# DbVisualizer 5.1

## Users Guide

DbVisualizer

The Universal Database Tool

http://www.dbvis.com

# Table of Contents

# Getting Started and General Overview

## Introduction

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

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

**Note:** All documents in the Users Guide are primarily focusing on the DbVisualizer Personal edition.

The screen shots throughout the users guide are produced on Windows XP using the Alloy look and feel.

## Installing

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

## Installation structure

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

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

There is basically nothing in this directory that is of general interest except the **dbvis.exe** file which is used to start DbVisualizer. For information how to increase the memory for the Java process that runs DbVisualizer and also how to modify the Java version being used please read the on-line FAQ for latest information.

## Java Properties

DbVisualizer relies on few Java properties that can be used to modify characteristics of the application. These DbVisualizer specific properties are available in the **DBVIS-HOME/resources/dbvis-custom.prefs** file.

**Note:** Modifying these properties are rarely needed as the default values are sufficient for most use.

The following are the properties handled by DbVisualizer:

| Property | Description |
|---|---|
| **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 ":".<br>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 is set too. |
| **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.formeditor.unlimitedfields**=false | Specify whether the form editor should ignore the max column length and allow any number of characters to be entered |
| **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.savedatacolumns**=false | Column layout changes such as |

| | |
|---|---|
| | reordering and/or visibility is 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 DbVis saving the layout for each table that is displayed in the Data grid = huge XML file... |
| **dbvis.disabledataedit**=false | Specifies if table data editing should be completely disabled, i.e. the form and inline editors. Note: This have only effect when used with a licensed edition. |
| **dbvis.showactionresult**=false | This defines whether the result for all actions should be displayed or only failures (default). |

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

## Install license key for DbVisualizer Personal

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

## Uninstalling the license key

There may be situations when uninstalling the license key is desirable. Do this by removing (or renaming) the following file:

| Operating System | File Name |
|---|---|
| **Windows** | `C:\Documents and Settings\<user>\.dbvis\dbvis.license` |
| **UNIX/Linux** | `/home/<user>/.dbvis/dbvis.license` |
| **Mac OS X** | `/Users/<user>/.dbvis/dbvis.license` |

# Useful Resources

Resources related to DbVisualizer that are useful

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

# Starting DbVisualizer

Starting DbVisualizer depends on what platform being used.

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

## Command line arguments

DbVisualizer supports a range of command line arguments. These are listed in the **Help->Usage Information** menu choice in DbVisualizer.

```
Usage: dbvis [-help] [-up <path>] [-sqlfile <path>]
             [-windowtitle <title>]
             [connect options] [remote options]

General Options:
  -help                 Display this help
  -up <path>            Use an alternate user preferences file
  -sqlfile <path>       Load file into the SQL Commander editor
  -windowtitle <title> Additional window title
  -execute              Will execute SQL file automatically
  -invisible            No windows will be displayed

Driver Connect Options:
  -driver Setup and connect using the following Driver options:
    -alias <name>       Database alias
    -drivername <name> Driver name
    -path <path>       Path to driver class
    -class <class>     JDBC Driver class
    -url <url>         Connection URL
    -userid <user>     Userid to connect as
    -password <pw>     Connect password

JNDI Connect Options:
  -jndi Setup and connect using the following JNDI options:
    -alias <name>      Database alias
    -drivername <name> Driver name
    -path <path>       Path to initial context class
    -class <class>     Initial context class
    -url <url>         Provider URL
    -lookup <name>     Lookup name
    -userid <user>     Userid to connect as
    -password <pw>     Connect password

Remote Options:
  -attachremote Attach to remote DbVisualizer instance
  -enableremote Enable remote attachment
  -host <host>  Remote host name (default: localhost)
  -port <port>  Remote port (default: 8787)
```

# The Main Window and Common Components

The DbVisualizer user interface screenshot below is organized with the database objects tree to the left and two tabs at the right.

**Database Objects Tree**

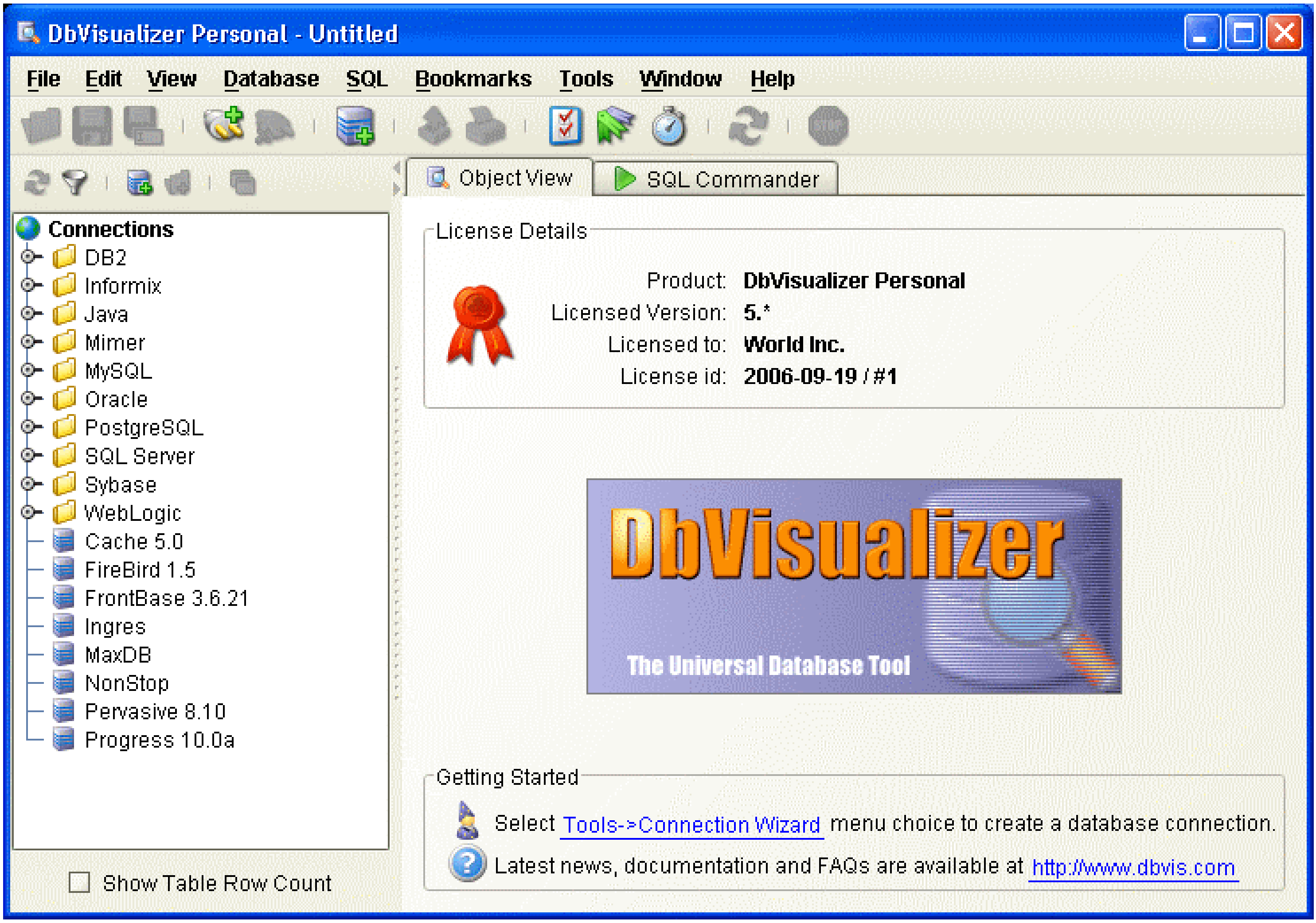
This tree keeps (at the top level) all the **Database Connection** objects (or folder objects with their purpose to organize Database Connections). Use this tree to navigate and explore the database. Clicking on an object will change the view in the **Object View** tab to show details about the selected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

**Object View**

This tab shows detailed information about the selected tree node. Every object type have their own representation in the object view tab.

**SQL Commander**

The SQL Commander is used to execute SQL statements and scripts.



License Details

Product: DbVisualizer Personal
Licensed Version: 5.*
Licensed to: World Inc.
License id: 2006-09-19 / #1

Getting Started

Select Tools->Connection Wizard menu choice to create a database connection.

Latest news, documentation and FAQs are available at http://www.dbvis.com

☐ Show Table Row Count

**Figure: The DbVisualizer main window**

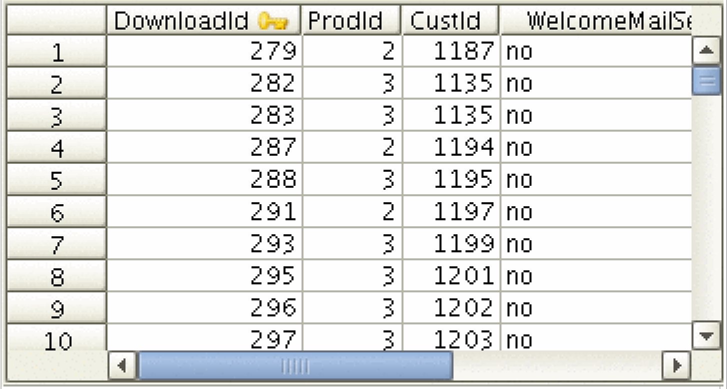
## Standard Components in the User Interface

The following section presents some generic topics that are worth knowing when you using DbVisualizer.

### Grid, Graph and Chart

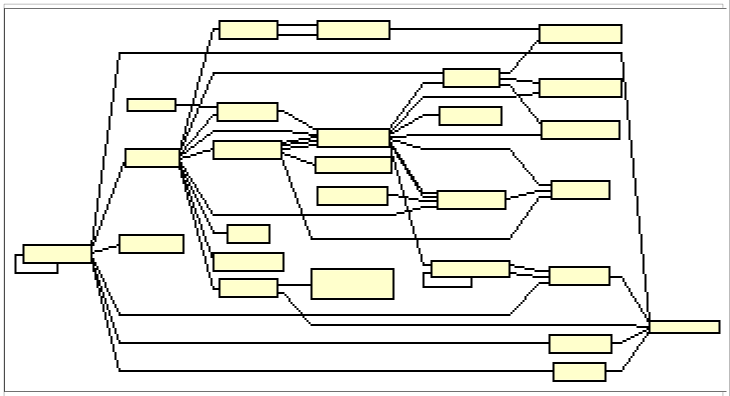
Grid, graph and chart are three terms that are often used in the application and in the documentation. The following explains what they represent.

**Grid**



| | DownloadId | ProdId | CustId | WelcomeMailSe |
|---|---|---|---|---|
| 1 | 279 | 2 | 1187 | no |
| 2 | 282 | 3 | 1135 | no |
| 3 | 283 | 3 | 1135 | no |
| 4 | 287 | 2 | 1194 | no |
| 5 | 288 | 3 | 1195 | no |
| 6 | 291 | 2 | 1197 | no |
| 7 | 293 | 3 | 1199 | no |
| 8 | 295 | 3 | 1201 | no |
| 9 | 296 | 3 | 1202 | no |
| 10 | 297 | 3 | 1203 | no |

**Graph**



**Chart**

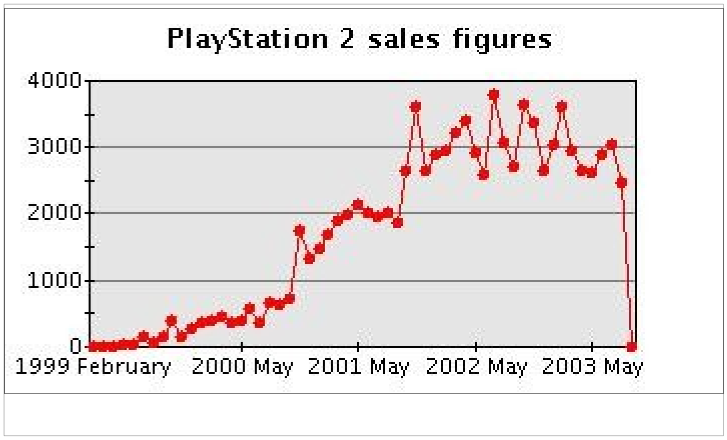


**Figure: The grid, graph and chart terms**

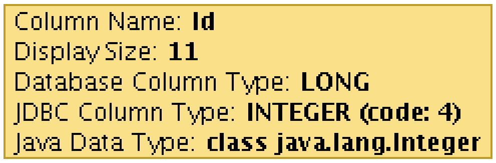
**Note:** The reason the documentation refer to **grid** rather then **table** is that table may be mixed with a database table.

### Context Sensitive Components

All components in the user interface are context sensitive i.e. buttons, menu items, etc.These are enabled only if they can be used in the current scope.

### Tool tips

Tooltips are used to explain a component. They are also used to express status information about components. An example is the grid column header tooltip that shows information about the column.

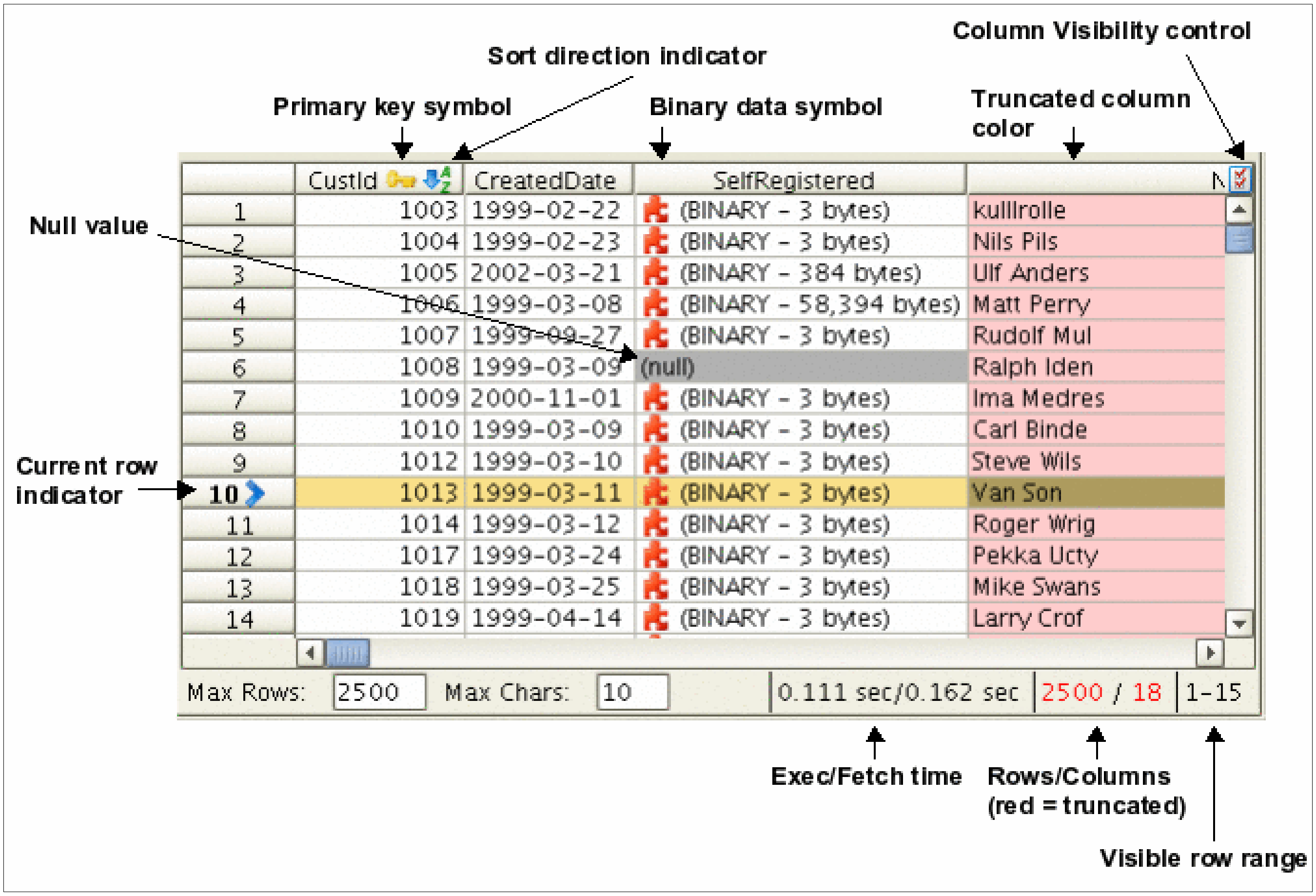


Column Name: **Id**
Display Size: **11**
Database Column Type: **LONG**
JDBC Column Type: **INTEGER (code: 4)**
Java Data Type: **class java.lang.Integer**

**Figure: Tooltip example**

## Grids

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



| | CustId | CreatedDate | SelfRegistered | N |
|---|---|---|---|---|
| 1 | 1003 | 1999-02-22 | (BINARY - 3 bytes) | kulllrolle |
| 2 | 1004 | 1999-02-23 | (BINARY - 3 bytes) | Nils Pils |
| 3 | 1005 | 2002-03-21 | (BINARY - 384 bytes) | Ulf Anders |
| 4 | 1006 | 1999-03-08 | (BINARY - 58,394 bytes) | Matt Perry |
| 5 | 1007 | 1999-09-27 | (BINARY - 3 bytes) | Rudolf Mul |
| 6 | 1008 | 1999-03-09 | (null) | Ralph Iden |
| 7 | 1009 | 2000-11-01 | (BINARY - 3 bytes) | Ima Medres |
| 8 | 1010 | 1999-03-09 | (BINARY - 3 bytes) | Carl Binde |
| 9 | 1012 | 1999-03-10 | (BINARY - 3 bytes) | Steve Wils |
| 10 | 1013 | 1999-03-11 | (BINARY - 3 bytes) | Van Son |
| 11 | 1014 | 1999-03-12 | (BINARY - 3 bytes) | Roger Wrig |
| 12 | 1017 | 1999-03-24 | (BINARY - 3 bytes) | Pekka Ucty |
| 13 | 1018 | 1999-03-25 | (BINARY - 3 bytes) | Mike Swans |
| 14 | 1019 | 1999-04-14 | (BINARY - 3 bytes) | Larry Crof |

Max Rows: 2500 Max Chars: 10 | 0.111 sec/0.162 sec | 2500 / 18 | 1-15

**Figure: Grid overview**

The screenshot shows the grid and controls that are available in the **Database Objects->Data tab** but the differences are minor compared to the standard grid.

### Right click menu

The generic right click menu contains the following operations:

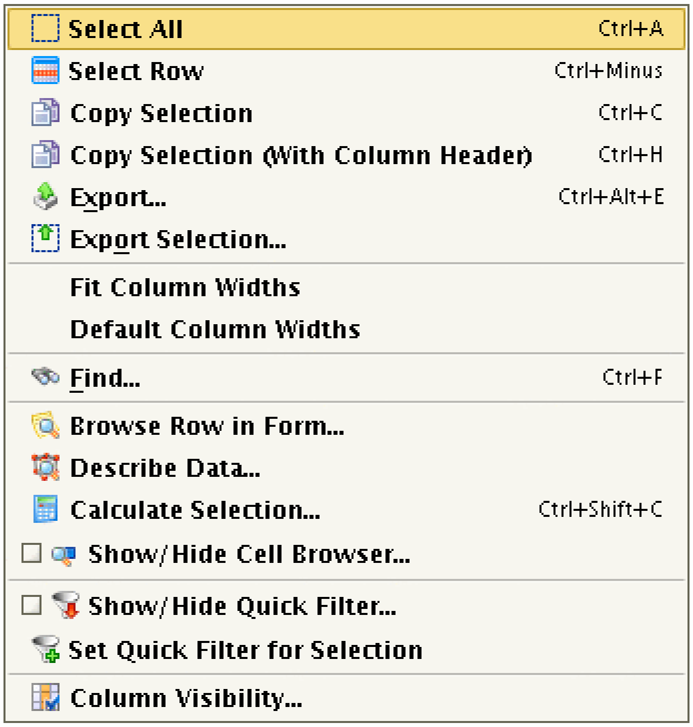


**Figure: Grid right click menu**

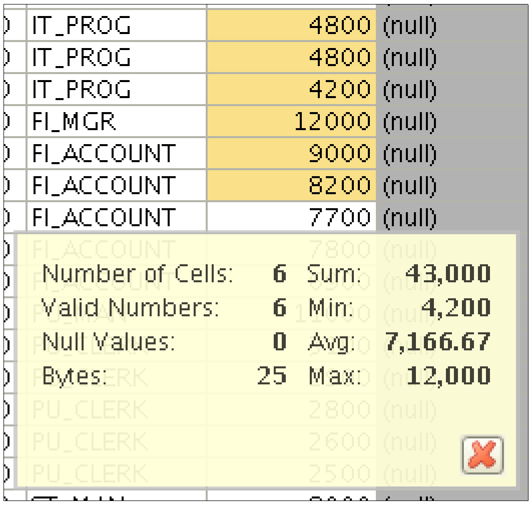
| Menu Choice | Description |
|---|---|
| **Select All** | Selects all cells (aka rows and columns) in the grid |
| **Select Row** | Selects all cells in the row |
| **Copy Selection** | Copy all selected cells onto the system clipboard |
| **Copy Selection (With Column Header)** | Copy all selected cells including column header onto the system clipboard |
| **Export** | Launch the export dialog |
| **Export Selection** | Export the selection using the standard export feature |
| **Fit Column Widths** | Automatically fit all column widths according to the widest cell value |
| **Default Column Widths** | Set the column width equally for all columns |
| **Find** | Launch the find dialog |
| **Browse Row in Form** | Displays all data for the selected row in a form. **Note:** this is just a read only form as editing is not |

| | |
|---|---|
| | allowed. |
| **Describe Data** | Show detailed information about the columns in the grid |
| **Calculate Selection** | Displays some metrics about the current selection. This is especially useful for numeric fields. Read more in Calculate Selection below. |
| **Show/Hide Cell Browser** | Displays or hides the cell browser. This browser shows the selected cell value below the grid. Useful when browsing complex data or images |
| **Show/Hide Quick Filter** | Displays or hides the quick filter pane. Read more about Quick Filters in the Database Objects Explorer document. |
| **Set Quick Filter for Selection** | Sets the selected value as the current quick filter |
| **Column Visibility** | Displays the column visibility menu. Use this to control what columns should be displayed in the grid. Read more in Column Visibility below. |

(The menu may contain additional entries based on the current scope).

### Calculate Selection

The **Calculate Selection** feature is used to perform some calculations on the current selection. It is primarily used to calculate on selections keeping numbers. The following is an example of what it shows.



```
IT_PROG       4800 (null)
IT_PROG       4800 (null)
IT_PROG       4200 (null)
FI_MGR       12000 (null)
FI_ACCOUNT    9000 (null)
FI_ACCOUNT    8200 (null)
FI_ACCOUNT    7700 (null)

 Number of Cells:   6  Sum:    43,000
 Valid Numbers:     6  Min:     4,200
 Null Values:       0  Avg:  7,166.67
 Bytes:            25  Max:    12,000
```

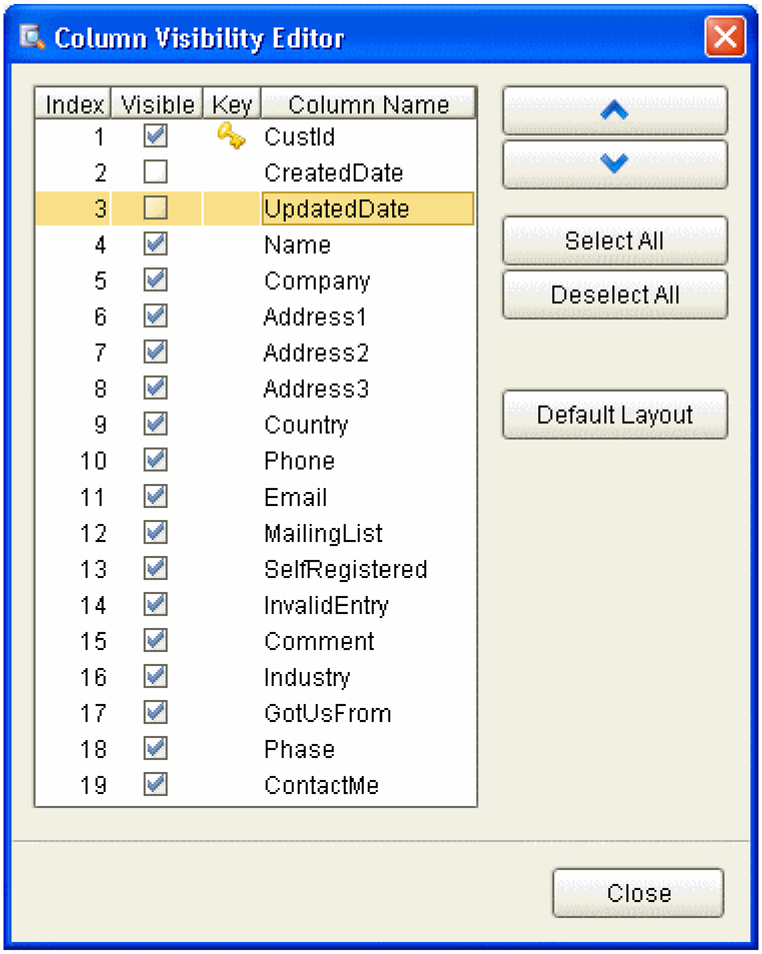
**Figure: The calculate selection popup**

| Property | Description |
|---|---|
| **Number of Cells** | shows the number of cells in the selection. |
| **Valid Numbers** | lists the number of valid numbers in the selection. |
| **Null Values** | shows the total of null values in the selection. |
| **Bytes** | shows the total number of bytes in the selection after that the data has been translated to text |
| **Sum** | shows the total summary of the selection |
| **Min** | shows the minimum number in the selection |
| **Avg** | shows the average value of the selection by doing sum / number of valid numbers |
| **Max** | shows the maximum number in the selection |

Either click the red cross icon or anywhere in the popup to close it.

#### Column Visibility

The **Column Visibility** feature is used to control what columns should appear in a grid. The column visibility dialog is displayed either by choosing the **Column Visibility** right menu choice in the grid or by clicking the button above the vertical scrollbar in the grid.



| Index | Visible | Key | Column Name |
|---|---|---|---|
| 1 | ☑ | 🔑 | CustId |
| 2 | □ | | CreatedDate |
| 3 | □ | | UpdatedDate |
| 4 | ☑ | | Name |
| 5 | ☑ | | Company |
| 6 | ☑ | | Address1 |
| 7 | ☑ | | Address2 |
| 8 | ☑ | | Address3 |
| 9 | ☑ | | Country |
| 10 | ☑ | | Phone |
| 11 | ☑ | | Email |
| 12 | ☑ | | MailingList |
| 13 | ☑ | | SelfRegistered |
| 14 | ☑ | | InvalidEntry |
| 15 | ☑ | | Comment |
| 16 | ☑ | | Industry |
| 17 | ☑ | | GotUsFrom |
| 18 | ☑ | | Phase |
| 19 | ☑ | | ContactMe |

**Figure: The column visibility dialog**

The column visibility dialog shows all columns that are available in the grid. The check mark in front of a column name indicates that the column is visible in the grid while an unchecked box indicates that it will be invisible. Columns can be made invisible either by selecting a checked column name in this list or by using the **Remove Column** menu choice in the grid column header menu. The order of the columns can also be adjusted in this dialog. Just select a row and then move it up (left in grid) or down (right in grid).

The **Default Layout** resets the grid by making all column visible and put them in their default locations.

**Note 1:** Modifying column visibility in conjunction with column resizing and column ordering is saved between invocations of DbVisualizer for all grids in the various **Object Views** **except** the Data tab.

**Note 2:** If modifying column visibility in the **Data** tab then these changes will persist throughout the session i.e if you for example remove the column **Name** in the Data tab for the table **EMPLOYEE** then will **Name** not appear if doing a reload or subsequent shows of the Data tab for that table. You must manually make it visible again or simply select **Default Layout** to bring it back. Another solution is to restart the application.

# Problem resolution

There are situations when problems, errors or even bugs occur. The runtime environment for DbVisualizer is rather complicated when it comes to tracking the source of a potential problem since it's not only DbVisualizer that may cause the problem but also the actual JDBC driver(s).

There are a few things that you can do before reporting problems based on at what stage the problem occurs:

1. Make sure you are using the latest version of Java 1.4
2. Make sure you are using at least the version of the JDBC driver that we've tested DbVisualizer with
3. Read the DbVisualizer FAQ.
4. Check the on-line Forums.
5. Read the DbVisualizer Users Guide.

If you cannot find a solution to resolve the problem then please do the following and email us the debug output:

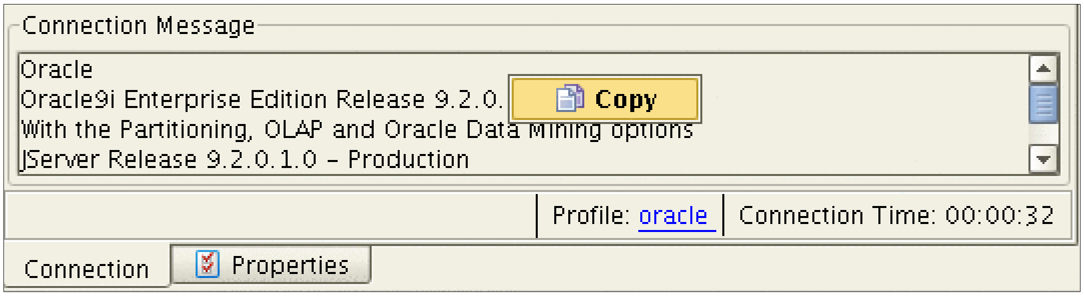
- Problem during installation or when starting DbVisualizer
  Debugging the installer
- Error or problem while using DbVisualizer
  Debugging DbVisualizer

Use the DbVisualizer problem report form or email support@dbvis.com. We appreciate detailed reports as well as screenshots when possible.

## How to satisfy the DbVisualizer support team

Quite often we get incomplete problem reports and need to follow-up for additional information. If an error or problem occur then you can do the following to let DbVisualizer create system details that you then paste into a support email or in the problem report form:

1. Select the **Connection** tab
2. In the **Connection Message** area select the right click menu
3. In the menu select **Copy**
4. This will copy system details to the clipboard. Then paste the details into an email or in the problem report form mentioned above.
5. A bonus is if you provide screen shots! An image says more then ... you know.



```
Oracle
Oracle9i Enterprise Edition Release 9.2.0.
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

**Figure: The connection message right click menu**

# Load JDBC Driver and Get Connected

## Introduction

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

**Fast track:** If you are impatient then please go ahead and read the Connection Wizard section. It is the recommended feature in DbVisualizer to create database connections.

## What is a JDBC Driver?

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

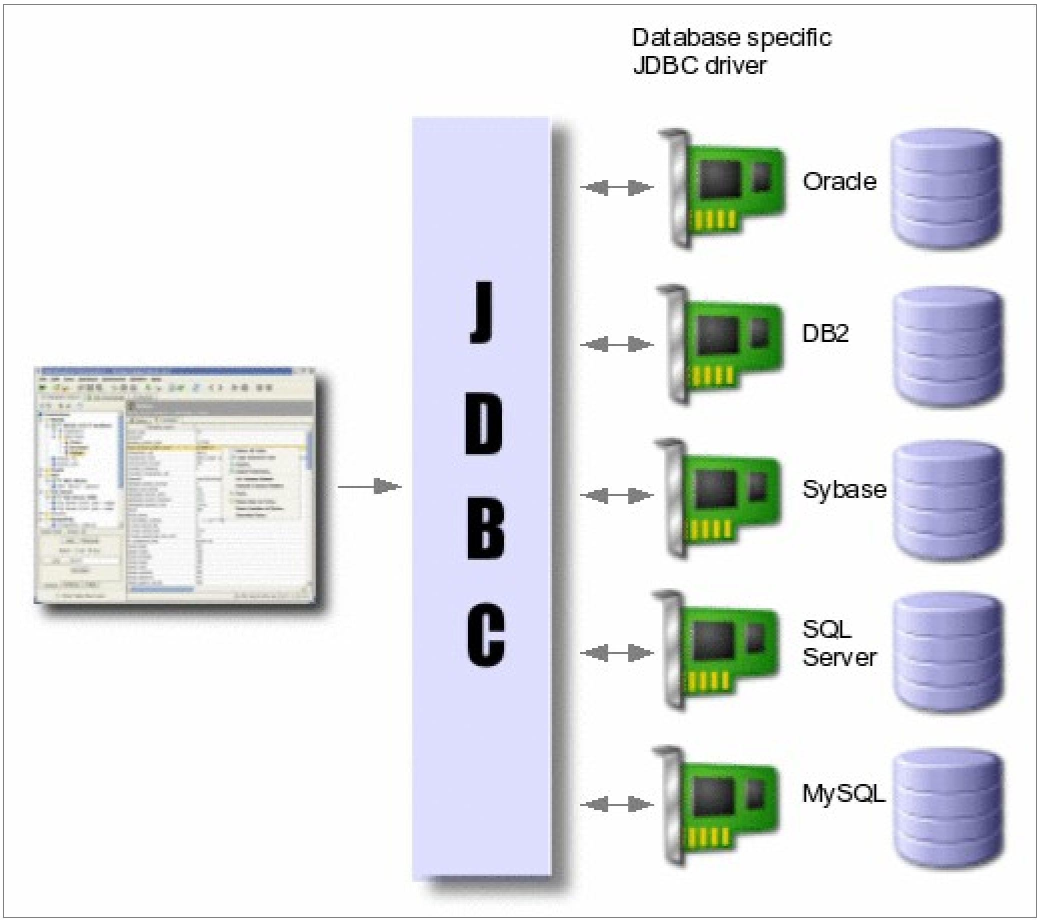


**Figure: The runtime environment with the JDBC interface, JDBC driver and sample databases**

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

A JAR, ZIP or directory that is loaded into the driver manager consists of a number of Java classes that forms the complete implementation of the JDBC driver. DbVisualizer automatically recognize the classes that are used to initiate the connection with the database and presents them in the **Driver Class** list. You must select the correct class in this list to make sure DbVisualizer successfully can initiate the connection. Consult the driver documentation for information of which class to select or if the number of found classes are low, figure out by trying each of them.

# Get the JDBC driver file(s)

DbVisualizer do not include any JDBC driver so first you must grab a JDBC driver file(s) that works with the actual database and the version of it. The following online web page lists an up to date listing of the tested combinations:

## Databases and JDBC Drivers

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

JDBC Data Access API - 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 in order to make the driver files visible to the Driver Manager.
The Databases and JDBC Drivers web page lists from where to download each driver and also what steps is needed to eventually unpack, install and load the driver in DbVisualizer.

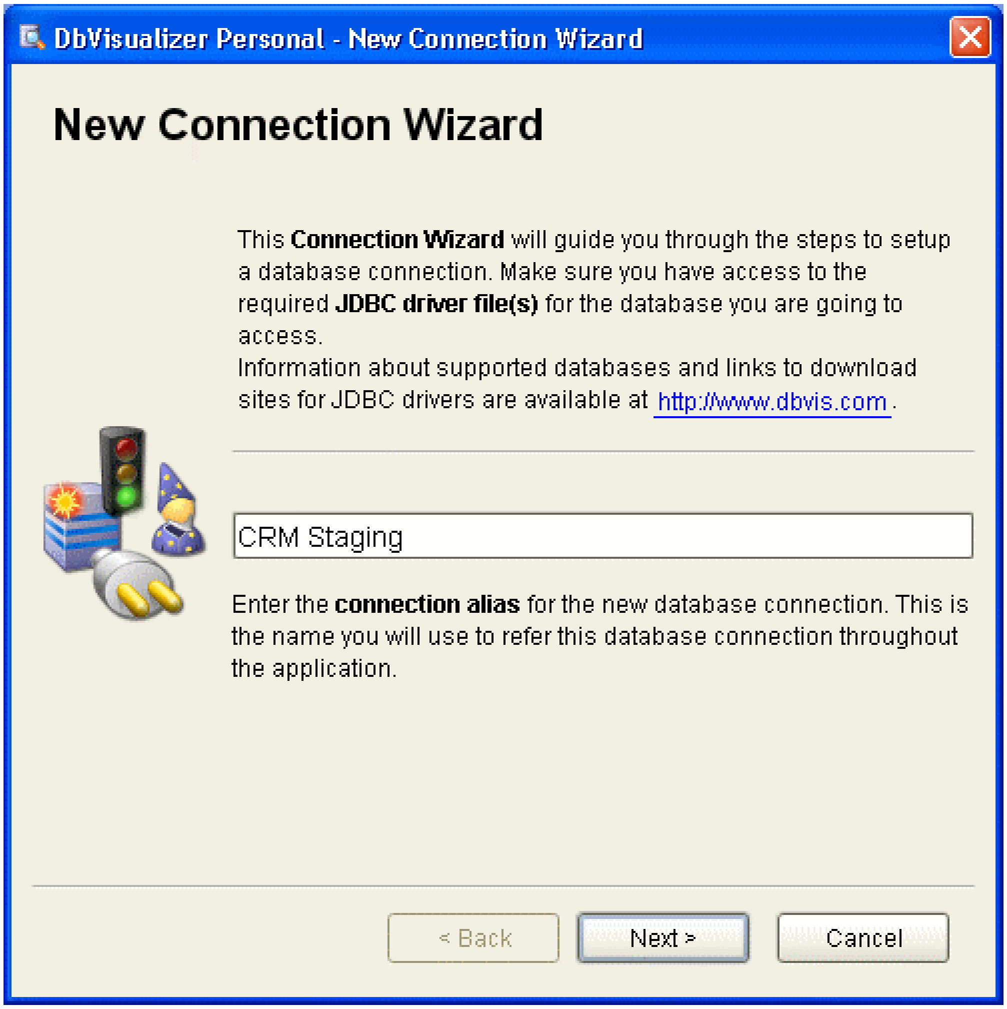
(Drivers are categorized into 4 types. We're not going to explain the differences here but just give a hint that the "type 4" aka "thin" drivers are easiest to maintain since they are pure Java drivers and do not depend on any external DLL's or dynamic libraries i.e try to get a type 4 driver even though DbVisualizer works with any type of driver).

# Connection Wizard

The Connection Wizard greatly simplifies the steps needed to load the JDBC driver and create a new database connection. It is based on a few wizard pages in which information about the driver file(s) and connection data should provided. Once the new database connection has been created it will appear in the database objects tree.

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

The first wizard screen look like this.



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 .

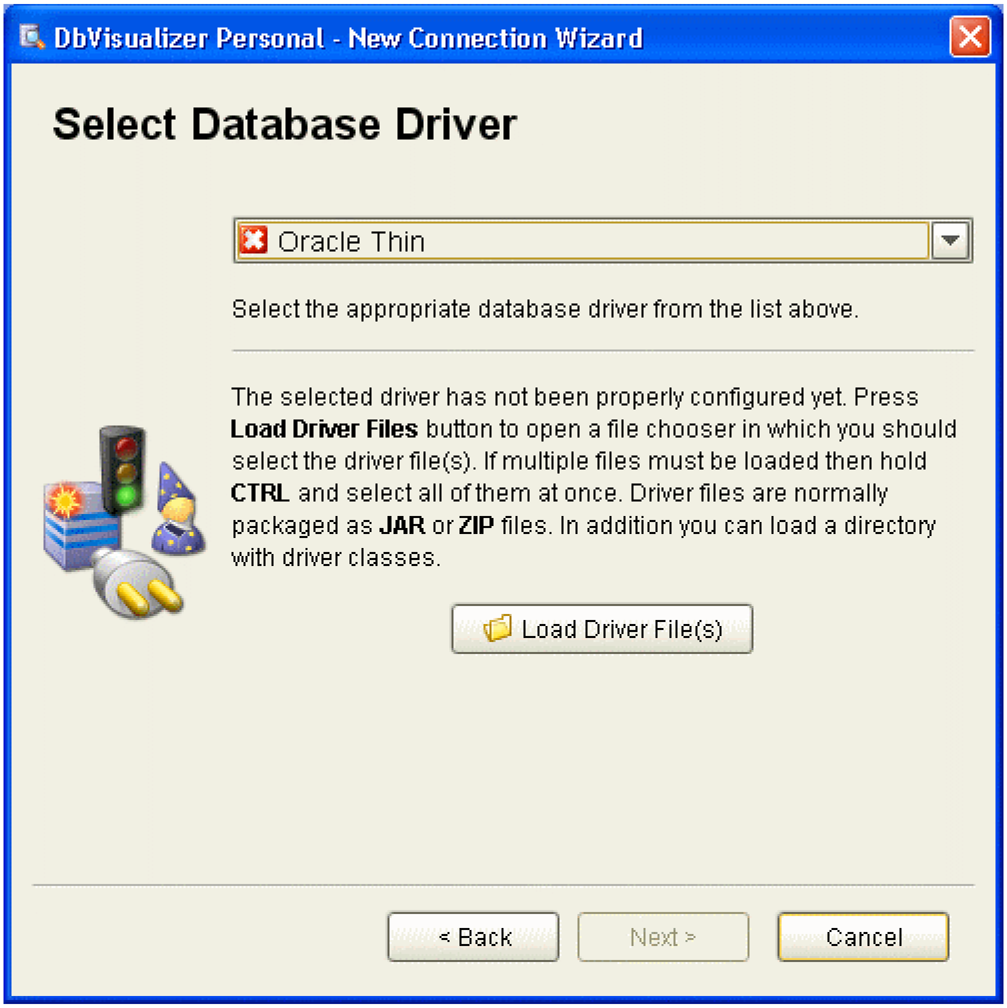
CRM Staging

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

**Figure: Connection Wizard - Page 1**

In the **connection alias** field enter the name of the new database connection. This is the name that will be used in DbVisualizer.
Press **Next** to connect to the next page.

In this page select the driver from the list that you are going to use. The red icon indicates that the driver is not ready yet while a green icon indicates that it has been properly configured (simply press Next to continue). If the driver you select is not yet configured the following will be displayed. Press the **Load Driver File(s)** button to open a file chooser in which you should select the JAR or ZIP file(s) that contain the driver implementation.



Select Database Driver

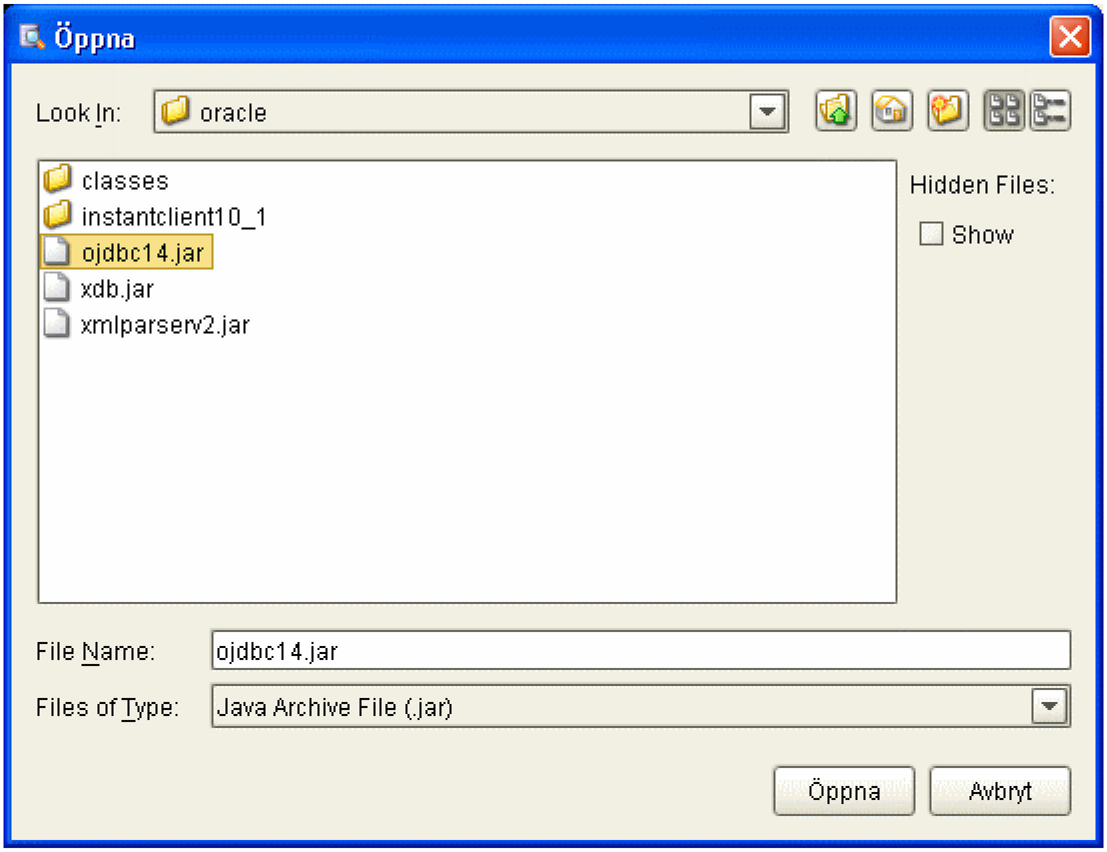
Oracle Thin

Select the appropriate database driver from the list above.

The selected driver has not been properly configured yet. Press **Load Driver Files** button to open a file chooser in which you should select the driver file(s). If multiple files must be loaded then hold **CTRL** and select all of them at once. Driver files are normally packaged as **JAR** or **ZIP** files. In addition you can load a directory with driver classes.

**Figure: Connection Wizard - Page 2**

In the file chooser locate the files needed to load the JDBC driver. (Select multiple files by pressing the SHIFT key).



Look In: oracle

classes
instantclient10_1
ojdbc14.jar
xdb.jar
xmlparserv2.jar

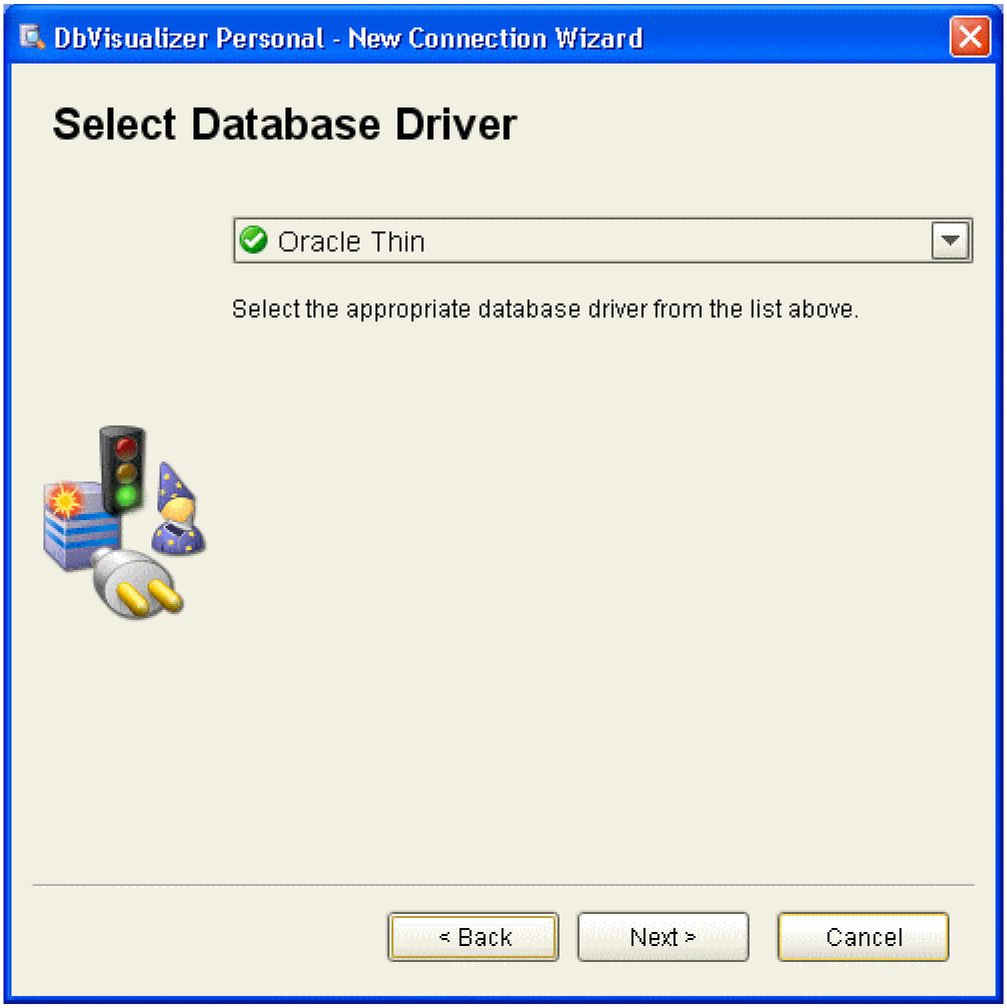
Hidden Files:
□ Show

File Name: ojdbc14.jar

Files of Type: Java Archive File (.jar)

**Figure: Connection Wizard - Page 3**

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



DbVisualizer Personal - New Connection Wizard

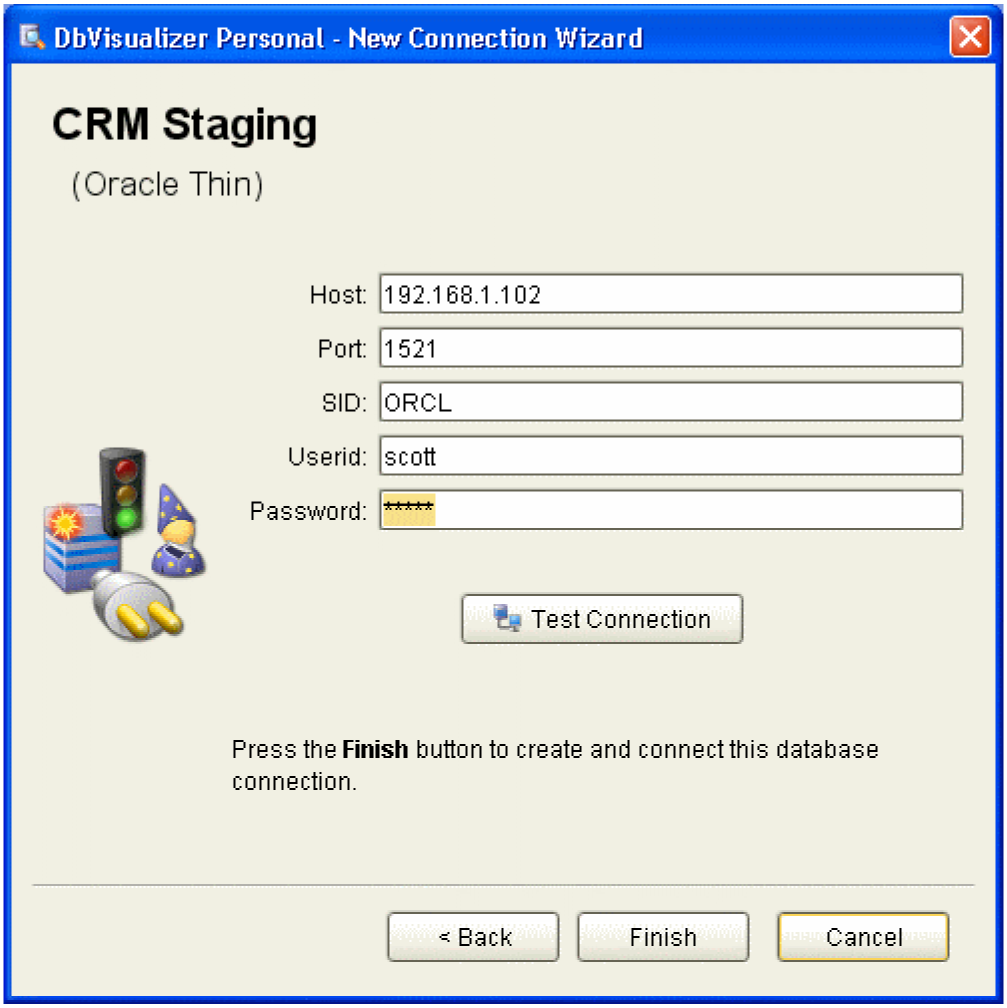
Select Database Driver

Oracle Thin

Select the appropriate database driver from the list above.

**Figure: Connection Wizard - Page 4**

In the last wizard pane enter details for the new database connection.



CRM Staging

(Oracle Thin)

Host: 192.168.1.102

Port: 1521

SID: ORCL

Userid: scott

Password: *****

Press the **Finish** button to create and connect this database connection.

**Figure: Connection Wizard - Page 5**

Press **Test Connection** to check if the connection can be established. Press **Finish** to create the new database connection and connect it.

It is recommended that you skip reading the rest of this document if you **don't**:

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

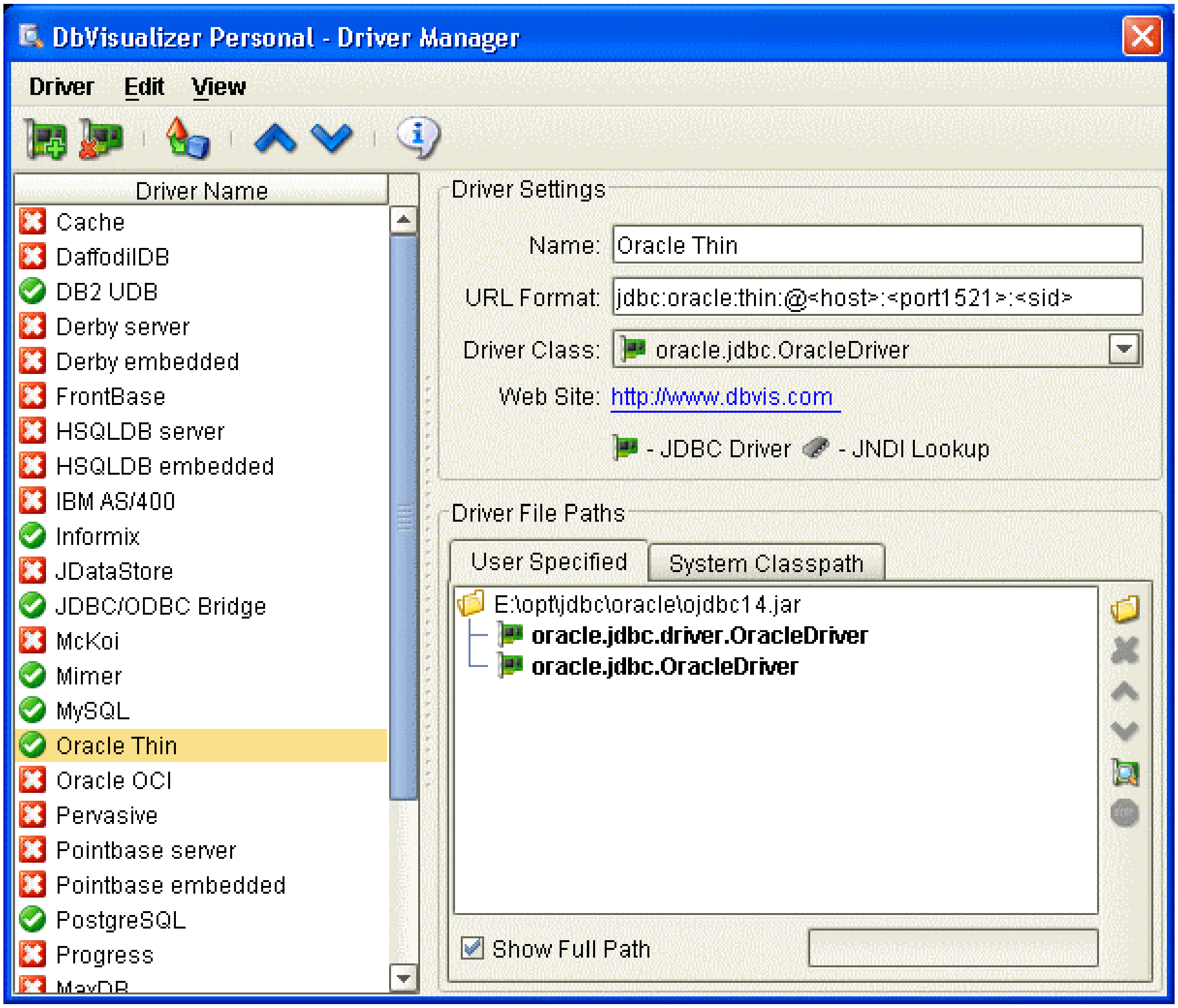
# Driver Manager

The **Driver Manager** in DbVisualizer is used to define the drivers that will be used to communicate with the actual databases. Start the driver manager dialog using the **Tools->Driver Manager** menu choice.

The left part of the driver manager dialog lists a collection of driver names and a symbol indicating whether the driver has been configured or not. The right part displays the definition of 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 name is later listed in the **Connection** tab setup when selecting what driver to use for a **database connection**
- **URL Format**
  The URL format specifies the pattern for the JDBC URL or a JNDI Lookup name. The purpose is to assist the user in the connection tab while entering the URL or lookup name
- **Default Class**
  Defines the default class to use when connecting
- **Web Site**
  Link to the DbVisualizer web site containing up to date information how to download the driver.
- **Driver File Paths**
  Defines all paths to search for JDBC drivers or Initial Contexts during connect with the database. The Driver File Paths is composed of two tabs, the **User Specified** tab is used to locate and identify dynamically loaded JDBC drivers or Initial Context classes. The **System Classpath** tab lists all paths that are part of the Java system classpath.

**Note:** Do not bother about the System Classpath tab unless you are using the JDBC-ODBC driver.



Driver Name:
- ✗ Cache
- ✗ DaffodilDB
- ✓ DB2 UDB
- ✗ Derby server
- ✗ Derby embedded
- ✗ FrontBase
- ✗ HSQLDB server
- ✗ HSQLDB embedded
- ✗ IBM AS/400
- ✓ Informix
- ✗ JDataStore
- ✓ JDBC/ODBC Bridge
- ✗ McKoi
- ✓ Mimer
- ✓ MySQL
- ✓ Oracle Thin
- ✗ Oracle OCI
- ✗ Pervasive
- ✗ Pointbase server
- ✗ Pointbase embedded
- ✓ PostgreSQL
- ✗ Progress
- ✗ MaxDB

Name: Oracle Thin
URL Format: jdbc:oracle:thin:@<host>:<port1521>:<sid>
Driver Class: oracle.jdbc.OracleDriver
Web Site: http://www.dbvis.com
- JDBC Driver - JNDI Lookup

E:\opt\jdbc\oracle\ojdbc14.jar
- oracle.jdbc.driver.OracleDriver
- oracle.jdbc.OracleDriver

**Figure: Driver Manager dialog**

The driver list contains initially a collection of default drivers. These are not fully configured as the actual paths used to search for the classes need to be identified. The list can be edited as drivers can be created, copied, removed and renamed. A driver is ready once a default class has been identified and this state is indicated with a green check icon in the list. Not ready drivers are indicated with a red cross icon.

**Note:** Only ready (configured) drivers will appear in the Connection tab driver list.

The figure shows four drivers that are ready, **MySQL RefFS, Oracle Thin**, **SQL Server** and **Sybase ASE**.

### Setup a JDBC driver

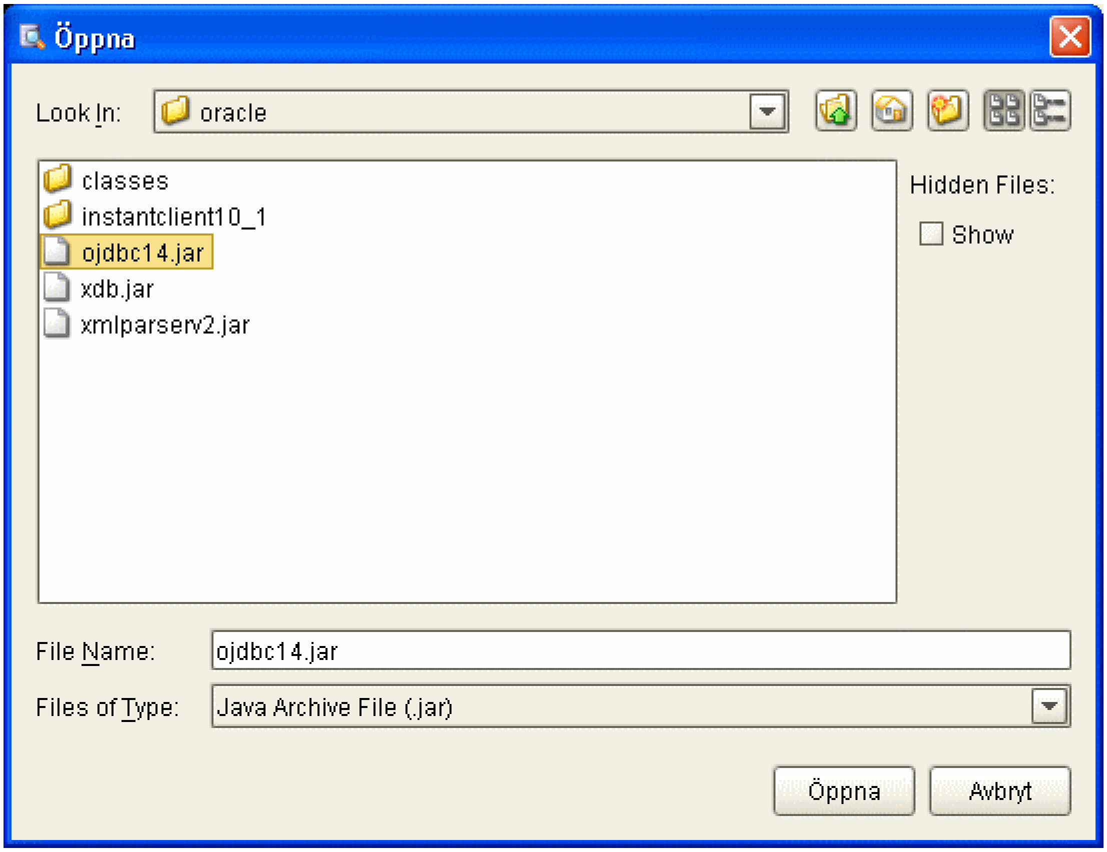
The recommended way to setup a driver is to pick a matching driver name from the list and then simply load the JAR, ZIP or directory that keeps the actual driver class(es) i.e if you are going to load the JDBC driver for **Oracle** then select the Oracle driver in the list. You can also create a new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

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

- It lists what databases and drivers that has been tested
- Download links to JDBC drivers
- Information of what files to load in the driver manager for each JDBC driver
- Information of what **Driver Class** that should be chosen

## Databases and JDBC Drivers

To load jar file(s) then press the **Load** button to the right of the **User Specified** paths tree to show the file chooser and load the driver jar(s).



Look In: oracle

File Name: ojdbc14.jar

Files of Type: Java Archive File (.jar)

**Figure: File Chooser dialog**

It is important to load the root of the JDBC Driver i.e a JDBC Driver implementation consists in most cases of several Java classes. These are also in most cases organized using the package mechanism in Java. Example:

```
oracle.jdbc.driver.OracleDriver
```

Each package part in the name above (separated by ".") will be represented by a directory in the file system. These directories are either explicitly visible in the file system or implicitly if the driver is packaged in a ZIP or JAR file. The root of the driver is in this case where the **oracle** directory is located. In the Oracle example this is the **ojdbc14.jar** JAR file so the driver manager must load this path in order to find the driver class. If the driver is packaged in a ZIP file or a directory then point the driver manager to that path in order for the driver manager to locate the driver class.

Once a connection is established in the Connection tab will DbVisualizer search the selected drivers path tree's in the following order:

1. User Specified
2. System Classpath

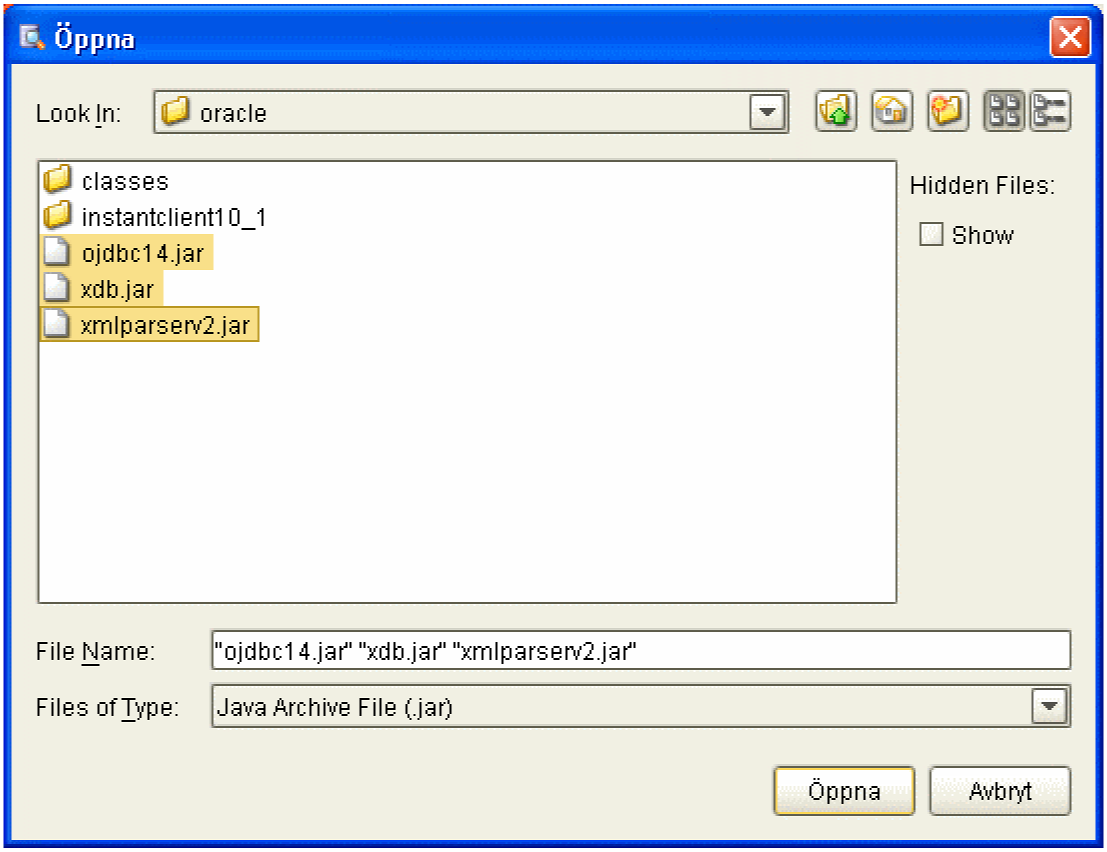
These are searched from the top of the tree i.e if there are several identical classes in for example the dynamic tree then the topmost class will be used. Loading several paths with different versions of the same driver in one driver definition is not recommended even though it works (if you do this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

Once one or several classes has been identified and listed in the **Driver Class** list then make sure you select the correct **Driver Class** from the list. See the table earlier for assistance.

### JDBC drivers that requires several JAR or ZIP files

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



File Name: "ojdbc14.jar" "xdb.jar" "xmlparserv2.jar"

Files of Type: Java Archive File (.jar)

**Figure: File Chooser dialog**

### The JDBC-ODBC bridge

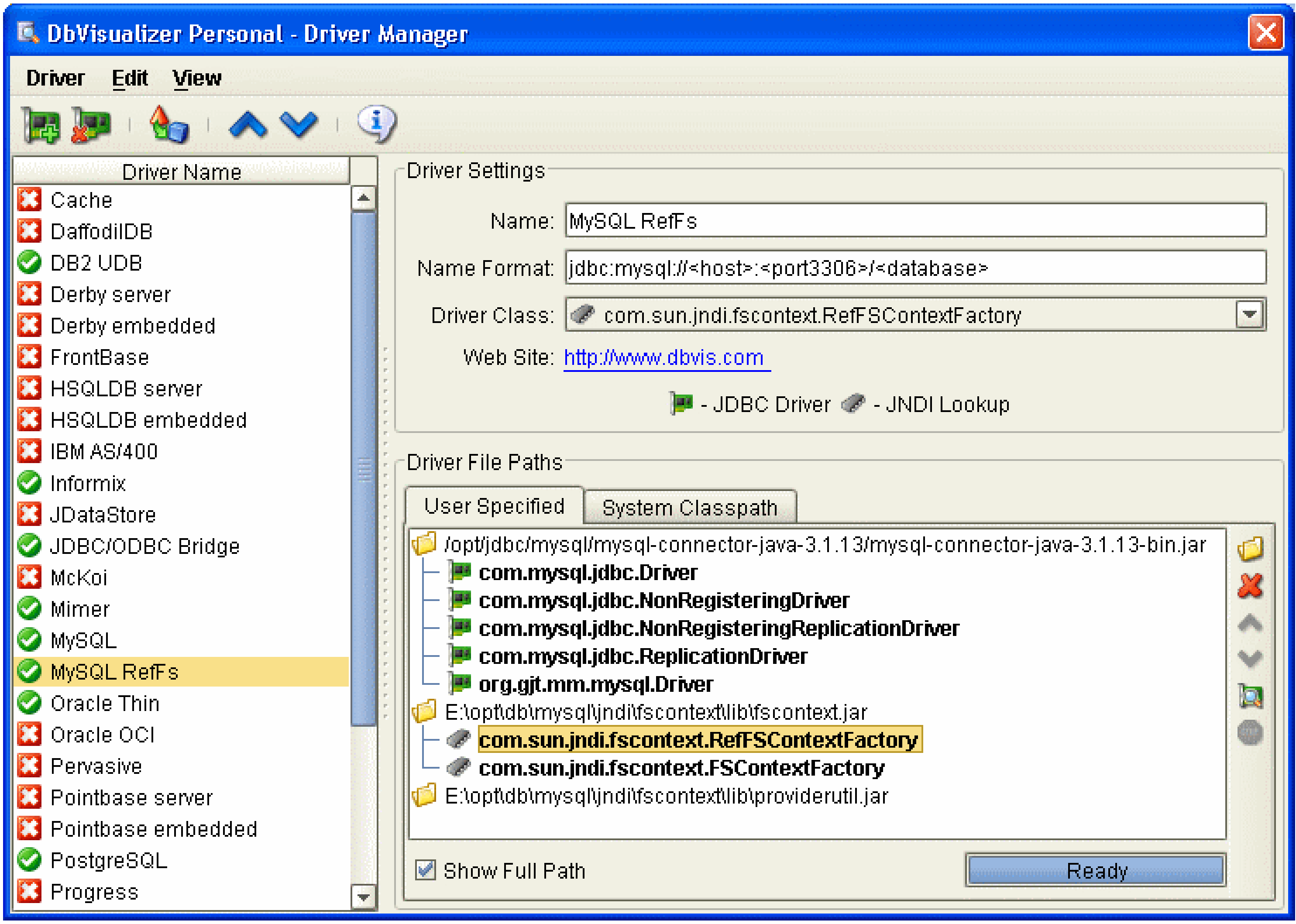
The JDBC-ODBC driver is by default part of most Java installations. The **JdbcOdbcDriver** class is included in a JAR file that is commonly named **rt.jar** and is stored somewhere in the Java directory structure. DbVisualizer automatically identifies this JAR file in the System Classpath tree. To locate the JdbcOdbcDriver simply press the **Find Drivers** button to the right of the System Classpath tree. Once found then make sure the **sun.jdbc.odbc.JdbcOdbcDriver** is selected as the **Default Class**.

## Loading JNDI Initial Contexts

Initial Context classes are needed in order to get a handle to a database connection that is registered in a JNDI lookup service. These classes are similar to JDBC driver classes since an Initial Context implementation is required.

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

To load Initial Context classes into the Driver Manager simply follow the steps outlined for loading JDBC drivers. The difference is that you will instead load locations that contain Initial Context classes instead of JDBC drivers. Once Initial Context classes have been found the following will appear in the Driver Manager list.



Driver Name:

- ☒ Cache
- ☒ DaffodilDB
- ✓ DB2 UDB
- ☒ Derby server
- ☒ Derby embedded
- ☒ FrontBase
- ☒ HSQLDB server
- ☒ HSQLDB embedded
- ☒ IBM AS/400
- ✓ Informix
- ☒ JDataStore
- ✓ JDBC/ODBC Bridge
- ☒ McKoi
- ✓ Mimer
- ✓ MySQL
- ✓ MySQL RefFs
- ✓ Oracle Thin
- ☒ Oracle OCI
- ☒ Pervasive
- ☒ Pointbase server
- ☒ Pointbase embedded
- ✓ PostgreSQL
- ☒ Progress

Name: MySQL RefFs

Name Format: jdbc:mysql://<host>:<port3306>/<database>

Driver Class: com.sun.jndi.fscontext.RefFSContextFactory

Web Site: http://www.dbvis.com

- JDBC Driver - JNDI Lookup

```
/opt/jdbc/mysql/mysql-connector-java-3.1.13/mysql-connector-java-3.1.13-bin.jar
  ├ com.mysql.jdbc.Driver
  ├ com.mysql.jdbc.NonRegisteringDriver
  ├ com.mysql.jdbc.NonRegisteringReplicationDriver
  ├ com.mysql.jdbc.ReplicationDriver
  └ org.gjt.mm.mysql.Driver
E:\opt\db\mysql\jndi\fscontext\lib\fscontext.jar
  ├ com.sun.jndi.fscontext.RefFSContextFactory
  └ com.sun.jndi.fscontext.FSContextFactory
E:\opt\db\mysql\jndi\fscontext\lib\providerutil.jar
```

☑ Show Full Path

**Figure: Driver Manager List with Initial Context classes**

The visual difference between the identified JDBC drivers and Initial Context classes is the icon in the tree.

The figure shows the required JAR files in order to first obtain the JNDI handle and then also the actual JDBC driver that is needed to interface the database. Check with the application server vendor or similar for more information of what files that need to be loaded to get connected via JNDI.

### Errors (why are some paths red?)

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

### Several versions of the same driver

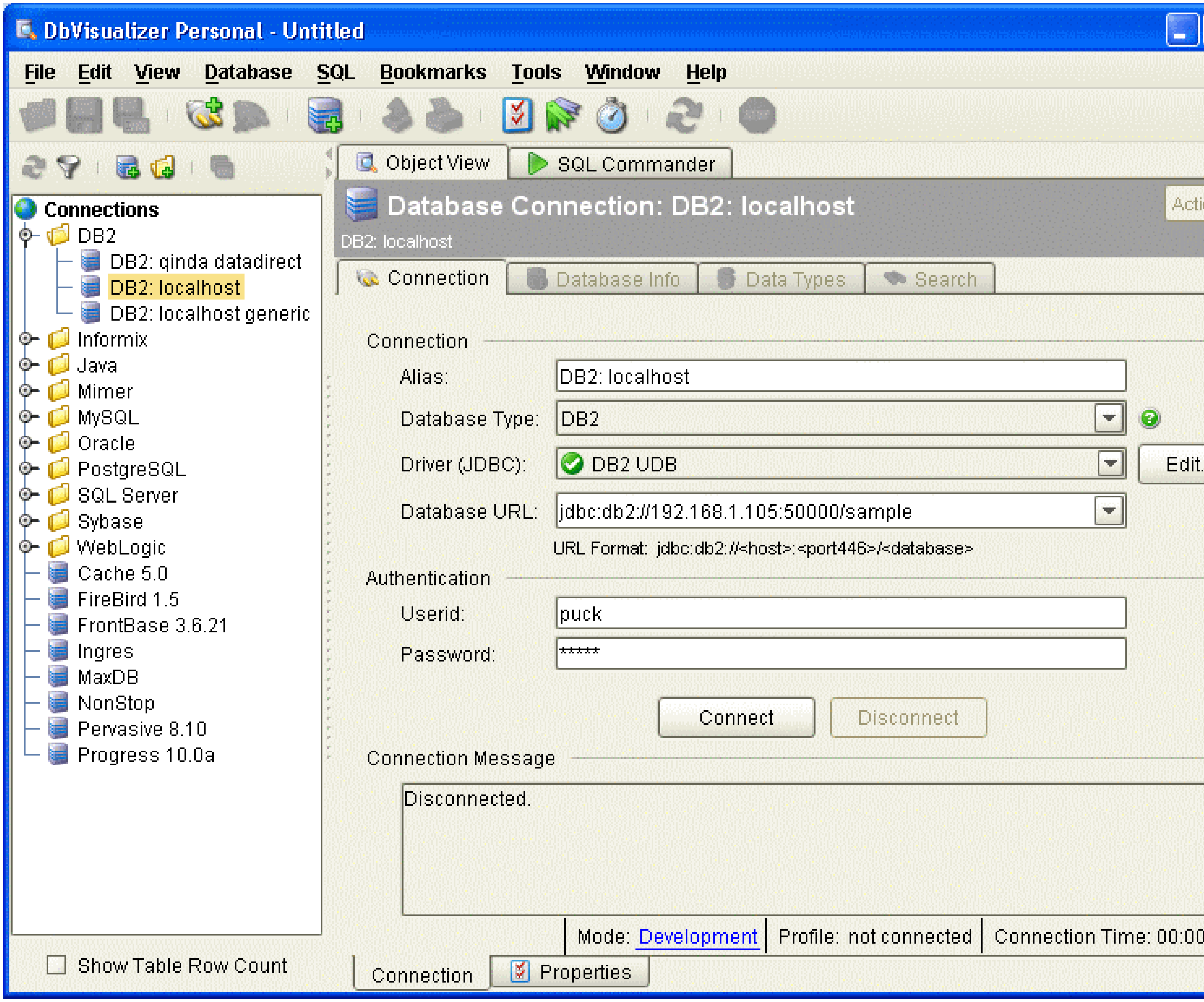
The Driver Manager supports loading and then using several versions of the same driver concurrently. The recommendation is to create a unique driver definition per version of the driver and then name the drivers properly. Ex. **Oracle 9.2.0.1**, **Oracle 10.2.1.0.1**, etc.

## Setup a database connection

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

### Setup using JDBC driver

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



Connection

Alias: DB2: localhost

Database Type: DB2

Driver (JDBC): DB2 UDB

Database URL: jdbc:db2://192.168.1.105:50000/sample

URL Format: jdbc:db2://<host>:<port446>/<database>

Authentication

Userid: puck

Password: *****

Connection Message

Disconnected.

Mode: Development | Profile: not connected | Connection Time: 00:00

**Figure: New Database Connection using JDBC driver**

The **Connection** tab is the only enabled tab if you are not already connected to the database. Database connection objects appear throughout the application and are by default listed by their **URL**. A URL can be, and often is, quite complex and long. The **Database Alias** is used to optionally set a more readable name of the database connection. The **Driver** list when opened shows all defined drivers that have been defined properly in the Driver Manager. Just open the list and select the appropriate driver. The URL Format lists the format  that the driver supports.

**Tip:** Put the mouse pointer on the URL Format and click with the mouse to copy the format template into the URL field.

The **<** and **>** characters indicates that they are the boundary for a placeholder and that they shall be replaced with appropriate values.  Ex.

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

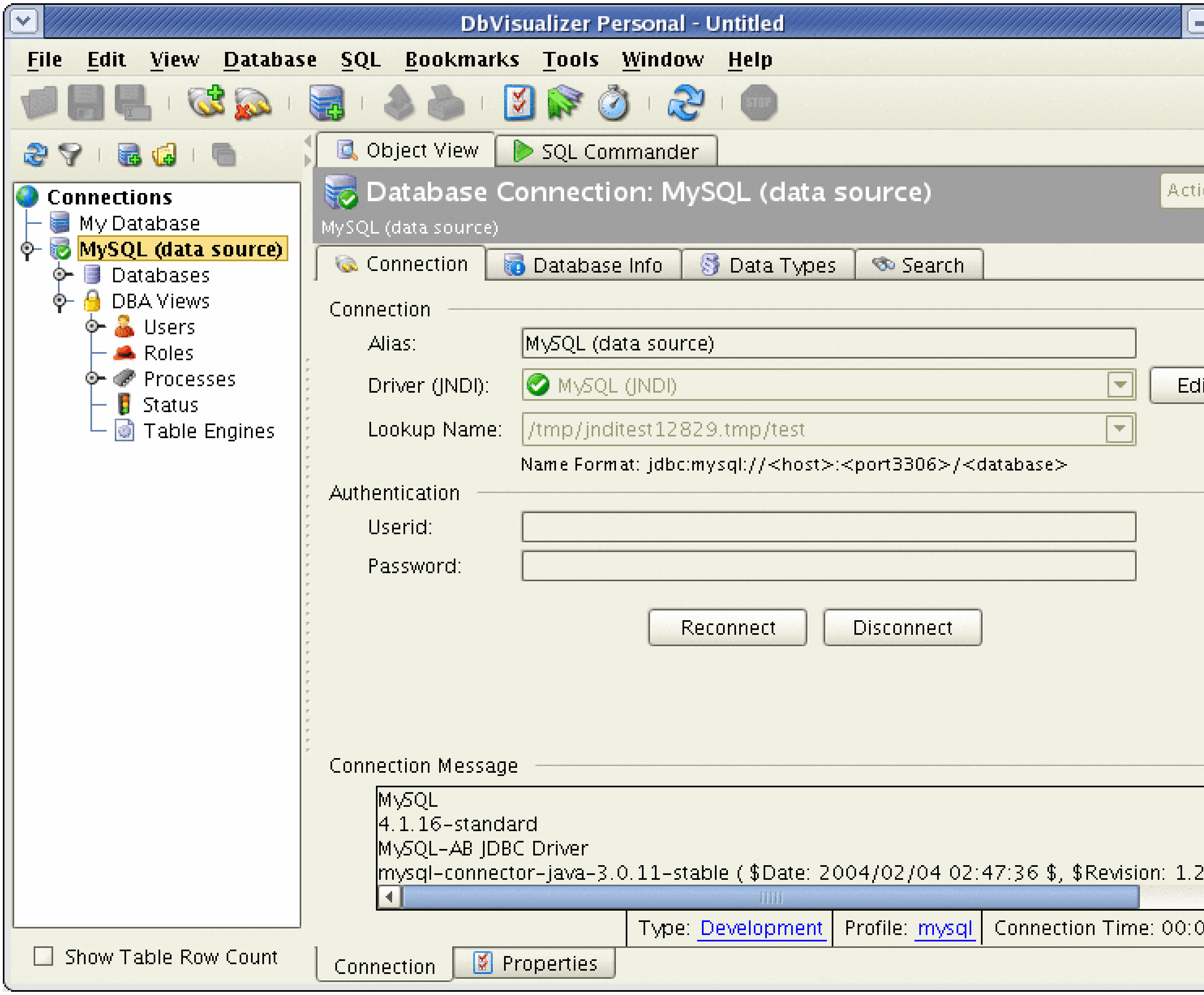
```
jdbc:db2://localhost/crm
jdbc:microsoft:sqlserver://localhost;DatabaseName=customers
```

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

Some drivers accept additional proprietary parameters described in the Connection Properties section.

## Setup using JNDI lookup

The information needed in order to obtain a database connection using JNDI lookup is similar to getting connected using a JDBC driver.



Alias: MySQL (data source)

Driver (JNDI): MySQL (JNDI)

Lookup Name: /tmp/jnditest12829.tmp/test

Name Format: jdbc:mysql://<host>:<port3306>/<database>

Userid:

Password:

```
MySQL
4.1.16-standard
MySQL-AB JDBC Driver
mysql-connector-java-3.0.11-stable ( $Date: 2004/02/04 02:47:36 $, $Revision: 1.2
```

Type: Development | Profile: mysql | Connection Time: 00:0

**Figure: New Database Connection using JNDI lookup**

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

## Connection Properties

In addition to the standard connection parameters (URL, Driver, Userid, Password, etc.) there are also a collection of connection properties. What properties are available depends on what database type is chosen. Some database types have more properties then others. What edition of DbVisualizer being used do also affect what connection properties are available.

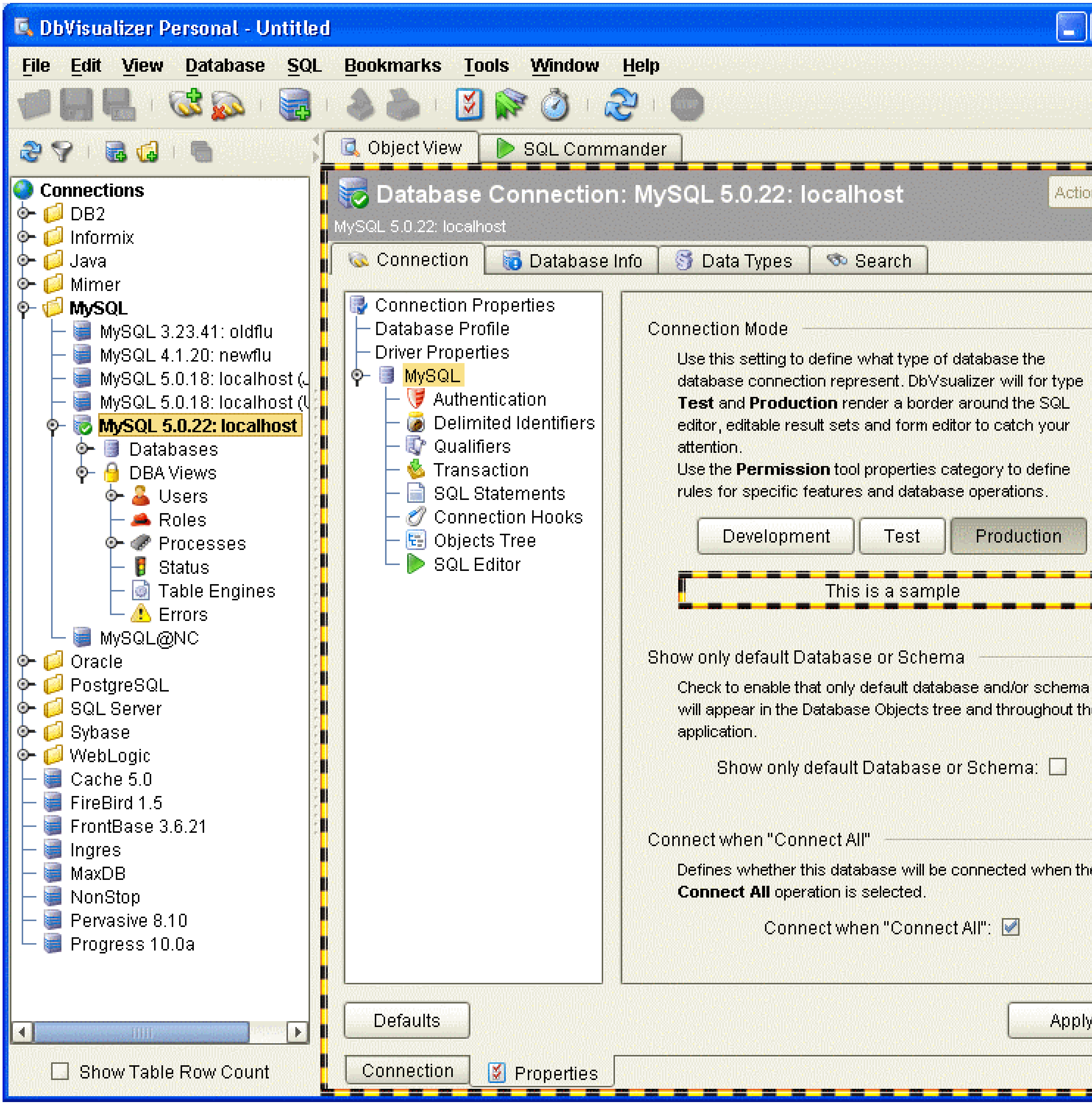
All supported database types (Oracle, Informix, Mimer, DB2, MySQL, etc.) are listed in the **Database** tab in the **Tool Properties** window. Each of the database types collects a number of properties that will be applied to any database connection of that type. Briefly it means that a database connection 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 a database connection. The advantage with this inheritance model is that property changes can be made for all database connections centrally instead of applying a common setting for every database connection of a specific database type.

The following summarize the organization of properties:

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

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

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



Connection Mode

Use this setting to define what type of database the database connection represent. DbVsualizer will for type **Test** and **Production** render a border around the SQL editor, editable result sets and form editor to catch your attention.
Use the **Permission** tool properties category to define rules for specific features and database operations.

Show only default Database or Schema

Check to enable that only default database and/or schema will appear in the Database Objects tree and throughout th application.

Show only default Database or Schema: □

Connect when "Connect All"

Defines whether this database will be connected when th **Connect All** operation is selected.

Connect when "Connect All": ☑

**Figure: Connection Properties**

The connection properties tab is organized in the same was as the tool properties window. The difference is that the list only includes the categories that are applicable for a database connection. The categories are briefly:

- Database Profile
- Driver Properties
- MySQL (The current Database Type)
    - Authentication
- Delimited Identifiers

- Qualifiers
- Transaction
- SQL Statements
- Connection Hooks
- Objects Tree
- SQL Editor

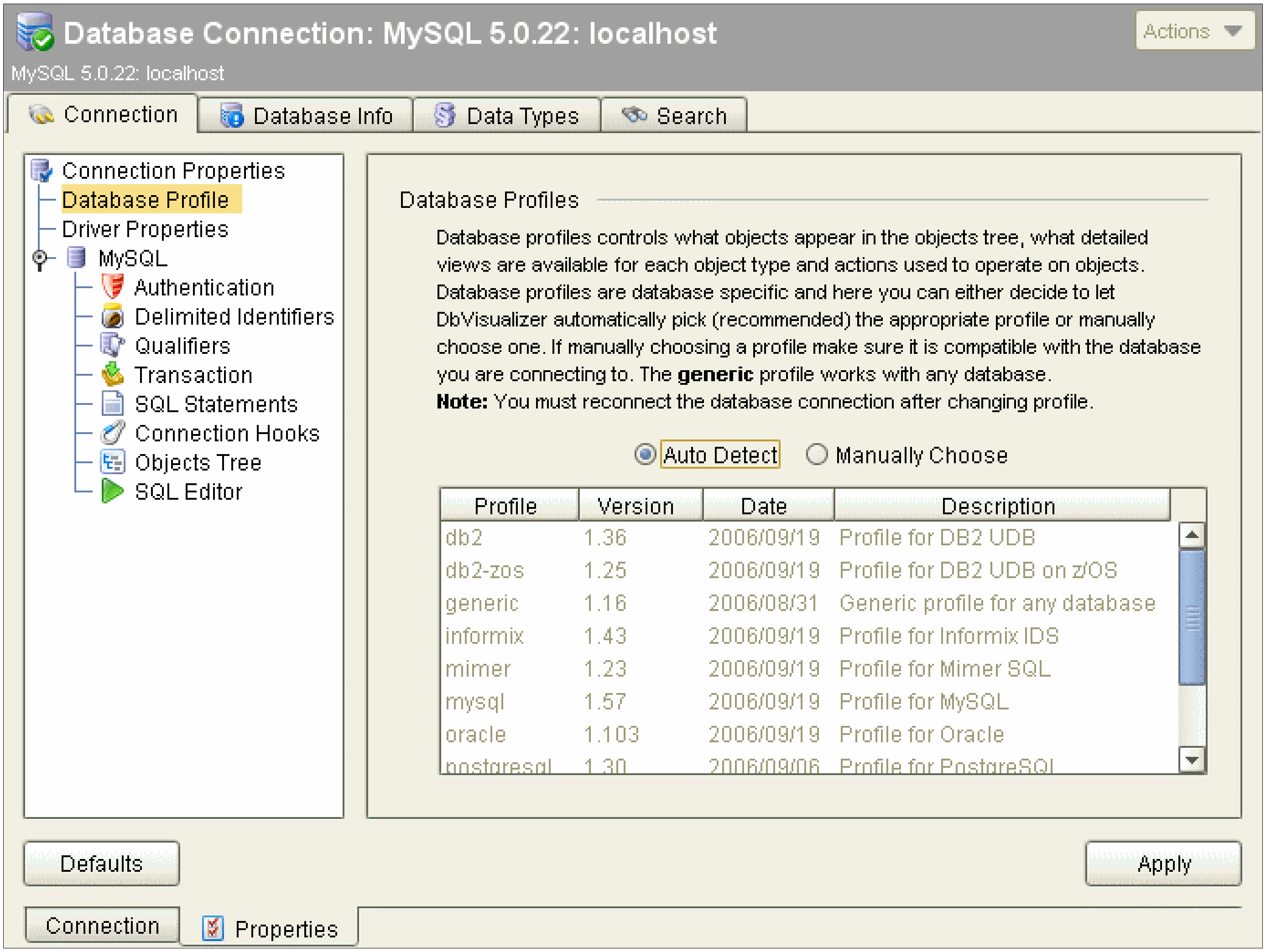
The **Database Profile** and **Driver Properties** categories are only available in the Connection Properties tab and not in Tool Properties. The next section explains the Database Profile and Driver Properties categories while the other categories are described in the Tool Properties document.

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

## Database Profile

Please read in the Database Objects Explorer document for detailed information about database profiles.

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



Database Profiles

Database profiles controls what objects appear in the objects tree, what detailed views are available for each object type and actions used to operate on objects. Database profiles are database specific and here you can either decide to let DbVisualizer automatically pick (recommended) the appropriate profile 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

| Profile | Version | Date | Description |
|---|---|---|---|
| db2 | 1.36 | 2006/09/19 | Profile for DB2 UDB |
| db2-zos | 1.25 | 2006/09/19 | Profile for DB2 UDB on z/OS |
| generic | 1.16 | 2006/08/31 | Generic profile for any database |
| informix | 1.43 | 2006/09/19 | Profile for Informix IDS |
| mimer | 1.23 | 2006/09/19 | Profile for Mimer SQL |
| mysql | 1.57 | 2006/09/19 | Profile for MySQL |
| oracle | 1.103 | 2006/09/19 | Profile for Oracle |
| postgresql | 1.30 | 2006/09/06 | Profile for PostgreSQL |

**Figure: Database Profile category for a database connection**

**Note:** The way DbVisualizer auto detects a profile is based on mappings in the **DBVIS-HOME/resources/database- mappings.xml** file.

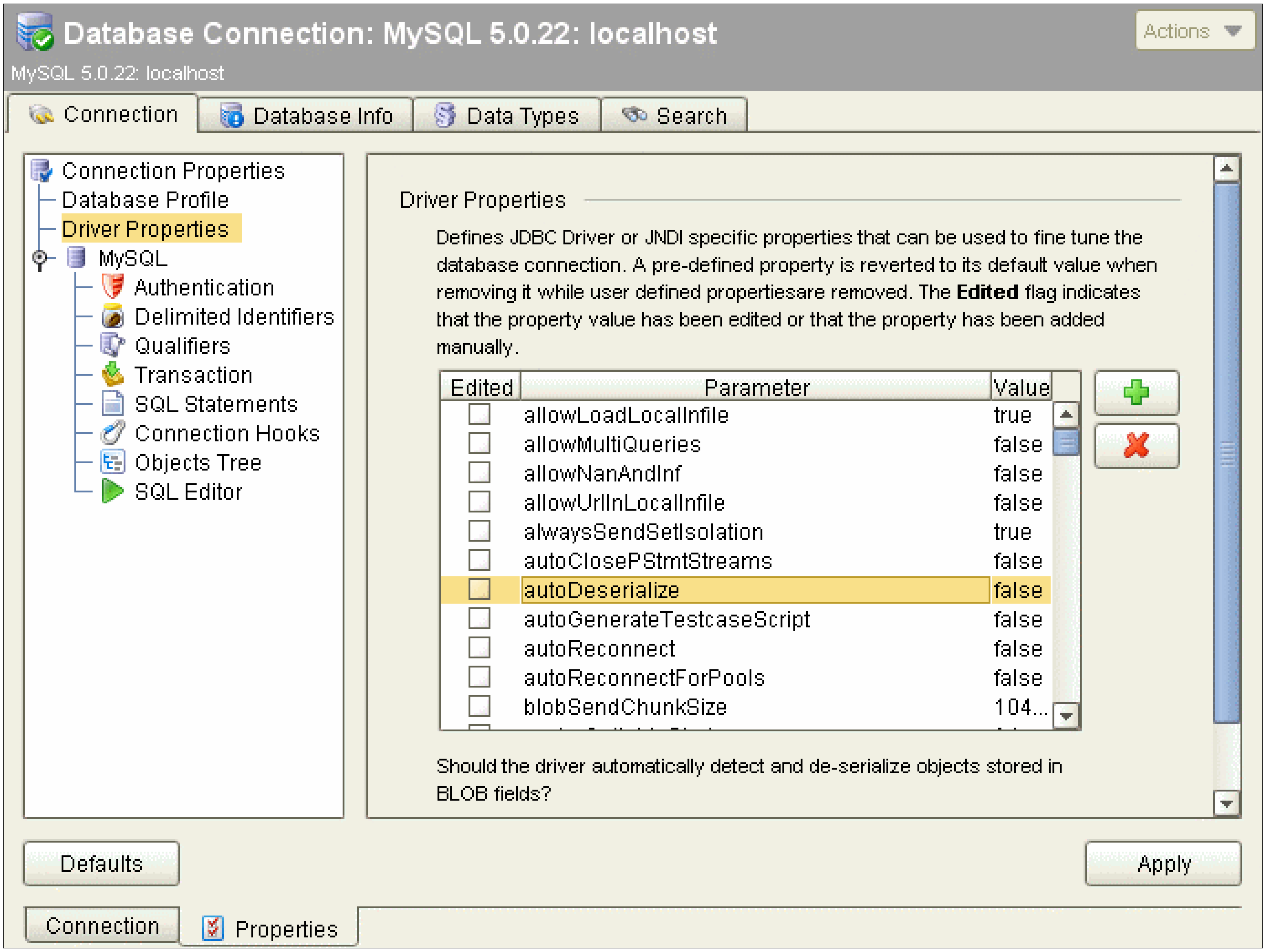
If you manually choose a database profile then this choice will be saved between invocations of DbVisualizer.

## Driver Properties

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

### Driver Properties for JDBC Driver

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



Driver Properties

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 propertiesare removed. The **Edited** flag indicates that the property value has been edited or that the property has been added manually.

| Edited | Parameter | Value |
|---|---|---|
| □ | allowLoadLocalInfile | true |
| □ | allowMultiQueries | false |
| □ | allowNanAndInf | false |
| □ | allowUrlInLocalInfile | false |
| □ | alwaysSendSetIsolation | true |
| □ | autoClosePStmtStreams | false |
| □ | autoDeserialize | false |
| □ | autoGenerateTestcaseScript | false |
| □ | autoReconnect | false |
| □ | autoReconnectForPools | false |
| □ | blobSendChunkSize | 104... |

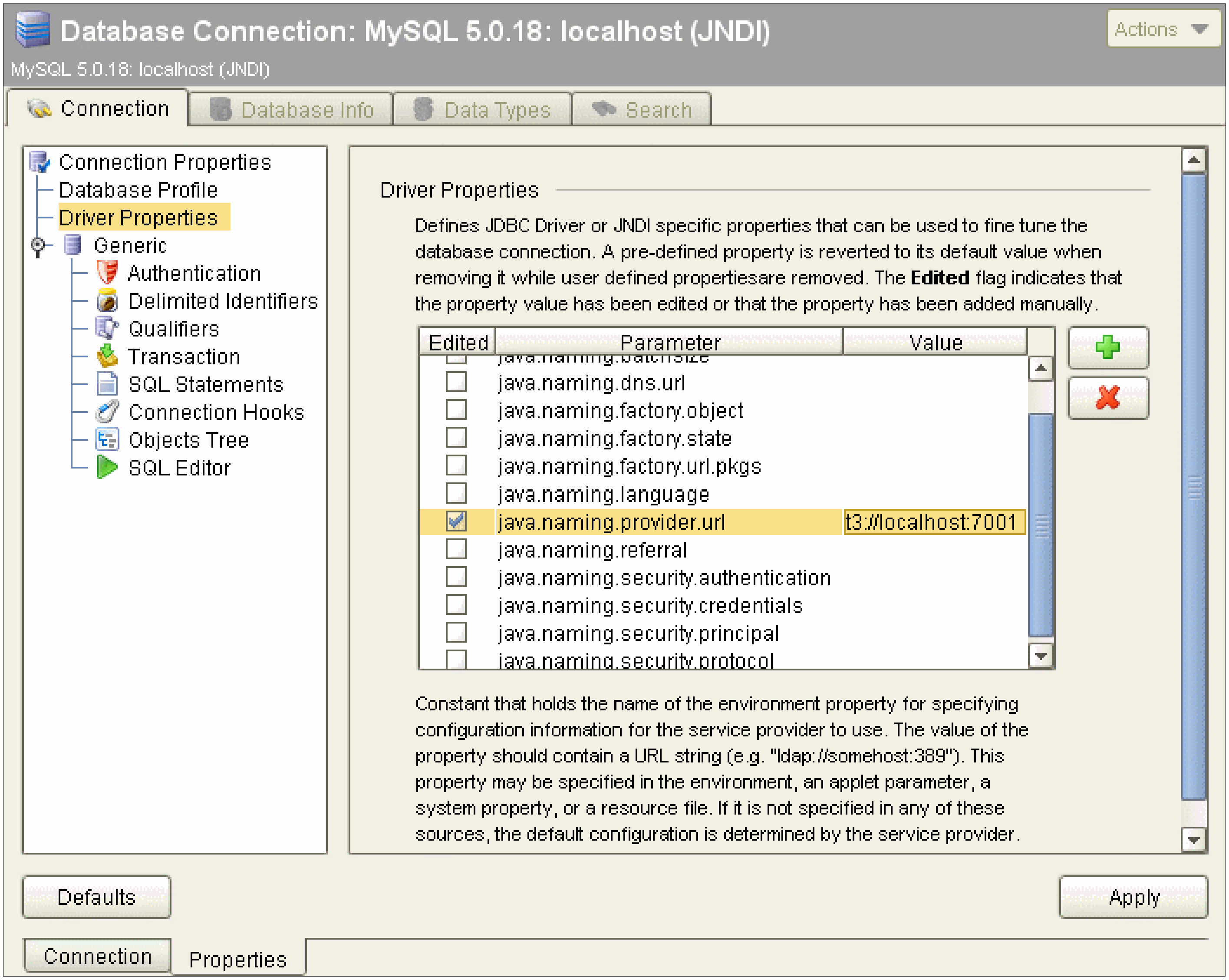
Should the driver automatically detect and de-serialize objects stored in BLOB fields?

**Figure: Driver Properties for JDBC Driver**

The list of parameters, their default values and parameter descriptions are determined by the actual driver. Not all drivers supports additional driver properties. To change a value just modify it in the list. The first column in the list indicates whether the property has been modified or not and so whether DbVisualizer will pass that parameter and value onto the driver at connect time.
New parameters can be added using the buttons at the bottom of the dialog. Be aware that additional parameters do not necessarily mean that the driver will do anything with them.

### Driver Properties for JNDI Lookup

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



Driver Properties

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 propertiesare removed. The **Edited** flag indicates that the property value has been edited or that the property has been added manually.

| Edited | Parameter | Value |
|---|---|---|
| □ | java.naming.batchsize | |
| □ | java.naming.dns.url | |
| □ | java.naming.factory.object | |
| □ | java.naming.factory.state | |
| □ | java.naming.factory.url.pkgs | |
| □ | java.naming.language | |
| ☑ | java.naming.provider.url | t3://localhost:7001 |
| □ | java.naming.referral | |
| □ | java.naming.security.authentication | |
| □ | java.naming.security.credentials | |
| □ | java.naming.security.principal | |
| □ | java.naming.security.protocol | |

Constant that holds the name of the environment property for specifying configuration information for the service provider to use. The value of the property should contain a URL string (e.g. "ldap://somehost:389"). This property may be specified in the environment, an applet parameter, a system property, or a resource file. If it is not specified in any of these sources, the default configuration is determined by the service provider.

**Figure: Driver Properties for JNDI lookup**

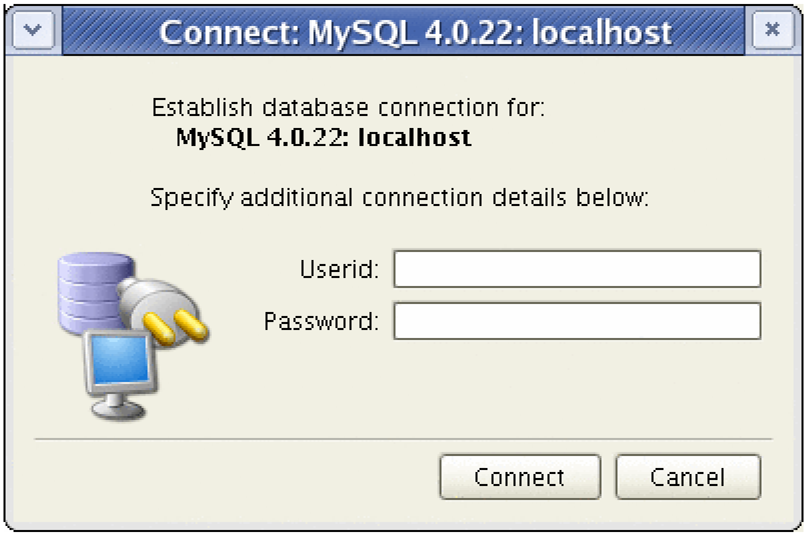
The list of options for JNDI lookup is determined by the constants in the **javax.naming.Context** class. To change a value just modify the value of the parameter. The first column in the list indicates whether the property has been modified or not and so whether DbVisualizer will pass that parameter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

## Always ask for userid and/or password

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

The **Require Userid** and **Require Password** connection properties can be enabled to control that DbVisualizer automatically should prompt for userid and/or password once a connection is established. Enabling either one or both of these and leaving the **Userid** and

**Password** fields blank for a database connection ensures that DbVisualizer will not keep this vital information between sessions. The following figure is displayed if requiring both userid and password.



Connect: MySQL 4.0.22: localhost

Establish database connection for:
**MySQL 4.0.22: localhost**

Specify additional connection details below:

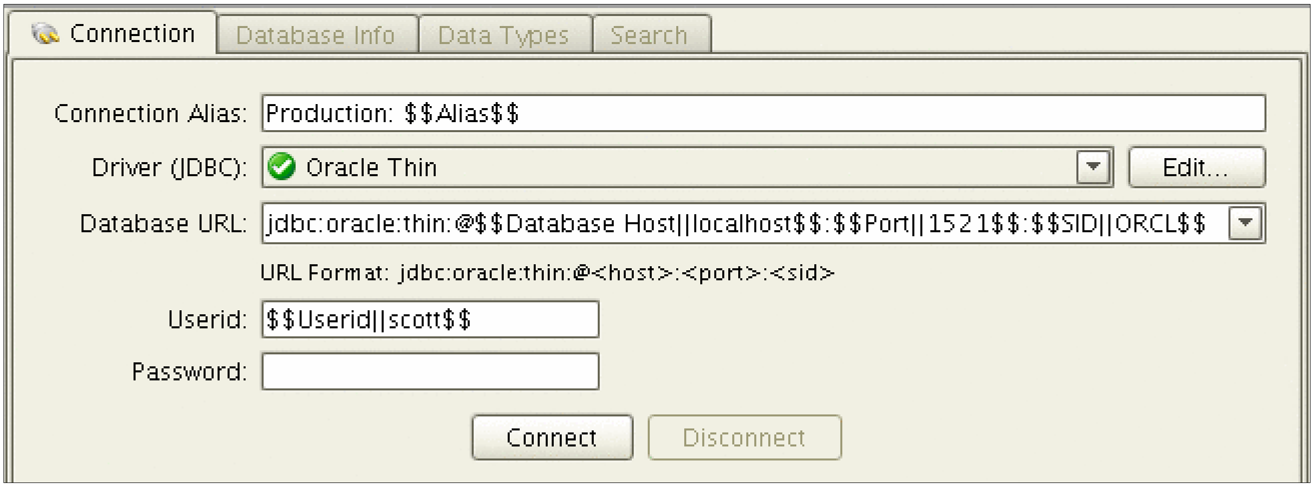
Userid:

Password:

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

## Using variables in the Connection details

Variables can be used in any of the fields in the Connection tab. This can be useful instead of 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 of variables that are identified by the dollar characters, **`$$...$$`**.



Connection Alias: Production: $$Alias$$

Driver (JDBC): Oracle Thin

Database URL: jdbc:oracle:thin:@$$Database Host||localhost$$:$$Port||1521$$:$$SID||ORCL$$

URL Format: jdbc:oracle:thin:@<host>:<port>:<sid>

Userid: $$Userid||scott$$

Password:

**Figure: Connection tab with variables**

The following variables appear in the figure:

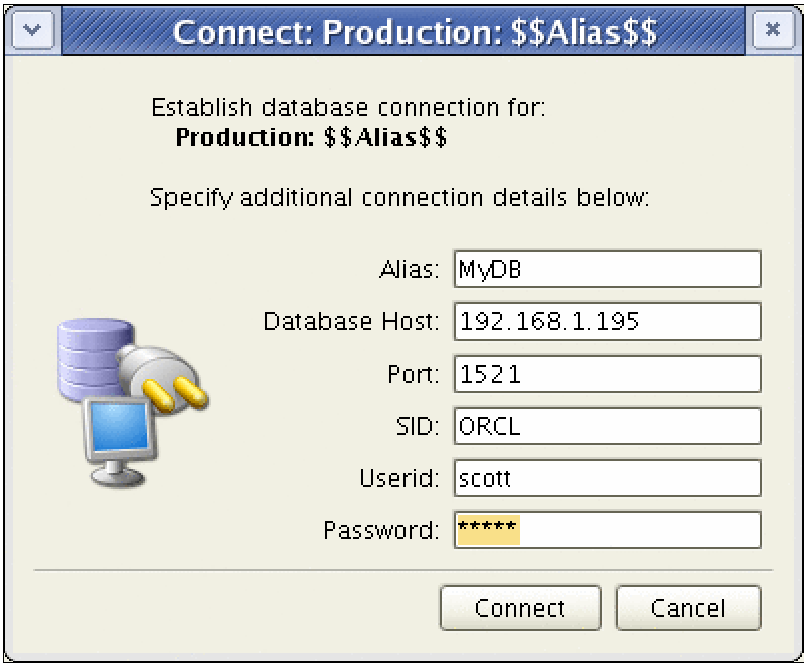
- `$$Alias$$`
- `$$Database Host||dbhost2|||choices=[dbhost1,dbhost2,dbhost3] $$`
- `$$Port||1521$$`
- `$$SID||ORCL$$`

- `$$Userid||scott$$`

All of these variables defines a default value after the "||" delimiter except **$$Alias$$** that have no default value. These default values will appear in the connect dialog once a connection is requested. The **$$Database Host$$** variable includes the **choices** option. Here you can specify a comma separated list of choices that will appear in a drop down. The drop down will be editable so the user is not locked to choose from the list only.

The following figure shows the connect dialog based on the information above.

**Note:** Using variables in conjunction with the **Require Userid** and/or **Require Password** settings also works.



Connect: Production: $$Alias$$

Establish database connection for:
**Production: $$Alias$$**

Specify additional connection details below:

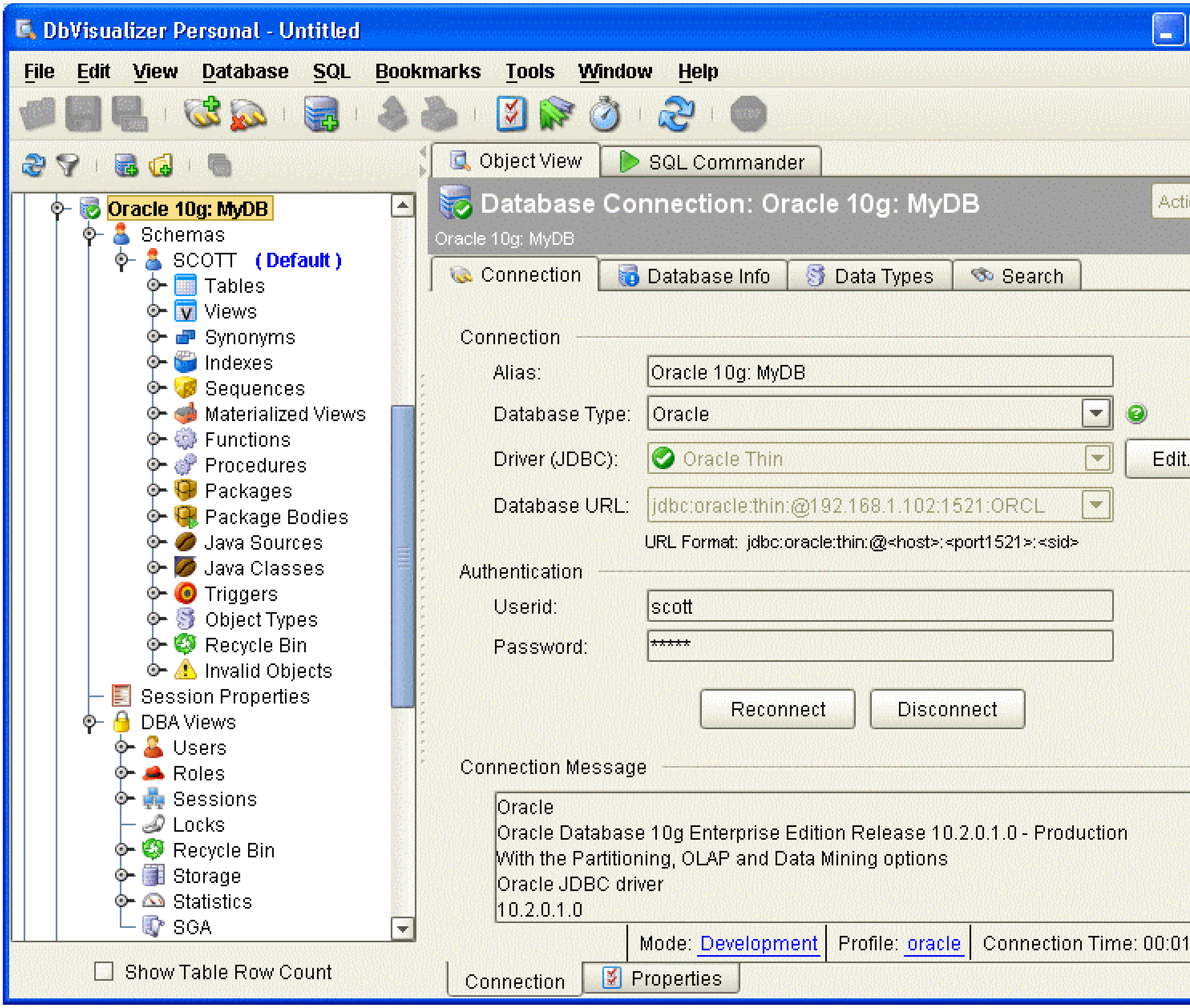
Alias: MyDB
Database Host: 192.168.1.195
Port: 1521
SID: ORCL
Userid: scott
Password: *****

**Figure: Connection tab with variables**

Enter the appropriate information in the fields and then press the **Connect** button to establish the connection. Once connected will DbVisualizer automatically substitute the variables in the Connection tab with the values entered in the connect dialog. These will at disconnect from the database revert back to the original variable definitions.

# Connect to the Database

Press **Connect** when all information has been specified. DbVisualizer will pass all entered information onto the selected driver and if the connection is established the following will appear.



Alias: Oracle 10g: MyDB

Database Type: Oracle

Driver (JDBC): Oracle Thin

Database URL: jdbc:oracle:thin:@192.168.1.102:1521:ORCL

URL Format: jdbc:oracle:thin:@<host>:<port1521>:<sid>

Userid: scott

Password: *****

