



Product Documentation

DB Optimizer™ XE and DB Optimizer™

User Guide

Version 2.5.1

Published August 2010

© 2010 Embarcadero Technologies, Inc. Embarcadero, the Embarcadero Technologies logos, and all other Embarcadero Technologies product or service names are trademarks or registered trademarks of Embarcadero Technologies, Inc. All other trademarks are property of their respective owners.

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for application developers and database professionals so they can design systems right, build them faster and run them better, regardless of their platform or programming language. Ninety of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero products to increase productivity, reduce costs, simplify change management and compliance and accelerate innovation. The company's flagship tools include: Embarcadero® Change Manager™, CodeGear™ RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and Rapid SQL®. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices located around the world. Embarcadero is online at www.embarcadero.com.

August 12, 2010

CORPORATE HEADQUARTERS

100 CALIFORNIA STREET
12TH FLOOR
SAN FRANCISCO, CALIFORNIA
94111 USA

EMEA HEADQUARTERS

YORK HOUSE
18 YORK ROAD
MAIDENHEAD, BERKSHIRE
SL6 1SF, UNITED KINGDOM

ASIA-PACIFIC HEADQUARTERS

L7. 313 LA TROBE STREET
MELBOURNE VIC 3000
AUSTRALIA

Contents

Welcome to Embarcadero DB Optimizer	9
New Features	11
New Features in 2.5.1	11
New Features in DB Optimizer 2.5	11
Configuring DB Optimizer™ XE and DB Optimizer™	13
Initial Setup	13
Specify a Workspace	14
License DB Optimizer™ XE and DB Optimizer™	14
Customizing DB Optimizer™ XE and DB Optimizer™ (Preferences)	14
Set Index Configuration Preferences	15
Set SQL Editor Preferences	18
Set SQL Execution Preferences	18
Set Code Assist Preferences	18
Set Code Formatter Preferences	19
Set Results View Preferences	19
Set Syntax Coloring Preferences	19
Set SQL Code Template Preferences	20
Set File Encoding Preferences	22
Using DB Optimizer	23
Working with Data Sources	23
Register Data Sources	24
Browse a Data Source	27
View Database Object Properties	27
Search for Database Objects	29
Filter Database Objects	30
Define Data Source-Specific Object Filters	31
Define Global Database Object Filters	31
Drop a Database Object	32
Working with SQL Projects	33
Create a New SQL Project	34
Open an Existing Project	34
Search a Project	35
Add Files to a Project	35

Delete a Project	36
Creating and Editing SQL Files (SQL Editor)	37
Create an SQL File	39
Open an Existing SQL File	39
Working in SQL Editor	39
Understanding Automatic Error Detection	41
Understanding Code Assist	43
Understanding Hyperlinks	46
Understanding Code Formatting	46
Understanding Code Folding	51
Understanding Code Quality Checks	51
Understanding SQL Templates	55
View Change History	56
Revert to an Old Version of a File	57
Delete an SQL File	57
Executing SQL Files	58
Associate an SQL File with a Data Source	59
Configure an SQL Session	60
Execute SQL Code	61
View and Save Results	61
Troubleshooting	62
View Log Details	64
Maintain Logs	65
Filter Logs	65
Import and Export Error Logs	68
Find and Fix SQL Code Errors	69
Find and Fix Other Problems	69
Using Profiling	71
Understanding Profiling	71
Understanding the Interface	72
Running a Profiling Session	73
Execute a Profiling Session	74
Work with Session Results	75
Opening an Existing Profiling Session	76
Analyze the Load Chart	76
Analyze the Top Activity Section	78
Top SQL Tab	80

Top Execution Activity Tab (DB2 Specific)	82
Top Events Tab	82
Top Sessions Tab.	83
Top Object I/O Tab (Oracle-Specific)	83
Top Procedures Tab (SQL Server and Sybase Specific)	84
Analyze Profiling Details	84
Viewing Details on the SQL Tab.	87
Viewing Details on the Sessions Tab	90
Viewing Details on the Events Tab.	94
Viewing Details on the Procedures Tab.	96
Saving Profiling Sessions	99
Work with the Profiling Repository	100
Import Statements to Tuning	101
Other Profiling Commands	101
Zooming In and Out	102
Filtering Results.	102
Configuring Profiling	103
Configuring DBMS Properties and Permissions	103
Configuring IBM DB/2 for Windows, Unix, and Linux	103
Configuring Microsoft SQL Server.	105
Configuring Oracle	106
Configuring Sybase.	106
Building Launch Configurations	107
Using Load Tester.	111
Using Tuning.	115
Introduction to Database Tuning	115
Introduction to DB Optimizer's Tuner	116
SQL Tuning Methodology	118
SQL Tuner Overview.	119
What's happening on the databases?	120
Tuning Example	123
The Database is Hanging or the Application has Problems	123
The Database Caused the Problem	125
The Machine Caused the Problem	126
Finding and Tuning Problem SQL.	128
Understanding the Tuner Interface.	128
Understanding the Input Tab	129

Understanding the Overview Tab	131
Understanding the Analysis Tab	133
Tuning SQL Statements	135
Create a New Tuning Job	136
Specify a Data Source	137
Add SQL Statements	138
Run a Tuning Job	140
Analyze Tuning Results.	143
Compare Cases.	145
Filter and Delete Cases	146
Create an Outline	146
Modify Tuning Results	147
Oracle Query Rewrites	149
Using the Analysis Tab	149
Implementing Index Analysis Recommendations	150
Visual SQL Tuning.	151
Changing Diagram Detail Display	152
Viewing Table Counts and Ratios	152
Viewing the VST Diagram in Summary Mode	153
Viewing the VST Diagram in Detail Mode	154
Changing Detail Level for a Specific Table	154
Viewing All Table Fields	155
Expanding Views in the VST Diagram	158
Expanding Subqueries and Nested Subqueries	161
Interpreting the VST Diagram Graphics	161
Viewing the Diagram Legend.	162
Colors	162
Connecting Lines/Joins.	162
One-to-One Join	163
One-to-Many Join	163
Cartesian Join	164
Implied Cartesian Join	165
Many-to-Many Join	166
Indirect Relationship	166
In or Exists Join	167
Not In or Not Exists Join	169
Viewing Object SQL	170
Refreshing the VST Diagram	170

Using Oracle-Specific Features	171
Using the Table Statistics Tab	172
Using the Column Statistics And Histograms Tab	173
Using the Outlines Tab	174
Tuning SQL Statements in the System Global Area (SGA)	175
Additional Tuning Commands	176
View the Source Code of Tuning Candidates	176
View Statement or Case Code in SQL Viewer	177
Open an Explain Plan for a Statement or Case	178
Executing a Session from the Command Line	179
Saving a Tuning Job	180
Configuring Tuning	180
Set Roles and Permissions on Data Sources	181
Set Tuning Job Editor Preferences	182
Set Generated Case Preferences	184
Examples of Transformations and SQL Query Rewrites	185
DBMS Hints	186
Oracle Hints	187
SQL Server Hints	193
DB2 Hints	194
Sybase Hints	195
Tutorials	199
Working with Data Source Explorer	200
Adding Data Sources	201
Browsing Data Sources	203
Profiling a Data Source	204
Starting a Profiling Session	206
Analyzing Session Data	206
Load Chart	207
Top Activity	208
Profiling Details	209
Saving a Profiling Session	212
Importing Statements to SQL Tuner	213
Tuning SQL Statements	214
Creating a New Tuning Job	215
Adding SQL Statements	217
Running a Tuning Job	218

Analyzing Tuner Results on the Overview Tab	220
Finding Missing Indexes and SQL Problems	222
Finding Missing Indexes	223
Applying Tuner Results to the Data Source	223
Implementing Recommendations on the Overview Tab	224
Implementing Recommendations on the Index Analysis Tab	224
SQL Code Assist and Execution	226
Code Extraction	227
Code Highlighting	228
Automatic Error Detection	229
Code Complete	230
Hyperlinks	230
Code Formatting	231
Code Folding	232
Code Quality Checks	232
SQL Execution	232
Configuring SQL Execution Parameters	234
Reference	245
Database Objects	245
DBMS Connection Parameters by Platform	254
IBM DB2 LUW	256
Microsoft SQL Server	256
JDBC Connection Parameters	258
Oracle Connection Parameters	258
Sybase Connection Parameters	259

WELCOME TO EMBARCADERO DB OPTIMIZER

Embarcadero DB Optimizer simplifies SQL optimization and development for application developers with many features for improving productivity and reducing errors. A rich SQL IDE with statement tuning, data source profiling, code completion, real-time error checking, code formatting and sophisticated object validation tools helps streamline coding tasks. DB Optimizer's user interface helps improve overall productivity with integrated development, monitoring, and tuning components. DB Optimizer offers native support for IBM® DB2® for LUW, Oracle®, Microsoft® SQL Server and Sybase® as well as JDBC support for other DBMS.

DB Optimizer has four components that when used together can optimize your database performance.

SQL Editor: A developer can write Java in Eclipse that calls to the database with SQL. The SQL that calls to the database can be written in the SQL Editor with type ahead, code assist and quick fixes to show the users syntax and correct mistakes. For more information, see "[Creating and Editing SQL Files \(SQL Editor\)](#)" on page 37.

Load Tester: The SQL code can be run in the Load Tester to test execution by multiple concurrent users. User load testing is so often done by one single user and then problems don't appear until production with multiple concurrent users. Concurrent user testing is a breeze in DB Optimizer. For more information, see "[Using Load Tester](#)" on page 111.

Profiler: You can run the Profiler while the Load Tester is executing to show clearly the impact on the database. The profiler can also be used by QA on load simulation. Finally the Profiler can be run on any production database to clearly show load, bottlenecks, and sources of bottlenecks or resource consumption. For more information, see "[Using Profiling](#)" on page 71.

Tuner: Finally if a problem SQL is found on the system the Tuner will show if it's correctly optimized by the database or not, and if not it will show the best plan and what hints or optimizer directives can be included in the SQL to force the database to use the optimal plan. For Oracle these hints can even be stored in the database so that there is no need to even change the original SQL text. For more information, see "[Using Tuning](#)" on page 115.

NOTE: This guide is supplemented by online information in the Embarcadero Documentation Wiki at <http://docwiki.embarcadero.com/DBOptimizer>.

NEW FEATURES

NEW FEATURES IN 2.5.1

Operating System Support

DB Optimizer platform support has been extended to include Windows XP x64, Windows Vista x64, Windows 7 x32, and Windows 7 x64.

Improvements to Tuning

- Visual SQL Tuning support for sub-queries as expandable nodes. For more information, see [Expanding Subqueries and Nested Subqueries](#) on page 161
- EXISTS, IN, NOT EXISTS, and NOT IN join annotation support in VST. For more information, see [Connecting Lines/Joins](#) on page 162.
- Complete state persistence when saving tuning editor. For more information, see "[Saving a Tuning Job](#)" on page 180.

NEW FEATURES IN DB OPTIMIZER 2.5

Improvements to Profiling

- New Procedures tab for SQL Server and Sybase platforms enables you to examine procedures and their effect on database efficiency. For more information, see "[Top Procedures Tab \(SQL Server and Sybase Specific\)](#)" on page 84.
- Top SQL tab now provides average execution time, Average Elapsed (sec) statistics for SQL Server, DB2, and Oracle platforms. For more information, see "[Top SQL Tab](#)" on page 80.
- Top Events tab now provides average wait time, Avg. Per Wait (sec) statistics for Oracle events. For more information, see "[Top Events Tab](#)" on page 82.
- Improved SQL text collection by querying the database cache more often and more intelligently thereby reducing the occurrences of UNKNOWN statements.
- Max CPU and Max Engine count can now be specified for cases when the database platform does not report how many CPUs or engines are available and for when you want to see the performance gain or hit created by increasing or decreasing the number of CPUs or engines. For more information, see "[Analyze the Load Chart](#)" on page 76.

- New Profiling Repository available for Oracle platforms offers the ability to gather and store more profiling information that is then readily available for analysis from the Data Source Explorer. For more information, see "[Building Launch Configurations](#)" on page 107, "[Saving Profiling Sessions](#)" on page 99, and "[Opening an Existing Profiling Session](#)" on page 76.
- Performance optimization enables the Profiler to better support longer sessions and more heavily loaded servers.

Improvements to Tuning

- New transformations or query rewrites available for Oracle data sources offer more tuning alternatives. For more information, see "[Oracle Query Rewrites](#)" on page 149.
- Extended support for statements with bind variables on SQL Server and Sybase. For more information, see "[Add SQL Statements](#)" on page 138.
- Filter ratios, and table size and table join size optionally added to Visual SQL Tuning diagrams. Collecting this data may be time-consuming for large databases. For more information, see [Viewing Table Counts and Ratios](#) on page 152.

CONFIGURING DB OPTIMIZER™ XE AND DB OPTIMIZER™

This section contains information on configuring DB Optimizer. It includes information on setting up the system directory for project files, as well as licensing information. Additionally, this section contains information on setting preferences within the application for the customization of various features and functionality.

- [Initial Setup](#) on page 13
- [Customizing DB Optimizer™ XE and DB Optimizer™ \(Preferences\)](#) on page 14

INITIAL SETUP

The following topics provide general help for configuring DB Optimizer:

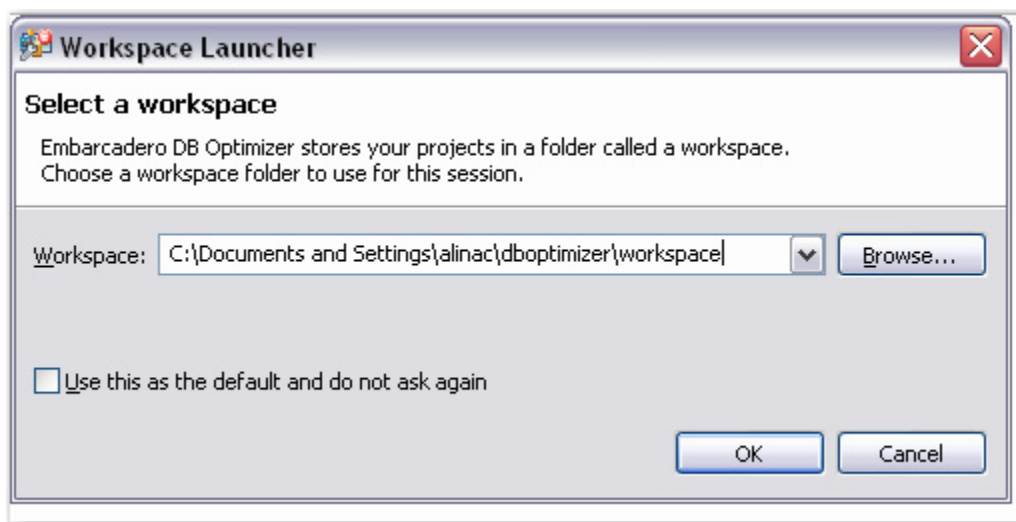
- [Specify a Workspace](#) on page 14
- [License DB Optimizer™ XE and DB Optimizer™](#) on page 14

Additionally, the following preferences are available to help you customize and tune functions within the application:

- [Set Index Configuration Preferences](#) on page 15
- [Set SQL Editor Preferences](#) on page 18
- [Set SQL Execution Preferences](#) on page 18
- [Set Code Assist Preferences](#) on page 18
- [Set Code Formatter Preferences](#) on page 19
- [Set Results View Preferences](#) on page 19
- [Set Syntax Coloring Preferences](#) on page 19
- [Set SQL Code Template Preferences](#) on page 20
- [Set File Encoding Preferences](#) on page 22

SPECIFY A WORKSPACE

When you start Eclipse or the DB Optimizer™ XE and DB Optimizer™ standalone application for the first time, you are prompted to create a workspace.



Click **Use this as the default and do not ask again** to set the specified folder as the permanent default workspace. For more information about workspaces, see **Help > Help Contents > Workbench User Guide**.

LICENSE DB OPTIMIZER™ XE AND DB OPTIMIZER™

The first time you first launch DB Optimizer™ XE and DB Optimizer™, you will be prompted to activate the product. Choose to activate by Internet and follow the prompts. During the activation process you will receive an email with an activation key; after you enter that key into the License Setup dialog, you will receive a free 14-day evaluation license.

If due to firewall or other restrictions you cannot use Internet activation, select the E-mail alternative. If that does not work either, select the Phone alternative.

To continue using DB Optimizer™ XE and DB Optimizer™ after the evaluation period, select **Help > Embarcadero Licensing > License Registration** and follow the prompts, or visit the Embarcadero online store at <http://www.embarcadero.com/store.html>.

CUSTOMIZING DB OPTIMIZER™ XE AND DB OPTIMIZER™ (PREFERENCES)

To customize various aspects of DB Optimizer™ XE and DB Optimizer™, select **Window > Preferences > SQL Development**. For information on categories that may not be covered in this section, see **Help > Help Contents > Workbench User Guide** or **Help > Help Contents > Debugger**, respectively.

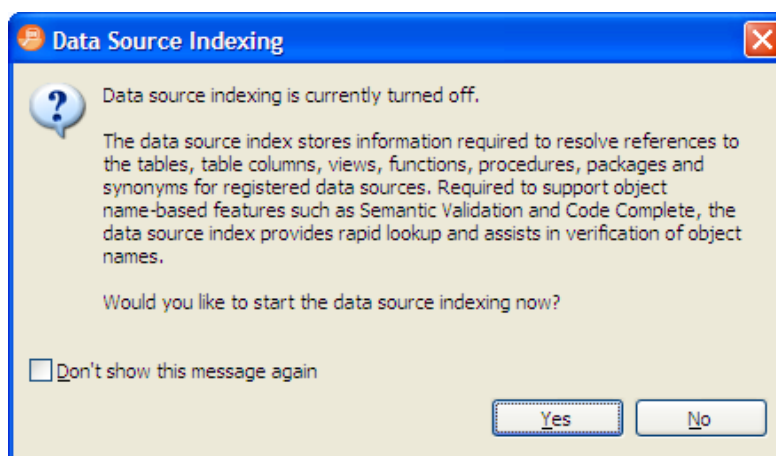
SET INDEX CONFIGURATION PREFERENCES

The Data Source Index is a local repository that stores the schema of registered data sources in DB Optimizer™ XE and DB Optimizer™. It is automatically set to index information about data sources registered in the development environment.

By default, the Data Source Index captures all catalogs, functions, procedures, tables, and views. Additionally, after the initial index, the index performs incremental captures of information.

However, there is a definitive trade-off when indexing a full database schema. The time it takes to fully capture a large schema and logical space considerations on local workstations, often makes it inefficient for DB Optimizer™ XE and DB Optimizer™ to perform this task each time a new data source is registered in DB Optimizer™ XE and DB Optimizer™. Thus, the Index can be configured via the DB Optimizer™ XE and DB Optimizer™ Preferences dialog to accommodate machine processing ability and speed.

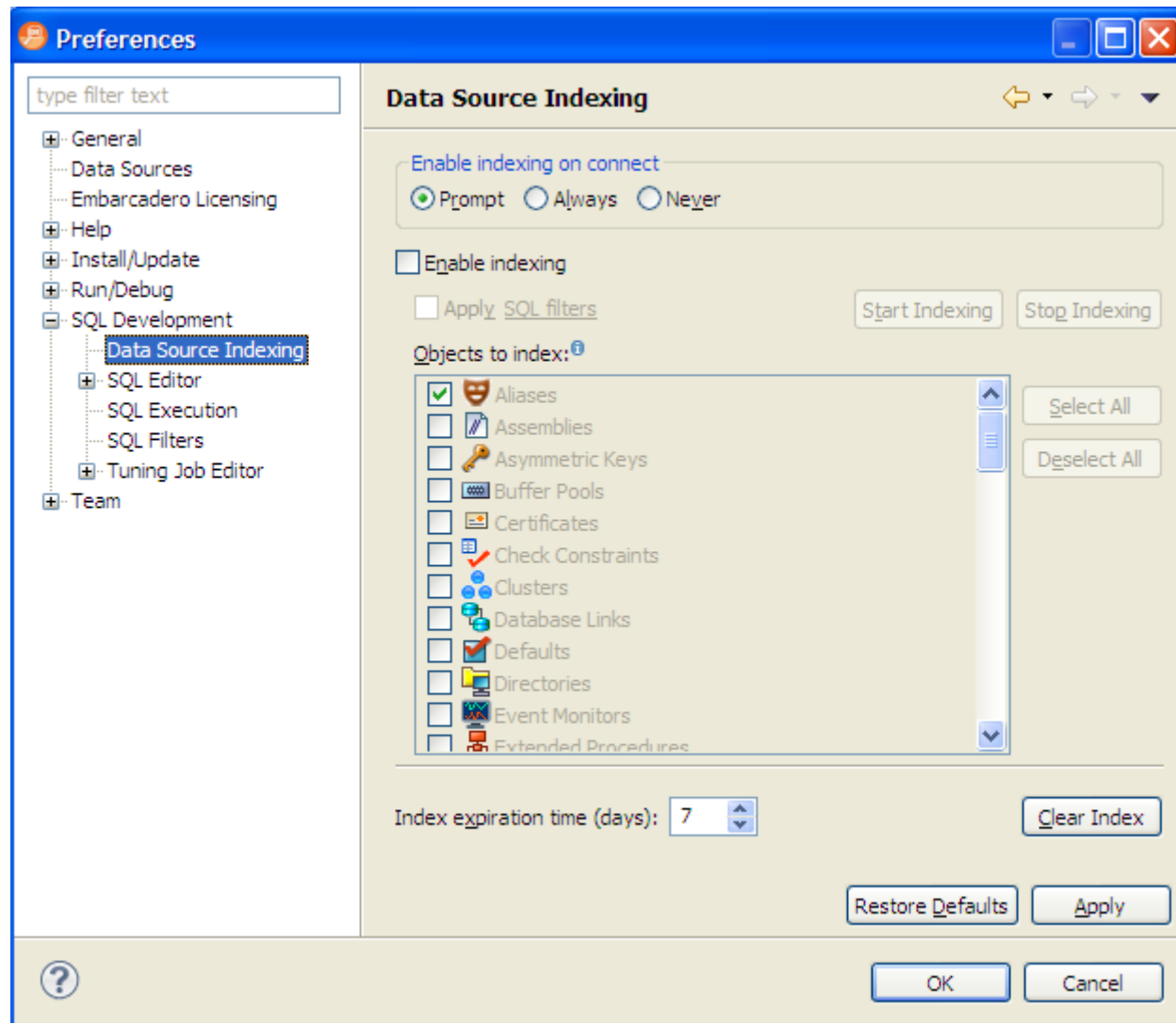
By default, when DB Optimizer™ XE and DB Optimizer™ connect to a data source, you are prompted to enable indexing. If the database is big indexing can put a lot of load on the database.



Index Configuration parameters enable you to indicate how schema caching behaves by specifying at what level data source objects will be indexed, the specific catalogs, schemas, and data source objects to index, and other factors that speed up the indexing process at a cost of slower retrieval for those objects not indexed by the process.

Additionally, over the course of a DB Optimizer™ XE and DB Optimizer™ session, index information is periodically updated by DB Optimizer™ XE and DB Optimizer™. The index refresh process uses the same specified parameters as the initial indexing process and therefore can cause application slowdown and performance issues if the index behavior has not been configured in an efficient manner.

DB Optimizer also provides the ability to index at the individual data source level. In Data Source Explorer, right-click on a data source and select **Properties**, then on the **Properties** dialog, choose **Data Source Indexing** to access index options for the specified data source.



To configure the global index:

1. Select **Window > Preferences > SQL Development > Data Source Indexing**. Change the settings as appropriate:
 - The tree view displays a list of database objects as they are organized in the Database Explorer view. Use the check boxes beside each object to specify the data sources that are to be included in the indexing process.
 - Select the **Apply SQL Filters** tab if you want to apply any pre-defined filters to the index.
 - If you are having performance problems due to a caching issue (such as a configuration error), the **Stop Indexing**, **Clear Index**, and **Start Indexing** buttons enable you to stop, clear, and/or restart the index process, respectively.

- The **Objects to include in index pane** contains a list of data source objects. Select or clear the check boxes beside each data source object to indicate the specific data source objects that are included and excluded, respectively, from the indexing process.
 - The **Index expiration time (days)** setting indicates that an index job will not start automatically until the specified number of hours have passed. The index can also be started manually via Start Indexing.
- 2 When you are finished configuring the Index, click **Apply** to save your changes.

To configure individual data source indexing:

- 1 In Data Source Explorer, right-click on the data source you want to specify indexing, and select **Properties**. The **Properties** dialog appears.
- 2 Choose **Data Source Indexing** and modify the properties, as required:
 - Choose **Enable Data Source Specific Settings** to indicate to DB Optimizer that you want to specify individual indexing properties for this data source. Data sources that do not have this option selected will index under global indexing parameters.
 - The tree view displays a list of database objects as they are organized in the Database Explorer view. Use the check boxes beside each object to specify the data sources that are to be included in the indexing process.
 - Select **Clear Index** to delete the Object Index each time the application is started.
 - Select the **Apply SQL Filters** tab if you want to apply any pre-defined filters to the index.
 - If you are having performance problems due to a caching issue (such as a configuration error), the **Stop Indexing**, **Clear Index**, and **Start Indexing** buttons enable you to stop, clear, and/or restart the index process, respectively.
 - The **Objects to include in index** pane contains a list of data source objects. Select or clear the check boxes beside each data source object to indicate the specific data source objects that are included and excluded, respectively, from the indexing process.
 - The **Index Expiration Time** (hours) setting indicates that an indexing job will not start automatically until the specified number of hours have passed. The index can also be started manually via **Start Indexing**.

SET SQL EDITOR PREFERENCES

- 1 Select **Window > Preferences > SQL Development > SQL Editor**.
- 2 Change the settings as appropriate in each section and then click **Apply**.
 - **Severity Level for Semantic Validation Problems** determines how semantic code errors are flagged in the editor and the Problems view.
 - The link to specify hyperlinks takes you to the Text Editors preference page.

NOTE: Clearing **Enable SQL Parser** will disable many of the “smart” SQL editor features, including code formatting, auto completion, semantic validation, and hyperlinks. For better performance, you may disable the parser for files above a specified size.

SET SQL EXECUTION PREFERENCES

Select **Window > Preferences > SQL Development > SQL Editor**.

NOTE: If you disable auto-commit for a platform, you must use SQL Editor’s transaction features to execute code on that platform.

SET CODE ASSIST PREFERENCES

The Code Assist panel is used to specify configuration parameters that determine how code completion features in SQL Editor behave.

Select **Window > Preferences > SQL Development > Code Assist**.

- **Enable Auto Activation** enables or disables code assist functionality with the Ctrl + Space command. If this option is selected, the code assist window automatically appears when you stop typing. Specify the amount of time in milliseconds that the window automatically appears in the Auto Activation Delay field beneath the option.
- **Insert Single Proposals Automatically** specifies if only a single code completion suggestion is returned, it is inserted automatically.
- **Fully Qualified Completions Automatically** specifies if code completion results are returned specific (fully qualified), rather than the minimum required to identify the object.
- **Code Assist Color Options** specifies the color formatting of code completion proposals. Select background or foreground options from the menu and modify them as appropriate.

SET CODE FORMATTER PREFERENCES

The Code Formatter pane provides configuration options for code formatting functionality in SQL Editor.

Select **Window > Preferences > SQL Development > Code Formatter**.

The panel provides a drop down list of formatting profiles and a preview window that displays how each profile formats code.

- Click **New** to define additional code formatting profiles.
- Click **Edit** to modify existing profiles. You can modify how code characters appear in the interface and how SQL Editor determines line breaks.
- Click **Rename** to change the name of an existing profile. The new name cannot be the same as another existing profile.

NOTE: If you create a new profile with a name that already exists in the system, a prompt will appear asking you to change the name of the new code formatting template.

SET RESULTS VIEW PREFERENCES

The Results Viewer pane provides configuration options that specify how the Results view displays results.

Select **Window > Preferences > SQL Development > Results Viewer**.

- **Grid Refresh Interval** indicates the speed in milliseconds that the Results view refreshes.
- **Stripe the Rows of the Results Table** adds intermittent highlighted bars in the Results view.
- **Display Results in Separate Tab in SQL Editor** opens the Results view in a separate window on the Workbench.

SET SYNTAX COLORING PREFERENCES

The Syntax Coloring panel provides configuration options that change the look and feel of code syntax in SQL Editor.

Select **Window > Preferences > SQL Development > Syntax Coloring**.

