sample.getLoginSchema . SCHEMA }** variable name consists of two parts, the name of the command: **sample.getLoginSchema** and the column name: **SCHEMA** in the result set the produced by the command.

As a sub node to the **My Schema** node there is the **DataNode type="Table"** definition for the Table object type. The complete declaration for the Table element and its sub elements has been copied from the parent profile:

```
<DataNode type="Table" label="${oracle.getTables.TABLE_NAME}" isLeaf="true">
   <SetVar name="objectname" value="${oracle.getTables.TABLE_NAME}"/>
   <SetVar name="rowcount" value="true"/>
   <SetVar name="acceptInQB" value="true"/>
   <Command idref="oracle.getTables">
      <Input name="owner" value="${schema}"/>
      <Input name="temporary" value="N"/>
      <ProcessDataSet action="sortcolumn" index="TABLE_NAME"/>
      <Filter index="TABLE_NAME" label="Table"/>
   </Command>
   <!--These are needed for the viewers defined in the parent profile-->
   <!--associated with the "Table" type-->
   <SetVar name="theTableName" value="${objectname}"/>
   <SetVar name="theParentName" value="${objectname}"/>
   <SetVar name="triggersCondition" value="and table_name = '${theTableName}'"/>
</DataNode>
```

The **Table** objects in the extended profile should not show any sub nodes such as columns or triggers and these declarations are then removed in the copied DataNode for the Table object. The **isLeaf="true"** attribute specifies that there will be no child nodes.

The new **All Schemas** node in the extended profile is supposed to have the exact same content and definition as in the parent profile when it was called **Schemas**. The following definition will just rename the label of the existing node.

```
<GroupNode type="Schemas" label="All Schemas"/>
```

This example show adding a new **Role** node under all **DBA / Users / User** objects.

In the parent Oracle profile there are no child nodes below **User**. To handle this all nodes from **DBA** down to **User** must be specified (aka the object type path). The only requirement is that the type attribute is specified and that it match the type in the parent profile. In addition, this example specify the **label** attribute for some of the nodes just to show that **overridden attributes** will replace any parent equvivalent node attributes.

> ℹ Only attributes for **GroupNode** and **DataNode** can be overridden. If you need to override for example a SetVar in a DataNode then all of the attributes in the DataNode and all its sub elements must be specified.

```
<GroupNode type="DBA" label="Database Administration">
   <GroupNode type="Users">
      <!--The "User" type don't allow child nodes in the parent profile.-->
      <!--Setting isLeaf="false" is needed to override this and allow the-->
      <!--new "Role" child node-->
      <DataNode type="User" isLeaf="false">
         <!--Here comes the new child "Role" DataNode-->
         <DataNode type="Role" isLeaf="true" label="${oracle.getGranteeRoles.GRANTED_ROLE} - ext">
            <SetVar name="objectname" value="${oracle.getGranteeRoles.GRANTED_ROLE}"/>
            <Command idref="oracle.getGranteeRoles">
               <Input name="grantee" value="${objectname}"/>
            </Command>
         </DataNode>
      </DataNode>
   </GroupNode>
</GroupNode>
```

The following are specified only to redefine the position of the **Locks** and **Sessions** nodes. One of **order-before** and **order-after** attributes are used to either identify a type for which the node should be positioned before or after, or an index. The index is the fixed position or 0 which means first or a somewhat high number means last. The following will move the **Sessions** first among the **DBA** child nodes.

```
<GroupNode type="Sessions" order-before="0"/>
<!--The following will move the "Locks" node before "Sessions"-->
<GroupNode type="Locks" order-before="Sessions"/>
```

22.3.3 Extending Actions

Extending ActionGroup and Action elements follow the same rules as for extending ObjectsTreeDef (see page 335) section. The following example show removing the **oracle-schema-stringsearch** action for the **Schema** object type in the parent profile. A new **ActionGroup**: **Extended Schema Actions** is added with a single new **Action**: **sample-schema-sample-action**.

```
<ObjectsActionDef extends="true">
   <ActionGroup type="Schema">
      <!--Remove action from parent profile for "Schema"-->
      <Action id="oracle-schema-stringsearch" action="drop"/>
      <!--Adds an "Extended Schema Actions" sub menu in the "Schema" actions menu-->
      <ActionGroup type="sample-schema-test" label="Extended Schema Actions">
         <!--Sample action that does nothing-->
         <Action id="sample-schema-sample-action" label="Sample Action"
                 reload="true" resetcatalogs="true" icon="remove">
            <Input label="Text Field" name="textField" style="text" editable="true"/>
            <Command>
```

```
               <SQL><![CDATA[Sample Action "${textField}"]]></SQL>
            </Command>
            <Confirm>
               Really run Sample Action "${textField}"?
            </Confirm>
            <Result>
               Sample Action "${textField}" processed!
            </Result>
         </Action>
      </ActionGroup>
   </ActionGroup>
</ObjectsActionDef
```

22.3.4 Extending Object Views

Extending **<ObjectView> and <DataView>** elements follow the same rules as previously explained.

```
<ObjectsViewDef extends="true">
   <!--Schema is dropped in the Oracle profile. Redefine it here and show the-->
   <!--dictionary views. That data is really not associated with the single schema-->
   <!--defined in this profile but is a way to have it quickly accessed from-->
   <!--a single node.-->
   <ObjectView type="Schema">
      <DataView id="sample-schema-dict" label="Dictionary"
                icon="sample-schema-dict" viewer="grid">
         <Command idref="sample.getDict"/>
         <Message>
            <![CDATA[
<html>
Simple viewer showing all dictionary tables with description. Easily accessed
by opening the Schema viewer since that is empty anyway in the
parent <b>oracle</b> profile.
</html>
            ]]>
         </Message>
      </DataView>
   </ObjectView>
</ObjectsViewDef>
```

22.3.5 Remove an Element

Removing an object in the parent profile is easy, just add the **action="drop"** attribute to any of **GroupNode**, **DataNode**, **ObjectView**, **DataView**, **ActionGroup** and **Action** elements. If there are any sub elements for the object being dropped, these are also removed.

22.3.6 Complete sample Database Profile

This document describe the different parts of a extended sample database profile for an Oracle database. Here follow the complete sample database profile. If you would like to test it with your Oracle database, then just copy and paste it to your *$HOME\.dbvis\ext\profiles* folder as *sample-oracle.xml*. In DbVisualizer open **Connection Properties** for your Oracle connection and click **Database Profiles** category. In the list of profiles chose **sample-oracle**, click **Apply** and then connect.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE DatabaseProfile SYSTEM "dbvis-defs.dtd">
<!--
   Copyright (c) DbVis Software AB. All Rights Reserved.
-->
<DatabaseProfile
        desc="Sample profile extending the Oracle profile"
        extends="oracle"
        minver="9.5">
   <!--Commands used in this profile-->
   <Commands extends="true">
      <Command id="sample.getLoginSchema">
         <SQL>
            <![CDATA[
select '${schema}' as schema from dual
            ]]>
         </SQL>
      </Command>
      <Command id="sample.getDict">
         <SQL>
            <![CDATA[
select * from dict order by table_name
            ]]>
         </SQL>
      </Command>
   </Commands>
   <ObjectsActionDef extends="true">
      <ActionGroup type="Schema">
         <!--Remove action from parent profile for "Schema"-->
         <Action id="oracle-schema-stringsearch" action="drop"/>
         <!--Adds an "Extended Schema Actions" sub menu in the "Schema" actions menu-->
         <ActionGroup type="sample-schema-test" label="Extended Schema Actions">
            <!--Sample action that does nothing-->
            <Action id="sample-schema-sample-action" label="Sample Action" reload="true"
                    resetcatalogs="true" icon="remove">
               <Input label="Text Field" name="textField" style="text" editable="true"/>
               <Command>
                  <SQL><![CDATA[Sample Action "${textField}"]]></SQL>
               </Command>
               <Confirm>
                  Really run Sample Action "${textField}"?
               </Confirm>
               <Result>
```

```
                  Sample Action "${textField}" processed!
               </Result>
            </Action>
         </ActionGroup>
      </ActionGroup>
   </ObjectsActionDef>
   <ObjectsTreeDef extends="true">
      <!--The following "Schema" definition shows the login schema directly below-->
      <!--the Database Connection for faster access. It is limited to only show-->
      <!--tables (by setting the "Table" DataNode to isLeaf="true")-->
      <DataNode type="Schema" label="My Schema: ${sample.getLoginSchema.SCHEMA}" order-before="0">
         <SetVar name="schema" value="${sample.getLoginSchema.SCHEMA}"/>
         <Command idref="sample.getLoginSchema">
            <Input name="schema" value="${#db.loginSchema}"/>
         </Command>
         <DataNode type="Table" label="${oracle.getTables.TABLE_NAME}" isLeaf="true">
            <SetVar name="objectname" value="${oracle.getTables.TABLE_NAME}"/>
            <SetVar name="rowcount" value="true"/>
            <SetVar name="acceptInQB" value="true"/>
            <Command idref="oracle.getTables">
               <Input name="owner" value="${schema}"/>
               <Input name="temporary" value="N"/>
               <ProcessDataSet action="sortcolumn" index="TABLE_NAME"/>
               <Filter index="TABLE_NAME" label="Table"/>
            </Command>
            <!--These are needed for the viewers defined in the parent profile-->
            <!--associated with the "Table" type-->
            <SetVar name="theTableName" value="${objectname}"/>
            <SetVar name="theParentName" value="${objectname}"/>
            <SetVar name="triggersCondition" value="and table_name = '${theTableName}'"/>
         </DataNode>
      </DataNode>
      <!--Renaming the standard Schemas node to "All Schemas"-->
      <GroupNode type="Schemas" label="All Schemas"/>
      <!--The main purpose with the following is to add a "Role" child DataNode -->
      <!--for each "User". In the parent Oracle profile there are no child-->
      <!--nodes below "User". To handle this all nodes from "DBA" down to "User"-->
      <!--must be specified. The only requirement is that the type attribute is-->
      <!--specified and that it match the type in the parent profile.-->
      <!--In addition, this example specify the label attribute for some of the-->
      <!--nodes just to show that overridden attributes will replace any parent-->
      <!--equvivalent node attributes.-->
      <GroupNode type="DBA" label="Database Administration">
         <GroupNode type="Users">
            <!--The "User" type don't allow child nodes in the parent profile.-->
            <!--Setting isLeaf="false" is needed to override this and allow the-->
            <!--new "Role" child node-->
            <DataNode type="User" isLeaf="false">
               <!--Here comes the new child "Role" DataNode-->
               <DataNode type="Role" label="${oracle.getGranteeRoles.GRANTED_ROLE} - ext" isLeaf="
true">
                  <SetVar name="objectname" value="${oracle.getGranteeRoles.GRANTED_ROLE}"/>
                  <Command idref="oracle.getGranteeRoles">
```

```
                     <Input name="grantee" value="${objectname}"/>
                  </Command>
               </DataNode>
            </DataNode>
         </GroupNode>
         <!--The following are specified only to re-define the position of the-->
         <!--"Locks" and "Sessions" nodes. One of "order-before" and "order-after" -->
         <!--attributes are used to either identify a type for which the node should-->
         <!--be positioned before or after, or an index. The index is the fixed position-->
         <!--or 0 which means first or a somewhat high number means last.-->
         <!--The following will move the "Sessions" first among the "DBA"-->
         <!--child nodes-->
         <GroupNode type="Sessions" order-before="0"/>
         <!--The following will move the "Locks" node before "Sessions"-->
         <GroupNode type="Locks" order-before="Sessions"/>
      </GroupNode>
   </ObjectsTreeDef>
   <ObjectsViewDef extends="true">
      <!--Schema is dropped in parent profile. Re-define it here and show the-->
      <!--dictionary views. That data is really not associated with the single schema-->
      <!--defined in this profile but is a way to have it quickly accessed from-->
      <!--a single node.-->
      <ObjectView type="Schema">
         <DataView id="sample-schema-dict" label="Dictionary" icon="sample-schema-dict" viewer="
grid">
            <Command idref="sample.getDict"/>
            <Message>
                  <![CDATA[
<html>
Simple viewer showing all dictionary tables with description. Easily accessed
by opening the Schema viewer since that is empty anyway in the
parent <b>oracle</b> profile.
</html>
               ]]>
            </Message>
         </DataView>
      </ObjectView>
   </ObjectsViewDef>
</DatabaseProfile>
```

22.4 Top level XML Elements

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

The top level XML elements in a database profile is as follows:

- InitCommands (see page 346) (optional)
  Defines SQLs that are executed before the profile is being loaded,
- Commands (see page 349)
  Defines the SQLs for the ObjectsTreeDef, ObjectsViewDef and ObjectsActionDef,
- ObjectsActionDef (see page 386) (optional)
  Defines actions for object types,
- ObjectsTreeDef (see page 359)
  Defines the structure and what objects should be visible in the objects tree,
- ObjectsViewDef (see page 369)
  Defines the object views for a specific object type.

All database connections loads a database profile from an XML file and if there is no matching database profile, the **generic** profile is used. This profile use standard JDBC metadata requests to obtain information about the (some) objects in the database. The generic profile is located in `DBVIS-HOME\resources\profiles\generic.xml`.

22.4.1 XML template

The following show an overview of a database profile with the top level XML elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DatabaseProfile SYSTEM "dbvis-defs.dtd">
<DatabaseProfile desc="Profile for Sybase ASE"
                 version="$Revision: 20369 $"
                 date="$Date: 2016-05-24 17:29:25 +0200 (Tue, 24 May 2016) $"
                 minver="9.5"
                 extends="generic">

  <InitCommands extends="true">
    ...
  </InitCommands>
  <Commands extends="true">
    ...
  </Commands>
  <ObjectsActionDef extends="true">
    ...
  </ObjectsActionDef>
  <ObjectsTreeDef extends="false">
    ...
  </ObjectsTreeDef>
  <ObjectsViewDef extends="true">
    ...
  </ObjectsViewDef>
</DatabaseProfile>
```

The DOCTYPE identifier at row 2 defines the DTD that is used to validate the XML.

The root element for the database profile framework is the DatabaseProfile (see page 345) element setting various attributes. Continue to the next sections for information about XML elements forming a database profile.

22.4.2 XML element - DatabaseProfile

Only in DbVisualizer Pro

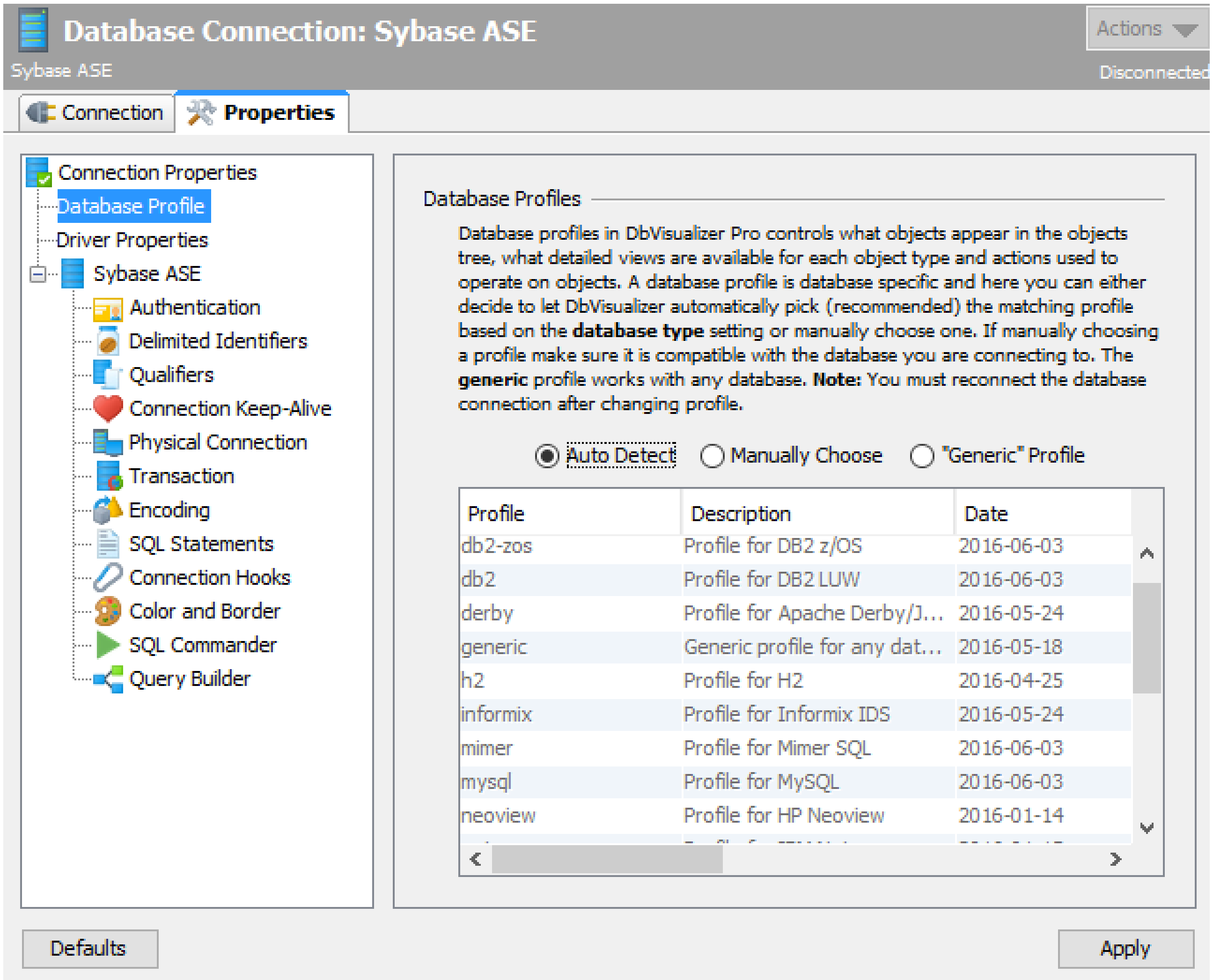
This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

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: 20369 $"
                 date="$Date: 2016-05-24 17:29:25 +0200 (Tue, 24 May 2016) $"
                 minver="9.5"
                 extends="generic">
  ...
</DatabaseProfile>
```

Attribute	Description
desc	The description of the profile
version	The version of the profile
date	The timestamp when the profile was last modified
minver	The minimum version of DbVisualizer that is required for the profile to work
extends	The parent profile that is extended. In most situations at least **generic** should be specified

Most attributes except **extends** are informative and are displayed in the **Database Profile** list when selecting the connection properties for a database connection:



Database Profiles

Database profiles in DbVisualizer Pro controls what objects appear in the objects tree, what detailed views are available for each object type and actions used to operate on objects. A database profile is database specific and here you can either decide to let DbVisualizer automatically pick (recommended) the matching profile based on the **database type** setting or manually choose one. If manually choosing a profile make sure it is compatible with the database you are connecting to. The **generic** profile works with any database. **Note:** You must reconnect the database connection after changing profile.

● Auto Detect ○ Manually Choose ○ "Generic" Profile

Profile	Description	Date
db2-zos	Profile for DB2 z/OS	2016-06-03
db2	Profile for DB2 LUW	2016-06-03
derby	Profile for Apache Derby/J...	2016-05-24
generic	Generic profile for any dat...	2016-05-18
h2	Profile for H2	2016-04-25
informix	Profile for Informix IDS	2016-05-24
mimer	Profile for Mimer SQL	2016-06-03
mysql	Profile for MySQL	2016-06-03
neoview	Profile for HP Neoview	2016-01-14

22.4.3 XML element - InitCommands

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

The **InitCommands** element define initialization commands (see page 349) that are executed just before the rest of the database profile is loaded. These commands are typically used to determine characteristics of the target database and database session. The result of these commands are stored in **variables** that can be used in conditions (see page 409) 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 in the database. If the user have limited privileges then some object types, views and actions should be disabled.

Multiple commands may be defined in the InitCommands element and these are executed in order.

The following sample is from the HP Neoview database profile. The main purpose with its commands is to first determine the database version by quering a system table. Based on the database version a condition controls which of two queries will be executed to find out another property from the database. The result of the executed query is stored in a the **METACAT** variable.

```
<InitCommands extends="true">
   <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 _ISO88591'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>
```

The **extends="true"** attribute specifies that the list of commands will extend the list of commands defined in the profile being extended (see page 333).

Initialization commands are processed in two stages:

1. First stage is to execute all commands having the attribute **method="runBeforeConditionsEval"** set. As the attribute reveal, these commands are execute before any conditions are evaluated,
2. The second and last stage will execute all commands **with no** method="runBeforeConditionsEval" set. This time any conditions are evaluated.

The reason for these stages is that the processing of initialization commands may also rely on conditions.

Here is an example how the new **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 _ISO88591'NONSTOP_SQLMX_%'
    AND CAT_NAME NOT IN (_ISO88591'NSMWEB', _ISO88591'NVSCRIPT', _ISO88591'METRIC',
                         _ISO88591'MATRIX', _ISO88591'GENUSCAT', _ISO88591'MANAGEABILITY')
ORDER BY
    CATALOG_NAME
FOR READ UNCOMMITTED ACCESS
      ]]>
   </SQL>
</Command>
```

Here is another example for Oracle getting the **instance_type** property from the **v$parameter** table and put the value in the **INSTANCE_TYPE** variable.

```
<InitCommands extends="true">
   <Command id="oracle.initGetInstanceType" method="runBeforeConditionsEval">
      <SQL>
         <![CDATA[
select value from v$parameter where name = 'instance_type';
         ]]>
      </SQL>
      <Output id="INSTANCE_TYPE" index="1"/>
   </Command>
</InitCommands>
```

Below show that only if **INSTANCE_TYPE** have the value **RDBMS** schema objects should be displayed in the database objects tree:

```
<ObjectsTreeDef extends="false">
   <If test="#INSTANCE_TYPE eq 'RDBMS'">
      <GroupNode type="Schemas" label="Schemas">
         ...
      </GroupNode>
   </If>

   <GroupNode type="Properties" label="Session Properties" isLeaf="true"/>
   <GroupNode type="DBA" label="DBA Views">
```

```
      ...
   </GroupNode>
</ObjectsTreeDef>
```

One more example showing the use of **<OnError>** and **<Message>** element **<InitCommands>**. Based on the database major version it runs one of two SQLs. If any **924** error (table not found) is raised the message will be logged in the DbVisualizer log. The `type="info"` defines that the log entry should be logged without further notification. Leaving the type out or setting it to `type="warning"` will also raise the error balloon in the DbVisualizer tool. Use this only when really necessary.

```
<InitCommands extends="true">
   <Command id="oracle.initInstanceNumber">
      <If test="#util.isDatabaseVersionLT(9)">
         <SQL><![CDATA[select -1 from dual;]]></SQL>
         <Output name="OWN_INSTANCE_NUMBER" index="1"/>
      </If>
      <Else>
         <SQL><![CDATA[select instance_number from v$instance;]]></SQL>
         <Output name="OWN_INSTANCE_NUMBER" index="1"/>
      </Else>
      <OnError>
         <If test="#result.getErrorCode() eq 942" context="runtime">
            <Message type="info">
               <![CDATA[
Could not to retrieve own INST_ID (probably due to missing privileges) which is needed to browse
DBA->Sessions, etc. DbVisualizer will work fine but related features will be disabled
               ]]>
            </Message>
         </If>
      </OnError>
   </Command>
</InitCommands>
```

Click these links for more information about the command (see page 349) element and conditions (see page 409).

22.4.4 XML element - Commands

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

The **Commands** element is a simple grouping element for **Command** elements.

```
<Commands extends="true">
  <Command id="profileName.xxx">
    ...
  </Command>
</Commands>
```

The **extends="true"** attribute specifies that the list of commands will extend the list of commands defined in the profile being extended (see page 333).

XML element - Command

The main purpose with the **Command** element is to run a single **SQL statement** or a **script** of SQL statements. In most cases, the script should return a result set with 0 or multiple rows with the exception for actions (see page 386) which not necessarily need to return a result set, e.g., a "drop" action). The following show the command element, its attributes with default values, and valid sub elements.

```
<Command     id="sybase-ase.getLogins"
         method="dynamic"
       exectype="script"
 processmarkers="true"
     autocommit="true"
    whensuccess="commit"
      whenerror="rollback">

   <SQL>
      ...
   </SQL>

   <Input>
      ...
   </Input>

   <Output>
      ...
   </Output>

   <Filter>
      ...
   </Filter>
```

```
    <ProcessDataSet>
       ...
    </ProcessDataSet>
    <ProcessSQLWarning>
       ...
    </ProcessSQLWarning>
</Command>
```

Attribute	Description
id	The command element is identified with a unique **id** attribute. This id is referred in ObjectsTreeDef (see page 359), ObjectsViewDef (see page 369) and ObjectsActionDef (see page 386) definitions using the **idref** attribute. The id naming convention for command elements is to prefix with the name of the profile and a dot. Example **oracle.xxx**, **sqlserver.xxx**, and so on.
method	- **dynamic** this is the default value and define that the SQL is a dynamic SQL statement as opposed to setting it to JDBC which defines that the SQL is really a JDBC meta data call rather than SQL, - jdbc See description for dynamic, - runBeforeConditionsEval this value is only considered if the command is define in the InitCommands (see page 346) section.
exectype	- **script** - asis - explain The default behavior is that the SQL may contains multiple SQL statements each delimited by a semi colon (';'). Set this attribute to false to disable multiple SQL statements.
autocommit	- **true** - false
whensuccess	- commit - rollback - ask

whenerror	• commit • rollback • ask • ignore

The following command queries 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 that profiles can be extended and id's need to be unique.

> This SQL example show a command with a **SELECT** statement using **column aliases**. If no aliases are specified the column names should be used to refer the data.