```
Oracle
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
Oracle JDBC driver
10.2.0.1.0
```

Mode: Development | Profile: oracle | Connection Time: 00:01

**Figure: A freshly initiated database connection using JDBC driver**

The **Connection Message** now lists the name and version of the database as well as the name and version of the JDBC driver. The database connection node in the tree indicates that it is connected. The connection properties cannot be edited once while a database connection is established. The **Alias** can be edited by selecting the database connection node in the tree and then clicking on the name.

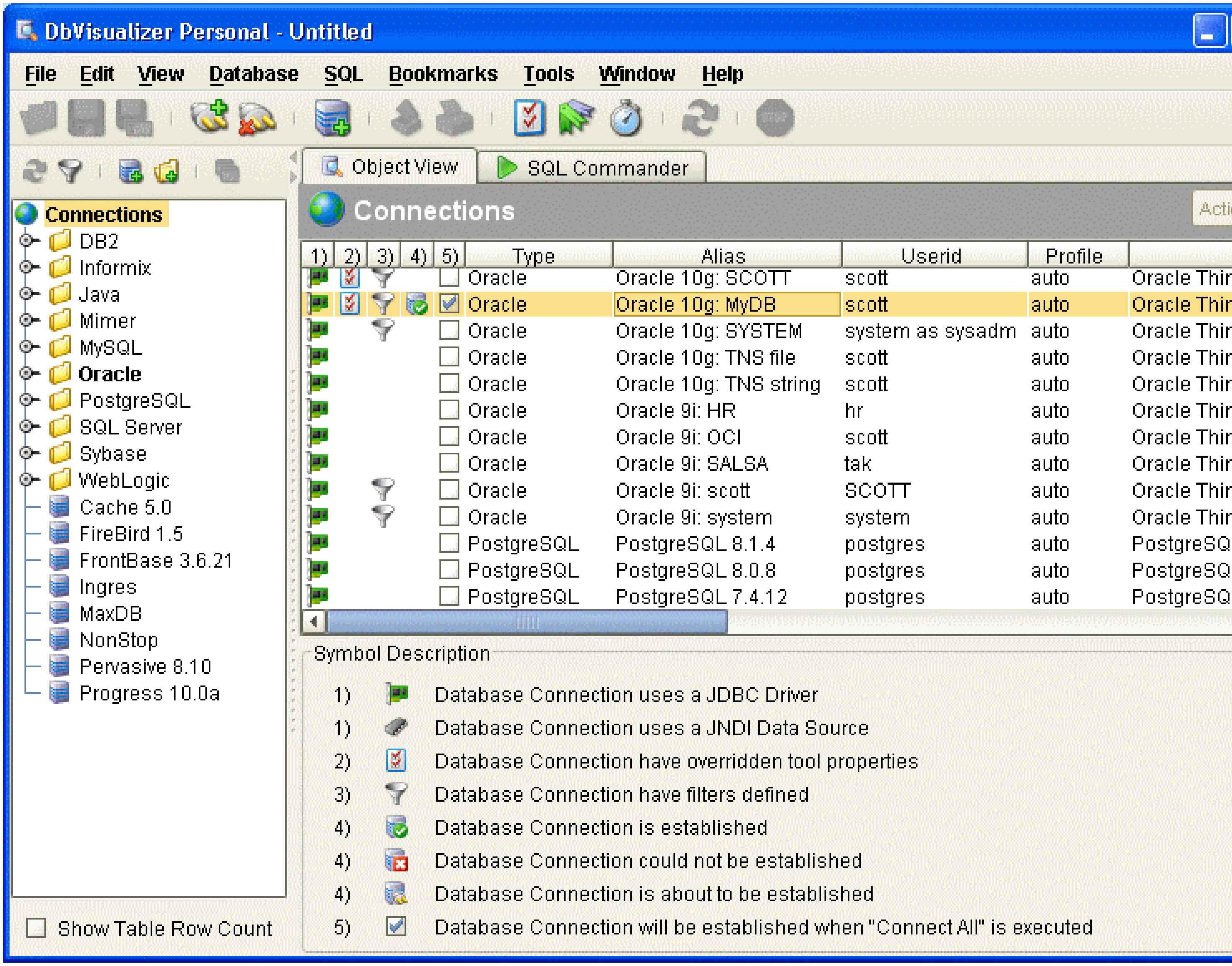
The figure above also shows that the database connection node in the tree has been expanded to show its child objects.

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

| Error Message | Explanation |
|---|---|
| No suitable driver.<br>There is no driver that can handle a connection for the specified URL. The most common reason is that the driver is not loaded in the Driver Manager. Also make sure the URL is correct spelled. | The JDBC support in Java determines what driver to load based on the database URL. If the URL is malformed then there might be no driver that is able to handle the database connection based on that URL. This error is produced when this situation occurs or when the driver is not loaded in the driver manager. The recommendation is to check the JDBC driver documentation for the correct syntax. |
| java.sql.SQLException: Io exception: Invalid number format for port number Io exception: Invalid number format for port number | The URL templates that are available in the Database URL list contains the "<" and ">" place holders. These are there to indicate that the value between them must be replaced with an appropriate value. The "<" and ">" characters must then be removed.<br>This example error message is produced by the Oracle driver when using the following URL:<br>`jdbc:oracle:thin:@<qinda>:<1521>:<fuji>`<br>Simply remove the "<" and ">" characters and try again. |

## Connections Overview

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



| 1) | 2) | 3) | 4) | 5) | Type | Alias | Userid | Profile | |
|---|---|---|---|---|---|---|---|---|---|
| ⚑ | ☑ | ▽ | | □ | Oracle | Oracle 10g: SCOTT | scott | auto | Oracle Thir |
| ⚑ | ☑ | ▽ | ✓ | ☑ | Oracle | Oracle 10g: MyDB | scott | auto | Oracle Thir |
| ⚑ | | ▽ | | □ | Oracle | Oracle 10g: SYSTEM | system as sysadm | auto | Oracle Thir |
| ⚑ | | | | □ | Oracle | Oracle 10g: TNS file | scott | auto | Oracle Thir |
| ⚑ | | | | □ | Oracle | Oracle 10g: TNS string | scott | auto | Oracle Thir |
| ⚑ | | | | □ | Oracle | Oracle 9i: HR | hr | auto | Oracle Thir |
| ⚑ | | | | □ | Oracle | Oracle 9i: OCI | scott | auto | Oracle Thir |
| ⚑ | | | | □ | Oracle | Oracle 9i: SALSA | tak | auto | Oracle Thir |
| ⚑ | | ▽ | | □ | Oracle | Oracle 9i: scott | SCOTT | auto | Oracle Thir |
| ⚑ | | ▽ | | □ | Oracle | Oracle 9i: system | system | auto | Oracle Thir |
| ⚑ | | | | □ | PostgreSQL | PostgreSQL 8.1.4 | postgres | auto | PostgreSQ |
| ⚑ | | | | □ | PostgreSQL | PostgreSQL 8.0.8 | postgres | auto | PostgreSQ |
| ⚑ | | | | □ | PostgreSQL | PostgreSQL 7.4.12 | postgres | auto | PostgreSQ |

Symbol Description

1) ⚑ Database Connection uses a JDBC Driver

1) Database Connection uses a JNDI Data Source

2) ☑ Database Connection have overridden tool properties

3) ▽ Database Connection have filters defined

4) ✓ Database Connection is established

4) ☒ Database Connection could not be established

4) Database Connection is about to be established

5) ☑ Database Connection will be established when "Connect All" is executed

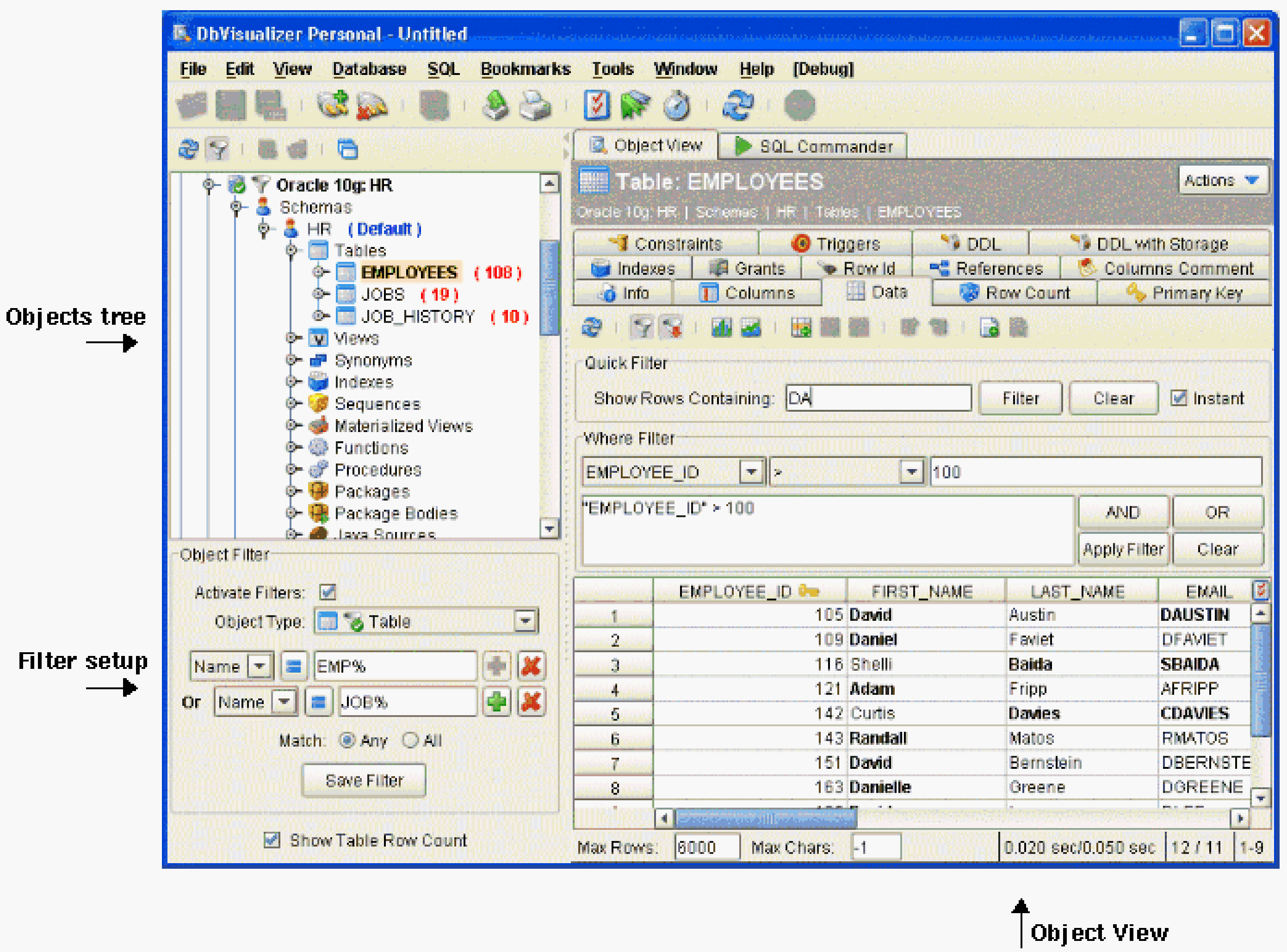
**Figure: The Connections Overview**

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

# Database Objects Explorer

## Introduction

The **Database Objects Tree** is used to explore databases and browse details about objects. What objects that may be explored and what object actions that exist is database dependent.



Activate Filters: ☑

Object Type: Table

Name = EMP%

Or Name = JOB%

Match: ● Any ○ All

☑ Show Table Row Count

Show Rows Containing: DA ☑ Instant

EMPLOYEE_ID > 100

"EMPLOYEE_ID" > 100

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL |
|---|---|---|---|---|
| 1 | 105 | David | Austin | DAUSTIN |
| 2 | 109 | Daniel | Faviet | DFAVIET |
| 3 | 116 | Shelli | Baida | SBAIDA |
| 4 | 121 | Adam | Fripp | AFRIPP |
| 5 | 142 | Curtis | Davies | CDAVIES |
| 6 | 143 | Randall | Matos | RMATOS |
| 7 | 151 | David | Bernstein | DBERNSTE |
| 8 | 163 | Danielle | Greene | DGREENE |

Max Rows: 8000 Max Chars: -1 0.020 sec/0.050 sec 12 / 11 1-9

**Figure: Database Objects tab**

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

The **Filter setup** pane below the tree is used to control what objects are displayed in the tree. It is handy in order to limit the number of objects.

Object actions used to typically create, alter, drop, etc. may exist for the objects in the tree. This is controlled by what database being connected and what database profile is used.

Check coming sections for more information.

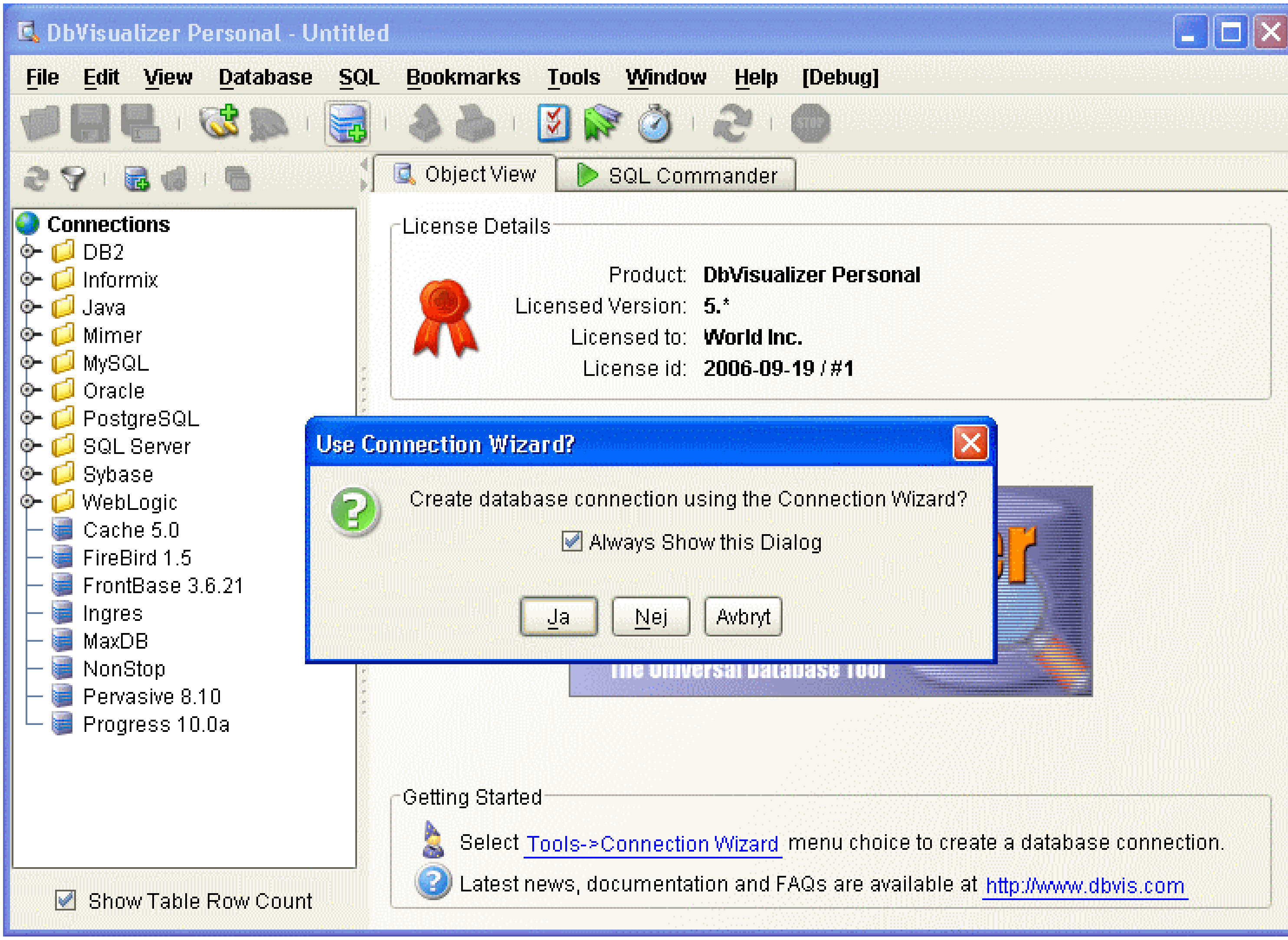
**Tip 1:** The Database Objects Tree is always visible to the left. If the currently selected main tab is the SQL Commander then you can double click on an object in the tree to automatically switch to the **Object View** tab.
**Tip 2:** All object names in the tree can be dragged to any editable text fields including the SQL Commander editor.

# Create Database Connection

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

## Database Connection object

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



License Details

Product: DbVisualizer Personal
Licensed Version: 5.*
Licensed to: World Inc.
License id: 2006-09-19 / #1

Use Connection Wizard?

Create database connection using the Connection Wizard?

☑ Always Show this Dialog

Getting Started

Select Tools->Connection Wizard menu choice to create a database connection.

Latest news, documentation and FAQs are available at http://www.dbvis.com

**Figure: Add database connection**

It is always recommended to use the connection wizard when creating new database connections as it hides the complexity loading drivers and syntax of database URLs. (Detailed information on how to establish a connection is provided in the Load JDBC Driver and Get Connected document).

**Tip 1:** Once a database connection has been setup properly then you just need to double click on the object to establish the connection.

**Tip 2:** The **Database- >Connect All** main menu choice is used to connect all enabled database connections with a single click. You make a database connection "Connect All" - aware in the **Database Properties** or in the **Connections** overview.

## Alias

The name of the database connection object as it appears in the tree is by default the URL of the connection. The **Connection Alias** can be used to override this name to something more descriptive and shorter. Either enter the new name in the Alias field in the Connection sub tab or click on the name in the tree and start editing the name.

### Default database and schema

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

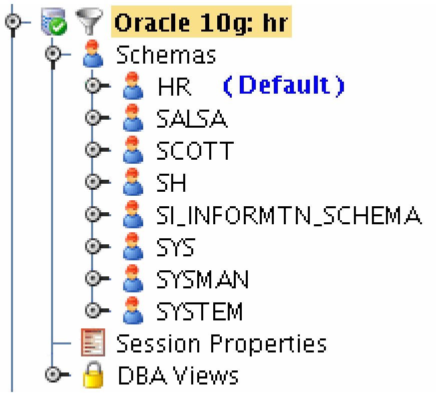


**Figure: The (default) indicator for database and schema objects**

**Tip:** In the Connection Properties you can define that only default database or schema should be visible in the tree.

### Remove and copy database connection objects

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

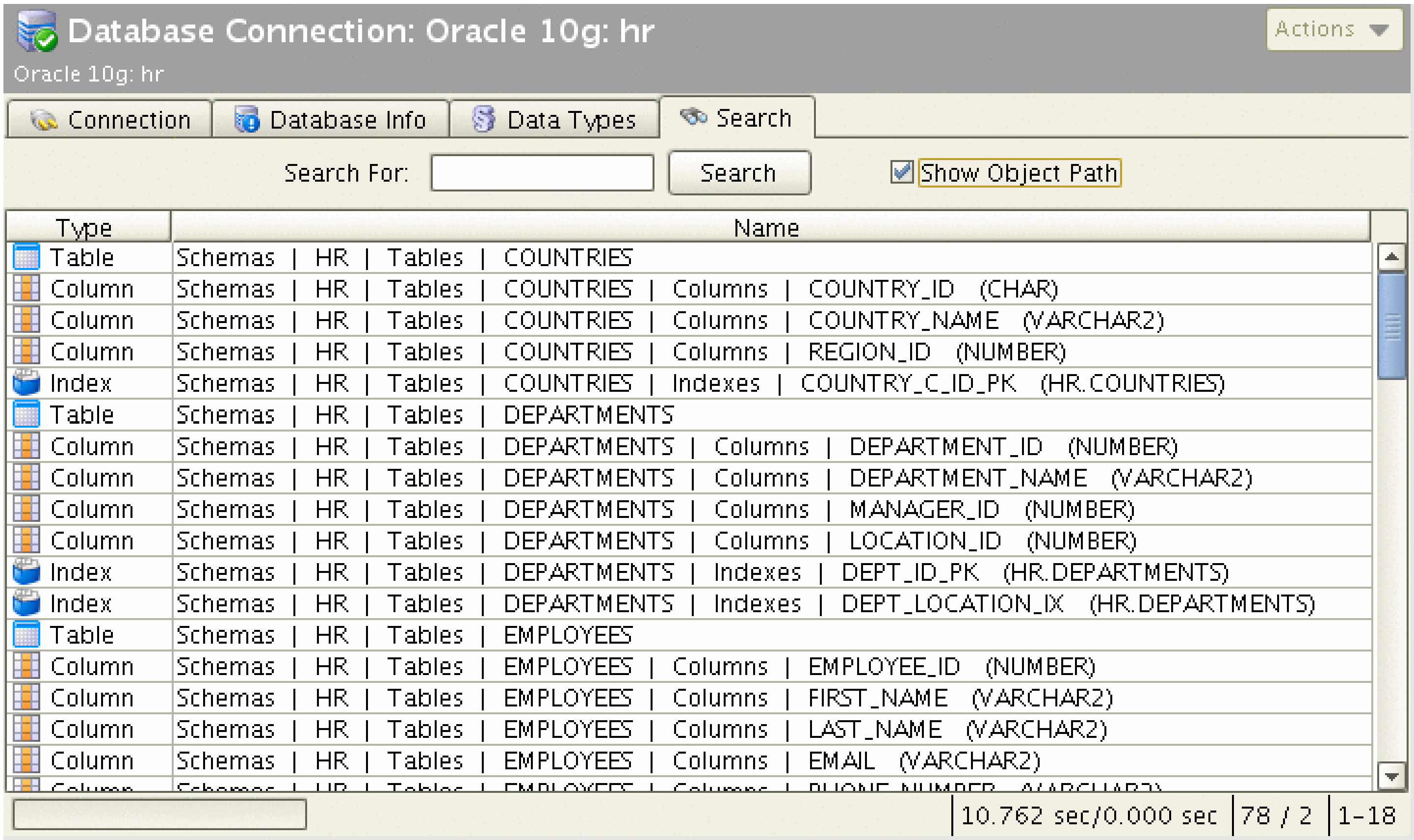
### Database Connection detailed information

The following section briefly explains the tabs in the objects view for a database connection.

| Tab | Description |
|---|---|
| **Connection** | This tab is always enabled and is used to setup the details for a database connection. This is also the place to control the connection state. |
| **Database Info** | When connected, the database info tab shows various information supplied by the driver. Much of this info is low level even though some may be useful. |
| **Data Types** | The data types tab lists all data types supported by the database. |
| **Search** | The search tab is used to search among the objects in the tree. Search operates on the content in the tree based on if there are any filter defined or if any other setting has been set that effects the content of the tree. See next section for more information about search. |

### Search

The Search tab is used to search among the objects in the tree by object name. This result will depend on if there are any tree filter defined or if any other property has been set that affects the content of the tree. The search operation is case insensitive.



Search For: [ ] Search ☑ Show Object Path

| Type | Name |
|---|---|
| Table | Schemas \| HR \| Tables \| COUNTRIES |
| Column | Schemas \| HR \| Tables \| COUNTRIES \| Columns \| COUNTRY_ID (CHAR) |
| Column | Schemas \| HR \| Tables \| COUNTRIES \| Columns \| COUNTRY_NAME (VARCHAR2) |
| Column | Schemas \| HR \| Tables \| COUNTRIES \| Columns \| REGION_ID (NUMBER) |
| Index | Schemas \| HR \| Tables \| COUNTRIES \| Indexes \| COUNTRY_C_ID_PK (HR.COUNTRIES) |
| Table | Schemas \| HR \| Tables \| DEPARTMENTS |
| Column | Schemas \| HR \| Tables \| DEPARTMENTS \| Columns \| DEPARTMENT_ID (NUMBER) |
| Column | Schemas \| HR \| Tables \| DEPARTMENTS \| Columns \| DEPARTMENT_NAME (VARCHAR2) |
| Column | Schemas \| HR \| Tables \| DEPARTMENTS \| Columns \| MANAGER_ID (NUMBER) |
| Column | Schemas \| HR \| Tables \| DEPARTMENTS \| Columns \| LOCATION_ID (NUMBER) |
| Index | Schemas \| HR \| Tables \| DEPARTMENTS \| Indexes \| DEPT_ID_PK (HR.DEPARTMENTS) |
| Index | Schemas \| HR \| Tables \| DEPARTMENTS \| Indexes \| DEPT_LOCATION_IX (HR.DEPARTMENTS) |
| Table | Schemas \| HR \| Tables \| EMPLOYEES |
| Column | Schemas \| HR \| Tables \| EMPLOYEES \| Columns \| EMPLOYEE_ID (NUMBER) |
| Column | Schemas \| HR \| Tables \| EMPLOYEES \| Columns \| FIRST_NAME (VARCHAR2) |
| Column | Schemas \| HR \| Tables \| EMPLOYEES \| Columns \| LAST_NAME (VARCHAR2) |
| Column | Schemas \| HR \| Tables \| EMPLOYEES \| Columns \| EMAIL (VARCHAR2) |
| Column | Schemas \| HR \| Tables \| EMPLOYEES \| Columns \| PHONE_NUMBER (VARCHAR2) |

10.762 sec/0.000 sec | 78 / 2 | 1-18

**Figure: The Search tab**

Search by specifying the name of the object or part of the name and press the **Search** button. The search operation can be stopped using the standard **Stop** button in the main tool bar. The **Show Object Path** check box is used to define whether the complete path for each found object should be displayed in the result or not. This path is the same as if navigating to each object manually in the objects tree.

**Note:** The search may take some time to perform the first time since all objects defined in the actual database profile are examined.

**Tip:** Detailed information of a specific object can be examined by double clicking on a row. This will display all information about the object in a separate window.

## Organizing Database Connections in Folders

The folder object is used to organize and group database connections. It allows child folder objects in an unlimited hierarchy. You can either use the **View- >Move Up/Down** main menu choices to organize the folders (and database connections) in the tree, or you can also use drag and drop to move nodes.

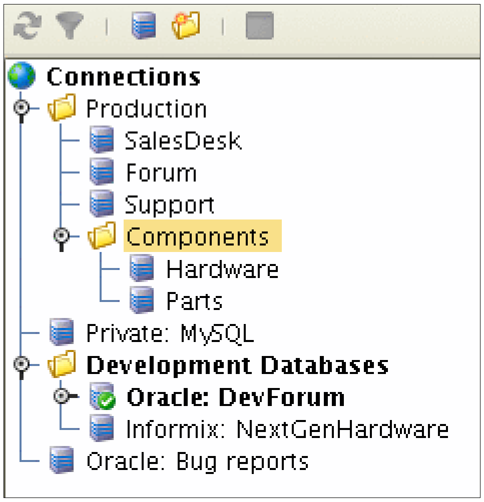
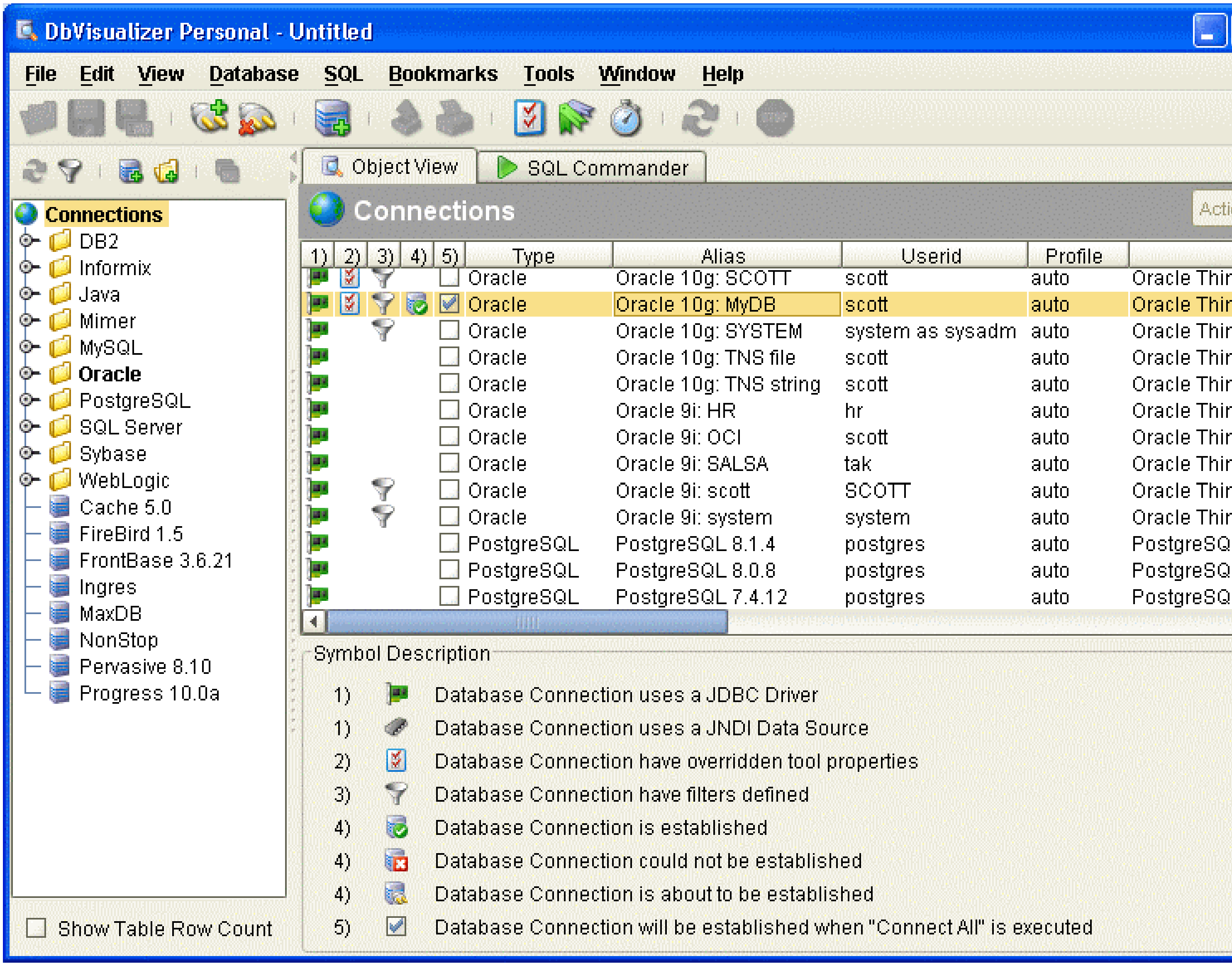


**Figure: The database objects tree and the folder object type**

## Connections overview

The **Connections** object is the root object in the tree and acts as a holder for all database connections and folders. The purpose of this is that when selected it displays an overview of all database connections in the details view. Here you can see the basic settings and states for your database connections. Read more about it in the Load JDBC Driver and Get Connected document.



| 1) | 2) | 3) | 4) | 5) | Type | Alias | Userid | Profile | |
|---|---|---|---|---|---|---|---|---|---|
| ⚑ | ✓ | ▽ | | ☐ | Oracle | Oracle 10g: SCOTT | scott | auto | Oracle Thir |
| ⚑ | ✓ | ▽ | ● | ☑ | Oracle | Oracle 10g: MyDB | scott | auto | Oracle Thir |
| ⚑ | | ▽ | | ☐ | Oracle | Oracle 10g: SYSTEM | system as sysadm | auto | Oracle Thir |
| ⚑ | | | | ☐ | Oracle | Oracle 10g: TNS file | scott | auto | Oracle Thir |
| ⚑ | | | | ☐ | Oracle | Oracle 10g: TNS string | scott | auto | Oracle Thir |
| ⚑ | | | | ☐ | Oracle | Oracle 9i: HR | hr | auto | Oracle Thir |
| ⚑ | | | | ☐ | Oracle | Oracle 9i: OCI | scott | auto | Oracle Thir |
| ⚑ | | | | ☐ | Oracle | Oracle 9i: SALSA | tak | auto | Oracle Thir |
| ⚑ | | ▽ | | ☐ | Oracle | Oracle 9i: scott | SCOTT | auto | Oracle Thir |
| ⚑ | | ▽ | | ☐ | Oracle | Oracle 9i: system | system | auto | Oracle Thir |
| ⚑ | | | | ☐ | PostgreSQL | PostgreSQL 8.1.4 | postgres | auto | PostgreSQ |
| ⚑ | | | | ☐ | PostgreSQL | PostgreSQL 8.0.8 | postgres | auto | PostgreSQ |
| ⚑ | | | | ☐ | PostgreSQL | PostgreSQL 7.4.12 | postgres | auto | PostgreSQ |

Symbol Description

1) ⚑ Database Connection uses a JDBC Driver
1) Database Connection uses a JNDI Data Source
2) ✓ Database Connection have overridden tool properties
3) ▽ Database Connection have filters defined
4) ● Database Connection is established
4) ☒ Database Connection could not be established
4) Database Connection is about to be established
5) ☑ Database Connection will be established when "Connect All" is executed

☐ Show Table Row Count

Figure: Connections object

# Database Objects Tree

## Standard Actions

The Database Objects tool bar buttons are used to do tree related operations. These are individually enabled or disabled based on what object is currently selected.

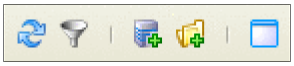


Figure: Objects tree toolbar

Description of the buttons from the left:

| Tool bar button | Description |
|---|---|
| **Reload** | Reloads the currently selected object by asking the JDBC driver to fetch information for the object from the database. This is useful if new objects have been created or removed. |
| **Show/Hide Tree Filter** | Is a toggle button that determines whether the Filter management pane will be displayed below the tree or not. |
| **Create Database Connection** | Adds a new Database Connection object in the tree. The location of the new object is determined based on the current selection. If no selection then the new is object added at the end of the list. |
| **Create Folder** | Creates a new folder object. |
| **Show in Window** | Request to display the details view in a separate window for the selected object. |

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

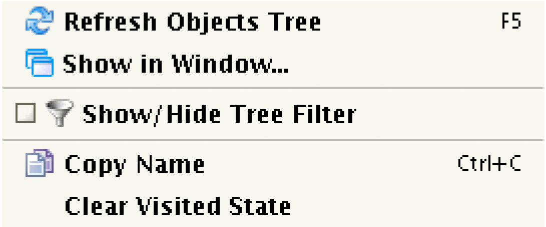


**Figure: Standard right click menu actions for all objects**

## Object Actions

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

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

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

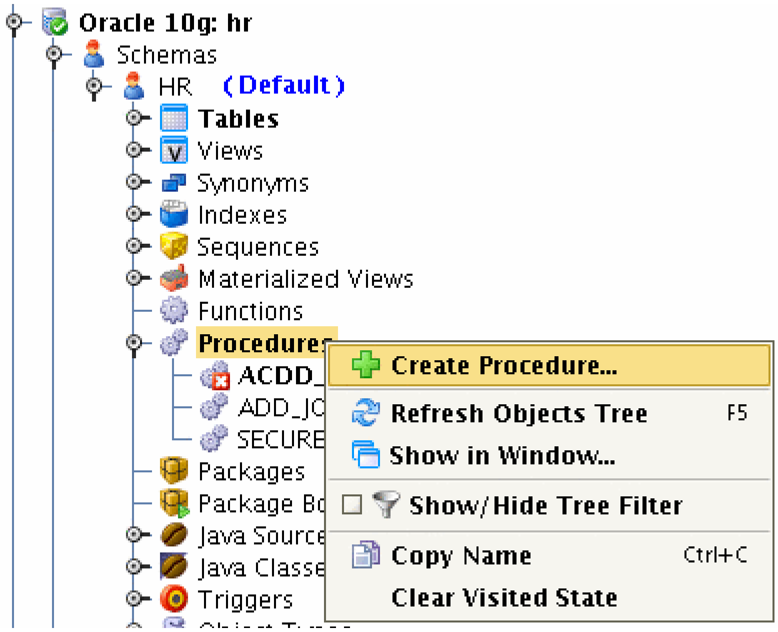


**Figure: Object actions menu**

## Common Object Actions

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

### Create Table

The create table action shows the Create Table assistant dialog. it is used to setup the columns their characteristics and primary keys for a new table. The final SQL that the assistant produce is then executed in the SQL Commander. Read more about this feature in Create Table and Index Assistants.

### Create Index

The create index action shows the Create Index assistant dialog. it is used to setup columns for new table indexes. The final SQL that the assistant produce is then executed in the SQL Commander. Read more about this feature in Create Table and Index Assistants.

### Import Table Data

Import Table Data shows a dialog used to import a CSV file into the actual table. Various configurations how the source file is organized and data mapping are offered. Read more in Export, Import and Print.

### Script Object to SQL Editor

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

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

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

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

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

The **Script Object to SQL Editor** is also used to generate the **DDL** for **Table** and **View** objects. (This is only supported in the database specific profiles).

#### Script Object to New SQL Editor

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

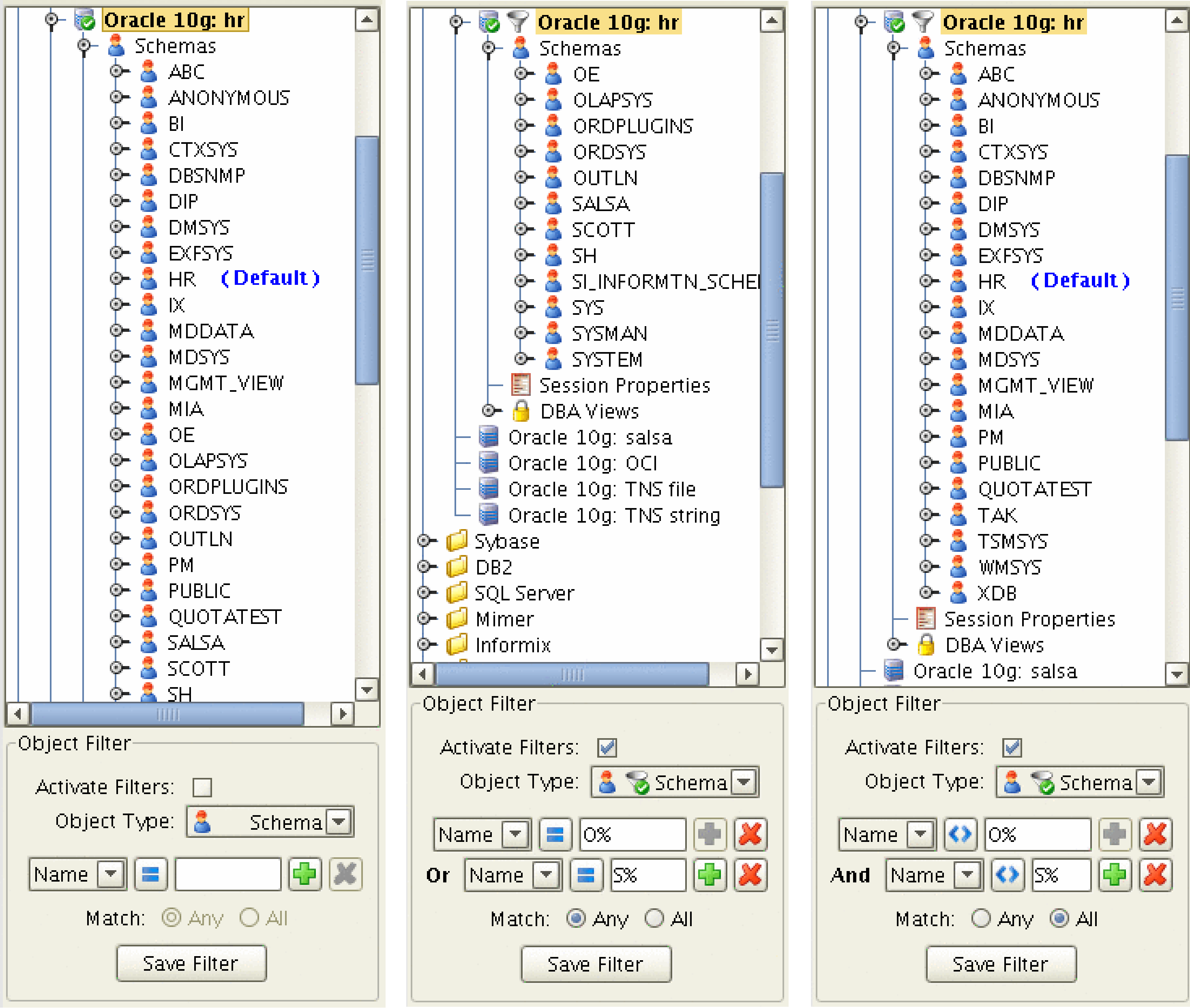
## Objects Tree Filtering

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

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

| The unfiltered schema objects | The same objects but now | Filter defined as all names that |
|---|---|---|

| | | |
|---|---|---|
| for an Oracle connection. | filtered based on all schema names starting with "O" or "S". | do not start with "O" and "S". |



Oracle 10g: hr
Schemas
ABC
ANONYMOUS
BI
CTXSYS
DBSNMP
DIP
DMSYS
EXFSYS
HR (Default)
IX
MDDATA
MDSYS
MGMT_VIEW
MIA
OE
OLAPSYS
ORDPLUGINS
ORDSYS
OUTLN
PM
PUBLIC
QUOTATEST
SALSA
SCOTT
SH

Object Filter
Activate Filters: ☐
Object Type: Schema
Name = 
Match: ○ Any ○ All
Save Filter

Oracle 10g: hr
Schemas
OE
OLAPSYS
ORDPLUGINS
ORDSYS
OUTLN
SALSA
SCOTT
SH
SI_INFORMTN_SCHE
SYS
SYSMAN
SYSTEM
Session Properties
DBA Views
Oracle 10g: salsa
Oracle 10g: OCI
Oracle 10g: TNS file
Oracle 10g: TNS string
Sybase
DB2
SQL Server
Mimer
Informix

Object Filter
Activate Filters: ☑
Object Type: Schema
Name = O%
Or Name = S%
Match: ● Any ○ All
Save Filter

Oracle 10g: hr
Schemas
ABC
ANONYMOUS
BI
CTXSYS
DBSNMP
DIP
DMSYS
EXFSYS
HR (Default)
IX
MDDATA
MDSYS
MGMT_VIEW
MIA
PM
PUBLIC
QUOTATEST
TAK
TSMSYS
WMSYS
XDB
Session Properties
DBA Views
Oracle 10g: salsa

Object Filter
Activate Filters: ☑
Object Type: Schema
Name <> O%
And Name <> S%
Match: ○ Any ● All
Save Filter

**Figure: Examples of tree filter settings**

An active filter for a database connection is represented by the funnel icon just before the database connection name. The active state for a filter is defined using the **Active** box in the name filter pane. A filter can only be activated if there are any filters defined.

Up to 15 filters can be defined per object type.

**Tip:** It is often desired to list only the default schema or catalog (database) in the database objects tree. This can be accomplished using the filtering functionality but the recommended place to do this is in the properties tab for the database connection. Please read more about the **Show only default Database or Schema** in Tool Properties document.

### Show Table Row Count

The **Show Table Row Count** setting below the database objects tree defines whether the number of rows for table objects will be listed after the name of the table.

**Note:** Enabling this property results in a performance degradation.

## Database Profiles

A Database Profile is the foundation in DbVisualizer used to express database specific support. A database profile is briefly a definition of what information should be presented in the database objects tree and in the various object views. In addition it define actions for the object types defined in the profile. DbVisualizer loads the matching database profile at connect. If no matching profile is found or if running DbVisualizer Free a Generic profile is loaded with rudimentary database support.

### Database Specific Support

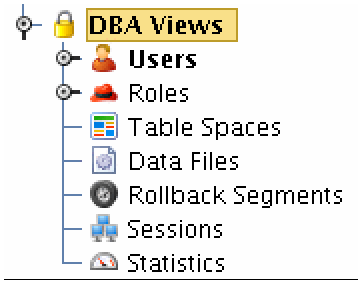
DbVisualizer Personal currently offer database specific support (database profiles) for the following databases (click links for details):

- Oracle
- DB2
- Sybase ASE
- SQL Server
- MySQL
- PostgreSQL
- Informix
- Mimer
- JavaDB/Derby

Since each of the specialized database profiles handles different object types will the database objects tree look different. The structure and organization of a database profile is also something that may impact the layout of the tree even though the provided ones are similar in their structure. There are two root nodes in the majority of profiles:

- User objects
- DBA objects

User objects are for example, tables, views, triggers, functions, etc. while DBA objects most probably requires certain privileges in the database in order to access them. DbVisualizer organizes all DBA objects in the **DBA Views** tree object. If privileges are not sufficient to access a DBA object may this result in an error. This is an example of the DBA sub tree.



DBA Views
Users
Roles
Table Spaces
Data Files
Rollback Segments
Sessions
Statistics

**Figure: The DBA Views tree object**

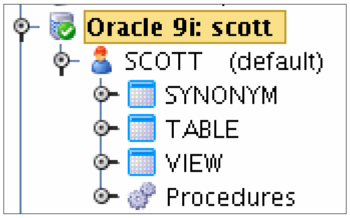
**Note:** Database profiles are defined in XML and it is quite easy to extend and modify them. Read more in the Plug-in Framework document.

## Generic profile

DbVisualizer supports a wide range of databases and since the nature of these and what they support is different from vendor to vendor so will the appearance and the structure of the tree below the database connection objects look different. The generic database profile and DbVisualizer Free display objects based on what JDBC offers in terms of database objects (aka meta data information). DbVisualizer does this simply by asking the actual JDBC driver for all schemas, databases, tables and procedures. It then builds the tree based on what it gets.

The advantage of using JDBC to get meta data about the database is that it's the responsibility of the driver to perform the operations in order to get the requested information. The drawback of letting the driver do this is that JDBC doesn't offer that much support for getting meta data information about all objects in a database i.e. the object types that are presented in the tree are sufficient for most database while there are obvious objects that are missing for some databases. The solution is simply to upgrade to the DbVisualizer Personal edition.

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



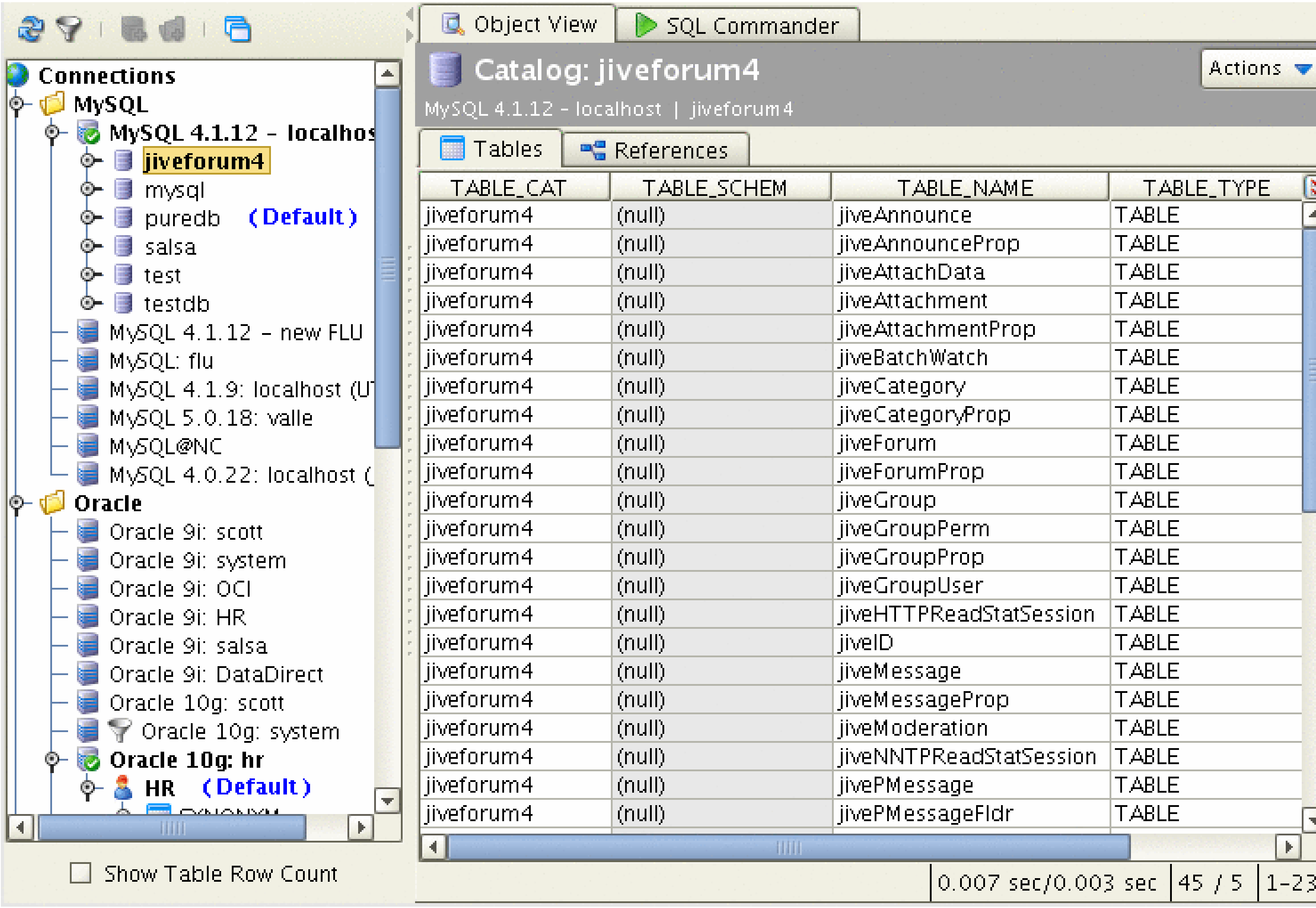
Oracle 9i: scott
SCOTT (default)
SYNONYM
TABLE
VIEW
Procedures

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

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

### Database (Catalog) object

The **Catalog** object is the generic JDBC term for a **Database** in for example Sybase, PostgreSQL, SQL Server and MySQL. It groups all objects for a logical database. The object view for a catalog is a pane with two tabs, **Tables** and **References**. The tables tab lists all the tables that are located in the catalog while references shows the exact same list of tables but instead as a referential integrity graph.



☐ Show Table Row Count

| TABLE_CAT | TABLE_SCHEM | TABLE_NAME | TABLE_TYPE |
|---|---|---|---|
| jiveforum4 | (null) | jiveAnnounce | TABLE |
| jiveforum4 | (null) | jiveAnnounceProp | TABLE |
| jiveforum4 | (null) | jiveAttachData | TABLE |
| jiveforum4 | (null) | jiveAttachment | TABLE |
| jiveforum4 | (null) | jiveAttachmentProp | TABLE |
| jiveforum4 | (null) | jiveBatchWatch | TABLE |
| jiveforum4 | (null) | jiveCategory | TABLE |
| jiveforum4 | (null) | jiveCategoryProp | TABLE |
| jiveforum4 | (null) | jiveForum | TABLE |
| jiveforum4 | (null) | jiveForumProp | TABLE |
| jiveforum4 | (null) | jiveGroup | TABLE |
| jiveforum4 | (null) | jiveGroupPerm | TABLE |
| jiveforum4 | (null) | jiveGroupProp | TABLE |
| jiveforum4 | (null) | jiveGroupUser | TABLE |
| jiveforum4 | (null) | jiveHTTPReadStatSession | TABLE |
| jiveforum4 | (null) | jiveID | TABLE |
| jiveforum4 | (null) | jiveMessage | TABLE |
| jiveforum4 | (null) | jiveMessageProp | TABLE |
| jiveforum4 | (null) | jiveModeration | TABLE |
| jiveforum4 | (null) | jiveNNTPReadStatSession | TABLE |
| jiveforum4 | (null) | jivePMessage | TABLE |
| jiveforum4 | (null) | jivePMessageFldr | TABLE |

**Figure: The view for Catalog objects**

The child objects shown for a catalog depends on the capabilities of the JDBC driver. Normally you will see a list of the supported table types that groups the tables of these types. The number within parentheses is the number of tables. The example shows a MySQL database. The driver reports that it can handle the table types, **TABLE** and **LOCAL TEMPORARY**. (These table types are the same as those listed in the **Table Types** tab when selecting a database connection object.

**Tip 1:** You can double click on a catalog object to display the detail view in a separate window.

**Tip 2:** Select one or several rows (cells) in the tables grid and then choose **Database->Build Select Script** to create a select script for the selected tables.

### Schema object

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

The following screen shot shows the information for the selected schema with the Reference tab selected.

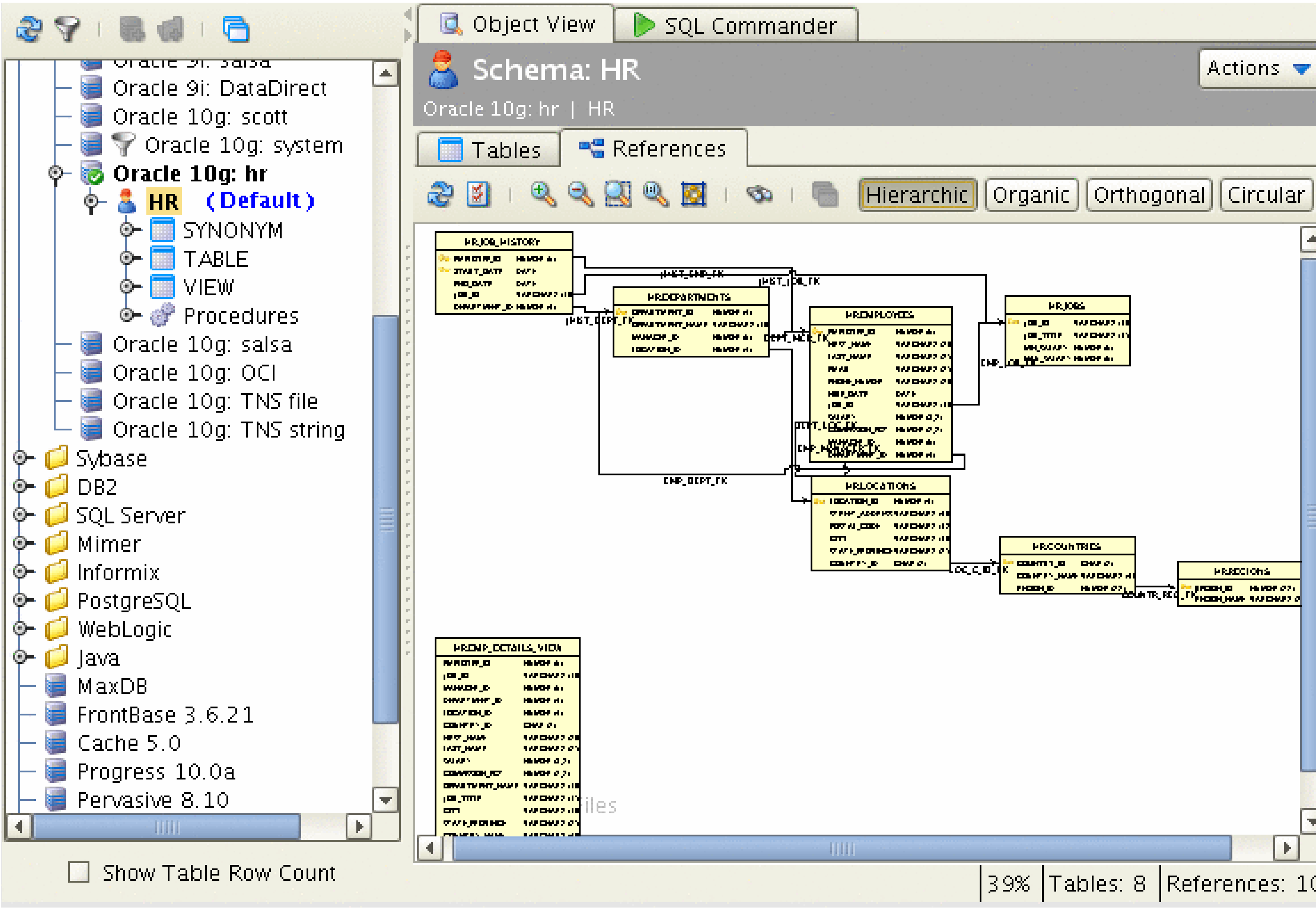
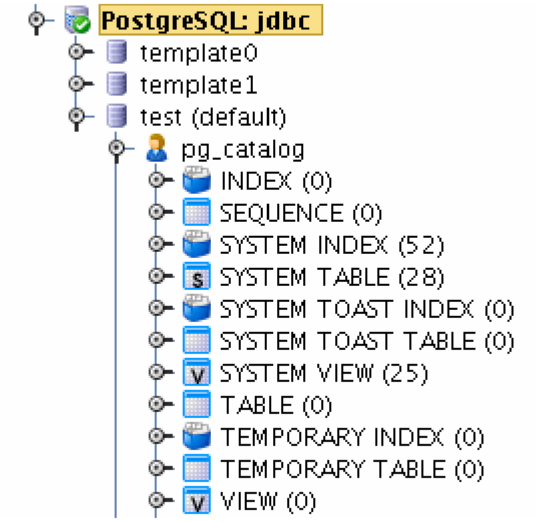


**Figure: The view for Schema objects**

## Table Type object

The **Table Type** object has been briefly explained earlier. The name and the number of table type objects are determined by the driver as DbVisualizer asks for the supported table types. When DbVisualizer retrieves all tables it checks each table's type and puts them into the matching table type object. The reason is simply to make the tree easier to browse.



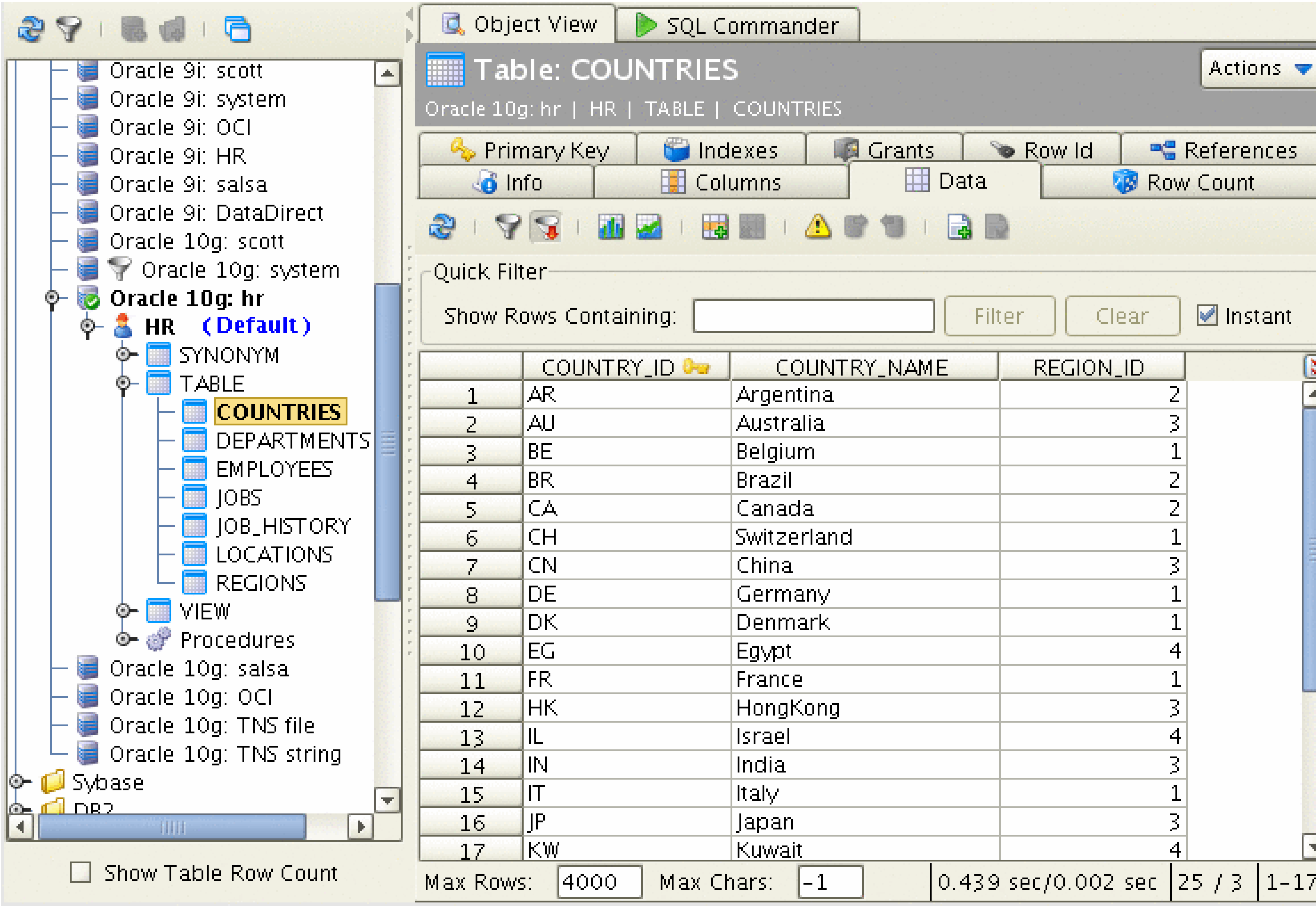
- PostgreSQL: jdbc
  - template0
  - template1
  - test (default)
    - pg_catalog
      - INDEX (0)
      - SEQUENCE (0)
      - SYSTEM INDEX (52)
      - SYSTEM TABLE (28)
      - SYSTEM TOAST INDEX (0)
      - SYSTEM TOAST TABLE (0)
      - SYSTEM VIEW (25)
      - TABLE (0)
      - TEMPORARY INDEX (0)
      - TEMPORARY TABLE (0)
      - VIEW (0)

**Figure: Example of table type objects for PostgreSQL**

Note: Even though the figure above lists objects as INDEX, SEQUENCE, VIEW, etc are all treated as tables by DbVisualizer.

### Table object

The **Table** object is probably the most frequently accessed object in the tree as when selected it shows not only a lot of information about the table but also the data in it. This is also the place where data edits are performed.



☐ Show Table Row Count

Show Rows Containing: ☑ Instant

| | 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 |

Max Rows: 4000 Max Chars: -1 0.439 sec/0.002 sec 25 / 3 1-17

**Figure: The view for Table objects**

The detailed view for table objects displays

| Tab | Description |
|---|---|
| **Info** | Brief information about the table object |
| **Columns** | This tab lists type information about all columns in the table |
| **Data** | Read more in Data tab |
| **Row Count** | Lists the table row count |
| **Primary Key** | Shows the primary key |
| Indexes | Lists all indexes for the table |
| Table | Displays any privileges for the table |

| | |
|---|---|
| Privileges | |
| Row Id | Displays the optimal set of columns that uniquely identifies a row |
| **References** | Read more in References tab |

### Data tab

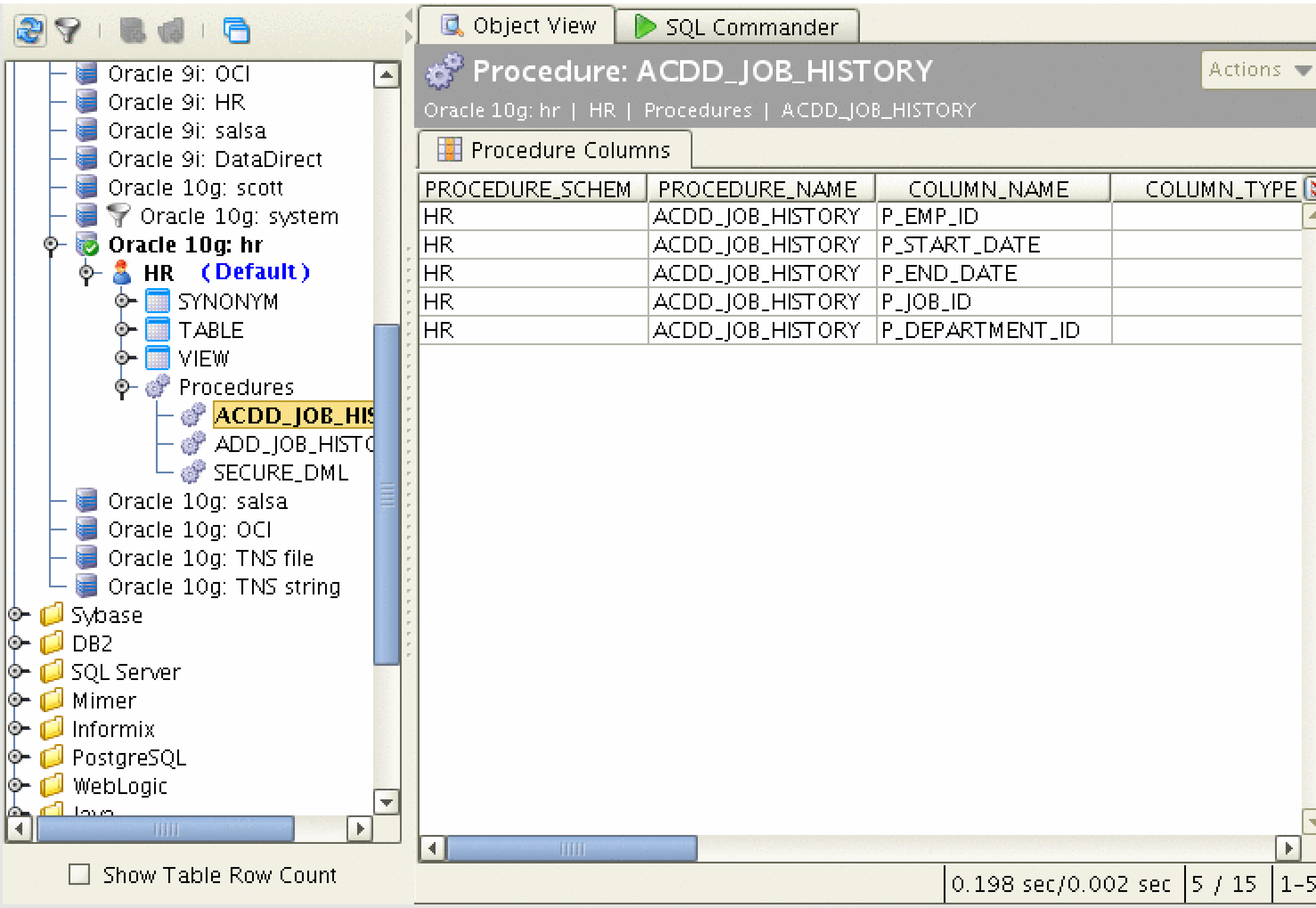
Read more about the Data tab in the Table Data section.

### References tab

Read more about the References tab in the References section.

## Procedure object

The procedure object is probably the simplest since it shows the name of the procedure or function in the tree, and in the object view lists the parameters that are used when calling it.



Procedure: ACDD_JOB_HISTORY

Oracle 10g: hr | HR | Procedures | ACDD_JOB_HISTORY

Procedure Columns

| PROCEDURE_SCHEM | PROCEDURE_NAME | COLUMN_NAME | COLUMN_TYPE |
|---|---|---|---|
| HR | ACDD_JOB_HISTORY | P_EMP_ID | |
| HR | ACDD_JOB_HISTORY | P_START_DATE | |
| HR | ACDD_JOB_HISTORY | P_END_DATE | |
| HR | ACDD_JOB_HISTORY | P_JOB_ID | |
| HR | ACDD_JOB_HISTORY | P_DEPARTMENT_ID | |

0.198 sec/0.002 sec | 5 / 15 | 1-5

**Figure: The procedure object**

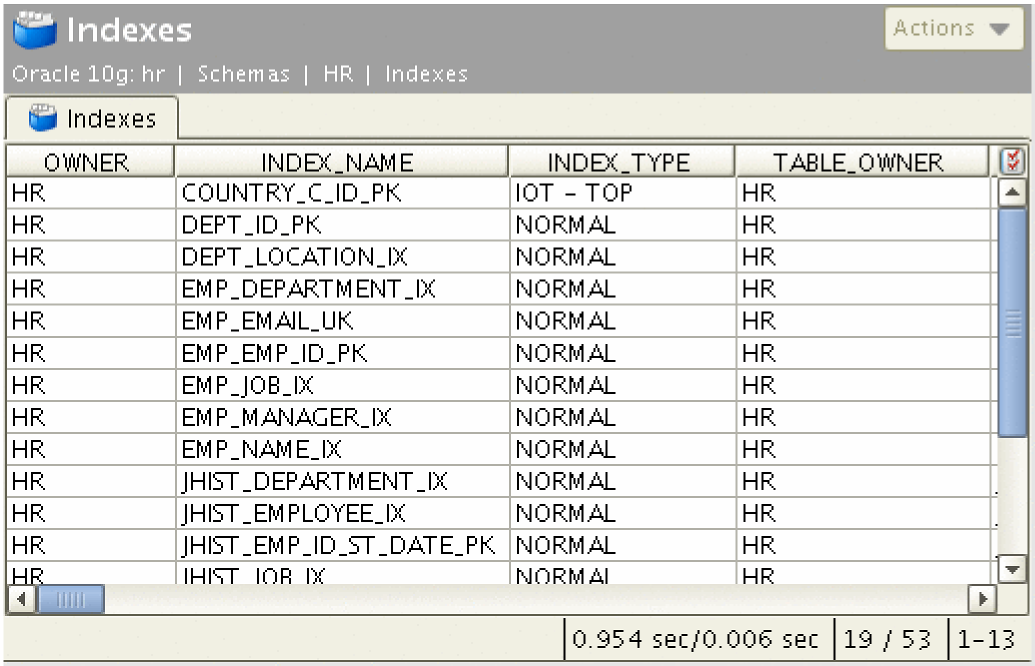
The object view shows a list of column names for the selected procedure.

# Object Views

The object views in the right area of the Database Objects tab shows detailed information about the selected tree object. The object view may contain several object view tabs depending on the current database profile. There are also several representations of a view to better illustrate the information. The following sections explains each of these visual presentation forms.

## Grid

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



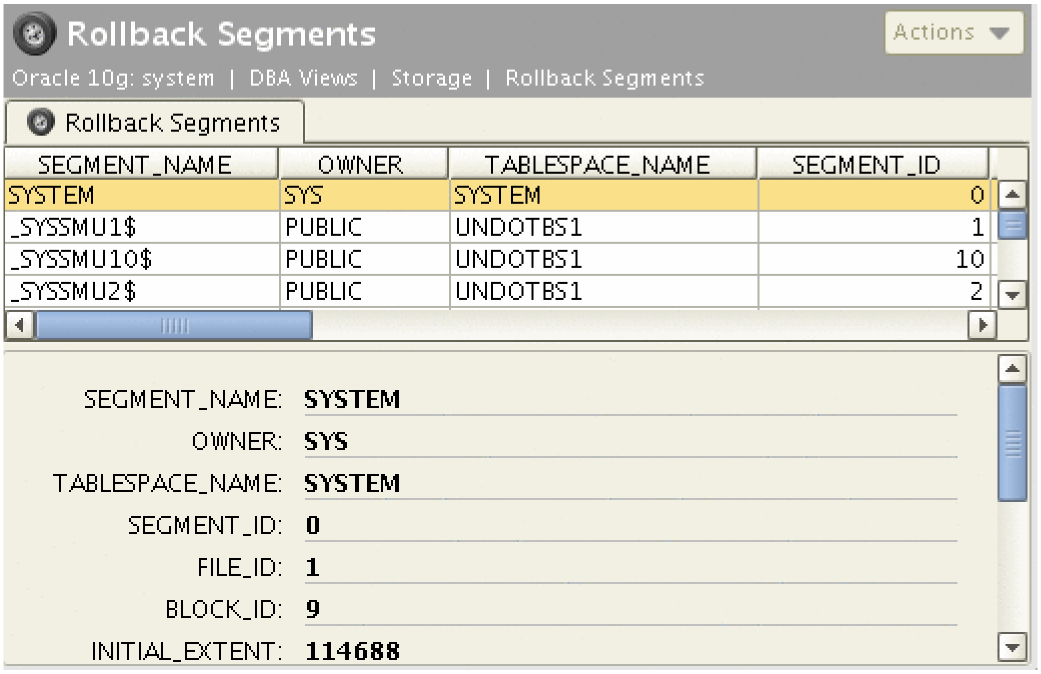
| OWNER | INDEX_NAME | INDEX_TYPE | TABLE_OWNER |
|---|---|---|---|
| HR | COUNTRY_C_ID_PK | IOT - TOP | HR |
| HR | DEPT_ID_PK | NORMAL | HR |
| HR | DEPT_LOCATION_IX | NORMAL | HR |
| HR | EMP_DEPARTMENT_IX | NORMAL | HR |
| HR | EMP_EMAIL_UK | NORMAL | HR |
| HR | EMP_EMP_ID_PK | NORMAL | HR |
| HR | EMP_JOB_IX | NORMAL | HR |
| HR | EMP_MANAGER_IX | NORMAL | HR |
| HR | EMP_NAME_IX | NORMAL | HR |
| HR | JHIST_DEPARTMENT_IX | NORMAL | HR |
| HR | JHIST_EMPLOYEE_IX | NORMAL | HR |
| HR | JHIST_EMP_ID_ST_DATE_PK | NORMAL | HR |
| HR | JHIST_JOB_IX | NORMAL | HR |

0.954 sec/0.006 sec | 19 / 53 | 1-13

**Figure: The Grid view**

## Form

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


Rollback Segments
Actions
Oracle 10g: system | DBA Views | Storage | Rollback Segments
Rollback Segments

| SEGMENT_NAME | OWNER | TABLESPACE_NAME | SEGMENT_ID |
|---|---|---|---|
| SYSTEM | SYS | SYSTEM | 0 |
| _SYSSMU1$ | PUBLIC | UNDOTBS1 | 1 |
| _SYSSMU10$ | PUBLIC | UNDOTBS1 | 10 |
| _SYSSMU2$ | PUBLIC | UNDOTBS1 | 2 |

| | |
|---|---|
| SEGMENT_NAME: | SYSTEM |
| OWNER: | SYS |
| TABLESPACE_NAME: | SYSTEM |
| SEGMENT_ID: | 0 |
| FILE_ID: | 1 |
| BLOCK_ID: | 9 |
| INITIAL_EXTENT: | 114688 |

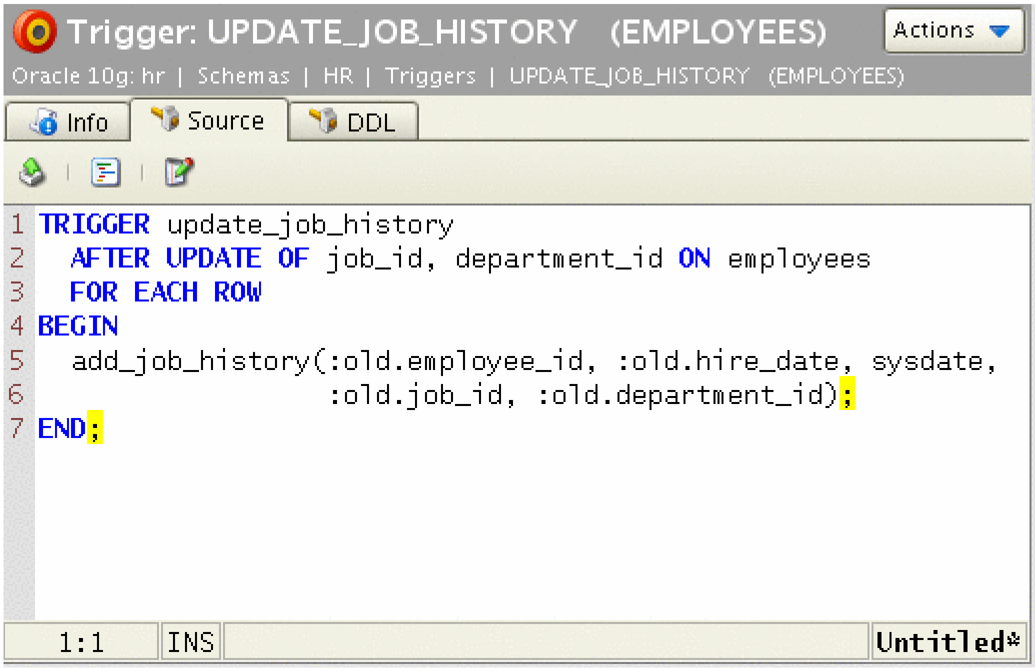
**Figure: The Form view**

If there is only one row in the result will no grid appear but only the form.

## Source

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

- Export data to file
- Wrap long lines
- Copy the data to SQL Commander

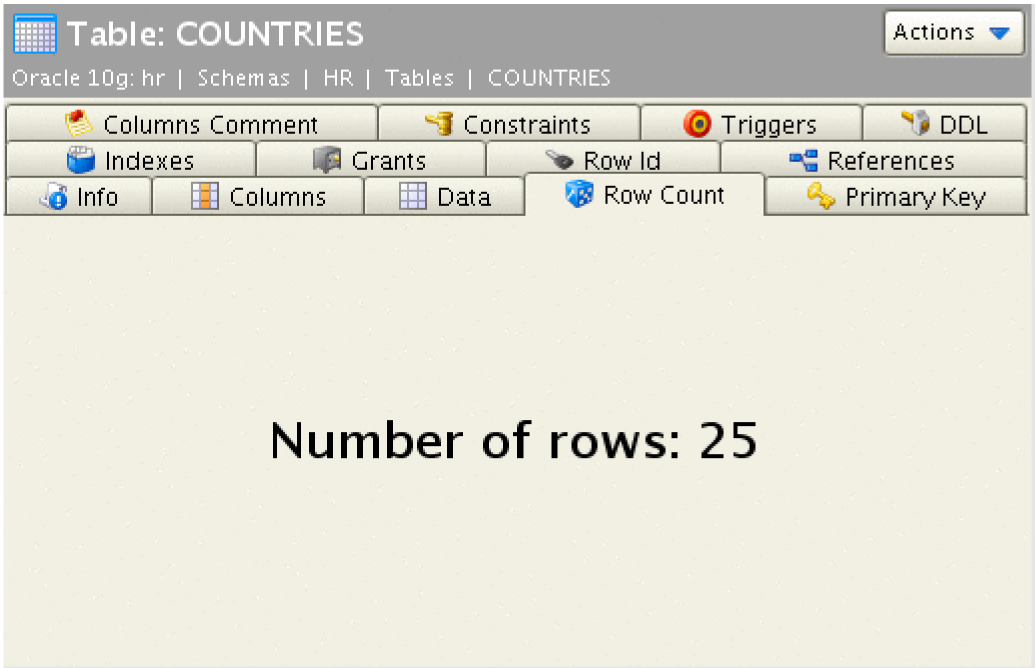

Trigger: UPDATE_JOB_HISTORY (EMPLOYEES)
Actions
Oracle 10g: hr | Schemas | HR | Triggers | UPDATE_JOB_HISTORY (EMPLOYEES)
Info
Source
DDL
1:1
INS
Untitled*

```
TRIGGER update_job_history
  AFTER UPDATE OF job_id, department_id ON employees
  FOR EACH ROW
BEGIN
  add_job_history(:old.employee_id, :old.hire_date, sysdate,
                  :old.job_id, :old.department_id);
END;
```

**Figure: The Source view**

## Table Row Count

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

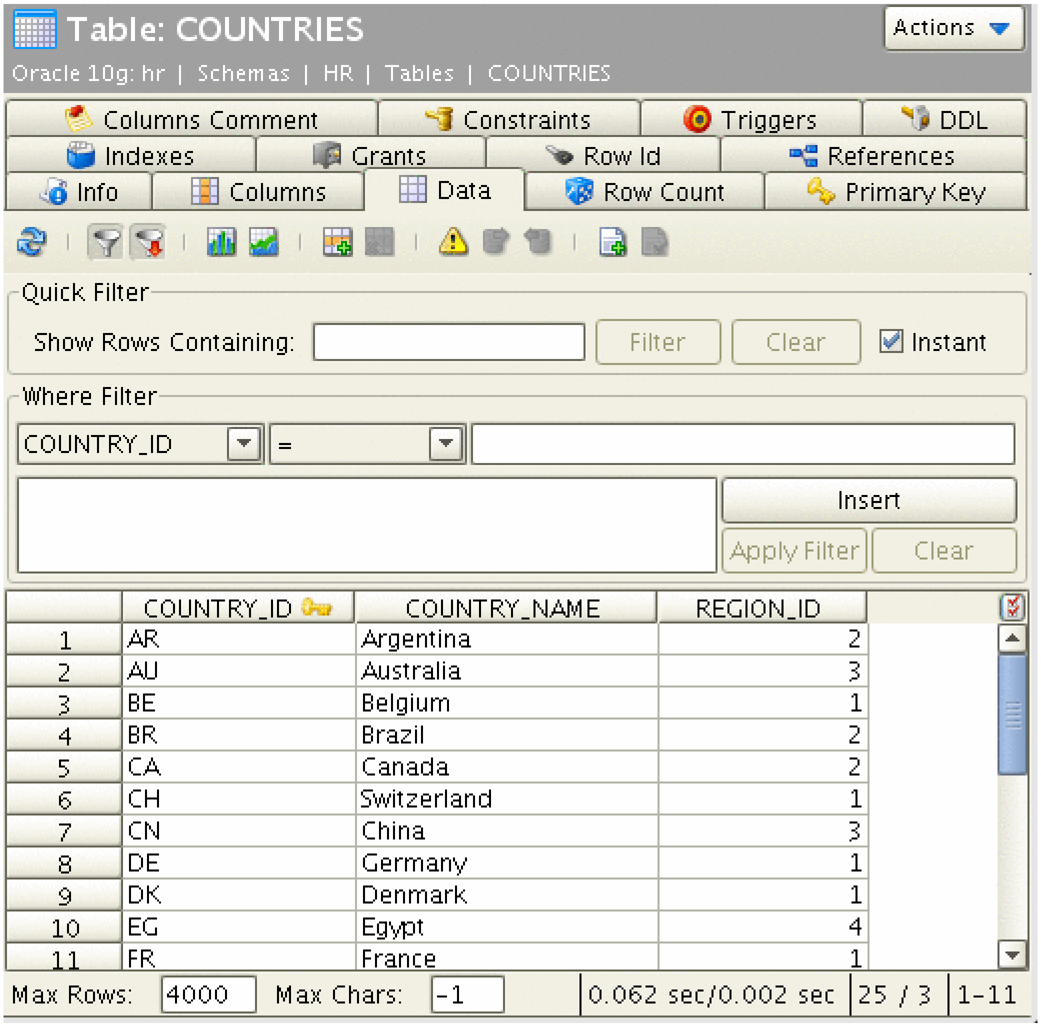


