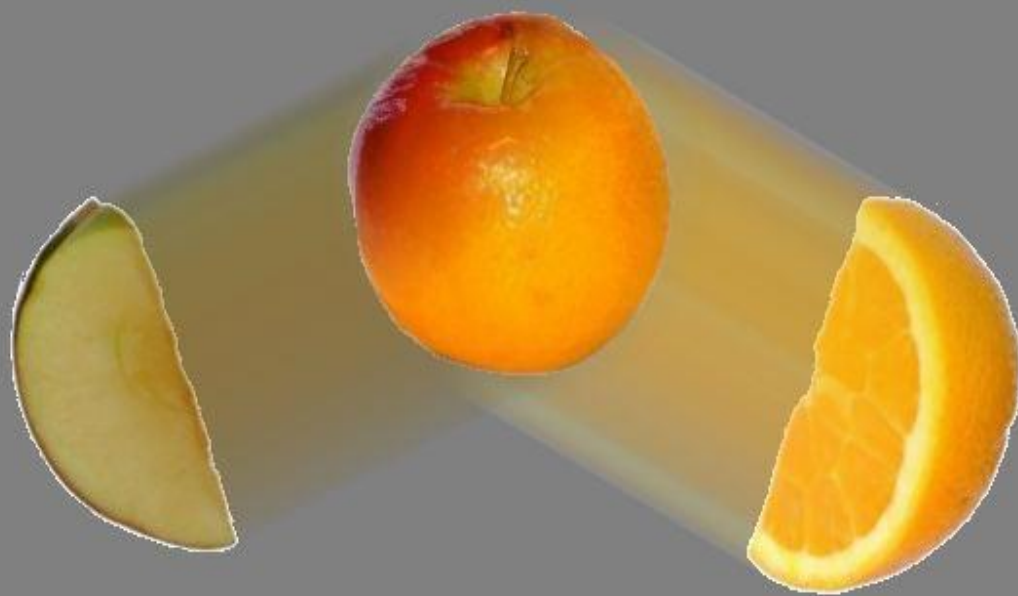




SQL DIFF FRAMEWORK

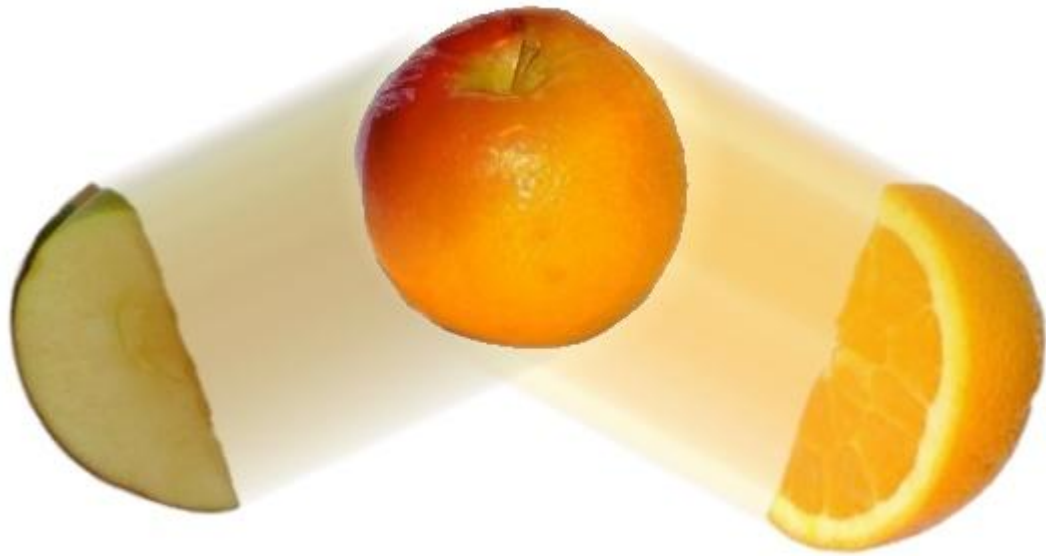


Now You *Can* Compare Apples & Oranges!

Michael Sorens

SqlDiffFramework

User Guide



Revision 1.0
April 30, 2010

Michael Sorens

SqlDiffFramework User Guide
Copyright © 2010 by Michael Sorens

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner. This publication, though related to the SqlDiffFramework software, is a separate work and is not published under the same license.

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, the author shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

Revision History

Version 1.0	2010.04.30	First Publication
-------------	------------	-------------------

SqlDiffFramework Software Published Under MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of the SqlDiffFramework software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents At A Glance

1	INTRODUCTION	1
1.1	FEATURE LIST	3
2	GETTING STARTED.....	5
2.1	INSTALLATION.....	5
2.2	ENTERPRISE CONFIGURATION	8
3	USING SQLDIFFFRAMEWORK	11
3.1	ANALYZING DIFFERENCES.....	12
3.2	ANALYZING ADDITIONS AND DELETIONS	14
3.3	INCLUDING AND EXCLUDING COLUMNS	17
3.4	SEARCHING AND FILTERING	21
3.5	SORTING AND TURBO-SORTING	30
3.6	WORKING WITH QUERIES AND SNAPSHOTS.....	38
3.7	WORKING WITH ODBC DATA SOURCES.....	44
3.8	WORKING IN TANDEM: DO TWO THINGS AT ONCE	48
3.9	WORKING WITH CONNECTIONS	51
3.10	WORKING WITH META-QUERIES	57
3.11	SYNTAX HIGHLIGHTING & KEYWORD COMPLETION.....	69
3.12	AUTOMATING DATA ANALYSIS.....	81
4	UNDERSTANDING SQLDIFFFRAMEWORK COMPONENTS	85
4.1	WORKING WITH THE APPLICATION.....	85
4.2	PERSISTENT SETTINGS	108
4.3	WORKING WITH THE EDITOR PANE.....	114
4.4	WORKING WITH THE QUERY EDITOR	123
4.5	WORKING WITH THE RESULTS GRID.....	133
5	APPENDICES.....	139
5.1	QUICK START.....	139
5.2	LIMITATIONS AND WORK TO BE DONE.....	141
5.3	MASTER FUNCTION REFERENCE.....	143
5.4	ON-SCREEN REFERENCE SHEETS	151
5.5	PREPARING AN ODBC CONNECTION	155
5.6	PROGRAMMING WITH SQLDIFFFRAMEWORK COMPONENTS	157

Detailed Table of Contents

1	INTRODUCTION	1
1.1	FEATURE LIST	3
2	GETTING STARTED.....	5
2.1	INSTALLATION.....	5
2.1.1	<i>Privacy Considerations</i>	6
2.1.2	<i>Component Options</i>	6
2.2	ENTERPRISE CONFIGURATION	8
2.2.1	<i>Provide Common Connections for Your Enterprise</i>	8
2.2.2	<i>Provide Update Notification within Your Enterprise</i>	9
2.2.3	<i>Track Users within Your Enterprise</i>	10
3	USING SQLDIFFFRAMEWORK	11
3.1	ANALYZING DIFFERENCES.....	12
3.2	ANALYZING ADDITIONS AND DELETIONS	14
3.3	INCLUDING AND EXCLUDING COLUMNS	17
3.3.1	<i>Create the Basic Query</i>	17
3.3.2	<i>Match Columns by Aliasing</i>	19
3.3.3	<i>Skip Columns With Matching Names</i>	20
3.4	SEARCHING AND FILTERING	21
3.4.1	<i>Searching the Query Editor</i>	21
3.4.2	<i>Searching the Results Grid</i>	25
3.4.3	<i>Filtering the Results Grid</i>	26
3.5	SORTING AND TURBO-SORTING	30
3.5.1	<i>Re-sorting Case Sensitive Results for Consistency</i>	30
3.5.2	<i>Re-sorting Static Snapshots</i>	30
3.5.3	<i>Re-sorting for NULL Consistency</i>	32
3.5.4	<i>Simple Sorting</i>	33
3.5.5	<i>Turbo-sorting</i>	34
3.6	WORKING WITH QUERIES AND SNAPSHOTS	38
3.6.1	<i>Organizing Your Files</i>	38
3.6.2	<i>Exporting to Excel</i>	39
3.6.3	<i>Saving Snapshots</i>	42
3.6.4	<i>Retrieving Snapshots</i>	42
3.7	WORKING WITH ODBC DATA SOURCES	44
3.7.1	<i>CSV Data Sources</i>	44
3.7.2	<i>Excel Data Sources</i>	45
3.7.3	<i>ODBC Example: Compare Two Excel Files</i>	46
3.8	WORKING IN TANDEM: DO TWO THINGS AT ONCE	48
3.8.1	<i>Tandem Operations</i>	48
3.8.1.1	<i>Load file</i>	48
3.8.1.2	<i>New file</i>	49
3.8.1.3	<i>Execute query</i>	49
3.8.1.4	<i>Toggle local / live mode</i>	49
3.8.1.5	<i>Enable auto-execute</i>	49

3.8.1.6	Filter result set	49
3.8.1.7	Scroll result set.....	49
3.8.1.8	Adjust grid row height	49
3.8.1.9	Adjust grid column widths.....	49
3.8.1.10	Enable auto-highlight	49
3.8.1.11	Enable case-sensitive syntax highlighting	50
3.8.1.12	Enable forcing keywords to upper or lower case	50
3.8.1.13	Enable auto-completion	50
3.8.1.14	Enable forcing auto-completed phrases to upper/lower/user case	50
3.8.1.15	Enable expanding tabs to spaces.....	50
3.8.1.16	Set number of spaces for tab expansion.....	50
3.8.1.17	Toggle tab and Control+tab actions.....	50
3.9	WORKING WITH CONNECTIONS	51
3.9.1	<i>Passwords</i>	55
3.9.1.1	Session and Persistent Passwords.....	55
3.9.1.2	Ephemeral Passwords.....	55
3.9.1.3	Determining When a Password is Required.....	56
3.10	WORKING WITH META-QUERIES	57
3.10.1	<i>Using Meta-Queries</i>	57
3.10.1.1	Invoking a Meta-Query.....	58
3.10.1.2	Manipulating a Meta-Query.....	59
3.10.1.3	The Meta-Query Library	60
3.10.2	<i>Meta-Query Templates</i>	61
3.10.2.1	Template Structure	61
3.10.2.2	Template Storage in the Library File	64
3.10.2.3	Customizing Meta-Queries.....	65
3.10.3	<i>Meta-Query Usage Summary</i>	68
3.11	SYNTAX HIGHLIGHTING & KEYWORD COMPLETION	69
3.11.1	<i>Syntax Highlighting</i>	69
3.11.1.1	Recognition	69
3.11.1.2	Stylization and Case Adjustment.....	69
3.11.2	<i>Keyword Completion</i>	70
3.11.2.1	Recognition	70
3.11.2.2	Stylization and Case Adjustment.....	70
3.11.3	<i>Highlighting and Keyword Completion Interactions</i>	71
3.11.4	<i>Macros</i>	71
3.11.4.1	Anonymous Macros vs. Named Macros.....	71
3.11.4.2	Macro Templates.....	72
3.11.5	<i>Customizing Syntax Highlighting and Macros</i>	76
3.11.5.1	Context File Structure.....	78
3.11.5.2	Controlling Whitespace in Macros	79
3.11.5.3	Highlighting Styles	80
3.12	AUTOMATING DATA ANALYSIS.....	81
4	UNDERSTANDING SQLDIFFFRAMEWORK COMPONENTS	85
4.1	WORKING WITH THE APPLICATION.....	85
4.1.1	<i>SqlDiffFramework Control ToolBar</i>	86
4.1.1.1	Next Difference.....	86
4.1.1.2	Previous Difference.....	86
4.1.1.3	First Difference	87
4.1.1.4	Current Difference	87

4.1.1.5	Last Difference.....	87
4.1.1.6	Set Current Difference	87
4.1.1.7	Toggle Turbo Sort Mode	87
4.1.1.8	Toggle Tandem Mode.....	87
4.1.1.9	Toggle Auto-Differencing Mode	87
4.1.1.10	Show Progress Monitor	88
4.1.2	<i>SqlDiffFramework Result ToolBar</i>	89
4.1.2.1	Difference Position Indicator	89
4.1.2.2	Match Quality (%).....	89
4.1.2.3	Count of Added Rows	89
4.1.2.4	Count of Missing Rows	89
4.1.2.5	Count of Changed Rows	90
4.1.2.6	Difference Navigation Shortcuts	90
4.1.3	<i>SqlDiffFramework Status Bar</i>	91
4.1.3.1	Status DropDown.....	91
4.1.3.2	Memory Indicator	91
4.1.3.3	Legend	91
4.1.4	<i>File Menu</i>	92
4.1.4.1	New workspace	92
4.1.4.2	New Query	92
4.1.4.3	Open Query.....	92
4.1.4.4	Save LEFT query, Save RIGHT query	93
4.1.4.5	Exit	93
4.1.5	<i>Edit Menu</i>	94
4.1.5.1	Find.....	94
4.1.5.2	Replace.....	94
4.1.5.3	Restore Settings... ..	94
4.1.5.4	Options... ..	95
4.1.6	<i>View Menu</i>	96
4.1.6.1	Next difference	96
4.1.6.2	Previous difference	96
4.1.6.3	First difference.....	96
4.1.6.4	Current difference	97
4.1.6.5	Last difference.....	97
4.1.6.6	Set current difference.....	97
4.1.6.7	Show progress monitor	97
4.1.6.8	Expand left pane.....	97
4.1.6.9	Expand right pane.....	98
	Show both panes	98
4.1.7	<i>Query Menu</i>	99
4.1.7.1	Execute query	99
4.1.7.2	Execute Batch.....	99
4.1.7.3	Meta-queries... ..	99
4.1.7.4	Edit Connections... ..	100
4.1.7.5	Mirror Query.....	100
4.1.8	<i>Help Menu</i>	102
4.1.8.1	Show Main Key Reference	102
4.1.8.2	Show Editor Pane Key Reference	102
4.1.8.3	Show Input Key Reference	102
4.1.8.4	Show Output Key Reference.....	103
4.1.8.5	About SqlDiffFramework.....	103

4.1.9	<i>Multiple Monitor Support</i>	104
4.1.9.1	Resolution and Orientation Impacts	104
4.1.9.2	Maximization Nuances	107
4.2	PERSISTENT SETTINGS	108
4.2.1	<i>Application-Scoped Global Values</i>	108
4.2.1.1	UpdateCatalogFile	108
4.2.1.2	UpdateRepository	108
4.2.2	<i>Global States</i>	109
4.2.2.1	WindowState.....	109
4.2.2.2	WindowPosition.....	109
4.2.2.3	UpdateChecked	109
4.2.2.4	NewVersion	109
4.2.2.5	UnreachableRepository	109
4.2.2.6	TandemButton_Checked.....	109
4.2.2.7	TurboSortButton_Checked.....	110
4.2.2.8	AutoDiffButton_Checked.....	110
4.2.3	<i>Editor Pane Shared States</i>	110
4.2.3.1	UseLocalDataButton_Checked	110
4.2.3.2	AutoHighlightButton_Checked	110
4.2.3.3	AutoExecuteButton_Checked.....	110
4.2.4	<i>Editor Pane Distinct States</i>	110
4.2.4.1	LeftSqlDirectory	110
4.2.4.2	RightSqlDirectory.....	110
4.2.4.3	LeftCsvDirectory	110
4.2.4.4	RightCsvDirectory	111
4.2.4.5	LeftConnectionName.....	111
4.2.4.6	RightConnectionName	111
4.2.5	<i>Database Connections</i>	111
4.2.5.1	ConnectionList.....	111
4.2.6	<i>Program Options</i>	111
4.2.6.1	UpdateCheckInterval.....	112
4.2.6.2	MaxColumnWidth	112
4.2.6.3	CommandTimeout	112
4.2.6.4	MaxHighlightedRowsPerChunk.....	113
4.2.6.5	MaxHighlightedRowsTotal.....	113
4.2.6.6	ShowElapsedTimes.....	113
4.3	WORKING WITH THE EDITOR PANE.....	114
4.3.1	<i>Top Control Bar Elements</i>	115
4.3.1.1	Auto-highlight on/off.....	115
4.3.1.2	Local/live mode	115
4.3.1.3	Auto-execute on query load.....	116
4.3.1.4	Execute or refresh.....	116
4.3.1.5	Meta-query dialog.....	116
4.3.1.6	Current file	117
4.3.1.7	File picker	117
4.3.2	<i>Bottom Control Bar Elements</i>	118
4.3.2.1	Grid navigation	118
4.3.2.2	Current DB Server	119
4.3.2.3	Connection Selector	119
4.3.2.4	New query.....	120
4.3.2.5	Save query	121
4.3.2.6	Save grid results	122

4.3.2.7	Execution time	122
4.3.2.8	Revert to saved query	122
4.4	WORKING WITH THE QUERY EDITOR	123
4.4.1	<i>Key Features</i>	123
4.4.1.1	Dialect-specific syntax highlighting.....	123
4.4.1.2	Recognizes SQL Server, Oracle, and MySql dialects out-of-the-box	124
4.4.1.3	Enable or disable automatic highlighting	124
4.4.1.4	Hide or display highlighting	124
4.4.1.5	Automatic or on-demand highlighting	124
4.4.1.6	Distinguishes between end-of-line comments and block comments	124
4.4.1.7	Multiple delimiters assignable for strings and variables	124
4.4.1.8	Instant conversion of keywords to uppercase or to lowercase	125
4.4.1.9	Instant conversion of variables to uppercase or to lowercase	125
4.4.1.10	Differentiate groups of keywords.....	125
4.4.1.11	Make keyword recognition case sensitive or insensitive	125
4.4.1.12	Enable or disable keyword completion.....	125
4.4.1.13	Invoke keyword completion.....	125
4.4.1.14	Instant case conversion of auto-completed phrases	126
4.4.1.15	Macros speed your typing	126
4.4.1.16	Macros may be static or dynamic	126
4.4.1.17	Navigate among place holders in a macro	126
4.4.1.18	Clean up unneeded place holders.....	127
4.4.1.19	Delete range by line boundaries	127
4.4.1.20	Comment or uncomment a region.....	127
4.4.1.21	Increase or decrease the font size.....	127
4.4.1.22	Increase or decrease the indent of a region.....	127
4.4.1.23	Customize the tab key to insert tab characters or spaces.....	128
4.4.1.24	Customize the tab key to act within the editor or not	128
4.4.1.25	Set the number of spaces inserted by the Tab key	129
4.4.1.26	Search by text, wildcard, or regular expression	129
4.4.2	<i>Types of Highlighting</i>	131
4.4.2.1	Keywords	131
4.4.2.2	Variables.....	131
4.4.2.3	Comments	131
4.4.2.4	Strings	131
4.4.2.5	Numbers.....	131
4.4.2.6	Place holders.....	132
4.5	WORKING WITH THE RESULTS GRID.....	133
4.5.1	<i>Key Features</i>	134
4.5.1.1	Filtering data.....	134
4.5.1.2	Reveal data type of a column.....	134
4.5.1.3	Adjusting column widths to fit data or headers	135
4.5.1.4	Increase or decrease row heights	136
4.5.1.5	Export to Excel.....	136
4.5.1.6	Change the format of all displayed date/time values.....	136
4.5.1.7	Reveal column numbers.....	136
4.5.1.8	Show or hide columns	137
4.5.1.9	Sort a column from the keyboard.....	137
4.5.1.10	Quick Find	138
4.5.1.11	Copy Field Names	138
5	APPENDICES.....	139

5.1	QUICK START.....	139
5.1.1	<i>Setup</i>	139
5.1.2	<i>Main Window</i>	139
5.2	LIMITATIONS AND WORK TO BE DONE.....	141
5.2.1	<i>False Positives from One Engine</i>	141
5.2.2	<i>Multi-Threaded Implementation</i>	141
5.2.3	<i>Not Instrumented for Internationalization</i>	141
5.2.4	<i>Binary Field Types not Supported</i>	141
5.2.5	<i>Data Must Fit in Memory</i>	141
5.2.6	<i>More Settings Should be Persistent</i>	142
5.3	MASTER FUNCTION REFERENCE.....	143
5.4	ON-SCREEN REFERENCE SHEETS	151
5.4.1	<i>Main Application Reference</i>	151
5.4.2	<i>Editor Pane Reference</i>	152
5.4.3	<i>Query Editor Reference</i>	153
5.4.4	<i>Results Grid Reference</i>	154
5.5	PREPARING AN ODBC CONNECTION	155
5.6	PROGRAMMING WITH SQLDIFFFRAMEWORK COMPONENTS	157
5.6.1	<i>Key URLs</i>	157
5.6.2	<i>Libraries</i>	158
5.6.3	<i>Component Descriptions</i>	160
5.6.3.1	<i>SqlEditor</i>	160
5.6.3.2	<i>Query Picker</i>	161
5.6.3.3	<i>ExtendedDataGridView</i>	161
5.6.3.4	<i>ChameleonRichTextBox</i>	162
5.6.3.5	<i>MultiConnectionStringManager</i>	162
5.6.3.6	<i>StructuredTraceSource</i>	163
5.6.3.7	<i>FileMaskControl</i>	163
5.6.3.8	<i>DisplayCommandsForm</i>	164
5.6.3.9	<i>ShadowTipForm</i>	164
5.6.3.10	<i>ResourceMgr</i>	164
5.6.3.11	<i>UsageTracker</i>	165
5.6.3.12	<i>UpdateCheck</i>	165
5.6.3.13	<i>WindowRestorer</i>	165
5.6.3.14	<i>MemoryGauge</i>	165
5.6.3.15	<i>MenuBuilder</i>	165
5.6.3.16	<i>ToolStripDropDownManager</i>	166

Table of Figures

Figure 2-1 Installer Snapshot.....	5
Figure 2-2 Accessing Boot Configuration Options.....	7
Figure 3-1 Main Window	11
Figure 3-2 A Differential Comparison	12
Figure 3-3 A Starting Query	18
Figure 3-4 Matching Columns.....	19
Figure 3-5 Skipping Columns.....	20
Figure 3-6 Query Editor Find and Replace	22
Figure 3-7 Result Grid Quick Find	25
Figure 3-8 Filtering Sequence	28
Figure 3-9 Snapshot Data in Native Order.....	31
Figure 3-10 Snapshot Data With Turbo-sort Applied.....	32
Figure 3-11 Native Sort Order of NULL	32
Figure 3-12 Turbo-sorting To Compensate for NULLs	33
Figure 3-13 Directory Structure	38
Figure 3-14 Ensuring Excel Interoperability	41
Figure 3-15 ODBC Example with Excel	47
Figure 3-16 Connection Editor Concepts.....	51
Figure 3-17 Connection Editor Examples	53
Figure 3-18 Login Dialog	55
Figure 3-19 Meta-Query Selector	58
Figure 3-20 Meta-Query Schema	65
Figure 3-21 Meta-Query Processing	67
Figure 3-22 Keyword Completion List	72
Figure 3-23 Instantiating a Macro.....	73
Figure 3-24 Context Definition Schema	77
Figure 3-25 Keyword Groups.....	79

Figure 3-26 Batch Execution Palette	82
Figure 3-27 Batch Execution Results	83
Figure 3-28 Batch Results Including Orphans	84
Figure 4-1 Components of SqlDiffFramework	85
Figure 4-2 Exploded View of Application's Control ToolBar.....	86
Figure 4-3 Progress Monitor.....	88
Figure 4-4 Exploded View of Application's Result ToolBar.....	89
Figure 4-5 Restore Settings Confirmation Dialog.....	95
Figure 4-6 Mirror Dialog.....	100
Figure 4-7 The About Box	103
Figure 4-8 Maximize to Multiple Monitors.....	104
Figure 4-9 Spanning Multiple Monitors with Varying Resolutions	105
Figure 4-10 Windows Display Control Panel	106
Figure 4-11 Options Dialogs.....	112
Figure 4-12 Editor Pane.....	114
Figure 4-13 Exploded View of Editor Pane's Top Control Bar.....	115
Figure 4-14 Exploded View of Editor Pane's Bottom Control Bar.....	118
Figure 4-15 Auto-Query Generation	121
Figure 4-16 Query Editor	123
Figure 4-17 Query Editor Context Menus	130
Figure 4-18 Results Grid.....	133
Figure 4-19 Revealing Column Information	134
Figure 4-20 Adjusting Column Widths.....	135
Figure 4-21 Column Shortcuts.....	137
Figure 5-1 Defining an ODBC Connection.....	155
Figure 5-2 Library Dependencies.....	159

Table of Tables



Table 3-1 Commonly Used Regular Expression Tokens	23
Table 3-2 Regular Expression Samples	23
Table 3-3 SQL Dialect for Grid Filtering	27
Table 3-4 Syntactical Variations Acceptable to Turbo-Sorting	36
Table 3-5 Comparing Live and Local Data Sources	43
Table 3-6 Standard Meta-Queries	61
Table 3-7 Meta-Query Usage Summary	68
Table 3-8 Sample Sequence Filling in a Query Template	75
Table 4-1 Sample Distribution of Chunks	90
Table 4-2 Actions of Maximize vs. Control+Maximize	107
Table 5-1 Key URLs for SqlDiffFramework Reference	157
Table 5-2 Libraries Used in SqlDiffFramework	158

Conventions, Credits, and Prerequisites

Conventions:

<i>Italic</i>	Indicates new terms, article titles, and email addresses.
Constant width	Filenames, directories.
Text	Hyperlinks.
<code>Shaded</code>	Code fragments.
Heavy Bold	Program components.
Text	On-screen buttons.
Text	Keyboard characters. Example: Control + Tab
Text	Standard menu.
Text	Context menu.

Credits:

- Basic apple and orange images (used to make the blend on the cover) come from <http://www.freeclipartpictures.com/0frames/food.html>
- Binary on front cover comes from Stock.XCHNG (<http://www.sxc.hu/>).
- Tip indicator () comes from <http://www.everystockphoto.com/photo.php?imageId=273127> and is used in accordance with their license terms.
- Difference navigation buttons () come from the WinMerge open source project (<http://winmerge.org/>) by permission from the author. (WinMerge is licensed under GPL: <http://www.opensource.org/licenses/gpl-2.0.php>)
- Included third-party libraries are described in section 5.6.2.
- SqlDiffFramework was created with Visual Studio 2008 and assistance from WinMerge, Subversion, LINQPad, NUnit, Ant, and other assorted development tools.
- The SqlDiffFramework installer was created with InnoSetup (<http://www.jrsoftware.org/isinfo.php>).
- This manual was created with Microsoft Word, Paint.NET, and SnagIt.

Prerequisites:

.NET Framework version 2.0 or later (According to Wikipedia this means you can effectively run on Windows 2000 or later; see http://en.wikipedia.org/wiki/.NET_Framework.)

Comments and Questions:

Please address your questions to *SqlDiffFramework at gmail.com*.

For more information on SqlDiffFramework, see <http://SqlDiffFramework.codeplex.com/>.

For more information on components and building blocks used to build SqlDiffFramework, see <http://cleancode.sourceforge.net/>.

About the Author



Michael Sorens is a freelance software engineer, spreading the seeds of good design wherever possible. He has more than two dozen articles published on Simple-Talk.com and DevX.com, has contributed to two books, been awarded a patent, taught computer science, and manages an open-source website (cleancode.sourceforge.net). With BS and MS degrees in computer science and engineering, he has worked at both Fortune 500 firms and startups using C#, SQL, XML, XSL, Java, Perl, C, Lisp, PostScript, and others.

His diverse assortment of technologies and products has included:

- color laser printer firmware at Xerox;
- websites at HP and OzEmail;
- content management systems at Tektronix;
- test automation at HP and AdvantageIQ;
- integrated circuit design using LISP(!) at HP;
- specialized network monitoring software at Itron.

But his favorite project was designing and implementing the world's smallest word processor, where the medium was silicon, the printer "head" was a laser, and the Declaration of Independence could literally fit on the head of a pin. (This advanced lasography system was *actually* designed for rapid prototyping of custom integrated circuits.)

I want to hear from you!

Does SqlDiffFramework make you more productive? Why or why not?

Is this user guide effective for you? Why or why not?

Please send your feedback to *SqlDiffFramework at gmail.com*.

1 Introduction



See Quick Start
on Page 139

Why SqlDiffFramework?

At any single moment, there are many companies in the midst of an upgrade, a migration, or a conversion of one software system to another. Most any such system has a database at its heart. The naïve approach of deciding on a launch date and switching over entirely from system A to system B is (one hopes!) a dying phenomenon. The much more prudent approach is to run the two systems in parallel during a specified transition period. This could certainly double the workload (for data entry folks, for DBAs, for analysts, etc.) but for something that can make or break your organization, it is well worth it.

So how do you judge whether the two systems are producing identical results? You can run reports on each and compare them, but chances are your reports will be formatted differently and contain different data; that is, after all, why you are migrating to a new system, right? Even if your reports do happen to contain all the same data, do you have reports that exhaustively cover all the data in your database? SqlDiffFramework gives you a much finer probe of your data, allowing you to cover all the data and to match arbitrarily different data between systems.

SqlDiffFramework is unique among database comparison utilities. The typical comparison utility lets you compare the structure (schema) of your database; others let you compare your data (e. g. SQL Compare and SQL Data Compare products from Redgate). But even Redgate's premium products are only for a single database type, SQL Server.

SqlDiffFramework is a data comparison tool that spans database types: it supports SQL Server, Oracle, and MySQL. It also supports ODBC data connections, allowing you to compare, for example, an Oracle table with an Excel spreadsheet, or compare a SQL Server table with a CSV file.

SqlDiffFramework takes a novel approach to comparing dissimilar databases: it is a differencing application that uses your custom-written SQL rules to compare apples and oranges. For users not familiar with the English idiom of "That's like apples and oranges", the classic phrase sums up the difficulty of comparing objects that are dissimilar—hence the blended apple/orange fruit on the cover! As a curious aside, Serbians refer to this challenge as "old ladies and frogs", according to National Geographic, March 2010, page 22. A simple example will illustrate the simplicity of monitoring diverse systems. Say, for

example, that your environment consists of a SQL Server installation and an Oracle installation. While they drive different front-ends there is a core set of tables that must be consistently maintained between the two databases. Your customer name table on both systems has `CustomerId` and `CustomerName` fields, among a few others. The list of ids and names should always match between the two systems as one is set up with appropriate triggers to copy any changes. SqlDiffFramework lets you verify that state on demand: load the same query into each of the dual edit panes in the application:

```
SELECT CustomerId, CustomerName
FROM dbo.CustomerNames
ORDER BY CustomerId
```

Run the query against both databases to generate dual result sets. The auto-differencing engine examines both result sets and visually highlights any differences, additions, or omissions between the two, allowing you to easily navigate among them just like any of a number of visual differencing tools for conventional text files.

SqlDiffFramework matches columns by name. This simple fact has several ramifications. It means first that you could include additional columns in your result set that will not be compared (but you want to see them for reference). More importantly, it means that any columns that you want to compare must either have the same name or be aliased to the same name. If one of the two databases used `FullName` instead of `CustomerName` due to legacy reasons, the second query need only be changed to:

```
SELECT CustomerId, FullName as CustomerName
FROM dbo.CustomerNames
ORDER BY CustomerId
```

Your goal, then, is to generate a result set with an appropriate `SELECT` statement that allows you to convert both your apple and your orange into a durian (substitute your own preference for a neutral fruit here). In practice, I have used some queries that are as simple as these for some tables. But other tables have required elaborate joins, filters, temporary tables, aggregated subqueries, and more to yield a suitable result set for comparison. Sharp SQL skills will definitely aid you on your journey.

After the next chapter on installation, the first part of this manual is a user guide, describing the concepts and techniques for using SqlDiffFramework. The remainder of the manual is a reference, providing details on every button, menu item, and command available to effectively use the application.

1.1 Feature List

- Provides two completely separate SQL editor windows, each of which may point to the same database or different databases. With a single click you may link them together so they will load queries, load results, scroll, and filter in synchronization. Click again to unlink them. Press an override key to temporarily reverse the link/unlink state for a single operation.
- Generates a typical visual difference display common to many diff programs for text files, only this operates on database result sets. Row differences are denoted similar to textual diff programs, showing additions, deletions and changes. The specific columns that differ within those rows are highlighted further, pinpointing the differences.
- To perform a difference operation, you simply need to specify two queries that provide matching column names. All matching columns will be used in the comparison; unmatched columns will be ignored. To match column names, you use the standard SQL mechanism to alias a column, as in `select name as CompanyName from . . .`. To skip some columns that match, simply add the column to an ignore list that resides with one or both of the queries as a SQL comment set off with special delimiters.
- Select from a choice of three difference engines that provide varying levels of performance and accuracy. Simply selecting a different engine will re-evaluate all the current data (without refetching anything from the source) and update the visual differencing on screen.
- Apply a filter defined by a general SQL predicate (e.g. `ID < 1000 and name like 'Sam%'`) and the filtered results are re-evaluated to update the visual differencing on screen.
- Search for values in the result set or search for text in the query.
- Load the result set from either a live database or from an archived, local copy (in CSV format).
- Operational configuration lets you tailor the application to your preferences, including: automatic differencing on/off when needed, automatic query execution upon query load on/off, results from live/local data, automatic syntax highlighting on/off, tandem/synchronized operation on/off.
- Connection editor lets you define a set of database connections to SQL Server, Oracle, or any ODBC data source. Passwords, if you choose to store them, are stored with secure encryption. The connection set is easily exported or imported, allowing you to copy it from one installation to another.

- Meta-queries provide a convenient mnemonic aid allowing you to search for tables or fields with wildcards, find non-empty tables, identify database version, and list other useful queries, based on the database type.
- Controls to navigate the differences allow you to scroll to next or previous difference, first or last difference, to return to the current difference if you have scrolled elsewhere, and to set the current difference to the current location.
- Typically you start with a query and obtain results from live databases or archived results. But you could also start with just data, then automatically generate a query from the data, and finally customize the queries to match columns, reorder rows, etc.
- You have sorting flexibility to manipulate the results to your liking: sort result sets on multiple columns via the standard SQL `order by` clause in your query, or sort on a single column by clicking a column header with the mouse or by single keystroke `(Control+Shift+digit)` where *n* may select any of the first nine columns.
- You have display flexibility to hide or show any number of columns via context menu; any of the first nine may be hidden/shown by single keystroke `(Control+Alt+digit)`.
- Adjust cell sizes: Adjust the column widths to fit either the header text or the data via keystroke or context menu. Increase or decrease row heights via keystroke.
- Automate data analysis by running any number of queries with a single click, generating a summary result table.



2 Getting Started

2.1 Installation

SqlDiffFramework includes a full-featured installer, providing a convenient and fast way to set up an entry in your start menu, on your quick launch bar, on your desktop – or none of these. You select which shortcuts to install. At the same time, the installation is lightweight: it has no hooks into your system other than a couple registry items whose sole purpose is to allow a quick uninstall at a later date.

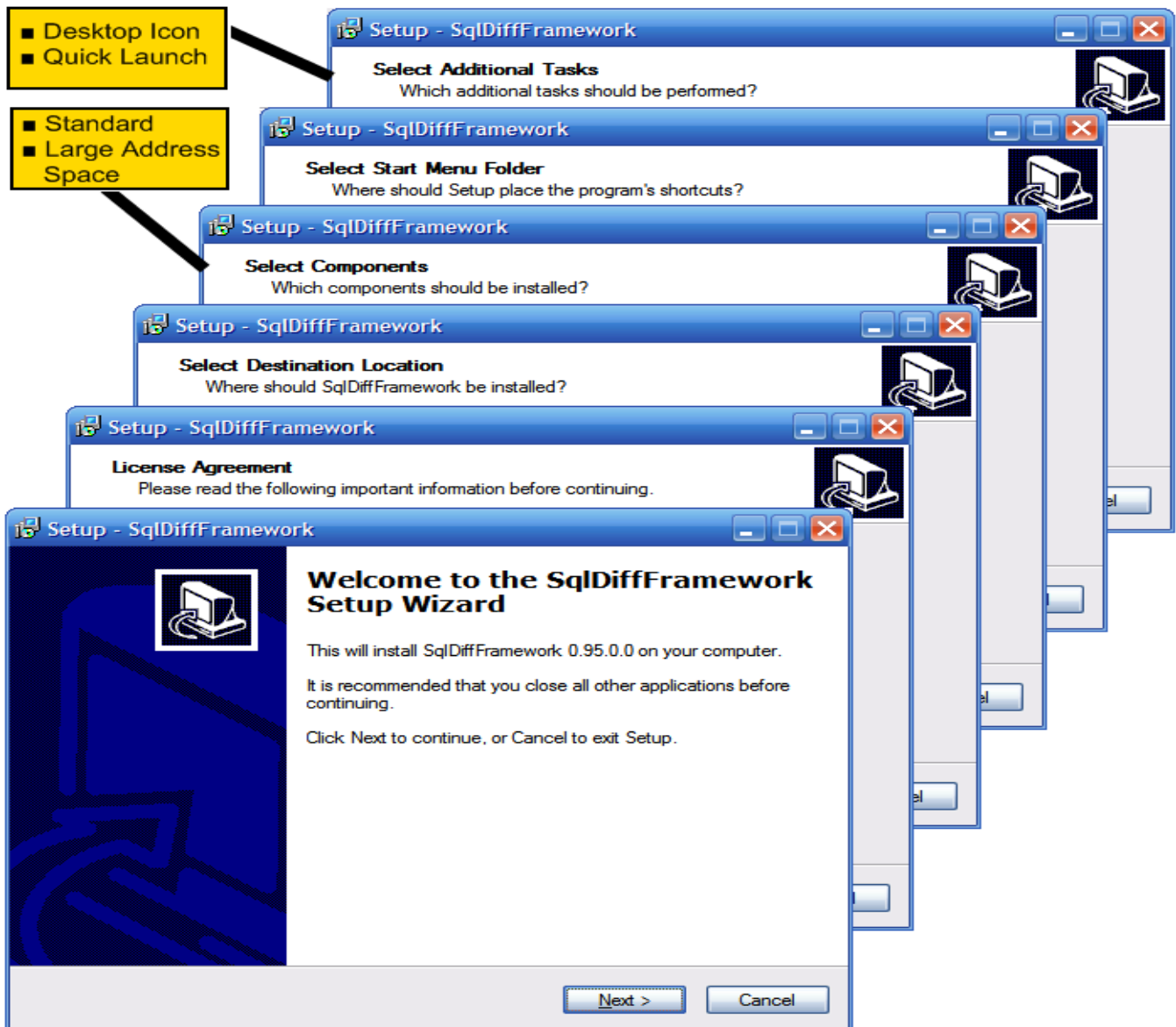


Figure 2-1 Installer Snapshot

2.1.1 Privacy Considerations

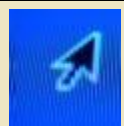
SqlDiffFramework saves passwords that you enter in an encrypted form. Neither you nor anyone else can ever see your passwords from inside the application. The passwords are saved—at your option—in your personal user .config file; however, they are also encrypted in the file. The encryption key is tied to the current machine so your configuration file cannot be used on another machine. Even if you export your passwords as a backup, the export stores them encrypted so they cannot be read.

SqlDiffFramework never “phones home” with any of your personal information. It may—if *you* configure it as described in section 2.2.3—store information about your installation within your corporate network, for use by your own system administrators.

2.1.2 Component Options

Everything in Figure 2-1 is completely standard for a Windows installer except for the choice of components, indicated as **Standard** or **Large Address Space**. The latter is available if the standard 2GB address space for Windows applications (Windows XP or earlier) just is not enough for the data you need to analyze. To give you some feel for this, I had not needed to implement this option for more than 6 months of intensive use of the application. But for a result set exceeding 500,000 records and 10 columns, SqlDiffFramework complained of exhausting memory at 1.5GB (presumably the other 0.5 GB was due to overhead from the .NET framework). By using the larger address space model, this query comparison was able to grab the remaining 300MB it needed to complete analysis.

In order to use the larger address space model you need to take an additional step beyond the selection in the installer. You must modify your system boot configuration to increase the size of the user process address space from 2GB to 3GB using the `/3GB` option in your `boot.ini` file. Figure 2-2 shows the simplest way to do this. (For further details, see Microsoft's *Boot INI Options Reference* at <http://support.microsoft.com/kb/833721>.)



TIP: To check whether you installed the standard or the large address space version of the application, use the `dumpbin` utility from a Visual Studio command prompt. Check the headers of the exe file (`/headers` option) with `dumpbin` and look for a line near the top stating: *Application can handle large (>2GB) addresses.*

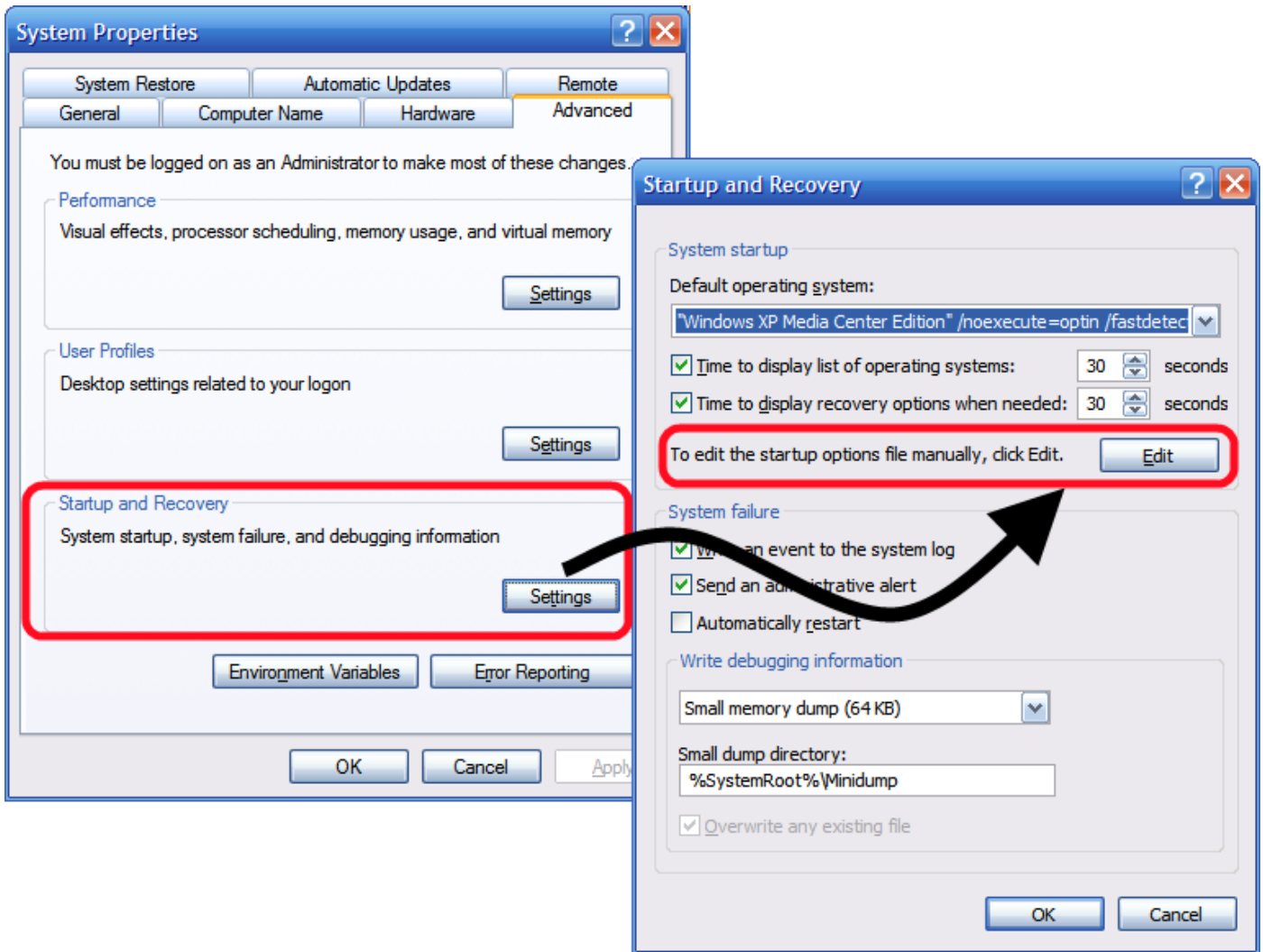


Figure 2-2 Accessing Boot Configuration Options
From the System Properties control panel, select the startup and recovery settings, then select the button to edit the startup options.

2.2 Enterprise Configuration

SqlDiffFramework provides a few optional features geared toward multiple installation deployment and corporate environments. The following sections describe these features that you would perform as a post-installation step. SqlDiffFramework does not provide an automatic mechanism for these. If your user base is small you can do these manually with a text editor. For a larger user base, use scripting tools to perform the actions described.

2.2.1 Provide Common Connections for Your Enterprise

A connection defines the credentials necessary to connect to a data source. Section 3.9 describes connections in detail including how to export a set of connections from one machine and import them to another. As part of your deployment, you may wish to define a set of connections common to all users or a group of users, then merge that connection set into each installation as a post-installation step. The connections are stored in the application's standard .NET configuration file (`SqlDiffFramework.exe.config`) located in the same directory as the application itself. The default value of the `ConnectionList` setting provides two SQL Server connections as an example, one to the master database available on any system and one to the commonly used AdventureWorks demo database.

```
<setting name="ConnectionList" serializeAs="Xml">
  <value>
    <ArrayOfConnectionDetails
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <ConnectionDetails>
        <ConnectionName>local-master</ConnectionName>
        <ConnectionString>Data Source=.\SQLEXPRESS;Initial
Catalog=master;Integrated Security=True;Persist Security
Info=True</ConnectionString>
        <DbType>SqlServer</DbType>
        <RememberPassword>>false</RememberPassword>
        <Description>local SExpress, master DB</Description>
      </ConnectionDetails>

      <ConnectionDetails>
        <ConnectionName>local-AdvWorks</ConnectionName>
        <ConnectionString>Data Source=.\SQLEXPRESS;Initial
Catalog=AdventureWorks;Integrated Security=True;Persist Security
Info=True</ConnectionString>
        <DbType>SqlServer</DbType>
        <RememberPassword>>false</RememberPassword>
        <Description>local SExpress, AdventureWorks DB</Description>
      </ConnectionDetails>

    </ArrayOfConnectionDetails>
  </value>
</setting>
```

You may either edit this `<setting>` with an XML editor or use the SqlDiffFramework's connection editor, export the settings to a file, then create a script which copies the XML in the exported file into the appropriate section of the `SqlDiffFramework.exe.config` file on each target machine. Note that this is a user-scoped setting so it may also be stored in the `user.config` file, depending on your needs.

2.2.2 Provide Update Notification within Your Enterprise

SqlDiffFramework checks for updates from a given location on your intranet allowing you, the system administrator, full control over when users should be notified of new versions. Note that it does *not* automatically install an update; it merely notifies users of the presence of an update.

Installation packages follow a naming convention of including the version number in the package name for use by the update notification mechanism. You may alter the package names if desired but to recognize the version of the package the version number must be embedded in the file name with components separated by either hyphens, underscores, or periods. For example, `SqlDiffFramework_1-1-0-0.exe` would be recognized as having version 1.1.0.0. The components of the version number are compared numerically so `SqlDiffFramework_1-1-10-0.exe` is correctly determined to be a later version than `SqlDiffFramework_1-1-2-0.exe` (if compared as simple character strings the reverse would be true).

To use this update notification feature you need to provide a value to the `UpdateRepository` (section 4.2.1.2) setting in the `SqlDiffFramework.exe.config` file. Set this to an empty value to globally disable the feature. Even if you provide a valid repository path, the user may use the **Options** panel to either change the notification check interval or disable it completely at the local level.

Update notification does not occur on every invocation; once notified, a user will not be notified for 5 days to prevent an annoying pop-up on every application start.

As a performance optimization, SqlDiffFramework learns if the search path contains no useful directories and eventually stops checking. Depending on your network this could improve startup time where there might be delays checking network directories only to find none of them exist. See the description of the `UnreachableRepository` setting (section 4.2.2.5) for further details.