Result Set

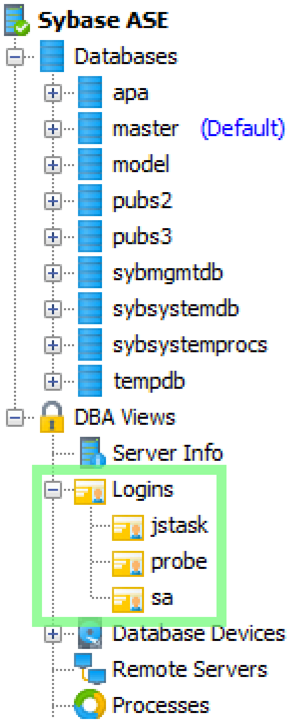
This is the result set for the previous query:

Name	SUID	Default Database	Full Name	Default Language	CPU Time	I/O Time	Password Set
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

How DbVisualizer handle 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 node in the tree. If executed in the object view, it is the viewer component that decide how the result will be displayed.

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 result set. The end user will see these objects (nodes) by some descriptive label, as defined in the ObjectsTreeDef. 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 ObjectsViewDef.

The following example put some light on this. Consider the previous result set and that it is used to create objects in the database objects tree. The end user will see the following in DbVisualizer. (The label for each row is the name column in the result set.):



Each of the **jstask**, **probe** and **sa** 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.Default Database,** 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 (see page 359) element definition presenting **jstask**, **probe** and **sa** nodes in the previous screenshot use the associated data for the **label** as follows:

```
<DataNode type="Login" label="${sybasease.getLogins.Name}" isLeaf="true">
   <SetVar name="objectname" value="${sybasease.getLogins.Name}"/>
   <Command idref="sybasease.getLogins">
      <Output id="sybasease.getLogins.Name" index="1"/>
      <Output id="sybasease.getLogins.suid" index="2"/>
   </Command>
</DataNode>
```

XML element - Input

The **Input** sub element for a Command is only used when a command is being referred with the **idref** attribute in any of ObjectsActionDef (see page 386), ObjectsTreeDef (see page 359) or ObjectsViewDef (see page 369). It has no effect specifying it for a Command in the Commands section .

There are two types of commands, with and without dynamic input. The difference is that dynamic commands accepts input data that is typically used to form the **WHERE** clause in SELECT SQLs. The previous example illustrates a static SELECT statement (without dynamic data).

To allow for dynamic input, just add variables at the positions (can be anywhere) in the SQL statement that should be replaced with 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>
```

This example add 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 that specify the **Input** sub elements to **map values** to the variables defined in the SQL.

```
<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 refer the command via the **idref** attribute which is then matched with the corresponding **id** for the Command).

The **${name}** variable in the SQL will be replaced with string **sa**.

The value for the **${suid}** variable will in this case get the value of another variable, **sybase-ase.getProcesses.suid**. So where is this variable defined? As explained in the Result Set (see page 352) section, all the data for a row in the result set is associated with the corresponding node in the database objects tree. In addition, it is possible to use all the data kept by the node and even its parent nodes (as presented in the objects tree) in the input to commands. So to evaluate the **${sybase-ase.getProcesses.suid}** variable, DbVisualizer first look for the variable in the current node. If it doesn't exist, it continues to look through the parent nodes until it reaches the root, which is the **Connections** node 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 use its value and stops searching.

XML element - Output

As mentioned earlier, a specific column value in a result set row is referenced by the name of the column and then 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.

```
<Output index="1" name="sybase-ase.getLogins.Name"/>
<Output index="2" name="sybase-ase.getLogins.suid"/>
```

(The index attribute accepts either the name of the column or index number in the result set starting from the first column at index 1).

Filter

The Filter element assists the Database Objects Tree filtering what fields are available for the user to filter on. The label attribute for DataNode is always available for filtering. Declaring a specific Filter is useful if for example using the label1 attribute to show additional information about an object and it should be possible to filter its label.

Here is an example of the Column sub-node to Table objects showing a filter definition:

```
<GroupNode type="Columns" label="Columns">
   <DataNode type="Column" label="${getColumnDefinitions.COLUMN_NAME}"
             label1="${getColumnDefinitions.TYPE_NAME}" isLeaf="true"
             icon="#dataMap.get('getColumnDefinitions.IS_PRIMARY_KEY') eq true ? 'PrimaryKey' : '
Column'">
      <SetVar name="objectname" value="${getColumnDefinitions.COLUMN_NAME}"/>
      <Command idref="getColumnDefinitions">
         <Input name="schema" value="${schema}"/>
         <Input name="objectname" value="${theTableName}"/>
         <Input name="tableType" value="Table"/>
         <Filter index="TYPE_NAME" label="Type"/>
      </Command>
   </DataNode>
</GroupNode>
```

Above is filter is defined for the **TYPE_NAME** column in the result set. The label for it as it will appear in the filtering pane in DbVisualizer is **Type**.

The **Show Default Database/Schema** pre-defined filter in DbVisualizer requires a Filter for the **DataNode type="schema"** command. The label for this filter must be **Name**:

```
<
DataNode

type

="Schema"

label

="${oracle.getSchemas.TABLE_SCHEM}"
>
   <
SetVar

name

="schema"

value

="${oracle.getSchemas.TABLE_SCHEM}"
/>
   <
Command
```

```
    idref

    ="oracle.getSchemas"
    >
          <
    Filter

    index

    ="TABLE_SCHEM"

    label

    ="Name"
    />
       </
    Command
    >

     ...
```

ProcessDataSet

The ProcessDataSet element is used to process the result set generated by a command. This process is performed just after the result set has been retrieved and before any **Output** elements are handled. Examples:

```
<ProcessDataSet action="renamecolumn" index="2" name="NewName"/>
<ProcessDataSet action="dropcolumn" index="AUTO_INCREMENT"/>
<ProcessDataSet action="dropidenticalcolumns" name="datname"/>
<ProcessDataSet action="addcolumn" index="Added Column" value="Add column with value 'Static string
${dbvis-jdbcURL}'"/>
<ProcessDataSet action="sortcolumn" index="LAST_NAME, FIRST_NAME"/>
<ProcessDataSet action="distinct" index="Country"/>
<ProcessDataSet action="removeisnullrows" index="VALUE"/>
<ProcessDataSet action="removerowsifequalto" index="DATA_LENGTH" value="16384"/>
<ProcessDataSet action="trimcolumn" index="PAD_COLUMN"/>
<ProcessDataSet action="printdataset"/>
```

Explanation of the actions. The index attribute for all actions identifies the column in the result set by either number starting with 1 for first column or the column name.

action	Description
renamecolumn	Renames the result set column identified by the index
dropcolumn	Drops the specified column
dropidenticalcolumns	Drops identically named columns (keeps first)
addcolumn	Adds a new column to all rows with the value specified in the value attribute. The value may contain variables using the ${...} notation
sortcolumn	Enter one or several column names (or indexes) separated with command (',') which are used to sort the result set
distinct	This action is used to remove all duplicates identified by the specified column index. What rows are removed may be different from time to time since matching is done on a single column value
removeisnullrows	Removes the row if the value in the specified result set column is null
removerowsifequalto	Removes the row if the data in the specified column is equal to the specified value
trimcolumn	Trims the data in the specified column index by removing all leading and trailing whitespaces
printdataset	Prints the result set to the debug log which is useful during profile development. Debug DbVisualizer must be enabled in the Tool->Debug Window.

ProcessSQLWarning

The ProcessSQLWarning element will look for any SQLWarning raised during execution of the command and convert it to a result set.

```
<InitCommands extends="true">
   <Command id="netezza.supportsMultipleSchemas">
      <SQL>
         <![CDATA[
show enable_schema_dbo_check
         ]]>
      </SQL>
      <ProcessSQLWarning action="converttodataset"/>
      <Output index="1" name="SUPPORTS_MULTIPLE_SCHEMAS"/>
   </Command>
</InitCommands>
```

Note: The SQLWarning support is quite brute as it will remove any other data sets produced by the command, and insert the SQLWarning alone as the only data set having a single 0,0 cell with the complete warning string.

Usage in <If>:

```
<If test="#SUPPORTS_MULTIPLE_SCHEMAS.matches('^.*ENABLE_SCHEMA_DBO_CHECK is [12]$')">
   ...
</If>
```

22.4.5 XML element - ObjectsTreeDef

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

The **ObjectsTreeDef** element section controls how the database objects tree should be presented and which commands should be executed to form its content (nodes).

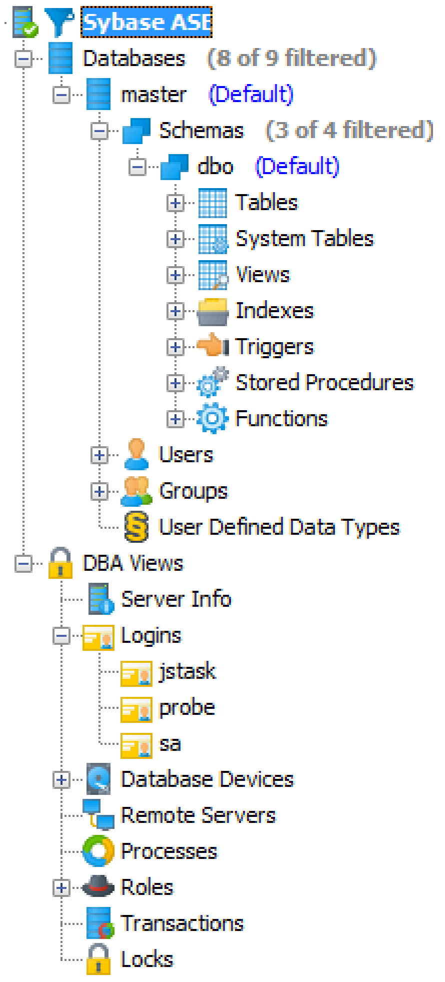
- XML element - GroupNode (see page 362)
- XML element - DataNode (see page 363)
  - XML element - Command (see page 366)
  - XML element - Filter (see page 366)
  - XML element - SetVar (see page 368)

```
<ObjectsTreeDef extends="false">
   <GroupNode type="xxx" label="xxx">
      <DataNode type="yyy" label="yyy">
         ...
      </DataNode>
   </GroupNode>
</ObjectsTreeDef>
```

Setting the **extends** attribute to **"true"** attribute specifies that the ObjectsTreeDef will extend the ObjectsTreeDef definition in the profile being extended (see page 333).

The mapping between the graphical representation in DbVisualizer and its ObejctsTreeDef XML is as quite straight forward:

Representation in DbVisualizer	XML specification



```
<ObjectsTreeDef extends="
false">
  <GroupNode type="
Databases">
    <DataNode type="
Catalog">
      <GroupNode type="
Schemas">
        <DataNode type="
Schema">
          <GroupNode type=
"Tables">
            <DataNode type
="Table"/>
          </GroupNode>
          <GroupNode type=
"SystemTables">
            <DataNode type
="SystemTable"/>
          </GroupNode>
          <GroupNode type=
"Views">
            <DataNode type
="View"/>
          </GroupNode>
          <GroupNode type=
"Indexes">
            <DataNode type
="Index"/>
          </GroupNode>
          <GroupNode type=
"Triggers">
            <DataNode type
="Trigger"/>
          </GroupNode>
          <GroupNode type=
"Procedures">
            <DataNode type
="Procedure"/>
          </GroupNode>
          <GroupNode type=
"Functions">
            <DataNode type
="Function"/>
          </GroupNode>
        </DataNode>
      </GroupNode>
      <GroupNode type="
Users">
```

Representation in DbVisualizer	XML specification

	`<DataNode type="User">`
	`</GroupNode>`
	`<GroupNode type="Groups">`
	`<DataNode type="Group"/>`
	`</GroupNode>`
	`<GroupNode type="Types"/>`
	`</DataNode>`
	`</GroupNode>`
	`<GroupNode type="DBA">`
	`<GroupNode type="ServerInfo"/>`
	`<GroupNode type="Logins">`
	`<DataNode type="Login"/>`
	`</GroupNode>`
	`<GroupNode type="Devices">`
	`<DataNode type="Device"/>`
	`</GroupNode>`
	`<GroupNode type="RemoteServers"/>`
	`<GroupNode type="Processes"/>`
	`<GroupNode type="ServerRoles">`
	`<DataNode type="ServerRole"/>`
	`</GroupNode>`
	`<GroupNode type="Transactions"/>`
	`<GroupNode type="Locks"/>`
	`</GroupNode>`
	`</ObjectsTreeDef>`

The screenshot in the above example show 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 how **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 screenshot is not listed in the example XML). The difference between a GroupNode and a DataNode is that GroupNode represent a static object in the tree while DataNode is dynamically created based on result sets produced by running a SQL statement. See GroupNode as a container holding other GroupNodes and DataNodes.

The database objects tree in DbVisualizer is the core visual component and it is the place where the user open object details and launch actions. To connect object actions (see page 386) and object views (see page 369) with a node in the objects tree, the **type** attribute is used. The type should be a descriptive word for a node such as **Table**, **Schemas**, **MaterializedQueryTable**, and so on. The type also map to a predefined icon (see page 407). Check the database profile utilities (see page 413) for information how to show all bundled icons and their type mappings.

There is no limitation on the number of levels in the objects tree expressed by nesting GroupNode and DataNode elements. A good rule is as always to keep it intuitive, simple and clean.

XML element - GroupNode

The GroupNode element represents a static object in the tree. A GroupNode do not 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 have the following attributes.

```
<GroupNode type="SystemTables" label="System Tables" isLeaf="false">
  ...
</GroupNode>
```

The **isLeaf** attribute is optional (default is **false**) and controls whether the GroupNode may have any child objects or not. It can always be set to false, the effect in the visual database objects tree is then that an expand handle always will be visible next to the icon, even if the node don't have any child nodes.

> ℹ If isLeaf is set to true and there are child Group and/or Data -nodes, these will not appear. The result may cause some frustration during the design of the database profile.

The complete set of attributes for the DataNode element:

Attribute	Value	Description
type		The type of node
label		The visual label for the node
label1		Optional label which shows additional information about the object
isLeaf	true/ **false**	Specifies if the node cannot have child objects

Attribute	Value	Description
icon		Icons are typically mapped using the type attribute with an icon name in the icon.prefs (see page 407) file(s). The **icon** attribute for a GroupNode can be set to specify an alternative icon. Conditions can be used to identify alternate icons (see below)
drop-label-not-equal		Do not add the node if the label is not equal to this value or variable
action	drop	**drop** is useful when extending another database profile to remove the DataNode and all its child nodes
order-before		Specifies the order of this GroupNode among a collection of nodes having the same parent node. It can either be an index starting at 0 (first) or a node type. Ex. **order-before="Views"** will order this GroupNode before nodes defined by the **type="Views"** attribute
order-after		Specifies the order of this GroupNode among a collection of nodes having the same parent node. It can either be an index starting at 0 (first) or a node type. Ex. **order-after="0"** will order this GroupNode after the first node definition

XML element - DataNode

The DataNode element feeds the tree with nodes produced by a **Command**. The example in the Command (see page 350) section querying for all logins in Sybase ASE look as follow in the ObjectsTreeDef:

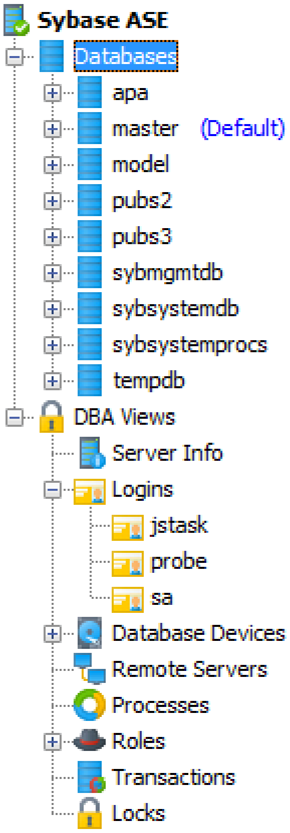
```
<GroupNode type="Logins" label="Logins">
  <DataNode type="Login" label="${sybase-ase.getLogins.Name}" isLeaf="true">
    <Command idref="sybase-ase.getLogins"/>
  </DataNode>
</GroupNode>
```

First, there is a **GroupNode** element with the purpose to group all child objects in a Logins node. The **DataNode** in this example have the same attributes as the GroupNode, the type is however singular **Login** instead of plural **Logins** (as it is for the GroupNode). This difference is important when the user decide to open one of the nodes, since the object view will show the matching views based on the object type. For Logins a list of all logins is displayed while opening a Login, details for that specific login is displayed.

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 introduce the use a variable. The real value for the label will, in this example, be the value in the **Name** column produced by the **sybase-ase.getLogins** command (variable names are automatically prefixed with the command id that produced them).

The **Command** element uses the **idref** attribute to identify the command that should be executed. The command in this case produce a result set (see page 352) with 3 rows and 8 columns. The result will be two nodes for each row, with the label of the **Name** column in the result set.



The label1 attribute can be set to any other valid variable, a combination of several variables or even static text:

```
label1="${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:

Attribute	Value	Description
type		The type of node
label		The visual label for the node

Attribute	Value	Description
label1		Optional label which shows additional information about the object
icon		Icons are typically mapped using the type attribute with an icon name in the icon.prefs (see page 407) file(s). The **icon** attribute for a DataNode can be set to specify an alternative icon. A condition can be specified which is used to choose an icon based on evaluating other variables for the node. Ex. `icon="#dataMap.get('mimer.getIndexes.IS_UNIQUE').equals('YES') ? 'uniqueIcon' : (#dataMap.get('mimer.getIndexes.IS_UNIQUE').equals('FK') ? 'fkIcon' : 'indexIcon')"`
actiontype		Object type used for object actions
isLeaf	true/ **false**	Specifies if the node cannot have child objects
drop-label-not-equal		Do not add the node if the label is not equal to this value or variable
stop-label-not-equal		The node will be a leaf if the label doesn't match the specified value or variable value
warnstate		A condition expression returning either true or false. For true, show a warning overlay icon for the node. Ex: `errorState="!#dataMap.get('oracle.getTriggers.STATUS'). equals('ENABLED')"`
errorstate		A condition expression returning either true or false. For true, show an error overlay icon for the node. Ex: `errorState="!#dataMap.get('oracle.getObjectsByType.STATUS'). equals('VALID')"`
is-empty-output	**continue** /stop	If result set is empty, use this to control whether child GroupNode/ DataNodes should be added anyway
action	**keep**/ drop	**drop** is useful when extending another database profile to remove the DataNode and all its child nodes
order-before		Specifies the order of this DataNode among a collection of nodes having the same parent node. It can either be an index starting at 0 (first) or a node type. Ex. **order-before="View"** will order this node before nodes defined by the **type="View"** attribute
order-after		

Attribute	Value	Description
		Specifies the order of this DataNode among a collection of nodes having the same parent node. It can either be an index starting at 0 (first) or a node type. Ex. **order-after="0"** will order this DataNode after the first node definition

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.

XML element - Command

The SQL used to generate the data used by the DataNodes are defined in the **Command** element.

A command is referenced by the **idref** attribute and that **id** must already be defined in the Commands (see page 349) section of the profile. For most DataNode definitions input must be supplied with the command and 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 syntax outlined in the result set (see page 352) section.

> For detailed information about the capabilities with the Command element, check the Command (see page 350) section.

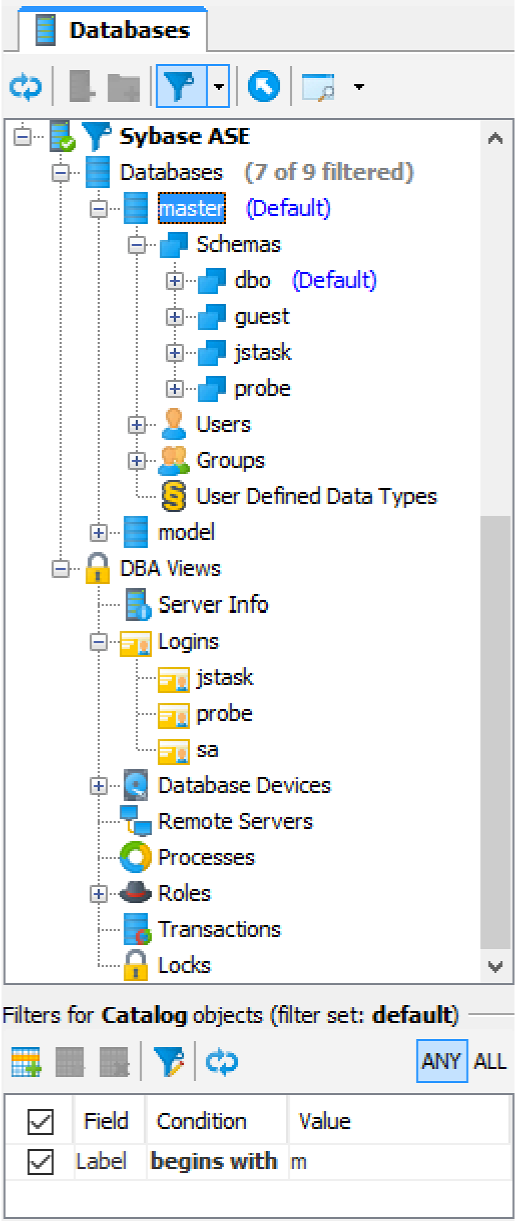
XML element - Filter

The Filter element is specific for Command elements that appear in the DataNode element. A filter define which data for a DataNode that can be searched in filter. This filter functionality is commonly referred as the Database Objects Tree Filtering (see page 154) 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 included in the filtering criteria.



☑	Field	Condition	Value
☑	Label	**begins with**	m

```
<DataNode type="View" label="${sybase-ase.getViews.Name}" isLeaf="true">
  <Command idref="sybase-ase.getViews">
    <Filter index="TABLE_NAME" name="View Table"/>
  </Command>
</DataNode>
```

The above filter definition specifies a filter for the Catalog (database) object type. The **index** attribute should specify a **column name** or **index** in the result set while the **name** attribute specifies the **label** for the filter as it appears in the object type drop-down list.

XML element - SetVar

The SetVar element is used in the ObjectsTreeDef for GroupNode and DataNode elements. 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 the name attribute set to **catalog** or **schema**, respectively.

```
<DataNode type="Catalog" label="${getCatalogs.TABLE_CAT}">
  <SetVar name="catalog" value="${getCatalogs.TABLE_CAT}">
</DataNode>
```

All DataNodes except Catalog and Schema must use SetVar to set the **objectname** variable:

```
<DataNode type="View" label="${sybase-ase.getViews.Name}" isLeaf="true">
  <SetVar name="objectname" value="${sybase-ase.getViews.Name}">
  <SetVar name="rowcount" value="true">
</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 (see page 369) and object actions (see page 386). Its value should be the identifier for the object as it is identified in the database, for example a table name or view name.

The **rowcount** variable is optional (default is false) and controls whether the object supports showing row count information when Show/Hide Table Row Count (see page 131) right-click menu choice is enabled for the database connection.

Another optional variable (not shown in the example above) is named **acceptInQB** (default is false). If set to true , nodes of this type can be used in the Query Builder (see page 191). 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.

Variables defined with SetVar are by default invisible in for example the node form viewer (see page 369). 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**.

Using SetVar for GroupNode's is used to set static values (since GroupNode's doesn't execute a Command). This may be used to pass a static value for later use in an Action or DataView. See vertica.xml which illustrates using SetVar for GroupNode's.

22.4.6 XML element - ObjectsViewDef

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

The ObjectsViewDef element define all views for the object types in the objects tree. These views are displayed in the Object View (see page 33) 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 (see page 369) definition.

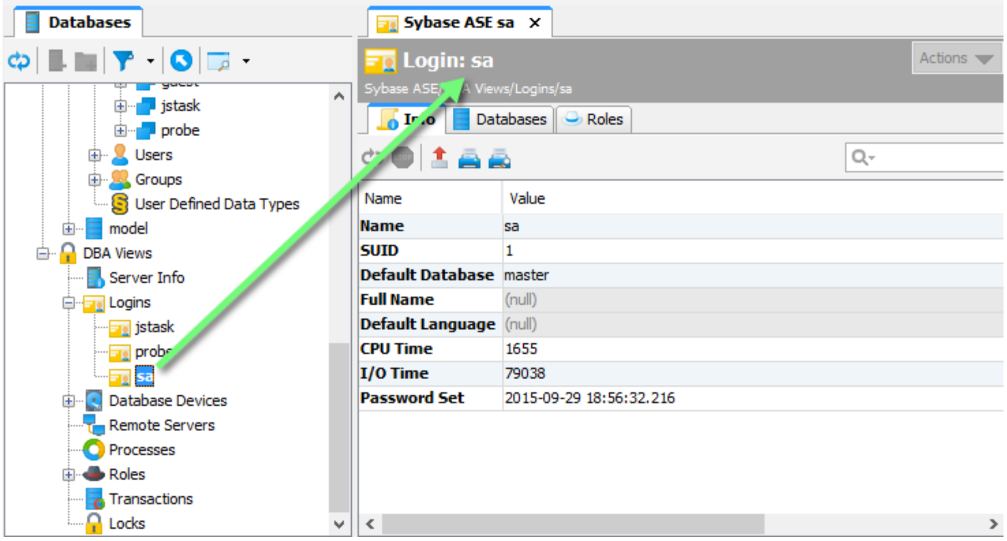
- XML element - ObjectView (see page 371)
- XML element - DataView (see page 372)
  - Viewers (see page 373)
    - Viewer - grid (see page 373)
      - Adding custom menu items in the grid (see page 375)
      - Setting initial max column width (see page 377)
    - Viewer - text (see page 378)
      - Specify what column to browse (see page 378)
      - Enable SQL formatting of the data (see page 379)
      - Adding newline to each row (see page 379)
    - Viewer - form (see page 379)
    - Viewer - node-form (see page 380)
      - Hiding columns (see page 381)
    - Viewer - table-refs (see page 381)
    - Viewer - tables-refs (see page 382)
    - Viewer - table-data (see page 383)
      - Disable data editing (see page 384)
    - Viewer - table-rowcount (see page 384)
  - XML element - Command (see page 385)
  - XML element - Input (see page 385)
  - XML element - Message (see page 385)

```
<ObjectsViewDef extends="true">
   <ObjectView type="xxx">
      <DataView id="yyy" label="yyy">
         ...
      </DataView>
   </ObjectView>
</ObjectsViewDef>
```

The **extends="true"** attribute specifies that this definition will extend the ObjectsViewDef definition in the database profile being extended (see page 333).