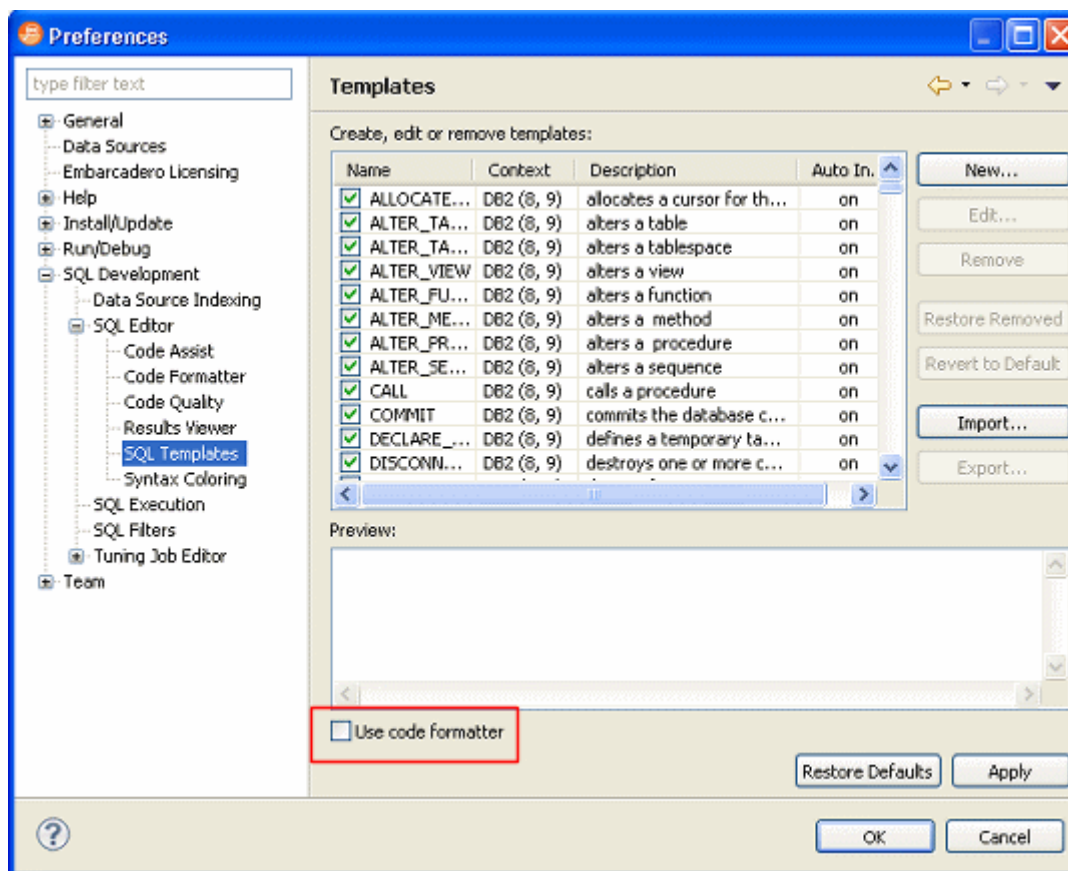
Use the tree view provided in the Element window to select the comment type or code element you want to modify. Select the options to the right-hand side of the window to modify it. The Preview window shows a piece of sample code that updates according to the changes you made.

SET SQL CODE TEMPLATE PREFERENCES

The SQL Templates panel provides customization options for creating and modifying SQL code templates.

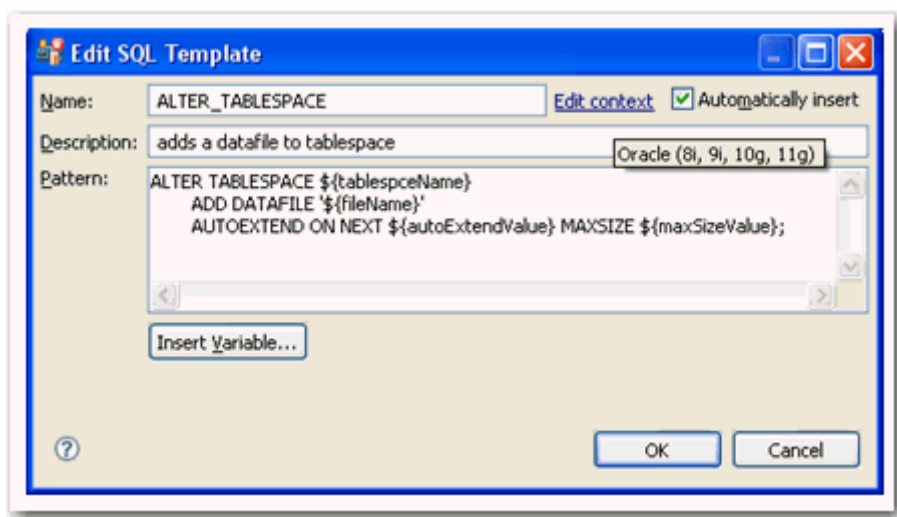
- Select **Window > Preferences > SQL Development > SQL Editor > SQL Templates**.

The SQL Templates panel displays a list of all SQL code templates currently available. Additionally, when you select a template from the list, the Preview section displays the code block as it will appear when the template is selected in SQL Editor.



Click on the check box beside each template to specify if it is included in the code assist check or not, within SQL Editor. Use the buttons on the right-hand side of the panel to create, edit, or delete SQL templates, as needed.

When you create or edit a template, the Edit SQL Template dialog appears.



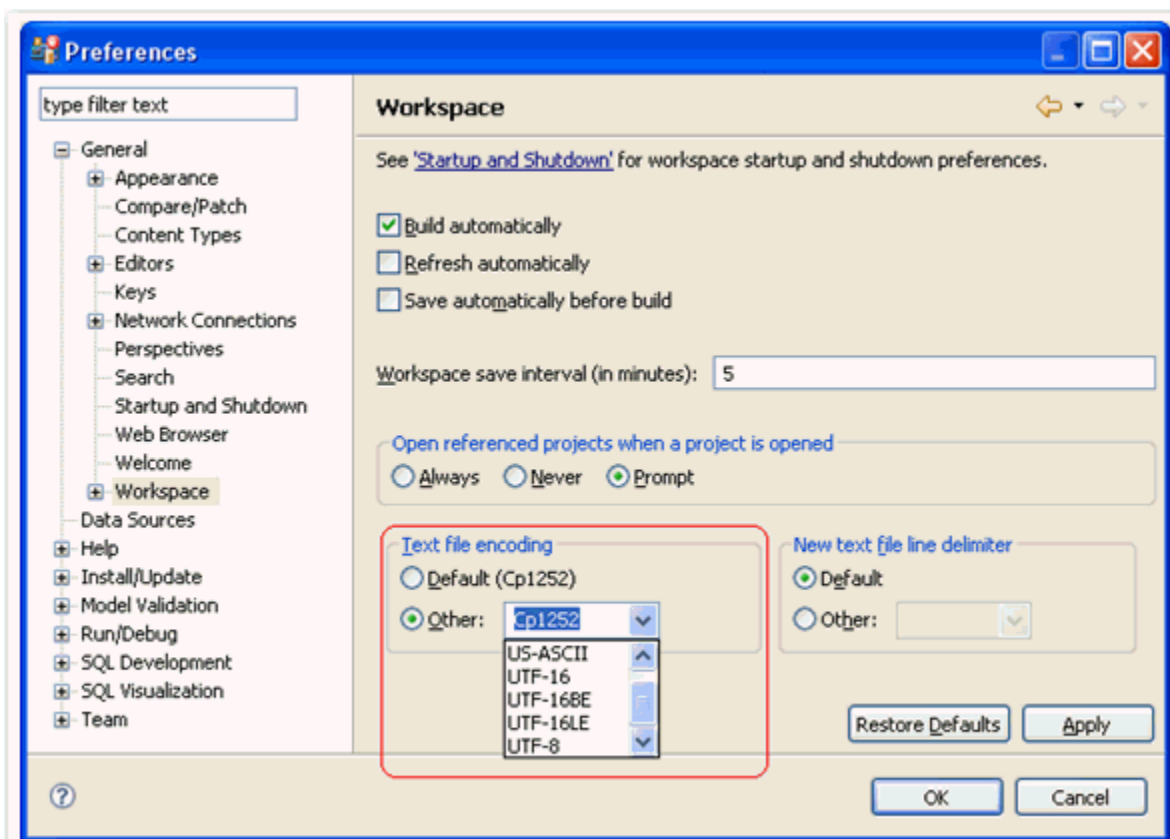
Enter a Name, Description, and Pattern in the fields provided, and click OK. If the template name doesn't match an existing SQL code template, your new template is added to the list, and will automatically be considered when the code assist function is executed in SQL Editor.

Select the Use Code Formatter check box to apply code formatting preferences to the specified template. See "[Set Code Formatter Preferences](#)" on page 19 for more information about setting code formatter preferences.

SET FILE ENCODING PREFERENCES

The Workspace panel provides options for unicode support in SQL files.

Select **Window > Preferences > General > Workspace**.



The default encoding for text files on Windows platforms is Cp1252. You can change unicode support in from file to file using the Text File Encoding options available on the Workspace panel.

To change text file encoding in the development environment:

- 1 Select **Window > Preferences > General > Workspace** and click the **Other** option under **Text File Encoding**.
- 2 Use the drop down menu and select an encoding mode from the list provided. Click **Apply** to keep your changes.

To change text file encoding on a specific, folder, or project in:

- 1 Right-click on the file, folder or project that you want to modify and choose **Properties**.
- 2 Modify the encoding selection on the Resource properties page that appears.

USING DB OPTIMIZER

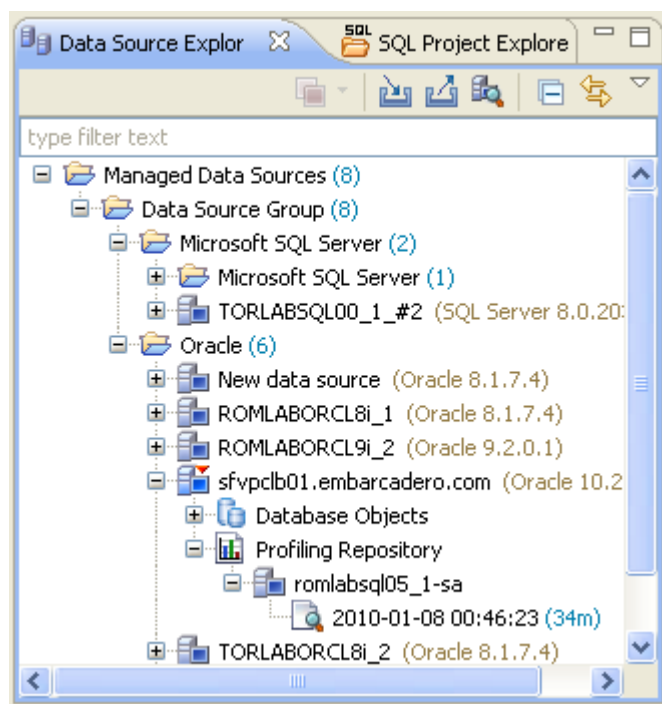
This section describes how to use the features of DB Optimizer to optimize your database operations. This section contains an overview of DB Optimizer functionality and also contains detailed instructions for

- [Working with Data Sources](#) on page 23
- [Working with SQL Projects](#) on page 33
- [Creating and Editing SQL Files \(SQL Editor\)](#) on page 37
- [Troubleshooting](#) on page 62.

WORKING WITH DATA SOURCES

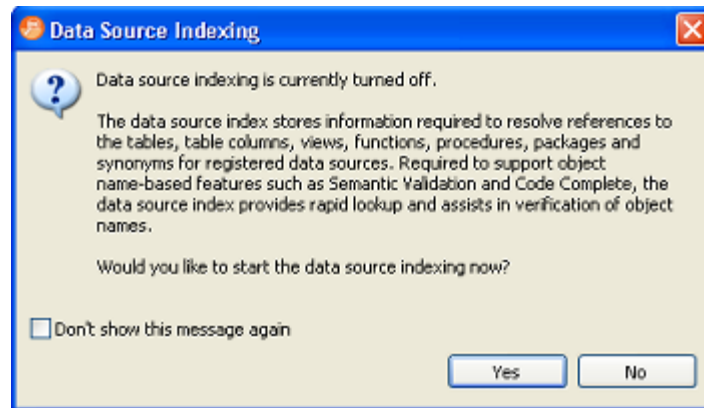
The Data Source Explorer provides a tree view of all registered data sources and associated database objects. When you first start DB Optimizer, a prompt appears and offers to populate Data Source Explorer from multiple sources on the system. This includes previously-registered data sources on other Embarcadero products, and third-party DBMS clients such as TOAD. If DB Optimizer cannot detect a data source, you can register it manually.

Additionally, you can initiate this feature by clicking the Auto-Discovery button on the Toolbar or via the **File > Import > Embarcadero > Data Sources > Previously Registered Embarcadero Data Sources (Registry)** command from the Main Menu.



The Profiling Repository entries in the Data Source Explorer are available only when configured in the **Profile Configuration Dialog** for Oracle data sources only. These are saved profiling sessions that you can share with other DB Optimizer users. For information on configuring the datasource profiles, see "[Building Launch Configurations](#)" on page 107.

The first time you connect to a datasource you will see a dialog prompting you to index the datasource. I



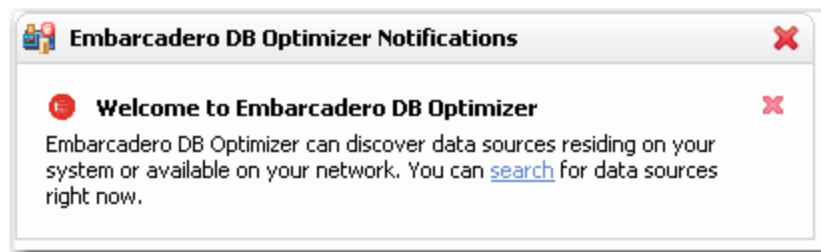
Data source indexing is important for the SQL editor for code completion and semantic validation, however, it is not necessary for profiling and tuning. For more information, see "[Set Index Configuration Preferences](#)" on page 15.

TIP: If you select **Don't show this message again**, and later want to be prompted to enable indexing on connect you can change the setting on the Preferences page, **Window > Preferences > SQL Development > Data Source Indexing**.

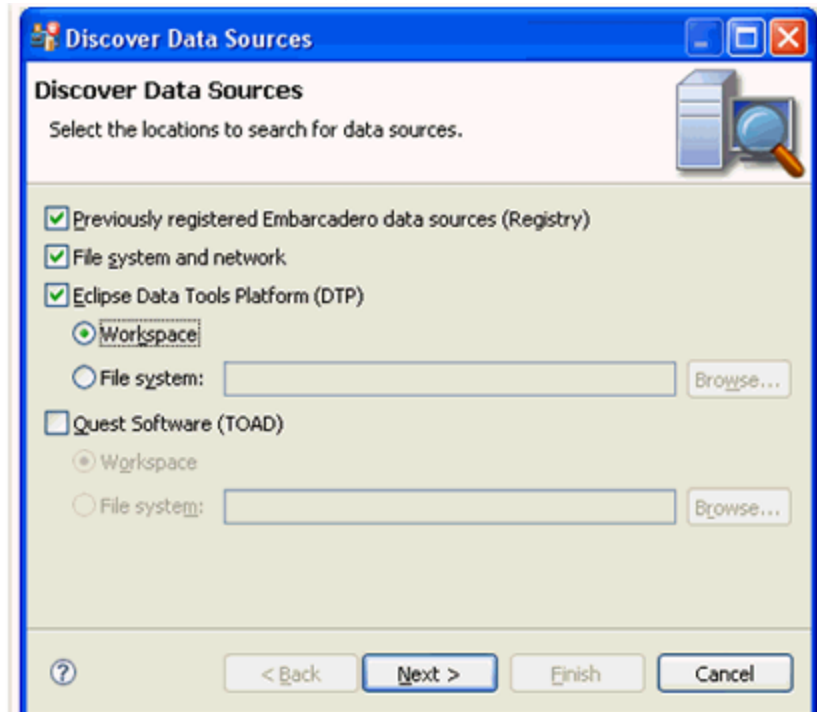
REGISTER DATA SOURCES

When DB Optimizer is started, it prompts you to discover data source catalogs that have been created by any previously installed Embarcadero products (DBArtisan, Rapid SQL, DB Optimizer), or other instances of DB Optimizer.

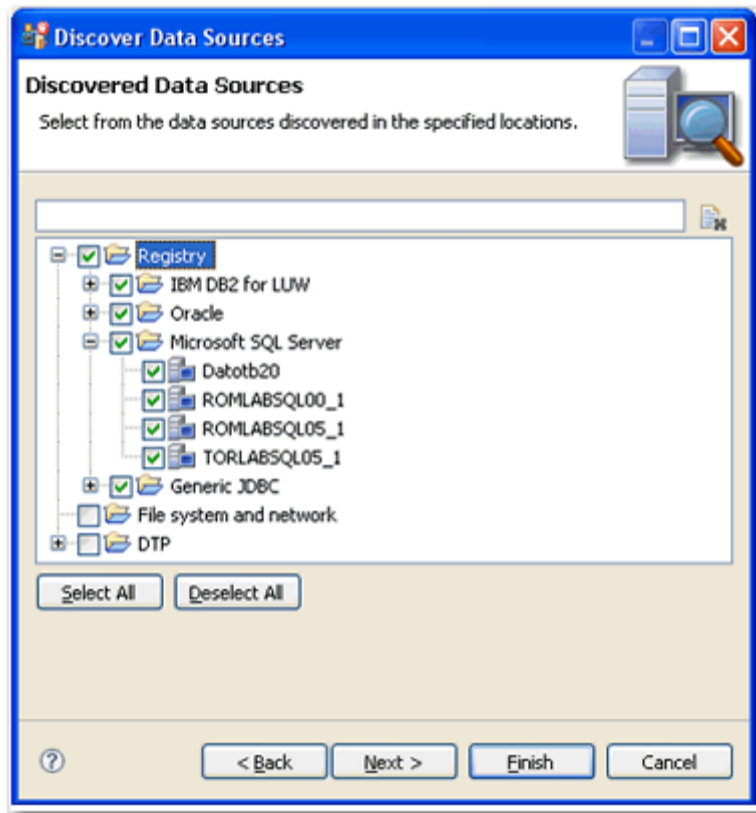
Additionally, the system scans your machine for the client software of all supported third-party DBMS platforms (TOAD, Eclipse Data Tools Platform, etc.). These data sources are automatically added to the data source catalog.



To manually initiate the scan later, click the Discover Data Sources icon at the top of Data Source Explorer. The Discover Data Sources dialog appears.



- 1 Choose the type of data sources you want to scan for and click **Next**. The wizard automatically returns all data sources it finds on your machine based on the criteria you specified



- 2 Choose the data sources you want to add to the DB Optimizer environment and click **Finish**. Data Source Explorer automatically populates with the new data source selections.

TIP: To add data sources manually, right-click Managed Data Sources in the Data Source Explorer tree, select **New > Data Source**, and enter the connectivity parameters as prompted.

For additional information on data source connection parameters, see "[DBMS Connection Parameters by Platform](#)" on page 254.

Once registered, the data source appears in the Data Source Explorer view. If you have created more than one workspace, they all share the same data source catalog.

Once a data source has been registered, the connection parameters are stored locally. In some cases, a user ID and password are required to connect to a registered data source. DB Optimizer™ XE and DB Optimizer™ can encrypt and save user IDs and passwords to connect automatically.

NOTE: In some cases, older versions of DB Optimizer and DB Artisan/Rapid SQL are not compatible with this version of DB Optimizer, and the methods listed above will not import these older data source catalogs. If you are experiencing difficulties, you can import the old data sources via the Windows registry by selecting **File > Import... > Embarcadero > Data Sources > Previously Registered Embarcadero Data Sources** (Windows Registry).

BROWSE A DATA SOURCE

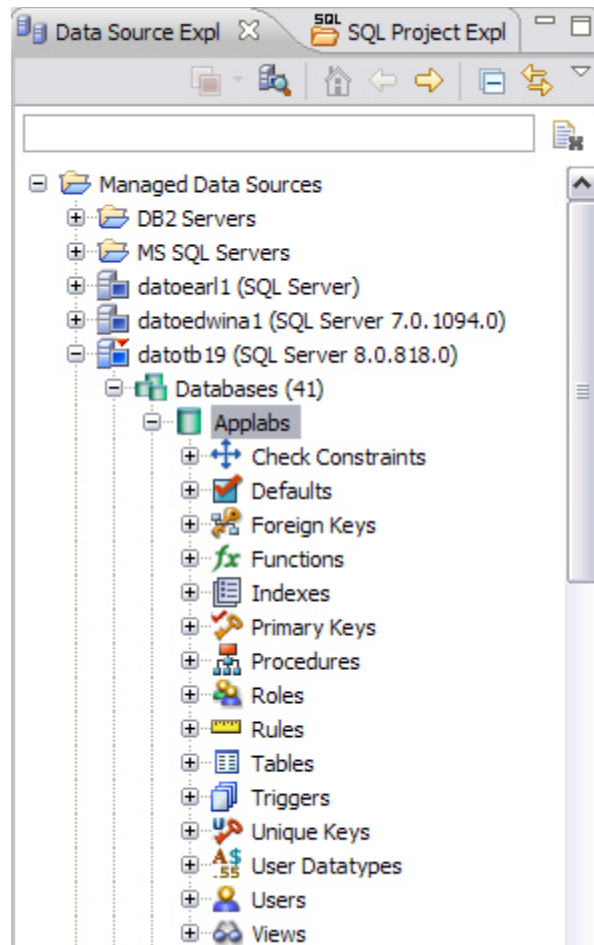
You can drill down in the Data Source Explorer tree to view registered databases on a server, and view tables, and other objects in a database. Additionally, you can view the structure of individual objects such as the columns and indexes of a table. Right-click the object for a menu of available commands, such as Extract to Project, which creates a new SQL file containing the object's DDL.

In most cases, whenever you browse a data source, DB Optimizer requires login information in order to connect with the data source. Enter a valid user name and password in the fields provided. The Auto Connect option retains your login credentials for future connections to the same data source.

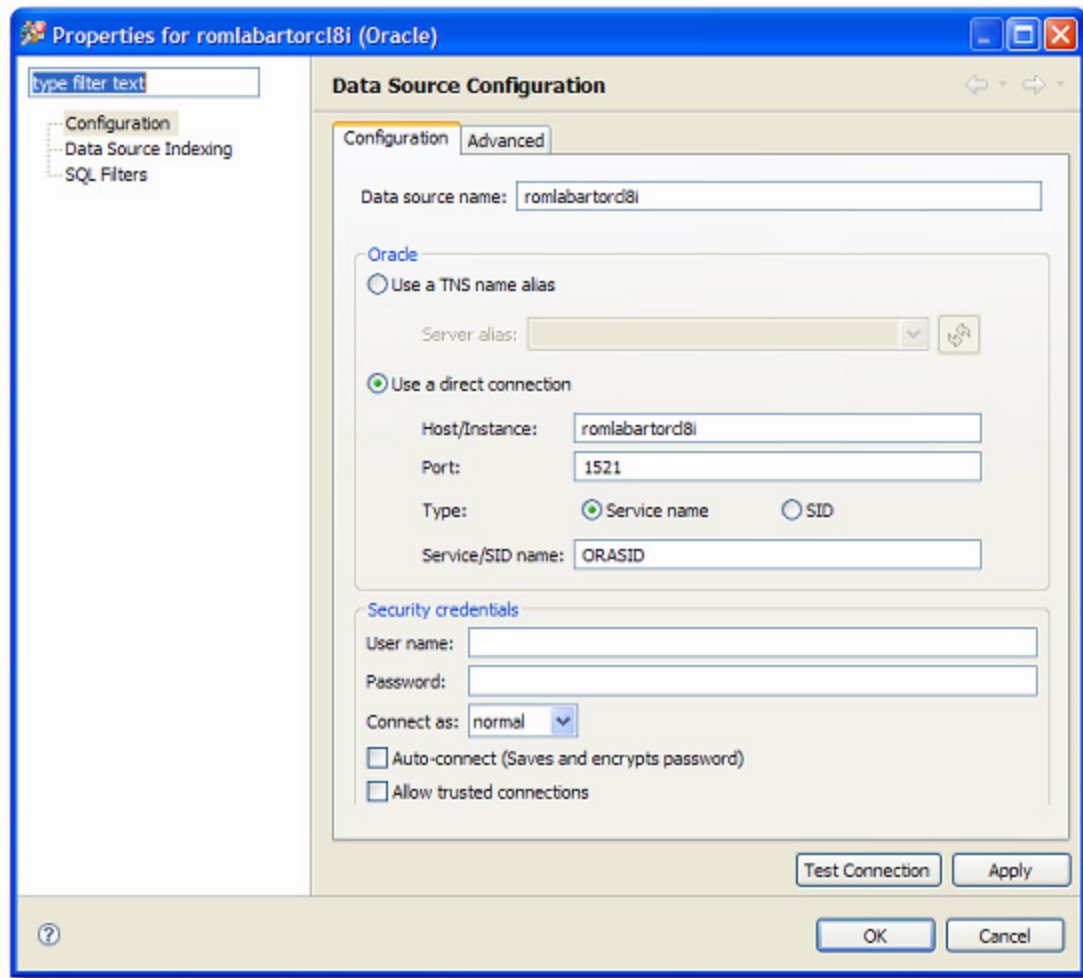
You can turn off the Auto Connect feature by right-clicking on a specified data source and toggling the Connect on Expand option. By default, when Connect on Expand is active, DB Optimizer automatically attempts to connect to the server each time you browse a data source.

VIEW DATABASE OBJECT PROPERTIES

All objects in Data Source Explorer contain properties as they relate to the DB Optimizer application.



DB Optimizer Object Properties are viewed via the Properties dialog. The dialog is accessed by right-clicking the object in Data Source Explorer.



To view Data Source Explorer object properties:

The Info properties node is accessed by right-clicking a data source in Data Source Explorer.

The dialog displays properties with regards to Configuration, Data Source Indexing, and SQL Filters.

As well, each node representing the actual data source connection (the uppermost parent in a list of data source objects), contains additional properties in addition to the Info node and its respective properties. With the exception of the Configuration node, these values can be modified in the Properties dialog.

The Configuration node is composed of:

- Data Source Name
- Data Source Type

- and three subnodes: Connection Information, Data Source Information, and Security Parameters.

These nodes are identical to the parameters used to initially define the data source during the data source registration process. For more information on these values and how to modify them, see "[Register Data Sources](#)" on page 24.

The SQL Filter node enables a developer to place filters on data source objects that appear in the Database Explorer. For more information, see [Filter Database Objects](#) on page 30.

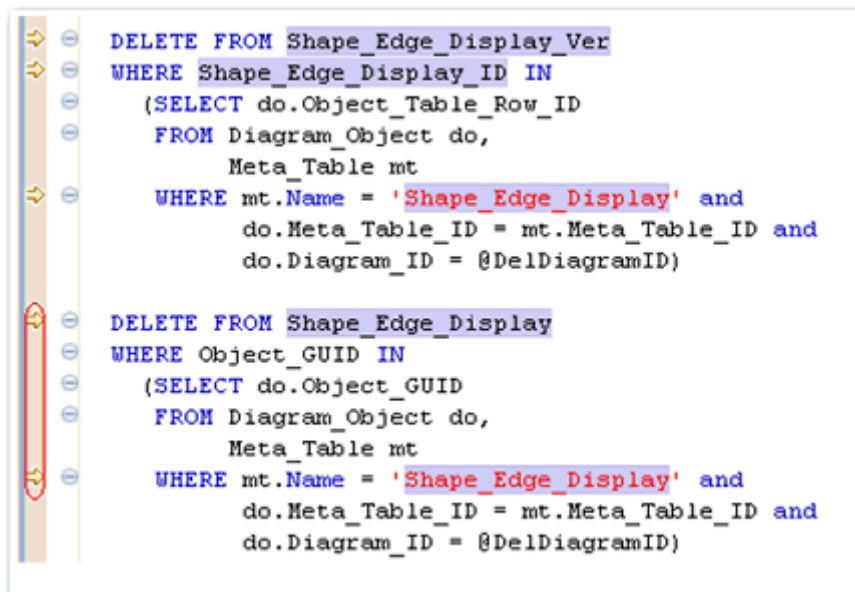
SEARCH FOR DATABASE OBJECTS

Database object searches rely on the Object Index when returning results. By default, caching is set to configure only parts of a database. To configure the Index to expand object searches, see [Set Index Configuration Preferences](#) on page 15.