```
Number of rows: 25
```

**Figure: The Row Count view**

## Table Data

The **Data** tab is used to browse the data in the table and to do various data related operations. This view is based on the generic grid but adds a few more visual components to limit the max number of rows, the width of text columns and the collection of data tab specific operations in the right click menu. In addition it is also possible to set a filter that will ensure that only the rows that match the filter will be displayed. The data tab is the place to do edits in DbVisualizer Personal.



Show Rows Containing: ☑ Instant

COUNTRY_ID =

| | 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 |

Max Rows: 4000 Max Chars: -1 0.062 sec/0.002 sec 25 / 3 1-11

**Figure: The Data tab for Table objects**

### Right click menu

The right click menu in the data tab grid menu adds some operations into the standard right click menu. These are primarily used to create SQL statements based on the current selection. Choosing any of these will create the appropriate SQL and then switch the view to the SQL Commander tab. These operations are used to edit table data in the DbVisualizer Free edition since the inline and form based editors are specifically for DbVisualizer Personal. (Information about the standard right click menu operations are available in the Getting Started and General Overview document).

The generated SQL can contain either static values as they appear in the grid or DbVisualizer **variables**. A variable is essentially used as a place holder for a value in an SQL statement. Once the statement is executed DbVisualizer will locate all variables and present them in a dialog. The values for the variables can then be entered or modified and DbVisualizer will in the final SQL replace the variable place holders with the new values.

Variables can be used in any SQL statement and DbVisualizer relies heavily on them. (Read more about variables in the SQL Commander document).

The use of variables in the SQL statements generated by the SQL operations in the right click menu depends on the **Table Data- >Include Variables in SQL** setting in **Tool Properties**. This setting is by default true (include variables) and will result in variables being used in the statement. Disabling the property will result in static SQL in the generated statement.

Here follows an example with the **Include Variables in SQL** setting enabled and then disabled. The SQL is generated when the **select * where** operation is selected based on the selection in the previous figure.

<table>
<tr><th>Include Variables in SQL is enabled</th></tr>
<tr><td>select *<br>from SCOTT.EMP<br>where ENAME = $$ENAME (where)||WARD||String||where ds=10 dt=VARCHAR nullable $$<br>and JOB = $$JOB (where)||SALESMAN||String||where ds=9 dt=VARCHAR nullable $$</td></tr>
<tr><th>Include Variables in SQL is disabled</th></tr>
<tr><td>select *<br>from SCOTT.EMP<br>where ENAME = 'WARD'<br>and JOB = 'SALESMAN'</td></tr>
</table>

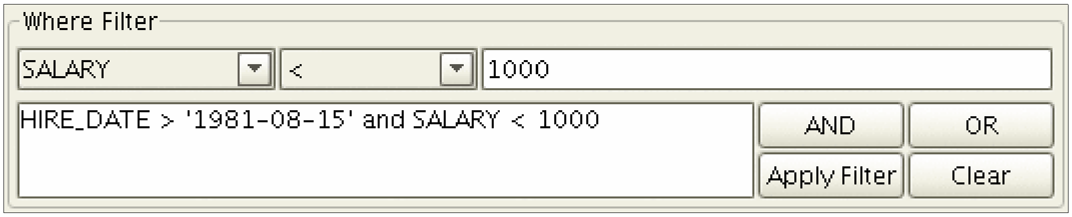
The following lists the generated SQL for each of the operations based on the selection of ENAME = WARD and JOB = SALESMAN.

| Operation | SQL Example |
|---|---|
| **Set Filter for Selection** | ENAME = 'WARD' and<br>JOB = 'SALESMAN' |
| **Script: SELECT ALL** | select *<br>from SCOTT.EMP |
| **Script: SELECT WHERE** | select *<br>from SCOTT.EMP<br>where ENAME = 'WARD'<br>and JOB = 'SALESMAN' |
| **Script: INSERT INTO TABLE** | insert into SCOTT.EMP<br>(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO) |

| | |
|---|---|
| | values (,'','',,'',,,) |
| **Script: INSERT COPY INTO TABLE** | insert into SCOTT.EMP<br>(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)<br>values (7521, 'WARD', 'SALESMAN', 7698, '1981-02-22 00:00:00.0', 1250, 500, 30) |
| **Script: UPDATE WHERE** | update SCOTT.EMP<br>set    EMPNO = 7521,<br>       ENAME = 'WARD',<br>       JOB = 'SALESMAN',<br>       MGR = 7698,<br>       HIREDATE = '1981-02-22 00:00:00.0',<br>       SAL = 1250,<br>       COMM = 500,<br>       DEPTNO = 30<br>where  ENAME = 'WARD'<br>and    JOB = 'SALESMAN' |
| **Script: DELETE WHERE** | delete from SCOTT.EMP<br>where  ENAME = 'WARD'<br>and    JOB = 'SALESMAN' |
| **Script: DROP TABLE** | drop table SCOTT.EMP |

### Where Filter

The filter capability in the Data tab is used to form the where clause that will limit the number of rows in the grid.

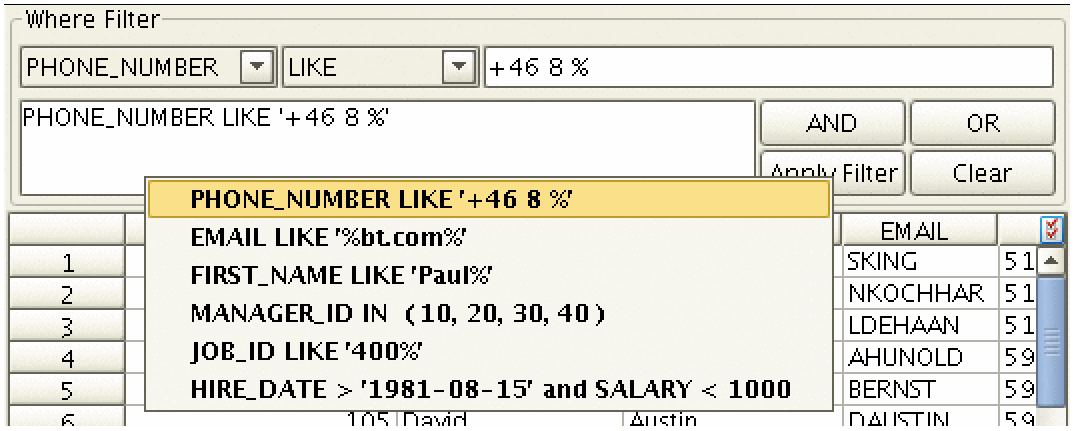


SALARY    <    1000

HIRE_DATE > '1981-08-15' and SALARY < 1000

**Figure: The Data tab filter**

The filter area is composed of two parts. The upper one is used to define the where clause for a single column. The available columns and operators are selected from two lists. The value of the column is specified in a text field. You can use **Ctrl- Enter** while editing the value to force a reload of the grid based on that single filter. The lower part displays the complete filter and the buttons are used to control whether the newly entered filter will be **AND**'ed or **OR**'ed with the complete filter. The buttons change appearance based on whether there is any filter or not. While in the complete filter you can use **Ctrl- Enter** to force a reload based on the complete filter. The right click menu lists the last 20 filters that have been applied to the grid.


Where Filter

PHONE_NUMBER LIKE +46 8 %

PHONE_NUMBER LIKE '+46 8 %'

**Figure: The filter history right click menu**

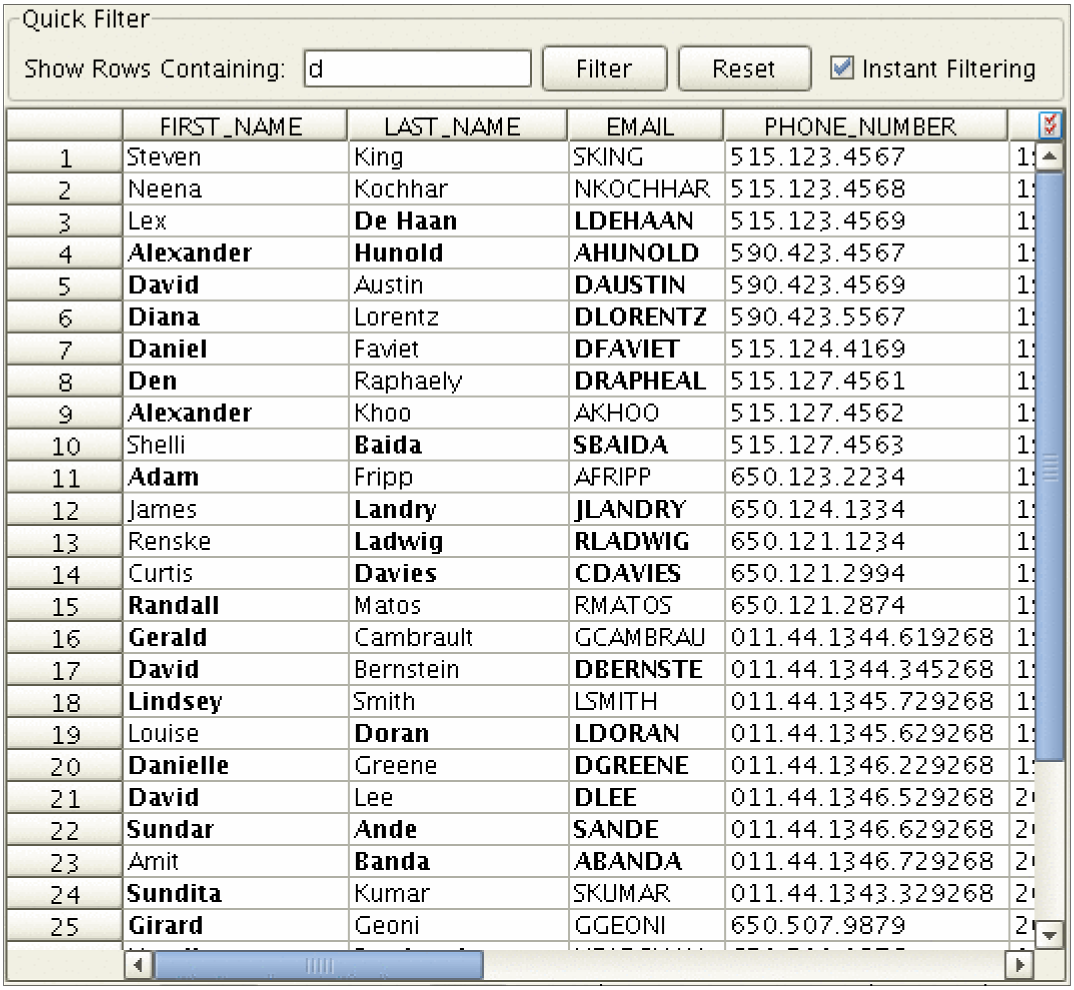
To reset the use of the filter select the **Reload** operation in the data tab tool bar.

(The visible state of the filter pane is controlled using the Filter toggle button in the data tab tool bar).

## Quick Filter

The quick filter acts on the data that is already in the grid as opposed of a **where filter** which is used to limit the number of rows fetched from the database. Quick filter is convenient as it is used to quickly list only those rows that match the entered search string.

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



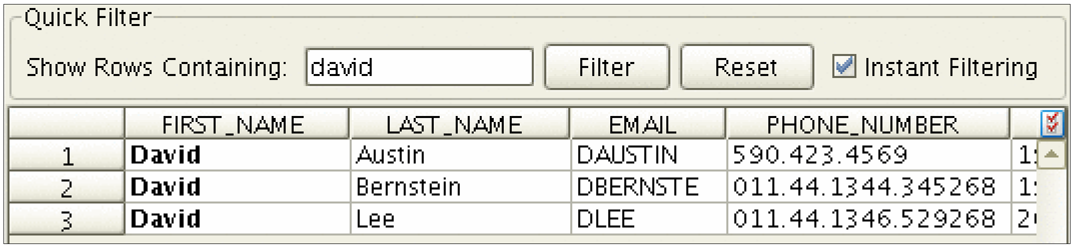
Quick Filter

Show Rows Containing: d

| | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER |
|---|---|---|---|---|
| 1 | Steven | King | SKING | 515.123.4567 |
| 2 | Neena | Kochhar | NKOCHHAR | 515.123.4568 |
| 3 | Lex | **De Haan** | **LDEHAAN** | 515.123.4569 |
| 4 | **Alexander** | **Hunold** | **AHUNOLD** | 590.423.4567 |
| 5 | **David** | Austin | **DAUSTIN** | 590.423.4569 |
| 6 | **Diana** | Lorentz | **DLORENTZ** | 590.423.5567 |
| 7 | **Daniel** | Faviet | **DFAVIET** | 515.124.4169 |
| 8 | **Den** | Raphaely | **DRAPHEAL** | 515.127.4561 |
| 9 | **Alexander** | Khoo | AKHOO | 515.127.4562 |
| 10 | Shelli | **Baida** | **SBAIDA** | 515.127.4563 |
| 11 | **Adam** | Fripp | AFRIPP | 650.123.2234 |
| 12 | James | **Landry** | **JLANDRY** | 650.124.1334 |
| 13 | Renske | **Ladwig** | **RLADWIG** | 650.121.1234 |
| 14 | Curtis | **Davies** | **CDAVIES** | 650.121.2994 |
| 15 | **Randall** | Matos | RMATOS | 650.121.2874 |
| 16 | **Gerald** | Cambrault | GCAMBRAU | 011.44.1344.619268 |
| 17 | **David** | Bernstein | **DBERNSTE** | 011.44.1344.345268 |
| 18 | **Lindsey** | Smith | LSMITH | 011.44.1345.729268 |
| 19 | Louise | **Doran** | **LDORAN** | 011.44.1345.629268 |
| 20 | **Danielle** | Greene | **DGREENE** | 011.44.1346.229268 |
| 21 | **David** | Lee | **DLEE** | 011.44.1346.529268 |
| 22 | **Sundar** | **Ande** | **SANDE** | 011.44.1346.629268 |
| 23 | Amit | **Banda** | **ABANDA** | 011.44.1346.729268 |
| 24 | **Sundita** | Kumar | SKUMAR | 011.44.1343.329268 |
| 25 | **Girard** | Geoni | GGEONI | 650.507.9879 |

**Figure: The filter history right click menu**

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



Quick Filter

Show Rows Containing: david

| | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER |
|---|---|---|---|---|
| 1 | **David** | Austin | DAUSTIN | 590.423.4569 |
| 2 | **David** | Bernstein | DBERNSTE | 011.44.1344.345268 |
| 3 | **David** | Lee | DLEE | 011.44.1346.529268 |

**Figure: The filter history right click menu**

When the **Instant Filtering** control is enabled then is the grid filtered while entering new characters. Having a lot of rows in the grid may slow down the search if having Instant Filtering enabled. If it is disabled then you must press the **Filter** button in order to apply the filter.

## Monitor row count

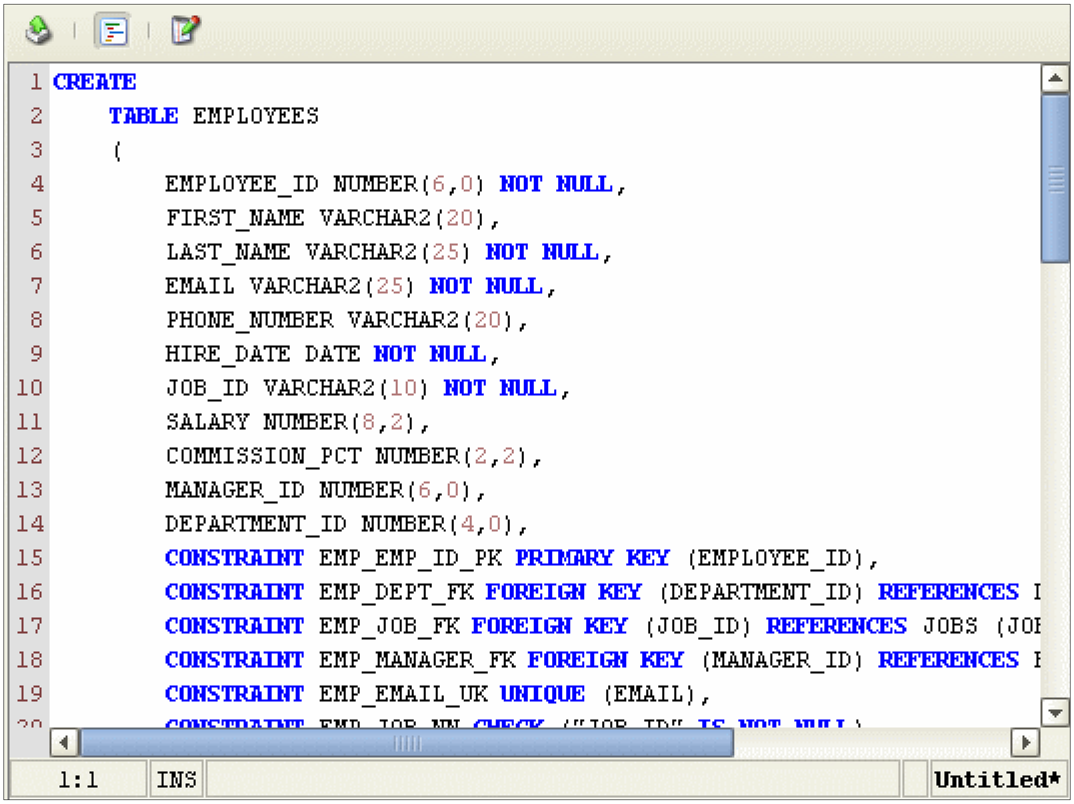
Read more about the **Monitor Row Count** and **Monitor Row Count Difference** in Monitor and Charts.

### Editing

Read about data editing in Edit Table Data

## DDL Viewer

DDL Viewer tabs appear only for **Table** and **View** objects and for databases that have specialized database profiles.

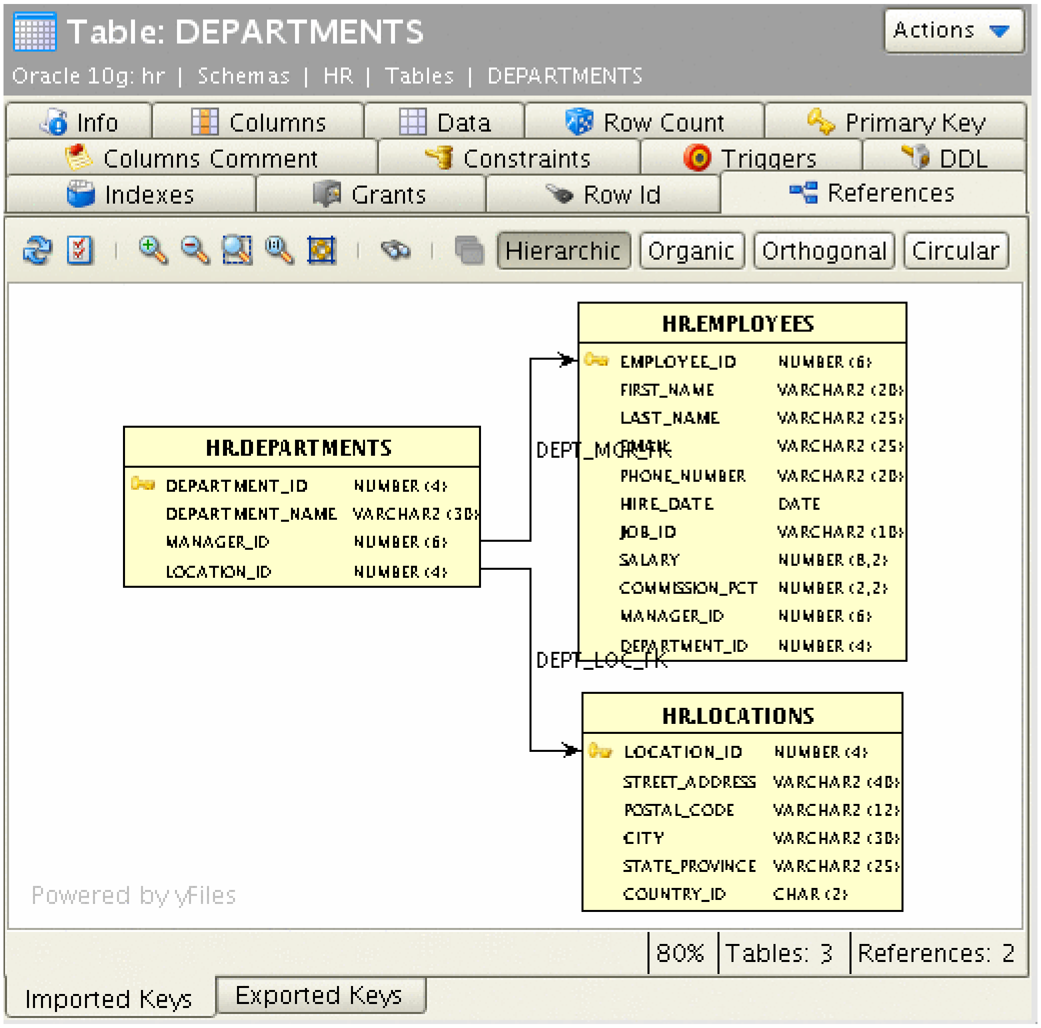


```
CREATE
    TABLE EMPLOYEES
    (
        EMPLOYEE_ID NUMBER(6,0) NOT NULL,
        FIRST_NAME VARCHAR2(20),
        LAST_NAME VARCHAR2(25) NOT NULL,
        EMAIL VARCHAR2(25) NOT NULL,
        PHONE_NUMBER VARCHAR2(20),
        HIRE_DATE DATE NOT NULL,
        JOB_ID VARCHAR2(10) NOT NULL,
        SALARY NUMBER(8,2),
        COMMISSION_PCT NUMBER(2,2),
        MANAGER_ID NUMBER(6,0),
        DEPARTMENT_ID NUMBER(4,0),
        CONSTRAINT EMP_EMP_ID_PK PRIMARY KEY (EMPLOYEE_ID),
        CONSTRAINT EMP_DEPT_FK FOREIGN KEY (DEPARTMENT_ID) REFERENCES I
        CONSTRAINT EMP_JOB_FK FOREIGN KEY (JOB_ID) REFERENCES JOBS (JOI
        CONSTRAINT EMP_MANAGER_FK FOREIGN KEY (MANAGER_ID) REFERENCES I
        CONSTRAINT EMP_EMAIL_UK UNIQUE (EMAIL),
        CONSTRAINT EMP_JOB_NN CHECK ("JOB_ID" IS NOT NULL)
```

Figure: The DDL viewer for a table

## References

The **references** tab is used to visualize the references from the table and what tables reference it. Use the sub tabs at the bottom of the display to show either view. The following shows the references from the table.



| HR.DEPARTMENTS | |
|---|---|
| DEPARTMENT_ID | NUMBER (4) |
| DEPARTMENT_NAME | VARCHAR2 (30) |
| MANAGER_ID | NUMBER (6) |
| LOCATION_ID | NUMBER (4) |

| HR.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) |

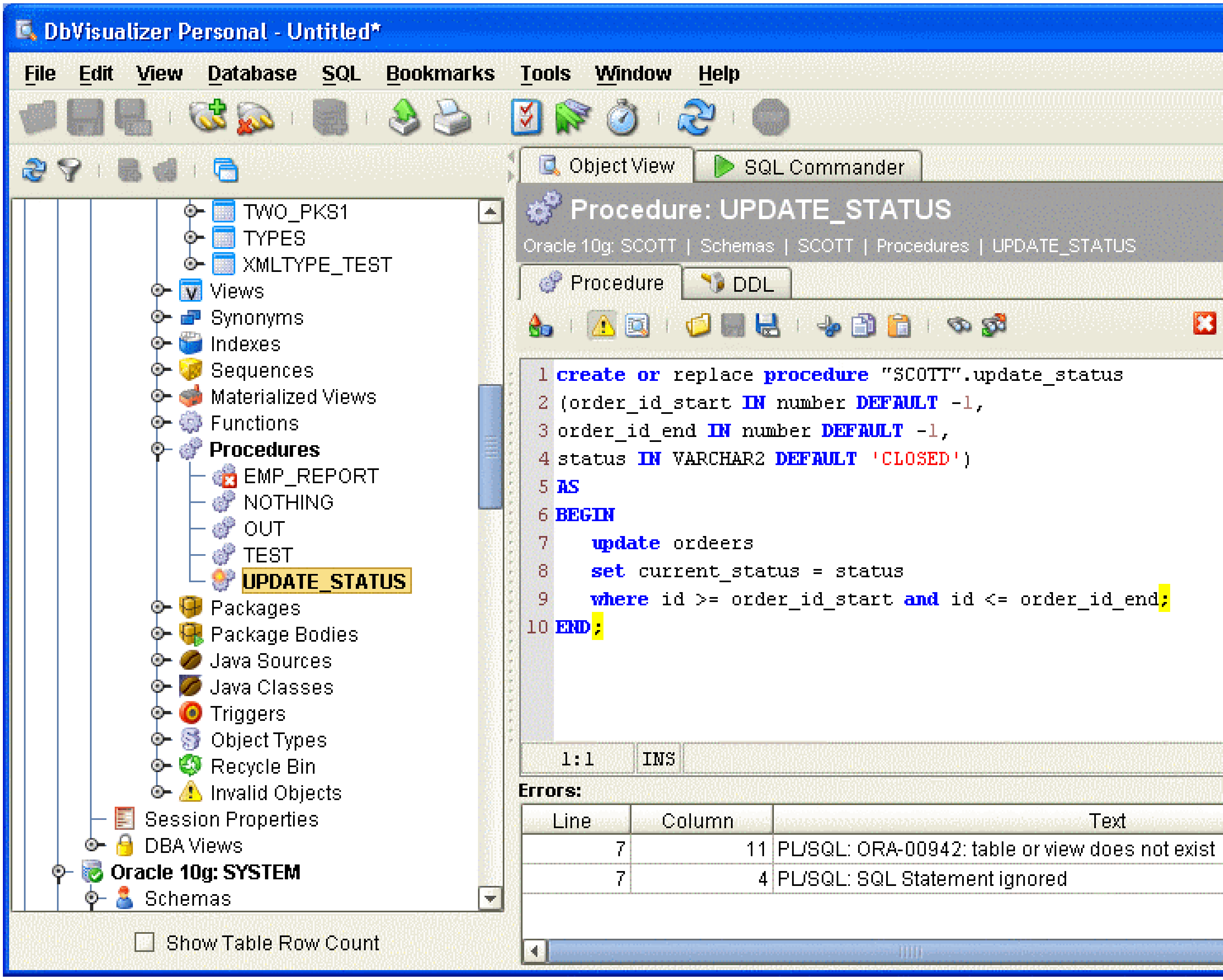
| HR.LOCATIONS | |
|---|---|
| LOCATION_ID | NUMBER (4) |
| STREET_ADDRESS | VARCHAR2 (40) |
| POSTAL_CODE | VARCHAR2 (12) |
| CITY | VARCHAR2 (30) |
| STATE_PROVINCE | VARCHAR2 (25) |
| COUNTRY_ID | CHAR (2) |

**Figure: The references graph showing imported keys for a table**

## Procedure Editor

The procedure editor is used to browse, edit and compile procedures, functions, packages and package bodies. The procedure contains the source editor and options to display parameter info and compilation error list. If error(s) occur during compilation then click the error and the related row in the source editor will be highlighted.

More information is can be read in the Procedure Editor document.



```
create or replace procedure "SCOTT".update_status
(order_id_start IN number DEFAULT -1,
order_id_end IN number DEFAULT -1,
status IN VARCHAR2 DEFAULT 'CLOSED')
AS
BEGIN
   update ordeers
   set current_status = status
   where id >= order_id_start and id <= order_id_end;
END;
```

| Line | Column | Text |
|---|---|---|
| 7 | 11 | PL/SQL: ORA-00942: table or view does not exist |
| 7 | 4 | PL/SQL: SQL Statement ignored |

**Figure: The procedure editor for functions, procedures, packages and package bodies**

# SQL Commander

## Introduction

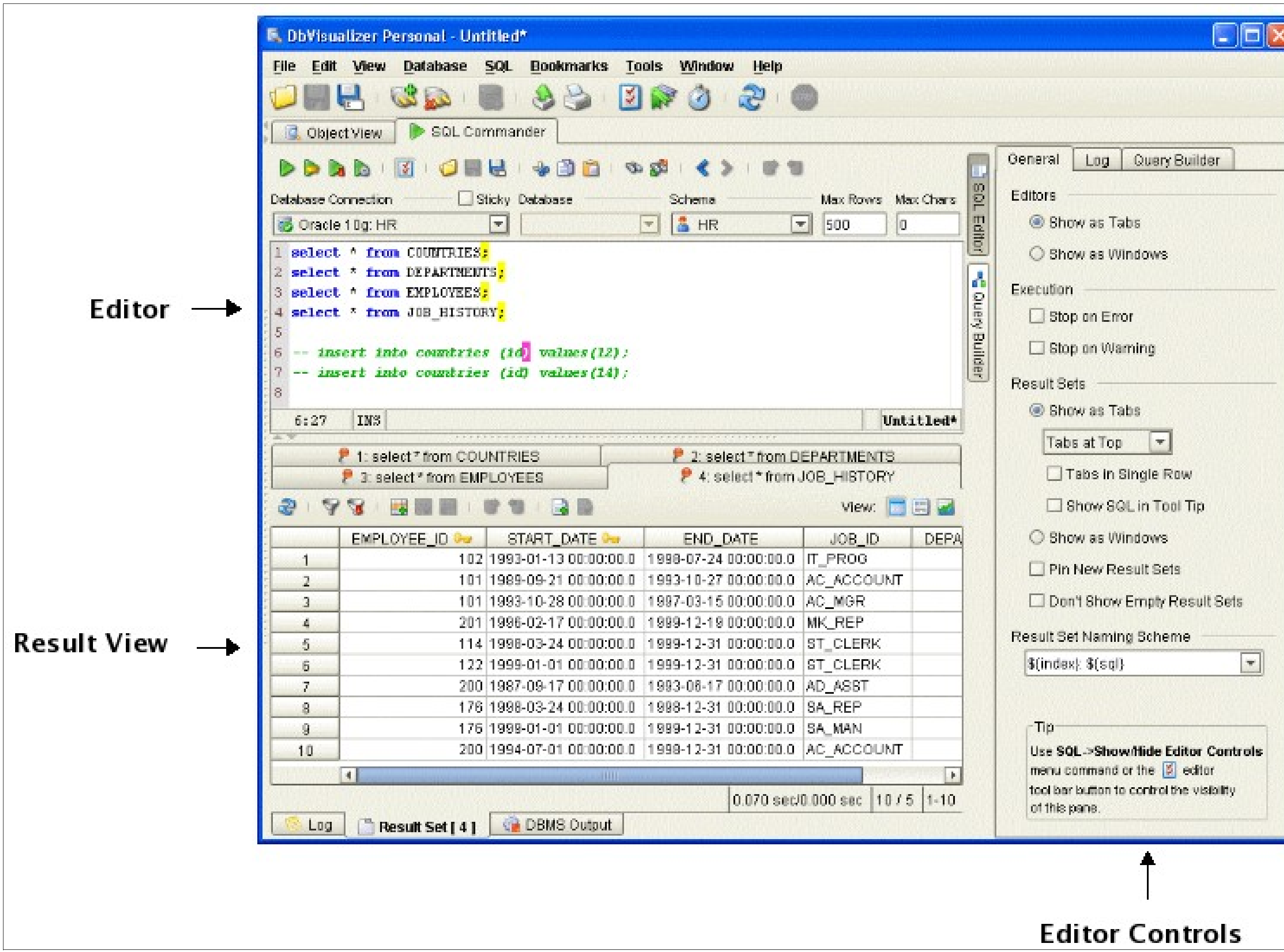
The SQL Commander is used to edit, format and execute SQL statements or SQL scripts. Multiple editors may be open at the same time each controlling its own SQL log and result sets. Result sets can be displayed in grid, text or chart formats.

The SQL Commander supports the following features:

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

Database specific support:

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


Editor
Result View
Editor Controls
DbVisualizer Personal - Untitled*
File Edit View Database SQL Bookmarks Tools Window Help
Object View
SQL Commander
SQL Editor
Query Builder
General
Log
Query Builder
1: select * from COUNTRIES
2: select * from DEPARTMENTS
3: select * from EMPLOYEES
4: select * from JOB_HISTORY
View:
Log
Result Set [ 4 ]
DBMS Output

Database Connection: Oracle 10g: HR □ Sticky Database: Schema: HR Max Rows: 500 Max Chars: 0

```
select * from COUNTRIES;
select * from DEPARTMENTS;
select * from EMPLOYEES;
select * from JOB_HISTORY;

-- insert into countries (id) values(12);
-- insert into countries (id) values(14);

```

6:27 INS Untitled*

| | EMPLOYEE_ID | START_DATE | END_DATE | JOB_ID | DEPA |
|---|---|---|---|---|---|
| 1 | 102 | 1993-01-13 00:00:00.0 | 1998-07-24 00:00:00.0 | IT_PROG | |
| 2 | 101 | 1989-09-21 00:00:00.0 | 1993-10-27 00:00:00.0 | AC_ACCOUNT | |
| 3 | 101 | 1993-10-28 00:00:00.0 | 1997-03-15 00:00:00.0 | AC_MGR | |
| 4 | 201 | 1996-02-17 00:00:00.0 | 1999-12-19 00:00:00.0 | MK_REP | |
| 5 | 114 | 1998-03-24 00:00:00.0 | 1999-12-31 00:00:00.0 | ST_CLERK | |
| 6 | 122 | 1999-01-01 00:00:00.0 | 1999-12-31 00:00:00.0 | ST_CLERK | |
| 7 | 200 | 1987-09-17 00:00:00.0 | 1993-06-17 00:00:00.0 | AD_ASST | |
| 8 | 176 | 1998-03-24 00:00:00.0 | 1998-12-31 00:00:00.0 | SA_REP | |
| 9 | 176 | 1999-01-01 00:00:00.0 | 1999-12-31 00:00:00.0 | SA_MAN | |
| 10 | 200 | 1994-07-01 00:00:00.0 | 1998-12-31 00:00:00.0 | AC_ACCOUNT | |

0.070 sec/0.000 sec 10 / 5 1-10

Editors
● Show as Tabs
○ Show as Windows

Execution
□ Stop on Error
□ Stop on Warning

Result Sets
● Show as Tabs
Tabs at Top
□ Tabs in Single Row
□ Show SQL in Tool Tip
○ Show as Windows
□ Pin New Result Sets
□ Don't Show Empty Result Sets

Result Set Naming Scheme
$(index): $(sql)

Tip
Use **SQL->Show/Hide Editor Controls** menu command or the editor tool bar button to control the visibility of this pane.

**Figure: SQL Commander overview**

The figure shows the editing area and controls above and the output view in the lower part of the screen. The following sections give a detailed explanation of all features and controls in the SQL Commander.

# Editor

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

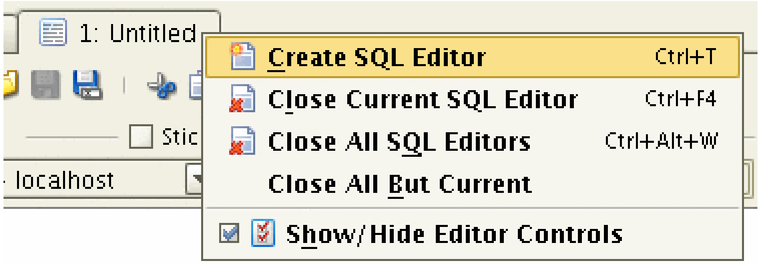

1: Untitled
Stic
localhost
Create SQL Editor Ctrl+T
Close Current SQL Editor Ctrl+F4
Close All SQL Editors Ctrl+Alt+W
Close All But Current
☑ Show/Hide Editor Controls

**Figure: Editor tab menu**

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

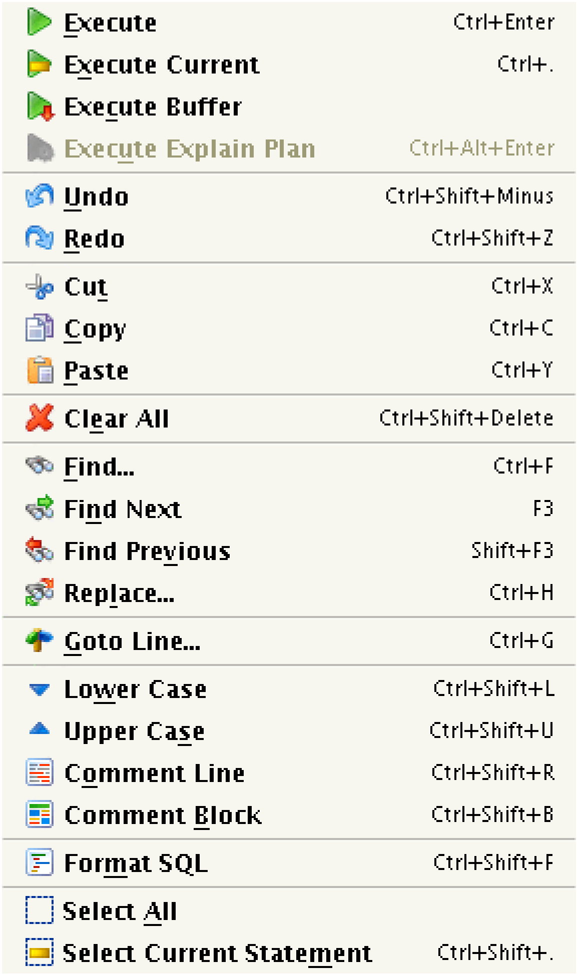
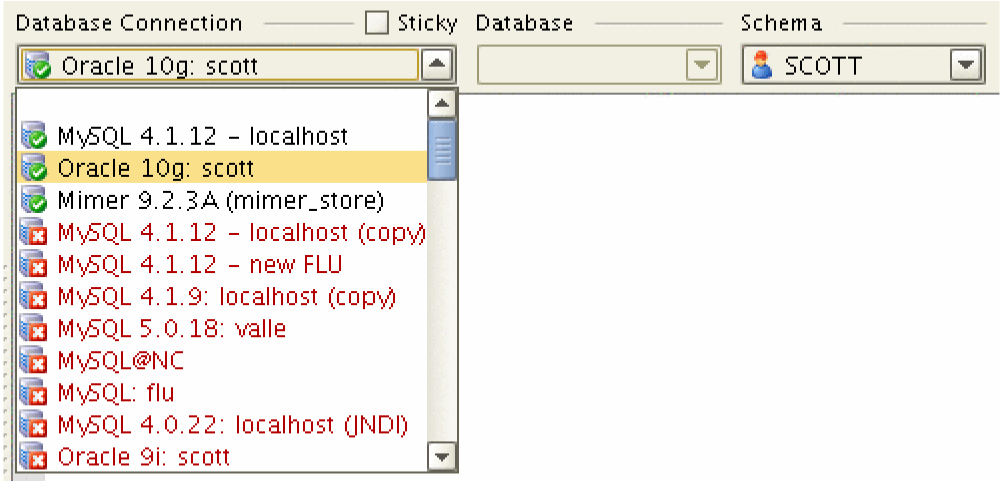


**Figure: The SQL editor right click menu**

The SQL editor is also used in the Bookmark Editor and when editing CLOB's in the form editor.

## Database Connection, Catalog and Schema

The Database Connection and Database (or Catalog) lists above the editor specify which connection and database the SQL in the editor will be executed by. The list of connections shows all connections as they are ordered in the Database Objects tree with the exception that all currently active connections are listed first.



Database Connection □ Sticky Database Schema

Oracle 10g: scott

SCOTT

MySQL 4.1.12 - localhost
Oracle 10g: scott
Mimer 9.2.3A (mimer_store)
MySQL 4.1.12 - localhost (copy)
MySQL 4.1.12 - new FLU
MySQL 4.1.9: localhost (copy)
MySQL 5.0.18: valle
MySQL@NC
MySQL: flu
MySQL 4.0.22: localhost (JNDI)
Oracle 9i: scott

**Figure: Database Connection, Database and Schema lists**

The **Sticky** box above the Database Connection list specifies when enabled that the current connection selection will not change automatically when passing SQL statements from other parts of DbVisualizer. One example is passing an SQL bookmark from the Bookmark Editor. Consider an SQL bookmark defined for database connection "**ProdDB**". If the Sticky setting is disabled the database connection will automatically be changed to ProdDB. If however the Sticky setting is enabled then the current setting of database connection will be unchanged. The Sticky setting is per SQL editor instance.

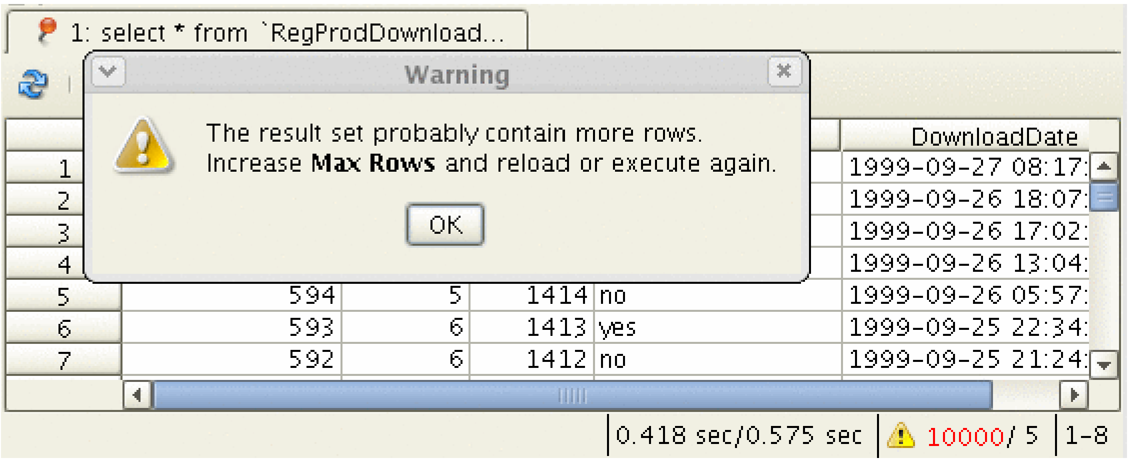
The **Database list** (or Catalog) is used to set what catalog in the connection will be the target for the execution. In the event of catalogs not being supported by the database connection the header will indicate this with **No Catalogs for the Database Connection**.

The **Schema list** is used only to help the auto completion feature to limit what tables to show in the completion pop up. It does not define that the actual SQL should be executed in the selected schema.

## Limiting Result Set size (Max Rows/Chars)

The **Max Rows** field is used to control how many rows that DbVisualizer will fetch for each result set. if there are more rows available then presented in the result set you will see a warning indicator in the grid status bar.

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



1: select * from `RegProdDownload...

Warning

The result set probably contain more rows.
Increase **Max Rows** and reload or execute again.

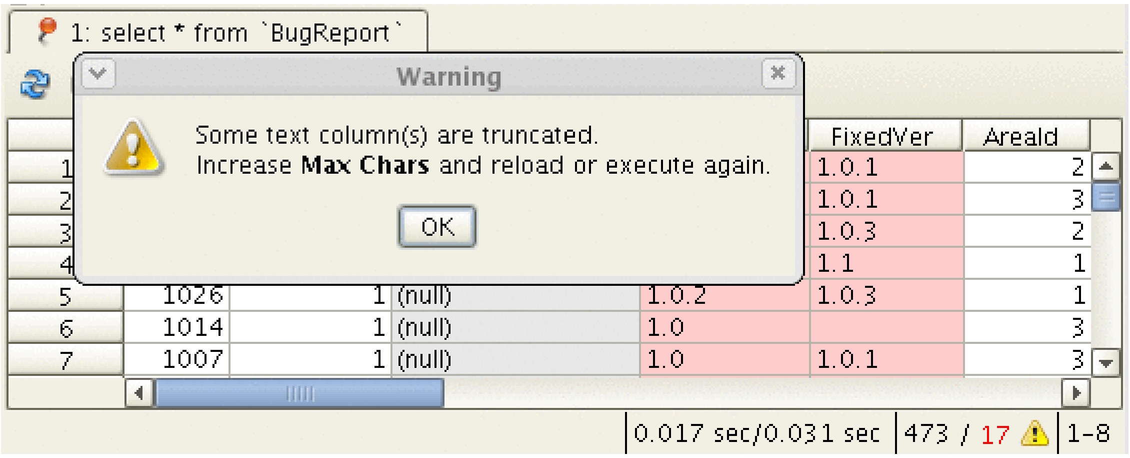
| | | | | | DownloadDate |
|---|---|---|---|---|---|
| 1 | | | | | 1999-09-27 08:17: |
| 2 | | | | | 1999-09-26 18:07: |
| 3 | | | | | 1999-09-26 17:02: |
| 4 | | | | | 1999-09-26 13:04: |
| 5 | 594 | 5 | 1414 | no | 1999-09-26 05:57: |
| 6 | 593 | 6 | 1413 | yes | 1999-09-25 22:34: |
| 7 | 592 | 6 | 1412 | no | 1999-09-25 21:24: |

0.418 sec/0.575 sec | 10000/ 5 | 1-8

**Figure: Max Rows exceeded warning**

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

Setting **Max Chars** limits the number of characters that are presented for text data. Columns that contain more characters then the specified Max Chars shows with red background color to highlight that it is truncated.



1: select * from `BugReport`

Warning

Some text column(s) are truncated.
Increase **Max Chars** and reload or execute again.

| | | | | | FixedVer | AreaId |
|---|---|---|---|---|---|---|
| 1 | | | | | 1.0.1 | 2 |
| 2 | | | | | 1.0.1 | 3 |
| 3 | | | | | 1.0.3 | 2 |
| 4 | | | | | 1.1 | 1 |
| 5 | 1026 | 1 | (null) | 1.0.2 | 1.0.3 | 1 |
| 6 | 1014 | 1 | (null) | 1.0 | | 3 |
| 7 | 1007 | 1 | (null) | 1.0 | 1.0.1 | 3 |

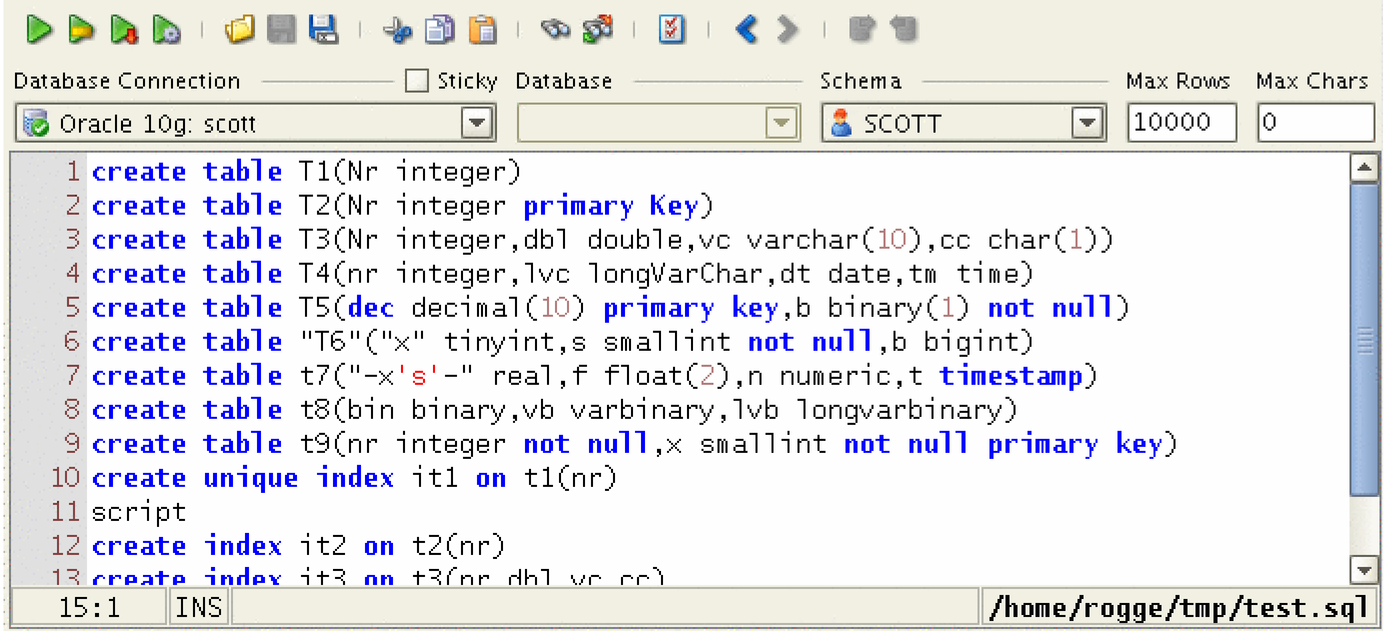
0.017 sec/0.031 sec | 473 / 17 | 1-8

**Figure: Max Chars exceeded warning**

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

## Load from and save to file

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



Database Connection: Oracle 10g: scott

□ Sticky

Database:

Schema: SCOTT

Max Rows: 10000

Max Chars: 0

```
create table T1(Nr integer)
create table T2(Nr integer primary Key)
create table T3(Nr integer,dbl double,vc varchar(10),cc char(1))
create table T4(nr integer,lvc longVarChar,dt date,tm time)
create table T5(dec decimal(10) primary key,b binary(1) not null)
create table "T6"("x" tinyint,s smallint not null,b bigint)
create table t7("-x's'-" real,f float(2),n numeric,t timestamp)
create table t8(bin binary,vb varbinary,lvb longvarbinary)
create table t9(nr integer not null,x smallint not null primary key)
create unique index it1 on t1(nr)
script
create index it2 on t2(nr)
create index it3 on t3(nr,dbl,vc,cc)
```

15:1 INS /home/rogge/tmp/test.sql

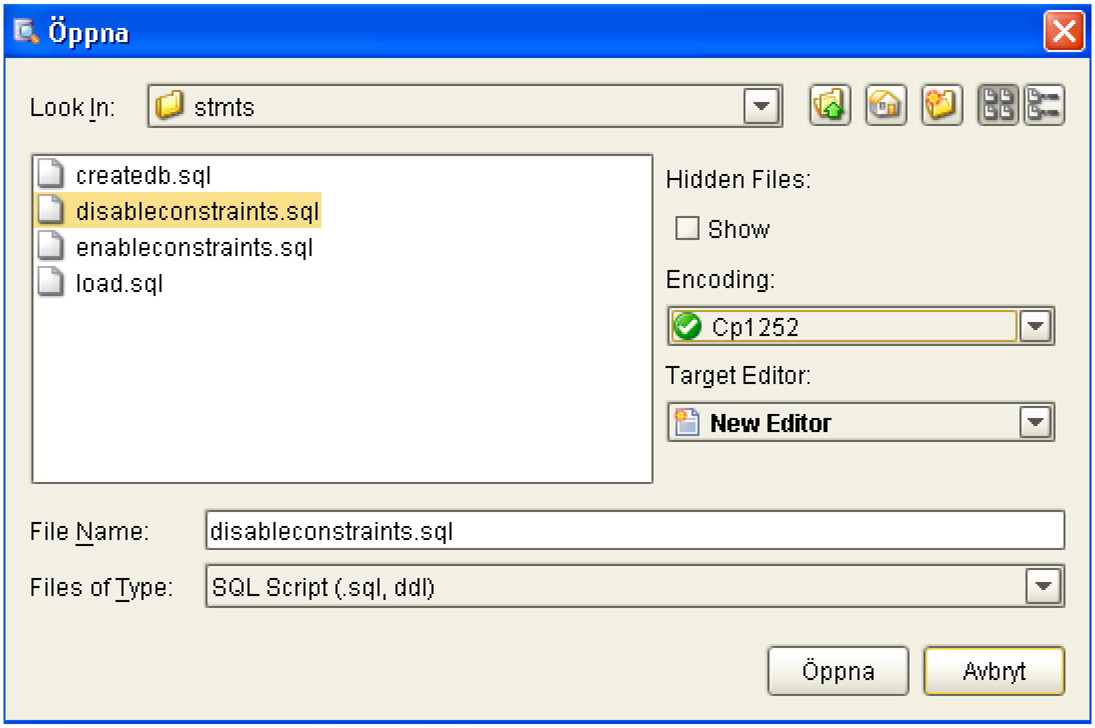
**Figure: Loading a file into the SQL Commander**

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

DbVisualizer will ask at exit if there are any pending edits that need to be saved.

### Load Recent

The File->Load Recent sub menu lists the recently loaded files. You may choose an entry and that file will be current in the file chooser. The file chooser allows setting what target SQL editor the file should be loaded in:



Öppna

Look In: stmts

createdb.sql
disableconstraints.sql
enableconstraints.sql
load.sql

Hidden Files: □ Show

Encoding: Cp1252

Target Editor: New Editor

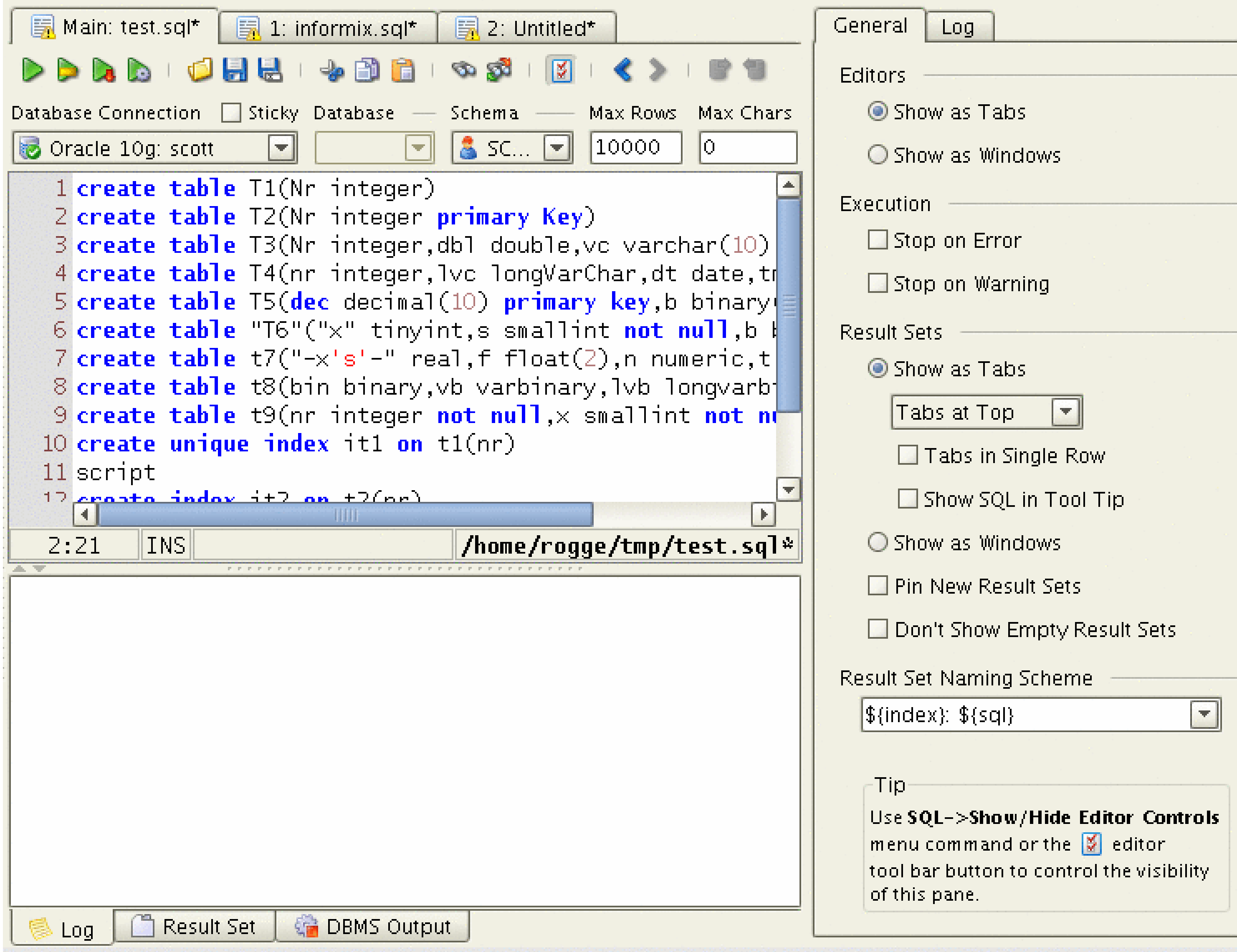
File Name: disableconstraints.sql

Files of Type: SQL Script (.sql, ddl)

**Figure: File Chooser**

## Editor Preferences

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



Database Connection: Oracle 10g: scott  □ Sticky  Database:  Schema: SC...  Max Rows: 10000  Max Chars: 0

```
create table T1(Nr integer)
create table T2(Nr integer primary Key)
create table T3(Nr integer,dbl double,vc varchar(10)
create table T4(nr integer,lvc longVarChar,dt date,tr
create table T5(dec decimal(10) primary key,b binary
create table "T6"("x" tinyint,s smallint not null,b b
create table t7("-x's'-" real,f float(2),n numeric,t
create table t8(bin binary,vb varbinary,lvb longvarb
create table t9(nr integer not null,x smallint not nu
create unique index it1 on t1(nr)
script
create index it2 on t2(nr)
```

2:21 INS /home/rogge/tmp/test.sql*

Editors
- ● Show as Tabs
- ○ Show as Windows

Execution
- □ Stop on Error
- □ Stop on Warning

Result Sets
- ● Show as Tabs
  - Tabs at Top
  - □ Tabs in Single Row
  - □ Show SQL in Tool Tip
- ○ Show as Windows
- □ Pin New Result Sets
- □ Don't Show Empty Result Sets

Result Set Naming Scheme: ${index}: ${sql}

Tip
Use **SQL->Show/Hide Editor Controls** menu command or the editor tool bar button to control the visibility of this pane.

**Figure: Editor preferences pane**

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

**Tip:** The **Result Set Naming Scheme** may include HTML code typically used to change the style of the elements.
Example: `<html>${index}: ${sql} <b>(${rows})</b></html>`

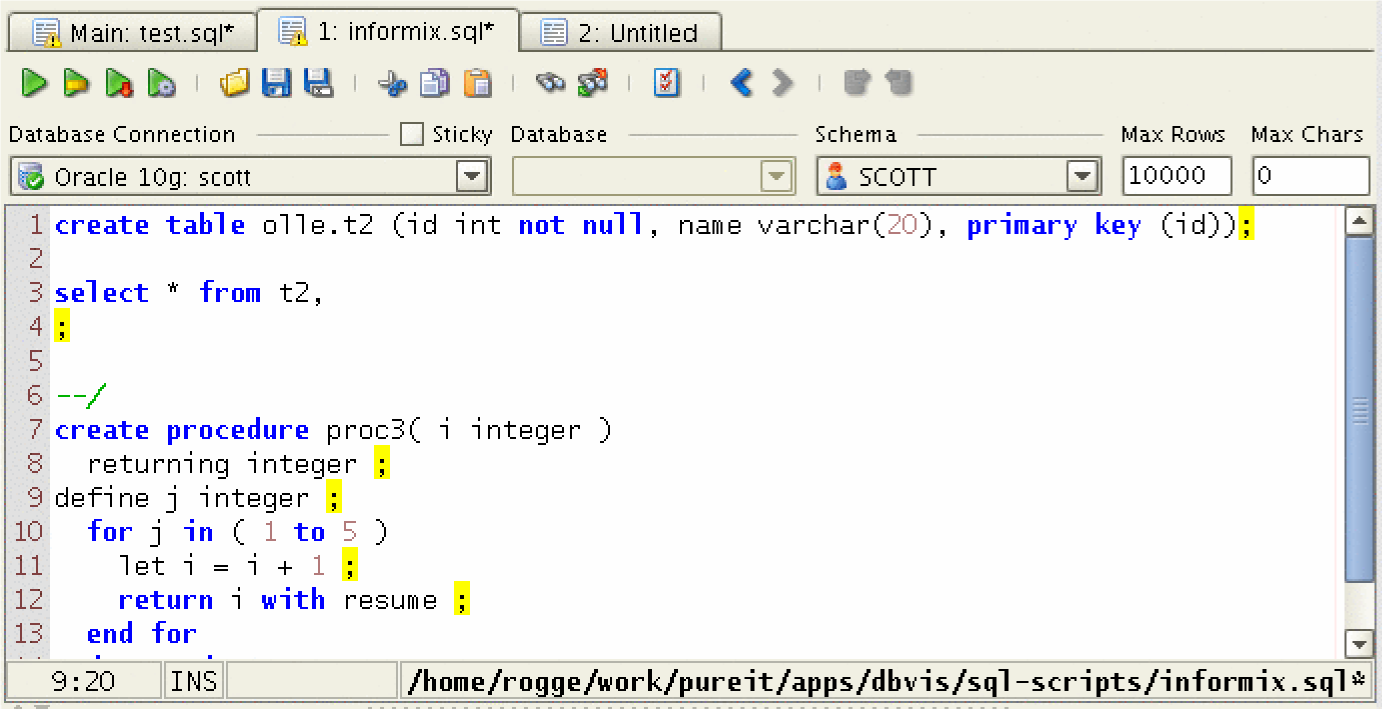
## Multiple editors

Multiple SQL editors can be created with the **File->Create SQL Editor** main menu operation. Editors can be organized as tabs or internal windows using the **View** buttons. There is always one default editor named **Main Editor**. This editor is used when passing SQL bookmarks from the Bookmarks Editor or when issuing requests from other parts of DbVisualizer that activate the SQL Commander. To remove all but the Main Editor select the **File->Close all SQL Editors** menu operation.

The following figures show 3 editors organized in the tabs style and the windows style

### Tabs style

The SQL editors in the figure below show the Main: test.sql, 1: informix.sql and 2: Untitled. A file has been loaded into Editor 1 and the label shows the file name and indicates with an asterisk if the content in the editor has been modified. Remove an editor by choosing the **Close** operation in the right click menu while over the tab header.



Database Connection: Oracle 10g: scott
□ Sticky
Database:
Schema: SCOTT
Max Rows: 10000
Max Chars: 0

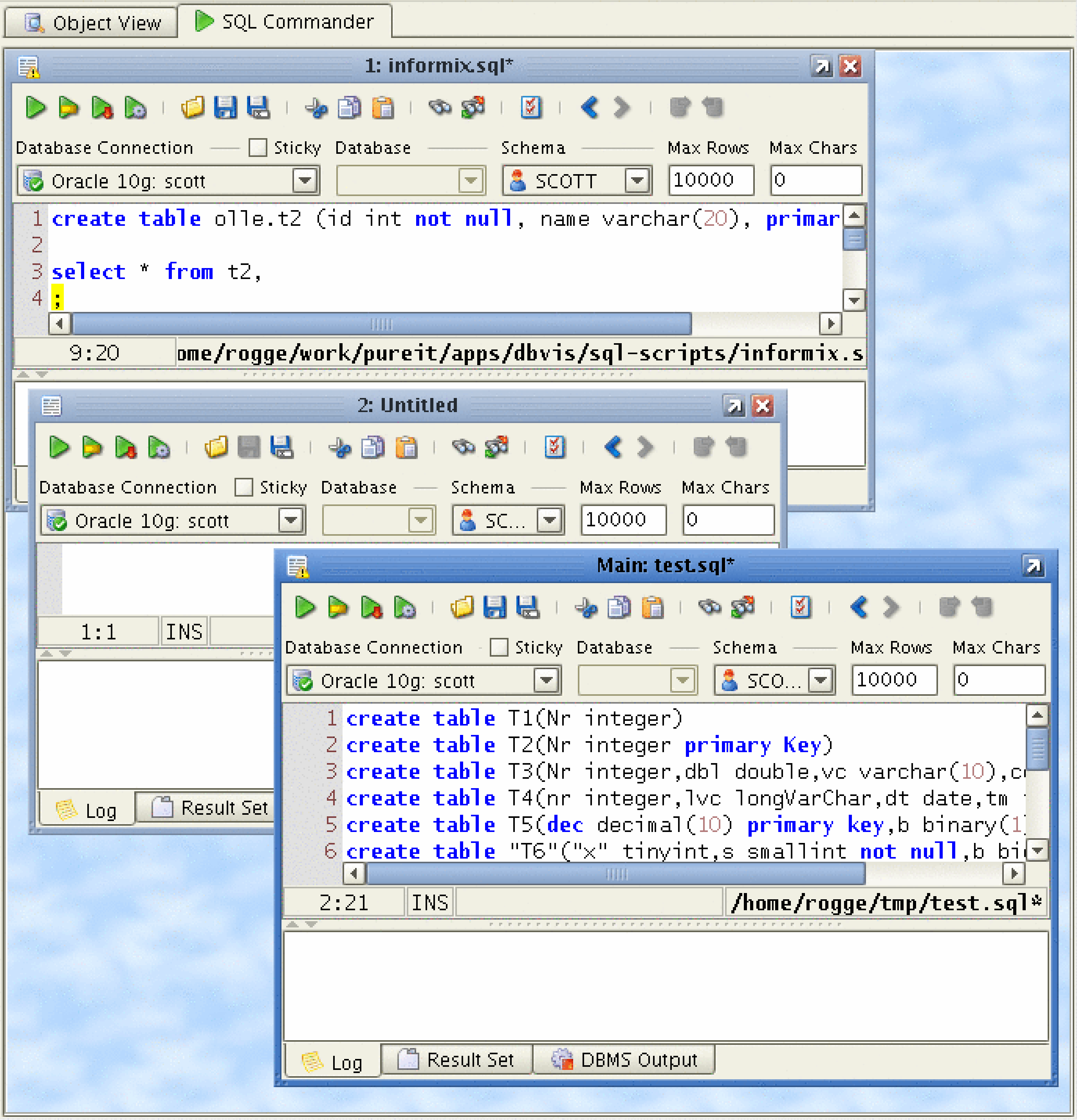
```
create table olle.t2 (id int not null, name varchar(20), primary key (id));

select * from t2,
;

--/
create procedure proc3( i integer )
  returning integer ;
define j integer ;
  for j in ( 1 to 5 )
    let i = i + 1 ;
    return i with resume ;
  end for
```

9:20 INS /home/rogge/work/pureit/apps/dbvis/sql-scripts/informix.sql*

**Figure: Multiple SQL editors in the Tabs view**

### Windows style

The following figure shows the same editors but in the Windows view.



Database Connection: Oracle 10g: scott ☐ Sticky Database: Schema: SCOTT Max Rows: 10000 Max Chars: 0

```
create table olle.t2 (id int not null, name varchar(20), primar

select * from t2,
;
```

9:20 ome/rogge/work/pureit/apps/dbvis/sql-scripts/informix.s

Database Connection: Oracle 10g: scott ☐ Sticky Database: Schema: SC... Max Rows: 10000 Max Chars: 0

1:1 INS

Database Connection: Oracle 10g: scott ☐ Sticky Database: Schema: SCO... Max Rows: 10000 Max Chars: 0

```
create table T1(Nr integer)
create table T2(Nr integer primary Key)
create table T3(Nr integer,dbl double,vc varchar(10),c
create table T4(nr integer,lvc longVarChar,dt date,tm
create table T5(dec decimal(10) primary key,b binary(1
create table "T6"("x" tinyint,s smallint not null,b bi
```

2:21 INS /home/rogge/tmp/test.sql*

**Figure: Multiple SQL editors in the Windows view**

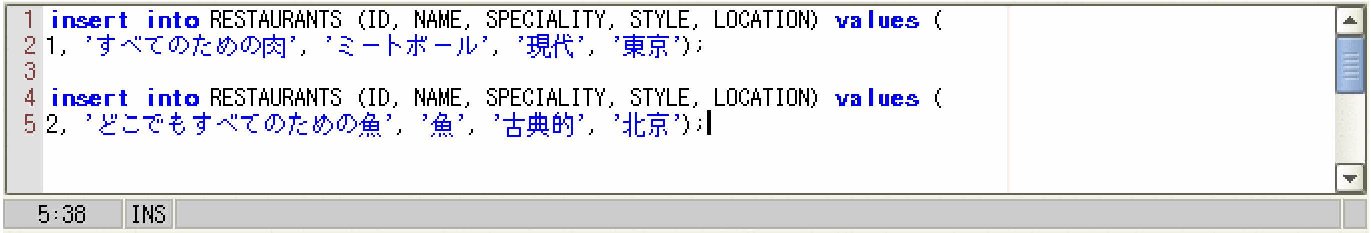
Remove an SQL editor window by selecting the close (red cross) button in the window header. Windows can be automatically organized using the **Tile** and **Cascade** operations in the **Window** main menu.

## Permissions

All SQL commands executed in the SQL Commanded are checked with the DbVisualizer Permission verifier before being executed by the database server. The permission verifier use various rules to determine if a specific SQL is allowed, denied or need confirmation before being executed. Specify in **Tool Properties- >Permissions** the rules for the verifier. The default behavior is that all SQL's are allowed withouyt delete operations must be confirmed while insert and update need no confirmation.

## Charsets and Fonts

The SQL editor supports changing font which is useful and necessary in order to display characters for languages like Chinese, Japanese, etc.



```
insert into RESTAURANTS (ID, NAME, SPECIALITY, STYLE, LOCATION) values (
1, 'すべてのための肉', 'ミートボール', '現代', '東京');

insert into RESTAURANTS (ID, NAME, SPECIALITY, STYLE, LOCATION) values (
2, 'どこでもすべてのための魚', '魚', '古典的', '北京');
```

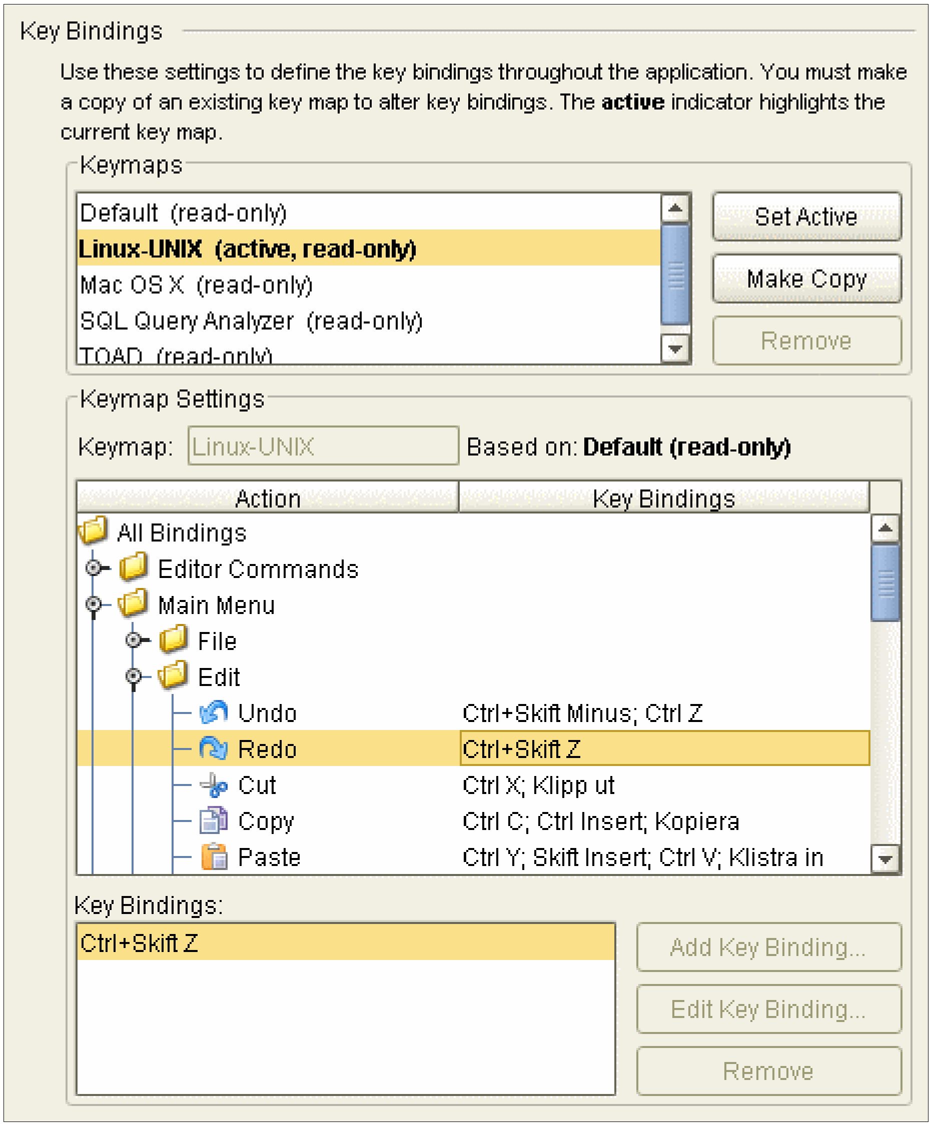
**Figure: SQL Editor with another font**

Open Tool Properties and select the Font category in order to set the font for the SQL Editor. **(**It is advisable to set the same font for both the SQL editor and the grid components).

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

## Key Bindings

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



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

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

Keymap: Linux-UNIX Based on: **Default (read-only)**

| Action | Key Bindings |
|---|---|
| All Bindings | |
| Editor Commands | |
| Main Menu | |
| File | |
| Edit | |
| Undo | Ctrl+Skift Minus; Ctrl Z |
| Redo | Ctrl+Skift Z |
| Cut | Ctrl X; Klipp ut |
| Copy | Ctrl C; Ctrl Insert; Kopiera |
| Paste | Ctrl Y; Skift Insert; Ctrl V; Klistra in |

Key Bindings:

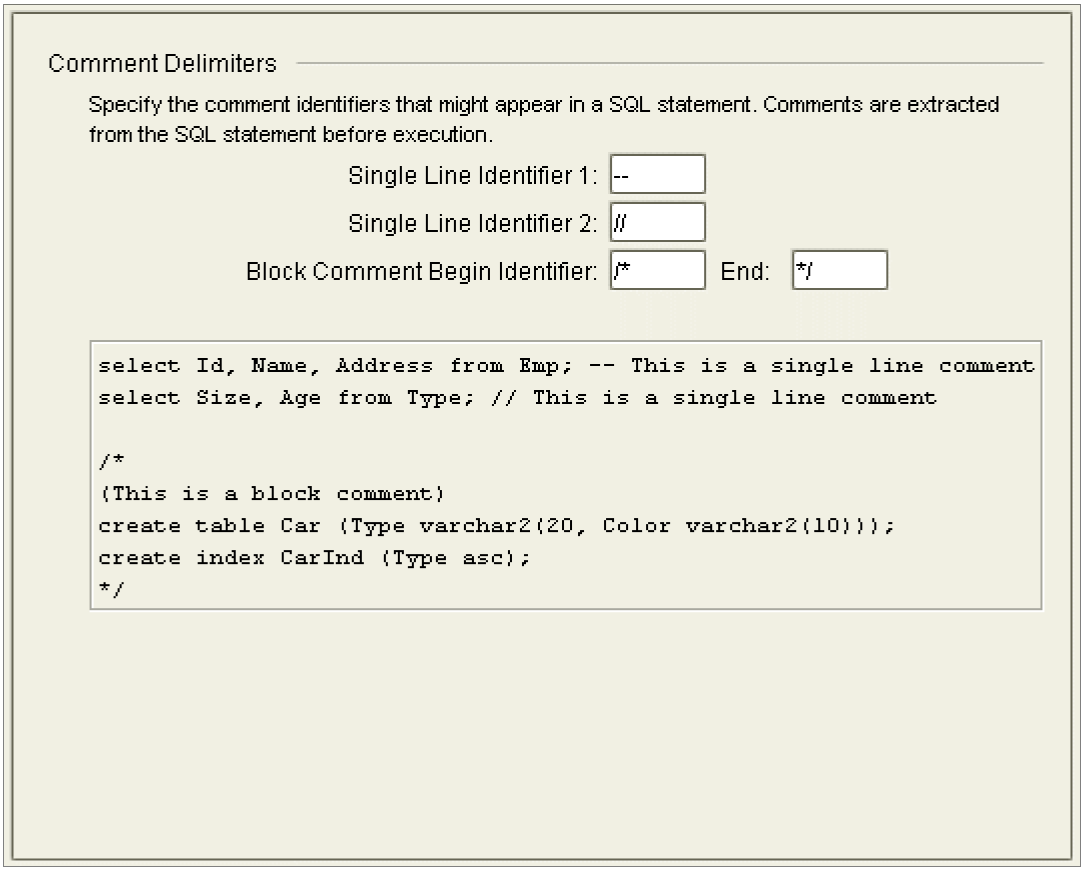
Ctrl+Skift Z

**Figure: The Key Bindings editor in Tool Properties**

Read more about configuring key bindings in the Tool Properties document.

## Client side Comments

Comments in the SQL editor are identified by the comment identifiers in Tool Properties. These are client side comments and are removed by DbVisualizer before execution. Oracle for example uses the block comment identifier to express "hints" for the database. These must be passed to the database for processing. To enable this simply change the delimiters for the block comment to something that doesn't interfere with the **/*+ ... */** notation that Oracle use.



Comment Delimiters

Specify the comment identifiers that might appear in a SQL statement. Comments are extracted 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);
*/
```

**Figure: The Comments category in Tool Properties**

## Auto Completion

Auto completion is a convenient feature used to assist in the editing of SQL statements. The auto completion support in DbVisualizer currently supports completing table and columns names for the following DML commands:

- SELECT
- INSERT
- UPDATE
- DELETE

To display the completion pop up then use the key binding Ctrl-SPACE. An entry is selected in the pop up via one of mouse double click, ENTER key or TAB key. To cancel the pop up press the ESC key.

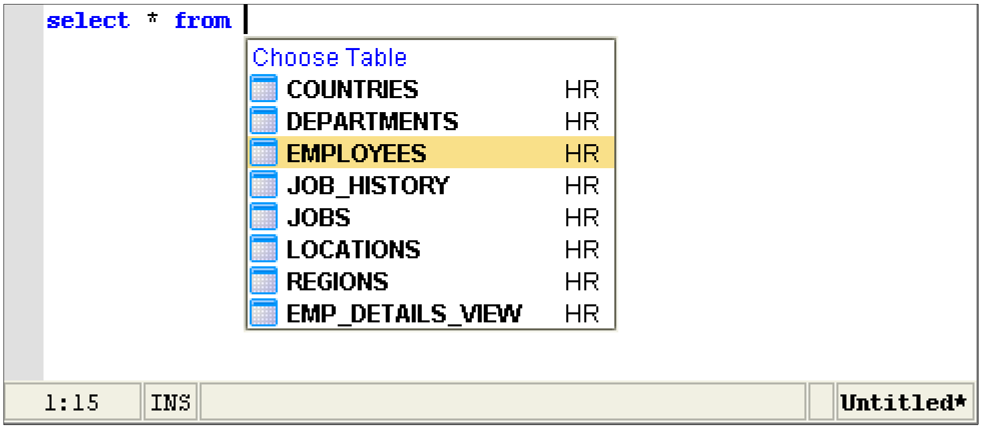
**Tip:** The **SPACE** key can be configured to select entries in the pop up. Do this in **Tool Properties->General->Key Bindings** category. Select the Editor Commands key bindings and add the SPACE key for the **Insert Newline** editor action.

**Note 1:** If there are several SQL statements in the editor then make sure to separate them using the statement delimiter character (default to ";").
**Note 2:** In order for the column name completion pop up to appear then you must first make sure there are table names in the statement.
**Note 3:** All table names that has been listed in the completion pop up are cached by DbVisualizer to make sure subsequent displays of the pop up is performed quickly without

asking the database. The cache is cleared only when doing a **Refresh** in the database objects tree or reconnecting the database connection.
**Note 4:** The **Schema** list above the editor is used only to assist the auto completion feature to limit what tables to list in the pop up.

General display settings for the auto completion feature is managed in Tool Properties.

The following shows the completion pop up with table names.



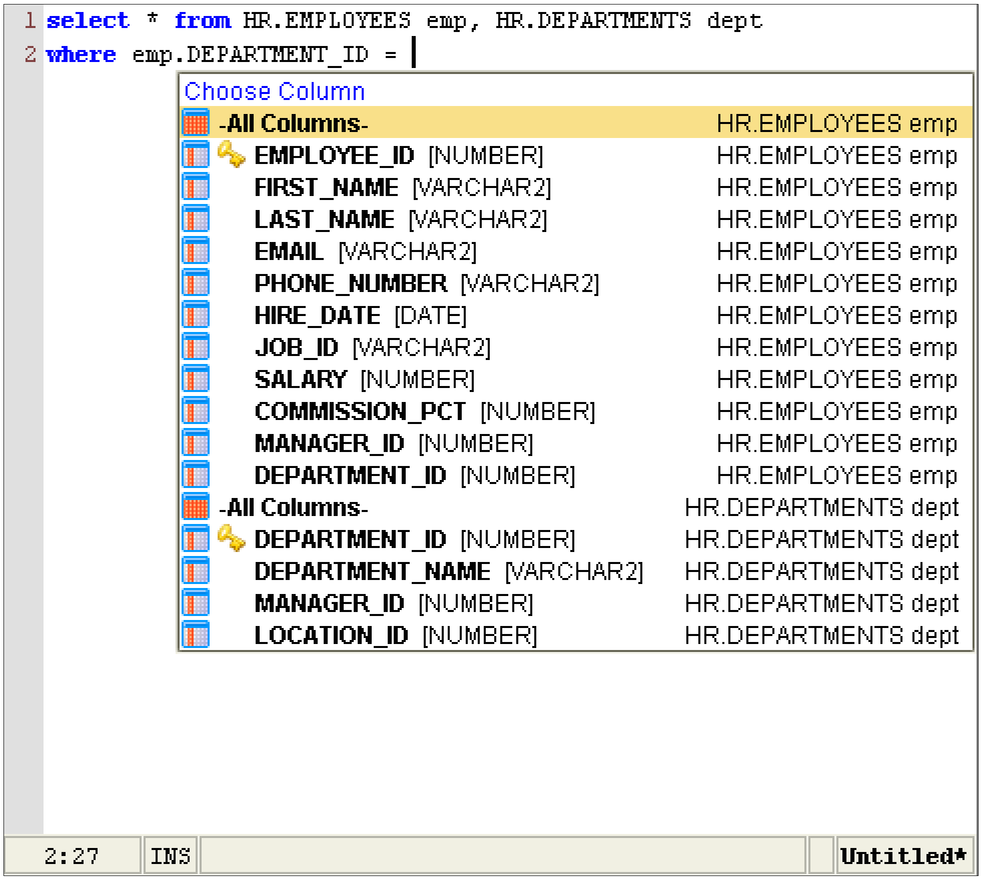
```
select * from 
```

Choose Table

|  |  |
|---|---|
| COUNTRIES | HR |
| DEPARTMENTS | HR |
| EMPLOYEES | HR |
| JOB_HISTORY | HR |
| JOBS | HR |
| LOCATIONS | HR |
| REGIONS | HR |
| EMP_DETAILS_VIEW | HR |

**Figure: Auto completion pop up showing table names**

Here is another completion pop up showing column names.



```
select * from HR.EMPLOYEES emp, HR.DEPARTMENTS dept
where emp.DEPARTMENT_ID = 
```

Choose Column

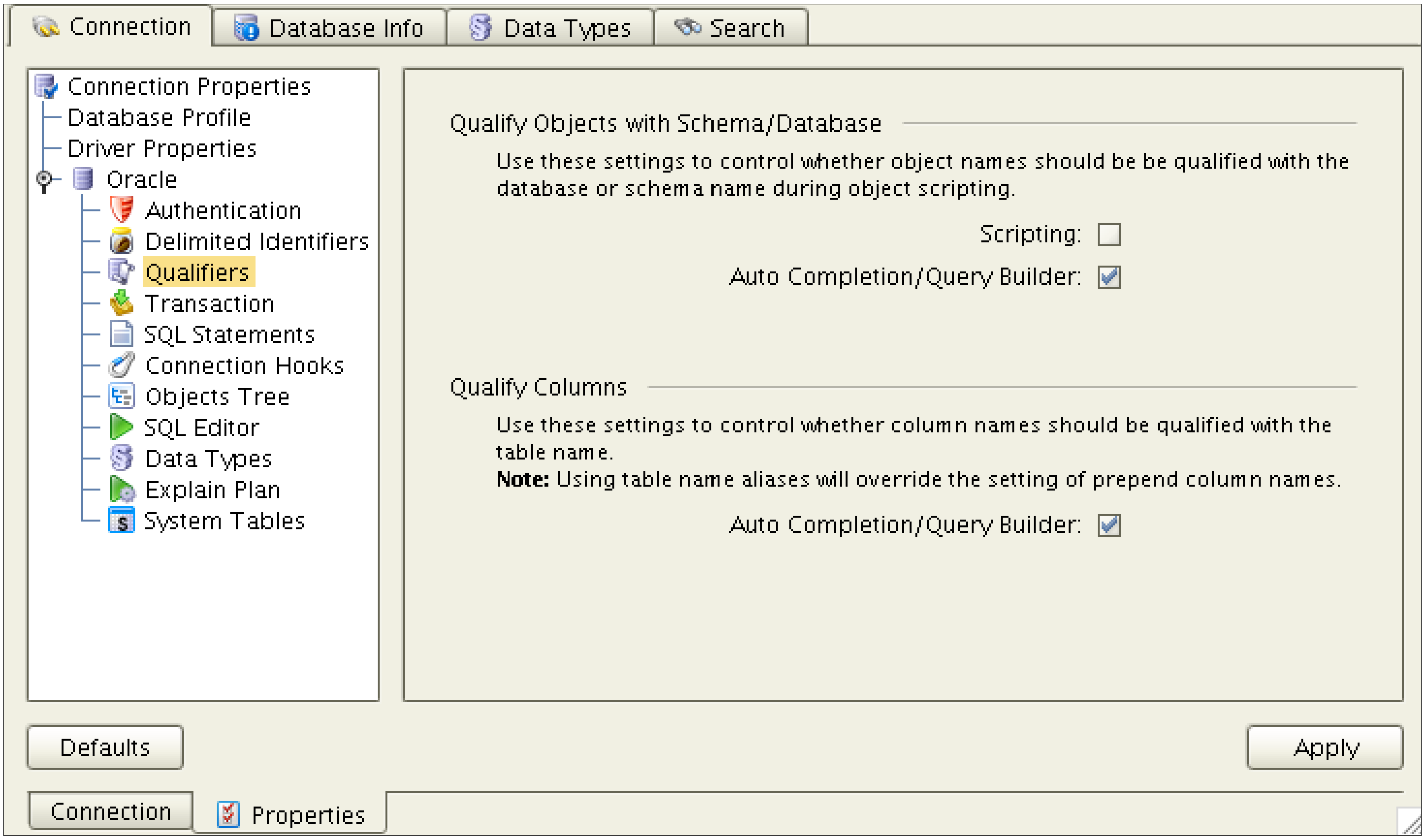
|  |  |
|---|---|
| -All Columns- | HR.EMPLOYEES emp |
| EMPLOYEE_ID [NUMBER] | HR.EMPLOYEES emp |
| FIRST_NAME [VARCHAR2] | HR.EMPLOYEES emp |
| LAST_NAME [VARCHAR2] | HR.EMPLOYEES emp |
| EMAIL [VARCHAR2] | HR.EMPLOYEES emp |
| PHONE_NUMBER [VARCHAR2] | HR.EMPLOYEES emp |
| HIRE_DATE [DATE] | HR.EMPLOYEES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

**Figure: Auto completion pop up showing column names**

Here follows a couple of examples. The <AC> symbol indicates the position where the auto completion pop up is requested. The currently selected catalog is empty and the selected schema is HR. (These examples are when accessing an Oracle database).

| | |
|---|---|
| select * from **<AC>** | Shows all tables in the **HR** schema (since HR is the selected schema) |
| select * from SYS.**<AC>** | The pop up will display all tables in the **SYS** schema independent of the schema list selection |
| select * from SYS.a**<AC>** | Lists all tables in the **SYS** schema beginning with the **A** character |
| select **<AC>** from SYS.all_objects | Lists all column in the **SYS.all_objects** table |
| select **<AC>** from SYS.all_objects all, EMPLOYEES | Lists all columns in the **SYS.all_objects** and **EMPLOYEES** table (in the **HR** schema) |
| select emp.**<AC>** from EMPLOYEES emp | Lists all columns in the **EMPLOYEES** table here identified by the alias **emp** |
| select emp.N**<AC>** from EMPLOYEES emp | Lists all columns in the **EMPLOYEES** table identified by alias **emp** starting with the **N** character |
| insert into EMPLOYEES (**<AC>** | Lists all columns in the **EMPLOYEES** table. Selecting the **- All Columns-** in the pop up will result in that all columns will be added. Each table is comma separated. |

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



Qualify Objects with Schema/Database

Use these settings to control whether object names should be be qualified with the database or schema name during object scripting.

Scripting: □

Auto Completion/Query Builder: ☑

Qualify Columns

Use these settings to control whether column names should be qualified with the table name.
**Note:** Using table name aliases will override the setting of prepend column names.

Auto Completion/Query Builder: ☑

**Figure: Properties controlling auto completion qualifiers**

Qualify disabled (for both table names and columns):

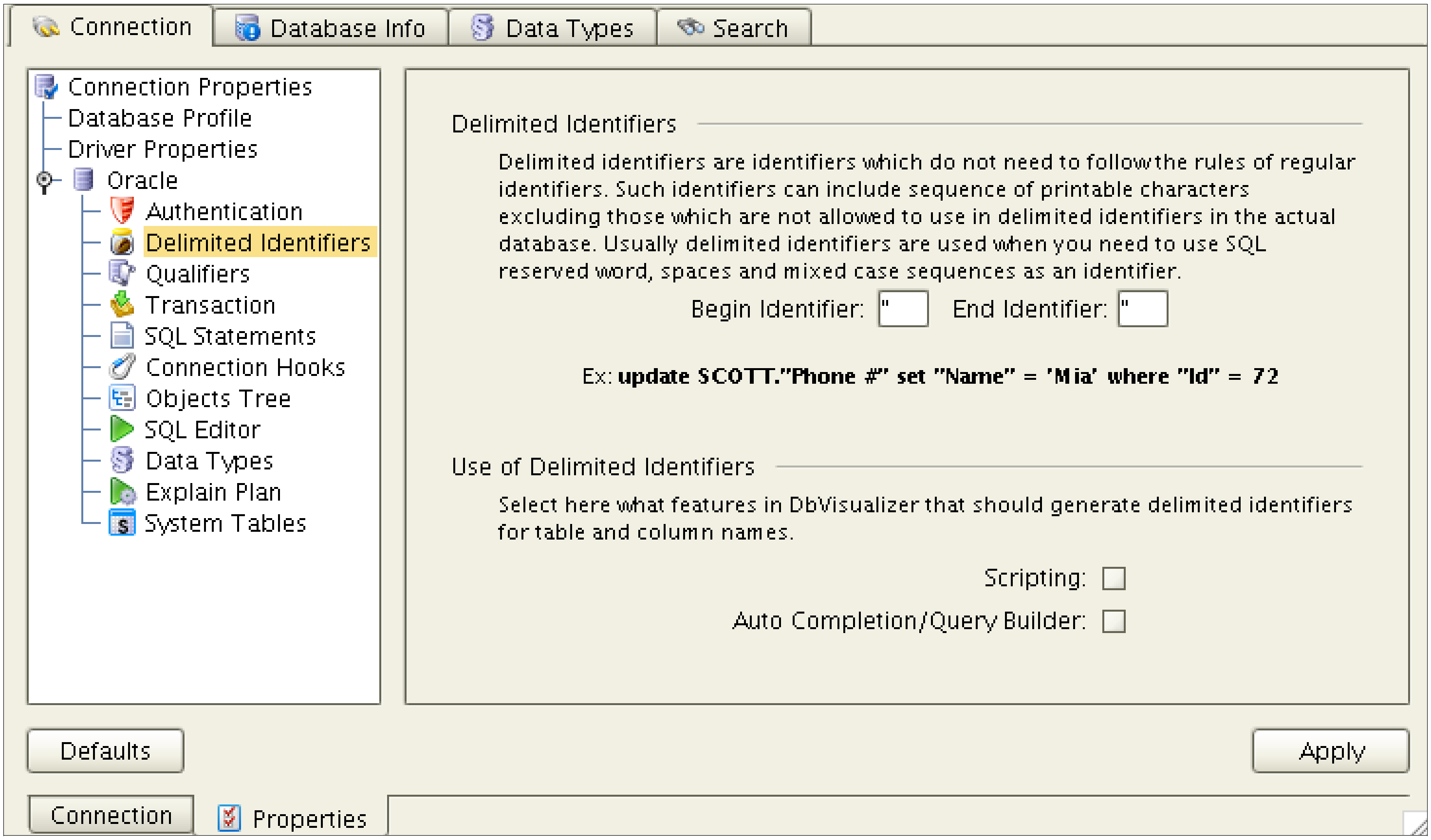
```
select Name, Address from EMPLOYEE where Id > 240
```

Qualify enabled:

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

(The setting of Qualify Columns is ignored if having table name aliases in the SQL).

These settings defines whether delimited identifiers should be part of the completed SQL.



Delimited Identifiers

Delimited identifiers are identifiers which do not need to follow the rules of regular identifiers. Such identifiers can include sequence of printable characters excluding those which are not allowed to use in delimited identifiers in the actual database. Usually delimited identifiers are used when you need to use SQL reserved word, spaces and mixed case sequences as an identifier.

Begin Identifier: " End Identifier: "

Ex: **update SCOTT."Phone #" set "Name" = 'Mia' where "Id" = 72**

Use of Delimited Identifiers

Select here what features in DbVisualizer that should generate delimited identifiers for table and column names.

Scripting: □

Auto Completion/Query Builder: □

**Figure: Properties controlling delimited identifiers for auto completion**

Delimited Identifiers disabled:

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

Delimited Identifiers enabled:

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

## SQL Formatter

The **SQL->Format SQL** feature is used to format the editor buffer according to the settings defined in **Tool Properties->SQL Editor->SQL Formatting**. If the default settings for the SQL formatter is not sufficient then edit the settings in Tool Properties, press Apply and format again to see the result. The formatter source SQL may be enclosed by quotes (as copied from a program or similar), it supports formatting the final SQL in a number of language formats such as Java, C#, PHP, VB, etc.

Example of the SQL before formatting:

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

And after formatting has been applied:

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

## SQL History

The **History** operations available in the **View** main menu are used to walk forward and backward through the history of executed SQL statements. These operations are performed in the currently selected editor and simply insert the next or previously executed SQL with accompanying settings for **Database Connection** and **Catalog** (if **Sticky** is disabled).

The history entries are in fact SQL Bookmarks and managed by the History root folder in the Bookmark Editor.

## SQL Bookmarks

SQL Bookmarks are used to manage favorite SQL statements between invocations of DbVisualizer. These are handled by the Bookmark Editor but the execution is performed in the SQL Commander. Please refer to the SQL Bookmarks document for how to use the **Bookmarks** main menu operations in the SQL Commander.

# Execution

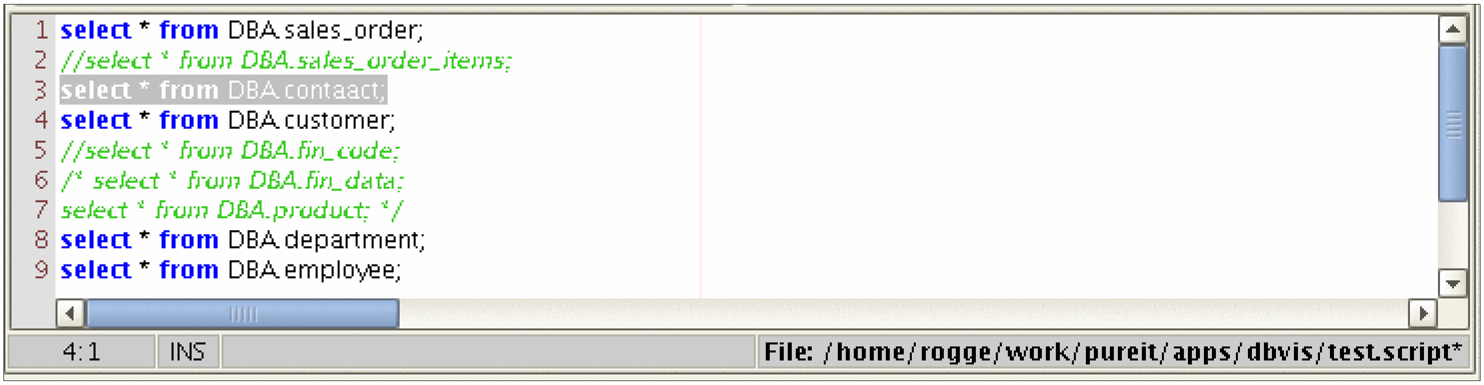
The execution of multiple SQL statements can be controlled using the **Stop Execution On** controls. These define whether the execution of the following SQL statements will be stopped based on two states:

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

**Note:** The **Stop Execution On** controls are only effective when executing multiple SQL statements

## Execute Selected Block

Selection Executes is useful when a batch of SQL statements are in the SQL editor and you just want to execute one or a few of the statement(s).



```
select * from DBA.sales_order;
//select * from DBA.sales_order_items;
select * from DBA.contaact;
select * from DBA.customer;
//select * from DBA.fin_code;
/* select * from DBA.fin_data;
select * from DBA.product; */
select * from DBA.department;
select * from DBA.employee;
```

**Figure: Selection execute**

The above figure will result in only the highlighted statement being executed.

## SQL->Execute

The **SQL->Execute** main menu operation is used to execute the SQL in the current (selected) SQL editor. The SQL Commander does this by analyzing the content in the editor to determine the SQL statements. It will then execute the statement(s) and indicate the progress. All statements in one editor are executed by the Database Connection that has been selected. The SQL Commander does not support executing SQL's for multiple database connections in one batch.

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

## SQL->Execute Current

The **Execute Current** operation is useful when having a script of several SQL statements. Use it to execute the statement at the cursor position without first needing to select the SQL statement. The default key binding for execute current is **Ctrl- PERIOD** (Ctrl- .).

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

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

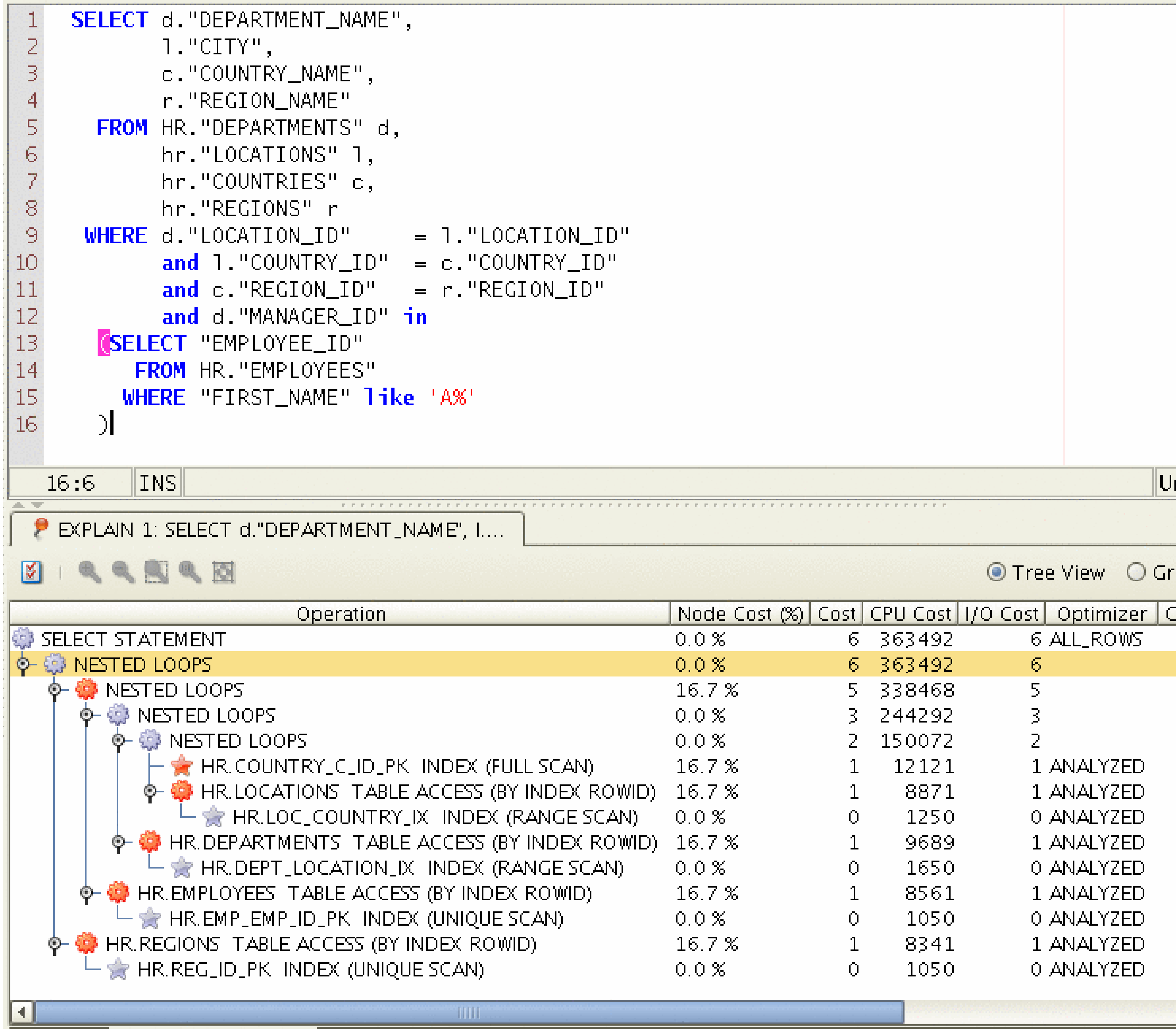
## SQL->Execute Buffer

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

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

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

DbVisualizer presents the plan either in a tree style format or in a graph. What information is shown depends on what database it is. In the tree view put the mouse pointer on the column header for a tool tip description what that column represents. The following screen shows the SQL in the editor at top and the resulting explain plan as 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 | C |
|---|---|---|---|---|---|---|
| SELECT STATEMENT | 0.0 % | 6 | 363492 | 6 | ALL_ROWS | |
| NESTED LOOPS | 0.0 % | 6 | 363492 | 6 | | |
| NESTED LOOPS | 16.7 % | 5 | 338468 | 5 | | |
| NESTED LOOPS | 0.0 % | 3 | 244292 | 3 | | |
| NESTED LOOPS | 0.0 % | 2 | 150072 | 2 | | |
| HR.COUNTRY_C_ID_PK INDEX (FULL SCAN) | 16.7 % | 1 | 12121 | 1 | ANALYZED | |
| HR.LOCATIONS TABLE ACCESS (BY INDEX ROWID) | 16.7 % | 1 | 8871 | 1 | ANALYZED | |
| HR.LOC_COUNTRY_IX INDEX (RANGE SCAN) | 0.0 % | 0 | 1250 | 0 | ANALYZED | |
| HR.DEPARTMENTS TABLE ACCESS (BY INDEX ROWID) | 16.7 % | 1 | 9689 | 1 | ANALYZED | |
| HR.DEPT_LOCATION_IX INDEX (RANGE SCAN) | 0.0 % | 0 | 1650 | 0 | ANALYZED | |
| HR.EMPLOYEES TABLE ACCESS (BY INDEX ROWID) | 16.7 % | 1 | 8561 | 1 | ANALYZED | |
| HR.EMP_EMP_ID_PK INDEX (UNIQUE SCAN) | 0.0 % | 0 | 1050 | 0 | ANALYZED | |
| HR.REGIONS TABLE ACCESS (BY INDEX ROWID) | 16.7 % | 1 | 8341 | 1 | ANALYZED | |
| HR.REG_ID_PK INDEX (UNIQUE SCAN) | 0.0 % | 0 | 1050 | 0 | ANALYZED | |

**Figure: Explain Plan presented as a tree**

The Graph View shows the plan in a graph. The graph can be exported to an image file or printed. Use the File menu choices to export and print.

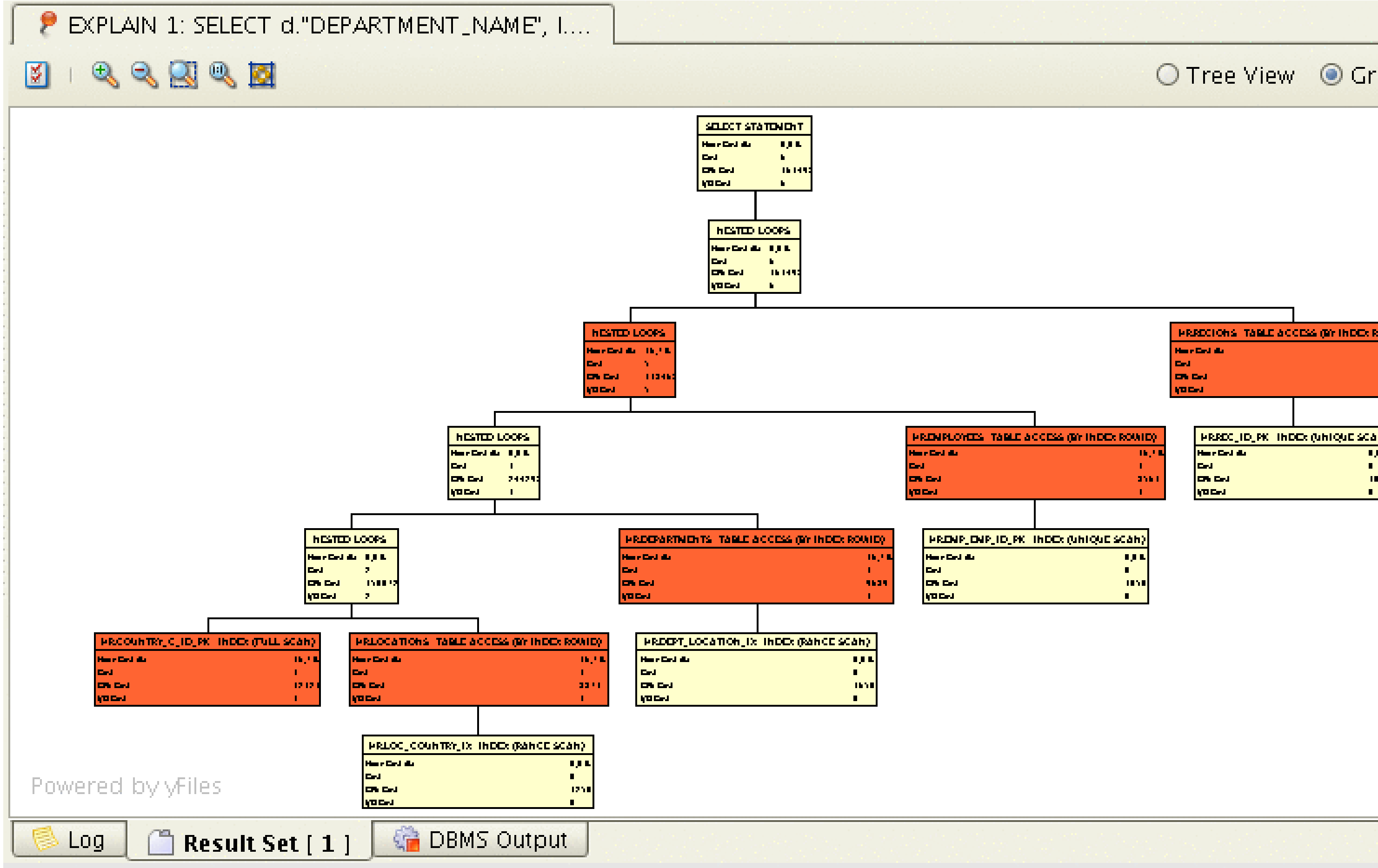
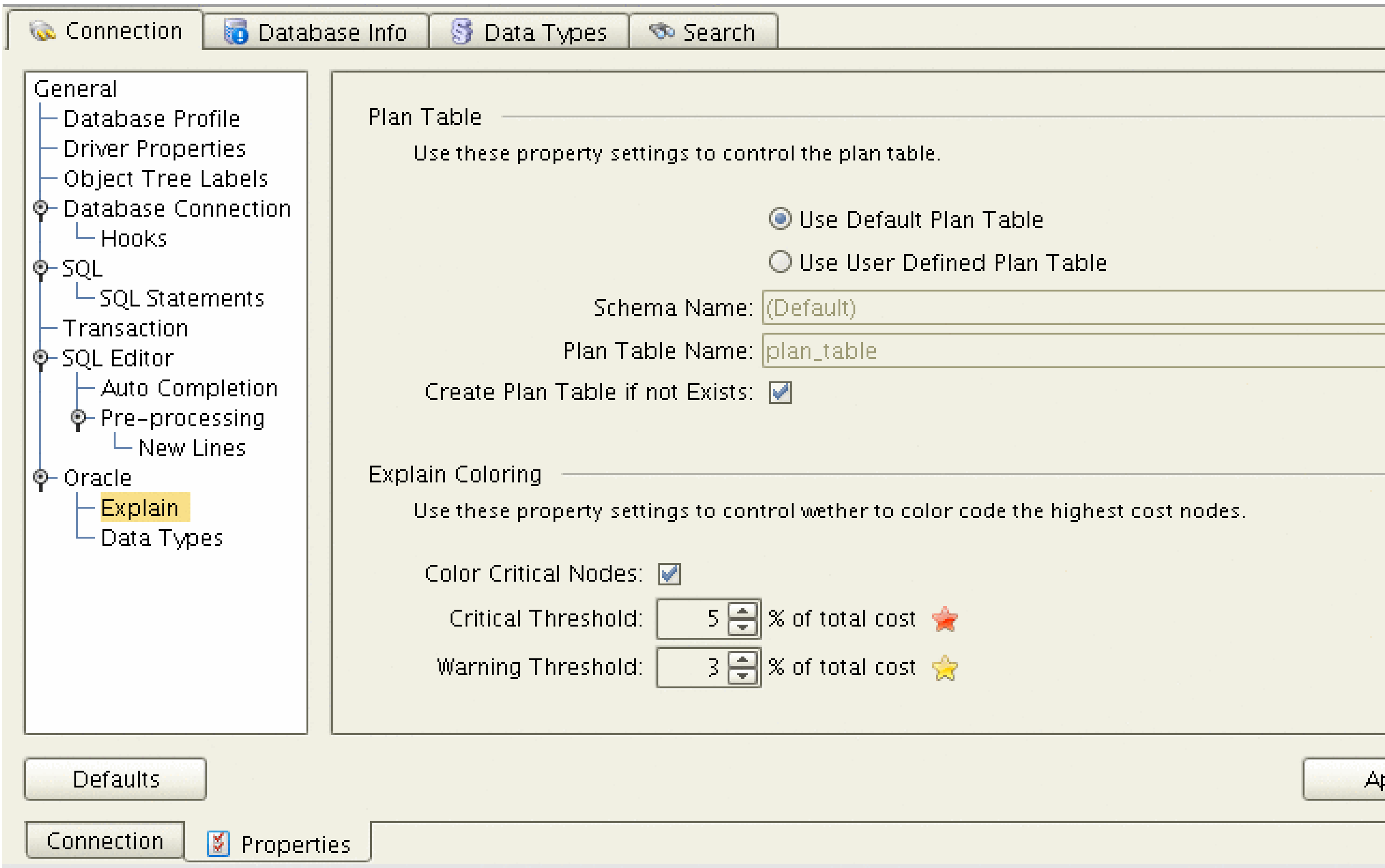

EXPLAIN 1: SELECT d."DEPARTMENT_NAME", l....
○ Tree View
● Gr
SELECT STATEMENT
NESTED LOOPS
NESTED LOOPS
HR.REGIONS TABLE ACCESS (BY INDEX R
NESTED LOOPS
HR.EMPLOYEES TABLE ACCESS (BY INDEX ROWID)
HR.REG_ID_PK INDEX (UNIQUE SCA
NESTED LOOPS
HR.DEPARTMENTS TABLE ACCESS (BY INDEX ROWID)
HR.EMP_EMP_ID_PK 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)
Powered by yFiles
Log
Result Set [ 1 ]
DBMS Output

**Figure: Explain Plan presented as a graph**

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



Plan Table

Use these property settings to control the plan table.

● Use Default Plan Table
○ Use User Defined Plan Table

Schema Name: (Default)
Plan Table Name: plan_table
Create Plan Table if not Exists: ☑

Explain Coloring

Use these property settings to control wether to color code the highest cost nodes.