When an object is opened in the database tree (**sa** in the screenshot below) a corresponding object view tab is created (right in the sample). Each of the DataView elements in the ObjectView will appear as sub tabs in the object view tab. The selected object and its information is passed to each of the data views for processing and presentation. The following example show the Object View in DbVisualizer and its ObjectView element definition .

Representation in DbVisualizer	XML definition

	`<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">`

Name	Value
Name	sa
SUID	1
Default Database	master
Full Name	(null)
Default Language	(null)
CPU Time	1655
I/O Time	79038
Password Set	2015-09-29 18:56:32.216

Representation in DbVisualizer	XML definition

	`<Command idref="sybase-ase.getLoginRoles"/>`
	`</DataView>`
	`</ObjectView>`

The screenshot and the database tree show both the **Logins** node and its child nodes, **jstask**, **probe** and **sa**. These nodes are instances of the object types **Logins** (labeled Logins in the screenshot) and **Login** (the three sub nodes: **jstask**, **sa** and **probe**).

The ObjectView XML definitions above shows the data views for these two types, **Logins** and **Login**. Opening the node labeled **Logins** in the tree will show the object view for the **<ObjectView type="Logins">** definition while opening 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 displayed as tabs in the object view are (by label):

- Info
- Databases
- Roles

XML element - 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>
```

The **type** attribute value is used when a node is clicked in the database objects tree to map with the corresponding ObjectView definition. The following lists the attributes for ObjectView:

Attribute	Value	Description
type		The type of the ObjectView as declared in the GroupNode and DataNode elements in the ObjectsTreeDef section
action	drop	

Attribute	Value	Description
		drop is useful when extending another database profile to remove the ObjectView and all its child views

XML element - DataView

The DataView element is comparable with the DataNode (see page 369) element in the ObjectsTreeDef. It defines what SQL (command) should be executed, labeling, viewer type (presentation form) and other characteristics. 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="sybasease-login-info" icon="Info" label="Info" viewer="node-form"/>
  <DataView type="sybasease-login-databases" icon="Databases" label="Databases" viewer="grid">
    <Command idref="sybase-ase.getLoginDatabases"/>
  </DataView>
  <DataView type="sybasease-login-roles" icon="Roles" label="Roles" viewer="grid">
    <Command idref="sybase-ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```

All three DataView elements have a **viewer** attribute identifying how the data in the view should be be presented , e.g., as a grid or a form. See the next sections for a list of viewers. The following lists all attributes for DataView:

Attribute	Value	Description
id		Every DataView element must have a unique id which is not only unique in the current profile but also with all id's in extended profiles To make sure the id is unique use the following recommended format: **profileName-objectViewType-viewerLabel**. Ex: sybasease-login-databases (The id should not contain any empty space or special characters other than dash ("-")).
label		The label for the viewer as it will appear in the tab
icon		The icon as defined in the icons.prefs (see page 407) file(s)

Attribute	Value	Description
viewer		One of: grid, text, form, node-form, table-refs, tables-refs, table-date, table-rowcount, message, navigator, ddl, ProcedureViewer. See the viewers section in this document for more information
drop-label-not-equal		Drop the viewer if its label is not equal to the value of this attribute
class		Used to specify a custom Java class used as the viewer
classargs		Used to pass arguments to a custom viewer
action	drop	**drop** is useful when extending another database profile to remove the DataView
doclink		Relative HTML link to the related chapter in the users guide
order-before		Specifies the order of this DataView among a collection of viewers having the same parent ObjectView. It can either be an index starting at 0 (first) or a node type. Ex. **order-before="sybasease-login-databases"** will order this DataView before viewers defined by the **id="sybasease-login-databases"** attribute
order-after		Specifies the order of this DataView among a collection of viewers having the same parent ObjectView. It can either be an index starting at 0 (first) or a node type. Ex. **order-after="sybasease-login-databases"** will order this DataView after viewers defined by the **id="sybasease-login-databases"** attribute

Viewers

The viewer attribute for a DataView define how the data for the viewer 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.

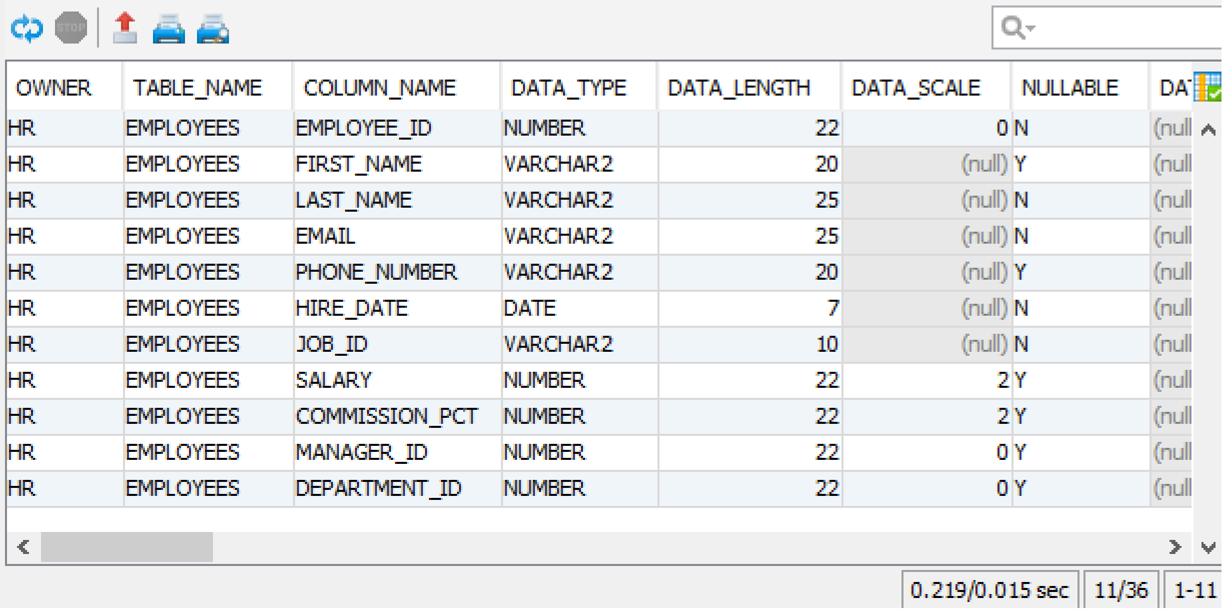
Viewer - grid

The **grid** viewer presents a result set in a grid with standard grid features such as search, copy, fit columns, export and so on. The result set is presented exactly as it is produced by the associated **Command** and any optional **Output** processing.

Here is a sample of the XML for the grid viewer:

```
<DataView type="oracle-columns-columns" icon="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 above definition.



OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_SCALE	NULLABLE	DA
HR	EMPLOYEES	EMPLOYEE_ID	NUMBER	22	0	N	(null
HR	EMPLOYEES	FIRST_NAME	VARCHAR2	20	(null)	Y	(null
HR	EMPLOYEES	LAST_NAME	VARCHAR2	25	(null)	N	(null
HR	EMPLOYEES	EMAIL	VARCHAR2	25	(null)	N	(null
HR	EMPLOYEES	PHONE_NUMBER	VARCHAR2	20	(null)	Y	(null
HR	EMPLOYEES	HIRE_DATE	DATE	7	(null)	N	(null
HR	EMPLOYEES	JOB_ID	VARCHAR2	10	(null)	N	(null
HR	EMPLOYEES	SALARY	NUMBER	22	2	Y	(null
HR	EMPLOYEES	COMMISSION_PCT	NUMBER	22	2	Y	(null
HR	EMPLOYEES	MANAGER_ID	NUMBER	22	0	Y	(null
HR	EMPLOYEES	DEPARTMENT_ID	NUMBER	22	0	Y	(null

0.219/0.015 sec | 11/36 | 1-11

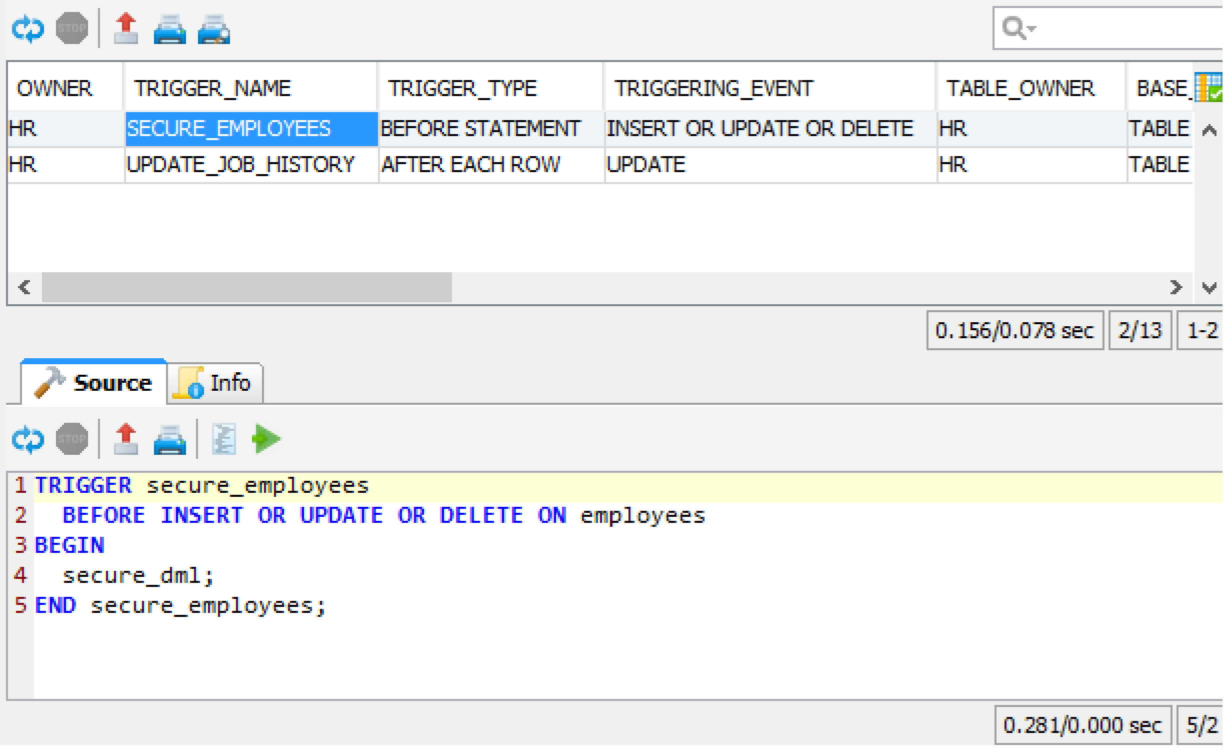
The nesting capability for grid viewers is really powerful, as it can be used to create a **drill-down** view of the data. Consider the scenario with a grid viewer showing all Trigger objects. Wouldn't it be nice to offer the user the capability to display the trigger source when selecting a row in the list? This is easily accomplished with the following definition:

```
<DataView id="oracle-table-triggers" icon="Trigger" label="Triggers" viewer="grid">
   <Command idref="oracle.getTriggers">
      <Input name="owner" value="${schema}"/>
      <Input name="condition" value="${triggersCondition}"/>
   </Command>
   <DataView id="oracle-table-triggers-source" icon="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 id="oracle-table-triggers-info" icon="Info" label="Info" viewer="node-form"/>
</DataView>
```

- The first DataView element define the top grid viewer labeled **Triggers** and the command to get the result set for it
- The next DataView is the **nested text viewer** labeled **Source**, 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 DataView labeled **Info** 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:



OWNER	TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_OWNER	BASE_
HR	SECURE_EMPLOYEES	BEFORE STATEMENT	INSERT OR UPDATE OR DELETE	HR	TABLE
HR	UPDATE_JOB_HISTORY	AFTER EACH ROW	UPDATE	HR	TABLE

```
TRIGGER secure_employees
  BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
  secure_dml;
END secure_employees;
```

Adding custom menu items in the grid

The grid right-click menu contain a lot of standard actions. Custom commands can be defined in the DataView element and these will appear last in the menu.

```
<Input name="menuItem" value="Open in New Tab...">
    <Input name="action" value="open-object-in-new-tab-command ${schema||OWNER}${object||TABLE_NAME}
"/>
</Input>
<Input name="menuItem" value="Open in Floating Tab...">
    <Input name="action" value="open-object-in-floating-tab-command ${schema||OWNER}${object||
TABLE_NAME}"/>
```

```
</Input>
   <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>
```

The **<Input name="menuItem">** element define a menu item entry that should appear in the grid right-click menu. The **value** for the menuItem is the **label** for the item as it will appear in the menu while the child **Input** element with **name="command"** is the **SQL command** that should be produced for all selected rows when the menu item is selected. Invoking a custom menu item will **not execute** the produced SQL directly but rather **copy the statements to a SQL Editor**. In the SQL Editor you will then need to manually execute the script and track the result.

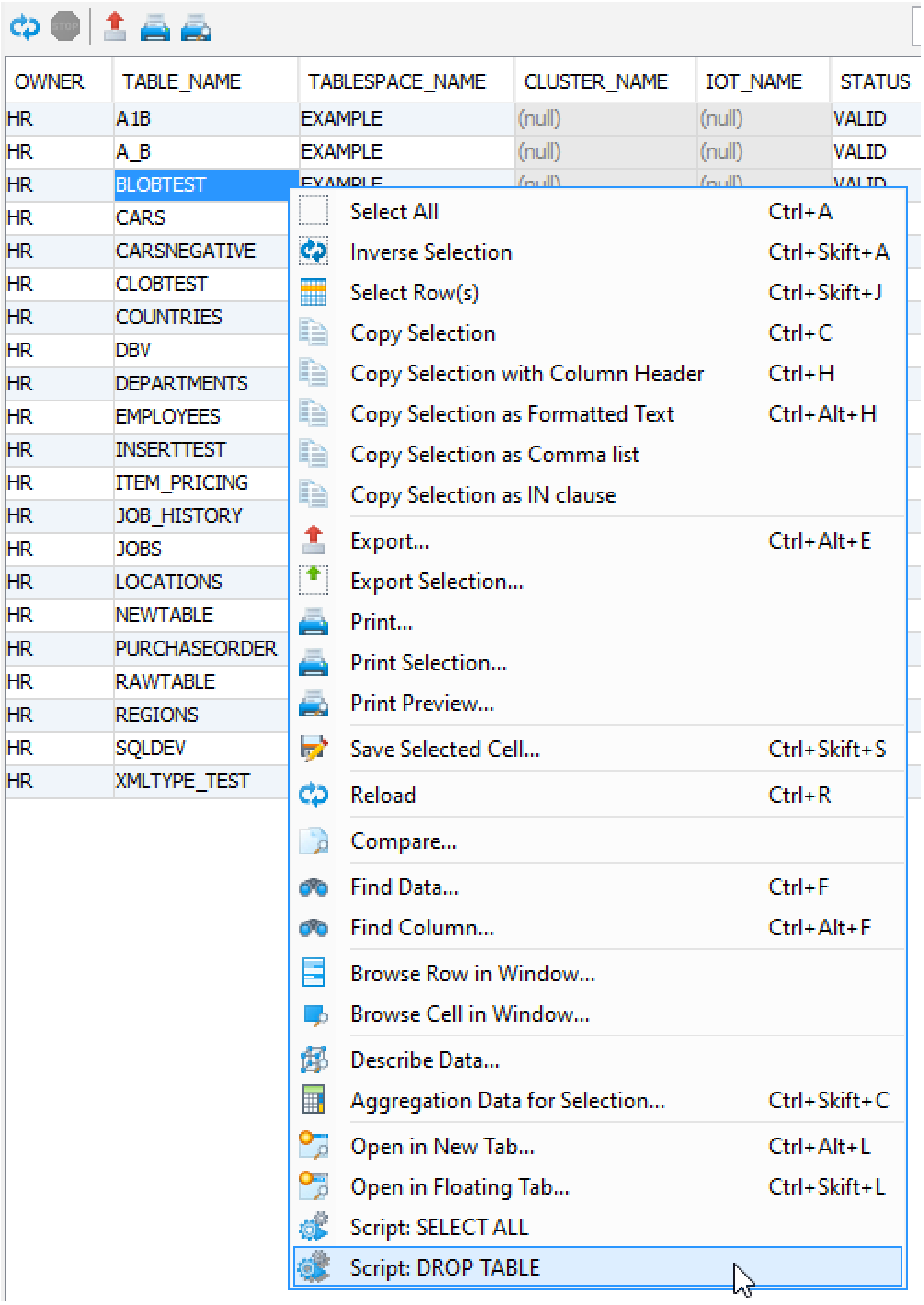
The **name="action"** attribute declare that the value is a pre-defined action. Valid actions are:

- open-object-in-new-tab-command
- open-object-in-floating-tab-command

Any variables in the SQL statement should identify column names in the result set. The user may select any cells in the 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 menuItem declaration.

The variables specified in these examples starts with **${schema=...}** and **${object=...}**. These define 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.

Here is a sample:



OWNER	TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME	IOT_NAME	STATUS
HR	A1B	EXAMPLE	(null)	(null)	VALID
HR	A_B	EXAMPLE	(null)	(null)	VALID
HR	BLOBTEST	EXAMPLE	(null)	(null)	VALID
HR	CARS				
HR	CARSNEGATIVE				
HR	CLOBTEST				
HR	COUNTRIES				
HR	DBV				
HR	DEPARTMENTS				
HR	EMPLOYEES				
HR	INSERTTEST				
HR	ITEM_PRICING				
HR	JOB_HISTORY				
HR	JOBS				
HR	LOCATIONS				
HR	NEWTABLE				
HR	PURCHASEORDER				
HR	RAWTABLE				
HR	REGIONS				
HR	SQLDEV				
HR	XMLTYPE_TEST				

Select All Ctrl+A
Inverse Selection Ctrl+Skift+A
Select Row(s) Ctrl+Skift+J
Copy Selection Ctrl+C
Copy Selection with Column Header Ctrl+H
Copy Selection as Formatted Text Ctrl+Alt+H
Copy Selection as Comma list
Copy Selection as IN clause
Export... Ctrl+Alt+E
Export Selection...
Print...
Print Selection...
Print Preview...
Save Selected Cell... Ctrl+Skift+S
Reload Ctrl+R
Compare...
Find Data... Ctrl+F
Find Column... Ctrl+Alt+F
Browse Row in Window...
Browse Cell in Window...
Describe Data...
Aggregation Data for Selection... Ctrl+Skift+C
Open in New Tab... Ctrl+Alt+L
Open in Floating Tab... Ctrl+Skift+L
Script: SELECT ALL
Script: DROP TABLE

Setting initial max column width

Some result sets may contain wide columns. The following parameter sets an initial maximum width for all columns in the grid.

```
<Input name="columnWidth" value=""/>
```

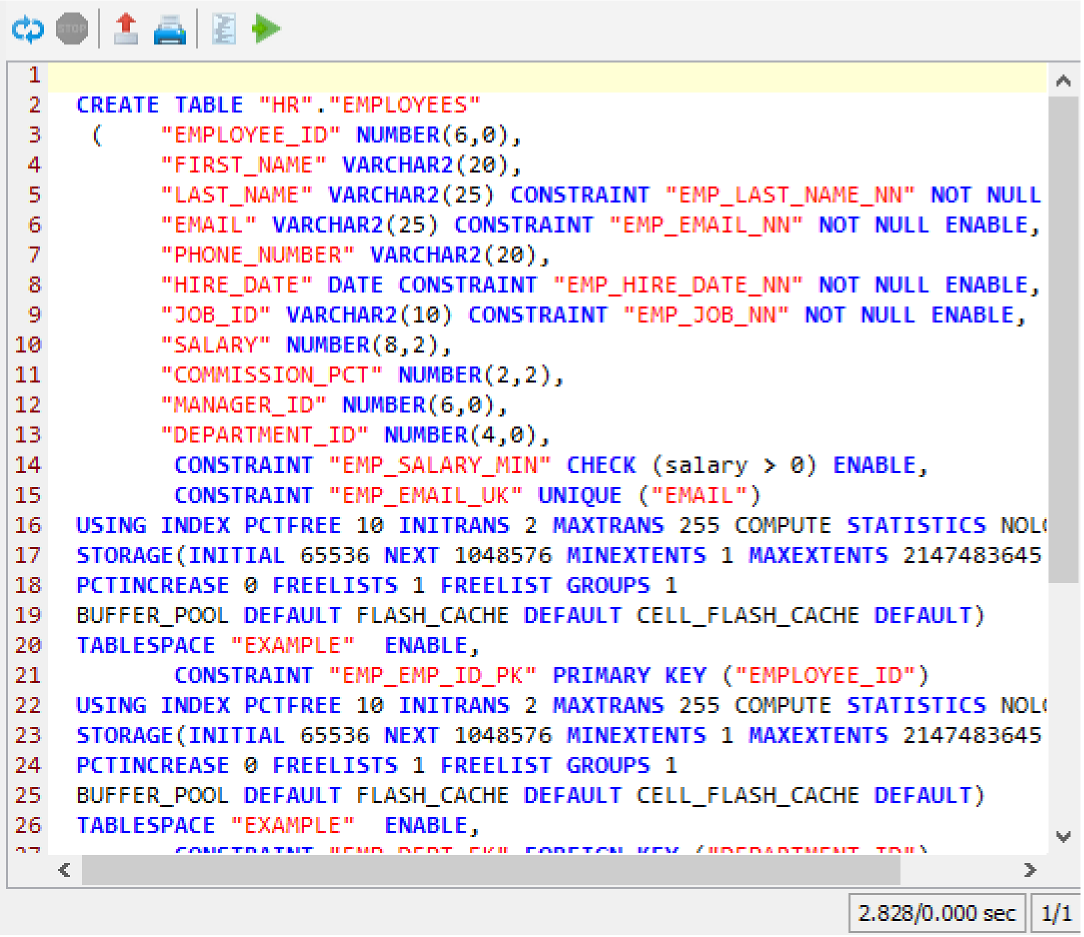
Viewer - text

The **text** viewer presents data from **one column** in a result set in a text browser (read only). This viewer is typically used to present large chunks of data, such as source code, SQL statements, etc. If the result set contain 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 id="oracle-table-triggers-source" icon="Source" label="Source" viewer="text">
   <Input name="dataColumn" value="text"/>
   <Input name="formatSQL" value="true"/>
   <Input name="newline" value""/>
   <Command idref="oracle.getTriggerSource">
      <Input name="owner" value="${OWNER}"/>
      <Input name="name" value="${TRIGGER_NAME}"/>
   </Command>
</DataView>
```

And here is a screenshot of the Source tab based on the previous definition.



```

CREATE TABLE "HR"."EMPLOYEES"
 (     "EMPLOYEE_ID" NUMBER(6,0),
       "FIRST_NAME" VARCHAR2(20),
       "LAST_NAME" VARCHAR2(25) CONSTRAINT "EMP_LAST_NAME_NN" NOT NULL
       "EMAIL" VARCHAR2(25) CONSTRAINT "EMP_EMAIL_NN" NOT NULL ENABLE,
       "PHONE_NUMBER" VARCHAR2(20),
       "HIRE_DATE" DATE CONSTRAINT "EMP_HIRE_DATE_NN" NOT NULL ENABLE,
       "JOB_ID" VARCHAR2(10) CONSTRAINT "EMP_JOB_NN" NOT NULL ENABLE,
       "SALARY" NUMBER(8,2),
       "COMMISSION_PCT" NUMBER(2,2),
       "MANAGER_ID" NUMBER(6,0),
       "DEPARTMENT_ID" NUMBER(4,0),
        CONSTRAINT "EMP_SALARY_MIN" CHECK (salary > 0) ENABLE,
        CONSTRAINT "EMP_EMAIL_UK" UNIQUE ("EMAIL")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS NOL
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "EXAMPLE"  ENABLE,
        CONSTRAINT "EMP_EMP_ID_PK" PRIMARY KEY ("EMPLOYEE_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS NOL
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "EXAMPLE"  ENABLE,
        CONSTRAINT "EMP_DEPT_FK" FOREIGN KEY ("DEPARTMENT_ID")
```

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 have the **SQL Formatting** function, which when invoked formats the SQL buffer in the viewer. The **formatSQL** input parameter is used to control whether formatting should be made automatically when the data first displayed. If formatSQL is not specified, no initial formatting is made.

```
<Input name="formatSQL" value=""/>
```

Adding newline to each row
For a result set containing multiple rows and all rows should be displayed in a text viewer, the **newline** parameter define the character(s) that should separate the rows in the viewer. A **\n** somewhere in the value will be converted to a platform dependent newline sequence in the viewer. By default there is no newline sequence between multiple rows.