- 1 Select **Search > Database**. By default, the search scope is all currently connected databases. Under Specify the scope for the search, clear any databases or server check boxes you do not want to search.
- 2 Specify the search criteria:
 - Type the value to search for in the **Search String** field. Use the * character to indicate wildcard string values and the ? character to indicate wildcard character values.
 - Select **Case Sensitive** to indicate to the search function that you want case sensitivity to be a factor when searching for appropriate string matches.
 - Select **Search Indexed Data** to indicate that the search function should read the Index. This increases the performance of the search function and will typically result in faster returns on any hits the search might make.
 - Select **Apply SQL Filters** to apply any relevant database or vendor filters to the search.
 - Choose **Declarations, References, or All Occurrences** to specify what the search is restricted to in terms of database objects.
 - A **Declaration** is an instance where an object is declared. For example, an object is declared in a CREATE table.
 - A **Reference** is an instance where an object is used or referred to. For example, an object is referred to in a procedure or as a foreign key in a table.
 - Choose **All Occurrences** to return both declarations and references in the search results.
 - Use the check boxes beside the database object panel to select and deselect the specific database objects that you want to be included in the search process.

3 Click **Search**.

The results of your search are generated in the **Search** view. When you open a matched file, references to the keyword are flagged with yellow arrow icons that appear in the left-hand column of the editor.



You can navigate between keywords within all returned files using the yellow “up” and “down” arrows that appear at the top of the **Search** view.

FILTER DATABASE OBJECTS

Filters can be placed on data sources and corresponding data source objects to restrict their display in Data Source Explorer. This feature is useful if you have data sources that contain large numbers of database objects. You can apply filters to view only the schema objects you need for the development process.

There are two types of data source filters available:

- **Global filters** that affect all registered data sources in the DB Optimizer™ XE and DB Optimizer™ development environment.
- **Data Source specific filters** affect only the specified data source for which they are defined.
 - On Sybase and SQL Server platforms, you can apply database filters, which enables you to set different filters on different databases within the same source.

In both cases, data source object filters are defined via the Object Filter Manager, through the development of filter templates. Once defined, filter templates can be activated and deactivated as you need them.

Several filter templates can be combined at a global level or applied to a specific data source.

See also:

[Define Global Database Object Filters](#) on page 31

[Define Data Source-Specific Object Filters](#) on page 31

DEFINE DATA SOURCE-SPECIFIC OBJECT FILTERS

Data source-specific object filters affect only the specified data source.

To define data source-specific filters:

- 1 In **Data Source Explorer**, right-click the data source and select **Properties**.
The **Properties** dialog appears.
- 2 Select the SQL Filters node and select **Enable data source specific settings**. The other controls on the dialog become enabled.
- 3 Click **New**. The **Filter Template** dialog appears.
- 4 Specify the parameters of the filter.
 - In the **Name** field, enter the name of the filter as you want it to appear in the selection window on the SQL Filter node.
 - The **Database Type** pane provides a list of data source objects. Deselect the data source objects that this template filters so that they do not appear in Database Explorer when displaying data source objects for the data source.
 - Click **New** to add filter parameters for data source object properties. The New SQL Filter Predicate dialog appears.
 - Use the **Property** and **Operator** fields to supply the filter criteria. Property specifies whether the value is a Name or Schema, and Operator specifies the matching type of the filter syntax. (Equals, Not Equals, Like, Not Like, In, Not In)
 - In the **Value** field, enter the full or partial syntax of the property or properties you want to filter in Data Source Explorer.
- 5 Click **OK**. The filter property specification is added to the Filter Template.
- 6 When you have finished defining the filter template, click **OK**. The template name is added to the Properties dialog. It can be enabled and disabled by selecting or deselecting the check box beside its name, respectively.

DEFINE GLOBAL DATABASE OBJECT FILTERS

Global filters affect all registered data sources in the DB Optimizer™ XE and DB Optimizer™ development environment. When you create and apply a global filter to a platform vendor in DB Optimizer™ XE and DB Optimizer™, all databases associated with that vendor are affected by the filter, as defined.

Individual global filter templates are separated, by supported data source platform, on tabs in the SQL Filter window. Select the appropriate tab to view existing filter templates or add new ones, as needed.

To define a global filter:

- 1 Select **Window > Preferences** from the Main Menu. The **Preferences** dialog appears.
- 2 Expand the **SQL Development** node and select the **SQL Filter subnode**. The **SQL Filter** pane appears.
- 3 Click **New**. The **Filter Template** dialog appears.
- 4 Specify the parameters of the filter template:
 - In the **Name** field, enter the name of the filter as you want it to appear in the selection window on the SQL Filter node.
 - The **Database Type** pane provides a list of data source objects. Deselect the data source objects that this template filters so that they do not appear in Database Explorer when displaying data source objects for the data source.
 - Click **New** to add filter parameters for data source objects properties. The **New SQL Filter Predicate** dialog appears.
 - Use the **Property** and **Operator** fields to supply the filter criteria. Property specifies whether the value is a Name or Schema, and Operator specifies the matching type of the filter syntax. (Equals, Not Equals, Like, Not Like, In, Not In)
 - In the **Value** field, enter the full or partial syntax of the property or properties you want the template to filter in data source Explorer.
- 5 Click **OK**. The filter property specification is added to the Filter Template.
- 6 When you have finished defining the filter template, click **OK**. The template name is added to the Properties dialog. It can be enabled and disabled by selecting or de-selecting the check box beside its name, respectively.

TIP: Data Source object filters are added and removed from the development environment by selecting and de-selecting the checkboxes associated with each filter template on both the global and data source-specific dialogs.

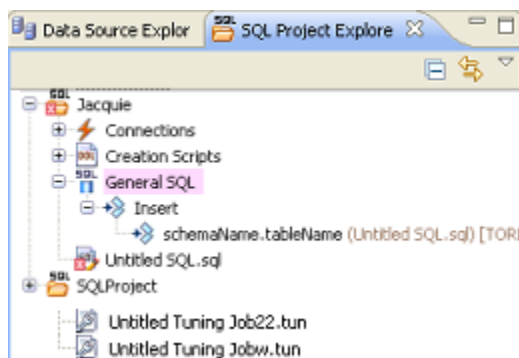
DROP A DATABASE OBJECT

To delete an object permanently from a database, right-click the object in Data Source Explorer and choose **Drop** from the menu. The **Drop Wizard** prompts you to confirm removal of the object and provides a DDL preview of the deletion code.

WORKING WITH SQL PROJECTS

You create projects to organize and store SQL development files. The purpose of projects is to keep your work-in-progress files organized, as well as maintain a common directory structure when developing code and executing files on registered data sources. Once a file has been developed and is ready for deployment, that file can then be executed on a registered data source.

SQL Project Explorer is used to view and access files. It uses a tree view to display the project as a series of folder directories with a folder labeled with the project name as the parent directory, and with project categories, and associated project files as its children.



All files in an SQL Project project are organized under the following categories:

- **Connections:** List the connections of any given SQL file of a data source associated with the project.
- **Creation Scripts:** Provide DDL statements and statements that define database objects.
- **General SQL:** Provide a category for all other SQL files that are not used in database object creation. This includes DML files, and so on.
- **Large Scripts:** Contain all files larger than the currently set SQL Editor preference. The file size limit can be modified on the Preferences panel by selecting **Window > Preferences** in the Main Menu.

Physically, the projects and files you create as you work in DB Optimizer™ XE and DB Optimizer™ are stored under the Workspace directory you specified at the prompt when DB Optimizer™ XE and DB Optimizer™ was started. The directory and files can be shared, and other tools may be used to work on the files, outside the DB Optimizer™ XE and DB Optimizer™ development environment.

You can move existing files within a project by clicking and dragging the file you want to move in the Project Explorer from one node to another, or via the **File > Move** command.

CREATE A NEW SQL PROJECT

- 1 Select **File > New > SQL Project** from the **DB Optimizer™ XE and DB Optimizer™** Main Menu. The New Project Wizard appears.
- 2 Enter the appropriate information in the fields provided:
 - **Name:** Enter the name of the project as you want it to display in the Project Explorer view.
 - **DBMS Platform:** Select the data source platform to which the new project will be associated. This enables DB Optimizer™ XE and DB Optimizer™ to properly parse SQL development code for project files.
 - **Location:** When selected, the **Use Default Location** check box indicates the project is to be created under the currently selected Workspace. Deselect the check box and specify a new folder path if you do not want to create the project in the currently selected Workspace.
- 3 Click **Finish**. The new project icon appears in the Project Explorer view under the name that you specified. If you did not select Use Default Location, the project will appear in the appropriate Workspace when you open it in DB Optimizer™ XE and DB Optimizer™.

NOTE: Alternatively, you can select **New > SQL Project** from the Main Menu or click the New Project icon in the Tool Bar to create a new project.

OPEN AN EXISTING PROJECT

You can open projects by navigating to SQL Project Explorer and expanding the node of the project that contains the files you want to access.

Below each project name are a series of nodes that categorize any existing SQL files by development type:

- **Connections:** Lists the connections of any given SQL file of a data source associated with the project.
- **Creation Scripts:** General data source object development scripts. This node contains DDL statements and statements that define database objects.
- **General SQL:** Provides a category for all other SQL files that are not used in database object creation. DML files, etc.
- **Large Scripts:** Contains all files larger than the currently set SQL Editor preference. The file size limit can be modified on the Preferences panel. (Choose **Window > Preferences** in the Main Menu to access the panel.)

NOTE: Physically, the projects and files you create as you work in DB Optimizer™ XE and DB Optimizer™ are stored under the project directory that you specified at the prompt when the project was created. The directory and files can be shared, and other tools may be used to work on the files, completely exempt from the DB Optimizer™ XE and DB Optimizer™ development environment.

SEARCH A PROJECT

- 1 Select **Search > File**.
- 2 Specify the search criteria:
 - Type the value to search in the **Containing Text** field. Use the * character to indicate wildcard string values, the ? character to indicate wildcard character values, and the \ character to indicate an escape character for literals (* ? /).
 - Select **Case Sensitive** and indicate to the search function that it should take into account case when searching for appropriate string matches.
 - Select **Regular Expression** to indicate to the search function that the string is a regular function.
 - In the **File Name Pattern** field, specify the extension name of the files to search for explicitly. If the value in this field is a * character, the search function searches all files regardless of extension. Manually type in the extensions to indicate file type (separate multiple file types with commas), or click Choose and use the Select Types dialog to select the file extensions the process will search for the string by.
 - Select **Consider Derived Resources** to include derived resources in the search.
 - Select **Workspace** or **Working Set** to choose the scope of the search. If you choose Working Set, specify the name of the defined working set manually, or click Choose and navigate to the working set you want to search for in the provided string.
- 3 Click **Search**. The results of your search are generated in the Search view on the Workbench.

ADD FILES TO A PROJECT

Existing files that reside in directories outside of the workspace can be added to a project via the following methods:

- Dragging and dropping the file set from a system directory to SQL Project Explorer.
- Copying and pasting the file set from a system directory to SQL Project Explorer.
- Executing the Import command.

To drag/drop or copy/paste files from a system directory to SQL Project Explorer:

- 1 With the SQL Project Explorer view open, navigate to the directory where the files you want to add to the project are located on the system.
- 2 Drag and drop the files you need from Windows Explorer into SQL Project Explorer. The files appear in the tree view under the appropriate categories.

NOTE: Alternatively, you can use the Copy command on the files you want to add in Windows Explorer, and then right-click the Project Explorer and select Paste from the menu. The files appear in the tree view under the appropriate categories.

To use the Import command:

- 1 Right-click anywhere on the Project Explorer and select **Import**. The **Import** dialog appears.
- 2 Expand the **General** node and double-click **File System**. A dialog containing the import specification parameters appears.
 - In the **From directory** field, manually type the directory location of the files you want to import to Project Explorer, or click **Browse** and navigate to the appropriate folder. The panels below the field populate with the folder selection and a list of suitable files contained in that folder. Use the check boxes beside each folder and file to specify what folders/files you want the import function to add in Project Explorer.
 - In the **Into folder** field, manually type the name of the folder within Project Explorer where you want to import the files specified in the panels above, or click **Browse** and navigate to the appropriate folder.
 - Select the **Overwrite existing resources without warning** check box if you do not want to be prompted when the import process overwrites Project Explorer files that contain the same name as the imported files.
 - Choose **Create complete folder structure** or **Create selected folders only**, depending on whether you want the import process to build the folder structure of the imported directory automatically, or only create those folders you selected in the panels above, respectively.
- 3 Click **Finish**. The import process moves all selected folders and files into Project Explorer and thus into the DB Optimizer™ XE and DB Optimizer™ development environment.

NOTE: In addition to accessing the Import command via the shortcut menu, you can also access the Import dialog by choosing **File > Import...** from the Main Menu.

DELETE A PROJECT

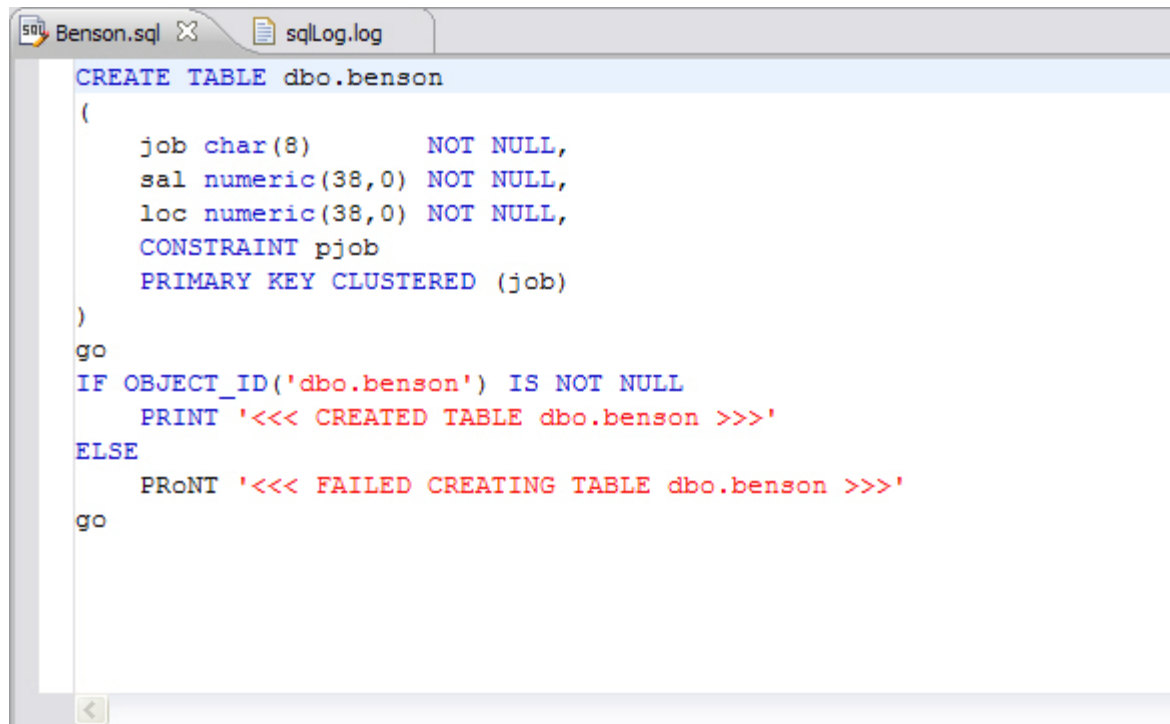
You can delete a project by right-clicking its folder in the SQL Project Explorer and selecting **Delete**.

When you delete a project, DB Optimizer™ XE and DB Optimizer™ will prompt you with a **Confirm Project Delete** dialog that asks you to confirm the deletion of the project, and offers you the option of deleting the project from the DB Optimizer™ XE and DB Optimizer™ interface, or deleting the project from the system.

- If you select **Do not delete contents**, the files and directory structure will be removed from SQL Project Explorer, but they will still exist on your machine.
- If you select **Also delete contents ...**, the files and directory structure will be removed from SQL Project Explorer and deleted from your machine.

CREATING AND EDITING SQL FILES (SQL EDITOR)

The SQL Editor is a Workbench interface component that enables the development, viewing, and formatting of SQL code.



The screenshot shows the SQL Editor window with two tabs: 'Benson.sql' and 'sqlLog.log'. The main editor area contains the following SQL code:

```
CREATE TABLE dbo.benson
(
    job char(8)          NOT NULL,
    sal numeric(38,0) NOT NULL,
    loc numeric(38,0) NOT NULL,
    CONSTRAINT pjob
    PRIMARY KEY CLUSTERED (job)
)
go
IF OBJECT_ID('dbo.benson') IS NOT NULL
    PRINT '<<< CREATED TABLE dbo.benson >>>'
ELSE
    PRINT '<<< FAILED CREATING TABLE dbo.benson >>>'
go
```

The Editor supports the following the functionality

- Code assist:
 - Code complete: Type Ahead and Name completion. For more information, see "[Understanding Code Assist](#)" on page 43.
 - Code templates: Templates for creation of tables, procedures, etc. For more information, see "[Understanding SQL Templates](#)" on page 55.
 - Hyper links. For more information, see "[Understanding Hyperlinks](#)" on page 46.
 - Semantic validation. For more information, see "[Sematic Validation](#)" on page 38.
 - Object hovering: Hover over an error found and an explanation of the cause of the error appears.
- Code formatter. For more information, see "[Understanding Code Formatting](#)" on page 46.
- Code correction and transformations, For more information, see "[Examples of Transformations and SQL Query Rewrites](#)" on page 185.
- Object indexing: For more information, see "[Set Index Configuration Preferences](#)" on page 15.
- SQL Project Explorer. For more information, see "[Working with SQL Projects](#)" on page 33.

SQL Editor contains context-sensitive command menus that are tailored with pertinent functionality for the specified file format.

If SQL Editor does not recognize a selected file format, DB Optimizer™ XE and DB Optimizer™ automatically launches the file externally in the system default application. External editors are not embedded in the Workbench. For example, on most machines, the default editor for HTML files is the system Web browser. SQL Editor does not, by default, recognize HTML files, and opening an HTML file from the Workbench launches the file in an instance of the Web browser instead of the Editor.

Any number of instances of SQL Editor can be open on the Workbench at the same time. Multiple instances of SQL Editor displaying different content may be open in the same Workbench. These instances will be stacked by default, but can also be tiled side-by-side so the content of various files can be viewed simultaneously for comparison or multi-tasking purposes. When an instance of SQL Editor is active, the Workbench Main Menu automatically contains commands applicable to the file format. If a view is active, SQL Editor commands are disabled automatically, except when commands are still valid between the selected view and the file displayed in the interface.

Sematic Validation

When working with code in SQL Editor, the window contains a number of features that provide an increase in the efficiency and accuracy of code development. The following syntax highlighting changes are automatically applied to code as a user adds lines in the interface.

Code	Formatting
Comments	Green font, italics
SQL Commands	Dark blue font
Coding Errors	Red underline
Strings	Red font
Non-Executable Command Line Commands	Aqua font

Single line and multiple line comments appear in different colors.

Furthermore, SQL Editor provides two column bars, one on either side of the code window. The purple change bar in the left-hand column indicates that the line of code has been modified. Hover over the change bar to display the original code text. The red square in the right-hand column indicates that there are errors in the code window. Hover the mouse over the square to view the error count. Click the red bar in this column to navigate directly to the line in which the SQL Editor detects the error. SQL Editor automatically highlights the appropriate code. Non-executable command line commands are displayed in a different formatting style than SQL commands. Syntactic and semantic errors are also highlighted.

SQL Editor also features dynamic error detection, object lookup and suggestion features, code folding, and auto-formatting. SQL Editor is able to identify different areas in a statement, and enables users to retrieve subclauses, resolve table aliases, and dynamically return lists of tables, views, and columns, as needed.

See also:

[Working in SQL Editor](#) on page 39

CREATE AN SQL FILE

- 1 Create or open a SQL project.
- 2 Select **File > New > SQL File**. A blank instance of **SQL Editor** appears.

NOTE: If you are not in a SQL project when you create a new SQL file, it will not open in SQL Editor.

OPEN AN EXISTING SQL FILE

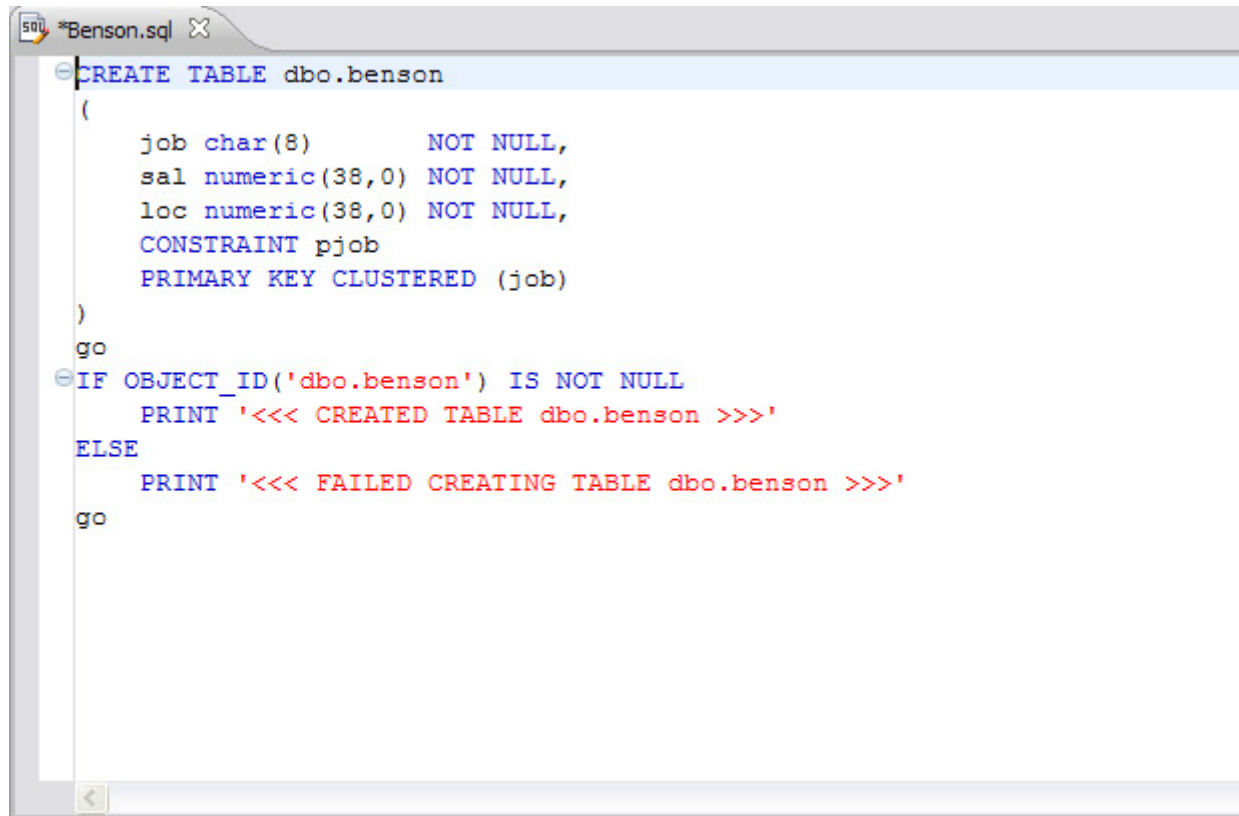
- 1 Open the SQL project containing the file, or that you want to contain the file.
- 2 If necessary, add the file to the project. (see "[Add Files to a Project](#)" on page 35)
- 3 In the SQL Project Explorer, double-click the file to open it in SQL Editor.

WORKING IN SQL EDITOR

SQL Editor handles SQL code formats and contains context-sensitive command menus, tailored with pertinent functionality for development purposes. Other files may be opened in DB Optimizer™ XE and DB Optimizer™, as well, but these are handled by other editors.

For example, if a text file is opened in the Workbench, DB Optimizer™ XE and DB Optimizer™ detects and opens the contents of that file in a text editor viewer with pertinent commands for that file type.

Any number of instances of SQL Editor can be active on the Workbench at the same time. Multiple instances of SQL Editor displaying different content may be active on the same Workbench. These instances will be stacked, by default, but can also be tiled side-by-side, so the content of various files can be view simultaneously for comparison or multi-tasking purposes. When an instance of SQL Editor is active, the Main Menu contains commands applicable to the file format. If a view is active, SQL Editor commands are disabled automatically, except when commands are still valid between the selected view and the file displayed in the interface.



Among the commands SQL Editor supports via the right-click menu:

- **Revert File:** Automatically restores the working file to the original text as it appeared the last time the **Save** command was issued.
- **Shift Right/Shift Left:** Indents the line of code in the working file to the right or left, respectively.
- **Toggle Comments:** Hides or displays comments in the code of the working file, depending on the current hide/show state.

- **Add Block Comment/Remove Block Comment:** A block comment is used to insert a comment into SQL code that spans multiple lines and begins with a forward slash and asterisk. While block comments are typically used to insert a command that spans multiple lines, some developers find them more useful than line comments, especially if a development team is using different text editors on an individual basis. Moving code from one text editor to another often breaks line comments in the middle of a line and causes errors. Block comments can be broken without causing errors.

NOTE: In addition to editing commands, some commands such as extract, drop, and execute can be accessed by right-clicking over statements in SQL code that are performed on specific tables, views, and columns. These commands will appear automatically in the appropriate menu when the code is highlighted. Full information on using these commands is found elsewhere in this documentation, based on the task each executable performs.

- **Explain Plan:** An explain plan details the steps that occur in SELECT, UPDATE, INSERT, and DELETE statements and is primarily used to determine the execution path followed by the database in its SQL execution.

See also:

- [Understanding Automatic Error Detection](#) on page 41
- [Understanding Code Assist](#) on page 43
- [Understanding Hyperlinks](#) on page 46
- [Understanding Code Formatting](#) on page 46
- [Understanding Code Folding](#) on page 51
- [Understanding Code Quality Checks on page 51](#)
- [Understanding SQL Templates](#) on page 55

UNDERSTANDING AUTOMATIC ERROR DETECTION

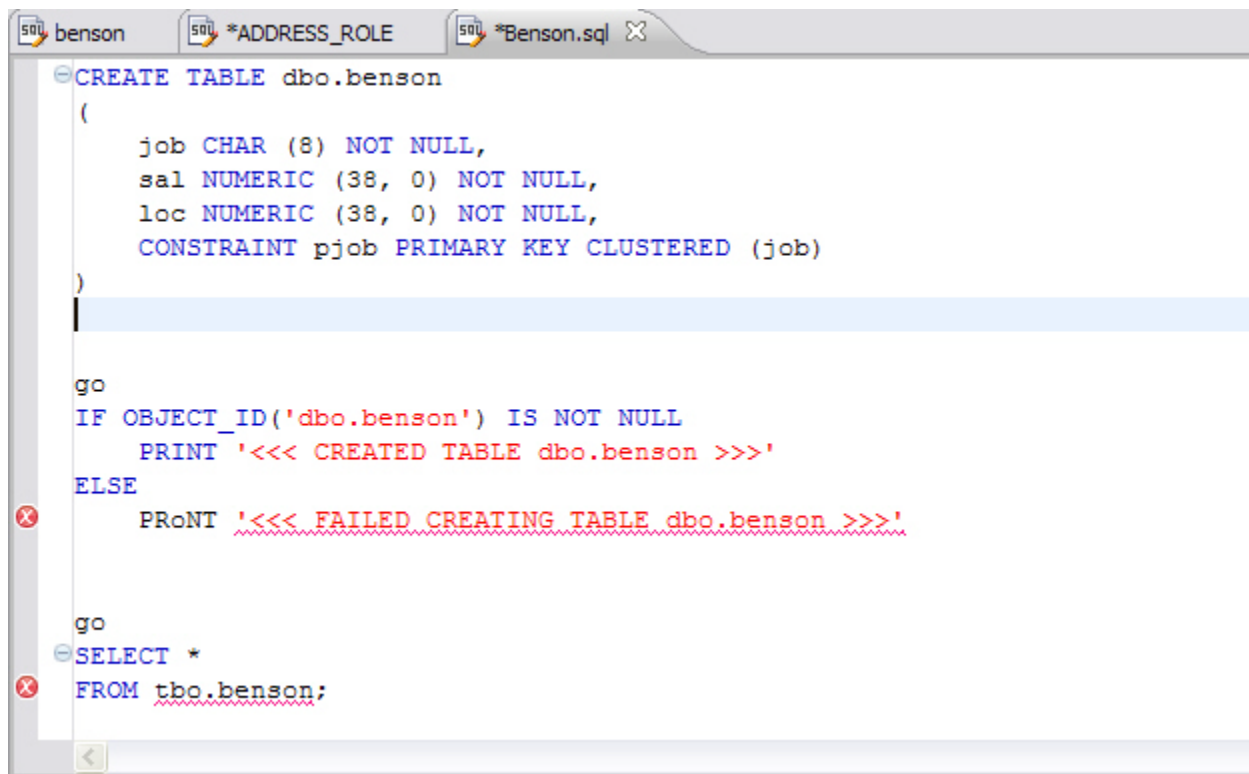
SQL Editor orders and classifies SQL statements. This enables it to edit code as you work within SQL Editor and highlight errors and typographical errors in “real time”. As you work, SQL Editor examines each clause in a statement and provides error reporting and other features as required.