Color Critical Nodes: ☑
Critical Threshold: 5 % of total cost
Warning Threshold: 3 % of total cost

**Figure: Explain Plan configuration**

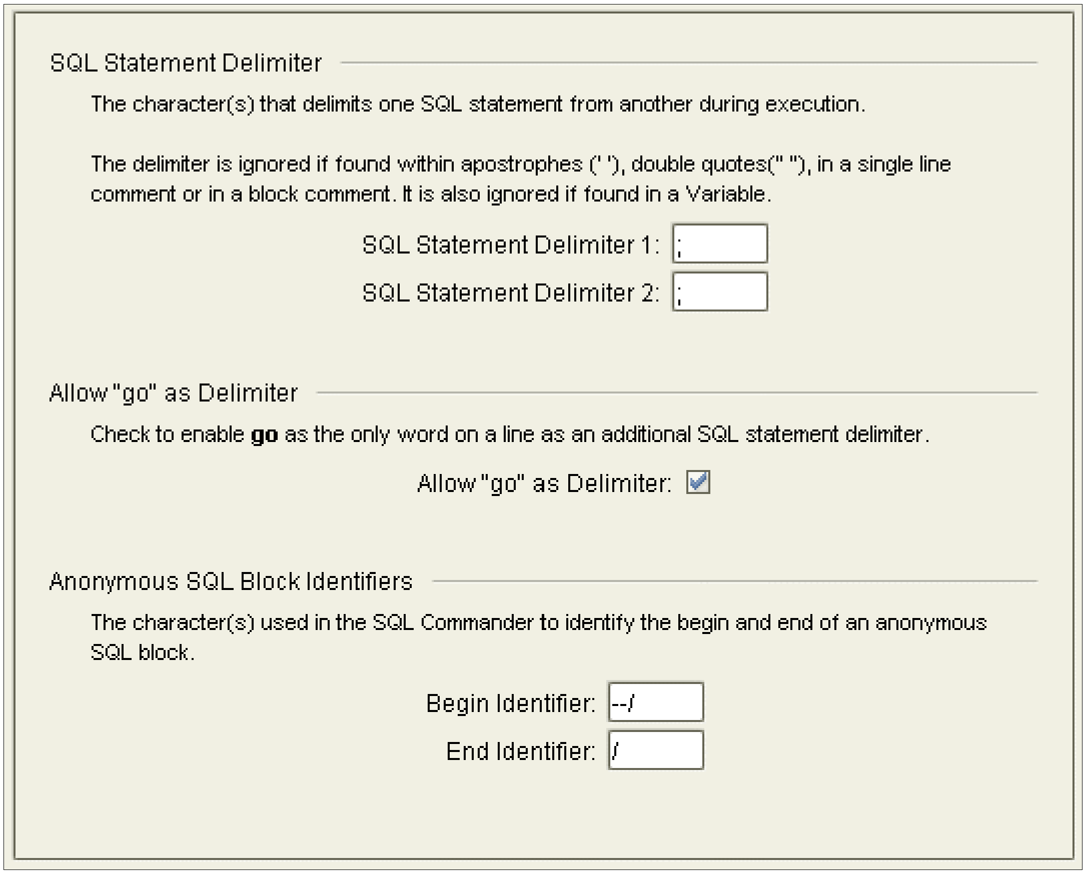
The configuration options for each of the supported database are different.

## Commit and Rollback

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

## SQL Scripts

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



SQL Statement Delimiter

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

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

SQL Statement Delimiter 1: ;

SQL Statement Delimiter 2: ;

Allow "go" as Delimiter

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

Allow "go" as Delimiter: ☑

Anonymous SQL Block Identifiers

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

Begin Identifier: --/

End Identifier: /

**Figure: Statement Delimiters**

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

```
select * from MyTable;                                       /* Stmt 1 */

insert into table MyTable
(Id, Name) /* This is a comment */ values (1, 'Arnold')      /* Stmt 2 */
go

update MyTable set Name = 'George' where Id = 1;             /* Stmt 3 */

select * from
MyTable; // This is a comment                                /* Stmt 4 */
```

## Anonymous SQL blocks

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

| | |
|---|---|
| **Begin Identifier:** | `--/` |
| **End Identifier:** | `/` |

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

```
--/ script to disable foreign keys

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

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

If you want to execute the complete editor buffer as an anonymous SQL block then use the SQLI->Execute Buffer operation. Doing this requires no begin or end identifiers.

## Stored Procedures

Executing stored procedures is not officially supported by DbVisualizer even though it works for some databases. The best way to figure it out is to test.

Our internal tests show that the Sybase ASE and SQL Server procedure calls work ok in the SQL Commander. DbVisualizer also presents multiple result sets from a single procedure call as of version 4.0 for these databases.

## Client Side Commands

The SQL Commander supports a number of DbVisualizer specific editor commands. An editor command begins with the at sign, "@". The following sections describe what commands are available.

### @run - run SQL script from file

### @cd <directory> - change directory

### @<file> - run SQL script from file

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

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

Example of a script utilizing the file referencing commands:

```
select * from MyTable;  -- Selects data from MyTable

                        -- Execute the content in the
@run createDB.sql;      -- createDB.sql file. The location
                        -- of this file is the same as the working
                        -- directory for DbVisualizer.

@cd /home/mupp;         -- Request to change directory to /home/mupp

                        -- Execute the content in the
@loadBackup.sql;        -- loadBackup.sql file. This file will now
                        -- be loaded from the /home/mupp directory.
```

### @export - export result sets to file

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

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

result sets are presented in the DbVisualizer tool

These are all supported parameters and their values:

| Parameter | Default Value | Valid Values |
|---|---|---|
| AppendFile | false | true, false, clear |
| BinaryFormat | Don't Export | Don't Export, Value, Hex, Base64 |
| CsvColumnDelimiter | \t (TAB) | |
| CsvIncludeColumnHeader | true | true, false |
| CsvIncludeSQLCommand | false | true, false |
| CsvRowCommentIdentifier | | |
| CsvRowDelimiter | \n | \n (UNIX/Linux/Mac OS X), \r\n (Windows) |
| DateFormat | yyyy-MM-dd | See valid formats in Tool Properties document |
| DecimalNumberFormat | Unformatted | See valid formats in Tool Properties document |
| Destination | File | File |
| Encoding | UTF-8 | |
| **Filename** | **REQUIRED** | |
| Format | CSV | CSV, HTML, XML, SQL |
| HtmlIncludeSQLCommand | false | true, false |
| HtmlIntroText | | |
| HtmlTitle | DbVisualizer export output | |
| NumberFormat | Unformatted | See valid formats in Tool Properties document |

| | | |
|---|---|---|
| QuoteTextData | None (ANSI if Format="SQL") | None, Single, Double, ANSI |
| SettingsFile | | |
| ShowNullAs | (null) | |
| SqlIncludeSQLCommand | false | true, false |
| SqlRowCommentIdentifier | -- | |
| SqlSeparator | ; | |
| TimeFormat | HH:mm:ss | See valid formats in Tool Properties document |
| TimeStampFormat | yyyy-MM-dd HH:mm:ss.SSSSSS | See valid formats in Tool Properties document |
| XmlIncludeSQLCommand | false | true, false |
| XmlIntroText | | |

**Example 1: @export with minimum setup**

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

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

select * from Orders;
```

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

This example shows that the file name will be 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 file name 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.

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

```
@export on;
```

```
@export set filename="c:\Backups\${table}" format="sql";

select * from Orders;
select * from Products;
select * from Transactions;
```

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

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

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

```
@export on;
@export set filename="c:\Backups\alltables.sql" appendfile="clear" format="sql

select * from Orders;
select * from Products;
select * from Transactions;
```

### Example 4: @export using pre-defined settings

The export grid wizard supports saving export settings to a file for later use in the export wizard. A export settings file can in addition be referenced in the **@export set** command.

```
@export on;
@export set settingsfile="c:\exportsettings\htmlsettings.xml" filename="c:\Bac

select * from Orders;
select * from Products;
select * from Transactions;
```

The example shows that all settings will be read from the **c:\exportsettings\html.xml** file.

### @exit [nocheck] - Exit DbVisualizer

The @exit command is the same as selecting the **File->Exit** operation. This is useful if starting DbVisualizer using the **-invisible**, **-sql** and **-execute** program arguments. Having **@exit** last in the loaded SQL file will force DbVisualizer to exit once the script has been executed. The **nocheck** argument defines that no confirmation dialogs should be displayed during exit.

### @window iconify - Iconify the main window

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

### @window restore - Raise the main window

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

### @desc table - Describe the columns in table

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

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

### @spool log - Save log to file

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

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

### @stop on error - Stop execution if any error occur

### @stop on warning - Stop execution if any warning occur

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

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

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

### @spool log - Save log to file

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

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

## Parameterized queries (variables)

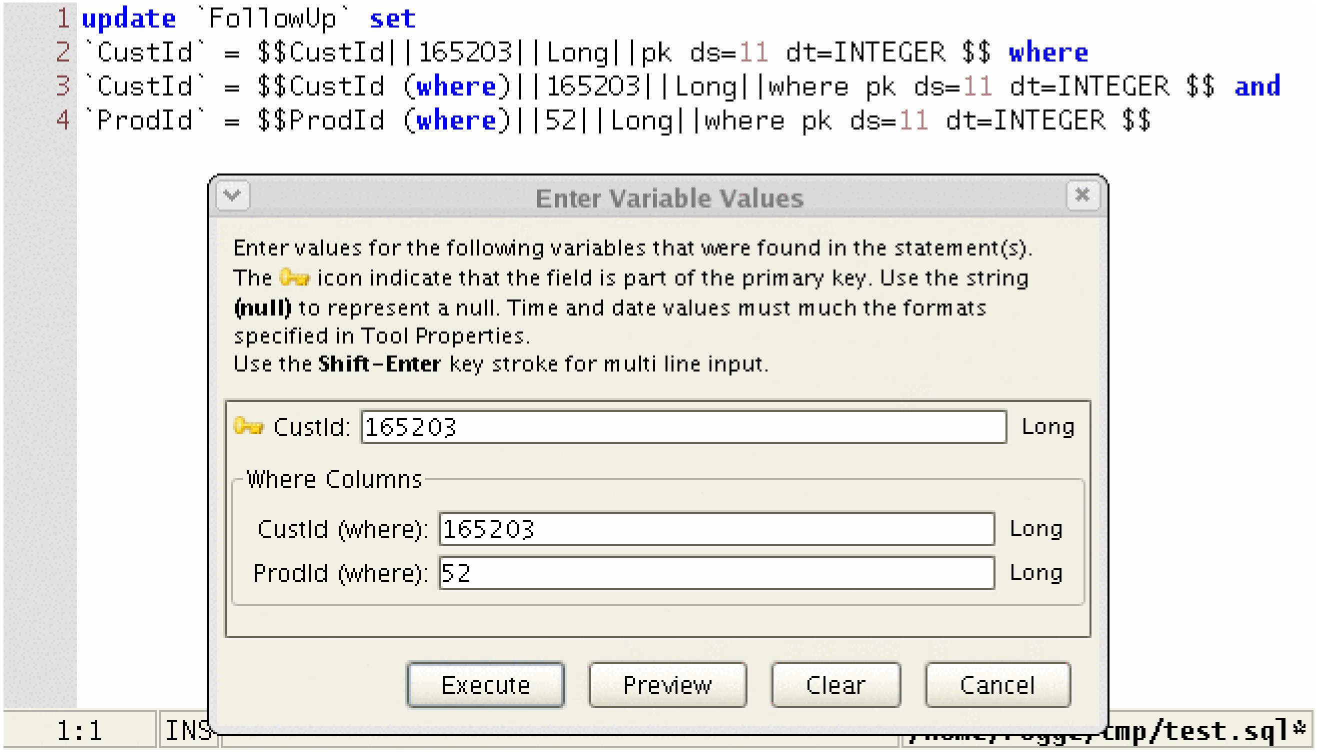
Variables can be used to build parameterized SQL statements. The SQL Commander will at execution check for variables and prompt for replacement values of the variables. Variables are also used internally in DbVisualizer. The SQL templates that are listed in the Tool Properties->SQL->SQL Statements category are used inside DbVisualizer in various situations. The difference with these is that DbVisualizer automatically substitutes the pre-defined variable names with correct values once the templates are used instead of prompting for values as the SQL Commander does.

A variable has the following format in its simplest use:

```
$$FullName$$
```

A variable must begin and end with the character(s) identified by the **Variable Identifier** property in the **Tool Properties->SQL** category (default is **$$** as in the example above). During execution the SQL Commander will search for variables and display a window with the name of each variable and an input (value) field. Enter the value for each variable and then press **Execute**. This will replace the original variable with the value and finally let the database execute the statement.

**Tip:** Use the **Ctrl->Enter** key binding as a shortcut for **Execute**.



```
update `FollowUp` set
`CustId` = $$CustId||165203||Long||pk ds=11 dt=INTEGER $$ where
`CustId` = $$CustId (where)||165203||Long||where pk ds=11 dt=INTEGER $$ and
`ProdId` = $$ProdId (where)||52||Long||where pk ds=11 dt=INTEGER $$
```

Enter values for the following variables that were found in the statement(s). The icon indicate that the field is part of the primary key. Use the string **(null)** to represent a null. Time and date values must much the formats specified in Tool Properties.
Use the **Shift-Enter** key stroke for multi line input.

| | | |
|---|---|---|
| CustId: | 165203 | Long |

Where Columns

| | | |
|---|---|---|
| CustId (where): | 165203 | Long |
| ProdId (where): | 52 | Long |

**Figure: The substitute variables window**

The above example is the simplest case as it only contains the variable name. In this case it is also necessary to place the text value within quotes since the substitution window cannot determine from the variable itself if it is a number or text variable.

The final substituted SQL statement that results from the initial SQL and variable value is:
`update Friends set LastName = 'Svensson' where Id = 100;` **Variable Syntax**

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

```
$$FullName||Swansons||String||where pk $$
```

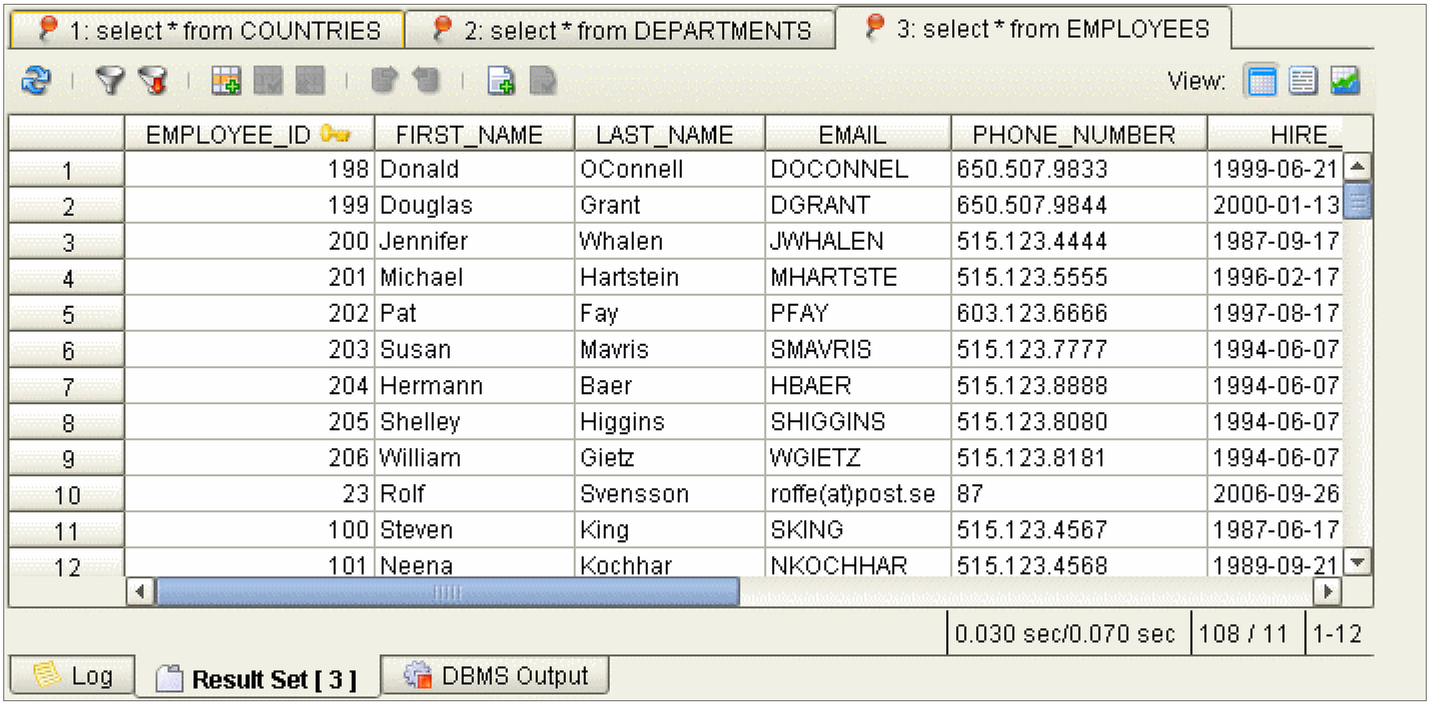
The full format of the variable syntax is:

```
$$variableName [|| defaultValue [|| type [|| options]]]
```

- **variableName**
  Required. This is the name that will appear in the substitution dialog. If several variables have the same name then the substitution dialog will show only one and the entered value will be applied to all variables of that name.
- **defaultValue**
  The default value that will appear in the substitution dialog
- **type**
  The type of variable - String, Integer, BinaryData, etc. This is used to determine if the value will be enclosed by quotes or not. If no type is specified then it is treated as an Integer (no quotes).
- **options**
  The options part is used to express various things. Most interesting are the **pk** and **where** keywords.
  (**Note:** There must be a whitespace character following a keyword).
    - **pk**
      Defines whether an icon will appear before the variable name in the substitution dialog to indicate that it is a primary key field.
- **where**
  Defines that the variable is part of the where clause and so will appear last in the list of variables.

## Output View

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


1: select * from COUNTRIES
2: select * from DEPARTMENTS
3: select * from EMPLOYEES
View:
0.030 sec/0.070 sec
108 / 11
1-12
Log
Result Set [ 3 ]
DBMS Output

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_ |
|---|---|---|---|---|---|---|
| 1 | 198 | Donald | OConnell | DOCONNEL | 650.507.9833 | 1999-06-21 |
| 2 | 199 | Douglas | Grant | DGRANT | 650.507.9844 | 2000-01-13 |
| 3 | 200 | Jennifer | Whalen | JWHALEN | 515.123.4444 | 1987-09-17 |
| 4 | 201 | Michael | Hartstein | MHARTSTE | 515.123.5555 | 1996-02-17 |
| 5 | 202 | Pat | Fay | PFAY | 603.123.6666 | 1997-08-17 |
| 6 | 203 | Susan | Mavris | SMAVRIS | 515.123.7777 | 1994-06-07 |
| 7 | 204 | Hermann | Baer | HBAER | 515.123.8888 | 1994-06-07 |
| 8 | 205 | Shelley | Higgins | SHIGGINS | 515.123.8080 | 1994-06-07 |
| 9 | 206 | William | Gietz | WGIETZ | 515.123.8181 | 1994-06-07 |
| 10 | 23 | Rolf | Svensson | roffe(at)post.se | 87 | 2006-09-26 |
| 11 | 100 | Steven | King | SKING | 515.123.4567 | 1987-06-17 |
| 12 | 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 1989-09-21 |

**Figure: The output view**

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

## Log

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

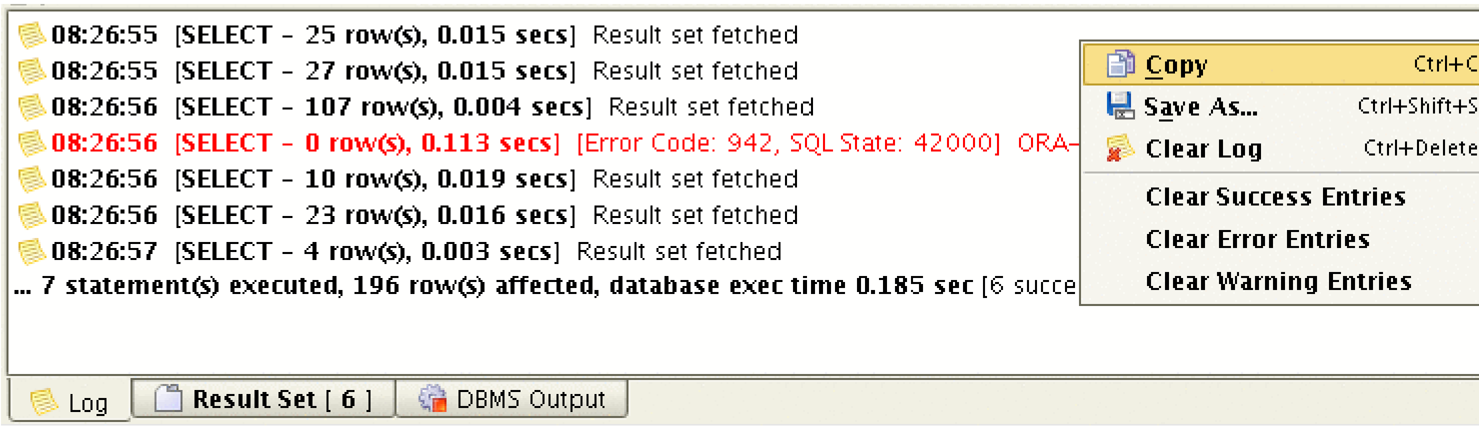

```
08:26:55 [SELECT - 25 row(s), 0.015 secs] Result set fetched
08:26:55 [SELECT - 27 row(s), 0.015 secs] Result set fetched
08:26:56 [SELECT - 107 row(s), 0.004 secs] Result set fetched
08:26:56 [SELECT - 0 row(s), 0.113 secs] [Error Code: 942, SQL State: 42000] ORA-
08:26:56 [SELECT - 10 row(s), 0.019 secs] Result set fetched
08:26:56 [SELECT - 23 row(s), 0.016 secs] Result set fetched
08:26:57 [SELECT - 4 row(s), 0.003 secs] Result set fetched
... 7 statement(s) executed, 196 row(s) affected, database exec time 0.185 sec [6 succe
```

**Figure: The Log with one failed statement**

The detail level in an error message is dependent on the driver and database that is being used. Some databases are very good at telling what went wrong and why while others are very quiet. The icon to the left of each log entry is used to pass the SQL for the entry into the current SQL editor when clicked.

### Log controls

The **Show** controls below the log are used to define what information will appear in the log. The **Filter** controls are used to specify what entries will be displayed.

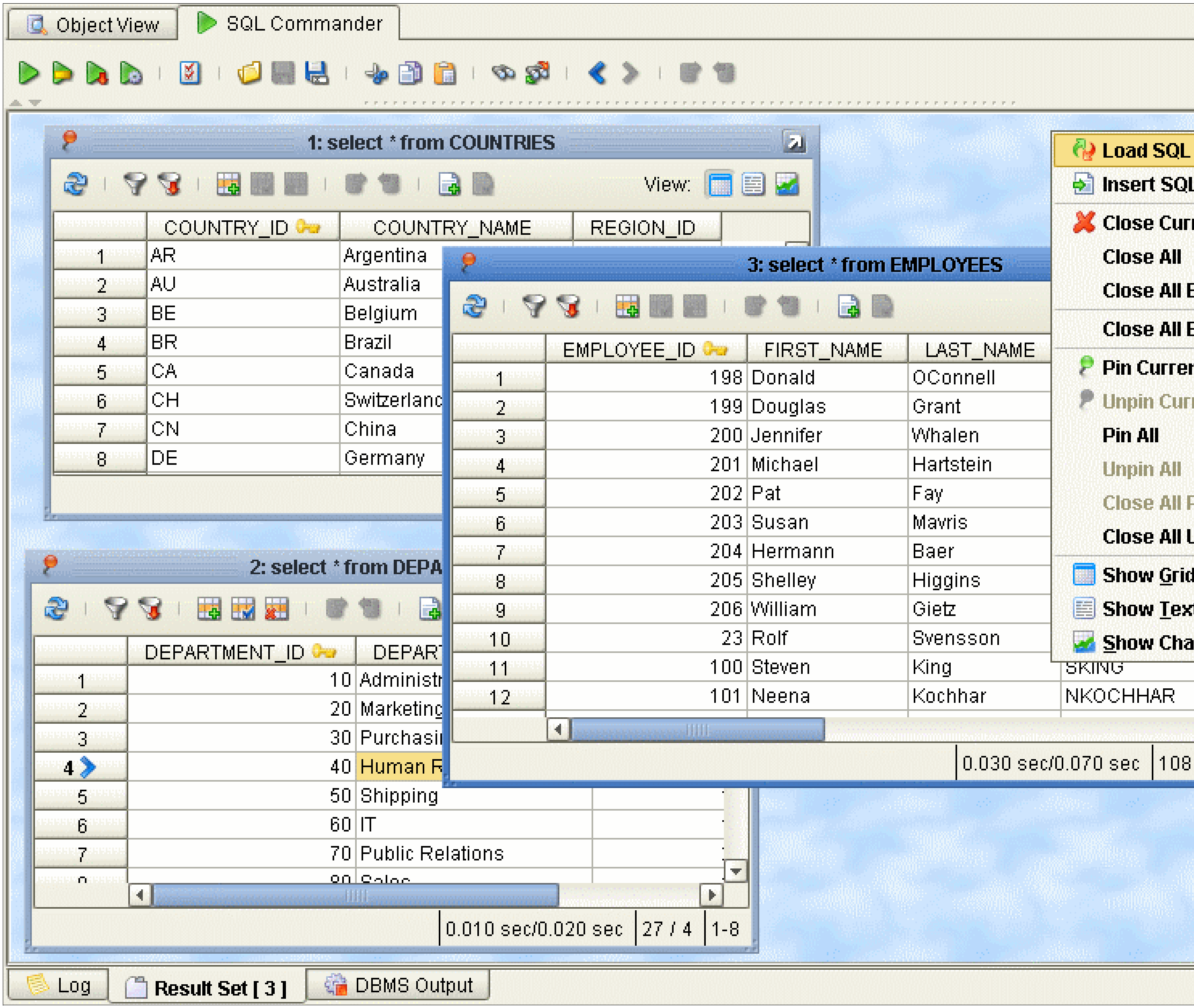
### Auto clear log

The **Auto Clear Log** control can be enabled to let the SQL Commander automatically clear the log between executions.

## Result Set

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

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



1: select * from COUNTRIES

| | COUNTRY_ID | COUNTRY_NAME | REGION_ID |
|---|---|---|---|
| 1 | AR | Argentina | |
| 2 | AU | Australia | |
| 3 | BE | Belgium | |
| 4 | BR | Brazil | |
| 5 | CA | Canada | |
| 6 | CH | Switzerlanc | |
| 7 | CN | China | |
| 8 | DE | Germany | |

2: select * from DEPA

| | DEPARTMENT_ID | DEPART |
|---|---|---|
| 1 | 10 | Administr |
| 2 | 20 | Marketing |
| 3 | 30 | Purchasi |
| 4 | 40 | Human R |
| 5 | 50 | Shipping |
| 6 | 60 | IT |
| 7 | 70 | Public Relations |
| 8 | 80 | Sales |

0.010 sec/0.020 sec | 27 / 4 | 1-8

3: select * from EMPLOYEES

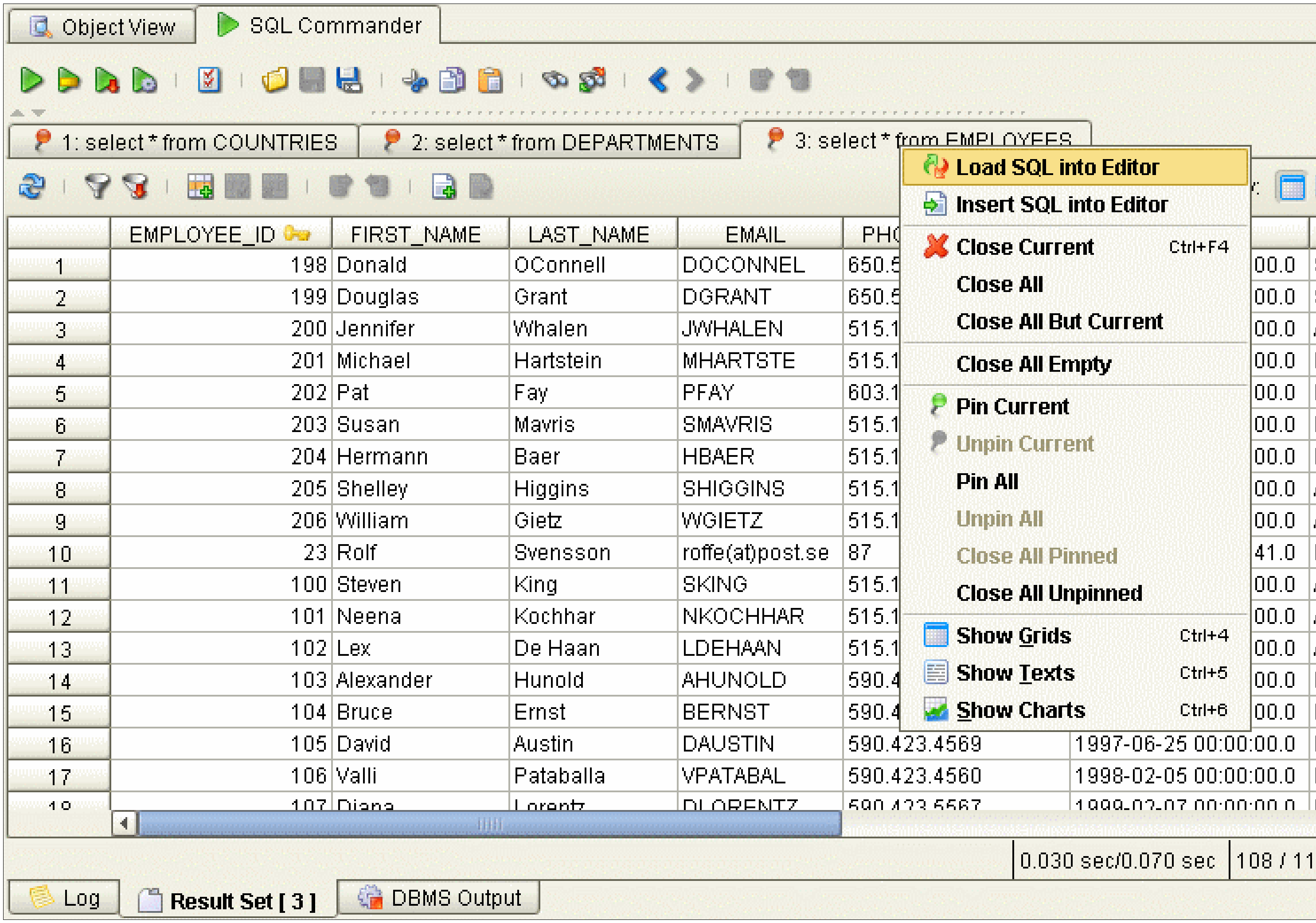
| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | |
|---|---|---|---|---|
| 1 | 198 | Donald | OConnell | |
| 2 | 199 | Douglas | Grant | |
| 3 | 200 | Jennifer | Whalen | |
| 4 | 201 | Michael | Hartstein | |
| 5 | 202 | Pat | Fay | |
| 6 | 203 | Susan | Mavris | |
| 7 | 204 | Hermann | Baer | |
| 8 | 205 | Shelley | Higgins | |
| 9 | 206 | William | Gietz | |
| 10 | 23 | Rolf | Svensson | |
| 11 | 100 | Steven | King | SKING |
| 12 | 101 | Neena | Kochhar | NKOCHHAR |

0.030 sec/0.070 sec | 108

**Figure: The windows output view**

This figure shows the **Windows** output view with three result set grids. The **Max Rows** and **Max Chars** fields at the bottom of the figure are used to set the maximum number of rows and columns (for text data) that will be fetched and presented in new result set grids. The labels for the number of rows and columns in the grid will be displayed in red if either of these exceed their respective maximum settings. A result set grid can be closed using the red cross in the window frame header.

If the output view is Tabs then use the **Close** right click menu choice when the mouse pointer is in the tab header:



| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHO | |
|---|---|---|---|---|---|---|
| 1 | 198 | Donald | OConnell | DOCONNEL | 650.5 | 00.0 |
| 2 | 199 | Douglas | Grant | DGRANT | 650.5 | 00.0 |
| 3 | 200 | Jennifer | Whalen | JWHALEN | 515.1 | 00.0 |
| 4 | 201 | Michael | Hartstein | MHARTSTE | 515.1 | 00.0 |
| 5 | 202 | Pat | Fay | PFAY | 603.1 | 00.0 |
| 6 | 203 | Susan | Mavris | SMAVRIS | 515.1 | 00.0 |
| 7 | 204 | Hermann | Baer | HBAER | 515.1 | 00.0 |
| 8 | 205 | Shelley | Higgins | SHIGGINS | 515.1 | 00.0 |
| 9 | 206 | William | Gietz | WGIETZ | 515.1 | 00.0 |
| 10 | 23 | Rolf | Svensson | roffe(at)post.se | 87 | 41.0 |
| 11 | 100 | Steven | King | SKING | 515.1 | 00.0 |
| 12 | 101 | Neena | Kochhar | NKOCHHAR | 515.1 | 00.0 |
| 13 | 102 | Lex | De Haan | LDEHAAN | 515.1 | 00.0 |
| 14 | 103 | Alexander | Hunold | AHUNOLD | 590.4 | 00.0 |
| 15 | 104 | Bruce | Ernst | BERNST | 590.4 | 00.0 |
| 16 | 105 | David | Austin | DAUSTIN | 590.423.4569 | 1997-06-25 00:00:00.0 |
| 17 | 106 | Valli | Pataballa | VPATABAL | 590.423.4560 | 1998-02-05 00:00:00.0 |
| 18 | 107 | Diana | Lorentz | DLORENTZ | 590.423.5567 | 1999-02-07 00:00:00.0 |

0.030 sec/0.070 sec | 108 / 11

**Figure: The right click menu for tabs**

### Result set menu

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

| Menu Choice | Description |
|---|---|
| **Load SQL into Editor** | Loads the SQL for the selected result set tab or window into the current editor. |
| **Insert SQL into Editor** | Inserts the SQL for the selected result set tab or window into the current editor at the cursor position. |
| **Close Current** | Close the current result set |
| **Close All** | Closes all result sets |

| | |
|---|---|
| **Close All But Current** | Closes all but the current result set |
| **Close All Empty** | Closes all result sets that are empty (no data) |
| **Pin Current** | Pin the current result set (prevent it from being removed at next execution). |
| **Unpin Current** | Unpin the current result set |
| **Pin All** | Pins all result sets. Pinning a result set will prevent it from being removed at the next execution. |
| **Unpin All** | Unpins any pinned result sets making them candidates for removal during the next execution. |
| **Close All Pinned** | Removes all pinned result sets directly. |
| **Close All Unpinned** | Removes all unpinned result sets directly. |
| **Show Grids** | Changes the display mode to show the grid tab for all result sets |
| **Show Texts** | Changes the display mode to show the text tab for all result sets |
| **Show Charts** | Changes the display mode to show the chart tab for all result sets |

### Editing

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

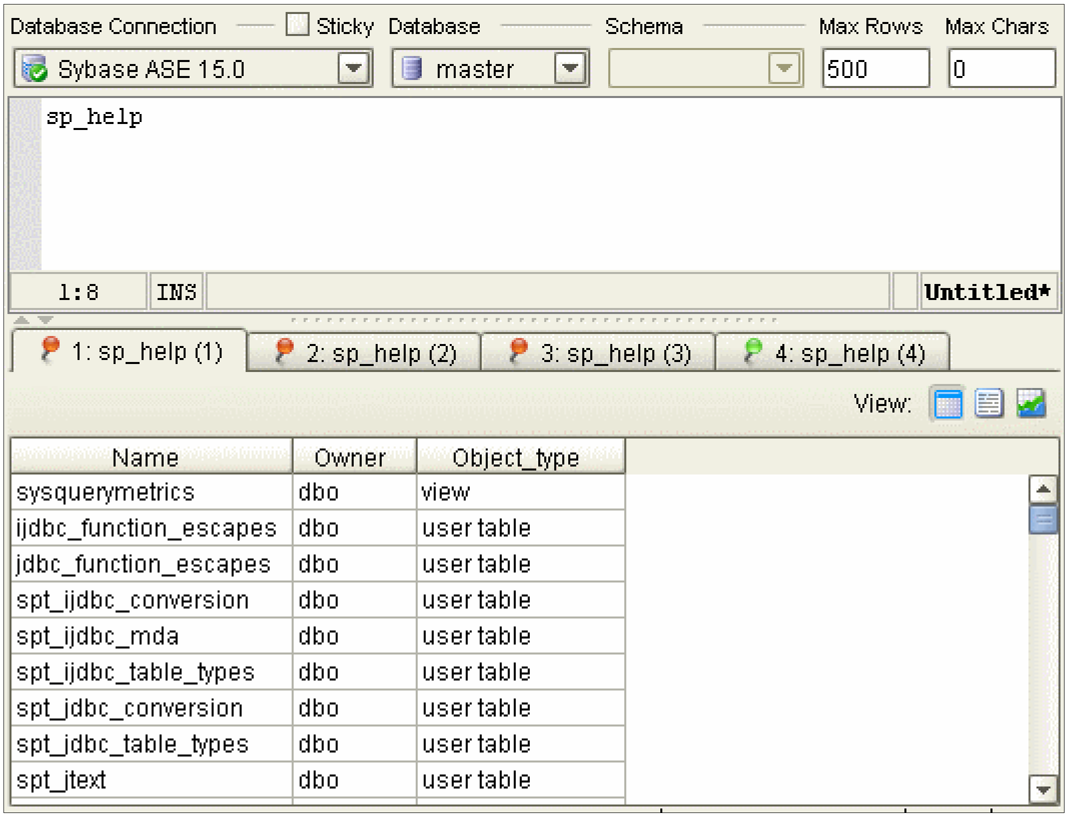
1. The result really is a result set
2. The SQL is a SELECT command
3. Only one table is referenced in the FROM clause
4. All columns in the result set exist in the table with exactly matching names

If all the above is true then the standard editing tool bar will appear just above the grid. Read more about editing in the Edit Table Data document.

If any of the above fail to comply will the editing tool bar not appear.

### Multiple result sets produced by a single SQL statement

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

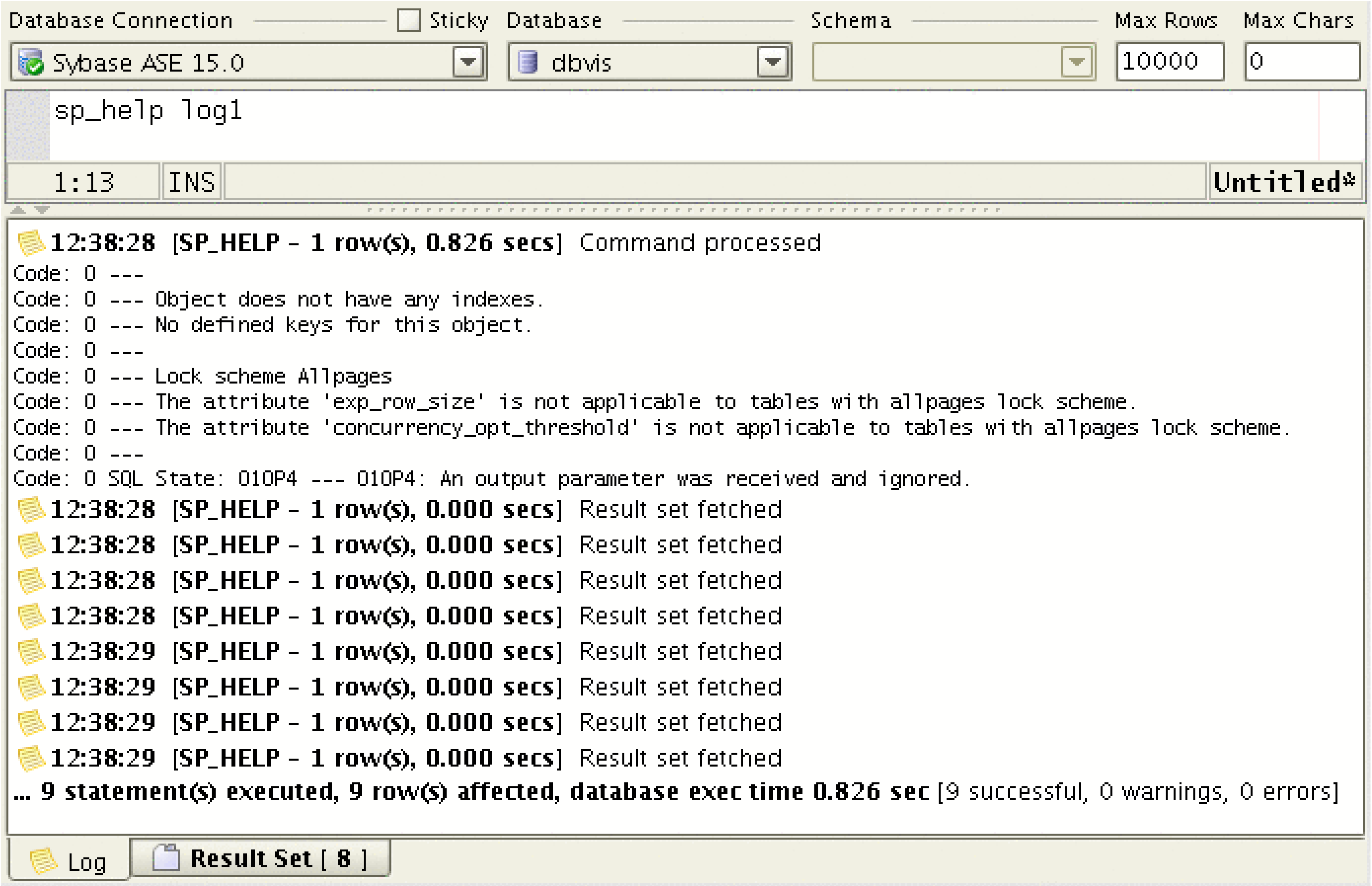


```
sp_help
```

| Name | Owner | Object_type |
|---|---|---|
| sysquerymetrics | dbo | view |
| ijdbc_function_escapes | dbo | user table |
| jdbc_function_escapes | dbo | user table |
| spt_ijdbc_conversion | dbo | user table |
| spt_ijdbc_mda | dbo | user table |
| spt_ijdbc_table_types | dbo | user table |
| spt_jdbc_conversion | dbo | user table |
| spt_jdbc_table_types | dbo | user table |
| spt_jtext | dbo | user table |

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

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



```
sp_help log1
```

```
12:38:28  [SP_HELP - 1 row(s), 0.826 secs]  Command processed
Code: 0 ---
Code: 0 --- Object does not have any indexes.
Code: 0 --- No defined keys for this object.
Code: 0 ---
Code: 0 --- Lock scheme Allpages
Code: 0 --- The attribute 'exp_row_size' is not applicable to tables with allpages lock scheme.
Code: 0 --- The attribute 'concurrency_opt_threshold' is not applicable to tables with allpages lock scheme.
Code: 0 ---
Code: 0 SQL State: 010P4 --- 010P4: An output parameter was received and ignored.
12:38:28  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:28  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:28  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:28  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:29  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:29  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:29  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
12:38:29  [SP_HELP - 1 row(s), 0.000 secs]  Result set fetched
... 9 statement(s) executed, 9 row(s) affected, database exec time 0.826 sec [9 successful, 0 warnings, 0 errors]
```

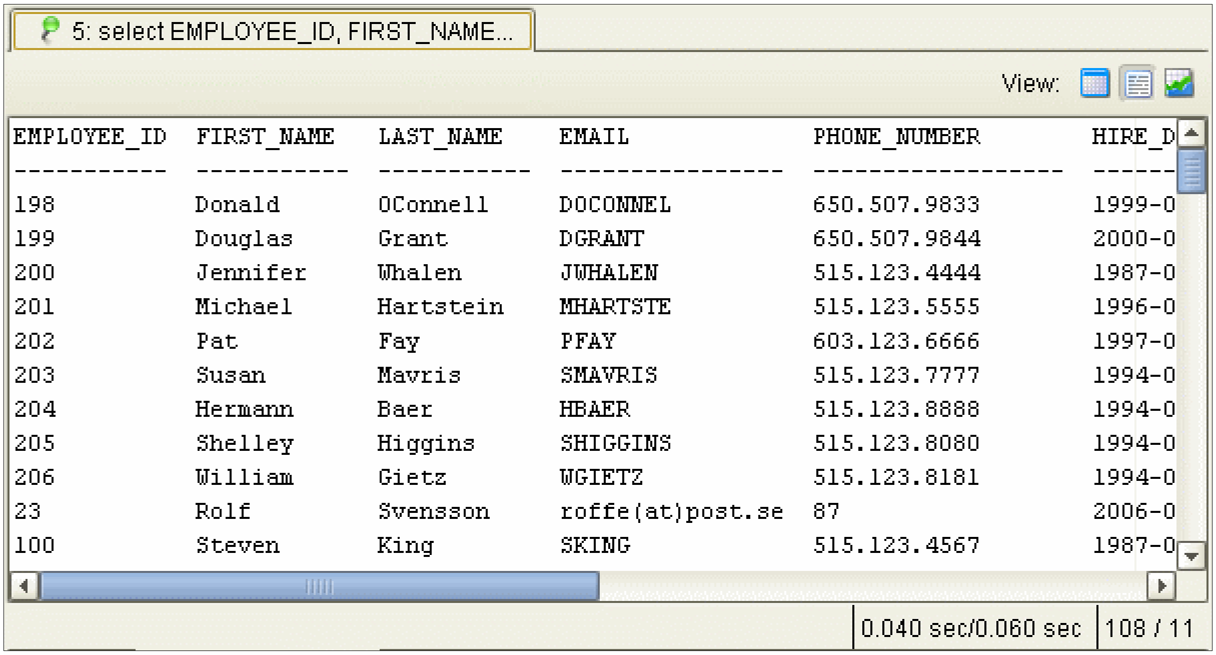
**Figure: The Log after executing an SQL statement that returns multiple results**

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

## Text

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

**Note:** The columns widths may vary between executions of the SQL.



```
EMPLOYEE_ID  FIRST_NAME   LAST_NAME    EMAIL             PHONE_NUMBER        HIRE_D
-----------  -----------  -----------  ----------------  ------------------  ------
198          Donald       OConnell     DOCONNEL          650.507.9833        1999-0
199          Douglas      Grant        DGRANT            650.507.9844        2000-0
200          Jennifer     Whalen       JWHALEN           515.123.4444        1987-0
201          Michael      Hartstein    MHARTSTE          515.123.5555        1996-0
202          Pat          Fay          PFAY              603.123.6666        1997-0
203          Susan        Mavris       SMAVRIS           515.123.7777        1994-0
204          Hermann      Baer         HBAER             515.123.8888        1994-0
205          Shelley      Higgins      SHIGGINS          515.123.8080        1994-0
206          William      Gietz        WGIETZ            515.123.8181        1994-0
23           Rolf         Svensson     roffe(at)post.se  87                  2006-0
100          Steven       King         SKING             515.123.4567        1987-0
```

0.040 sec/0.060 sec | 108 / 11

**Figure: The Text result set format**

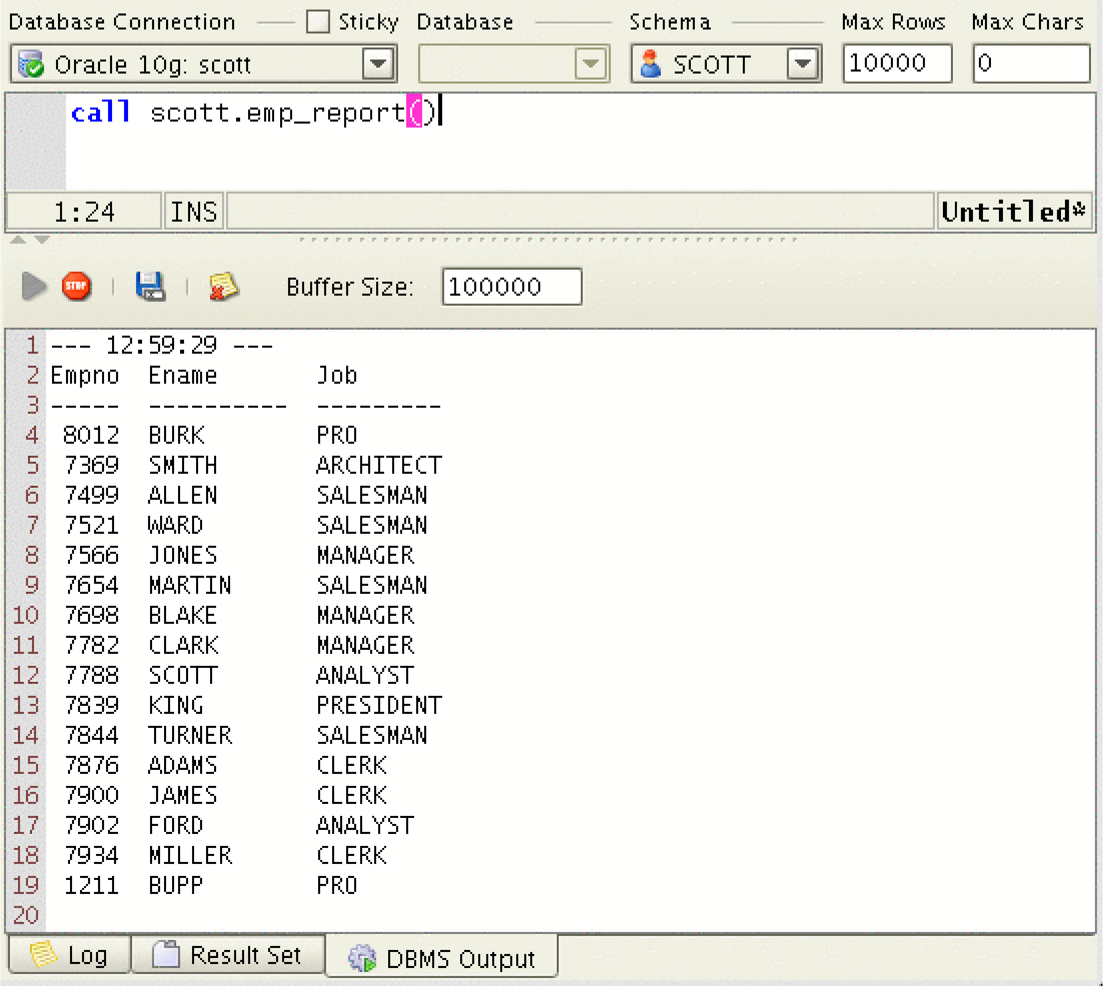
### Chart

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

## DBMS Output (Oracle)

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

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



Database Connection: Oracle 10g: scott
Database:
Schema: SCOTT
Max Rows: 10000
Max Chars: 0

```
call scott.emp_report()
```

Buffer Size: 100000

```
--- 12:59:29 ---
Empno  Ename       Job
-----  ----------  ---------
 8012  BURK        PRO
 7369  SMITH       ARCHITECT
 7499  ALLEN       SALESMAN
 7521  WARD        SALESMAN
 7566  JONES       MANAGER
 7654  MARTIN      SALESMAN
 7698  BLAKE       MANAGER
 7782  CLARK       MANAGER
 7788  SCOTT       ANALYST
 7839  KING        PRESIDENT
 7844  TURNER      SALESMAN
 7876  ADAMS       CLERK
 7900  JAMES       CLERK
 7902  FORD        ANALYST
 7934  MILLER      CLERK
 1211  BUPP        PRO
```

**Figure: DBMS Output tab**

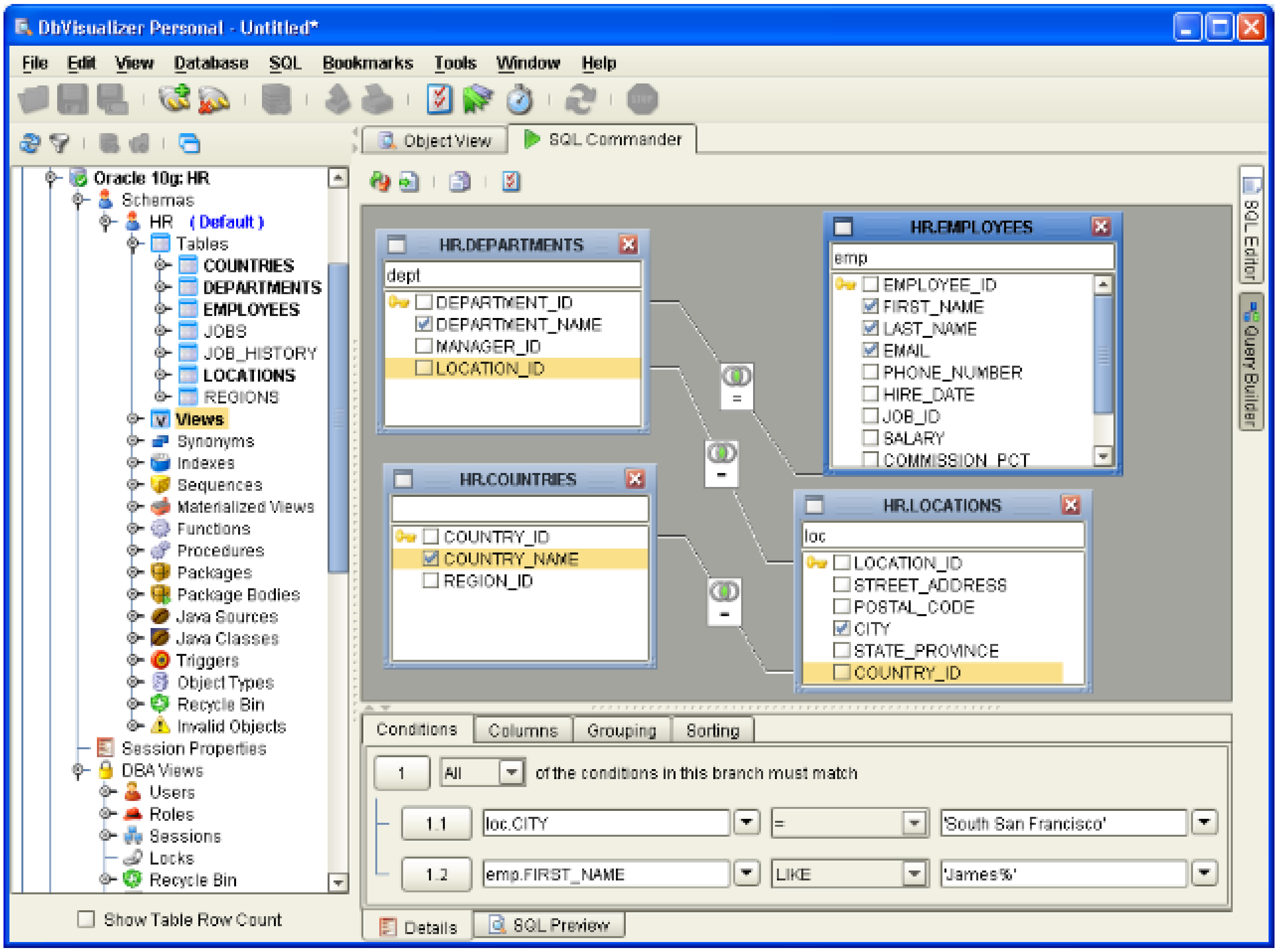
# Query Builder

## Introduction

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

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

**Note:** This document talks only about **Tables** even though the query builder supports both table and view objects.



1 All of the conditions in this branch must match

1.1 loc.CITY = 'South San Francisco'

1.2 emp.FIRST_NAME LIKE 'James%'

← Diagra

← Query

↑ Drag tables/views from objects tree

**Figure: The query builder**

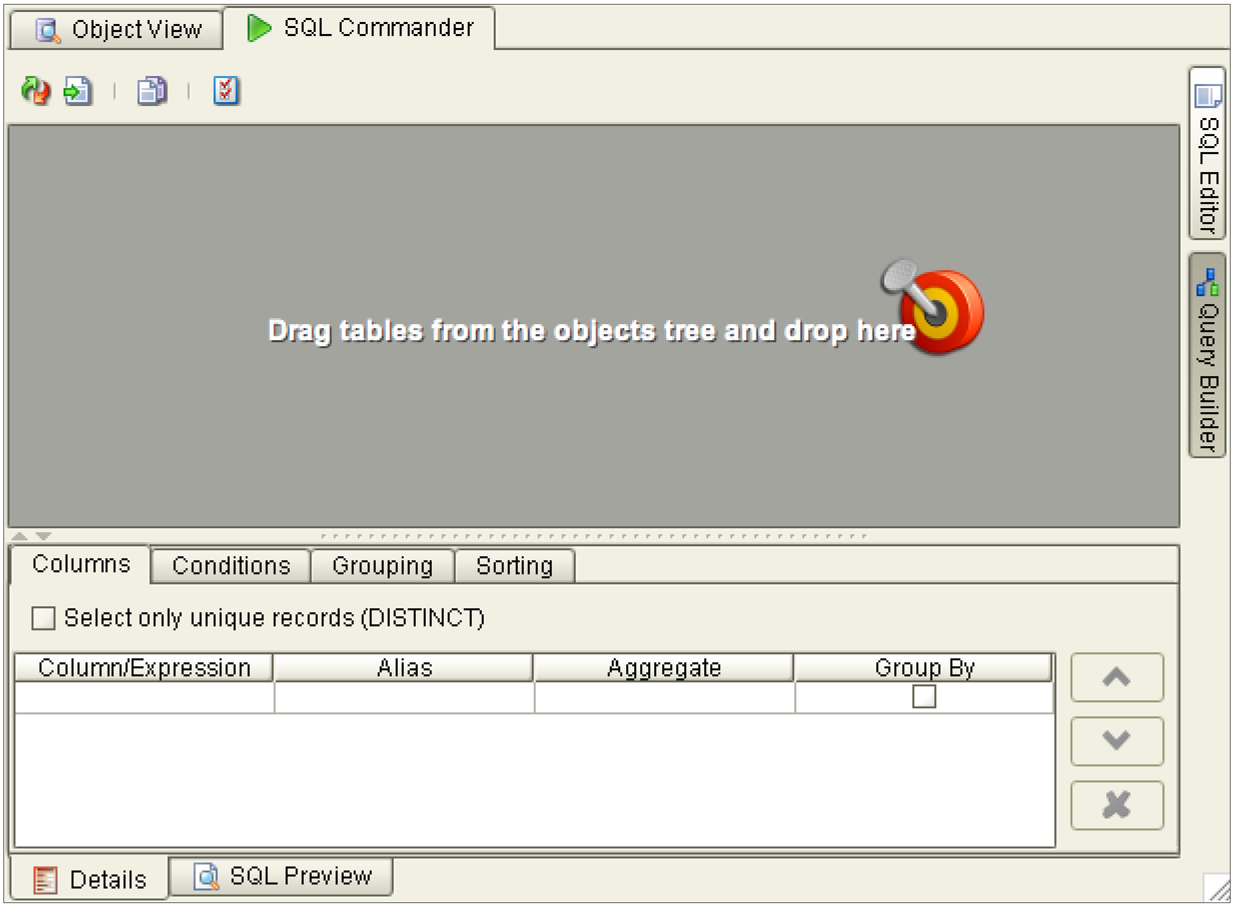
### Current Limitations

These are the current limitations in the query builder:

- It is not possible to generate a query builder definition from on an existing SQL statement (reverse engineer)
- Unions and sub selects are not supported.
- Not all join types are supported when joins are expressed as WHEREclause 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 ignored.

## Creating a Query

To create a query, make sure the SQL Commander tab is current and open the query builder using the **SQL->Show Query Builder** menu choice or Query Builder button as described earlier.



□ Select only unique records (DISTINCT)

| Column/Expression | Alias | Aggregate | Group By |
|---|---|---|---|
| | | | □ |

**Figure: The initial appearance of the query builder**

The easiest way to jump between the query builder and the SQL editor is by clicking the vertical control buttons to the right of the editor. 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 query builder toolbar buttons at the top of the query builder:

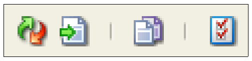


**Figure: Query builder toolbar**

1. The first button (from left) replaces the content of the SQL editor with the query SQL
2. The second button adds the query last in the SQL editor
3. The third button copies the query to the system clipboard
4. The fourth button opens the editor properties

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

## Adding Tables

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

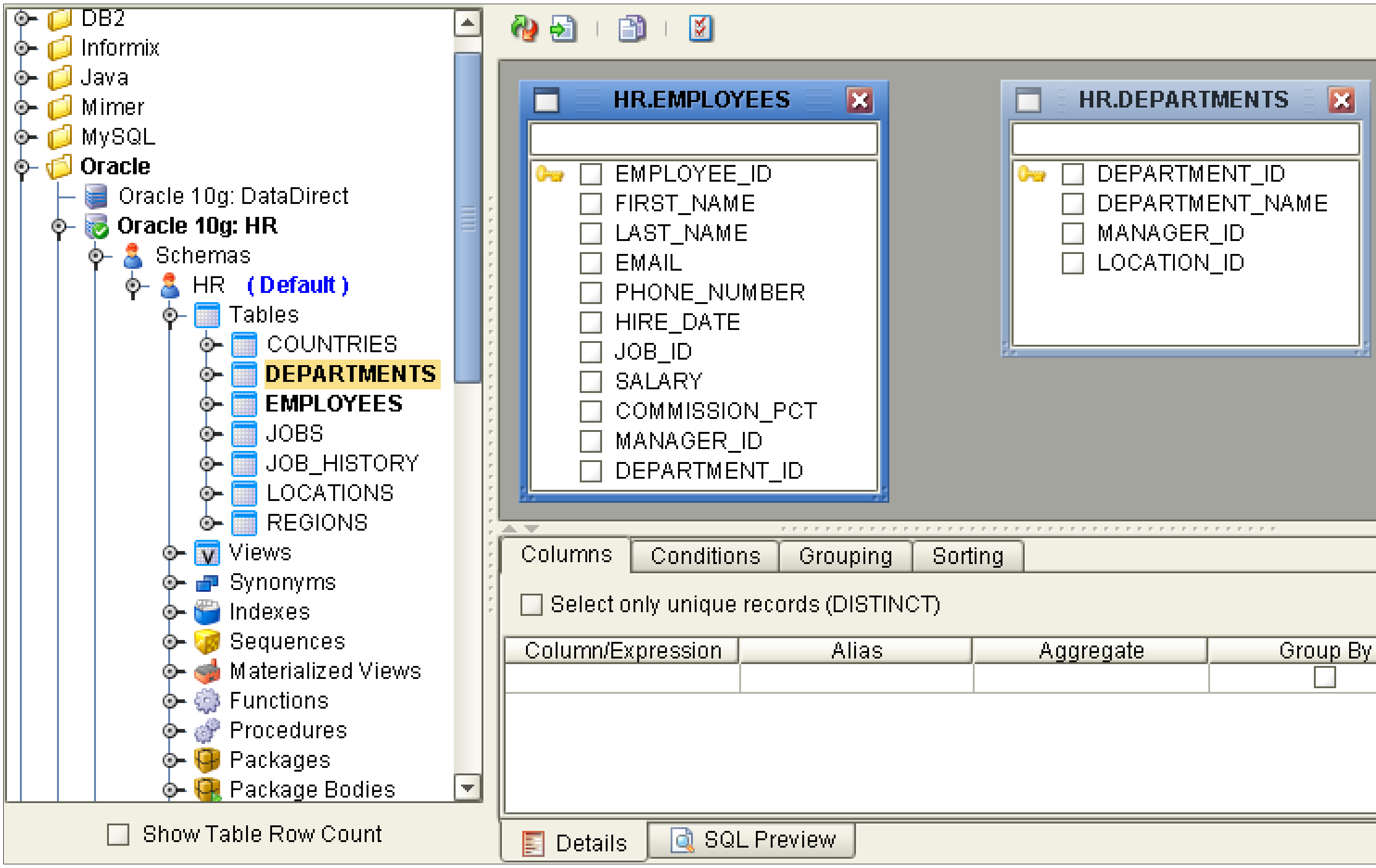


**Figure: Adding tables to the query builder**

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

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

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

## Joining Tables

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

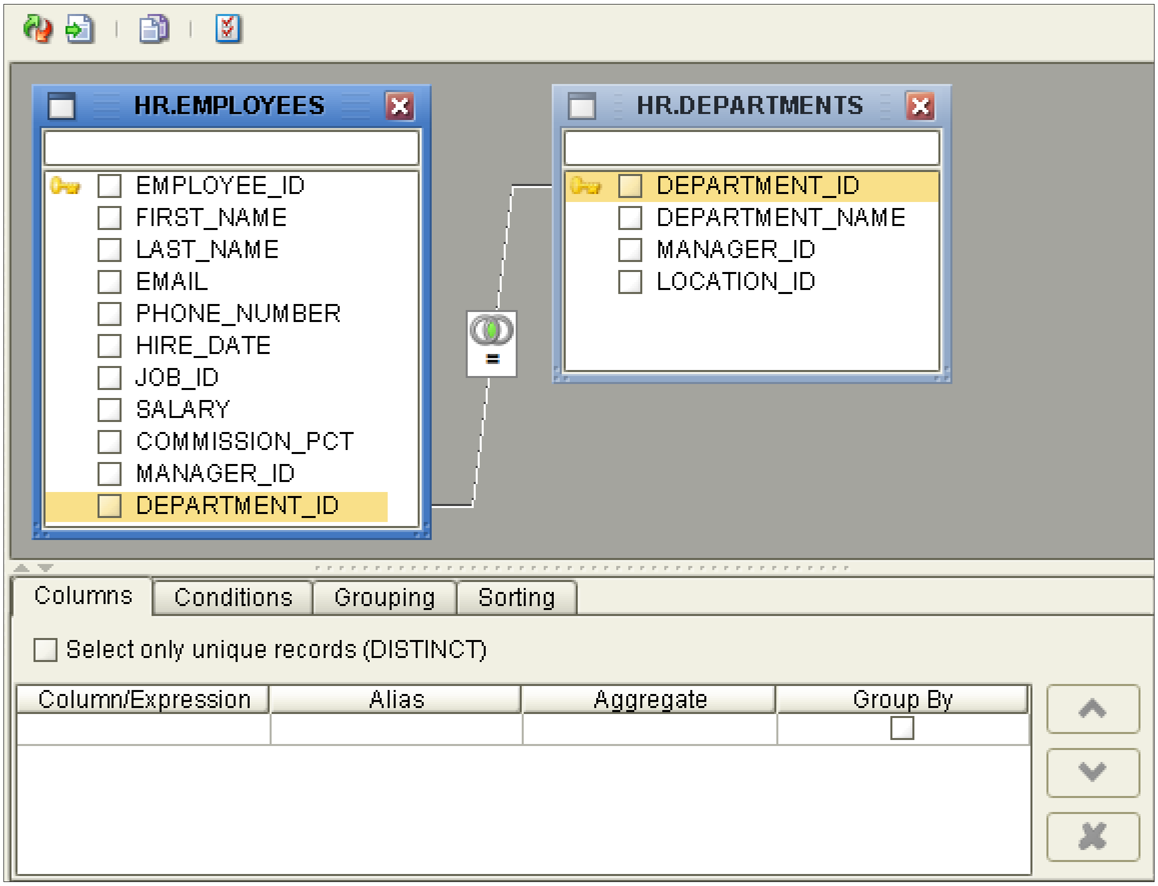


**Figure: Joining two tables**

The two columns now represents a join condition, represented by 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.

### Join Properties

The join characteristics can be modified by either double-clicking the icon or selecting Join Properties from the right click menu. The join properties window shows the source and target table columns and the conditional operator.

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.

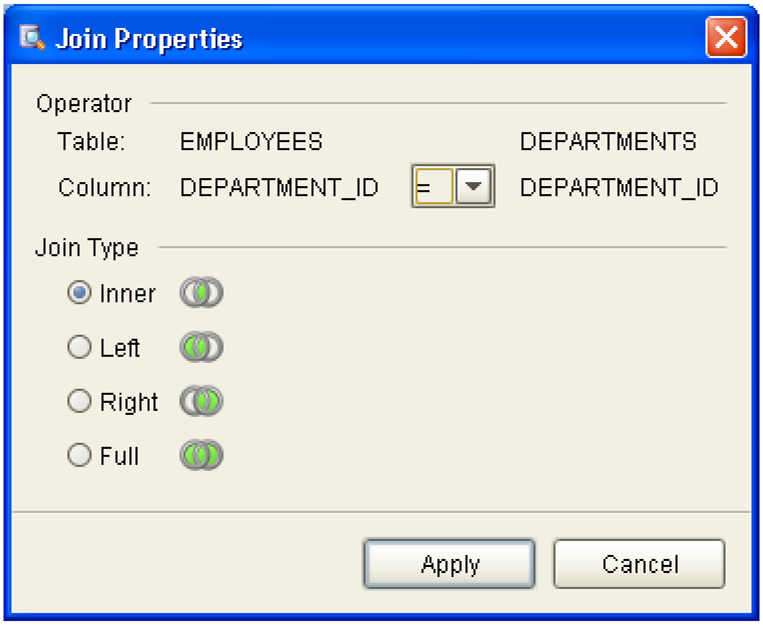


Table: EMPLOYEES DEPARTMENTS

Column: DEPARTMENT_ID = DEPARTMENT_ID

● Inner

○ Left

○ Right

○ Full

**Figure: Join Properties window**

You can change the join type and the conditional operator in the Join Properties dialog.

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

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

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

## Remove Tables and Joins

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

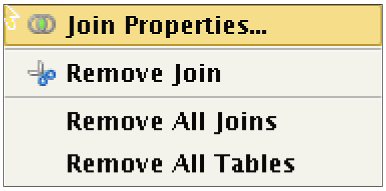


**Figure: Diagram right click menu**

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

## Query Details

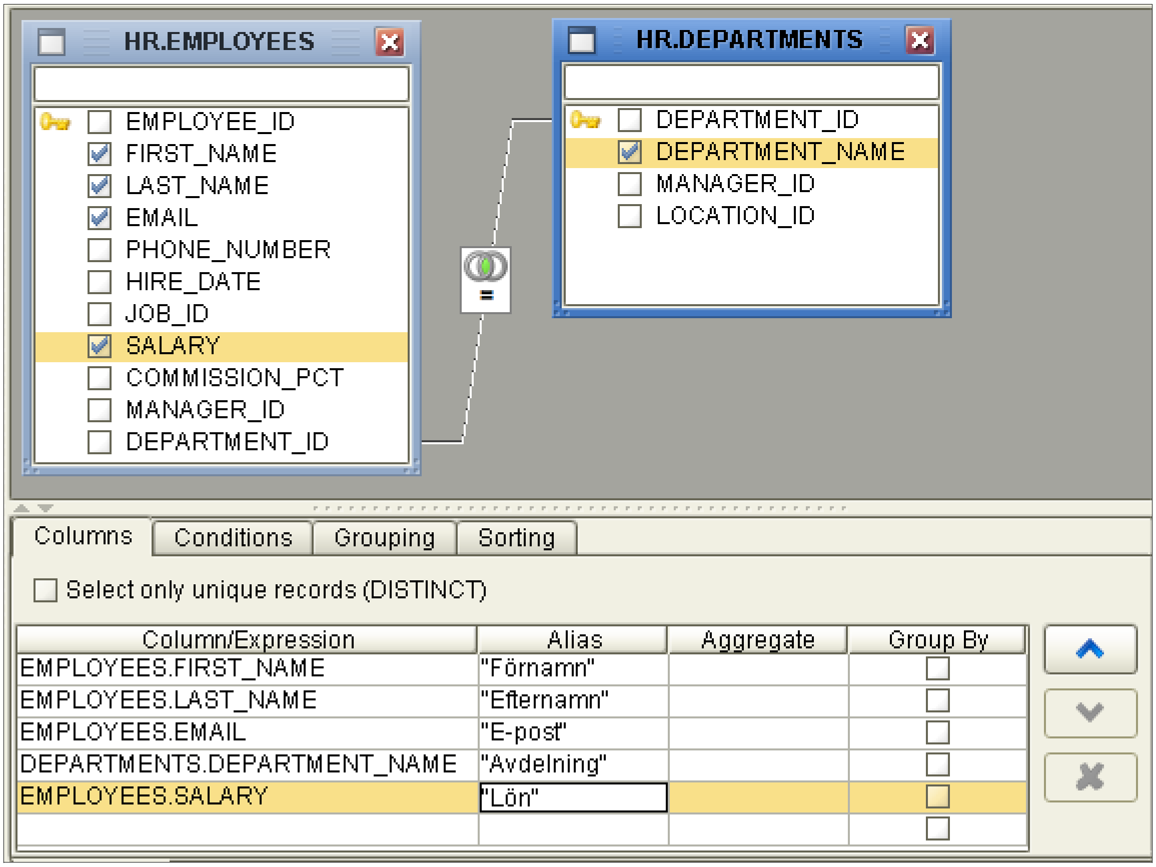
The Details tabs below the diagram area are used to define the various parts of the query. The tabs briefly represents the following parts of the final SQL:

```
  SELECT <Columns>
    FROM <tables>
   WHERE <Conditions>
GROUP BY <Columns>
  HAVING <Grouping>
ORDER BY <Sorting>
```

(The **<tables>** clause is defined in the diagram).

### Columns

Use the Columns tab to specify characteristics of the columns that are included in the query. The list is initially empty until a column is checked in a table window or  if manually adding a column expression. Columns will appear in the list in the same order as they are checked but may be manually 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

□ Select only unique records (DISTINCT)

| Column/Expression | Alias | Aggregate | Group By |
|---|---|---|---|
| EMPLOYEES.FIRST_NAME | "Förnamn" | | □ |
| EMPLOYEES.LAST_NAME | "Efternamn" | | □ |
| EMPLOYEES.EMAIL | "E-post" | | □ |
| DEPARTMENTS.DEPARTMENT_NAME | "Avdelning" | | □ |
| EMPLOYEES.SALARY | "Lön" | | □ |
| | | | □ |

**Figure: The columns tab**

The previous screenshot shows a total of 5 checked columns in the two tables. These are presented in the columns list by their full column identifier (not by any table alias). To remove a column from the list then uncheck the corresponding column in the table window.

The **alias** field is used to specify an optional alias identifier for the column. The alias will be the identifier for the column in the final query and will also appear as the column name in the result set produced by the query. Check the documentation for the actual database whether 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

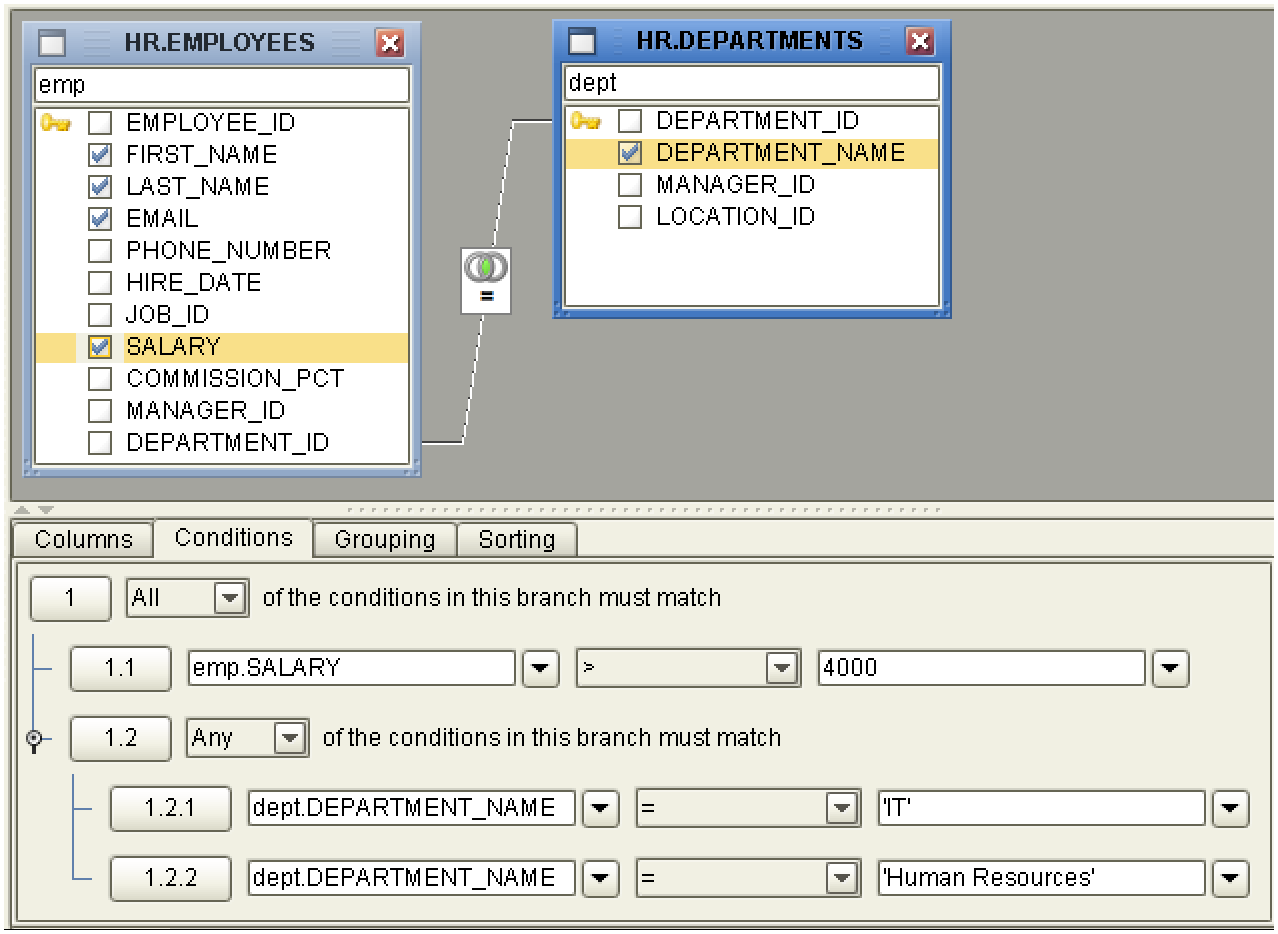
If an aggregate is selected then all non-aggregate columns must be **Group By** enabled.

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

## Conditions

This Conditions tab is used to manage the **WHERE** statement for the query. A where statement may consist of several where conditions each connected by AND or OR. The evaluation order for each where condition is defined by indention 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.



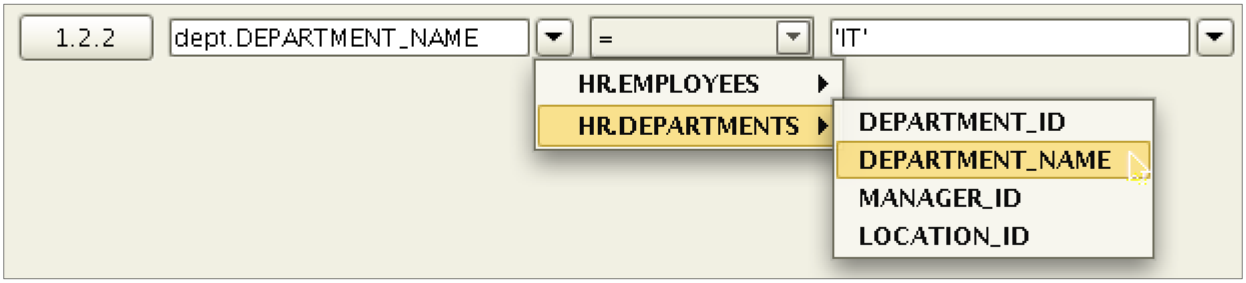
1 All of the conditions in this branch must match
1.1 emp.SALARY > 4000
1.2 Any of the conditions in this branch must match
1.2.1 dept.DEPARTMENT_NAME = 'IT'
1.2.2 dept.DEPARTMENT_NAME = 'Human Resources'

**Figure: Condition settings**

To create a new where condition press the indexed button in the list. In the menu that is displayed you may choose to create a new condition on the same level, a compound condition or delete current.
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 previous conditions 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. Pick columns from the list instead of typing these manually.



1.2.2 dept.DEPARTMENT_NAME = 'IT'

**Figure: List of columns in the conditions tab**

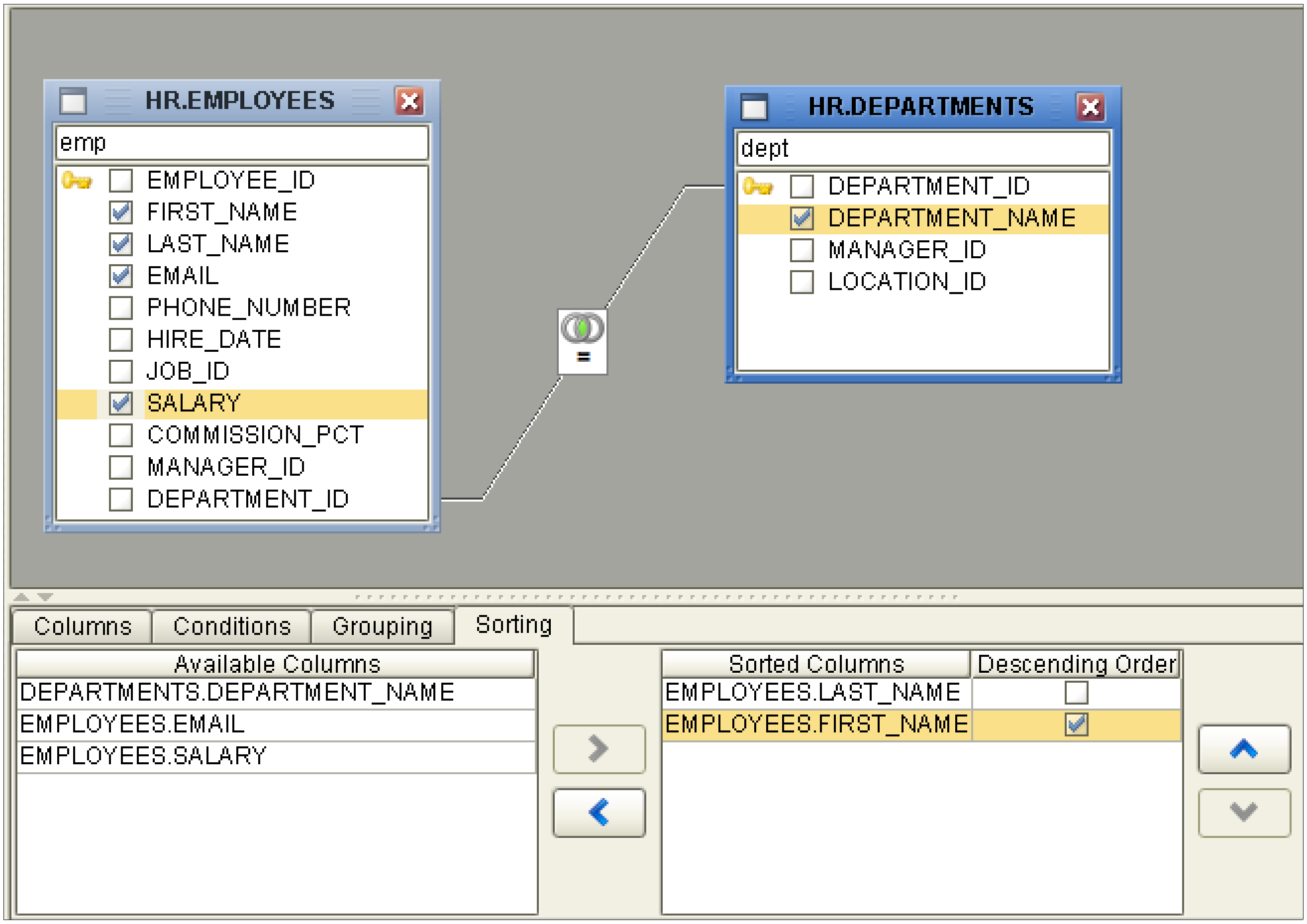
The values specified in the input fields will be used in the query exactly as specified. You need to manually quote text data.

### Grouping

The grouping tab is used to define the conditions for the **HAVING** statement. The capabilities for this statement are the same as for WHEREstatements except that grouping process the result set after any summary functions has been applied. Please read the Conditions section for more information.

### Sorting

The sorting tab is used to specify how the final result set will be sorted. The listed columns are the same as in the Columns tab.



HR.EMPLOYEES — emp

- □ EMPLOYEE_ID
- ☑ FIRST_NAME
- ☑ LAST_NAME
- ☑ EMAIL
- □ PHONE_NUMBER
- □ HIRE_DATE
- □ JOB_ID
- ☑ SALARY
- □ COMMISSION_PCT
- □ MANAGER_ID
- □ DEPARTMENT_ID

HR.DEPARTMENTS — dept

- □ DEPARTMENT_ID
- ☑ DEPARTMENT_NAME
- □ MANAGER_ID
- □ LOCATION_ID

| Available Columns |
|---|
| DEPARTMENTS.DEPARTMENT_NAME |
| EMPLOYEES.EMAIL |
| EMPLOYEES.SALARY |

| Sorted Columns | Descending Order |
|---|---|
| EMPLOYEES.LAST_NAME | □ |
| EMPLOYEES.FIRST_NAME | ☑ |

**Figure: The sorting tab**

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

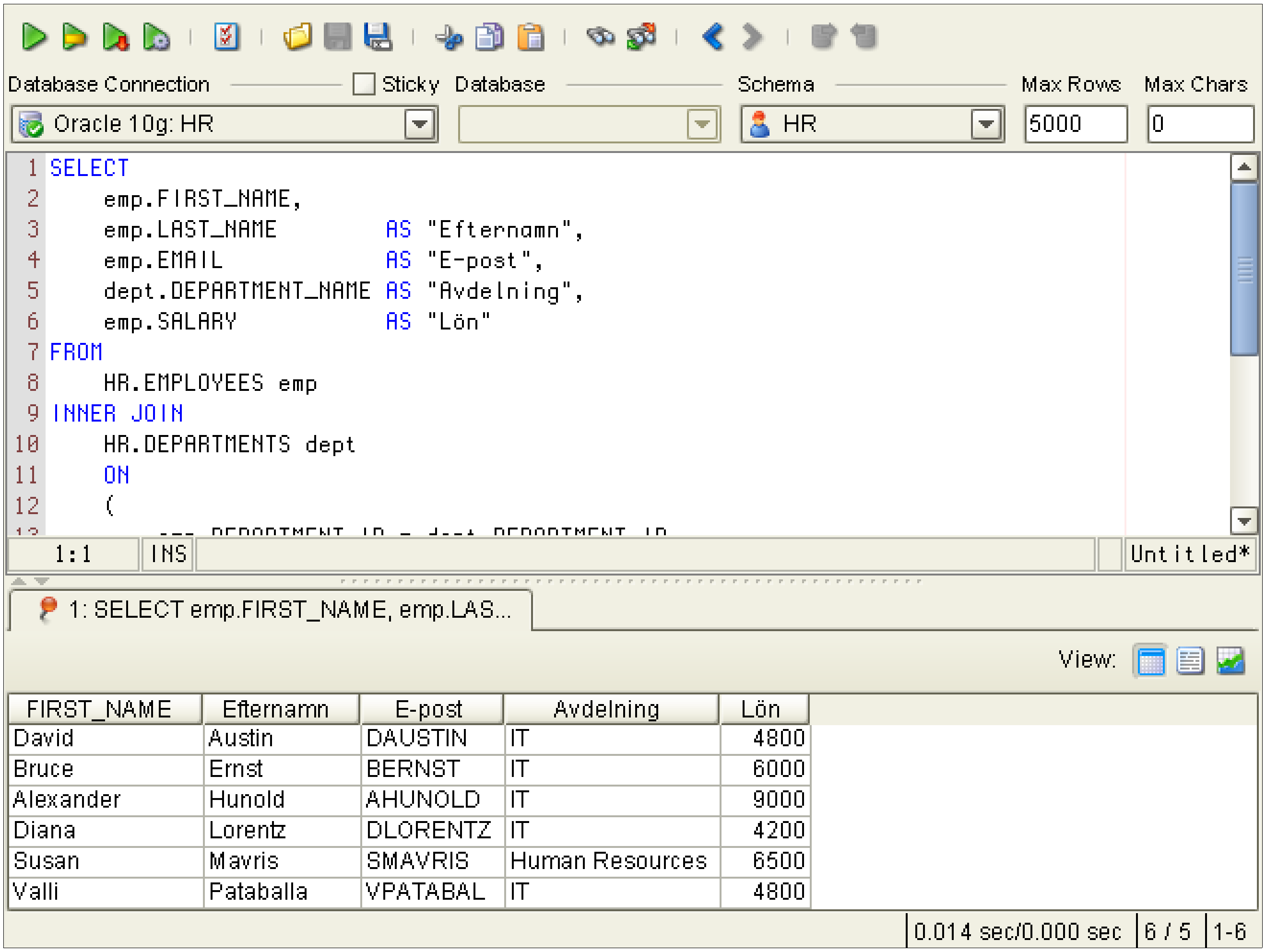
In the Sorted Columns table, you can change the default sort order (ascending) by clicking the check box in the **Descending Order** column. You can remove columns from the sorting criteria by selecting them in the Sorted Columns table and clicking the Move Right button.

### SQL Preview

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

## Testing the Query

To test the query, simply press the appropriate toolbar buttons in the query builder to copy the SQL to the SQL editor. Then execute the SQL as usual in the SQL editor.



Database Connection: Oracle 10g: HR
☐ Sticky
Database:
Schema: HR
Max Rows: 5000
Max Chars: 0

```
SELECT
    emp.FIRST_NAME,
    emp.LAST_NAME        AS "Efternamn",
    emp.EMAIL            AS "E-post",
    dept.DEPARTMENT_NAME AS "Avdelning",
    emp.SALARY           AS "Lön"
FROM
    HR.EMPLOYEES emp
INNER JOIN
    HR.DEPARTMENTS dept
    ON
    (
```

1:1 INS Untitled*

1: SELECT emp.FIRST_NAME, emp.LAS...

| FIRST_NAME | Efternamn | E-post | Avdelning | Lön |
|---|---|---|---|---|
| David | Austin | DAUSTIN | IT | 4800 |
| Bruce | Ernst | BERNST | IT | 6000 |
| Alexander | Hunold | AHUNOLD | IT | 9000 |
| Diana | Lorentz | DLORENTZ | IT | 4200 |
| Susan | Mavris | SMAVRIS | Human Resources | 6500 |
| Valli | Pataballa | VPATABAL | IT | 4800 |

0.014 sec/0.000 sec | 6 / 5 | 1-6

**Figure: Testing the SQL**

To further refine the SQL press the Query Builder button and apply the neccessary changes.

# Properties controlling Query Builder

There are a few properties that control how the query builder works and the SQL it generates. Check the following sections for details.

## Express joins as JOIN clause or WHERE condition

This property is available via **Connection Properties->[Database Type]->SQL Editor->Generate JOIN Clause in SQL Builder.**
Joins can be expressed either via the standardized SQL notation or by database specific syntax. The database specific syntax is somewhat different between the supported databases and the **Full** outer join type is generally not supported. The default setting of this property is by 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 sets the correct join notation when generating joins as WHERE conditions for databases that support left and right joins using WHERE conditions.

## Table and Column Name qualifiers

Qualifying table names with the schema or database name and qualifying column names with table name are defined in **Connection Properties- >[Database Type]- >Qualifiers.**

## Delimited Identifiers

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

## Drag style and Diagram Size

In the editor properties it is possible to set what style the windows in the query builder diagram should have when moving them. It is also possible to set the default size for newly added table windows.

# Monitor and Charts

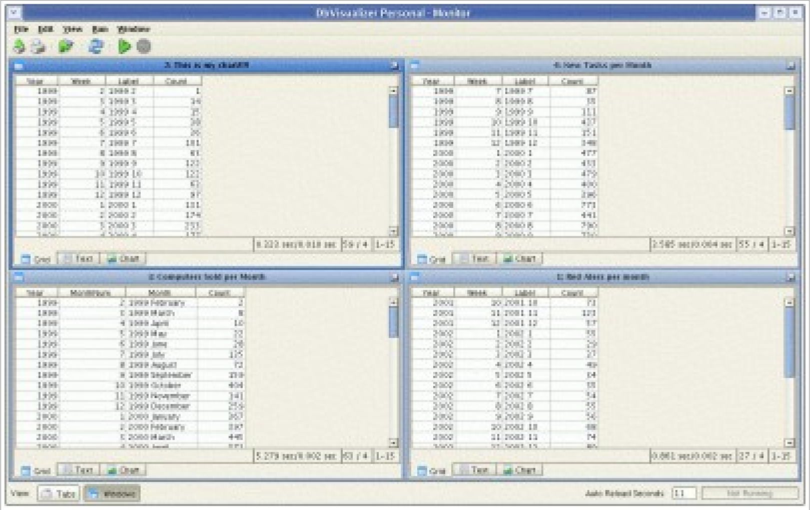
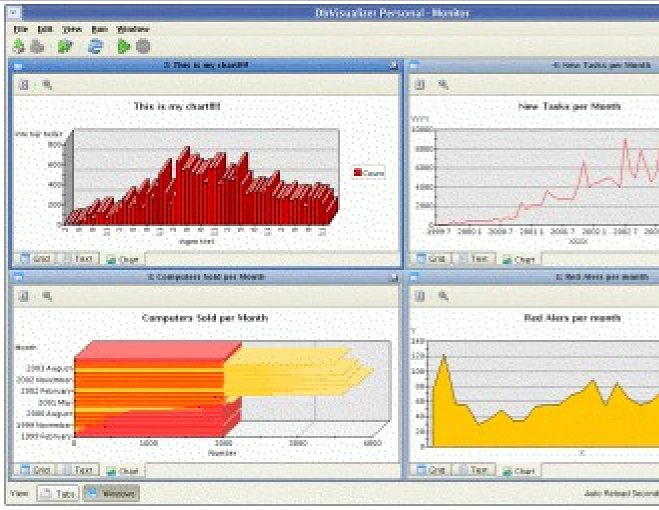
## Introduction

The monitor feature is used to show the results of one or many SQL statements in the Monitor window. These monitors can be updated manually or automatically based on an update period. A monitored SQL statement is an SQL Bookmark and the definition and management of which SQL Bookmarks are monitored is controlled in the Bookmark Editor. Any SQL Bookmark that produces a result set (data in a grid) can be monitored. The monitor feature supports monitoring SQL Bookmarks for different database connections concurrently.

The monitoring feature in conjunction with the charting capability in DbVisualizer Personal is really powerful since it delivers real time charts of many result sets simultaneously. Typical scenarios when monitoring is useful are to see live trends in a production database, surveillance, statistics, database metrics and so on. It is just a matter of imagination and the level of SQL expertise that sets the limit. We (Minq Software) as an example have a dedicated workstation that automatically presents live chart information from our internet servers and customer database.

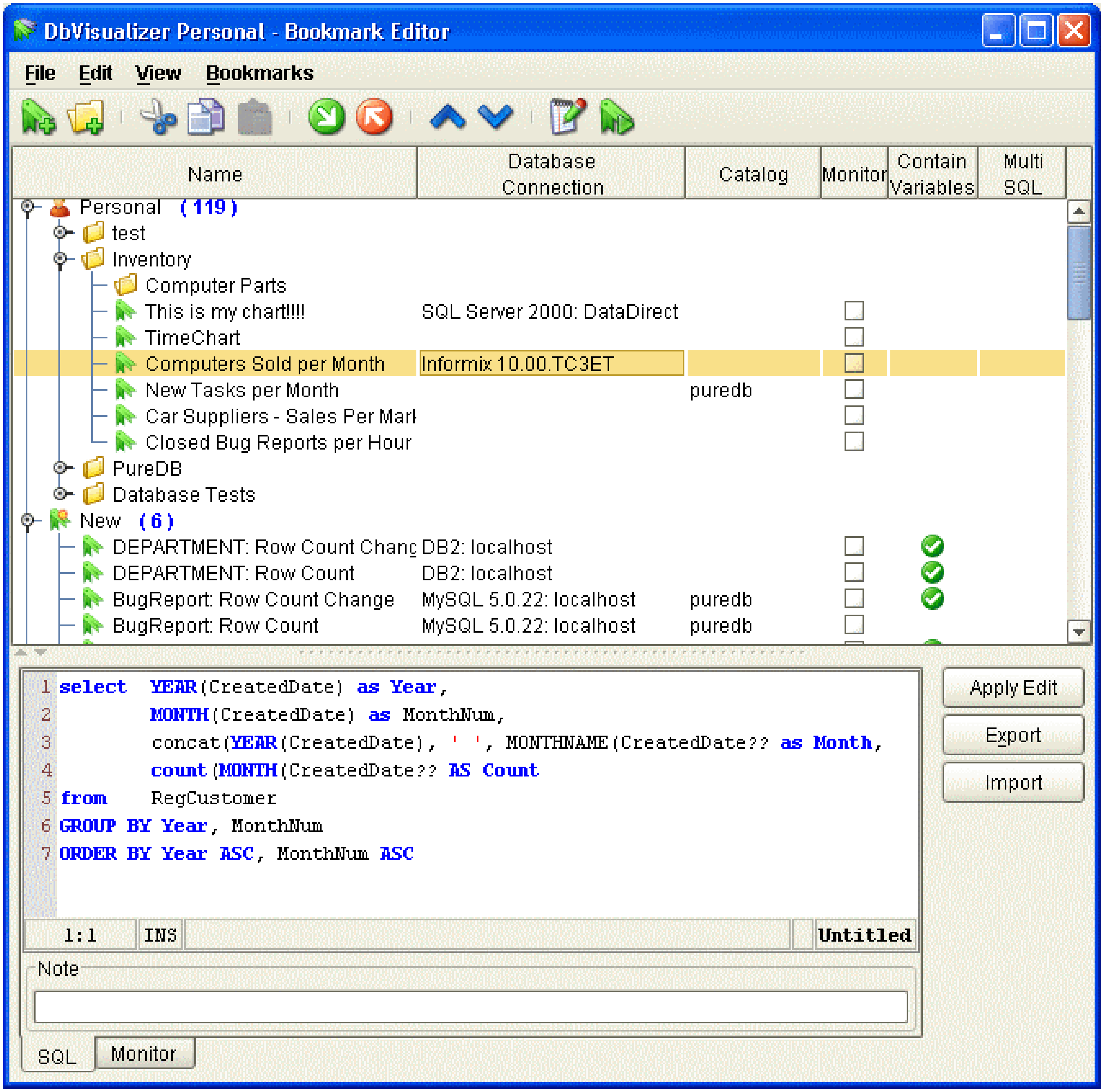
Charts can be exported to JPEG and PNG files.

**Note:** Charts cannot be printed directly. You must first export and then use another tool to print.
**Note:** The chart customization covered in this document is also applicable in the SQL Commander (DbVisualizer Personal).

| | |
|---|---|
|  |  |
| **The Monitor window with four monitored SQL Bookmarks. The results can be viewed as windows or tabs. This example shows the grid data as returned from each SQL statement.** | **The same monitored SQL Bookmarks as left figure but here presented as charts. (DbVisualizer Personal)** |

# Monitor an SQL statement

An SQL statement to be monitored must be defined as an SQL Bookmark in DbVisualizer. A bookmark is briefly an SQL statement with associated information about the target database connection and an optional catalog (generic JDBC denomination which translates to a *database* in for example Sybase, MySQL, SQL Server, etc). The Bookmark Editor supports organizing SQL bookmarks in a tree structured folder view and the complete structure and all SQL Bookmarks are saved in the XML file between invocations of DbVisualizer. It is the Bookmark Editor that is used to enable what SQL Bookmarks should be visible in the monitor feature.



| Name | Database Connection | Catalog | Monitor | Contain Variables | Multi SQL |
|---|---|---|---|---|---|
| Personal (119) | | | | | |
| test | | | | | |
| Inventory | | | | | |
| Computer Parts | | | | | |
| This is my chart!!!! | SQL Server 2000: DataDirect | | □ | | |
| TimeChart | | | □ | | |
| Computers Sold per Month | Informix 10.00.TC3ET | | □ | | |
| New Tasks per Month | | puredb | □ | | |
| Car Suppliers - Sales Per Marl | | | □ | | |
| Closed Bug Reports per Hour | | | □ | | |
| PureDB | | | | | |
| Database Tests | | | | | |
| New (6) | | | | | |
| DEPARTMENT: Row Count Chang | DB2: localhost | | □ | ✓ | |
| DEPARTMENT: Row Count | DB2: localhost | | □ | ✓ | |
| BugReport: Row Count Change | MySQL 5.0.22: localhost | puredb | □ | ✓ | |
| BugReport: Row Count | MySQL 5.0.22: localhost | puredb | □ | | |

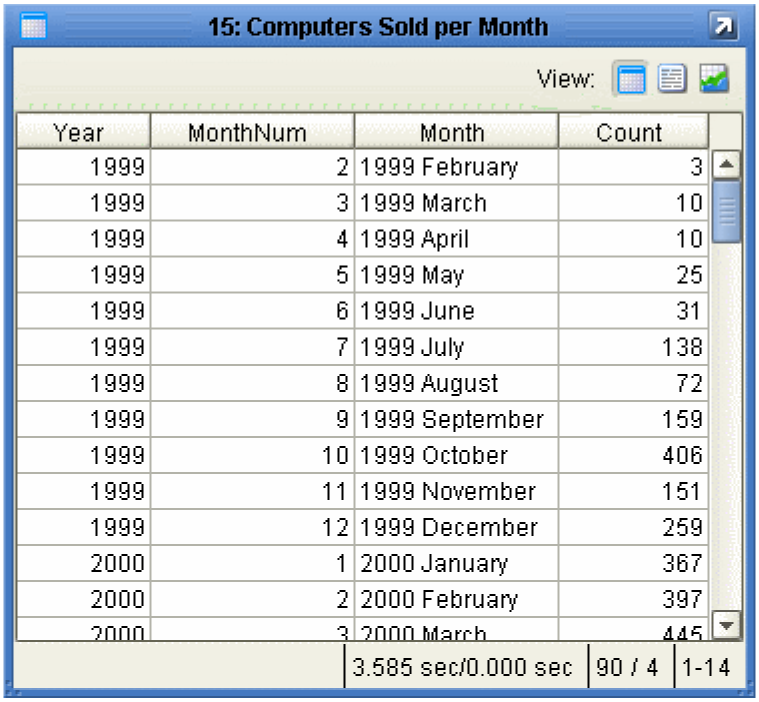
```
select  YEAR(CreatedDate) as Year,
        MONTH(CreatedDate) as MonthNum,
        concat(YEAR(CreatedDate), ' ', MONTHNAME(CreatedDate?? as Month,
        count(MONTH(CreatedDate?? AS Count
from    RegCustomer
GROUP BY Year, MonthNum
ORDER BY Year ASC, MonthNum ASC
```

```
1:1   INS                                   Untitled
```

**Figure: Bookmark Editor**

The figure shows the **Computers Sold per Month** bookmark and the SQL that is associated with it. The **Monitor** field in the tree is used to determine whether the SQL Bookmark is a monitor or not. Just click on the check mark and the SQL Bookmark will appear in the Monitor window. Uncheck it to remove the monitor.