```
<Input name="newline" value="\n"/>
<Input name="newline" value="as-sql"/>
```

The **as-sql** value will append an SQL statement delimiter and a platform specific newline sequence at the end of every row.

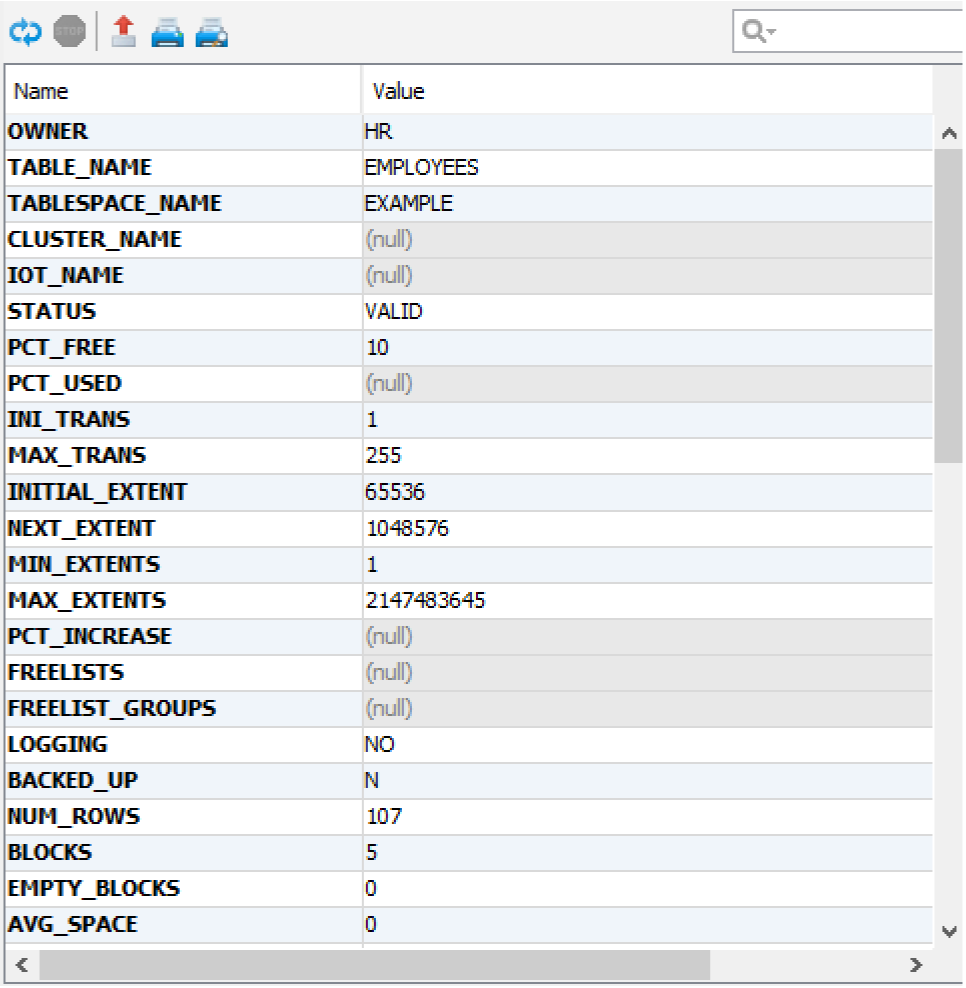
Viewer - form

The **form** viewer displays 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 id="oracle-table-info" icon="Info" label="Info" viewer="form" order-before="0">
   <Command idref="oracle.getTable">
      <Input name="owner" value="${schema}"/>
      <Input name="table" value="${objectname}"/>
   </Command>
</DataView>
```

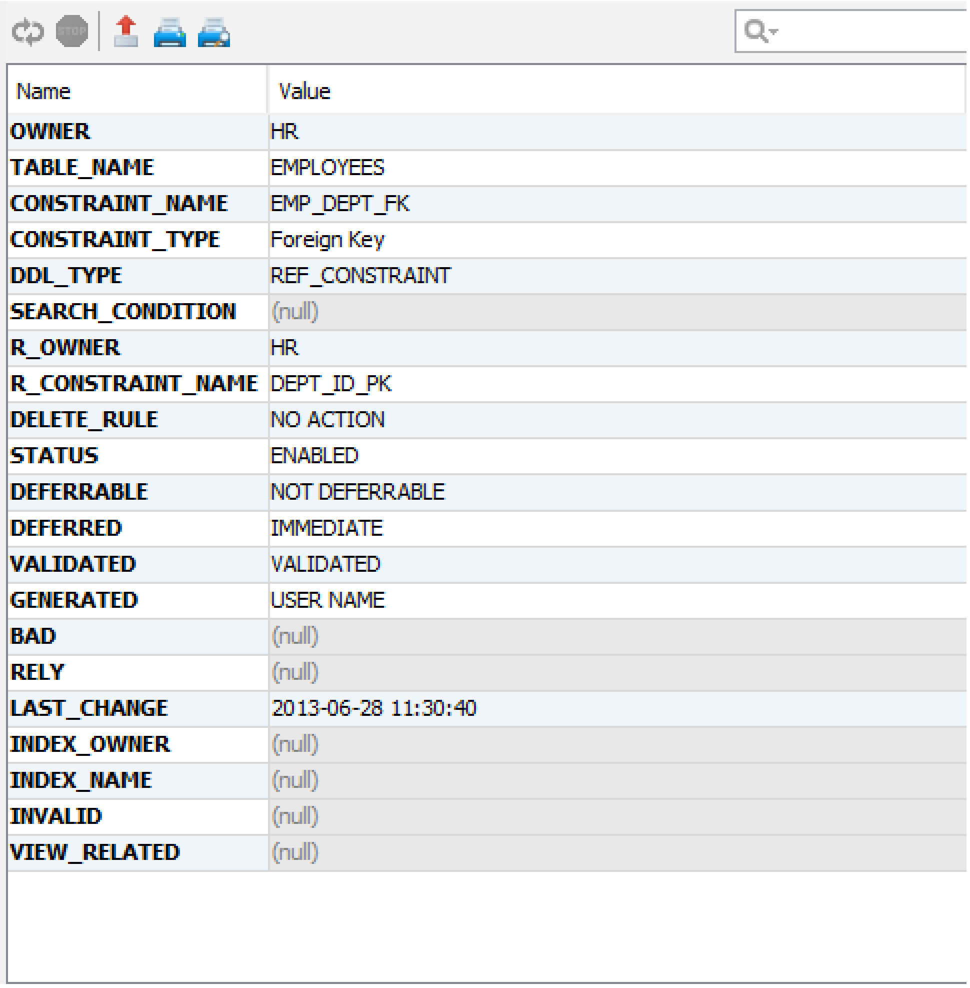
And here is a screenshot of the Info tab based on the previous definition.



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	1048576
MIN_EXTENTS	1
MAX_EXTENTS	2147483645
PCT_INCREASE	(null)
FREELISTS	(null)
FREELIST_GROUPS	(null)
LOGGING	NO
BACKED_UP	N
NUM_ROWS	107
BLOCKS	5
EMPTY_BLOCKS	0
AVG_SPACE	0

Viewer - node-form

The **node-form** viewer presents all data associated with the selected DataNode (variables). Here is a sample of the XML for the node-form viewer:



Name	Value
OWNER	HR
TABLE_NAME	EMPLOYEES
CONSTRAINT_NAME	EMP_DEPT_FK
CONSTRAINT_TYPE	Foreign Key
DDL_TYPE	REF_CONSTRAINT
SEARCH_CONDITION	(null)
R_OWNER	HR
R_CONSTRAINT_NAME	DEPT_ID_PK
DELETE_RULE	NO ACTION
STATUS	ENABLED
DEFERRABLE	NOT DEFERRABLE
DEFERRED	IMMEDIATE
VALIDATED	VALIDATED
GENERATED	USER NAME
BAD	(null)
RELY	(null)
LAST_CHANGE	2013-06-28 11:30:40
INDEX_OWNER	(null)
INDEX_NAME	(null)
INVALID	(null)
VIEW_RELATED	(null)

Hiding columns

There may be data associated with the object that you don't want to present in the node form. The **hidecolumn** input parameter control what data for the object that should be invisible and you may repeat this option as many times you like to handle multiple variables that shouldn't be displayed.

```
<Input name="hidecolumn" value="oracle.getKeys.TABLE_OWNER"/>
```

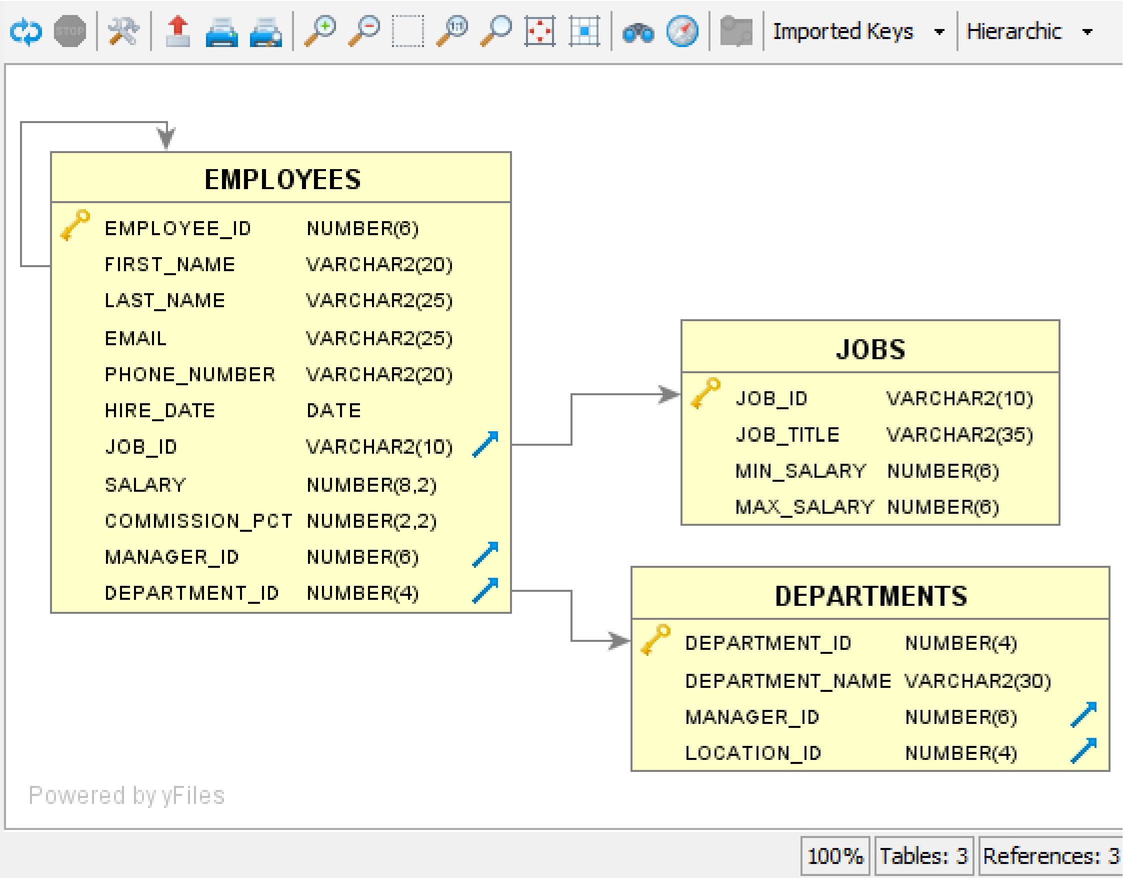
Viewer - table-refs

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 id="generic-table-references" icon="References" label="References" viewer="table-refs"/>
```

And here is a screenshot of the References tab based on the previous definition.



EMPLOYEES

EMPLOYEE_ID	NUMBER(6)
FIRST_NAME	VARCHAR2(20)
LAST_NAME	VARCHAR2(25)
EMAIL	VARCHAR2(25)
PHONE_NUMBER	VARCHAR2(20)
HIRE_DATE	DATE
JOB_ID	VARCHAR2(10)
SALARY	NUMBER(8,2)
COMMISSION_PCT	NUMBER(2,2)
MANAGER_ID	NUMBER(6)
DEPARTMENT_ID	NUMBER(4)

JOBS

JOB_ID	VARCHAR2(10)
JOB_TITLE	VARCHAR2(35)
MIN_SALARY	NUMBER(6)
MAX_SALARY	NUMBER(6)

DEPARTMENTS

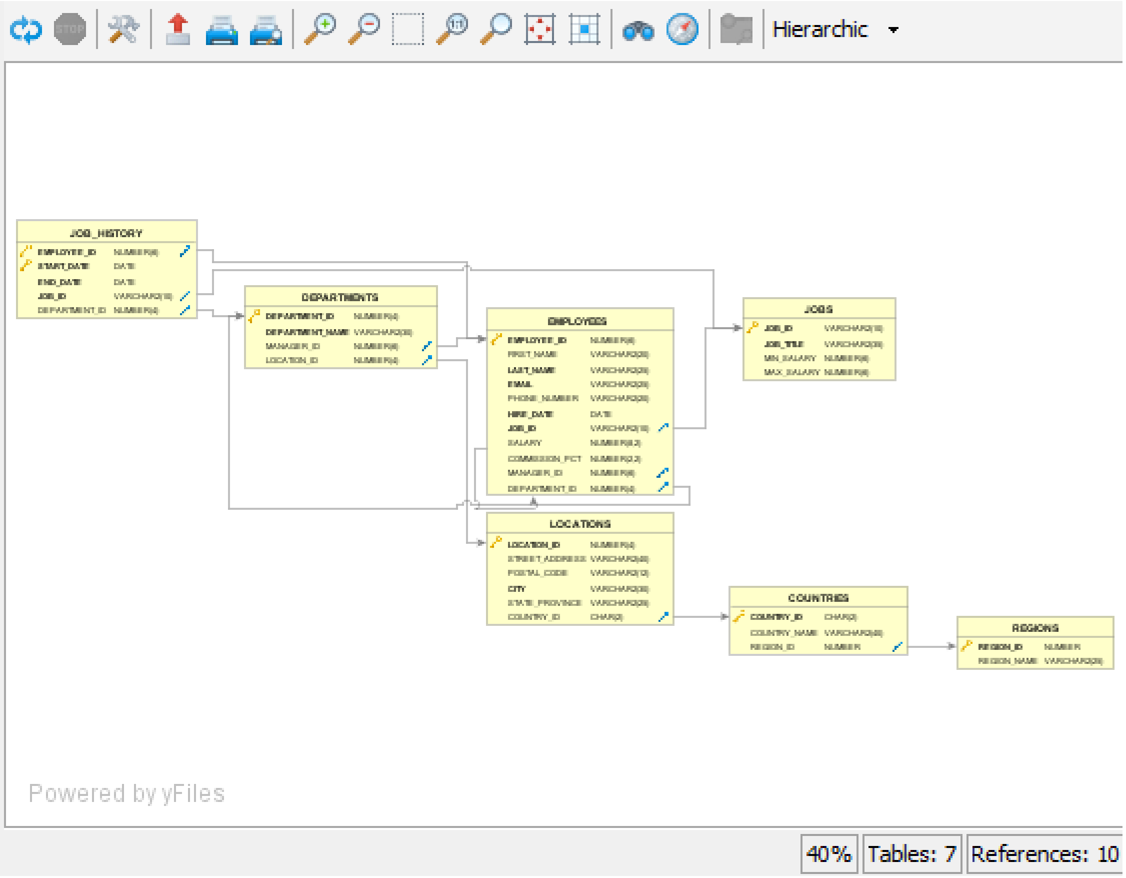
DEPARTMENT_ID	NUMBER(4)
DEPARTMENT_NAME	VARCHAR2(30)
MANAGER_ID	NUMBER(6)
LOCATION_ID	NUMBER(4)

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 id="oracle-tables-references" icon="References" label="References" viewer="tables-refs">
   <Command idref="oracle.getTables">
      <Input name="owner" value="${schema}"/>
      <ProcessDataSet action="renamecolumn" index="OWNER" name="TABLE_SCHEM"/>
      <ProcessDataSet action="renamecolumn" index="TABLE_NAME" name="TABLE_NAME"/>
   </Command>
</DataView>
```

And here is a screenshot of the References tab based on the previous definition.



Viewer - table-data

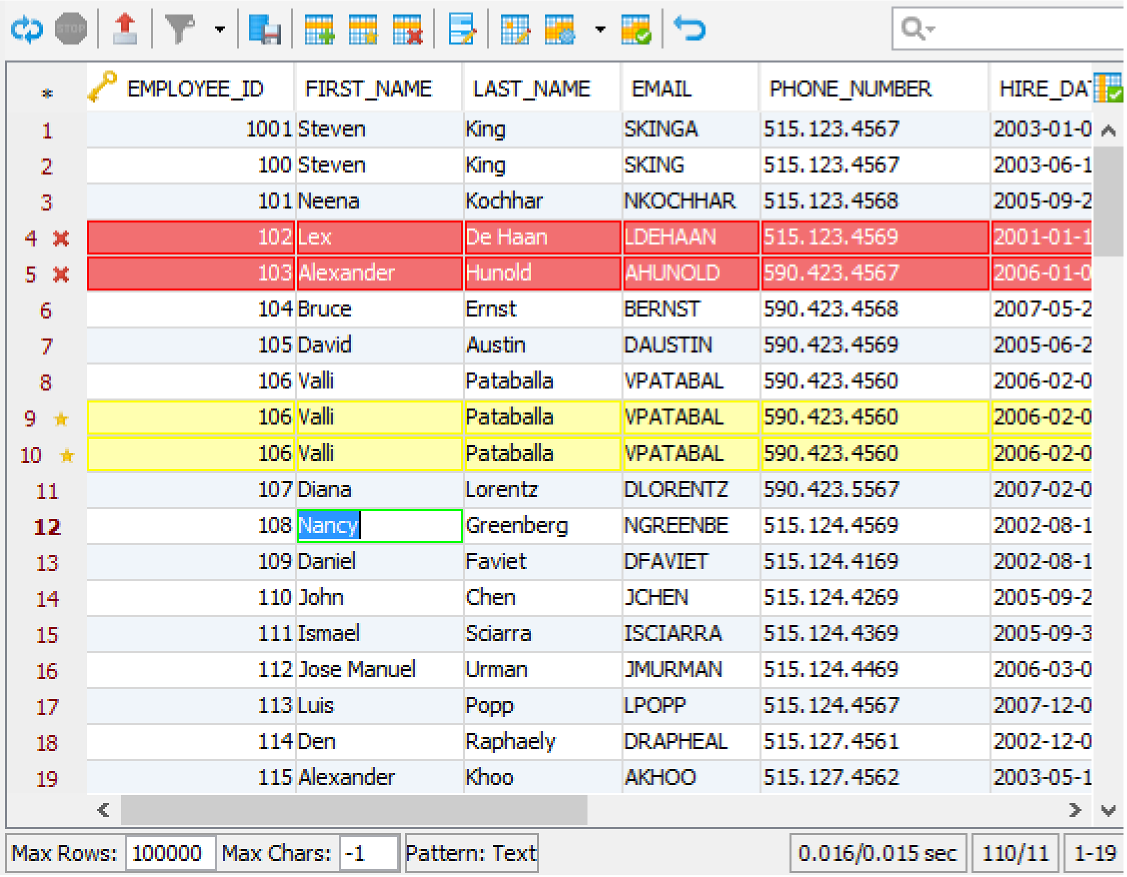
The **table-data** viewer shows the data for a table in a grid with various features such as filtering and editing (if licensed) functionality.

> Information presented in the grid is obtained automatically by the viewer via a standard **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.
>
> It is important that the **Database Type** in the connection setup is properly set to match the database being accessed. The reason is that the identifiers (schema, database, table) are delimited automatically. Delimiters are database specific and if having the wrong database type set it may result in an error getting the result.

Here is a sample of the XML for the table-data viewer:

```
<DataView id="oracle-view-data" icon="Data" label="Data" viewer="table-data"/>
   <Input name="disableEdit" value="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_DA
1		1001	Steven	King	SKINGA	515.123.4567	2003-01-0
2		100	Steven	King	SKING	515.123.4567	2003-06-1
3		101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-2
4	✖	102	Lex	De Haan	LDEHAAN	515.123.4569	2001-01-1
5	✖	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-0
6		104	Bruce	Ernst	BERNST	590.423.4568	2007-05-2
7		105	David	Austin	DAUSTIN	590.423.4569	2005-06-2
8		106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-0
9	★	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-0
10	★	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-0
11		107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-0
12		108	Nancy	Greenberg	NGREENBE	515.124.4569	2002-08-1
13		109	Daniel	Faviet	DFAVIET	515.124.4169	2002-08-1
14		110	John	Chen	JCHEN	515.124.4269	2005-09-2
15		111	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-3
16		112	Jose Manuel	Urman	JMURMAN	515.124.4469	2006-03-0
17		113	Luis	Popp	LPOPP	515.124.4567	2007-12-0
18		114	Den	Raphaely	DRAPHEAL	515.127.4561	2002-12-0
19		115	Alexander	Khoo	AKHOO	515.127.4562	2003-05-1

Max Rows: 100000 Max Chars: -1 Pattern: Text 0.016/0.015 sec 110/11 1-19

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="true"/>
```

Viewer - 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.

It is that the **Database Type** in the connection setup is properly set to match the database being accessed. The reason is that the identifiers (schema, database, table) are delimited automatically. Delimiters are database specific and if having the wrong database type set it may result in an error getting the result.

Here is a sample of the XML for the table-rowcount viewer:

```
<DataView id="generic-table-rowcount" icon="RowCount" label="Row Count" viewer="table-rowcount"/>
```

XML element - Command

Check the commands (see page 349) section for more information.

XML element - Input

The **Input** element is supported for some of the data viewers. Check the viewer sections for more information.

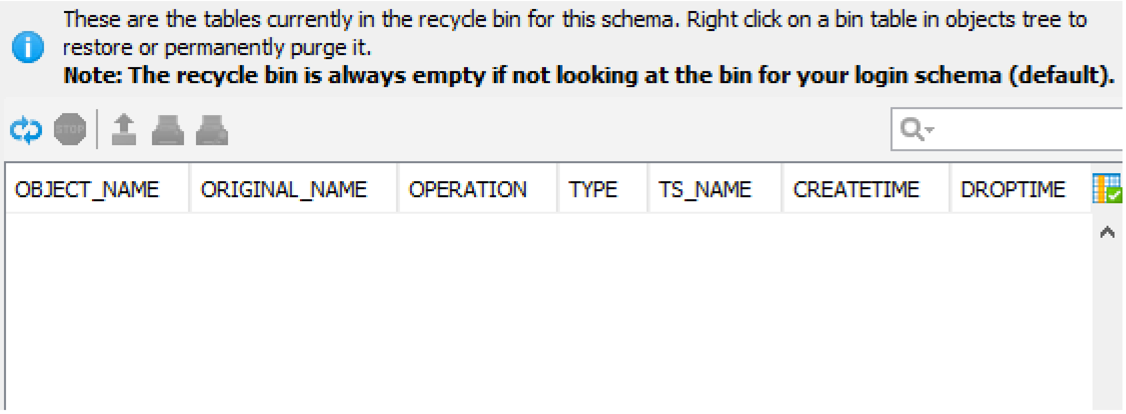
XML element - Message

The **Message** element is very simple as it just define a message that should appear at the top of the viewer. The text in the message may contain HTML tags such as <b> (bold), <i> (italic), <br> (line break), etc.