SQL Editor identifies the following clauses and elements:

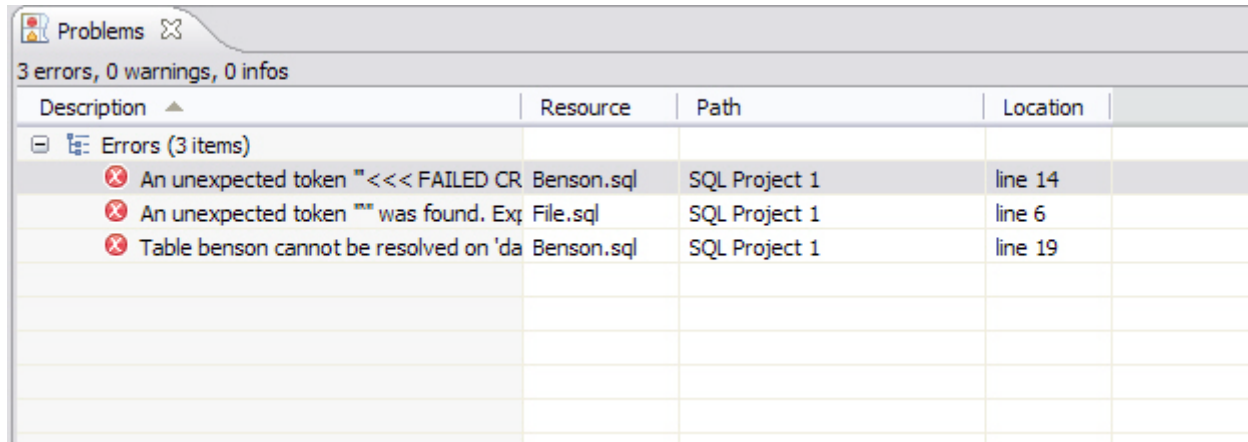
- **SELECT:** Specifies the field, constants, and expressions to display in the query results.
- **FROM:** Specifies one or more tables containing the data that the query retrieves from.
- **WHERE:** Specifies join and filter conditions that determine the rows that query returns. Join operations in a WHERE clause function in the same manner as JOIN operations in a FROM clause.

- **GROUP BY:** Specifies one or more columns used to group rows returned by the query. Columns referenced in the SQL SELECT statement list, except for aggregate expressions, must be included in the GROUP BY clause. You cannot group by Memo, General or Blob fields.
- **HAVING:** Specifies conditions that determine the groups included in the query. If the SQL statement does not contain aggregate functions, you can use the SQL SELECT statement containing a HAVING clause without the GROUP BY clause.
- **ORDER BY:** Specifies one or more items used to sort the final query result set and the order for sorting the results.

As you develop code in SQL Editor, it automatically detects semantic errors on a line-by-line basis. Whenever an error is detected, the line is flagged by an icon located in the left-hand column of the editor.



Additionally, all semantic errors detected in SQL Editor are displayed in the Problems view.



Right-click the an error and select **Go To** in order to find the error. DB Optimizer™ XE and DB Optimizer™ opens and navigates to the specific line of code containing the specified error.

UNDERSTANDING CODE ASSIST

When SQL Editor has finished analyzing a partial piece of code, it displays a list of data source objects for you to select from.

SQL Editor takes the following into consideration when analyzing code for a list of possible data source objects for insertion:

- Text to be inserted
- Original text to be replaced
- Content assist request location in original text
- The database object represented by the insertion text

Generally, insertion suggestions use the following format:

```
<insertion_text > - <qualification_information >
```

Code assist is available for SELECT, UPDATE, INSERT, and DELETE statements, as well as stored procedures, and functions (built-in and user defined.)

Additionally, code suggestions can be made for DML statements nestled within DDL statements. This functions in the same manner as code assist for statements that are not nestled, and applies to CREATE PROCEDURE, FUNCTION, TRIGGER, TABLE, and VIEW statements.

When the code assist window is open, you can filter out singular object suggestions by pressing (Ctrl + Spacebar). This removes all objects from the assist window while retaining procedures and functions. To display objects again, press (Ctrl + Spacebar) again.

The following table displays a list of all possible object suggestions, and the format in which SQL Editor inserts the suggestions into a statement:

Object and Stored Procedure Suggestions

Object Suggestion	Syntax/Example
Table	(TABLE) [catalog].[schema] EMPLOYEE - (TABLE)HR
Alias Table	(TABLE ALIAS) [catalog].[schema]tableName EMPLOYEE-(TABLE ALIAS)HRJOBS
Column	datatype - (Column) [catalog].[schema].tableName JOB_TITLE: varchar(20)- (Column)HRJOBS
Alias Column	datatype - (COLUMN ALIAS) [catalog].[schema].tableName. columnName JOB_TITLE:int-(COLUMN ALIAS)HR.JOBS.JOB_ID
Schema	(SCHEMA) [catalog] dbo-(SCHEMA)NorthWind
Catalog	(CATALOG)
Call	Call HR.ADD_JOB_HISTORY

Function Suggestions

Function Suggestion	Syntax/Example
Built-in	SELECT A FROM HR.DEPARTMENTS WHERE HR.DEPARTMENTS AVG
User-Defined	SELECT + FROM HR.CLIENTS WHERE HR.F_PERSONAL

NOTE: Function suggestions are only available for Oracle and DB2 platforms.

SQL Editor detects incomplete or erroneous code, processes the code fragments, and then attempts to apply the appropriate logic to populate the code.

As code is typed into SQL Editor, the application 'reads' the language and returns suggestions based on full or partial syntax input.

Depending on the exact nature of the code, the automatic object suggestion feature behaves differently; this enables SQL Editor to provide reasonable and 'intelligent' suggestions on coding.

Additionally, semantic validations can be made for DML statements nestled within DDL statements. This functions in the same manner as validation for top-level statements, and applies to CREATE PROCEDURE, FUNCTION, TRIGGER, TABLE, and VIEW statements.

The following chart displays the possible statement fragments that SQL Editor will attempt to suggest/populate with objects:

Statement Fragment Elements	Object Suggestion Behavior
SELECT	A list of tables, when selected automatically, prompts the user to select a column.
UPDATE and DELETE	A list of tables appears in the FROM and/or WHERE clause.
INSERT	A list of tables and views appears in the INSERT INTO and OPEN BRACKET clause prior to values. A list of columns based on the table or view name appears in the OPEN BRACKET or VALUES clause.

In addition to DML statements, SQL Editor also suggests objects based on specific fragmented syntax per line of code:

Statement Syntax	Object Suggestion Behavior
A partial DML statement (for example SEL ... indicates a fragment of the SELECT clause)	The keyword is completed automatically, assuming SQL Editor can match it. Otherwise, a list of suggested keywords is displayed. If the preceding character is a period, and the word prior is a table or view, a list of columns appears. If the word being typed is a part of a table name (denoted by a schema in front of it) the table name is autocompleted. If the word being typed has a part of a column name (denoted by a table in front of it) the column name is autocompleted.
Without typing anything.	A list of keywords appears.
A period is typed.	If the word prior to the period is a name of a table or view, a list of columns is displayed. If the word prior to the period is a schema name, a list of table names is displayed. If the word prior to the period is either a table name or a schema name, then both a list of columns and a list of table names is displayed.

To activate code suggestions:

By default, code suggestions are automatically offered if you stop typing in SQL Editor for one second. You can turn off the automated suggestion feature on the Code Assist preferences page.

If automated code suggestion is disabled, you can still access the suggestion window using the following method:

- 1 Click the line that you want SQL Editor to suggest an object for.
- 2 Press (**CTRL + Spacebar**) on your keyboard. SQL Editor 'reads' the line and presents a list of tables, views or columns as appropriate based on statement elements.

NOTE: On a per platform basis, auto-suggestion behavior may vary. (For example, the WITH statement on DB2 platforms.)

To modify object suggestion parameters, including setting it from automatic to manual, see "[Set Code Assist Preferences](#)" on page 18.

You can speed up the performance of the code assist functionality by enabling datasource indexing either when you connect to the datasource, see "[Working with Data Sources](#)" on page 23 or on the Preferences page, see "[Set Index Configuration Preferences](#)" on page 15.

UNDERSTANDING HYPERLINKS

SQL Editor supports hyperlinks that are activated when a user hovers their mouse over a word and presses the CTRL key. If a hyperlink can be created, it becomes underlined and changes color. When the hyperlink is selected, the creation script for the hyperlink object is opened in a new editor.

Hyperlinks can be used to link to tables, columns, packages, and other reference objects in development code. Additionally, hovering over a hyperlink on a procedure or function of a call statement will open it. You can also use the hyperlink feature on function calls in DML statements.

Clicking a hyperlink performs an action. The text editor provides a default hyperlink capability. It allows a user to click on a URL (for example, www.embarcadero.com) and database object links.

Hyperlink options (look and feel) can be modified via the Hyperlinking subnode in the Editors > Text Editors node of the Preferences panel.

NOTE: Hyperlink functionality relies on certain objects being captured in the Object Index. If the index is turned off, or has been restricted in what information it captures, users will be unable to link them (as they are non-existent within the Index.) To specify object index types, see "[Set Index Configuration Preferences](#)" on page 15.

UNDERSTANDING CODE FORMATTING

Code formatting provides automatic code formatting in SQL Editor while you are developing code.

To access the code formatter, select the open editor you want to format and select Ctrl+Shift+F. The code is formatted automatically based on formatting parameters specified in the Code Formatter subnode of the SQL Editor node in the Preferences panel.

You can also format an entire group of files from Project Explorer. To do so, select the directory or file and execute the Format command via the shortcut menu. The files will be formatted automatically based on your formatting preferences. See "[Set Code Formatter Preferences](#)" on page 19 for more information.

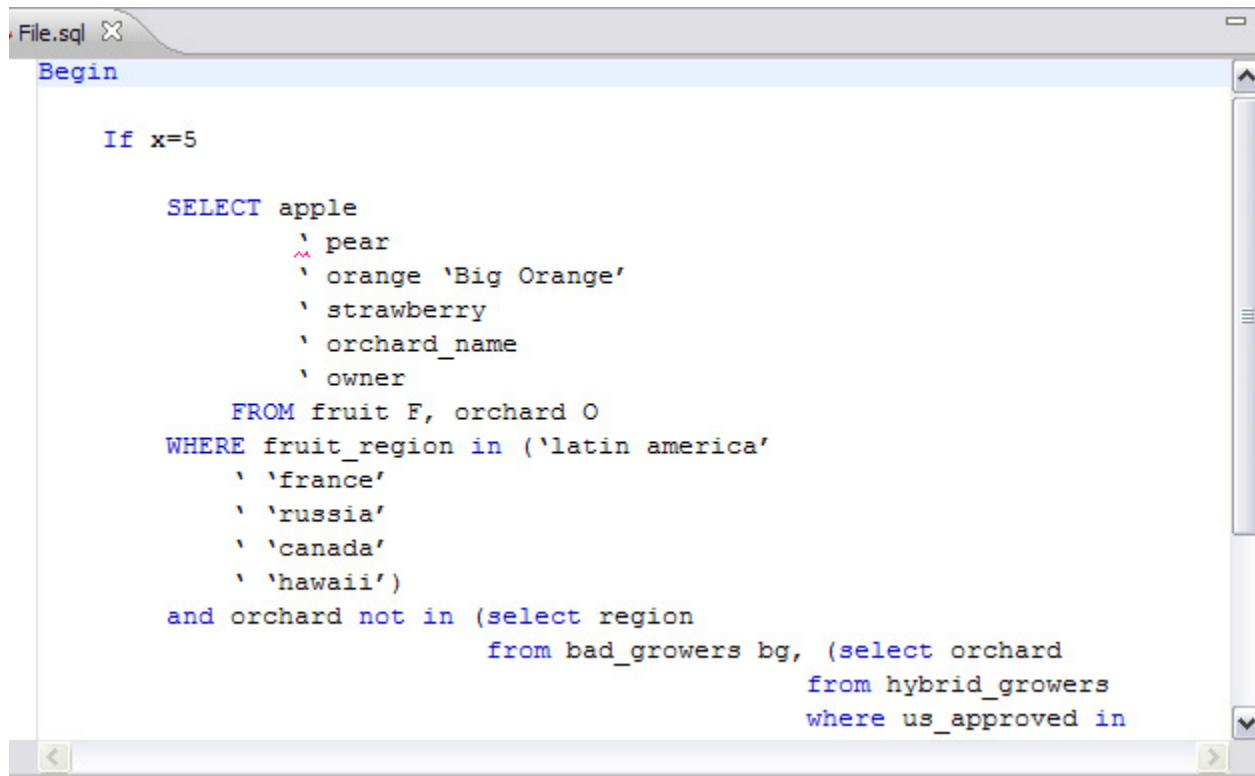
The following examples display a list of code formatting parameters and the resultant output in SQL Editor, based on the same set of SQL statements.

Custom Code Formatting Example 1

The following chart indicates a list of custom code formatting parameters and their corresponding values. The chart is followed by the actual syntax as it would appear in SQL Editor, based on the formatting parameter values. Compare the parameters and formatted code in Example 2 with this example for a concept of how custom formatting works.

Custom Code Formatting Parameter	Value (if applicable)
Stack commas separated by lists?	Yes
Stack Lists with ___ or more items.	3
Indent Size?	2
Preceding commas?	Yes
Spaces after comma?	1
Trailing commas?	--
Spaces before comma?	--
Right align FROM and WHERE clauses with SELECT statement?	Yes
Align initial values for FROM and WHERE clauses with SELECT list?	Yes
Place SQL keywords on their own line?	No
Indent size?	--
Indent batch blocks?	Yes
Number of new lines to insert	1
Indent Size	5
Right Margin?	80
Stacked parentheses when they contain multiple items?	No
Stack parentheses when list contains ___ or more items.	--
Indent Size?	5
New line after first parentheses?	No
Indent content of conditional and looping constructs?	Yes

Custom Code Formatting Parameter	Value (if applicable)
Number of new lines to insert?	1
Indent size?	5



The screenshot shows a window titled 'File.sql' with a 'Begin' button. The SQL code is as follows:

```
Begin

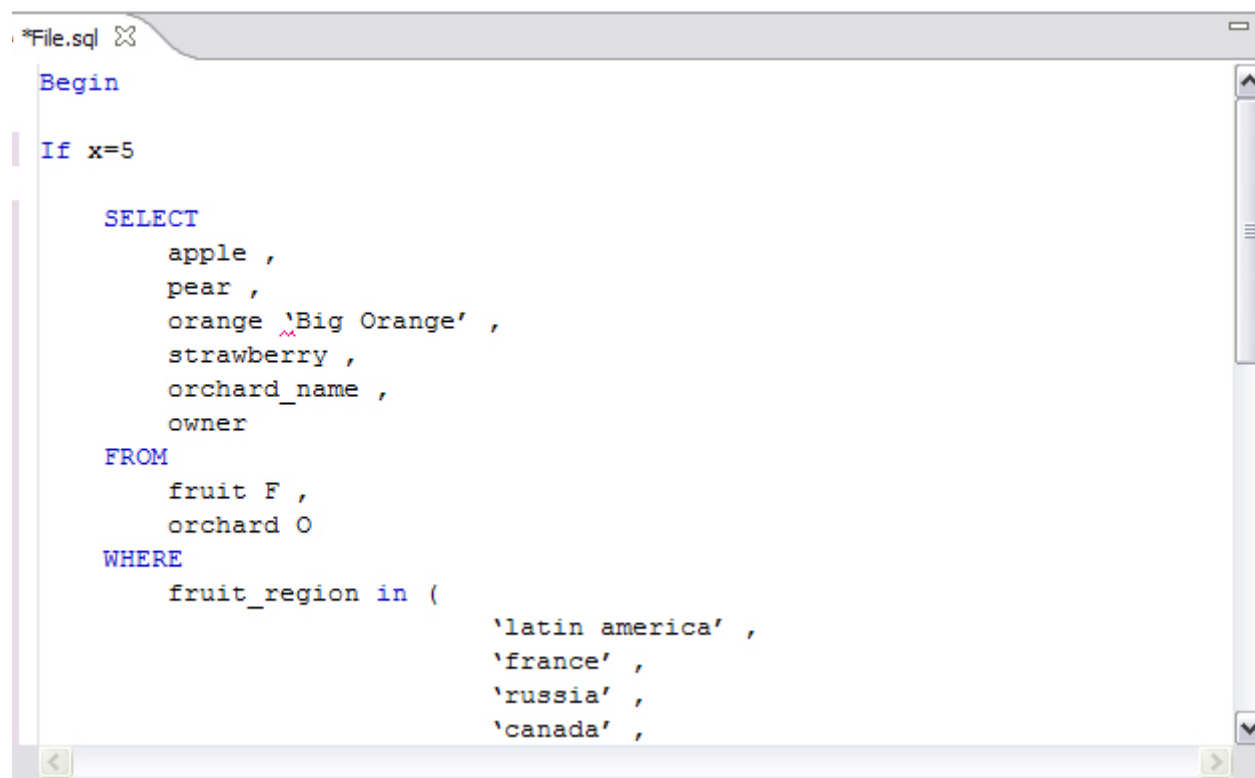
If x=5

    SELECT apple
        \ pear
        \ orange 'Big Orange'
        \ strawberry
        \ orchard_name
        \ owner
    FROM fruit F, orchard O
    WHERE fruit_region in ('latin america'
        \ 'france'
        \ 'russia'
        \ 'canada'
        \ 'hawaii')
    and orchard not in (select region
                        from bad_growers bg, (select orchard
                        from hybrid_growers
                        where us_approved in
```


Custom Code Formatting Example 2

The following chart indicates a list of custom code formatting parameters and corresponding values. The chart is followed by the actual syntax as it would appear in SQL Editor based on the formatting parameter values. Compare the parameters and formatted code in Example 1 with this example for a concept of how custom formatting works.

Custom Code Formatting Parameter	Value (if applicable)
Stack commas separated by lists?	Yes
Stack Lists with ____ or more items.	2
Indent Size?	0
Preceding commas?	--
Spaces after comma?	Yes
Trailing commas?	Yes
Spaces before comma?	2
Right align FROM and WHERE clauses with SELECT statement?	No
Align initial values for FROM and WHERE clauses with SELECT list?	--
Place SQL keywords on their own line?	Yes
Indent size?	4
Indent batch blocks?	No
Number of new lines to insert	1
Indent Size	5
Right Margin?	80
Stacked parentheses when they contain multiple items?	Yes
Stack parentheses when list contains ____ or more items.	2
Indent Size?	2
New line after first parentheses?	Yes
Indent content of conditional and looping constructs?	--
Number of new lines to insert?	1
Indent size?	5



The screenshot shows a SQL Editor window with a single file named *File.sql. The query is as follows:

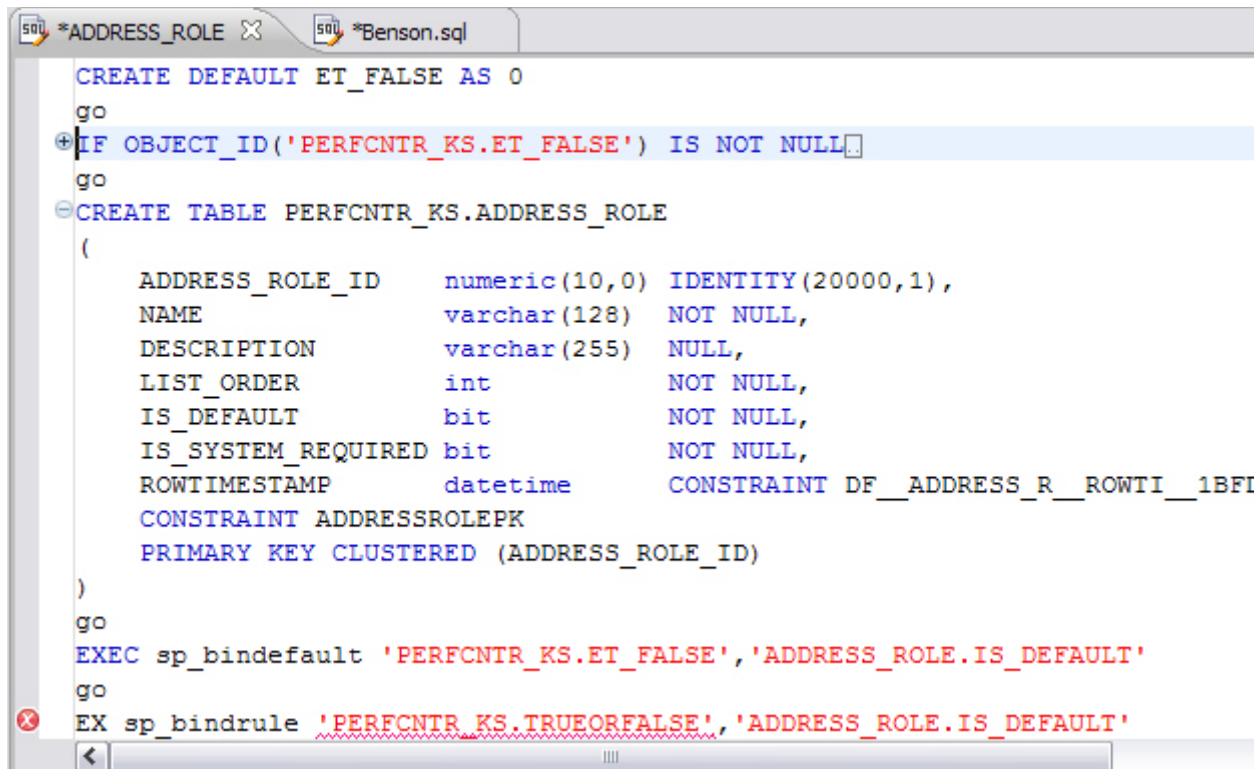
```
Begin

If x=5

    SELECT
        apple ,
        pear ,
        orange 'Big Orange' ,
        strawberry ,
        orchard_name ,
        owner
    FROM
        fruit F ,
        orchard O
    WHERE
        fruit_region in (
                                'latin america' ,
                                'france' ,
                                'russia' ,
                                'canada' ,
```

UNDERSTANDING CODE FOLDING

SQL Editor features code folding that automatically sorts code into an outline-like structure within the editor window for easy navigation and clarity while developing code.



The editor window automatically inserts collapsible nodes in the appropriate lines of code for organizational purposes. This enables you to expand and collapse statements, as needed, while developing code in particularly large or complicated files.

UNDERSTANDING CODE QUALITY CHECKS

Code quality markers provide annotations that prevent and fix common mistakes in the code.

These notes appear in a window on any line of code where the editor detects an error, and are activated by clicking the light bulb icon in the margin or by pressing Ctrl + I.

For example, if a statement reads `select * from SCOTT.EMP, SCOTT.DEPT`, when you click the light bulb icon or press Ctrl + I, a window appears beneath the line of code that suggests Add join criteria.

When you click on a proposed fix, the statement is automatically updated to reflect your change.

The following common errors are detected by the code quality check function in the editor:

Code Quality Check Type	Definition
Statement is missing valid JOIN criteria	<p>If a SELECT statement contains missing join criteria, when it is executed, it can produce a Cartesian product between the rows in the referenced tables. This can be problematic because the statement will return a large number of rows without returning the proper results.</p> <p>The code quality check detects missing join criteria between tables in a statement and suggests join conditions based on existing foreign keys, indexes, and column name/type compatibility.</p> <p>Example</p> <p>The following statement is missing a valid JOIN criteria:</p> <pre>SELECT * FROM employee e, customer c, sales_order s WHERE e.employee_id = c.salesperson_id</pre> <p>The code quality check fixes the above statement by adding an AND clause:</p> <pre>SELECT * FROM employee e, customer c, sales_order s WHERE e.employee_id = c.salesperson_id AND s.customer_id = c.customer_id</pre> <p>Note: This code quality check is valid for Oracle, DB2, and Sybase-specific join conditions.</p>
Invalid or missing outer join operator	<p>When an invalid outer join operator exists in a SELECT statement, (or the outer join operator is missing altogether), the statement can return incorrect results.</p> <p>The code quality check detects invalid or missing join operators in the code and suggests fixes with regards to using the proper join operators.</p> <p>Example</p> <p>The following statement is missing an outer join operator:</p> <pre>SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id (+) AND c.state = 'CA'</pre> <p>The code quality check fixes the above statement by providing the missing outer join operator to the statement:</p> <pre>SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id(+) AND c.state(+) = 'CA'</pre>

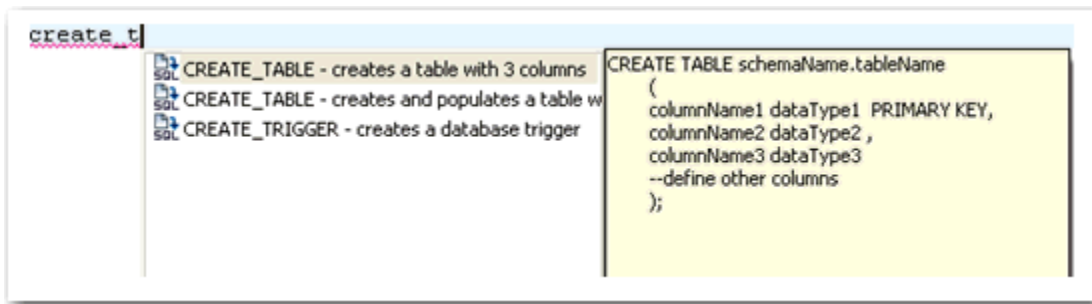
Code Quality Check Type	Definition
Transitivity issues	<p>The performance of statements can sometimes be improved by adding join criteria, even if a join is fully defined. If this alternate join criteria is missing in a statement, it can restrict the selection of an index in Oracle's optimizer and cause performance problems.</p> <p>The code quality check detects possible join conditions by analyzing the existing conditions in a statement and calculating the missing, alternative join criteria.</p> <p>Example</p> <p>The following statement contains a transitivity issue with an index problem:</p> <pre>SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id</pre> <p>The code quality check fixes the above statement with a transitivity issue by adding the missing join condition:</p> <pre>SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id AND i.product_id = pr.product_id</pre>
Nested query in WHERE clause	<p>It is considered bad format to place sub-queries in the WHERE clause of a statement, and such clauses can typically be corrected by moving the sub-query to the FROM clause instead, which preserves the meaning of the statement while providing more efficient code.</p> <p>The code quality check fixes the placement of sub-queries in a statement, which can affect performance. It detects the possibility of moving sub-queries from the FROM clause of the statement.</p> <p>Example</p> <p>The following statement contains a sub-query that contains an incorrect placement of a WHERE statement:</p> <pre>SELECT * FROM employee WHERE employee_id = (SELECT MAX(salary) FROM employee)</pre> <p>The code quality check fixes the above statement by correcting the sub-query issue:</p> <pre>SELECT employee.* FROM employee (SELECT DISTINCT MAX(salary) col1 FROM employee) t1 WHERE employee_id = t1.col1</pre>

Code Quality Check Type	Definition
Wrong place for conditions in a HAVING clause	<p>When utilizing the HAVING clause in a statement</p> <p>It is recommended to include as few conditions as possible while utilizing the HAVING clause in a statement. DB Optimizer™ XE and DB Optimizer™ detects all conditions in a given HAVING statement and suggests equivalent expressions that can benefit from existing indexes.</p> <p>Example</p> <p>The following statement contains a HAVING clause that is in the wrong place:</p> <pre>SELECT col_a, SUM(col_b) FROM table_a GROUP BY col_a HAVING col_a > 100</pre> <p>The code check fixes the above statement by replacing the HAVING clause with equivalent expressions:</p> <pre>SELECT col_a, SUM(col_b) FROM table_a WHERE col_a > 100 GROUP BY col_a</pre>
Index suppressed by a function or an arithmetic operator	<p>In a SELECT statement, if an arithmetic operator is used on an indexed column in the WHERE clause, the operator can suppress the index and result in a FULL TABLE SCAN that can hinder performance.</p> <p>The code quality check detects these conditions and suggests equivalent expressions that benefit from existing indexes.</p> <p>Example</p> <p>The following statement includes an indexed column as part of an arithmetic operator:</p> <pre>SELECT * FROM employee WHERE 1 = employee_id - 5</pre> <p>The code quality check fixes the above statement by reconstructing the WHERE clause:</p> <pre>SELECT * FROM employee WHERE 6 = employee_id</pre>
Mismatched or incompatible column types	<p>When the data types of join or parameter declaration columns are mismatched, the optimizer is limited in its ability to consider all indexes. This can cause a query to be less efficient as the system might select the wrong index or perform a table scan, which affects performance.</p> <p>The code quality check flags mismatched or incompatible column types and warns that it is not valid code.</p> <p>Example</p> <p>Consider the following statement if Table A contains the column col int and Table B contains the column col 2 varchar(3):</p> <pre>SELECT * FROM a, b WHERE a.col = b.col;</pre> <p>In the above scenario, the code quality check flags the 'a.col = b.col' part of the statement and warns that it is not valid code.</p>

Code Quality Check Type	Definition
Null column comparison	<p>When comparing a column with NULL, the !=NULL condition may return a result that is different from the intended command, because col=NULL will always return a result of false. Instead, the NULL/IS NOT NULL operators should be used in its place.</p> <p>The code quality check flags occurrences of the !=NULL condition and replaces them with the IS NULL operator.</p> <p>Example</p> <p>The following statement includes an incorrect col = NULL expression:</p> <pre>SELECT * FROM employee WHERE manager_id = NULL</pre> <p>The code quality check replaces the incorrect expression with an IS NULL clause:</p> <pre>SELECT * FROM employee WHERE manager_id IS NULL</pre>

UNDERSTANDING SQL TEMPLATES

DB Optimizer provides code templates for DML and DDL statements that can be applied to the Editor via the (Ctrl + Spacebar) command. When you choose a template from the menu that appears, SQL Editor automatically inserts a block of code with placeholder symbols that you can modify to customize the code for your own purposes.