The following is an example of what the above SQL Bookmark produces:



| Year | MonthNum | Month | Count |
|---|---|---|---|
| 1999 | 2 | 1999 February | 3 |
| 1999 | 3 | 1999 March | 10 |
| 1999 | 4 | 1999 April | 10 |
| 1999 | 5 | 1999 May | 25 |
| 1999 | 6 | 1999 June | 31 |
| 1999 | 7 | 1999 July | 138 |
| 1999 | 8 | 1999 August | 72 |
| 1999 | 9 | 1999 September | 159 |
| 1999 | 10 | 1999 October | 406 |
| 1999 | 11 | 1999 November | 151 |
| 1999 | 12 | 1999 December | 259 |
| 2000 | 1 | 2000 January | 367 |
| 2000 | 2 | 2000 February | 397 |
| 2000 | 3 | 2000 March | 445 |

**Figure: Monitor showing the result in Grid format**

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

There are alternative operations to simplify creation of monitored SQL Bookmarks. Read the following sections to find out how this is done.

## Monitor table row count

The Monitor Table Row Count operation is activated in the Data tab for a table (left button below):

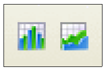


Figure: Data tab tool bar buttons that are used to create monitors

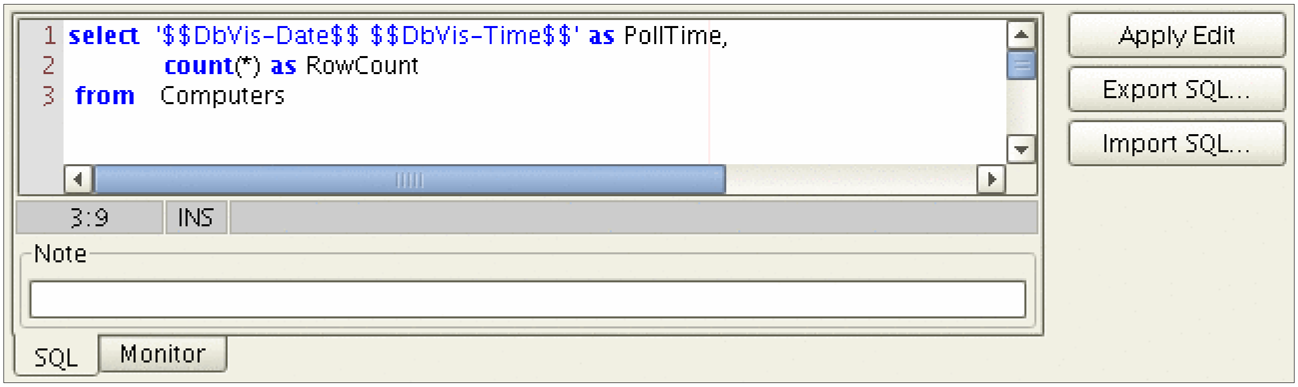
It is used to create a monitor that displays in a single row the current time stamp for when the monitor is executed and the total number of rows (count(*)) in the table. Each execution of the monitor will result in one row being added to the grid. The monitoring feature in this example keeps a pre defined number of rows until the oldest rows are removed.

Example:

| **Computers: Row Count** | |
|---|---|
| **PollTime** | **RowCount** |
| 2003-01-23 12:19:10 | 43123 |
| 2003-01-23 12:11:40 | 43139 |
| 2003-01-23 12:21:10 | 43143 |
| 2003-01-23 12:22:40 | 43184 |
| ... | ... |

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

The SQL for this monitor introduces two variables, **DbVis-Date** and **DbVis-Time**. These variables are substituted with the current date and time formatted according to the formats in Tool Properties. The reason these variables are used instead of using appropriate SQL functions to retrieve them is simply because it is almost impossible to get the values of these in a database independent way. Another reason is that we want to set the time of the client machine rather than the database. The SQL can of course be modified to contain whatever SQL that is appropriate as long as the **PollTime** and **RowCount** labels are not changed.

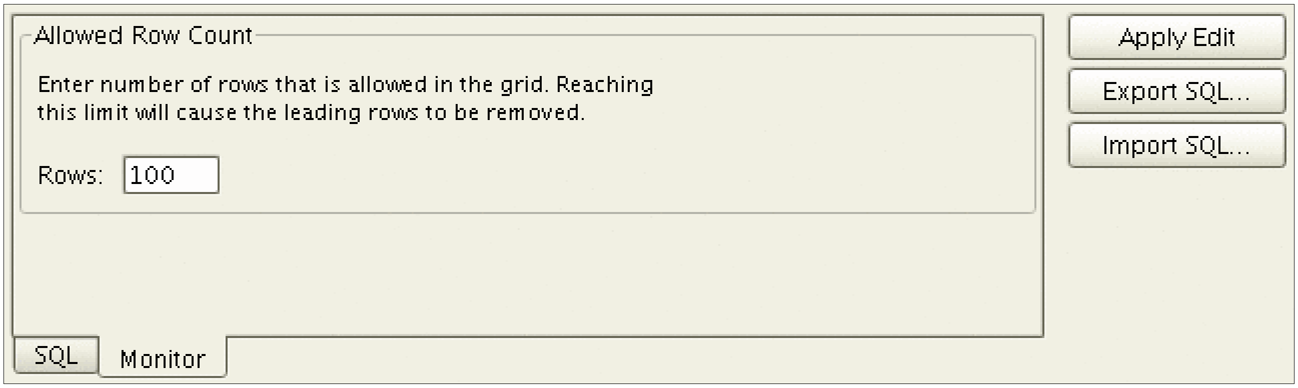


```
select '$$DbVis-Date$$ $$DbVis-Time$$' as PollTime,
       count(*) as RowCount
 from  Computers
```

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

The above does not introduce any big news for an SQL hacker.

The magic of this monitor is that it keeps a pre defined number of rows in the grid. This is managed by the **Allowed Row Count** property in the Bookmark Editor. This property is automatically set when creating a row count monitor. The default value is to keep the 100 latest rows added to the grid (one per execution).



Allowed Row Count

Enter number of rows that is allowed in the grid. Reaching this limit will cause the leading rows to be removed.

Rows: 100

**Figure: Allowed Row Count property pane**

The above setting can be modified to limit or extend the number of rows that the monitored grid will keep. Setting it to 0 or a negative number tells DbVisualizer to always clear the grid between executions of monitors.

## Monitor table row count difference

The Monitor Table Row Count Difference operation is activated in the Data tab for a table buttons (right button below):

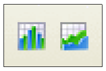


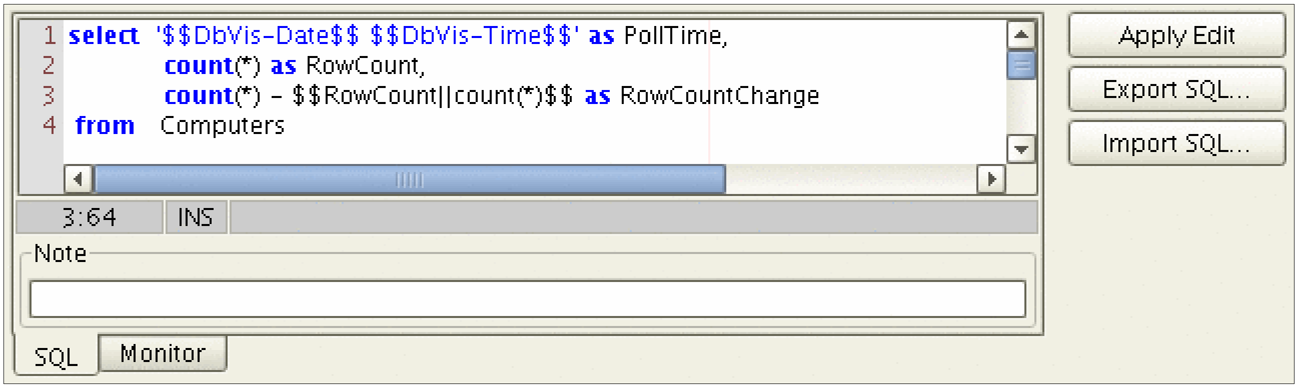
**Figure: Data tab tool bar buttons that are used to create monitors**

Its purpose is similar to Monitor Table Row Count except that this monitor reports the difference between the two latest executions in the result grid:

| Computers: Row Count Change | | |
|---|---|---|
| **PollTime** | **RowCount** | **RowCountChange** |
| 2003-01-23 12:19:10 | 43123 | 0 |
| 2003-01-23 12:11:40 | 43139 | 16 |
| 2003-01-23 12:21:10 | 43143 | 4 |
| 2003-01-23 12:22:40 | 43184 | 41 |
| ... | ... | ... |

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

The SQL for this monitor adds a third field which is the **RowCountChange**. It is rather simple since the current count(*) in the table is used when subtracting the RowCount in the previous execution round or count(*) if there are no previous rows in the grid. This gives the difference. The trick here is that DbVisualizer always keeps all values of the last row that was added in the grid. Any of its fields can be referenced in the succeeding execution of the monitor.

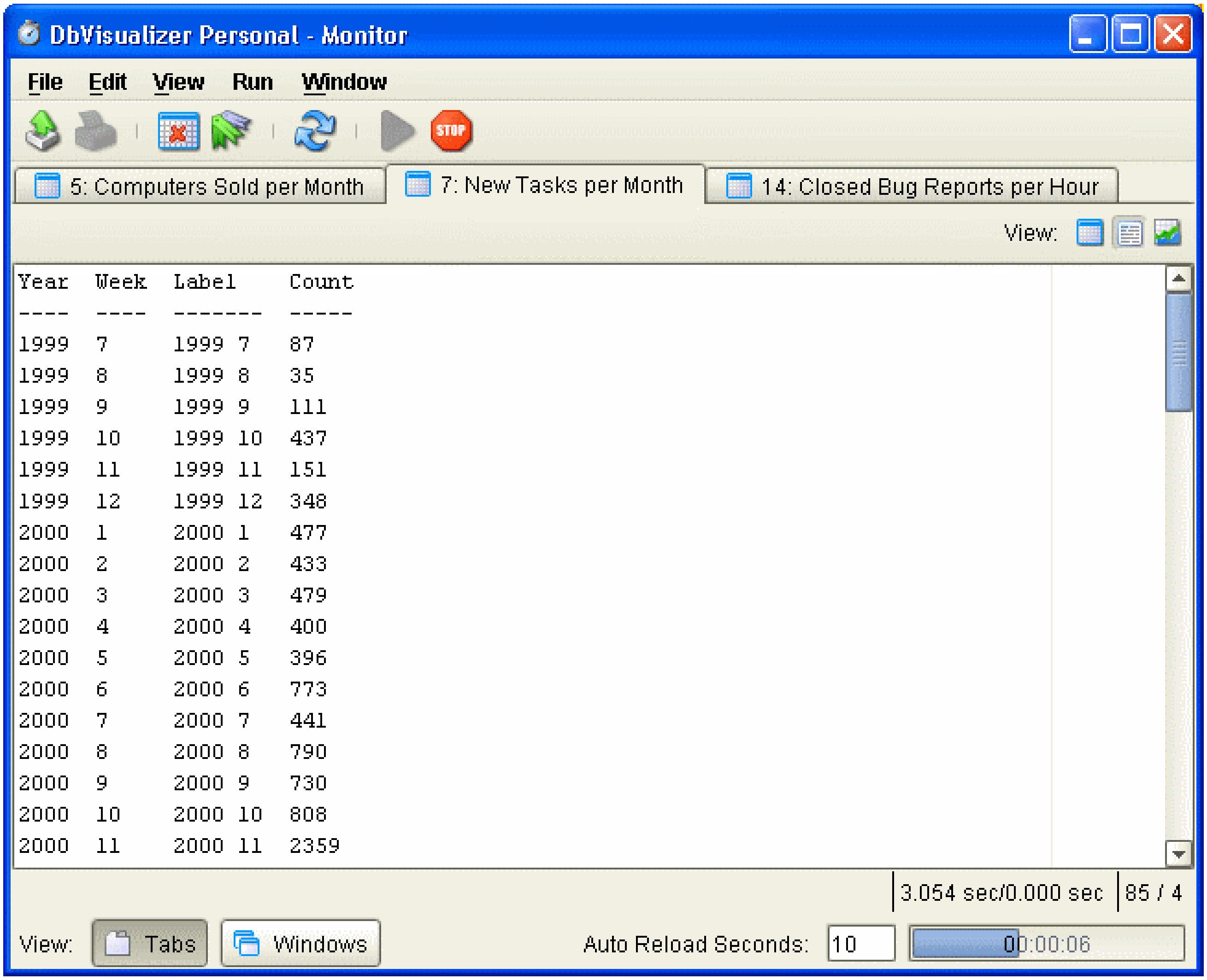


```
select '$$DbVis-Date$$ $$DbVis-Time$$' as PollTime,
        count(*) as RowCount,
        count(*) - $$RowCount||count(*)$$ as RowCountChange
from   Computers
```

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

## Monitor window

The Monitor feature launched via the **Tools->Monitor** menu option is used to browse the active monitors. The monitors can be organized either as tabs or internal windows. In DbVisualizer Free the monitor results can be viewed as grids while DbVisualizer Personal adds the capability to view them as charts. The following figure is a screen shot of the Monitor window:


DbVisualizer Personal - Monitor
File
Edit
View
Run
Window
STOP
5: Computers Sold per Month
7: New Tasks per Month
14: Closed Bug Reports per Hour
View:
Tabs
Windows

```
Year  Week  Label    Count
----  ----  -------  -----
1999  7     1999 7   87
1999  8     1999 8   35
1999  9     1999 9   111
1999  10    1999 10  437
1999  11    1999 11  151
1999  12    1999 12  348
2000  1     2000 1   477
2000  2     2000 2   433
2000  3     2000 3   479
2000  4     2000 4   400
2000  5     2000 5   396
2000  6     2000 6   773
2000  7     2000 7   441
2000  8     2000 8   790
2000  9     2000 9   730
2000  10    2000 10  808
2000  11    2000 11  2359
```

3.054 sec/0.000 sec | 85 / 4

Auto Reload Seconds: 10 00:00:06

**Figure: The Monitor window with all monitors organized as tabs**

The screen shot is from DbVisualizer Personal as the selected monitor has the **View** buttons at the top which are not there in DbVisualizer Free. The **Auto Reload** feature at the bottom of the main window is used to control whether auto update of all monitors is enabled or not. The **Seconds** field specifies how many seconds the Monitor feature should wait before doing an auto reload. If auto reloading is enabled then the monitor toolbar icon in the main window is displayed to indicate its state. The **Edit Bookmark** button is used to open the Bookmark Editor for the currently selected monitor. The Bookmark Editor will automatically locate the actual SQL Bookmark for the monitor.

**Note:** The specified number of seconds may be increased automatically by DbVisualizer if the total execution time for all monitors is longer.

The Window menu contains choices to control the appearance in the Monitor:

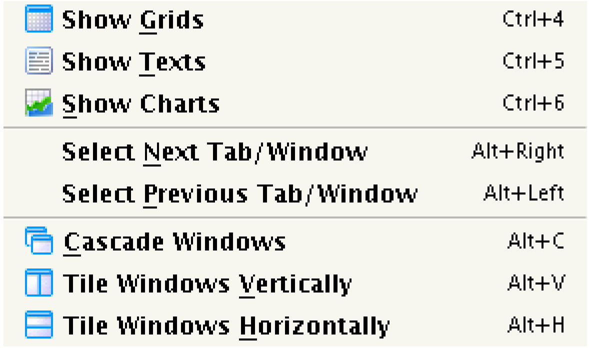

Show Grids Ctrl+4
Show Texts Ctrl+5
Show Charts Ctrl+6
Select Next Tab/Window Alt+Right
Select Previous Tab/Window Alt+Left
Cascade Windows Alt+C
Tile Windows Vertically Alt+V
Tile Windows Horizontally Alt+H

**Figure: Window menu operations**

The **Show Grids, Show Texts** and **Show Charts** toggles the monitors to display the monitors in the selected view. **Cascade** and **Tile** are used to automatically arrange the windows in the Windows view.

# Charts

**This section is only applicable for DbVisualizer Personal.**

Charts in conjunction with the Monitor feature is really powerful since monitored data is very often a good candidate to be charted. The charting capability in DbVisualizer Personal is also available in the SQL Commander feature even though this document does not cover it.

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

The chart view is controlled by sub toolbar for each monitor:

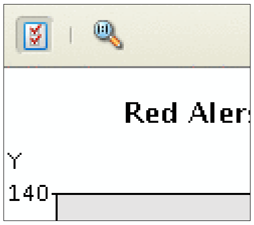


**Figure: Chart control buttons**

The controls are from the left:

1. Show/Hide chart controls pane
2. Reset any zoom

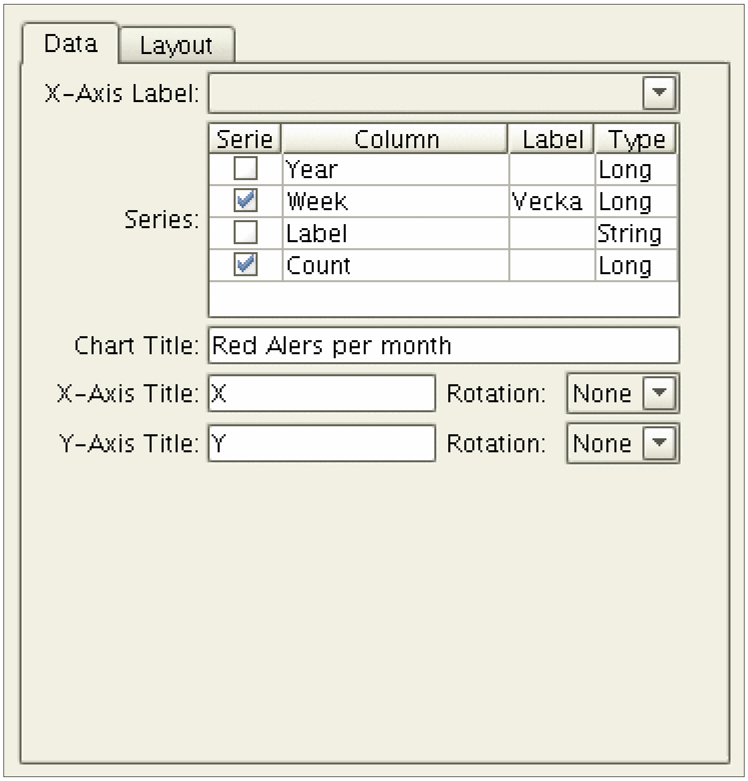
The following sections explain the features and how to setup the chart.

## Chart Controls

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

### Data

Specify in the Data customization which data shall appear in the chart.



X-Axis Label:

Series:

| Serie | Column | Label | Type |
|---|---|---|---|
| ☐ | Year | | Long |
| ☑ | Week | Vecka | Long |
| ☐ | Label | | String |
| ☑ | Count | | Long |

Chart Title: Red Alers per month

X-Axis Title: X Rotation: None

Y-Axis Title: Y Rotation: None

**Figure: Data customizer**

Select at least one **Series** from the list of columns and the chart is ready! Selecting several series will show them accordingly in the chart. The **Label** field can be used to specify an optional label for the series as it will appear in a legend. The name of the column is used if no label is specified.

The **X- Axis Label** box is used to specify the column in the result that should be used to render the labels of the X-axis. **Chart Title** specifies the main title of the chart. This is the same title as the SQL Bookmark in the Bookmark Editor. **X- Axis Title** and **Y- Axis Title** specifies the titles for the X and Y axis. The **Rotation** settings are used to set the rotation of the X and Y axis.

### Layout

The layout tab is used to configure the appearance of the chart and primarily what type of chart that will be displayed. Note that all settings are per monitor. The following screen shots show some of the most commonly used chart types.

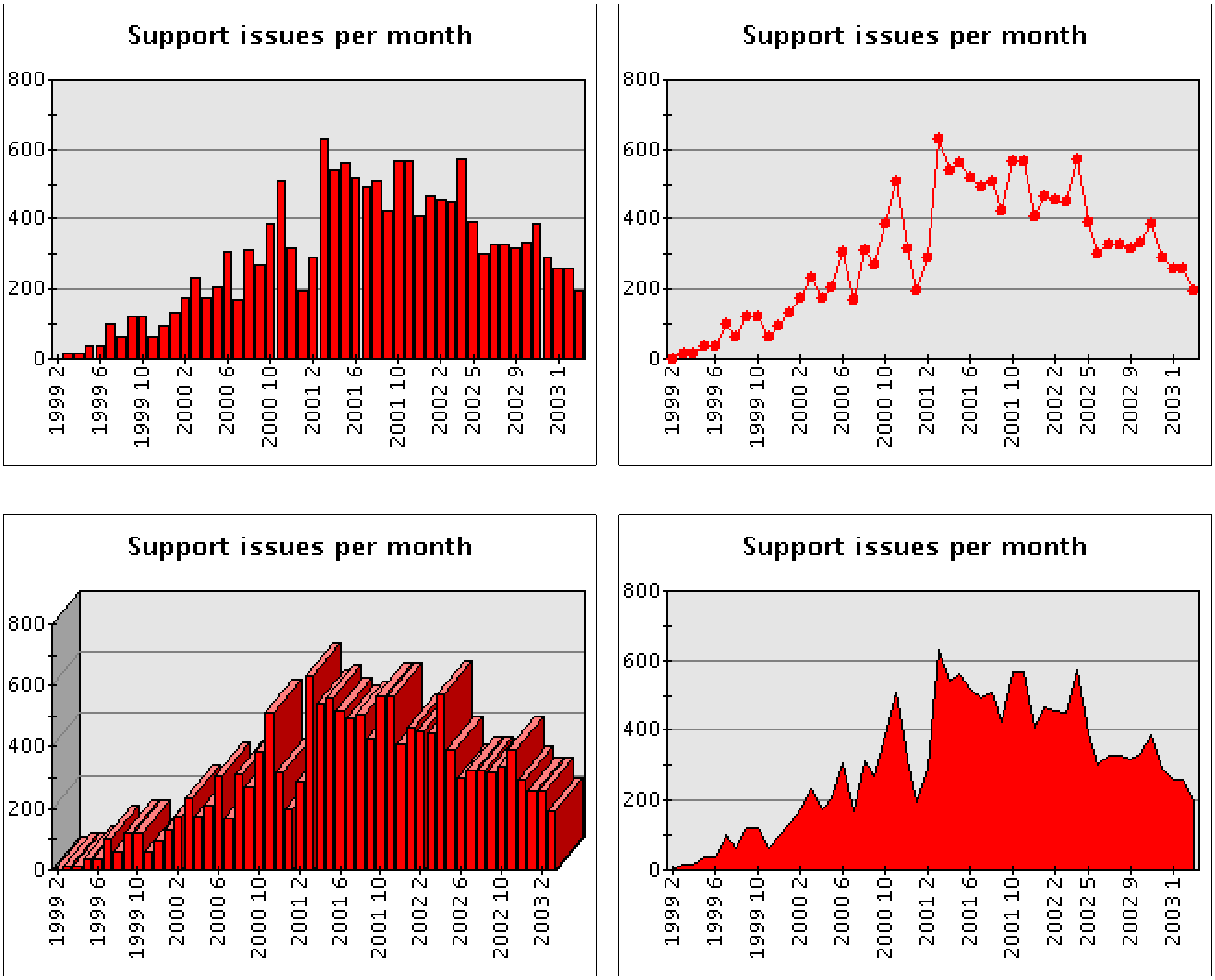
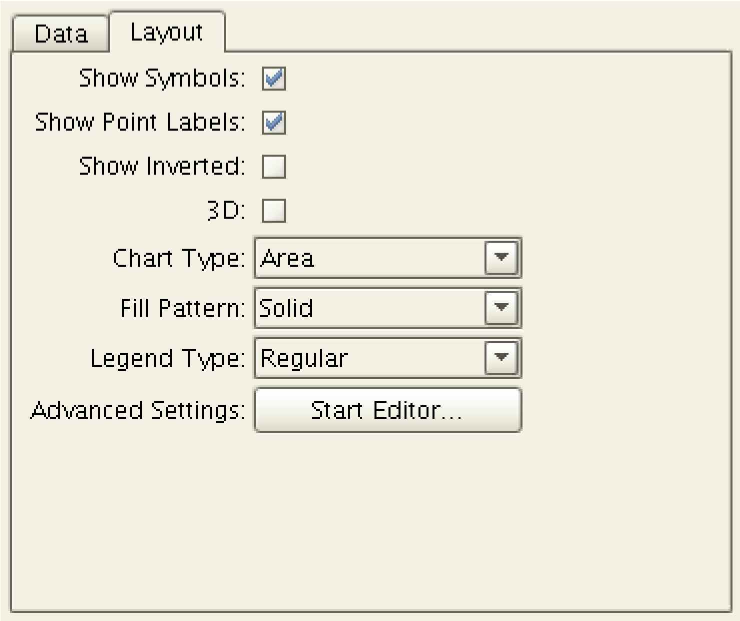


**Figure: Chart type examples**

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



Show Symbols: ☑

Show Point Labels: ☑

Show Inverted: □

3D: □

Chart Type: Area

Fill Pattern: Solid

Legend Type: Regular

Advanced Settings: Start Editor...

Figure: Layout customizer

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

The **Advanced Settings** editor is used to customize all the bits and pieces of the chart. This document does not explain all the configurations that can be done using this editor since that would result in a 100 page book.

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

## Chart View

### Zooming

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

### Rotating

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

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

Examples:

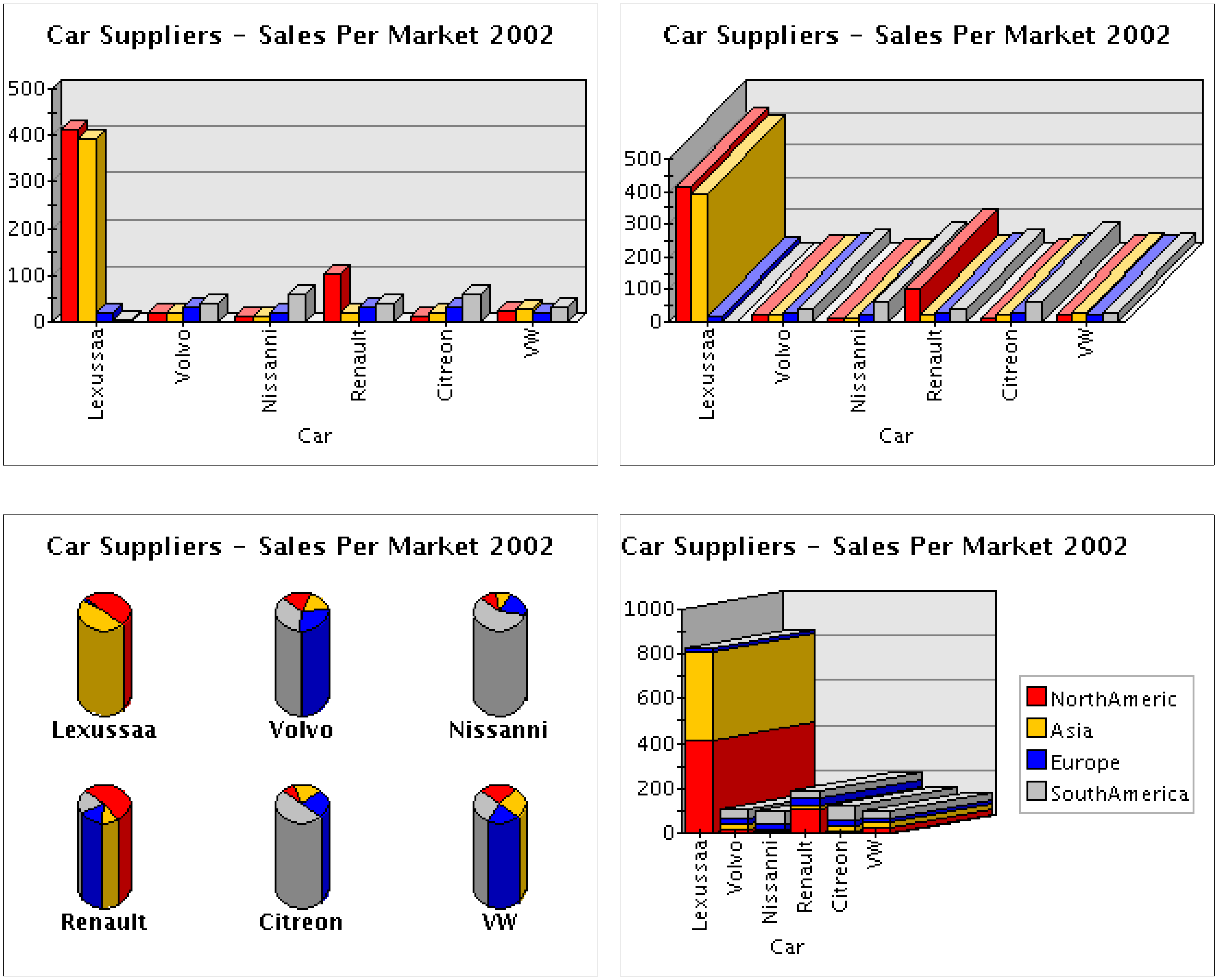
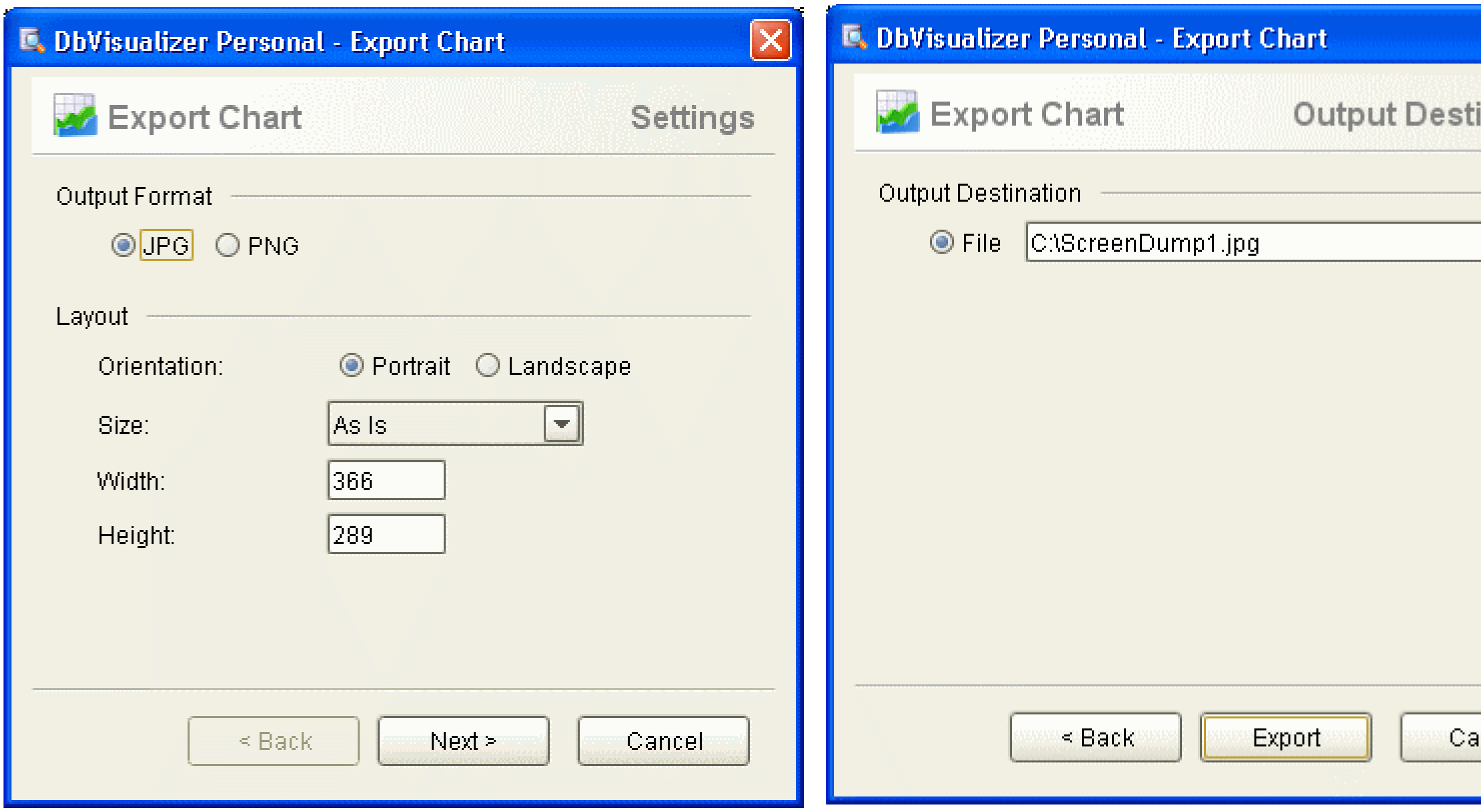


**Figure: Example of 3D charts**

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

## Export

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



Output Format

● JPG ○ PNG

Layout

Orientation: ● Portrait ○ Landscape

Size: As Is

Width: 366

Height: 289

Output Destination

● File C:\ScreenDump1.jpg

**Figure: Export dialog for charts**

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

# Edit Table Data

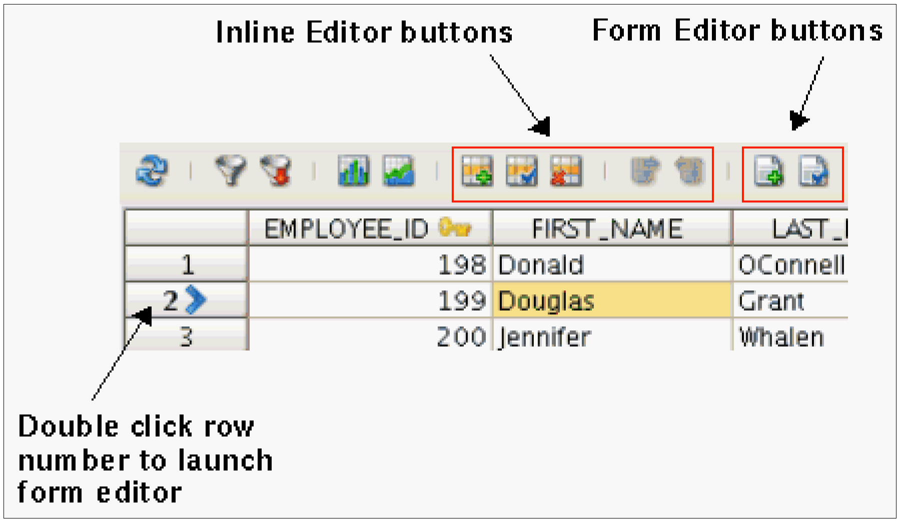
## Introduction

The editing support in DbVisualizer Personal is used to insert, update or remove single rows in a database table. Editing is performed using two different editors:

- Inline Editor
- Form Editor

The Inline Editor is convenient in situations when fast edits of single columns need to be made. The Form Editor presents all columns in a form and some users prefer it since it is easier to get an overview of the data.

The following figure shows what buttons in the **Data** tab and in a result set grid that are used specifically for editing.



Inline Editor buttons

Form Editor buttons

| | EMPLOYEE_ID | FIRST_NAME | LAST_I |
|---|---|---|---|
| 1 | 198 | Donald | OConnell |
| 2 | 199 | Douglas | Grant |
| 3 | 200 | Jennifer | Whalen |

Double click row number to launch form editor

**Figure: The buttons in the Data tab used to control the inline and form editors**

### Permissions

All of the insert, update and delete requests performed by the inline and form data editors may be confirmed before being executed by the database server. Specify in **Tool Properties->Permissions** the confirmation state. The default behavior is that delete operations must be confirmed while insert and update need no confirmation.

### Features that support editing

Editing of table data can be performed in the **Database Objects->Data** tab or in the results from an SQL statement in the **SQL Commander**.

There are a few rules that must be fulfilled in order to enable editing in the SQL

Commander:

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

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

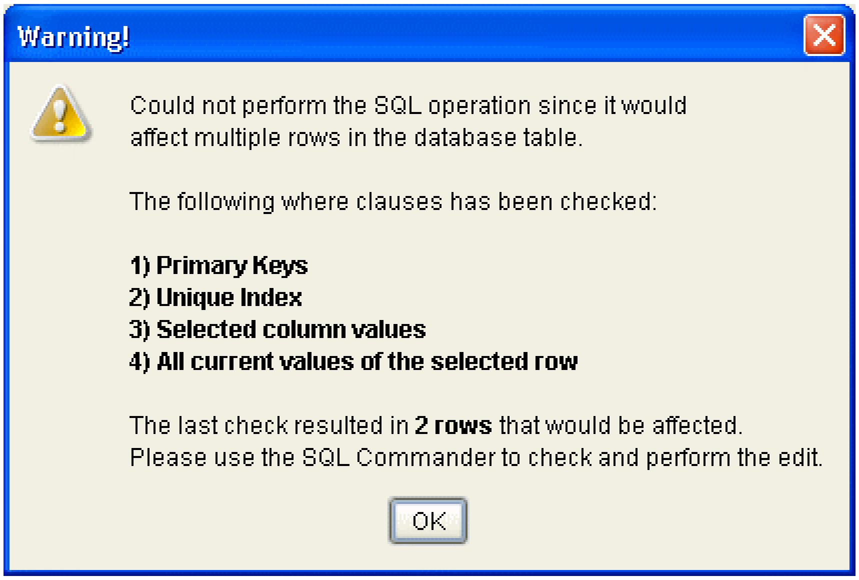
## Edits might be denied

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

1. Primary Key
2. Unique Index
3. Selected Columns
4. All Previous Values

The **Primary Key** concept is widely used in databases to uniquely identify the key columns in tables. If the table has a primary key then DbVisualizer will use it. There are situations when primary keys are not supported by a database or when primary keys are supported but not used. If no primary key is defined then DbVisualizer will check if there is a unique index. If there are several unique indices then DbVisualizer will pick one of them. So why not only try to identify the row by all its previous values? The reason **Selected Columns** is searched before checking **All Previous Values** is because not all data types are allowed to be used in a where clause. Typical examples are BLOB and CLOB data types. The **Selected Columns** check simply uses the selected columns in the inline editor or checked columns in the form editor to find out if the values identify a unique row. If that check fails then **All Previous Values** is searched.

The following dialog appears when none of these strategies can uniquely identify the row.



Could not perform the SQL operation since it would affect multiple rows in the database table.

The following where clauses has been checked:

**1) Primary Keys**
**2) Unique Index**
**3) Selected column values**
**4) All current values of the selected row**

The last check resulted in **2 rows** that would be affected.
Please use the SQL Commander to check and perform the edit.

**Figure: Warning dialog when a unique row could not be identified**

## Commit

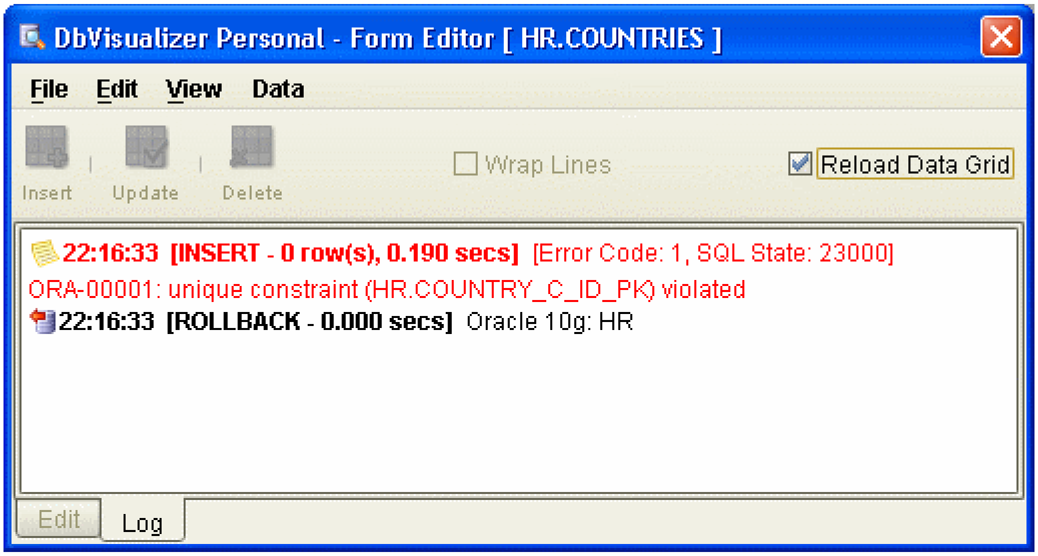
DbVisualizer issues an implicit database commit after each successful edit operation (the value of the **Auto Commit** property in Tool Properties is bypassed during editing). The commit request is per database connection and it commits all pending updates even those that have been executed in the SQL Commander.

The following example explains the effect of the implicit commit:

1. The **Auto Commit** property is disabled in Tool Properties.
2. An SQL statement that deletes some rows is executed in the SQL Commander but it is not explicitly committed.
3. We now go to the Data tab for a table and edit the value of a column.
4. Once the update in the grid is performed the update is committed. This commit will also commit the delete statement that was executed earlier in the SQL Commander. The reason is that all commits are performed per Database Connection.

## Error Log

If the insert, update or delete request is successfully executed by the database then the row in the grid will be updated accordingly. There are however situations when errors occur during the database execution phase. These messages are presented in the execution log.



```
22:16:33 [INSERT - 0 row(s), 0.190 secs] [Error Code: 1, SQL State: 23000]
ORA-00001: unique constraint (HR.COUNTRY_C_ID_PK) violated
22:16:33 [ROLLBACK - 0.000 secs] Oracle 10g: HR
```

**Figure: Error Log view**

The log keeps information of all edits that have been executed in both the inline and form editors. It presents not just failed operations (red color) but also all successful edits. All error messages that appear in the log are produced by either the JDBC driver or the database. Refer to the driver or database documentation for explanation of the error messages.

### Binary data/BLOB and CLOB

Binary data/BLOB and CLOB data can only be edited in the form editor. Read more about it in Editing Binary/BLOB data and CLOB.

## Inline Editor

The inline editor is handy when fast edits need to be made. The editor is simple to use since it is activated by typing characters in the cell (column in a row) that is to be modified. The inline editor is data type aware and checks that the entered values are valid for each column's data type. Any errors are reported in the error log. Only those cells that have been modified (indicated by a yellow color) during the editing session will be propagated to the database.

The inline editor keeps all edits that have been made in a single row until the edit is implicitly committed by selecting another row in the grid or by using the update tool bar button (see below). All other operations such as selecting another table in the **Database Objects** tree will silently revert any edit.

The following tool bar buttons are used to control the inline editor:

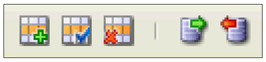


**Figure: Tool bar buttons to control the inline editor**

Description of the buttons (from left):

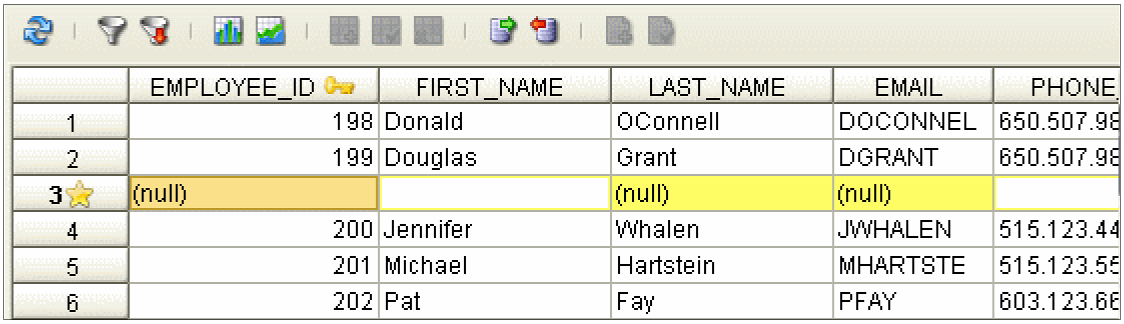
- Insert row
- Duplicate row
- Delete row
- Perform the current edit
- Revert the currently edited row

A row is edited by typing characters in a cell. All cells that have been edited are indicated by a yellow background color. Only these columns will be updated once the final SQL is executed to perform the change. All cells irrespective of whether they have been edited or not are highlighted with a yellow border to indicate which row is being edited.

*It is not possible to edit binary data, BLOB and CLOB values in the inline editor. Use the form editor to manipulate values for these data types.*

### Insert a new row

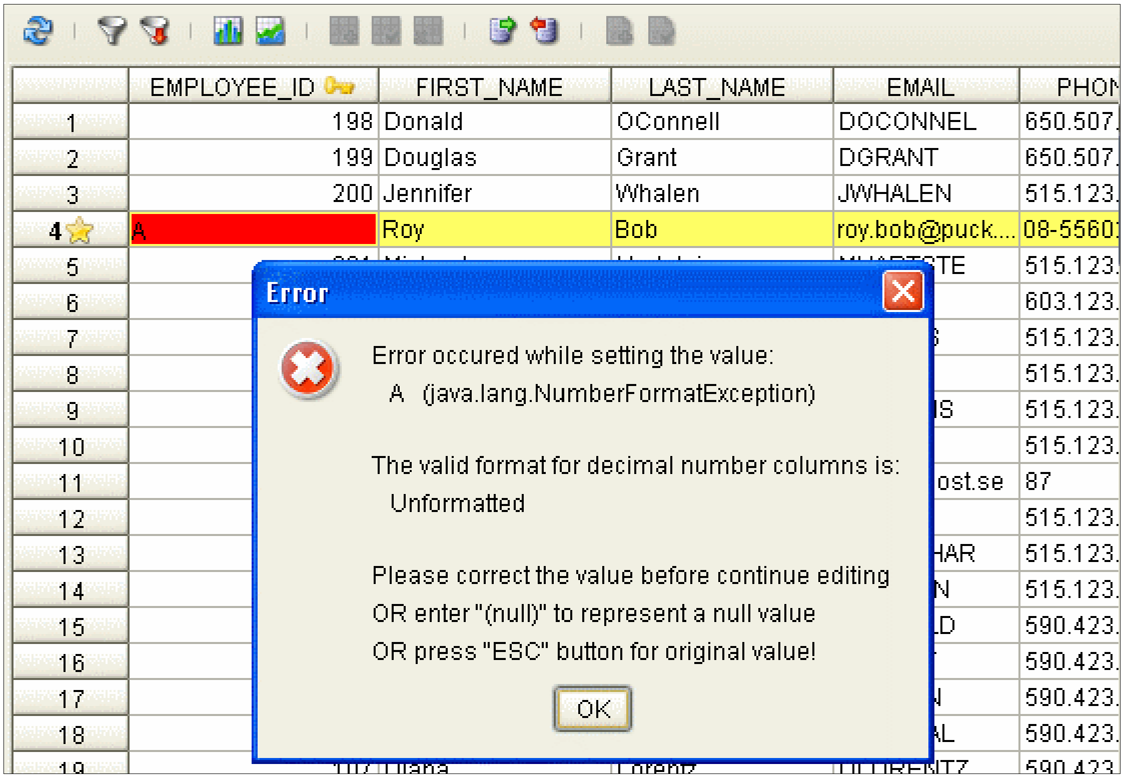
To insert a new row just press the **Insert row** tool bar button. All columns in the table that don't allow nulls are by default switched to the yellow background color. Now just start typing in the cells to set the values of the new row.



|  | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE |
|---|---|---|---|---|---|
| 1 | 198 | Donald | OConnell | DOCONNEL | 650.507.98 |
| 2 | 199 | Douglas | Grant | DGRANT | 650.507.98 |
| 3 ☆ | (null) |  | (null) | (null) |  |
| 4 | 200 | Jennifer | Whalen | JWHALEN | 515.123.44 |
| 5 | 201 | Michael | Hartstein | MHARTSTE | 515.123.55 |
| 6 | 202 | Pat | Fay | PFAY | 603.123.66 |

**Figure: Initial view of the inline editor when a new row is about to be edited**

If a value is entered that is not valid for a cell then an error dialog is displayed. The color of the invalid cell is at the same time switched to red.



|  | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHON |
|---|---|---|---|---|---|
| 1 | 198 | Donald | OConnell | DOCONNEL | 650.507. |
| 2 | 199 | Douglas | Grant | DGRANT | 650.507. |
| 3 | 200 | Jennifer | Whalen | JWHALEN | 515.123. |
| 4 ☆ | A | Roy | Bob | roy.bob@puck.... | 08-5560 |
| 5 |  |  |  | MHARTSTE | 515.123. |
| 6 |  |  |  |  | 603.123. |
| 7 |  |  |  | S | 515.123. |
| 8 |  |  |  |  | 515.123. |
| 9 |  |  |  | IS | 515.123. |
| 10 |  |  |  |  | 515.123. |
| 11 |  |  |  | ost.se | 87 |
| 12 |  |  |  |  | 515.123. |
| 13 |  |  |  | HAR | 515.123. |
| 14 |  |  |  | N | 515.123. |
| 15 |  |  |  | LD | 590.423. |
| 16 |  |  |  |  | 590.423. |
| 17 |  |  |  | N | 590.423. |
| 18 |  |  |  | AL | 590.423. |
| 19 | 107 | Diana | Lorentz | DLORENTZ | 590.423 |

Error

Error occured while setting the value:
A (java.lang.NumberFormatException)

The valid format for decimal number columns is:
Unformatted

Please correct the value before continue editing
OR enter "(null)" to represent a null value
OR press "ESC" button for original value!

**Figure: Error dialog when the data type is invalid for a column**

Follow the instructions in the dialog to either correct the value to match the data type of the column or revert to its original value.

If the newly added row is the only row in the table then there is no other row to select in order to perform the insertion. You must in this situation explicitly press the **Perform the current edit** button in the tool bar.

## Update an existing row

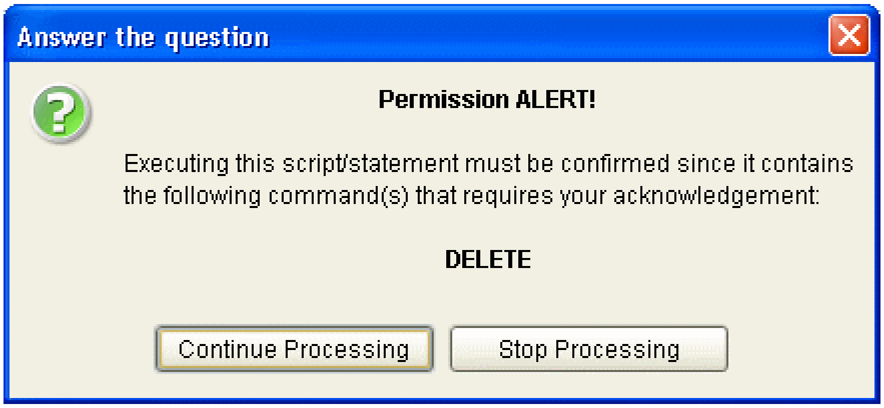
To update the value of a cell then just select the cell and start typing. The same checks as when inserting a new row are done here as well. To perform the edit just press the **Perform the current edit** button in the tool bar or select a cell in another row.

DbVisualizer use three different strategies to determine the row that will be updated. In order to let DbVisualizer use **Selected Columns** then first edit the cells that should be updated. Now select each cell that should be part of the selected columns list. Press the **Perform the current edit** button in the tool bar to let DbVisualizer use the selected columns in the final **where** clause.

## Delete a row

To delete a row then select at least one cell in the row to be removed. DbVisualizer will use one of the strategies in order to determine the row that will be deleted. Selecting one or several cells in the row will form the **where** clause that optionally will be used if the **Selected Columns** strategy is used to identify the row.

The default behavior when deleting a row is that it has to be confirmed. You may modify whether the confirmation dialog should appear or not in the **Permission** category **in Tool Properties**.



Permission ALERT!

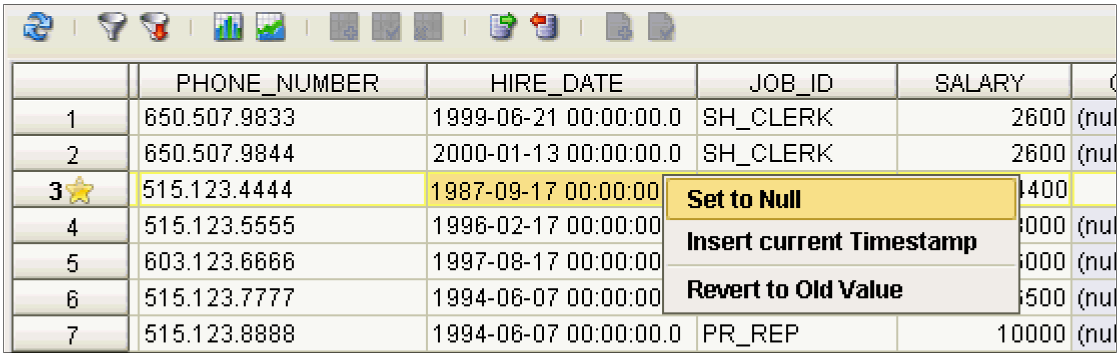
Executing this script/statement must be confirmed since it contains the following command(s) that requires your acknowledgement:

DELETE

**Figure: Confirmation dialog when deleting a row**

## Cell pop up menu

The cell pop up menu is active while editing a cell value and it is displayed using the right mouse button.



| | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | ( |
|---|---|---|---|---|---|
| 1 | 650.507.9833 | 1999-06-21 00:00:00.0 | SH_CLERK | 2600 | (nul |
| 2 | 650.507.9844 | 2000-01-13 00:00:00.0 | SH_CLERK | 2600 | (nul |
| 3 | 515.123.4444 | 1987-09-17 00:00:00 | | 400 | |
| 4 | 515.123.5555 | 1996-02-17 00:00:00 | | 000 | (nul |
| 5 | 603.123.6666 | 1997-08-17 00:00:00 | | 000 | (nul |
| 6 | 515.123.7777 | 1994-06-07 00:00:00 | | 500 | (nul |
| 7 | 515.123.8888 | 1994-06-07 00:00:00.0 | PR_REP | 10000 | (nul |

**Figure: Cell pop up menu**

The list of operations in the cell pop up menu is different depending on what data type the column is. Common for all data types are the **Set to Null** and **Uncheck Column** operations.

The Uncheck Column is used to remove the editing state of the cell which means that it will not be included in the update of the row (the yellow background color is removed).

Remember that the format of time stamp, date or time values must match the format settings in Tool Properties.

**Insert current Timestamp** (can also be **Insert current Time** and **Insert current Date**) menu choice is valid only for the appropriate data types and simply inserts the current time stamp. The format of these values matches the format settings in Tool Properties.

# Form Editor

The Form Editor enables editing of data in a form based dialog. 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 and the inline editor are based on the same set of rules and the visual appearance is similar.

The following tool bar buttons are used to start the form editor:

**Figure: Tool bar buttons to control the form editor**

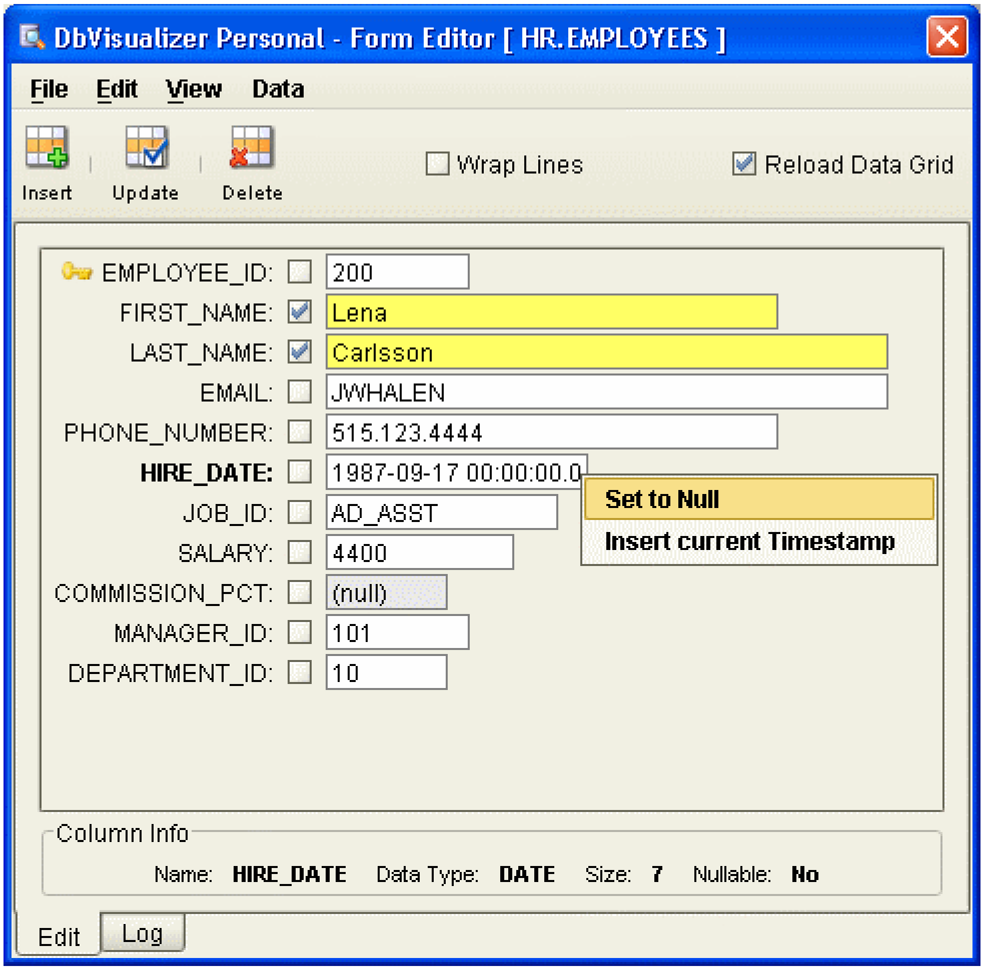
Description of the buttons (from left):

- Insert row
- Edit row (update, delete or insert copy)

An alternative way to start the form editor in edit mode is to double click on the row header.

## Form editor controls

The following explains how the form editor is organized and what controls are available.



☐ Wrap Lines ☑ Reload Data Grid

| | | |
|---|---|---|
| EMPLOYEE_ID: | ☐ | 200 |
| FIRST_NAME: | ☑ | Lena |
| LAST_NAME: | ☑ | Carlsson |
| EMAIL: | ☐ | JWHALEN |
| PHONE_NUMBER: | ☐ | 515.123.4444 |
| **HIRE_DATE:** | ☐ | 1987-09-17 00:00:00.0 |
| JOB_ID: | ☐ | AD_ASST |
| SALARY: | ☐ | 4400 |
| COMMISSION_PCT: | ☐ | (null) |
| MANAGER_ID: | ☐ | 101 |
| DEPARTMENT_ID: | ☐ | 10 |

Column Info

Name: **HIRE_DATE** Data Type: **DATE** Size: **7** Nullable: **No**

**Figure: The form editor**

The form editor is composed of two main views (at the bottom of the dialog), they are the **Edit** and **Log** views. The edit view is as its name implies the place where the actual edits take place. The log view maintains a list of log messages for all edits that have been performed. The log view is displayed automatically if an error occurs during the database operation.

The tool bar buttons are used to **Insert**,**Update** and **Delete** the current row. (Only the insert button is enabled if the form editor has been launched to insert a new row).

The **Wrap Lines** check box is used to control whether long values should be wrapped automatically.

The **Reload Data Grid** check box determines whether the Data tab grid should be automatically re-loaded once the form editor is closed. If this box is disabled then make sure to reload the data tab grid manually to reflect the changes that have been made in the form editor.

## Row Values

The **Row Values** section in the form editor lists all columns and values (fields) as they appeared in the data tab grid when the form editor was launched (i.e if a column was manually removed from the grid then it wont show in the form editor). Each field is composed of a label which is the column name and the value. Any primary key columns are indicated with a key-like icon to the left of the label.

The check box between the label and the value indicates whether the column will be part of the SQL that is finally executed to perform the edit. The checked state is automatically enabled if the field of the value is being edited and the color of the value field is automatically switched to yellow to indicate what fields will be part of the database request.

Additional information about a column is listed below the list of fields. This information is updated when the pointer is in a value field.

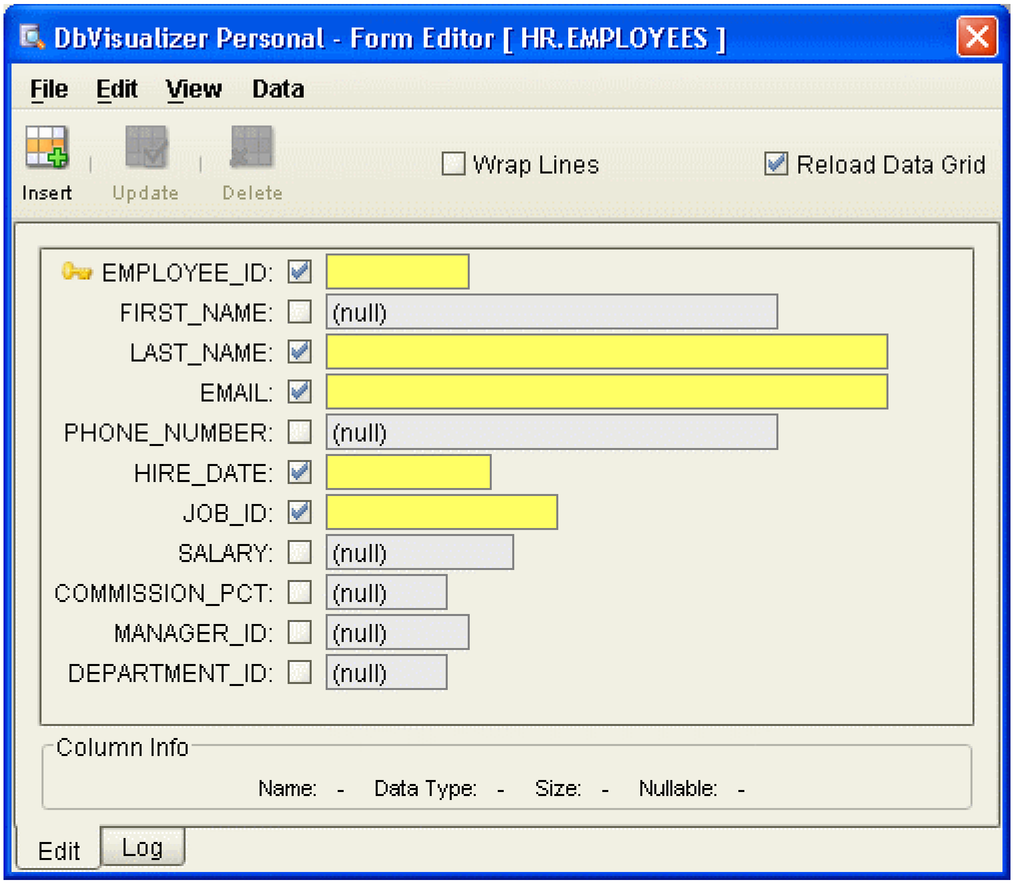
All text fields allow data to be entered in multiple lines (press the enter keyboard button). Boolean and Bit fields are managed by toggle buttons. The state of these fields can be either true, false or null. All fields except boolean fields have a pop up menu associated with them similar to the one in the inline editor. For all fields it contains the **Set to Null** operation. For time stamp, date and time columns it also contains **Insert current Timestamp**, **Insert current Time** or **Insert current Date**. The formats of time stamp, date and time are the same as specified in Tool Properties.

Remember to use the same format for time stamp, date and time fields as specified in the formats section in Tool Properties.

Each of the fields limits the amount of data that can be entered. Entering more characters than allowed will be denied. Some JDBC drivers report invalid column lengths such as 0 or a negative size. The form editor tries to figure out if the length is invalid and adjust the width accordingly.

## Insert a row

The following dialog is displayed when choosing to insert a new row using the Form Editor.



☐ Wrap Lines ☑ Reload Data Grid

EMPLOYEE_ID: ☑
FIRST_NAME: ☐ (null)
LAST_NAME: ☑
EMAIL: ☑
PHONE_NUMBER: ☐ (null)
HIRE_DATE: ☑
JOB_ID: ☑
SALARY: ☐ (null)
COMMISSION_PCT: ☐ (null)
MANAGER_ID: ☐ (null)
DEPARTMENT_ID: ☐ (null)

Name: - Data Type: - Size: - Nullable: -

**Figure: The form editor as it appears when a new row is about to be edited**

The form editor automatically enables the checked state for all columns that do not accept nulls. All other fields are unchecked with the value of (null) (or whatever the text representation of null has been set to in Tool Properties). Now specify the values accordingly and press the **Insert** button to perform the insertion. The form editor is closed if the execution was successful.

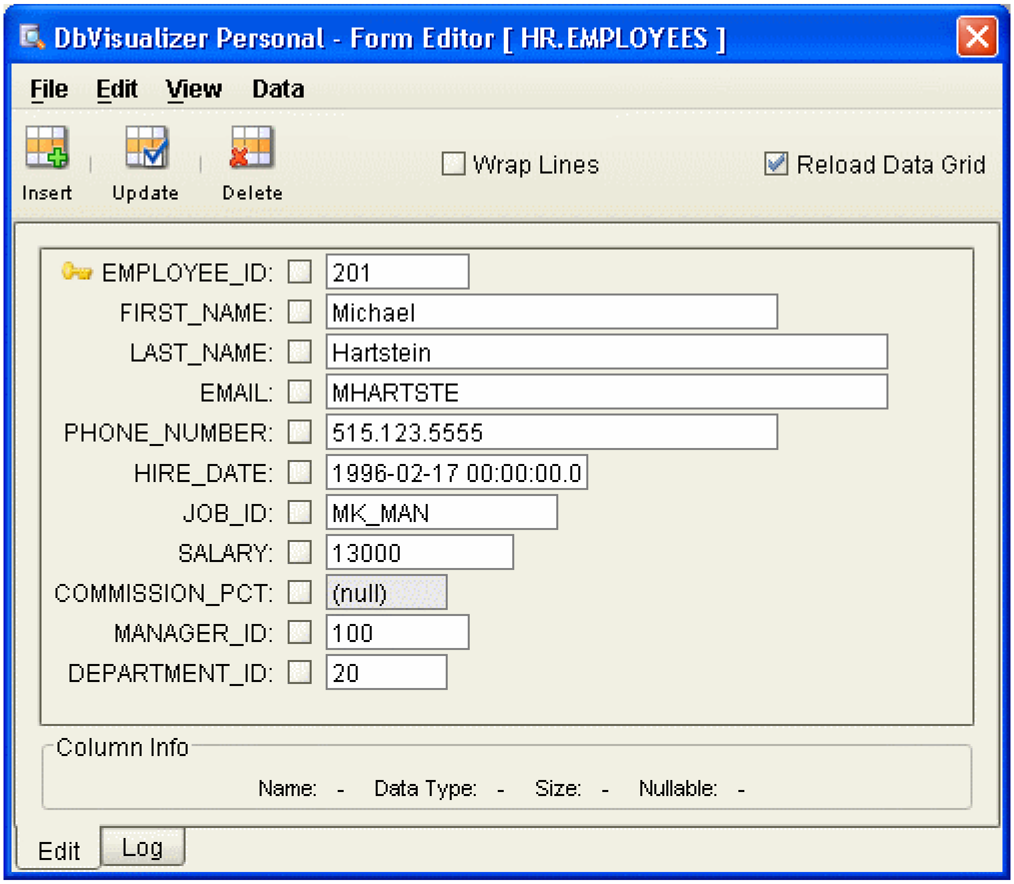
The **Data- >Insert (Keep Window)** is an alternate insert operation that is used to insert the new row. The difference is that the form editor dialog is kept. This is useful if successive inserts need to be made.

## Edit a row (update, delete or insert copy)

Do the following in the Data tab grid to edit a row using the form editor:

1. Double click on the row header for the row that is to be edited
2. Select one or more cells in the row and press the **Edit row in a form** tool bar button

The first choice is useful if the table has primary keys or if the database table accepts an update request based on all current values for the row. The second choice is useful when the table has primary keys or when one must be able to select the columns (cells) that will form the **Selected Columns** where clause. See Edits might be denied for more information.



☐ Wrap Lines ☑ Reload Data Grid

|  |  |  |
|---|---|---|
| EMPLOYEE_ID: | ☐ | 201 |
| FIRST_NAME: | ☐ | Michael |
| LAST_NAME: | ☐ | Hartstein |
| EMAIL: | ☐ | MHARTSTE |
| PHONE_NUMBER: | ☐ | 515.123.5555 |
| HIRE_DATE: | ☐ | 1996-02-17 00:00:00.0 |
| JOB_ID: | ☐ | MK_MAN |
| SALARY: | ☐ | 13000 |
| COMMISSION_PCT: | ☐ | (null) |
| MANAGER_ID: | ☐ | 100 |
| DEPARTMENT_ID: | ☐ | 20 |

Column Info

Name: - Data Type: - Size: - Nullable: -

**Figure: The form editor as it appears when edit of an existing row has been requested**

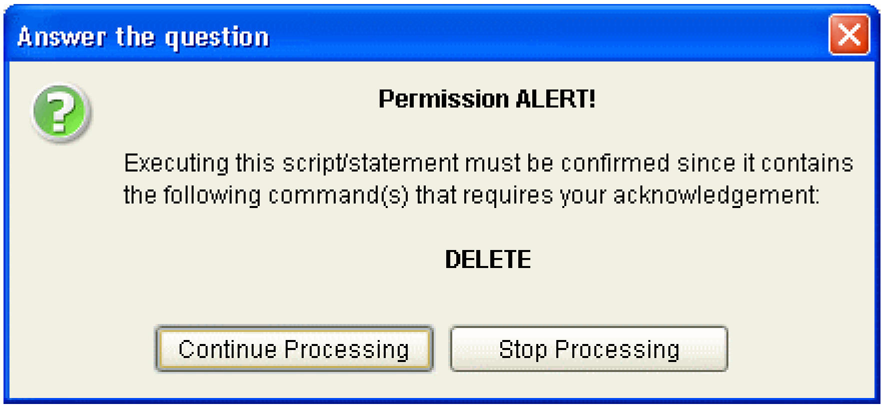
### Update the row

All fields are automatically unchecked when the form editor appears in edit mode. Change the desired values and press the **Update** button to perform the update request.

The form editor is closed if the update operation is successful.

### Delete the row

To delete the current row in the form editor just press the **Delete** tool bar button. A confirmation dialog might be displayed (the appearance of this dialog can be specified in Tool Properties) in which the deletion must be confirmed.



Answer the question

Permission ALERT!

Executing this script/statement must be confirmed since it contains the following command(s) that requires your acknowledgement:

DELETE

**Figure: Confirmation dialog when deleting a row**

The form editor will be closed if the delete operation is successful.

### Insert a copy of a row

The form editor allows a new row to be inserted based on the values that are currently in the editor. Make sure to set each fields check state or use the **Edit->Check All Values** menu choice to enable the checked state for all fields. Use the **Insert** tool bar button or **Data->Insert (Keep Window)** menu operation to perform the insertion.

## Import from File

The **File->Import From File** operation can be used to load any file into the currently selected field. Importing from file is exactly the same as manually entering the data i.e the max field sizes are considered and it might not be possible to load a file for which content does not fit into the actual field.

Imported binary data with a recognized binary viewer will be displayed accordingly.

## Export from File

The **File->Export To File** is used to export the content of any field including binary data to a file.

## Editing Binary data/BLOB and CLOB

There are a few constraints specifically for editing of BLOB and CLOB data types in DbVisualizer:

- They can only be edited in the form editor
- A primary key is recommended to successfully update these data types
- **Note:** A primary key is **required** for update of BLOB and CLOB in **Oracle**

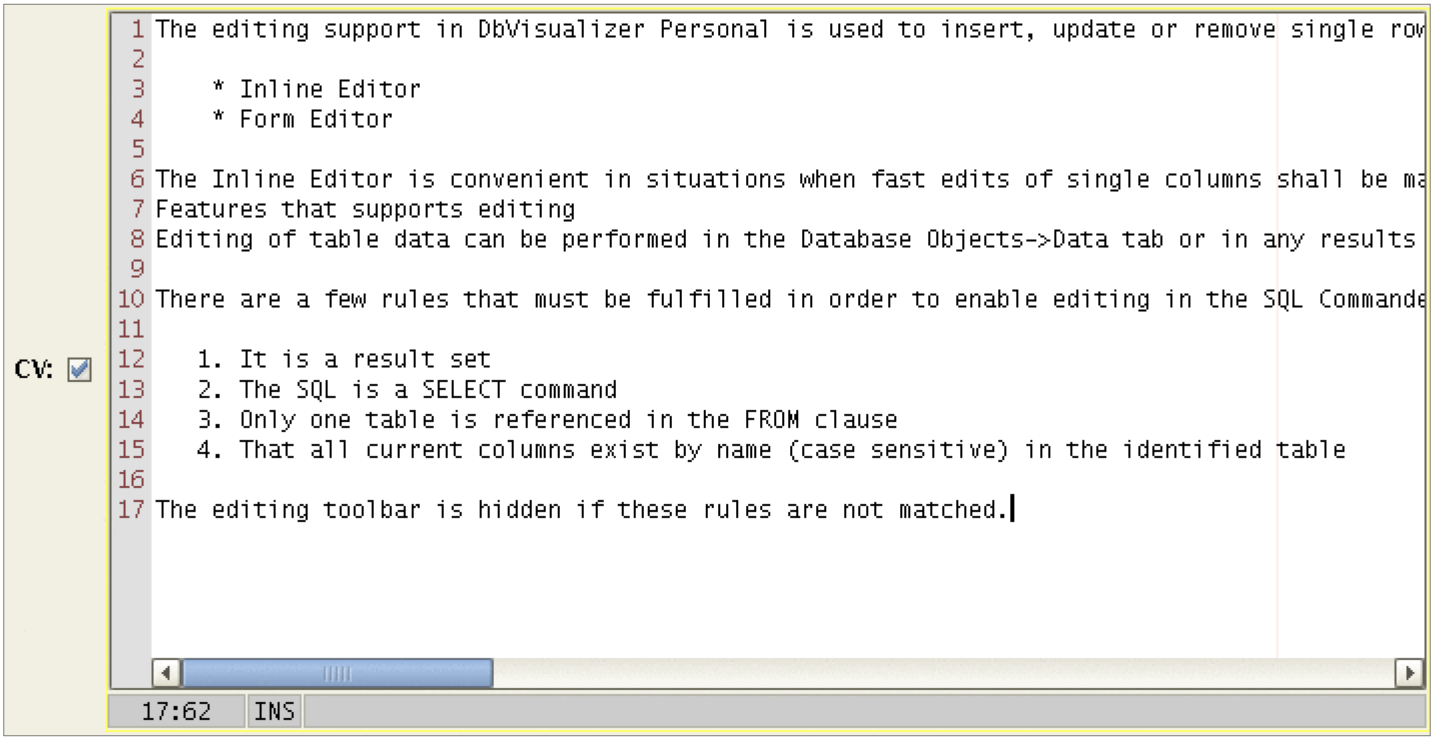
**Binary data** in DbVisualizer is the generic term for several common binary database types:

- LONGVARBINARY
- BINARY
- VARBINARY
- BLOB

Read the following sections about CLOB and Binary/BLOB data for specific information

### CLOB

CLOB data appears (apart from other data types) in a multi line text field in the form editor. Data can be entered manually or imported using the **File->Import From File** operation.



CV: ☑

```
The editing support in DbVisualizer Personal is used to insert, update or remove single row

    * Inline Editor
    * Form Editor

The Inline Editor is convenient in situations when fast edits of single columns shall be ma
Features that supports editing
Editing of table data can be performed in the Database Objects->Data tab or in any results

There are a few rules that must be fulfilled in order to enable editing in the SQL Commande

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

The editing toolbar is hidden if these rules are not matched.
```

**Figure: The CLOB text editor**

### Binary data/BLOB

Binary data can by its nature not be manually edited in DbVisualizer, it can only be imported from a file.

The form editor recognizes some common formats and presents them in an appropriate viewer.

### GIF, JPEG and PNG viewer

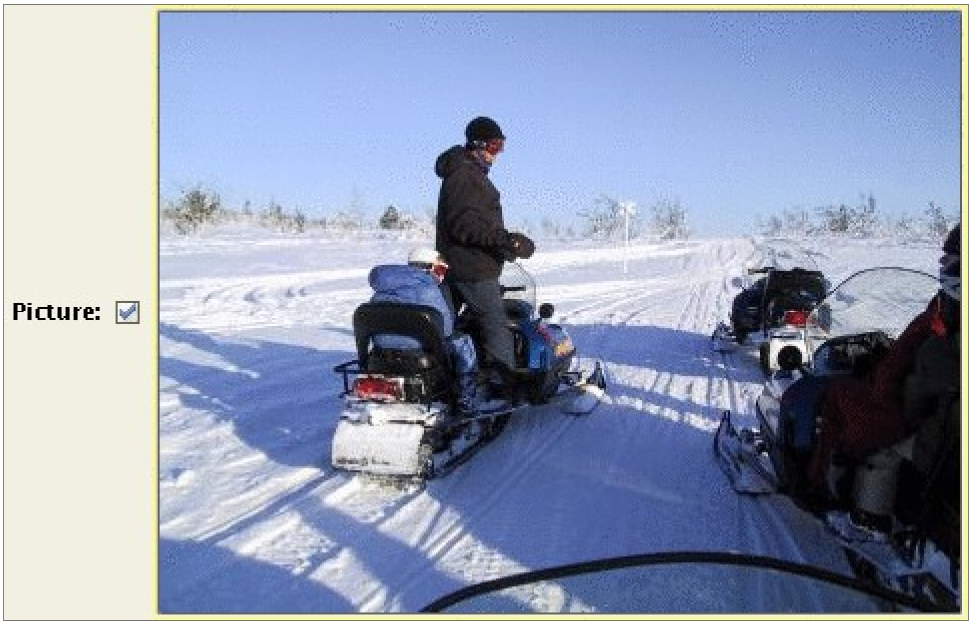
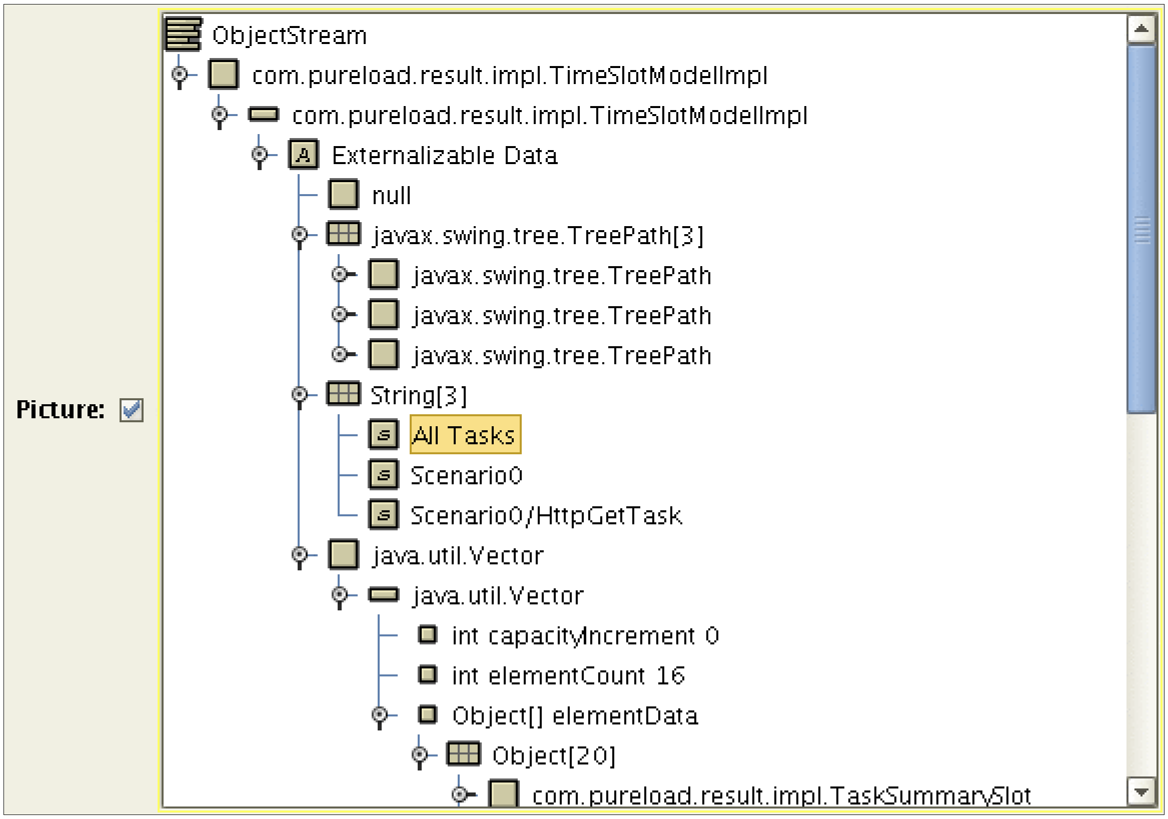


**Figure: Binary data viewer for common image formats**

### Serialized Java objects viewer



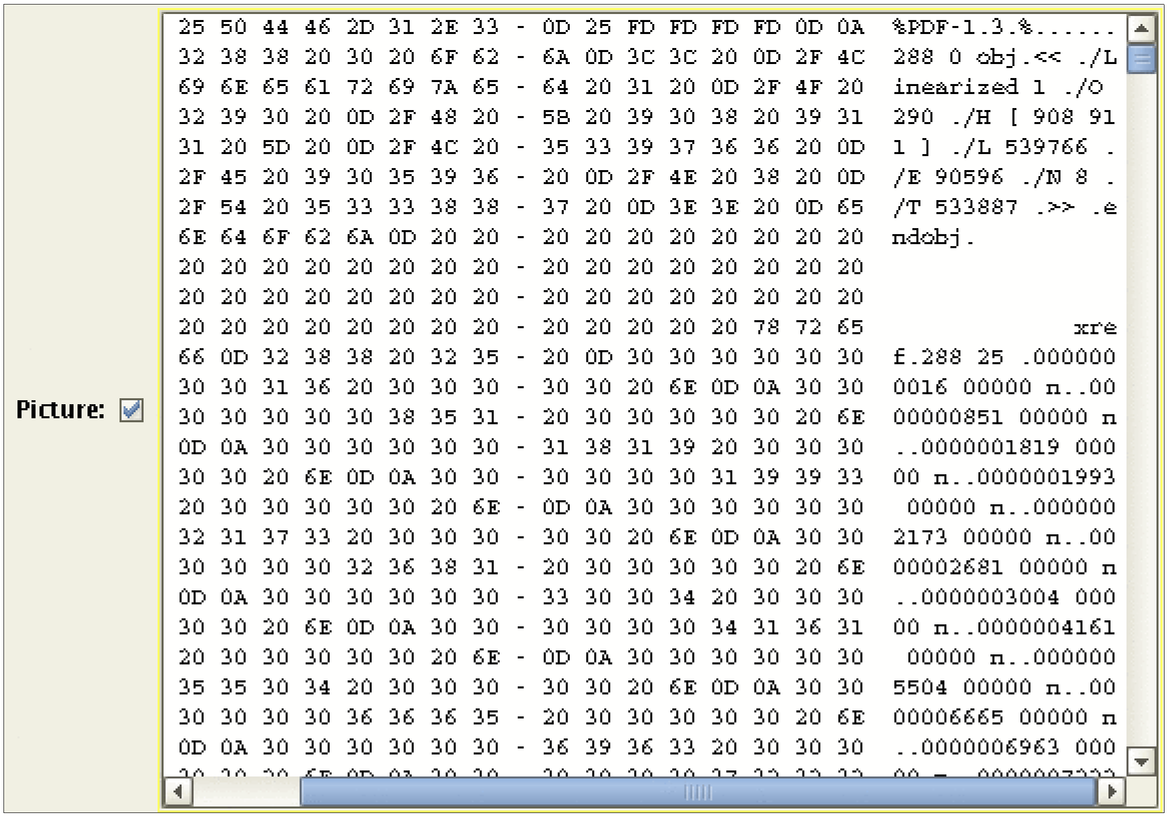
Picture: ☑

```
ObjectStream
  com.pureload.result.impl.TimeSlotModelImpl
    com.pureload.result.impl.TimeSlotModelImpl
      Externalizable Data
        null
        javax.swing.tree.TreePath[3]
          javax.swing.tree.TreePath
          javax.swing.tree.TreePath
          javax.swing.tree.TreePath
        String[3]
          All Tasks
          Scenario0
          Scenario0/HttpGetTask
        java.util.Vector
          java.util.Vector
            int capacityIncrement 0
            int elementCount 16
            Object[] elementData
              Object[20]
                com.pureload.result.impl.TaskSummarySlot
```

**Figure: Binary data viewer for serialized Java objects**

### Hex/Ascii viewer

The generic hex/ascii viewer is used if the data format is not recognized.



Picture: ☑

```
25 50 44 46 2D 31 2E 33 - 0D 25 FD FD FD FD 0D 0A  %PDF-1.3.%......
32 38 38 20 30 20 6F 62 - 6A 0D 3C 3C 20 0D 2F 4C  288 0 obj.<< ./L
69 6E 65 61 72 69 7A 65 - 64 20 31 20 0D 2F 4F 20  inearized 1 ./O 
32 39 30 20 0D 2F 48 20 - 5B 20 39 30 38 20 39 31  290 ./H [ 908 91
31 20 5D 20 0D 2F 4C 20 - 35 33 39 37 36 36 20 0D  1 ] ./L 539766 .
2F 45 20 39 30 35 39 36 - 20 0D 2F 4E 20 38 20 0D  /E 90596 ./N 8 .
2F 54 20 35 33 33 38 38 - 37 20 0D 3E 3E 20 0D 65  /T 533887 .>> .e
6E 64 6F 62 6A 0D 20 20 - 20 20 20 20 20 20 20 20  ndobj.
20 20 20 20 20 20 20 20 - 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 - 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 - 20 20 20 20 20 78 72 65               xre
66 0D 32 38 38 20 32 35 - 20 0D 30 30 30 30 30 30  f.288 25 .000000
30 30 31 36 20 30 30 30 - 30 30 20 6E 0D 0A 30 30  0016 00000 n..00
30 30 30 30 30 38 35 31 - 20 30 30 30 30 30 20 6E  00000851 00000 n
0D 0A 30 30 30 30 30 30 - 31 38 31 39 20 30 30 30  ..0000001819 000
30 30 20 6E 0D 0A 30 30 - 30 30 30 30 31 39 39 33  00 n..0000001993
20 30 30 30 30 30 20 6E - 0D 0A 30 30 30 30 30 30   00000 n..000000
32 31 37 33 20 30 30 30 - 30 30 20 6E 0D 0A 30 30  2173 00000 n..00
30 30 30 30 32 36 38 31 - 20 30 30 30 30 30 20 6E  00002681 00000 n
0D 0A 30 30 30 30 30 30 - 33 30 30 34 20 30 30 30  ..0000003004 000
30 30 20 6E 0D 0A 30 30 - 30 30 30 30 34 31 36 31  00 n..0000004161
20 30 30 30 30 30 20 6E - 0D 0A 30 30 30 30 30 30   00000 n..000000
35 35 30 34 20 30 30 30 - 30 30 20 6E 0D 0A 30 30  5504 00000 n..00
30 30 30 30 36 36 36 35 - 20 30 30 30 30 30 20 6E  00006665 00000 n
0D 0A 30 30 30 30 30 30 - 36 39 36 33 20 30 30 30  ..0000006963 000
```

**Figure: Hex/Ascii viewer for unrecognized binary data**

# Create Table and Index Assistants

## Introduction

The Create Table and Index Assistants are used to create new tables and indexes. The assistants are quite simple to use since they examine various meta data in the database (depending on what assistant is used) and then let the user point and click to define the actual table or index. Both assistants finally generate the appropriate SQL and pass it over to the SQL Commander which is used to execute it.

These utilities are launched from the **Database** main menu or in the **Database Objects** tree right click menu. The menu choices are enabled only if a table or index can be created for the selected node in the Database Objects tree.

**Note:** Remember to manually select the **Reload** right click menu choice in the Database Objects tree once a new table has been created.

## Create Table

Start the table creation assistant by choosing the **Create Table** action in the objects tree right click menu or from the Actions menu. Make sure you have located and selected the appropriate object in the **Database Objects** tree as this selection is used to define which database, schema, etc. the table will be created in.

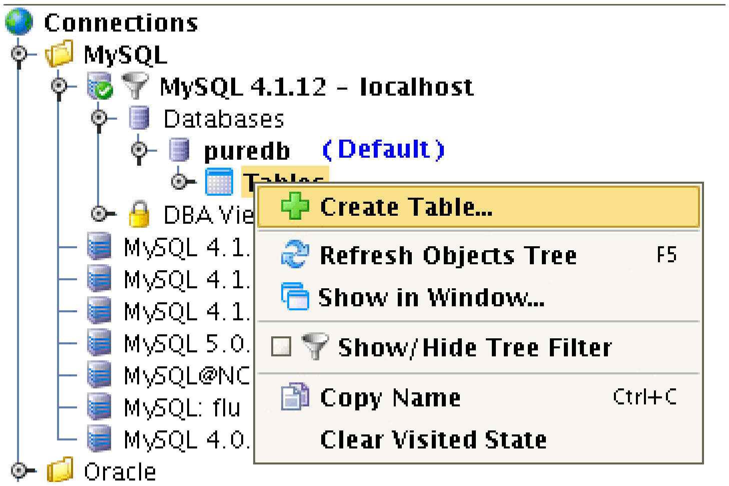


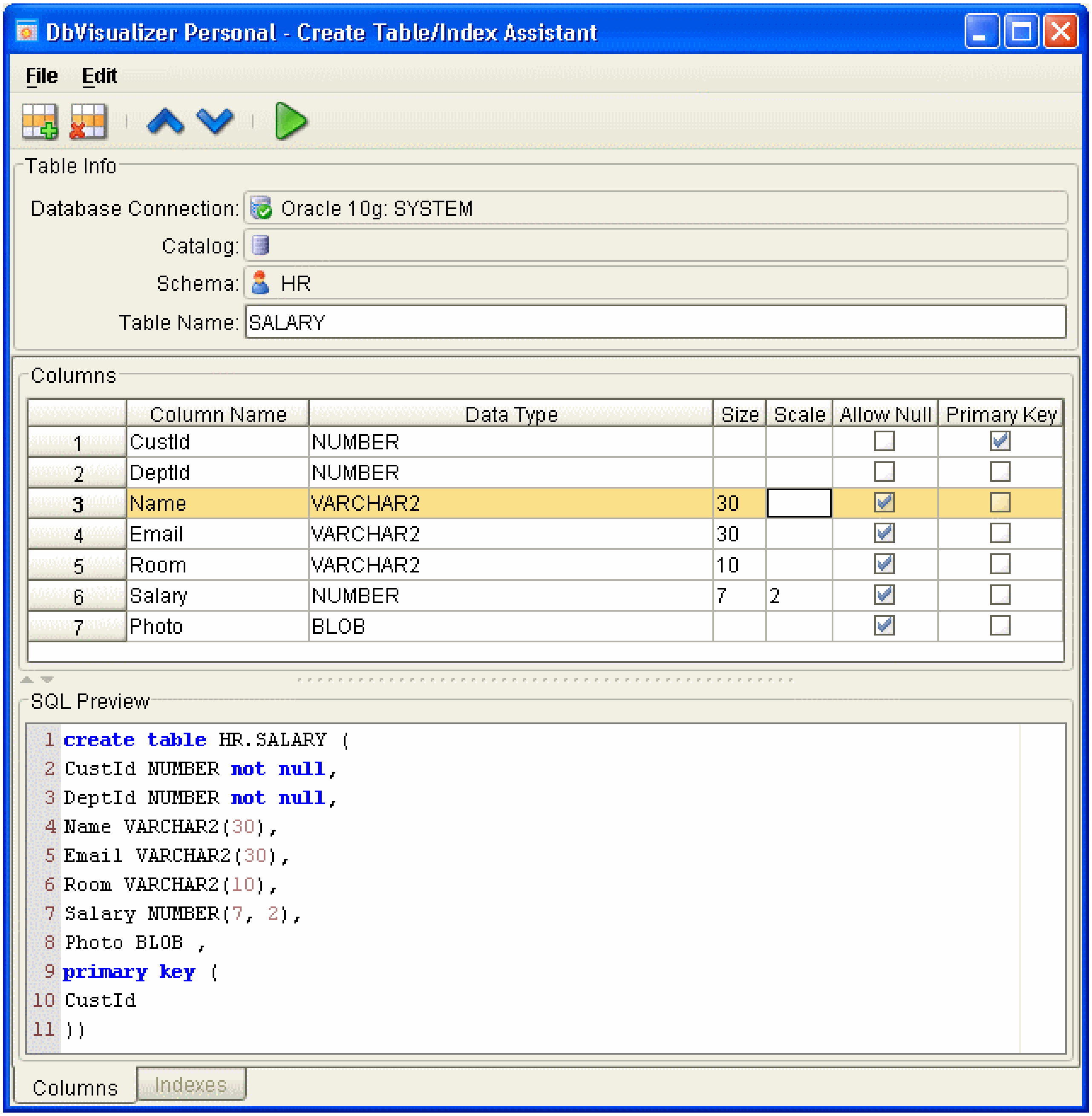
**Figure: The right click menu in the Database Objects tree**

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

- **Table Info**
  Specifies the owning database connection, database and/or schema. These are picked up from the selection in the tree when the assistant is started. Table name is empty and must be specified.
- **Columns**

The list is used to organize the columns that will be in the final create statement. The Data Type column is a list of supported data types for the actual database. Columns can be re-ordered using the main controls.

- **SQL Preview**
  The SQL previewer instantly shows the SQL that is used to create the table.



Database Connection: Oracle 10g: SYSTEM
Catalog:
Schema: HR
Table Name: SALARY

| | Column Name | Data Type | Size | Scale | Allow Null | Primary Key |
|---|---|---|---|---|---|---|
| 1 | CustId | NUMBER | | | □ | ☑ |
| 2 | DeptId | NUMBER | | | □ | □ |
| 3 | Name | VARCHAR2 | 30 | | ☑ | □ |
| 4 | Email | VARCHAR2 | 30 | | ☑ | □ |
| 5 | Room | VARCHAR2 | 10 | | ☑ | □ |
| 6 | Salary | NUMBER | 7 | 2 | ☑ | □ |
| 7 | Photo | BLOB | | | ☑ | □ |

```
create table HR.SALARY (
CustId NUMBER not null,
DeptId NUMBER not null,
Name VARCHAR2(30),
Email VARCHAR2(30),
Room VARCHAR2(10),
Salary NUMBER(7, 2),
Photo BLOB ,
primary key (
CustId
))
```

**Figure: The table assistant**

## Columns

Add columns by using the **Edit->Insert** menu choice and **Edit->Delete** to remove the currently selected row. They can be re-organized using the **Edit->Move Up** and **Edit->Move Down** menu operations.

The assistant is based on generic JDBC and it is the responsibility of the user to enter the required fields i.e. specifying size for text data types, ignore size for some BLOB types, enter scale for decimal types, etc.

The **Data Type** field for a column when selected contains a list of valid data types for the actual database connection.

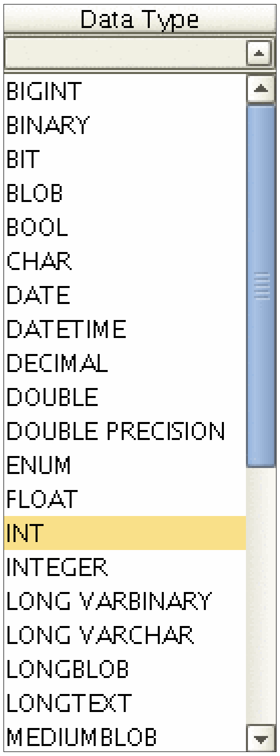
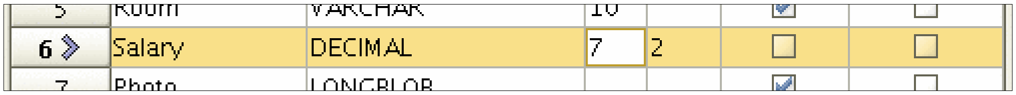


**Figure: Data Type list (for MySQL)**

The **Size** and **Scale** fields are used either in conjunction or else the size field on its own. Size is often used to set the max length of text columns. It is also used in combination with the Scale field when defining decimal boundaries.



| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | Room | VARCHAR | 10 | | ☑ | □ |
| 6 | Salary | DECIMAL | 7 | 2 | □ | □ |
| 7 | Photo | LONGBLOB | | | ☑ | □ |

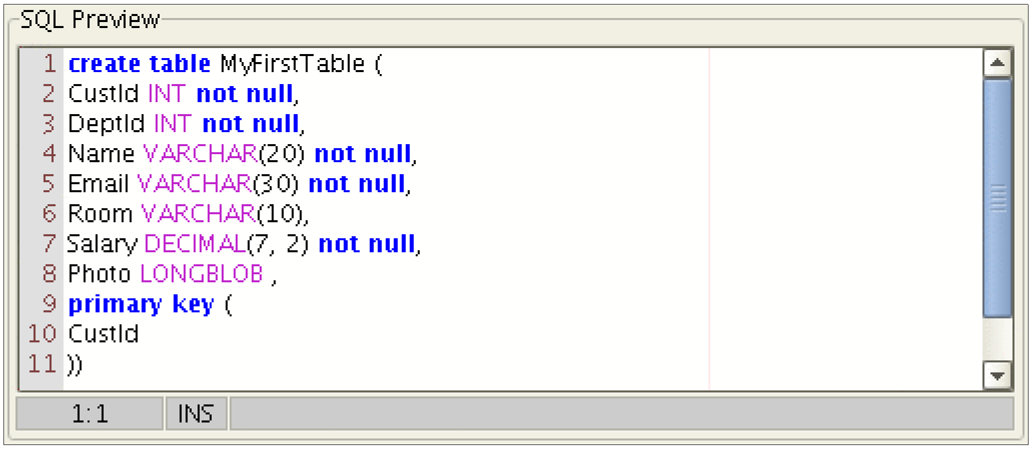
**Figure: Size and scale for a DECIMAL data type**

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

```
      1.02
   9871.1
   8172.0
  18291.22
          12.112 <- Error
1921211.11   <- Error
```

## SQL Preview

The SQL Preview area is updated automatically to match the edits made in the assistant. The preview is read only.



```
create table MyFirstTable (
CustId INT not null,
DeptId INT not null,
Name VARCHAR(20) not null,
Email VARCHAR(30) not null,
Room VARCHAR(10),
Salary DECIMAL(7, 2) not null,
Photo LONGBLOB ,
primary key (
CustId
))
```

**Figure: The SQL Preview for a table**

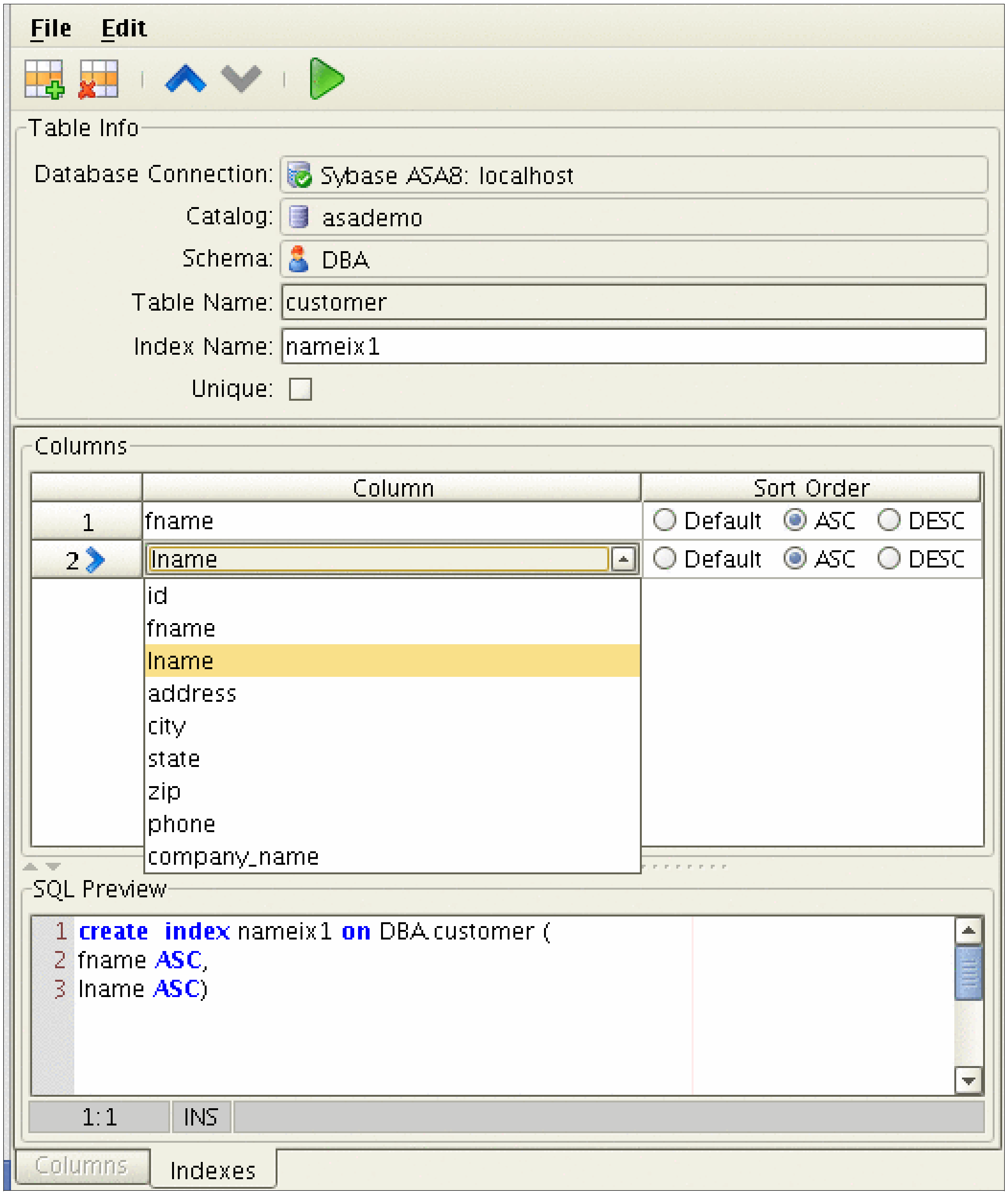
## Execute

When you are satisfied with the table then choose **File- >Pass SQL to SQL Commander** menu choice. The SQL is then inserted into the SQL Commander. Now select **Database->Execute** main menu choice to execute it as per any regular SQL. If any errors appear during the execution then go back to the Table Assistant and make the necessary changes.

**Note:** If you want to refine the setup of a table then just select the already visible Table Assistant window. Do not choose the Create Table menu choice again since it will then clear the assistant from its current setup.

# Create Index

The Create Index Assistant is much the same as the Create Table Assistant. It is launched from the same menus and the overall layout and controls are the same. The only difference is the **Columns** list which now lists the columns that will be part of the index. The **Column** field when selected lists the current columns in the table.



Database Connection: Sybase ASA8: localhost

Catalog: asademo

Schema: DBA

Table Name: customer

Index Name: nameix1

Unique: □

| | Column | Sort Order |
|---|---|---|
| 1 | fname | ○ Default ● ASC ○ DESC |
| 2 | lname | ○ Default ● ASC ○ DESC |

```
create index nameix1 on DBA.customer (
fname ASC,
lname ASC)
```

**Figure: The Index creation assistant**

The example shows that a new **unique** index, **UniqueName** will be created for the **MyFirstTable**. It will index the **Name** column in **Ascending** order.