Here is a sample of the XML for using the message element in a grid viewer:

```
<ObjectView type="RecycleBin">
   <DataView id="oracle-recyclebin-recyclebin" icon="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	CREATETIME	DROPTIME

22.4.7 XML element - ObjectsActionDef

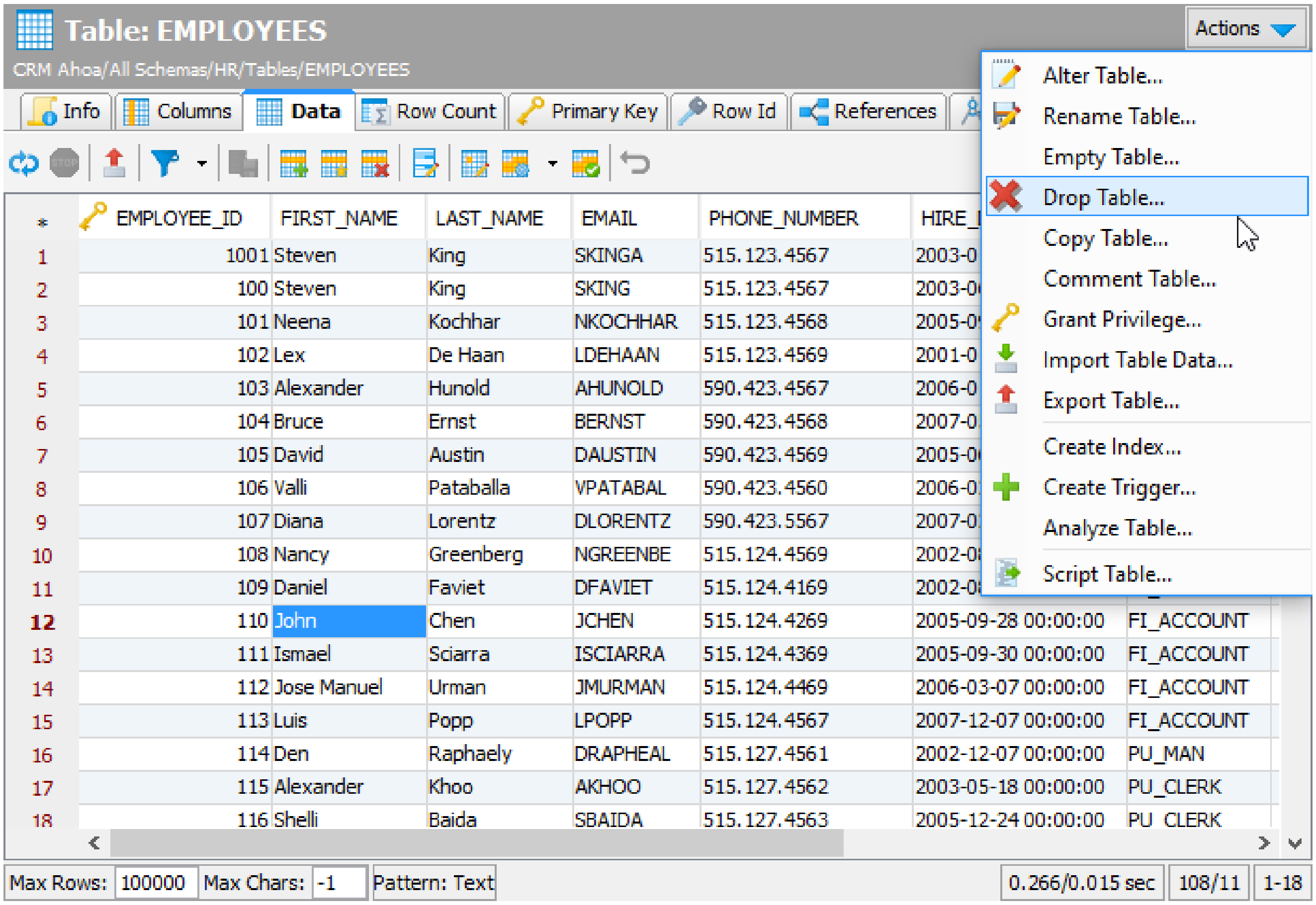
Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

Introduction

Objects actions (ObjectsActionDef) define what operations are available for the object types defined in the **ObjectsTreeDef**. Object actions are powerful, as they offer an extensive number of features to define actions for almost any type of object operation.

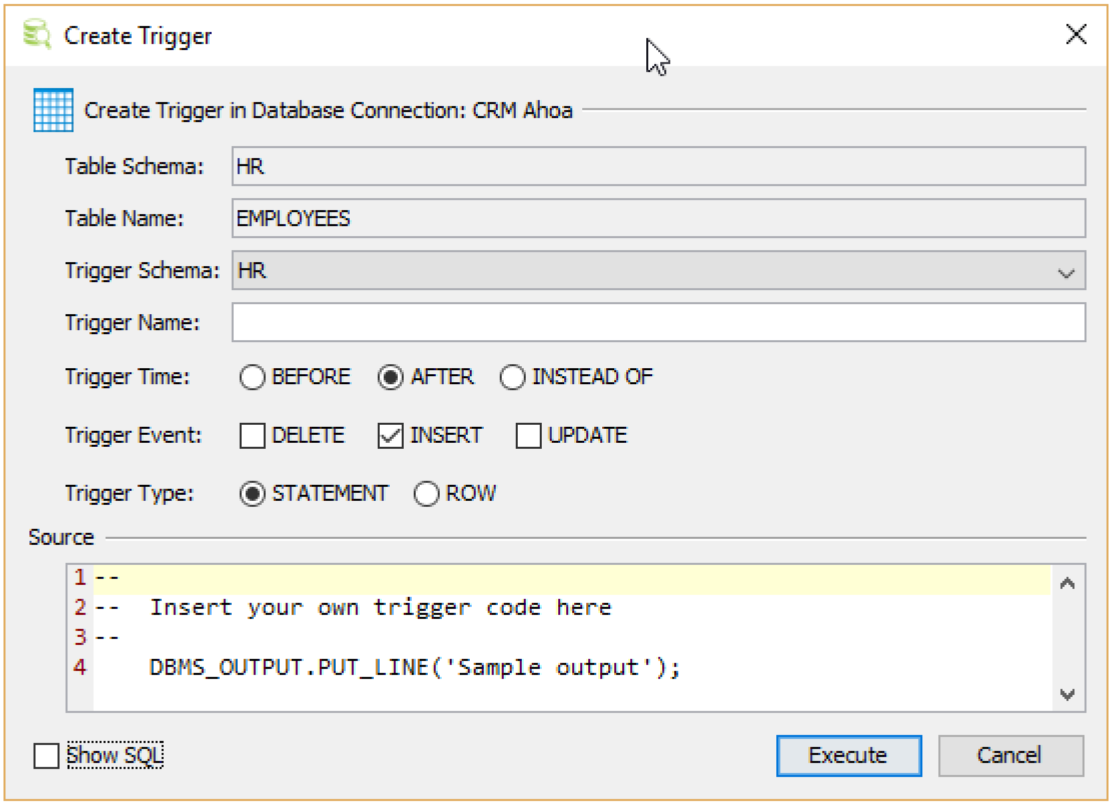
In DbVisualizer, the object actions menu is accessed via the right-click menu in the objects tree or via the **Actions** button in the object view:



*	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_	
1	1001	Steven	King	SKINGA	515.123.4567	2003-0	
2	100	Steven	King	SKING	515.123.4567	2003-0	
3	101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-0	
4	102	Lex	De Haan	LDEHAAN	515.123.4569	2001-0	
5	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-0	
6	104	Bruce	Ernst	BERNST	590.423.4568	2007-0	
7	105	David	Austin	DAUSTIN	590.423.4569	2005-0	
8	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-0	
9	107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-0	
10	108	Nancy	Greenberg	NGREENBE	515.124.4569	2002-0	
11	109	Daniel	Faviet	DFAVIET	515.124.4169	2002-0	
12	110	John	Chen	JCHEN	515.124.4269	2005-09-28 00:00:00	FI_ACCOUNT
13	111	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-30 00:00:00	FI_ACCOUNT
14	112	Jose Manuel	Urman	JMURMAN	515.124.4469	2006-03-07 00:00:00	FI_ACCOUNT
15	113	Luis	Popp	LPOPP	515.124.4567	2007-12-07 00:00:00	FI_ACCOUNT
16	114	Den	Raphaely	DRAPHEAL	515.127.4561	2002-12-07 00:00:00	PU_MAN
17	115	Alexander	Khoo	AKHOO	515.127.4562	2003-05-18 00:00:00	PU_CLERK
18	116	Shelli	Baida	SBAIDA	515.127.4563	2005-12-24 00:00:00	PU_CLERK

Max Rows: 100000 Max Chars: -1 Pattern: Text 0.266/0.015 sec 108/11 1-18

All of the operations for the current **Table** object in the figure above are expressed in the ObjectsActionDef section in the database profile. The implementation for these actions are either declared entirely in XML via standard definitions, or via custom definitions. (The Java API for action handlers is not yet documented). The following screenshot shows the dialog appearing when executing an action via a standard XML definition:



Create Trigger in Database Connection: CRM Ahoa

Table Schema: HR

Table Name: EMPLOYEES

Trigger Schema: HR

Trigger Name:

Trigger Time: ○ BEFORE ● AFTER ○ INSTEAD OF

Trigger Event: □ DELETE ☑ INSERT □ UPDATE

Trigger Type: ● STATEMENT ○ ROW

Source

```
--
--   Insert your own trigger code here
--
     DBMS_OUTPUT.PUT_LINE('Sample output');
```

□ Show SQL

The first field in the dialog, **Database Connection**, is always present and shows the alias of the database connection the current object is associated with. At the bottom, there is a **Show SQL** control that, when checked , 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

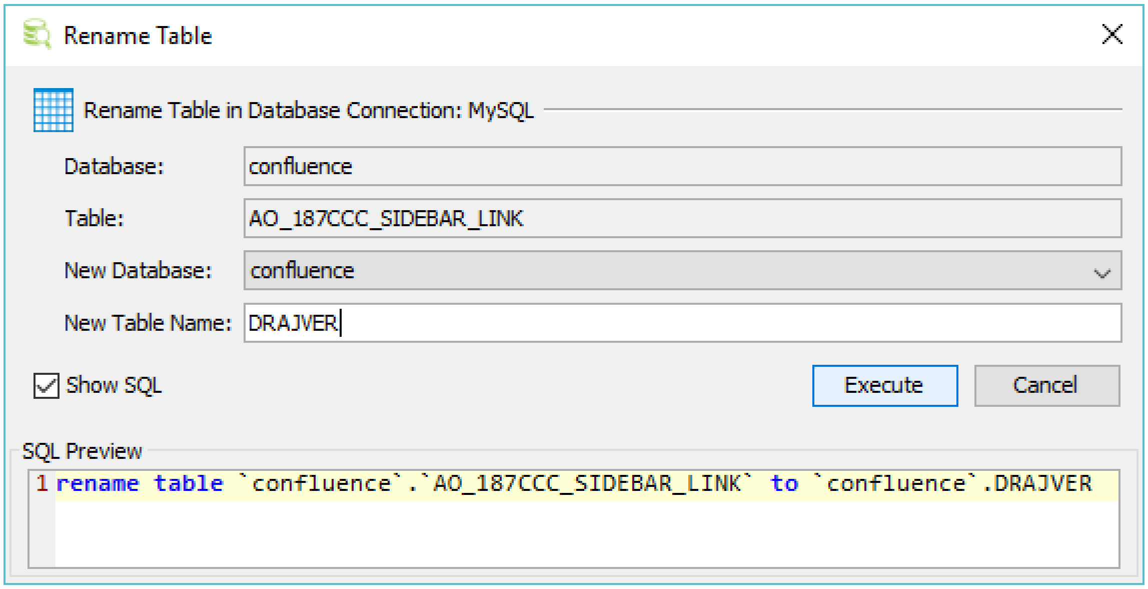
Variables are used to reference data for the object for which the action was launched, and the data for all its parent objects in the objects tree. Variables are also used to reference input data specified by the user in the actions dialog. Variables are typically used in the **Command**, **Confirm**, **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) 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** component 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 checked, you will see any edits in the dialog being reflected immediately in the final **SQL Preview**.



Database: confluence

Table: AO_187CCC_SIDEBAR_LINK

New Database: confluence

New Table Name: DRAJVER

☑ Show SQL

```
rename table `confluence`.`AO_187CCC_SIDEBAR_LINK` to `confluence`.DRAJVER
```

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 current object in the objects tree (**GroupNode** or **DataNode**). 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 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 (see page 359) section).

The **New Database** input field is a list component showing 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 (the new table name) .

Both the **New Database** and **New Table Name** fields are editable and should be specified by the user. This data is then available 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.

XML element - ActionGroup

The **ActionGroup** element is a container and groups a collection of **ActionGroup**, **Action** and **Separator** elements. It is used to define what actions should be present for a particular object type. It also define in what order the actions should appear in the menu and where any separators should be located. ActionGroup elements can be nested and these will be displayed as sub menus in DbVisualizer.

```
<ActionGroup type="Table">
   <Action id="xxx">
      ...
   </Action>
</ActionGroup>
```

The attributes for an ActionGroup are:

Attribute	Value	Description
type		This defines what object type the ActionGroup is mapped to. This attribute is required and valid only for top level ActionGroup elements (not nested ActionGroup elements). An example is the **Table** object type, the corresponding **<ActionGroup type="Table">** will only be displayed when the current object is a Table
label		This attribute is required for nested ActionGroup elements and is the label displayed in the sub menu. (This attribute have no effect on top level ActionGroup elements)
action	drop	**drop** is useful when extending another database profile to remove the ActionGroup and all its child elements
order-before		Specifies the order of this ActionGroup among a collection of ActionGroup elements located at the same level. It can either be an index starting at 0 (first) or a node type. Ex . **order-before="Views"** will order this ActionGroup before ActionGroup elements defined by the **type="Views"** attribute
order-after		Specifies the order of this ActionGroup among a collection of ActionGroup elements located at the same level. It can either be an index starting at 0 (first) or a node type. Ex . **order-after="Views"** will order this ActionGroup after ActionGroup elements defined by the **type="Views"** attribute

XML element - Action

The **Action** element defines the characteristics of the action. The following show the complete definition of the **Drop Table** action in Oracle.

```
<Action id="oracle-table-drop" label="Drop Table" reload="true" icon="remove">
   <Input label="Schema" style="text" editable="false">
      <Default>${schema}</Default>
   </Input>
   <Input label="Table" style="text" editable="false">
      <Default>${objectname}</Default>
   </Input>
   <Input label="Drop Referential Integrity Constraints" name="cascade" style="check"
          tip="Enable this to drop all referential integrity constraints
               that refer to primary and unique keys in the dropped table">
      <Values>cascade constraints</Values>
   </Input>
   <If test="#util.isDatabaseVersionGTE(10)">
      <Input label="Purge Space" name="purge" style="check"
         tip="Enable this if you want to drop the table and
              release the space associated with it in a single step">
         <Values>purge</Values>
      </Input>
   </If>
   <Else>
```

```
      <SetVar name="purge" value="''"/>
   </Else>
   <Command>
      <SQL>
         <![CDATA[drop table "${schema}"."${objectname}" ${cascade} ${purge}]]>
      </SQL>
   </Command>
   <Confirm>
      Really drop table ${schema}.${objectname}?
   </Confirm>
   <Result>
      Table ${schema}.${objectname} has been dropped!
   </Result>
</Action>
```

The available attributes for the **Action** element:

Attribute	Value (bold = default)	Description
id		Every Action element must have a unique id which is not only unique in the current profile but also with all id's in extended profiles. The recommended format is **profileName-actionGroupType-action**. Ex: oracle-table-drop
icon		The name of the icon that should be displayed next to the label in the actions menu
label		The label for the action as it should be displayed in the list of actions and in the actions dialog
reload	true/**false**	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	• script show the action dialog, process user input and send the final SQL	Specifies how the action will be prepared and displayed

Attribute	Value (bold = default)	Description
	to the SQL Commander without executing the command - script-immediate will not show the action dialog but instead pass the final SQL directly to the SQL Commander without executing the command	
resultaction	- mergeasscript - mergeastext	- **mergeastext** will merge multiple result sets to to a single result tab - **mergeasscript** will merge multiple result sets to a single result tab, each row will be terminated with a semi-colon (";") Default is that result sets are displayed in individual tabs
resulttarget	- editor	Only applicable if the **resultaction** attribute is specified. With the value **editor** the merged results will be opened in a new SQL Commander tab
hideif		There may be situations when an action should be dropped due to a condition. The hideif attribute is used to express a condition which is evaluated when the list of actions is created. Example: **hideif="#dataMap.get(' actionlevel') neq 'toplevel'"**
resetcatalogs	true/**false**	Setting this attribute to true will reset any cached databases for the actual database connection. Useful when for example the action create, rename or delete a database
resetschemas	true/**false**	Setting this attribute to true will reset any cached schemas for the actual database connection. Useful when for example the action create, rename or delete a schema

Attribute	Value (bold = default)	Description
supportsmultipleobjects	**true**/false	An action support processing multiple objects if the style attribute for all input elements is one of: • check • check-list • list • radio • separator • node • label • read-only text The **supportsmultipleobjects="true"** attribute is used to disable multi object processing even if the previous criteria is satisfied
class		Used to specify a custom Java class used as the action
classargs		Used to pass arguments to a custom action
doclink		Relative HTML link to the related chapter in the users guide
action	drop	**drop** is useful when extending another database profile to remove the Action
order-before		Specifies the order of this Action among a collection of Action elements located at the same level. It can either be an index starting at 0 (first) or a node type. Ex. **order-before="View"** will order this Action before Action elements defined by the **type="View"** attribute
order-after		Specifies the order of this Action among a collection of Action elements located at the same level. It can either be an index starting at 0 (first) or a node type. Ex. **order-after="View"** will order this Action after Action elements defined by the **type="View"** attribute

XML element - Input

An Input element specifies the characteristics of a field component in 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 name of the variable in which the user input is stored.

Attribute	Value (bold = default)	Description
label		The label for the input component
name		For editable input this should be the name of the variable holding the value specified by the user
tip		Message displayed when hovering over the component
editable	**true**/false	Enables or disables editing of the component
linebreak	true/**false**	If set to true, no line break will be made after the input component. This is useful when for example having multiple **<Input style="check">** elements in a single row
style	list, radio, **text**, check, check-list, password, number, text-editor, wrapped-text-editor, grid, separator, label, note	The style of the input element. See following sections for more details
hideif		There may be situations when an **Input** element should be dropped due to a condition. The hideif attribute is used to express a condition which is evaluated when the action is initialized. Example: **hideif="#dataMap.get('actionlevel') neq ' toplevel'"**
runsetdefaultwhenvaluechanged		The **runsetdefaultwhenvaluechanged** attribute defines what other inputs default command should be triggered when the value for the input is changed

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 hover 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" label="Unit" style="list" linebreak="true">
  <Labels>KB|MB</Labels>
  <Values>K|M</Values>
  <Default>M</Default>
</Input>
```

The previous example show the use of the linebreak attribute. The **Size** number field and the **Unit** list will appear in the same row.

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>
```

The **Default** definition above will execute a **SQL** statement, it will automatically pick the value in the first row's first column and present it as the default value for the input component. SQL may be specified in the Default element for all styles while SQL in **Values** and the **Labels** elements are **valid only for list, radio, and check** styles. In some rare situations it may not be possible to express a SQL statement that will return a single column that should 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. Its value must be a **column name** or **column index** in the result set.

column index (**column="2"** attribute):

```
<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** (**column="THE_SIZE"** attribute):

```
<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 one of the previous examples, is to refer to a command (see page 349) 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 using **<Command idref="xxx">** elements instead.

ⓘ Referring commands in actions via the idref attribute is recommended when the same SQL is used in several actions. Use Input elements for the Command to pass parameters to the command.

The following sections presents the **supported styles** that can be used in the Input element.

Style - 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 specify **editable="false"**

Style - 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">
   <Arg name="height" value="30"/>
</Input>
```

The **Arg** element defines the height (in DLU) for the text-editor. The default height is 80 DLU's.

Style - wrapped-text-editor (multi line)

This is exactly the same as text-editor except that the wrapped-text-editor wraps long lines.

Style - 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"/>
```

Style - 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"/>
```

Style - list (large number of exclusive choices)

The **list** style displays a list of choices in a drop-down component. Only one choice can be selected. 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 the following example and 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.

```
<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 show 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"**.

Style - check-list (large number non exclusive choices)

The **check-list** style is much like the **list** style except that each choice have a check box in front allowing multiple values to be selected

```
<Input label="Weekdays" name="weekdays" style="check-list">
   <Arg name="output" value="jschours[${value}] = true"/>
   <Arg name="newline" value=", "/>
   <Labels>Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday</Labels>
   <Values>00|11|22|33|44|55|66</Values>
   <Default>22|44</Default>
</Input>
```

To produce a desirable output from the check-list it is possible to define how each choice should be generated in the final result. The **output** argument and its **value** attribute is used to define how each value should be generated. In the example each choice will be formatted as: **jschours[22] = true**. The **newline** argument defines any delimiter between each choice. If the user selects both Monday and Friday in the action window the following will be generated:

jschours[00] = true, jschours[44] = true

Style - radio (limited number of choices)

The **radio** style display 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 **<Arg name="direction" value="vertical"/>** element can be specified for radio style to present the radio choices vertically (default is horizontally)

See the **list** style for complete capabilities of the radio style.

Style - check (true/false, on/off, selected/unselected)

The **check** style is suitable for **yes/no**, **true/false**, **here/there** types of input. Its checked state indicates that the **Value** for the input will be set in the final variable. If the check box is unchecked, the variable value is blank.

```
<Input label="Cascade Constraints" name="cascade" style="check">
   <Values>compact</Values>
</Input>
```

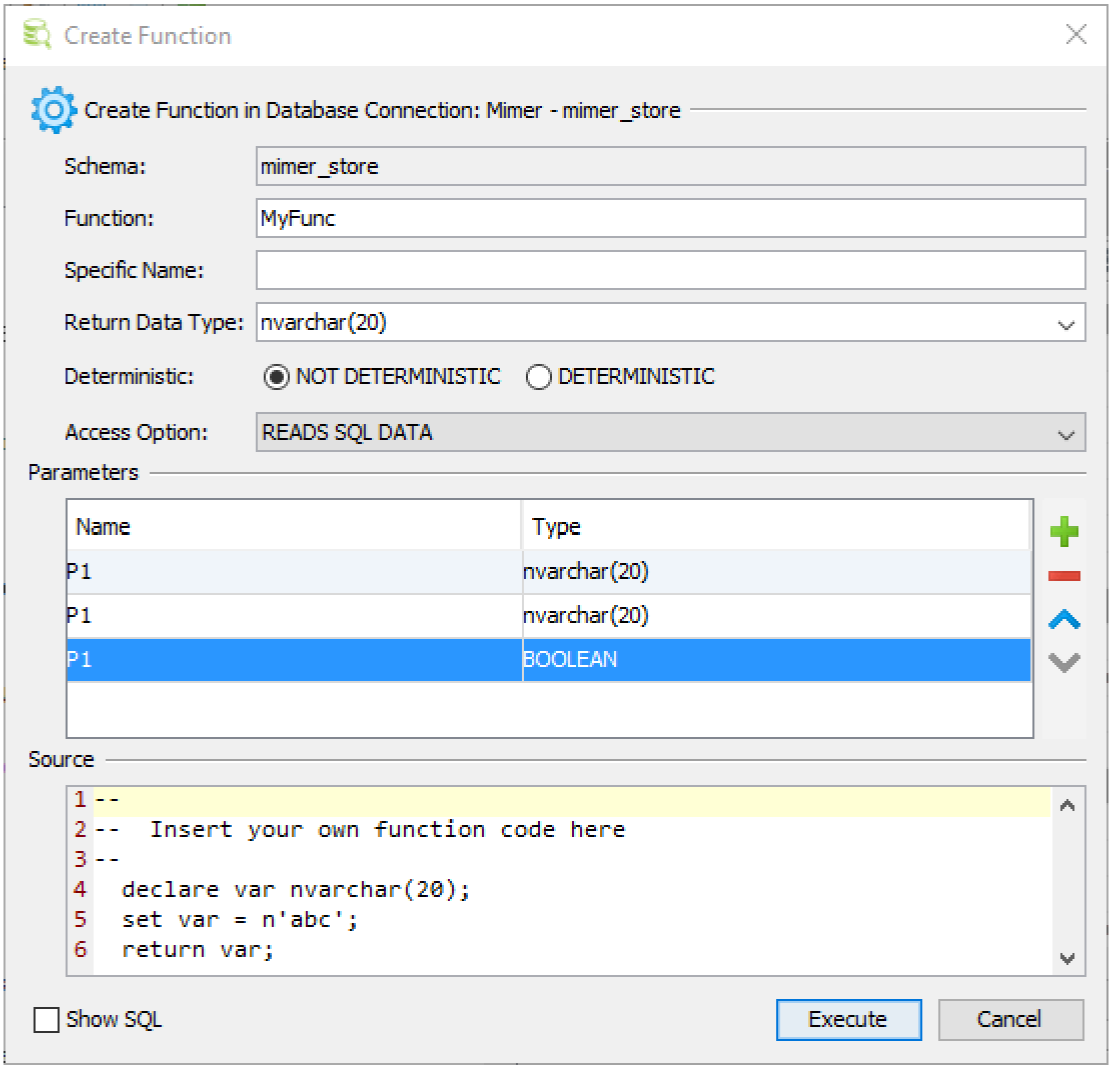
- This will create a check component with the label **Cascade Constraints**
- Checking the check box will set the value of the variable identified by name (cascade) to the value of **Value**, which is **compact**
- If the check box is unchecked, the variable value will be blank

Style - separator (visual divider between input controls)

The **separator** style is not really an input element but is used to visually divide input components in the in the action 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:



Create Function

Create Function in Database Connection: Mimer - mimer_store

Schema: mimer_store

Function: MyFunc

Specific Name:

Return Data Type: nvarchar(20)

Deterministic: ● NOT DETERMINISTIC ○ DETERMINISTIC

Access Option: READS SQL DATA

Parameters

Name	Type
P1	nvarchar(20)
P1	nvarchar(20)
P1	BOOLEAN

Source

```
--
--  Insert your own function code here
--
 declare var nvarchar(20);
 set var = n'abc';
 return var;
```

□ Show SQL

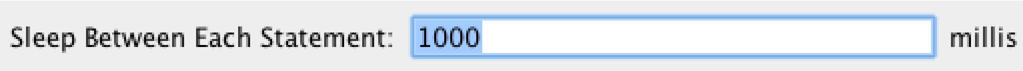
The figure shows the use of separators and two fields that extend to the full width of the action dialog. The separators for **Parameters** and **Source** are here used as alternatives to labels for the fields below them.

Style - note

The **note** style is especially useful when a description or text should be anchored directly after another input.

```
<Input label="Sleep Between Each Statement" name="sleep" style="number" linebreak="false">
   <Default>1000</Default>
</Input>
<Input label="millis" style="note" linebreak="true"/>
```

The previous sample looks like this:


Sleep Between Each Statement: 1000 millis

Style - label

The **label** style is useful when the standard left aligned label should not be displayed. The following example will show seven check boxes each labeled individually and all displayed in the same row.