2.2.3 Track Users within Your Enterprise

Within a corporate environment the IT department often wants to keep track of installations of any application. SqlDiffFramework automatically catalogs the first use of each user after installation. It writes user details to a common XML catalog file; here is a sample showing a succession of versions from 1.0.1.0 through 1.0.3.0, including several different users, including a couple re-installations of the same version.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<catalog>
  <version id="1.0.1.0">
    <user name="Sorens, Michael">
      <event ip="149.93.16.52" date="3/22/2009 9:09:08 AM" os="Win XP Pro
5.1.2600 SP2" machine="dellxp" />
      <event ip="149.93.16.52" date="3/22/2009 9:23:57 AM" os="Win XP Pro
5.1.2600 SP2" machine="dellxp" />
    </user>
    <user name="Smith, James">
      <event ip="149.93.16.157" date="3/22/2009 9:34:38 AM" os="Win 2000
Pro 5.0.2195 SP4" machine="smithpc" />
    </user>
  </version>
  <version id="1.0.2.0">
    <user name="Sorens, Michael">
      <event ip="149.93.16.52" date="3/22/2009 9:39:02 AM" os="Win XP Pro
5.1.2600 SP2" machine="dellxp" />
    </user>
  </version>
  <version id="1.0.3.0">
    <user name="Sorens, Michael">
      <event ip="149.93.16.52" date="3/23/2009 1:56:29 PM" os="Win XP Pro
5.1.2600 SP2" machine="dellxp" />
      <event ip="149.93.16.52" date="3/26/2009 8:07:26 AM" os="Win XP Pro
5.1.2600 SP2" machine="dellxp" />
      <event ip="149.93.16.52" date="3/26/2009 10:42:24 AM" os="Win XP Pro
5.1.2600 SP2" machine="dellxp" />
    </user>
    <user name="Smith, James">
      <event ip="149.93.16.157" date="3/26/2009 10:48:42 AM" os="Win 2000
Pro 5.0.2195 SP4" machine="smithpc" />
    </user>
    <user name="Valentine, Michael">
      <event ip="149.93.16.157" date="3/29/2009 3:43:17 PM" os="Win 2000
Pro 5.0.2195 SP4" machine="valentinepc" />
    </user>
  </version>
</catalog>
```

To use this cataloging feature you need to provide a value to the `UpdateRepository` (section 4.2.1.2) and `UpdateCatalogFile` (section 4.2.1.1) settings in the `SqlDiffFramework.exe.config` file. Set either one or both of these settings to an empty value to disable the feature.

As a performance optimization, SqlDiffFramework learns if the search path contains no useful directories and eventually stops checking. Depending on your network this could improve startup time where there might be delays checking network directories only to find none of them exist. See the description of the `UnreachableRepository` setting (section 4.2.2.5) for further details.

3 Using SqlDiffFramework

You begin with the application's main window with no content—see Figure 3-1. There are two identical editor panes side by side. You load a query into each of these then execute those queries against two different databases. SqlDiffFramework then analyzes the two result sets and color codes any discrepancies.

The queries need to be written with certain guidelines in mind so that SqlDiffFramework understands how to compare the results. You will see how to do that shortly, but first take a look at what the application can really do.

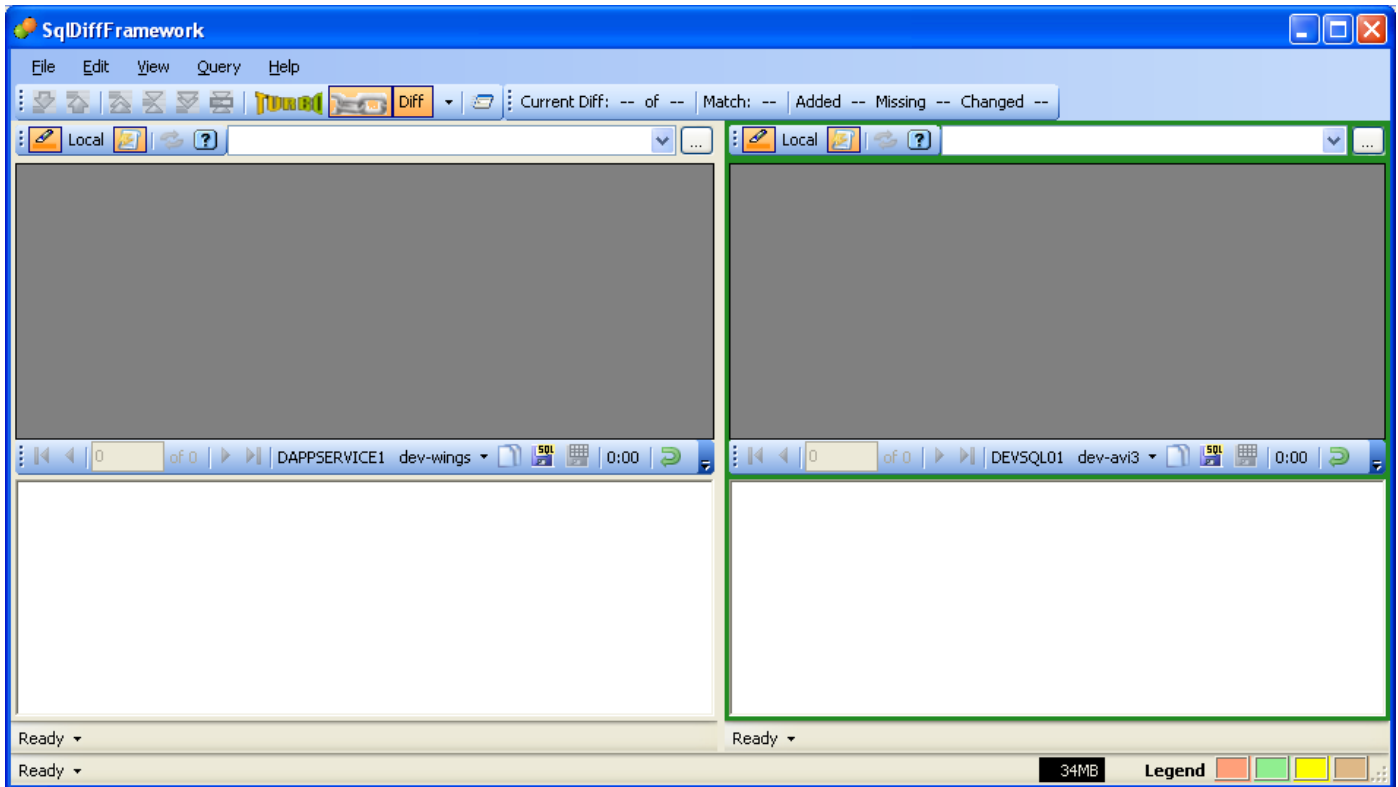


Figure 3-1 Main Window

Main Window: A fresh invocation of SqlDiffFramework shows a menu bar and two toolbars at the top and a status bar at the bottom. The middle contains two identical editor panes.

3.1 Analyzing Differences

A politician, a writer, and an economist wash up on a desert island. Next to them on the beach is a case of food—canned goods. How to open the cans? The banker is stumped; the writer has no ideas. They look at the economist, who says: “Assume we have a can opener...”

To you, dear reader, I analogously begin: For the moment, assume you know how to load a file, execute a query, and perform an analysis. Having done all that, here is the differential comparison of two small data sets. The editor pane on the left is pointing to an Oracle 10g database; the editor pane on the right is pointing to a SQL Server 2005 database. These two systems maintain some common tables between them and it is crucial that they stay precisely synchronized. You, as the system administrator, fire up SqlDiffFramework on a regular basis to monitor the synchronization.

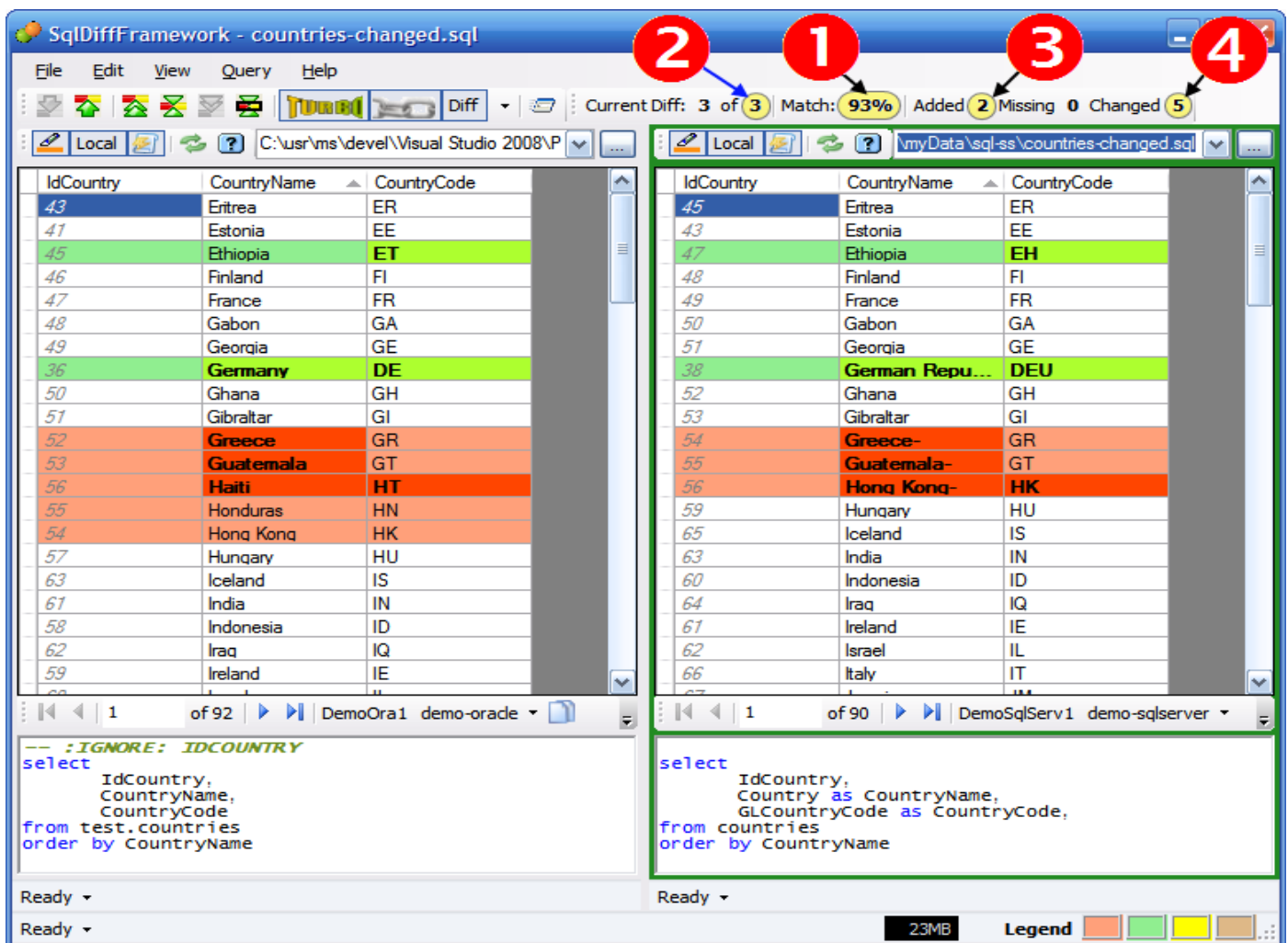


Figure 3-2 A Differential Comparison

There are several changes noted between a table in the Oracle database and a corresponding table in the SQL Server database.

Figure 3-2 shows that there are some differences. Glance first at the results toolbar (see 4.1.2); this is the toolbar at the top of the window on the right. It reports a 93% match; this is highlighted in the figure as point 1. The differences in this case are in three clusters (point 2) and the screen shot happens to indicate that the current difference cluster is the last of the three. The results toolbar further indicates that among the collective set of differences, there are 2 rows added to the left side that are not on the right side (point 3) and 5 rows that exist on both sides but contain different data (point 4).

Now take a look at the color-coded portions of the result sets. Each pair of rows that differ is colored light green, with the exception of the *current* difference, colored light red. Thus, when you navigate back to the second difference, the third set reverts to green and the second set becomes red. Within a row, SqlDiffFramework shows precisely which columns differ by increasing the color saturation and emboldening the text. The first difference cluster, consisting of a single row containing Ethiopia, has a discrepancy in just the CountryCode column, “ET” on the left and “EH” on the right. The second difference cluster has a discrepancy in two columns, the CountryName and the CountryCode, so both are highlighted in deeper green.

The third difference cluster is more interesting. Observe that on the left side there are 5 rows marked but only 3 of them highlight specific discrepancies. Discrepant rows in one editor pane are always paired up with rows in the partner pane. Therefore there can only be discrepancies marked for the minimum row count between the left rows and the right rows in the cluster. Because the right side has the smaller number—3 rows—this dictates how many rows on the left can show discrepancies.

But why, you may wonder, does the grid pair “Haiti” on the left with “Hong Kong-” on the right? It is “obvious” that the two “Hong Kong” rows should be paired together, right? But the right side value is “Hong Kong-” with a (presumably accidental) trailing hyphen. Differencing algorithms typically compare two values—in this case the entire left-side row and the entire right-side row—and determine that they match or they do not; there is no “almost”. So as an observer, *you* can see that there are two “Hong Kong” rows and those should be compared but the application only sees that the first three rows differ. What this implies, though, is that the more you clean up the data, the more difference clusters will be meaningful, allowing you to clean up the data better still, to get more meaningful clusters, ad infinitum.

I will mention just briefly here that differential analysis is a challenging problem that does not have one optimal solution. As such, SqlDiffFramework contains three different differencing engines, switchable with a single mouse click (see 4.1.1.9). Even on the small example presented above you will get different results between the Tauberer engine and the Potter engine. On large data sets, therefore, differences may diverge even more.

Notice in Figure 3-2 that the values in the IdCountry column differ markedly yet those differences are ignored. See 3.3 for details on how the IdCountry column was skipped, as well as how the country name and code were properly paired.

3.2 Analyzing Additions and Deletions

The previous section focused on *changed data*; data that exists in both left and right editor panes but with different values. This section considers the other major category: *added data*. Added data from one side is, of course, missing data from the other side.

SqlDiffFramework always considers the left editor pane as the reference point so *added rows* means rows appearing on the left that do not exist on the right while *missing rows* means rows appearing on the right that do not exist on the left. The results toolbar in Figure 3-2 reports the values for added, missing, and changed rows.

For changed data you have seen that discrepant rows are marked in green on both left and right editor panes. Added rows are similarly marked in green but only in the left pane. Green continues to mean that those values are different from the partner pane. The right pane, however, uses brown colored rows to indicate a location where the added rows on the left would be if they were present. Deleted rows do just the reverse: these are marked with green on the right and brown on the left.

*The brown rows in themselves do **not** indicate different data; they are reference indicators only!*

Normally there will always be *two* brown rows. Together they indicate the single reference point right between them. At the very top and very bottom of the data grid, though, there will only be a single brown row. Think of the second one as existing just off the grid so that, again, the reference point is right between them. In the case of a brown first row, the reference point is therefore above it at the top boundary of the grid. Similarly, a brown last row means the reference point is below it at the bottom boundary of the grid. The following storyboard should help make this clear.

IdCountry	CountryName	CountryCode
9	Anqola	AO
10	Arqentina	AR
7	Amenia	AM
12	Australia	AU
11	Austria	AT
13	Azerbaijan	AZ
17	Bahrain	BH
14	Banqladesh	BD
23	Belarus	BY
18	Benin	BJ
20	Bolivia	BO
22	Botswana	BW
21	Brazil	BR
19	Brunei Darus...	BN
16	Bulqaria	BG
28	Cameroon	CM
2	Canada	CAN
27	Chile	CL
29	China	CN
30	Colombia	CO
24	Conqo	CG

IdCountry	CountryName	CountryCode
8	Albania	AL
41	Alqeria	DZ
11	Anqola	AO
12	Arqentina	AR
9	Amenia	AM
14	Australia	AU
13	Austria	AT
15	Azerbaijan	AZ
19	Bahrain	BH
16	Banqladesh	BD
25	Belarus	BY
17	Belqium	BE
20	Benin	BJ
22	Bolivia	BO
24	Botswana	BW
18	Bulqaria	BG
30	Cameroon	CM
2	Canada	CAN
29	Chile	CL

This snapshot shows a fresh analysis indicating these four difference clusters in order:

- two missing rows at the top of the grid;
- one missing row in the middle;
- two added rows in the middle; and
- three added rows at the bottom of the grid.

Since this is a fresh analysis, there is not yet any current cluster and all rows are either brown (indicating reference points) or green (indicating discrepant data).

IdCountry	CountryName	CountryCode
9	Anqola	AO
10	Arqentina	AR
7	Amenia	AM
12	Australia	AU
11	Austria	AT
13	Azerbaijan	AZ
17	Bahrain	BH
14	Banqladesh	BD
23	Belarus	BY
18	Benin	BJ
20	Bolivia	BO
22	Botswana	BW
21	Brazil	BR
19	Brunei Darus...	BN
16	Bulqaria	BG
28	Cameroon	CM
2	Canada	CAN
27	Chile	CL
29	China	CN
30	Colombia	CO
24	Conqo	CG

IdCountry	CountryName	CountryCode
8	Albania	AL
41	Alqeria	DZ
11	Anqola	AO
12	Arqentina	AR
9	Amenia	AM
14	Australia	AU
13	Austria	AT
15	Azerbaijan	AZ
19	Bahrain	BH
16	Banqladesh	BD
25	Belarus	BY
17	Belqium	BE
20	Benin	BJ
22	Bolivia	BO
24	Botswana	BW
18	Bulqaria	BG
30	Cameroon	CM
2	Canada	CAN
29	Chile	CL

Upon selecting **Move to Next Difference** the first difference becomes the *current difference*.

Consistent with the previous section, changed data in the current difference cluster changes from green to red; in this case the Albania and Algeria rows on the right. The brown reference on the left also changes color—this time to yellow—to indicate the current difference.

Interpretation of this current difference:

The Albania and Algeria rows are missing from the left; if they were present they would be *before* Angola.

IdCountry	CountryName	CountryCode
9	Anqola	AO
10	Arqentina	AR
7	Amenia	AM
12	Australia	AU
11	Austria	AT
13	Azerbaijan	AZ
17	Bahrain	BH
14	Banqladesh	BD
23	Belarus	BY
18	Benin	BJ
20	Bolivia	BO
22	Botswana	BW
21	Brazil	BR
19	Brunei Darus...	BN
16	Bulqaria	BG
28	Cameroon	CM
2	Canada	CAN
27	Chile	CL
29	China	CN
30	Colombia	CO
24	Conqo	CG

IdCountry	CountryName	CountryCode
8	Albania	AL
41	Alqeria	DZ
11	Anqola	AO
12	Arqentina	AR
9	Amenia	AM
14	Australia	AU
13	Austria	AT
15	Azerbaijan	AZ
19	Bahrain	BH
16	Banqladesh	BD
25	Belarus	BY
17	Belqium	BE
20	Benin	BJ
22	Bolivia	BO
24	Botswana	BW
18	Bulqaria	BG
30	Cameroon	CM
2	Canada	CAN
29	Chile	CL

Move to Next Difference makes the second difference the current difference.

Interpretation of this current difference:

The missing Belgium row, if it existed on the left, would be between Belarus and Benin, just as it is on the right.

Observe that both Belarus and Benin exist on both the left and the right and they match. This confirms that the brown (and the yellow) do *not* indicate differences, just reference points.

IdCountry	CountryName	CountryCode
9	Anqola	AO
10	Arqentina	AR
7	Amenia	AM
12	Australia	AU
11	Austria	AT
13	Azerbaijan	AZ
17	Bahrain	BH
14	Banqladesh	BD
23	Belarus	BY
18	Benin	BJ
20	Bolivia	BO
22	Botswana	BW
21	Brazil	BR
19	Brunei Darus...	BN
16	Bulqaria	BG
28	Cameroon	CM
2	Canada	CAN
27	Chile	CL
29	China	CN
30	Colombia	CO
24	Conqo	CG

IdCountry	CountryName	CountryCode
8	Albania	AL
41	Alqeria	DZ
11	Anqola	AO
12	Arqentina	AR
9	Amenia	AM
14	Australia	AU
13	Austria	AT
15	Azerbaijan	AZ
19	Bahrain	BH
16	Banqladesh	BD
25	Belarus	BY
17	Belqium	BE
20	Benin	BJ
22	Bolivia	BO
24	Botswana	BW
18	Bulqaria	BG
30	Cameroon	CM
2	Canada	CAN
29	Chile	CL

Move to Next Difference makes the third difference the current difference.

Interpretation of this current difference:

The added rows for Brazil and Brunei would be between Botswana and Bulgaria.

IdCountry	CountryName	CountryCode
9	Anqola	AO
10	Argentina	AR
7	Amenia	AM
12	Australia	AU
11	Austria	AT
13	Azerbaijan	AZ
17	Bahrain	BH
14	Banqladesh	BD
23	Belarus	BY
18	Benin	BJ
20	Bolivia	BO
22	Botswana	BW
21	Brazil	BR
19	Brunei Darus...	BN
16	Bulqaria	BG
28	Cameroon	CM
2	Canada	CAN
27	Chile	CL
29	China	CN
30	Colombia	CO
24	Congo	CG

IdCountry	CountryName	CountryCode
8	Albania	AL
41	Algeria	DZ
11	Anqola	AO
12	Argentina	AR
9	Amenia	AM
14	Australia	AU
13	Austria	AT
15	Azerbaijan	AZ
19	Bahrain	BH
16	Banqladesh	BD
25	Belarus	BY
17	Belqium	BE
20	Benin	BJ
22	Bolivia	BO
24	Botswana	BW
18	Bulqaria	BG
30	Cameroon	CM
2	Canada	CAN
29	Chile	CL

Move to Next Difference

makes the fourth and final difference the current difference.

Interpretation of this current difference:

The added rows for China, Colombia, and Congo follow Chile on the left. If they were also on the right, they would similarly follow Chile, highlighted in yellow.

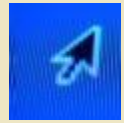
3.3 Including and Excluding Columns

3.3.1 Create the Basic Query

The crux of differential comparison is specifying what to compare and what to ignore. Figure 3-3 again shows the example country query that you saw when it was tailored for SqlDiffFramework use, but here it is rewound back to the first cut. The Oracle query on the left includes the three fields to display and the SQL Server query on the right shows the corresponding three fields, though two of them have different field names. Ironically, the field name that you do *not* want to compare—IdCountry—happens to be the same on both sides. In your environment you may want to match an ID field. For this example, I am making a reasonable assumption that the ID field for each table is a primary key, meaningless outside that table but vital to the table itself. As a primary key, it is often helpful to display the ID in order to uniquely identify a record but it is meaningless to compare it to another ID.

SqlDiffFramework matches columns by their display names. A display name is just the name shown in the result set grid. In Figure 3-3, the display names are identically the field names referenced in the query. Since IdCountry is present in both result sets, those columns are matched for comparison. The name of the country, though, is called

CountryName on the left but Country on the right. Since these display names are different those columns are not matched—this is noted in the grids by marking the entire column in italics and using grey text instead of black. The country code similarly does not match and is marked in italics.



TIP: Note that column matching is *not* case-sensitive: You may use `IdCountry` on one side and `IDCOUNTRY` on the other and they will still match.

The screenshot displays the SqlDiffFramework application with two editor panes. The left pane shows a query for 'test.countries' with columns IdCountry, CountryName, and CountryCode. The right pane shows a query for 'countries' with columns IdCountry, Country, and GLCountryCode. The diff view highlights differences in column names and codes, with some columns marked as italicized and greyed out to indicate they do not match between the two queries.

IdCountry	CountryName	CountryCode	IdCountry	Country	GLCountryCode
43	Eritrea	ER	45	Eritrea	ER
41	Estonia	EE	43	Estonia	EE
45	Ethiopia	ET	47	Ethiopia	EH
46	Finland	FI	48	Finland	FI
47	France	FR	49	France	FR
48	Gabon	GA	50	Gabon	GA
49	Georgia	GE	51	Georgia	GE
36	Germany	DE	38	German R...	DEU
50	Ghana	GH	52	Ghana	GH
51	Gibraltar	GI	53	Gibraltar	GI
52	Greece	GR	54	Greece-	GR
53	Guatemala	GT	55	Guatemala-	GT
56	Haiti	HT	56	Hona Kong-	HK

The left query is: `select IdCountry, CountryName, CountryCode from test.countries order by CountryName`

The right query is: `select IdCountry, Country, GLCountryCode from countries order by Country`

Figure 3-3 A Starting Query

These queries identify the columns to be displayed but produce a poor match. The IdCountry field is matched between editor panes but the country name and code are not, just the reverse of what would be most useful.

3.3.2 Match Columns by Aliasing

Once you have created the basic query to return the data you need, the second step is to alias field names to matching display names. You may do this on either editor pane (or even both if desired). As shown in Figure 3-4, the right side query is modified to alias the native Country field to the CountryName display name. Similarly, GLCountryCode is aliased to CountryCode. The alias names were assigned by observing the field names from the left editor pane.

With these aliases in place, re-executing the queries yields the result sets shown in Figure 3-4. Now all three columns are shown in as non-italics indicating they are all actively being compared. You do not, however, want the IdCountry column to be an active column; the next section shows how to disable it.

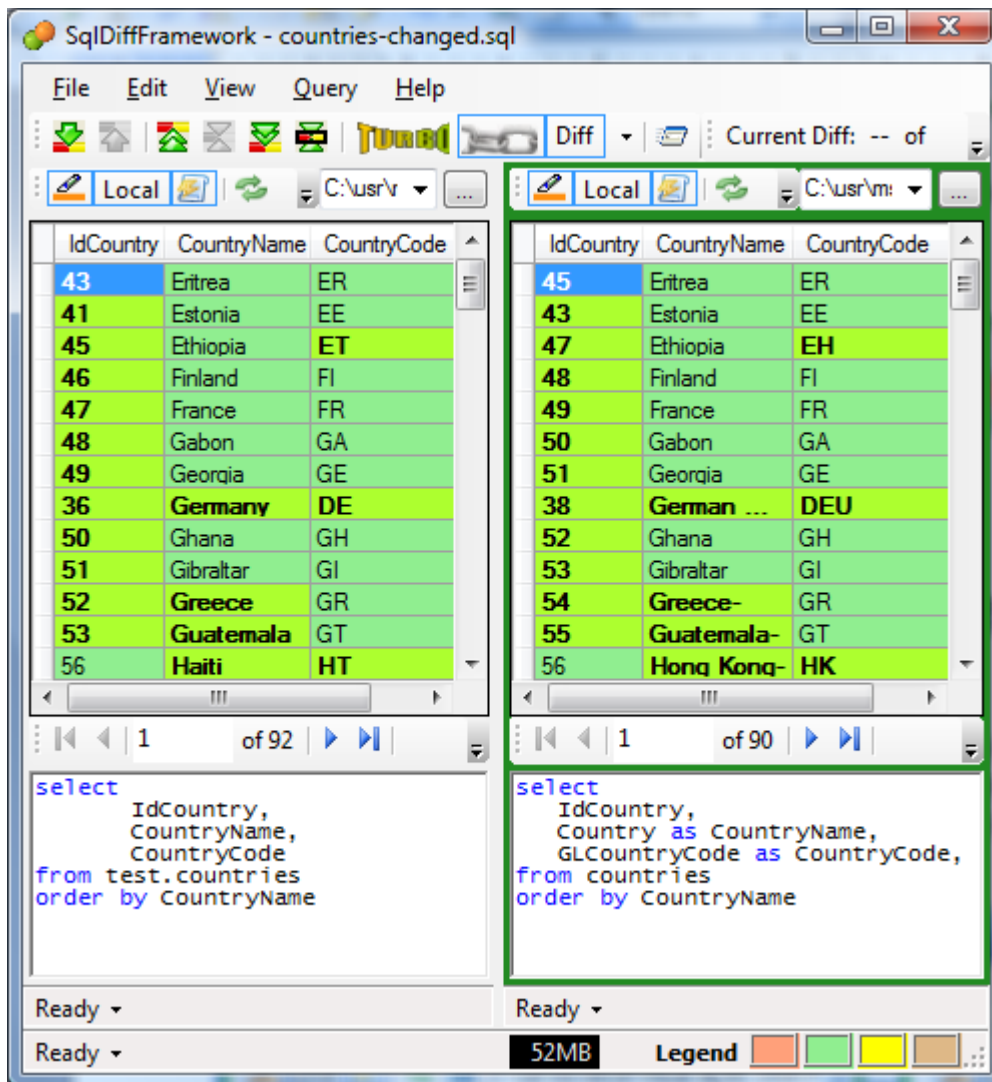


Figure 3-4 Matching Columns

These queries have been correlated to produce a better match by aliasing the field names to matching display names.

3.3.3 Skip Columns With Matching Names

There are actually two ways to skip columns. The first technique you already know: make the display names different. The second involves a special syntactic construct in your query. Use the `:IGNORE:` clause embedded into a comment so that it does not cause errors when you send the query to the database. Following the `:IGNORE:` keyword include a comma-separated list of field names that you want to skip. Figure 3-5 shows the `IdCountry` field being skipped with the `:IGNORE:` clause.

The screenshot displays the SqlDiffFramework application window titled "countries-changed.sql". It features a menu bar (File, Edit, View, Query, Help) and a toolbar with various icons. Below the toolbar are two data grids and a SQL editor pane.

The left data grid shows a table with columns `IdCountry`, `CountryName`, and `CountryCode`. The rows are:

IdCountry	CountryName	CountryCode
43	Eritrea	ER
41	Estonia	EE
45	Ethiopia	ET
46	Finland	FI
47	France	FR
48	Gabon	GA
49	Georgia	GE
36	Germany	DE
50	Ghana	GH
51	Gibraltar	GI
52	Greece	GR
53	Guatemala	GT
56	Haiti	HT

The right data grid shows a similar table with different values:

IdCountry	CountryName	CountryCode
45	Eritrea	ER
43	Estonia	EE
47	Ethiopia	EH
48	Finland	FI
49	France	FR
50	Gabon	GA
51	Georgia	GE
38	German ...	DEU
52	Ghana	GH
53	Gibraltar	GI
54	Greece-	GR
55	Guatemala-	GT
56	Hong Ko...	HK

The bottom SQL editor pane contains the following query:

```
-- :IGNORE: IdCountry
select
  IdCountry,
  CountryName,
  CountryCode
from test.countries
order by CountryName
```

The status bar at the bottom indicates "Ready" and "53MB" of data, along with a legend.

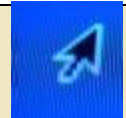
Figure 3-5 Skipping Columns

Use the special syntactic construct shown in the left editor pane to ignore particular fields.

Figure 3-5 shows the `:IGNORE:` clause as the first line in the left editor pane but it does not matter whether you put the clause in the left or the right editor pane nor where it appears within the query. The only restrictions to keep in mind are that the list of fields must all be on a single line with the opening token and nothing else except whitespace. The formal syntax for an `<ignore-clause>` is just this:

```
<ignore-clause> ::= '--' ':IGNORE:' <display-name> { ',' <display-name> }
```

Whitespace may appear anywhere within the line and is ignored.



TIP: If you change the elements in the ignore list, generally you should re-execute *both* queries, not just re-apply differencing or re-execute one query. That may work sometimes but other times you may see some residual difference highlighting. Re-executing the queries will clean that up.

3.4 Searching and Filtering

The search capability of the query editor and the search and filtering capabilities of the result grid together provide a useful means for finding and isolating information.

3.4.1 Searching the Query Editor

To search the query editor, invoke the Find command via the keyboard (`Control+F`) or via the menu (`Edit → Find...`); the Replace command (`Control+H`) brings up the same dialog with additional settings for replacement.

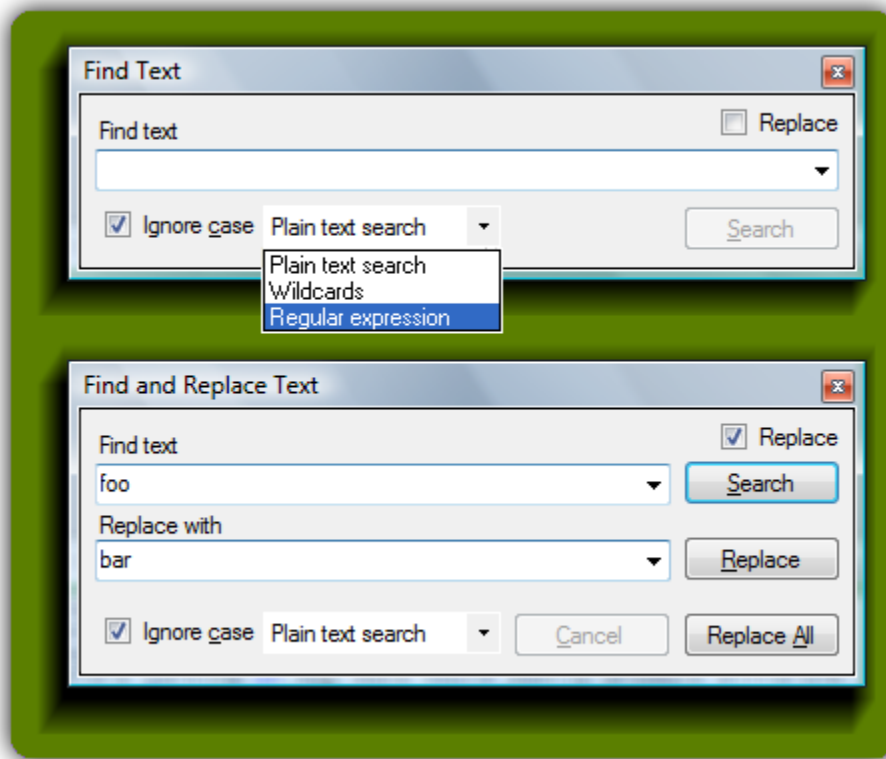


Figure 3-6 Query Editor Find and Replace

Both the Find (Control+F) and Replace (Control+H) dialogs allow searching by text, wildcards, or regular expressions. Switch the find mode with the drop-down as shown in the Find Text box at top. Switch between find and replace using the Replace check box in the upper right corner.

You may set the search mode to any of these choices:

Plain text – finds exactly what you type.

Wildcards – use `*` to match any number of characters and `?` to match any single character. These are standard wildcard characters to specify multiple files on the command line with Windows or Unix-like operating systems. This mechanism is provided as a convenience for users familiar with it, but use the regular expression mechanism for more flexibility.

Regular expressions – mix text and regular expression notation to create a powerful matching expression. A comprehensive description of regular expressions is beyond the scope of this manual but if you are not familiar with them I encourage you to learn a bit about them. They allow you to express elaborate constraints on a value.

Table 3-1 shows the most commonly used tokens in regular expressions.

Token	Description
^	Anchor to the start of a string
\$	Anchor to the end of a string
.	Match any character except newline
[...]	Match any character in the specified range
(...)	Group part of an expression
{ <i>m,n</i> }	Preceding expression must occur at least <i>m</i> times and at most <i>n</i> times
	Match one of alternate expressions
*	0 or more occurrences of preceding expression
+	1 or more occurrence of preceding expression
?	0 or 1 occurrences of preceding expression
\w	Match any letter or digit
\s	Match any whitespace character
\S	Match any non-whitespace character
\d	Match any digit

Table 3-1 Commonly Used Regular Expression Tokens

Table 3-2 shows some examples using the above tokens. Some regular expressions are simple and clear; others are far from it: recognizing something as seemingly simple as an e-mail address depends on what you mean by *e-mail address*. The last row of the table encompasses a very broad definition.

Regular Expression	Meaning
^{.5}\$	value must contain exactly 5 characters
.{5}	value must contain at least 5 characters
^\S+\$	value may not contain spaces
^-?(?:\d*\.\d+ \d+\.)\$	value must be a number
^\d{3}-\d{3}-\d{4}\$	value must be a canonical US phone number
^\w[\w\.]*@\w+\.\w[\w\.]*\$	value must be an e-mail address

Table 3-2 Regular Expression Samples

Here are just a few of many good references to learn more about regular expressions:

- Wikipedia (http://en.wikipedia.org/wiki/Regular_expression) has good background information.
- Espresso (<http://www.ultrapico.com/Espresso.htm>) is a free GUI-tool for building, interpreting, and testing regular expressions.
- RegExLib.com (<http://regexlib.com/CheatSheet.aspx>) has a quick reference chart as well as a library of regular expressions.
- Zvon.org (<http://www.zvon.org/other/reReference/Output/index.html>) has an interactive reference to regular expression tokens.
- Regular-expressions.info (<http://www.regular-expressions.info/>) has tutorial and reference material.

3.4.2 Searching the Results Grid

The results grid has a convenient mechanism for moving around a large data set quickly. Invoking the **Quick Find** command (see 4.5.1.10) opens a mini-dialog at the bottom of the result grid, confirming your primary sort column. Type one or more characters in the input field. As you enter each character, the current selection changes to reflect the first entry matching the prefix you have typed. For example, if you starting typing with an “L”, the current selection advances to the first entry in the sorted column beginning with “L”, say “LL Bottom Bracket” for example. If you add an “o” to make it “Lo” the current selection jumps to “Long Sleeve”, and so forth—see Figure 3-7.

This **Quick Find** is indeed quick, as it is optimized to leverage the current sorting of the result set. This mechanism provides a very fast way of moving around a large data set. It is

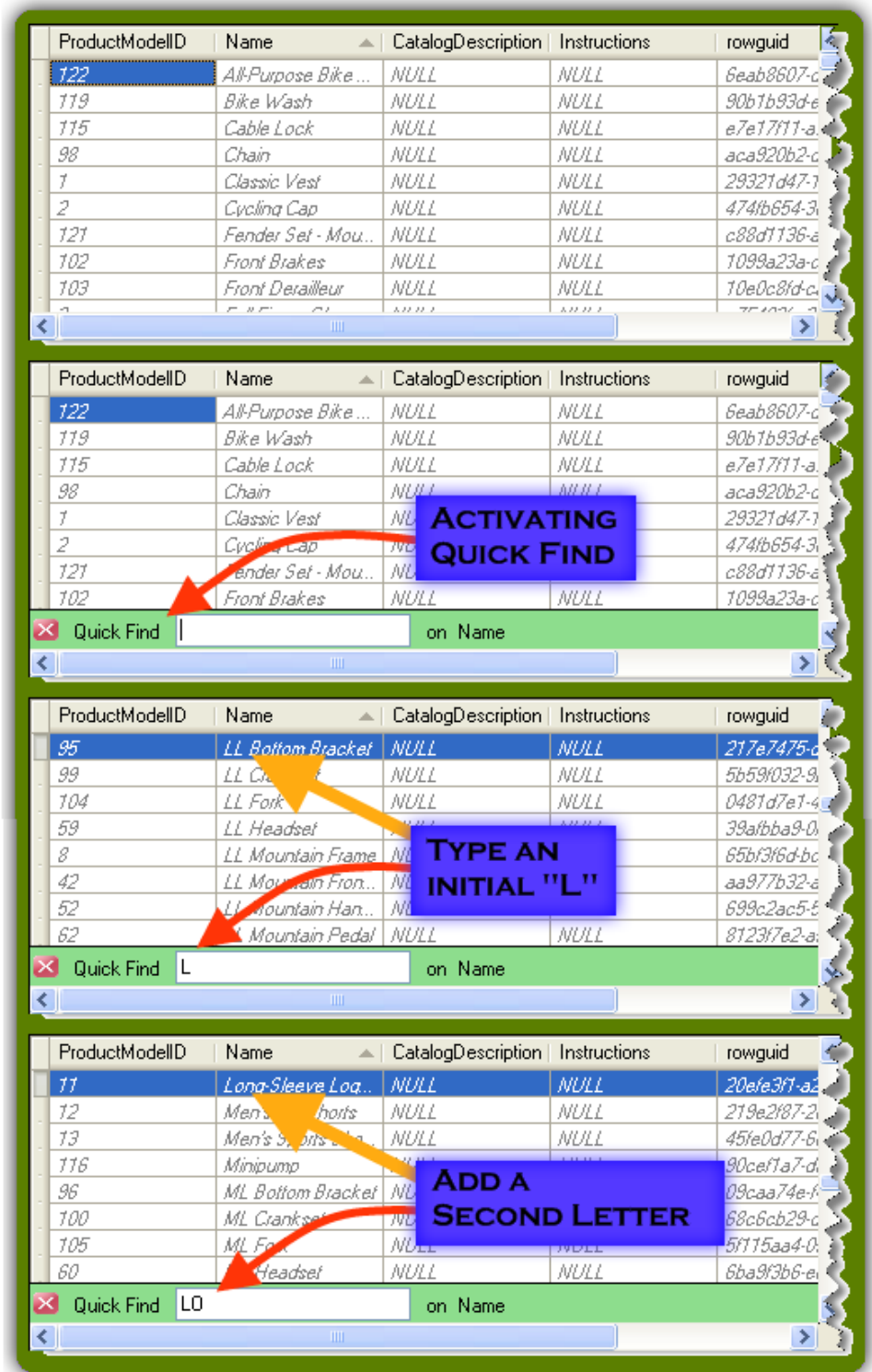


Figure 3-7 Result Grid Quick Find


This storyboard illustrates the process of searching a single manually sorted column or the primary column of a turbo-sorted result grid.

constrained, however, to sorting done within the application; it does not have hooks into the sorting done at the target database. There are two ways to sort within the application. Either click on a column header to sort by that column (see 4.5.1.9), or enable turbo-sorting to sort by any number of columns (see 3.5). With turbo-sorting, the primary sort column is available to use with the **Quick Find** operation.

3.4.3 Filtering the Results Grid

Rather than jumping to different rows in the result grid you also have the option to reduce the result set within the application; that is, it does not need to communicate with the database. This actually allows you to apply two levels of filters. First the `WHERE` clause in your query is applied at the database, returning a filtered view of a possibly larger dataset. Next, the `WHERE` clause of the result grid is applied directly within the grid. This latter predicate uses a special subset of SQL presented in Table 3-3.

To learn more about using this special SQL dialect in .NET applications see my article *Exploring Secrets of BindingSource Filters* at <http://www.devx.com/dotnet/Article/34451>.

When you specify a filter expression then press the Apply button () the result grid is immediately updated to reflect the predicate. Furthermore, if you have tandem mode enabled the partner grid is also updated with the same predicate and, if auto-differencing is enabled, the difference engine re-evaluates the two grids. Figure 3-8 illustrates the filtering process starting with a query with 389 rows.

Category	Element	Description
Comparison Operators	< > <= >= <> = IN LIKE	less than greater than less than or equal greater than or equal unequal equal contains matches
Arithmetic Operators	+ - * / %	addition subtraction multiplication division modulus
String Operators	+	concatenation
Logical Operators	AND OR NOT ()	both operands must be true either one or both operators must be true operand must be false used to force precedence
Wildcards <i>Allowed only at the start or end of a string</i>	* or %	match any number of characters
Aggregate Operators	SUM AVG MIN MAX COUNT STDEV VAR	sum average minimum maximum count deviation variance
Functions	CONVERT LEN ISNULL IIF TRIM SUBSTRING	type conversion string length returns second expression if first is null returns one of two values based on an expression removes leading and trailing white space returns a portion of a string
References	Parent Child	refers to parent table refers to child table
Delimiters	[] '' ## \ \'	wraps column names wraps column names wraps strings wraps dates—used with certain data providers escapes next character

Table 3-3 SQL Dialect for Grid Filtering

These details were extracted from this MSDN reference page:

<http://msdn2.microsoft.com/en-us/library/system.data.datacolumn.expression.aspx>.

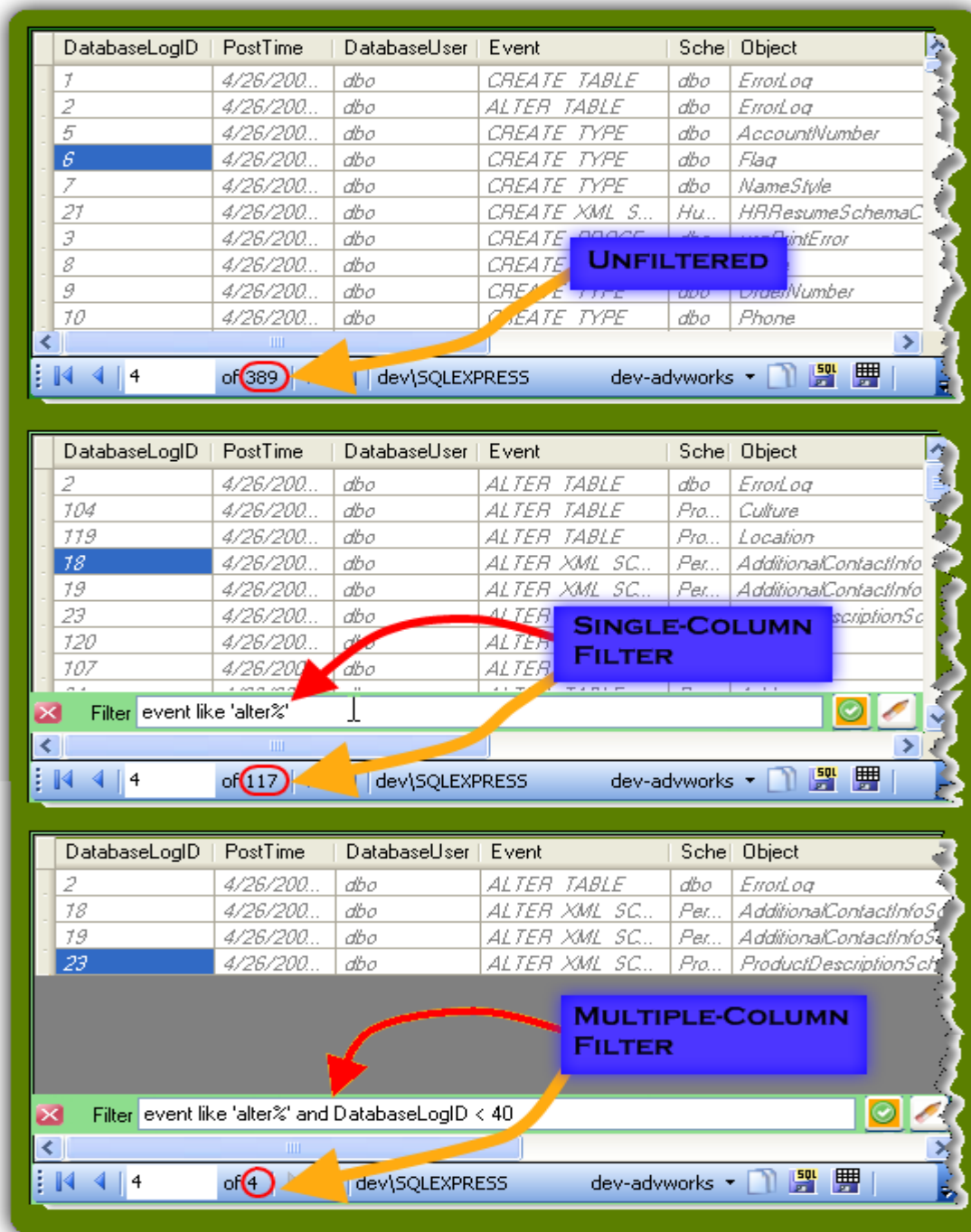


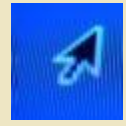



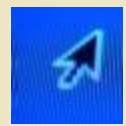
Figure 3-8 Filtering Sequence


The filter predicate will remain active until you explicitly remove it, even when you load data from a completely different result set. If a result set does not have the columns referenced in the filter query it will have no affect on your data, lying dormant until you load another query that does have the same column names. Remove the filter by either

deleting the text in the filter panel and clicking the Apply button () , or just press the Erase button () in the filter panel.



TIP: You may apply the filter predicate either by pressing **Enter** on the keyboard or clicking the Apply button () on the filter panel. But note that the former leaves the panel open while the latter closes the panel.



TIP: If you define a filter predicate with a column that is present in one result grid but not in its partner and tandem mode is enabled, SqlDiffFramework will apply the filter in the current grid and notify you that it does not exist in the partner. To avoid such warnings, use the tandem override (hold down **Control**) when you press **Enter** or click the Apply button () , so SqlDiffFramework will not attempt to process the partner grid.

3.5 Sorting and Turbo-sorting

Notice that in Figure 3-2 both queries specify sorting by the single `CountryName` field (`order by CountryName`). You may include any number of fields in the order by clause. SqlDiffFramework sends what you specify with the rest of the query to the respective databases where the database engines process the queries, sort the data as you specified, and return result sets. SqlDiffFramework then compares the result sets line by line.

3.5.1 Re-sorting Case Sensitive Results for Consistency

The most common pitfall of this seemingly straightforward operation is case sensitivity of data. Oracle, for example, is case sensitive while SQL Server (by default) is not. Therefore sorting of the data could easily produce different orderings for essentially the same data.

One solution is to change SQL Server's behavior: there are a number of ways to make SQL Server operate in a case-sensitive fashion. If you have the wherewithal to modify the table in question, make a column permanently case sensitive. Example:

```
ALTER TABLE dbo.Customers
    ALTER COLUMN CustID char(8) COLLATE SQL_Latin1_General_CP1_CS_AS NOT
NULL
```

If your database cannot be changed you could instead specify case-sensitivity in a query.:

```
SELECT *
FROM dbo.Customers
WHERE CustID = @CustID COLLATE SQL_Latin1_General_CP1_CS_AS
AND CustPassword = @CustPassword COLLATE SQL_Latin1_General_CP1_CS_AS
```

Or this:

```
SELECT
    companyName COLLATE SQL_Latin1_General_CP1_CS_AS,
    count(*)
FROM dbo.Customers
GROUP BY companyName COLLATE SQL_Latin1_General_CP1_CS_AS
```

These and several more techniques are well-documented at *Case sensitive search in SQL Server queries* (http://vyaskn.tripod.com/case_sensitive_search_in_sql_server.htm).

A more general solution, however, is to simply ask SqlDiffFramework to sort both results for you, overriding the sort order received from the databases. Since both result sets will then be sorted with the same rules—in this case rules of the .NET framework—there can be no sorting inconsistencies. This is called the *turbo-sort* feature. When you enable turbo-sorting, SqlDiffFramework locally sorts the results received from each executed query.

3.5.2 Re-sorting Static Snapshots

Another important use of turbo-sorting is for re-sorting static data, i.e. retrieving result sets from stored CSV snapshots. When you load a snapshot the currently loaded query is irrelevant; the result set is determined completely from the snapshot file and the order of the rows in the file is the order in which they are displayed. However, SqlDiffFramework

can apply turbo-sorting to local data as well as live data, allowing you to re-sort CSV snapshots on the fly.

As an example, Figure 3-9 shows snapshots loaded into both editor panes where the data was originally sorted---then saved---by CountryName. When it is reloaded as in Figure 3-9, it is still displayed in the same order. Enabling turbo-sorting that specifies sorting by CountryCode updates the display to that shown in Figure 3-10.

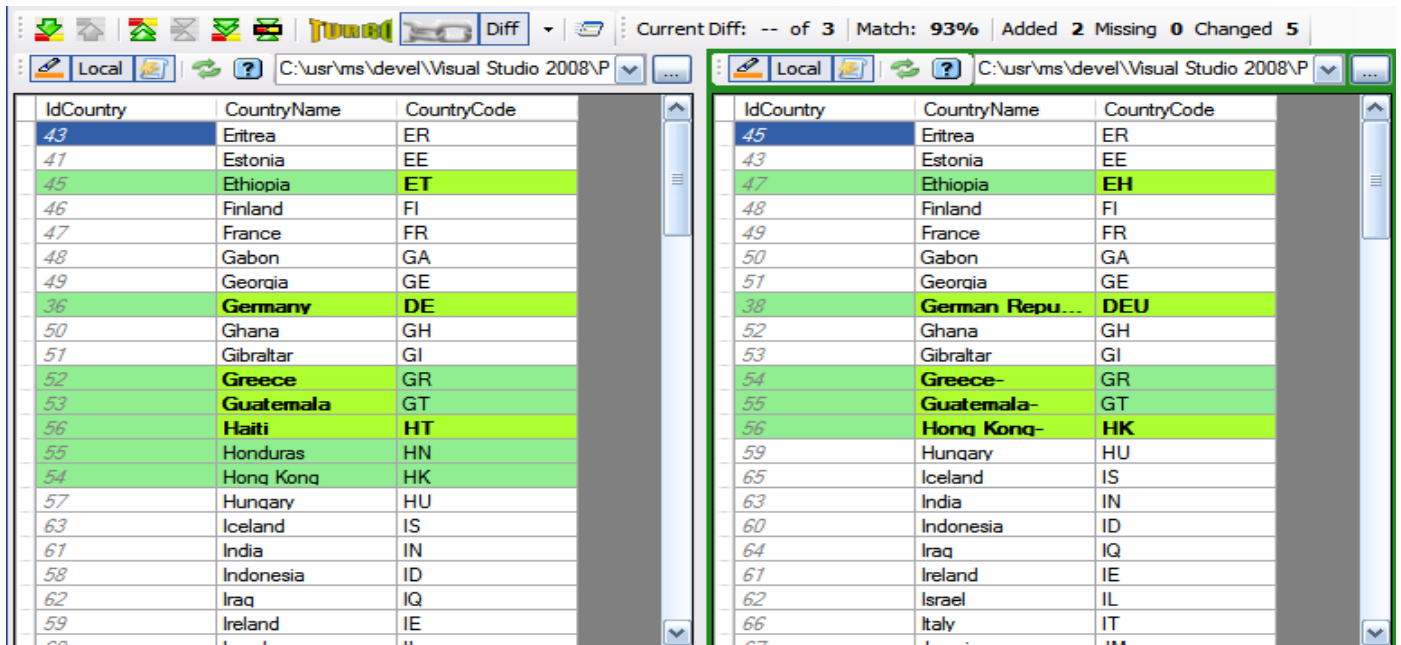


Figure 3-9 Snapshot Data in Native Order

Without turbo-sorting enabled, the snapshot displays data in the same order as it was saved in, in this case sorted by CountryName.

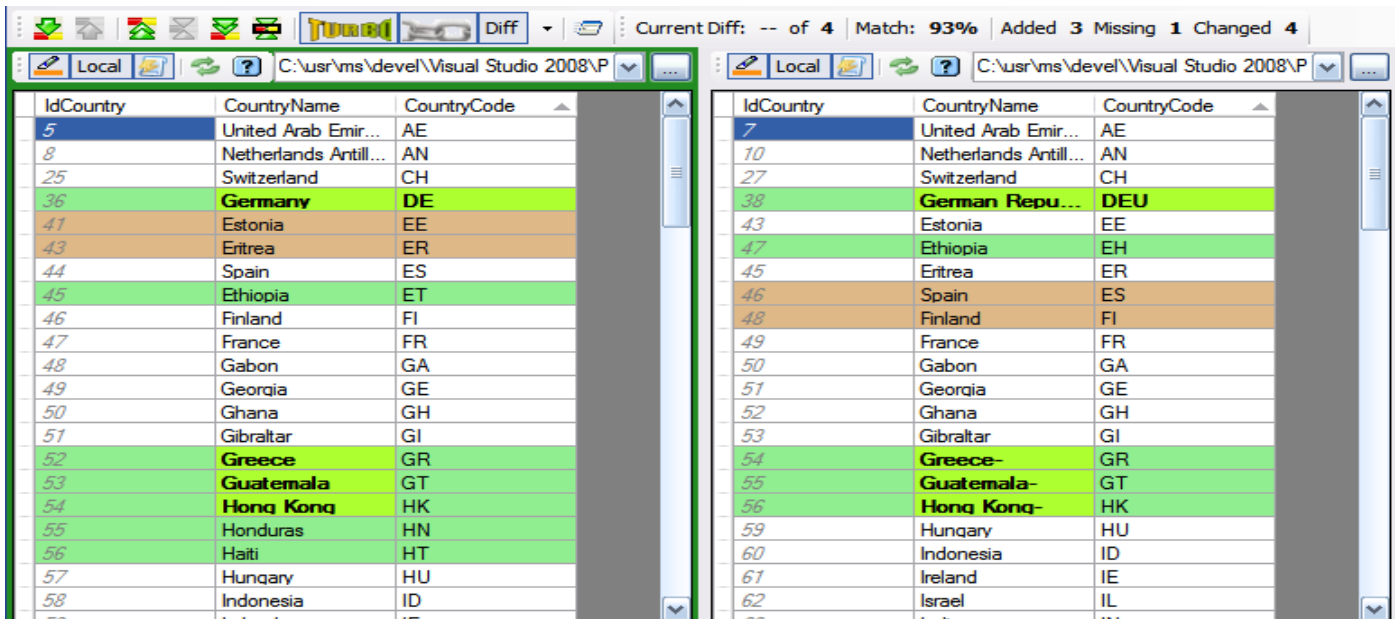


Figure 3-10 Snapshot Data With Turbo-sort Applied
 Specifying a turbo-sort by CountryCode resorts even static data from a CSV snapshot.

3.5.3 Re-sorting for NULL Consistency

Just as different databases handle case differently, there may also be differences in how NULL is treated when sorting. Oracle sorts NULLs at the end while SQL Server sorts NULLs at the beginning—see Figure 3-11.

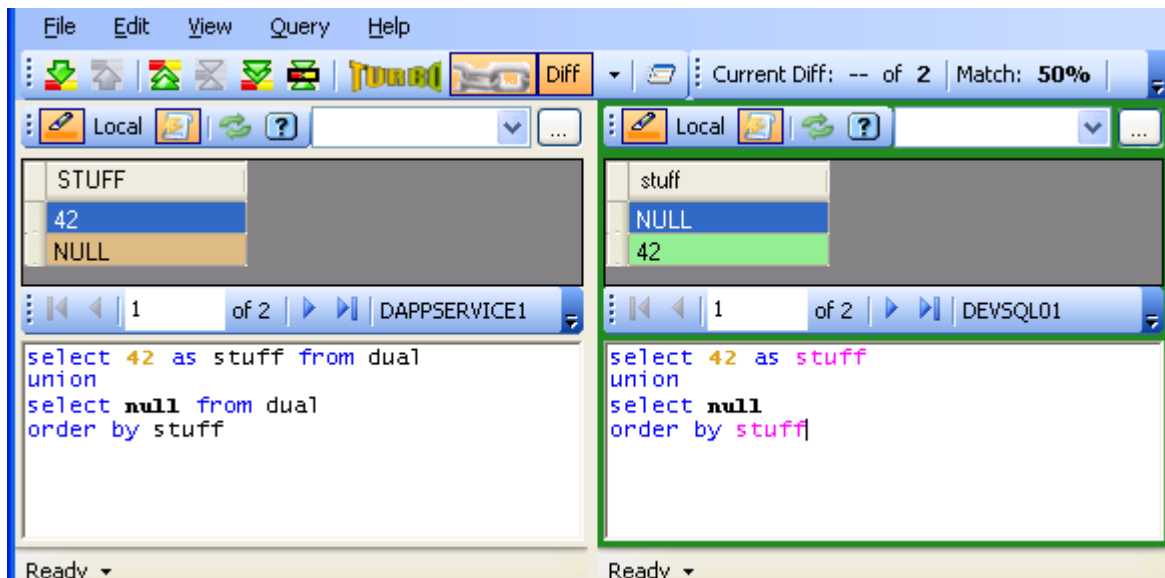


Figure 3-11 Native Sort Order of NULL
 Oracle sorts NULL at the bottom while SQL Server sorts NULL at the top.

Applying turbo-sorting allows you to compensate for this difference: since turbo-sorting sorts both result sets locally, the sort rules are the same. The .NET framework sorts NULLs at the beginning—see Figure 3-12.

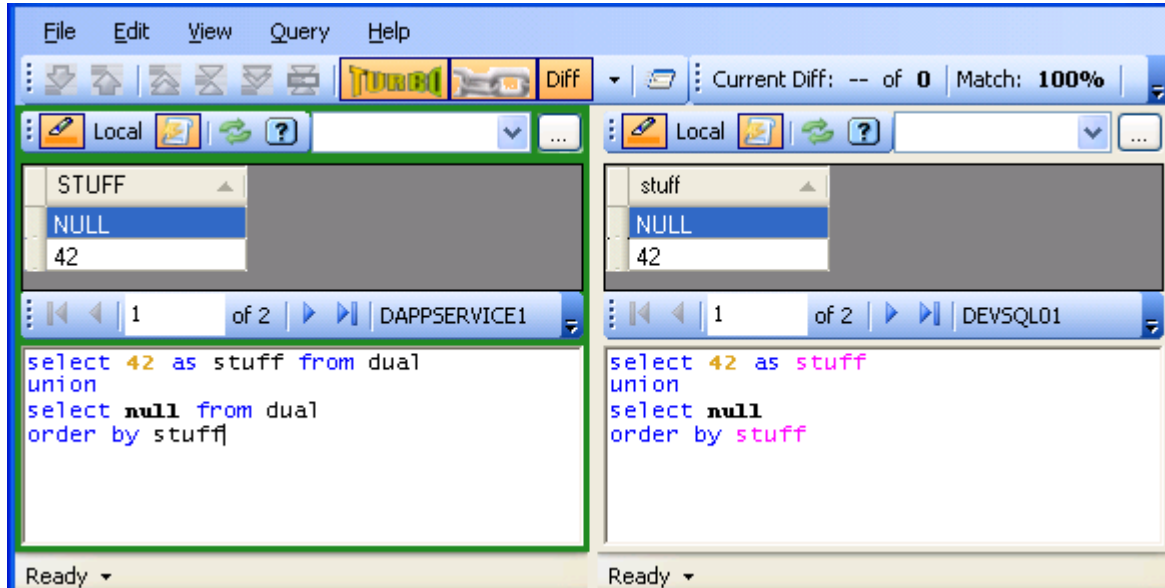
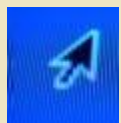


Figure 3-12 Turbo-sorting To Compensate for NULLs

By applying turbo-sorting both result sets will put NULL values at the top of the data.

3.5.4 Simple Sorting

If you have need on occasion to just sort by a single column, there is no need to modify your query and re-execute it: just click on the column header for the field you wish to sort. SqlDiffFramework re-sorts your data by that column and immediately re-compares it to its partner editor pane.



TIP: SqlDiffFramework re-compares immediately upon sorting which is useful sometimes but may not be optimal in some cases. If you have sizable result sets and want to re-sort both sides, first disable differencing, click the appropriate column header on each side to sort both sides, then re-enable differencing to re-compare the data.

3.5.5 Turbo-sorting

Turbo-sorting compensates for any lexicographic differences between two databases by re-sorting both results according to the specifications in their respective queries. To turn turbo-sorting on or off you simply press the turbo-sort button (**Turbo**) – see 4.1.1.7. But you must also indicate how you want to sort. SqlDiffFramework leverages your existing order by clause when turbo-sorting. Say you execute two partner queries with native sorting then realize that there are case sensitivity differences you want to compensate for. Just press the turbo-sort button to turn on turbo-sorting and re-execute both queries. The result sets will sort on precisely the same criteria: the elements specified in your order by clause but this time with internal sorting. If you want to change the ordering with turbo-sort, edit the order by clause and re-execute.

Turbo-sorting has an additional benefit: it allows you to use the **Quick Find** feature to move around the results grid – see section 4.5.1.10 for more details.

Turbo-sorting is more particular about the elements in the `order by` clause. Each element must refer to a *display name* in the result set grid rather than to a *database field name*. The display names are determined from the elements of the select clause. Any compound expressions (e.g. `len(name)`) must therefore be aliased. Though, for example, SQL Server could process this query:

```
select name, len(name)
from accounts
order by len(name)
```

...SqlDiffFramework can only use the actual column names in the result set. Since an unaliased column such as that shown above will receive a generic *Column n* label, it could not easily be associated with the compound expression. The solution is to require an alias as in:

```
select name, len(name) as myLength
from accounts
order by myLength
```

This allows SqlDiffFramework to convert the specified sort expression of `len(name)` to a realized sort expression of `myLength`. Note that most databases (SQL Server and Oracle, for example) allow you to use *either* the actual expression *or* the alias in the `order by` clause itself, so SqlDiffFramework allows this as well. Thus this will also work because the column name is derivable from the original alias:

```
select name, len(name) as myLength
from accounts
order by len(name)
```

There are some drivers, however (notably the Microsoft Jet driver used by Access as well as ODBC/CSV connections) that do *not* allow aliases in the `order by` clause. In those situations, you must *define* the alias but then you must *use the original expression* in the `order by` as in the last example above.

Some valid SQL code will not quite work properly due to a limitation of SqlDiffFramework. An expression such as this:

```
SELECT users.name
FROM users
ORDER BY users.name
```

...is fine for a SQL interpreter but SqlDiffFramework operates on column names or aliases to column names. Since the resulting column from the above example would be `name` SqlDiffFramework cannot match the element `users.name` in the order by clause to it. The workaround to this is simply to alias any compound names such as `users.name` to a simple name, as in:

```
SELECT users.name as name
FROM users
ORDER BY users.name
```

Table 3-4 illustrates a sampling of statements that all work in SqlDiffFramework.

Description	Expression
Unaliased simple name	SELECT name FROM NameTable ORDER BY name
Aliased simple name ordered by alias	SELECT name as myName FROM NameTable ORDER BY myName
Aliased simple name ordered by name	SELECT name as myName FROM NameTable ORDER BY name
Aliased compound name ordered by alias	SELECT table.name as myName FROM NameTable ORDER BY myName
Aliased compound name ordered by compound name	SELECT table.name as myName FROM NameTable ORDER BY table.name
Aliased expression ordered by alias	SELECT len(name) as myNameLength FROM NameTable ORDER BY myNameLength
Aliased expression ordered by expression	SELECT len(name) as myNameLength FROM NameTable ORDER BY len(name)
Delimited simple name ordered by non-delimited name	SELECT [name] FROM NameTable ORDER BY name
Delimited simple name ordered by delimited name	SELECT [name] FROM NameTable ORDER BY [name]
Aliased, delimited compound name ordered by delimited, compound name	SELECT [nt].[name] as myName FROM NameTable nt ORDER BY [nt].[name]
Aliased, delimited compound name ordered by alias	SELECT [nt].[name] as myName FROM NameTable nt ORDER BY myName

Table 3-4 Syntactical Variations Acceptable to Turbo-Sorting

SqlDiffFramework is flexible in field name delimiters: you may use square brackets as shown in the table as well as single quotes, double quotes, or back quotes. Since different databases have different requirements on these, SqlDiffFramework allows the superset.

Section 3.5.2 stated that when loading local data (CSV snapshots) instead of live data the query is effectively ignored. That is not quite true: with turbo-sort enabled SqlDiffFramework processes the `SELECT` clause to determine the display names and the `ORDER BY` clause to determine sort order, just as it does with live data.

Turbo-sorting uses a simple parsing technique for identifying the `SELECT` and `ORDER BY` clauses. Complex queries—those with sub-queries or those containing multiple queries—could result in an incorrect determination, which typically manifests as a status bar message indicating missing aliases for certain fields. As one example, the parser expects that the list of fields in the `ORDER BY` clause is comprised of everything from the `ORDER BY`

literal until the end of the query text. This code violates that, by having a `DROP TABLE` statement as part of the cleanup at the end of the query:


```
SELECT field1, field2, field3
FROM #myTempTable
ORDER BY field1, field2, field3
DROP TABLE #myTempTable
```

You can provide a hint for such cases where it cannot properly identify the `ORDER BY` clause by including special comments demarcating the `ORDER BY` clause. Note that the hints are embedded in end-of-line comments but that the comment token is doubled (4 hyphens rather than 2):

```
SELECT field1, field2, field3
FROM #myTempTable
---- [ORDERING START]
ORDER BY field1, field2, field3
---- [ORDERING END]
DROP TABLE #myTempTable
```

Similarly, you can provide a hint for the `SELECT` clause by itself, or in conjunction with the hint for the `ORDER BY` clause. You must include at a minimum the `SELECT` and `FROM` keywords:

```
---- [SELECT START]
SELECT field1 as ClientName, field2 as ClientNumber, field3
FROM #myTempTable
---- [SELECT END]
---- [ORDERING START]
ORDER BY field1, field2, field3
---- [ORDERING END]
DROP TABLE #myTempTable
```

SqlDiffFramework provides a mechanism that lets you verify your sorting criteria and alias matching, assisting you in determining whether you need to provide hints or not. Depress the `Alt` key when you press the Execute button () and after your query executes you will see your sort fields and aliases added to the bottom of your query as comments. The above query, for example, would yield this:

```
-- SORT FIELDS: field1[ASC], field2[ASC] , field3[ASC]
-- ALIASES: field1=>ClientName, field2=>ClientNumber
```

3.6 Working With Queries and Snapshots

3.6.1 Organizing Your Files

Figure 3-13 illustrates one convenient directory structure to store both your SQL queries and saved snapshots. It is not actually necessary that the two SQL directories be direct siblings, but rather it is important to have separate directories for them. Similarly, the two snapshot directories should be separate from each other rather than comingled.

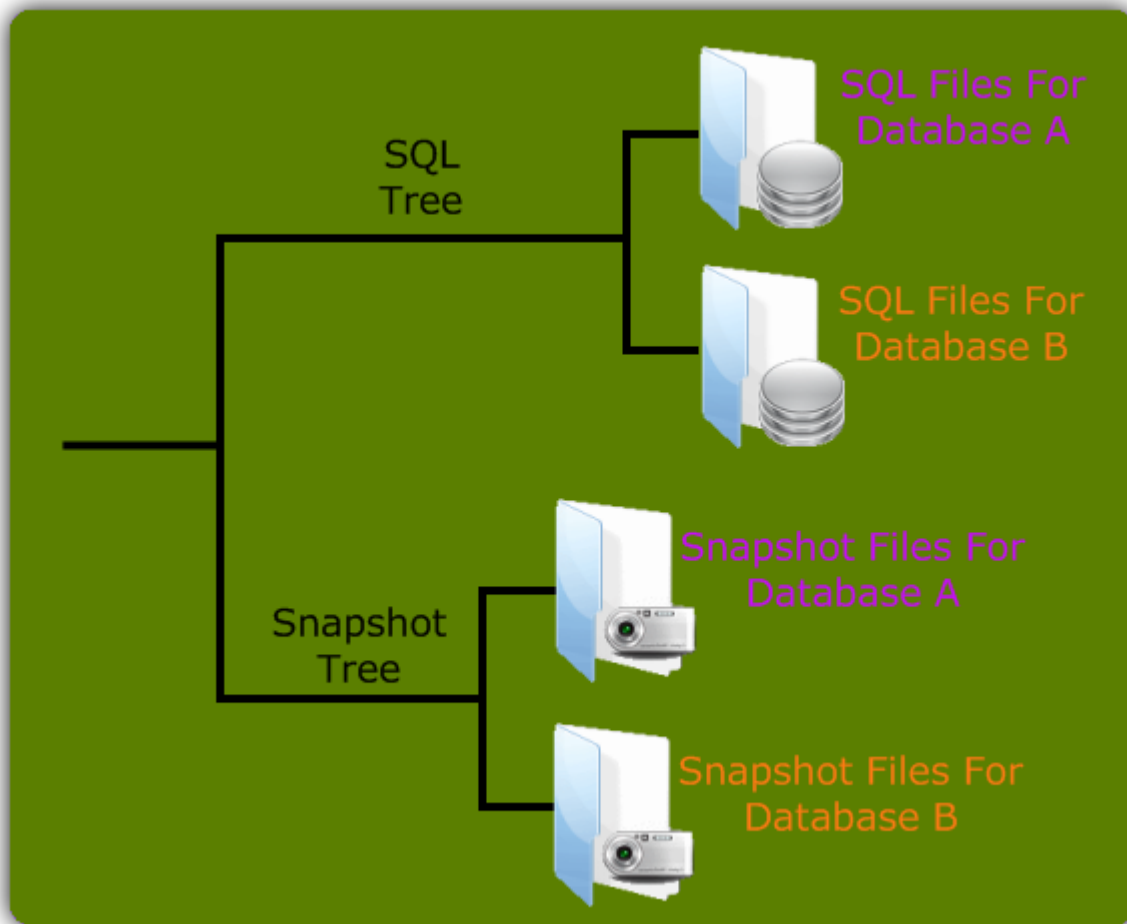


Figure 3-13 Directory Structure

Separating SQL scripts and snapshots between target databases allows you to leverage the time-saving operations of tandem mode.

Within the directory containing SQL queries for database A you will have a set of scripts to query database A. Since SqlDiffFramework is all about comparison, each of these needs something to compare to, which will be the corresponding script for database B stored in its directory. The number of scripts in the two directories, therefore, should typically be the same. Furthermore, the *names* of the scripts in the two directories should be the same. So if

you have a query entitled `CheckStringLengthsForPartsTable.sql` for database A, the query for database B should use exactly the same name. When you keep the names the same, you can then make best use of SqlDiffFramework's *tandem mode*. With tandem mode enabled, many operations that you perform on one editor pane will automatically be mirrored in the other. If you scroll down the grid on the left, for example, the right scrolls down the same amount. More to the point here, if you open `CheckStringLengthsForPartsTable.sql` on the left for database A, then the corresponding query for database B of the same name is automatically opened on the right.

Not only should you use the same names for queries between A and B, but you should also use the same names for snapshots as you do for queries as this will similarly streamline loading snapshots. Thus, when you save a snapshot for the same query, name it `CheckStringLengthsForPartsTable.csv` for both result sets. Having these four files with identical base names, you can then load two queries and two result sets with a *single* file dialog instead of four separate file dialogs!

As a final tip, the reason that the SQL tree and snapshot trees are separate in Figure 3-13 rather than having the four leaf directories all in one subdirectory is for ease of archiving. The SQL queries are your source code files, which you may want to manage in a version control system, while the CSV files tend to be more transient. By having the trees separate you can more easily operate on just the files you need.

3.6.2 Exporting to Excel

Sending a result set to Excel is one of two techniques for exporting data from SqlDiffFramework. Once you have executed a query against a live data source, open the context menu for the result set by right-clicking in the header row of the result grid (see also 4.5.1.5). The top-level menu will contain an **Export to Excel** menu item but *only* if the application can detect that it is installed on your system. Selecting this Excel export sends the result set directly to Excel; it does not save it to an Excel file. If Excel is already running, it just opens a new window with the data. If Excel is not running, SqlDiffFramework launches and opens a new worksheet with the data.

My initial development environment for SqlDiffFramework was Windows XP with Excel 2003. Though I had two systems with presumably the same environment, one showed the Excel menu choice while the other did not. From my reading, this may have come about by the order I installed programs (Office vs. the .NET framework). If you do not have the menu item to export to Excel and you want it, first check your installed assemblies (see the last frame of Figure 3-14). You might have some or none of the interop assemblies showing up in your assembly cache. You can easily update your Office installation through **Add/Remove programs**, as diagrammed in the figure. Select the **Add or Remove Features** options then make sure you check the box for advanced customization. Turn the **.NET**

Programmability Support option on, close out of the installer, and you should then have the assembly after the installer finishes the minor update. (Thanks to Gill Cleeren at <http://www.snowball.be/MicrosoftOfficeInteropExcel+Not+Found.aspx> for this tip.) Another way to get to the same point, I believe, is by manually installing the OfficeXP primary interop assemblies from <http://support.microsoft.com/kb/328912/> .

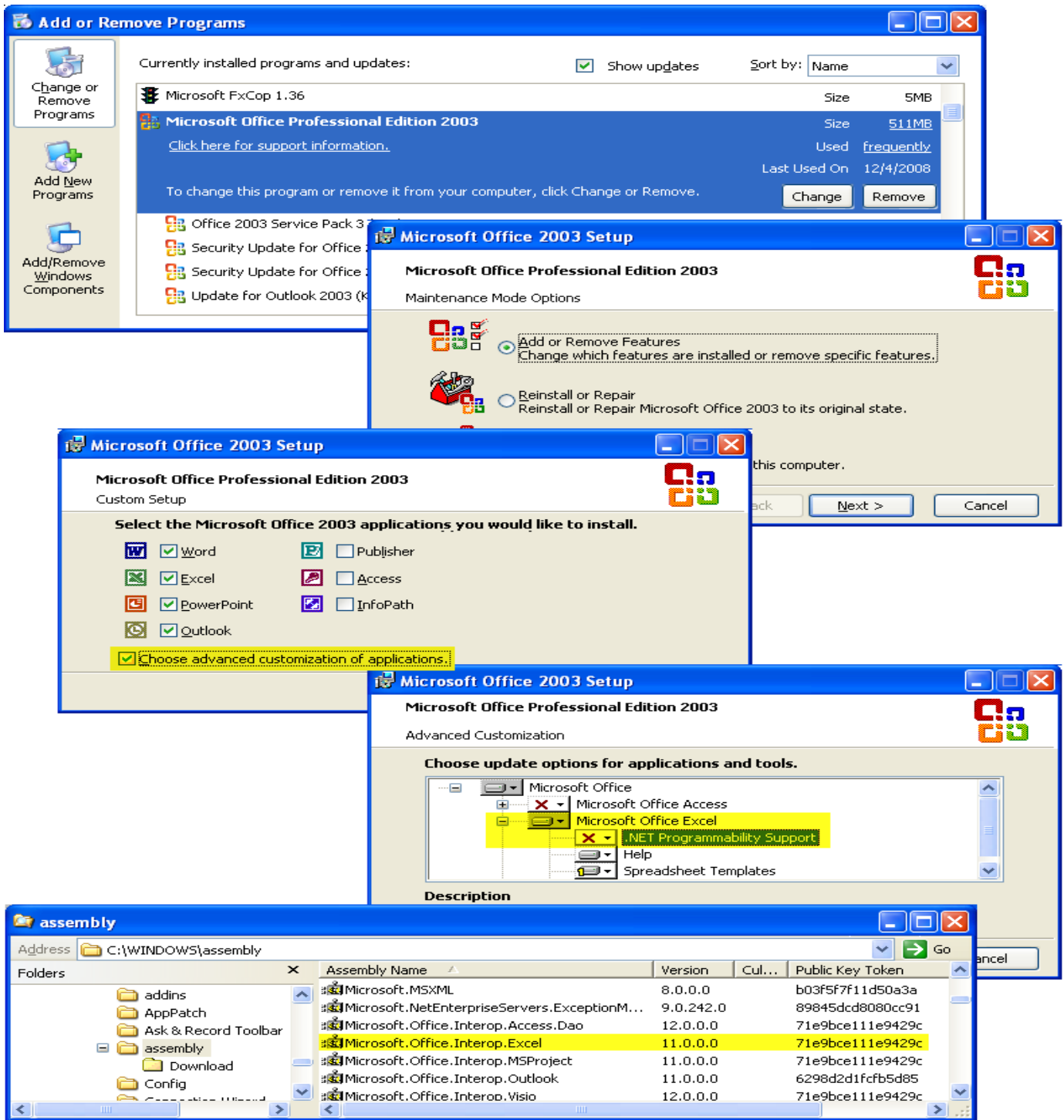

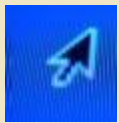


Figure 3-14 Ensuring Excel Interoperability
 If you do not have the DLL for Excel (bottom frame), open your Office installer from Add/Remove Programs and navigate to the .NET Programmability Support option for Excel as shown. Update that item and you should then have the assembly showing up in your assembly cache.

3.6.3 Saving Snapshots

Saving a snapshot is the other technique for exporting data from SqlDiffFramework. Use the Save Grid button () to generate a snapshot. SqlDiffFramework prompts you with a standard **Save File Dialog**, allowing you to specify the path and file name (see 4.3.2.6). A snapshot is stored as a CSV file so it may be read by any application that understands CSV format. Note, however, that CSV files generated by SqlDiffFramework have extra information in the headers to allow restoring the data in a strongly-typed fashion (integers, dates, etc.) rather than restoring all data as strings. Loading CSV files generated by any other means does render all columns as just strings.



TIP: You can sometimes tell just by looking whether numbers in the grid, for example, are represented as numbers or as strings. But SqlDiffFramework can also reveal a data type for you at any time: Depress `Control+Shift` or `Control+Alt` then hover over the column header you are interested in. See 4.5.1.2 for an illustration.

A normal CSV file often includes (though it is not required to) the display names as the first row of the file. A typical row appears like this:

```
IdCustomer,ClientName,CreateDate
```

The same header row when emitted by SqlDiffFramework to a decorated CSV file, appears like this:

```
IdCustomer:System.Decimal,ClientName:System.String,CreateDate:System.DateTime
```

More generally, a header row follows this grammatical construction:

```
<header-row> ::= <element>,<element>,<element>,. . .
```

For a normal CSV file these are just display names:

```
<element> ::= <display-name>
```

But for a decorated CSV file, an element is defined like this:

```
<element> ::= <display-name> ':' <.NET-data-type>
```

When you load a decorated CSV file back into SqlDiffFramework the data types are used internally but not displayed. Loading the same file into Excel or some other application that reads CSV files, will actually show the full `<element>` values as the column headers, including the data types.

3.6.4 Retrieving Snapshots

To SqlDiffFramework, snapshots provide another data source, one that gives you the ability to track changes from a live data source over time. One scenario where this is very useful is a development cycle, where there are multiple releases to test over a short time span. Take a snapshot before each release to obtain a baseline of your data then compare early and often during your test cycle. This allows you to identify anomalies very soon after they appear, giving you the best chance to identify who and what caused it.

Each editor pane operates in either *live* mode or *local* mode. When you execute a query in live mode, the query is sent to the database (Oracle, MySQL, SQL Server, or ODBC data source) and the database returns a result set. When you switch to local mode with the local/live mode switch (see 4.3.1.2) and then execute a query, SqlDiffFramework loads the snapshot corresponding to the current query. If it cannot properly identify a correlated snapshot (see 3.6.1) SqlDiffFramework then prompts you to select a file.

Either editor pane may be in live or local mode separately. You therefore have three operating modes as shown in Table 3-5.

Mode	Use
Live vs. Live	Compare current data from two different databases
Local vs. Local	Review the history of two different databases over time (assumes you have saved a collection of snapshots for each)
Live vs. Local	Analyze the current state of a single data source compared to its progression over time (assumes you have saved a collection of snapshots)

Table 3-5 Comparing Live and Local Data Sources

SqlDiffFramework provides a convenience feature when you want to analyze a single data source on both sides comparing current data with a snapshot, two different snapshots, or any other analysis you might want to do on the same data source. First set up one editor pane with mode settings, directory paths, and connection settings, even loading a query if you wish. Then invoke the **Query → Mirror Query** command to copy all those settings to the partner pane – see 4.1.7.5.

3.7 Working With ODBC Data Sources

With ODBC connections, you can extend the reach of SqlDiffFramework to reach into Excel spreadsheets, CSV or other text files, Access database files, and more. The following sections describe in detail two of these common data sources, CSV files and Excel files.

You will see shortly (Figure 3-15) that you can query ODBC data sources just like you can query a database. As you probably know, SQL Server uses T-SQL, Oracle uses PL/SQL. ODBC data sources use yet another SQL variation, Jet SQL, which uses Microsoft's Jet Database Engine (http://en.wikipedia.org/wiki/Microsoft_Jet_Database_Engine) developed way back in 1992, and most generally known as the language of Microsoft Access. The official Microsoft reference for Jet SQL is available at <http://office.microsoft.com/en-us/access/CH062526881033.aspx>. But perhaps an even more useful reference is this T-SQL to Jet SQL cross reference, posted on a blog by Jeff Smith: <http://weblogs.sqlteam.com/jeffs/archive/2007/03/30/Quick-Access-JET-SQL-to-T-SQL-Cheatsheet.aspx>. Here you will learn, for example, that instead of `substring` you must use the `mid` function.

What I just stated is fine in theory but not quite as straightforward in practice. Perhaps it is just that my knowledge of which JET version applies to which type of ODBC connection is lacking. I found, for example, that though this cross-reference says JET queries use double quotes for strings, Excel 2003 still requires single quotes, just like T-SQL. Similarly, Excel 2003 uses the percent character for wildcards, not the asterisk character. I suspect that more recent versions of JET SQL have just been brought more in line with standard SQL, and this is not reflected on that web page.

3.7.1 CSV Data Sources

You can reload a saved snapshot of a live data source by toggling the local/live mode switch then refreshing, as described in the previous section. But snapshots are just a special case of static data that may be retrieved as live data using an ODBC connection. ODBC connections allow you to treat Excel spreadsheets, CSV files, and other objects as live data sources. For the case of saved snapshots this gives you more flexibility because you can engage the data source with a query, whereas just reloading a snapshot, as you learned, effectively ignores the query and just loads the whole thing.

You define an ODBC connection for a **CSV file** using the standard ODBC Administrator utility—see Appendix 5.5. For the case of a CSV file, you specify a target *directory* in the ODBC definition. Next, use the connection editor (section 3.9) to apply the connection within SqlDiffFramework's context: specifying a data source type of ODBC enables the `Data Source` dropdown selector, which shows all the ODBC connections defined on your machine. Finally, once you close the connection editor select the connection by name in the

connection selector to point the editor pane to that data source. You can then invoke a query to retrieve data from any CSV file in the directory pointed to by that ODBC connection. Examples:

```
select * from "myfile.csv"

select ServiceCode as Code, Description, [User Name] from [test file.csv]
```

If you specify an invalid field name, the status bar indicates this with a partially useful error message—it is not quite clear what “Expected 1” means:

```
Error "[07002][Microsoft][ODBC Text Driver]Unknown field(s). Expected 1."
```

But note that the *same* error message is used for other reasons—for example, if the CSV file is in Unicode format!

3.7.2 Excel Data Sources

You define an ODBC connection for an **Excel file** using the standard ODBC Administrator utility—see Appendix 5.5. For the case of an Excel file, you specify a target *spreadsheet* in the ODBC definition. Next, use the connection editor (section 3.9) to apply the connection within SqlDiffFramework’s context: specifying a data source type of ODBC enables the **Data source** dropdown selector, which shows all the ODBC connections defined on your machine. Finally, once you close the connection editor select the connection by name in the connection selector to point the editor pane to that data source. You can then invoke a query to retrieve data from any sheet, or any named range, within the specified Excel workbook.

By default Excel creates worksheets Sheet1, Sheet2, and Sheet3 in a new workbook. To access, for example, all the data on Sheet2 use (note the trailing dollar sign):

```
select * from [sheet2$]
```

If your sheet contains disjoint areas, or you are only interested in a portion of the data on a sheet, assign a name to the target range within Excel (**Insert → Name → Define** in Excel 2003). Named ranges may be referenced just like ordinary table names from other data sources:

```
select [Name] as [Full Name], Created, [Quantity] as "Qty"  
from [top10table]  
order by Name
```

For further information on reading Excel data through an ADO interface, see *ExcelADO demonstrates how to use ADO to read and write data in Excel workbooks* at <http://support.microsoft.com/default.aspx?scid=kb;en-us;278973>.

3.7.3 ODBC Example: Compare Two Excel Files

Microsoft Word provides a powerful differencing utility to compare two versions of a document and find out exactly what has changed. This is extremely useful, yet the same facility does not exist for Microsoft Excel. You have seen above how

In this example, the goal is to find differences in two versions of a spreadsheet, named `Book1-NEW.xls` and `Book1-OLD.xls`. Two ODBC data sources are created with similar names, `Book1-NEW-data` and `Book1-OLD-data`. Inside SqlDiffFramework, the **Connection Editor** then creates two connections, `book1-NEW` and `book1-OLD`. Each side of the main SqlDiffFramework is then set to one of these connections from the drop-down selector. SqlDiffFramework conveniently displays both the connection name and the data source name, as indicated in the figure. Finally, the same query is used on both sides, referencing all fields from the first sheet in the Excel file revealing, in this case, the single cell that is different.

You will note that the processes for CSV files and for Excel files described above are largely the same. Figure 3-15, then, while using an Excel example, is largely applicable whatever data source you use.

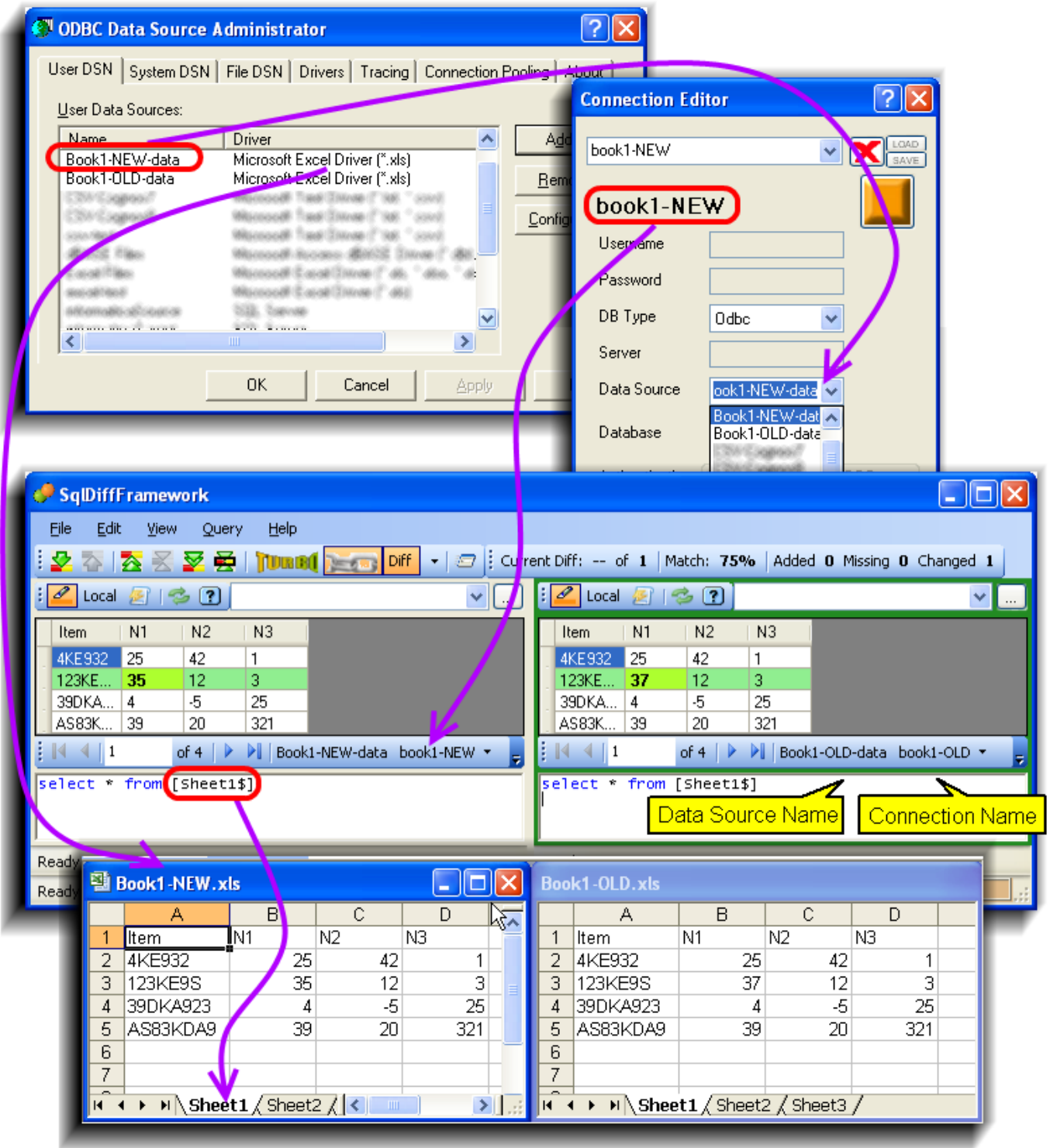



Figure 3-15 ODBC Example with Excel

Create an ODBC connection named Book1-NEW-data to point to the Book1-NEW.xls spreadsheet. Create a SqlDiffFramework connection named Book1-NEW to point to the ODBC connection. Reference worksheets and columns in the originating spreadsheet by making the Excel ODBC connection active.

3.8 Working in Tandem: Do Two Things At Once

With tandem mode enabled, many operations that you perform on one editor pane will automatically be mirrored in the other. If you scroll down the grid on the left, for example, the right scrolls down the same amount. Enable or disable tandem mode with the Tandem button () on the main toolbar. Whether you have tandem mode enabled or disabled you may toggle the setting for a solitary operation by holding down the **Control** key when you invoke a tandem-aware operation. For example, with tandem mode enabled using the scroll arrows in the result grid on one editor pane adjusts the partner editor pane in the same way. If you want to scroll only one editor pane, depress **Control** first then use the scroll arrows to adjust only the single grid. Similarly, with tandem mode disabled you can still scroll both grids by holding **Control** before using the scroll arrows.


Some tandem actions are initiated by on-screen controls appearing in both editor panes. Therefore if tandem mode is disabled the control will operate only in its own editor pane. But other tandem actions are initiated by menu commands, raising the question of which editor pane the command affect with tandem mode disabled. In such cases, the action applies to the *most recently active* editor pane, the one where you were last typed or clicked or even just hovered over with your mouse. For example, the **Open File** menu item, in tandem mode, opens files for both editor panes. With tandem mode disabled, if you had just typed in the left editor pane it opens a file in that pane.

It is generally better to use the corresponding keyboard shortcut rather than using the mouse to open the menu and select a menu item. Consider, for example, if you are working in the right-hand editor pane. You move the mouse across to the menu bar, but happen to just cross into the left-hand editor pane as your mouse travels. That makes the left-hand editor pane the most recently active one, so by the time you open the menu, the **Open File** command will act on the left-hand pane.


3.8.1 Tandem Operations

Section 5.4.2 shows a quick reference chart for the editor panes that indicates which operations are tandem-aware. Details of these are given below, indicating tandem-aware and solitary operations where appropriate.


3.8.1.1 Load file

Load a new file either with on-screen controls—typing a file path into the current file selector, selecting a previously loaded file from the current file drop down, or using the file picker button ()—or menu command (or its keyboard shortcut). See section 4.1.4.3.

3.8.1.2 New file

The on-screen New File button ()—and its related actions—are all tandem-aware, as is the associated menu item. See section 4.1.4.2.

3.8.1.3 Execute query

Execute a query from the menu, its keyboard equivalent, or from the Execute/Refresh button (). When you edit only one of the two current queries, though, it is natural to just press **F5** to re-execute. But in tandem mode, this will re-execute both queries, which is not strictly necessary. Instead, use **Control+F5** to override tandem mode to solitary mode and re-execute just that one query. Since executing either one or both queries automatically triggers the differencing analysis (assuming auto-differencing is enabled), that is sufficient to re-compare the result set with its partner. But you will have saved not one, but two steps: executing the partner query and scanning the partner data.

3.8.1.4 Toggle local / live mode

Each editor pane has its own local / live mode switch.





3.8.1.5 Enable auto-execute

Each editor pane has its own auto-execute switch. When enabled, as soon as you load a query file by any means it is executed automatically.

3.8.1.6 Filter result set

You may apply a local filter to the result set to focus on some particular data.

3.8.1.7 Scroll result set

You may scroll up or down with the vertical arrow keys ( and ) or **Control+Home** / **Control+End** keys in one grid and have the partner grid mirror the movements. Similarly, the horizontal arrow keys ( and ) scroll wide grids horizontally in one or both grids.

3.8.1.8 Adjust grid row height

You may adjust the height of rows in the result set.

3.8.1.9 Adjust grid column widths

You may adjust the widths of columns in the result set to match the header cells or to match the data cells in one or both grids.

3.8.1.10 Enable auto-highlight

Each editor pane has its own auto-highlight switch. When enabled, syntax highlighting is applied automatically to your queries when you type or when you load a file.

3.8.1.11 Enable case-sensitive syntax highlighting

The query editor highlights SQL keywords either exactly as they are defined or independent of case.

3.8.1.12 Enable forcing keywords to upper or lower case

Recognized keywords may be forced to upper or lower case with this setting.

3.8.1.13 Enable auto-completion

Each editor pane has its own auto-completion switch. When enabled, you can invoke the command-completion facility to select keywords or macros and reduce typing.

3.8.1.14 Enable forcing auto-completed phrases to upper/lower/user case

With this setting, auto-completed words and phrases may be forced to upper or lower case or to match the case of the prefix you entered.

3.8.1.15 Enable expanding tabs to spaces

Change the action of inserting a tab to insert spaces or vice versa.

3.8.1.16 Set number of spaces for tab expansion

Adjust the number of spaces inserted when the tab key is set to insert spaces.

3.8.1.17 Toggle tab and Control+tab actions

Toggle the actions of tab and **Control+Tab** between inserting spaces or tabs and moving to the next control.

3.9 Working with Connections

A connection defines the credentials necessary to connect to a data source.

SqlDiffFramework supports connections to SQL Server, Oracle, MySql, and ODBC. Though a fairly old technology, ODBC broadens the spectrum of available data sources to include Access databases, Excel spreadsheets, plain text files and more.

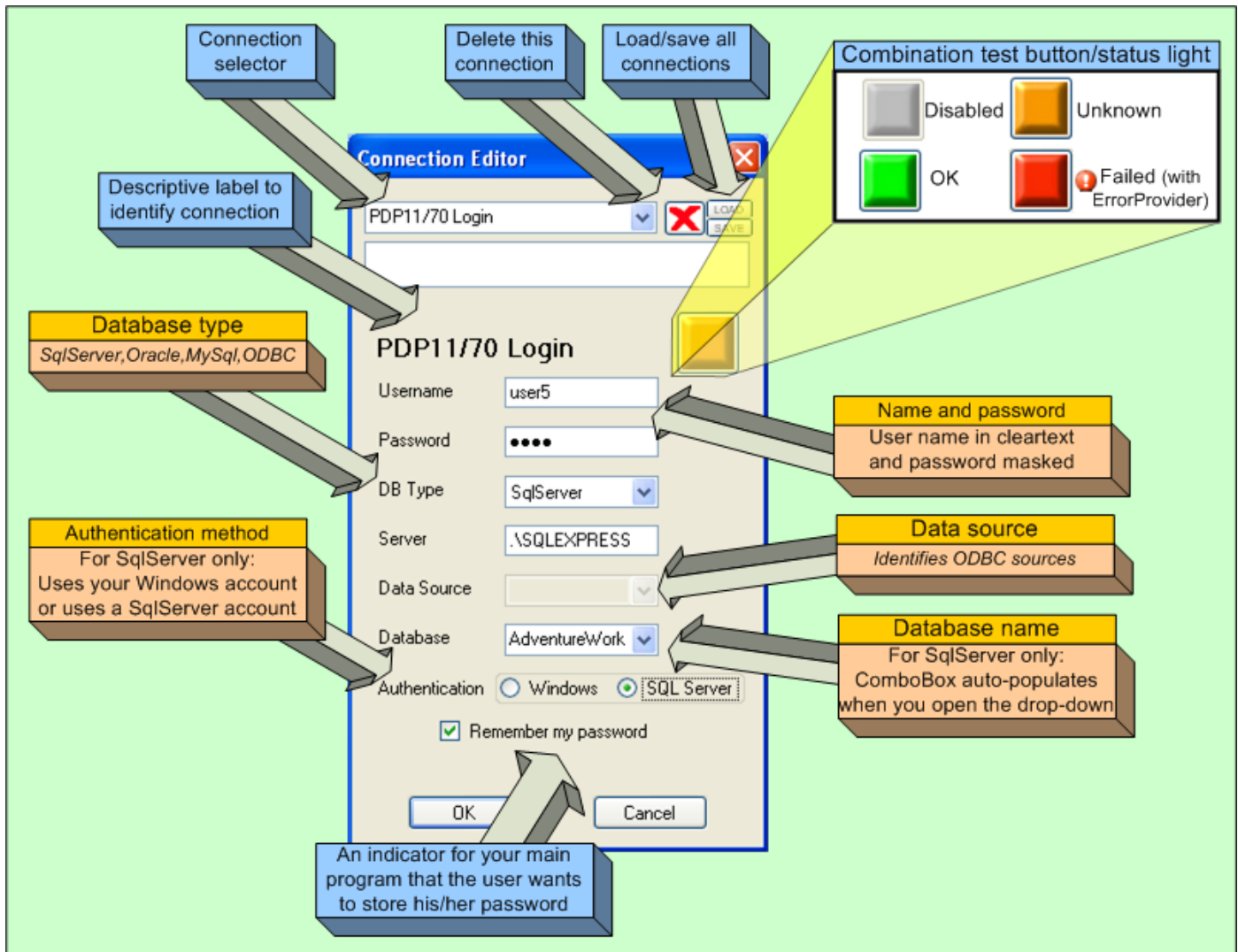


Figure 3-16 Connection Editor Concepts

This exploded view shows the connection editor for a representative data source. Note the particularly useful test button/status light in the upper right.

The combination test button/status light provides a convenient way to check credentials without leaving the dialog. The test button's color provides immediate visual feedback. It starts out as grey (status disabled), and turns orange (status unknown) only after you have entered data in all the appropriate fields. Of course, the list of *appropriate* fields varies based

on your entries. For example, if you select Oracle as the database type then you must enter a username, password, and server, but because Oracle does not have the concept of database names, the database selector is disabled. After the button turns orange, you may click it to attempt to connect to the specified system. When the connection succeeds, the button changes to green; otherwise, it turns red and displays an error indicator (the red exclamation symbol in Figure 3-16).

The exploded view in Figure 3-16 shows a representation of the connection editor for a single data source. Within SqlDiffFramework the connection editor can handle an arbitrary number of data sources and, in fact, lets you import or export a set of connections to port them between different installations. Figure 3-17 shows actual screen shots of the connection editor within the application, illustrating settings for three different types of data source.

With SQL Server, you have the choice of allowing either Windows authentication or SQL Server authentication. Microsoft recommends using Windows authentication whenever possible – see <http://msdn2.microsoft.com/en-us/library/ms143705.aspx>. Windows authentication validates the user's currently logged on Windows account to determine whether the user has access to the given database. SQL Server authentication, on the other hand, uses the account registry contained within SQL Server itself. So if you select Windows authentication, you do not need to provide a name and password. As Figure 3-17 illustrates, the username and password fields become disabled with Windows authentication selected.

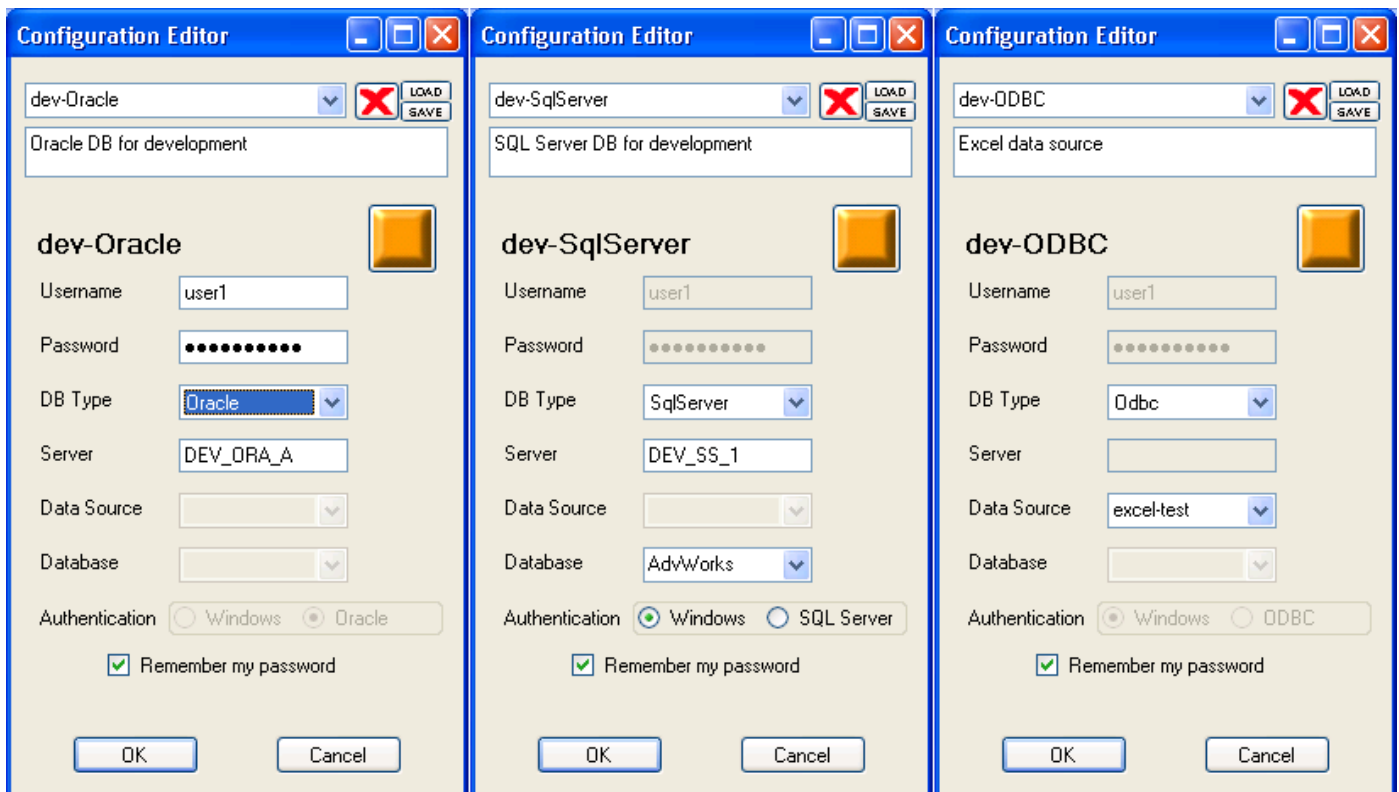


Figure 3-17 Connection Editor Examples

Depending on your choice of database type and authentication, different fields will be enabled. Here you see examples for Oracle, SQL Server, and ODBC data sources.

SQL Server and MySQL have the concept of separate databases on a given server while Oracle does not. Therefore, selecting Oracle disables the database drop-down box while selecting SQL Server or MySQL enables it, as shown in Figure 3-17. Similarly, ODBC data sources require a data source (defined using the ODBC Administrator utility—see Appendix 5.5) while the other database types do not.

You create a new connection simply by typing a new name in the drop-down at the top. While it appears as if you are just changing the name, you are in reality making a copy with all of the settings from the previous instantly copied to the new connection with the new name, allowing you to create a similar one with little effort. Then go ahead and change any or all of the values to suit the new connection. To access previously defined connections, either type an existing name in or select one from the drop-down choices.

To delete a connection use the delete button (✖) in the upper right. Adjacent to that are the **Load** and **Save** buttons allowing you to import or export all of your settings, respectively. Note that when you load a connection set it replaces any existing connections. If you want to instead merge your current set with a set to be imported, you must do that outside the program with any text or XML editor:

- (1) Export your current connection set with the **Save** button.
- (2) With a text editor or XML editor open the exported connection set file.
- (3) Manually merge the second set in the editor.
- (4) Import the combined connection set into the connection editor with the **Load** button.

Once you load a connection set with the **Load** button, that connection set becomes part of your session information that is saved when you close the application and then restored upon your next invocation.

3.9.1 Passwords

3.9.1.1 Session and Persistent Passwords

Both the load and save operations normally do *not* include passwords from your connection set though you may override this. Passwords, if included, are stored encrypted and the encryption key ties to the specific machine. Thus it is not possible to transfer a set of encrypted passwords to an installation on a *different* computer. If, however, you wish to make a backup or restore from a backup on the *same* machine, the connection editor is flexible enough to allow you to include passwords simply by pressing the **Control** key when you click **Load** or **Save**.

Passwords are only saved with your session information for those connections where you have checked the **Remember my password** box near the bottom of the connection editor. If you do not check the box, the passwords you enter in the connection editor are remembered only for the duration of the current session and disappear when you close the application.

Storing passwords in your connection set is generally an unsafe practice. It is useful and common, though, in development or test environments, where SqlDiffFramework is often used. The next section explains how to require passwords at runtime.

3.9.1.2 Ephemeral Passwords

If you do not enter a password in the connection editor for a connection that requires one (see next section), then the first time you select that connection SqlDiffFramework pops up a login dialog (Figure 3-18). Information entered in this login dialog lasts for the duration of the current session only. If you close the application and restart it, SqlDiffFramework will present the same login dialog during startup.

Since the login prompt appears automatically, and only when the information is missing, you need to be able to force it to appear in case you want to make a change. The most common need for this would be if you entered an incorrect password.

To change the username or password for the current connection, depress the **Shift** key and click the current connection name on the connection selector (see 0).

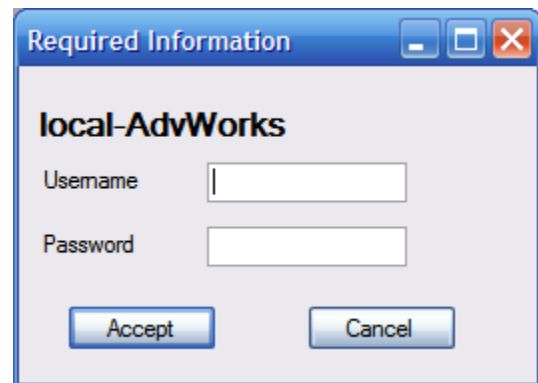


Figure 3-18 Login Dialog

This window pops up when you select a connection that needs a password but does not have one stored in your connection set.

To switch to a different connection and force the login prompt at the same time, depress the **Shift** key while selecting a different connection from the dropdown selector of the connection selector.

3.9.1.3 Determining When a Password is Required

The previous section indicated you will receive a login prompt for connections that require one.

A password is *required* if the connection specifies:

- The database type is Oracle
— or —
the database type is ODBC and the ODBC underlying referent is Oracle.
- The database type is MySQL
— or —
the database type is ODBC and the ODBC underlying referent is MySQL.
- The database type is SQL Server with SQL Server authentication specified
— or —
the database type is ODBC and the ODBC underlying referent is SQL Server with SQL Server authentication specified.

Or, to look at the converse, a password is *not required* when:

- The database type (or the ODBC-derived type) is SQL Server with Windows authentication specified.
- The database type is ODBC and the underlying referent is Excel, Access, text file, etc.

3.10 Working with Meta-Queries

Meta-queries are queries that provide information about your data (or database) rather than providing the data itself, queries that might display the database version, list available databases, find specific tables or columns, or find non-empty tables. The two main sub-sections that follow describe how to use meta-queries and then how to modify or create new meta-query templates. (Some of the material for this section first appeared in my published three-part article series starting with *A Unified Approach to Multi-DataBase Query Templates* available at <http://www.simple-talk.com/dotnet/.net-tools/a-unified-approach-to-multi-database-query-templates/>).

3.10.1 Using Meta-Queries

Each of these queries is specified separately for each supported database type (currently SQL Server, Oracle, and MySql). Some — such as identifying the database version — require no input while others require you to specify parameters like a table name or a column name. Each meta-query, therefore, is actually provided as a *meta-query template*. Each template contains place holders of the form `{ name }` for any parameters that you need to supply at runtime. When you select a specific meta-query, all the place holders in its template are mapped to input fields displayed one per line, ready for you to enter values.

3.10.1.1 Invoking a Meta-Query

Once you open the meta-query selector, drill down through the query tree on the left to find the meta-query of interest. Say for example, you wanted to find all tables that contain a column named `IdCustomer`. Select an appropriate meta-query from the Columns category and the dialog generates input fields for each of the relevant place holders. Note that most queries are

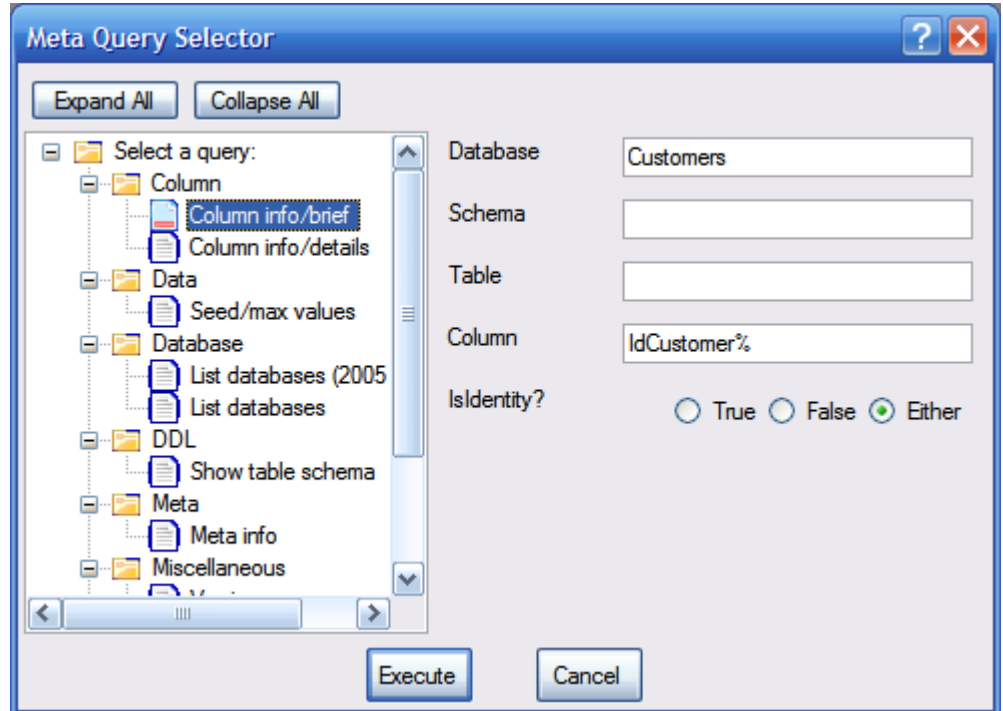
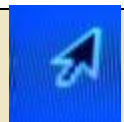


Figure 3-19 Meta-Query Selector

The dialog (showing a SQL Server-specific meta-query) presents queries available for the current database type in a tree on the left. Selecting one generates input fields required by that query on the right.

available for each database type but they may generate different field lists. Oracle, for example, includes the concept of a schema owner while SQL Server includes the notion of multiple database names. Thus the database field shown in the SQL Server meta-query in Figure 3-19 is omitted from the equivalent Oracle meta-query.



TIP: The meta-query selector remembers any values you enter for each editor pane. Say, for example, that after entering the values in Figure 3-19 you switch to the `version` meta-query. This query takes no parameters so the input field list is removed from the dialog. If you then switch back to the Column Info meta-query, the fields are restored with the previous values you entered. Even if you close the meta-query selector and later return to it within the same session, SqlDiffFramework will redisplay your entered values.

Fill in just as much information as you wish; an unfilled field implicitly matches all possible values. For those fields you do fill in, you may either use an exact literal or a term with SQL wildcards. The example in Figure 3-19 lists columns beginning with `IdCustomer` due to the presence of the (%) character. It matches not only `IdCustomer` but `IdCustomerDisabled`, `IdCustomer_old`, etc. You could broaden the search further with

something like `%Id%` to match any column with Id in its name or restrict it to an exact name by omitting the wildcard. See Table 3-7 for all the SQL wildcards available.

3.10.1.2 Manipulating a Meta-Query

When you press the Execute button (🔄) your values replace the place holders in the template, the query executes, and the database returns the result set. The contents of your query editor are unchanged; you never see the meta-query. Rather, meta-queries are designed to be used as command extensions to SqlDiffFramework. When you invoke the menu command to open a new workspace you do not want the application describing to you how it is going to do that—you just want it to do it. Similarly, when you invoke a meta-query to find specific tables it just reports the tables found. Normally you do not care about how it identified those tables. But SqlDiffFramework is, after all, a utility to work with queries! So unlike regular menu commands, with meta-queries you have the option to see—and manipulate if desired—the contents of a meta-query: simply depress the `Shift` key while pressing the Execute button (🔄). SqlDiffFramework then adds the text of the meta-query to the end of the query editor.

When you direct SqlDiffFramework to add the meta-query to the query editor (by depressing `Shift`) for you to view or change, you have a choice of two representations. The default representation *suppresses* universal matches; the alternate representation *reveals* universal matches. (Invoke the alternate representation by also depressing the `Alt` key.) A *universal match* corresponds to a field that you leave blank in the meta-query selector. As indicated above, leaving it blank is implicitly asking for all possible matches on that field. In the query, a universal match may be specified in one of two ways: implicitly (say nothing about the field in the WHERE predicate), or explicitly (specify that the field should return all values with an expression of the form `fieldname LIKE "%"`). Consider the SQL Server meta-query in Figure 3-19 filled out with the values shown in the figure.

The default representation with universal matches suppressed is:

```
SELECT
    ist.table_catalog as Db,
    ist.table_schema as 'Schema',
    so.name as TableName,
    sc.name as ColumnName,
    sc.colstat as IsIdentity
FROM syscolumns sc
JOIN sysobjects so on sc.id = so.id
JOIN INFORMATION_SCHEMA.TABLES ist on ist.table_name = so.name
WHERE ist.table_catalog LIKE 'Customers'
    AND sc.name LIKE 'IdCustomer%'
    AND so.xtype in ('U','V')
ORDER BY so.name, sc.name
```

The alternate representation with universal matches revealed (and highlighted) is:

```
SELECT
    ist.table_catalog as Db,
    ist.table_schema as 'Schema',
    so.name as TableName,
    sc.name as ColumnName,
    sc.colstat as IsIdentity
FROM syscolumns sc
JOIN sysobjects so on sc.id = so.id
JOIN INFORMATION_SCHEMA.TABLES ist on ist.table_name = so.name
WHERE ist.table_catalog LIKE 'Customer'
    AND ist.table_schema LIKE '%'
    AND so.name LIKE '%'
    AND sc.name LIKE 'IdCustomer%'
    AND sc.colstat LIKE '%'
    AND so.xtype in ('U','V')
ORDER BY so.name, sc.name
```

Why, you may wonder, would you ever want to include universal matches? The answer is: it depends on your working style. Some people prefer filling out forms (i.e. entering values directly in the meta-query selector input fields) while others would rather edit the text of a query directly. For the latter approach, once you select a query in the tree of the meta-query selector just *leave all the fields blank*. Depress **Shift** (to display the query) and **Alt** (to reveal universal matches) and press the Execute button (🔄). This gives you a query with every input field in the `WHERE` predicate formatted as `fieldname LIKE "%"`. Now you are free to modify any one or more of them to pare down the result set.

3.10.1.3 The Meta-Query Library

Table 3-6 describes the meta-queries supplied with the application. They are generally simple in concept but they can be a great aid in learning about a database or just finding something for which you cannot quite remember the details. Also, they allow you to do some interesting searches. Say, for example, you wanted to list all identity columns in all tables in a SQL Server database. The `List columns/brief` meta-query includes an input field `IsIdentity?` that lets you filter a search by this property.

Some of these meta-queries are extremely short; others are deceptively long. Consider the `Show table schema` query: in MySQL this is just four words, while in SQL Server it is a query exceeding 250 lines in length! In either case, having them available effectively as extension commands saves you the effort in looking up how to do them.

Category	Description	SQL Server	Oracle	MySQL
System	Display DB version	Yes	Yes	Yes
	Databases	Yes	NA	Yes
	Constraints	Yes		
	Indexes	Yes		
	Primary keys	Yes		Yes
	Map primary/foreign keys	Yes		
	Procedures and functions	Yes		Yes
	International Characteristics		Yes	
	Session information/brief	Yes		
	Session information/details	Yes		
	Users and Roles	Yes		
	Currently running statements	Yes		
Table	Row/column counts	Yes		
	Non-empty tables	Yes	Yes ¹	
	Space utilization	Yes		
	Object level details	Yes		
	Copy a table	Yes	Yes	Yes
Column	Column info/brief	Yes	Yes	Yes
	Column info/details	Yes	Yes	Yes
Data	Seeds, row counts, and maxima	Yes		
DDL	Show table schema	Yes ²	Yes	Yes

Table 3-6 Standard Meta-Queries

¹ Accuracy depends on when and how often Oracle gathers statistics.

² SS2005 or later; limited to 8000 characters per element.

3.10.2 Meta-Query Templates

The previous sections have described how to use meta-queries strictly as commands, and how to modify them after invoking them. This section presents details about the meta-query templates, giving you all the information you need to modify the templates themselves or add new ones.

3.10.2.1 Template Structure

Section 3.10.1.2 showed a filled out meta-query, i.e. *after* your input data had been applied to the template. Below you see the original template before data is applied. This template has five place holders, highlighted for clarity. Notice that each place holder appears in a predicate with the `LIKE` operator rather than the equality operator. Using the matching operator provides flexibility when you supply input values to fill out the place holders: you

may provide either a specific value for an exact match or include wildcards for a broader match. (See Table 3-7 for all the SQL wildcards available.)

```
SELECT
    ist.table_catalog as Db,
    ist.table_schema as 'Schema',
    so.name as TableName,
    sc.name as ColumnName,
    sc.colstat as IsIdentity
FROM syscolumns sc
JOIN sysobjects so on sc.id = so.id
JOIN INFORMATION_SCHEMA.TABLES ist on ist.table_name = so.name
WHERE ist.table_catalog LIKE '{Database}'
    AND ist.table_schema LIKE '{Schema}'
    AND so.name LIKE '{Table}'
    AND sc.name LIKE '{Column}'
    AND sc.colstat LIKE '{IsIdentity?True/False/Either}'
    AND so.xtype in ('U','V')
ORDER BY so.name, sc.name;
```

Each term surrounded by braces defines a place holder. Each unique place holder drives the generation of one input field in the meta-query selector when you select a query. Observe that each input line in Figure 3-19 derives from the place holders specified in this query. By default, this manifests as a single-line text box for you to type a value. That is the case with four out of the five place holders here. The final one, `IsIdentity?`, maps to a set of mutually exclusive radio buttons representing a Boolean state. Though you could use a generalized text box to type a 1 or a 0 for a Boolean field, it makes more sense to let the system handle this for you so you do not waste time accidentally entering, e.g. “Alaska” when it is meaningless to do so. In this query, the `IsIdentity` column in the result set is a Boolean value indicating whether the named column is an identity column or not. The question mark (?) character in the place holder drives this distinction: any place holder name ending in a question mark generates a similar set of three buttons. The place holder name—and the value displayed as the field name on the meta-query selector—is the portion up to and including the question mark. The remaining optional text specifies the labels of the three buttons (words separated by virgules) indicating a true value, a false value, and either value, in that order. If omitted, the defaults of `Yes/No/Any` are used. You may specify one, two, or three of these labels. For example `{IsIdentity?True/False}` would generate labels of `True`, `False`, and `Any`.

Place holders do not have to be unique in the template. If the query template had, for example, a line like this:

```
WHERE {MyFieldName} is null or {MyFieldName} LIKE "{ProductId}"
```

... then `MyFieldName` would appear as an input field just once; the value you enter in the input form is applied to all occurrences of the `{MyFieldName}` place holder.

The query template shown above is perfectly adequate except that it is not instrumented for universal match suppression (section 3.10.1.2). Here is the template in its final form, including bracketing for universal match suppression. Each place holder is placed within a *suppression region*—a portion of the text surrounded with double square brackets. If the place holder located somewhere within the boundaries of the suppression region receives no input (i.e. you leave the field blank on the meta-query selector) then the suppression region is completely omitted from the final query.

```
SELECT
    ist.table_catalog as Db,
    ist.table_schema as 'Schema',
    so.name as TableName,
    sc.name as ColumnName,
    sc.colstat as IsIdentity
FROM syscolumns sc
JOIN sysobjects so on sc.id = so.id
JOIN INFORMATION_SCHEMA.TABLES ist on ist.table_name = so.name
WHERE [[ist.table_catalog LIKE '{Database}'
    AND ]][[ist.table_schema LIKE '{Schema}'
    AND ]][[so.name LIKE '{Table}'
    AND ]][[sc.name LIKE '{Column}'
    AND ]][[sc.colstat LIKE '{IsIdentity?True/False/Either}'
    AND ]][so.xtype in ('U','V')]
ORDER BY so.name, sc.name;
```

The boundaries of the suppression region are important for two reasons. First, the brackets must be placed so that if either the brackets are removed by themselves, or the brackets and their contents are removed, the remaining text is still syntactically valid. Thus, for the `{Database}` place holder, the suppression region is:

```
ist.table_catalog LIKE '{Database}' AND
```

...rather than just:

```
ist.table_catalog LIKE '{Database}'.
```

Second, the boundaries of the suppression region should be placed so that the query cosmetically has consistent white space whether the brackets are removed by themselves or removed with their contents. The template above consistently follows this pattern (ignoring line breaks in the template for simplicity):

SPACE OUTSIDE

SPACE INSIDE

```
preceding-text [[ist.table_catalog LIKE '{Database}' AND ]][following-text
```

No space after the opening bracket and a single space just before the end bracket ensure there are always single spaces between all words whether any or all of the suppression regions are suppressed or not, shown here:

- With input the region remains in place:

```
preceding-text ist.table_catalog LIKE 'demo_db' AND following-text
```

- When no input is supplied the region is suppressed:

```
preceding-text following-text
```

Notice that the `WHERE` predicate in the previous query is composed of six clauses, but only the first five are in suppression regions. If all the fields are left blank when invoking this query, the five clauses are suppressed leaving the predicate with a single, syntactically valid clause: `WHERE so.xtype in ('U','V')`. But consider what happens if all the clauses are in suppression regions and all are left blank. You would have a `WHERE` with no clauses at all, which is not valid syntax. Since a query template should be built robust enough to handle any combination of inputs, you need to make sure it handles the no-input case as well. A simple technique for this, borrowed from the toolbox of SQL injection, is to add a conjunction with an always true clause, e.g.:

```
WHERE [[isr.routine_catalog LIKE '{Database}'
AND ]][[isr.routine_schema LIKE '{Schema}'
AND ]][[isr.routine_name like '{Procedure/function}'
AND ]]1=1
```

3.10.2.2 Template Storage in the Library File

Each query template is stored in the default query library, an XML data dictionary. A complete dictionary entry specifies not just the text of the query but also: the database type; the query type (which is just a display name for the query); a category (indicating where to place this query in the query tree of the meta-query selector); and an optional tool tip (which displays when you hover over the query in the query tree). Here is the complete entry for the Oracle-variant of the previous query:

```
<Query>
  <DbType>Oracle</DbType>
  <Category>Column</Category>
  <QueryType>Column info/brief</QueryType>
  <ToolTip>Shows just table and column names for matching
columns</ToolTip>
  <QueryText>
    SELECT table_name as TableName, column_name as ColumnName
    FROM sys.ALL_TAB_COLS
    WHERE [[owner LIKE '{Owner}'
      AND ]][[table_name LIKE '{Table}'
      AND ]][[column_name LIKE '{Column}'
      AND ]][NOT regexp_like(table_name, '[$_]', 'i')
    ORDER BY table_name, column_name
  </QueryText>
</Query>
```

The default query library is embedded in the application package. The first time you open the meta-query selector, SqlDiffFramework creates the query file (`QueryLibrary.xml`) by unwrapping the embedded resource. The application stores this and other externalized resources in the Application Data/SqlDiffFramework directory. The location of your Application Data directory depends upon the Windows version you are using; you can identify the precise path using the `Help → About` menu item.

Important note: The query library file is purged and updated each time you install a new version of the application *unless* you have made the file read-only to preserve any customized changes you may have made.

3.10.2.3 Customizing Meta-Queries

Figure 3-21 on the next page describes the steps to go from the meta-query selector and the query template library to a finished query instance. The act of opening the meta-query selector loads it with the set of meta-query templates germane to the current database type, which is determined from the connection editor. Once you select a specific meta-query from the query tree the template for that meta-query is scanned for place holders. Each place holder is turned into an input field on the meta-query selector dialog, either a text box for general input or a set of radio buttons for Boolean input. You then proceed to fill out any or all of the input fields provided. When you press execute, your values are plugged into the template to produce a completed query that is sent off to the current database. (Note that you will not see the finished query unless you depress the `Shift` key when you execute.)

To modify any of the existing meta-queries open the query library file (`QueryLibrary.xml`) in a text editor or XML editor and edit its template. To add a new one you only need to conform to the simple schema in Figure 3-20.

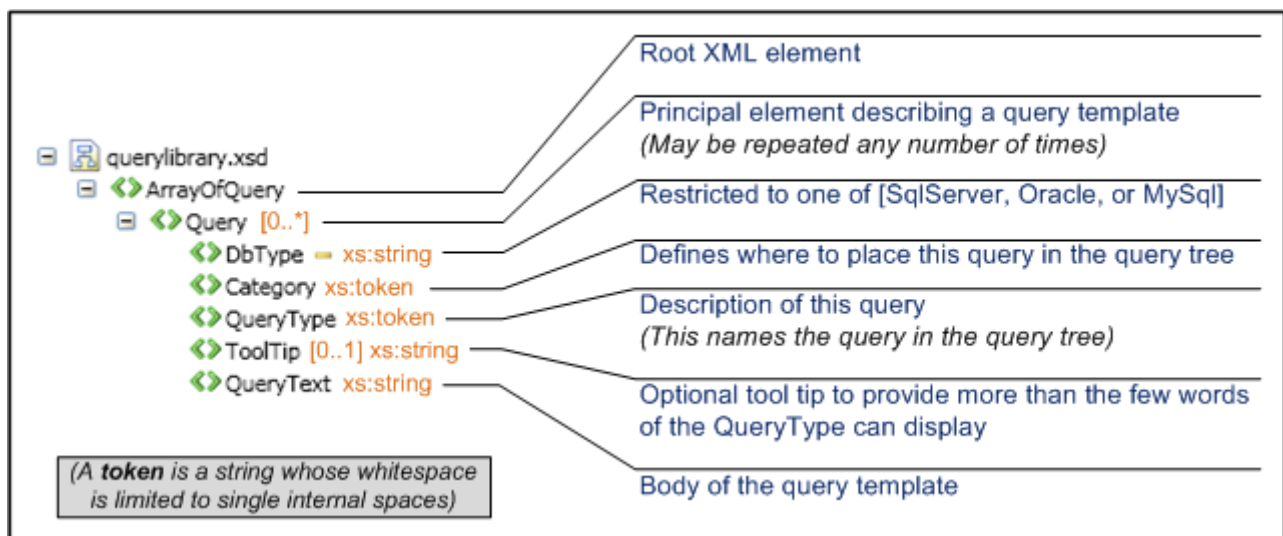
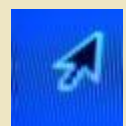


Figure 3-20 Meta-Query Schema

The XML Schema for the query template library file consists of a simple array of queries, where each query contains 4 or 5 elements. The notations describe how these elements are used.

That is, the file consists of an unordered array of `<Query>` elements, each of which must contain a `<DbType>`, a `<Category>`, a `<QueryType>`, an optional `<ToolTip>`, and a `<QueryText>`. The `<Query>` elements may appear in any order in the file. They are organized in the query tree of the meta-query selector (Figure 3-19) first by filtering by the current `<DbType>` (possible values for `<DbType>` are shown in the figure). All the query templates that match are then grouped and sorted by `<Category>` as the first level of nodes in the query tree. Finally, each `<QueryType>` is listed in the bottom level of the query tree in its respective category. As you add new `<Query>` elements you should be consistent with `<Category>` and `<QueryType>` values throughout the file.

Note that if you wish to edit the query library file in an XML-editor that provides guidance based on the schema, the schema file is included in the same directory containing the executable `sqlDiffFramework.exe` file (XML Schema/QueryLibrary.xsd).



TIP: You will find references in the library to system stored procedures (`sp_some_name`) but only internal to some meta-query or as a comment for suggested alternatives. But you will not find any meta-queries that simply call a system stored procedure because they lack the desired flexibility in parameterization. For example, the SQL Server procedure `sp_databases` takes no parameters, listing *all* your databases. The meta-query used to list databases access the `sysdatabases` table instead because it can then filter the results based on your input. A procedure like `sp_tables` on the other hand, takes a single, fixed parameter so can provide information only regarding a *single* table. The actual meta-query to list tables uses the INFORMATION_SCHEMA views to again allow appropriate filtering.

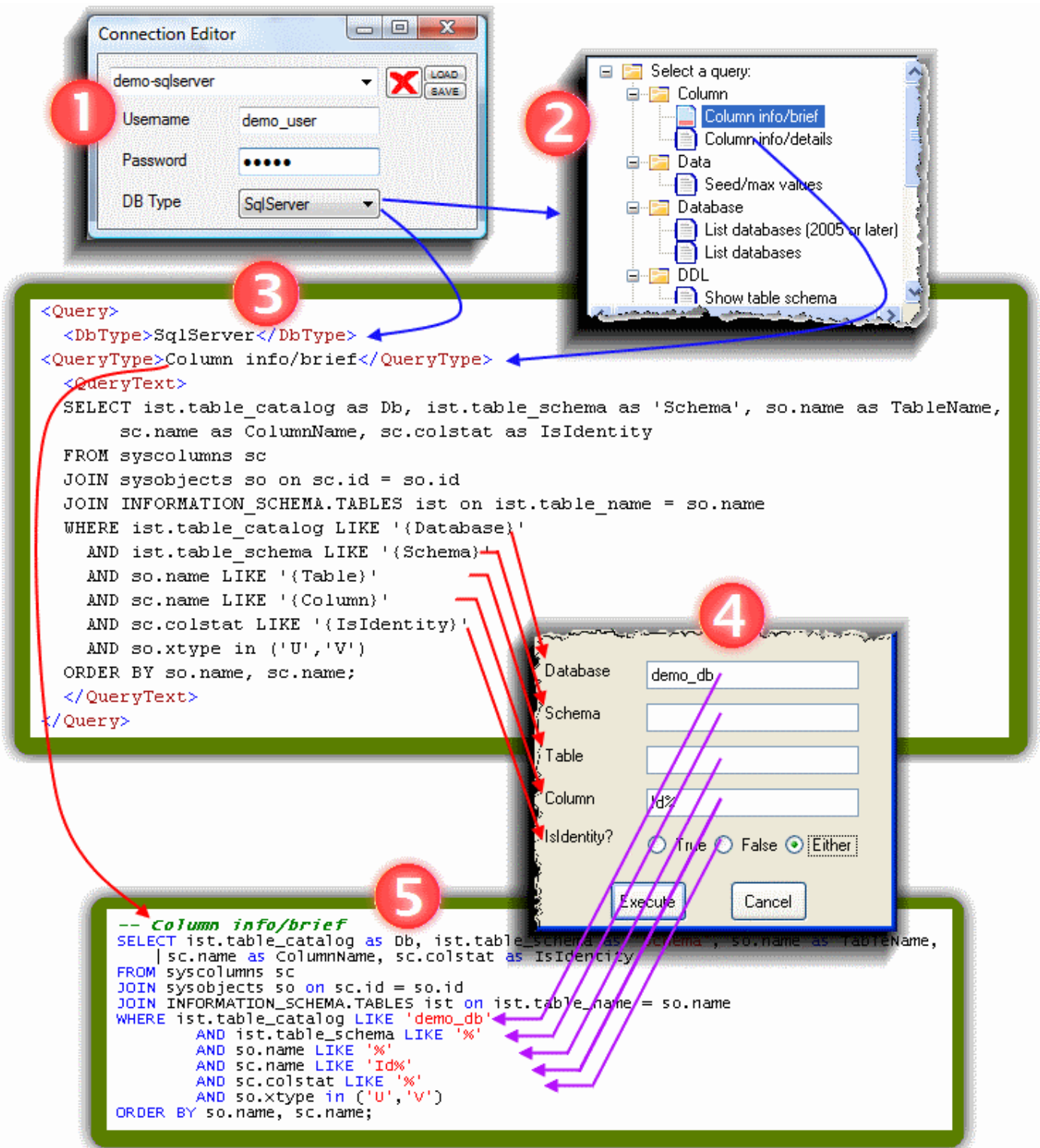


Figure 3-21 Meta-Query Processing

The database type of your current connection (1) determines the available set of meta-queries when you open the meta-query selector (2). Choosing a meta-query accesses the query template (3) which generates input fields (4) in the meta-query selector. Your entered values fill the place holders in the query template to produce a completed query (5).

3.10.3 Meta-Query Usage Summary

Once you have digested the detailed discussion of meta-queries, the summary in Table 3-7 provides a quick reference to make it fast and easy to find the key details about using meta-queries and meta-query templates.

Place holder	Generate a text box	{field_name}
	Generate Boolean choices with default labels	{field_name?}
	Generate Boolean choices with custom labels	{field_name?yes_label/no_label/any_label}
	Require input for a field	{field_name!}
Universal match suppression region		[[any SQL text containing a place holder]]
Execution	As a silent command	Execute
	Display query without universal matches	Shift+Execute
	Display query and reveal universal matches	Alt+Shift+Execute
Input Wildcards	Percent (%)	Any string of 0 or more characters
	Underscore (_)	Any single character
	[<i>characters</i>] or [<i>character-character</i>]	Any single character within the enumerated set or the specified range, respectively
	[<i>^characters</i>] or [<i>^character-character</i>]	Any single character not within the set or range, respectively

Table 3-7 Meta-Query Usage Summary

3.11 Syntax Highlighting & Keyword Completion

3.11.1 Syntax Highlighting

SqlDiffFramework recognizes a number of different SQL language constructs as you type and highlights them to make the text clearer to read. Section 4.4.2 details each of the language constructs. This section focuses on *keywords*, predefined words that are part of the SQL language. Each database type has a slightly different set of language keywords, so SqlDiffFramework dynamically adjusts its context depending on what database type you are using in each editor pane (SQL Server, Oracle, or MySql).

Within each context, keywords are grouped by category: functions, keywords, constants, and macros. The context maintains style characteristics at the group level, so the words in each category are all highlighted in the same way, but the highlighting between groups is different. Constants, for example, are stylized in bold black and Courier while functions appear in magenta in the default font.

Highlighting is actually a two-step process: *recognition* and *stylization*.

3.11.1.1 Recognition

When you pause typing, SqlDiffFramework compares what you have typed with the defined keywords in the current context. This is the *recognition process*. If it recognizes the new word, it moves on to the second step, stylization. By default, recognition occurs independent of case. So if the context includes the keyword `SELECT` and you type `select`, your typed word will be recognized and highlighted but otherwise unchanged. With the **CaseSensitive** setting you can modify this behavior. Setting this to true then requires the case you type to match the case of the keyword in the context definition: `SELECT` will then only match `SELECT` and not `select`. To adjust the **CaseSensitive** setting, open the context menu by right-clicking in the query editor: `Highlighting` → `Case Sensitive`.

3.11.1.2 Stylization and Case Adjustment

By default, highlighting affects only the color and style—but not the case—of the word. For example, `RTRIM`, a common SQL function, is rendered in magenta as mentioned earlier. If you type `RTRIM` it will be rendered as `RTRIM`; `rtrim` yields `rtrim`; `Rtrim` yields `Rtrim`. The word is colored and styled but the case remains unaltered. You may alter this behavior, though, via the `Highlighting` → `Keyword` submenu on the context menu. You have options to convert recognized keywords to uppercase or to lowercase. If you specify that keywords should all be uppercase, then even if you type `rtrim` it will render as `RTRIM`.

3.11.2 Keyword Completion

Keyword completion increases productivity by saving typing: bring up a list of all words beginning with the current prefix you have typed with a single keystroke and then click to insert the one you want. Just as with highlighting, keyword completion involves the same two-step process, recognition and stylization.

3.11.2.1 Recognition

When you invoke keyword completion a list of keywords appears. The contents of the completion list depend on the location of the cursor:

- If a partial word or even just a single character precedes the cursor (e.g. `wher`) and has multiple possible matches, then the list shows all keywords beginning with the target prefix you have entered, allowing you to select the one you want.
- If your target prefix has only one possible match, the pop-up list is skipped and the sole matching word is immediately inserted in your text.
- If the cursor immediately follows whitespace—that is, there is no target prefix—then all keywords are listed (giving you a rather lengthy list from which to choose).

Note that recognizing the target prefix is *always* case-insensitive: whether you type `WHER`, `wher`, or even `wHeR`, they are all treated identically in generating the completion list.

3.11.2.2 Stylization and Case Adjustment

Unless your target prefix had only one possible match, you must next select one choice from the completion list. Select a word or phrase from the list with keyboard arrows then press either `Space` or `Enter`. Alternatively, use the mouse to scroll then click. Either way, the completion list closes and the item you selected replaces the target prefix.

Auto-completion does *no* stylization (color or font); it only involves potential case adjustments. The case of the auto-completed keyword depends on the rule you have selected via the context menu. By default, the case of the inserted word is exactly what you see in the list of auto-complete candidate words, whether upper or lower case. If you see `SELECT` in the list, then whether your target prefix is `sel` or `SEL` or `seL`, it is replaced by `SELECT`. You may alter this behavior, though, via the `Keyword Completion` submenu on the context menu. If you select `Uppercase`, the word is converted to uppercase regardless of the case of the defined word and regardless of the case of the prefix you have typed. Similarly if you select `Lowercase`, it is converted to lowercase. Finally, if you select `Match user case` the case of the inserted keyword changes to match the case of your target prefix. (More precisely, it matches the case of the first letter of the target prefix.) Thus, if you type `se` the completion list displays `SECTION`, `SELECT`, `SESSION`, and others. If you

choose `SELECT` from the completion list, it is converted to `select` because your target prefix (`se`) was in lowercase.

3.11.3 Highlighting and Keyword Completion Interactions

The previous sections introduced a number of settings relating to *case sensitivity* and *case matching* on both auto-highlighting and auto-completion, settable interactively on the context menu. To use these settings effectively it is important to understand the interrelations between highlighting (3.11.1) and completion (3.11.2). (If you are just typing without invoking auto-completion then you only have to be concerned with highlighting.)

Keyword completion occurs first. You type a target prefix, press `Control+Space` to open the completion list, select a word or phrase, and the selected word replaces your typed target prefix. Syntax highlighting follows immediately thereafter. Once a keyword is entered—whether by you manually typing it or by the auto-completion facility—it is highlighted according to the settings for syntax highlighting.

The trap to be aware of is when you use a non-default option on case for *both* keyword completion and syntax highlighting. If you specify, for example, that keyword completion should complete words in lowercase but also specify syntax highlighting to highlight words in uppercase, an auto-completed word will always end up in uppercase by consecutive application of your rules. Thus, if you type `sel` or `SEL` and auto-complete this to `SELECT`, the word `select` will be inserted in lowercase. A fraction of a second later this word will be recognized by the highlight engine and converted to uppercase. So some combinations are incompatible but all are available so you may tailor the user experience as you see fit.

3.11.4 Macros

There are a few general macros included for each supported SQL dialect (SQL Server, Oracle, and MySQL). Macros are defined and used just like regular keywords. That is, you use the standard keyword completion invoked with `Control+Space` to insert a macro. The only differences are that SqlDiffFramework considers an element to be a macro if it contains any whitespace (and therefore multiple words). Also, any elements with place holders are macros.

3.11.4.1 Anonymous Macros vs. Named Macros

Macros come in two varieties: *anonymous macros* and *named macros*. An anonymous macro has no name, just a body; the macro body itself appears in the auto-completion list. A named macro has both a body and a name, and the macro name appears in the auto-completion list. For example, say you have an anonymous macro that inserts the phrase `SELECT * FROM`. When you type `SEL` then press `Control+Space`, the text of the macro appears among the displayed choices (see the second item in Figure 3-22). Note that if a macro is

too long it is truncated and marked with an ellipsis to indicate there is more that is not visible (fourth item in Figure 3-22). Anonymous macros therefore should be relatively short; longer ones should be defined as named macros.

Besides length, the other reason macros would be defined as named macros is positioning. Named macros appear in the keyword completion list sorted by the name whereas anonymous macros are sorted by the macro body. If the anonymous macro example above was instead a named macro with a name of `ALL` or `ALL-FIELDS` (since the meaning of the asterisk may not be obvious to everyone) the macro will show up in the list alphabetized under A for `ALL` rather than S for `SELECT`.

Macros are intermixed with simple keywords in the list; macro names for named macros, macro bodies for anonymous macros, and keywords are all sorted together. Named macros, then, are often indistinguishable from regular keywords at first glance—consider the keyword `SELECT` compared to the named macro `SELECT-FULL` in Figure 3-22. It is only when you choose an entry from the completion list and it appears in your editor that you know you have just used a macro. The keyword will appear unaltered; the named macro will have the macro body rather than the macro name that was in the list.

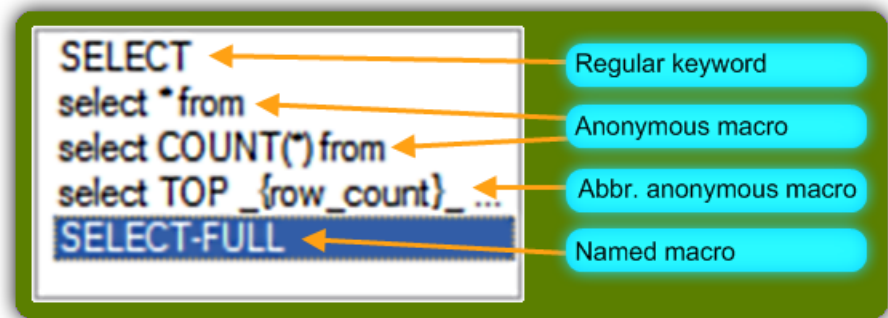


Figure 3-22 Keyword Completion List

Anonymous macros may be distinguished from regular keywords by the presence of whitespace and/or place holders. Named macros, however, are indistinguishable in the list.

3.11.4.2 Macro Templates

You can insert an arbitrarily complex query or query fragment with just a few keystrokes. The query as it exists in the library is called a *query template* because it may contain place holders to be filled out upon use. Macros are not required to have place holders but generally macros with place holders are more versatile and will be used more often than those without. A place holder is delimited by the `{` and `}` brackets and may contain only letters, digits, or underscores. Figure 3-23 shows an example of instantiating a macro named `SELECT-FULL` containing 6 place holders.

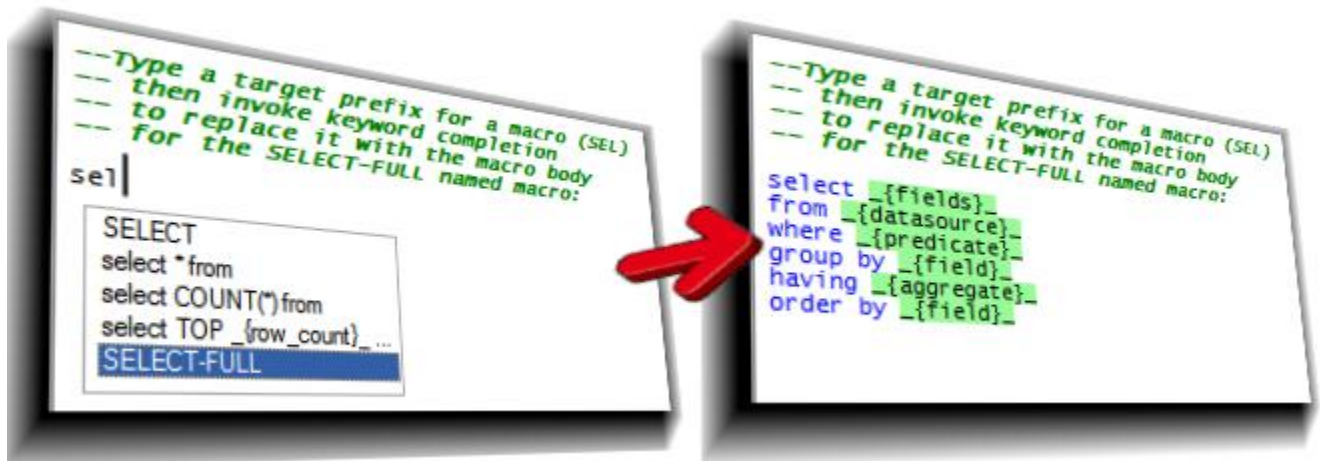


Figure 3-23 Instantiating a Macro

After typing the target prefix “sel”, Control+Space opens the completion list with candidate values; choosing the SELECT-FULL macro name inserts its corresponding macro body into the query editor with place holders highlighted.

Start by typing the first one or more letters of the macro name, in this case `SEL`. Press `Control+Space` to display the completion list. Choose the `SELECT-FULL` macro name.

SqlDiffFramework replaces the target prefix with the body of the macro, which is a template for a `SELECT` query. Once the body of the macro is inserted in the editor the syntax highlighting engine processes the text, highlighting according to the style characteristics defined in the current context for each keyword. In this case recognized keywords are highlighted in blue and place holders are shaded in green.

But clearly you are not finished yet. Your goal is to transform this query template:

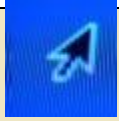
```
select {fields}_
from {datasource}_
where {predicate}_
group by {field}_
having {aggregate}_
order by {field}_
```

to a specific query instance that you can actually execute, something like this:

```
select x.ProductId, y.Yield, y.Frequency
from products x join data y on x.ProductId = y.ProductId
where y.yield > 100
group by x.Color
order by x.ProductId, y.Yield, y.Frequency
```

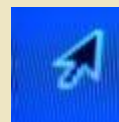
SqlDiffFramework provides two shortcut keys to make it quick and easy to fill out the template:

F4 advances to the next place holder and selects it, allowing you to just type to replace it. So do a sequence of **F4**, type something, **F4**, type something, etc. to fill in all the place holders.



TIP: You do not have to edit the place holders in order: skip a place holder by just pressing **F4** again to advance to the next one. Once you reach the last place holder the next **F4** wraps back to the first unaltered place holder.

To optimize typing **Control+Shift+Delete** deletes the line containing the currently selected place holder. Templates may often have more than you need since it is often quicker to prune than to add. Assuming that a template is designed with each place holder on a separate line (as in the above example), when you land on a place holder in a line that you do not need, just press **Control+Shift+Delete** to delete its containing line. The above example query instance does not use the `HAVING` clause; the line containing it was removed after advancing to the place holder then pressing **Control+Shift+Delete**.



TIP: The **Control+Shift+Delete** key combination is not just for use with macros: it deletes the currently selected range but *snaps to line boundaries*. See 4.4.1.19 for more details.

Table 3-8 shows the sequence of steps to fill in the sample query template above.

<p>Original template inserted</p>	<pre>select {fields} from {datasource} where {predicate} group by {field} having {aggregate} order by {field}</pre>
<p>F4 advances to first place holder</p>	<pre>select {fields} from {datasource} where {predicate} group by {field} having {aggregate} order by {field}</pre>
<p>Type the desired phrase.</p>	<pre>select x.ProductId, y.Yield, y.Frequency from {datasource} where {predicate} group by {field} having {aggregate} order by {field}</pre>
<p>Select the phrase just typed and copy it.</p>	<pre>select x.ProductId, y.Yield, y.Frequency from {datasource} where {predicate} group by {field} having {aggregate} order by {field}</pre>
<p>Skip to the next place it is needed; in this case press F4 five times.</p>	<pre>select x.ProductId, y.Yield, y.Frequency from {datasource} where {predicate} group by {field} having {aggregate} order by {field}</pre>
<p>Paste the copied phrase here.</p>	<pre>select x.ProductId, y.Yield, y.Frequency from {datasource} where {predicate} group by {field} having {aggregate} order by x.ProductId, y.Yield, y.Frequency</pre>
<p>Press F4 to advance to the next place holder; this wraps back to the first unfilled place holder. Type or Control+Shift+Delete then F4 again until finished.</p>	<pre>select x.ProductId, y.Yield, y.Frequency from {datasource} where {predicate} group by {field} having {aggregate} order by x.ProductId, y.Yield, y.Frequency</pre>

Table 3-8 Sample Sequence Filling in a Query Template

3.11.5 Customizing Syntax Highlighting and Macros

Each SQL dialect supported (SQL Server, Oracle, MySQL) has an associated context file that specifies the style characteristics of keywords, comments, variables, strings, numbers, and place holders. It also specifies the list of language keywords and macros. Each default context file is embedded in the application package. The first time you connect to a database of a given type, SqlDiffFramework creates the context file (`Context-dbtype.xml`) by unwrapping the embedded resource. The application stores this and other externalized resources in the Application Data/SqlDiffFramework directory. The location of your Application Data directory depends upon the Windows version you are using; you can identify the precise path using the `Help → About` menu item.

Important note: All context files are purged each time you install a new version of the application *unless* you have made a given file read-only to preserve any customized changes you may have made.

To modify any of the existing context files open the appropriate context file in a text editor or XML editor. The context definition must conform to the XML Schema shown below. This schema file is included with SqlDiffFramework so you can validate any changes you make, or if your XML editor can make use of it to provide guidance while editing. The schema file is included in the same directory containing the executable `SqlDiffFramework.exe` file (XML Schema/`ContextDefinition.xsd`). Figure 3-24 illustrates the schema here for your convenience.



Figure 3-24 Context Definition Schema

3.11.5.1 Context File Structure

Under the root `EditorContext` node, there is a `Delimiters` node containing all the details except for keywords and macros, which are under the `WordsAndPhrases` node. Note that all nodes under `EditorContext` are optional; you only have to include nodes when you want to specify a value different from the default. The default for the special `SeparatorTokens` node is shown below; the defaults for all the other nodes are shown in section 4.4.2.

The `Delimiters` element allows you to define styles for comments, strings, variables, separators, numbers, and place holders, and to define delimiters for all of these except numbers (which are just occurrences of 0 through 9). Some of these may have only one delimiter (e.g. block comments, end-of-line comments, and place holders) while others may have any number (strings, variables, and separators). You define, for example, an arbitrary number of string delimiters by adding a `QuoteToken` node for each (typically single quote and double quote). Similarly you may have an arbitrary number of variable designators by adding a `VariableToken` node for each (e.g. @ for parameters and @@ for system variables in SQL Server), where each `VariableToken` represents a sigil.

Separator tokens are unique in that each is always a single character and you do not specify any styling—they are present for proper parsing of language elements. You can specify any number of separator characters but you do this by specifying a single string in the `SeparatorTokens` node containing all of the separator characters. The default string is usually sufficient, so unless you want to modify the default you do not need to include the node at all. The `SeparatorTokens` default is the set of non-alphabetic, non-numeric characters:

```
`-~!@#$$%^&()+[\]\{\};':\",./<>?(return)(newline)(tab)(space)
```

The `WordsAndPhrases` element specifies keywords and macros in one or more `WordGroup` nodes. You could include all under a single group; specify multiple groups to apply different styles to different keywords, e.g. red and bold for function names, blue and italic for command words, etc. The `WordGroup` node also takes an optional type attribute whose only purpose is to label the group when you display the context details via `Control+F10`. (see TBD). If you do not provide a type the groups are simply labeled *Group 1*, *Group 2*, etc. Figure 3-25 shows a portion of the SQL Server context file specifying reserved word groups, each with an explicit type name that is reflected in the context summary dialog.



Figure 3-25 Keyword Groups

Type attributes on WordGroup elements are reflected in the context summary, showing how many of each type are defined.

The `keyword` node is the only element the `WordGroup` may contain. A `keyword` node specifies either a reserved word in the language or a macro. If the latter, then it may contain an `alias` attribute (this determines whether it is a named macro or an anonymous macro) and it may contain a `whiteSpace` attribute, described next.

3.11.5.2 Controlling Whitespace in Macros

Macro templates may contain one word or many words and extend one line or multiple lines. SqlDiffFramework provides some flexibility in whitespace between how a template is defined in the library and rendering that template when you insert it into your editor. That is, a macro template may be defined on multiple lines in the context file but depending on its properties it could end up in your editor on multiple lines or on a single line.

The `whiteSpace` attribute attached to a macro template controls how whitespace in the context file is processed for that node. The possible values and semantics for this attribute are borrowed exactly from the XML Schema definitions:

- `preserve` indicates to retain any white space as it is entered;
- `replace` replaces each occurrence of a non-space white space character (tab, return, newline) with an actual space character, effectively merging everything onto a single line;
- `collapse` (the default) replaces runs of any white space with a single space and trims leading/trailing white space.

Macros with place holders should generally be set to `preserve` (as the `SELECT-FULL` macro is) to provide the flexibility of using the `Control+Shift+Delete` shortcut key to snap to line boundaries and delete as described earlier.

3.11.5.3 Highlighting Styles

Comments, strings, variables and keywords use a common set of attributes, identified as the `fontStyleAttributes` attribute group in the schema. This includes:

- the font family (a string indicating the font name as in `Verdana` or `Times New Roman`);
- the font color (a string indicating a .NET color name—see <http://www.pardesiservices.com/Softomatix/ColorChart.asp> for a reference chart);
- a bold flag (`true` or `false`); and
- an italic flag (`true` or `false`).

Numbers and place holders, on the other hand, use the `fontStyleAttributes_WithBgColor` attribute group. As the name implies, this is a superset of the `fontStyleAttributes` attribute group that adds a background color choice (using the same set of color names as the font color).

See also section 4.4.2 for more on the different highlighting elements.

3.12 Automating Data Analysis

Up until now you have learned how to manually run a single pair of queries to obtain a single comparison result. SqlDiffFramework also provides an automation or batch facility where you may execute any number of query pairs, save snapshots of every result set, and generate a summary table for all the executed queries.

Open the **Batch Execution Palette** via the `Query → Execute Batch...` menu item (Figure 3-26). The top portion of the palette indicates the key details to be used during execution. Use this information to make sure you are pointing to the correct systems (System row), the correct input directories (source row), and correct output directories (snapshot row). Note that in the figure the source row displays an error indicator because the source directory has not yet been set after a fresh installation—it still shows the default dot, indicating the current directory. While you could run manual queries with that setting, batch execution requires you to be more meticulous and provide an explicit path. You do not set the paths here, though: close the palette, return to the main window, and load a query—any query—from the appropriate path, on both editor panes. Re-open the palette and you should then see the source directories filled out.

The snapshot row, as mentioned, indicates where the output is stored, the output being CSV snapshot files for each query executed. The Snapshot row will similarly show the default (in this case empty values) until you manually save a snapshot from the main window. (Note that an error indicator is *not* present on the snapshot row because the **Save Snapshots** checkbox in the lower left corner is disabled.)

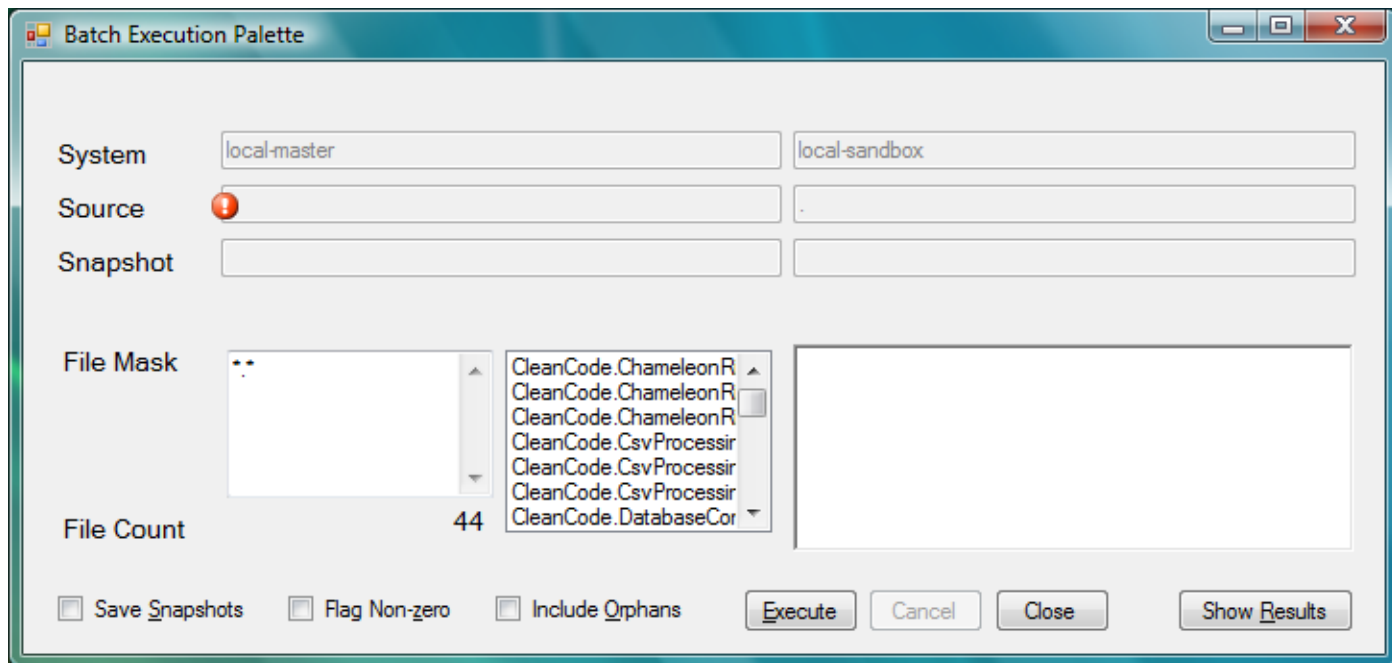


Figure 3-26 Batch Execution Palette

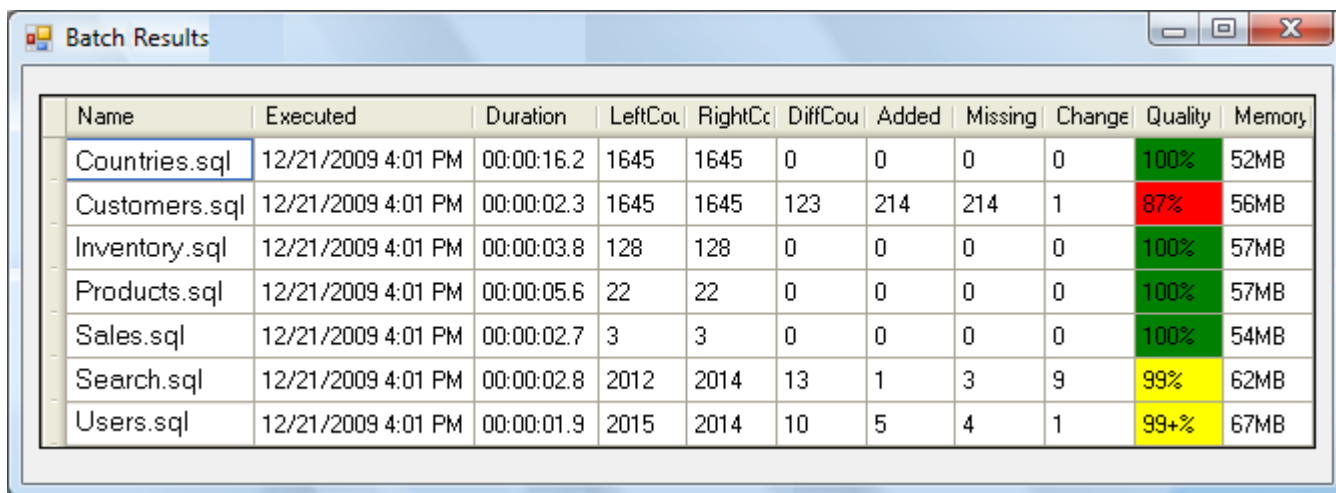
The top portion of the palette identifies the dual systems you are pointing to along with the source directories and snapshot directories. The bottom portion allows you to specify one or more file masks, identifies the files selected by those masks, and provides a notification window used once you initiate execution. The source directories have not yet been set so it shows an error indicator and will not let you proceed.

The middle portion of the **Batch Execution Palette** allows you to specify one or more file masks to filter a list of files from the source directory. Enter masks in the left text box and the list of matched files in the middle text box updates immediately, showing the files filtered by your masks. With the single default `*.*` specification, the list will show all files that are in *both* source directories—since comparison requires two queries, any file in one source directory that does not have a partner file of the same name in the other source directory is ignored.

A single mask may contain standard Windows wildcards (i.e. question mark or asterisk) to specify multiple files to match. You could use `*.*` to match all files, or something more specific, such as `*.sql` or `Cr*.sql`, etc. Sometimes it is difficult or impossible to specify a single mask to collect all the files you want, though, so you may use multiple masks as well. Separate each mask with lines (i.e. just press return in the input box), commas, spaces, or vertical bars.

Once you are satisfied with your list of files and the connections, press **Execute**. The large text box to the right of the file list displays the status of the run, indicating each file as it processes it. Starting execution also opens the batch results pane shown in Figure 3-27. You may manually open this results pane in advance—or at a later time after you have closed

it—using the **Show Results** button in the lower right of the palette (Figure 3-26). Also, the **Cancel** button becomes enabled upon starting execution so that you can abort the batch run if desired. Note that cancelling does not interrupt a running query but rather stops after the currently executing query comparison finishes. Just closing the execution pane will not abort the batch run; you must use the **Cancel** button if you wish to cancel.



Name	Executed	Duration	LeftCou	RightCo	DiffCou	Added	Missing	Change	Quality	Memory
Countries.sql	12/21/2009 4:01 PM	00:00:16.2	1645	1645	0	0	0	0	100%	52MB
Customers.sql	12/21/2009 4:01 PM	00:00:02.3	1645	1645	123	214	214	1	87%	56MB
Inventory.sql	12/21/2009 4:01 PM	00:00:03.8	128	128	0	0	0	0	100%	57MB
Products.sql	12/21/2009 4:01 PM	00:00:05.6	22	22	0	0	0	0	100%	57MB
Sales.sql	12/21/2009 4:01 PM	00:00:02.7	3	3	0	0	0	0	100%	54MB
Search.sql	12/21/2009 4:01 PM	00:00:02.8	2012	2014	13	1	3	9	99%	62MB
Users.sql	12/21/2009 4:01 PM	00:00:01.9	2015	2014	10	5	4	1	99+%	67MB

Figure 3-27 Batch Execution Results

This panel displays a summary of all the queries executed so far, one row per comparison. It duplicates the information from the main window detailing the quality of the match, including the total percentage and the individual numbers for added, missing, and changed rows.

As shown in Figure 3-27, the results pane enumerates each file, the record count for the left editor pane and the right editor pane, then the match quality statistics—the same information displayed in the results toolbar (section 4.1.2). The match percentages are color-coded here: green indicates a perfect match, yellow indicates 99% or better, and red is used for anything less.

Referring back to the **Batch Execution Palette** (Figure 3-26) there are two more options for executing batches: **Flag non-zero** lets you alter the semantics of the result grid from flagging differences to flagging any results any invalid. That is, instead of running queries that return the “good” data and then compare the left and right data results, you may elect to run queries that return “bad” data. For example, identifying product IDs where the color field references a non-existent color, identifying duplicate records in a table, etc. With a set of queries designed to ferret out bad data, any returned results are invalid. When you enable this checkbox, any non-zero results are flagged on the count columns, as shown in Figure 3-28.

Include orphans refers not to orphans in your data but orphaned queries, i.e. queries that exist only on the left or only on the right. Normally you need dual queries to do a

comparison but on occasion you may want to see a count of something in with your results. Or, more likely, when looking for bad data with **Flag non-zero**, the **Include orphans** option lets you look independently on both sides and report all the results together. Figure 3-28 shows the effective use of both of these options together. Where orphans exist, most values in the table are grayed out since you are no longer considering differences.

Name	Executed	Duration	LeftCount	RightCount	DiffCount	Added	Missing	Changed	Quality	Memory
	2/11/2010 ...	00:00:00.1		58						43MB
	2/11/2010 ...	00:00:00.1	0							57MB
	2/11/2010 ...	00:00:00.2	0	6	-1	-1	-1	-1	--	55MB
	2/11/2010 ...	00:00:00.2		0						57MB
	2/11/2010 ...	00:00:00.2		0						57MB
	2/11/2010 ...	00:00:00.3		0						57MB
	2/11/2010 ...	00:00:00.5		0						43MB
	2/11/2010 ...	00:00:00.5	0	265	-1	-1	-1	-1	--	55MB
	2/11/2010 ...	00:00:00.8		0						43MB
	2/11/2010 ...	00:00:00.8		0						53MB
	2/11/2010 ...	00:00:00.8	0	872	-1	-1	-1	-1	--	54MB
	2/11/2010 ...	00:00:00.8		0						55MB
	2/11/2010 ...	00:00:00.9	0	0	0	0	0	0	100%	55MB
	2/11/2010 ...	00:00:01.2	0	0	0	0	0	0	100%	53MB
	2/11/2010 ...	00:00:01.3	0	0	0	0	0	0	100%	57MB
	2/11/2010 ...	00:00:01.5		0						57MB
	2/11/2010 ...	00:00:01.6		0						57MB
	2/11/2010 ...	00:00:01.8		25						55MB
	2/11/2010 ...	00:00:02.4	0	9	-1	-1	-1	-1	--	43MB
	2/11/2010 ...	00:00:02.4	0	0	0	0	0	0	100%	53MB
	2/11/2010 ...	00:00:05.7		0						43MB
	2/11/2010 ...	00:00:08.2	539	539	2	2	2	0	99+%	54MB
	2/11/2010 ...	00:00:08.8	121	121	0	0	0	0	100%	54MB

Figure 3-28 Batch Results Including Orphans

Normally batch processing requires matched queries for the left and right data sources. You may override this using the Include Orphans checkbox on the setup form, which then includes all files on both sides matching the file masks. Differencing details are not applicable for such files, so they are grayed out for orphans.

4 Understanding SqlDiffFramework Components

4.1 Working with the Application

Figure 4-1 shows an exploded view of SqlDiffFramework. The main application window contains menus and a two toolbars at the top, and a common status bar at the bottom. The main body of the window consists of two identical Editor Panes (see 4.1.4) each of which in turn consists of a **Query Editor** (see 4.4) for input and a **Result Grid** (see 4.5) for output.

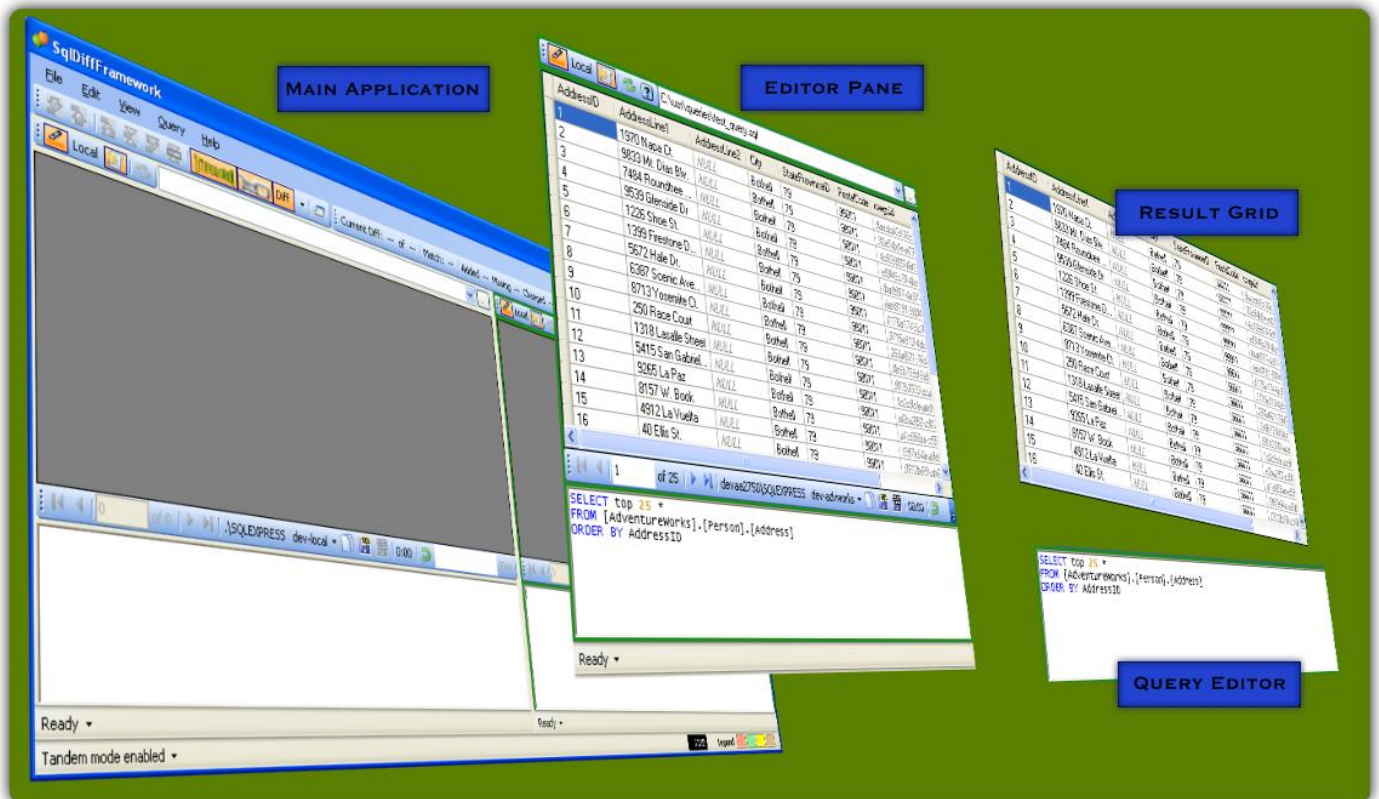


Figure 4-1 Components of SqlDiffFramework

Starting from the left, the main application includes menus and toolbars at the top; a status bar at the bottom. It contains two identical Editor Panes, each of which contains a Query Editor for input and a Result Grid for output.

There are two toolbars in the main window, the *control toolbar* and the *results toolbar*. They are positioned in the same row initially but you are free to relocate them, either stacking them at the top of the window or even to the sides or bottom of the window. Just drag the handle at the extreme left edge of each to move them.

4.1.1.1 SqlDiffFramework Control ToolBar

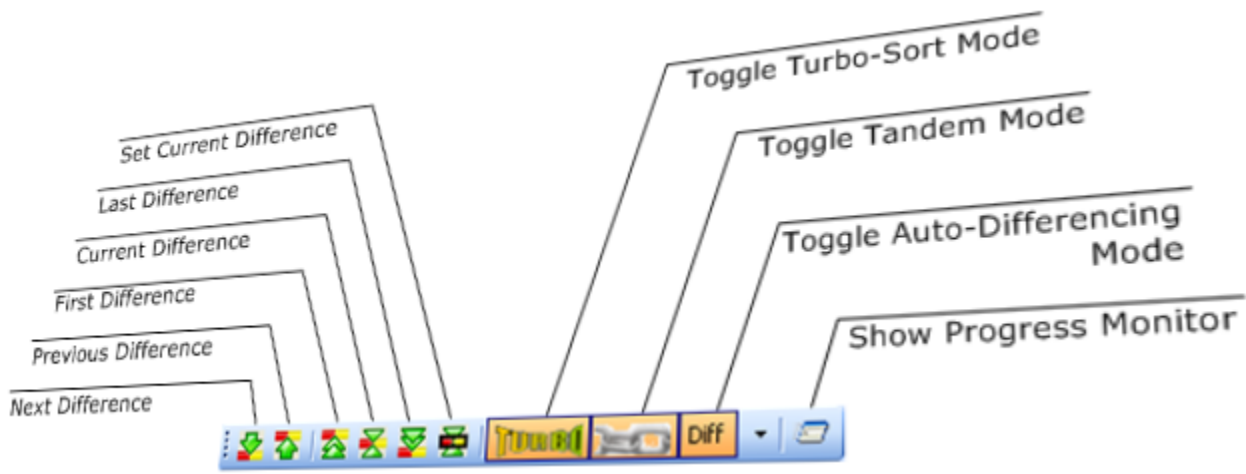


Figure 4-2 Exploded View of Application's Control ToolBar

4.1.1.1.1 Next Difference

Moves down to the next difference, scrolling as necessary, but does not go beyond the last difference. The difference position indicator (see 4.1.2.1) updates to reflect the movement.

If the current difference engine has reported differences where there are none (a false positive), a single invocation of this command will continue hunting through subsequent differences until it finds a real one. You may suppress this hunt mode by holding down the **Shift** key when you click the button.

You may adjust the behavior of this command to act only on a subset of the difference categories (added, missing, changed) using the results toolbar (see 4.1.2.6).

Alternatives: **Alt** + **↓** or menu command (see 4.1.6.1)

4.1.1.1.2 Previous Difference

Moves up to the previous difference, scrolling as necessary, but does not go beyond the first difference. The difference position indicator (see 4.1.2.1) updates to reflect the movement.



If the current difference engine has reported differences where there are none (a false positive), a single invocation of this command will continue hunting through antecedent differences until it finds a real one. You may suppress this hunt mode by holding down the **Shift** key when you press the button.

You may adjust the behavior of this command to act only on a subset of the difference categories (added, missing, changed) using the results toolbar (see 4.1.2.6).

Alternatives: **Alt** + **↑** or menu command (see 4.1.6.2)



4.1.1.3 First Difference

Moves to the first difference, scrolling as necessary. The difference position indicator (see 4.1.2.1) updates to reflect the movement.

Alternatives:  +  or menu command (see 4.1.6.3)

4.1.1.4 Current Difference

Moves the current difference into view in case you have scrolled it off-screen.

Alternatives:  +  or menu command (see 4.1.6.4)

4.1.1.5 Last Difference

Moves to the last difference, scrolling as necessary. The difference position indicator (see 4.1.2.1) updates to reflect the movement.

Alternatives:  +  or menu command (see 4.1.6.5)

4.1.1.6 Set Current Difference


Sets the current difference near the currently active cell. That is, you may advance the current difference by an arbitrarily large step rather than the single steps available on the previous/next difference commands.

If you have selected more than one row, the first row is used as the target.


The difference position indicator (see 4.1.2.1) updates to reflect the movement.

Alternatives:  +  or menu command (see 4.1.6.6)


4.1.1.7 Toggle Turbo Sort Mode

This mode switch () enables or disables turbo-sort mode. When you enable turbo-sorting, SqlDiffFramework locally sorts the results received from each executed query. Turbo-sorting compensates for any lexicographic differences between two databases by re-sorting both results according to the specifications in their respective queries. See 3.5 for details on sorting and turbo-sorting.

4.1.1.8 Toggle Tandem Mode

This mode switch () enables or disables tandem mode. When you enable tandem mode, many operations that you perform on one editor pane are automatically mirrored in its partner, including loading, executing, scrolling, and much more. See 3.7 for details on tandem operations.

4.1.1.9 Toggle Auto-Differencing Mode


This mode switch () enables or disables differencing mode. Additionally, it includes a drop-down selector that lets you switch among three separate differencing engines. When

enabled, a differencing engine automatically compares both result sets whenever you refresh one or both grids. If you initiate a tandem action to execute both queries, the differencing analysis does not begin until both queries have executed. But if you execute a query in just one editor pane the differencing engine compares result sets as long as there is a second result set to compare to. Furthermore if you turn off differencing with this switch, the difference engine will automatically perform analysis as soon as you turn it back on as long as there are two result sets to work with. Switching difference engines also performs an immediate re-analysis. Note that this does not require re-executing the queries, only re-analyzing the existing data so it is a bit faster.

Each of the three differencing engines performs differently and you may find, through experimentation, one that works best for you. The Tauberer engine is often faster than the Potter engine on large data sets, but Tauberer is prone to false positives with highly dissimilar data sets. The Hertel engine is the most memory intensive; as you work with larger and larger data sets Hertel will report out of memory well before the others.

Regarding false positives just mentioned, SqlDiffFramework cannot suppress them but it does two things to ameliorate their effect. First, you will see an asterisk attached to the changed row count (4.1.2.5) on the results toolbar indicating one or more false positives have been detected. Second, when you navigate among differences with the next/previous difference menu command or keystroke and land on a false positive it will skip it and move on to the next real difference (see 4.1.1.1).

4.1.1.10 Show Progress Monitor

During normal operation the progress monitor appears automatically whenever differencing analysis runs. Depending on the complexity of your queries and the size of your data, it may take up to several minutes to complete a full analysis so the progress monitor provides some visual feedback. When analysis is complete the progress monitor is turned off. In case you want to review the numbers it reports, this button  lets you see it again.

SqlDiffFramework does not currently run its time-consuming tasks as background operations (though it should!). Even so, it does provide some limited capability to cancel between steps via the progress monitor.

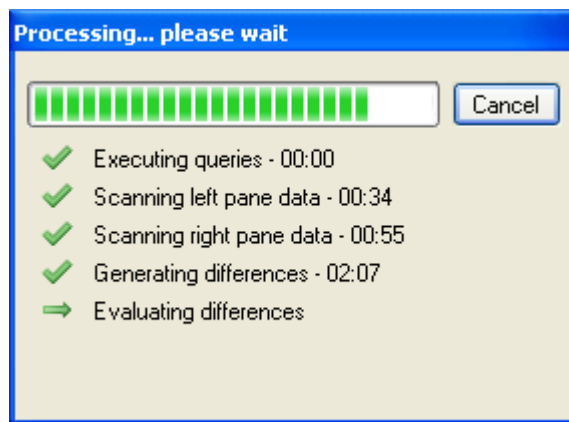


Figure 4-3 Progress Monitor

As the data analysis may take up to several minutes, SqlDiffFramework reports the progress of each step.

4.1.2 SqlDiffFramework Result ToolBar

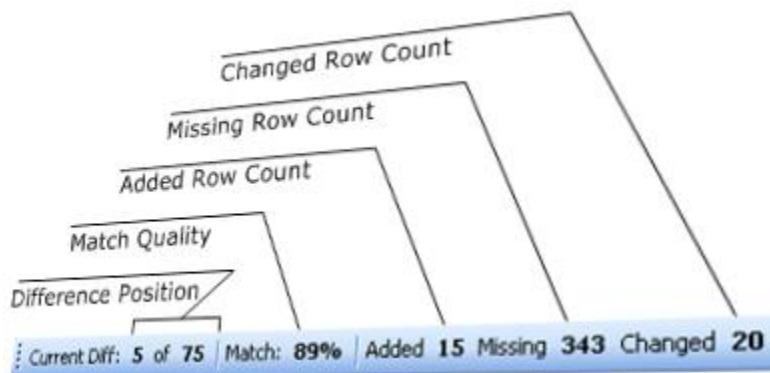


Figure 4-4 Exploded View of Application's Result ToolBar

4.1.2.1 Difference Position Indicator

This reflects movements among difference sections as performed by the [Next Difference](#), [Previous Difference](#), etc., commands discussed in the previous section. Note that difference navigation movements, however, do *not* affect the current record number (section 4.3.2.1) as they do not change the active cell. Hover over the current difference indicator to see how many rows are in the current difference on both the left and right editor panes.

4.1.2.2 Match Quality (%)

This indicates numerically how good the match is between the left pane and the right pane. The number is calculated by first combining the row count of the left result grid and right result grid to find the total rows. Then the total number of difference rows are counted on both left and right sides, whether they are changes or additions. The percentage is then simply the total count less the difference count all divided by the total count. Hover over the indicator to see the total number of rows with mismatches out of the sum of rows from both left and right editor panes.

4.1.2.3 Count of Added Rows

The added row count is relative to the *left* result grid. For each difference section, if the count of rows on the **left** is greater than that on the **right**, that difference is added to a cumulative total, reported in this field. The [Added](#) label is actually a button that allows you to refine behavior of the difference navigation keys—see 4.1.2.6.

4.1.2.4 Count of Missing Rows



The missing row count is relative to the *left* result grid. For each difference section, if the count of rows on the **right** is greater than that on the **left**, that difference is added to a


cumulative total, reported in this field. The **Missing** label is actually a button that allows you to refine behavior of the difference navigation keys—see 4.1.2.6.

4.1.2.5 Count of Changed Rows

For each difference section, the smaller of the number of rows in the left result grid and the right result grid is added to a cumulative total to arrive at the changed row count. Taking the smaller of the two reflects that *both* sides have that many rows that are changed; the “leftover” rows on the side with a larger count are already included in either the added row count or missing row count. The **Changed** label is actually a button that allows you to refine behavior of the difference navigation keys—see 4.1.2.6.

4.1.2.6 Difference Navigation Shortcuts


.By default, the Next Difference button ( (4.1.1.1) advances from the current difference to the very next one. Thus, if you are on difference 42, you will advance to difference 43. Similarly, the Previous Difference button ( (4.1.1.2) moves the other direction. But if, for example, you have a preponderance of just one of the three categories (added, missing, change) it is tedious to hunt for the few instances of one of the other categories.

As a concrete example, assume you have 25 differences reported of varying sizes, distributed as shown in Table 4-1. There is a preponderance of missing chunks, with very few added or changed chunks—the result toolbar reports 4 added, 21 missing, and 5 changed. If you are interested in examining the *added* chunks, you need to repeatedly press the Next Difference button () to advance to the fifth difference, then again to get to the eighteenth difference, etc. That is tedious but nothing compared to, say 500 differences and only two of them for added chunks—tedium to the extreme!

SqlDiffFramework provides a shortcut mechanism to let you go from one added chunk to the next. Click on the **Added** label (4.1.2.3) in the toolbar to enable *category filtering*. Once enabled, the difference navigation controls (toolbar buttons, keystrokes, or menu items) then move between just those chunks of the selected category. Each of the Added, Missing, and Changed labels on the toolbar may be enabled or disabled independently. You may use them individually or in combination.

	Added	Missing	Changed
1		X	
2		X	
3		X	
4		X	
5	X		
6		X	
7		X	
8		X	
9		X	X
10		X	
11		X	X
12		X	X
13		X	X
14		X	
15		X	
16		X	
17		X	
18	X		X
19		X	
20		X	
21		X	
22	X		
23	X		
24		X	
25		X	



Table 4-1 Sample Distribution of Chunks

Thus, with just Added enabled, the Next Difference button () will follow this sequence of chunks:

5 → 18 → 22 → 23

With both the Added and Changed categories enabled, you get:

5 → 9 → 11 → 12 → 13 → 18 → 22 → 23

In addition to the targeted chunks, the Next Difference () and Previous Difference () buttons will each eventually reach a logical boundary (the first chunk or last chunk, respectively) when searching for the next chunk to display. Once reaching that boundary, SqlDiffFramework displays that first or last chunk *even if it is not in the currently selected category filter*. So the sequences shown above should really begin with 1 and end with 25.



TIP: The default with *no* categories enabled is equivalent to having *all* three categories enabled as a convenience (since disabling all categories is not particularly useful).

4.1.3 SqlDiffFramework Status Bar

4.1.3.1 Status DropDown

Application-level messages are displayed in the application's status bar. (Just above that are separate status bars for each editor pane.) The last anomalous message is displayed right on the status bar; previous errors or warnings may be viewed by opening the dropdown. For long messages, hover over an item *in the dropdown* list to see a tooltip with the longer message.

4.1.3.2 Memory Indicator

The memory indicator shows, by default the memory used by the workspace, in megabytes. Click on the indicator to toggle to display a percentage of available memory. This indicator starts out black, indicating minimal usage. It changes color as you consume more memory resources as you work, from black to grey to plum to coral and finally red if memory tops 75% of available.

4.1.3.3 Legend

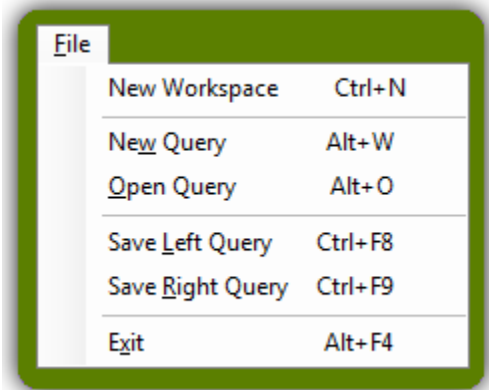
Just as an on-screen reminder, each of the difference bar colors is shown in the bottom right corner; hover over each to see a tooltip describing what the color means.

4.1.4 File Menu


4.1.4.1 New workspace


New workspace creates a new SqlDiffFramework window allowing you to compare another pair of data sets. Each time you invoke this command your current settings (paths, modes, etc.) are first saved. This has two side effects. First, since all workspaces use the same configuration file, a new workspace opens with the same settings that your original workspace has at the moment you create the new workspace (rather than with the settings you had when you opened the very first workspace). Second, the **Edit → Restore Settings...** command will be able to restore settings only back to this save point.

Alternatives: **Control** + **N**



4.1.4.2 New Query


New Query unloads the current query file (if any) and deletes the query text from the query editor. The current file indicator and the **query editor** will both be blank upon completing this operation. The application first prompts you if you have any unsaved changes. If you use the New Query button () instead of the menu command you have additional options, on the drop-down portion of the button.

- **New file** With no modifier keys the New Query button () is equivalent to the menu command.
- **New file but retain buffer** SqlDiffFramework still disassociates the file name but it retains the buffer contents. This is a convenient way to start a new query based on an existing one.
- **Generate new query from results** SqlDiffFramework clears the query window, disassociates the file name, and generates a base query from the currently loaded data, if any. See section 4.3.2.4.

Alternatives: **Alt** + **W** or New Query button ()

4.1.4.3 Open Query


Open Query presents a standard **open file dialog** box allowing you to select a query file (*.sql) to load into the appropriate pane. The selected file name appears in the current file indicator and the contents of the file loads into the **query editor**. The file name is also entered into the dropdown list attached to the current file indicator for ease of access later



in the same session. Therefore, any previously loaded file may be reloaded either by using the open command or by opening the dropdown list. The File Picker button () is equivalent to the open query menu command.

By enabling tandem mode, you may load both editor panes with a single action. Each editor pane remembers the last directory from which it loaded a file. When you select a file to load in one editor pane, the same base name is then attached to the remembered directory in the other editor pane and that file is loaded, if it exists.

Alternatives:  +  or File Picker button ()

4.1.4.4 Save LEFT query, Save RIGHT query

If no filename is associated with the current query, either save command presents a standard **save file dialog** box allowing you to specify a query file name. Upon pressing  in the dialog box the contents of the **query editor** are written to the specified file name. If, on the other hand, a filename is associated with the current query (either by a previous save or a previous open), then the application immediately saves the query to the same name, updating and overwriting its previous contents.

If you use the Save Query button () instead of the menu, you have one more option when saving: pressing  when selecting the Save Query button will let you save the contents of the query editor to a different name (a standard *Save as* operation).

Alternatives:  +  and  +  or Save Query button ()

4.1.4.5 Exit

Exit closes the current workspace, saving any changes to your current settings. Note that if you have multiple workspaces open, each time you close one it overwrites the stored settings with its settings, so the last one to close determines the settings that the next invocation of the application will open with.

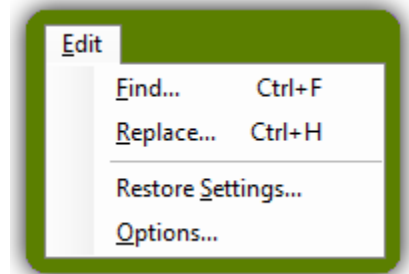
Alternatives:  +  or Application Close button ()

4.1.5 Edit Menu

4.1.5.1 Find...

Find... opens a dialog box to search for text in the query editor. You have the option of matching or ignoring the case of the term you enter. Furthermore, you may elect to search by plain text, by wildcards, or by regular expressions. See section 3.4.1.

Alternatives: **Ctrl** + **F**



4.1.5.2 Replace...

Replace... opens a dialog box to replace text in the query editor. You have the option of matching or ignoring the case of the term you enter to search for. Furthermore, you may elect to search by plain text, by wildcards, or by regular expressions. See section 3.4.1.

Alternatives: **Ctrl** + **H**

4.1.5.3 Restore Settings...

Restore Settings... allows you to restore your settings back to the last time they were saved or to the original factory settings. Be aware that this not only resets modes, paths, etc., but also wipes out your list of database connections if you restore to factory defaults. If you have an extensive list of customized connections you should go to the connection editor (see **Query** → **Edit Connections...**), export the DB connections to a file, then do **Restore Settings...**, and finally re-import the DB connections you saved.

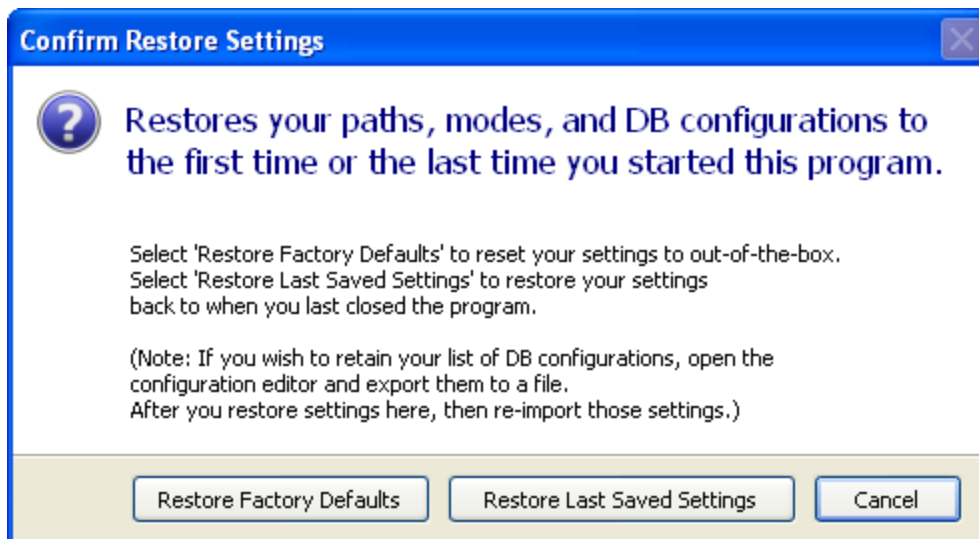


Figure 4-5 Restore Settings Confirmation Dialog

You may rollback your settings to those with which you began the current session, or all the way back to when you installed the application.

Section 4.2 lists all of the relevant settings. The list contains settings that represent both visual attributes and internal states. Among those representing visual attributes, the user interface is immediately updated to reflect the restored values *except* for window state (4.2.2.1) and window position (4.2.2.2). For example, the state of the Auto-Highlight button (🔍) will revert to the factory default or the last saved (depending on your choice) but the window position will not revert.

4.1.5.4 Options...

options... opens the **Option Dialog** (see 4.2.6) allowing you to adjust some global program settings.

4.1.6 View Menu

The first two groups of commands involve navigating among the difference regions once you load both grids and run a comparison.

4.1.6.1 Next difference

Moves down to the next difference, scrolling as necessary, but does not go beyond the last difference. The difference position indicator (see 4.1.2.1) updates to reflect the movement.

If the current difference engine has reported differences where there are none (a false positive), a single invocation of this command will continue hunting through subsequent differences until it finds a real one. If you use the toolbar button instead of the menu command, it has one additional capability: you may suppress this hunt mode by holding down the **Shift** key when you click the button.

You may adjust the behavior of this command to act only on a subset of the difference categories (added, missing, changed) using the results toolbar (see 4.1.2.6).

Alternatives: **Alt** + **↓** or toolbar button, section 4.1.1.1 (📌)

4.1.6.2 Previous difference

Moves up to the previous difference, scrolling as necessary, but does not go beyond the first difference. The difference position indicator (see 4.1.2.1) updates to reflect the movement.

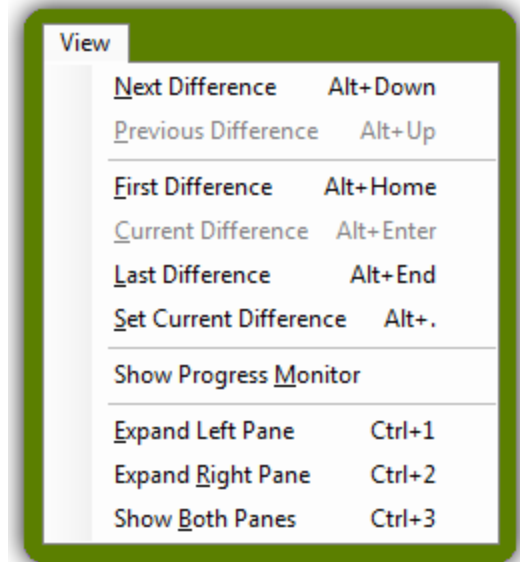
If the current difference engine has reported differences where there are none (a false positive), a single invocation of this command will continue hunting through antecedent differences until it finds a real one. If you use the toolbar button instead of the menu command, it has one additional capability: you may suppress this hunt mode by holding down the **Shift** key when you press the button.

You may adjust the behavior of this command to act only on a subset of the difference categories (added, missing, changed) using the results toolbar (see 4.1.2.6).

Alternatives: **Alt** + **↑** or toolbar button, section 4.1.1.2 (📌)

4.1.6.3 First difference


Moves to the first difference, scrolling as necessary. The difference position indicator (see 4.1.2.1) updates to reflect the movement.



Alternatives: **Alt** + **Home** or toolbar button, section 4.1.1.3 ()

4.1.6.4 Current difference

Moves the current difference into view in case you have scrolled it off-screen.

Alternatives: **Alt** + **Enter** or toolbar button, section 4.1.1.4 ()

4.1.6.5 Last difference

Moves to the last difference, scrolling as necessary. The difference position indicator (see 4.1.2.1) updates to reflect the movement.

Alternatives: **Alt** + **End** or toolbar button, section 4.1.1.5 ()

4.1.6.6 Set current difference

This command sets the current difference near the currently active cell. That is, you may advance the current difference by an arbitrarily large step rather than the single steps available on the previous/next difference commands.

If you have selected more than one row, the first row is used as the target.

The difference position indicator (see 4.1.2.1) updates to reflect the movement.

Alternatives: **Alt** + **.** or toolbar button, section 4.1.1.6 ()

4.1.6.7 Show progress monitor

The progress monitor normally opens during processing then closes again automatically. This menu command lets you open it on demand if you want to see, for example, the time each step took from the last run. See section 4.1.1.10.

Alternatives: Toolbar button ()

4.1.6.8 Expand left pane

You may wish to focus on a single query, or perhaps you have a result set with a lot of columns. Drag the center splitter rail to the left or right to give more area to the side of interest. This menu command provides a handy shortcut to move the splitter rail all the way to the *right*.

Alternatives: **Ctrl**+**1**

4.1.6.9 Expand right pane

You may wish to focus on a single query, or perhaps you have a result set with a lot of columns. Drag the center splitter rail to the left or right to give more area to the side of interest. This menu command provides a handy shortcut to move the splitter rail all the way to the *left*.

Alternatives: 

Show both panes

In contrast to the previous commands that maximize area for a single query, this command restores the splitter rail to the center with a single keystroke.

Alternatives: 

4.1.7 Query Menu

This menu provides actions relating to queries and database connections.

4.1.7.1 Execute query

Executes the query in the **query editor** and loads the **results grid** if the **local/live** mode switch is set to live.

If the local/live mode is set to live, the query is sent to the currently selected database connection (indicated by the current db value) to retrieve the data. See also section 3.9.

If the local/live mode is set to local, the query is not executed but the **results grid** is loaded from a previously saved data set. Even though the results are already fixed and therefore the query is not driving the results, the contents of the query editor are still related to the results. Any ignore clause effects what columns are compared and, if turbo sort is active, the order by predicate of the query is used for sorting. Any CSV file may be used as a data source, but CSV files saved by SqlDiffFramework carry extra information allowing the data table to be strongly typed rather than just all strings. See also: Turbo sort, Ignore clause, CSV import/export.

Whether live or local mode is active, this command operates on one or both editor panes depending on other conditions. If tandem mode is enabled both editor panes are refreshed. Otherwise, just the current editor pane [current editor pane = last editor pane the mouse was in] is refreshed. If the tandem override key is depressed, the setting of the tandem mode is reversed just for this single operation.

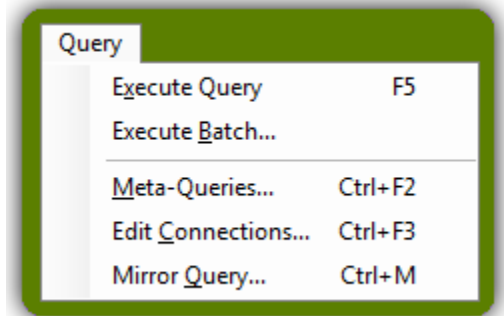
Alternatives: **F5** or Execute/Refresh button (🔄) on either editor pane.

4.1.7.2 Execute Batch

Execute Batch... opens the batch execution palette, allowing you to run many queries without user intervention. See section 3.12.

4.1.7.3 Meta-queries...

This command opens up a **meta-query dialog**, providing a selection of queries that can provide information about your database schema rather than about your data. Place the mouse over the editor pane of interest to make it the active pane then invoke the command. You will be prompted to select a meta-query and then supply its necessary parameters. Since this menu item is unique in that it acts on the active editor pane, it is advisable to use keyboard accelerators rather than the mouse to activate this menu item, lest you sweep



over the other editor pane on your way to the menu, inadvertently switching that pane to be the active one. See section 3.10.

Alternatives: **Ctrl** + **F2** or Meta-Query Dialog button (?) on either editor pane

4.1.7.4 Edit Connections...

This command opens a connection editor that lets you define connections to any number of SQL Server, Oracle, or MySQL databases as well as ODBC data sources. You can either manually enter connections or import a previously defined set that you have saved. The connection editor provides a combination test button and status indicator that lets you immediately test a connection you are defining. It also has **Save** and **Load** buttons for exporting and importing respectively. See section 3.9.

Alternatives: **Ctrl** + **F3** or depress the **Alt** key while clicking on the connection selector in either editor pane (see 4.3.2.3).

4.1.7.5 Mirror Query...

This command opens a dialog allowing you to copy all the settings and the query text from one pane to the other in one click.

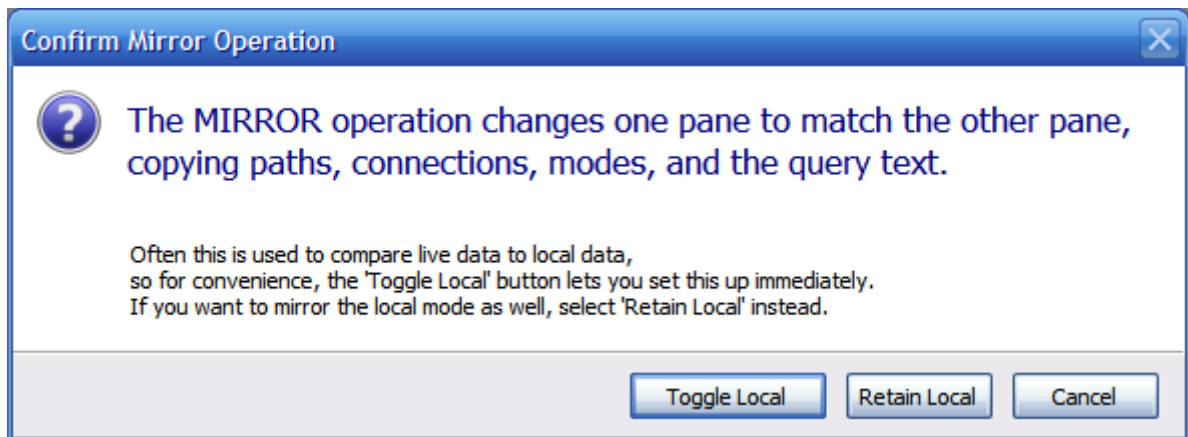


Figure 4-6 Mirror Dialog

Use the Mirror Query command to duplicate settings and query text from one editor pane to the other. You have a choice on how to set the local/live mode switch during the process.

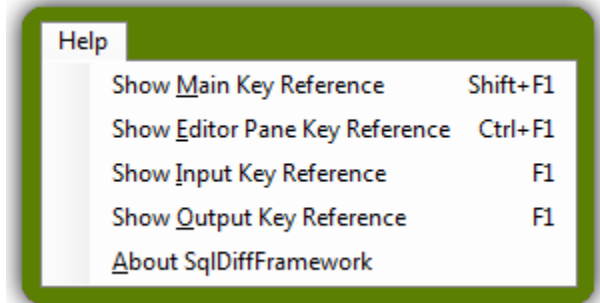
As the dialog states, this is commonly used to compare live data with local data; that is, data from the same source taken in the past vs. current data. The other common use is for comparing two past snapshots of data. There are two choices available when you perform the mirror operation. The only distinction between the two buttons is the setting of the local/live mode switch. The **Toggle Local** choice makes the local/live mode switch in the target pane have the opposite state as that in the source pane. The **Retain Local** choice mirrors the current setting so that both have the same state.

Which pane is the source and which is the target is determined by which editor pane is the current pane at the instant you open the menu item. It is advisable, therefore, to use the keyboard shortcut to invoke the menu item rather than using the mouse to select the menu item. If you use the mouse and travel through the other editor pane on your way to the menu, you then switch which editor pane is the source. There is no direct way to undo this operation; there is an indirect way that may work, depending on your activity in the current session. The `Edit → Restore Settings...` command includes a choice to `Restore Last Saved Settings`. That command discards any changes made since the start of the session or since the last `File → New workspace` command, whichever is more recent. So if you do not mind rolling back your settings to the last saved settings, you can undo the mirror operation. To limit rollback to just the changes made by the mirror operation, invoke `File → New workspace` just before you invoke `Query → Mirror Query...`

Alternatives: `Ctrl` + `M`

4.1.8 Help Menu

This menu provides a collection of quick reference sheets and information about the application. The commands available lend themselves to natural groupings related to different levels of operation. Snapshots of each of these quick references begin on the next page.



4.1.8.1 Show Main Key Reference

This quick reference sheet shows the command and shortcut keys available at the top-level, operations that primarily involve both editor panes (navigating data differences, etc.).

Alternatives: **Shift** + **F1**

4.1.8.2 Show Editor Pane Key Reference

This quick reference sheet shows commands pertaining to each editor pane individually (loading and saving inputs and outputs) as well as identifying those commands where both editor panes may act in unison (synchronized scrolling, synchronized loading, etc.)

Alternatives: **Ctrl** + **F1**

4.1.8.3 Show Input Key Reference

Within an editor pane, you use the query editor (the bottom half) for input; this sheet describes commands for highlighting, formatting, editing, and navigation within the query editor.

Alternatives: **F1** when mouse is over either query editor.

4.1.8.4 Show Output Key Reference

Within an editor pane, you use the results grid (the top half) for output; this sheet describes commands for selection, navigation, filtering, and formatting within the results grid.

Alternatives: **F1** when mouse is over either result grid.

4.1.8.5 About SqlDiffFramework

The **About** command displays a variety of useful information about SqlDiffFramework. The top of the box shows the current release version of the application. The scrolling box at the bottom includes the versions of all loaded libraries. Note that libraries are only loaded as needed so if you come back after using the application for awhile you may see more libraries listed.

In the middle are several hyperlinks: the first two are URLs that take you to the SqlDiffFramework website in your default browser, while the last two are local file paths that open Windows Explorer to the indicated paths. (For paths that are too long to fit, hover over the link and the tooltip will display the full path.)

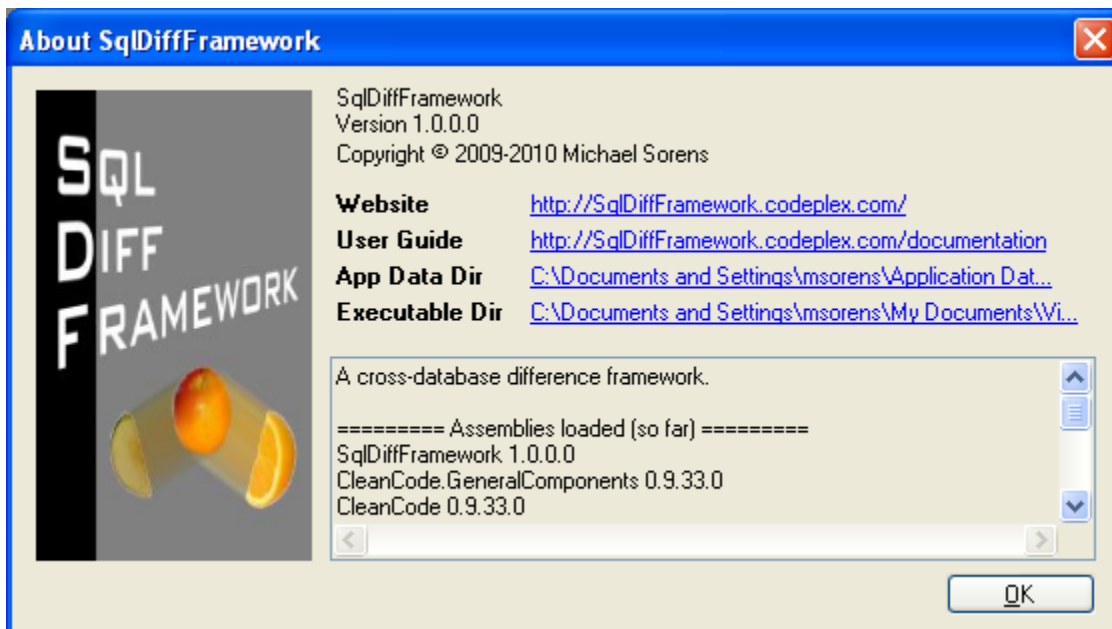


Figure 4-7 The About Box

4.1.9 Multiple Monitor Support

The output of a single query may easily be very wide if you need to display a lot of columns. Even when you maximize your querying tool's window and have a widescreen monitor, you still end up scrolling back and forth horizontally.

Since SqlDiffFramework displays two result sets side by side this exacerbates the problem even further. If you have two monitors side by side, then at least you could stretch the application window across both monitors so you are no worse off than displaying a single query. Windows gives you a one-click technique for maximizing a window to a single monitor but not for

multiple monitors. To stretch across two monitors you have to un-maximize, reposition the window to one edge, then carefully drag the opposite edge all the way to the far side of the second (or third) monitor. SqlDiffFramework gives you back a one-click technique: simply depress **Control** when you maximize your window with the standard window maximize button (☐). If, as shown in Figure 4-8, the splitter between the two editor panes is right in the middle (i.e. the two editor panes are equal width), then doing a multiple-maximize automatically gives you one editor pane on each monitor.



Figure 4-8 Maximize to Multiple Monitors

Use the standard window maximize button—while depressing the Control key—to maximize the application across multiple horizontally-aligned monitors in a single click.

4.1.9.1 Resolution and Orientation Impacts

The multiple-monitor maximization works intelligently across multiple monitors of varying resolution with a logical side-by-side organization. That is, vertical stacking of

monitors is not supported. When you invoke this feature, the resolutions of each screen are examined; the smallest one drives the height of the maximized window, as shown in Figure 4-9. The figure illustrates one example where you have a larger, desktop monitor on the left and a smaller, laptop monitor on the right and their tops are aligned. The window expands to fill the smaller of the two but no further. (If it expanded to fill the *larger* monitor instead, you would then have to frequently scroll back and forth vertically to see the whole window.)



Figure 4-9 Spanning Multiple Monitors with Varying Resolutions

All of the parts and controls of the two editor panes are labeled for easy identification. The query text shown in the Query Editor (near the bottom) is the contents of the Current File (near the top). The results of executing the query appear in the Result Grid.

The two editor panes were equal width before the maximization—and are *still* equal width after the maximization—but notice that a portion of the *right* editor pane is on the *left* monitor. That is just an artifact of the different screen resolutions. Say, for example, the left monitor is 1000 pixels wide and the right is 500 pixels wide. Half of the 1500 total puts the midpoint at 750 pixels, putting it about three-fourths the way across the left monitor, as the figure shows.

If instead of aligning the window *tops* you align the window *bottoms* the maximized window will follow suit; the left monitor in Figure 4-9 would show its portion of the window at the bottom of the screen instead of at the top.

Use the standard Windows **Display control panel** (or other manufacturer-supplied video control panel) to set the logical arrangement of your monitors (see Figure 4-10).

Also note that even for monitors of the *same* size and resolution there will be some automatic compensation due to the Windows taskbar. (The two monitors in Figure 4-8 both show screens filled to the edges because the taskbar is hidden.)

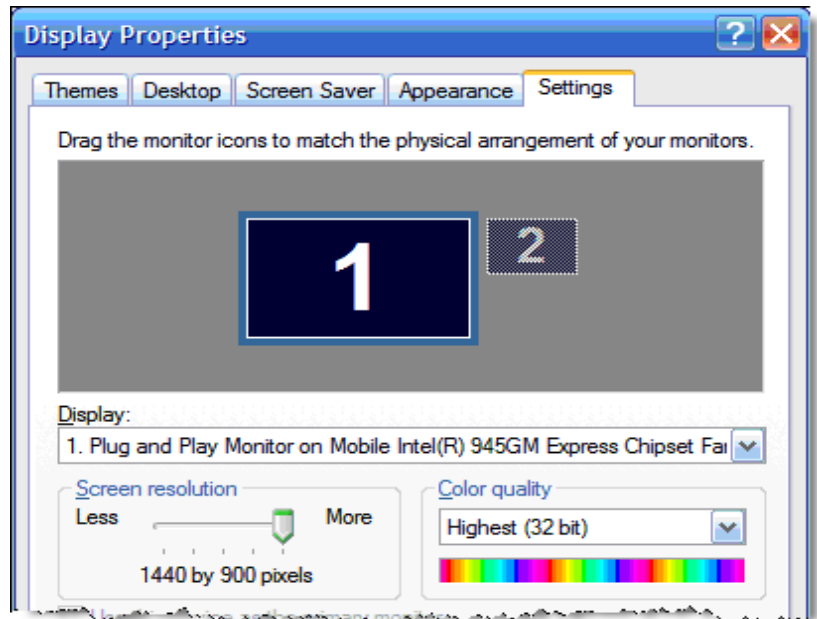





Figure 4-10 Windows Display Control Panel

Simply drag the monitor icons around the workspace to align their tops or bottoms, or to set which is on the left and which on the right.

4.1.9.2 Maximization Nuances

Assume that you have a non-maximized window (referred to as the *normal* state) as a starting point. In the simplest case when you press the maximize button () the window expands to fill the single monitor (the *maximized* state); pressing it again restores the window to the normal state. Similarly pressing  +  from the normal state expands the window across multiple monitors (the *multi-maximized* state) and pressing it again returns the window to the normal state. Table 4-2 outlines the actions that transpire from each button depending on the current window state.












































From this state...	With this button...	This action happens:
normal		Sets the window to maximized .
multi-maximized		Sets the window to maximized .
maximized		Sets the window <i>either</i> to normal <i>or</i> to multi-maximized , depending on its prior state.
normal	 + 	Sets the window to multi-maximized .
multi-maximized	 + 	Restores the window to the normal state.
maximized	 + 	Sets the window <i>either</i> to normal <i>or</i> to maximized , depending on its prior state.

Table 4-2 Actions of Maximize vs. Control+Maximize

Because the new state after leaving the maximized state is influenced by its prior state, mixing  with  +  may produce a behavior that is not always what you might expect. Here are some useful combinations (starting from the **normal** state):

Toggle max / normal					...
Toggle max / normal		 + 		 + 	...
Toggle multi-max / normal	 + 	 + 	 + 	 + 	...
Toggle multi-max / max	 + 		 + 		...

A conventional window remembers only *two* sizes: the normal size and the maximized size. SqlDiffFramework's windows, however, remember *three* sizes: normal, maximized, and multi-maximized. The above combinations show simple keystrokes for moving between two of the three states. Here is a sequence for switching between all three sizes:

Press this...	 + 		 + 	 + 
... to go to:	multi-max	max	multi-max	normal

4.2 Persistent Settings

SqlDiffFramework uses the standard .NET framework configuration file scheme: it reads an application-level configuration file (`SqlDiffFramework.exe.config`) on first launch to drive its behavior. All users on the same computer read this same configuration file. Once you close the application the first time, it records any alterations you may have made in a user-level configuration override file (`user.config`). On subsequent invocations, the application reads first the application-level configuration file to establish defaults, then reads your user-level overrides. The net effect is that many settings that you made on one invocation are restored on subsequent invocations. This includes the window position, the state of various buttons, and so forth. While many of these settings are present for the application itself to manipulate you are free to manipulate them yourself to suit your needs. Of course, only do this when the application is not running, otherwise your values will be overwritten when you close the application.

Most settings are user-scoped, meaning that actions you take will directly or indirectly change these values and store them in your own `user.config` file. The first sub-section below details settings that are application-scoped, meaning they are read-only values and only appear in the `SqlDiffFramework.exe.config` file. The subsequent sections detail the user-scoped settings.

4.2.1 Application-Scoped Global Values

These values are application-scoped and appear only in the primary `SqlDiffFramework.exe.config` file.

4.2.1.1 UpdateCatalogFile

The `UpdateCatalogFile` setting specifies the name of the catalog file within the directory specified by `UpdateRepository` (section 4.2.1.2). This file records the first use of each user after a new installation.

4.2.1.2 UpdateRepository

The `UpdateRepository` setting specifies a search path (a semi-colon separated list of paths); the first path in the list that points to an actual directory will be used. You may include more than one path in the repository search path; this allows deploying the file repository in more than one location. This is useful in two typical scenarios where the search paths are network drives. The first is as a backup, so if one server is down you have a backup repository. The second is where you have two deployments that are not physically networked, e.g. a development environment and a test environment. Simply include both paths and they will be used appropriately.

4.2.2 Global States

These states are user-scoped and stored in the individual `user.config` file for each user. They are automatically updated based on your use of the application.

4.2.2.1 WindowState

This value records the state of the window (normal, maximized, or minimized) upon closing the application. If the window is normal or maximized upon closing, that state is restored. If minimized at closing, the state is restored to normal.

4.2.2.2 WindowPosition

This value records the size and position of the window upon closing the application. Both size and position are restored upon reinvocation, automatically taking into account a reduction of monitors or reduction of screen size.

4.2.2.3 UpdateChecked

This timestamp records the last time the application checked for an update. Reminders about a new version will not occur if this value is less than the number of days specified in `UpdateCheckInterval` beyond the current date/time value.

4.2.2.4 NewVersion

This Boolean allows the application to recognize the first run of a new version and to migrate settings from the previous version, if any. In the application-level configuration file this value defaults to true. Upon creating the user-level configuration file, it gets a permanent value of false. Since the latter overrides the former, it will always be read as false upon subsequent checks.


4.2.2.5 UnreachableRepository

This is a countdown counter (starting at five) that tracks unsuccessful attempts to reach the repository containing new releases of SqlDiffFramework. Each time you start SqlDiffFramework it attempts to check for a new release. If it is unable to complete its check—either due to a network problem or an undefined update location—it decrements the count by one. If it is successful before this count reaches zero, it resets to five. Once it reaches zero further update checks are disabled. If you resolve a network issue and want to reinstitute checking for updates, change this to a non-zero value. Contrariwise, if you never want to check for updates you can set this value to zero.

4.2.2.6 TandemButton_Checked

This Boolean value records whether the Tandem button () is enabled or not.

4.2.2.7 TurboSortButton_Checked

This Boolean value records whether the turbo-sort button () is enabled or not.

4.2.2.8 AutoDiffButton_Checked

This Boolean value records whether the Diff button () is enabled or not.

4.2.3 Editor Pane Shared States


These states are user-scoped and stored in the individual user.config file for each user. They are automatically updated based on your use of the application.

These buttons exist separately in both the left and right editor panes but only a single state is recorded, the one from the editor pane that is active at the time you close the program; of course this only matters if the two panes have different values.


4.2.3.1 UseLocalDataButton_Checked

Records whether the access is set to live or local for the last referenced editor pane.

4.2.3.2 AutoHighlightButton_Checked

Records whether the Auto-Highlight button () for the last referenced editor pane is enabled or not.

4.2.3.3 AutoExecuteButton_Checked

Records whether the Auto-Execute button () for the last referenced editor pane is enabled or not.

4.2.4 Editor Pane Distinct States

These states are user-scoped and stored in the individual user.config file for each user. They are automatically updated based on your use of the application.

4.2.4.1 LeftSqlDirectory

The file path for queries for the left editor pane.

4.2.4.2 RightSqlDirectory

The file path for queries for the right editor pane.

4.2.4.3 LeftCsvDirectory

The file path for CSV snapshots for the left editor pane.

4.2.4.4 RightCsvDirectory

The file path for CSV snapshots for the right editor pane.

4.2.4.5 LeftConnectionName

The selected DB connection name for the left editor pane.

4.2.4.6 RightConnectionName

The selected DB connection name for the right editor pane.

4.2.5 Database Connections

These values are user-scoped and stored in the individual `user.config` file for each user. You may explicitly adjust them in the connection editor (`Query → Edit Connections...`).

4.2.5.1 ConnectionList

This container holds an XML structure defining the set of database connections. Each connection specifies:

- the connection string needed to establish a connection with a target database;
- the database type (SQL Server, Oracle, or MySql);
- whether to store the password or not;
- a display name and description.

Section 2.2.1 shows examples of the actual format. While you could edit this directly in the configuration file if needed, it is simpler and safer to use the connection editor (see 3.9).

4.2.6 Program Options

These values are user-scoped and stored in the individual `user.config` file for each user. You may explicitly adjust them in the program options dialog (`Edit → options...`).

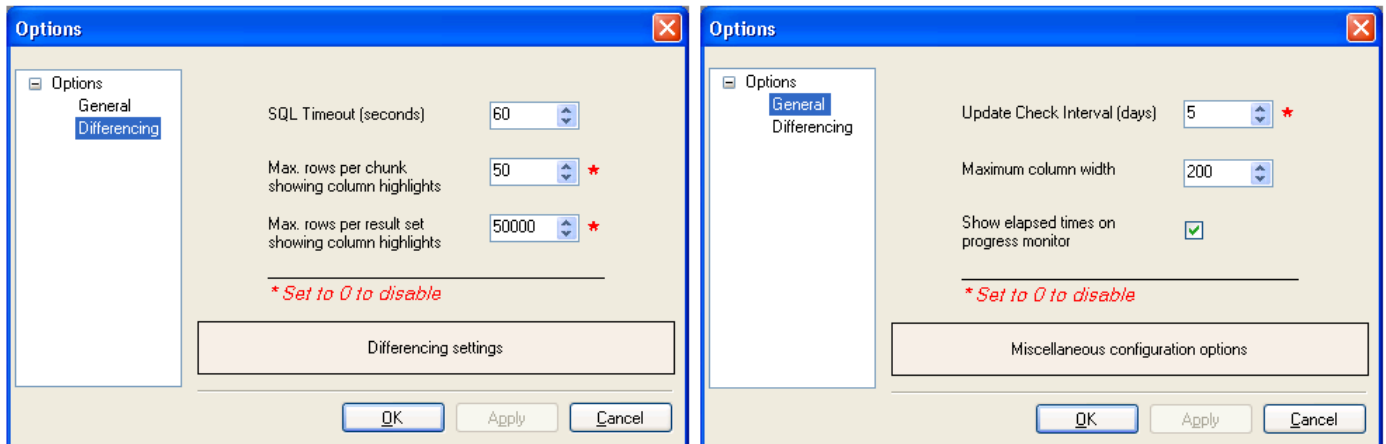


Figure 4-11 Options Dialogs

These dialogs show the program settings that you may explicitly set; most program settings are determined indirectly through your use of the application.

4.2.6.1 UpdateCheckInterval

This integer (in days) determines how often the application checks for updates upon startup. Set to zero to disable checking for updates.

Default=5 days, minimum=0, maximum=365.

4.2.6.2 MaxColumnWidth

This value (in pixels) limits the width of columns when you adjust column widths to fit the data (see section 4.5.1.2). Since a text column may contain lengthy strings (into thousands of characters), fitting the data with no constraint on column width would be hazardous.

Default=200 pixels, minimum=50, maximum=5000.

4.2.6.3 CommandTimeout

This value (in seconds) sets the timeout for SQL commands sent to a database. If the database has not responded within the specified interval, the SQL connection cancels the operation and notes this in the status bar for the editor pane.

Default=60 seconds, minimum=30, maximum=600.

4.2.6.4 MaxHighlightedRowsPerChunk

The difference analysis is performed first at the row level, then at the column level for those rows reporting a difference. A difference chunk consists of the maximum number of adjacent rows with differences. This integer value specifies the maximum number of rows in each chunk on which to perform the column-level difference analysis. Very large values could significantly slow the performance.

Default=50 rows, minimum=0, maximum=10,000.

4.2.6.5 MaxHighlightedRowsTotal

The difference analysis is performed first at the row level, then at the column level for those rows reporting a difference. This integer value specifies the maximum number of rows in the entire result set on which to perform the column-level difference analysis. Very large values could significantly slow the performance.

Default=50,000 rows, minimum=0, maximum=250,000.

4.2.6.6 ShowElapsedTimes

A Boolean flag indicating whether the progress monitor that pop-ups during execution should list tasks only or list tasks with elapsed time values (see section 4.1.1.10).

4.3 Working with the Editor Pane

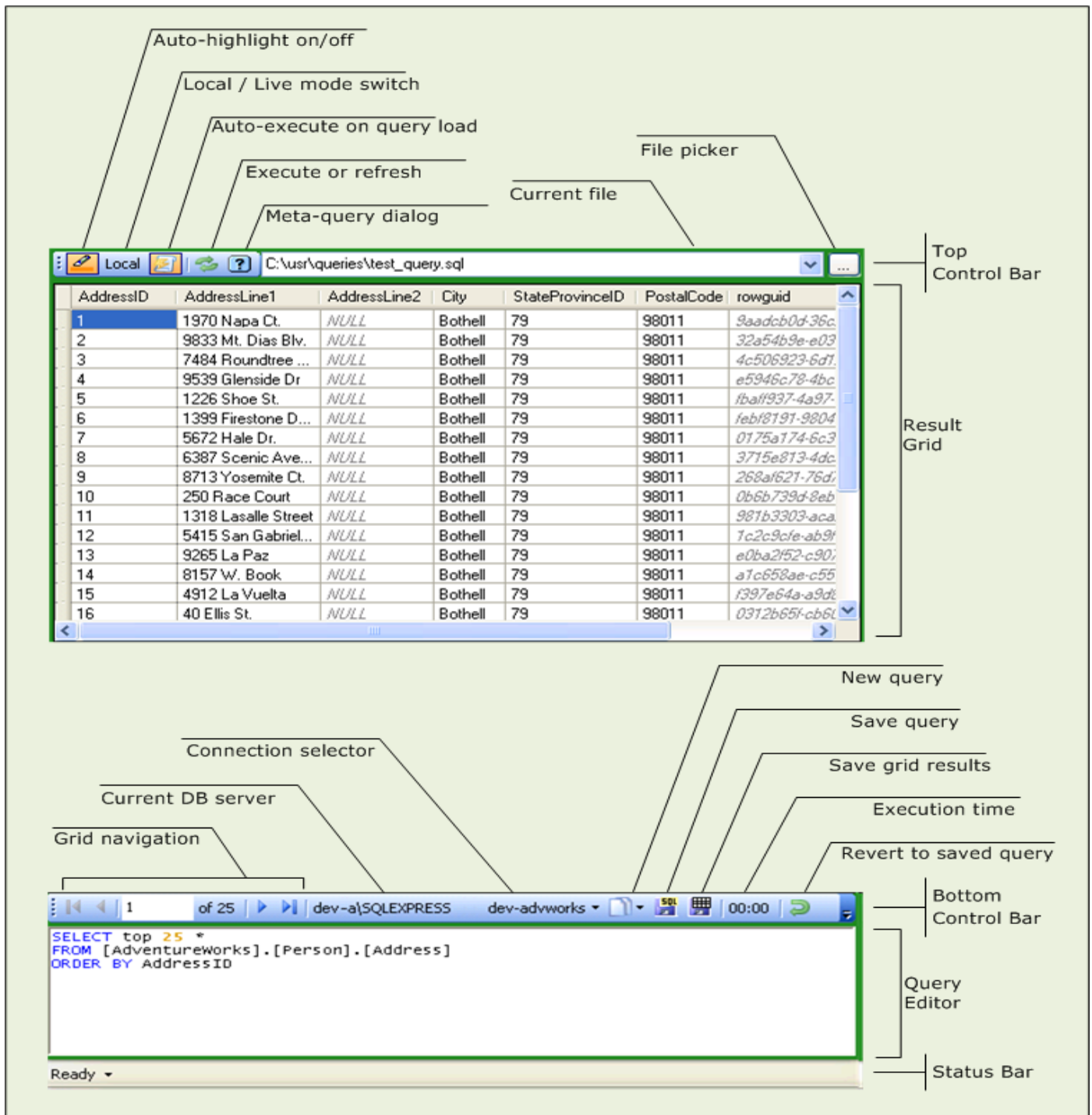


Figure 4-12 Editor Pane

All of the parts and controls of the two editor panes are labeled for easy identification. The query text shown in the Query Editor (near the bottom) is the contents of the Current File (near the top). The results of executing the query appear in the Result Grid.

4.3.1 Top Control Bar Elements

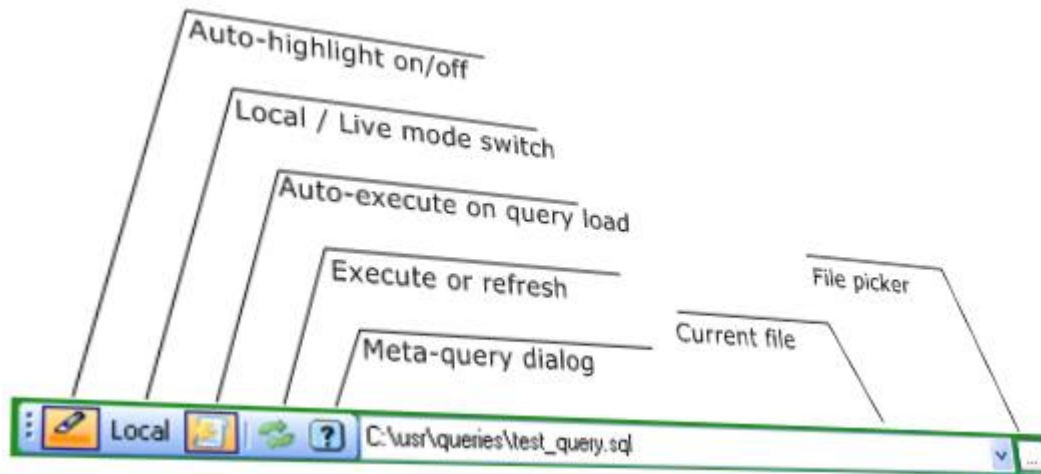





Figure 4-13 Exploded View of Editor Pane's Top Control Bar

4.3.1.1 Auto-highlight on/off

This mode switch () enables or disables auto-highlighting. Whether the existing text retains its highlighting or reverts to plain black text depends on the hide/display highlighting setting (see 4.4.1.4). Manual highlighting (see 4.4.1.5) may also be used instead of automatic highlighting.

Alternatives: `Highlight` → `Auto-highlight` context menu item (see 4.4.1.3).



4.3.1.2 Local/live mode

This mode switch () determines whether data is sourced from a live data source (database or ODBC data source) or from a CSV file. Local mode is useful to load archival snapshots of live data that you have saved with the Save Grid button () (see 4.3.2.6) that you can then compare to other archival snapshots or to live data. This gives you a powerful mechanism for looking for changes from the same data source.


Note that CSV files generated by SqlDiffFramework have extra information in the headers to allow the application to convert the data to actual data types (integers, dates, etc.) rather than treat all data as strings. Thus, if you load CSV files generated by any other means all columns in the result grid are just strings.


Toggling this mode does not automatically reload the result grid. To refresh the result grid from the alternate source, use the Execute/Refresh button () (see 4.3.1.4).

4.3.1.3 Auto-execute on query load

If this mode switch () is enabled when you load a new file into the **query editor** (see 4.3.1.7) the query will automatically be executed to retrieve the data, either from a live data source or from a data snapshot depending on the local/live mode () (see 4.3.1.2). If this switch is disabled and there is a result set from any previous operation showing, then the background of the query editor changes to pale blue to indicate the query is out of sync with the result set. (Note that technically any typing in the query editor immediately makes the query out of sync with the result set as well but presumably you know you are changing it so the background color does not change.)



4.3.1.4 Execute or refresh

For live mode (see 4.3.1.2): this button () executes the currently loaded text in the **query editor**, sending it to the currently selected database connection and displays the result set from the database in the **result grid**.


For local mode: this button () loads a CSV data file, which may be a file previously exported from SqlDiffFramework, or any other CSV file. Note that CSV files generated by SqlDiffFramework have extra information in the headers to allow the application to convert the data to actual data types (integers, dates, etc.) rather than treat all data as strings. Thus, if you load CSV files generated by any other means all columns in the result grid are just strings.

The first time you load data in local mode you are prompted with a standard file dialog, allowing you to navigate to the appropriate directory and select a file. SqlDiffFramework remembers the directory so the next time you do the same operation the file dialog is already in the correct directory (assuming you have your collection of CSV files in a single directory). That is standard behavior for .NET applications, of course. What goes above and beyond that however is this: SqlDiffFramework looks at the base name of your SQL file and automatically selects a CSV file with the same base name. If it finds a matching file it skips the file dialog and just processes the file.

Hover over the button when in local mode to see the directory from which local files will be retrieved.

Alternatives:  or  depending on whether you want to refresh one or both editor panes (see Working in Tandem).


4.3.1.5 Meta-query dialog

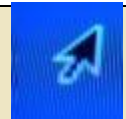
This button () opens up the **meta-query dialog**, a dialog designed to give you a selection of queries that can provide information about your database schema rather than about your data. The meta-query dialog is specific to the type of database so the current connection of the editor pane dictates the contents of its meta-query dialog. The meta-query dialog for

the partner editor pane would have different queries available if its connection is set to a different database type. Upon opening the dialog, you are prompted to select an individual meta-query and then supply its necessary parameters. See section 3.10.

Alternatives: The **Query → Meta-Queries...** menu item opens the meta-query dialog for the active editor pane (see 4.1.7.3).


4.3.1.6 Current file

This combination drop-down/type-in box serves dual duty, both displaying the currently loaded query file and allowing you to select a different one. The immediately adjacent control, the file picker button () is also available for selecting files (see 4.3.1.7). Any previous files that you have loaded in the editor pane during the current session are displayed when you open the drop-down. Alternatively, you could type directly in the text field; the file you picked loads upon pressing **Tab** or **Enter**. Note that as you type the combo box provides auto-completion—any files from the session history that have the same prefix are immediately shown as you press each character. Of course, what you type in is not limited to only what is in the history; typing a new name loads the new file and then adds it to the history as well.



TIP: Since all your SQL files will typically be in one directory and the path to that directory might be several levels down from the root, it could be a lot of typing if you (a) click in the type-in field and (b) press Delete or Backspace to clear the field. Instead do this: (a) click in the field; (b) press End to position the cursor at the end of the existing text; and (c) hold down Backspace to erase just the file name portion of the path. Then start typing and the combo box can quickly auto-complete the new file for you.

4.3.1.7 File picker

The file picker button () opens a standard file dialog, allowing you to navigate to the appropriate directory and select a file. SqlDiffFramework remembers the directory so the next time you do the same operation the file dialog is already in the correct directory (assuming you have your collection of SQL files in a single directory). The Save query command (see 4.3.2.5) uses the same location. The immediately adjacent control, the current file combo box, is also available for selecting files (see 4.3.1.6).

Alternatives: **Control+F6** for the left editor pane and **Control+F7** for the left editor pane.

4.3.2 Bottom Control Bar Elements

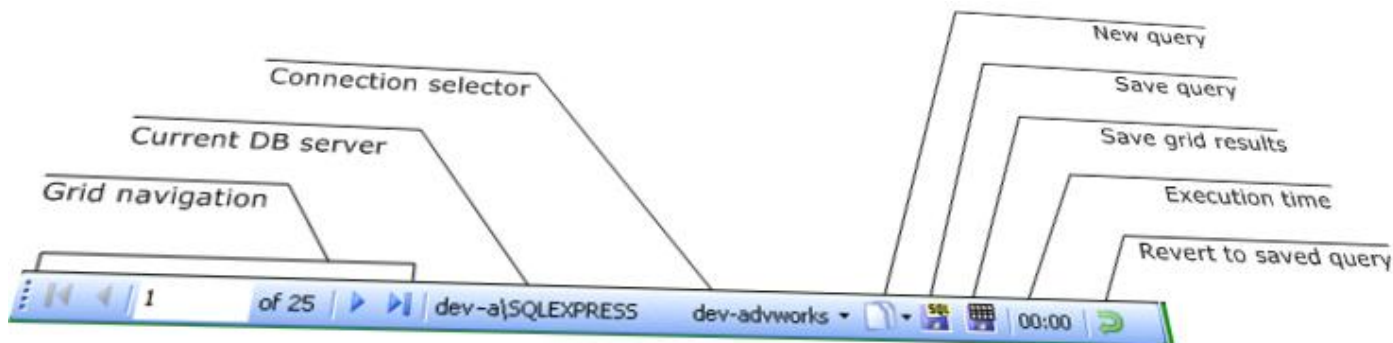


Figure 4-14 Exploded View of Editor Pane's Bottom Control Bar



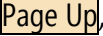
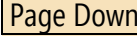
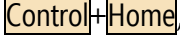
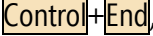


4.3.2.1 Grid navigation

This is a collection of controls letting you navigate among rows in the result grid. On the extreme ends you have buttons to move to the first or last row of the grid (keyboard equivalents: **Control+Home**, **Control+End**). Just inside those are buttons to move to the previous or the next row from your current location (keyboard equivalents ((**↑**) or **↓**)). That leaves the direct access control in the middle. The example in Figure 4-12 shows 1 of 25, indicating the current record is the first record and that there are 25 records in total. You may move to any other row simply by typing in the desired record number.

Hover over the total row count indicator to see the data source for the currently displayed grid. This is handy in two instances. First, although it usually mirrors the current DB server indicator, when you change the connection via the connection selector, the current DB server reflects this immediately. This tooltip does not—it always points to the source for the currently loaded grid. The second instance where this tooltip is useful is when you are loading local rather than live data and you have custom file names. Normally if your query is, say, `customers.sql`, the corresponding local version of the data would be `customers.csv`. But let's say you want to compare two slightly different versions of the same query *without* saving the query itself under two different names. Execute one version of the query against a live data source, then save the grid to `customers-old.csv`. Modify the query and execute again, then save the grid to `customers-new.csv`. Now switch to local mode, mirror the settings on the other editor pane, then load one CSV file on each side (by depressing **Alt** when you click **Refresh**). This tooltip will indicate the names of the respective CSV file—that saves you from having to remember which one you loaded on which side.



TIP: Of course, you probably will not know that you need to look at record 1643 but if you have 2000 records and you know you want to look about three-quarters of the way down, you can enter 1500.

Note that mouse actions (clicking a cell) and keyboard navigation movements ((, , , , , , , ) also update this current record number. Difference navigation movements, however, do *not* affect the current record number as they do not change the active cell. Difference navigation movements are reflected on the difference position indicator (section 4.1.2.1) of the results toolbar (section 4.1.2).

4.3.2.2 Current DB Server

This label identifies the database server associated with the current connection selection immediately adjacent (see 4.3.2.3). Specifically it is the value of the `Server` field in the connection editor (Figure 3-16).



TIP: Hover over this label to reveal the name of the user associated with the current connection.

Click this label to reveal all the details of the current connection except for the password.

Oracle Example:

```
-- Data Source=DevServer01;Unicode=True; User ID=test1
```

SQL Server Example:




```
-- Data Source=DevSqlServerA;Initial Catalog=AdventureWorks; User ID=test1
```

ODBC Example:

```
-- Dsn=MyOdbcConnection
-- (C:\Store\Version1\Test CSV files)
```


Notice that for an ODBC connection it not only reveals the connection string but digs further and indicates the file or path inside that ODBC connection. (This is something even the **ODBC Administrator** utility is reluctant to share: if your path is more than 20 characters or so—which most paths are—it does not show the full path!)

4.3.2.3 Connection Selector

Select any of your defined connections with this drop-down. A *connection* is a set of details to connect to a database. In order to have any choices available you need to use the connection editor, which you can open in two ways: either click the connection selector while holding down the  key, or use the `Query → Edit Connections...` menu item (see 4.1.7.4). The connection editor lets you define connections to any number of SQL Server, Oracle, or MySQL databases as well as ODBC data sources. You can either manually enter connections or import a previously defined set that you have saved. The connection editor provides a combination test button and status indicator that lets you immediately test a connection you are defining. It also has  and  buttons for exporting and importing respectively. See: section 3.9.

If you do not include passwords when you create connections in the connection editor, SqlDiffFramework prompts you for a password when you select a connection that requires one. It does *not* validate the password at that time; it validates the password only when you attempt to execute a query against the database. If you have entered the wrong password the status bar lets you know. You can correct the password in one of two ways; either go into the connection editor and enter a password, or click the connection selector while you depress the **Shift** key. SqlDiffFramework will then prompt you for a password. See 3.9.1 for more details on passwords.

4.3.2.4 New query

This button () is overloaded with three related functions, one on the main button itself and two on the attached dropdown.

(1) Main button

Pressing the main button unloads the current query file and empties the query editor, leaving you a blank slate.

(2) Dropdown selection: New file but retain buffer


To unload the current query *file* but keep its *contents* in the query editor, open the dropdown and select this choice.

(3) Dropdown selection: Generate new query from results

Unload the current query file and then auto-generate a query from the current result set with this choice from the dropdown. The query will not match any row filtering predicates you might have in mind. Rather, it is a quick way to enumerate all the fields in the current result set. Once supplied, prune the list to just those fields you are interested in.

Type a minimal query to grab all the fields from one or more tables using the asterisk (*) wildcard:

```
SELECT * FROM some_table
```

Execute that query then depress **Shift** as you click the New Query button (). The application will prompt you to save the temporary query; just click **No** unless you want to save it for something. SqlDiffFramework then converts the temporary query into a query enumerating all the fields in the current result set, putting them in both a **SELECT** clause and an **ORDER BY** clause. You need only supply the data source in between. Figure 4-15 illustrates the steps. (Also see section 4.5.1.11 if you just want to copy some or all of the field names from the grid.)

Having the **ORDER BY** clause is particularly useful if you are reading data in local mode from a snapshot (CSV) file. Snapshot files are loaded *as is*. That is, they ignore the contents of the

query editor. Unless, that is, you turn on Turbo mode to do internal sorting in which case the `ORDER BY` clause is used. See section 3.5.2 for more.

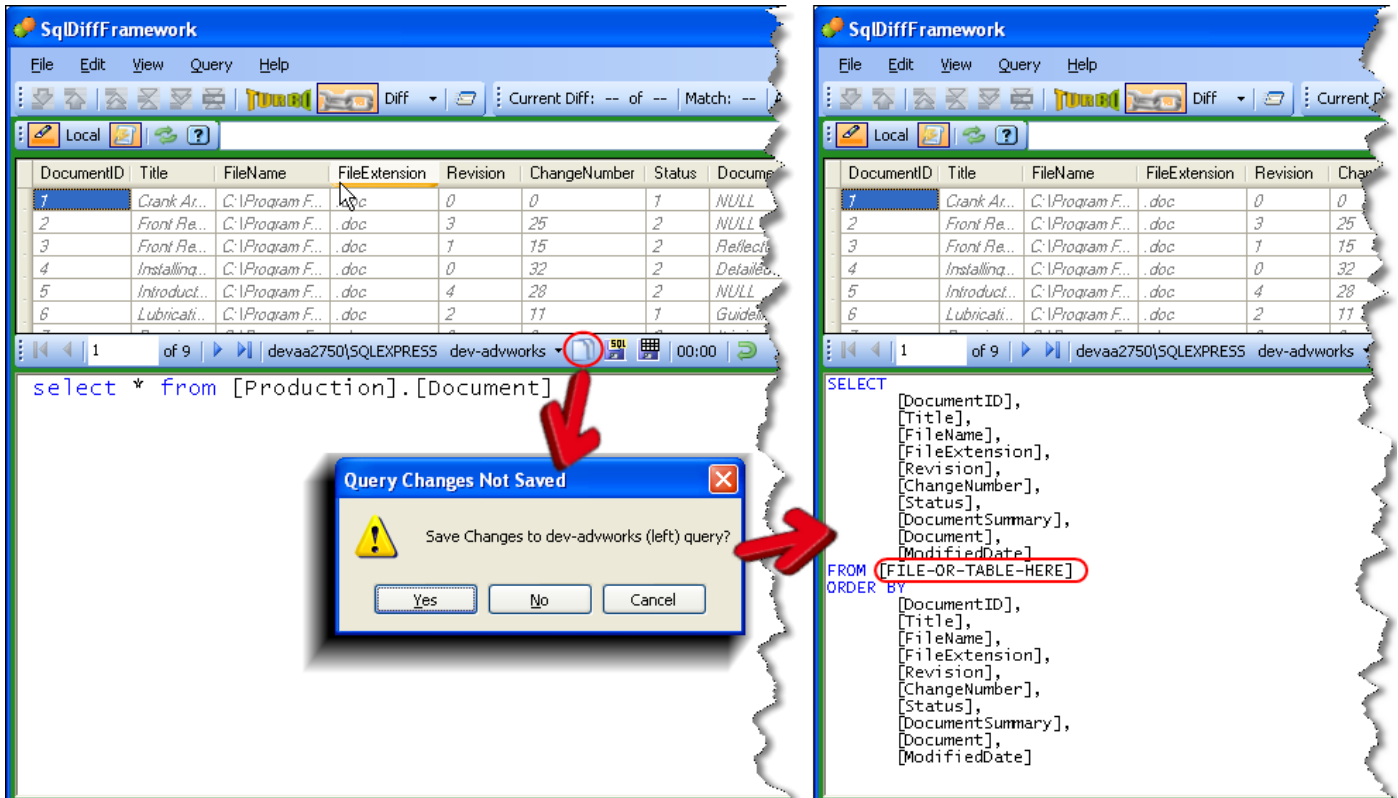




Figure 4-15 Auto-Query Generation

Generate an enumerated field list for your data in a couple easy steps: Execute a query to select all fields with the asterisk (*) shortcut as shown on the left. Invoke New Query → *Generate new query from results* to transform the query into an enumerated list of the fields in the result set; you just need to supply the data source in the FROM clause.

4.3.2.5 Save query

This button () saves the query to the current file name or, if none, prompts you with a standard save dialog, allowing you to navigate to the appropriate directory and enter a file name. SqlDiffFramework remembers the directory so the next time you do the same operation the file dialog is already in the correct directory (assuming you keep your collection of SQL files in a single directory). The **file picker** (see 4.3.1.7) uses the same location.

The button is disabled unless you have made changes to the currently loaded file.




TIP: To save the query to a new file name or directory, depress **Alt** as you click the button.

Hover over the button to see the current directory for queries.

Alternatives: **Control+F8** for the left editor pane and **Control+F9** for the left editor pane.

4.3.2.6 Save grid results

This button () saves the current result set to a CSV file, a snapshot that you can use later for comparing data from the same source over time. CSV files saved by SqlDiffFramework carry extra information in the headers to allow the application to convert the data to actual data types (integers, dates, etc.) rather than treat all data as strings. If you load CSV files generated by any other means all columns in the result grid are just strings. Note that you could also export the data directly into Excel if you just want to manipulate it rather than save a snapshot for later (see 4.5.1.5).


See also: CSV import/export.

4.3.2.7 Execution time

This indicator displays the time of the last query execution in minutes and seconds.

Hover over the indicator to obtain further resolution of the execution time (down to milliseconds) along with the file name and when the query was executed.

4.3.2.8 Revert to saved query

This button () discards any changes you have made in the query editor, effectively reloading the last saved contents. If you are working in an unnamed buffer (i.e. you have neither loaded a file into the buffer nor saved the buffer to a file) the buffer reverts to empty. Once you do this, or when you load a fresh file, the button is disabled since it has nothing to do.

4.4 Working with the Query Editor

While you could use your favorite database-specific utility to design your queries (e.g. SQL Server Management Studio, Query Analyzer, Oracle's SQL Developer, SQLyog, etc.), SqlDiffFramework includes a smart and flexible editor in its own right. Indeed, its powerful syntax-highlighting capabilities include some features that the above commercial tools do not have. Furthermore, the built-in editor distinguishes between different SQL dialects for SQL Server, Oracle, and MySQL. It maintains a separate lexicon for each dialect so it highlights the keywords specific. Out of the box the highlighting styles between dialects are all the same but it is easy to customize to, for example, highlight keywords in blue for SQL Server and green for Oracle if you wish. (See: Syntax Highlighting Customization).

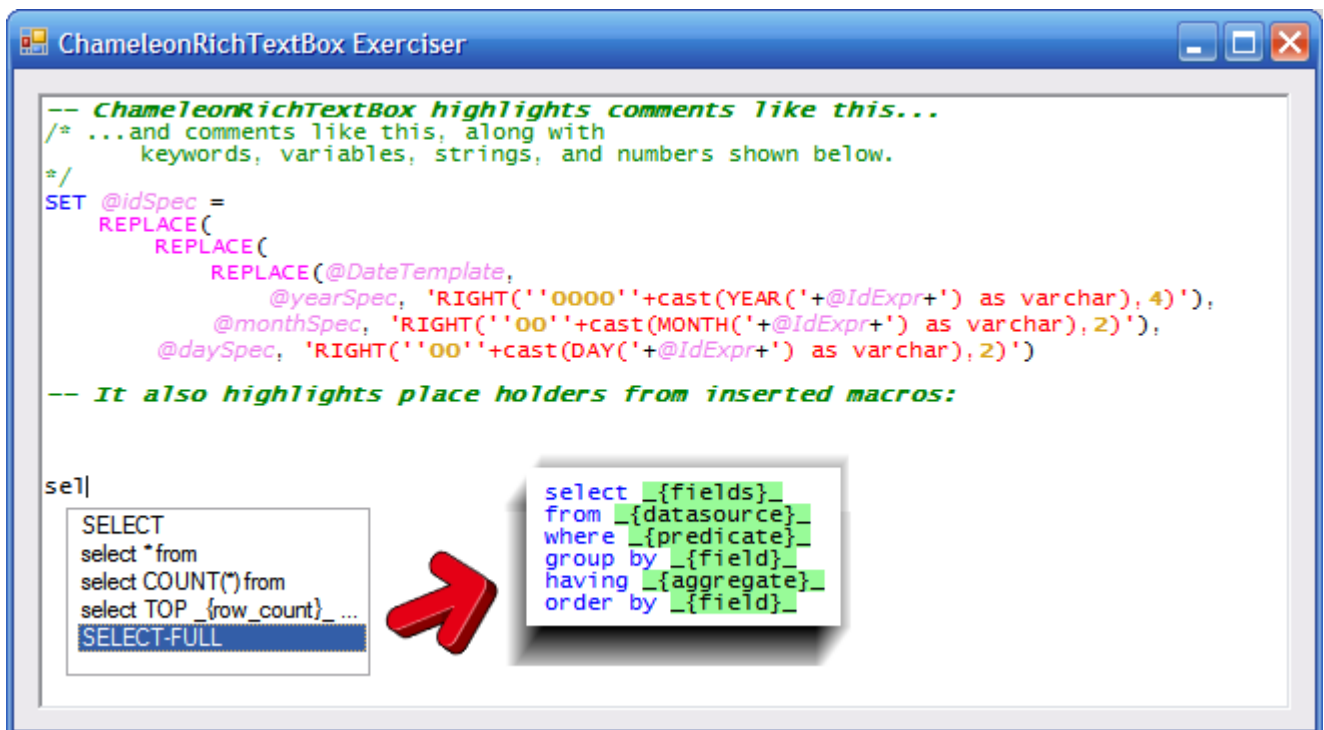


Figure 4-16 Query Editor

This figure illustrates the different types of highlighting available (keywords, comments, variables, strings, numbers, and place holders) as well as keyword completion.

4.4.1 Key Features

4.4.1.1 Dialect-specific syntax highlighting

As shown in Figure 4-16, the query editor recognizes six different elements to highlight: keywords, strings, comments, variables, numbers, and macro place holders. Though some of these are not dialect-specific, these elements are defined separately for each dialect (SQL


Server, Oracle, and MySQL) allowing for customization if desired. You might, for example, want to stylize imperative keywords differently from function names or constant names. See section 3.11.5 for details on customization.

4.4.1.2 Recognizes SQL Server, Oracle, and MySQL dialects out-of-the-box

The query editor supports three common SQL dialects: SQL Server, Oracle, and MySQL, as well as a generic SQL context usable for ODBC data sources. Each SQL dialect has its own context file, detailing how to stylize each of the six elements from 4.4.1.1.

4.4.1.3 Enable or disable automatic highlighting

CM: **Highlighting** → **Auto-highlight** (see Figure 4-17, pane 2)

Disabling automatic highlighting suspends further highlighting as you type; it performs exactly the same function as the editor pane's Auto-highlight button ()—see 4.3.1.1. Whether the existing text retains its highlighting or reverts to plain black text depends on the hide/display highlighting setting (see 4.4.1.4). Manual highlighting (see 4.4.1.5) may be used instead of automatic highlighting.

Alternatives: Auto-highlight on/off button on the top control bar (see 4.3.1.1).

4.4.1.4 Hide or display highlighting

CM: **Highlighting** → **Hide highlight when disabled** (see Figure 4-17, pane 2)

When you disable highlighting (see 4.4.1.2) you may elect to retain what has already been highlighted, or you can revert to plain black text. Even if you hide the highlighting everything will be highlighted again if you later re-enable automatic highlighting.

4.4.1.5 Automatic or on-demand highlighting

Shortcut: **Control**+**Shift**+**H**




If automatic highlighting is disabled (see 4.4.1.2), you may manually highlight at any time using this shortcut.

4.4.1.6 Distinguishes between end-of-line comments and block comments

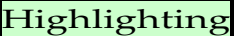
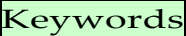
Since standard SQL includes two forms of comments the editor highlights them differently, letting you easily distinguish between the two. As shown in Figure 4-16, end-of-line comments are emboldened while block comments are not. See also: section 4.4.2.3.

4.4.1.7 Multiple delimiters assignable for strings and variables

Strings are recognized as any characters delimited by either **'** (apostrophe) or **"** (double-quotation mark). Both are, by default, stylized the same but you may customize them to stylize differently. Similarly, T-SQL recognizes two types of parameters, user parameters

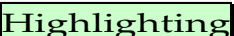
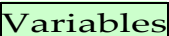
(beginning with ) and system parameters (beginning with  ). These may be stylized separately as well if desired. . See also: section 4.4.2.2 and 4.4.2.4.

4.4.1.8 Instant conversion of keywords to uppercase or to lowercase

CM:  →  (see Figure 4-17, pane 3)

By default, keywords that you type are stylized but the case of the words is unaltered—if you type in lowercase it remains in lowercase. But from the context menu you may set all keywords to be uppercase or lowercase. So even if you are typing in lowercase as soon as the editor recognizes a keyword it converts it to uppercase. See section 3.11.3 for subtle points to keep in mind.

4.4.1.9 Instant conversion of variables to uppercase or to lowercase

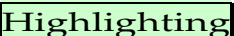
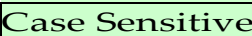
CM:  →  (see Figure 4-17, pane 3)

By default, variables that you type are stylized but the case of the words is unaltered—if you type in lowercase it remains in lowercase. But from the context menu you may set all variables to be uppercase or lowercase. So even if you are typing in lowercase as soon as the editor recognizes a variable it converts it to uppercase.

4.4.1.10 Differentiate groups of keywords

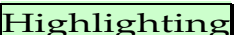
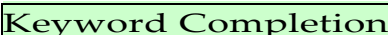
Imperative keywords, function names, and constants are stylized differently because they are in separate groups in the context definition for each SQL dialect. See also: section 4.4.2.1.

4.4.1.11 Make keyword recognition case sensitive or insensitive

CM:  →  (see Figure 4-17, pane 2)



When enabled, the keywords you type must match the case exactly as the keyword is defined in the current context. When disabled, upper or lower case is ignored.

4.4.1.12 Enable or disable keyword completion

CM:  →  →  (see Figure 4-17, pane 4)

Keyword completion saves you typing: instantly see a list of all words beginning with the current prefix you have typed and insert it in the editor.

4.4.1.13 Invoke keyword completion

Shortcut: +

When you invoke keyword completion the editor looks at the text before your cursor and displays a list of all words or phrases beginning with the prefix you typed. (If the cursor is

adjacent to a space, then all available words and phrases will be included.) You select a word or phrase from the list with keyboard arrows then press either **Space** or **Enter**. Alternatively, use the mouse to scroll then click. Either way, the list closes and the item you selected replaces the prefix you typed in the editor.

4.4.1.14 Instant case conversion of auto-completed phrases

CM: **Highlighting** → **Keyword Completion** (see Figure 4-17, pane 4)

By default, auto-completed phrases are inserted in the case defined by the context for the particular SQL dialect. That is, if the context defines **SELECT** in uppercase, it will be inserted in uppercase regardless of whether you used the prefix **SEL** or **sel** or even **sel**. You may modify this behavior to insert auto-completed phrases in uppercase or in lowercase, or have the case match the case of your entered prefix (Match user case on the context menu). See 3.11.3 for subtle points to keep in mind.

4.4.1.15 Macros speed your typing

Shortcut: **Control+Space**

Each SQL dialect (SQL Server, Oracle, and MySql) comes with several general purpose macros. Macros may be either *anonymous macros*, where the macro phrase itself appears in the auto-completion list, or *named macros*, where just the name of the macro appears in the auto-completion list. Either way a macro may consist of a single word or multiple words on a single line or on multiple lines. Long one-line macros or any multiple-line macros should generally be named macros in order to keep the auto-completion list clean and compact. See 3.11.4 for more on defining, editing, and using macros.

4.4.1.16 Macros may be static or dynamic

Shortcut: **Control+Space**

Macros may either contain static, literal text or they may contain text mixed with place holders to mark where you should insert actual values. Place holders are stylized uniquely for easy identification. See 3.11.4 for more on using macros.

4.4.1.17 Navigate among place holders in a macro

Shortcut: **F4**

Once you insert a macro with place holders (see 4.4.1.16), use the **F4** shortcut to advance to each place holder in turn. When you land on a place holder in this fashion it is automatically selected so that you need only start typing to replace it. Advancing through place holders goes only in one direction through your text, but if you miss one, keep going: when you reach the last one the next **F4** takes you back to the first one. See 3.11.4 for more on using macros.

4.4.1.18 Clean up unneeded place holders

Shortcut: **Control+Shift+Delete**

Some macros include optional components, so you may not need every element every time you use it. The `SELECT-FULL` macro is a good example. This is a template for a simple select statement, including where, group by, having, and order by clauses, each with a place holder. Say you do not need the having clause. Advance through the place holders using F4 (see 4.4.1.17); when you reach the line with the having clause, just press **Control+Shift+Delete** to remove the entire line. This shortcut deletes not just the selected phrase but the entire line which contains it. Therefore, good macro design dictates separable elements should be on separate lines. See 3.11.4 for more on using macros.

4.4.1.19 Delete range by line boundaries

Shortcut: **Control+Shift+Delete**

This shortcut is actually not tied specifically to place holders (4.4.1.18). You may, in fact, use it to delete complete lines touched by the current selection irrespective of the presence of a place holder. If nothing is selected, it deletes the current line containing the cursor.

4.4.1.20 Comment or uncomment a region

Shortcut: **Control+Shift+C** [comment] and **Control+Shift+U** [uncomment]

Use **Control+Shift+C** to quickly comment a range of lines in the editor. An end-of-line comment token plus a single space are appended to the beginning of each line that your current selection touches. If you have just an insertion point rather than a text selection, just the single line containing the insertion point is commented.

Use **Control+Shift+U** to uncomment a range of lines in the editor. Only lines within your selected text that begin with an end-of-line comment token will be affected (though you may have leading whitespace on the line as well). The comment token and a single following whitespace character (tab or space), if any, are removed from each such line. Any whitespace before the comment tokens is unaffected. See 4.4.2.3 for more information on comments.

4.4.1.21 Increase or decrease the font size

Shortcut: **Control+>** [increase] and **Control+<** [decrease]

Increase or decrease the font size of the current selection. When auto-highlighting is enabled, this works only if all text is selected. With auto-highlighting disabled, it works on any selection.

4.4.1.22 Increase or decrease the indent of a region

Shortcut: **Alt+>** [increase] and **Alt+<** [decrease]

Use **Alt+>** to shift the current selection to the right by adding a tab (or spaces depending on the **ExpandTab** setting) to the start of each line. If **ExpandTab** is enabled, the number of spaces used is determined by the **TabSize** setting.

Use **Alt+<** to shift the current selection to the left by removing a tab (or spaces depending on the **ExpandTab** setting) at the start of each line. See 4.4.1.23 for more on the **ExpandTab** setting.

4.4.1.23 Customize the tab key to insert tab characters or spaces

CM: **Tab Control** → **Expand Tab** (see Figure 4-17, pane 5)

Working in conjunction with the **Tab Inserts Characters** setting, **Expand Tab** is another mode toggle relevant to editing in the **query editor**. When turned on and you insert a tab character in the query editor (via either **Tab** or **Control+Tab** depending on the **Tab Inserts Characters** setting), this setting determines whether that actually inserts a tab character or an equivalent number of spaces. The number of spaces is determined by the setting (2, 4, or 8) in the same context menu.

4.4.1.24 Customize the tab key to act within the editor or not

CM: **Tab Control** → **Tab Inserts Characters** (see Figure 4-17, pane 5)

Standard behavior of Windows applications is that the **Tab** key advances focus from one field to the next—the classic example of this is a form where you are filling out name, address, etc. You type something in one text field then press **Tab** to advance to the next text field, etc. That works fine for single-line text boxes as well as buttons, check boxes, and other elements. The model becomes slightly muddled though when you get to a multiple-line text container such as this query editor or the result grid. In the editor, should the **Tab** key still advance to the next element on the screen (perhaps a button) or should it insert 8 spaces in your text? In the result grid, should it advance out of the grid to the next element or should it advance to the next cell within the grid? Both scenarios describe standard behaviors depending on your situation so this setting allows you to decide for yourself which way to operate. When enabled, the **Tab** key acts within the editor or the result grid, inserting characters or moving among cells; when this setting is disabled, the **Tab** key advances to the next element. Note that whichever way you set it, you may temporarily toggle it to the opposite setting by depressing the **Control** key in conjunction with the **Tab** key. Another way to say this is that **Tab** and **Control+Tab** always do opposite actions: if **Tab** acts within the element, then **Control+Tab** advances to the next element and vice versa.

4.4.1.25 Set the number of spaces inserted by the Tab key

CM: **Tab Control** → **n Spaces** (see Figure 4-17, pane 5)

If **Expand Tab** is enabled, then when you press the **Tab** key it inserts a fixed number of spaces determined by this setting rather than a tab character.

4.4.1.26 Search by text, wildcard, or regular expression

Shortcut: **Control+F** [search] and **Control+H** [replace]

SqlDiffFramework provides a standard search-and-replace mechanism for editing convenience. Beyond just a basic text search, though, you have the flexibility to alternately search by wildcards or by regular expressions.

See 3.4.1 for more on the search-and-replace facility.

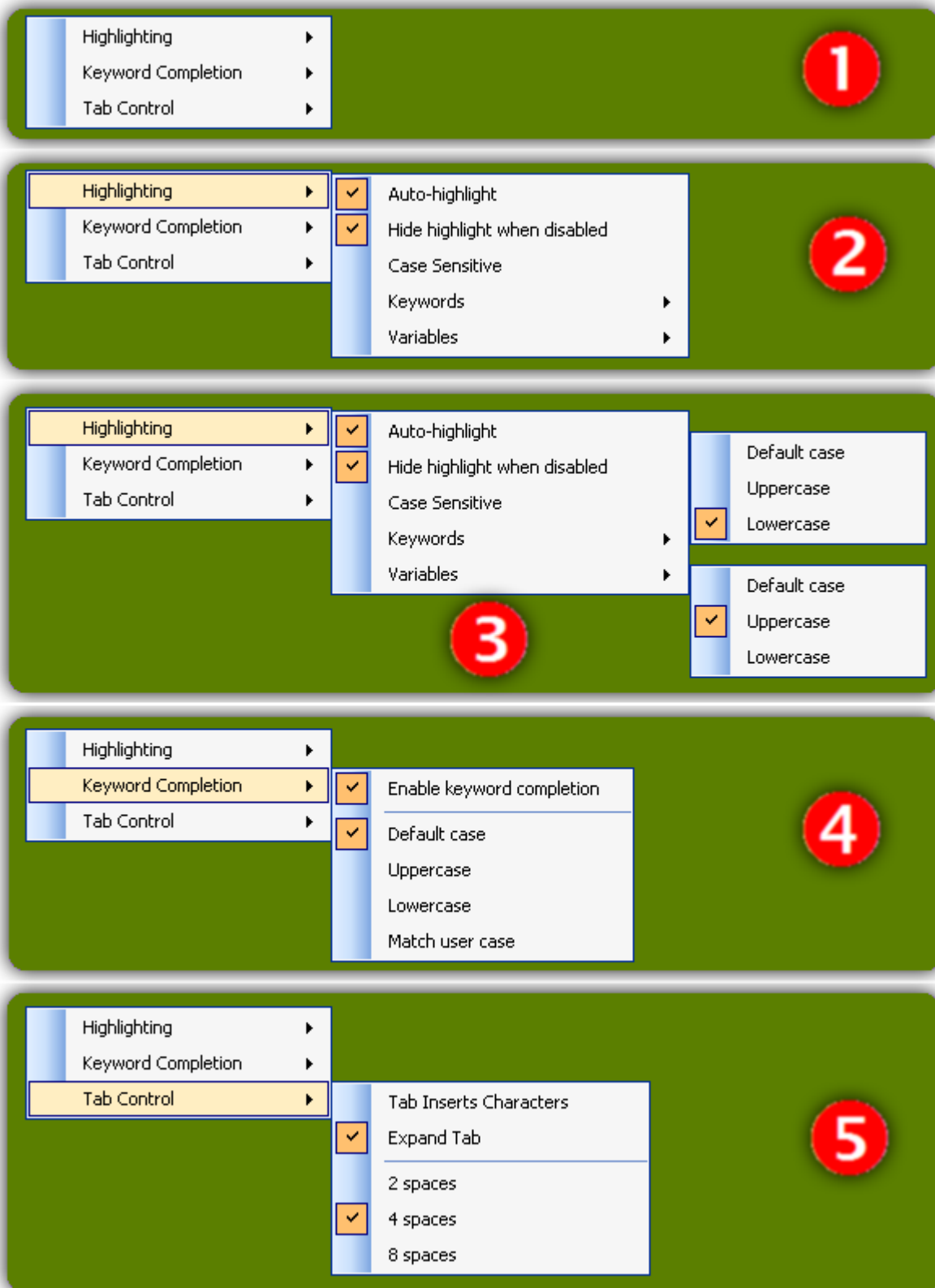


Figure 4-17 Query Editor Context Menus
 Menus are expanded in the five panes shown to display all available settings.

4.4.2 Types of Highlighting

The following categories of elements may be stylized by specifying a font or a color, or italicizing, or emboldening. Each supported dialect (SQL Server, Oracle, MySql) has a separate specification for each of these, though by design the default specifications mirror each other. You may, however, customize any of them individually. (See: Syntax Highlighting Customization 3.11.5).

4.4.2.1 Keywords

These are reserved words of the language, including imperatives (select, join), function names (min, max, len), operators (and, or), constants (null), etc. These groups of keywords each have their own rendering styles. Imperatives are blue and function names are magenta, for example. Keyword highlighting is adjustable via the **Highlighting** → **keyword** submenu of the context menu.

Default: none

4.4.2.2 Variables

A variable (or parameter) name consists of a sigil and its immediately adjacent word (e.g. @param or @@sys_info in T-SQL). Variable highlighting is adjustable via the **Highlighting** → **variable** submenu of the context menu. SQL Server recognizes @ and @@ while Oracle and MySql recognize just @.

Default: @

4.4.2.3 Comments

Comments come in two varieties, *block comments* (those delimited with both a starting and an ending token), and *end-of-line comments* (those with a starting token and an implicit ending token of the end of the same line).

Default: /* and */ for block comments, -- (double-hyphen) for end-of-line comments.

4.4.2.4 Strings

A string is any sequence of characters bracketed on either end by the same delimiter.

Default: " (double-quotation mark)

4.4.2.5 Numbers

A number is any contiguous sequence of digits.

Default: not applicable



4.4.2.6 Place holders

A place holder occurs only when you invoke a macro expansion at runtime. Any word (letters, digits, or underscores) bracketed by the place holder tokens will be highlighted as specified in the context. You may then sequence through them with the F4 key. Examples:

`_{field1}_` or `_{table name}_`.

Default: `_{ and }_`

4.5 Working with the Results Grid

The **results grid** displays the result set from the query you execute or, in local mode, from the stored data snapshot. Figure 4-18 shows sample data loaded from a query of the AdventureWorks database. The grid has a context menu that may be opened by right-clicking on any part of the grid header; the following sections describe the commands on the context menu. The grid also provides a powerful filtering mechanism that allows you to post-process the data you receive from your database. When you invoke the data filter (either by context menu or by shortcut), it opens as a one-line panel at the bottom of the grid with an input box, an Apply button () , and an Erase button () , as shown in the figure.

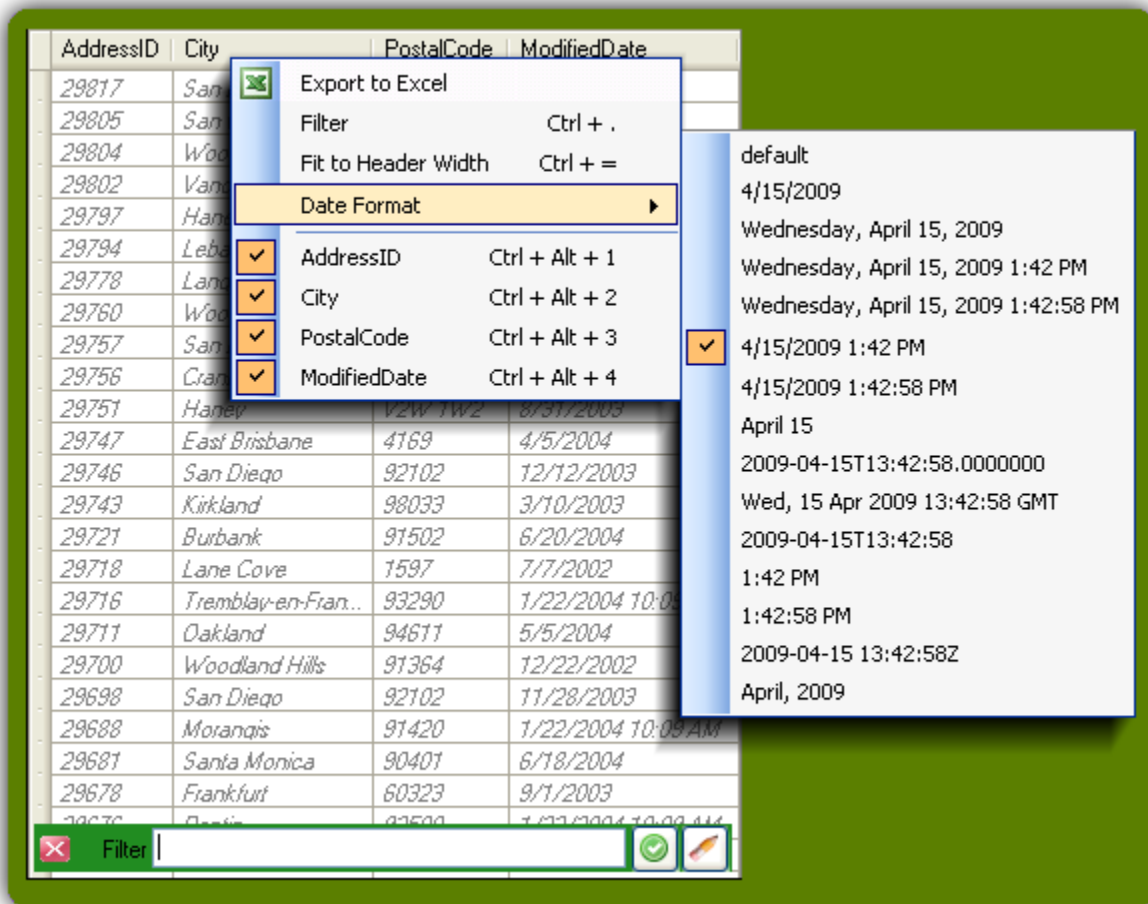


Figure 4-18 Results Grid

The results grid has *context menus* and a *filter capability*. The two-level context menu is partly dynamic, listing the current fields for showing or suppressing. The on-demand filter panel accepts simple SQL expressions for filtering the data to reduce the number of rows locally (i.e. without a round-trip to the database).

4.5.1 Key Features

4.5.1.1 Filtering data

Shortcut: **Control**+**'** (apostrophe)

CM: **Filter**

A useful feature of the result grid is that you can modify the result set *after* you get it from your database. Upon opening the filter panel type in a SQL expression then press the

Apply button (). For the data shown in Figure 4-18 you could, for example use `(AddressID < 29000) or (City like 'East%')`. See section 3.4.3 for more details.

4.5.1.2 Reveal data type of a column

Shortcut: **Control**+**Shift** and **Control**+**Alt** then hover over column header

When a results grid is active (i.e. keyboard actions are going to the results grid rather than, say, the query editor) then pressing either of these key combinations and hovering your mouse over a column header reveals the data type of that column. See Figure 4-19 for an illustration.

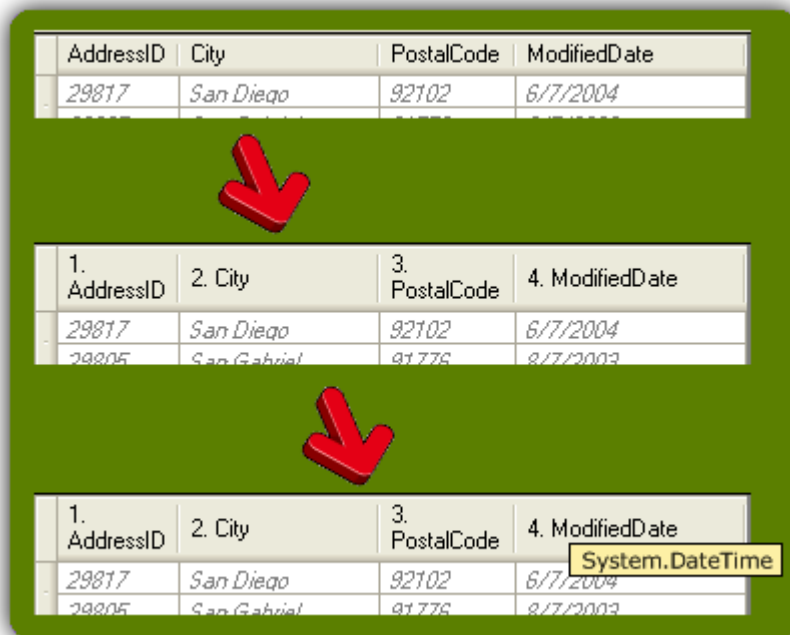


Figure 4-19 Revealing Column Information

The top fragment shows the grid in its normal state. When you press **Control**+**Shift** or **Control**+**Alt** column numbers are added; these are references for hiding or sorting columns from the keyboard. With those keys depressed, hovering over a particular column also reveals the data type of that column.

4.5.1.3 Adjusting column widths to fit data or headers

Shortcut: **Alt+=**

CM: **Fit to Header Width**

When you load a result set all columns start out at a standard 100 pixel width. This toggle lets you use two more useful configurations:

- When enabled: column widths snap to the size of the header cell text.
- When disabled: columns shrink or expand to be just wide enough to accommodate the longest datum they contain.

Figure 4-20 demonstrates the difference. Note that since text fields may contain very long strings and screen width is quite limited by comparison, any column may grow to at most 200 pixels wide.

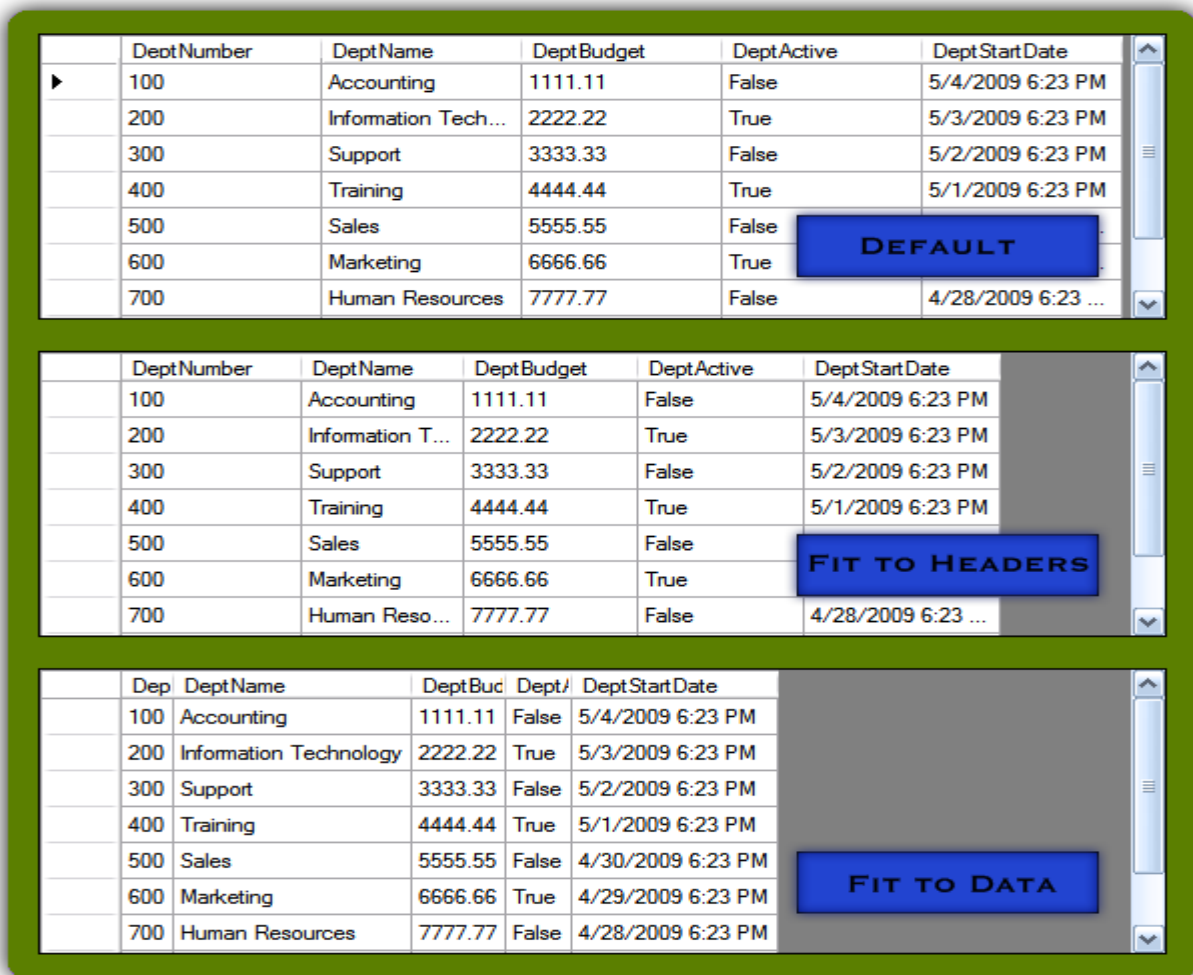


Figure 4-20 Adjusting Column Widths

Use the two fitting commands to toggle the data to fit the header widths or fit the largest data width.

4.5.1.4 Increase or decrease row heights

Shortcut: **Control**+**↑** and **Control**+**↓**

These two shortcuts allow you to increase or decrease grid row heights to either make the data easier to read or to pack in more on the screen.

4.5.1.5 Export to Excel

CM: **Export to Excel**

The controls on the editor pane provide a button to save the result grid to a CSV file, typically for later reloading of the data (see 4.3.2.6). This menu item provides another option, allowing you to export the data directly into Excel. Note that the Export to Excel menu item will only appear on your context menu if the application can detect that it is installed. I have found that even with a single configuration (Windows XP and Excel 2003) the installations may still vary, because on one system I have the Excel export option and on another I do not.

4.5.1.6 Change the format of all displayed date/time values

CM: **Date format**

Date/time values are typically stored in a database with quite precise values, often down to microseconds. When you display a date/time value, however, usually a resolution of seconds is sufficient. Sometimes minutes or even days will do. So you may select any of the available data formats shown in Figure 4-18 and your choice applies to all date/time values in all columns. Note that this does not just change the cosmetic appearance of the value; it also affects the value used for comparison between this results grid and its partner. Say, for example, the two result sets you are comparing have a date/time column where the values, for some reason, may sometimes differ by a fraction of a second. If your date/time values include seconds (as in 4/15/2009 1:48:52 PM) then you will inevitably have some values that end up differing by one second. Use the context menu to change the format to resolve to minutes (4/15/2009 1:48 PM) in order to significantly reduce the noise in the data.

4.5.1.7 Reveal column numbers

Shortcut: **Control**+**Shift** or **Control**+**Alt**

When a results grid is active (i.e. keyboard actions are going to the results grid rather than, say, the query editor) then pressing either of these key combinations alters the column headings to include column numbers. When you release the keys the numbers disappear. This is not a terribly exciting feature by itself but this feature allows you to instantly hide (see 0) or to sort (see 4.5.1.9) columns from the keyboard. See Figure 4-19 for an illustration.

4.5.1.8 Show or hide columns

Shortcut: **Control**+**Alt**+**1** through **Control**+**Alt**+**9**

CM: **field name**

The lower half of the top-level context menu in Figure 4-18 enumerates up to the first nine columns in the results grid along with an associated shortcut key combination. Either the context menu item or the key combination will toggle the visibility of the specified column. Initially all columns are displayed, indicated by the check marks in the context menu. When you select an item it toggles the state, removing the check mark and hiding the column. The particular number assigned to the column (e.g. since the context menu shows **Control**+**Alt**+**1** for AddressID, that indicates its column number is 1) is the same number you will see when you reveal the column numbers (see 4.5.1.7). Note that hiding a column does not affect assigned column numbers. So if you have hidden column 2 and then reveal column numbers you will still see AddressID with a 1 and PostalCode with a 3. So to re-display column 2 your choices are the same: either use the context menu or press **Control**+**Alt**+**2**.

Figure 4-21 illustrates that shortcut keys are assigned only to the first nine fields, using keys **Control**+**Alt**+**1** through **Control**+**Alt**+**9**. Any additional fields may still be shown or hidden but only from the context menu.

4.5.1.9 Sort a column from the keyboard

Shortcut: **Control**+**Shift**+**1** through **Control**+**Shift**+**9**

Mouse: click a column header

The standard technique in .NET applications for sorting a grid by a particular column is to click the header of the column with your mouse. The results grid provides a convenience feature letting you also do this from the keyboard for up to the first 9 columns. The particular number assigned to each column is the same number you will see when you reveal the column numbers (see 4.5.1.7).

If Auto-Differencing is enabled (see 4.1.1.9) the two result sets will immediately be re-analyzed.

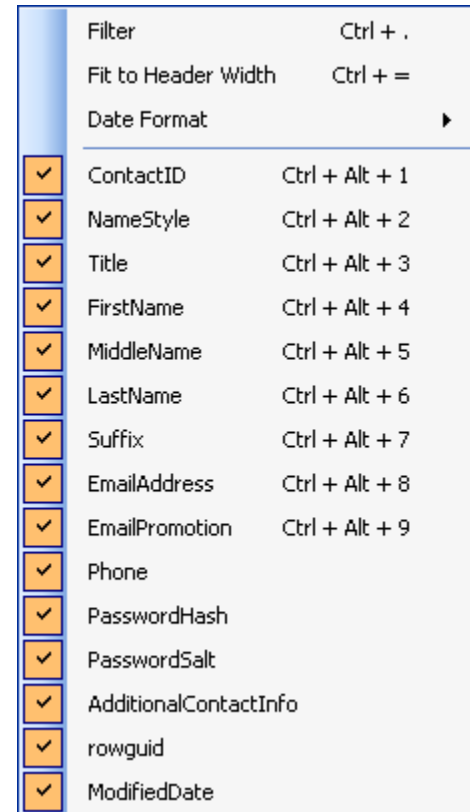


Figure 4-21 Column Shortcuts
Fields beyond 9 are accessible only through the context menu.

4.5.1.10 Quick Find

Shortcut: **Control**+**/**

You can move around the results grid from the keyboard by typing a prefix of a cell value from the primary sort column. Invoke **Quick Find** then start typing a prefix; as you enter each character the display immediately jumps to the first cell matching that value. See 3.4.2.

4.5.1.11 Copy Field Names

Shortcut: **Control**+**;**

Select one or more cells in the grid then invoke this command to copy the field names that the selection touches. This loads the names in a comma-separated string onto the clipboard, ready for pasting into a query for example. Note that there is also a mechanism to generate a complete query from the result grid — see section 4.3.2.4.

5 Appendices

5.1 Quick Start

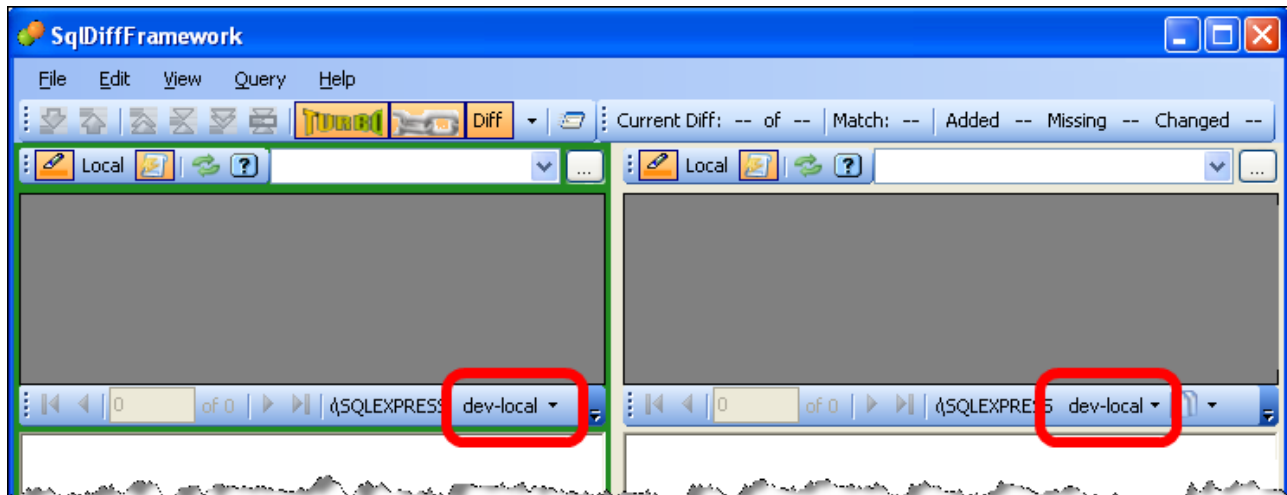
5.1.1 Setup

- Define one or more connections in the **Connection Editor** (**Ctrl** + **F3**) to specify the data source(s) you wish to use (see 3.9); alternatively, import a pre-defined set of connections that your company has standardized on.
- If a data source is an ODBC data source, use Window's **ODBC Data Source Administrator** (see 3.7 and 5.5) first; entries there will populate the Data Source dropdown here in the **Connection Editor**.



5.1.2 Main Window

- Select your target connections with the **Connection Selector** dropdown (main window).



- For maximum effect from minimum effort, enable **tandem** mode (3.8), **auto-diff** mode (4.1.1.9), **auto-execute** mode (4.3.1.3), and **auto-highlight** mode (4.3.1.1).
- If you will be mixing database types (e.g. SQL Server and Oracle), strongly consider enabling **turbo-sort** mode (but note that every query run under turbo-sort mode will need a little extra diligence to always provide an `ORDER BY` clause and all fields will need to be properly aliased—see 3.5.5).
- If you want to retrieve data snapshots, enable **local** mode (4.3.1.2); otherwise leave it off.
- Write (or load) a query in each editor pane, then execute (**F5**).



5.2 Limitations and Work To Be Done

5.2.1 False Positives from One Engine

The Tauberer difference engine is prone to false positives on complex result sets (see 4.1.1.9). SqlDiffFramework still displays the false positive results but when you navigate among them they will automatically be skipped over. Using a different engine, of course, is another workaround.

5.2.2 Multi-Threaded Implementation

SqlDiffFramework does a lot of things well, but I have not made it properly multi-threaded. Once you start a long-running operation it locks up the user interface with all the horrid consequences of that act, though it does display a progress meter. The only (meager) workaround at present is that before starting a major operation open up a new workspace (from the file menu). Each workspace is a separate process so you may continue to work in the new instance without interference.

5.2.3 Not Instrumented for Internationalization

There are as yet no provisions for non-English users to adapt to local languages.

5.2.4 Binary Field Types not Supported

This is a limitation of the .NET framework: it assumes that binary columns are always images, so non-image data appears as the canonical *broken link*:

IdStuff	ProblemColumn	name
7		abc

Microsoft affirms that this behavior is by design! See this defect report on Microsoft Connect:

<http://connect.microsoft.com/VisualStudio/feedback/ViewFeedback.aspx?FeedbackID=93639>

My locally filed defect report is available here:

https://sourceforge.net/tracker/?func=detail&aid=2912137&group_id=101363&atid=629536

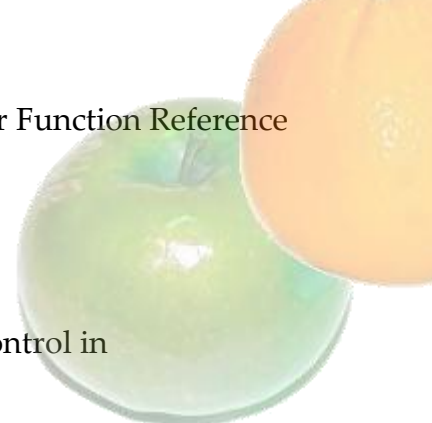
5.2.5 Data Must Fit in Memory

SqlDiffFramework operates with everything in memory. It can handle sizable data sets but you will probably need to close most of your other applications when doing so. Also, be sure to install for the **large address space** (see 2.1.2) on WindowsXP, allowing you to use 3GB instead of the standard 2GB for a single application. To eke out just a bit more, switch

from 32-bit Windows to a 64-bit version. On a 32-bit Windows, 1GB of the total 4GB potential is always used for the kernel. On a 64-bit OS with more memory (e.g. 8GB), you get a full 4GB space devoted to a single application.

5.2.6 More Settings Should be Persistent

Many of the user settings you make, explicitly or implicitly or persisted across invocations. But many others are not and should be.



5.3 Master Function Reference

This table lists every operation, keystroke, menu item, and on-screen control in SqlDiffFramework, organized by groups of related functions.

A mark in the **Tandem** column indicates that whenever you perform that action in one of the two editor panes, the same action will automatically occur in its partner when Tandem mode is enabled. Any such tandem-aware operation may be overridden by depressing **Control** before you perform the action to temporarily make the action apply to a single editor pane.


The notation **CM** in the **Menus** column is an abbreviation for **Context Menu**.

Items beginning with **>** are operations performed in a sub-context (i.e. inside the Meta-Query Selector or inside the Connection Editor).


Item	Tan-dem	Keys	Menus	On screen
Difference Navigation				
Go to first difference		Alt + Home	View → First difference	
Go to last difference		Alt + End	View → Last difference	
Go to previous difference		Alt + ↑	View → Previous difference	
Go to next difference		Alt + ↓	View → Next difference	
Go to current difference		Alt + Enter	View → Current difference	
Set current difference to cursor		Alt + .	View → Set current difference	
Modify next/previous movements to go specifically to added rows.				
Modify next/previous movements to go specifically to missing rows.				
Modify next/previous movements to go specifically to changed rows.				
View total number of lines present on the left that are absent from the right				

Item	Tan-dem	Keys	Menus	On screen
View total number of lines present on the right that are absent from the left				Missing 12
View total number of lines present on both sides that differ from each other				Changed 279
View total number of difference chunks				Current Diff: 1 of 2
View current difference chunk number				Current Diff: 1 of 2
Display number of lines in current difference chunk				[Hover] + Current Diff: 1 of 2
View overall match quality as a percentage				Match: 65%
Display total number of different lines as compared to total number of lines				[Hover] + Match: 65%
Result Set Navigation				
Go to first record				⏪
Go to previous record				⏴
Go to next record				⏵
Go to last record				⏩
Set current record to cursor				Click a cell or row
View current record number				14 of 128
View total number of records				14 of 128
Set current record to any record number				Type row number: 14 of 128
Load & Save Files				
Open query file from file dialog	•	Alt + O	File → Open query	
Open query file from history	•			
Save current file		Ctrl + F8 / Ctrl + F9	File → Save Left/Right Query	
Save file to new name or directory				Alt +
Display last used SQL directory				[Hover] +
Save result set to new name				

Item	Tan-dem	Keys	Menus	On screen
New, unnamed file with empty query	•		File → New query	
New, unnamed file keeping current query	•			→ retain buffer
New, unnamed file with query derived from result set	•			→ generate new query
Restore query to last saved version				
Mirror query from current pane to partner			Query → Mirror query...	
Load & Execute Results				
Refresh result set	•		Query → Execute Query	
Refresh local result set from a different CSV directory	•			
Display last used CSV directory				[Hover] +
Display last query execution duration (resolved to seconds)				00:03
Display last execution time, duration (to milliseconds), and file name				[Hover] + 00:03
Operating Modes				
Toggle linked editor panes	—			
Toggle auto-execute	•			
Toggle turbo-sort	—			
Toggle auto-differencing or change difference engine	—			
Toggle local/live mode	•			
Toggle auto-highlight	•		CM → Highlighting → Auto-highlight	
Toggle case-sensitive auto-highlighting	•		CM → Highlighting → Case sensitive	
Toggle highlighting when auto-highlighting is disabled	•		CM → Highlighting → Hide highlight when disabled	
Set keywords forced to upper/lower case	•		CM → Highlighting → Keywords → default/upper/lower	

Item	Tan-dem	Keys	Menus	On screen
Set variables forced to upper/lower case	•		CM → Highlighting → Variables → default/upper/lower	
Toggle auto-completion	•		CM → Keyword completion → Enable keyword completion	
Set auto-completion forced to upper/lower/user case	•		CM → Keyword completion → default / upper / lower / user case	
View				
Maximize left pane		Ctrl + 1	View → Expand left pane	Drag splitter bar right
Maximize right pane		Ctrl + 2	View → Expand right pane	Drag splitter bar left
Equalize both panes		Ctrl + 3	View → Show both panes	Drag splitter bar to middle
Maximize window across multiple horizontal monitors				Control + 
Working in Query Editor				
Comment / Uncomment selection		Ctrl + Shift + C / Ctrl + Shift + U		
Shift select left / right		Alt + < / Alt + >		
Increase / decrease font size (with auto-highlighting off)		Ctrl + > / Ctrl + <		
Find		Ctrl + F	Edit → Find...	
Find again		F3		
Replace		Ctrl + H	Edit → Replace...	
Undo / Redo		Ctrl + Z / Ctrl + Y		
Complete current word or macro		Ctrl + Space		
Select next place holder in macro		F4		
Reapply highlighting		Ctrl + Shift + H		

Item	Tan-dem	Keys	Menus	On screen
Display column data type		Ctrl+Alt+hover or Ctrl+Shift+hover		
Scroll vertically	•			Vertical scroll bar
Scroll horizontally	•			Horizontal scroll bar
Connections				
Select a connection				local-master ▾
Open connections editor		Ctrl + F3	Query → Edit connections...	Alt + local-master ▾
Clear non-persistent usernames/passwords				Shift + local-master ▾
Show username in tooltip				[Hover] + .SQLEXPRESS
Insert connection string in editor as a comment				[Click] + .SQLEXPRESS
> Test a connection				
> Insert new connection		Type new name		
> Delete current connection				
> Load connection set from file				Load
> Load connection set with passwords				Ctrl + Load
> Save connection set to file				Save
> Save connection set with passwords				Ctrl + Save
Meta-Queries				
Open meta-query dialog		Ctrl + F2	Query → Meta-queries	
> Navigate up/down in query tree		↑ / ↓		
> Expand/collapse category folders		→ / ←		
> Expand/collapse all query folders				Expand All / Collapse All
> Execute meta-query silently		Enter		Execute
> Execute meta-query, inserting into editor, suppressing unused inputs		Shift + Enter		Shift + Execute

Item	Tan-dem	Keys	Menus	On screen
> Execute meta-query, inserting into editor, revealing all inputs		Alt + Shift + Enter		Alt + Shift + Execute
Batch Execution				
Open batch execution palette			Query → Execute batch...	
> Toggle save snapshots		Alt + S		<input type="checkbox"/> Save Snapshots
> Toggle check for good vs. bad data		Alt + Z		<input type="checkbox"/> Flag Non-zero
> Toggle including orphans		Alt + O		<input type="checkbox"/> Include Orphans
> Begin execution		Alt + E		Execute
> Cancel execution after current query				Cancel
> Close batch execution palette				Close or Esc
> Close batch execution palette and results form				Shift + Close
> Open results form		Alt + R		Show Results
Miscellaneous				
New workspace		Ctrl + N	File → New workspace	
Toggle memory used between % and MB				52MB / 2%
Display progress monitor			View → Show progress monitor	
Restore last saved settings			Edit → Restore settings... → Last Saved	
Restore factory settings			Edit → Restore settings... → Factory Settings	
Exit		Alt + F4	File → Exit	



5.4 On-Screen Reference Sheets

These are available under the **Help** menu.

5.4.1 Main Application Reference

Shortcut Key Reference ✖

SqlDiffFramework Quick Reference

DIFFERENCE NAVIGATION	
First difference	Alt Home or
Last difference	Alt End or
Previous difference	Alt UpArrow or
Next difference	Alt DownArrow or
Current difference	Alt Enter or
Set current difference to cursor	Alt Period or
Limit traversal to added, missing, and/or changed rows	Added or Missing or Changed
QUERY LOAD/SAVE	
Open query	Alt O
New query	Alt W
Save LEFT query	Ctrl F8
Save RIGHT query	Ctrl F9
VIEW	
Expand LEFT pane to full width	Ctrl 1
Expand RIGHT pane to full width	Ctrl 2
Display both panes equally	Ctrl 3
MODES	
Turbo-sort	?
Tandem	?
Diff	?
Show Progress Monitor	?
MISCELLANEOUS	
New workspace	Ctrl N
Maximize across multiple monitors	Ctrl Maximize
Edit connections	Ctrl F3
Open meta-query selector	Ctrl F2
View absolute or percentage memory used	Click memory indicator ?

5.4.2 Editor Pane Reference

Shortcut Key Reference

DB Pane Quick Reference

Operations that work in tandem with a partner are marked with (•) and may be overridden with the Control key

DB CONNECTIONS	
Select connection (add Shift to erase transitory login parameters)	button
Edit connections (or import/export them)	Alt button
Display current connection details	Click server name label ?
Display user for current connection	Hover over server name label
QUERY	
Load file •	file drop down or ...
New file •	?
New file but retain buffer •	>> retain ?
New file and auto-generate query •	>> generate ?
Save query (add Alt to save to new file)	
Open meta-query dialog (add Shift to reload library first)	?
EXECUTION and RESULTS	
Execute Query (live) or load CSV snapshot (local) •	F5 or ?
Load CSV snapshot from different directory (local)	Alt ?
Reveal turbo-sort criteria	Alt ?
Save data to CSV file	
SET MODES	
Toggle local mode vs. active connection •	Local ?
Toggle auto-execute upon loading query •	?
SEARCH and FILTER	
Search query / search again	Ctrl F / F3
Search-and-replace in query	Ctrl H
Search result set	Ctrl / ?
Filter result set •	Ctrl * ?
GENERAL	
Move to next/previous control	Tab / Shift Tab ?
OUTPUT GRID OPERATIONS THAT SYNCHRONIZE WITH A PARTNER	
Horizontal/vertical grid scrolling •	scroll keys
Adjust grid row height larger / smaller•	Alt ↑ / Alt ↓
Adjust column widths •	Alt = ?
INPUT EDITOR OPERATIONS THAT SYNCHRONIZE WITH A PARTNER	
Auto-Highlight as you type •	or context menu
Case-sensitive syntax highlighting •	context menu
Keywords / variables forced to upper/lower case •	context menu
Auto-completion enabled •	context menu
Keywords forced to upper/lower/user case on auto-completion •	context menu
Expand tab to spaces / set number of spaces •	context menu
Swap Tab and Ctrl+Tab for editor (and output grid) •	context menu ?

5.4.3 Query Editor Reference

Shortcut Key Reference	
Input Quick Reference	
BLOCK OPERATIONS	
Comment/uncomment selection	Ctrl Shft C / Ctrl Shft U
Shift selection left / right	Alt < / Alt > ?
HIGHLIGHTING	
Reapply highlighting	Ctrl Shft H ?
Convert to upper/lower case	context menu >> highlighting >> keywords or variables
Case-sensitive matching	context menu >> highlighting >> case sensitive
KEYWORD COMPLETION	
Complete current word or macro	Ctrl Space
Convert to upper/lower/matching case	context menu >> completion
EDITING	
Cut	Ctrl X
Copy	Ctrl C
Paste	Ctrl V
Undo / Redo	Ctrl Z / Ctrl Y
Toggle insert / overwrite	Insert
Delete next/previous character	Delete / Bksp
Delete next/previous word	Ctrl Delete / Ctrl Bksp
Delete range by line boundaries	Ctrl Shft Delete
SEARCH and REPLACE	
Open search dialog	Ctrl F
Open replace dialog	Ctrl H
Search again	F3 ?
FORMATTING (when auto-highlighting is OFF)	
Increase / decrease font size	Ctrl > / Ctrl <
Toggle bullets	Ctrl Shift L
Align left / right / center	Ctrl L / Ctrl R / Ctrl E
TAB CONTROL	
Insert TAB character or SPACES	Tab ?
Select TAB or SPACES mode for tab key	context menu >> tab control >> expand tab
Set number of spaces for SPACES mode	context menu >> tab control >> n spaces
Toggle whether control accepts TAB or Ctrl+TAB	context menu >> tab control >> tab inserts characters
SELECTION	
Select all	Ctrl A
Select range	Shift navigation key
Select next placeholder in macro expansion	F4 ?
Display language context details	Ctrl F10
NAVIGATION	
Start / end of line	Home / End
Start / end of text	Ctrl Home / Ctrl End
Start / end of window	Ctrl PageUp / Ctrl PageDown
Left / right one word	Ctrl LeftArrow / Ctrl RightArrow
Up / Down one line	UpArrow / DownArrow
Up / Down one window	PageUp / PageDown
Move to next/previous control	Ctrl Tab / Shift Ctrl Tab ?
Close	

5.4.4 Results Grid Reference

Shortcut Key Reference

Output Quick Reference

Some commands are available only via the context menu, opened by right-clicking the grid header

SEARCH AND FILTER	
Quick-Find	Ctrl / ?
Quick-Filter	Ctrl ' ?
COLUMN OPERATIONS	
Mask columns by keystroke	Ctrl Alt 1 to Ctrl Alt 9 ?
Sort columns by keystroke	Ctrl Shift 1 to Ctrl Shift 9 ?
Display column datatype	Ctrl Shift or Ctrl Alt ?
Adjust column widths	Alt = ?
Change date formatting	context menu ?
Copy field names	Ctrl ; ?
ROW OPERATIONS	
Increase row height	Alt]
Decrease row height	Alt [
MISCELLANEOUS	
Export result set	context menu ?
SELECTION	
Select all	Ctrl A
Select range	Shift <i>navigation key</i>
Select non-contiguous range	Ctrl <i>navigation key</i>
Copy selected cells	Ctrl C
NAVIGATION-CELLS	
Move to next cell forward/backward	Tab / Shift Tab ?
Move to first/last cell in row	Home / End
Move to first/last cell in grid	Ctrl Home / Ctrl End
Move up/down one window height	PageDown / PageUp
Move to next/previous cell in row	RightArrow / LeftArrow ?
Move to next/previous cell in column	DownArrow / UpArrow
Move to next/previous control	Ctrl Tab / Ctrl Shift Tab ?
NAVIGATION-SCROLL BARS	
Up/Down a page	Click scroll track
Move to percentage location	Shift Click scroll track
Up/Down a row	Click scroll arrow

Close

5.5 Preparing an ODBC Connection

Open the **ODBC Data Source Administrator**. On Vista, this is called **Data Sources (ODBC)** and located under the **Administrative Tools** submenu of the **Start** menu.

Figure 5-1 walks through configuring and testing an existing connection called `testDB` (1). If you need to create a new one, select **Add** instead of **Configure** at (2) when you are on the **User DSN** tab. When creating a new data source, you will have one extra step before getting to (3) in the figure: you will be prompted to select the driver. You will find entries for a variety of data sources, including Excel files (**Microsoft Excel Driver (*.xls)**), text files (**Microsoft Text Driver (*.txt; *.csv)**), and databases (**SQL Server**). Select an appropriate one and click the misleadingly-named **Finish**.

You arrive at (3) then with a defined data source. Here you specify the name and other details. For an Excel data source you may select a specific workbook. For a SQL Server database, you select the database server. Point (4) in the figure is the next dialog if you chose SQL Server as the driver type. Here you specify Windows or SQL Server authentication. In (5) you may specify the default database for this connection, and finally in (6) you may test the connection.

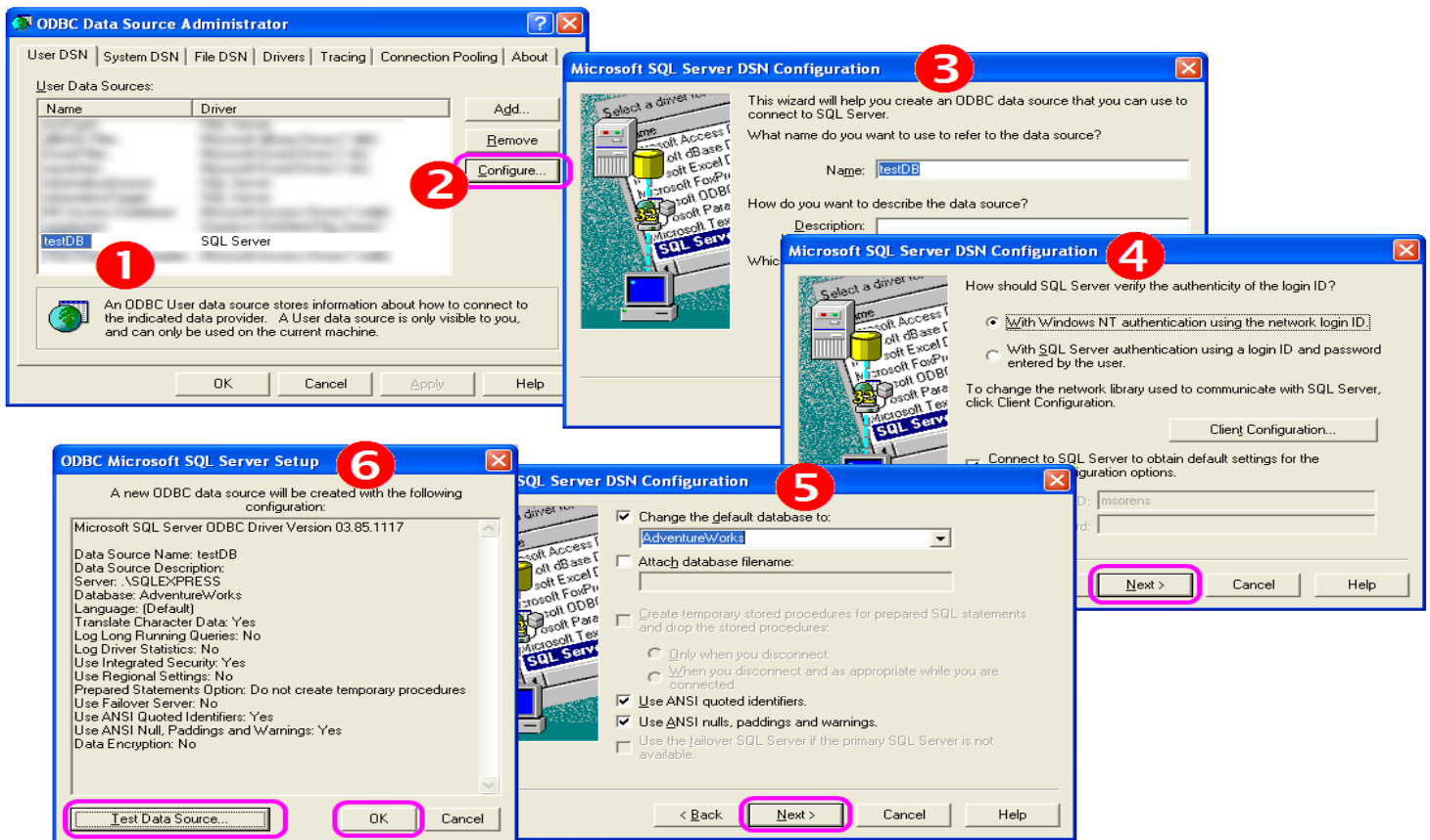
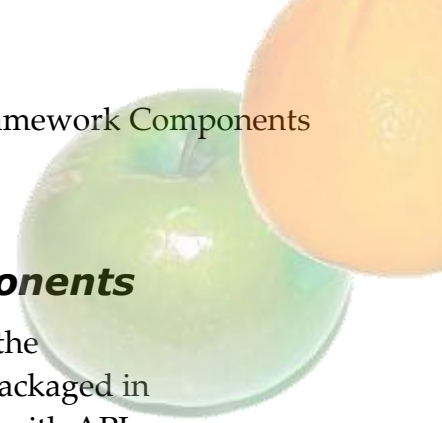


Figure 5-1 Defining an ODBC Connection



5.6 Programming with SqlDiffFramework Components

If you are a developer and are interested in leveraging components of the SqlDiffFramework application, most bits of functionality are already packaged in standalone libraries on my open source web site CleanCode, complete with API documentation.

5.6.1 Key URLs

Item	URL
Home page	http://cleancode.sourceforge.net/
Software highlights	http://cleancode.sourceforge.net/wwwdoc/software.html
Download page	http://cleancode.sourceforge.net/wwwdoc/download.html
Design notes	http://cleancode.sourceforge.net/wwwdoc/colophon.html
.NET API	http://cleancode.sourceforge.net/api/csharp/index.html

Table 5-1 Key URLs for SqlDiffFramework Reference

5.6.2 Libraries

The libraries listed here contain substantial portions of the SqlDiffFramework functionality. The first few (shaded in the table) are my own, fully custom libraries. The remaining ones are work that I have leveraged. Details of these are on the main download page on my web site (<http://cleancode.sourceforge.net/wwwdoc/download.html>).

Library	Description
CleanCode	CleanCode core classes
CleanCode.CsvProcessing	Import/export for CSV data
CleanCode.ChameleonRichTextBoxControls	RichTextBox enhancements for applying SQL contexts, macro support, and more
CleanCode.DatabaseControls	Meta-Query Selector and Connection Editor components
CleanCode.DataGridViewControls	DataGridView enhancements
CleanCode.SqlEditorControls	SqlEditor pane
Diff	Tauberer differencing engine
HertelDifferenceEngine	Hertel differencing engine
PotterDifferenceEngine	Potter differencing engine
LumenWorks.Framework.IO	CSV import/export
DPAPI	Password encryption
LinqBridge	LINQ for .NET 2.0
MySql.Data	MySql connections
OptionsLib	Option form support
VDialog	Advanced dialog box formatting
SyntaxHilightingTextBox	RichTextBox enhancements for syntax highlighting and keyword completion
Searchable Controls	RichTextBox enhancements for search and replace

Table 5-2 Libraries Used in SqlDiffFramework

The shaded ones are custom libraries from CleanCode. The remaining ones are freely available libraries, used either unchanged or with custom enhancements.

Figure 5-2 shows the dependency graph of the utilized libraries. I have omitted the LinqBridge library because it is almost used universally; also, if you use .NET 3.5 rather than .NET 2.0 to compile, it is not required.

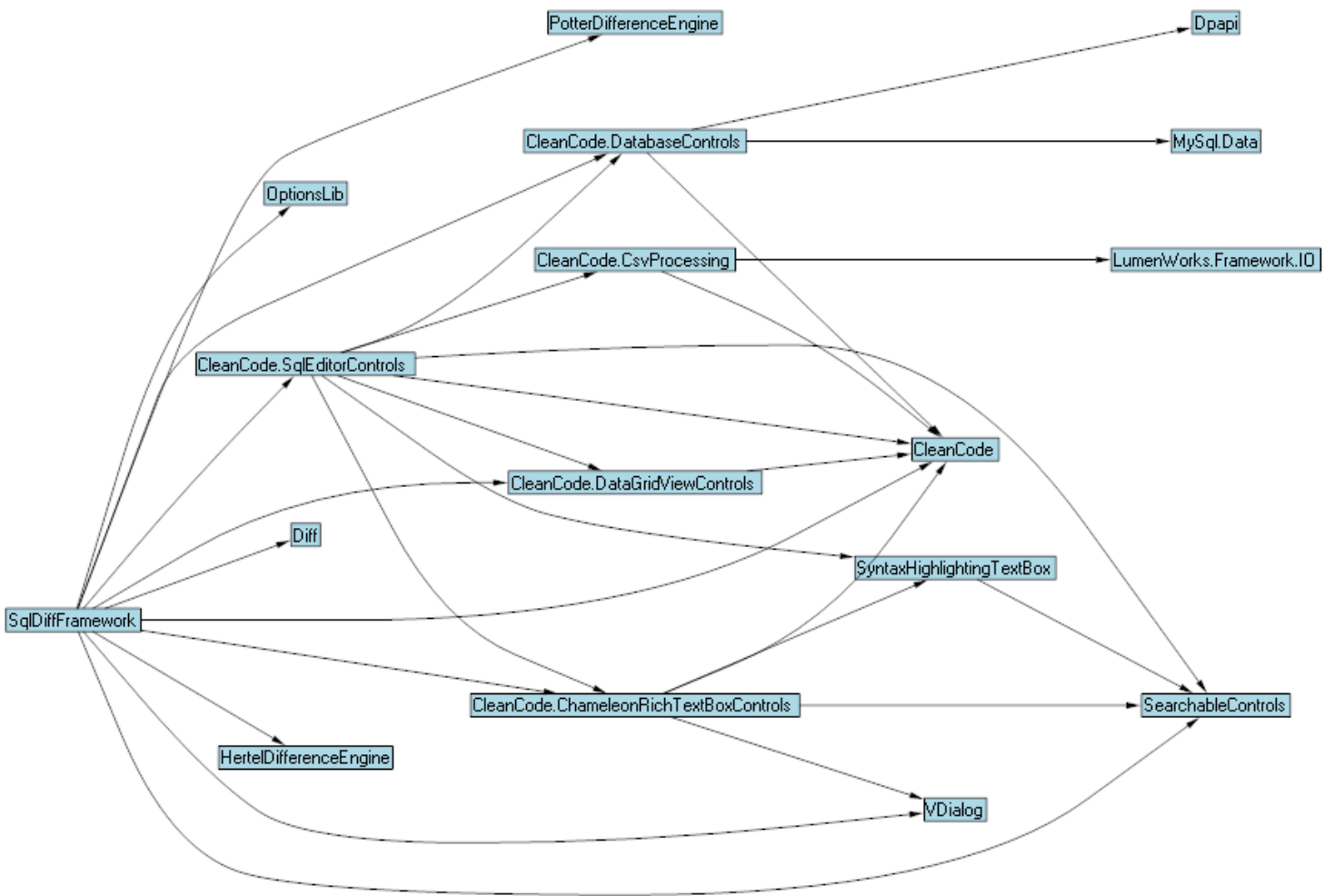


Figure 5-2 Library Dependencies

The dependency graph here was automatically generated using the .NET Refractor tool.

5.6.3 Component Descriptions

The following sections provide a brief description (and illustration where appropriate) of the major components from my open source libraries that are used as building blocks in SqlDiffFramework. Further details on each of these, including in-depth articles on some of them, are available on the main software page on my web site (<http://cleancode.sourceforge.net/wwwdoc/software.html>).

5.6.3.1 SqlEditor

The screenshot shows the SqlEditor control with a table listing database tables and a SQL query editor below it.

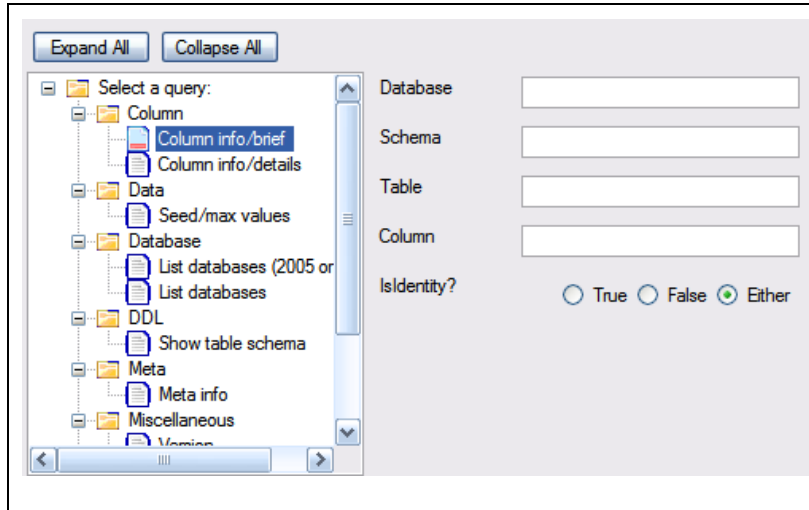
table_catalog	table_schema	Table Name	Rows
AdventureWorks	Person	Address	19614
AdventureWorks	Person	AddressType	6
AdventureWorks	dbo	AWBuildVersion	1
AdventureWorks	Production	BillOfMaterials	2679
AdventureWorks	Person	Contact	19972
AdventureWorks	Sales	ContactCreditCard	19118
AdventureWorks	Person	ContactType	20
AdventureWorks	Person	CountryRegion	238
AdventureWorks	Sales	CountryRegionCu...	109
AdventureWorks	Sales	CreditCard	19118
AdventureWorks	Production	Culture	8
AdventureWorks	Sales	Currency	105
AdventureWorks	Sales	CurrencyRate	13532
AdventureWorks	Sales	Customer	19185
AdventureWorks	Sales	CustomerAddress	19220
AdventureWorks	dbo	DatabaseLog	389
AdventureWorks	HumanResources	Department	16
AdventureWorks	Production	Document	9
AdventureWorks	HumanResources	Employee	290
AdventureWorks	HumanResources	EmployeeAddress	290
AdventureWorks	HumanResources	EmployeeDepart...	296
AdventureWorks	HumanResources	EmployeePayHist...	316
AdventureWorks	Production	Illustration	5
AdventureWorks	Sales	Individual	10404