Please read the previous sections on how to use the Table assistant to edit and create the actual index.

# Procedure Editor

## Introduction

Many databases offers the capability to store programs in the database. These programs may be of two types, functions and procedures. The difference is that functions return a value while procedures don't. Some databases do in addition offer the package concept which means that a collection of functions and/or procedure are grouped together in one unit. A package is the interface describing the functions and procedures while the package body contains the implementation. The related functionality to create and drop these procedural object types in DbVisualizer are activated via the object actions menu. The procedure editor is used to browse, edit and compile these object types.

The examples throughout this document refer to the procedure object type while all described features can also be applied to function, package and package body objects.

## Create Procedure

To create a new procedure simply select the **Procedures** node in the objects tree and choose **Create Procedure** from its action menu.

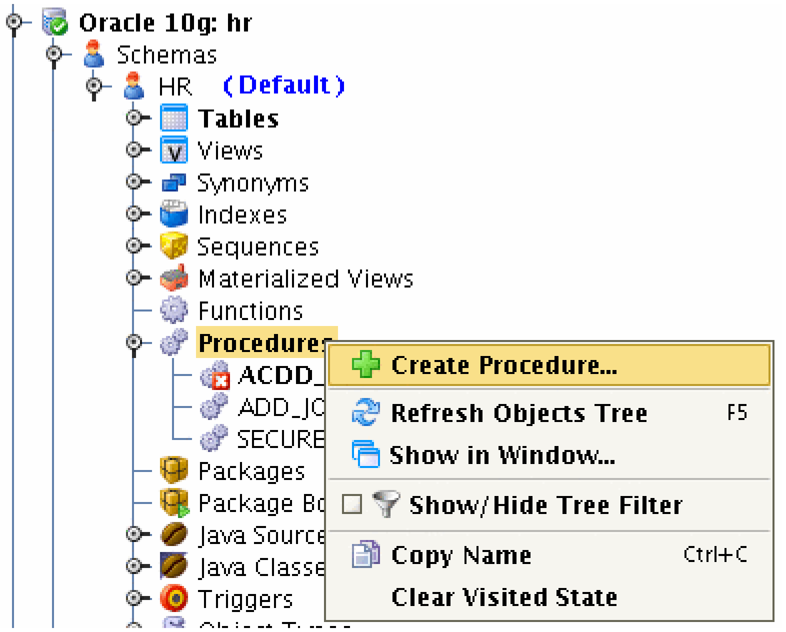
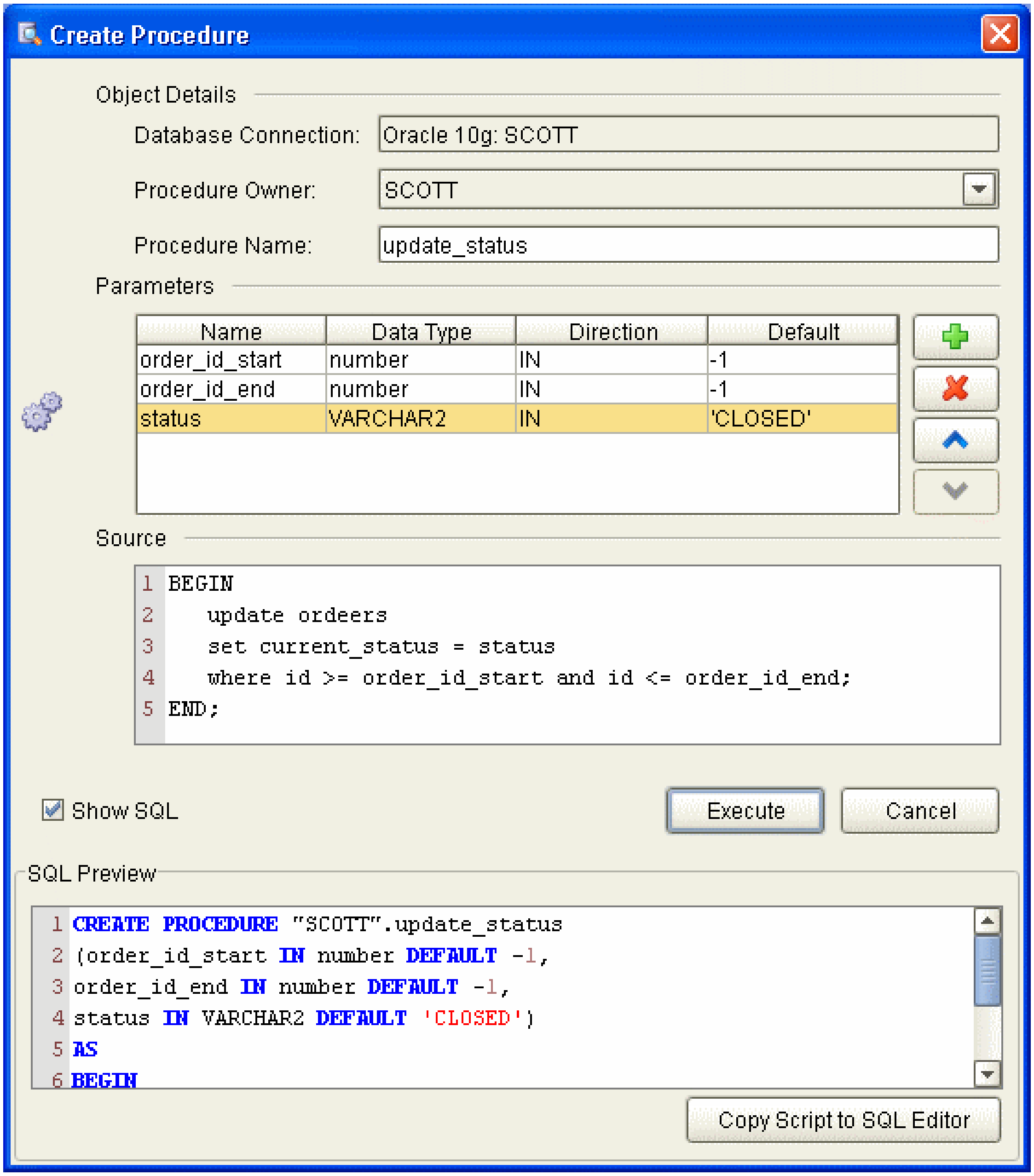


**Figure: The actions menu for the Procedures node**

Next a dialog will be displayed in which parameters for the new procedure are entered. This data forms the interface for the procedure. You can leave the parameters and edit them later. The source for the procedure is edited in a later step.



Database Connection: Oracle 10g: SCOTT

Procedure Owner: SCOTT

Procedure Name: update_status

| Name | Data Type | Direction | Default |
|---|---|---|---|
| order_id_start | number | IN | -1 |
| order_id_end | number | IN | -1 |
| status | VARCHAR2 | IN | 'CLOSED' |

```
BEGIN
   update ordeers
   set current_status = status
   where id >= order_id_start and id <= order_id_end;
END;
```

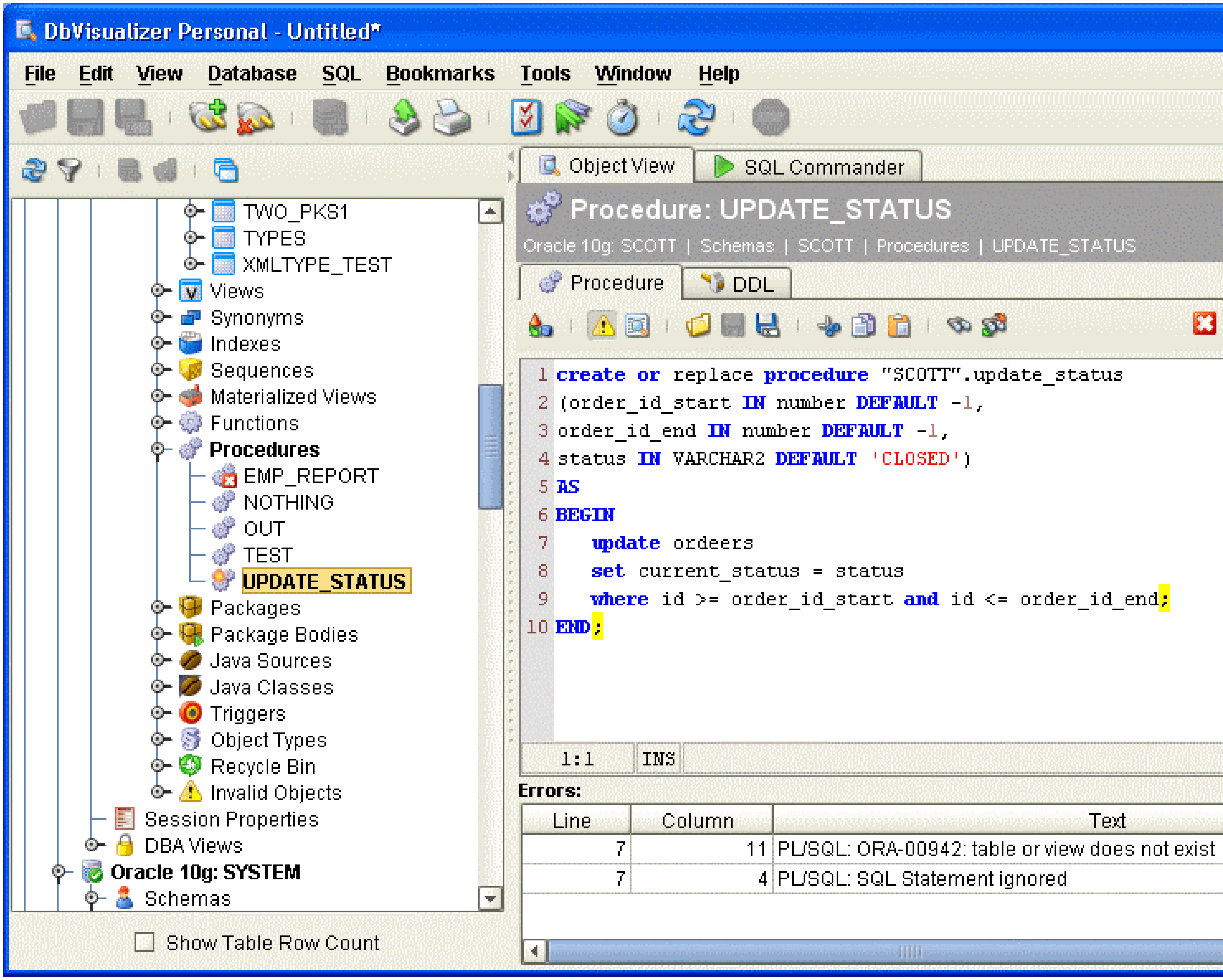
☑ Show SQL

```
CREATE PROCEDURE "SCOTT".update_status
(order_id_start IN number DEFAULT -1,
order_id_end IN number DEFAULT -1,
status IN VARCHAR2 DEFAULT 'CLOSED')
AS
BEGIN
```

**Figure: The create procedure dialog**

Use the buttons to the right of the parameter list to insert, remove and move the entries. For every parameter you must supply its **Name**, leaving **Data Type** results in the VARCHAR datatype and the **Direction** is by default set to IN.

Now press **Ok** in the dialog to create the new procedure.



```
create or replace procedure "SCOTT".update_status
(order_id_start IN number DEFAULT -1,
order_id_end IN number DEFAULT -1,
status IN VARCHAR2 DEFAULT 'CLOSED')
AS
BEGIN
   update ordeers
   set current_status = status
   where id >= order_id_start and id <= order_id_end;
END;
```

| Line | Column | Text |
|---|---|---|
| 7 | 11 | PL/SQL: ORA-00942: table or view does not exist |
| 7 | 4 | PL/SQL: SQL Statement ignored |

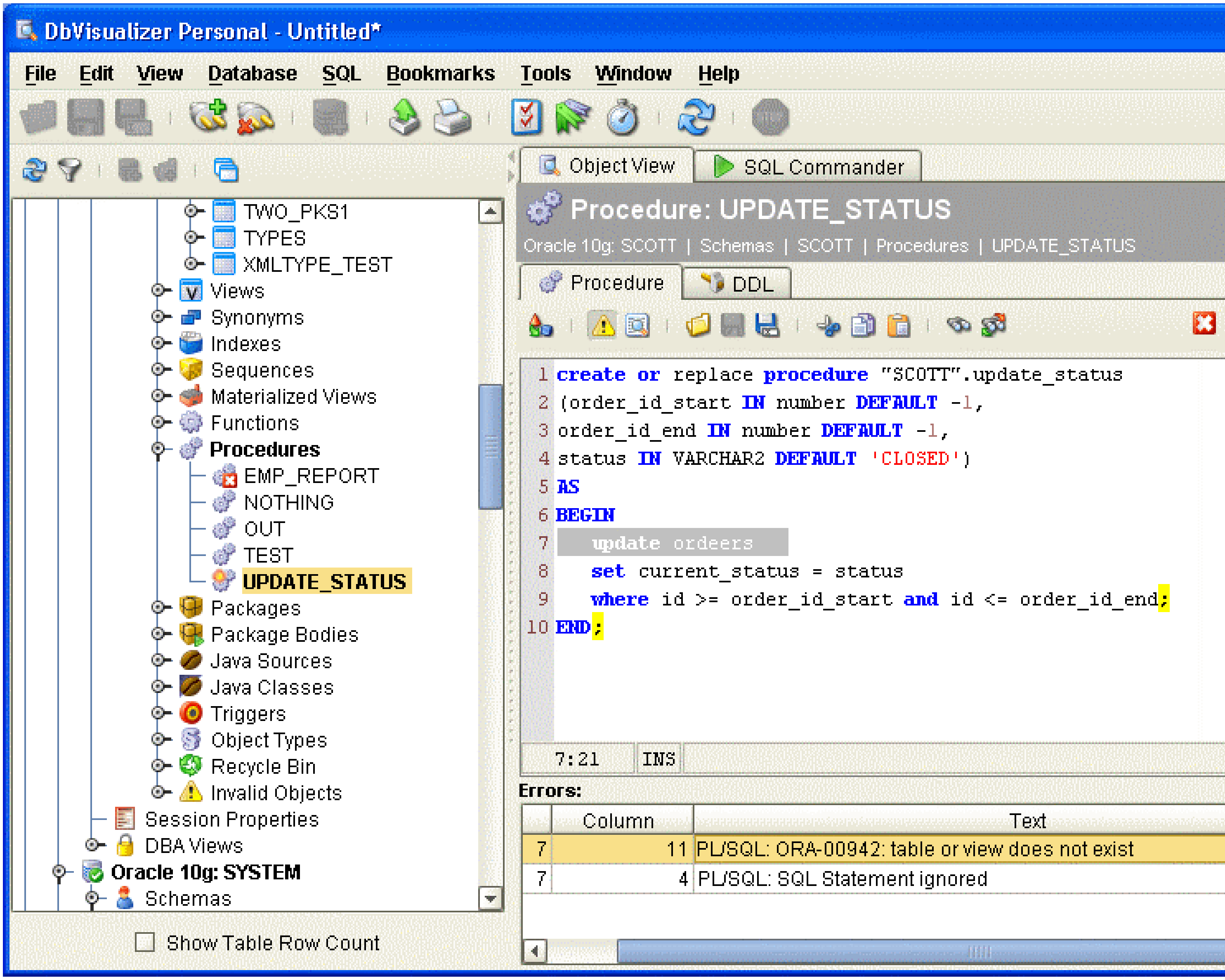
**Figure: The newly created procedure**

Selecting the newly created procedure in the tree will show the source for it in the procedure editor.

# Edit and Compile

**Note:** This editor is currently supported only for Oracle databases!
The editor have a small toolbar with various actions to start compilation, save and load to/from file, show parameter information and common editing operations. The **Status** indicator shows whether the procedure is valid or invalid based on last compilation.

To start editing then just edit the content. Compile the procedure using the compile toolbar button.



```
 1 create or replace procedure "SCOTT".update_status
 2 (order_id_start IN number DEFAULT -1,
 3 order_id_end IN number DEFAULT -1,
 4 status IN VARCHAR2 DEFAULT 'CLOSED')
 5 AS
 6 BEGIN
 7    update ordeers
 8    set current_status = status
 9    where id >= order_id_start and id <= order_id_end;
10 END;
```

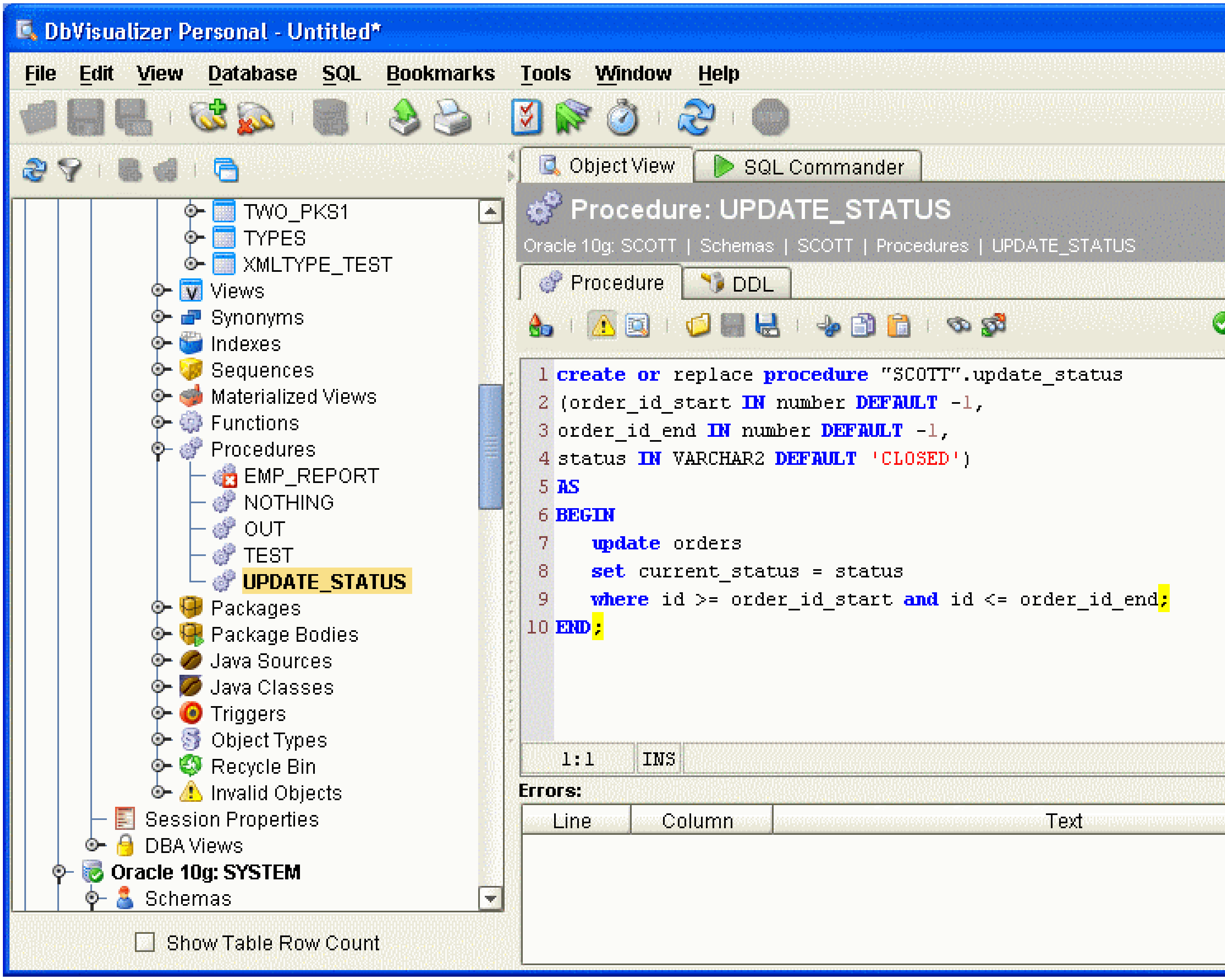
Errors:

| | Column | Text |
|---|---|---|
| 7 | 11 | PL/SQL: ORA-00942: table or view does not exist |
| 7 | 4 | PL/SQL: SQL Statement ignored |

**Figure: Compiling procedure with errors**

If error(s) occur during compilation then the error list will appear below the editor. It lists the row number in the source editor where the error is and an error message. Click the error in the list and it will highlight the corresponding row in the editor. The **Status** indicator is switched to **INVALID** if errors are in the procedure. The same applies for the object icon in the tree which shows a little red cross for invalid procedures.

Correcting the error yields correct results as in the next figure.



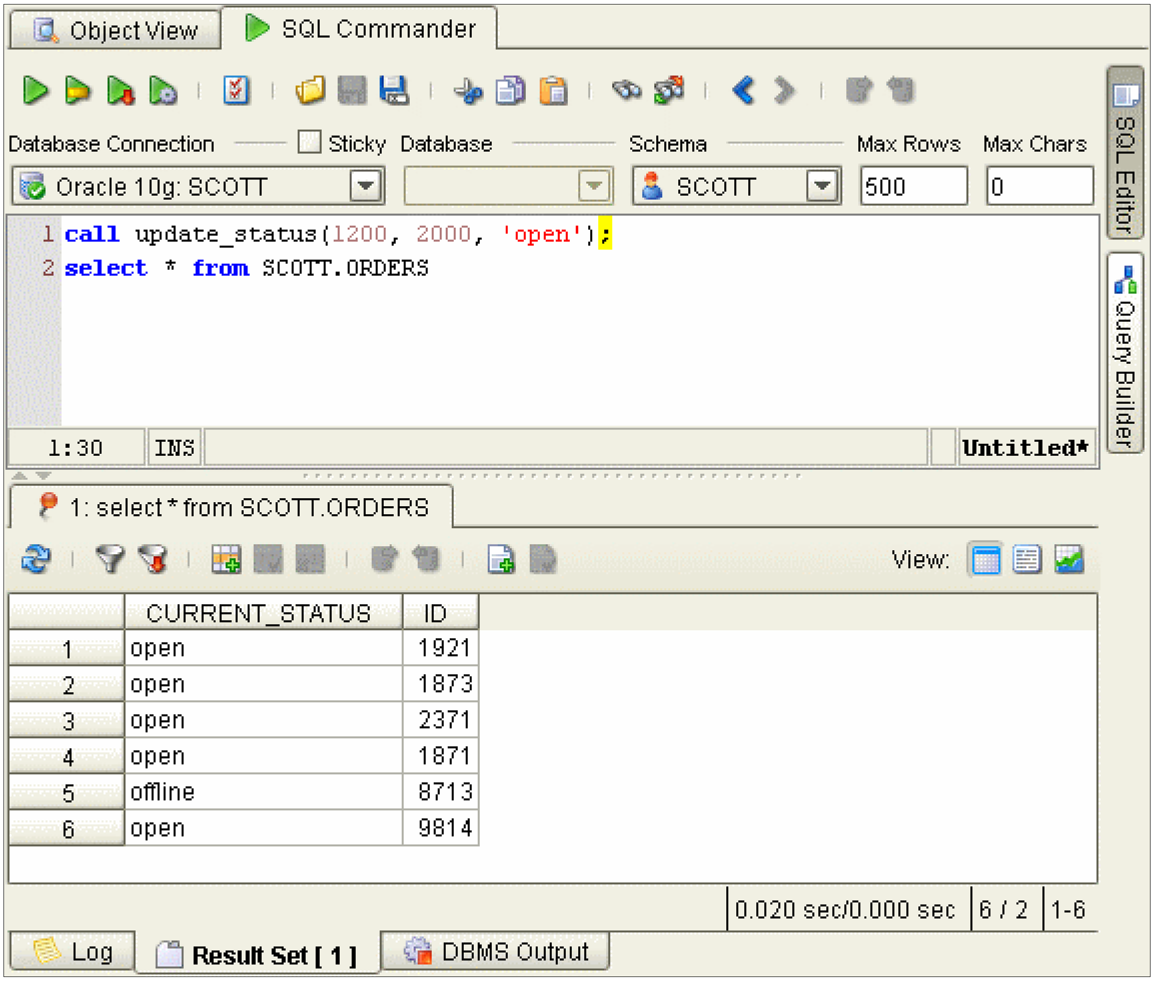
```
create or replace procedure "SCOTT".update_status
(order_id_start IN number DEFAULT -1,
order_id_end IN number DEFAULT -1,
status IN VARCHAR2 DEFAULT 'CLOSED')
AS
BEGIN
    update orders
    set current_status = status
    where id >= order_id_start and id <= order_id_end;
END;
```

**Figure: Compiling procedure with successful result**

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

## Running in SQL Commander

You can now test the procedure in the SQL Commander as in the next screen shot



Max Rows: 500

Max Chars: 0

```
call update_status(1200, 2000, 'open');
select * from SCOTT.ORDERS
```

| | CURRENT_STATUS | ID |
|---|---|---|
| 1 | open | 1921 |
| 2 | open | 1873 |
| 3 | open | 2371 |
| 4 | open | 1871 |
| 5 | offline | 8713 |
| 6 | open | 9814 |

**Figure: Running the procedure in SQL Commander**

The figure shows the invocation of the **moveorder** procedure with parameters meaning that all IDs in the ORDERStable between 1 and 3 should be set to Pre-Closed. The second statement selects from the updated table.

# SQL Bookmarks

## Introduction

The purpose with the bookmark management is to offer a way to save SQL statements between invocations of DbVisualizer and make it easy to execute them. Another important requirement is to organize SQL statements in folders for structural and grouping purposes. The core of the bookmark management is the Bookmark Editor. It is here the bookmarks are organized.

The bookmark editor depends heavily on the SQL Commander since when requesting to execute an SQL Bookmark the bookmark editor will pass the actual SQL along with the connection data to the SQL Commander. It is then the SQL commander that is used to edit and test the SQL until it is complete.

### What's a bookmark in DbVisualizer?

An SQL Bookmark is generally an SQL statement that is saved between invocations of DbVisualizer. In addition it also keeps related information needed to execute the SQL and present the result accordingly once it is requested.

- SQL statement
- Bookmark name
- Database Connection
- Catalog (aka Database)
- Chart settings (optional)

The bookmark management is primarily used to save SQL statements that are used often or for whatever reason there might be. There are different types of bookmarks and DbVisualizer automatically creates bookmarks in the following function areas:

- Each SQL that is executed in the SQL Commander is saved as an SQL bookmark in the History folder
- Each monitored SQL statement in the Monitor feature is an SQL bookmark and is added to the New folder

(Read more about this in the following sections).

## The Bookmarks Main Menu

The bookmarks main menu in the DbVisualizer window contains the following choices:

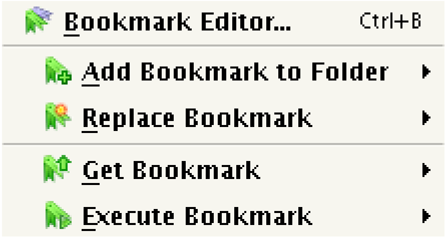
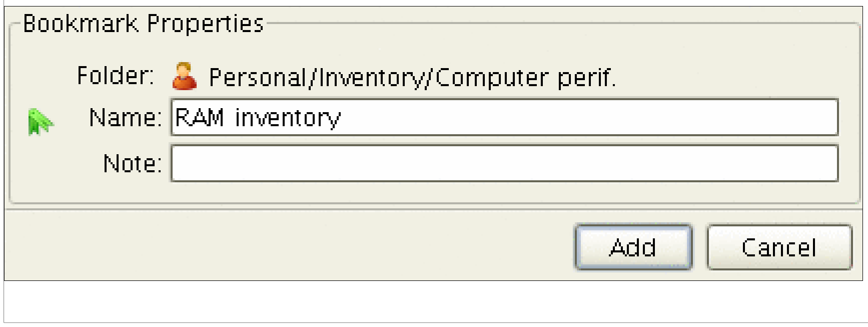
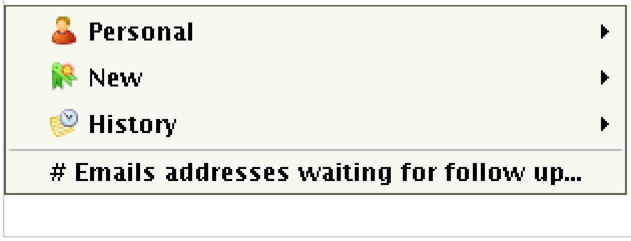


Figure: The Bookmark main menu

All except the Bookmark Editor choice are disabled if you are not in the SQL Commander tab.

| Menu Choice | Description |
|---|---|
| **Bookmark Editor** | Requests to display the **Bookmark Editor**. |
| **Add Bookmark to Folder** | This is composed of a sub menu in which all folders are displayed. This list displays the paths for all folders (i.e the folder hierarchy from the root). The root folders are **Personal**, **New** or **History** (read more about these in the sections below). Once a folder has been selected the following dialog is displayed. Here you can change the default name and add an optional note.<br><br>Bookmark Properties<br>Folder: Personal/Inventory/Computer perif.<br>Name: RAM inventory<br>Note:<br> |
| **Replace Bookmark** | This option is used to replace the chosen SQL bookmark with the SQL and connection data that is in the current SQL Commander editor. The replace bookmark menu consists of the root folders and last in the menu possibly the name of the last SQL bookmark that was passed from the bookmarks editor. If you want to replace the data for that SQL Bookmark just select its name in the menu.<br><br> Personal New History # Emails addresses waiting for follow up... |
| **Get Bookmark** | Get Bookmark shows the same menu hierarchy as Replace Bookmark except that it is used to fetch the chosen SQL Bookmark and insert it into the current SQL Commander |

| | |
|---|---|
| | editor. |
| **Execute Bookmark** | Same as **Get Bookmark** but this one also executes the SQL statement(s) |

# Bookmark Editor

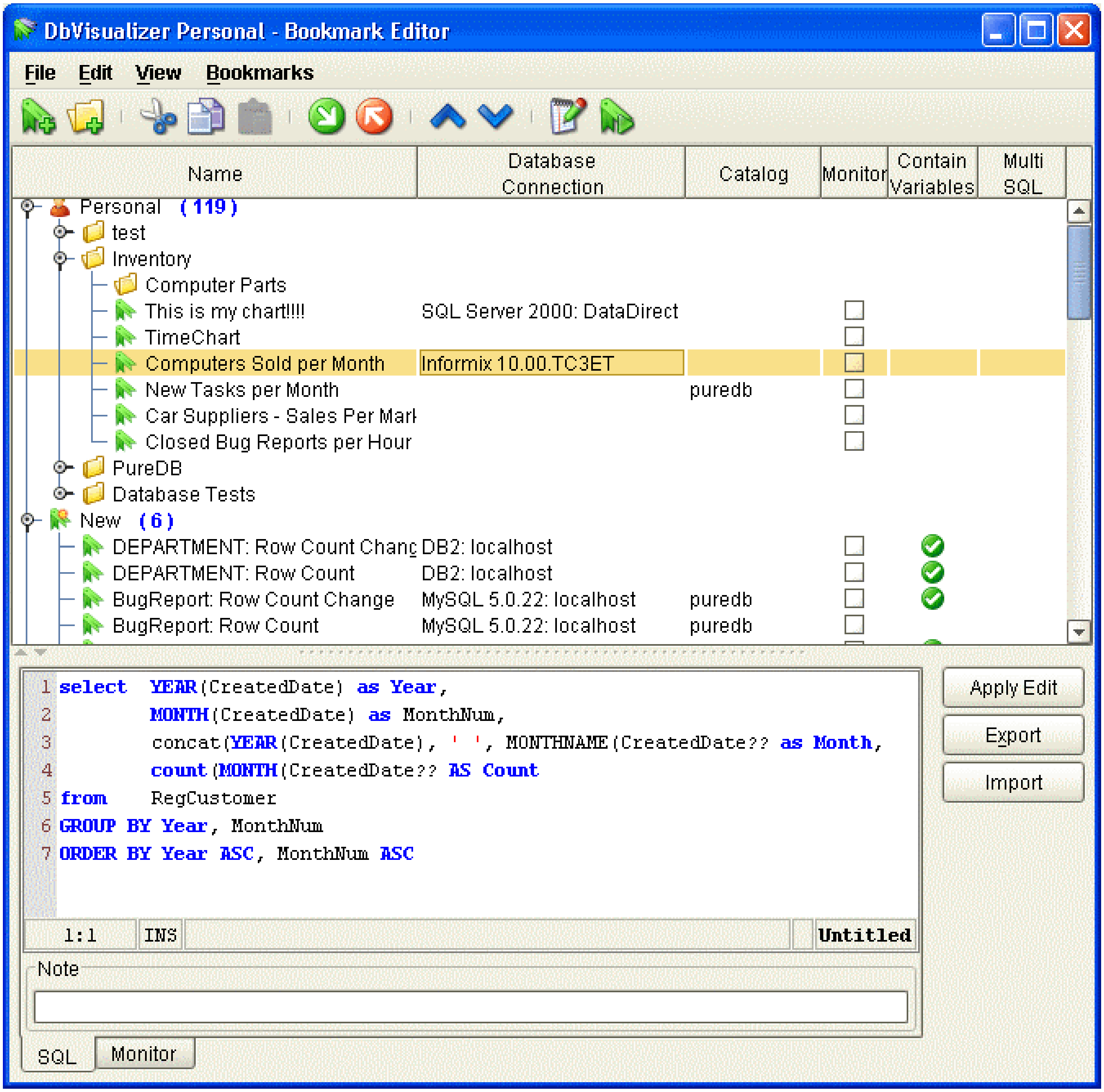
The bookmark editor is the core of the bookmark management and is used to organize SQL bookmarks in folders and to do various adjustments.

## Bookmark list

The editor is based on a tree list with the same structure as the tree that appears in the **Bookmarks** main menu options. The tree has three root folders that cannot be changed, moved or removed. There is basically no difference between these root folders except that they are used in different contexts in DbVisualizer.

- **Personal**
  This root folder is supposed to hold the structure of favorite SQL bookmarks. By putting SQL bookmarks in folders you get a better organization and overview of your bookmarks. All nodes in the root folder are manually maintained.
- **New**
  Creating **Row Count Monitors** in the **Database Objects- >Data** tab will add these monitors (as SQL Bookmarks) to the **New** root folder.
- **History**
  All SQL statements or scripts that are executed in the SQL Commander are automatically added in the **History** root folder. The latest executed statement appear first in the list.

(The number after the root folder names indicates the number of SQL bookmarks that are in that root folder).



| Name | Database Connection | Catalog | Monitor | Contain Variables | Multi SQL |
|---|---|---|---|---|---|
| Personal (119) | | | | | |
| test | | | | | |
| Inventory | | | | | |
| Computer Parts | | | | | |
| This is my chart!!!! | SQL Server 2000: DataDirect | | □ | | |
| TimeChart | | | □ | | |
| Computers Sold per Month | Informix 10.00.TC3ET | | □ | | |
| New Tasks per Month | | puredb | □ | | |
| Car Suppliers - Sales Per Marl | | | □ | | |
| Closed Bug Reports per Hour | | | □ | | |
| PureDB | | | | | |
| Database Tests | | | | | |
| New (6) | | | | | |
| DEPARTMENT: Row Count Chang | DB2: localhost | | □ | ✓ | |
| DEPARTMENT: Row Count | DB2: localhost | | □ | ✓ | |
| BugReport: Row Count Change | MySQL 5.0.22: localhost | puredb | □ | ✓ | |
| BugReport: Row Count | MySQL 5.0.22: localhost | puredb | □ | | |

```
select  YEAR(CreatedDate) as Year,
        MONTH(CreatedDate) as MonthNum,
        concat(YEAR(CreatedDate), ' ', MONTHNAME(CreatedDate?? as Month,
        count(MONTH(CreatedDate?? AS Count
from    RegCustomer
GROUP BY Year, MonthNum
ORDER BY Year ASC, MonthNum ASC
```

Figure: The Bookmark Editor

You cannot create folders or SQL bookmarks in the **New** or **History** root folders. The way to work with these folders is to copy the SQL bookmarks you want from them into the appropriate location in the Personal root folder.

The tree of folders and SQL bookmarks is contains the following information:

| Column in list | Description |
|---|---|
| **Name** | The name of the node (folder or SQL bookmark). Modify the name by selecting the column and click once to get into editor mode. The **Edit->Change Name** menu choice can be used for the same purpose. If a SQL bookmark was created by some |

| | |
|---|---|
| | other function in DbVisualizer then the name will be the first 40 characters of the SQL statement. |
| **Database Connection** | The database connection column when clicked displays a list of all defined database connections. The list indicates whether a connection is established or closed. It is here you specify using another database connection for an SQL bookmark. |
| **Catalog** | This column lists the Catalog (aka Database) that was current when the bookmark was created. You can change the Catalog by clicking in it. A list of accessible catalogs is then displayed.<br>**Note:** The list of catalogs is empty if the Database Connection is not established. |
| **Monitor** | Check this box to enable the SQL bookmark to become a monitor and thereby appear in the Monitor main tab. SQL's that returns results are the most obvious candidates for being monitored. |
| **Contain Variables** | This column is read only and indicates whether the SQL statement includes any DbVisualizer variables. (I.e `$$variable name$$`) |
| **Multi SQL** | This column is also read only and indicates whether the SQL statement is composed of several SQL statements (aka script). This is determined by looking for statement delimiters in the SQL. |

## New and History root folders

The number of SQL bookmarks that may be added by DbVisualizer to the **New** and **History** root folders are specified in the **Tool Properties->Bookmarks** category. Bookmarks in these folders can be removed one by one or each folder can be cleared using the **File->Clear all New entries** or **File->Clear all History** entries.

## SQL Editor

### Monitor information

The monitor sub tab controls the total number of rows that will be kept in the result grid until rows are automatically removed. This feature is specific to the monitor feature. Please see Charts and Monitors for more information.

### The Note field

The note field can be used to write a short note about the SQL Bookmark. This note will appear as a tool tip in the Bookmarks main menu.

## Executing an SQL bookmark or folder of SQL bookmarks

The SQL editor in the bookmark editor can be used to modify the SQL but it is not the place to execute SQL statements. Instead it is the SQL Commander that is used to execute SQL's. Select the **Edit->Copy to SQL Commander** or **Edit->Execute Bookmark** menu operation to copy the selected SQL bookmark into the SQL Commander. The SQL Commander is then used to execute and edit the SQL. Once you are satisfied with it then select the last entry in the Bookmarks menu:

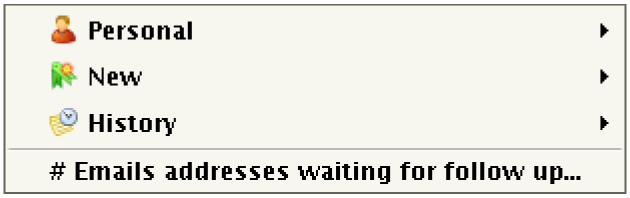
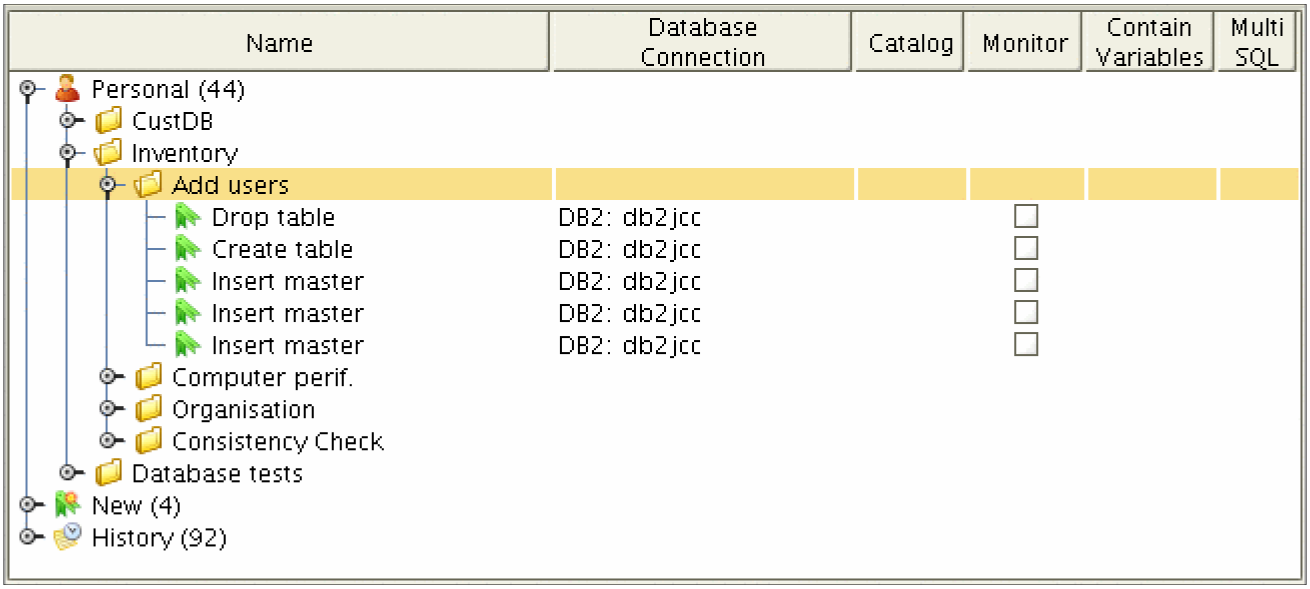


Figure: The Bookmark->Replace sub menu

If the last entry displays "No current bookmark" then it indicates that the currently edited SQL was not passed from the Bookmark Editor. You can use the other menu choices to locate the actual bookmark that will be replaced.

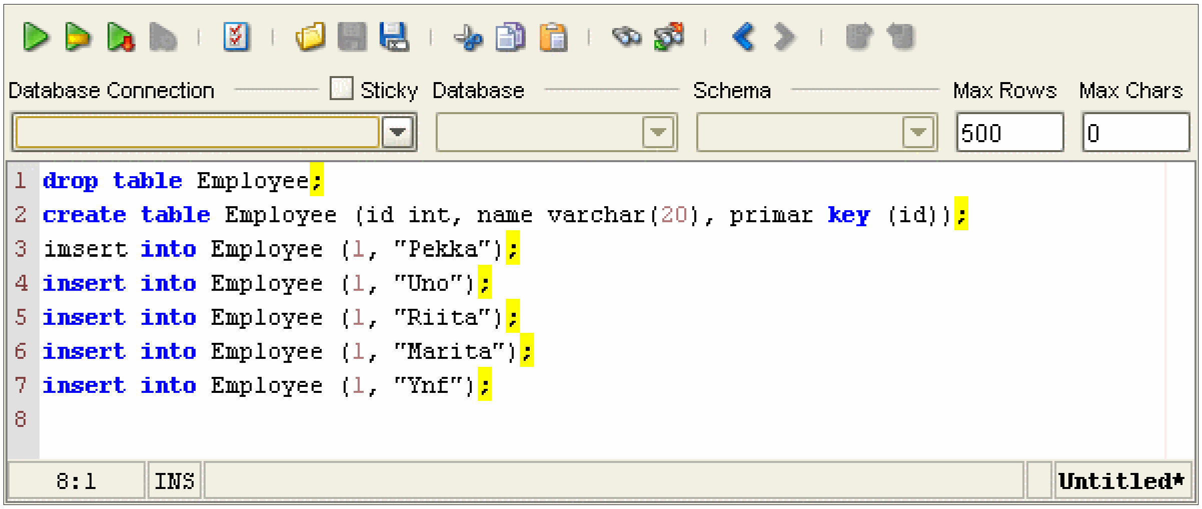
The **Copy to SQL Commander** and **Execute Bookmark** operations can also operate on a folder. Doing this will result in a script of all direct child SQL bookmarks that are located in that folder. Each of the SQL statements will be delimited by the delimiter as specified in Tool Properties.



| Name | Database Connection | Catalog | Monitor | Contain Variables | Multi SQL |
|---|---|---|---|---|---|
| Personal (44) | | | | | |
| CustDB | | | | | |
| Inventory | | | | | |
| Add users | | | | | |
| Drop table | DB2: db2jcc | | □ | | |
| Create table | DB2: db2jcc | | □ | | |
| Insert master | DB2: db2jcc | | □ | | |
| Insert master | DB2: db2jcc | | □ | | |
| Insert master | DB2: db2jcc | | □ | | |
| Computer perif. | | | | | |
| Organisation | | | | | |
| Consistency Check | | | | | |
| Database tests | | | | | |
| New (4) | | | | | |
| History (92) | | | | | |

Figure: Selecting a folder for execution

Choosing the **Copy to SQL Commander** for a folder of bookmarks will result in the following being produced in the SQL Commander:



Max Rows 500

Max Chars 0

```
drop table Employee;
create table Employee (id int, name varchar(20), primar key (id));
imsert into Employee (1, "Pekka");
insert into Employee (1, "Uno");
insert into Employee (1, "Riita");
insert into Employee (1, "Marita");
insert into Employee (1, "Ynf");
```

Figure: The SQL Commander editor

Note that the **Database Connection** and **Catalog** lists are empty. You need to select these from the lists when a script of SQL bookmarks is passed from the Bookmark Editor.

# Tool Properties

## Customizing DbVisualizer

DbVisualizer is highly customizable, you can control formatting, layout and the way DbVisualizer interacts with databases. The default settings are good enough for the normal user but sometimes it is necessary to modify these properties. This chapter guides you through all the properties.

Properties are divided into two groups:

- **General Settings**
  These settings controls DbVisualizer in general such as fonts, colors, data formats, etc.
- **Database Settings**
  These settings are per supported database type and defines properties that are used in database specific operations. Changing a database property in Tool Properties will then apply for all database connections defined for that database type. In Connection Properties it is also possible to override database properties specifically for a database connection.

### The user preferences (XML) file

All properties are saved in an XML file. The exact location of this file is platform dependent. The location on your system is listed in the first, **General** category. The XML file contains, in addition to all properties, also the information about drivers, database connections, bookmarks, etc. The general recommendation is to **not edit** this file manually even though it is quite easy to do so.

DbVisualizer automatically creates a backup copy of the XML file when the application is started. The location of this file is the same as for the standard XML file except that the **.bak** suffix is appended to the file name. The standard XML file might get broken for various reasons. If a warning message that the XML file could not be read is displayed during launch of DbVisualizer then simply copy the backup file to the standard location and restart the application. If the XML file is moved from its standard location or if it is removed then DbVisualizer will automatically create a new one.

**Tip:** the **- up** command line argument is used to identify the file name (and path) to an alternate XML file.

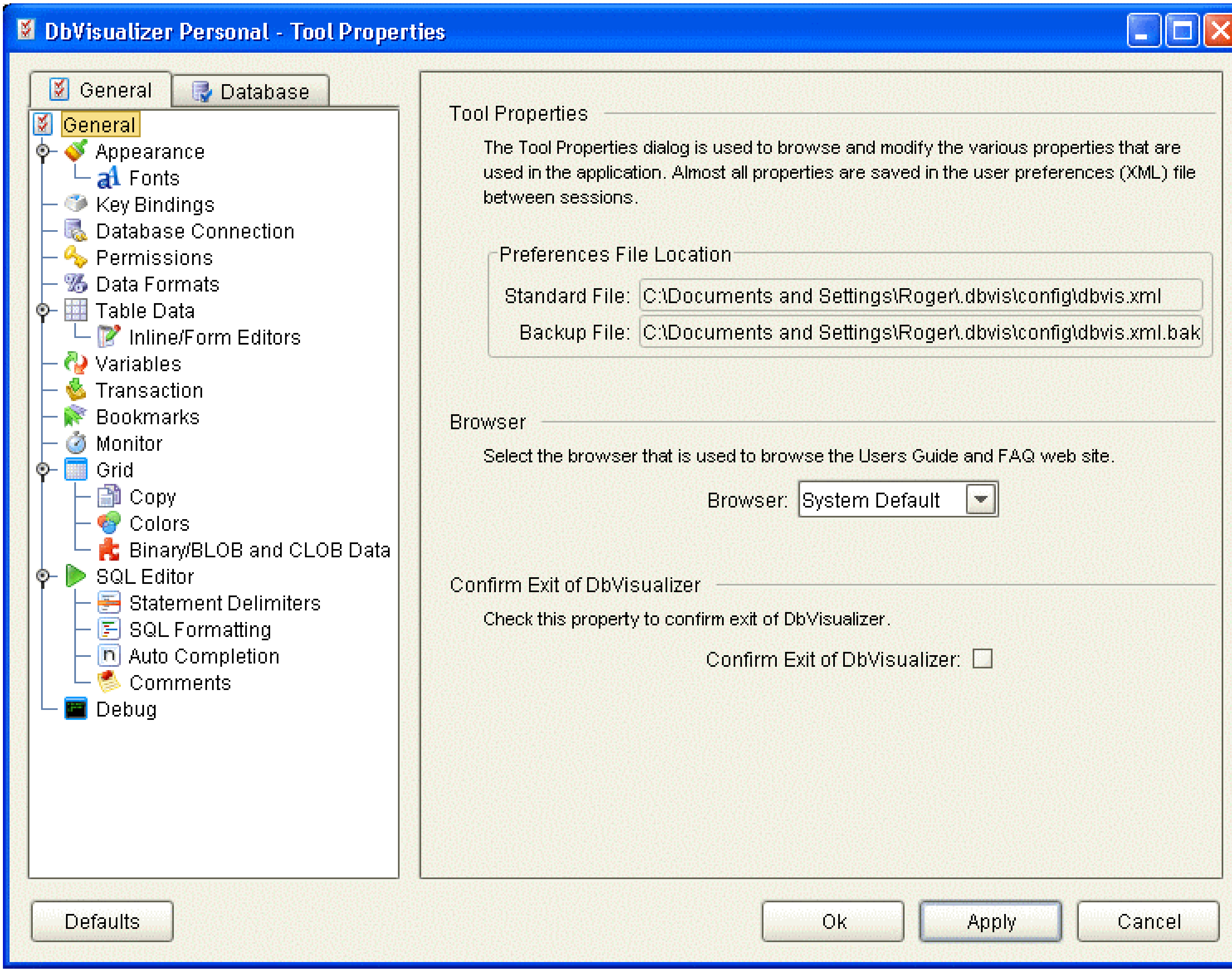
## General Settings

The General settings tab collects all categories that are used to control the general aspects of DbVisualizer.

The buttons at the bottom of the window control whether the changed properties should be applied using the **Ok** (this also closes the window) and **Apply** button, if changes should be reverted using the **Cancel** button or if the factory defaults should be applied using the **Defaults** button.

Changes are tracked on a per category basis. If any changes has been made then a question will be displayed whether the changes should be applied or not. **Defaults** can be initiated either to revert all properties (both **General** and **Database** properties) to their default settings or just the current category.

This is a screenshot of the **General** category tree.



Tool Properties

The Tool Properties dialog is used to browse and modify the various properties that are used in the application. Almost all properties are saved in the user preferences (XML) file between sessions.

Preferences File Location

Standard File: C:\Documents and Settings\Roger\.dbvis\config\dbvis.xml

Backup File: C:\Documents and Settings\Roger\.dbvis\config\dbvis.xml.bak

Browser

Select the browser that is used to browse the Users Guide and FAQ web site.

Browser: System Default

Confirm Exit of DbVisualizer

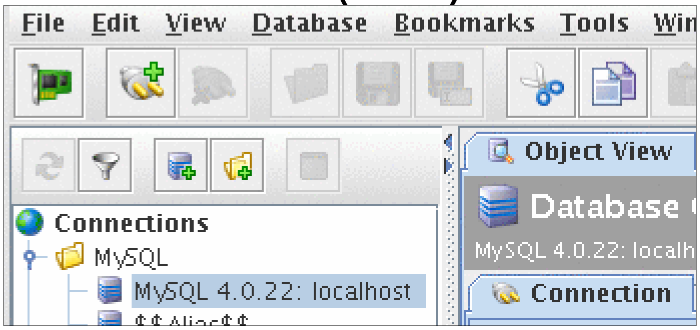
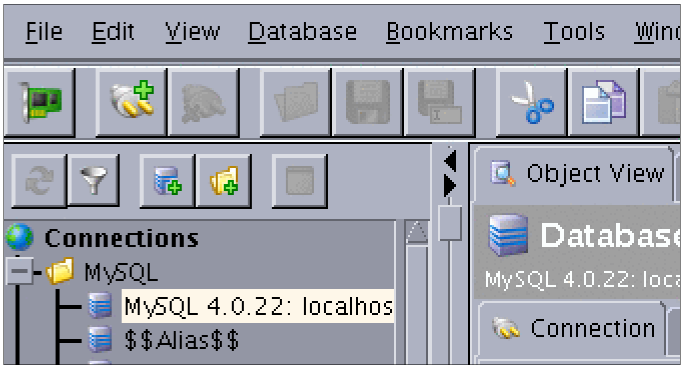
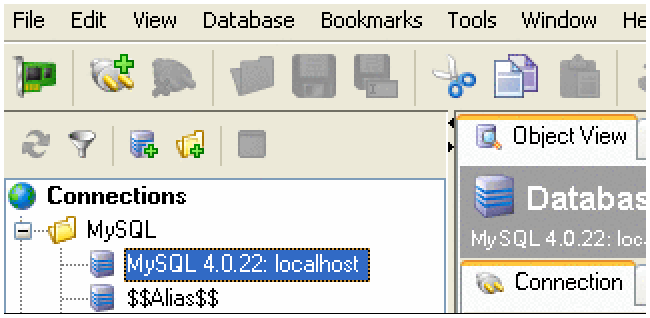
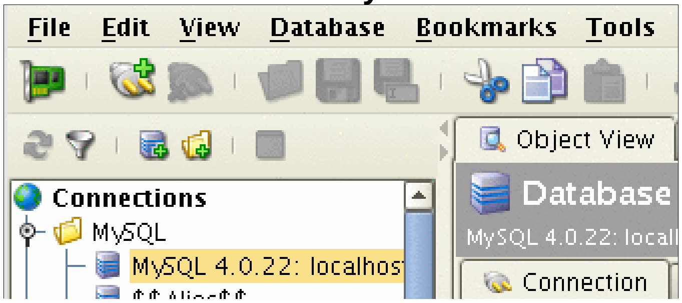
Check this property to confirm exit of DbVisualizer.

Confirm Exit of DbVisualizer: ☐

**Figure: The Tool Properties window showing the tree with General categories**

## Appearance

| Property | Description |
|---|---|

| | |
|---|---|
| **Look and Feel** | Controls which look and feel will be used.<br>**Note 1:** You must restart DbVisualizer in order to use a new look and feel.<br>**Note 2:** Some look and feels are platform specific and do not appear on all OS'es<br>**Metal (Ocean)**<br><br> File Edit View Database Bookmarks Tools Win<br>Object View<br>Database<br>MySQL 4.0.22: localh<br>Connection<br>Connections<br>MySQL<br>MySQL 4.0.22: localhost<br>$$Alias$$<br>**Motif**<br><br> File Edit View Database Bookmarks Tools Wind<br>Object View<br>Database<br>MySQL 4.0.22: loca<br>Connection<br>Connections<br>MySQL<br>MySQL 4.0.22: localhos<br>$$Alias$$<br>**Windows**<br><br> File Edit View Database Bookmarks Tools Window He<br>Object View<br>Databas<br>MySQL 4.0.22: loc<br>Connection<br>Connections<br>MySQL<br>MySQL 4.0.22: localhost<br>$$Alias$$<br>**Alloy**<br><br> File Edit View Database Bookmarks Tools<br>Object View<br>Database<br>MySQL 4.0.22: locall<br>Connection<br>Connections<br>MySQL<br>MySQL 4.0.22: localhost<br>$$Alias$$ |

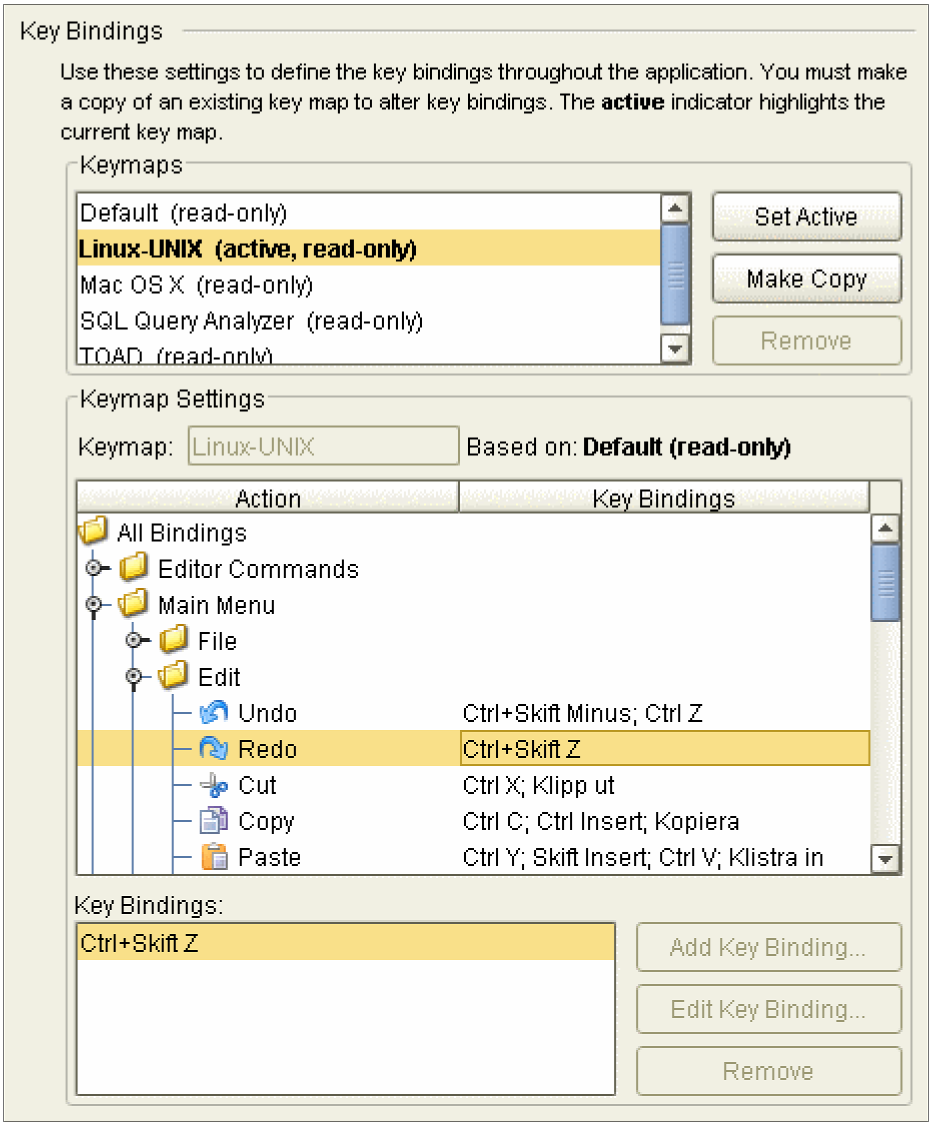
| | |
|---|---|
| **Icon Sizes** | The **Menus**, **Main Tool Bars**, **Sub Tool Bars** settings are used to control the size of the icons. |
| **Show Tab Icons** | Specifies whether an icon will appear in the header of all object view tabs. |

### Fonts

Individual fonts can be defined for **SQL editors**, **Grids** and **Text** output data. The **Application Font Size** settings is used to control the size of the font for all other components in the user interface. Increasing the application font size is useful at demos or presentations.

## Key Bindings

The key binding function is used to define key bindings for almost all operations and editor commands in DbVisualizer. Key bindings are grouped in **Key Maps**. DbVisualizer includes a set of pre- defined key maps targeted for the supported operating systems. These key maps cannot be deleted or modified. To customize key bindings, then copy an existing key map and make your changes.



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

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

Keymap: Linux-UNIX Based on: **Default (read-only)**

| Action | Key Bindings |
|---|---|
| All Bindings | |
| Editor Commands | |
| Main Menu | |
| File | |
| Edit | |
| Undo | Ctrl+Skift Minus; Ctrl Z |
| Redo | Ctrl+Skift Z |
| Cut | Ctrl X; Klipp ut |
| Copy | Ctrl C; Ctrl Insert; Kopiera |
| Paste | Ctrl Y; Skift Insert; Ctrl V; Klistra in |

Key Bindings:

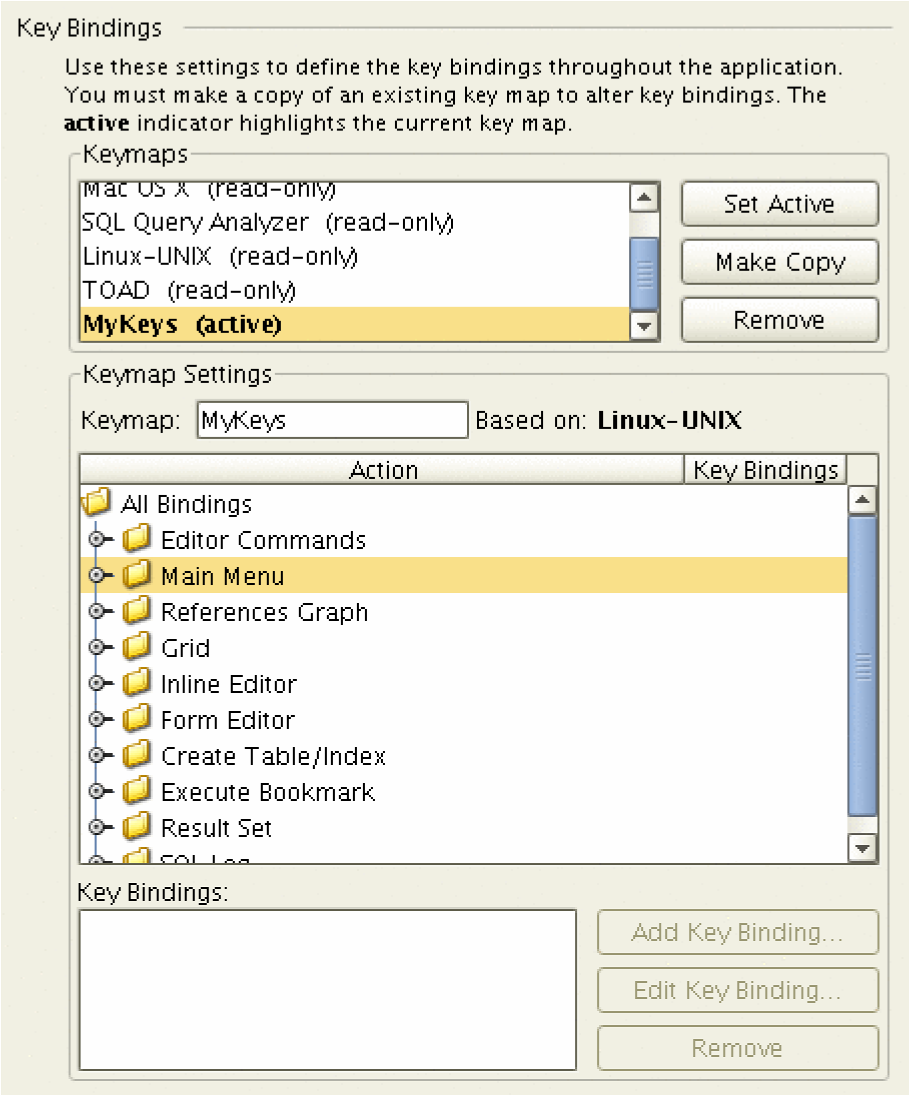
Ctrl+Skift Z

**Figure: The key binding editor**

All user defined key maps are stored in your **$HOME/.dbvis/config/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 press 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 will now have the exact same key bindings as the parent key map.

**Note:** Key maps must be uniquely named.



Key Bindings

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

Keymaps

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

Keymap Settings

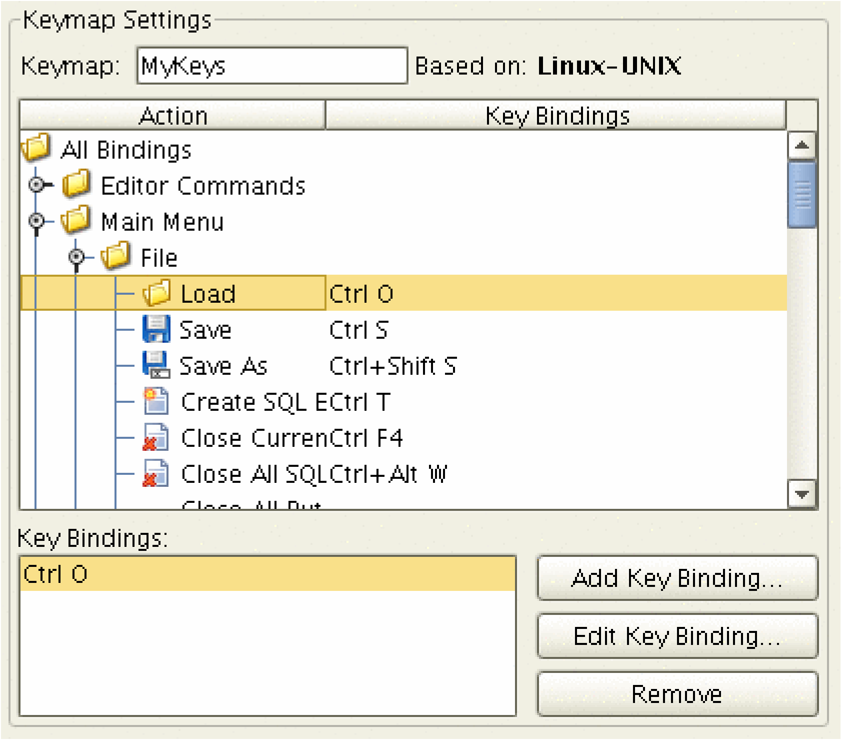
Keymap: MyKeys Based on: **Linux-UNIX**

All Bindings
Editor Commands
Main Menu
References Graph
Grid
Inline Editor
Form Editor
Create Table/Index
Execute Bookmark
Result Set
SQL Log

**Figure: User defined key map**

The action list is organized in folders. The **Editor Commands** folder lists all actions available in the SQL Commander editor and their current key bindings. The **Main Menu** folder contain sub folders each representing the main window menu actions. 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 then select the action from the action list. The current key bindings are listed in the **Key Bindings** list.



Keymap Settings

Keymap: MyKeys Based on: **Linux-UNIX**

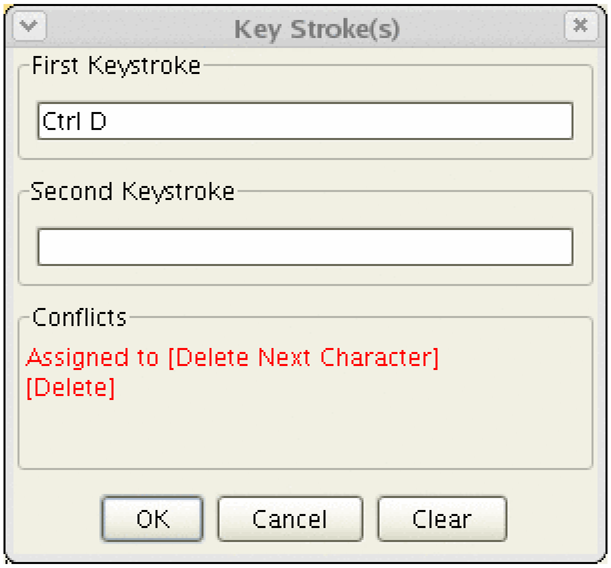
| Action | Key Bindings |
|---|---|
| All Bindings | |
| Editor Commands | |
| Main Menu | |
| File | |
| Load | Ctrl O |
| Save | Ctrl S |
| Save As | Ctrl+Shift S |
| Create SQL E | Ctrl T |
| Close Curren | Ctrl F4 |
| Close All SQL | Ctrl+Alt W |
| Close All Dat | |

Key Bindings:

Ctrl O

**Figure: User defined key map**

To add an additional key binding press **Add Key Binding** or press **Edit Key Binding** to edit the selection.



Key Stroke(s)

First Keystroke

Ctrl D

Second Keystroke

Conflicts

Assigned to [Delete Next Character]
[Delete]

**Figure: Key stroke dialog**

The key stroke dialog controls whether a key binding is already assigned somewhere else. In the conflict box you'll see 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.

**Note:** It is not recommended to assign several key bindings for different actions. The reason is that you may get different results between invocations of such key bindings.

**Note:** Menu items and tool tips shows the first defined key binding in the list.

## Database Connection

| Property | Description |
|---|---|
| **Run "Connect All" at Startup** | Defines whether a database connection will be connected when the **Connect All** operation is selected in the main window menu bar. |
| **Confirm "Disconnect All"** | Checking this property will force a dialog to be displayed before disconnecting all current database connections using the Disconnect All operation. |
| **Connection Timeout** | Specify number of seconds that the driver will wait until terminating ongoing connection request.<br>**Note:** This property is handled by JDBC drivers and might not be supported. |

## Permissions

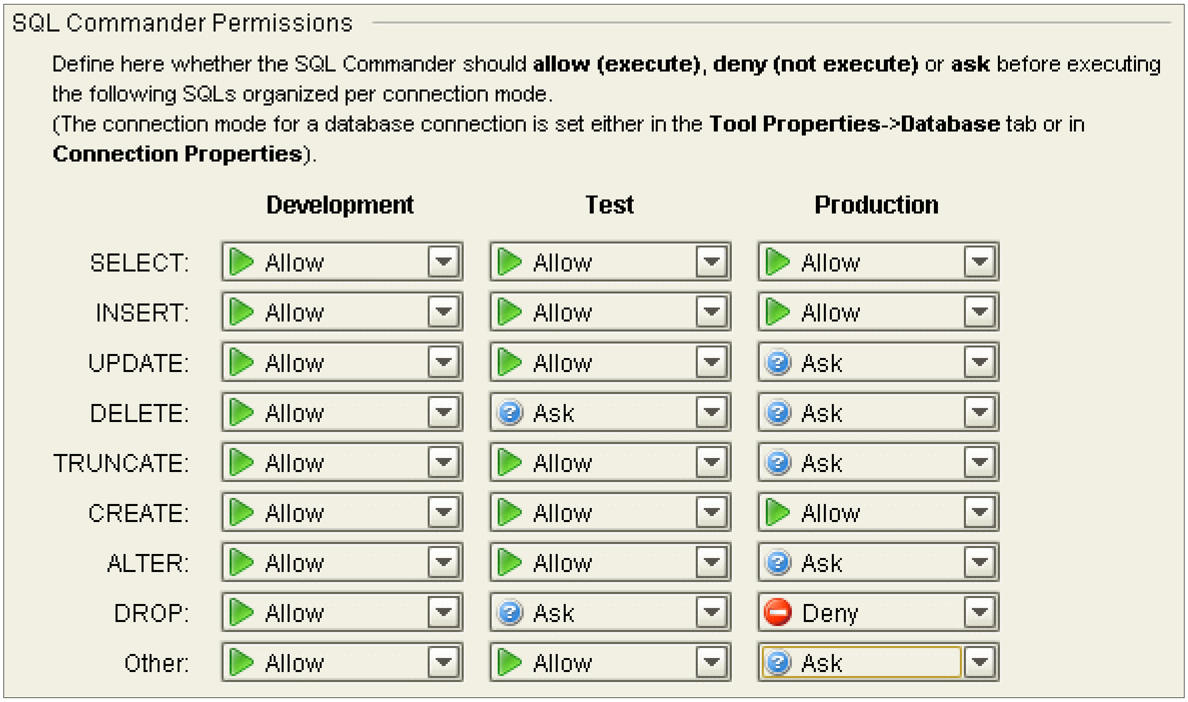
The Permission functionality is a security mechanism preventing from running certain database operations unconfirmed. Permissions are configured per connection mode and are categorized into the following feature areas.

**Note:** The permission feature is part of DbVisualizer and should not be mixed with any authorization system in the actual database.

### SQL Commander Permissions

For the SQL Commander we define via a drop down the permission type for each SQL command:

- **Allow**
  This type will run the actual SQL without any confirmation
- **Deny**
  This type will simply ignore running the actual SQL command
- **Ask**
  When executing the SQL statement or script of statements the SQL Commander will first ask the user whether the actual SQL command(s) should be executed or not.



SQL Commander Permissions

Define here whether the SQL Commander should **allow (execute)**, **deny (not execute)** or **ask** before executing the following SQLs organized per connection mode.
(The connection mode for a database connection is set either in the **Tool Properties->Database** tab or in **Connection Properties**).

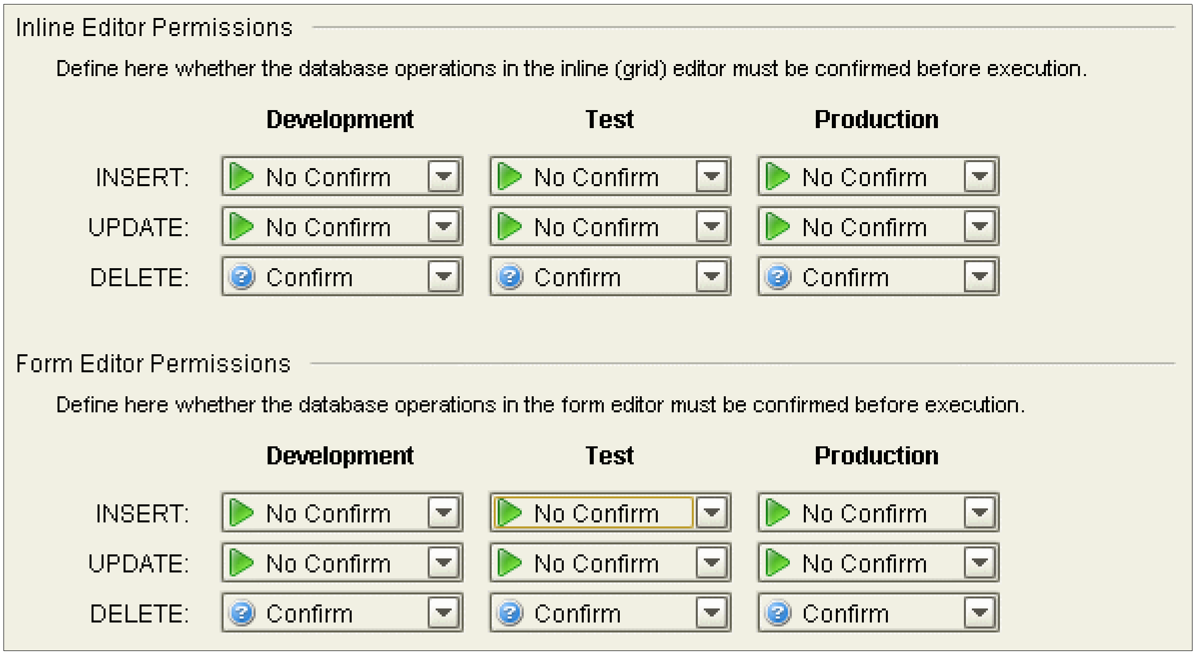
| | Development | Test | Production |
|---|---|---|---|
| SELECT: | Allow | Allow | Allow |
| INSERT: | Allow | Allow | Allow |
| UPDATE: | Allow | Allow | Ask |
| DELETE: | Allow | Ask | Ask |
| TRUNCATE: | Allow | Allow | Ask |
| CREATE: | Allow | Allow | Allow |
| ALTER: | Allow | Allow | Ask |
| DROP: | Allow | Ask | Deny |
| Other: | Allow | Allow | Ask |

**Figure: SQL Commander Permissions**

## Inline and Form Editor Permissions

The permissions for inline and form editors are:

- **Confirm**
  A confirmation window will be displayed in which the user must accept the operation or cancel it
- **No Confirm**
  The SQL operation is performed without any confirmation being displayed



Inline Editor Permissions

Define here whether the database operations in the inline (grid) editor must be confirmed before execution.

| | Development | Test | Production |
|---|---|---|---|
| INSERT: | No Confirm | No Confirm | No Confirm |
| UPDATE: | No Confirm | No Confirm | No Confirm |
| DELETE: | Confirm | Confirm | Confirm |

Form Editor Permissions

Define here whether the database operations in the form editor must be confirmed before execution.

| | Development | Test | Production |
|---|---|---|---|
| INSERT: | No Confirm | No Confirm | No Confirm |
| UPDATE: | No Confirm | No Confirm | No Confirm |
| DELETE: | Confirm | Confirm | Confirm |

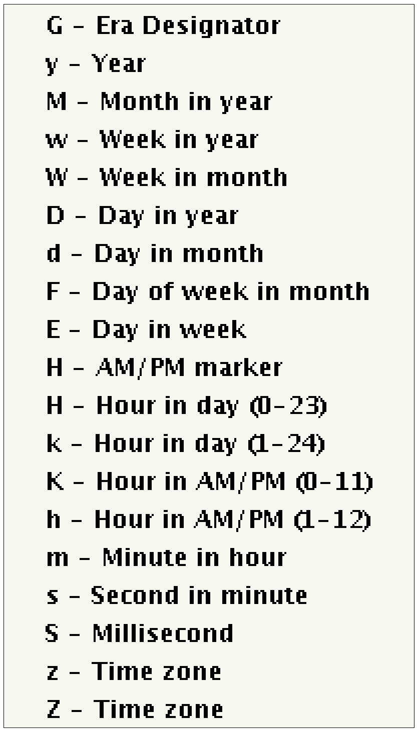
**Figure: Inline and Form Editor Permissions**

## Data Formats

| Property | Description |
|---|---|
| **Date Format** | Select the date format that will be used throughout the application (i.e grids, forms and during editing). More information below. |
| **Time Format** | Select the time format that will be used throughout the application (i.e grids, forms and during editing). More information below. |
| **Timestamp Format** | Select the timestamp format that will be used throughout the application (i.e grids, forms and during editing). More information below. |
| **Numbers Format** | Specifies how numbers will be formatted. |
| **Decimal Number Format** | Specifies how decimal numbers will be formatted. |
| **Null String** | This is the string representation of the null value. This string is the readable form of null and appears in grids, forms, exports and during editing. |

### Date, Time and Timestamp formats

The lists for date, time and timestamp format contains a collection of standard formats. If these formats are not suitable then you can enter your own format in the appropriate field. The tokens used to define the format is listed in the right click menu while 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
H – AM/PM marker
H – Hour in day (0-23)
k – Hour in day (1-24)
K – Hour in AM/PM (0-11)
h – Hour in AM/PM (1-12)
m – Minute in hour
s – Second in minute
S – Millisecond
z – Time zone
Z – Time zone

**Figure: The date and time right click menu**

The complete documentation for these tokens are listed in the following web page SimpleDateFormat.

## Table Data

| Property | Description |
|---|---|
| **Show Table Row Count** | Specifies if the number of rows in a table will be displayed in the header of the table when in the Database Objects->Data tab. Enabling this property will cause an extra round trip to the database (i.e minor performance penalty) |
| **Highlight Primary Key Columns** | Specifies if Primary Key columns will be indicated in the Database Objects->Data tab, Variable Substitution dialog, SQL Commander Result grids and in the References Graph. |
| **Include Variables in SQL** | Specifies if the right click menu operations in the Data tab will create appropriate SQL statements that include DbVisualizer variables or if the generated statements are plain SQL. Letting DbVisualizer generating statements with variables results in the variable substitution dialog being displayed when these statements are executed in the SQL |

| | |
|---|---|
| | Commander. |
| **Max Rows at First Display** | Set the number of rows that will be fetched for a table in the Data tab when a table is first displayed. |

### Inline/Form Editors

| Property | Description |
|---|---|
| **Reload Grid after Edit** | Check this to enable auto reloading of the grid after a successful edit in the inline editor. |
| **Image Max Size in Form Editor** | The default size for images displayed in the form editor. |

## Variables

Variables can be used in the SQL executed in the SQL Commander and in Connection details. Before executing an SQL statement or connecting a database connection a window is display asking for replacement values.

These settings define what character sequence identifies a variable.

| Property | Description |
|---|---|
| **Variable Identifier** | The identifier for a variable. A variable starts and ends with this identifier. Default is "$$". |
| **Variable Delimiter** | The delimiter used to identify the parts of a variable. Default is "\|\|". |

## Transaction

| Property | Description |
|---|---|
| **Pending Transactions at Disconnect** | Defines what DbVisualizer will do on exit from the application when the auto commit setting is disabled. |

## Bookmarks

| Property | Description |
|---|---|
| **Number of Bookmarks Limit** | Specifies the number of SQL bookmarks that the **New** and **History** bookmark object may keep until the lists are truncated. |

## Monitor

| Property | Description |
|---|---|
| **Start Monitors Automatically** | Check to enable start of monitors automatically when database connections are established. |

## Grid

| Property | Description |
|---|---|
| **Fit Grid Column Widths** | Enable to let DbVisualizer automatically fit the content in each grid column based on the widest cell value. |
| **Max Column Separator Width** | This setting is used only **when Fit Grid Column Widths** is enabled and is used to set a maximum visual column width for grids. |
| **Meaning of setting Max Chars** | The Max Chars property in the Database Objects Data tab and in the SQL Commander is used to control the max number of characters that text values can hold. If the number of characters for a text column is wider then this setting then the column is colored in a light red color.<br>The meaning of setting this property can be one of the following:<br>• **Truncate Values**<br>Will truncate the original value to be less then the setting of Max Chars.<br>**Note:** this will affect any subsequent edits and SQL operations that use the value since it's truncated. This setting is only useful to save memory if viewing very large text |

| | |
|---|---|
| | columns.<br>• **Truncate Values Visually**<br>Will truncate the visible value only and leave the original value intact. This is the preferred setting since it will not harm the original value. The disadvantage is that more memory is needed if dealing with large text columns. |

### Copy

The copy category groups properties that are used to control the result of using **Copy Selection** and **Copy Selection (With Column Header)** via the grid right click menu.

| Property | Description |
|---|---|
| **Column Delimiter** | Specifies the delimiter between columns in a multi column copy |
| **End of Line Delimiter** | Specifies the new line control characters for multi row copy requests |

### Colors

The Colors category defines how odd and even numbered rows in grids should be presented.

### Binary/BLOB and CLOB Data

| Property | Description |
|---|---|
| **BLOB** | Specifies how BLOB and binary data values will be represented in grids. Setting this property to **By Value** will result in performance penalties and the memory consumption will increase dramatically. |
| **CLOB** | Specifies how BLOB and binary data values will be represented in grids. Setting this property to **By Value** will result in performance penalties and the memory consumption will increase dramatically. |

## SQL Editor

The editor category controls various settings specific for the SQL Commander editor.

| Property | Description |
|---|---|
| **Tabs** | This is used to define settings for the tab keyboard key. |
| **Recent Files Limit** | Specifies the max number of files listed in the **File->Load Recent** sub menu. |
| **Confirm Close of Unsaved Editors** | Enable this and DbVisualizer will ask for unsaved editors (and not only editors loaded from file) whether to save to file or not. |
| **Set "Sticky" for SQL Editor(s)** | When this is enabled new SQL Editors automatically will be defined as Sticky meaning that the database connection details only can be changed manually. |

**Statement Delimiters**

Statement delimiters define how a script should be divided into specific SQL statements in the pre-processing phase.

| Property | Description |
|---|---|
| **SQL Statement Delimiter 1** | Defines the character(s) used to delimit one SQL statement from another in a SQL script |
| **SQL Statement Delimiter 2** | Defines the additional character(s) used to delimit one SQL statement from another in a SQL script. If there is no need for more then one SQL statement delimiter then set this one to the same as delimiter 1. |
| **Allow "go" as Delimiter** | Specifies whether **go** as the first word on a single line will be interpreted as a statement delimiter. |
| **Begin Identifier** | Defines the character(s) that identifies the start of an anonymous SQL block |
| **End Identifier** | Defines the character(s) that identifies the end of an anonymous SQL block |

### SQL Formatting

The SQL formatting category groups properties to control the SQL formatting feature in the SQL Commander. To see the effect of each property, modify it, press **Apply** and format the SQL in the SQL Commander to see the result.

### Auto Completion

These category is used to define the visual appearance of the auto completion popup in SQL Editors.

| Property | Description |
|---|---|
| **Sort Tables List** | Enable this to always present tables sorted in the auto completion popup |
| **Sort Columns List** | Enable this to always present column names sorted in the auto completion popup |
| **Display Automatically** | Enable this and the auto completion popup is automatically displayed whenever possible |
| **Instant Substitution** | Enable this and the auto completion popup is only displayed when needed |
| **Display Delay** | Specifies the time in milliseconds until the auto completion popup is displayed automatically |

### Comments

| Property | Description |
|---|---|
| **Single Line Identifier 1** | Defines the character(s) that identifies the beginning of a one line comment |
| **Single Line Identifier 2** | Defines the additional character(s) that identifies the beginning of a one line comment |
| **Block Comment Begin Identifier** | Specifies the character(s) that identifies the start of a multi line comment block |
| **End** | Specifies the character(s) that identifies the end of a multi line comment block |

### Debug

The debug category is used to control the amount of output that is produced when setting various debug modes. Normally only error messages are displayed in the default debug destination which is the **Tools->Debug Window**. The support team often refer to the debug properties when we want more information in a problem situation.

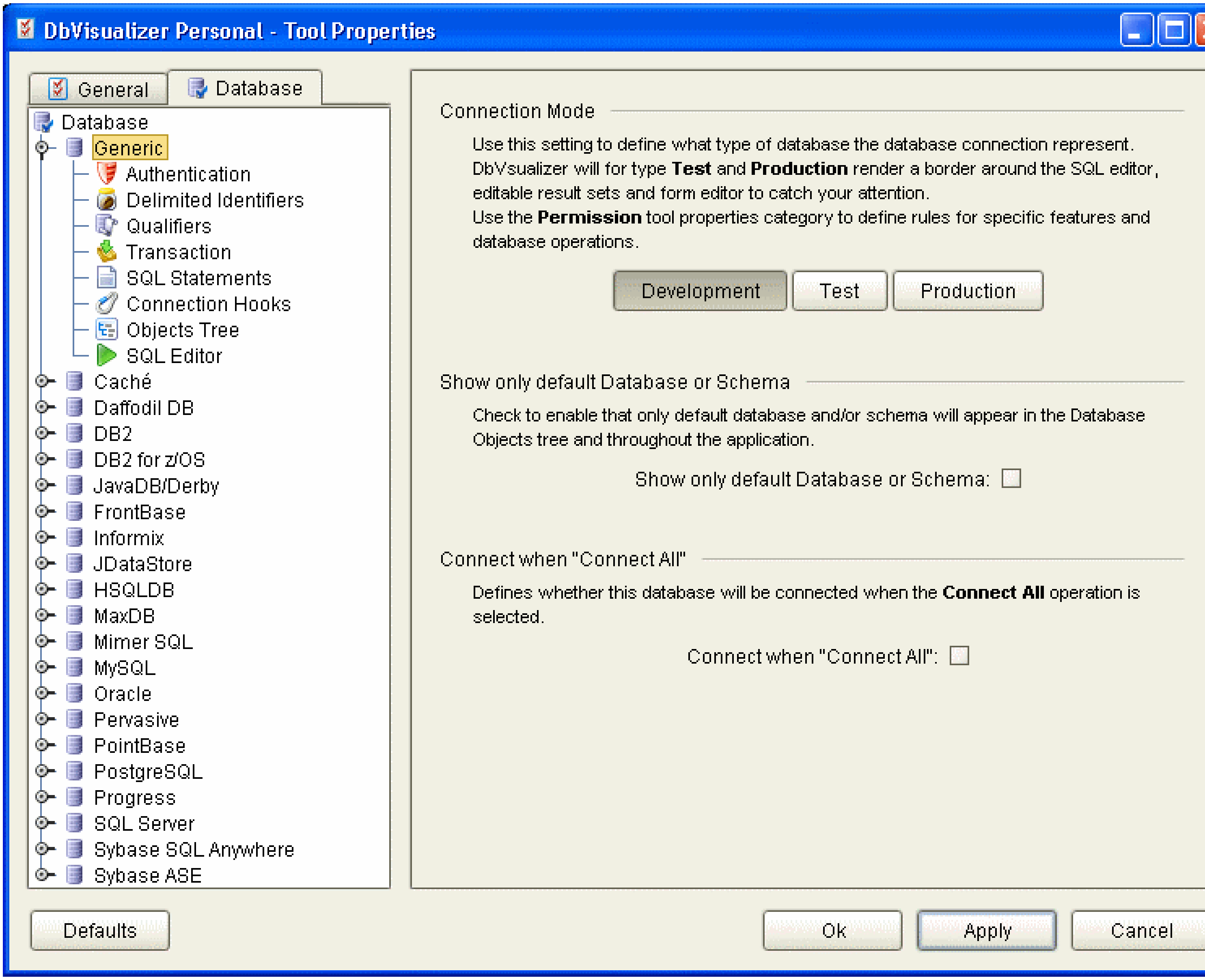
| Property | Description |
|---|---|
| **Debug Output Destination** | Specifies the destination to which all debug messages will be written to. It is not advisable to set this to **Off** since then also error messages will then also be ignored. **Standard Out** is only useful if the debug mode of the DbVisualizer launcher is enabled. |
| **Debug DbVisualizer** | Defines the amount of logging that will be produced. **Full** output is when **Log Level** is set to Debug and lowest output is **Error**. Setting **Detail Level** to **Full** produces the most detail and also consume more resources. |
| **Debug JDBC Drivers** | This property enables any debug output produced by a JDBC driver. The amount of output depends on the actual drivers. |

Read more about Problem Resolution.

## Database Settings

Database settings extends the General settings with properties that are defined per supported database type. The selection of what database type is current for a database connection choose the appropriate type in the **Database Type** list in the Connection tab. If there is no matching entry use the **Generic** database type.

Having database type specific properties is useful as settings can be defined for all database connections instead of per individual database connections. It is also possible to override these properties in the Connection Properties tab.



Connection Mode

Use this setting to define what type of database the database connection represent. DbVsualizer will for type **Test** and **Production** render a border around the SQL editor, editable result sets and form editor to catch your attention.
Use the **Permission** tool properties category to define rules for specific features and database operations.

[✓ Development] Test Production

Show only default Database or Schema

Check to enable that only default database and/or schema will appear in the Database Objects tree and throughout the application.

Show only default Database or Schema: ☐

Connect when "Connect All"

Defines whether this database will be connected when the **Connect All** operation is selected.

Connect when "Connect All": ☐

**Figure: The Tool Properties window showing the tree with Database categories**

The following properties are displayed when selecting a database type in the tree.

| Property | Description |
|---|---|
| **Connection Mode** | Specify here what mode the database connection is. **Permissions** are based on connection mode as well as a visual border around critical features in DbVisualizer. |
| **Show only default Database or Schema** | Enable this if you only want the default database or schema listed in the database objects tree. |
| **Connect when "Connect All"** | The **Connect All** feature is used with a |

| | |
|---|---|
| | single click to connect all database connections that have this setting enabled. |

## Authentication

| Property | Description |
|---|---|
| **Save Password** | Enable this and DbVisualizer will save the password for the database connection between invocations. (The password is saved encrypted) |
| **Clear Password at Disconnect** | Enable this and the password will be cleared at disconnect |
| **Require Userid** | Ask the user to enter userid whenever the database connection is established |
| **Require Password** | Ask the user to enter password whenever the database connection is established |

## Delimited Identifiers

Delimited identifiers are identifiers which do not need to follow the rules of regular identifiers. Such identifiers can include sequence of printable characters excluding those which are not allowed to use in delimited identifiers in the actual database. Usually delimited identifiers are used when you need to use SQL reserved word, spaces and mixed case sequences as an identifier.

| Property | Description |
|---|---|
| **Begin Identifier** | Defines the start character for a delimited identifier. Normally this is a double quote (") |
| **End Identifier** | Defines the end character for a delimited identifier. Normally this is a double quote (") |
| **Scripting** | Enable this to use delimited identifiers in the Scripting features |
| **Auto Completion/Query Builder** | Enable this to use delimited identifiers in the auto completion and query builder features |

## Qualifiers

Use these settings to control whether column names should be qualified with the table name.
**Note:** Using table name aliases will override the setting of prepend column names.

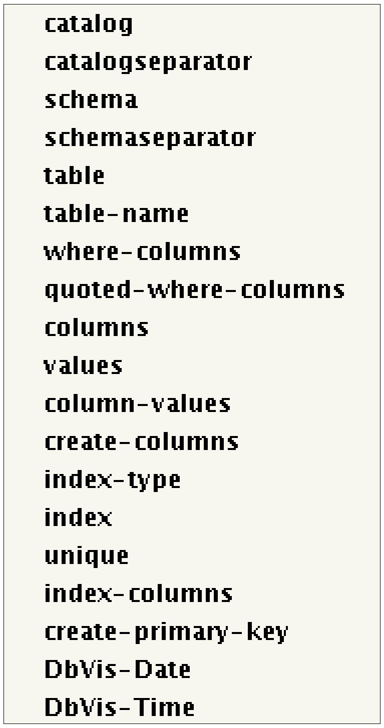
| Property | Description |
|---|---|
| **Qualify with Schema/Database: Scripting** | Enable this to qualify object names with schema/database in the Scripting features |
| **Qualify with Schema/Database: Auto Completion/Query Builder** | Enable this to qualify object names with schema/database in the auto completion and query builder features |
| **Qualify Columns: Auto Completion/Query Builder** | Enable this to qualify column names with the table name in the auto completion and query builder features |

## Transaction

| Property | Description |
|---|---|
| **Auto Commit** | Defines if each executed SQL statement will be auto committed or not. This setting applies for all SQL's that are executed in the SQL Commander. The inline and form editors in DbVisualizer Personal handles the commit and rollback management independently of the setting of Auto Commit. |
| **Transaction Isolation** | Attempts to change the transaction isolation level for all database connections.<br>**Note:** If this property is changed during a transaction, the result is JDBC driver specific. |

## SQL Statements

This category controls the SQL templates that DbVisualizer uses internally throughout the application. Each SQL template is composed of the standard SQL and variables. Variables are identified with **$$...$$**. DbVisualizer relies on a list of pre- defined variable names that are accessed in the **SQL Templates** right click menu:



catalog
catalogseparator
schema
schemaseparator
table
table-name
where-columns
quoted-where-columns
columns
values
column-values
create-columns
index-type
index
unique
index-columns
create-primary-key
DbVis-Date
DbVis-Time

**Figure: All pre-defined variables**

A specific pre-defined variable can be used in on or more of the SQL templates. Using a variable in a SQL statement that is not valid will result in the variable appearing as is once the statement is executed.

There is normally no reason to modify the SQL templates nor the variable identifier or delimiter settings. There might however be circumstances when edits are needed:

- To put quotes or brackets around table names
- To change the variable identifier or variable delimiter since the default settings may interfere with object names in the database
- To modify the appearance of the where clause or the list of columns

| Property | Name | Description |
|---|---|---|
| **SQL Templates** | **SELECT ALL** | Command used when selecting all rows for a table |
| | **SELECT ALL WHERE** | Command used when selecting some rows for a table |
| | **SELECT COUNT** | Command used to get the number of rows in a table |
| | **INSERT INTO** | Command used to insert a new row into a table |
| | **UPDATE WHERE** | Command used to update an existing row in a |

| | | |
|---|---|---|
| | | table |
| | **DELETE WHERE** | Command used to delete a specific row in a table |
| | **DROP TABLE** | Command used to drop a specific table |
| | **CREATE TABLE** | Command used to create a new table with an optional primary key |
| | **CREATE INDEX** | Command used to create an index for a specific table |
| | **Monitor Row Count** | Command used to get the number of rows in a table and the current time stamp |
| | **Monitor Row Count Change** | Command used to get the row count difference in a table compared to the previous execution. The calculated row count and the current time stamp is returned |

## Connection Hooks

Connection hooks defines optional SQL commands that are sent to the database at connect and just before disconnect. They are typically used to initialize the database session with custom settings and at disconnect clean up various resources.

| Property | Description |
|---|---|
| **Run SQL at Connect** | Defines the SQL that will be executed just after the connection has been established |
| **Run SQL at Disconnect** | Defines the SQL that will be executed just before the connection will be disconnected |

## Objects Tree

| Property | Description |
|---|---|
| **Custom Object Tree Labels** | Here you can define custom tree labels that will appear in the database objects tree. The **Object Type** must match the corresponding type in the actual database |

| | |
|---|---|
| | profile. |

## SQL Editor

| Property | Description |
|---|---|
| **Remove New Line Characters** | Specifies whether any new line characters should be removed from any SQL statement executed in the SQL Commander and in the implicit SQL execution functionality in DbVisualizer. Some drivers/databases such as DB2 requires that no new line characters are part of any executed SQL. |
| **Generate JOIN clauses in Query Builder** | Specifies whether the Query Builder will generate JOINs as JOIN clauses or WHERE conditions.<br>**JOIN clause:**<br>`SELECT     *`<br>`FROM       HR.EMPLOYEES emp`<br>`INNER JOIN HR.DEPARTMENTS dept`<br>`ON         (emp.DEPARTMENT_ID = dept.DEPARTMENT_ID)`<br>**WHERE condition:**<br>`SELECT *`<br>`FROM   HR.EMPLOYEES emp,`<br>`    HR.DEPARTMENTS dept`<br>`WHERE  (emp.DEPARTMENT_ID = dept.DEPARTMENT_ID)` |

## Database Specific settings

Some databases are supported more in DbVisualizer then others and so requires extended configuration capabilities.

### Data Types (Oracle)

In Oracle it is sometimes desired to treat DATE data types as TIMESTAMP. Enable **Handle DATE as TIMESTAMP** and DbVisualizer will automatically convert DATE's.

### Explain Plan (Oracle, SQL Server and DB2)

The explain plan feature supported for Oracle, SQL Server and DB2 can be configured to highlight certain threshold levels.

| Property | Description |
|---|---|
| **Color Critical Nodes** | Enable this and critical nodes in the explain plan feature will be highlighted. |
| **Critical Threshold** | The threshold for when a node should be handled as critical |
| **Warning Threshold** | The threshold for when a node should be handled as a warning |

### Explain Plan (Oracle)

The explain plan feature for Oracle can be configured to define the management of the underlying plan table in which the explain plan result is stored.

### Explain Plan (DB2)

The explain plan feature for DB2 can be configured to define the management of the underlying plan tables in which the explain plan result is stored.

### System Tables (Oracle)

Select here whether the database profile for Oracle should retrieve database information from the **DBA** or **ALL** system tables.
**Note:** If choosing DBA make sure the appropriate privileges are granted for the user you are connecting as.

# Export, Import and Print

## Introduction

The export feature is used to export data that has been fetched and presented in DbVisualizer. The export wizard dialog looks different depending on whether a grid, graph or chart data is going to be exported. The following sections describe the settings that can be made in each of these contexts. There are major differences between DbVisualizer Free and Personal when exporting grid data. This document explains the complete functionality in the Personal edition even though it implicitly covers the export functionality in DbVisualizer Free.

The import feature is used to import data stored in CSV (Character Separated Values) format from files.

The printing feature can be used to print grid and graph data to printer or file.

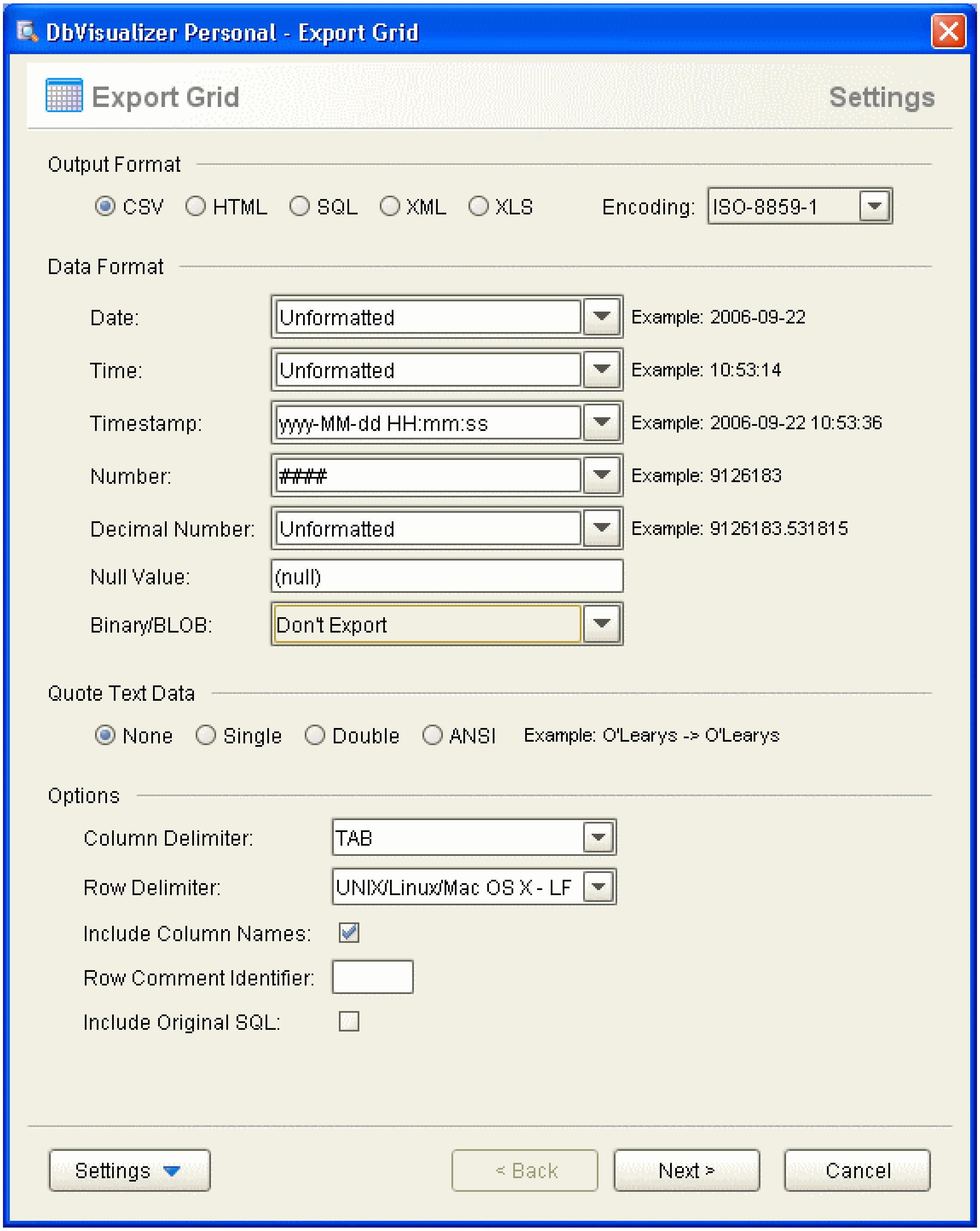
Exporting very large result sets using the standard export feature may fail with memory problems since all data must first be presented in DbVisualizer. Using the @export client side command in the SQL Commander solves this problem since the data is exported on the fly while it is fetched from the database.

## Export Grid data

The Export wizard is primarily initiated using the **File->Export** main menu choice. This operation examines the current context and displays the appropriate wizard. The **File->Export Selection** main menu choice is specifically for **Grid** contexts and is used to export the current selection instead of all data in the grid. It is only enabled if the current context is a grid and if there are any selected cells in it. In addition all grids throughout DbVisualizer offer the right click menu choice for **Export Selection.** It is a shortcut for the **File->Export Selection** main menu operation.

### Settings page

There are a number of options to configure how the data should be exported. The settings page contain general properties that control how the exported data should be formatted. All settings in the settings page can be saved to a file for later use in the export wizard or in the SQL Commander when exporting result sets using the @export editor command.