```
<Input style="label" linebreak="false"/>
<Input label="Mon" name="mon" style="check">
   <Values>mon</Values>
</Input>
<Input label="Tue" name="tue" style="check">
   <Values>tue</Values>
</Input>
<Input label="Wed" name="wed" style="check">
   <Values>wed</Values>
</Input>
<Input label="Thu" name="thu" style="check">
   <Values>thu</Values>
</Input>
<Input label="Fri" name="fri" style="check">
   <Values>fri</Values>
</Input>
<Input label="Sat" name="sat" style="check">
   <Values>sat</Values>
</Input>
<Input label="Sun" name="sun" style="check">
   <Values>sun</Values>
</Input>
```

Style - grid (configurable multi row/columns input)

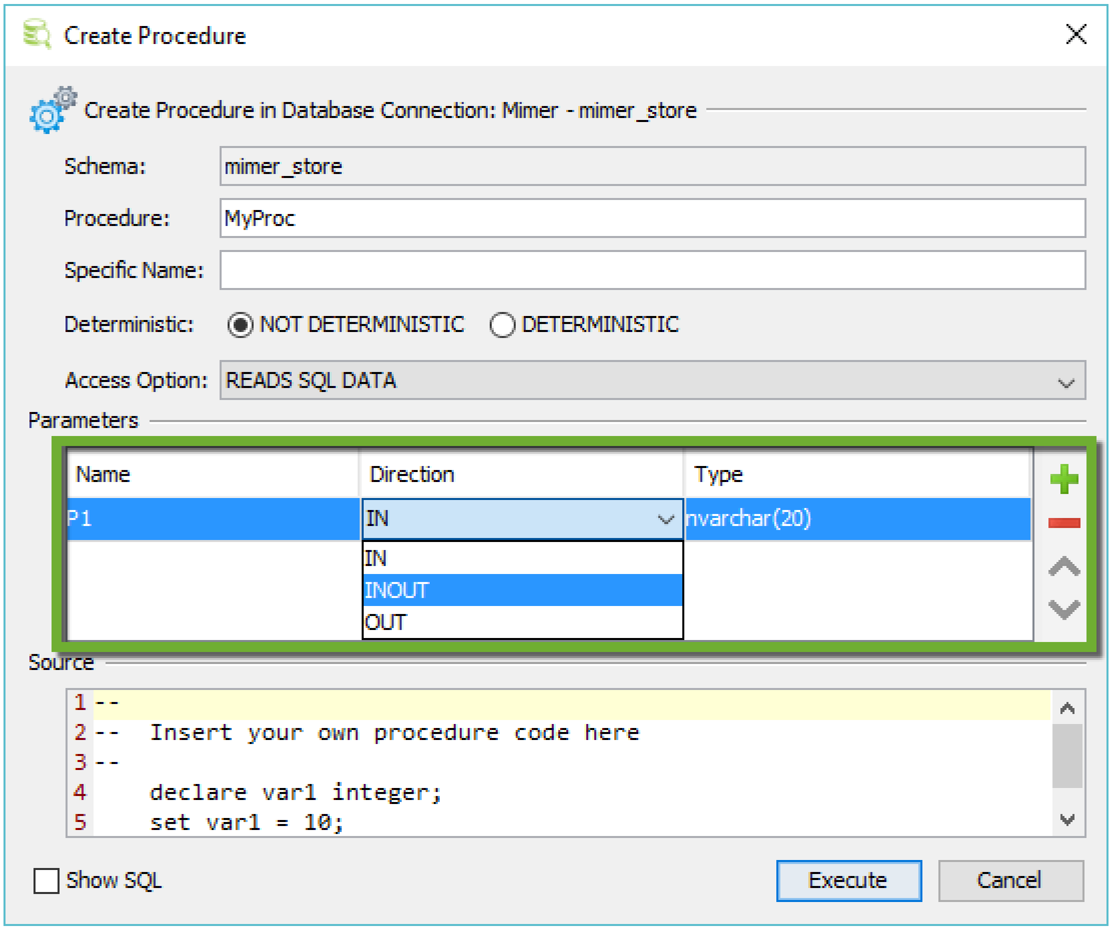
The **grid** input style is presented as a grid with user controls to **add**, **remove** and **move** rows. The columns that should appear in the grid are defined by using any of the primitive styles: **text**, **number**, **password**, **check**, **list** and **radio**. The grid style is useful for 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 show 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 the result:



Create Procedure

Create Procedure in Database Connection: Mimer - mimer_store

Schema: mimer_store

Procedure: MyProc

Specific Name:

Deterministic: ● NOT DETERMINISTIC ○ DETERMINISTIC

Access Option: READS SQL DATA

Parameters

Name	Direction	Type
P1	IN	nvarchar(20)

Source

```
--
--   Insert your own procedure code here
--
     declare var1 integer;
     set var1 = 10;
```

The sub elements for the grid style is different from the other input styles as it accepts nested **Input** elements. These input styles define 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 newline sequence 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.

XML element - SetVar

The **SetVar** element is used to do conditional processing and create new variables based on the content of other variables or static text.

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 three attributes:

Attribute	Description
name	The name of the new variable
value	Should contain the expression that will be evaluated. The expression is based on the OGNL (http://commons.apache.org/proper/commons-ognl/) toolkit. This is an expression library that mimics most of what is being supported by Java. Variables are referenced as **#variableName**
action	This attribute is optional and can have any of the following values: • **show** - default and indicates that the variable (and its value will appear in node-form viewers) • **hide** - the variable will not display in node-form viewers • **drop** - the SetVar element is not evaluated when actions is being processed • **runwheninit** - runs the SetVar only during initialization of the action (before window is displayed)

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.

XML element - 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. Otherwise the message is displayed in a confirmation dialog.

```
<Confirm>Really drop table ${table}?</Confirm>
```

Note that the message text can be composed of HTML tags such as <b>, <i>, <br>, etc.

XML element - Result

The **Result** element is optional and if specified, it is shown in a dialog after successful execution.

> ℹ 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.

XML element - 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>
```

For more information about the Command element check the XML element - Commands (see page 349) section
.

XML element - Message

The **Message** element can be used to specify a highlight message that will appear at the top of the action window.

```
<Message>
    <![CDATA[<html>
This action will be <b>deprecated</b> in a future version as it use database calls that has been
declared by the database vendor as <b>extremely bad performing</b>.</html>
    ]]>
</Message>
```

You may use simple HTML tags in the message content.

Action showing just a message

There may be situations when an action should show a message in a simple dialog with just an OK button. One scenario when this is useful is when a condition is evaluated for an action requiring certain DB privileges to run it . If proper authorization is missing a message should be displayed.

This is accomplished by having a single **Confirm** element for the **Action** element. The following illustrates an example:

```
<If test="#SUPERUSER">
   <Action id="vertica-table-analyze-workload" label="Analyze Workload For Table" resultaction="
show">
      <Input label="Schema" style="text" editable="false">
         <Default>${schema}</Default>
      </Input>
      <Input label="Table" style="text" editable="false">
         <Default>${objectname}</Default>
      </Input>
      <Command>
         <SQL>
            <![CDATA[SELECT analyze_workload('${schema}.${objectname}')]]></SQL>
      </Command>
      <Confirm>
         Really Analyze Workload on ${schema}.${objectname}?
      </Confirm>
   </Action>
</If>
<Else>
   <Action id="vertica-table-analyze-workload-INFO" label="Analyze Workload For Table">
      <Confirm>
         <![CDATA[
            This feature requires the <b>super user</b> authorization.
         ]]>
      </Confirm>
   </Action>
</Else>
```

22.5 Icons

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

22.5.1 Introduction

Icons related to functionality defined in a database profile are displayed in the database objects tree, actions and object viewers. Icons are declared by mapping a logical name with the file name for the icon. For database profiles provided with DbVisualizer, the **type="xxx"** attribute for **GroupNode** and **DataNode** elements map the **xxx** with a matching icon file name. Icons are normally of minor interest until you decide to build your own database profile or extend an existing one.

The following show the use of the **icon** attribute for a **DataNode** element. A condition control what icon to use based on the value of **getCatalogs.TABLE_CAT**.

```
<DataNode type="Catalog" label="${getCatalogs.TABLE_CAT}"
          icon="#dataMap.get('getCatalogs.TABLE_CAT').equals('sales') ? 'salesIcon' :
               (#dataMap.get('getCatalogs.TABLE_CAT').equals('support') ? 'supportIcon' : null)">
</DataNode>
```

The following sections explain how icons are handled and what choices you have to add your own icons.

22.5.2 icons.prefs file

Icons are defined in a simple text file with each row in the format: **name=iconFileName**. In DbVisualizer there is a *icons.prefs* file provided with the installation and it maps all icons used not only in database profiles but for all features. Here is a sample of the icons.prefs file.

```
PhysicalStandby=                server_certificate
Plan=                           FIX_execute_explain
PrimaryKey=                     key
Privileges=                     key
Procedure=                      gears
Procedures=                     gears
Process=                        cpu
Processes=                      cpu
Program=                        hat_green
Programs=                       hat_green
Properties=                     control_panel
PublicDatabaseLinks=            FIX_public_link
PublicDatabaseLink=             FIX_public_link
```

The first name is the logical name used in the database profile. For all object types such as **Table**, **Column**, **View**, **Source** and so on these names are defined in the *icons.prefs* file. For non object types icons are named with the name the icon represents such as **cut**, **copy**, **paste**, **open** and so on. The value for each logical name is the file name without the extension .png.

The object type may refer grouping objects (**GroupNode**) such as **Tables**, **Views**, **Procedures** and specific a objects (**DataNode**) such as **Table**, **View**, **Procedure**. The general recommendation is to name the object type for a GroupNode in a plural form and in singular form for DataNode objects. The icon representing for example **Tables** and **Table** is in most cases the same, still there must be two definitions in the *icons.prefs* file.

22.5.3 Icons Search Path

The database profile search path defined in **Tool Properties / General / Database Connection / Database Profile** not only define what directories are searched for profiles but also icons. These are the default folders searched:

```
${dbvis.prefsdir}/ext/profiles
${dbvis.home}/resources/profiles
```

`${dbvis.prefsdir}` is replaced with the setting directory for DbVisualizer on the platform being used. On Windows this is **C:\Users\<user>\.dbvis** while `${dbvis.home}` is the installation directory for DbVisualizer. If there is an icons.prefs file available in the scanned directories the actual icon files bust be available in the **images/16x16** and **images/24x24** sub directories such as `${dbvis.prefsdir}/ext/profiles/images/16x16`. The 16x16 directory should contain a 16 by 16 sized icon and the 24x24 a 24 by 24 sized icon.

This is an example of the `${dbvis.prefsdir}/ext/profiles/icons.prefs` file:

```
sample-schema-dict=dict
```

The actual icon represented by the **dict** file name is located in:

```
${dbvis.prefsdir}/ext/profiles/images/16x16/dict.png
${dbvis.prefsdir}/ext/profiles/images/24x24/dict.png
```

22.6 Conditional Processing

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

22.6.1 Introduction

Conditional processing simply means that a profile can adjust its content based on certain conditions. A few examples:

- Which version of the database is being accessed
- The format of the database URL
- The client environment i.e Java version, vendor, etc.
- User properties
- Database connection properties

Conditional processing is especially useful when adapting the profile for different versions of the database (and/or JDBC driver). Another use is to replace generic error messages with more user friendly messages.

If you have some programming skills conditions are expressed using **if**, **else if** and **else** statements.

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 of the two phases. If this attribute is set to the value **runtime**, it will be processed in the second phase. If it is not specified, it will be processed in the first phase.

22.6.2 Conditional processing when database connection is established

These are the call signatures for the utilities used when processing conditions:

```
boolean #util.isNull(String string)
boolean #util.isNullOrWhitespace(String string)
boolean #util.isDatabaseVersionLT(Integer major)
boolean #util.isDatabaseVersionLT(Integer major, Integer minor)
boolean #util.isDatabaseVersionLTE(Integer major)
boolean #util.isDatabaseVersionLTE(Integer major, Integer minor)
boolean #util.isDatabaseVersionEQ(Integer major)
boolean #util.isDatabaseVersionEQ(Integer major, Integer minor)
boolean #util.isDatabaseVersionGTE(Integer major)
boolean #util.isDatabaseVersionGTE(Integer major, Integer minor)
boolean #util.isDatabaseVersionGT(Integer major)
boolean #util.isDatabaseVersionGT(Integer major, Integer minor)
boolean #util.isDatabaseType(String type)
boolean #util.isNotDatabaseType(String type)
```

The following example shows the use of conditions that are processed during connect of the database connection.

```
<Command id="sybase-ase.getLogins">
   <If test="#util.isDatabaseVersionLTE(5)">
      <SQL>
         <![CDATA[
select name from master.dbo.syslogins
         ]]>
     </SQL>
   </If>
   <ElseIf test="#util.isDatabaseVersionEQ(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 5, the first SQL is used. If the major version is equal to 9, the second SQL is used, and the last SQL is used for all other versions. 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="#util.isDatabaseVersionGTE(4, 1)">
   <GroupNode type="TableEngines" label="Table Engines" isLeaf="true"/>
   <!-- "Errors" was added in MySQL 5 -->
   <If test="#util.isDatabaseVersionGTE(5)">
```

```
      <GroupNode type="Errors" label="Errors" isLeaf="true"/>
   </If>
</If>
<Commands>
   <OnError>
      <!-- The ORA-942 error means "the table or view doesn't exist" -->
      <!-- It is catched 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>
```

As you can see, this example contains **nested** uses of **If**.

22.6.3 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 catched 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 its commands. If it's part of a specific Command, it is processed only for that command.

22.7 Database Profile Utilities

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

In the **Database Profiles** list in **Connection Properties** there is a right-click menu with various commands to process the database profile and list of profiles. These

commands are useful while developing a database profile. Read the following sections for more details about each command.

- Analyze Database Profile (see page 413)
- Show All Type and Icon Attributes (see page 414)
- Show Available Icons (see page 415)
- Export Merged Profile (see page 416)
- Configure Search Path (see page 416)
- Reload Database Profiles List (see page 417)

22.7.1 Analyze Database Profile

This command verify the loaded database profile and perform various consistency checks and may report:

Check
GroupNode and DataNode elements with no matching ObjectView (by "type" attribute)

Check
ObjectView elements with no matching GroupNode or DataNode (by "type" attribute)
ActionGroup elements with no matching GroupNode or DataNode (by "type" attribute)
DataView elements located in the same ObjectView having same "label" attribute
No matching icon for these elements (by the "type" or "icon" attributes)

22.7.2 Show All Type and Icon Attributes

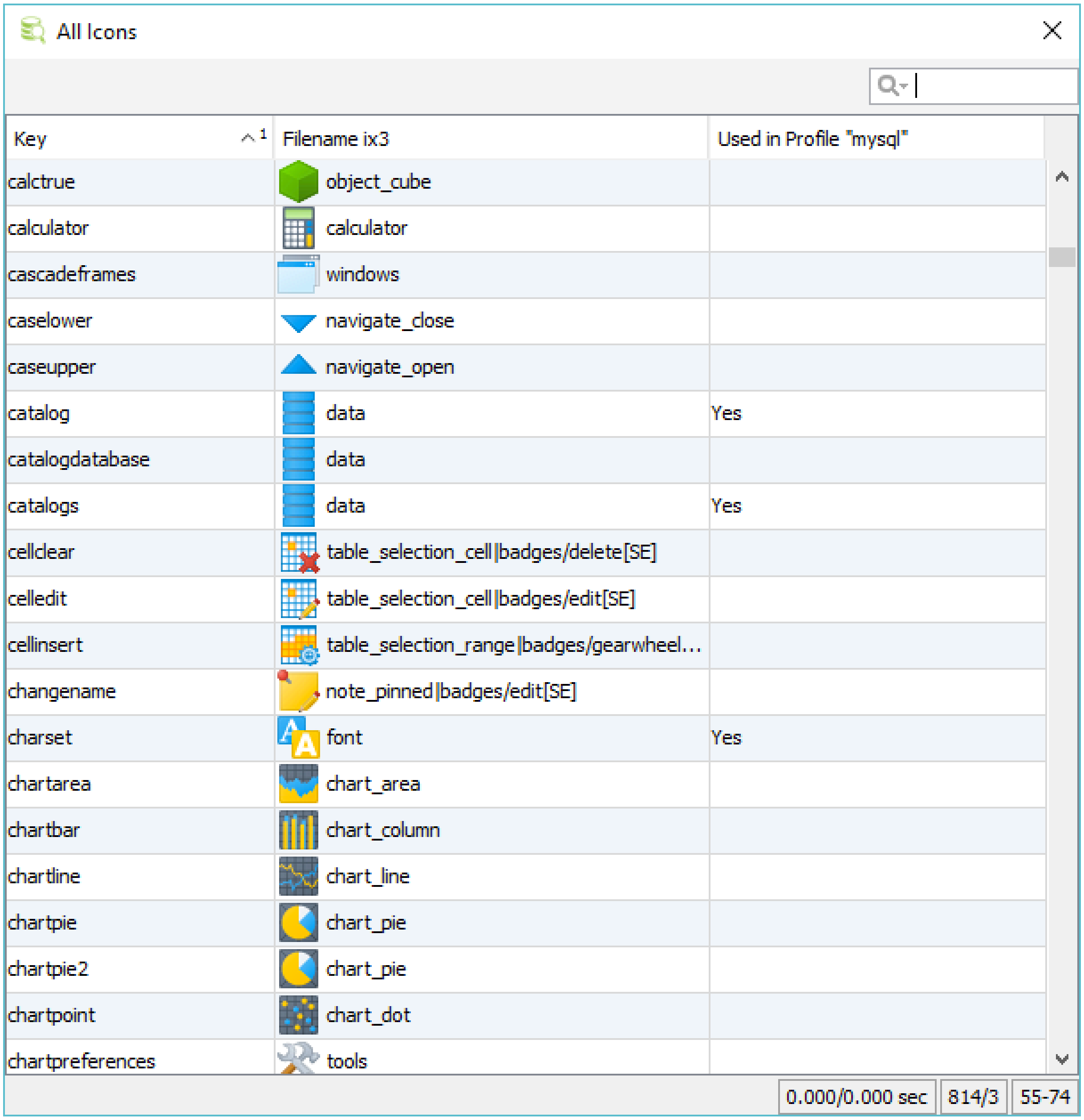
This command examine the loaded profile and report all used icons (logical name) as referenced by the object type and specific icon attributes. The following is an example doing running this command with the MySQL profile loaded.

```
These are all "type" and "icon" references in the profile:
add
catalog
catalogs
charset
collation
column
columns
data
databases
dba
edit
errors
event
events
export
function
functions
import
index
indexes
info
login
navigator
primarykey
privileges
procedure
procedures
process
processes
references
remove
rename
```

```
role
rowcount
rowid
schema
schemas
scriptobject
slavestatus
source
sourceeditor
stateerror
status
table
tableengine
tableengines
tableprivileges
tables
trigger
triggers
users
variables
view
views
warning
warnings
```

22.7.3 Show Available Icons

This will show a window with all available icons, their name, icon file name and an indicator if the icon is used in the loaded database profile.



All Icons

Key	Filename ix3	Used in Profile "mysql"
calctrue	object_cube	
calculator	calculator	
cascadeframes	windows	
caselower	navigate_close	
caseupper	navigate_open	
catalog	data	Yes
catalogdatabase	data	
catalogs	data	Yes
cellclear	table_selection_cell|badges/delete[SE]	
celledit	table_selection_cell|badges/edit[SE]	
cellinsert	table_selection_range|badges/gearwheel...	
changename	note_pinned|badges/edit[SE]	
charset	font	Yes
chartarea	chart_area	
chartbar	chart_column	
chartline	chart_line	
chartpie	chart_pie	
chartpie2	chart_pie	
chartpoint	chart_dot	
chartpreferences	tools	

0.000/0.000 sec | 814/3 | 55-74

22.7.4 Export Merged Profile

This will export the currently loaded profile. If the profile has been merged using the extend (see page 345) attribute the exported profile file is the complete and processed profile.

22.7.5 Configure Search Path

This will open **Tool Properties / General / Database Connection / Database Profile** pane used to configure the search path for database profile loading.

22.7.6 Reload Database Profiles List

This will reload the list of database profiles in case a new profile has been added (or renamed) since last launch of DbVisualizer.

22.8 Database Profile changes in 9.5

This document collects changes in the database profile framework in DbVisualizer 9.5.

22.8.1 New/changed attributes for Command

These are the changes in the attributes for the Command element.

- exectype="**script**/asis/explain" (replaces parsesql="true/false")
  - **script** will execute each individual statement in a multi statement script, **asis** will execute all of the SQL as one statement and **explain** will try generate an explain plan for the single statement
- processmarkers="**true**/withdriver"
  - Defines whether markers such as :name, ? should be processed either internally by DbVisualizer or by letting the driver (if it supports it) do the parsing
- autocommit="**true**/false"
  - Specifies if the command should be auto committed after execution
- whensuccess="commit/rollback/ask"
  - If autocommit="false" then this attribute defines what to do after successful execution
- whenerror="commit/rollback/ask/ignore"
  - If autocommit="false" then this attribute defines what to do after failed execution

22.8.2 Action element improvements

- Moved processmarkers="true|false" to **Command processmarkers="true|withdriver"**

- Removed **execute** value for the **mode** attribute (since it is default)
- Removed **resulttype="resultset|dbmsoutput"** and replaced by using the **@set server output** client side command in script when needed
- Replaced **resultaction="ask|show|showtext|script"** with **resultaction="mergeastext/mergeasscript"**
- Added **resulttarget="editor"**

Notes:

- If none of **resultaction** and **resulttarget** is present and there are results generated, the results are presented as tabs (same as SQL Commander). The result set tab option to **Merge All Result Sets** tabs is available
- If **resultaction** is present all result sets will be merged into a single result tab
- If **resulttarget** is present it will copy the merged results to a new SQL Commander tab

New runsetdefaultwhenvaluechanged attribute

The **runsetdefaultwhenvaluechanged** attribute defines what other inputs should be triggered when the value for the input is changed. Whenever this happens the other inputs Default commands are executed. This is useful if there are for example two list inputs. Changing the selected entry in the first list will then notify the second input so that its Default command is re-executed using the new value as condition.

```
<Action id="postgresql-settings-alter" label="Alter Setting" reload="true" icon="edit">
   <Input name="setting" label="Setting" runsetdefaultwhenvaluechanged="value,desc" style="list">
      <Values>
         <Command><SQL><![CDATA[select name from pg_settings]]></SQL></Command>
      </Values>
   </Input>
   <Input name="value" label="Value" style="text">
      <Default>
         <Command><SQL><![CDATA[select setting from pg_settings where name = '${setting}']]></SQL><
/Command>
      </Default>
   </Input>
   <Input name="desc" label="Description" style="text" editable="false">
      <Default>
         <Command><SQL><![CDATA[select short_desc from pg_settings where name = '${setting}']]></SQ
L></Command>
      </Default>
   </Input>
   <Command>
      <SQL><![CDATA[alter system set ${setting} = ${value}]]></SQL>
   </Command>
   <Confirm>
      Really alter setting ${setting}?
   </Confirm>
</Action>
```

Check list style

The new check-list input shows a list with options allowing multiple selections.

```
<Input label="Weekdays" name="weekdays" style="check-list">
   <Arg name="output" value="jschours[${value}] = true"/>
   <Arg name="newline" value=", "/>
   <Labels>Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday</Labels>
   <Values>00|11|22|33|44|55|66</Values>
   <Default>22|44</Default>
</Input>
```

The `<Arg name="output" value="jschours[${value}] = true"/>` defines that for each selected value in the list (ex Monday, Thursday, Friday), the output in the **weekdays** variable will be:

`jschours[11] = true,` jschours[33] = true, jschours[44] = true

The **newline** argument defines the string used to separate the output entries.

Check style

Check inputs now supports a second values argument:

```
<Input label="Check with true/false value" name="enabled" style="check">
   <Values>On|Off</Values>
   <Default>Off</Default>
</Input>
```

The first, "On" is set if the checkbox is checked. "Off" is set if the checkbox is unchecked. If the second argument is not specified an empty string is set if the checkbox is unchecked.

Action layout

- Improved support for having input fields on the same row
- Previously when using **linebreak="false"** any label on that row was not displayed. Now if there is a label it is shown
- New: **type="label"** input attribute. Used to create just a label. Useful when **linebreak="false"** by having a general label for the row and then each component can have their individual label. Ex: "Days: Mon: X Tue: X"
- **New: type="note"** input attribute. Is the exact samt as "label" but can be used to specify some informative text that is not suffixed with a colon

22.8.3 Changes for DataNode and GroupNode

- In addition to the label attribute the new label1 is now available. In the objects tree it will show optional information about the object. This is displayed in italic gray text.
- The **sort** attribute has been replaced with the **ProcessDataSet action="sortcolumn"**

22.8.4 New utility class

The new #util class can be used in profiles to primarily perform conditional checks such as what target database version is used. This is useful to adjust what parts of the database profile should be visible to the user. These are the methods available:

```
boolean #util.isNull(String string)
boolean #util.isNullOrWhitespace(String string)
boolean #util.isDatabaseVersionLT(Integer major)
boolean #util.isDatabaseVersionLT(Integer major, Integer minor)
boolean #util.isDatabaseVersionLTE(Integer major)
boolean #util.isDatabaseVersionLTE(Integer major, Integer minor)
boolean #util.isDatabaseVersionEQ(Integer major)
boolean #util.isDatabaseVersionEQ(Integer major, Integer minor)
boolean #util.isDatabaseVersionGTE(Integer major)
boolean #util.isDatabaseVersionGTE(Integer major, Integer minor)
boolean #util.isDatabaseVersionGT(Integer major)
boolean #util.isDatabaseVersionGT(Integer major, Integer minor)
boolean #util.isDatabaseType(String type)
boolean #util.isNotDatabaseType(String type)
boolean #util.replaceAll(String source, String search, String replacement )
```

22.8.5 Changed icons definition

Many icons are composed of a full size (16x16, 24x24) base icon and a badge icon (half the size) representing things like, new, delete, error, clock, and so on. The badge icon is typically anchored south east on the base icon.

Prior to DbVisualizer 9.5 there were no separate badge icons as these were integrated with the base icon resulting in many duplicated base icons only with different badge symbols.

In DbVisualizer 9.5 a new icon library was introduced and now base icons are separate from the badge icons. Any badge icons that should be attached are defined in the icons.prefs file and the final icon is dynamically created.