```
-- Find non-empty tables
SELECT ist.table_catalog, ist.table_schema, convert(varchar(30),object_name
(id)) [Table Name], Rows FROM sysindexes JOIN INFORMATION_SCHEMA.TABLES
ist on ist.table_name = object_name(id) WHERE ist.table_catalog LIKE
'AdventureWorks' AND ist.table_schema LIKE '%' AND object_name(id) not like
'sys%' and indid IN (0,1) and rows > 0 ORDER BY object_name(id)
```

This control is a true SQL-editor-in-a-box. Drag it onto your Windows Form and you have a fully functioning editor that may be configured at runtime to connect to any number of different servers and database types (it supports SQL Server, Oracle, and any ODBC data source). Once you define a set of connections in the connection editor, you may export the list and then import it into another copy of the application running on other machines.

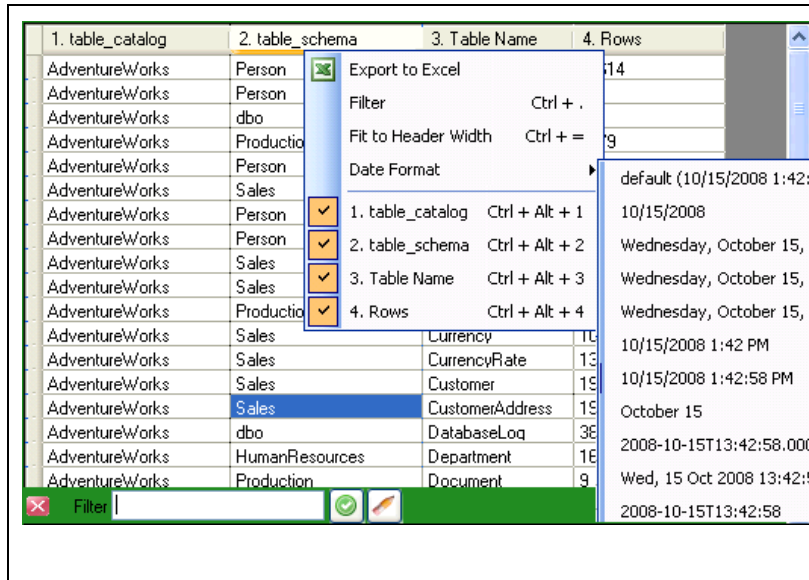
The SqlEditor control has built-in support for synchronizing its actions with a partner. That is, instantiate two copies of this control and they can scroll together, load files together, and change settings together, once you enable tandem operation (just what SqlDiffFramework needs!). The user may override the tandem mode setting (either on or off) for a single action simply by depressing the **Control** key when performing an action.

5.6.3.2 Query Picker



Presents a library of meta-query templates in a tree, then dynamically generates an input form based on the place holders in that template. Once the user supplies values for the inputs, the control builds a finished meta-query from the template. Templates are included for SQL Server, Oracle, and MySql.

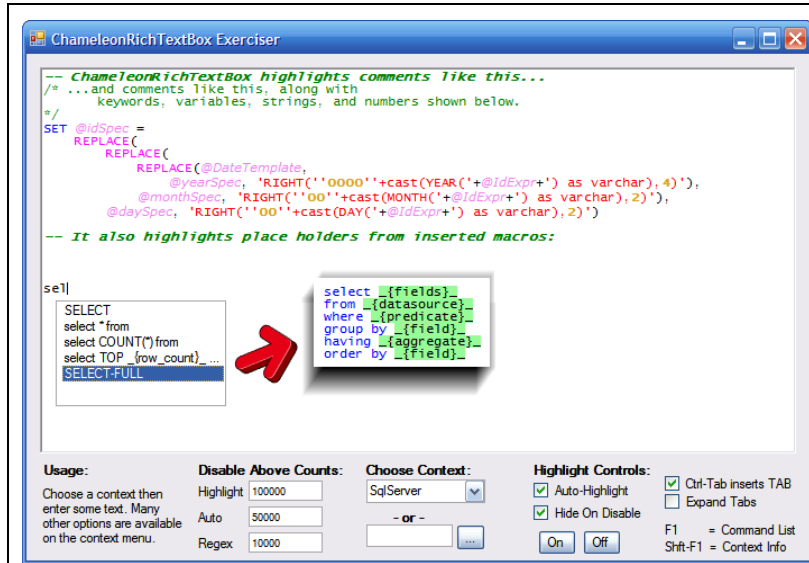
5.6.3.3 ExtendedDataGridView



An ExtendedDataGridView, as the name suggests, extends a standard DataGridView with useful functionality that includes:

- Quick-Find
- Quick-Filter
- Hide/show columns
- Adjust column widths
- Adjust row heights
- Adjust date formatting
- Reveal column data type
- Export result set

5.6.3.4 ChameleonRichTextBox

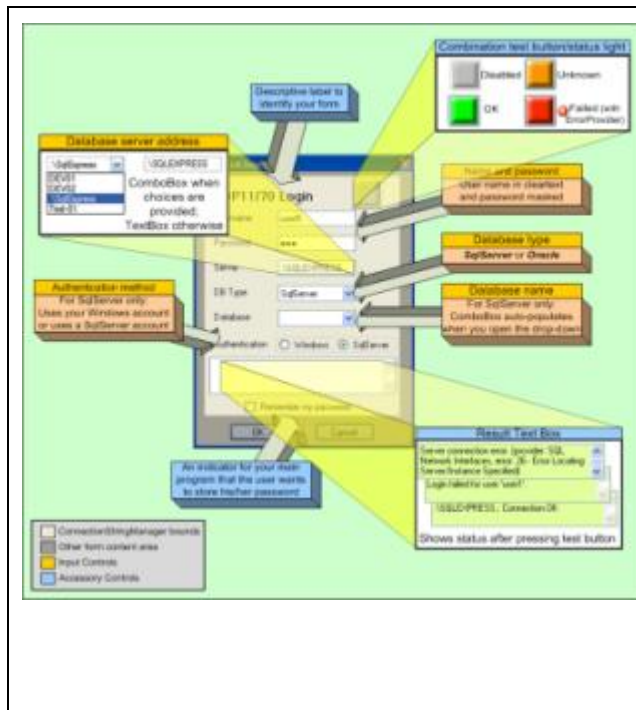


A ChameleonRichTextBox enhances a RichTextBox with language-specific syntax highlighting, keyword completion (a modest Intellisense), search-and-replace capability, macro support, plus some additional keyboard formatting controls.

A complete feature list is available on this comparison chart (URL should be all on one line):

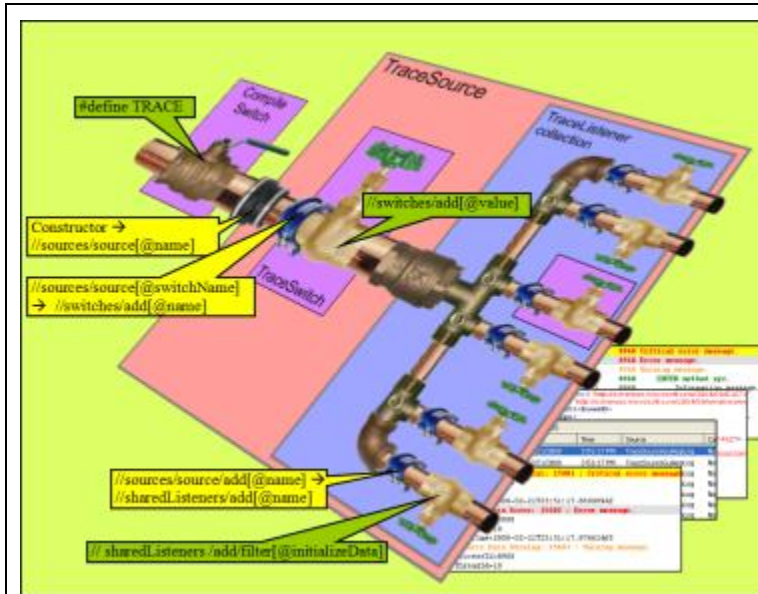
<http://cleancode.sourceforge.net/article/SyntaxHighlightImprovements.html>

5.6.3.5 MultiConnectionStringManager



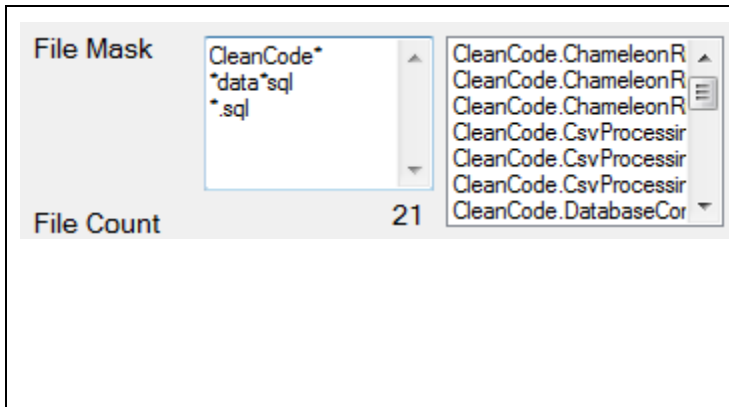
The MultiConnectionStringManager control provides a way to easily manage not just one connection string but a set of connection strings, allowing one-click switching between databases whether they be on the same server, on different servers, or even dissimilar systems (e.g. SQL Server vs. Oracle.) The control supports connections to SQL Server, Oracle, and MySQL, as well as ODBC data sources. It allows you to test the connection immediately, as well as to save or load sets of connections so you can easily migrate a set to other machines.

5.6.3.6 StructuredTraceSource



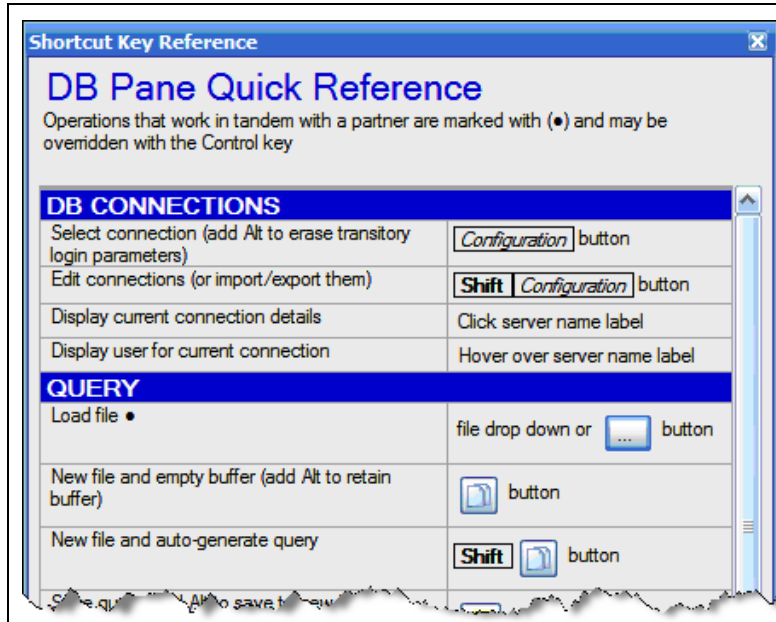
The several diagnostic classes (StructuredTraceSource and its associated AlignedTextWriterTraceListener) integrate into and extend the .NET diagnostic system. These are quite useful in following the execution flow of your instrumented application, providing a flexible mechanism for tracing and debugging.

5.6.3.7 FileMaskControl



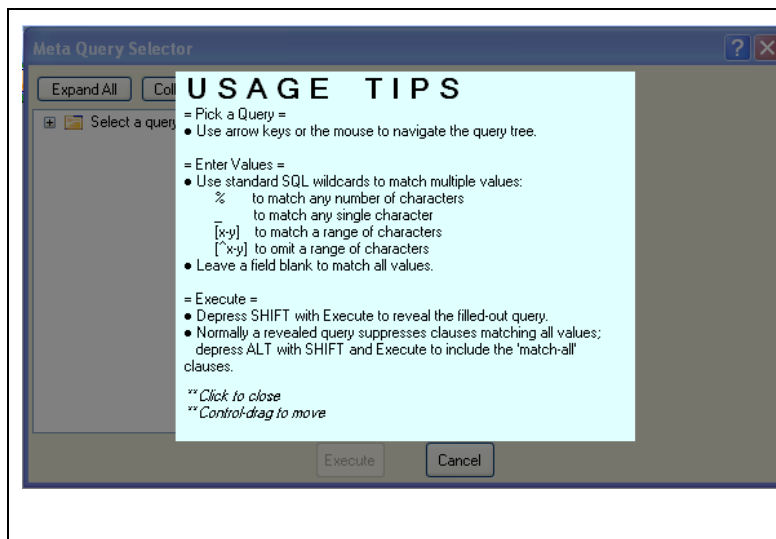
Filters a directory using one or more file masks, displaying both the *list* of matching files and a *count* of matched files. You may customize this generic control with additional arbitrary criteria such as: restricting to files modified after a certain date, ignoring locked files, etc.

5.6.3.8 DisplayCommandsForm



Use this sub-form to display available user commands for a control or application (a quick reference list). Buttons are displayed as images, keys are shown as icons, and tooltips may also be used for further details. You can see several of these in SqlDiffFramework's Help menu. Each is implemented in the user control to which it pertains. Thus, for example, anywhere you embed a ChameleonRichTextBox you can display its quick reference.

5.6.3.9 ShadowTipForm



Use this sub-form to display more of a freeform text passage (compared to the DisplayCommandForm) to guide the user. It partially obscures the underlying form with an opaque island of text floating on a semi-transparent layer covering the parent form. By pressing Control, you can even move the opaque panel out of the way to see the underlying form.

5.6.3.10 ResourceMgr

<p>CleanCode.IO .ResourceMgr</p>	<p>The ResourceMgr class provides a powerful mechanism for your application to purge and refresh external file resources embedded in your binaries. Often an application includes files other than just the executable and associated DLLs, perhaps an XML file, text files, languages files, etc. One approach to file management is to package these files with an installer for your application. A second</p>
--------------------------------------	---

	approach, taken here, is to include these files as resources packaged within your executable or DLL, which are then unpackaged not by an installer but at runtime by the application itself and only if needed.
--	---

5.6.3.11 UsageTracker

CleanCode.IO .UsageTracker	For custom applications used in-house, the UsageTracker class provides a mechanism for recording first use of each version of your application by each user. This provides a convenient history for managing updates and releases.
-------------------------------	--

5.6.3.12 UpdateCheck

CleanCode.IO .UpdateCheck	The UpdateCheck class provides a mechanism for notifying in-house users when new versions of your application become available.
------------------------------	---

5.6.3.13 WindowRestorer

CleanCode.Forms .WindowRestorer	Used in conjunction with application settings this helper class tracks a form's window state and position and enables an application to restore it upon subsequent invocations. It accounts for whether the window is minimized or maximized at the time the application is terminated. Most importantly, it accounts for multiple monitors and changes in monitors. You can also use this class to add a maximize-across-multiple-monitors function and to automatically position sub-forms on the same monitor before displaying them.
------------------------------------	--

5.6.3.14 MemoryGauge

CleanCode.Forms .MemoryGauge	Provides a convenient mechanism for displaying the amount of memory used by the current application on either a ToolStripItem or a Control as an absolute number in megabytes or a relative percentage of memory.
---------------------------------	---

5.6.3.15 MenuBuilder

CleanCode.Forms .MenuBuilder	This class simplifies the process of building—and using—a multi-level context menu in a Windows Form application.
---------------------------------	---

5.6.3.16 ToolStripDropDownManager

CleanCode.Forms. ToolStripDropDownManager	This class enhances a plain status label in a ToolStrip to support history, include a time stamp, include a category, and differentiate errors and warnings.
--	--