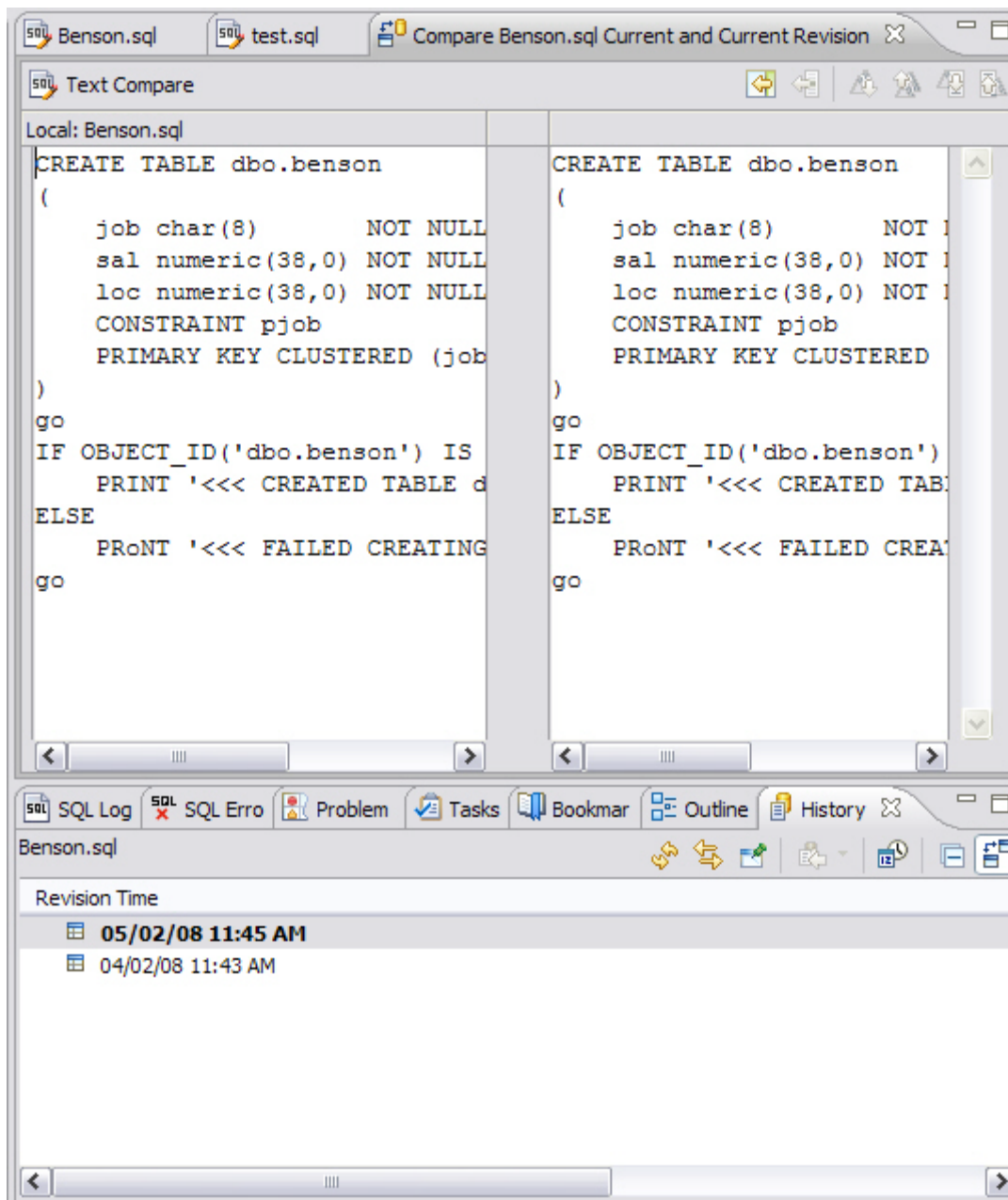


Code templates are available for DML, ALTER, DROP, CREATE, and platform specific commands.

There are 267 templates available for all supported platforms and respective versions. You can modify and create new templates via the SQL Templates panel on the Preferences dialog. See "[Set SQL Code Template Preferences](#)" on page 20 for more information on how to create and alter SQL code templates.

VIEW CHANGE HISTORY

Each time an SQL file is saved, the local history of that file is recorded (changes made). Using the Local History command, you can view all changes made to the file. Local History is accessed via the shortcut menu of SQL Editor and selecting **Compare With > Local History**.



- The History view displays all recorded times the file was changed since its inception/introduction into the workspace.
- Double-click a time in the **History** view to access the **Text Compare** panel. It displays the text of the file after the change occurred at the time indicated in the **History** view.

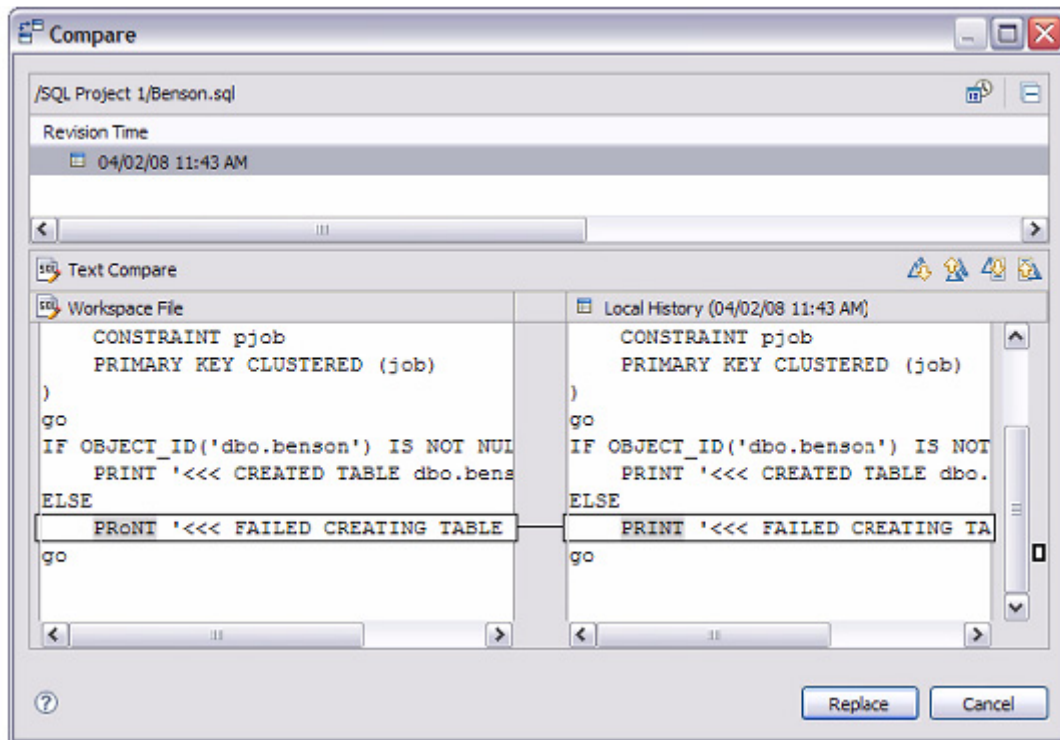
REVERT TO AN OLD VERSION OF A FILE

The **Replace With > Local History** command provides you with the ability to revert a SQL file back to a previously recorded local history.

To replace the contents of a file with the contents of a previously saved version via local history:

- 1 Right-click the SQL Editor and select **Replace With > Local History** from the shortcut menu.

The **Replace from Local History** dialog appears.



- 2 In the **Local History of ...** panel, select a previously recorded version of the file by clicking the appropriate timestamp.
- 3 Click **Replace**.

The contents of the currently-opened file revert to the contents of the file at the history point you selected in the dialog.

Alternatively, from the shortcut menu, select **Replace With > Previous from Local History** to replace the contents of the file with DB Optimizer™ XE and DB Optimizer™'s last recorded history point.

DELETE AN SQL FILE

To delete a file, right-click its icon in the SQL Project Explorer and select **Delete**. This will remove the file from both the SQL project and the file system.

EXECUTING SQL FILES

DB Optimizer™ XE and DB Optimizer™ can execute SQL code directly on registered data sources.

Files are executed via the Execute SQL command in the Run menu, or by clicking the green arrow button on the toolbar.

When an SQL file is open in the Workspace, select it and choose a database and an associated catalog on which you want to execute the file via the lists in the Toolbar.

You can click the execute icon to execute code on the specified database and catalog, start a transaction or commit a transaction, or modify SQL session options prior to execution.

To execute a file:

Open the SQL file you want to run, ensure it is associated with the correct database, and click **Execute**. DB Optimizer™ XE and DB Optimizer™ executes the code on the data source you specified. Results are displayed in the Results view and can be exported into a file via the Data Export wizard, or displayed in multiple file formats (HTML, XML, and TXT formats).

To execute a transaction:

To execute transactions, you need to ensure that the auto commit feature is turned off. See "[Set SQL Execution Preferences](#)" on page 18 for more information on how to turn off auto commit.

Open the transaction file you want to run, ensure it is associated with the correct database, and click Start Transaction. DB Optimizer™ XE and DB Optimizer™ executes the transaction on the data source you specified.

Once the transaction runs, you can execute the file as normal.

NOTE: Click **Commit** or **Rollback** to finish or cancel a transaction.

To commit a transaction:

Open the transaction file you want to commit, ensure it is associated with the correct database, and click **Commit Transaction**. DB Optimizer™ XE and DB Optimizer™ commits the transaction on the data source you specified.

TIP: You can set transactions to auto-commit prior to execution on the SQL Execution node of the **Preferences** panel.

See Also:

- [Associate an SQL File with a Data Source](#) on page 59
- [Configure an SQL Session](#) on page 60
- [Execute SQL Code](#) on page 61

- [View and Save Results](#) on page 61

ASSOCIATE AN SQL FILE WITH A DATA SOURCE

When working with files, SQL Editor enables developers to view and change the data source to which they are connected.

The bread crumb line in SQL editor is used to display and specify a data source in relation to the specified SQL Editor file. The menu contains a list of all registered data sources. Additionally, on platforms that support catalogs, these are displayed as well.



Changing a catalog via the drop down lists is the equivalent of issuing a USE DATABASE command on SQL Server, Sybase, and MySQL platforms. Any change will not affect the current connection, and the list automatically updates to display the name of the newly selected data source.

If no registered database is associated with a SQL file (as would be the case if a user started a new, unsaved file), the list is empty. This indicates that the file is not connected to a registered data source.

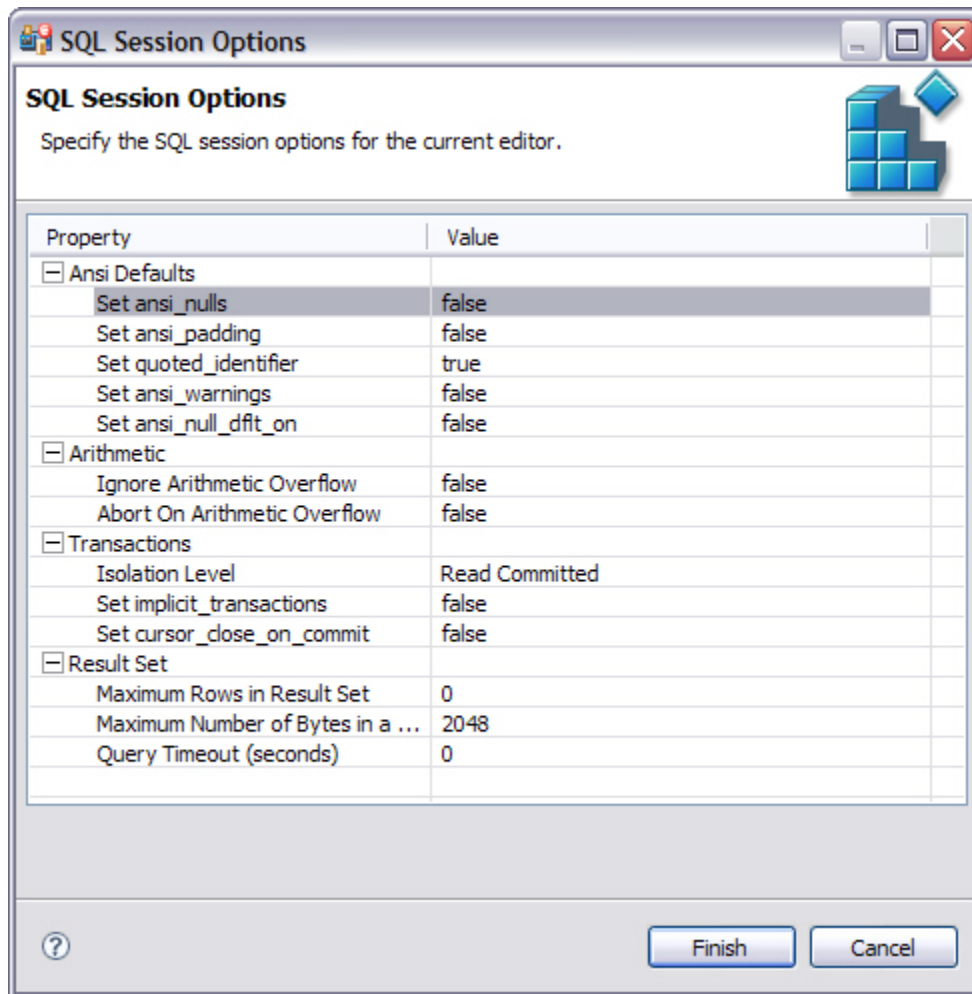
To change or associate a registered data source with a SQL file:

Click the database list and select the name of a registered database from the list provided. Depending on the state of the code in SQL Editor, DB Optimizer™ XE and DB Optimizer™'s behavior differs when the connection is made:

TIP: If you are receiving multiple syntax errors, always check that the file is associated with the correct data source and corresponding database/catalog before troubleshooting further.

CONFIGURE AN SQL SESSION

The SQL Session Options dialog provides configuration parameters that indicate to DB Optimizer™ XE and DB Optimizer™ how to execute code in the development environment.



To modify SQL session options:

- 1 Click the SQL Session Options icon in the Toolbar.

The **SQL Session Options** dialog appears.

- 2 Click on individual parameters in the Value column to change the configuration of each property, as specified.
- 3 Click **Finish**.

The session options will be changed and DB Optimizer™ XE and DB Optimizer™ will execute the code as specified when you execute it.

Session options only apply to the corresponding editor and are not retained when executing multiple SQL files.

EXECUTE SQL CODE

Files can be launched from within the DB Optimizer™ XE and DB Optimizer™ development environment for execution on a registered data source. Files are executed via the commands in the Run menu.

When a SQL file is open in the Workspace, select it and choose a database and an associated catalog on which you want to execute file using the drop down menus in the Toolbar. You can click the execute icon to execute the code on the specified database and catalog, start a transaction or commit a transaction, or modify the SQL session options prior to execution.

To execute code:

Open the SQL file you want to run, ensure it is associated with the correct database and click the Execute icon. DB Optimizer™ XE and DB Optimizer™ executes the code on the data source you specified. Results are displayed in the same tab or in a new tab.

To execute a transaction:

Open the transaction file you want to run and ensure it is associated with the correct database, and then click the Start Transaction icon. DB Optimizer™ XE and DB Optimizer™ executes the transaction on the data source you specified.

To commit a transaction:

Open the transaction file you want to commit, ensure it is associated with the correct database, and then click the Commit Transaction icon. DB Optimizer™ XE and DB Optimizer™ commits the transaction on the data source you specified.

TIP: You can set transactions to auto-commit prior to execution on the SQL Execution node of the Preferences panel in DB Optimizer™ XE and DB Optimizer™.

VIEW AND SAVE RESULTS

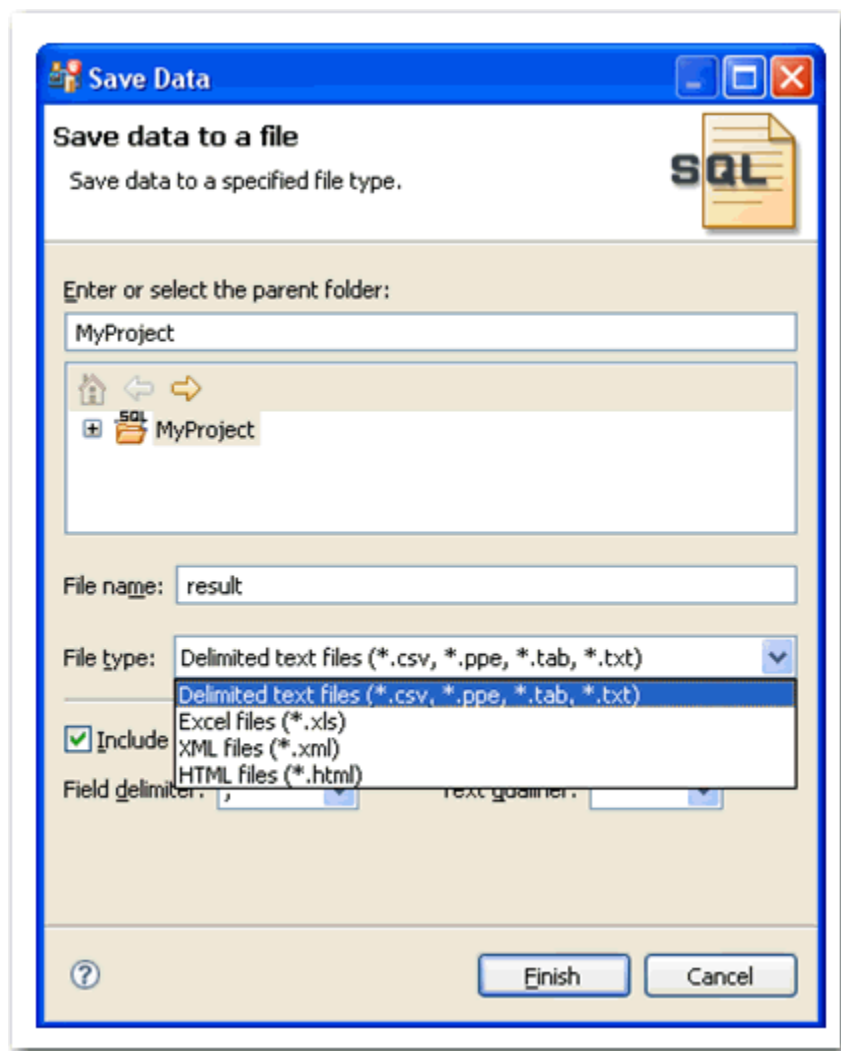
Once a file has been executed, the results are displayed in the Results view. Here, you can examine the outcome of the execution process, as well as save the results in other file formats, as needed.

You can view results in the following formats:

- HTML
- XML
- TXT

To save results:

- 1 Right-click on the **Results** view and select **Save Data**. The **Save Data** dialog appears.

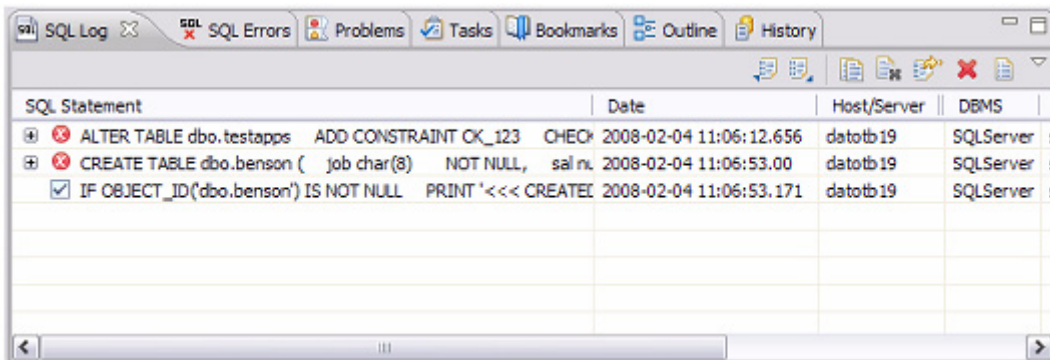


- 2 Select the project name to which you want to save the results, enter a file name, choose the file parameters, and then choose a file format from the drop down menu. You can select delimited text file, Excel, XML, or HTML file formats.
- 3 Click **Finish**. The results are saved in the directory location and format that you specified.

TROUBLESHOOTING

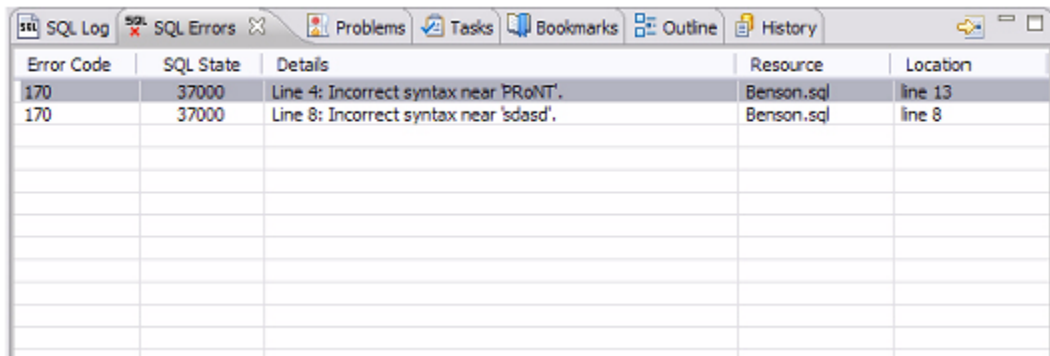
DB Optimizer™ XE and DB Optimizer™ contains a number of views used exclusively to log and monitor the SQL development process.

- The **SQL Log** captures all SQL commands executed by SQL Editor and the system. SQL Log entries are listed by SQL Statement name, Date issued, Host/Server, Service, User, Source, and the Time (in milliseconds) it took to execute the command.



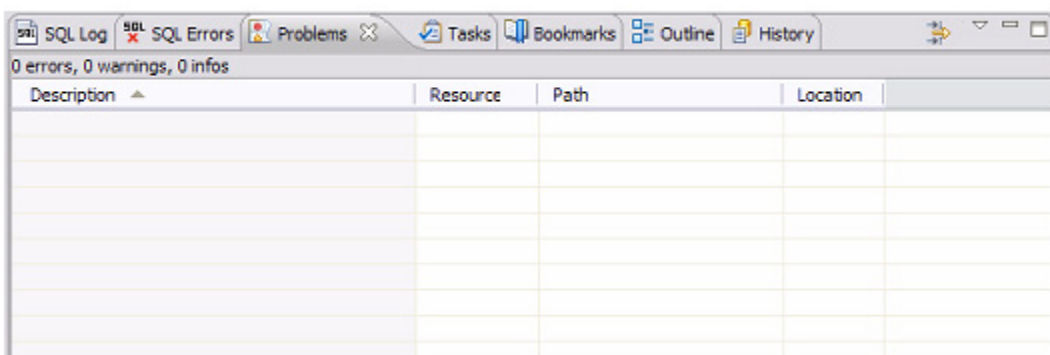
SQL Statement	Date	Host/Server	DBMS
ALTER TABLE dbo.testapps ADD CONSTRAINT CK_123 CHECK	2008-02-04 11:06:12.656	datotb19	SQLServer
CREATE TABLE dbo.benson (job char(8) NOT NULL, sal n	2008-02-04 11:06:53.00	datotb19	SQLServer
IF OBJECT_ID('dbo.benson') IS NOT NULL PRINT '<<< CREATE	2008-02-04 11:06:53.171	datotb19	SQLServer

- The **SQL Errors** log automatically logs all SQL errors encountered when SQL commands are executed through DB Optimizer™ XE and DB Optimizer™. Errors are listed by Error Code, SQL State, error Details, Resource, and the Location of the error in the SQL file.



Error Code	SQL State	Details	Resource	Location
170	37000	Line 4: Incorrect syntax near 'PRINT'.	Benson.sql	line 13
170	37000	Line 8: Incorrect syntax near 'sdasd'.	Benson.sql	line 8

- The **Problems** view captures syntactic and semantic errors and warnings in the files of the workspace. These entries typically take the form of error messages or warnings issued by the system over the course of a procedure execution. Problems are organized by Description (which indicates the type of problem logged), Resource, file Path, and Location. Using the Problems view, you can apply quick fixes to issues that DB Optimizer detects, as well as locate other problems that have similar attributes.



Description	Resource	Path	Location
0 errors, 0 warnings, 0 infos			

See Also:

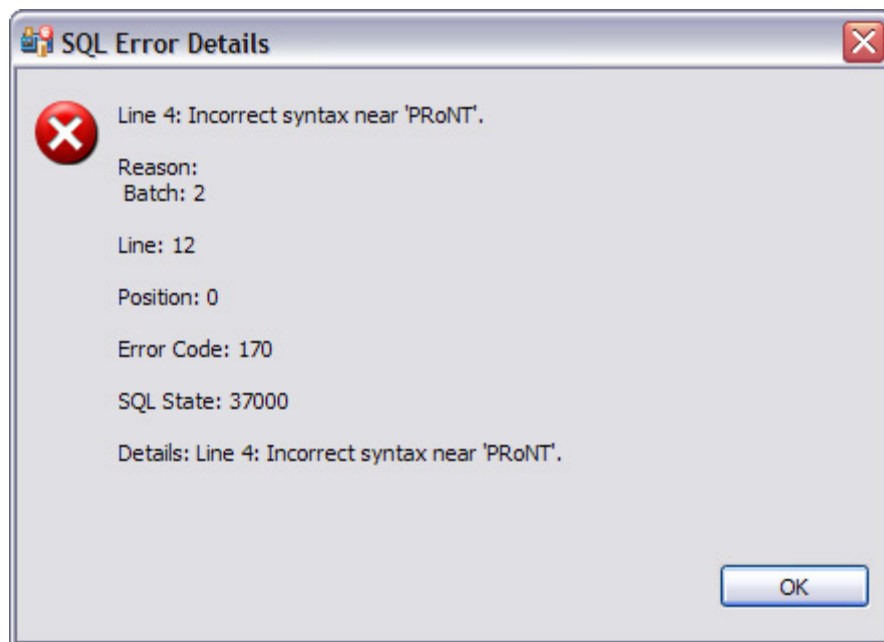
- [View Log Details](#) on page 64
- [Maintain Logs](#) on page 65
- [Filter Logs](#) on page 65
- [Import and Export Error Logs](#) on page 68
- [Find and Fix SQL Code Errors](#) on page 69
- [Find and Fix Other Problems](#) on page 69

VIEW LOG DETAILS

The SQL Error Log and Problems views contain functionality that enable you to view details regarding individual log entries, and in some cases, locate or fix those issues automatically.

To view details about SQL Errors entries:

Right-click the error whose details you want to view and select SQL Error Details.



The SQL Error Details dialog provides information about the specified SQL error.