23 Troubleshooting

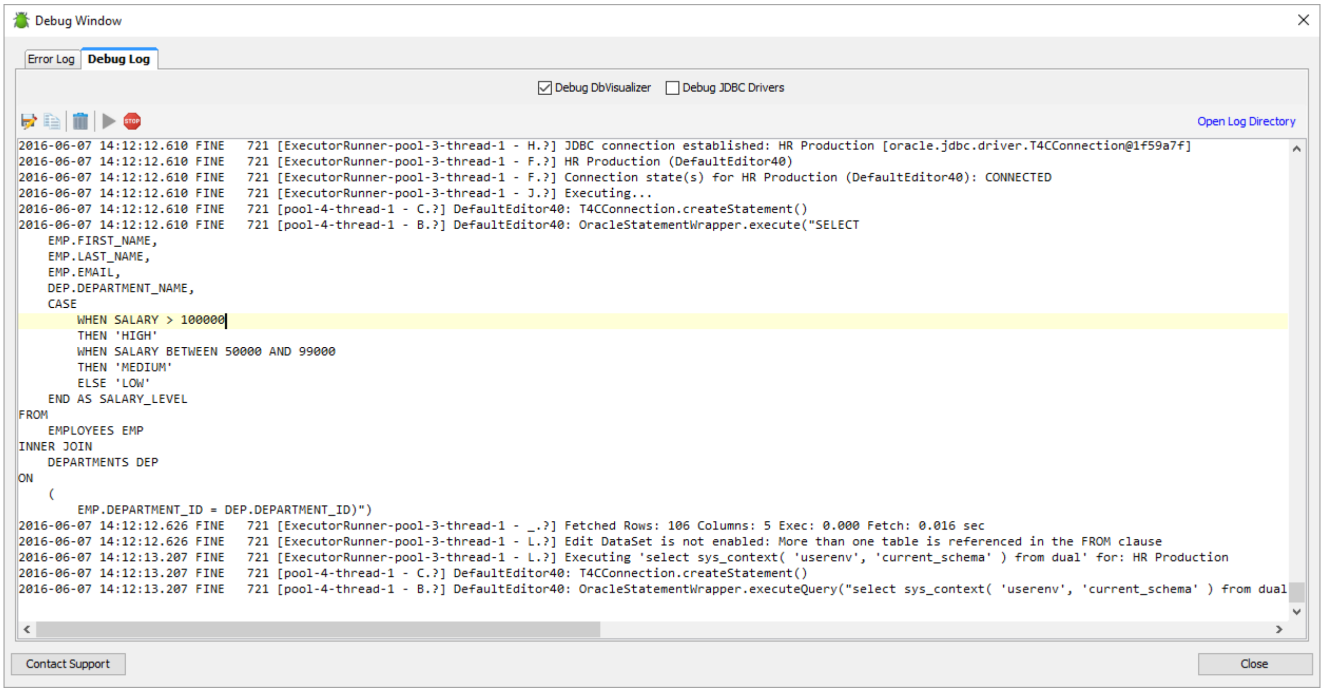
Even though we make our very best to ensure the quality of DbVisualizer, you may run into problems of different kinds. The runtime environment for DbVisualizer is rather complicated when it comes to tracking the source of a problem, since it's not only DbVisualizer that may cause the problem but also 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 (http://www.java.com/) available for your platform (Java 6 or later),
2. Make sure you are using a version (http://www.dbvis.com/doc/database-drivers/) of the JDBC driver that we've tested DbVisualizer with, or a later, production quality version,
3. Read the DbVisualizer FAQ (http://confluence.dbvis.com/display/FAQ),
4. Check the online Forums (http://www.dbvis.com/forum/),
5. Read the DbVisualizer Users Guide (http://confluence.dbvis.com/display/UG/Users+Guide),
6. ... the last resort is to post a question via the problem report form (see page 426) or send an email to support@dbvis.com (mailto:support@dbvis.com). (Note that we generally love detailed reports as well as screenshots when possible)

23.1 Debugging DbVisualizer

The **Tools->Debug Window** is useful to see what is going on in DbVisualizer and the JDBC driver(s). The checkboxes at the top of the **Debug** tab controls 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.



```
2016-06-07 14:12:12.610 FINE   721 [ExecutorRunner-pool-3-thread-1 - H.?] JDBC connection established: HR Production [oracle.jdbc.driver.T4CConnection@1f59a7f]
2016-06-07 14:12:12.610 FINE   721 [ExecutorRunner-pool-3-thread-1 - F.?] HR Production (DefaultEditor40)
2016-06-07 14:12:12.610 FINE   721 [ExecutorRunner-pool-3-thread-1 - F.?] Connection state(s) for HR Production (DefaultEditor40): CONNECTED
2016-06-07 14:12:12.610 FINE   721 [ExecutorRunner-pool-3-thread-1 - J.?] Executing...
2016-06-07 14:12:12.610 FINE   721 [pool-4-thread-1 - C.?] DefaultEditor40: T4CConnection.createStatement()
2016-06-07 14:12:12.610 FINE   721 [pool-4-thread-1 - B.?] DefaultEditor40: OracleStatementWrapper.execute("SELECT
    EMP.FIRST_NAME,
    EMP.LAST_NAME,
    EMP.EMAIL,
    DEP.DEPARTMENT_NAME,
    CASE
        WHEN SALARY > 100000
        THEN 'HIGH'
        WHEN SALARY BETWEEN 50000 AND 99000
        THEN 'MEDIUM'
        ELSE 'LOW'
    END AS SALARY_LEVEL
FROM
    EMPLOYEES EMP
INNER JOIN
    DEPARTMENTS DEP
ON
    (
        EMP.DEPARTMENT_ID = DEP.DEPARTMENT_ID)")
2016-06-07 14:12:12.626 FINE   721 [ExecutorRunner-pool-3-thread-1 - _.?] Fetched Rows: 106 Columns: 5 Exec: 0.000 Fetch: 0.016 sec
2016-06-07 14:12:12.626 FINE   721 [ExecutorRunner-pool-3-thread-1 - L.?] Edit DataSet is not enabled: More than one table is referenced in the FROM clause
2016-06-07 14:12:13.207 FINE   721 [ExecutorRunner-pool-3-thread-1 - L.?] Executing 'select sys_context( 'userenv', 'current_schema' ) from dual' for: HR Production
2016-06-07 14:12:13.207 FINE   721 [pool-4-thread-1 - C.?] DefaultEditor40: T4CConnection.createStatement()
2016-06-07 14:12:13.207 FINE   721 [pool-4-thread-1 - B.?] DefaultEditor40: OracleStatementWrapper.executeQuery("select sys_context( 'userenv', 'current_schema' ) from dual
```

The log window shows the end of the log files produced by DbVisualizer. The **Debug Log** tab shows the debug log file and the **Error Log** tab the error log file. DbVisualizer writes its log files to a log directory which may be opened using the **Open Log Directory** link at the top right. DbVisualizer maintains the content of the log directory in a way that limits the size of its content. The content of the log tabs is also automatically truncated to preserve memory.

The toolbar contains the following buttons:

- **Save** saves the content of the visible log tab to file
- **Copy** copies any selection to the clipboard
- **Clear** clears the content of the log tab
- **Start** starts monitoring of the corresponding log file. This button is of course disabled if monitoring is already started
- **Stop** stops the monitoring of the log files (if started)

The button labeled **Contact Support** serves as a shortcut to open the window for Reporting Issues (see page 426) where you may optionally attach the content of the DbVisualizer log files to a support ticket.

23.2 Fixing Connection Issues

There may be many reasons for why you cannot connect to a database, but some of the most common are:

- Incorrect values for the **Database Server** or **Database Port** fields in the **Object View** tab for the connection,
- TCP/IP access is not enabled in the database server,
- A firewall between the client and the database server blocks connections to the database port,

- A syntax error in a manually entered JDBC URL,
- The user account is not authorized to connect from the client where you run DbVisualizer,
- Native libraries for a JDBC driver are not found.

The first three problems usually results in a somewhat cryptic message about I/O errors or a time-out. You can use the **Ping Server** button to make sure that the a TCP/IP network connection can be established to the specified server and port. If this test fails, please ask your system or database administrator for help.

JDBC syntax errors typically result in one of two error messages:

- "The selected Driver cannot handle the specified Database URL. The most common reason for this error is that the database URL contains a syntax error preventing the driver from accepting it. The error also occurs when trying to connect to a database with the wrong driver. Correct this and try again."
- "java.sql.SQLException: Io exception: Invalid number format for port number Io exception: Invalid number format for port number"

In both cases, we recommend using the **Server Info** settings format instead of the **Database URL** format for the connection, and let DbVisualizer build a valid JDBC URL for you. If you must enter a JDBC URL manually, make sure that you replace possible placeholders enclosed with "<" and ">" in a template you have copied, such as < 1521>, and look for other syntax errors. Also verify that the JDBC driver is installed (see page 430) correctly.

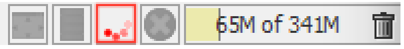
Authorization problems are usually described by more straight forward messages. Ask you database administrator to help you get it resolved.

If you get a message about native libraries not being found, e.g. " no ocijdbc11 in java.library.path" or similar, it is because you have not installed these in a location where DbVisualizer can read them. Unless you have a very good reason for using a JDBC driver that requires native code, we recommend that you use a pure Java JDBC driver (a Type 4 driver) instead, like the "Oracle Thin" driver for Oracle.

If none of this helps, please contact us using the **Help->Contact Support** form. Most of the information we can gather about the problem is typically already filled in, but please add any additional details that may help us figure out what is wrong.

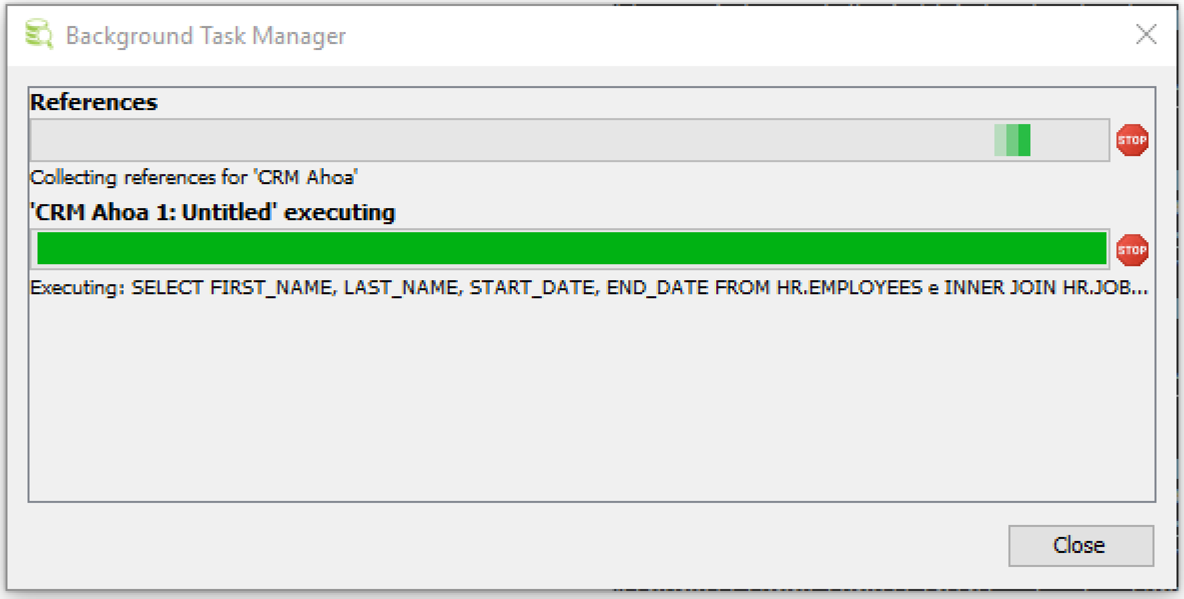
23.3 Handling Dropped Connections

All tasks that may potentially take a bit of time to perform or that may not finish at all because a database connection has dropped are executed in the background so that you can still do other things. You can see that background tasks are running by looking at an indicator icon in the status bar.



If there are tasks running, the indicator icon is animated, showing a spinning pattern.

You can see exactly which tasks are running by clicking on the icon. This opens the **Task Manager** window.



References

Collecting references for 'CRM Ahoa'

'CRM Ahoa 1: Untitled' executing

Executing: SELECT FIRST_NAME, LAST_NAME, START_DATE, END_DATE FROM HR.EMPLOYEES e INNER JOIN HR.JOB...

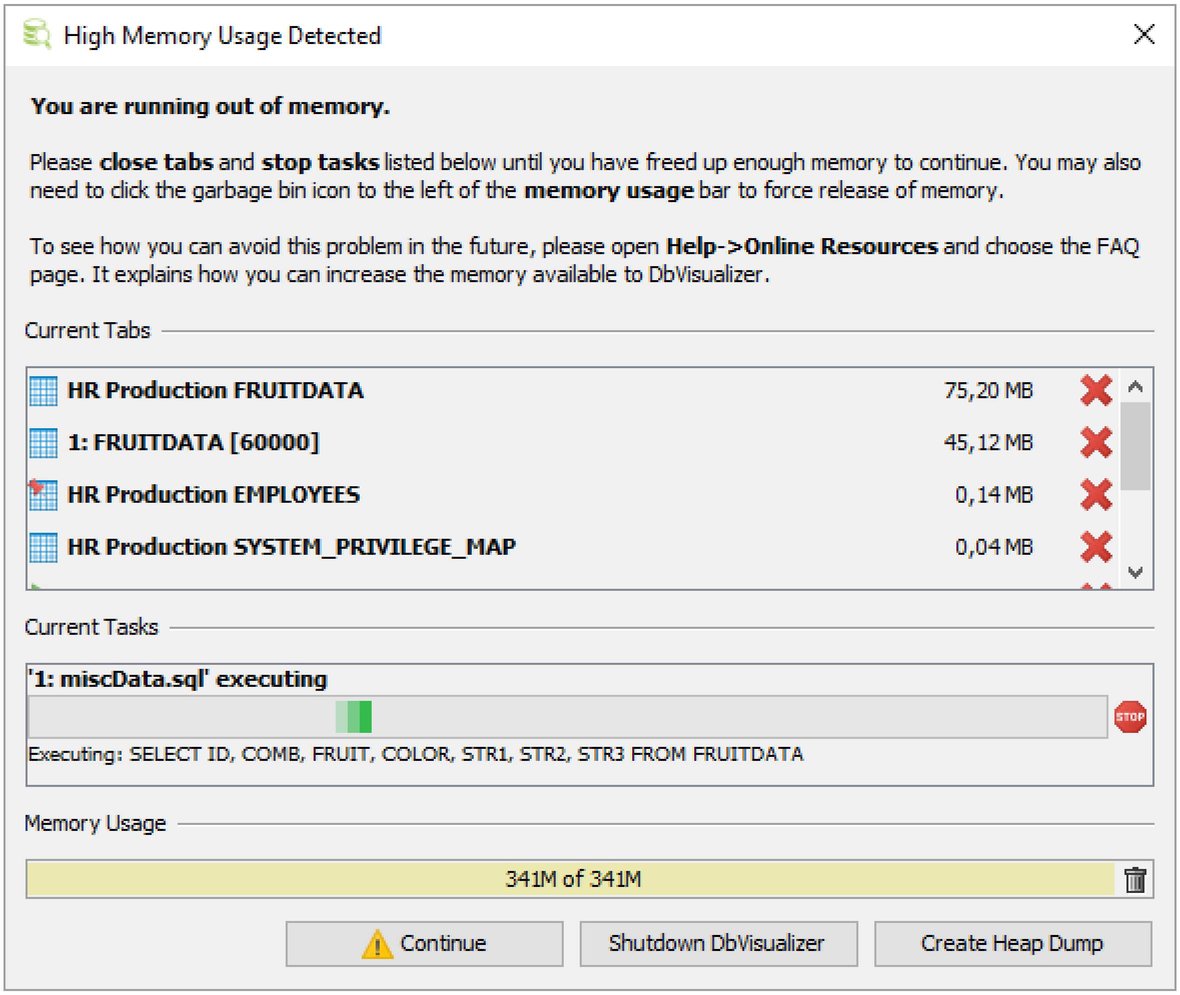
You can abort a long running task by clicking the **Stop** button next to the progress bar.

If the task did not complete because the connection dropped, you also need to reconnect to the database. If this happens frequently, you can enable Connection Keep-Alive (see page 298) to send a dummy SELECT statements to the database occasionally to prevent time-outs.

23.4 Handling Memory Constraints

DbVisualizer has a fixed amount of memory available, and things may go bad if you load so much data that you're getting close to this limit. The most common effect is that the GUI becomes very unresponsive or freezes completely.

To minimize the risk of this happening, DbVisualizer keeps track of the memory usage when you load tables or files and similar tasks that consume memory. If you're getting so close to the limit that problems are likely to begin to show, all memory consuming background tasks are suspended and the High Memory Usage window pops up.



High Memory Usage Detected

You are running out of memory.

Please **close tabs** and **stop tasks** listed below until you have freed up enough memory to continue. You may also need to click the garbage bin icon to the left of the **memory usage** bar to force release of memory.

To see how you can avoid this problem in the future, please open **Help->Online Resources** and choose the FAQ page. It explains how you can increase the memory available to DbVisualizer.

Current Tabs

HR Production FRUITDATA	75,20 MB
1: FRUITDATA [60000]	45,12 MB
HR Production EMPLOYEES	0,14 MB
HR Production SYSTEM_PRIVILEGE_MAP	0,04 MB

Current Tasks

'1: miscData.sql' executing

Executing: SELECT ID, COMB, FRUIT, COLOR, STR1, STR2, STR3 FROM FRUITDATA

Memory Usage

341M of 341M

All open tabs are listed along with an estimate for how much memory each tab uses. All background tasks are also shown. You need to resolve the high memory usage problem before you can continue working with DbVisualizer. Click on the red cross next to tabs that use lots of memory to close them and stop tasks that consume memory.

When you have released enough memory to get below the critical level, the icon in the Continue button changes to a green checkmark. Click it to close the window and continue to work.

If you often see this window, first consider using features that minimizes the memory usage, such as using the `@run` command to execute your script (see page 180) and the `@export` command to write query results to a file (see page 214). As a last resort, you can increase the amount of memory DbVisualizer can use. Please see the FAQ page (http://confluence.dbvis.com/pages/viewpage.action?pageId=3146118) on out web site for how to do so.

In rare cases, closing tabs and stopping tasks do not release enough memory to continue, possibly due to memory leaks in the DbVisualizer code. If this happens, you can click the **Create Heap Dump** button. The heap dump file is named *heap.bin* and stored in the preferences folder, typically the *.dbvis* folder in your home folder. It may help us find memory leaks and fix them, so please send it to us with a description of what you were doing when you ran into this problem. The file can be huge, so please compress it (e.g. using the zip command) before mailing it to us.

If you cannot release enough memory to continue, you can use the **Shutdown DbVisualizer** button to shut down and start fresh.

23.5 Reporting Issues

23.5.1 Contacting support

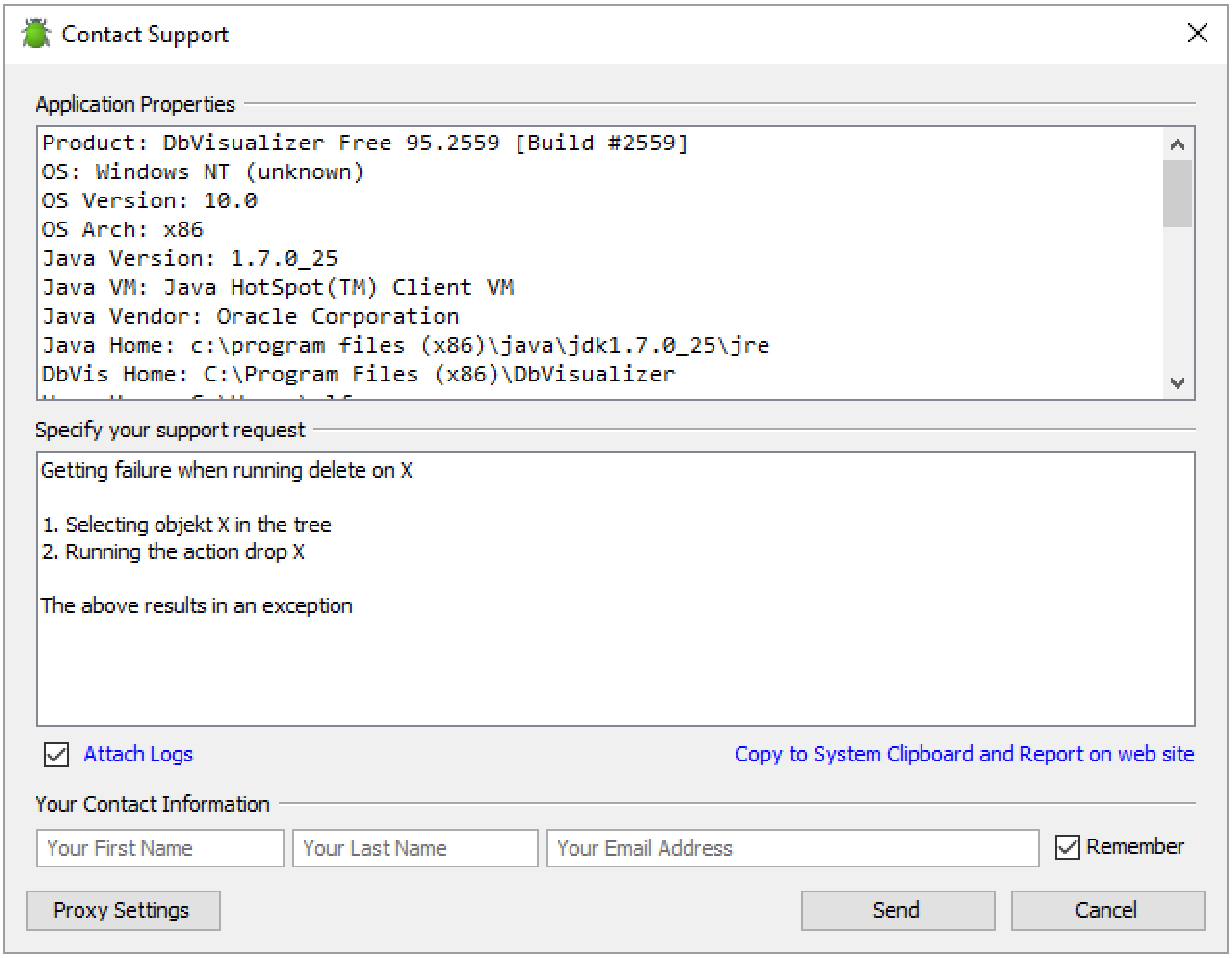
There following are the alternatives getting help with issues or support requests:

- Using Help->Contact Support in the DbVisualizer application (Recommended)
- Using the web form at https://www.dbvis.com/company/contact/
- Using the forums and knowledge base at http://support.dbvis.com

Using Help-Contact Support

This is the recommend way to report an issue or support requests as it gathers information about your settings and connections that help us provide you with better analysis of any problems without having to get back to you and ask for additional information.

Please describe as detailed as possible the actions leading to the issue.



```
Product: DbVisualizer Free 95.2559 [Build #2559]
OS: Windows NT (unknown)
OS Version: 10.0
OS Arch: x86
Java Version: 1.7.0_25
Java VM: Java HotSpot(TM) Client VM
Java Vendor: Oracle Corporation
Java Home: c:\program files (x86)\java\jdk1.7.0_25\jre
DbVis Home: C:\Program Files (x86)\DbVisualizer
```

Getting failure when running delete on X

1. Selecting objekt X in the tree
2. Running the action drop X

The above results in an exception

☑ Attach Logs

☑ Remember

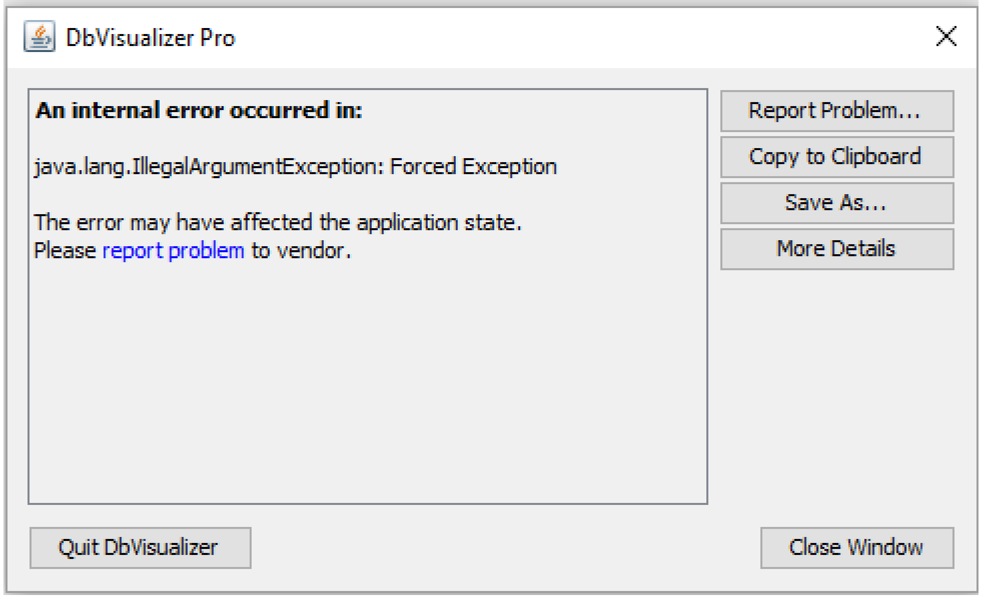
23.5.2 Encountering Errors

DbVisualizer sometimes detects errors that need the attention from the user. This can be done in two ways

- The pop up of an Error dialog
- The pop up of a "Red balloon" in the status bar of DbVisualizer

The Error Dialog

If you encounter an error that causes an error alert dialog to pop up, please click the **Report Problem** button to open the contact form with the details of the error filled in.


DbVisualizer Pro

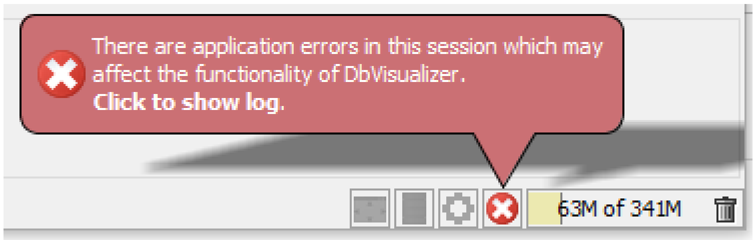
An internal error occurred in:

java.lang.IllegalArgumentException: Forced Exception

The error may have affected the application state.
Please report problem to vendor.

The Red Balloon - log errors

DbVisualizer constantly monitor the log entries it writes to it's log files. If log entries representing errors are detected, a "Red balloon" pops up in the DbVisualizer status bar as indicated below.