Output Format

● CSV ○ HTML ○ SQL ○ XML ○ XLS    Encoding: ISO-8859-1

Data Format

| | | |
|---|---|---|
| Date: | Unformatted | Example: 2006-09-22 |
| Time: | Unformatted | Example: 10:53:14 |
| Timestamp: | yyyy-MM-dd HH:mm:ss | Example: 2006-09-22 10:53:36 |
| Number: | #### | Example: 9126183 |
| Decimal Number: | Unformatted | Example: 9126183.531815 |
| Null Value: | (null) | |
| Binary/BLOB: | Don't Export | |

Quote Text Data

● None ○ Single ○ Double ○ ANSI    Example: O'Learys -> O'Learys

Options

| | |
|---|---|
| Column Delimiter: | TAB |
| Row Delimiter: | UNIX/Linux/Mac OS X - LF |
| Include Column Names: | ☑ |
| Row Comment Identifier: | |
| Include Original SQL: | □ |

**Figure: The grid export wizard**

Read the sections below for detailed information on each field and what settings that can be made.

### Output Format

Grid data can be exported in the following formats.

| Format | Description |
|---|---|
| **CSV** | The CSV format (Character Separated Values) is used to export the grid of data to a file in which each column is separated with a character or several. It is even possible to specify the row delimiter (aka newline sequence of characters).<br>`5,Hepp,59248`<br>`15,Hopp,41993`<br>`16,Hupp,44115`<br>The above example use a "," as the column delimiter and a "\n" sequence as the row delimiter (invisible above). |
| **HTML** | The data is exported in HTML format using the <TABLE> and associated tags. |
| **SQL** | The SQL format simply creates an SQL INSERT statement for each row in the grid. It also uses the column names from the grid to define the column list in the SQL statement.<br>`insert into table1 (Column1, Column2, Column3) values (5, 'Hepp', 59248);`<br>`insert into table1 (Column1, Column2, Column3) values (15, 'Hopp', 41993);`<br>`insert into table1 (Column1, Column2, Column3) values (16, 'Hupp', 44115);` |
| **XML** | The XML format is handy when importing or using the exported data in an XML enabled application. The structure of the XML format is:<br>`<ROWSET>`<br>`  <ROW>`<br>`    <Column1>5</Column1>`<br>`    <Column2>Hepp</Column2>`<br>`    <Column3>59248</Column3>`<br>`  </ROW>`<br>`  <ROW>`<br>`    <Column1>15</Column1>`<br>`    <Column2>Hopp</Column2>`<br>`    <Column3>41993</Column3>`<br>`  </ROW>`<br>`  <ROW>`<br>`    <Column1>15</Column1>` |

```
    <Column2>Hupp</Column2>
    <Column3>44115</Column3>
  </ROW>
</ROWSET>
```

### Encoding

The encoding choice controls what encoding the data will be exported in. This will also set the encoding in the HTML and XML headers. The default choice is based on your systems default encoding.

### Data Format

The data format settings defines how the data for each of the data types will be formatted.

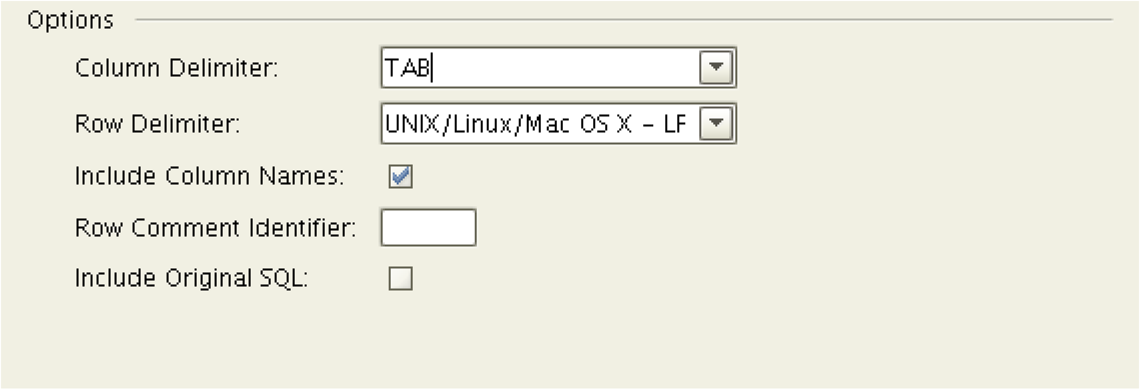
### Quote Text Data

Defines if text data should appear between quotes or not. Selecting the ANSI choice will automatically prefix any single quotes with an additional one.

### Options

The options section is used to define settings that are specific for the selected output format.

#### CSV



Options

Column Delimiter: TAB

Row Delimiter: UNIX/Linux/Mac OS X - LF

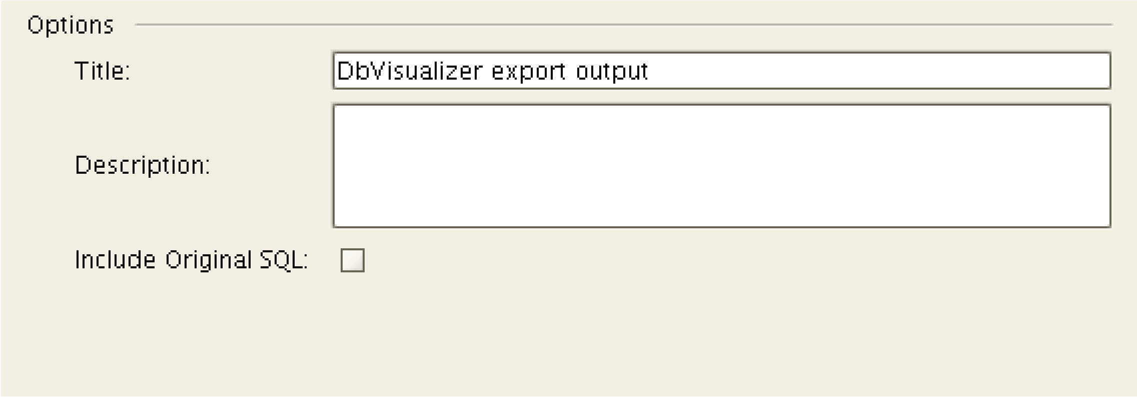
Include Column Names: ☑

Row Comment Identifier:

Include Original SQL: □

**Figure: CSV specific export options**

### HTML


Options

Title: DbVisualizer export output

Description:

Include Original SQL: □

**Figure: HTML specific export options**

### SQL

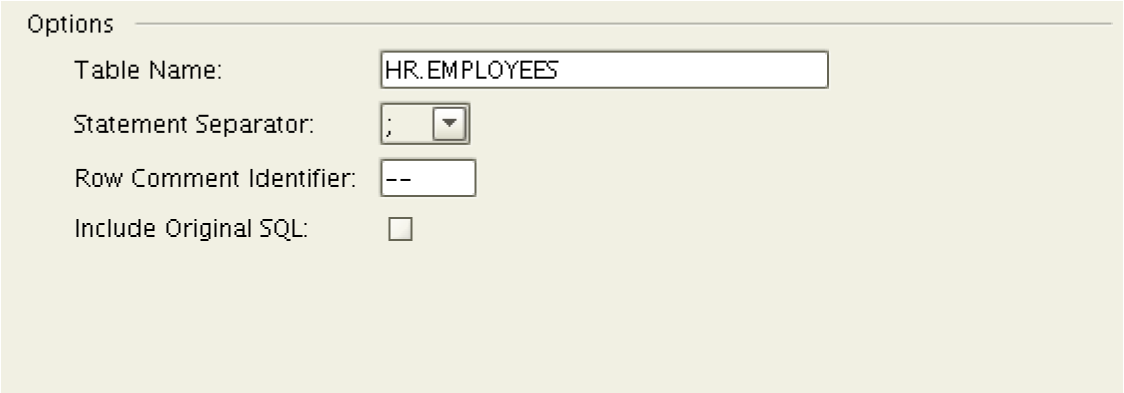

Options

Table Name: HR.EMPLOYEES

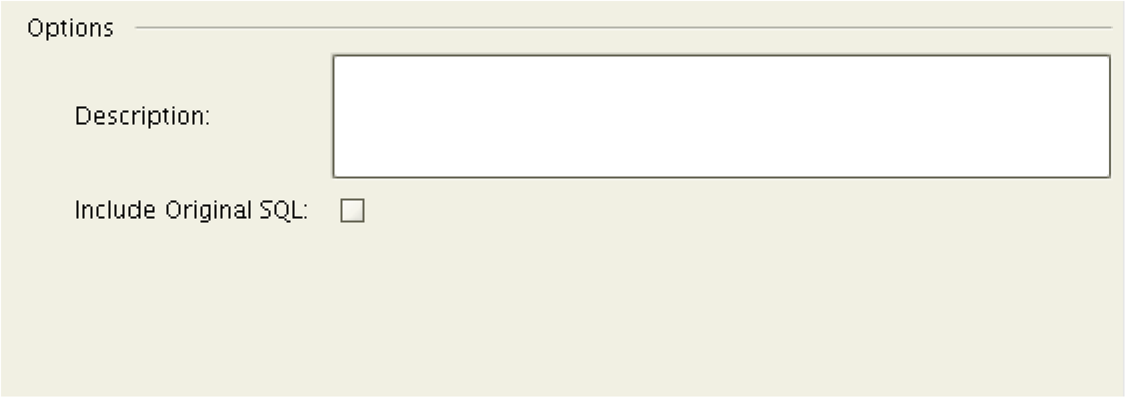
Statement Separator: ;

Row Comment Identifier: --

Include Original SQL: □

**Figure: SQL specific export options**

### XML


Options

Description:

Include Original SQL: □

**Figure: XML specific export options**

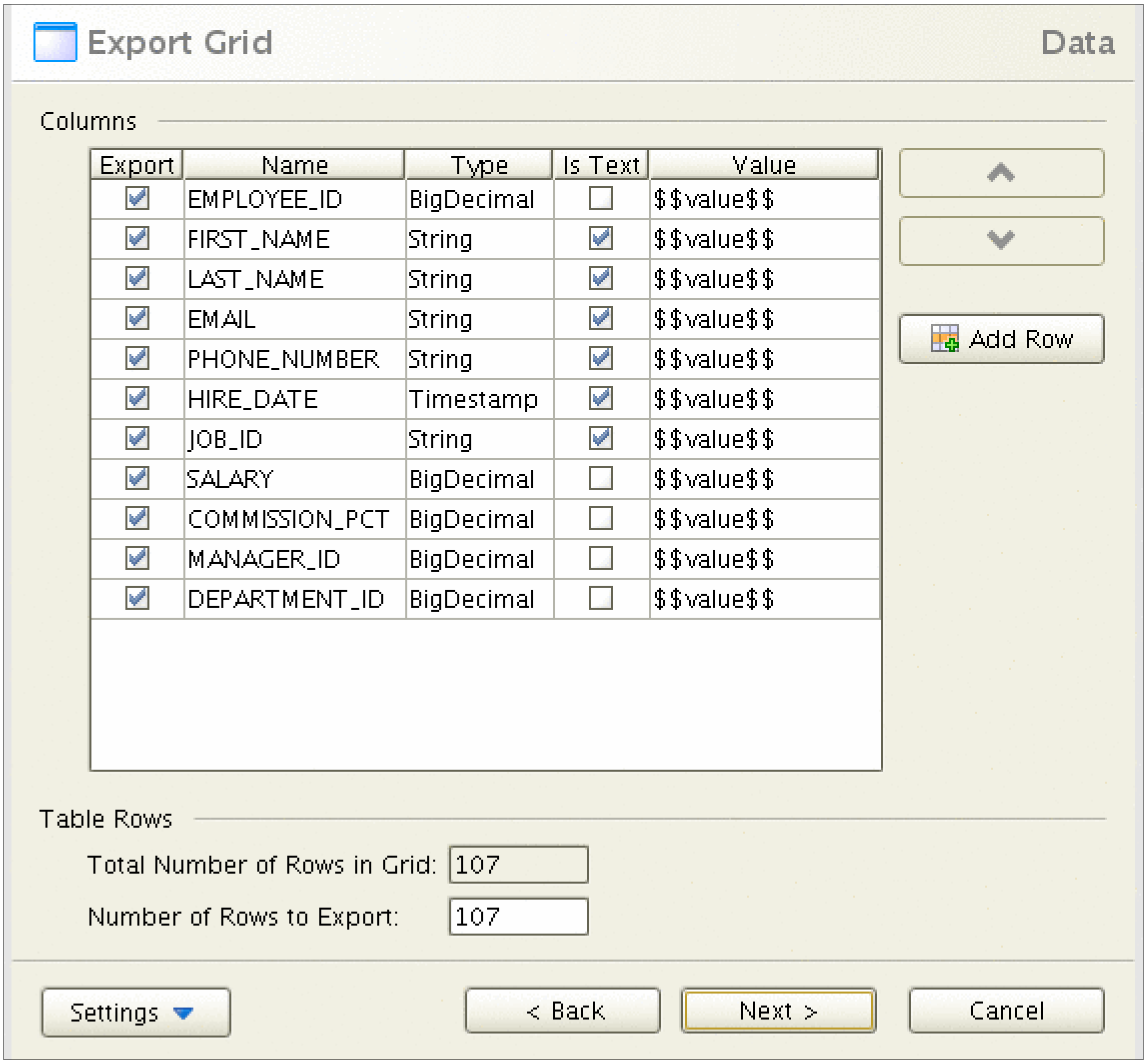
## Settings

The Settings button lists when pressed a menu with options to save and load settings to and from a file.

- **Use Default Settings**
  Press this button to initiate the settings with default values. Some of the settings will be fetched from the general tool properties dialog.
- **Load**
  Press this button to open the file choose 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 @export editor commands.

## Data page

The columns list is used to control what columns will be exported and the format of their data. 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 export then it will not appear in this list.



| Export | Name | Type | Is Text | Value |
|---|---|---|---|---|
| ☑ | EMPLOYEE_ID | BigDecimal | □ | $$value$$ |
| ☑ | FIRST_NAME | String | ☑ | $$value$$ |
| ☑ | LAST_NAME | String | ☑ | $$value$$ |
| ☑ | EMAIL | String | ☑ | $$value$$ |
| ☑ | PHONE_NUMBER | String | ☑ | $$value$$ |
| ☑ | HIRE_DATE | Timestamp | ☑ | $$value$$ |
| ☑ | JOB_ID | String | ☑ | $$value$$ |
| ☑ | SALARY | BigDecimal | □ | $$value$$ |
| ☑ | COMMISSION_PCT | BigDecimal | □ | $$value$$ |
| ☑ | MANAGER_ID | BigDecimal | □ | $$value$$ |
| ☑ | DEPARTMENT_ID | BigDecimal | □ | $$value$$ |

Total Number of Rows in Grid: 107

Number of Rows to Export: 107

**Figure: The grid export wizard**

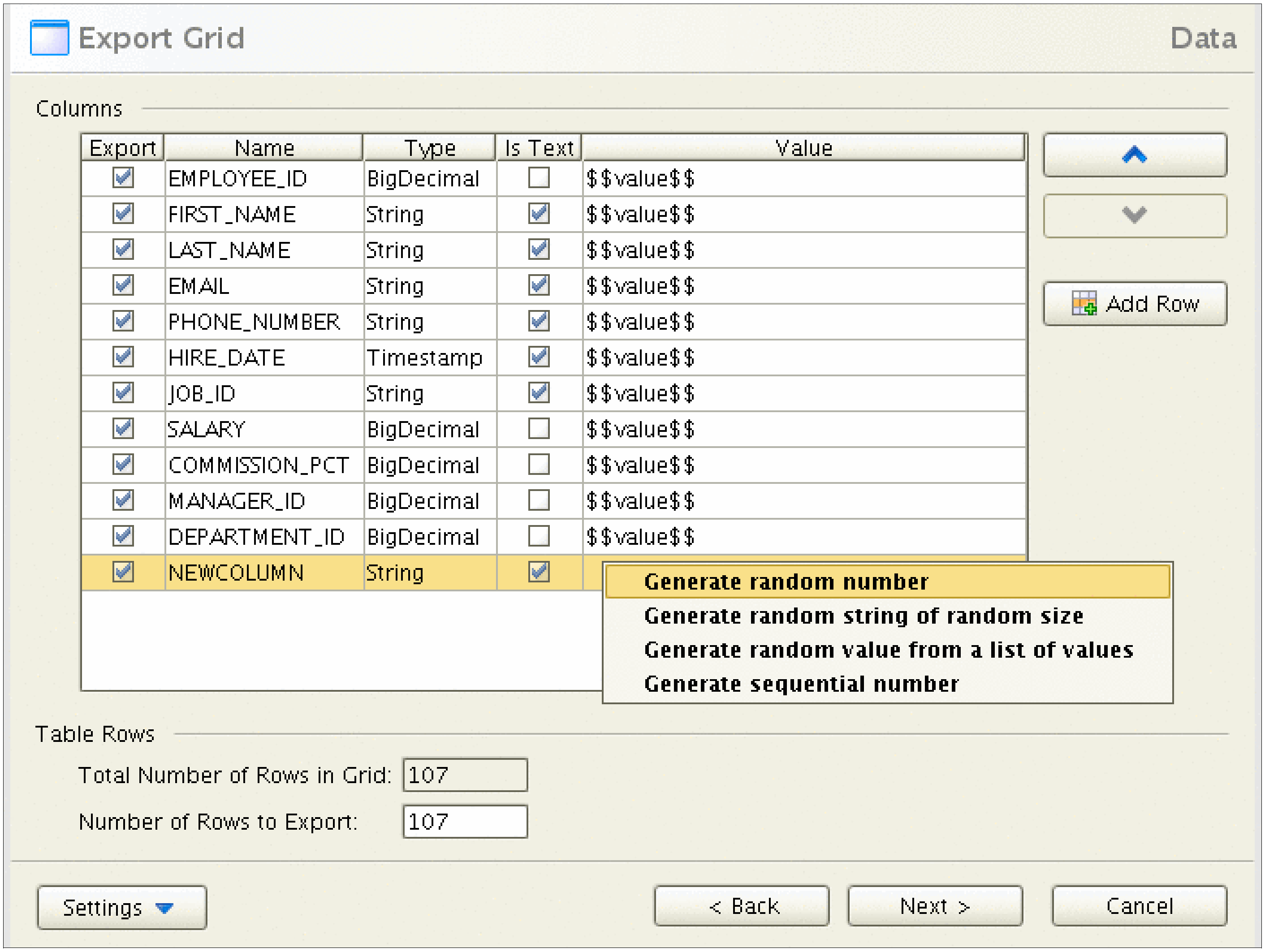
The **Table Rows** fields tells how many rows that are available and the choice to optionally specify the number of rows to export. This setting along with the **Add Row** button is especially useful if using the test data generation feature described in the next section.

Here follows information about the columns in the list.

| Field | Description |
|---|---|
| **Export** | Defines whether the column will be exported or not. Uncheck it to ignore the column in the exported output. |
| **Name** | The name of the column. This is only used if exporting in HTML, XML or SQL format. Column headers are optional in the CSV output format. |
| **Type** | The internal DbVisualizer type for the column. This type is used to determine if the column is a text column (i.e if the data will be surrounded in 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 will be enclosed in quotes. |
| **Value** | The default "`$$value$$`" variable will simply be substituted by the actual value in the exported output. You can enter additional static text in the value field. This is also the place where any test data generators are defined. |

### Generate Test Data

The test data generator is useful when you need to add random column data to the exported output. The actual value of the data that is going to be in the exported output is referenced by the `$$value$$` variable in the Value field. This variable is simply replaced by the real value during the export process. Additional static data can be added before and after the `$$value$$` field and will be exported as entered. The value field is also the place to setup any test data generators. While in the editing mode of the value field there is a right click menu with the supported generator functions.



Export Grid
Data
Add Row
Generate random number
Generate random string of random size
Generate random value from a list of values
Generate sequential number
Settings
< Back
Next >
Cancel

Columns

| Export | Name | Type | Is Text | Value |
|---|---|---|---|---|
| ☑ | EMPLOYEE_ID | BigDecimal | ☐ | $$value$$ |
| ☑ | FIRST_NAME | String | ☑ | $$value$$ |
| ☑ | LAST_NAME | String | ☑ | $$value$$ |
| ☑ | EMAIL | String | ☑ | $$value$$ |
| ☑ | PHONE_NUMBER | String | ☑ | $$value$$ |
| ☑ | HIRE_DATE | Timestamp | ☑ | $$value$$ |
| ☑ | JOB_ID | String | ☑ | $$value$$ |
| ☑ | SALARY | BigDecimal | ☐ | $$value$$ |
| ☑ | COMMISSION_PCT | BigDecimal | ☐ | $$value$$ |
| ☑ | MANAGER_ID | BigDecimal | ☐ | $$value$$ |
| ☑ | DEPARTMENT_ID | BigDecimal | ☐ | $$value$$ |
| ☑ | NEWCOLUMN | String | ☑ | |

Table Rows

Total Number of Rows in Grid: 107

Number of Rows to Export: 107

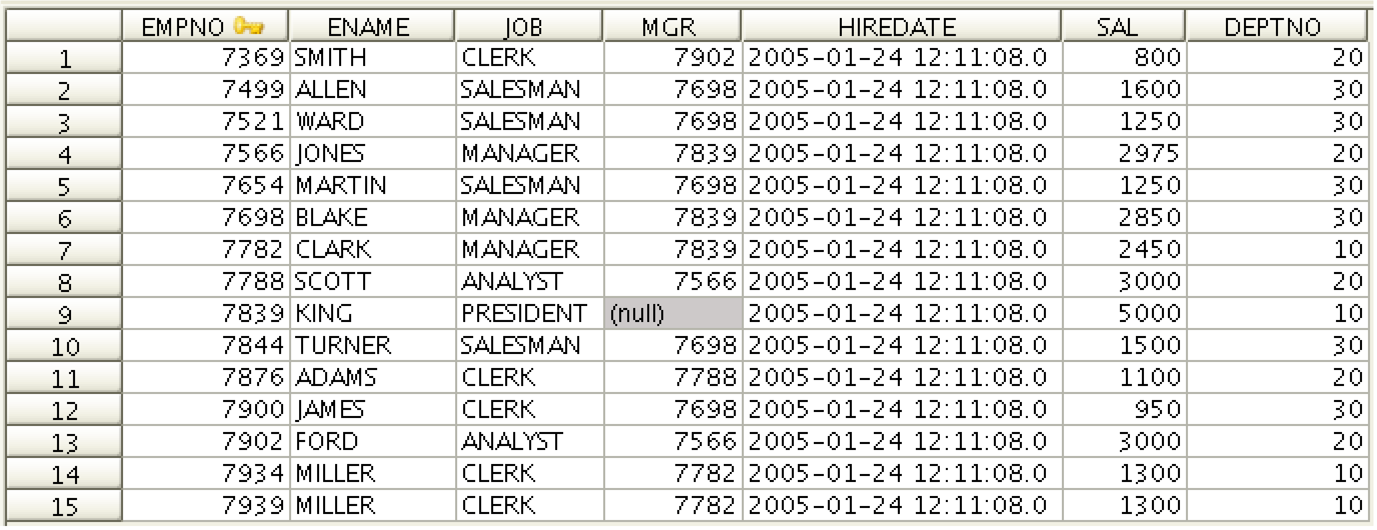
**Figure: Right click menu with the test data generator functions**

| 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 |
| **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. |

**Test data generator example**

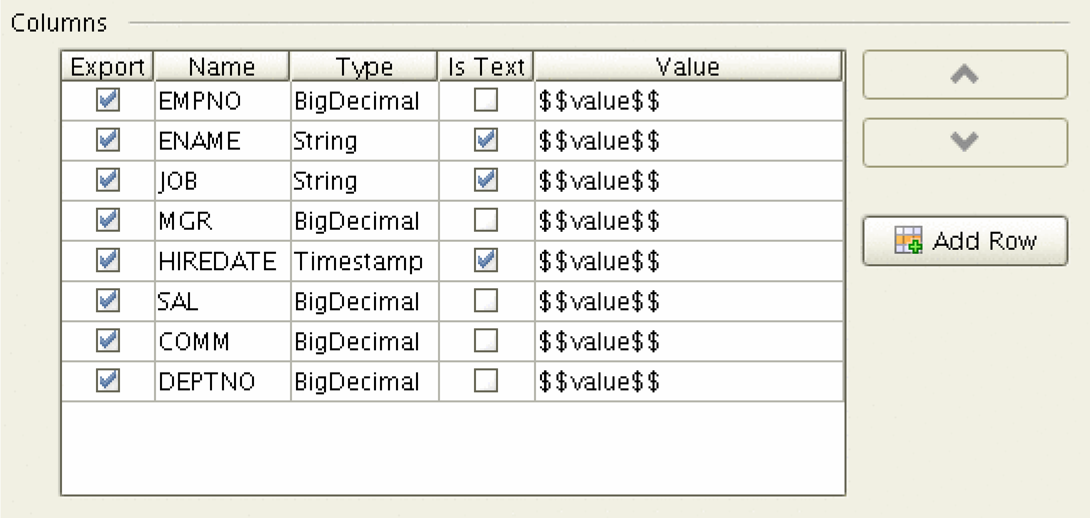
Here follows an example that utilizes the test data generators. Consider this data:



| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | DEPTNO |
|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 2005-01-24 12:11:08.0 | 800 | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 2005-01-24 12:11:08.0 | 1600 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 2005-01-24 12:11:08.0 | 1250 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 2005-01-24 12:11:08.0 | 2975 | 20 |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 2005-01-24 12:11:08.0 | 1250 | 30 |
| 6 | 7698 | BLAKE | MANAGER | 7839 | 2005-01-24 12:11:08.0 | 2850 | 30 |
| 7 | 7782 | CLARK | MANAGER | 7839 | 2005-01-24 12:11:08.0 | 2450 | 10 |
| 8 | 7788 | SCOTT | ANALYST | 7566 | 2005-01-24 12:11:08.0 | 3000 | 20 |
| 9 | 7839 | KING | PRESIDENT | (null) | 2005-01-24 12:11:08.0 | 5000 | 10 |
| 10 | 7844 | TURNER | SALESMAN | 7698 | 2005-01-24 12:11:08.0 | 1500 | 30 |
| 11 | 7876 | ADAMS | CLERK | 7788 | 2005-01-24 12:11:08.0 | 1100 | 20 |
| 12 | 7900 | JAMES | CLERK | 7698 | 2005-01-24 12:11:08.0 | 950 | 30 |
| 13 | 7902 | FORD | ANALYST | 7566 | 2005-01-24 12:11:08.0 | 3000 | 20 |
| 14 | 7934 | MILLER | CLERK | 7782 | 2005-01-24 12:11:08.0 | 1300 | 10 |
| 15 | 7939 | MILLER | CLERK | 7782 | 2005-01-24 12:11:08.0 | 1300 | 10 |

**Figure: Sample of grid data**

The Data page will look like this based on exporting the previous grid.

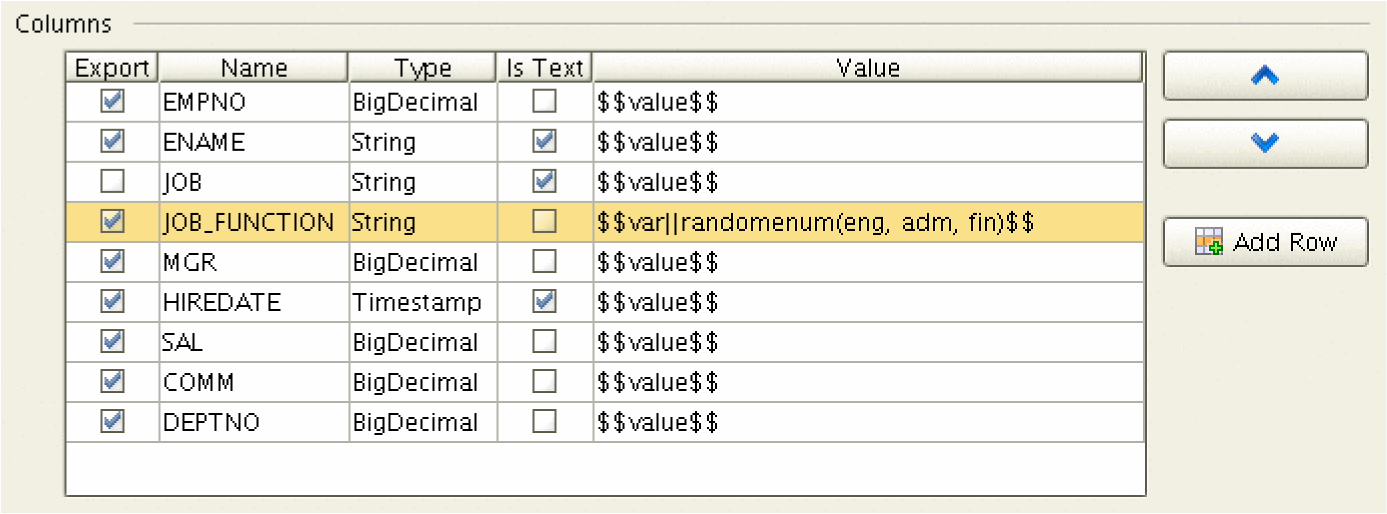


Columns

| Export | Name | Type | Is Text | Value |
|---|---|---|---|---|
| ☑ | EMPNO | BigDecimal | □ | $$value$$ |
| ☑ | ENAME | String | ☑ | $$value$$ |
| ☑ | JOB | String | ☑ | $$value$$ |
| ☑ | MGR | BigDecimal | □ | $$value$$ |
| ☑ | HIREDATE | Timestamp | ☑ | $$value$$ |
| ☑ | SAL | BigDecimal | □ | $$value$$ |
| ☑ | COMM | BigDecimal | □ | $$value$$ |
| ☑ | DEPTNO | BigDecimal | □ | $$value$$ |

**Figure: The export window**

The **JOB** column should not appear in the output and the new **JOB_FUNCTION** should contain abbreviated job functions. To accomplish this we simply uncheck the **Export** field for **JOB** entry. The Value for the **JOB_FUNCTION** is set to use the **Generate random value from a list of values** function.



| Export | Name | Type | Is Text | Value |
|---|---|---|---|---|
| ☑ | EMPNO | BigDecimal | ☐ | $$value$$ |
| ☑ | ENAME | String | ☑ | $$value$$ |
| ☐ | JOB | String | ☑ | $$value$$ |
| ☑ | JOB_FUNCTION | String | ☐ | $$var\|\|randomenum(eng, adm, fin)$$ |
| ☑ | MGR | BigDecimal | ☐ | $$value$$ |
| ☑ | HIREDATE | Timestamp | ☑ | $$value$$ |
| ☑ | SAL | BigDecimal | ☐ | $$value$$ |
| ☑ | COMM | BigDecimal | ☐ | $$value$$ |
| ☑ | DEPTNO | BigDecimal | ☐ | $$value$$ |

**Figure: Customized columns list with a generator function**

Previewing the data (or exporting it) in CSV format results in this:

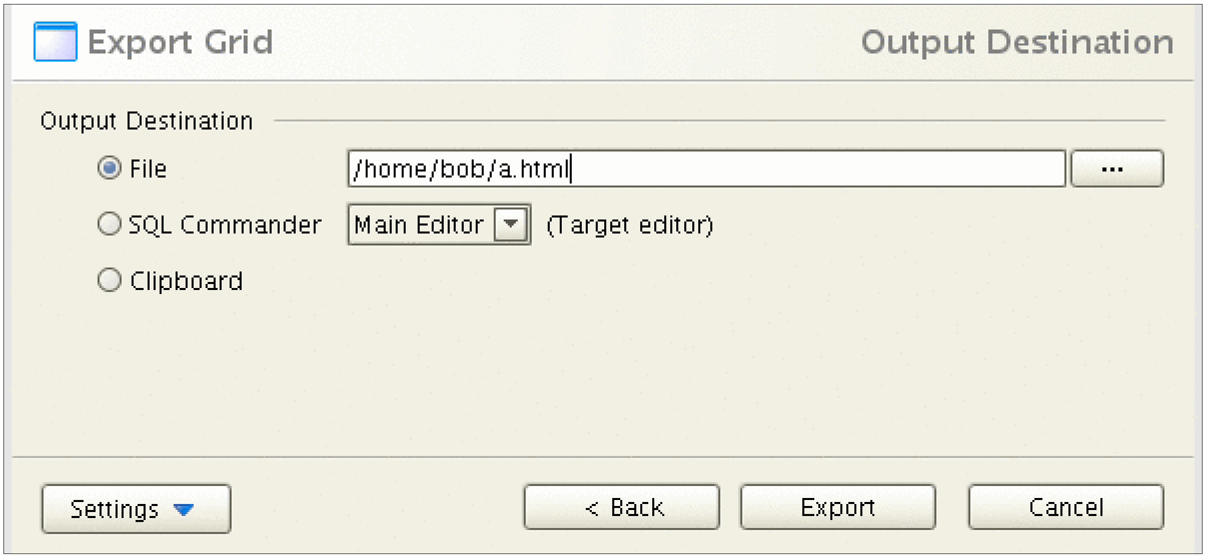
```
EMPNO, ENAME, JOB_FUNCTION, MGR, HIREDATE, SAL, COMM, DEPTNO
7369, SMITH, adm, 7902, 2005-01-24 12:11:08, 800, (null), 20
7499, ALLEN, adm, 7698, 2005-01-24 12:11:08, 1600, 300, 30
7521, WARD, eng, 7698, 2005-01-24 12:11:08, 1250, 500, 30
7566, JONES, adm, 7839, 2005-01-24 12:11:08, 2975, (null), 20
7654, MARTIN, eng, 7698, 2005-01-24 12:11:08, 1250, 1400, 30
7698, BLAKE, eng, 7839, 2005-01-24 12:11:08, 2850, (null), 30
7782, CLARK, eng, 7839, 2005-01-24 12:11:08, 2450, (null), 10
7788, SCOTT, eng, 7566, 2005-01-24 12:11:08, 3000, (null), 20
7839, KING, eng, (null), 2005-01-24 12:11:08, 5000, (null), 10
7844, TURNER, eng, 7698, 2005-01-24 12:11:08, 1500, 0, 30
7876, ADAMS, fin, 7788, 2005-01-24 12:11:08, 1100, (null), 20
7900, JAMES, eng, 7698, 2005-01-24 12:11:08, 950, (null), 30
7902, FORD, eng, 7566, 2005-01-24 12:11:08, 3000, (null), 20
7934, MILLER, fin, 7782, 2005-01-24 12:11:08, 1300, (null), 10
7939, MILLER, fin, 7782, 2005-01-24 12:11:08, 1300, (null), 10
...
```

## Preview

The preview page shows 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 then just use the back button to modify the settings.

## Output Destination

The destination field specifies the target destination for the exported data.



Output Destination

● File /home/bob/a.html

○ SQL Commander Main Editor (Target editor)

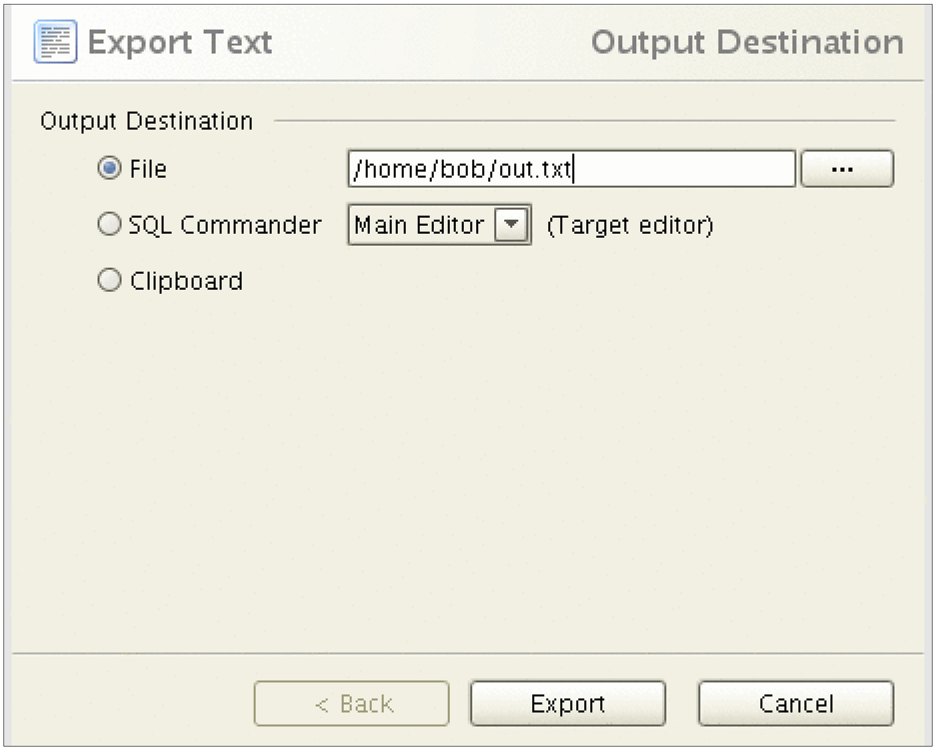
○ Clipboard

**Figure: The output destination and final page for grid export**

| Destination | Description |
|---|---|
| **File** | This option outputs the data to a named file. |
| **SQL Commander** | This destination will transfers the export data to the SQL Commander editor. It is primarily useful when exporting the SQL output format. |
| **Clipboard** | Export to the (system) clipboard is convenient if you want to use the exported data in another application without the extra step of exporting to file first. Data can even be pasted into a spreadsheet application such as Excel or StarOffice and the cells in the grid will appear as cells in the spreadsheet. Read more about the CSV format in the Format section. |

# Export Text data

The wizard when exporting result sets in **Text** format is very simple as it is only possible to specify where the exported output should go.



Output Destination

● File /home/bob/out.txt

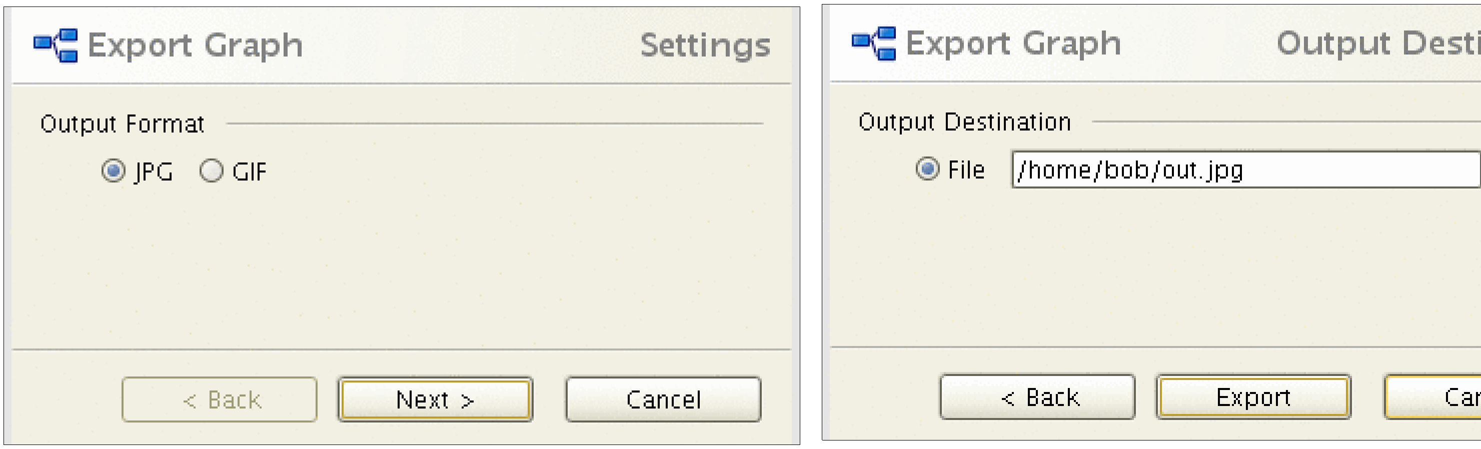
○ SQL Commander Main Editor (Target editor)

○ Clipboard

**Figure: Export window for text format result sets**

## Export Graph data

Exporting references graphs will export the graph in the same zoom level as it appears on the screen. The export window when exporting graphs looks like this:



Output Format

● JPG ○ GIF

Output Destination

● File /home/bob/out.jpg

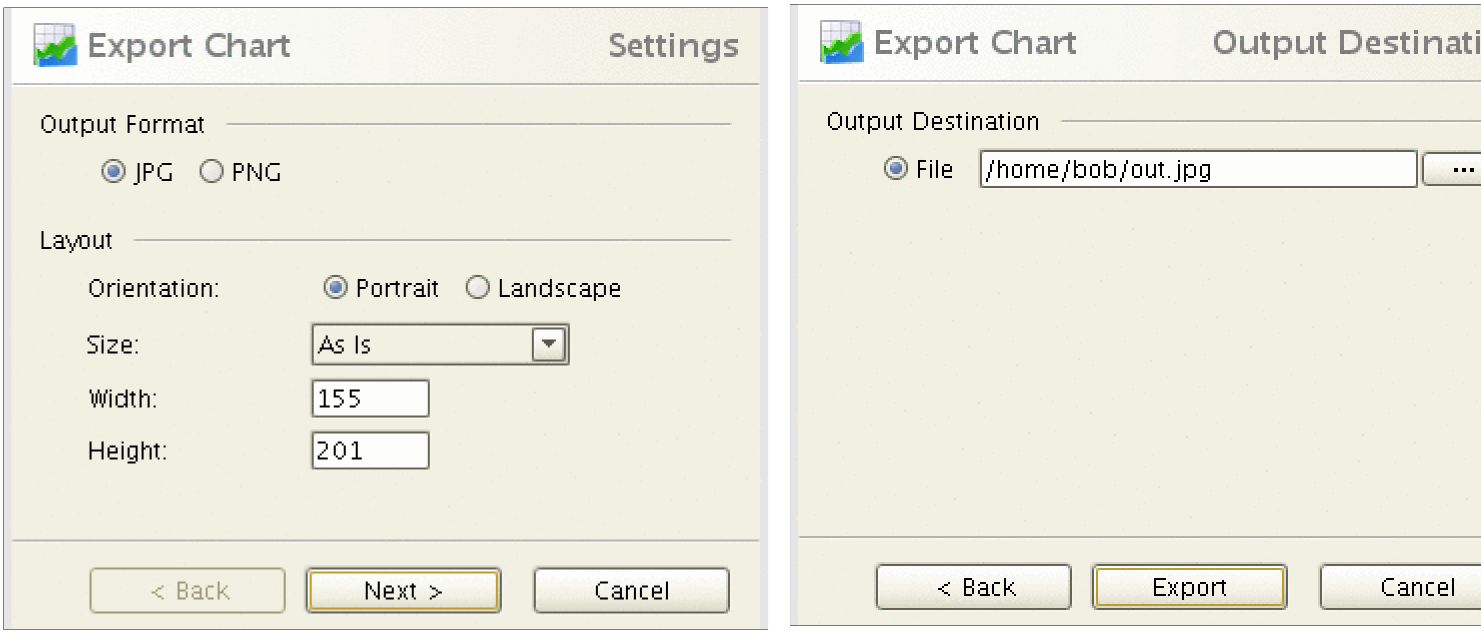
**Figure: Export window for graphs**

The export window is quite limited as compared to when exporting grids. The graph can only be exported to a **File** in the **JPEG** or **GIF** formats.

Export of graphs cannot be previewed or exported to any other destination then file.

# Export Chart data

Exporting charts adds the capabilities to set the size and orientation of the exported file.



Output Format: ● JPG ○ PNG

Orientation: ● Portrait ○ Landscape

Size: As Is

Width: 155

Height: 201

Output Destination: ● File /home/bob/out.jpg

**Figure: Export window for charts**

A chart can only be exported to a **File** in the **JPEG** and **PNG** formats. The optional **Layout** settings are used to control the size of the image. The initial width and height are the same as the size of the chart as it appear on the screen. The **Size** list when clicked shows a list of well known paper formats. The **Width** and **Height** will be changed to match the selected size. Setting the width and height or selecting a pre- defined size will scale the exported image accordingly.

Export of charts cannot be previewed or be exported to any other destination then file.

# Import Table Data

The import table data feature is used to import files whose data is organized as columns with separator characters between them. The destination for the imported data can be to a database table or a grid in DbVisualizer. The grid option is convenient for smaller files as the grid functionality then can be used to do various things with the data. An example is that a CSV file rather easily can be converted into an XML file or a HTML document by using the data import feature to grid and then use the export functionality in the grid to output the grid in the desired format.

**The import wizard is launched via the right click menu for table objects or via the Actions menu.**

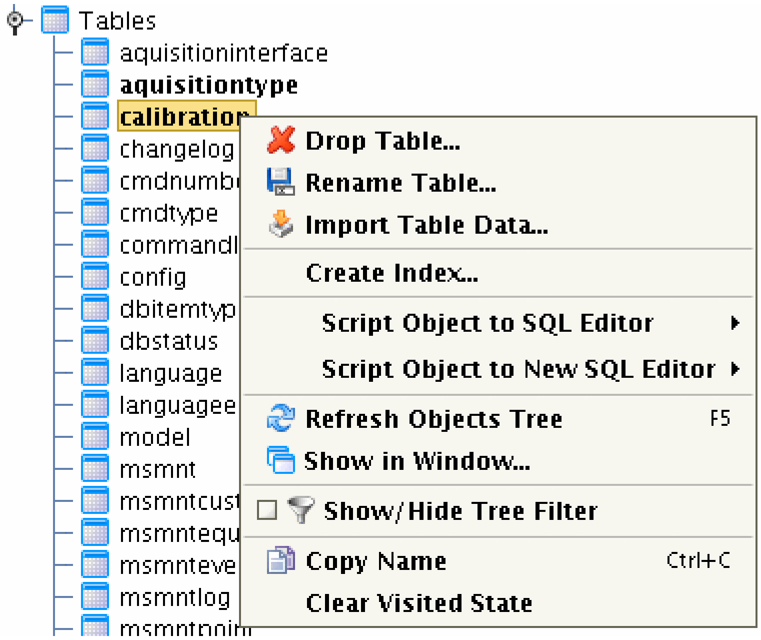
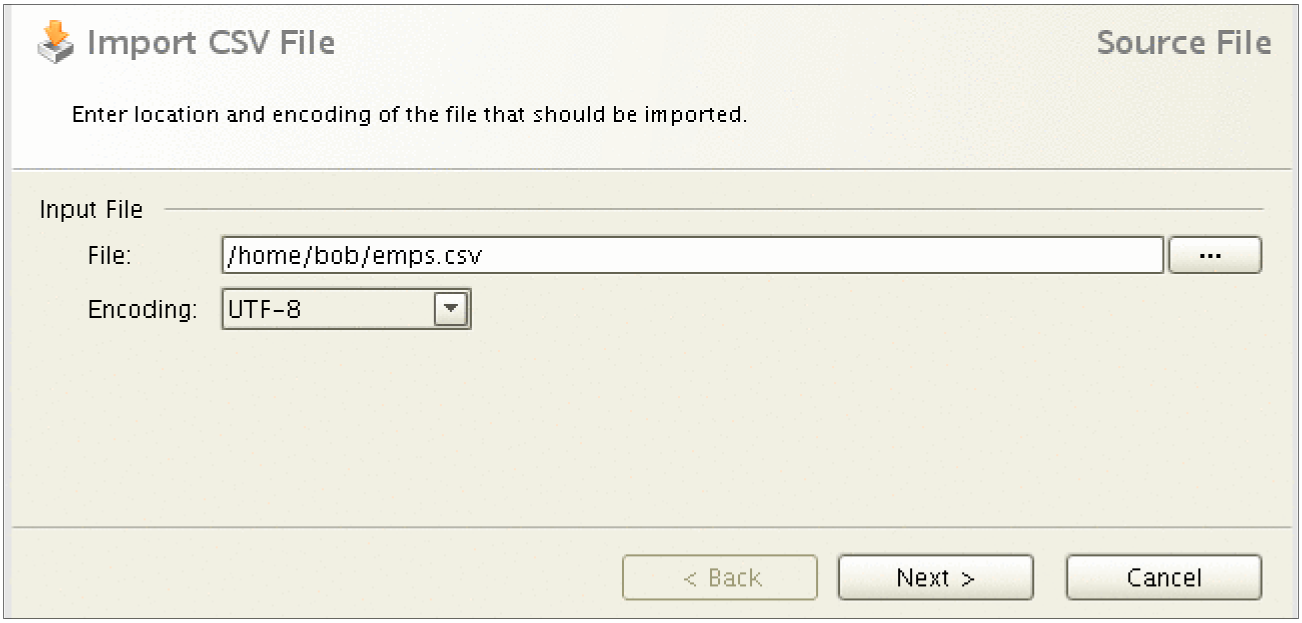


**Figure: Import Table Data action in the right click menu for table objects**

**Note 1:** The first row in the source file is used to find out the actual columns.
**Note 2:** The import wizard can not be used to import binary data.

## Source File

In the first wizard page select the source file to import.



Enter location and encoding of the file that should be imported.

File: /home/bob/emps.csv

Encoding: UTF-8

**Figure: The Source File import wizard page**

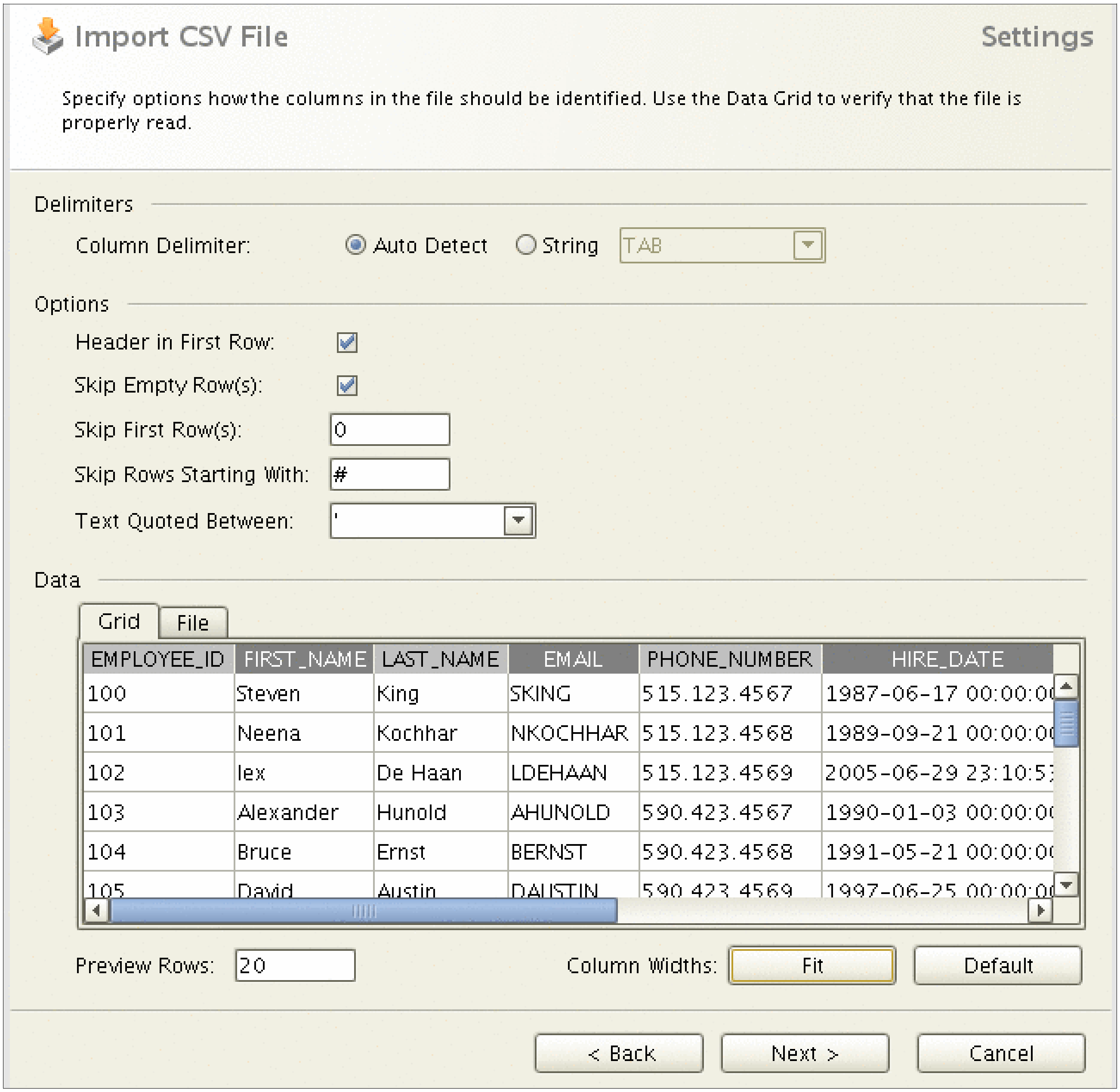
## Settings

In the settings page you specify options how the data in the file is organized. The **Data** section at the bottom of the page lists a preview of the parsed data in the **Grid** tab while the **File** tab shows the original source file. If a row in the Grid tab is red then it indicates that the row will be ignored during the import process. This happens if setting any of the **Options** that will result in rows not being qualified.

In the Delimiters section define what character that separates the columns in the file. If enabling the **Auto Detect** choice then DbVisualizer will try the following characters:

- comma ","
- tab "TAB"
- semicolon ";"
- percent "%"

Use the Options section to further define how the data should be read.



Import CSV File

Settings

Specify options how the columns in the file should be identified. Use the Data Grid to verify that the file is properly read.

Delimiters

Column Delimiter: ● Auto Detect ○ String TAB

Options

Header in First Row: ☑

Skip Empty Row(s): ☑

Skip First Row(s): 0

Skip Rows Starting With: #

Text Quoted Between: '

Data

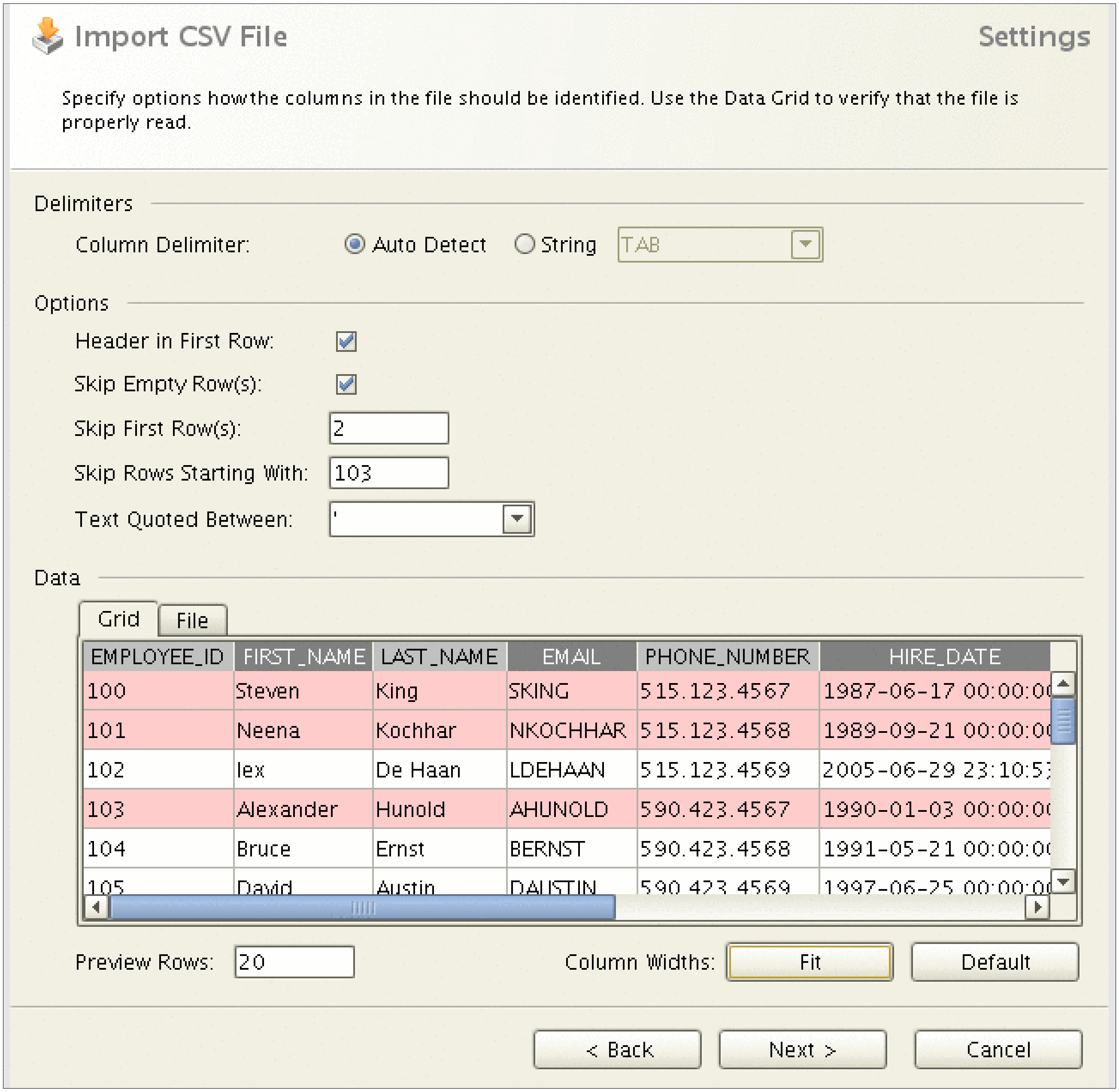
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE |
|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 1987-06-17 00:00:0 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 1989-09-21 00:00:0 |
| 102 | lex | De Haan | LDEHAAN | 515.123.4569 | 2005-06-29 23:10:5 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 1990-01-03 00:00:0 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 1991-05-21 00:00:0 |
| 105 | David | Austin | DAUSTIN | 590.423.4569 | 1997-06-25 00:00:0 |

Preview Rows: 20

Column Widths:

**Figure: The Settings wizard page**

The following shows the preview grid with some rows red. The reason is that the **Skip First Row(s)** and **Skip Rows Starting With** is set i.e the first two rows and the rows starting with 103 will not be imported.



Import CSV File

Settings

Specify options how the columns in the file should be identified. Use the Data Grid to verify that the file is properly read.

Delimiters

Column Delimiter: ● Auto Detect ○ String TAB

Options

Header in First Row: ☑

Skip Empty Row(s): ☑

Skip First Row(s): 2

Skip Rows Starting With: 103

Text Quoted Between: '

Data

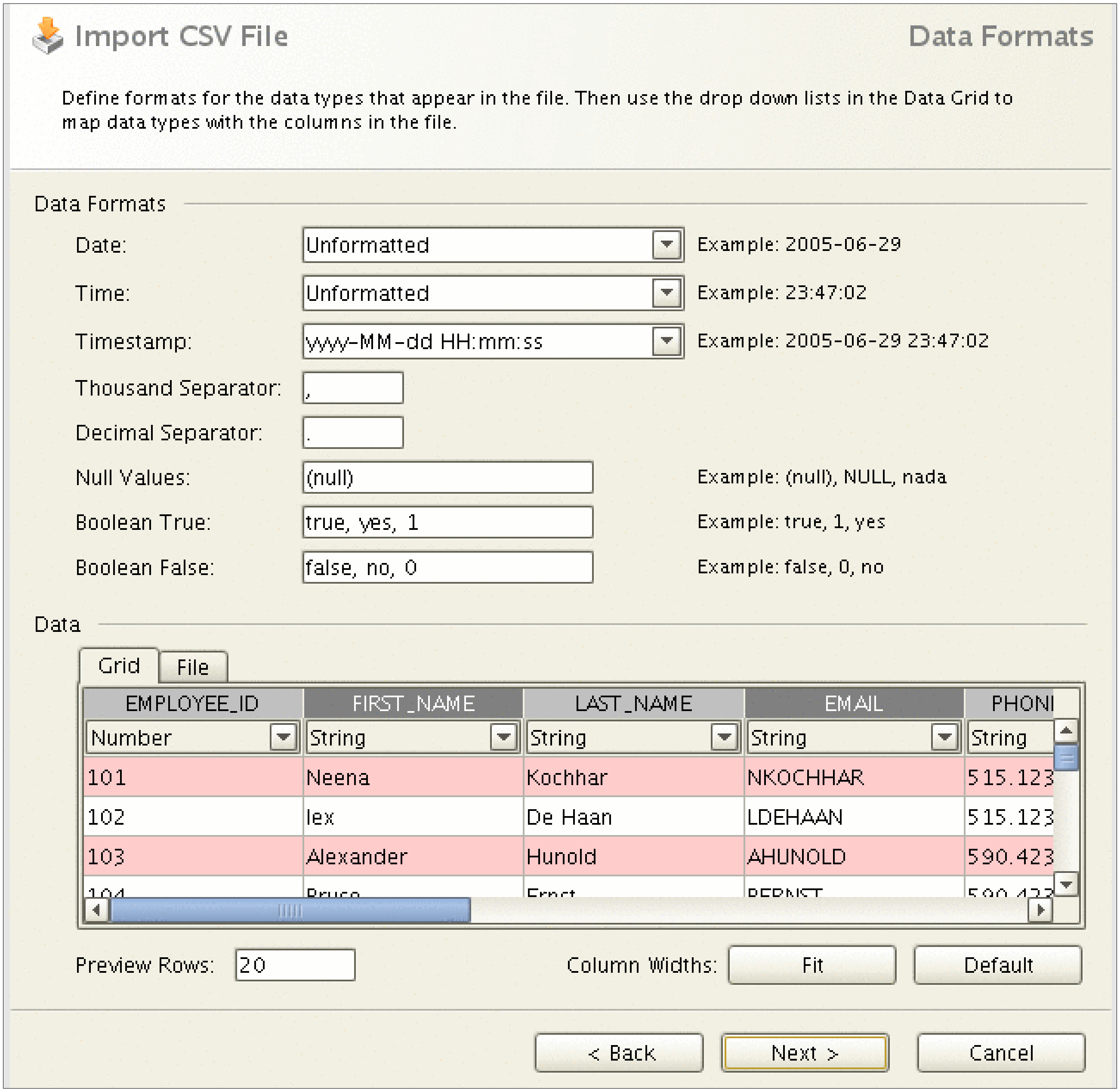
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE |
|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 515.123.4567 | 1987-06-17 00:00:0 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 1989-09-21 00:00:0 |
| 102 | lex | De Haan | LDEHAAN | 515.123.4569 | 2005-06-29 23:10:5 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 1990-01-03 00:00:0 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 1991-05-21 00:00:0 |
| 105 | David | Austin | DAUSTIN | 590.423.4569 | 1997-06-25 00:00:0 |

Preview Rows: 20

**Figure: The Settings wizard page**

## Data Formats

The Data Formats page is used to define formats for some data types. The first row in the preview grid here contains a drop down box which lists the actual data types. Just select the appropriate type for the column.



## Import CSV File

Data Formats

Define formats for the data types that appear in the file. Then use the drop down lists in the Data Grid to map data types with the columns in the file.

Data Formats

| | | |
|---|---|---|
| Date: | Unformatted | Example: 2005-06-29 |
| Time: | Unformatted | Example: 23:47:02 |
| Timestamp: | yyyy-MM-dd HH:mm:ss | Example: 2005-06-29 23:47:02 |
| Thousand Separator: | , | |
| Decimal Separator: | . | |
| Null Values: | (null) | Example: (null), NULL, nada |
| Boolean True: | true, yes, 1 | Example: true, 1, yes |
| Boolean False: | false, no, 0 | Example: false, 0, no |

Data

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONI |
|---|---|---|---|---|
| Number | String | String | String | String |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123 |
| 102 | lex | De Haan | LDEHAAN | 515.123 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423 |
| 104 | Bruce | Ernst | BERNST | 590.423 |

Preview Rows: 20

**Figure: The Settings wizard page**

The following is displayed when selecting the drop down box in the preview grid.

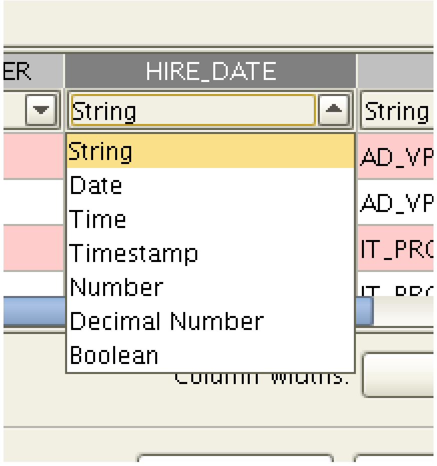
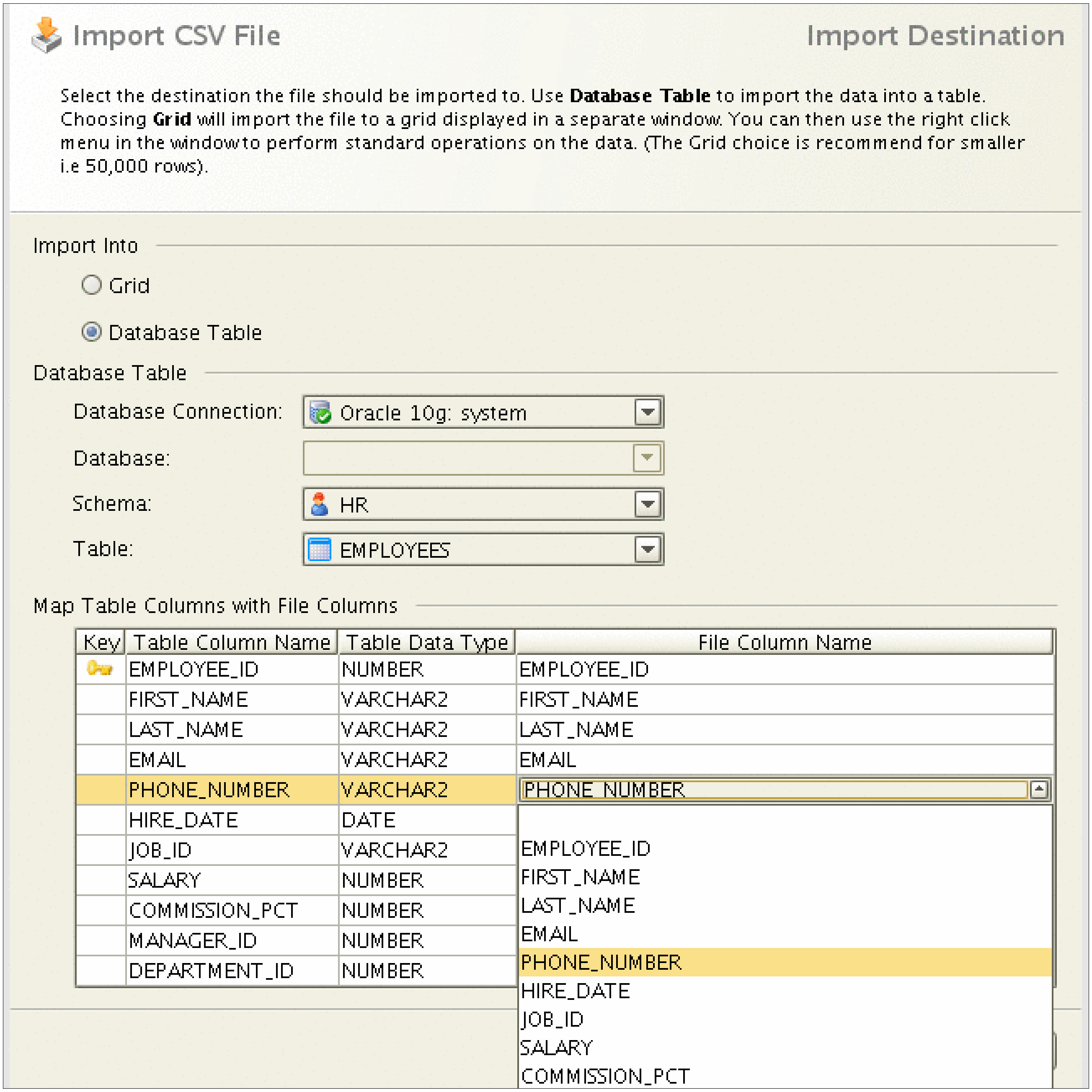


**Figure: The data type drop down**

## Import Destination

The import destination page initially shows two options, **Grid** and **Database Table**. The Grid choice is used to import the data into a grid that will be presented in its own window in DbVisualizer. When the Database Table choice is selected it will show information about the table in which the data will be imported. The Map Table Columns with File Columns grid will show what columns are in the selected database table and a column with the columns in the source file. You can here select what fields in the source file should be imported into what columns.

DbVisualizer automatically assigns the columns in the source file with the first columns in the target table. You can then manually assign them. Choose the empty choice in the columns drop down to ignore the column during import.



Import CSV File

Import Destination

Select the destination the file should be imported to. Use **Database Table** to import the data into a table. Choosing **Grid** will import the file to a grid displayed in a separate window. You can then use the right click menu in the window to perform standard operations on the data. (The Grid choice is recommend for smaller i.e 50,000 rows).

Import Into

○ Grid

● Database Table

Database Table

Database Connection: Oracle 10g: system

Database:

Schema: HR

Table: EMPLOYEES

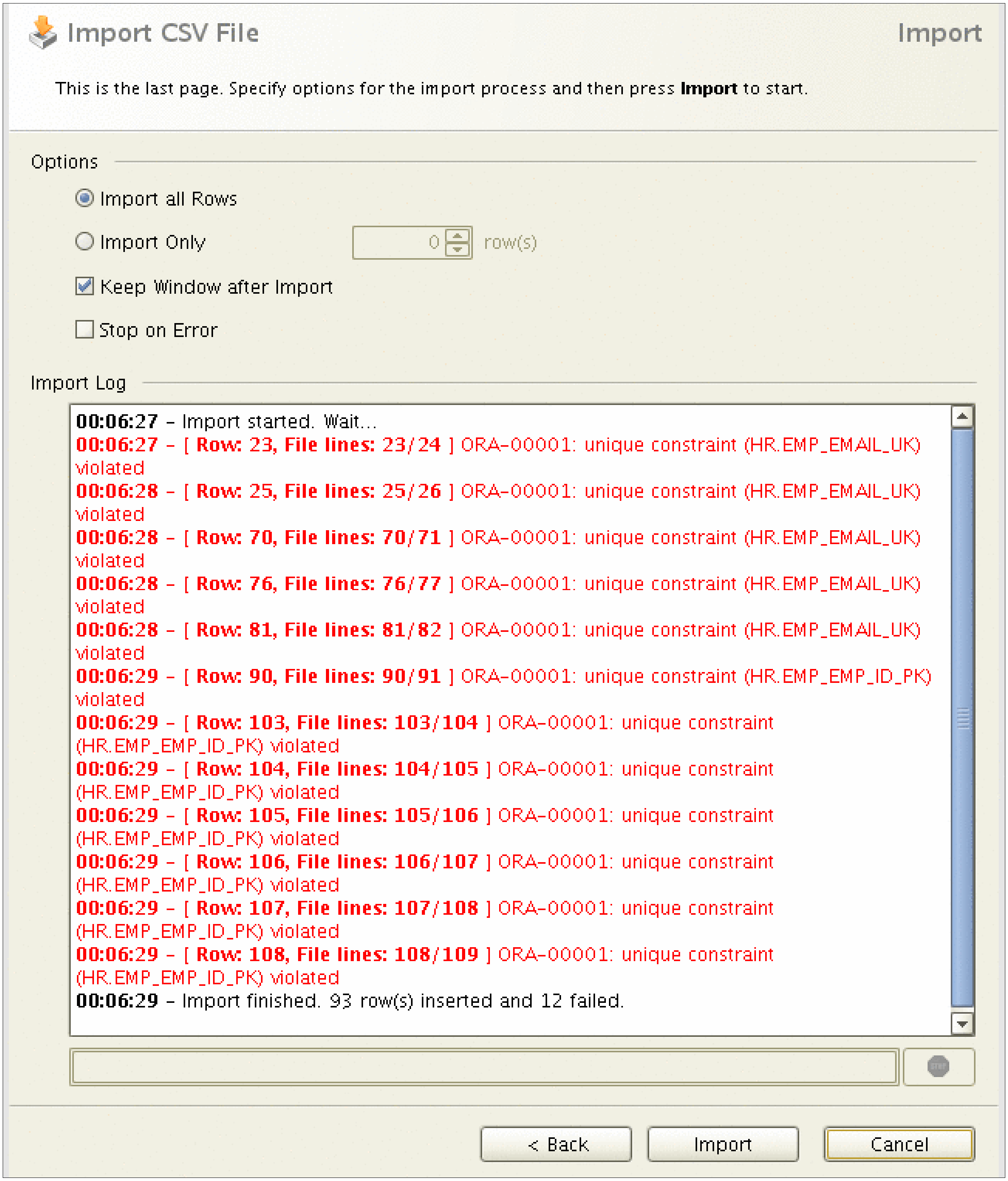
Map Table Columns with File Columns

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

**Figure: The data type drop down**

## Import process

The last wizard page is used to start and monitor the import process. Here you can select whether all rows in the source file should be imported or only a portion. Errors that occur during the import process will be presented in the log.



This is the last page. Specify options for the import process and then press **Import** to start.

Options

● Import all Rows

○ Import Only 0 row(s)

☑ Keep Window after Import

□ Stop on Error

Import Log

```
00:06:27 - Import started. Wait...
00:06:27 - [ Row: 23, File lines: 23/24 ] ORA-00001: unique constraint (HR.EMP_EMAIL_UK)
violated
00:06:28 - [ Row: 25, File lines: 25/26 ] ORA-00001: unique constraint (HR.EMP_EMAIL_UK)
violated
00:06:28 - [ Row: 70, File lines: 70/71 ] ORA-00001: unique constraint (HR.EMP_EMAIL_UK)
violated
00:06:28 - [ Row: 76, File lines: 76/77 ] ORA-00001: unique constraint (HR.EMP_EMAIL_UK)
violated
00:06:28 - [ Row: 81, File lines: 81/82 ] ORA-00001: unique constraint (HR.EMP_EMAIL_UK)
violated
00:06:29 - [ Row: 90, File lines: 90/91 ] ORA-00001: unique constraint (HR.EMP_EMP_ID_PK)
violated
00:06:29 - [ Row: 103, File lines: 103/104 ] ORA-00001: unique constraint
(HR.EMP_EMP_ID_PK) violated
00:06:29 - [ Row: 104, File lines: 104/105 ] ORA-00001: unique constraint
(HR.EMP_EMP_ID_PK) violated
00:06:29 - [ Row: 105, File lines: 105/106 ] ORA-00001: unique constraint
(HR.EMP_EMP_ID_PK) violated
00:06:29 - [ Row: 106, File lines: 106/107 ] ORA-00001: unique constraint
(HR.EMP_EMP_ID_PK) violated
00:06:29 - [ Row: 107, File lines: 107/108 ] ORA-00001: unique constraint
(HR.EMP_EMP_ID_PK) violated
00:06:29 - [ Row: 108, File lines: 108/109 ] ORA-00001: unique constraint
(HR.EMP_EMP_ID_PK) violated
00:06:29 - Import finished. 93 row(s) inserted and 12 failed.
```

**Figure: The import process page**

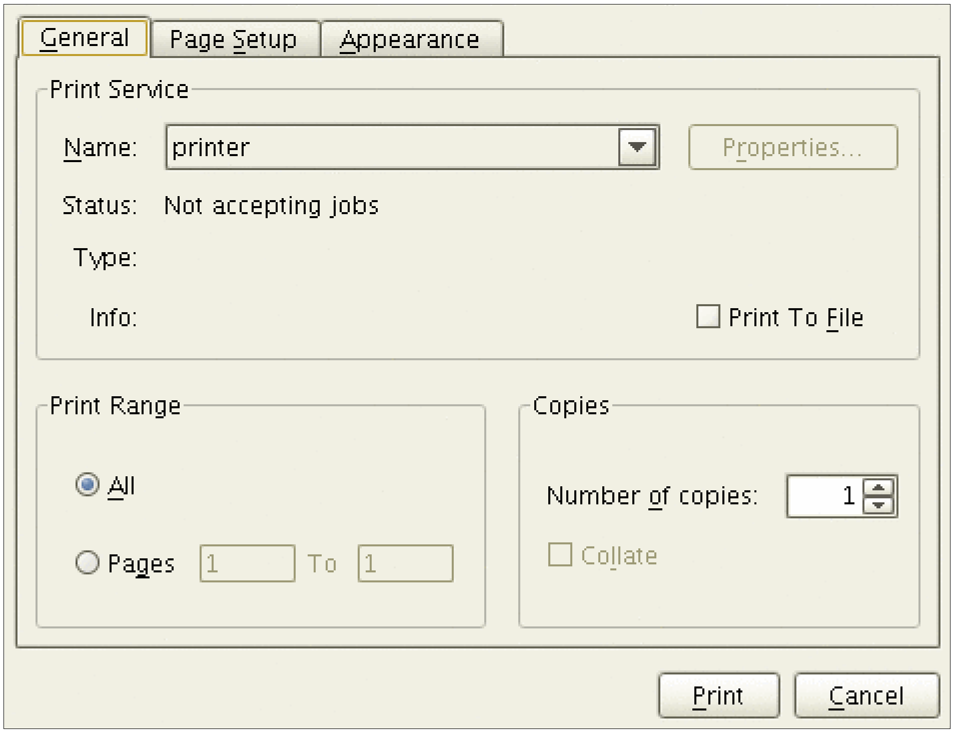
# Print

The printing support in DbVisualizer supports printing of Grids and Graphs. The print dialog looks somewhat different depending on what is printed.

**Note:** Printing of charts is currently not supported. The workaround is simply to export the chart and then use your favorite printing tool to get it on paper (a standard web browser is sufficient).

## Grid

Printing a grid in DbVisualizer causes the visual grid to be output on paper. This includes the table headers, sort and primary key indicator, etc. It can be output as a screen shot that spans several pages depending on the number of rows and columns that are printed. The other solution to printing grids is to export to HTML and then use a web browser to print it. The choice of which is more attractive than the other is up to you to decide.



Print Service

Name: printer

Status: Not accepting jobs

Type:

Info:

☐ Print To File

Print Range

● All

○ Pages 1 To 1

Copies

Number of copies: 1

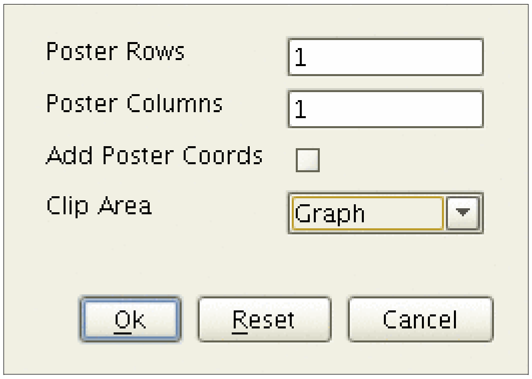
☐ Collate

**Figure: Standard print dialog**

The content and layout of the print dialog is platform specific. The above screen shot is from Linux/RedHat.

## Graph

The graph printing setup dialog adds a step before the standard printing dialog is displayed.



Poster Rows: 1

Poster Columns: 1

Add Poster Coords: □

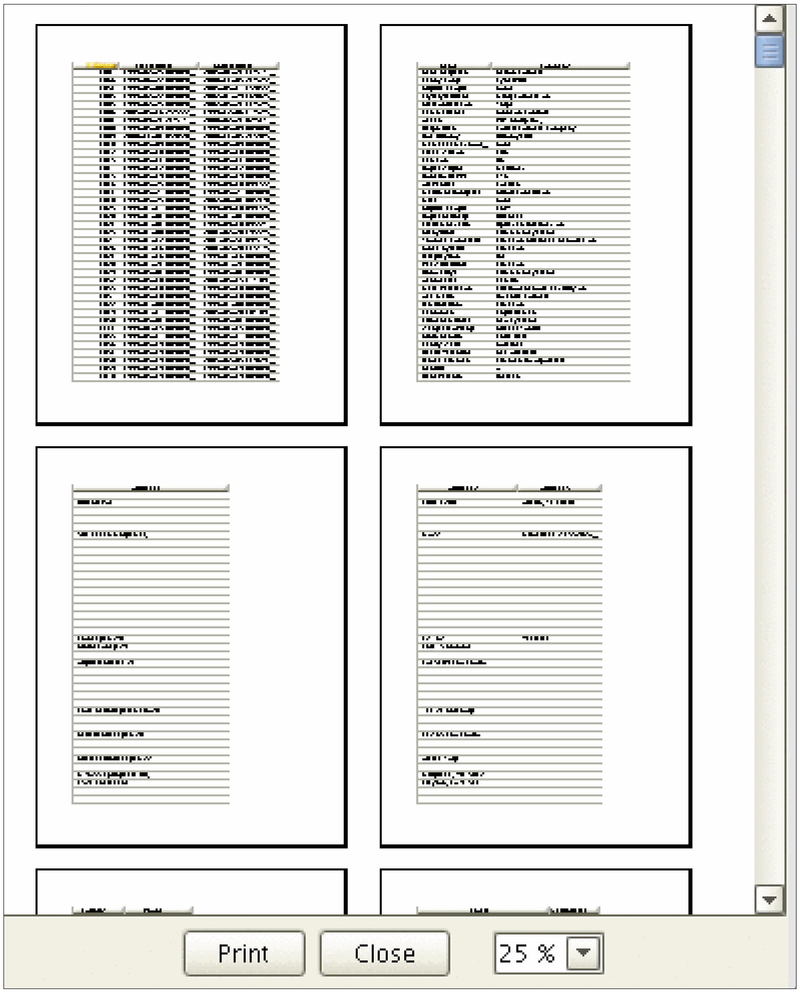
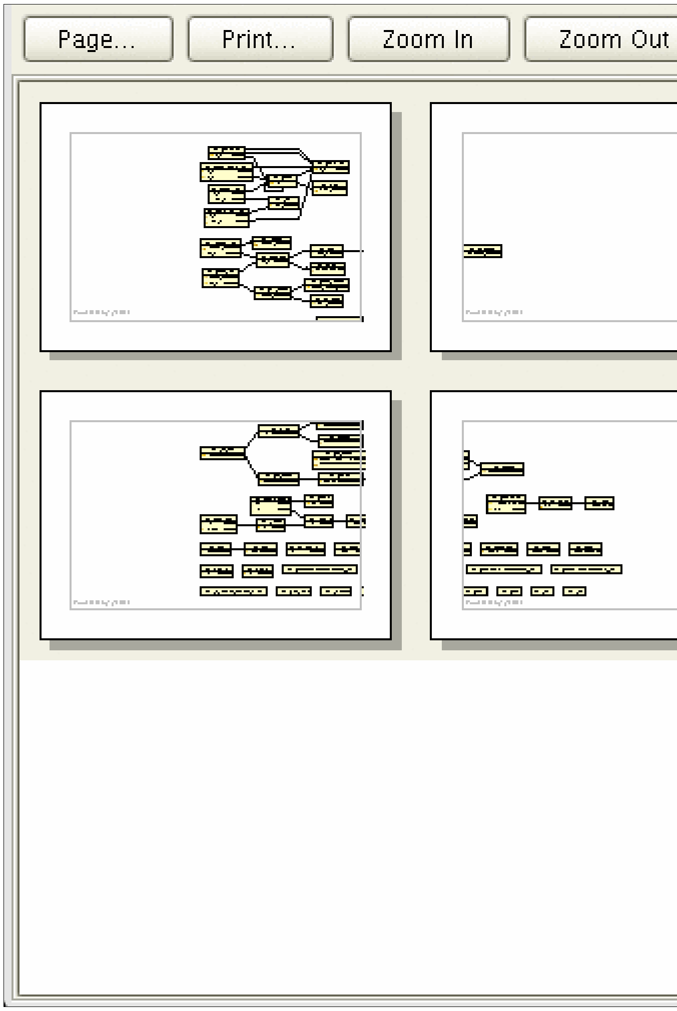
Clip Area: Graph

**Figure: Print options when printing graphs**

It is possible to specify the number of rows (pages) and columns (pages) that the complete image will be divided into. It is also possible to select whether the view as it appears on the screen will be printed or the complete graph.

# Print Preview

The **File->Print Preview** feature is used to preview a grid or graph before print.

| Grid | Graph |
|---|---|
|  |  |

### Figure: Grid and graph print previews

# Plug-in Framework

## Introduction

**Note: The plug-in framework is supported only by the DbVisualizer Personal edition.**

This document explains the database profile framework which is the base for how DbVisualizer presents information in the **Database Objects tree** and in the **Object View**. In addition it is also used to define object actions such as drop, rename, compile, create, comment, alter, etc.

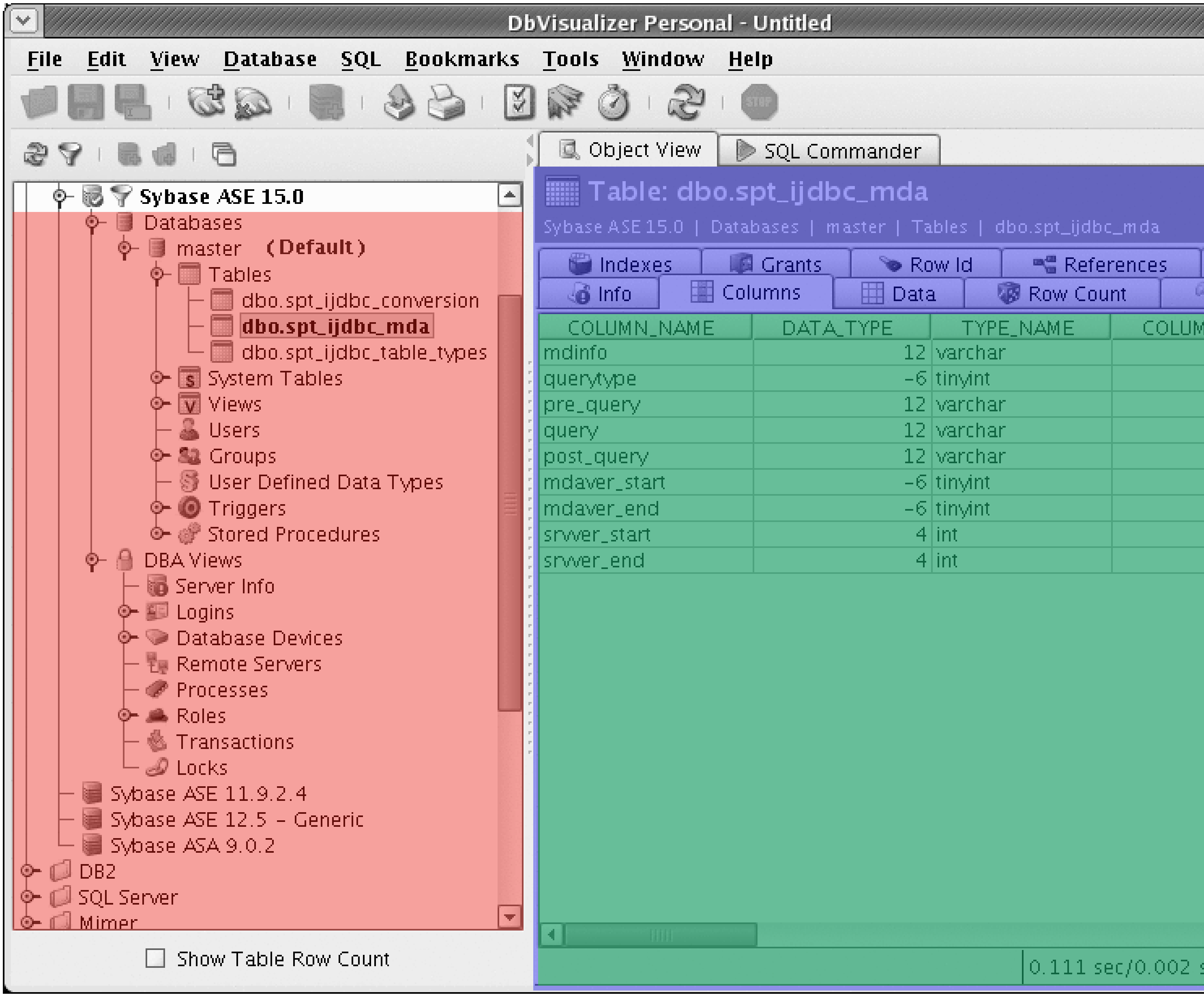
## What features in DbVisualizer relies on the database profile?

One of the most important and central features in DbVisualizer is the database objects tree used to navigate databases and the object view showing details about specific objects. The general problem exploring any database is that they are all different with respect to the information describing what's in the database (also called **system tables** or **database meta data**). This briefly means that it's rather complex for a product such as DbVisualizer since each database must be handled specifically. All existing database products do in addition 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 databases. Technically is a database profile an XML document keeping all of the logic, structure and actions easily mapped to the visual components in DbVisualizer. Another great benefit separating the database specific logic from the implementation of DbVisualizer is that anyone with some degree of domain knowledge may create a database profile. All that is needed is a text editor (preferably with XML support) and some ideas of what should be the final result.

A great source for inspiration (except for this document) is all the existing database profiles that comes with DbVisualizer. All database profiles are (and must be) stored in the **DBVIS-HOME/resources/profiles** directory.

The following figure illustrates what features in DbVisualizer that is controlled by the database profile.



| COLUMN_NAME | DATA_TYPE | TYPE_NAME | COLUM |
|---|---|---|---|
| mdinfo | 12 | varchar | |
| querytype | -6 | tinyint | |
| pre_query | 12 | varchar | |
| query | 12 | varchar | |
| post_query | 12 | varchar | |
| mdaver_start | -6 | tinyint | |
| mdaver_end | -6 | tinyint | |
| srvver_start | 4 | int | |
| srvver_end | 4 | int | |

**Figure: What the database profile control in DbVisualizer**

The red box at the left shows the **database objects tree**. This tree is used to navigate the objects in the database. Selecting an object in the tree will show the **object view** (blue box) specifically for the selected object type. An object view may have several **data views** (green) showing object information. DbVisualizer show these as labeled **tabs**. The green box shows in this screen shot the content of the data view labeled **Columns**. The type of viewer that is presenting the data in the screenshot is the **grid** viewer. Read more about all data viewers in the Viewers section.

Common for 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 will show a menu will all valid actions for the selected object. These are defined per database profile and object type. Read more about the capabilities of actions in the Definition of user actions section.

### How does DbVisualizer know what database profile to use?

DbVisualizer automatically load the appropriate database profile (XML file) based on the following:

1. The **Database Type** for the database connection is matched with the information in the **DBVIS-HOME/resources/database-mappings.xml** file to find out if there is a database profile available. If it finds one then it is used.
2. If there is no matching profile then the **generic** profile will be used. (This is very basic profile and shows only rudimentary information about the objects in the database). This is the profile used in the DbVisualizer Free edition.

A specific database profile can be manually set for a database connection. This is defined in the database connection properties. Manually choosing a profile requires that the profile supports the actual database. If it doesn't then various errors will be reported once the database objects tree is explored. (Whenever the profile is changed you must re-connect the actual database connection).

The name of the loaded profile is listed in the **Connection** tab status bar when the connection has been established. Click the profile link and the **Database Profile** list will be displayed.

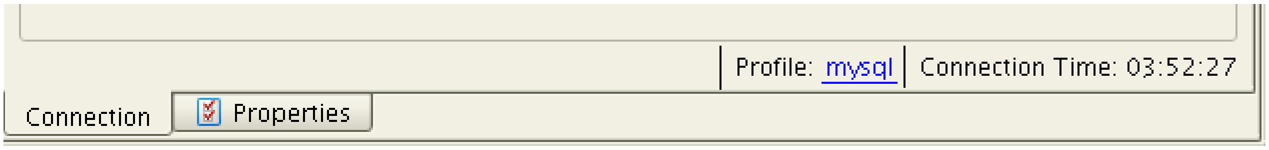


**Figure: The status bar in the Connection tab when connected**

## XML structure

The mapping from the visual components in the user interface described earlier and the element definitions in the XML file is briefly as follows:

- The database objects tree (green box) is described by the **ObjectsTreeDef** root element. (The Database Connections node is mandatory and its appearance cannot be controlled by the profile).
- The object views (green and blue boxes) are described by the **ObjectsViewDef** root element.
- The commands used to execute the SQL in order to get the information for **ObjectsTreeDef**, **ObjectsViewDef** and optionally **ObjectsActionDef** definitions are defined by the **Commands** root element.
- All **Actions** for an object is defined in the **ObjectsActionDef** root element. (Actions are optional).

The XML for a database profile is quite simple but there are a few things that need to be highlighted. All database connections loads a database profile from an XML file. If there is no matching database profile then the **generic** profile is used. This profile uses the standard JDBC meta data calls in order to obtain information about the structure and objects in the database. The generic profile is not one XML file as database specific profiles but instead four files:

- **generic-commands.xml**
- **generic-actions.xml**
- **generic-tree.xml**
- **generic-view.xml**

All these files a referred in the **generic.xml** file as include statements i.e. each of the above files will be included in the generic.xml file when loaded. The reason for this is that these files can be included and extended in a specialized profile. See later for more information.

The XML structure used to represent the database profile is organized as follows (click on the link to read more about each specific section):

- **Commands**
  Defines the SQLs for the **ObjectsTreeDef**, **ObjectsViewDef** and optionally **ObjectsActionDef**.
- **ObjectsActionDef** (optional)
  Defines actions for object types.
- **ObjectsTreeDef**
  Defines the structure and what objects should be visible in the objects tree.
- **ObjectsViewDef**
  Defines the object views for a specific object type.

## XML skeleton

The following is a minimal XML file showing its structure.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE DatabaseProfile SYSTEM "dbvis-defs.dtd" [
  <!ENTITY generic-commands SYSTEM "generic-commands.xml">
  <!ENTITY generic-view SYSTEM "generic-view.xml">
]>

<DatabaseProfile desc="Profile for Sybase ASE"
              version="$Revision: 1.20 $"
                 date="$Date: 2006/09/29 12:30:10 $"
               minver="5.0">

<!-- ================================================================== -->
<!-- Definition of the commands -->
<!-- ================================================================== -->

  <Commands>
    &generic-commands;
    ...
  </Commands>

<!-- ================================================================== -->
```

```
<!-- Definition of the object actions that are used by the tree -->
<!-- ============================================================ -->

  <ObjectsActionDef>
    ...
  </ObjectsActionDef>

<!-- ============================================================ -->
<!-- Definition of the database objects tree structure -->
<!-- ============================================================ -->

  <ObjectsTreeDef id="sybase-ase">
    ...
  </ObjectsTreeDef>

<!-- ============================================================ -->
<!-- Definition of the database objects views -->
<!-- ============================================================ -->

  <!-- Include the generic-view -->
  &generic-view;

  <ObjectsViewDef id="sybase-ase" extends="generic">
    ...
  </ObjectsViewDef>

</DatabaseProfile>
```

**Note:** The name of the XML file (sybase-ase) and the values for the **name** attribute for the **ObjectsTreeDef** and **ObjectsViewDef** elements must be the same.

The first rows in the XML defines external dependencies and their URI's. The **DOCTYPE** identifier defines the DTD that is used to verify the XML with. The **ENTITY** identifiers lists URI's for external references. In this case they identify the **generic-commands.xml** and **generic-view.xml** files. They can then be referred in the XML as **&generic-commands;** and **&generic-view;** and simply means that the related XML files will be included in the final document once the profile is loaded.

The root of the database profile is the **DatabaseProfile** element. Continue to the next sections for information about the elements forming the database profile.

**Tip:** If you are using an XML editor to edit the profile then it is very convenient loading the DTD in the editor as you then will get color and error highlighting.

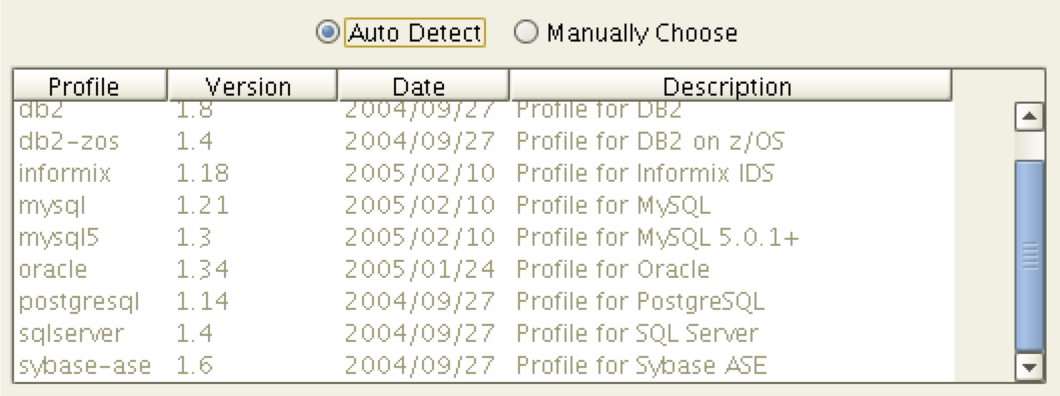
## <DatabaseProfile>

The **DatabaseProfile** is the root element in the XML file. It is required and have the following attributes.

```
<DatabaseProfile desc="Profile for Sybase ASE"
```

```
               version="$Revision: 1.20 $"
                  date="$Date: 2006/09/29 12:30:10
$"
                minver="5.0">

  ...

</DatabaseProfile>
```

The attributes specified for the **DatabaseProfile** element will appear in the **Database Profile** list when selecting the connection properties for a database connection:



● Auto Detect ○ Manually Choose

| Profile | Version | Date | Description |
|---|---|---|---|
| db2 | 1.8 | 2004/09/27 | Profile for DB2 |
| db2-zos | 1.4 | 2004/09/27 | Profile for DB2 on z/OS |
| informix | 1.18 | 2005/02/10 | Profile for Informix IDS |
| mysql | 1.21 | 2005/02/10 | Profile for MySQL |
| mysql5 | 1.3 | 2005/02/10 | Profile for MySQL 5.0.1+ |
| oracle | 1.34 | 2005/01/24 | Profile for Oracle |
| postgresql | 1.14 | 2004/09/27 | Profile for PostgreSQL |
| sqlserver | 1.4 | 2004/09/27 | Profile for SQL Server |
| sybase-ase | 1.6 | 2004/09/27 | Profile for Sybase ASE |

**Figure: The list of available database profiles**

## <Commands> - The SQLs used to interact with the database

This element keeps all **Command** elements with SQL sub element. A **Command** element is identified by a unique **id** attribute which is then referred in **ObjectsTreeDef**, **ObjectsViewDef** and (optionallty) **ObjectsActionDef** definitions.

```
<Commands>
  &generic-commands;

  <Command>
    ...
  </Command>

</Commands>
```

The first statement in the **<Commands>** element is:

```
  &generic-commands;
```

This means that the **generic- commands** entity defined at the top of the XML file will be included in the XML i.e. all its definitions will be accessible from the **ObjectsTreeDef**,

**ObjectsViewDef** and **ObjectsActionDef**. If you don't plan to use any of the generic command then simply ignore this include statement.

## <Command>

The **Command** element identifies the SQL associated with the command. The SQL should in most cases return a result set with 0 or several rows. (The exception is actions which not necessarily need to return a result set). The following command queries for login information in Sybase ASE.

```
<Command id="sybase-ase.getLogins">
  <SQL>
    <![CDATA[
select name, suid, dbname, fullname, language, totcpu,
totio, pwdate from master.dbo.syslogins
    ]]>
  </SQL>
</Command>
```

The **id** for this command is **sybase-ase.getLogins**. The reason for prefixing the id with the name of the profile is for maintainability. Since the **generic-commands.xml** file is included in most profiles it is easier to set unique prefixes for all commands so that they are not mixed with the commands in the generic-commands.xml file.

### Result set

The result set for the previous query look as follows:

| name | suid | dbname | fullname | language | totcpu | totio | pwdate |
|---|---|---|---|---|---|---|---|
| sa | 1 | master | (null) | (null) | 0 | 0 | 2005-02-24 23:59:14 |
| probe | 2 | subsystemdb | (null) | (null) | 0 | 0 | 2005-02-25 00:01:15 |

The way DbVisualizer handles the result set depends on whether the command is executed as a request in the database objects tree (**ObjectsTreeDef**) or in the object view (**ObjectsViewDef**). If executed in the database objects tree then each row in the result set will be represented by a new node in the tree. If executed in the object view then it's the actual viewer component that decides how the result will be presented. For more information how a result set is used in the **ObjectsTreeDef** or **ObjectsViewDef** then read the specific sections.

Another important difference between the database objects tree and object view is that the tree is a hierarchical structure of objects while object view presents information about a specific object. An object that is inserted in the database objects tree is a 1..1 mapping with a row from the actual result set. The end user will see these objects (nodes) by some descriptive label as defined in the **ObjectsTreeDef**. However, all data for the row from the original result set is stored with the object in the tree and may be used in the label, variables, conditions, etc. This is not the case in the **ObjectViewDef**.

The following example put some light on this. Consider the previous result set and that it's used to create objects in the database objects tree. The end user will see the following in DbVisualizer. The visible name for each row is the **name** column in the result set.

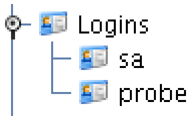


**Figure: Sample of the Logins node having two child nodes**

Each of the **sa** and **probe** nodes have all their respective data from the result set associated with the nodes. The data is referenced as **commandId.columnName** i.e. **sybase-ase.getLogins.name**, **sybase- ase.getLogins.dbname**, etc. All associated data for the **sa** node in the example are listed next:

```
sybase-ase.getLogins.name = sa
sybase-ase.getLogins.suid = 1
sybase-ase.getLogins.dbname = master
sybase-ase.getLogins.fullname = (null)
sybase-ase.getLogins.language = (null)
sybase-ase.getLogins.totcpu = 0
sybase-ase.getLogins.totio = 0
sybase-ase.getLogins.pwdate = 2005-02-24 23:59:14
```

The DataNode definition presenting **sa** and **probe** in the previous screenshot example is as follows:

```
label="${sybase-
ase.getLogins.name}"
```

### <Input> - Setting command input

There are two types of Commands, with or without dynamic input. The difference is that dynamic input Commands accepts input data that is typically used to form the **WHERE** clause in SELECT SQLs. The previous example illustrates a static SQL (without dynamic data).

To allow for dynamic input just add variables at the positions in the statement that should get dynamic values. The following is an extension to the previous example allowing for dynamic input.