Additionally, you can double-click the error to view the problem code in SQL Editor.

To view details about Problems

Right-click the entry whose details you want to view and select **Properties**. The **Properties** dialog appears, summarizing the issue.

MAINTAIN LOGS

The SQL Log and SQL Errors views both contain commands that enable you to save, restore, or otherwise move log entries into files outside of DB Optimizer™ XE and DB Optimizer™. Additionally, both views also contain commands that enable the clearing of the view.

The current editor option will only show users statements as generated by the active editor.

To maintain log entries:

All entries automatically captured by the Error Log are written to a file (.log suffix) that resides in the Workspace .metadata folder.

- From DB Optimizer™ XE and DB Optimizer™, right-click in the SQL Log and select **Clear Log Viewer** to remove all messages.
- In the shortcut menu, select **Delete Log** to delete the .log file. If entries are created after the Delete Log command is issued, DB Optimizer™ XE and DB Optimizer™ will automatically generate a new .log file in the .metadata subfolder.

NOTE: Old Error Log entries cannot be recovered once the .log file is deleted. To prevent data loss, archive the .log file via the Export command prior to deletion.

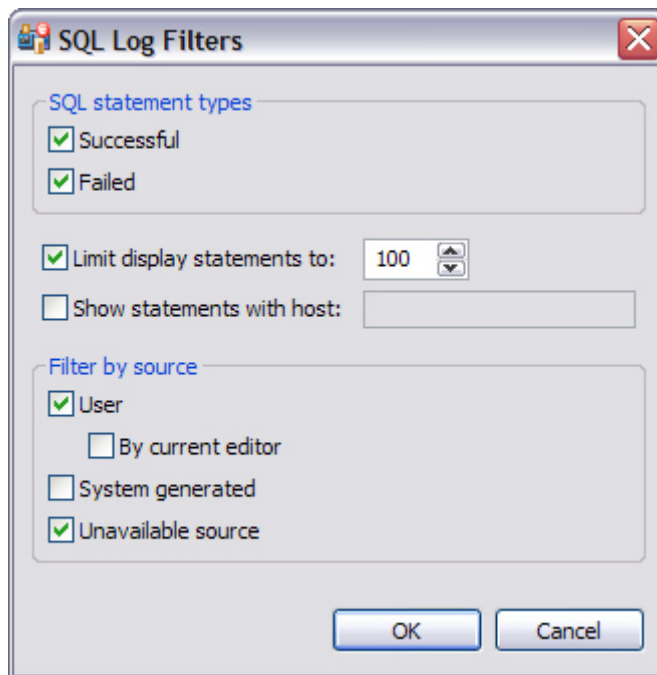
- To clear the Error Log view without deleting the .log file, select **Clear Log Viewer** from the shortcut menu. The View will be cleared of entries, but these entries will still be contained in the .log file.
- To restore the **Error Log** view based on the entries contained in the .log file, select **Restore Log** from the shortcut menu. The View is restored based on the entries in the .log file.

FILTER LOGS

Filters can be applied to Problems, SQL Log, and the SQL Error Log to limit searches when troubleshooting and pinpointing specific processes within the system.

To filter the SQL Log:

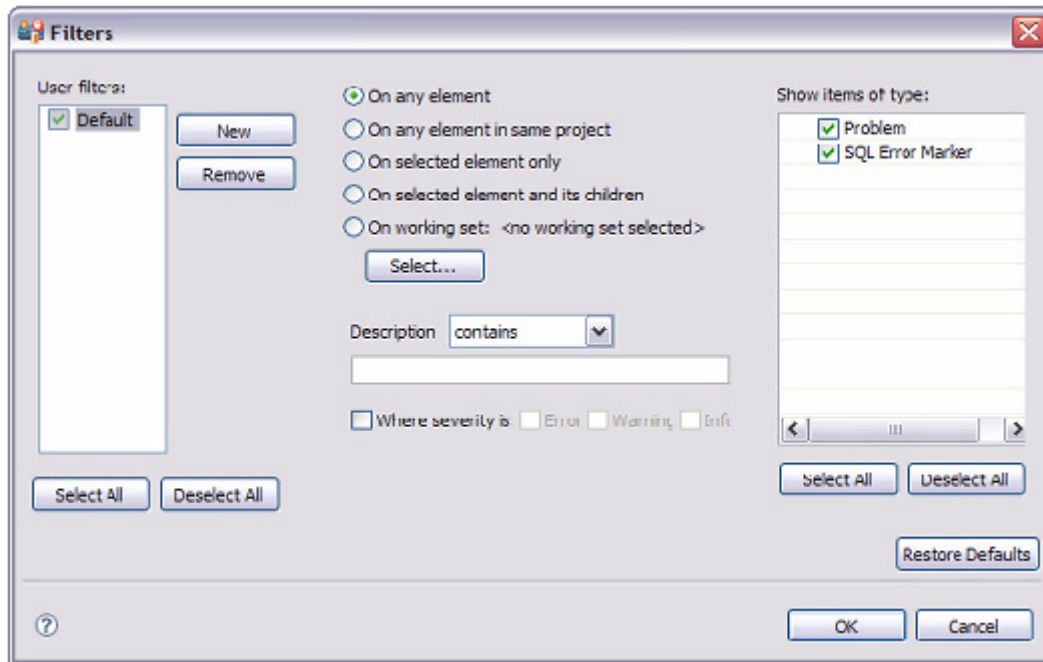
- Select the Toolbar Menu icon (the downward-pointing arrow in the right-hand corner of the view) and choose **Filters**. The **SQL Log Filters** dialog appears.



- In the SQL Statement Types frame, select **Successful** or **Failed** to filter by the type of Error Log entries.
- Select **Limit display statements** to indicate a maximum limit of the number of entries displayed in the Error Log, and enter the maximum entry value in the corresponding field.
- Select **Show statements with host** to indicate that only entries from a specific data source are to be displayed, then type the name of the data source (as it appears in the Database Explorer) in the corresponding field.
- In the **Filter by Source** pane, specify **User**, **System Generated** or **Unavailable Source** to filter statements by the type of source from where they originated.

To filter the Problems log:

Select the Toolbar Menu icon and choose **Configure Filters**. The **Filters** dialog appears.



The Filters dialog enables the creation of multiple filter profiles that can be applied to the log via the Toolbar Menu. The User Filters panel on the left-hand side of the dialog displays all existing filter profiles stored in the Workspace. Initially, the Workspace only contains the Default filter profile. Selecting it displays its filter parameters, and selecting the check box associated with its name enables the filter in the Problems view (only problems that match the criteria defined in the Filters dialog will appear in the view).

The ability to define different profiles enables the selection of multiple filter profiles. For each profile selected, the profile criteria is applied to the View, concurrently. You can filter problems by:

- Working Set
- Character String
- Problem Severity
- Problem Type
- A combination of the above four filter options

Profile Criteria	Description
Working Set	<p>The options located in the center of the dialog enable you to filter problems based on defined Working Sets. A Working Set is a collection of user-defined Project files that you can organize, as needed, in DB Optimizer™ XE and DB Optimizer™. Select an option, and then click Select to define a Working Set to which the parameters apply. If no Working Sets exist, you need to define one or more via the New button on the Select Working Set dialog.</p> <p>Select one or more Working Sets to which you want the criteria to apply. If no Working Sets exist, or none suitably match the current filter criteria, click New or Edit to define a new Working Set, or edit an exist Working Set, respectively.</p>
Character String	Use the Description list to select contains or doesn't contain, as needed, and type the character string in the field below the list. The Problems view is filtered to only contain, or omit, problem descriptions that fully or partially match the string value.
Problem Severity	Select the Where severity is check box and choose Error , Warning , Info or some combination of the three check boxes. Only entries whose severity matches the check boxes you have selected remain visible in the Problems view.
Problem Type	The options in the Show items of type list on the right-hand side of the dialog enable you to filter problems by type. Deselect Problem to remove any system entries from the view, or deselect SQL Error Marker to remove any SQL code entries from the view.

Once you have defined and/or selected the appropriate filter profiles, the profiles will appear in the **Filters** submenu in the Toolbar Menu of the Problems view. Select or deselect the profiles from the submenu, as needed.

IMPORT AND EXPORT ERROR LOGS

Error messages are written to a file named .log located in the Workspace directory .metadata folder. This file can (and should) be cleared periodically via the Delete Log command to prevent performance issues with regards to system memory and file size. However, the Export command enables you to archive log files prior to deletion. The files created by the Export command can then be imported back into the Error Log as needed at a later point in time.

To export the SQL Log:

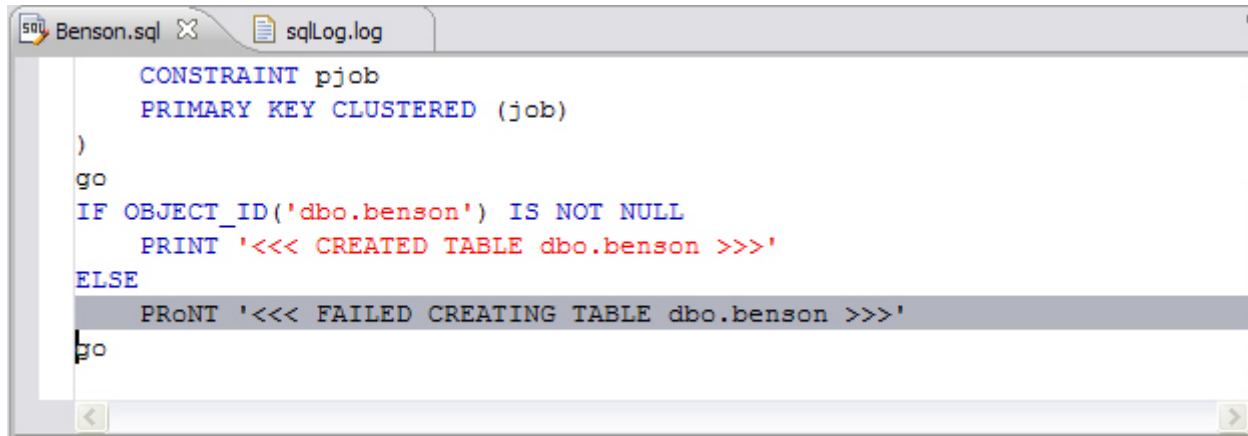
Right-click the SQL Log view and choose **Export Log**. The log is saved in the specified directory path with a .log extension.

To import the Error Log:

Right-click the SQL Log view and choose **Import Log**. Select the previously exported .log file. The Error Log view is restored with the entries from the specified export file.

FIND AND FIX SQL CODE ERRORS

The SQL Errors view contains an option that enables you to navigate directly to the resource associated with an error entry.



To navigate to the source of a SQL error entry:

Right-click the entry to which you want to navigate and select **Go To**. The file to which the error applies automatically opens in a new instance of SQL Editor, and the line is highlighted in the window.

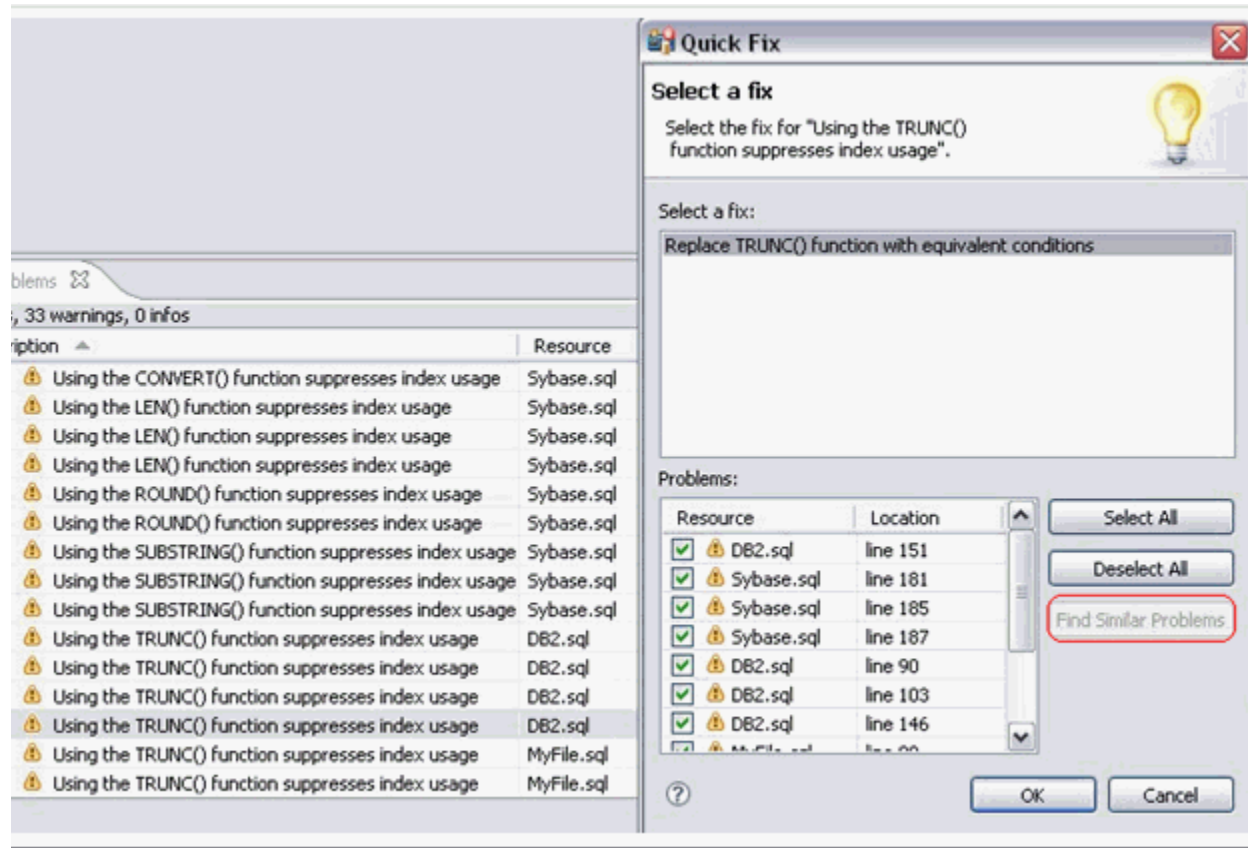
FIND AND FIX OTHER PROBLEMS

By default, the Problems view organizes problems by severity. You can also group problems by type, or leave them ungrouped.

The first column of the Problems view displays an icon that denotes the type of line item, the category, and the description. Click the problem and DB Optimizer™ XE and DB Optimizer™ will open the SQL file and automatically highlight the line that triggered the issue.

You can filter Problems to view only warnings and errors associated with a particular resource or group of resources. You can add multiple filters to the view, as well as enable/disable them as required. Filters are additive, so any problem that satisfies at least one of the filters will appear.

Problems can sometimes be fixed via the Quick Fix command in the shortcut menu. The **Quick Fix** dialog enables you to apply a fix to a problem detected by the view. The dialog also provides a list of similar problems to the one you selected, and enables you to apply a fix to multiple problems at the same time.



To apply a quick fix to an issue in the Problem view:

- 1 Right-click on a problem in the list and select **Quick Fix** from the menu. The **Quick Fix** dialog appears.
- 2 Select a fix from the list provided and click **OK**. DB Optimizer attempts to resolve the issue.

To find similar issues:

- 1 In the **Quick Fix** dialog, click **Find Similar Problems**. The Problems list populates with all of the issues that are similar to your initial selection.
- 2 Use the check boxes beside the problems to select them, and then choose a fix and click **OK**. DB Optimizer attempts to resolve all of the specified issues.

USING PROFILING

For details on working with profiling, see the following topics:

- [Understanding Profiling](#) on page 71
- [Understanding the Interface](#) on page 72
- [Running a Profiling Session](#) on page 73
- [Configuring Profiling](#) on page 103
- [Using Load Tester](#) on page 111

UNDERSTANDING PROFILING

Profiling continuously samples the data source to build a statistical model of the load on the database. Profiling can be used to locate and diagnose problematic SQL code and event-based bottlenecks. Additionally, profiling enables you to investigate execution and wait time event details for individual stored routines. Results are presented in the profiling editor, which enables users to identify problem areas and subsequently drill down to individual, problematic SQL statements.

Profiling filters out well performing light weight SQL and collects information on heavy weight SQL. SQL that is heavy weight are either long running queries or queries that are short but run so often that they put load on the database

Profiler takes snapshots of user and session activity once a second and builds up a statistical model of the load on the database. The sampled data is displayed in three ways:

- Load on the database measured in average number of sessions active
- Top Activity - Top SQL, Event, and Sessions, Object I/O, and Procedures
- Details - Detail on a SQL statement, Session, Event, Object I/O or Procedure

The graph on the top of the screen shows the load on the database and can quickly indicate visually how the database is functioning. The database could be

- Idle
- Lightly loaded
- Heavily loaded
- Bottlenecked

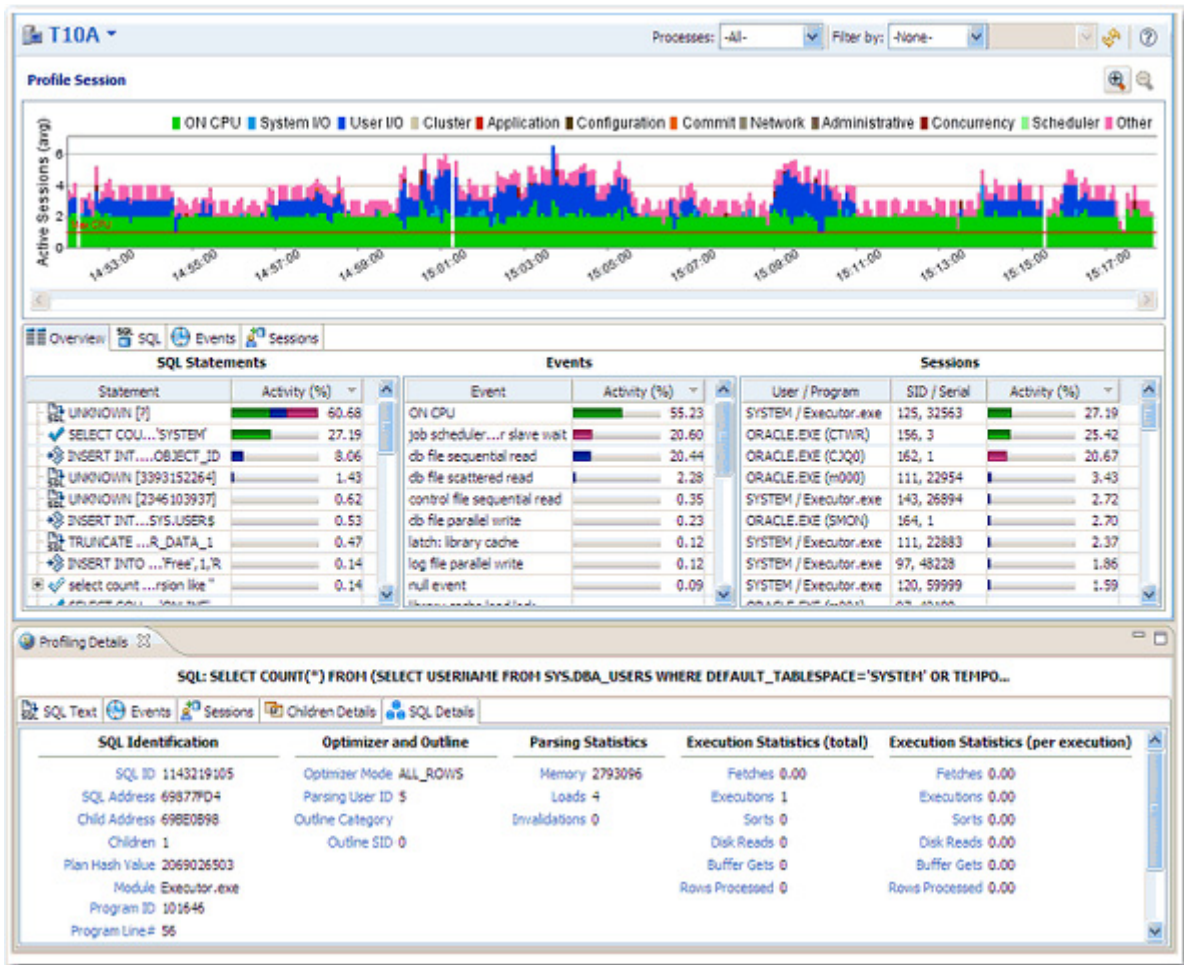
Problems can come from any one or more of four areas

- Machine CPU or Engine, slow disks (network)
- Application locks, invalid SQL

- Database cache sizes, log files, etc
- Inefficient SQL

UNDERSTANDING THE INTERFACE

The profiling interface is divided into three major parts:



- The **Load Graph** is located on the top section of the editor and provides a display of the overall load on the system. The bars represent individual aspects of the enterprise, and the view can be used to find bottlenecks.
- **Top Activity** is located on the middle section of the editor and displays where the load originates. Specifically, the top SQL statements, top events that the database spends time in, as well as the top activity sessions.
- The **Profiling Details View** is located on the bottom section of the editor and displays detailed information on any item selected in the middle section. For example, an SQL statement, an Event, or a Session.

The graphical portion of the profiling editor presents the distribution of sessions executed over the length of the profiling process, and those that were waiting in DBMS-specific events. It provides a first and most important step in identifying problem areas. Results can be viewed in real-time.

The Load Graph and Top Activity Section compose one view in the editor, while the Profiling Details view is a separate interface component that only activates when an item in the Top Activity Section is specified.

NOTE: Use a 1280 x 1024 monitor resolution when viewing profiling information. Smaller resolution sizes can obscure details in the view.

RUNNING A PROFILING SESSION

Profiling provides the continuous monitoring of a data source and builds a statistical model based of database load based on the users state every second. The created profile can then be saved to file, and the data can be saved, analyzed, and optimized by importing statements into the tuning component and running a tuning job.

The following list provides the general workflow and overhead tasks, when attempting to monitor data sources and store query information.

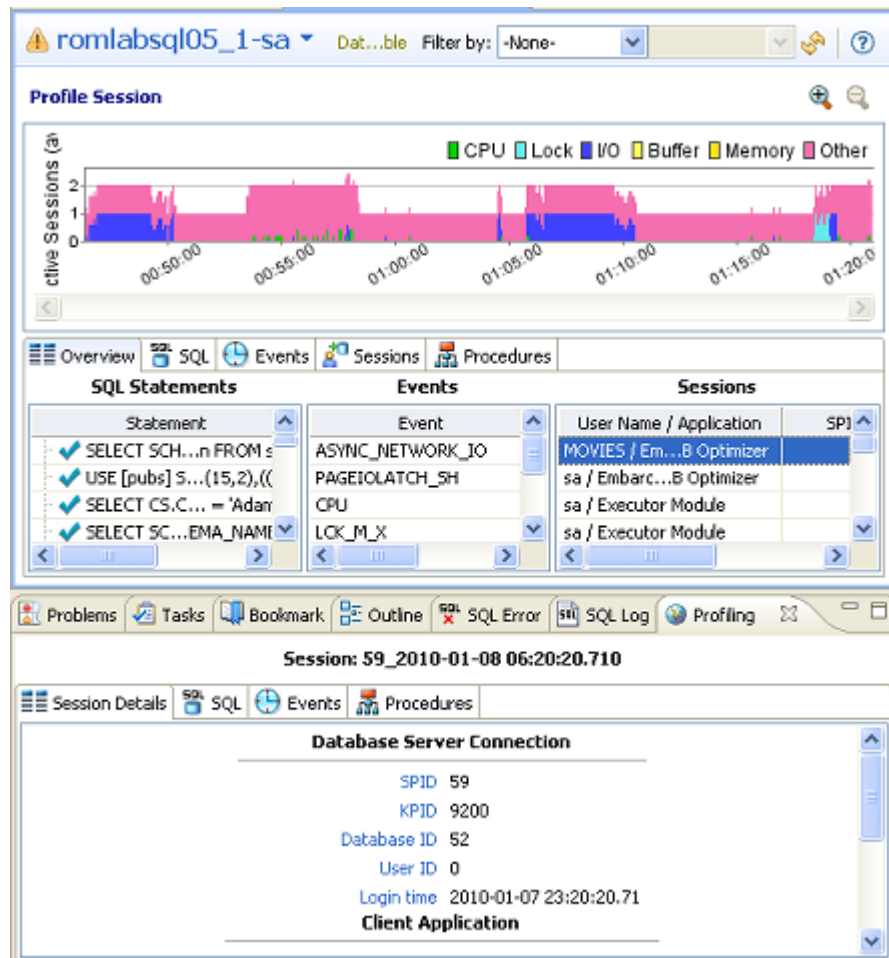
- 1 [Execute a Profiling Session](#) on page 74
- 2 [Work with Session Results](#) on page 75
- 3 [Saving Profiling Sessions](#) on page 99
- 4 [Import Statements to Tuning](#) on page 101

In addition to the workflow tasks outlined above, the profiling interface also enables a number of important functions to help in statement analysis and diagnosis. This additional, or extra, functionality can be found in "[Other Profiling Commands](#)" on page 101

Furthermore, in some cases you will need to configure system variables and parameters in order to get the results you need from the application. See "[Configuring DBMS Properties and Permissions](#)" on page 103 for more information on how to configure profiling and your registered data sources prior to running a session.

EXECUTE A PROFILING SESSION

Profiling is monitored and managed via profiling's three major interface components: the Load Chart, Top Activity Section, and Profiling Details view.



To execute a profiling session:

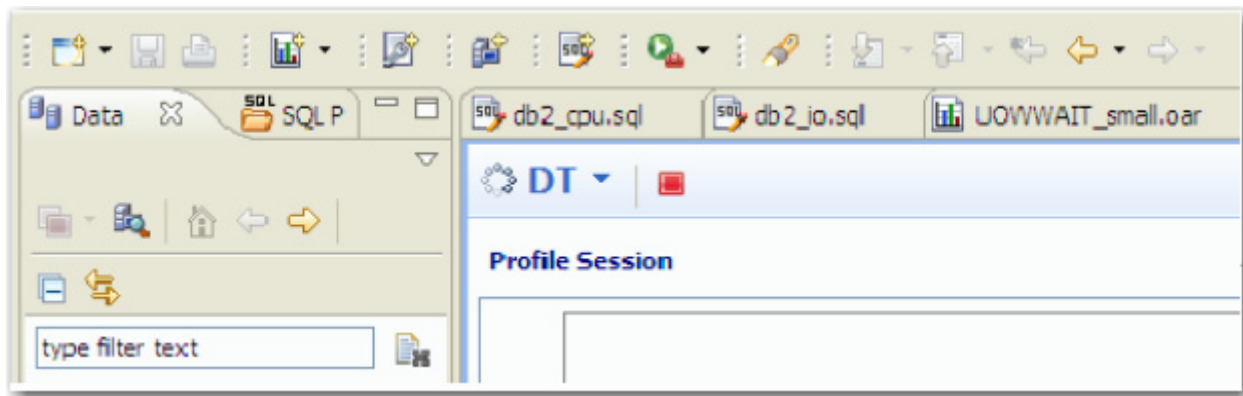
- 1 In Data Source Explorer, right click on the data source you want to profile and select **Profile As** from the menu, and then choose **Data Source 1**.
- 2 In the **Profiling Session Launch Configuration** dialog, select the configuration to use for this profiling session.

The profiling session begins. Alternatively, clicking the Profiling icon on the Toolbar automatically runs a profiling session for the last data source you selected.

Once a profiling session launches, it runs until you stop it. When a session has run for a length of time, you can then interpret and analyze the results. See "[Work with Session Results](#)" on page 75.

To stop a profiling session:

You can stop a profiling session at any time by clicking the **Stop** icon in the upper left-hand side of the Profile Session screen.

**Executing a Session from the Command Line**

You can launch a profiling session from the command line using the following syntax:

```
dboptimizer.exe profile ds:ROM*L*ABORCL10G_1 duration:20 tofile:c:\testprofile.oar
```

In the above command, the user has specified ROM*L*ABORCL10G_1 as the data source, and indicates a profiling session of 20 minutes. The tofile variable specifies the directory and name of the file to which the profiling session will be saved.

WORK WITH SESSION RESULTS

Results are displayed in the Profiling Session editor whenever a profiling session is executed. Results can appear in real time (if real time profiling is enabled) or once a session as finished its execution.

Results are displayed in the three major interface components of the editor, which you can use to analyze the overall efficiency and capacity of queries running on the data source, to various levels of detail:

The Profiling UI has three correlated sections:

- Selection in Chart will fill the top activity section data, distributed in Overview/SQL/Events/Sessions/Object I/O.
- Selection in any tab of Top Activity will fill the Profiling Details with top selection type related data.