There are application errors in this session which may affect the functionality of DbVisualizer.
Click to show log.

Clicking the red button will bring up the debug window (see page 421).

23.6 Using special characters in passwords

Passwords entered for connections etc. can optionally be saved between sessions by DbVisualizer. Passwords containing special characters such as can not be saved when the default encoding for passwords is used. More specifically DbVisualizer only supports characters in the 8859-1 character set with the default encoding; other characters are corrupted when saved.

To be able to save passwords containing any characters, you need to enter a Master Password. Please see the section Setting a Master Password (see page 294) for more information.

24 Reference Material

Here you find details about areas that are not covered elsewhere.

24.1 GUI Command Line Arguments

As an alternative to start DbVisualizer via the menu items and icons created by the installer, you can also start DbVisualizer from a shell (terminal) on all operating systems using the following scripts:

```
# DbVisualizer GUI on Windows:
DBVIS-HOME\dbvisgui.bat
# DbVisualizer GUI on Linux/UNIX:
DBVIS-HOME/dbvisgui.sh
# DbVisualizer GUI on macOS:
DbVisualizer.app/Contents/java/app/dbvisgui.sh
```

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

```
Usage: dbvisgui [-connection <name>] [-userid <userid>] [-password <password>]
                [<filename>] [-encoding <encoding>]
                [-prefsdir <directory>]
                [-windowtitle <title>]
                [-help] [-version]
General Options:
 -connection <name>    Database connection name (created with the GUI)
 -userid <userid>      Userid to connect as
 -password <password>  Password for userid
 <filename>            SQL script file to load into editor
 -encoding <encoding>  Encoding for the SQL script file
 -prefsdir <directory> Use an alternate user preferences directory
 -windowtitle <title>  Additional window title
 -help                 Display this help
 -version              Show version info
```

Please note that these scripts use the first Java version that is found in the PATH. The result may be that a non supported Java version is used.

24.2 Installation Structure

The installer and launcher for DbVisualizer is based on the install4jTM product (http://www.install4j.com (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/
editor/
jdbc/
lib/
resources/
wrapper/
dbvis.vmoptions
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 (http://confluence.dbvis.com/display/FAQ) for the latest information.

24.3 Installing a JDBC Driver

DbVisualizer bundles JDBC drivers for most common databases, so typically you do not need to install a JDBC driver.

This page describes the way JDBC drivers are managed in DbVisualizer. If a JDBC driver for your database is bundled with DbVisualizer, see Driver Info on the Supported Databases (http://www.dbvis.com/doc/database-drivers/) page, you typically do not need to read this chapter.

If, however, any of the these things apply to you, keep on reading:

- want to learn how the Driver Manager in DbVisualizer works,
- need to have several versions of the same JDBC driver loaded simultaneously,
- need to add a Driver that does not exist in the list of default drivers .

24.3.1 What is a JDBC Driver?

DbVisualizer is a generic tool for administration and exploration of databases. DbVisualizer does not deal directly with how to communicate with each database type. That job is done by a JDBC driver, which is a set of Java classes. All JDBC drivers conform to the JDBC specification and its standardized Java programming 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 drivers available from the database vendors themselves as well as from third parties. To establish a connection to a database, DbVisualizer loads the driver and then gets 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 DbVisualizer Driver Manager. This context is then used to lookup a database connection.

The following sections describe the steps for installing a JDBC Driver, and also how to configure DbVisualizer to use JNDI to obtain a database connection.

24.3.2 Get the JDBC driver file(s)

DbVisualizer comes bundled with all commonly used JDBC drivers that have licenses that allow for distribution with a third party product. Currently, drivers for DB2, H2, JavaDB/Derby, Mimer SQL, MySQL, NuoDB, Oracle, PostgreSQL, SQLite, Vertica 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 chapter and jump straight to the Creating a Connection (see page 17) page, 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 web page contains an up-to-date listing of the database/driver combinations we have tested:

http://www.dbvis.com/doc/database-drivers/

To find a JDBC driver for your database, go to the database vendor's website or search for the name of the database plus the word `JDBC`.
(http://www.dbvis.com/doc/database-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 (http://www.dbvis.com/doc/database-drivers/) web page describes where you can download some drivesr 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.

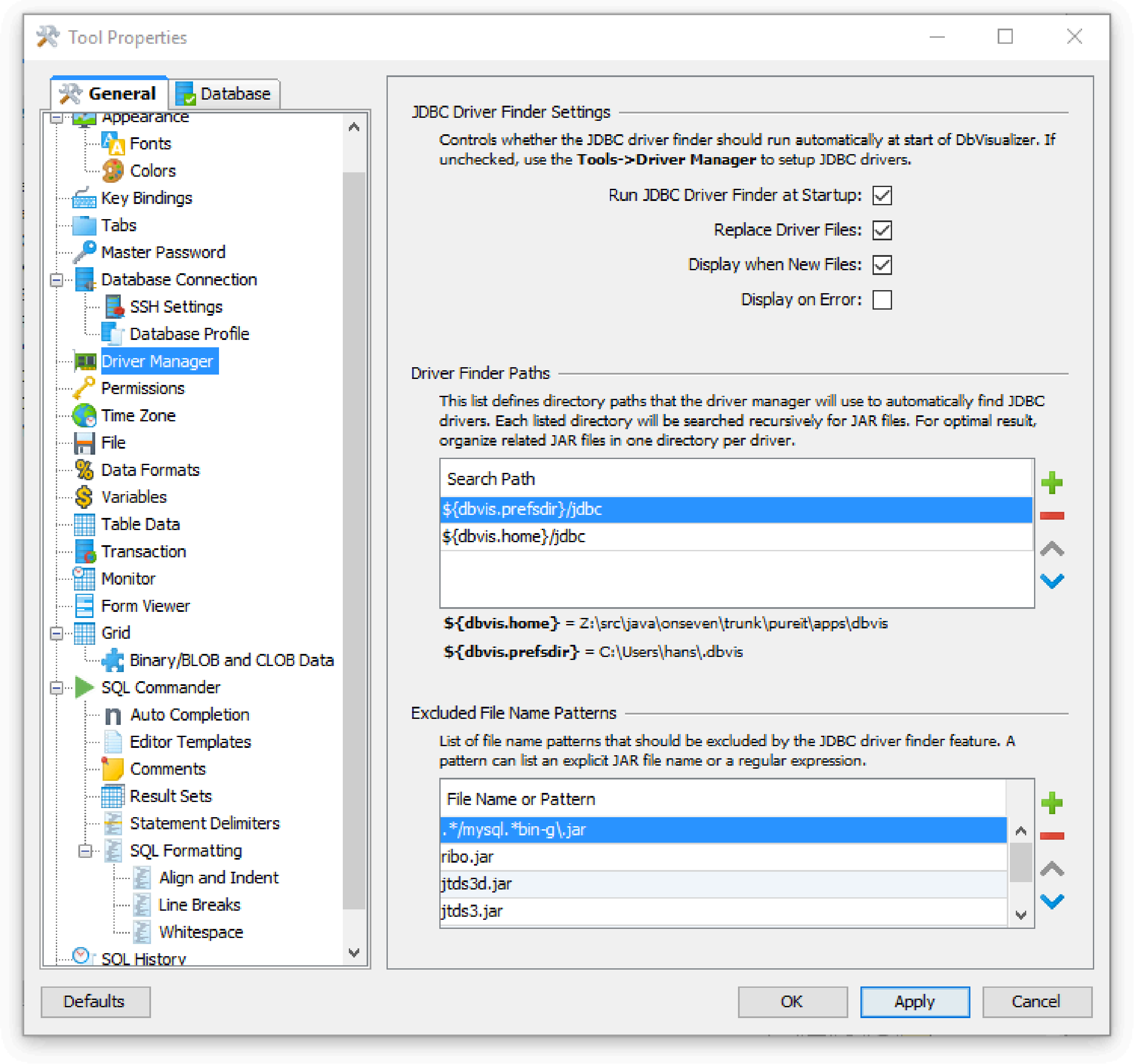
When you have downloaded the JDBC driver into a local folder (and unpacked it, if needed), you can go ahead and create a database connection with the Connection Wizard, as described in the Creating a Connection (see page 17) page. You will then be asked to load the driver files when the wizard needs 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 (see page 432) the next time you start DbVisualizer.

24.3.3 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->Tools Properties**, in the **Driver Manager** category under the **General** tab.



JDBC Driver Finder Settings

Controls whether the JDBC driver finder should run automatically at start of DbVisualizer. If unchecked, use the **Tools->Driver Manager** to setup JDBC drivers.

Run JDBC Driver Finder at Startup: ☑

Replace Driver Files: ☑

Display when New Files: ☑

Display on Error: □

Driver Finder Paths

This list defines directory paths that the driver manager will use to automatically find JDBC drivers. Each listed directory will be searched recursively for JAR files. For optimal result, organize related JAR files in one directory per driver.

Search Path
${dbvis.prefsdir}/jdbc
${dbvis.home}/jdbc

${dbvis.home} = Z:\src\java\onseven\trunk\pureit\apps\dbvis

${dbvis.prefsdir} = C:\Users\hans\.dbvis

Excluded File Name Patterns

List of file name patterns that should be excluded by the JDBC driver finder feature. A pattern can list an explicit JAR file name or a regular expression.

File Name or Pattern
.*/mysql.*bin-g\.jar
ribo.jar
jtds3d.jar
jtds3.jar

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.
	If enabled, the driver files are replaced for the matching driver even if the driver already has proper driver files.

Property	Description
Replace Driver Files	
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 **Tools->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.prefsdir}`). 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.

⚠ The Driver Finder is always activated when upgrading from an older DbVisualizer version.

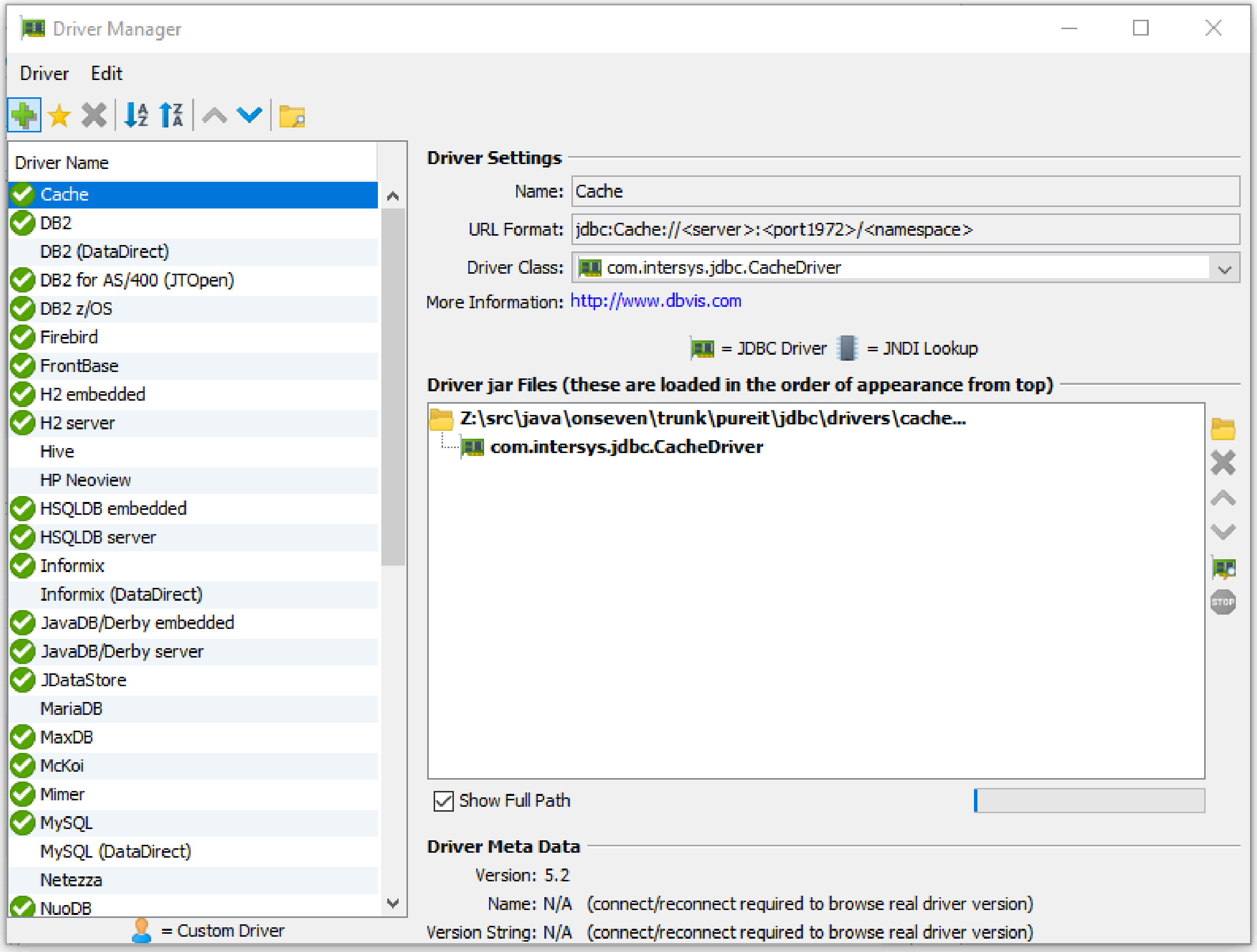
Loading and Configuring Drivers Manually

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 Connectiontab 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 URL information or a lookup name. See Using Variables in Connection Fields (see page 304) for more about how you can make it really easy to create Database Connections for this driver later on.
- **Driver Class**
  Defines the main class for the JDBC driver, used for connecting to the database.
- **Driver Version**
  Shows the version for a loaded driver.
- **Web Site**
  Link to the DbVisualizer web site, where you can get up-to-date information about how to download the drivers for many databases.
- **Driver JAR Files**
  Defines all paths to search for JDBC drivers or Initial Contexts when connecting to the database.



Driver Name

- ✓ Cache
- ✓ DB2
- DB2 (DataDirect)
- ✓ DB2 for AS/400 (JTOpen)
- ✓ DB2 z/OS
- ✓ Firebird
- ✓ FrontBase
- ✓ H2 embedded
- ✓ H2 server
- Hive
- HP Neoview
- ✓ HSQLDB embedded
- ✓ HSQLDB server
- ✓ Informix
- Informix (DataDirect)
- ✓ JavaDB/Derby embedded
- ✓ JavaDB/Derby server
- ✓ JDataStore
- MariaDB
- ✓ MaxDB
- ✓ McKoi
- ✓ Mimer
- ✓ MySQL
- MySQL (DataDirect)
- Netezza
- ✓ NuoDB

= Custom Driver

Driver Settings

Name: Cache
URL Format: jdbc:Cache://<server>:<port1972>/<namespace>
Driver Class: com.intersys.jdbc.CacheDriver
More Information: http://www.dbvis.com

= JDBC Driver = JNDI Lookup

Driver jar Files (these are loaded in the order of appearance from top)

```
Z:\src\java\onseven\trunk\pureit\jdbc\drivers\cache...
    com.intersys.jdbc.CacheDriver
```

☑ Show Full Path

Driver Meta Data

Version: 5.2
Name: N/A (connect/reconnect required to browse real driver version)
Version String: N/A (connect/reconnect required to browse real driver version)

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 shown without an icon, or with a red cross icon if an error has been detected (such as a missing file) .

Setup a JDBC driver

The recommended way to setup a predefined driver without bundled driver files 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 **DB2 (DataDirect)**, select the corresponding driver entry in the list . You can also create a new driver or copy an existing one.

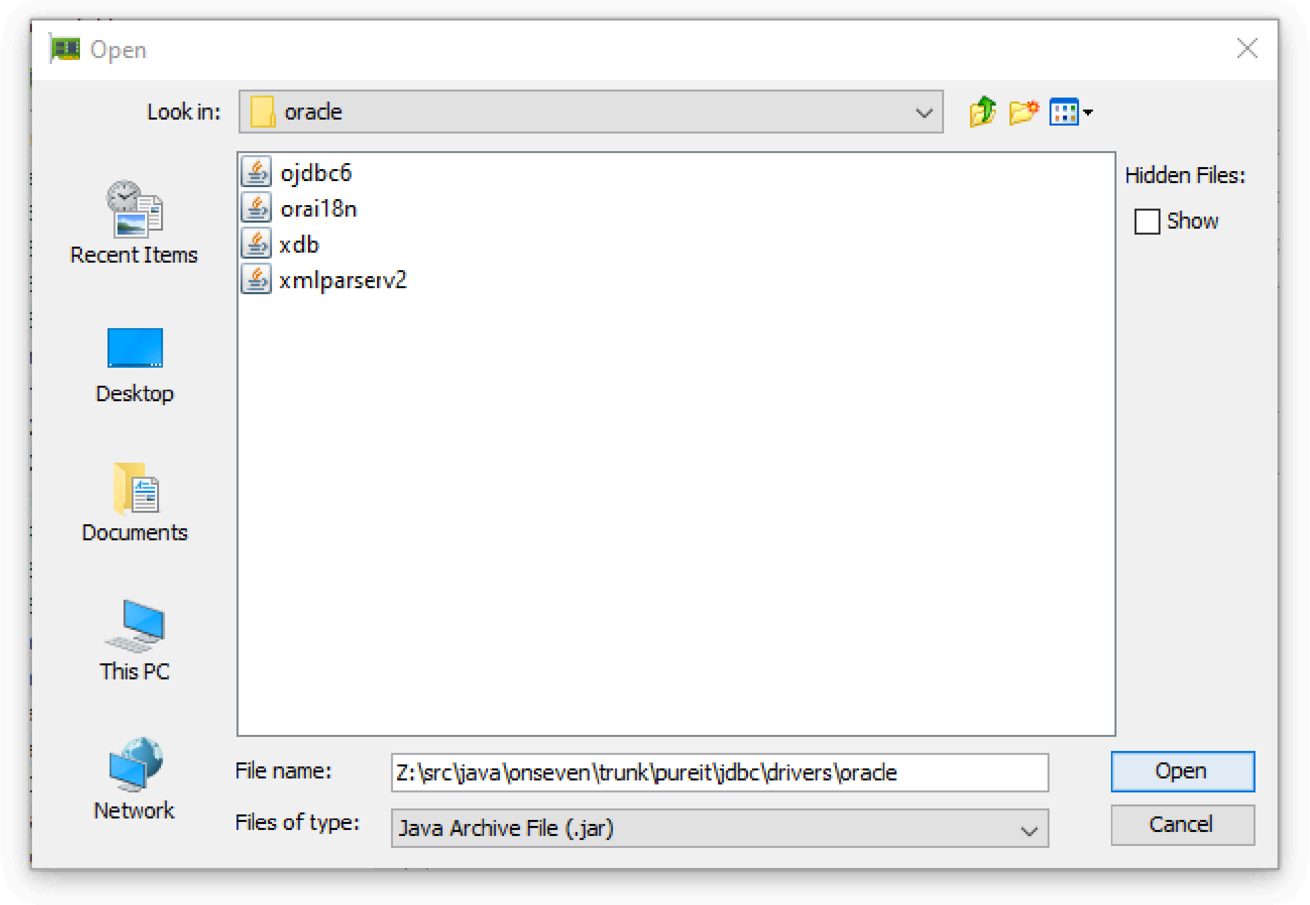
If you want to use other driver files for a bundled driver, it is best to create a new driver entry. For instance, to use an older Oracle driver, you may copy the default Oracle entry and name the new driver entry Oracle Old, and replace the bundled files with the old files for that entry. If you just replace the bundled file for the existing entry, they may be reset when you upgrade to a new DbVisualizer version.

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

http://www.dbvis.com/doc/database-drivers/

- It lists which databases and drivers we have tested
- Download links to JDBC drivers
- Information about which files to load in the driver manager for each JDBC driver
- Information about 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.



Look in: oracle

ojdbc6
orai18n
xdb
xmlparserv2

Hidden Files: □ Show

File name: Z:\src\java\onseven\trunk\pureit\jdbc\drivers\oracle

Files of type: Java Archive File (.jar)

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 ojdbc6.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.

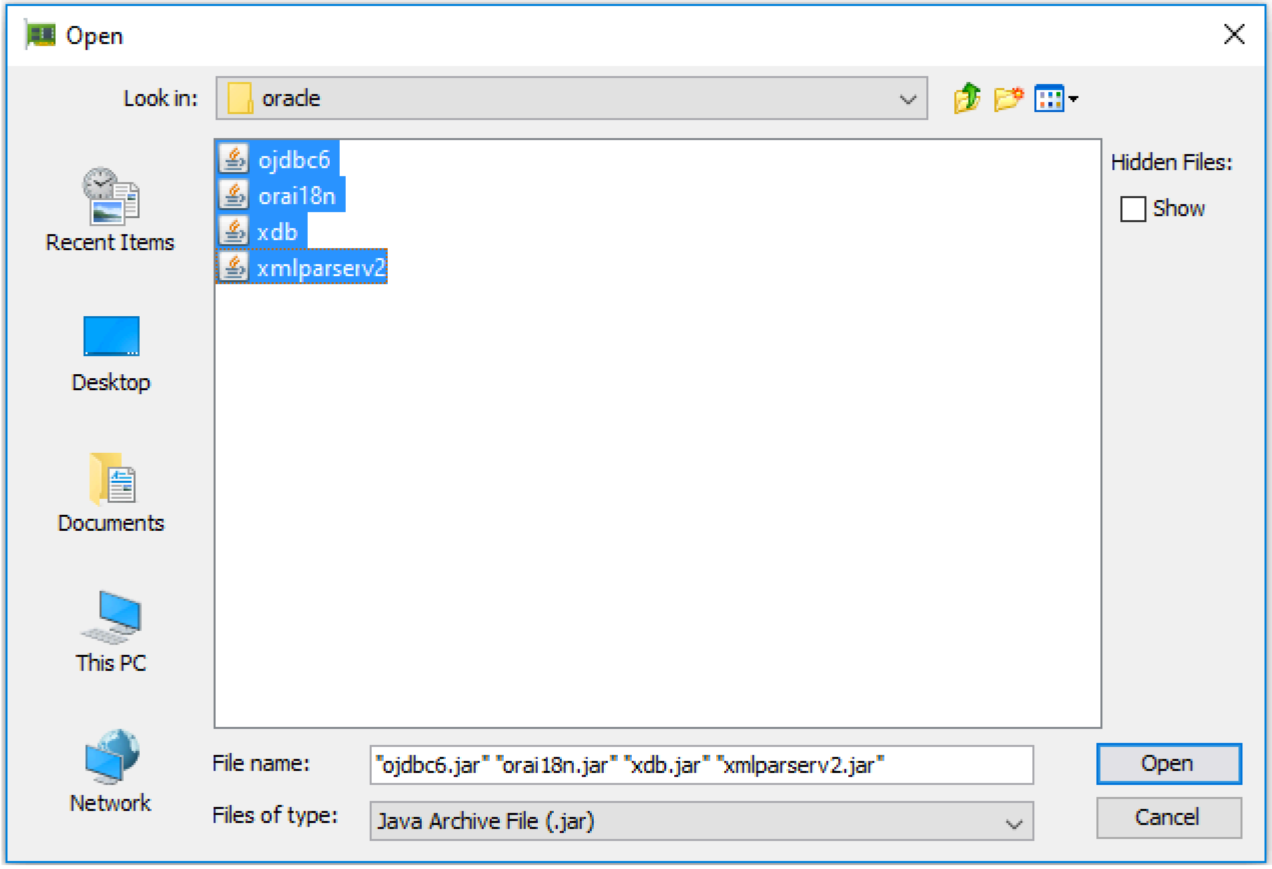
When a connection is established in the Connection tab, DbVisualizer searches the selected drivers files tree. The files are searched from the top of the tree, i.e., if there are several identical classes, the topmost class will be used. Loading several paths containing different versions of the same 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 Driver JAR Files 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 Driver JAR Files list, 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 (http://www.dbvis.com/doc/database-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. One 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.



ojdbc6
orai18n
xdb
xmlparserv2

File name: "ojdbc6.jar" "orai18n.jar" "xdb.jar" "xmlparserv2.jar"

Files of type: Java Archive File (.jar)

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.

24.4 Setting Up a JNDI Connection

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 installing a custom JDBC driver (see page 430). The difference is that you will 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.

When you create a database connection using a JNDI Lookup driver, the ***Properties sub*** tab in the connection's Object View tab will then, in this case, contain the same set of driver properties.

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 to the right of the list. Be aware that additional parameters do not necessarily mean that the InitialContext class will do anything with them.

24.5 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.

Property	Description
dbvis.-AutoSaveRunInterval=30	The number of seconds between auto-saving open SQL editors.
dbvis.disabledataedit=false	

Property	Description
	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

Property	Description
dbvis.-ConnectionTestTimeout =20	The timeout in seconds for the "Ping Server" feature.
dbvis.<database>. IgnoreMaxRowsForNonSELECT =true	Ignore the Max Rows setting for statements other than SELECT. MS SQL Server applies Max Rows also to DELETE, INSERT and UPDATE (upto and including SQL Server 2008).
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
dbvis.-FileForceSync=true	By default, all XML settings files are synced with the underlying storage device at the time when these are saved. Use this property to disable the syncing and instead rely on OS syncs. **Note**: relying on OS syncs and performing an uncontrolled shutdown of Windows may corrupt files.
dbvis.-MasterPasswordRule=.{8,}	By default, a Master Password must be at least 8 characters. This is the definition of the default implementation. Please see the *dbvis-custom.prefs* file for some additional examples.
dbvis.-MasterPasswordRuleDescr=\ The new password must be at least 8 characters long	Use this to specify a description of the master password rule.

⚠ You rarely need to modify these properties, as the default values are sufficient for most usage. Also note that these properties may change in future versions of DbVisualizer. Some are also experimental and may be removed or instead introduced in the DbVisualizer GUI.

Index