```
<Command id="sybase-ase.getLogins">
  <SQL>
    <![CDATA[
select name, suid, dbname, fullname, language, totcpu,
totio, pwdate from master.dbo.syslogins
where name = '${name}' and suid = '${suid}'
```

```
    ]]>
  </SQL>
</Command>
```

The example above adds two input variables: **${name}** and **${suid}**. Values for these variables should then be supplied wherever the command is referred for execution via the **Input** element.

The following is an example from the **ObjectsTreeDef** and its use of the **sybase-ase.getLogins** command:

```
<GroupNode type="Logins" label="Logins">
  <DataNode type="Login" label="${sybase-ase.getLogins.Name}
isLeaf="true">
    <SetVar name="objectname" value="${sybase-ase.getLogins.Name}">
    <Command idref="sybase-ase.getLogins">
      <Input name="name" value="sa">
      <Input name="suid" value="${sybase-ase.getProcesses.suid}">
    </Command>
  </DataNode>
</GroupNode>
```

(Note that the **Command** element refers the command via the **idref** attribute which will be matched with the corresponding **id** for the Command).

There is no magic with this definition since the **${name}** variable in the final SQL will be replaced with string **"sa"**.

The value for the **${suid}** definition will in this case get the value of the **sybase-ase.getProcesses.suid** when the SQL is executed. So where is this variable defined? As explained in the Result Set section we introduced how all the data for a row in the result set is associated with the objects in the database objects tree. In addition it is possible to use all the data kept by the current object and all its parent objects (as presented in the objects tree) in the input to commands. So the variable **${sybase- ase.getProcesses.suid}** means that DbVisualizer will first look if the variable is found in the current object. If it doesn't exist it will continue looking through the parent objects until it reaches the root which is the **Connections** object in the objects tree. If the variable is not found it will be set to the string representation for null which is **(null)** by default. Whenever a matching variable is found DbVisualizer will use the value of it and stop searching.

### <Output> - Redefine command output

As mentioned earlier is a specific column value in a result set row referenced by the name of the column prefixed by the command id. Sometimes this is not desirable and the **Output** definition can be used to change this behavior. The following identifies a column in the result set by its index number starting from 1 and then force its name to be set to the value of the **id** attribute.

```
<Output>
```

```
  <Column id="sybase-ase.getLogins.Name" index="1">
  <Column id="sybase-ase.getLogins.suid" index="2">
</Output>
```

Another option using the **Output** element is to alter the structure of columns in the result set by either adding, renaming or removing columns.

```
<Output>
  <Column modelaction="add" index="THIS_IS_A_NEW_COLUMN" value="Rattle
and Hum">
  <Column modelaction="rename" index="ADDR" name="ADDRESS">
  <Column modelaction="rename" index="2" name="PHONE">
  <Column modelaction="drop" index="MOBILE_PHONE">
  <Column modelaction="drop" index="4">
</Output>
```

(The rename and drop actions accepts either the name of the column or index number starting from left at index 1).

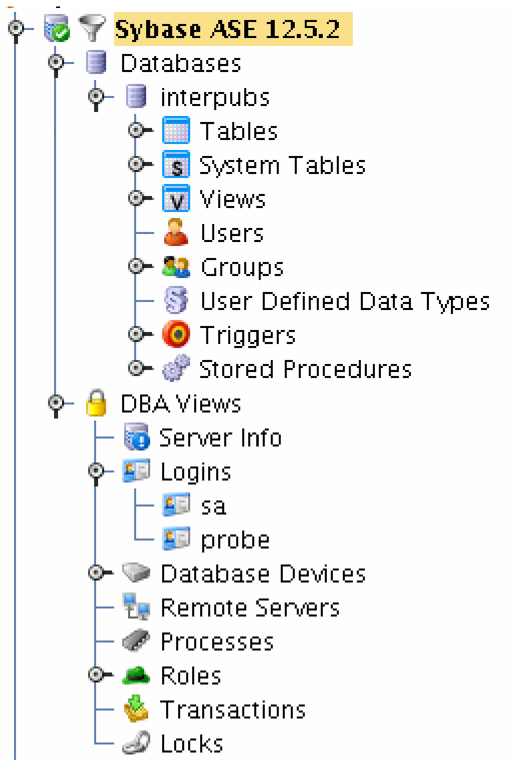
- The **add** operation is used to add a new column to all rows. The value attribute accepts variables using the **${...}** syntax.
- The **rename** operation simply renames a column.
- The **drop** operation drops the specified column.

The **rename** operation is primarily used when building a custom command that is supposed to be used by a viewer that requires pre-defined input by specific column names. Read more in the ObjectsViewDef section.

## <ObjectsTreeDef> - Definition of the Database Objects Tree

The **ObjectsTreeDef** element section controls how the database objects tree will be presented and what commands should be executed to form its content (nodes). The mapping between the graphical representation in DbVisualizer and its **ObejctsTreeDef** XML is as straight forward it can be:

```
<ObjectsTreeDef id="sybase-ase">
  <GroupNode type="Databases">
    <DataNode type="Catalog">
      <GroupNode type="Tables">
        <DataNode type="Table"/>
      </GroupNode>
      <GroupNode type="SystemTables">
        <DataNode type="SystemTable"/>
      </GroupNode>
      <GroupNode type="Views">
        <DataNode type="View"/>
      </GroupNode>
```



```
      <GroupNode type="Users"/>
      <GroupNode type="Groups">
        <DataNode type="Group"/>
      </GroupNode>
      <GroupNode type="Types"/>
      <GroupNode type="Triggers">
        <DataNode type="Trigger"/>
      </GroupNode>
      <GroupNode type="Procedures">
        <DataNode type="Procedure"/>
      </GroupNode>
    </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>
```

**Figure: The visual database objects tree and its XML definition**

The screenshot shows all nodes representing the **GroupNode** definitions in the **ObjectsTreeDef**. One exception is the **Logins** object that has been expanded (**sa** and **probe** child objects) to illustrate how **DataNode** objects look. The **ObjectsTreeDef** in the example has been simplified to show only the **type** attribute. (The label of the nodes as they appear in the visual tree is not listed in the **ObjectsTreeDef** example). The type attribute is primarily used internally in the profile as an identifier between the **ObjectsTreeDef** and the **ObjectsViewDef**. The type is also visible in the DbVisualizer GUI when either the tool tip for a tree node is displayed and in the object view header. The type is also used to identify what icon will be used to represent the object type.

There are no limitation on the number of levels in the **ObjectsTreeDef**. A good rule is however to keep it simple, clean and intuitive.

The **DataNode** definitions are the most important objects in the **ObjectTreeDef**. These also defines what object tree filters are available for each object type, if overlay'ed icons should appear (and the criteria), etc. Read the next sections for details.

## <GroupNode> - Static objects used for grouping

The **GroupNode** element is used to represent a static object in the tree. These don't have any associated SQL and appear only once where they are defined. A **GroupNode** is primarily used for structural and grouping purposes. The **GroupNode** element have the following attributes.

```
<GroupNode type="SystemTables" label="System Tables" isLeaf="false">
  ...
</GroupNode>
```

The **isLeaf** attribute is optional and controls whether the **GroupNode** may have any child objects or not. It can always be set to true but the effect in the visual database objects tree is then that the expand icon to the left of the group node icon will always be displayed even though it can never have any child objects. The default setting for **isLeaf** is false.

**Note:** If **isLeaf** is set to false and there are child Group and/or Data - nodes then these will not appear. The result may be some frustration during the design...

## <DataNode> - Dynamic objects created via SQL

The **DataNode** element feeds the tree with nodes produced by a **Command**. The example in the Command section querying for all logins in Sybase ASE look as 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** have in this example the same attributes as the **GroupNode**, the type is however **"Login"** instead of **"Logins"** as for the **GroupNode**. This difference is important once the user click on either of the objects since the the Object View will show the appropriate views based on object type. The **DataNode** definition can be seen as a template as the associated command will fetch rows of data from the database and DbVisualizer will use the **DataNode** definition to create one node per row in the result set.

The **label** attribute for the data node is somewhat different as it introduces the use of a variable (or several). The real value for the label will in this example be the value in the **Name** column produced by the **sybase- ase.getLogins** command as you can see in the **Command** definition (variable names are automatically prefixed with the command id).

The **Command** element defines by the **idref** attribute what command should be executed. The command in this case and in the Result set section produced a result set with 2 rows and 8 columns. The result will be two nodes each with the label of the **Name** column in the result set.

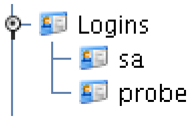


**Figure: Sample of the Logins node having two child nodes**

The label can be changed by setting it to any other valid variable or a composition of several variables. (It's even possible to specify static text in the label):

```
label="${sybase-ase.getLogins.Name} (${sybase-ase.getLogins.dbname})"
```

Will result in following being displayed:

```
sa (system)
probe (subsystemdb)
```

The complete set of attributes for the **DataNode** element is:

```
        type="value"         - The type of node (required)
          actiontype="value"         - Object type used for object
actions (optional)
               label="value"         - The visual label (required)
              isLeaf="true/false"    - Specifies if the node can have
child objects (default true)
                sort="col1,col2"     - A comma separated list of
names/variables used for sorting
drop-label-not-equal="value"         - Do not add the node if the
label is not equal to this value
                                       or variable
           warnstate="condition"     - If condition is true then show
an overlay icon for the node
          errorstate="condition"     - If condition is true then show
an overlay icon for the node
stop-label-hot-equal="value"         - The node will be a leaf if
label don't match this value
                                       or variable
     is-empty-output="continue/stop" - If result set is empty then use
this to control whether child
                                       GroupNode/DataNodes should be
added anyway or ignored
```

The **Command** definition in this example is basic since it doesn't use any variables in the SQL. Continue reading the next section for details about passing input data to commands.

## <Command>

Commands are referenced in the **DataNode** definition by the **idref** attribute. Sometimes its is required that a specific **DataNode** must supply input to a command. This is done by adding **Input** elements as children to the **Command**.

```
<DataNode type="Login" label="${sybase-ase.getLogins.Name}"
isLeaf="true">
  <Command idref="sybase-ase.getLogins">
    <Input name="name" value="sa">
    <Input name="suid" value="${sybase-ase.getProcesses.suid}">
  </Command>
</DataNode>
```

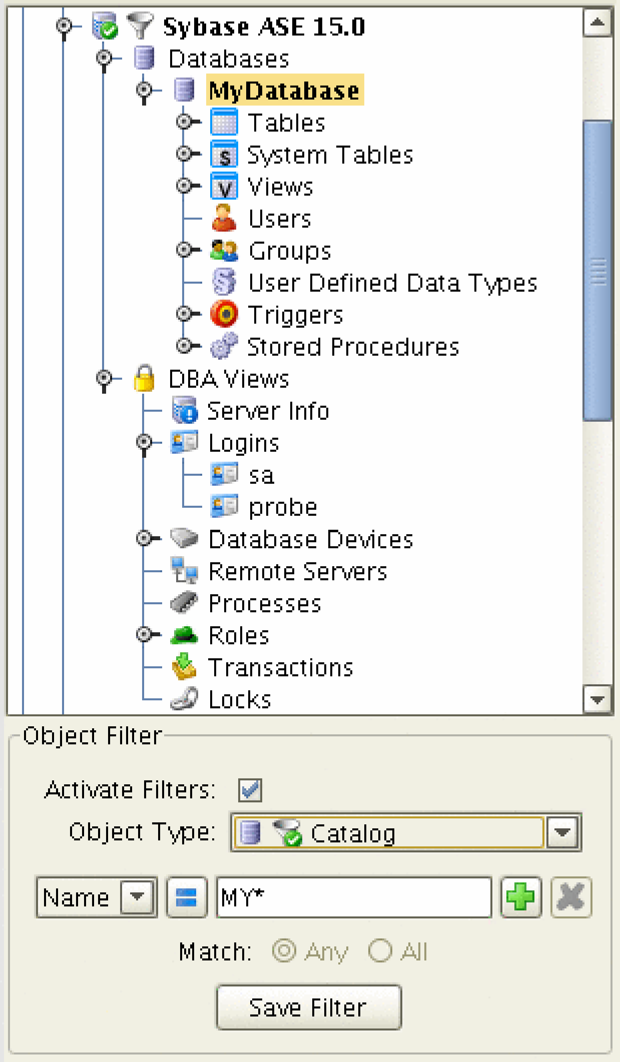
The value for variable(s) specified in the **Input** elements will be searched based on the same strategy outlined in the Result set section.

### <Filter>

The **Filter** element is specific for **Command** elements that appear in the **ObjectsTreeDef** section. A filter define what data for a **DataNode** that are allowed to use in filters. This filter functionality is commonly refered as the **Database Objects Tree Filtering** in DbVisualizer. The filtering setup appears below the database objects tree and the following example shows that filtering may be specified for these object types:

- Catalog
- Table
- System Table
- View

For each of the **Filter** definitions are one or several columns that can be used to filter on.



Activate Filters: ☑

Object Type: Catalog

Name = MY*

Match: ◉ Any ○ All

**Figure: Screen shot showing the filter pane**

```
<DataNode type="Views" label="${sybase-ase.getViews.Name}"
isLeaf="true">
  <Command idref="sybase-ase.getViews">
    <Filter type="View" name="View Table">
      <Column index="TABLE_NAME" name="Name"/>
    </Filter>
  </Command>
</DataNode>
```

The previous filter definition specifies a filter for the **View** object type. The **name** specifies the name for the filter as it will appear in the object type drop down box. The **Column** element then define the **index** which should be either a column name in the result set or an index number representing the actual column. The **name** attribute specifies the name of the column as it will appear in the filter pane.

Several **Column** elements may be specified for a **Filter** element.

**<SetVar>**

The **SetVar** element is needed in the **ObjectsTreeDef** for **DataNode**'s. DbVisualizer relies on some object types as these have special meaning. Two examples are the **Catalog** and **Schema** object types. For **DataNode** objects that you now will represent these types there must a **SetVar** identifying them. The name attribute should then be set to **"catalog"** and **"schema".**

```
<DataNode type="Catalog" label="${getCatalogs.TABLE_CAT}"
isLeaf="false">
  <SetVar name="catalog" value="${getCatalogs.TABLE_CAT}">
</DataNode>
```

All non **Catalog** or **Schema DataNode**'s must use **SetVar** to set the **"objectname"** variable:

```
<DataNode type="Views" label="${sybase-ase.getViews.Name}"
isLeaf="true">
  <SetVar name="objectname" value="${sybase-ase.getViews.Name}">
  <SetVar name="rowcount" value="true/false">
</DataNode>
```

The **objectname** variable is used to identify the object so that it can be uniformly referenced in object views and object actions. Its value should be the identifier for the object as it is identified in the database.

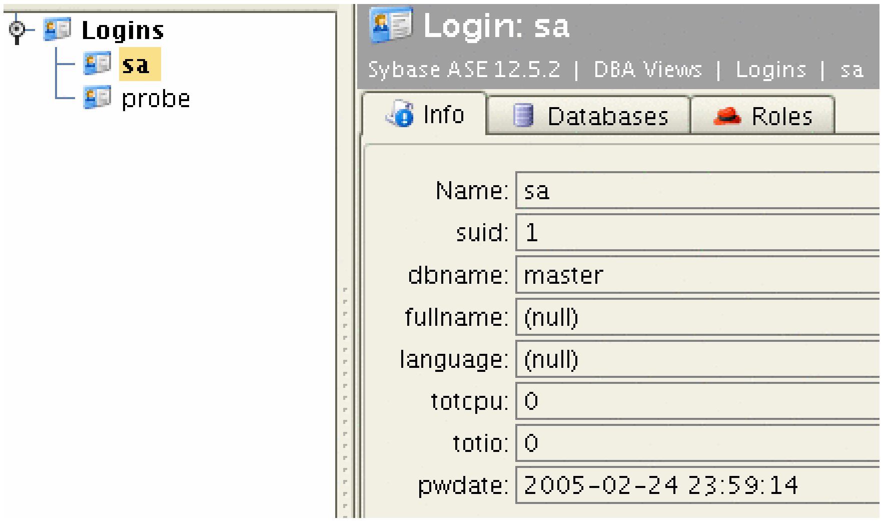
The **rowcount** setting is optional and control whether the object supports getting row count via the **select count(*)** SQL statement. This setting is also used to identify if the object is allowed for use in the **Query Builder**.

## <ObjectsViewDef> - Definition of the Object Views

The **ObjectsViewDef** element defines all views for the object types in the objects tree. These views are displayed in the **Object View** for the selected object. What views should appear when selecting a node in the tree is based on the object type for the tree node and the corresponding object view definition.

When an object is selected in the tree (**sa** in the screenshot below) its complete information is passed to the object view handler (right in the sample). This handler determines based on the object type what object view will present the information. When the object view is found all data views are created as tabs in the user interface. The selected object and its information is passed to each of the data views for processing and presentation. The following shows how the Object View look in DbVisualizer and its accompanying **ObjectView** definitions.

```
<ObjectView
type="Logins">
  <DataView
type="Logins"
label="Logins"

viewer="grid">
    <Command
idref="sybase-
ase.getLogins"/>
  </DataView>
</ObjectView>

<ObjectView
type="Login">
  <DataView type="Info"
label="Info"

  viewer="node-form"/>
  <DataView
type="Databases"
label="Databases"

viewer="grid">
    <Command
idref="sybase-
ase.getLoginDatabases"/
>
  </DataView>
  <DataView
type="Roles"
label="Roles"

viewer="grid">
    <Command
idref="sybase-
ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```



Name: sa
suid: 1
dbname: master
fullname: (null)
language: (null)
totcpu: 0
totio: 0
pwdate: 2005-02-24 23:59:14

**Figure: The visual database objects tree, object view and the XML definition**

The screenshot shows both the **Logins** node and its child nodes, **sa** and **probe**. What is not obvious in the screenshot is the object types for these objects. The Logins node is of type **Logins** while the sub nodes are **Login** types.

The **ObjectView** XML definitions shows the views for two types, **Logins** and **Login**. Clicking on the node labeled **Logins** in the tree will show the object view for the **<ObjectView type="Logins">** definition while clicking on the node labeled **sa** or **probe** will show the object view for the **<ObjectView type="Login">** .

The example shows **sa** being selected. Its **DataView** definitions are (by label):

- Info
- Databases
- Roles

These views are presented in DbVisualizer as tabs. The label of each tab is the label defined in the **DataView** and the icons are defined by the respective object type.

The **ObjectsViewDef** root element have the following attributes

```
<!-- Include the generic-view -->
&generic-view;

<ObjectsViewDef id="Views" extends="generic" >
  ...
</ObjectsViewDef>
```

The first statement for the **ObjectsViewDef** elements is:

```
  &generic-view;
```

This simply means that the generic- view entity defined at the top of the XML file will be included in the XML i.e. all its definitions will be accessible as is. An example is the **ObjectView** definition in the generic- view.xml file for the **Table** object type. It contains a lot of **DataView** elements that identifies all viewers for the **Table**. If you now want to use the generic Table **DataView**'s but add a new **Abbreviations** data view then simply extend the generic Table **DataView**. This is briefly done by adding for example a **extends="generic"** attribute in the **ObjectsViewDef** element. Then by using the exact same object type in the extended **ObjectView** you will get this behavior. Read more about extending **ObjectView**'s in the Extending ObjectView section.

## <ObjectView>

The **ObjectView** element is identified by an object type and groups all **DataView** elements that appear when the object type is selected in the database objects tree. Here follows the **ObjectView** definition for the **Login** object type.

```
<ObjectView type="Login">
  ...
</ObjectView>
```

This element is simple as its only attribute is the **type** attribute. The **type** is used when a node is clicked in the database objects tree to map the object of the type clicked and its **ObjectView**.

## <DataView>

The **DataView** element is as important as the **DataNode** is in the **ObjectsTreeDef**. It defines how the viewer should be labeled in DbVisualizer, what viewer (presentation form) it should use, commands and other things. The following is the **DataView** definitions for the **Login** object type. (The **ObjectView** element is part of the sample just for clarification).

```
<ObjectView type="Login">
  <DataView type="Info" label="Info" viewer="node-form"/>
  <DataView type="Databases" label="Databases" viewer="grid">
    <Command idref="sybase-ase.getLoginDatabases"/>
  </DataView>
  <DataView type="Roles" label="Roles" viewer="grid">
    <Command idref="sybase-ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```

This definition will be presented in DbVisualizer as described in the introduction of the **ObjectsViewDef** section. These three data view elements have the **viewer** attribute. It identifies how the data in the view will be presented. See next section for a list of viewers.

### Viewers

The **viewer** attribute for a **DataView** specifies how the data for the view should be presented. The following sections walk through the supported viewers.

The following sample illustrates the **viewer** attribute.

```
<ObjectView type="Login">
  <DataView type="Info" label="Info" viewer="node-form"/>
</ObjectView>
```

**DataView** definitions may be nested and the viewers are then presented with the nested DataView being presented in the lower part of the screen.

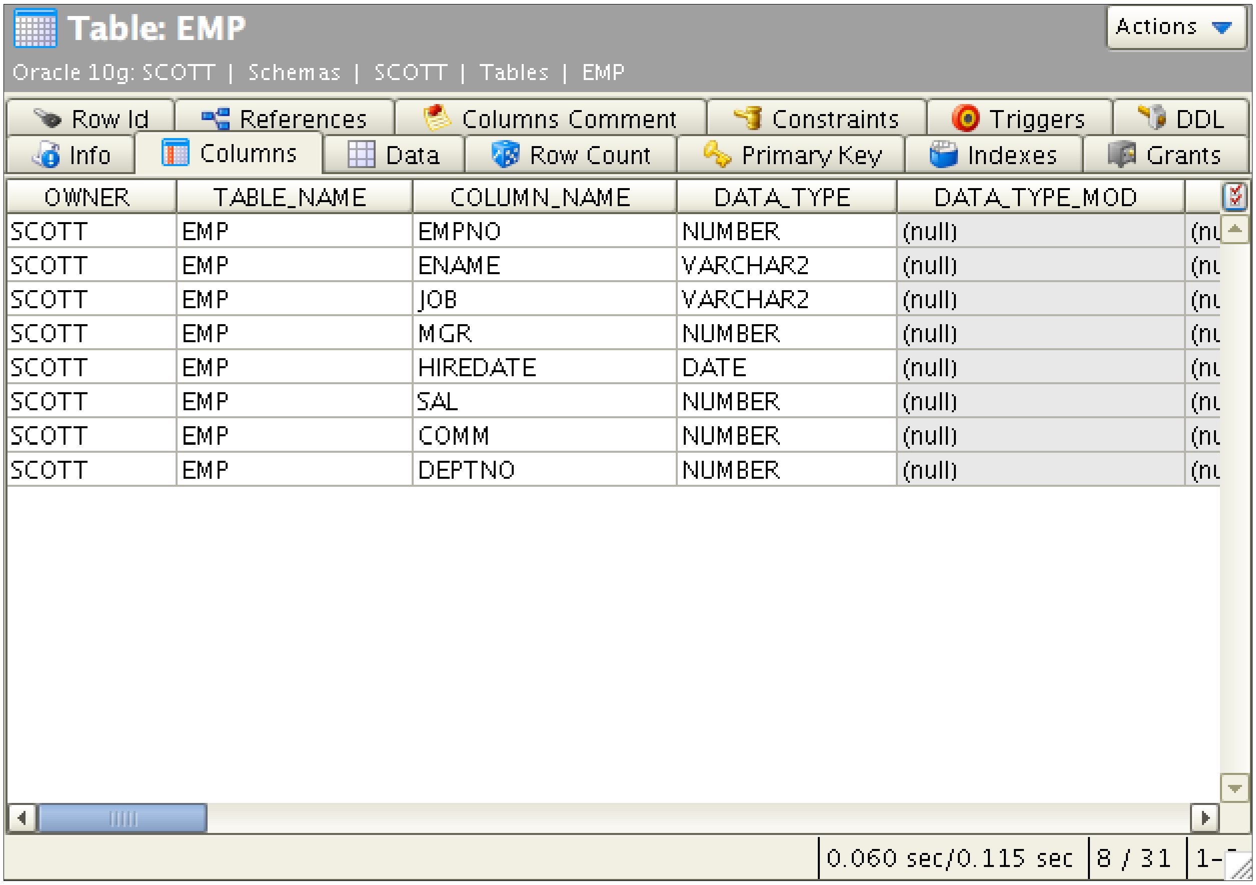
#### grid

The **grid** viewer presents a result set in a grid with standard grid features such as search, copy, fit, export, etc. The result set is presented exactly as it is produced by the **Command** and any optional **Output** processing.

Here is a sample XML for the grid viewer:

```
<DataView type="Columns" label="Columns" viewer="grid">
  <Command idref="oracle.getColumns">
    <Input name="owner" value="${schema}"/>
    <Input name="table" value="${objectname}"/>
  </Command>
</DataView>
```

Screenshot of the previous definition.



| OWNER | TABLE_NAME | COLUMN_NAME | DATA_TYPE | DATA_TYPE_MOD | |
|---|---|---|---|---|---|
| SCOTT | EMP | EMPNO | NUMBER | (null) | (nu |
| SCOTT | EMP | ENAME | VARCHAR2 | (null) | (nu |
| SCOTT | EMP | JOB | VARCHAR2 | (null) | (nu |
| SCOTT | EMP | MGR | NUMBER | (null) | (nu |
| SCOTT | EMP | HIREDATE | DATE | (null) | (nu |
| SCOTT | EMP | SAL | NUMBER | (null) | (nu |
| SCOTT | EMP | COMM | NUMBER | (null) | (nu |
| SCOTT | EMP | DEPTNO | NUMBER | (null) | (nu |

**Figure: The grid viewer**

The nesting capability for grid viewers is really powerful as it can be used to create a drill down view of the data. Consider the scenario with a **grid** viewer showing all Trigger objects. Wouldn't it be nice offering the capability to display the trigger source when selecting a row in the list? This is easily accomplished with the following:

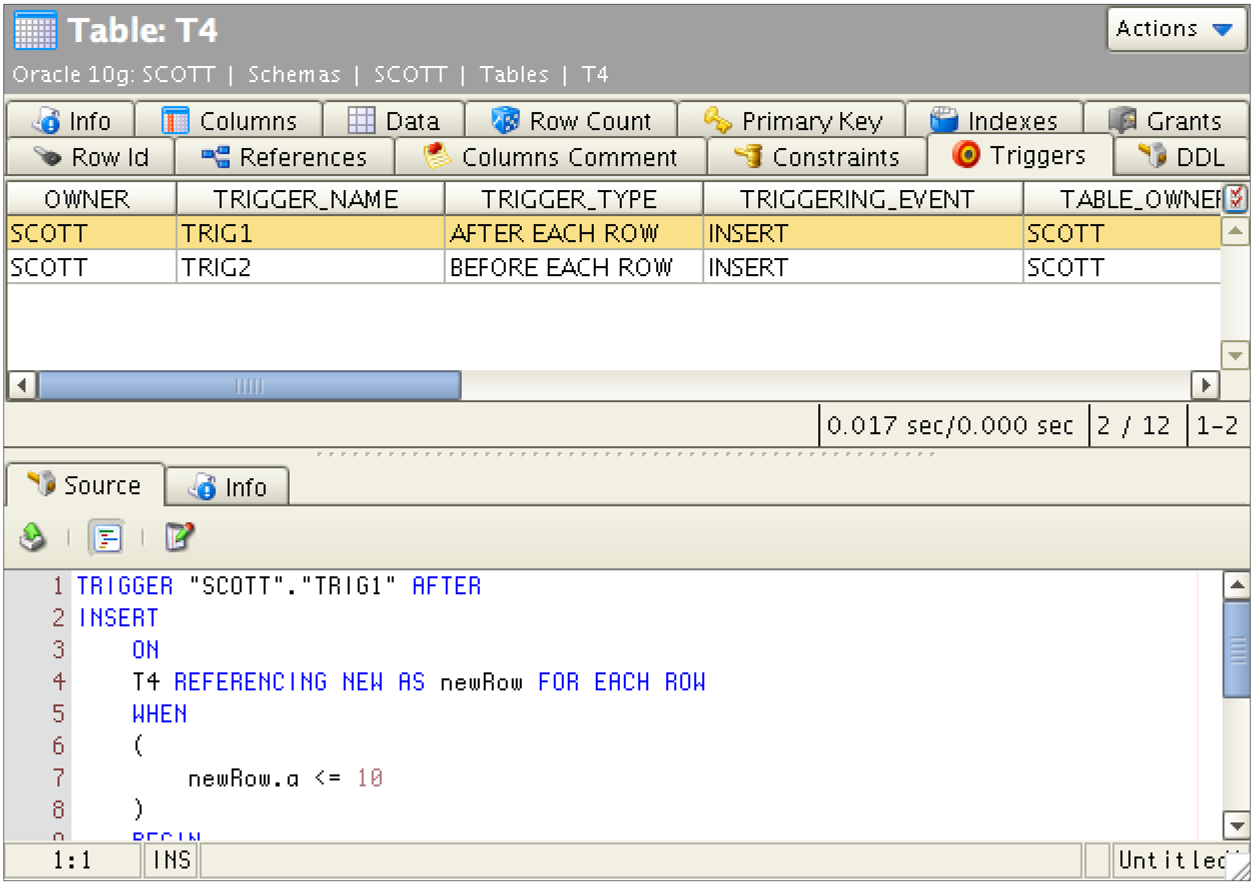
```
<DataView type="Trigger" label="Triggers" viewer="grid">
  <Command idref="oracle.getTriggers">
    <Input name="owner" value="${schema}"/>
    <Input name="table" value="${objectname}"/>
  </Command>
  <DataView type="Source" label="Source" viewer="text">
    <Input name="dataColumn" value="text"/>
    <Input name="formatSQL" value="true"/>
    <Command idref="oracle.getTriggerSource">
      <Input name="owner" value="${OWNER}"/>
      <Input name="name" value="${TRIGGER_NAME}"/>
    </Command>
  </DataView>
  <DataView type="Info" label="Info" viewer="node-form"/>
</DataView>
```

- The first **DataView** definition defines the top grid viewer and the command to get the result set for it.
- The next **DataView** is the nested text viewer specifying various input parameter for the viewer along with the command to get the source for the trigger. The difference

here is that the input parameters for this command reference column names in the top grid. Since this viewer is nested it will automatically be notified whenever an entry in the top grid is selected.

- The third nested **DataView** will be presented as a tab next to the **Source** viewer and presents info about the selected trigger.

The following screenshot illustrates the above sample:



| OWNER | TRIGGER_NAME | TRIGGER_TYPE | TRIGGERING_EVENT | TABLE_OWNE |
|---|---|---|---|---|
| SCOTT | TRIG1 | AFTER EACH ROW | INSERT | SCOTT |
| SCOTT | TRIG2 | BEFORE EACH ROW | INSERT | SCOTT |

```
TRIGGER "SCOTT"."TRIG1" AFTER
INSERT
    ON
    T4 REFERENCING NEW AS newRow FOR EACH ROW
    WHEN
    (
        newRow.a <= 10
    )
    BEGIN
```

**Figure: Example use of nested DataViews**

### Adding custom menu items in the grid

The **menuItem** parameter specifies entries that will appear in the right click menu in the grid. The value for the **menuItem** is the label for the item while the child **Input** specifies the SQL command that will be produced for all selected rows when the menu item is selected. The result of a custom menu item is that the grid viewer will create a statement that it copies to the SQL Commander, it will never execute the produced SQL in the scope of the viewer.

The following is an example with two menu items:

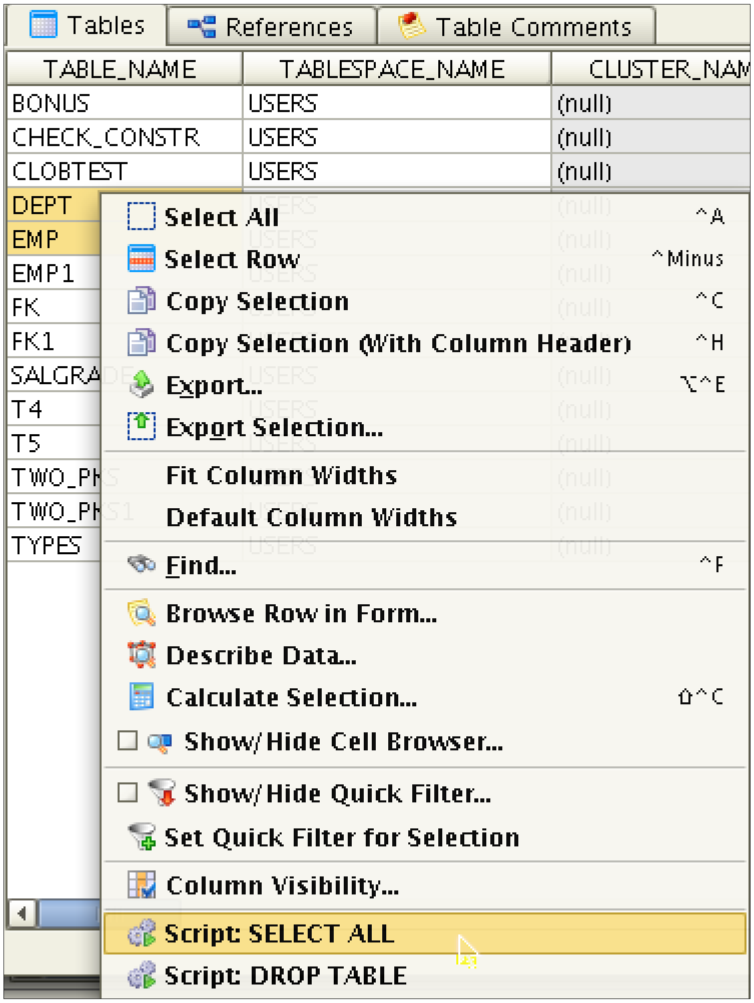
- **Script: SELECT ALL**
- **Script: DROP TABLE**

The variables in the SQL statement should identify column names in the result set. The user may select any columns in the visual grid and choose a custom menu item. It is only the actual rows that are picked from the selection as the columns are pre-defined by the

**menuItem** declaration. The variables specified in these examples starts with **${schema=...}** and **${object=...}**. These defines that the first variable represents a **schema** variable while the second defines it to be an **object**. This is needed for DbVisualizer to determine whether **delimited identifiers** should be used and if identifiers should be **qualified** as defined in connection properties for the actual database.

```
<Input name="menuItem" value="Script: SELECT ALL">
 <Input name="command" value="select * from
${schema=OWNER}${object=TABLE_NAME}"/>
</Input>

<Input name="menuItem" value="Script: DROP TABLE">
 <Input name="command" value="drop table
${schema=OWNER}${object=TABLE_NAME}"/>
</Input>
```

Here is a sample:



| TABLE_NAME | TABLESPACE_NAME | CLUSTER_NAM |
|---|---|---|
| BONUS | USERS | (null) |
| CHECK_CONSTR | USERS | (null) |
| CLOBTEST | USERS | (null) |
| DEPT | USERS | (null) |
| EMP | USERS | (null) |
| EMP1 | USERS | (null) |
| FK | | (null) |
| FK1 | | |
| SALGRADE | USERS | (null) |
| T4 | USERS | (null) |
| T5 | USERS | (null) |
| TWO_PKS | | (null) |
| TWO_PKS1 | | (null) |
| TYPES | USERS | (null) |

**Figure: Custom menu items in grid viewer**

**Note:** The result of selecting a menu item defined as a **menuItem** input parameter is that the specified command will be copied to the current SQL editor.

### Setting initial max column width

Some result sets may contain columns with very wide data. The following parameter sets an initial maximum column width for all columns in the grid.

```
<Input name="columnWidth" value="<pixels>"/>
```

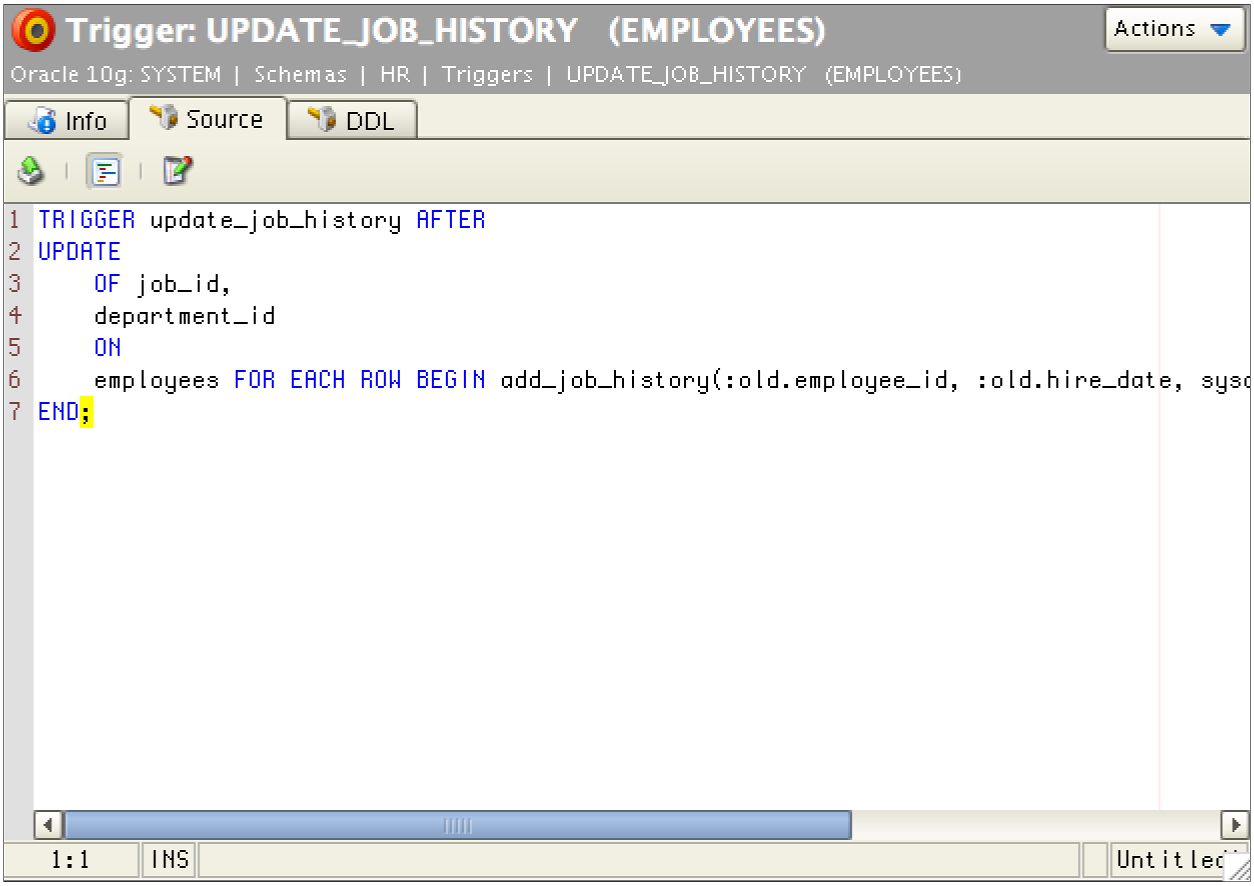
text

The **text** viewer presents data from one column in a result set in a text browser (read only editor). This viewer is typically used to present large chunks of data such as source code, SQL statements, etc. If the result set contains several rows then this viewer will fetch the data in the actual column for each row and present the combined data in the text viewer.

Here is a sample XML for the text viewer:

```
<DataView type="Source" label="Source" viewer="text">
  <Input name="dataColumn" value="text"/>
  <Input name="formatSQL" value="true"/>
  <Command idref="oracle.getTriggerSource">
    <Input name="owner" value="${schema}"/>
    <Input name="name" value="${objectname}"/>
  </Command>
</DataView>
```

Screenshot of the previous definition.


Trigger: UPDATE_JOB_HISTORY (EMPLOYEES)
Actions
Oracle 10g: SYSTEM | Schemas | HR | Triggers | UPDATE_JOB_HISTORY (EMPLOYEES)
Info
Source
DDL
1:1
INS
Untitled

```
TRIGGER update_job_history AFTER
UPDATE
    OF job_id,
    department_id
    ON
    employees FOR EACH ROW BEGIN add_job_history(:old.employee_id, :old.hire_date, sysd
END;
```

Figure: The text viewer

### Specify what column to browse

The text viewer automatically picks the data in first column. This behaviour 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 left).

```
<Input name="dataColumn" value="<column-name>"/>
```

### Enable SQL formatting of the data

The text viewer includes the **SQL Formatting** toolbar button which when pressed will format the content in the viewer. The **formatSQL** input parameter is used to control whether formatting should be enabled by default. If **formatSQL** is not specified no initial formatting is made.

```
<Input name="formatSQL" value="<true/false>"/>
```

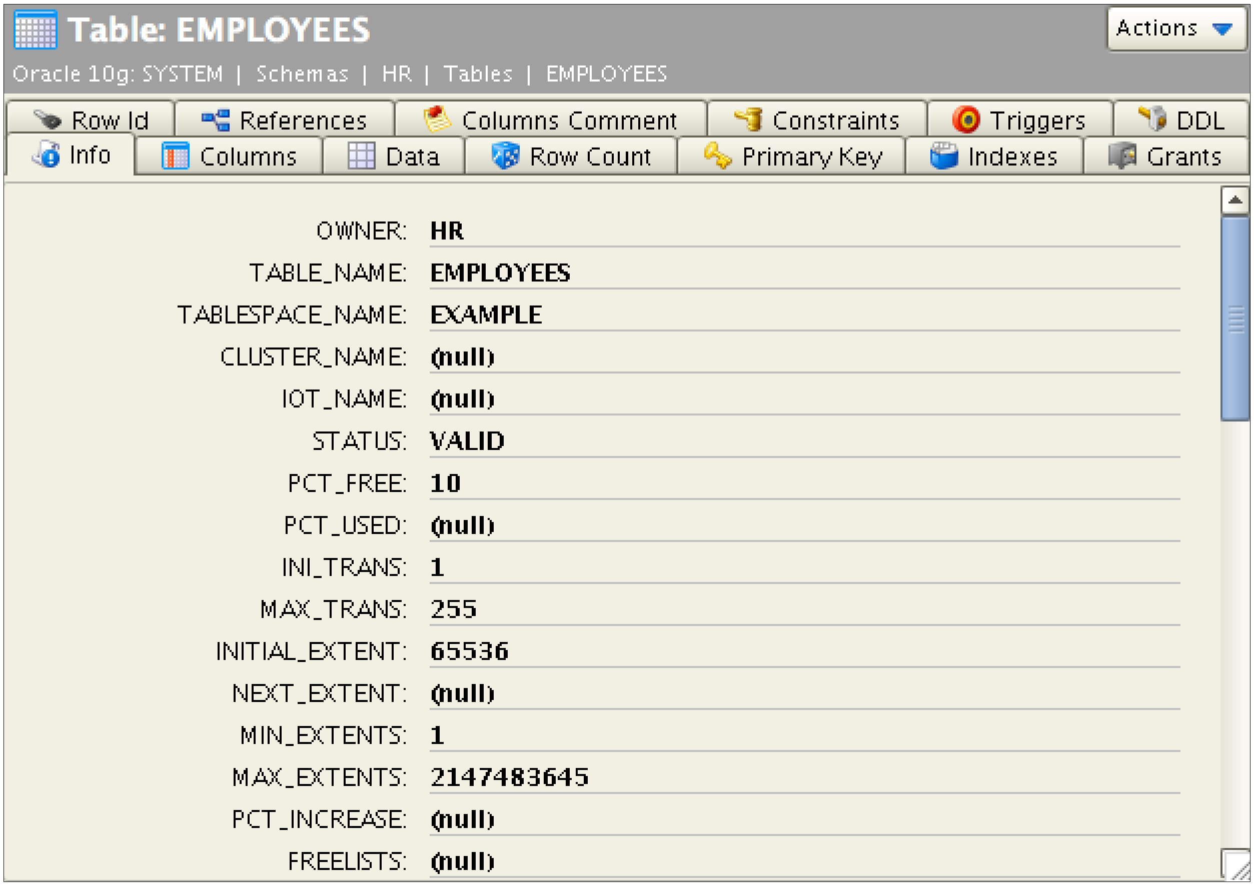
#### form

Presents row(s) from a result set in a form. If several rows are in the result then these are presented in a list. Selecting one row from the list will present all columns and data for that row in a form.

Here is a sample XML for the form viewer:

```
<DataView type="Info" label="Info" viewer="form">
 <Command idref="oracle.getTable">
  <Input name="owner" value="${schema}"/>
  <Input name="table" value="${objectname}"/>
 </Command>
</DataView>
```

Screenshot of the previous definition.



| | |
|---|---|
| OWNER: | HR |
| TABLE_NAME: | EMPLOYEES |
| TABLESPACE_NAME: | EXAMPLE |
| CLUSTER_NAME: | (null) |
| IOT_NAME: | (null) |
| STATUS: | VALID |
| PCT_FREE: | 10 |
| PCT_USED: | (null) |
| INI_TRANS: | 1 |
| MAX_TRANS: | 255 |
| INITIAL_EXTENT: | 65536 |
| NEXT_EXTENT: | (null) |
| MIN_EXTENTS: | 1 |
| MAX_EXTENTS: | 2147483645 |
| PCT_INCREASE: | (null) |
| FREELISTS: | (null) |

**Figure: The form viewer**

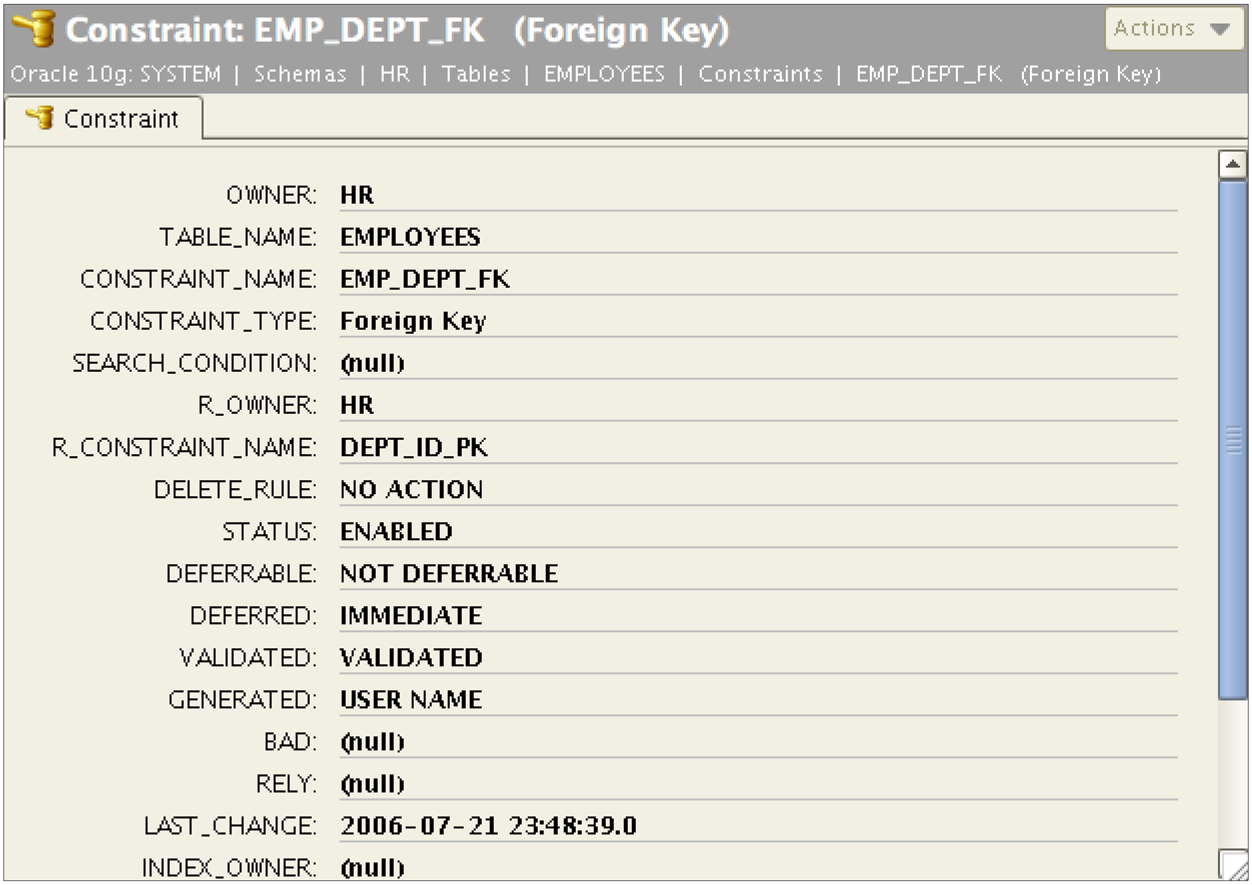
### node-form

Presents all data associated with the selected object (variables).

Here is a sample XML for the node-form viewer:

```
<DataView type="Constraint" label="Constraint" viewer="node-form"/>
```

Screenshot of the previous definition.


Constraint: EMP_DEPT_FK (Foreign Key)
Actions
Oracle 10g: SYSTEM | Schemas | HR | Tables | EMPLOYEES | Constraints | EMP_DEPT_FK (Foreign Key)
Constraint

| | |
|---|---|
| OWNER: | HR |
| TABLE_NAME: | EMPLOYEES |
| CONSTRAINT_NAME: | EMP_DEPT_FK |
| CONSTRAINT_TYPE: | Foreign Key |
| 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: | 2006-07-21 23:48:39.0 |
| INDEX_OWNER: | (null) |

**Figure: The node-form viewer**

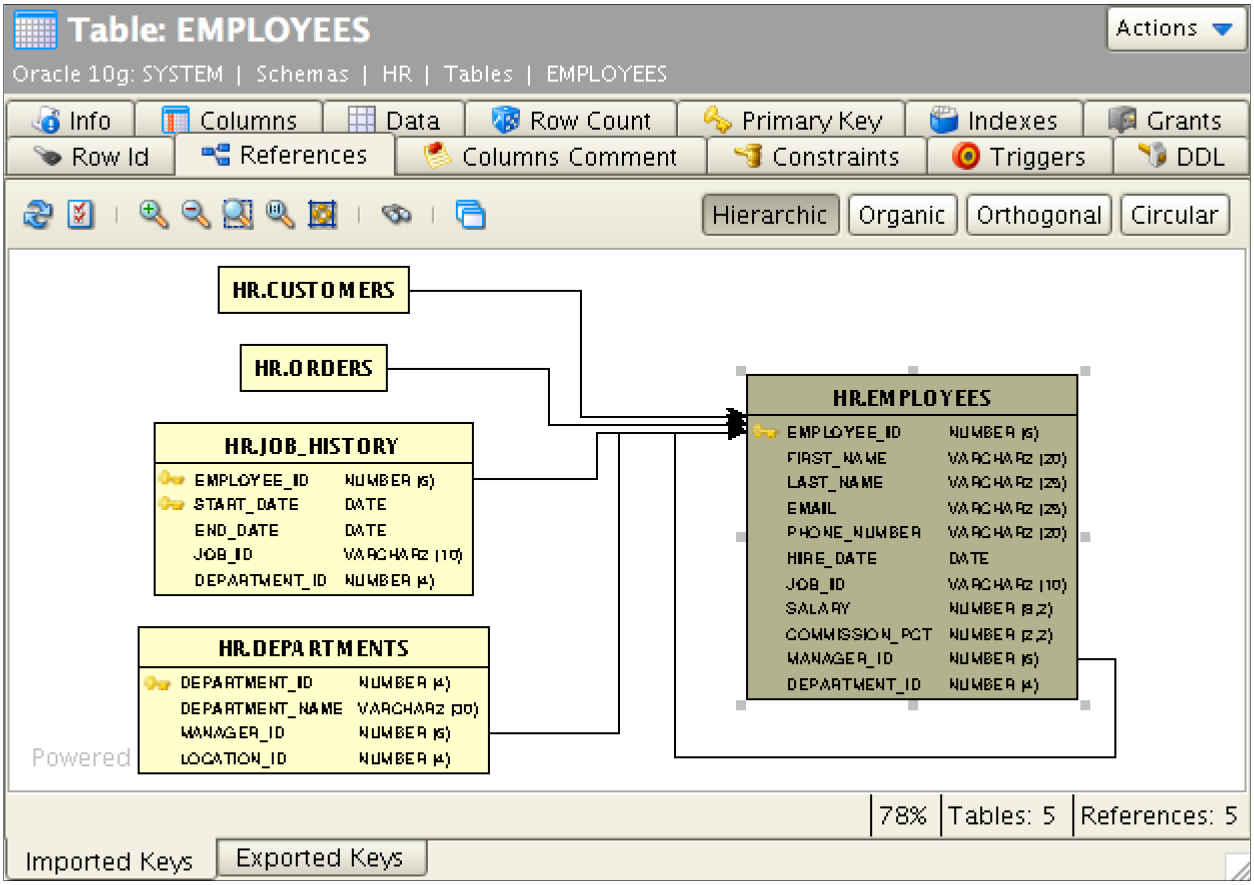
### table-refs

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 XML for the table-refs viewer:

```
<DataView type="References" label="References" viewer="table-refs"/>
```

Screenshot of the previous definition.



| HR.JOB_HISTORY | |
|---|---|
| EMPLOYEE_ID | NUMBER (6) |
| START_DATE | DATE |
| END_DATE | DATE |
| JOB_ID | VARCHAR2 (10) |
| DEPARTMENT_ID | NUMBER (4) |

| HR.DEPARTMENTS | |
|---|---|
| DEPARTMENT_ID | NUMBER (4) |
| DEPARTMENT_NAME | VARCHAR2 (30) |
| MANAGER_ID | NUMBER (6) |
| LOCATION_ID | NUMBER (4) |

| HR.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) |

**Figure: The table-refs viewer**

### tables-refs

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 XML for the tables-refs viewer:

```
<DataView type="References" label="References" viewer="tables-refs">
  <Command idref="getTables">
    <Input name="catalog" value="${catalog}"/>
    <Input name="schema" value="${schema}"/>
    <Input name="table" value="${objectname}"/>
    <Input name="type" value="${tableType}"/>
  </Command>
</DataView>
```

Screenshot of the previous definition.

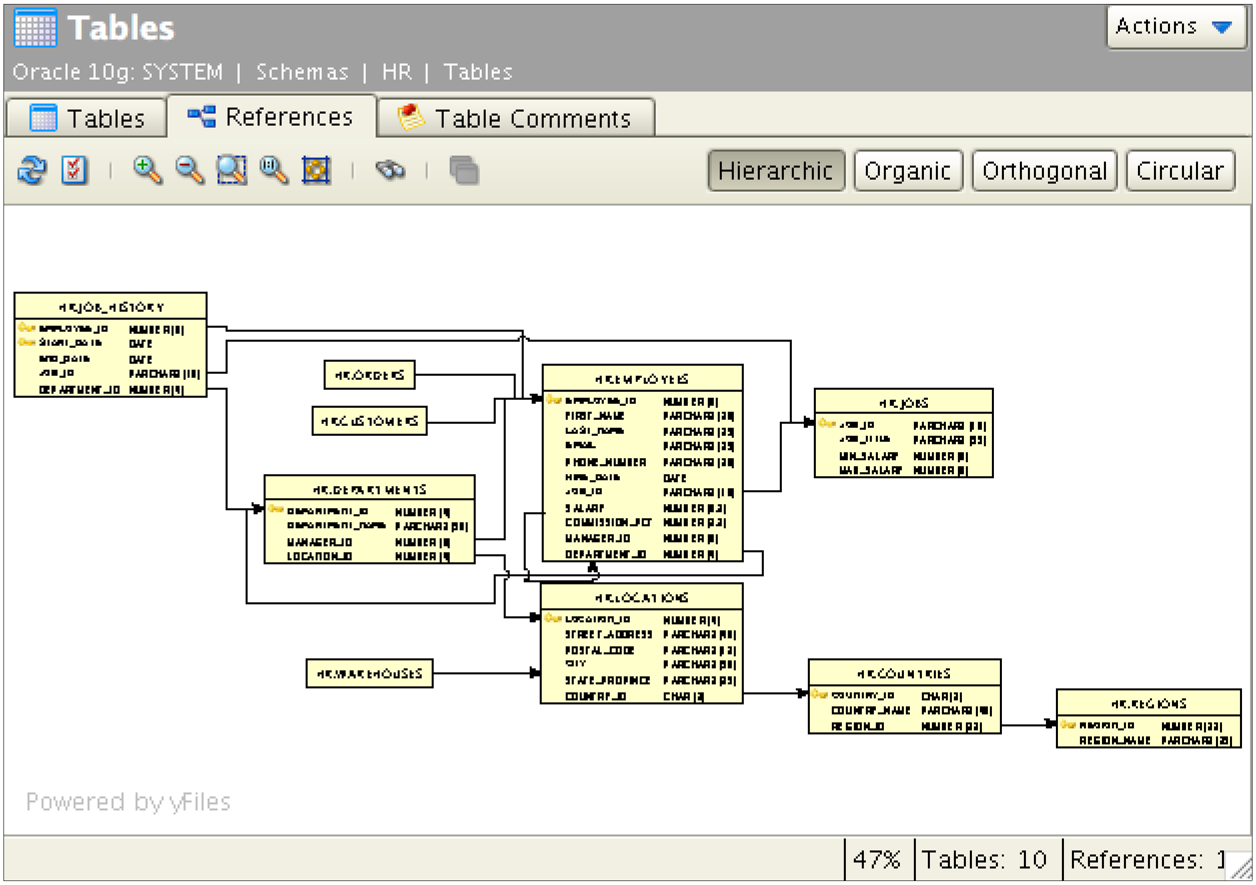


**Figure: The tables-refs viewer**

### table-data

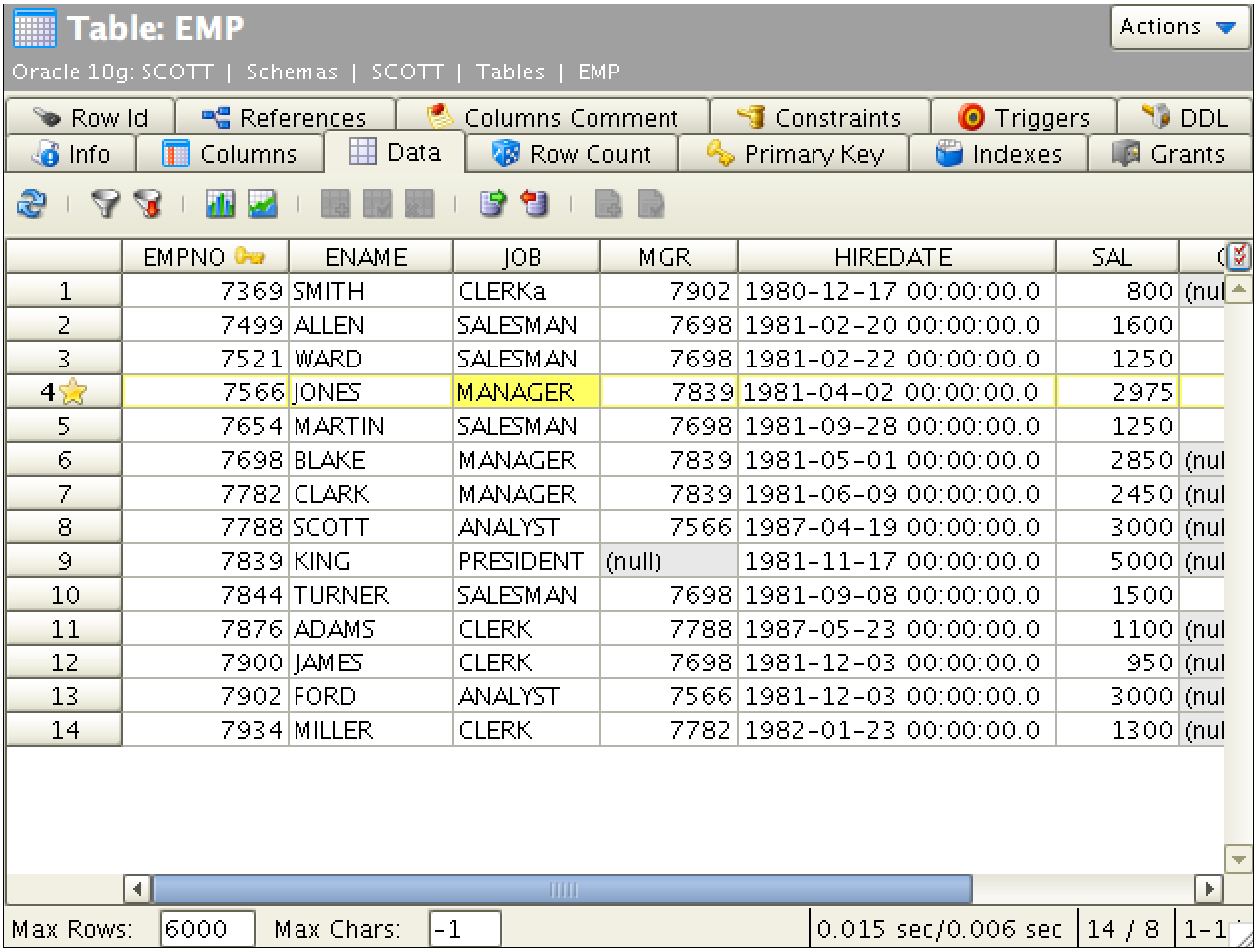
Shows the data for a table in a grid with editing features.

**Note:** information presented in the grid is obtained automatically by the viewer via a traditional **SELECT * FROM <schema>.table** statement i.e. the object type having this viewer defined must be able to support getting a result set via this SQL statement.

Here is a sample XML for the table-data viewer:

```
<DataView type="Data" label="Data" viewer="table-data"/>
```

Screenshot of the previous definition.



| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | ( |
|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERKa | 7902 | 1980-12-17 00:00:00.0 | 800 | (nul |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 00:00:00.0 | 1600 | |
| 3 | 7521 | WARD | SALESMAN | 7698 | 1981-02-22 00:00:00.0 | 1250 | |
| 4 ☆ | 7566 | JONES | MANAGER | 7839 | 1981-04-02 00:00:00.0 | 2975 | |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 00:00:00.0 | 1250 | |
| 6 | 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 00:00:00.0 | 2850 | (nul |
| 7 | 7782 | CLARK | MANAGER | 7839 | 1981-06-09 00:00:00.0 | 2450 | (nul |
| 8 | 7788 | SCOTT | ANALYST | 7566 | 1987-04-19 00:00:00.0 | 3000 | (nul |
| 9 | 7839 | KING | PRESIDENT | (null) | 1981-11-17 00:00:00.0 | 5000 | (nul |
| 10 | 7844 | TURNER | SALESMAN | 7698 | 1981-09-08 00:00:00.0 | 1500 | |
| 11 | 7876 | ADAMS | CLERK | 7788 | 1987-05-23 00:00:00.0 | 1100 | (nul |
| 12 | 7900 | JAMES | CLERK | 7698 | 1981-12-03 00:00:00.0 | 950 | (nul |
| 13 | 7902 | FORD | ANALYST | 7566 | 1981-12-03 00:00:00.0 | 3000 | (nul |
| 14 | 7934 | MILLER | CLERK | 7782 | 1982-01-23 00:00:00.0 | 1300 | (nul |

Max Rows: 6000 Max Chars: -1 0.015 sec/0.006 sec 14 / 8 1-1

**Figure: The table-data viewer**

**table-rowcount**

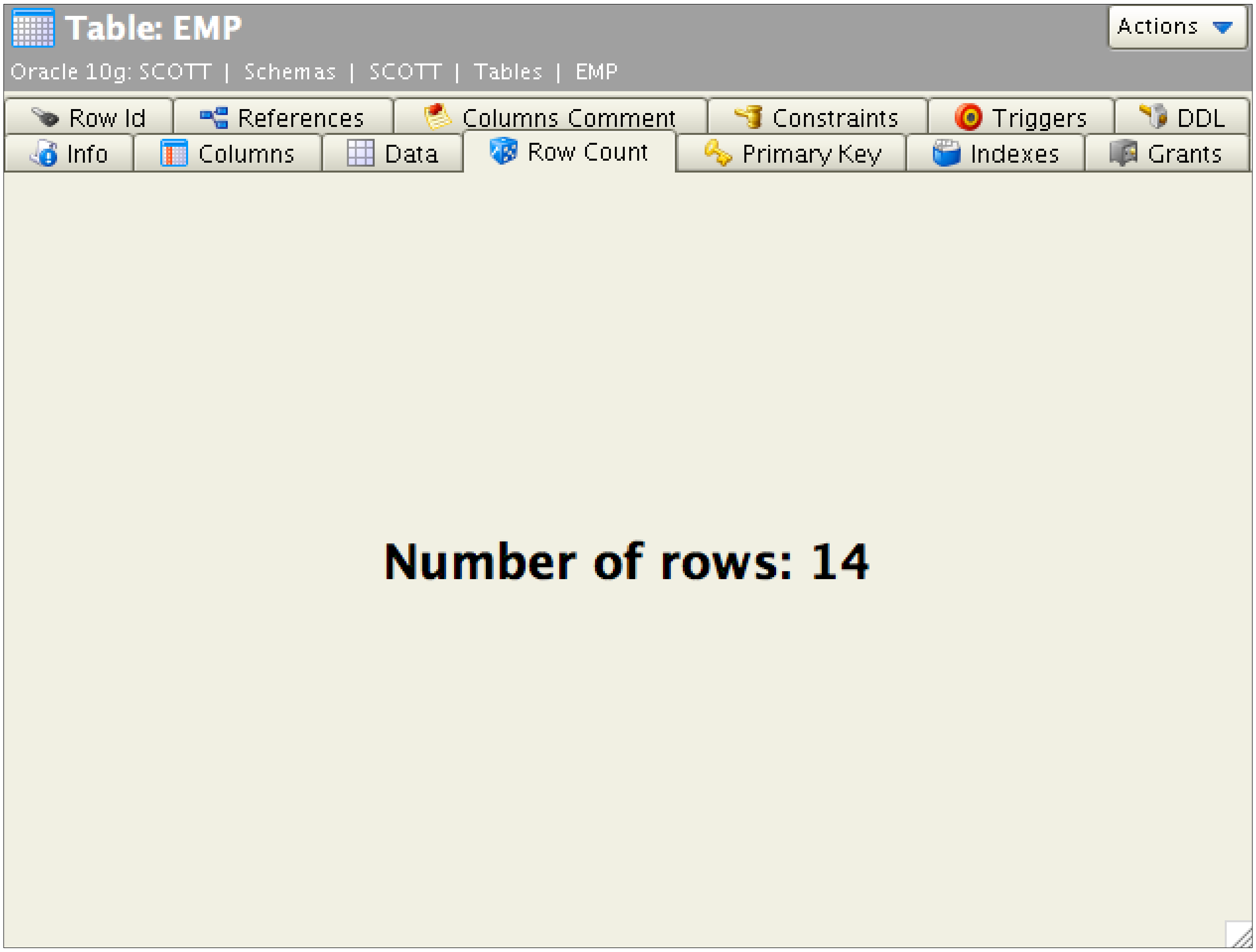
This viewer shows the row count for a (table) object.

**Note:** The row count is obtained automatically by the viewer via a traditional **SELECT COUNT(*) FROM <schema>.table** statement i.e. the object type having this viewer defined must be able to support getting a result set via this SQL statement.

Here is a sample XML for the table-rowcount viewer:

```
<DataView type="RowCount" label="Row Count" viewer="table-rowcount"/>
```

Screenshot of the previous definition.



Number of rows: 14

**Figure: The table- rowcount viewer**

### <Command>

Please read the Command section earlier as the capabilities here are the same.

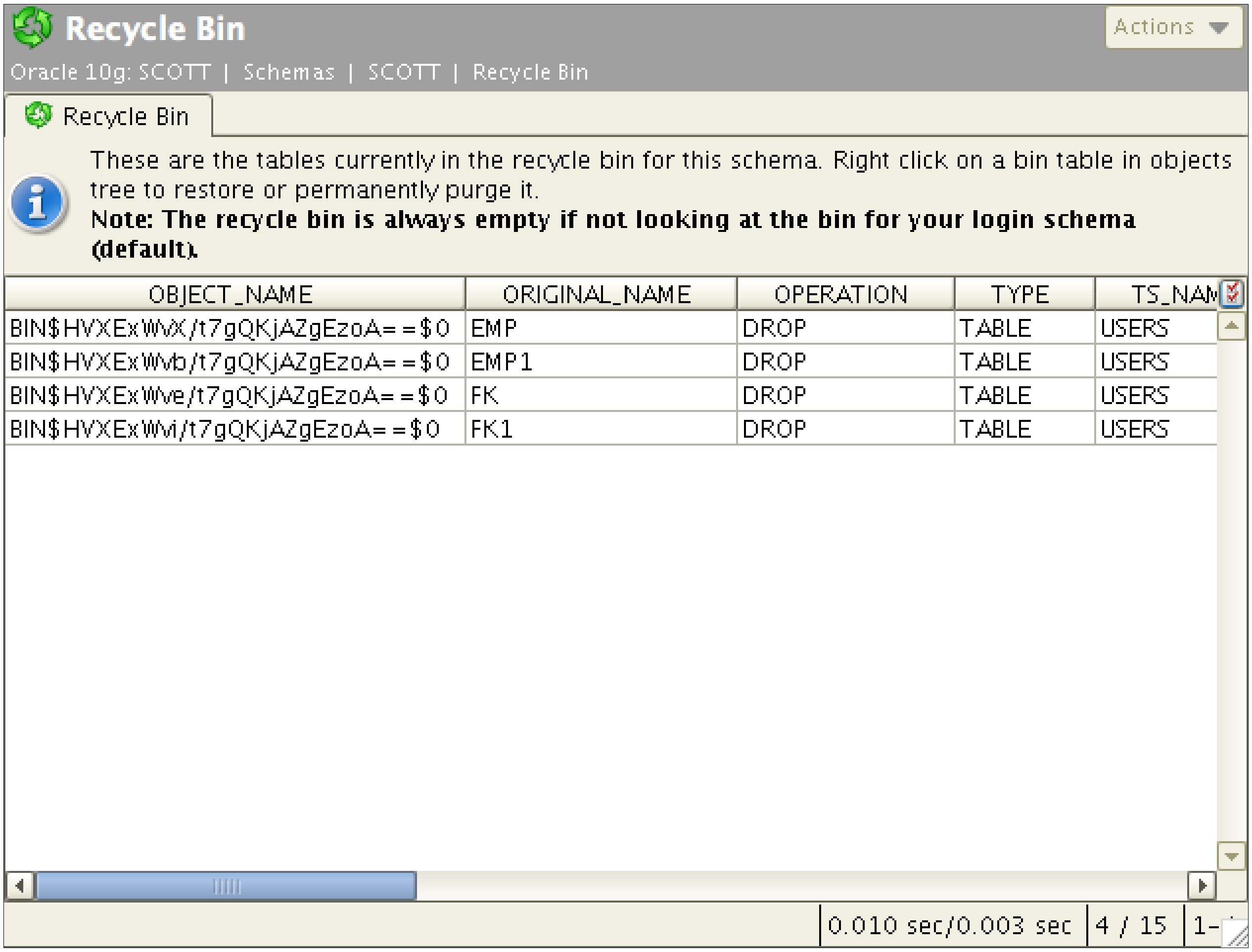
### <Message>

The **Message** element is very simple as it defines a message that will appear at the top of the viewer. The Message element is used to explain what is presented in the viewer. The text in the message may contain common HTML tags such as <b> (bold), <i> (italic), <br> (line break), etc.

Here is a sample XML using the Message element in a grid viewer:

```
<ObjectView type="RecycleBin">
  <DataView type="RecycleBin" label="Recycle Bin" viewer="grid">
    <Command idref="oracle.getRecycleBin">
      <Input name="schema" value="${schema}"/>
      <Input name="login_schema" value="${dbvis-
defaultCatalogOrSchema}"/>
    </Command>
    <Message>
      <![CDATA[
<html>
These are the tables currently in the recycle bin for this schema.
Right click on a bin
table in objects tree to restore or permanently purge it.<br>
<b>Note: The recycle bin is always empty if not looking at the bin for
your
login schema (default).</b>
</html>
      ]]>
    </Message>
  </DataView>
</ObjectView>
```

Screenshot of the previous definition.



Recycle Bin

Oracle 10g: SCOTT | Schemas | SCOTT | Recycle Bin

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_NAM |
|---|---|---|---|---|
| BIN$HVXExWvX/t7gQKjAZgEzoA==$0 | EMP | DROP | TABLE | USERS |
| BIN$HVXExWvb/t7gQKjAZgEzoA==$0 | EMP1 | DROP | TABLE | USERS |
| BIN$HVXExWve/t7gQKjAZgEzoA==$0 | FK | DROP | TABLE | USERS |
| BIN$HVXExWvi/t7gQKjAZgEzoA==$0 | FK1 | DROP | TABLE | USERS |

0.010 sec/0.003 sec | 4 / 15 | 1-

**Figure: The appearance of a Message in a viewer**

## Extending ObjectView

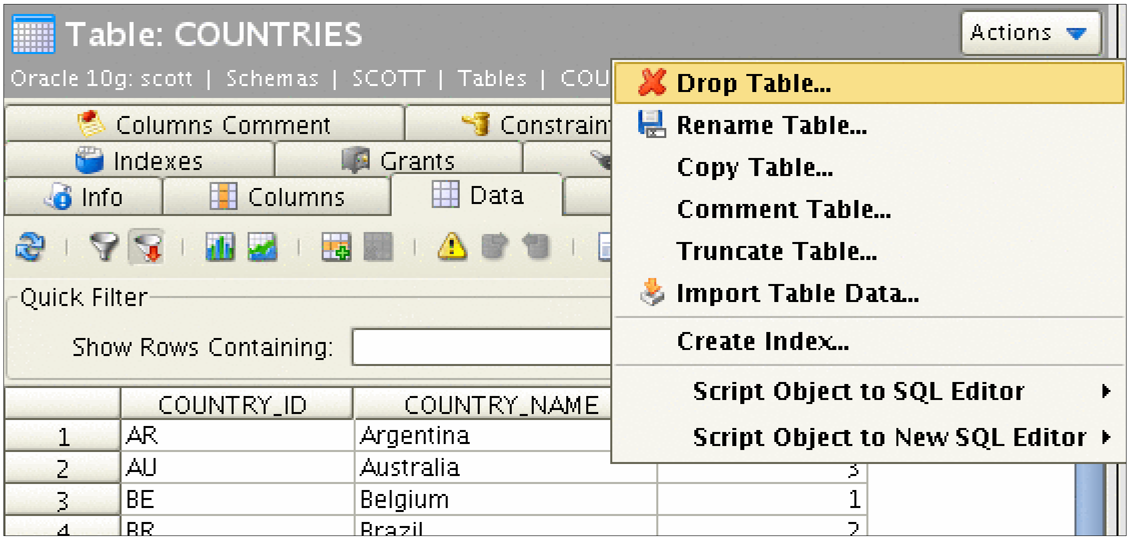
An existing **ObjectView** definition made in for example the generic- view.xml file can be extended in a database profile by using a few action attributes for each of the **DataView** elements. To accomplish extensions the object type specified in the **ObjectView** type attribute must match the type in the parent profile. Now you have the following options:

- **Adding a DataView**
  Simply add the **DataView** definition and it will be added to the current list of DataView definitions
- **Dropping an existing DataView**
  Add the `<DataView type="xxx" action="drop">` to drop the dataview type named "xxx"
- **Replacing a DataView**
  Just add the DataView with the exact same type as in the parent DataView. All the settings of the new DataView will replace the old one

# <ObjectsActionDef> - Definition of user actions

The previous sections have clarified how to define what objects should appear in the objects tree and what views will be displayed when selecting an object in the tree.
The **ObjectsActionDef** section in the profile defines what operations are available for the object types defined in the **ObjectTreeDef**. Object actions are very powerful as they offers an extensive number of features used to define actions for almost any type of object operation.

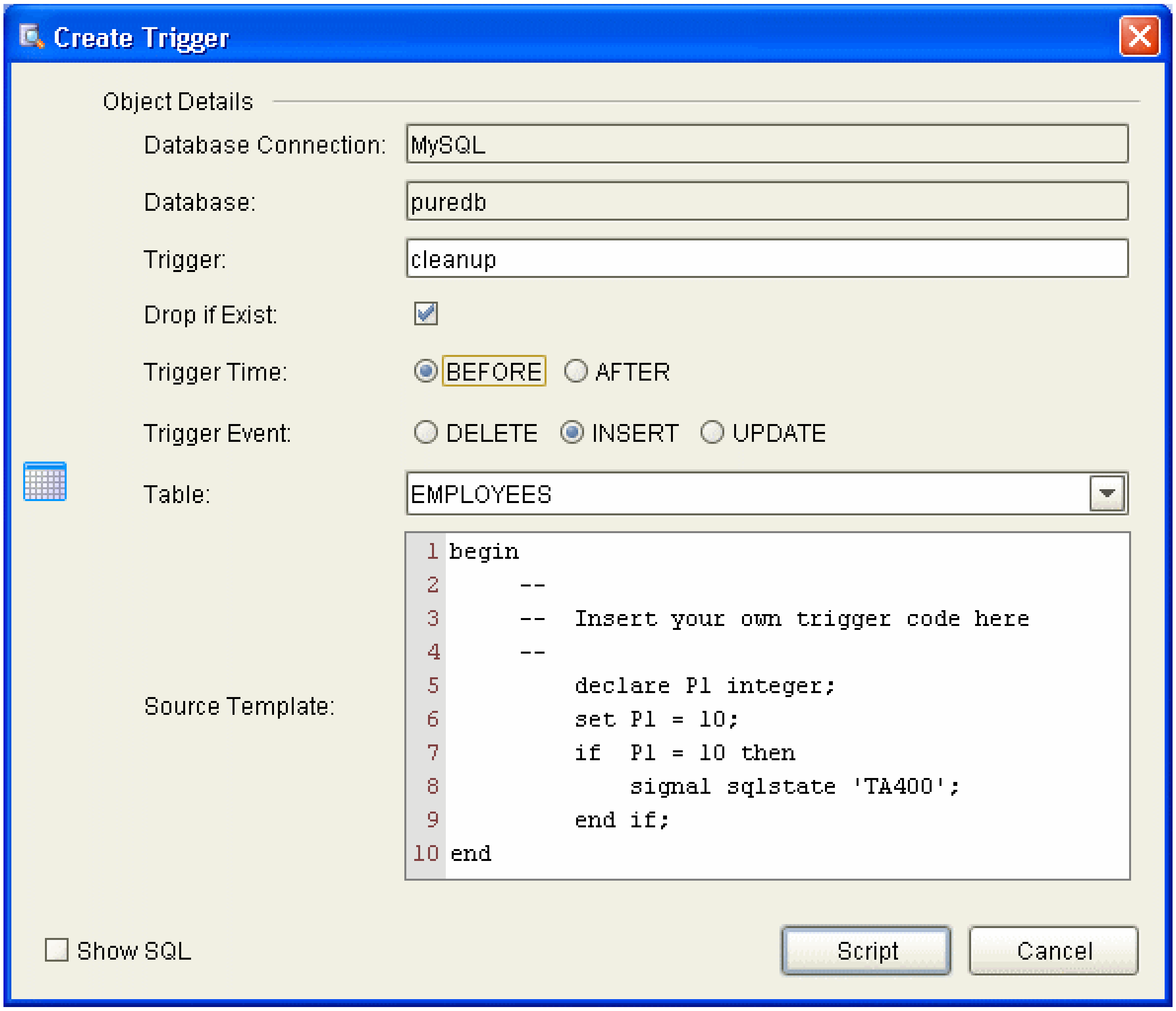
In DbVisualizer is the object type actions menu accessed via the right click menu in the objects tree or via the **Actions** button in the object view:



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

**Figure: The Actions menu for the selected object**

All of the operations for the selected **Table** object listed in the previous screenshot are expressed in the **ObjectsActionDef** section. The implementation for these actions are either expressed completely in XML via standard object actions or via specialized action handlers. (The API for action handlers is not yet documented). The following screenshot shows the dialog appearing when executing an action via the default action handler:



Object Details

Database Connection: MySQL

Database: puredb

Trigger: cleanup

Drop if Exist: ☑

Trigger Time: ● BEFORE ○ AFTER

Trigger Event: ○ DELETE ● INSERT ○ UPDATE

Table: EMPLOYEES

Source Template:

```
begin
     --
     --   Insert your own trigger code here
     --
          declare P1 integer;
          set P1 = 10;
          if  P1 = 10 then
              signal sqlstate 'TA400';
          end if;
end
```

☐ Show SQL

**Figure: The default action handler**

The first field in the dialog **Database Connection** is always present and shows the alias of the actual database connection. At the bottom there is a **Show SQL** control that when enabled will show 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 **Cancel** the action completely.

## Variables

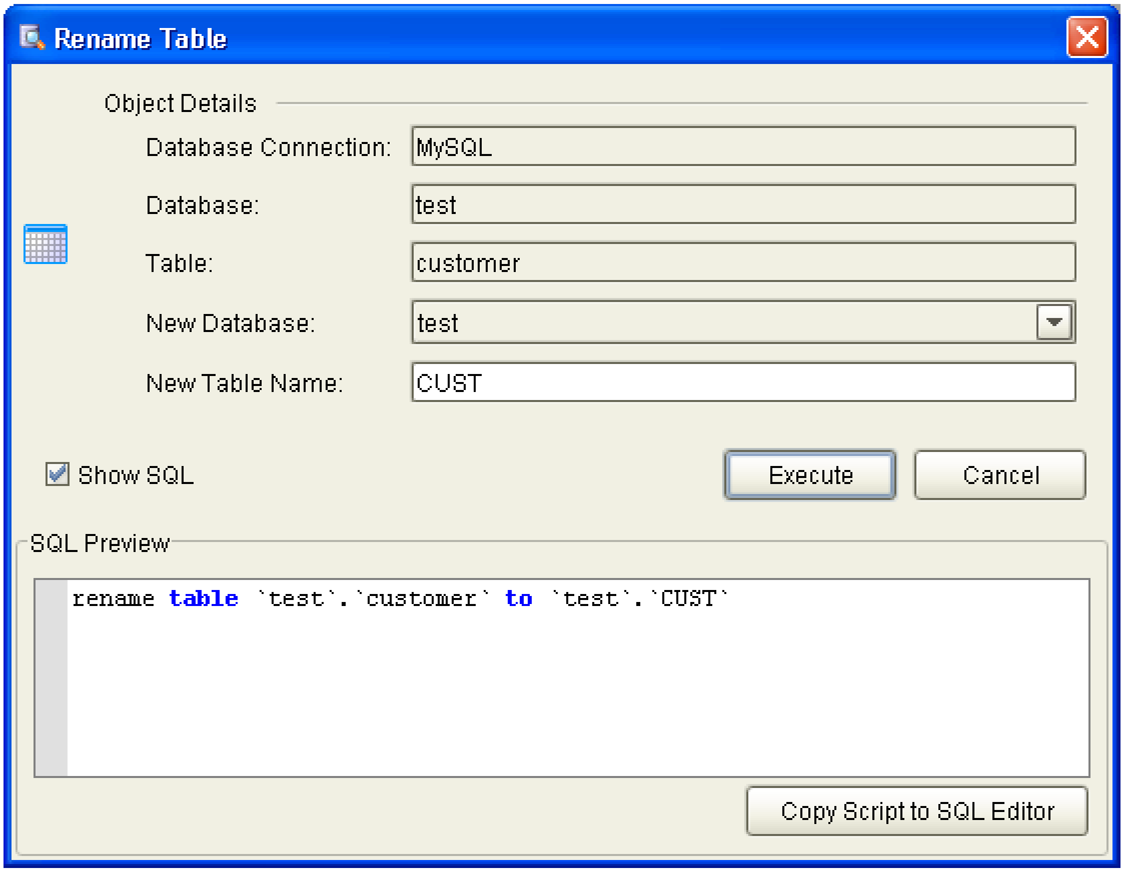
Variables are used to reference data for the object for which the action was launched and data for all its parent objects in the objects tree. Variables are also used to reference input data specified by the user in the actions dialog. Variables are typically used in the **Command**, **Confirm**, **Result** and **SetVar** elements.

Variables are specified in the following format:

```
${variableName}
```

Here follows an example for a **Rename Table** action. It first shows the name of the database connection (which is always present) along with information about the table being renamed. The last two input fields should be entered by the user and identify the new name of the table. The **New Database** control is a list from which the user should select the name of the new database. In the second **New Table Name** field should the new name of the table be entered.

If the **Show SQL** control is enabled you will see any edits in the dialog being directly reflected in the final SQL Preview.



Database Connection: MySQL
Database: test
Table: customer
New Database: test
New Table Name: CUST

☑ Show SQL

```
rename table `test`.`customer` to `test`.`CUST`
```

**Figure: The default action handler**

The complete action definition for the previous **Rename Table** action 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) in should be reloaded when the action is executed.

The next block of elements are **Input** fields defining the data for the action. As you can see in the sample there is a variable **${catalog}** in the **Default** element for the **Database** input and the **${objectname}** variable  in the **Default** element for the **Table** input. The values for these variables are fetched from the actual object in the objects tree. Briefly are values for variables retrieved by first checking if the variable is in the scope of the action dialog i.e. another input field, then the actual object for which the action was launched is checked, if the variable is not found the action then asks all of the parent objects until it reach the root object in the tree (Connections node). If a variable is not found its value will be **(null)**.

In the previous sample XML will the value of **${catalog}** be the name of the database in which the table object is stored. The **${objectname}** will present the current name of the table (these variables are described in the ObjectsTreeDef section).

The **New Database** input field is a list component and shows a list of databases based on the result set of the specified SQL command. The **Default** setting for the database will be the same as in which the table is currently stored based on the **${catalog}** variable.

The **New Table Name** input field is a simple text field in which the user may enter any text.

Both the **New Database** and **New Table Name** fields are editable and should be specified by the user. This data is then accessible via the variables specified in the **name** attribute, i.e. **newCatalog** and **newTable**.

The **Command** element should list the final SQL that will be executed by the action. The SQL is in this sample combined with static SQL along with variables.

## <ActionGroup>

The **ActionGroup** element is a container and groups **ActionGroup**, **Action** and **Separator** elements. It is used to define what actions should be present for a particular object type. It also define in what order the actions will appear in the menu and where any separators should appear. **ActionGroup** elements can be nested and will as so appear as sub menu(s).

```
<ActionGroup type="Table">
```

The attributes for an **ActionGroup** are:

- **type**
  this defines what object type the ActionGroup represents. This attribute is valid only for top level action groups. An example is the object of type **Table**, the corresponding **ActionGroup** will only be displayed when the actual object is a **Table**.
- **label**
  this attribute is required for nested action groups. This label will be displayed as the sub menu label for the nested action group. (The label attribute have no effect on top level action groups).

## <Action>

The action element defines the action.

```
<Action       id="oracle-table-drop"
           icon="remove"
          label="Drop Table..."
         reload="true"
           mode="execute"
resultsetaction="ask">
```

The attributes for an action are:

- **id**
  the id for the action. The recommended syntax for the id is "**profileName-objectType- someGoodActionName**"
- **icon**
  specifies an optional icon that will be displayed next to the label in the menu
- **label**
  the label for the action as it will appear in the menu in the action dialog
- **reload**
  specifies if the parent node (in the objects tree) should be reloaded after successful execution. This is recommended for actions that change the visual appearance of the object, such as remove, add or name change

- **mode** attribute can be set to any of these:
    - **execute**
      (default) - show the action dialog, process user input and execute the final SQL within the scope of the action dialog
- **script**
  show the action dialog, process user input and send the final SQL to the SQL Commander
- **script- immediate**
  will not show the action dialog but instead pass the final SQL directly to the SQL Commander
- **resultsetaction** attribute is only valid in combination with **mode="execute"**. It can be set to any of:
    - **ask**
      if the final SQL produced a result set then a query will ask whether the result set should be displayed in a window or copied as text to the SQL Commander
- **show**
  if the final SQL produced a result set then show it in a window
- **script**
  if the final SQL produced a result set then copy it to the SQL Commander.
- **class**
  used to launch a custom class. The **execute** attribute is obsolete if class is set
- **classargs**
  optional attribute used to specify arguments to the action hander defined by the class attribute

### <Input>

An **Input** element specifies the characteristics of a visible field component as it will appear in the actions dialog. The label attribute is recommended and is presented to the left of input field. If label is not specified then the input field will occupy the complete width of the action dialog. All input fields are editable by default and then requires the **name** attribute. This should specify the identity of the variable in which the user input will be stored.

A minimal definition of an input field is the following. It will show a read only text field control labeled **Table**.

```
<Input label="Size" editable="false"/>
```

If changing the input field to be editable we also need to supply the **name** attribute with 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 with the input field and is displayed as a tool tip when the user hovers the mouse pointer over it.

```
<Input label=Size" editable="true" name="theSize" tip="Please enter
```

```
the size of the new xxx"/>
```

Specifying the default value as a result from an SQL statement is a trivial task:

```
<Input label=Size" editable="true" name="theSize">
  <Default>
    <Command>
      <SQL>
select size from systables where tablename = '${objectname}'
      </SQL>
    </Command>
  </Default>
</Input>
```

Since **Default** here will execute a SQL statement it will automatically pick the value in the first row's first column and present it as the default. SQL may be specified in the **Default** and **Values** elements (also in **Labels** element for **list** and **radio** styles). An alternative of embedding the SQL as in the previous example is to refer a command via the standard **idref** attribute:

```
<Input label=Size" editable="true" name="theSize">
  <Default>
    <Command idref="getSize">
      <Input name"objectname" value="${objectname}"/>
    </Command>
  </Default>
</Input>
```

Instead of having duplicated SQLs in multiple actions consider replacing these with **Command** elements referred via the **idref** attribute.

Referencing commands in actions via the **idref** attribute is recommended when the same SQL is used in several actions. Use Input elements to pass parameters to the command.

The following sections presents the supported **styles** that can be used in the **Input** element.

#### text (single line)

The **text** style is used to present single line data in a text field.

```
<Input label="Enter your userid" name="userid" style="text">
  <Default>agneta</Default>
</Input>
```

- The optional **Default** element is used to define a default value for the field. Variables can be used here and **Command** (SQL) expressions
- A text input is editable by default. To make it read only just specify **editable="false"**

### text- editor (multi line)

A **text- editor** field is the same as the **text** style except that it presents a multi line field.

```
<Input label="Description" name="desc" style="text-editor"
editable="true" args="height=50"/>
```

- The `args="height=50"` attribute defines the height (in DLU) for the text- editor. The deafault height is 30 DLU's.

### number

A **number** style is the same as **text** except that it only accept number values.

```
<Input label="Size" name="size" style="number" editable="true"/>
```

### password

A **password** field is the same as **text** except that it masks the value as "*****".
Note that the password in visible in plain text in the SQL Preview.

```
<Input label="Password" name="pw" style="password" editable="true"/>
```

### list (large number of choices)

The list style displays a list of choices in a drop down component. The **list** can be editable meaning that the field showing the selection may be editable by the user. Here is a sample XML for the list style.

```
<Input label="Select index type" name="type" style="list">
  <Values>Pizza|Pasta|Burger</Values>
  <Default>Pasta</Default>
</Input>
```

The **Values** element should for static entries list all choices separated by a vertical bar (|) character. A **Default** value can either list the name of the default choice or the index number (first choice starts at 0). In the example above setting **Default** to **{2}** would set **Burger** to the default selection.

It is also possible to use the **Labels** element. If present then this should list all choices as they will appear in the actions dialog. Consider these as being the labels shown for the user while **Values** in this case should list the choices that will go into the final SQL via the variable. Here is an example:

```
<Input label="Select index type" name="type" style="list">
  <Values>Pizza|Pasta|Burger</Values>
  <Labels>Pizza the French style|Pasta Bolognese|Texas Burger</Labels>
  <Default>Pasta</Default>
```

```
</Input>
```

If the users selects **Texas Burger** then the value for variable **type** will be **Burger**.

The following shows how to use SQL to feed the list of values:

```
<Input label="New Database" name="newCatalog" style="list">
  <Values>
    <Command>
      <SQL>
        <![CDATA[
show databases
        ]]>
      </SQL>
    </Command>
  </Values>
  <Default>${catalog}</Default>
</Input>
```

Here a **Command** element is specified as sub element to **Values**. The result of the **show databases** SQL will be presented in the list component.

To make the list editable then specify the attribute **editable="true"**.

#### radio (limited number of choices)

The **radio** style displays a list of choices organized as button components. The only difference between the radio and list styles are:

- All choices for a radio style is displayed on the screen (better overview of choices but suitable only for a limited number of choices)
- The `args="vertical"` attribute can be specified for radio style and will present the radio choices vertically

See the list style for complete capabilities of the radio style.

#### check (true/false, on/off, selected/unselected)

The **check** style is suitable for yes/no, true/false, here/there types of input. Its enable state indicates that the **Value** for the input will be set in the final variable. If the check box is disabled then the variable is blank

```
<Input label="Cascade Constraints" name="cascade" style="check">
  <Values>compact</Values>
</Input>
```

- This will create a check component with the label **Cascade Constraints**
- Enabling the check box will set the value of the variable identified by **name** (cascade)

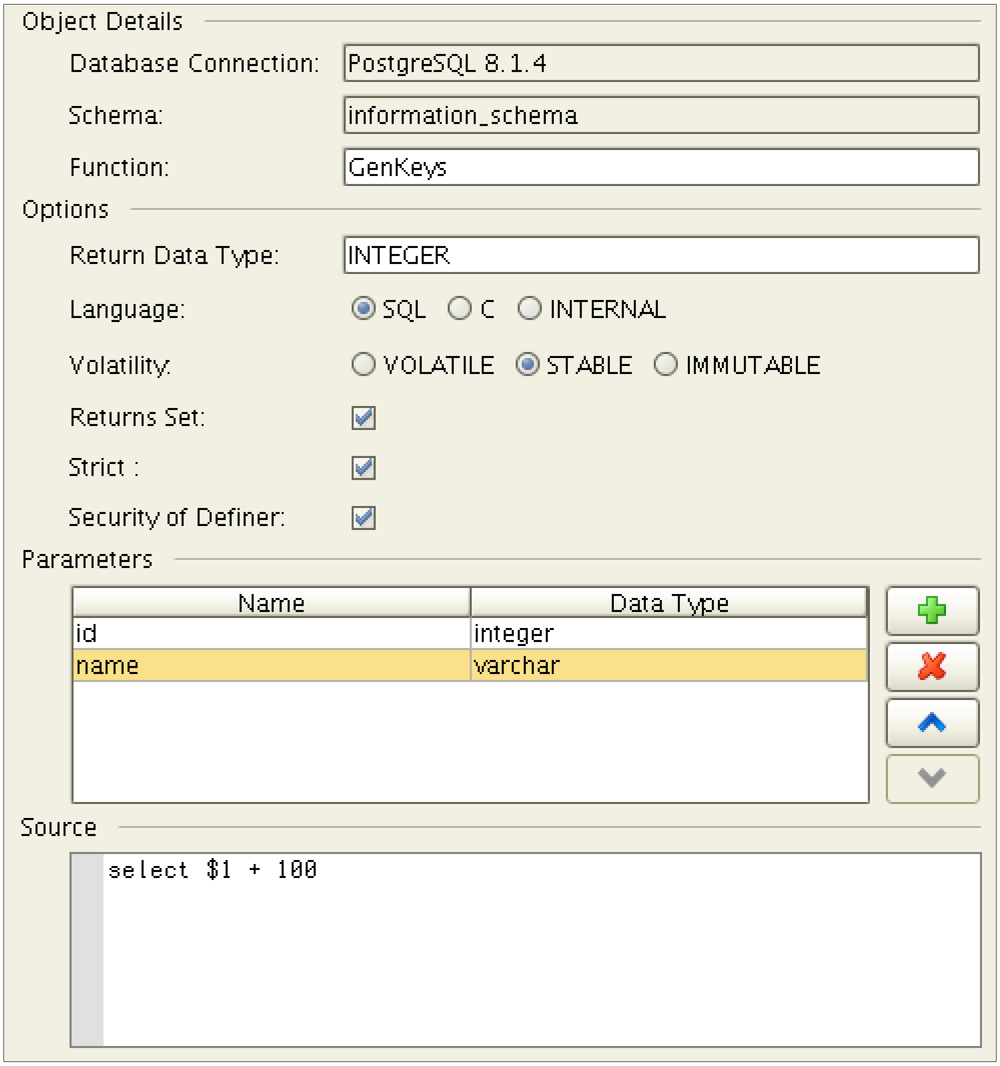
to the value of **Value**, which is **compact**.
- If the check box is unchecked then the variable will be blank

### separator (visual divider between input controls)

The **separator** style is not really an input element and is rather used to visually divide the fields in the in the actions dialog. If the **label** attribute is specified then it will be presented to the left of the separator line. If no label is specified only the separator is displayed.

```
<Input label="Content" style="separator"/>
```

The separator is a useful substitute for the standard label presented to the left of every input field. Here is a sample:



Object Details

Database Connection: PostgreSQL 8.1.4

Schema: information_schema

Function: GenKeys

Options

Return Data Type: INTEGER

Language: ● SQL ○ C ○ INTERNAL

Volatility: ○ VOLATILE ● STABLE ○ IMMUTABLE

Returns Set: ☑

Strict : ☑

Security of Definer: ☑

Parameters

| Name | Data Type |
|---|---|
| id | integer |
| name | varchar |

Source

```
select $1 + 100
```

**Figure: Sample showing separators and wide fields**

The previous figure shows the use of separators and two fields that extend to the full width of the action dialog. The separators for **Parameters** and **Source** are here used as an alternate label for the fields below them.

### procedure- params

The **procedure- params** style is a specialized style used primarily to manage parameters for procedure and function object types.

```
<Input name="parameters" style="procedure-params"
  args="procedure,defaultType=VARCHAR(20),parmOrder=direction|name|
type,directions=IN|INOUT|OUT/>
```

This example doesn't specify the label attribute as we want the field to extend the full width of the actions dialog. The **procedure- params** style use the **args** attribute extensively to costumize the appearance and function of the field. The following content in the args attribute is handled by the **procedure- params** style:

- **procedure**
  Defines what object type being handled. Can be one of **procedure** or **function**
- **defaultType=VARCHAR(20)**
  Defines the default data type when the user adds a new row in the list
- **parmOrder=direction|name|type|default**
  Specifies the order of the parameters for each row as they will appear in the final variable. The identifers above are static but can be ordered to comply with the wanted order of the parameters
- **directions=IN|INOUT|OUT**
  This lists the available values for the **Direction** column (if it's present)

The resulting parameter list is created automatically by the control and is available in the variable name specified in the example to be **parameters**.

### <SetVar>

The **SetVar** element is very powerful as it is used to do conditional processing and create new variables based on the content of other variables.

Consider an SQL statement for creating new users in the database:

```
create user 'user' identified by 'password'
```

In this case it is quite easy to map the **user** field to an **Input** element for the action since it is a required field. The question arise for **password** which is optional. The **identified by** clause should only be part of the final SQL if the password is entered by the user. The solution for this scenario is to use the **SetVar** element. Here is the complete action definition:

```
<Action id="mydb-user-create" label="Create User" reload="true"
icon="add">
  <Input label="Userid" name="userid" style="text"/>
  <Input label="Password" name="password" style="password"/>

  <SetVar name="_password" value='#password.equals("") ? "" : "
```

```
identified by \"" + #password + "\""'/>

  <Command>
    <SQL>
      <![CDATA[
create user ${userid} ${_password}
      ]]>
    </SQL>
  </Command>
</Action>
```

The **SetVar** element accepts two attributes:

- **name**
  should list the name of the new variable
- **value**
  this should contain the expression that will be evaluated. The expression is based on the OGNL toolkit provided by www.ognl.org. This is an expression library that mimics most of what is being supported by Java. Variables are referenced as **#variableName**.

The expression in the example above checks whether the **#password** variable is empty. If it is empty then a blank value is being set in 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}**.

**Note:** It is recommended that variables produced via **SetVar** elements are prefixed with an underline (_) to highlight were they come from.

### <Confirm>

The **Confirm** element will be displayed for the user when a request to **Execute** the action is made. If there are only read only input fields in the action this message is displayed in the body of the action dialog. The message is displayed in a confirmation dialog if there are editable fields.

Note that the message text can be composed of HTML tags such as <b>, <i>, <br>, etc.

```
<Confirm>Really drop table ${table}?</Confirm>
```

### <Result>

The **Result** element is optional and if specified it will show a dialog after successful execution.

**NOTE:** Result elements are currently not displayed in DbVisualizer. It is however recommend that you specify these as they will most probably 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 then a generic error dialog is displayed and the **Result** is not displayed.

### <Command>

The **Command** element specifies the SQL code that is executed by the action.

```
<Command>
  <SQL>
    <![CDATA[
drop table ${table} mode ${mode} including constraints
${includeconstraints}
    ]]>
  </SQL>
</Command>
```

## Conditional processing

Conditional processing briefly means that a profile can adjust its content based on conditions. A few examples:

- What version of the database it is
- The format of the database URL
- The client environment i.e Java versions, vendor, etc.
- User properties
- Database connection properties

Conditional processing is especially useful to adopt the profile for different versions of the database (and/or JDBC driver). Up to DbVisualizer 4.3.3 was a profile tested with a "minimum" of a database version. Accessing for example an Oracle 8 database using the Oracle profile supplied with DbVisualizer works most of the time but fails is some situations since it require at least Oracle 9. Another advantage with the conditional processing is to replace generic error messages with more user friendly messages.

Programmers familiar with **if**, **else if** and **else** will easily learn the conditional elements.

Depending on which of the two phases the conditions should be processed some restrictions and rules apply. Please read the following sections for more information.

### When are conditional expressions processed?

There are two phases when conditions are processed:

1. **Conditional processing when database connection is established**
   <If>, <ElseIf> and <Else> elements can be specified almost everywhere in the profile.
2. **Conditional processing during command execution**
   The <OnError> element is used to define a message that will appear in DbVisualizer if a command fail. To control what message should appear conditions are used.

DbVisualizer determines what **If** elements should be executed in what phase by the **type** attribute. If this attribute has the value **type="runtime"** then it will be processed in the second phase. If it is not specified or set to **type="load"** if will be processed in the first phase.

### Conditional processing when database connection is established

The following example shows the use of conditions that are processed during connect of the database connection.

```
<Command id="sybase-ase.getLogins">
  <If test="#DatabaseMetaData.getDatabaseMajorVersion() lte 8">
    <SQL>
      <![CDATA[
select name from master.dbo.syslogins
      ]]>
    </SQL>
  </If>
  <ElseIf test="#DatabaseMetaData.getDatabaseMajorVersion() eq 9">
    <SQL>
      <![CDATA[
select name, suid from master.dbo.syslogins
      ]]>
    </SQL>
  </ElseIf>
  <Else>
    <SQL>
      <![CDATA[
select name, suid, dbname from master.dbo.syslogins
      ]]>
    </SQL>
  </Else>
</Command>
```

The above briefly means that if the major version of the database being accessed is less then or equal to 8 the first SQL will be used. If the version is equal to 9 then the second SQL is used, the last SQL will be used for all other version. The test attribute may contain conditions that are AND'ed or OR'ed. Conditions can contain multiple evaluations combined using parenthesis.
The **If**, **ElseIf** and **Else** elements may be placed anywhere in the XML file.

Here is another example that controls whether certain nodes will appear in the database objects tree or not.

```
<!-- Getting Table Engines was added in MySQL 4.1 -->
<If test="(#dm.getDatabaseMajorVersion() eq 4 and
#dm.getDatabaseMinorVersion() gte 1)
          or #dm.getDatabaseMajorVersion() gte 5">
  <GroupNode type="TableEngines" label="Table Engines" isLeaf="true"/>

  <!-- "Errors" was added in MySQL 5 -->
  <If test="#dm.getDatabaseMajorVersion() gte 5">
    <GroupNode type="Errors" label="Errors" isLeaf="true"/>
  </If>
</If>
```

As you can see, this example contains nested uses of If.

### Conditional processing during command execution

Using conditional processing to evaluate any errors from a **Command** may be useful to re-phrase 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 will be processed for all commands. If it is part of a specific **Command** it will be processed only for that command.

## Current limitations

- The **SQL's** in the profile must be statements that DbVisualizer can execute with JDBC. It can not contain any executables, scripts or OS specific calls
- It is not possible to specify conditions or compound commands i.e. all needed to execute a command must be expressed in a single SQL statement.