For more information, see:

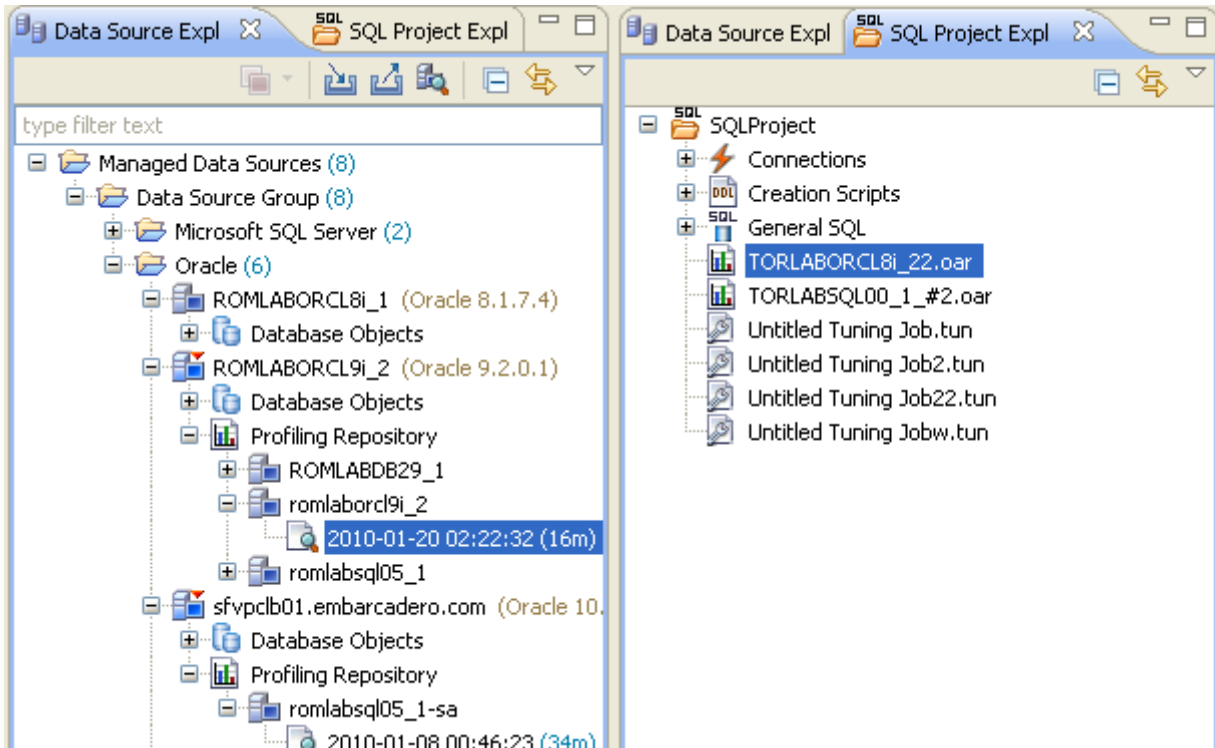
- [Opening an Existing Profiling Session](#) on page 76
- [Analyze the Load Chart](#) on page 76
- [Analyze the Top Activity Section](#) on page 78

- [Analyze Profiling Details](#) on page 84

OPENING AN EXISTING PROFILING SESSION

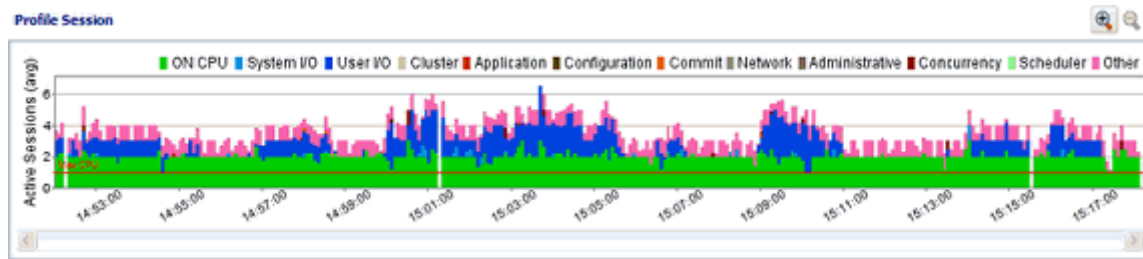
Saved profiling session data is stored in either an SQL Project or in a Profiling Repository on an Oracle data source.

To view a saved profiling session, locate it in either the **SQL Project Explorer** or in the **Data Source Explorer** and double-click the icon to open it in the Profiling Session window.



ANALYZE THE LOAD CHART

The Load Chart is located on the top section of the Profile Session editor and provides a display of the overall load on the system. The bars represent individual aspects of the enterprise, and the view is used to discover bottlenecks.



The most important part of the screenshot above is the Average Active Sessions (AAS) graph. AAS shows the performance of the database measured in the single powerful unified metric AAS. AAS easily and quickly shows any performance bottlenecks on the database when compared to the Max Engines (for Sybase) or Max CPU (for Oracle) line. The Max Engines line is a yardstick for performance on the database. When AAS is larger than the Max CPU line there is a bottleneck on the database. Bottleneck identification is that easy.

AAS or the average number of sessions active, shows how many sessions are active on average (over a 5 second range in DB Optimizer) and what the breakdown of their activity was. If all the users were running on CPU then the AAS bar is all green. If some users were running on CPU and some were doing I/O, represented by blue, then the AAS bar will be partly green and partly blue.

The line Max Engines or Max CPU represents the number of CPU processors on the machine. If we have one CPU/Engine then only one user can be running on the CPU/Engine at a time. If we have two CPUs/Engines then only two users can be on CPU at any instant in time. Of course users can go on and off the CPU/Engine extremely rapidly. When we talk about sessions on the Engines we are talking about the average number of sessions on CPU/Engine. A load of one session on the Engine, thus would be an average which could represent one user who is consistently on the CPU/Engine or many users who are on the CPU for short time slices. When the number of Engines becomes a resource bottleneck on the database we will the average active sessions in CPU/Engine state go over the Max Engine/Max CPU line. The number of sessions above the Max Engine line is the average number of sessions waiting for CPU/Engine.

The Max CPU is a yardstick for performance on the database. The number of CPUs or Engine on the data source is information DB Optimizer obtains during the profiling process. However, sometimes the number of CPUs or Engines is not reported. In these cases it might be desirable to change the default number of CPUs/Engines from one to a number more closely matching the actual system running the data source. You might also want to change the Max CPU/Engine line to reflect the performance impact of adding or removing a CPU or Engine from the system.

To change the Max CPU or Max Engine count in the Load Graph:

- 1 From the Profile Session window, right-click anywhere on the AAS graph and select **Edit Engine Count** or **Edit CPU Count**.
- 2 In the **Engine Count** dialog that appears select **Use a custom value**, enter a new value, and then click **OK**.

The AAS or Load Chart **Max CPU** or **Max Engine** line is updated immediately to reflect the change.

The Load Chart is designed as a high level entry point to profile session results. Subsequently, you can use the Top Activity and Profiling Details views to examine more detailed information on waiting and executing sessions over the length of the session. Alternatively, you can select one or more bars on the graph to populate the Top Activity section (and subsequently, the Details View) with information on a specific subset of the graph.

The Load Chart displays the distribution of waiting and executing sessions over the length of a profiling session.

- Time is displayed on the X axis. You can zoom in and zoom out on the graph via the icons in the upper right hand corner of the graph, once a profiling session is stopped.
- The Y axis shows the average number of sessions waiting or executing. Each supported platform has a specific set of wait event times.

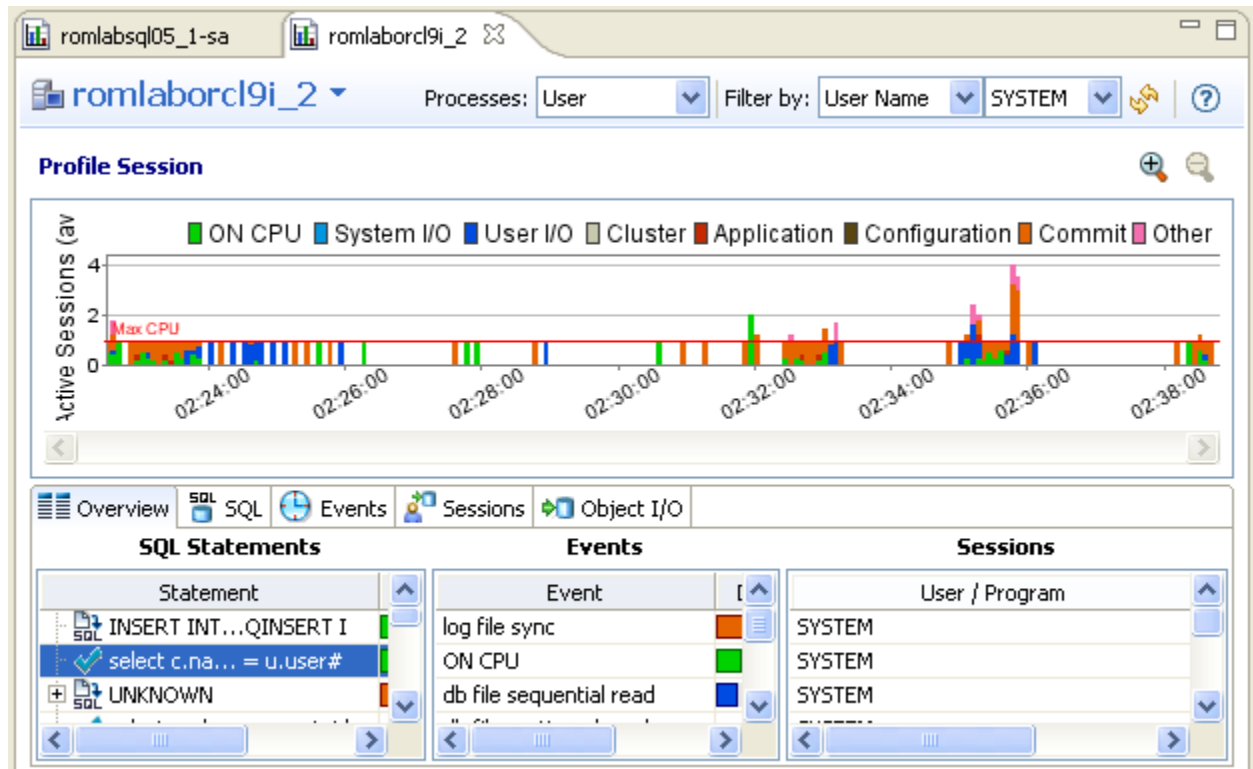
DBMS	Wait Event Category
IBM DB2	Fetch, Cursor, Execution, Operation, Transaction, Connectivity, Lock, Other
Oracle	On CPU, System I/O, User I/O, Cluster, Application, Configuration, Commit, Other
SQL Server	CPU, Lock, Memory, Buffer, I/O, Other
Sybase	CPU, Lock, Memory, I/O, Network, Other

- A chart legend displays a color and code scheme for executing and waiting session categories, in the upper right-hand corner of the view.

ANALYZE THE TOP ACTIVITY SECTION

The Top Activity Section is located in the middle section of the editor and displays where the load originates. Specifically, the top SQL statements, top events that the database spends time in, as well as the top activity sessions.

The Top Activity Section is composed of a series of tabs that provide detailed statistics on individual SQL statements and sessions that were waiting or executing over the length of a profiling session.



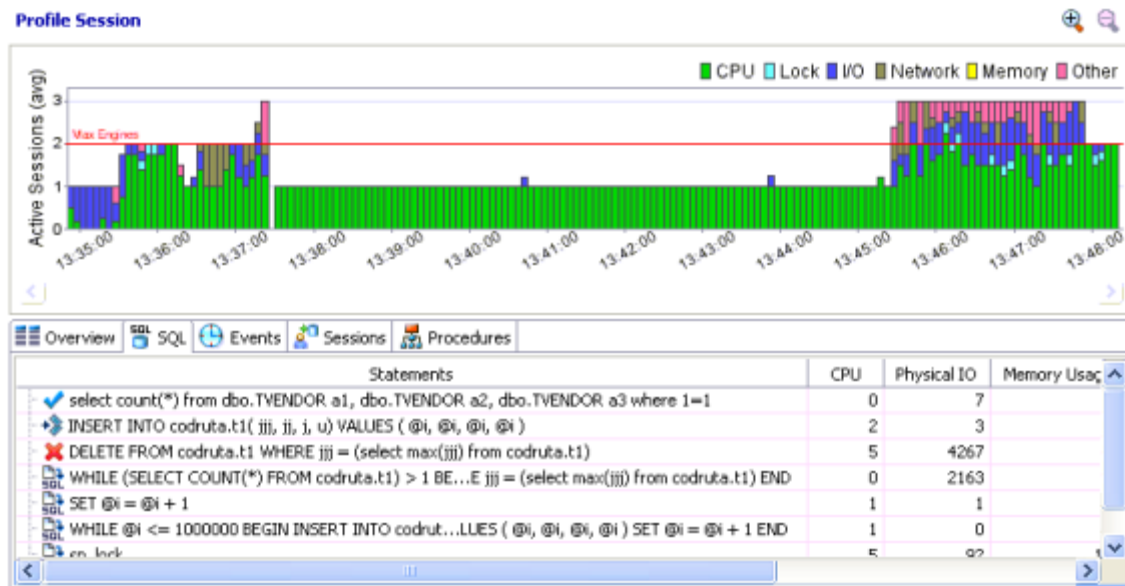
- The top **SQL** tab provides more detailed information than provided on the Overview tab, in terms of executing SQL statements and procedures. For more information, see "[Top SQL Tab](#)" on page 80.
- The top **Execution Activity** (DB 2 Specific) tab provides details about the statements and procedures that ran. This is DB 2 specific. For more information, see "[Top Execution Activity Tab \(DB2 Specific\)](#)" on page 82.
- The top **Events** tab displays information about wait events profiled by the execution process. For more information, see "[Top Events Tab](#)" on page 82.
- The top **Sessions** tab displays information about sessions profiled by the execution process. For more information, see "[Top Sessions Tab](#)" on page 83.
- The top **Object I/O** tab (Oracle-Specific) tab does not appear in the Top Activity Section unless the data source being profiled is an Oracle platform. This tab displays information about the I/O profiled by the execution process. For more information, see "[Top Object I/O Tab \(Oracle-Specific\)](#)" on page 83.
- The top **Procedures** tab (SQL Server and Sybase Specific) displays information about procedures profiled by the execution process. For more information, see "[Top Procedures Tab \(SQL Server and Sybase Specific\)](#)" on page 84.

When you select any item from the Top Activity tabs, details are displayed in the Profiling Details view. The tabs that appear in **Profiling Details** will be different depending on the database platform and whether you selected a statement, session, or an event. This is to accommodate the parameter specifics of the item you selected.

TOP SQL TAB

The Profile editor's SQL tab shows a representation of all SQL statements that are executing or waiting to execute over the length of the profiling session or within the currently selected graph bars.

NOTE: The image below depicts results achieved for a Sybase database. The columns displayed on this tab differ depending on the database platform.



Statements

Statements can be grouped by type by right-clicking the view and selecting **Organize > By Type**. The statement types are: INSERT, SELECT, DELETE, and UPDATE

TIP: Statements are grouped when they differ only by their clause values. This enables the roll-up of SQL statements that only differ by a variable value. For example: select * from emp where empno=1; and select * from emp where empno=2. A '+' symbol appears beside rollup statements. You can click the symbol to expand and view the different statement predicates.

Additionally, the SQL tab displays two other groupings of statements:

Group	Description
OTHER	Includes all recognized statements other than INSERT, SELECT, UPDATE, and DELETE statements.

Group	Description
UNKNOWN	Statements that are not recognized by the application. DB Optimizer has been improved to query the caching more often and more intelligently so that UNKNOWN appears less frequently in the Top SQL tab. The system queries the data source for SQL text in 15 second intervals. Unknown may still appear infrequently as the SQL text may have been removed by the database.

All statements are displayed in a tree structure with the following statement components:

Statement Component	Description
Subject	The DML statement type (and FROM clause, as appropriate).
Predicate	The WHERE clause.
Remainder	Any statement component following the WHERE clause.

For example, all statements with common subjects are shown as a single entry with multiple children; one child for each unique predicate. Predicates are similarly broken down by remainders.

NOTE: Right-clicking the **SQL** tab and selecting **Organize By** lets you choose between **Statement Type** grouping and **None**. The **None** option disables grouping by statement.

Statistics

Statistics are provided for statements and statement components. The statistics let you evaluate costs and spot wait event problems not just at the level of entire SQL statements, but also at the level of statement components. For each subject, predicate or remainder entry, the following statistics are provided:

NOTE: Columns displayed on the top SQL tab differ depending on the data source platform.

Statistic	Shown for Platform	Notes
Executions	SQL Server, Oracle, Sybase, DB2	The number of active executions for the statement or statement component over the length of the profiling session or the selected graph bars.
Avg. Elapsed (sec)	Oracle, DB2	The average amount of time that elapsed while executing the statement during the profiling period. This column appears for only SQL Server, DB2 and Oracle datasources.
DB Activity (%)	SQL Server, Oracle, Sybase, DB2	A graphical representation of the distribution of execution and wait time for the statement or statement component.
SQL ID	Oracle	The ID value of the SQL statement. This statistic only appears on Oracle data sources.
Child Number	Oracle	The child number in the database. This statistic only appears on Oracle data sources.
Parsing User ID	Oracle	The ID of the user who parsed the statement. This statistic only appears on Oracle data sources.

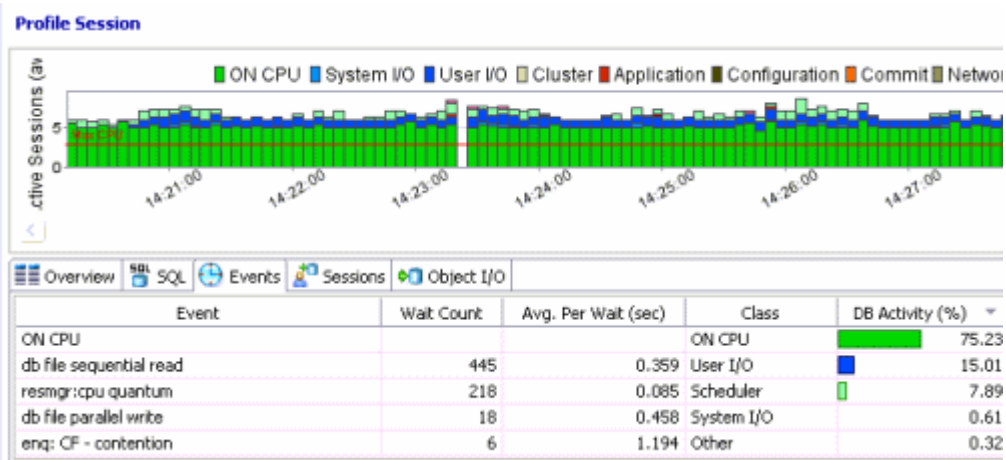
Statistic	Shown for Platform	Notes
Plan Hash Value	Oracle	The execution value of the statement. This statistic only appears on Oracle data sources.

TOP EXECUTION ACTIVITY TAB (DB2 SPECIFIC)

In addition to the statistics displayed on the Top SQL tab, DB2 platforms have an additional tab in the Profile Session editor named Execution Activity, which contains the following statistical rows: Rows Read, Rows Written, Fetch Count, Statement Sorts, Sort Time, and Sort Overflows.

TOP EVENTS TAB

The Top Events tab displays information about wait events on the resources involved in the profiling process. This display is used to tune at the application or database configuration level. For example, if the top events are locks, then application logic needs to be examined. If top events are related to database configuration, then database setup should be investigated.



NOTE: The columns that display are data source-dependant. For example, the Wait Count and Avg. Per Wait (sec) columns display only for an Oracle data source.

TOP SESSIONS TAB

The Sessions Tab provides information about individual sessions. This tab provides information about sessions that are very active or bottlenecked.

User Name	Program	SID	Serial#	Activity (%)	Machine	Session Type
SYSTEM	Executor.exe	125	32563	27.19	EMBARCADERO\ROWEBITA01	USER
	ORACLE.EXE (CTWR)	156	3	25.42	TORLABORCL10G_1	BACKGROUND
	ORACLE.EXE (CJQ0)	162	1	20.67	TORLABORCL10G_1	BACKGROUND
	ORACLE.EXE (m000)	111	22954	3.43	TORLABORCL10G_1	BACKGROUND
SYSTEM	Executor.exe	143	26894	2.72	EMBARCADERO\ROWEANITUCA01	USER
	ORACLE.EXE (SMON)	164	1	2.70	TORLABORCL10G_1	BACKGROUND
SYSTEM	Executor.exe	111	22883	2.37	EMBARCADERO\ROWSNOVAC01	USER
SYSTEM	Executor.exe	97	48228	1.86	EMBARCADERO\ROWSNOVAC01	USER
SYSTEM	Executor.exe	120	59999	1.59	EMBARCADERO\ROWSNOVAC01	USER
	ORACLE.EXE (m001)	97	48199	1.54	TORLABORCL10G_1	BACKGROUND
	ORACLE.EXE (SMON)	164	1	1.54	TORLABORCL10G_1	BACKGROUND

TOP OBJECT I/O TAB (ORACLE-SPECIFIC)

The **Object** I/O Tab is specific to the Oracle data source platform, and displays information about Oracle I/O loads on the profiled data source.

Object	Type	DB Activity (%)	Tablespace	File ID
EMP	TABLE	100.00	SYSTEM	1

The following parameters are displayed on the I/O tab:

Value	Description
Object	The name of the data source object affecting the Oracle I/O.
Type	The object type. For example, table, partition, or index.
DB Activity (%)	Use the color chart on the right-hand side of the I/O tab to view the I/O load on the data source during the profiling session.
Tablespace	The name of the tablespace where the object resides.
File ID	The unique ID value of the file from where specified object resides.

TOP PROCEDURES TAB (SQL SERVER AND SYBASE SPECIFIC)

The Procedures Tab is specific to the SQL Server and Sybase data source platforms. It displays information about Procedure loads on the profiled data source.

Overview	SQL	Events	Sessions	Procedures
Procedure Name	Database Name	Procedure ID	Executions	DB Activity (%)
TEST_PROC2	codruta	850099038	1	50.00
TEST_PROC1	codruta	1842102572	1	50.00

The following parameters are displayed on the Procedures tab:

Value	Description
Procedure Name	The name of the procedure affecting the database performance.
Database Name	The name of the database where the procedure resides.
Procedure ID	The unique ID value of the file where the specified procedure resides.
Executions	The number of times the procedure was executed during the session.
DB Activity (%)	Use the color chart on the right-hand side of the Procedures tab to view the procedures load on the data source during the profiling session.

ANALYZE PROFILING DETAILS

The Profiling Details view displays detailed information on any item selected in the Top Section View, such as an SQL statement, an Event, a Session or a Procedure.

SQL: INSERT INTO PERFCTR_DATA_1(QUERYID, VALUE1) SELECT 951, COUNT(*) FROM SYS.V_\$LOCK L, SYS.DBA_OBJECT...				
SQL Identification	Optimizer and Outline	Execution Statistics (total)	per execution	per row
SQL ID: 1135785965	Optimizer Mode: ALL_ROWS	Fetches: 0.00	0.00	0.00
SQL Address: 6998CE60	Parsing User ID: 5	Executions: 1	1.00	1.00
Child Address: 698AA520	Outline Category	Sorts: 0	0.00	0.00
Children: 1	Outline SID: 0	Disk Reads: 1004	1,004.00	1,004.00
Plan Hash Value: 3592252508	Parsing Statistics	Buffer Gets: 13261	13,261.00	13,261.00
Module: Executor.exe	Memory: 162433	Rows Processed: 1	1.00	1.00
Action	Loads: 134	CPU Time: 93,750.00	93,750.00	93,750.00
SQL Operation Code: 2	Invalidations: 132	Elapsed Time: 70,279,820.00	70,279,820.00	70,279,820.00
Program ID: 101644				
Program Line#: 195				

Depending on the data source platform you have specified, the tabs that appear in the view will be different, in order to accommodate the parameter specifics of the statement you have selected.

Depending on the top activity selected and the profiled platform types, some tabs may not be available.

Statement Selected

When a **Statement** is selected the following Profile Detail tabs are available.

Tab Name	Description	Supported Platform			
		Oracle	Sybase	DB2	SQL Server
SQL text	Displays the full code of the selected SQL statement.	yes	yes	yes	yes
SQL Details	Provides details on statement, like execution statistics.	yes		yes	
Events	Provides database activity details about events the statement is associated with.	yes	yes	yes	yes
Sessions	Shows which sessions executed this statement.	yes	yes	yes	yes
Children Details	Lists all copies of the cursor or sql query, if Oracle has cached multiple copies of the same statement.	yes			
Object I/O	If the SQL query has done physical I/O, then these are the objects, such as tables, and indexes that were read to satisfy the query. Temporary objects with not have values in Object and Type columns.	yes			
Procedures	Shows which procedures contain the selected statement.		yes		yes

Event Selected

When an **Event** is selected the following Profile Detail tabs are available.

Tab Name	Description	Supported Platform			
		Oracle	Sybase	DB2	SQL Server
SQL	Shows which SQL statements waited on this event.	yes	yes	yes	yes
Sessions	Provides information about the sessions associated with the event.	yes	yes	yes	yes
Procedures	Shows which procedures contain the selected event.		yes		yes
Raw Data	Raw data that was sampled from the database, specifically the following: <ul style="list-style-type: none"> • Sample time • SID • Serial # • User name • Program • Sql ID • P1 • P2 • P3 	yes			
Analysis	Displays for "buffer busy waits" and "cache buffer chains latch" waits. The analysis shows data and documentation to assist in solving these bottlenecks.	yes			

Session Selected

When a **Session** is selected the following Profile Detail tabs are available.

Tab Name	Description	Supported Platform			
		Oracle	Sybase	DB2	SQL Server
Sessions	Provides parameters regarding the session. For example, database server connection information, and data regarding the client tool and application.	yes	yes		yes
SQL	Shows which SQL statements this session ran.	yes	yes	yes	yes
Events	Shows which events this session waited on.	yes	yes	yes	yes
Procedures	Shows which procedures ran the selected session.		yes		yes

NOTE: When right-clicking on a SQL statement in the Top Activity Section in Profiling, if the SQL statement is run by a different user than the user who is running DBO, then the User Mismatch dialog appears, with an example of the following message: "This query was executed by [SOE] and you are currently connected as [system]. We recommend you reconnect as [SOE] to tune the SQL. Would you like to continue anyway?" This message indicates that the statement is being tuned by a user other than the user who originally ran the query, and tables may be missing based on the different schemas. Click OK to run the query, or click Cancel and run tuning under the original user.

Procedure Selected

When a **Procedure** is selected, the following Profile Details become available

Tab Name	Description	Supported Platform			
		Oracle	Sybase	DB2	SQL Server
SQL Text	Shows the SQL text of the selected procedure.		yes		yes
SQL	Shows which SQL statements this procedure ran.		yes		yes
Events	Shows which events the selected procedure waited on.		yes		yes
Sessions	Provides parameters regarding the session. For example, database server connection information, and data regarding the client tool and application.		yes		yes

This section also addresses the following topics:

- [Viewing Details on the SQL Tab](#) on page 87
- [Viewing Details on the Sessions Tab](#) on page 90
- [Viewing Details on the Events Tab](#) on page 92

- [Viewing Details on the Procedures Tab](#) on page 96

VIEWING DETAILS ON THE SQL TAB

In the **Top Activity Session**, selecting a statement entry on the **SQL** tab displays information in the **Profiling Details** view. The graph portion and details on the event category tabs on the new editor pertain only to the selected statement. Additionally, new tabs become available:

- **SQL Text tab:** Shows the full code of the SQL statement. For more information, see "[SQL Text](#)" on page 88.
- **SQL Details tab:** Displays execution details. This tab is only displayed for Oracle data sources. For more information, see "[SQL Details](#)" on page 88.
- **Events tab:** Displays information about the events the selected statement is associated with. For more information, see "[Events](#)" on page 89.
- **Sessions tab:** Displays information about the sessions that the selected statement is associated with. This tab is displayed only for Oracle data sources. For more information, see "[Sessions](#)" on page 89.
- **Procedures tab:** Displays information about the procedures that contain the selected statement. This tab is displayed only for SQL Server and Sybase datasources. For more information, see "[Procedures](#)" on page 90.

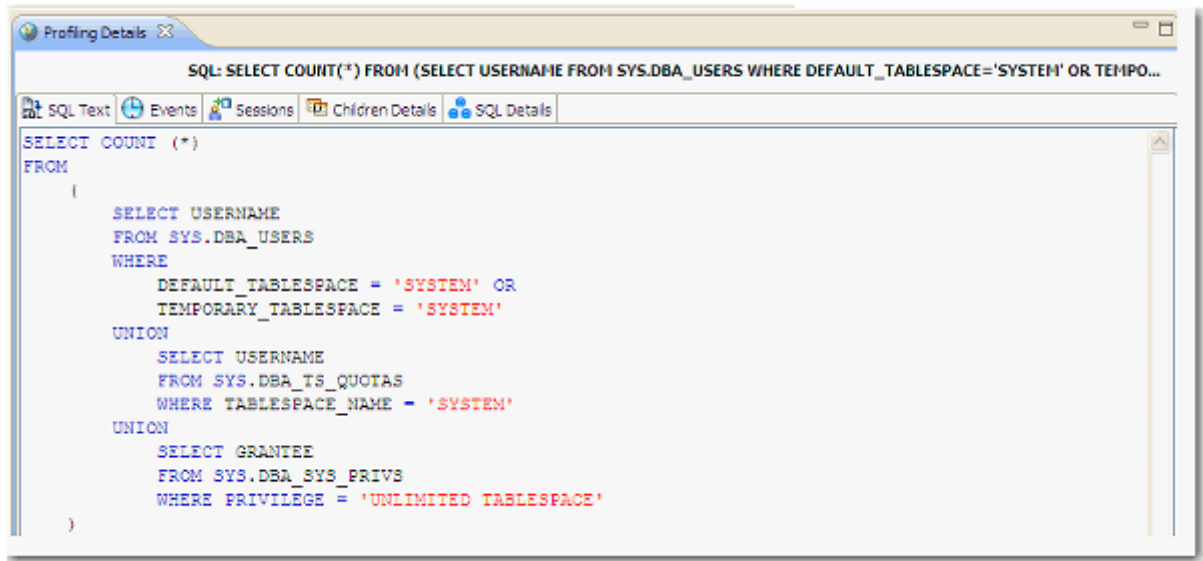
To select a SQL tab statement entry:

- On the **SQL** tab, click on a statement with no child nodes or on a leaf node in the statement structure.

The new profiling editor page opens, as reflected by the bread crumb trail at the top left of the editor. You can continue to drill down into the statement, as needed.

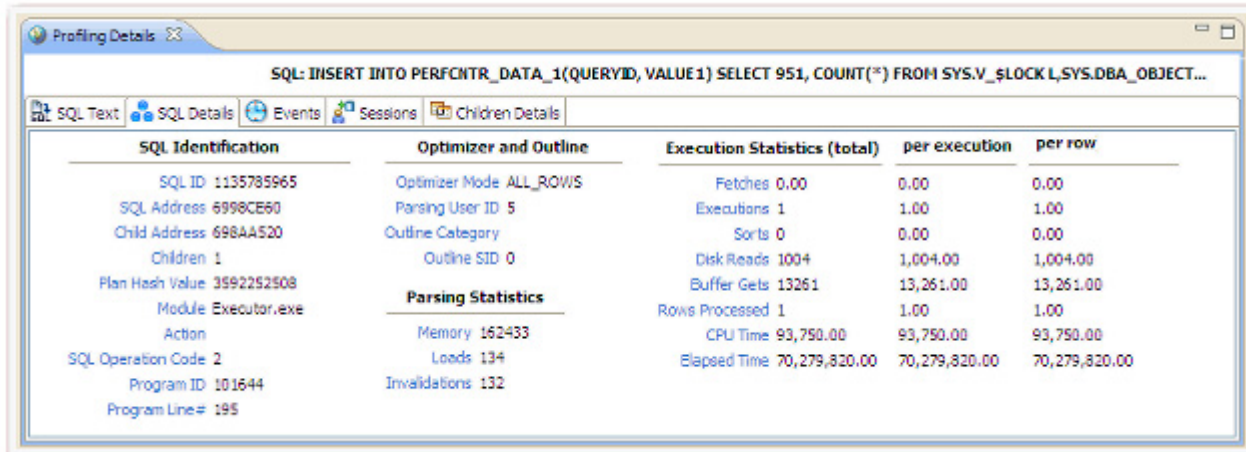
SQL Text

The SQL Text tab displays the full code of the SQL statement.



SQL Details

The SQL Details tab provides information and the execution of the statement and other information related to how it is running. It is only applicable to Oracle data sources:

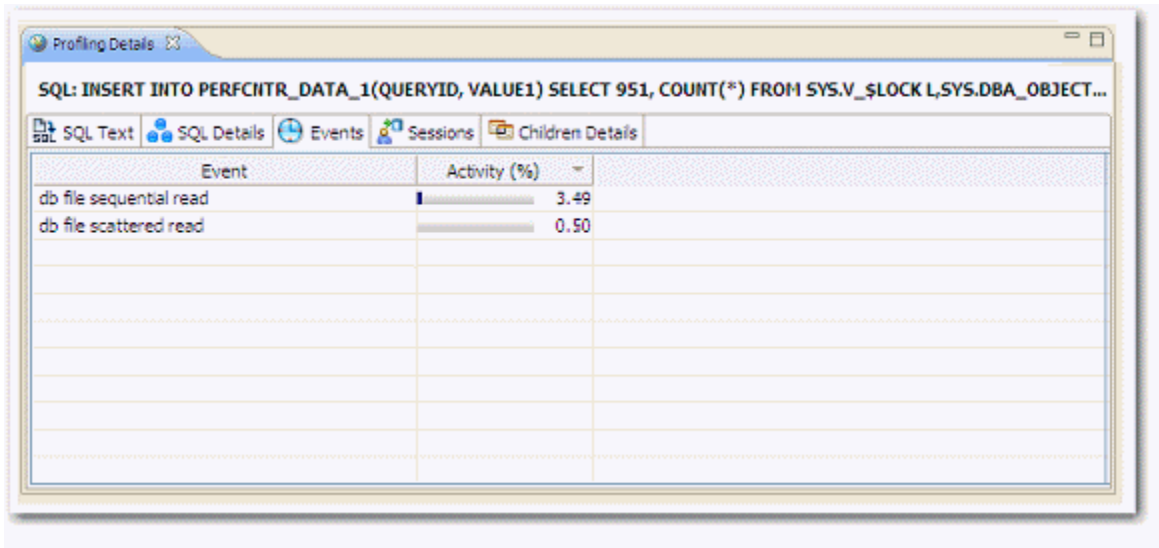


SQL Details include:

Parameters	Description
SQL Identification Values	The SQL ID value of the statement.
Optimizer and Outline Values	Optimizer-specific values pertaining to the parsing user ID value and outline SID.
Parsing Statistics	Information regarding memory, loads, and invalidation values.
Execution Statistics	The execution statistics of the statement. This category includes disk reads, buffer gets, rows, and values that represent CPU and elapsed time.

Events

The Events tab provides details about the events that the statement is associated with.



Sessions

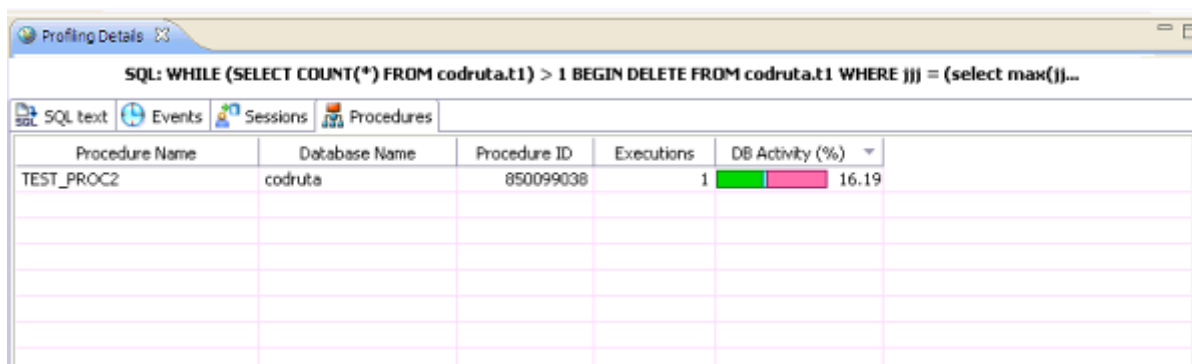
The Sessions tab provides information about any sessions the statement is associated with:

User Name	Program	SID	Serial#	Activity (%)	Machine	Session Type
SYSTEM	Executor.exe	145	9180	3.14	EMBARCADERO\ROWSNOVAC01	USER
SYSTEM	Executor.exe	145	9242	0.85	EMBARCADERO\ROWSNOVAC01	USER

Session details include information on different parameters, depending on the platform. For example, on Oracle platforms, the following parameters are displayed: User Name, Program, SID, Serial #, Activity (%), Network Machine Name, and Session Type.

Procedures

The Procedures tab provides information about any procedures containing the selected statement.



The following parameters are displayed on the Procedures tab:

Value	Description
Procedure Name	The name of the procedure that contains the selected statement.
Database Name	The name of the database where the procedure resides.
Procedure ID	The unique ID value of the file where the specified procedure resides.
Executions	The number of times the procedure was executed.
DB Activity (%)	Use the color chart on the right-hand side of the Procedures tab to view the procedures load on the data source during the profiling session.

VIEWING DETAILS ON THE SESSIONS TAB

In the **Top Activities Section**, selecting a statement entry on the **Sessions** tab displays information in the **Profiling Details** view. The graph portion and details on the event category tabs on the new editor pertain only to the selected statement. Additionally, new tabs become available.

Selecting an event type entry on an event category tab opens a new profiling editor page. The graph portion and details on the **Sessions** tab and event category tabs on the new editor page pertain only to the selected wait event and to SQL statements that waited in that event.

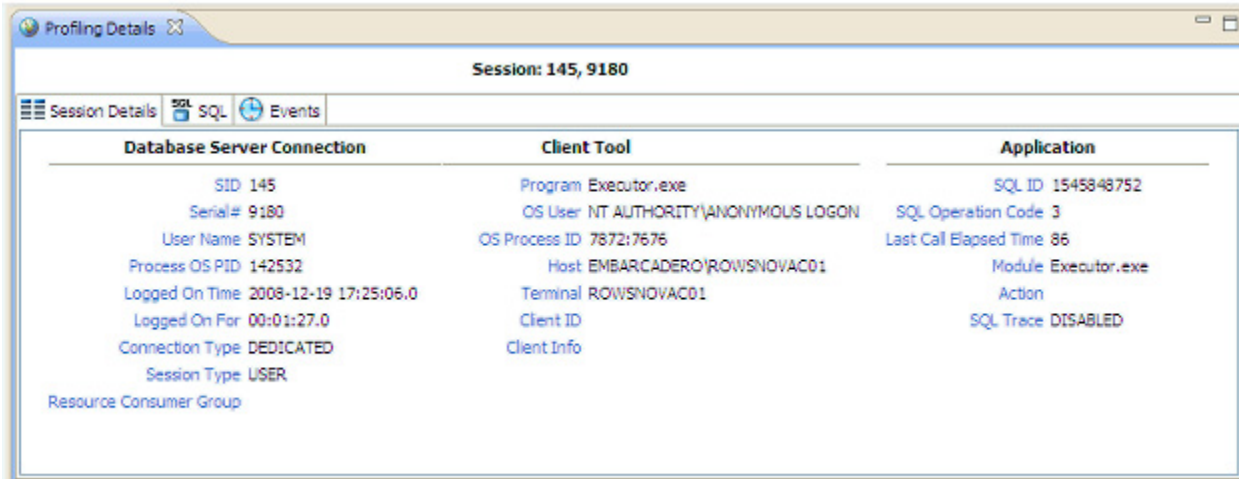
- **Session Details tab:** Shows system details about the selected session. For more information, see "[Session Details](#)" on page 91.
- **SQL tab:** Displays information about the SQL files that the selected session is associated with. This tab only appears on Oracle platforms. For more information, see "[SQL](#)" on page 92.
- **Events tab:** Displays the time and parameter information about the selected session. For more information, see "[Events](#)" on page 93.
- **Procedures tab:** Displays the details of any procedures run in the selected session. For SQL Server and Sybase datasources only. For more information, see "[Procedures](#)" on page 93.

Session Details

The Session tab provides further information about the selected session. The following are examples of the session details provided for different platforms.

NOTE: The fields that display vary depending on the database platform.

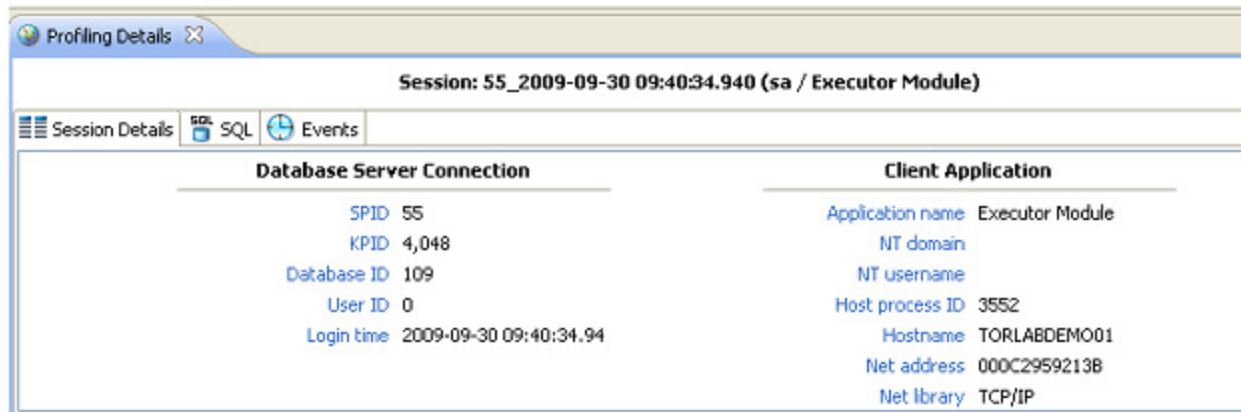
Oracle Profiling Details



The screenshot shows the 'Profiling Details' window for Session: 145, 9180. The window has tabs for 'Session Details', 'SQL', and 'Events'. The 'Session Details' tab is active, displaying information organized into three columns: Database Server Connection, Client Tool, and Application.

Database Server Connection	Client Tool	Application
SID 145	Program Executor.exe	SQL ID 1545848752
Serial# 9180	OS User NT AUTHORITY\ANONYMOUS LOGON	SQL Operation Code 3
User Name SYSTEM	OS Process ID 7872:7676	Last Call Elapsed Time 86
Process OS PID 142532	Host EMBARCADERO\ROWSNOVAC01	Module Executor.exe
Logged On Time 2008-12-19 17:25:06.0	Terminal ROWSNOVAC01	Action
Logged On For 00:01:27.0	Client ID	SQL Trace DISABLED
Connection Type DEDICATED	Client Info	
Session Type USER		
Resource Consumer Group		

Microsoft SQL Server

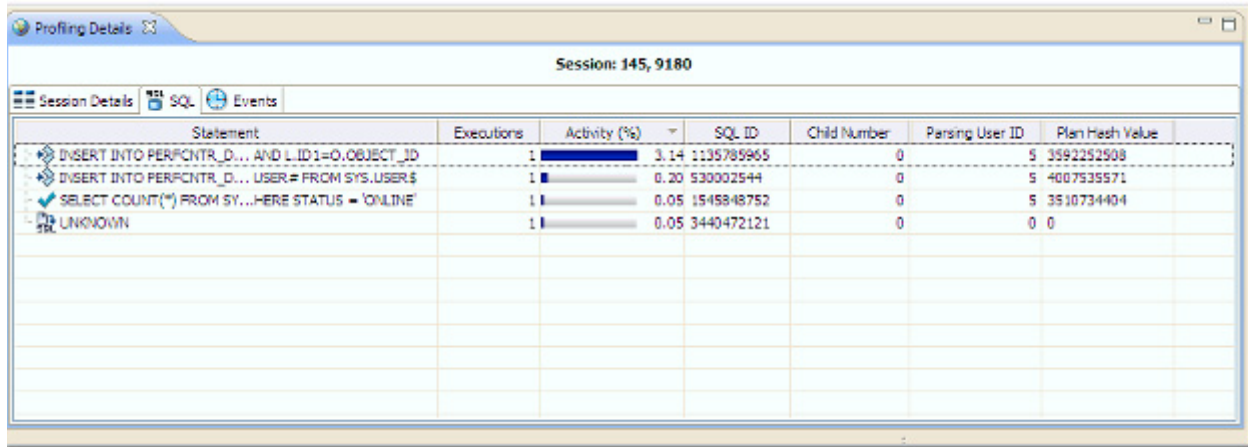


The screenshot shows the 'Profiling Details' window for Session: 55_2009-09-30 09:40:34.940 (sa / Executor Module). The window has tabs for 'Session Details', 'SQL', and 'Events'. The 'Session Details' tab is active, displaying information organized into two columns: Database Server Connection and Client Application.

Database Server Connection	Client Application
SPID 55	Application name Executor Module
KPID 4,048	NT domain
Database ID 109	NT username
User ID 0	Host process ID 3552
Login time 2009-09-30 09:40:34.94	Hostname TORLABDEMO01
	Net address 000C2959213B
	Net library TCP/IP

SQL

The SQL tab displays information about the statements associated with the session.



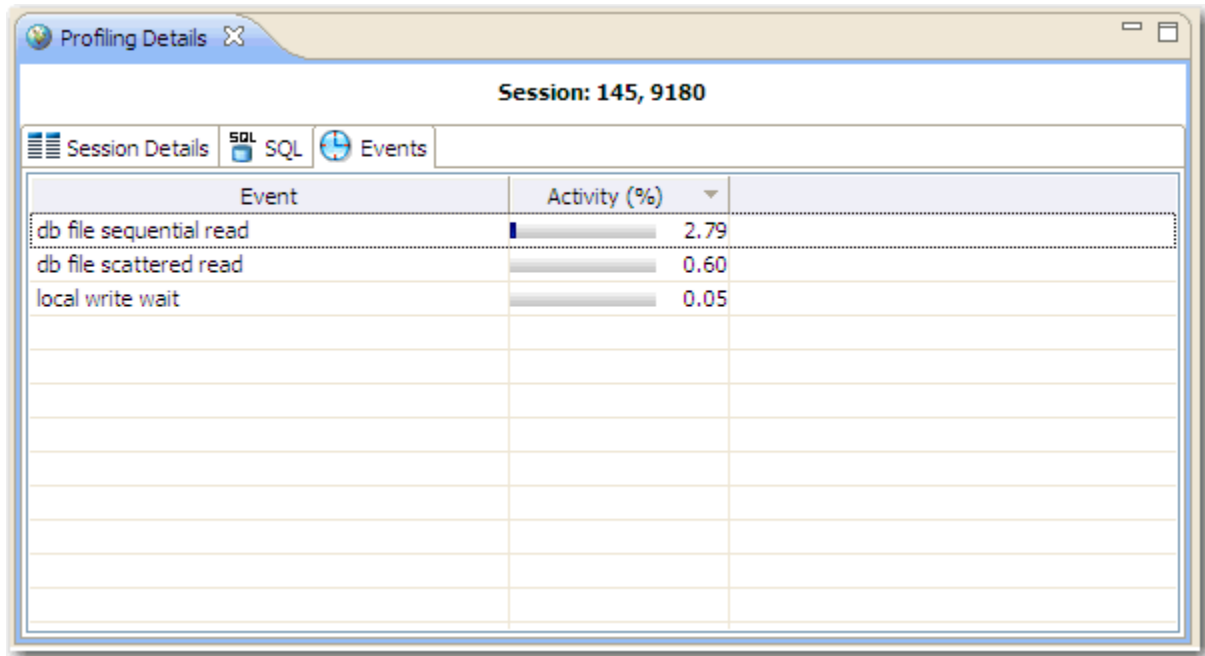
Statement	Executions	Activity (%)	SQL ID	Child Number	Parsing User ID	Plan Hash Value
INSERT INTO PERF/CNTR_D... AND L_ID1=O.OBJECT_ID	1	3.14	1135785965	0	S	3592252508
INSERT INTO PERF/CNTR_D... USER# FROM SYS.USER\$	1	0.20	530002544	0	S	4007535571
SELECT COUNT(*) FROM SY...HERE STATUS = 'ONLINE'	1	0.05	1545848752	0	S	3510734404
UNKNOWN	1	0.05	3440472121	0	0	0

SQL statements are listed by the following parameters:

Value	Notes
Statement	The name of the statement.
Executions	The number of times the statement was executed during the session.
Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.
SQL ID	The SQL ID value of the statement.
Child Number	The child number in the database.
Parsing User ID	The ID of the user who parsed the statement.
Plan Hash Value	The execution value of the statement.

Events

The Events tab provides details about the events that the session is associated with.

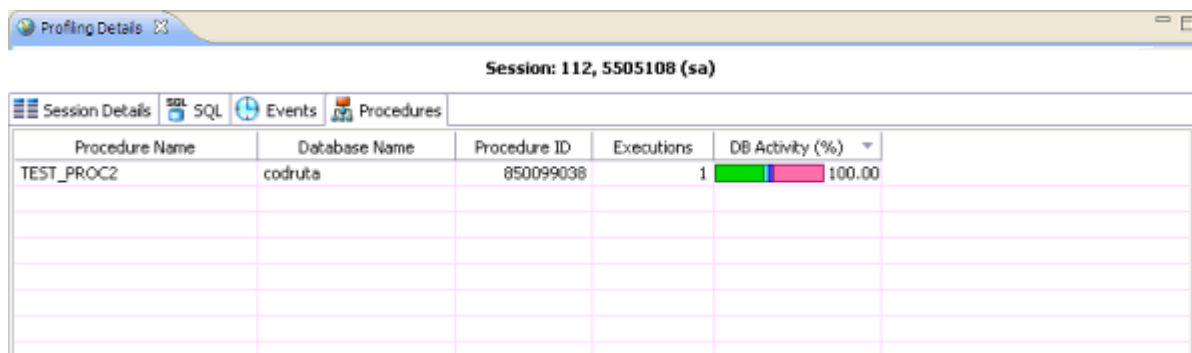


Events are listed by the following values:

Value	Notes
Event Name	The name of the event.
Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.

Procedures

For SQL Server and Sybase data sources only, the Procedures tab provides details about the procedures that the session is associated with



The following parameters are displayed on the Procedures tab:

Value	Description
Procedure Name	The name of the procedure that ran during the selected session.
Database Name	The name of the database where the procedure resides.
Procedure ID	The unique ID value of the file where the specified procedure resides.
Executions	The number of times the procedure was executed during the session.
DB Activity (%)	Use the color chart on the right-hand side of the Procedures tab to view the procedures load on the data source during the profiling session.

VIEWING DETAILS ON THE EVENTS TAB

In the **Top Activities Section**, selecting a statement entry on the **Event** tab displays information in the **Profiling Details** view. The graph portion and details on the event category tabs on the new editor pertain only to the selected statement. Additionally, new tabs become available.

Selecting an event type entry on an event category tab opens a new profiling editor page. The graph portion and details on the **Events** tab and event category tabs on the new editor page pertain only to the selected wait event and to SQL statements that waited in that event.

- **SQL** tab: Shows the statements involved in the selected event. For more information, see "[SQL](#)" on page 94.
- **Sessions** tab: Displays information about the sessions that the selected event is associated with. For more information, see "[Sessions](#)" on page 95.
- **Procedures** tab: Displays information about the procedures that ran during the selected event. For more information, see "[Procedures](#)" on page 96.

SQL

The SQL tab displays information about the SQL statements involved in the selected event.

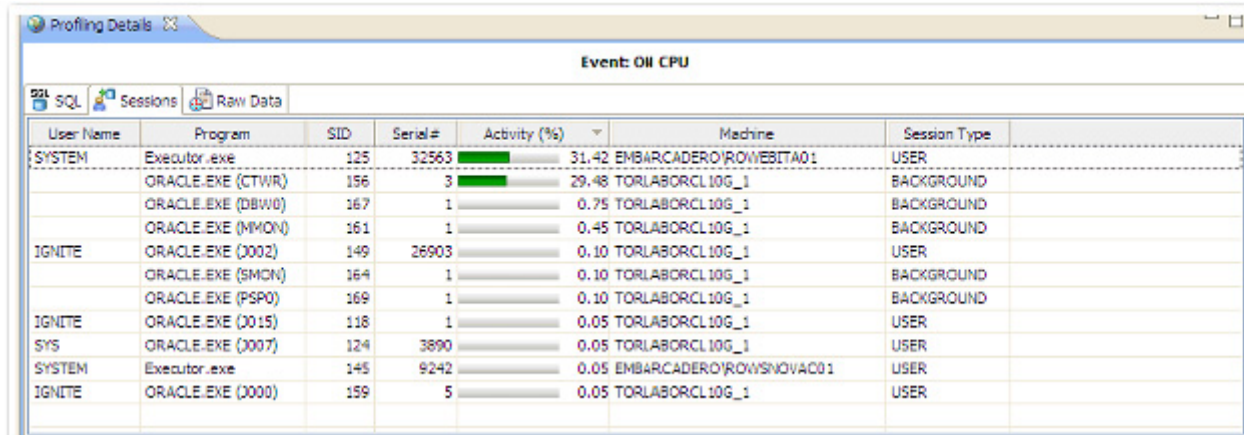
Event: wait for someone else to finish reading in mass				
Statements	CPU	Physical IO	Memory Usage	Executions
DELETE FROM codruta.t1 WHERE jjj = (select max(jjj) from codruta.t1)	0	3036	8	
WHILE (SELECT COUNT(*) FROM codruta.t1) > 1 BEGIN jjj = (select max(jjj) from codruta.t1) END	0	1625	8	

Value	Notes
Statement	The name of the statement.
SQL ID	The ID value of the SQL statement.
Child Number	The child number in the database.
Parsing User ID	The ID of the user who parsed the statement.
Plan Hash Value	The execution value of the statement.

Value	Notes
CPU	Cumulative CPU time for the process. (measured in "ticks", an arbitrary unit of time)
Physical IO	Cumulative disk reads and writes for the process. (total count)
Memory Usage	Number of pages in the procedure cache that are currently allocated to this process. A negative number indicates that the process is freeing memory allocated by another process.
Executions	The number of times the statement was executed.
Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.

Sessions

The Sessions tab displays the sessions and related information regarding those that were associated with the selected event.



User Name	Program	SID	Serial#	Activity (%)	Machine	Session Type
SYSTEM	Executor.exe	125	32563	31.42	EMBARCADERO\ROWEBITA01	USER
	ORACLE.EXE (CTWR)	156	3	29.48	TORLABORCL10G_1	BACKGROUND
	ORACLE.EXE (DBW0)	167	1	0.75	TORLABORCL10G_1	BACKGROUND
	ORACLE.EXE (MMON)	161	1	0.45	TORLABORCL10G_1	BACKGROUND
IGNITE	ORACLE.EXE (J002)	149	26903	0.10	TORLABORCL10G_1	USER
	ORACLE.EXE (SMON)	164	1	0.10	TORLABORCL10G_1	BACKGROUND
	ORACLE.EXE (PSP0)	169	1	0.10	TORLABORCL10G_1	BACKGROUND
IGNITE	ORACLE.EXE (J015)	118	1	0.05	TORLABORCL10G_1	USER
SYS	ORACLE.EXE (J007)	124	3890	0.05	TORLABORCL10G_1	USER
SYSTEM	Executor.exe	145	9242	0.05	EMBARCADERO\ROWSNOVAC01	USER
IGNITE	ORACLE.EXE (J000)	159	5	0.05	TORLABORCL10G_1	USER

The following parameters are displayed on the Sessions tab:

Value	Notes
User Name	The user name under which the session was run.
Program	The name of the executable under which the session was run.
SID	The SID value of the session.
Serial Number	The serial number of the machine from which the session executed.
Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.
Machine	The machine name and network location of the machine from which the session executed.
Session Type	The type of session.

Procedures

The Procedures tab displays the procedures and related information regarding those that were associated with the selected event.

Procedure Name	Database Name	Procedure ID	Executions	DB Activity (%)
TEST_PROC1	codruta	1842102572	1	100.00

The following parameters are displayed on the Procedures tab:

Value	Notes
Procedure Name	The name of the procedure that ran during the event.
Database Name	The name of the database where the procedure resides.
Procedure ID	The unique ID of the procedure.
Executions	The number of times the procedure ran during the event.
DB Activity (%)	A graphical representation of the distribution of execution and wait time for the procedure.

VIEWING DETAILS ON THE PROCEDURES TAB

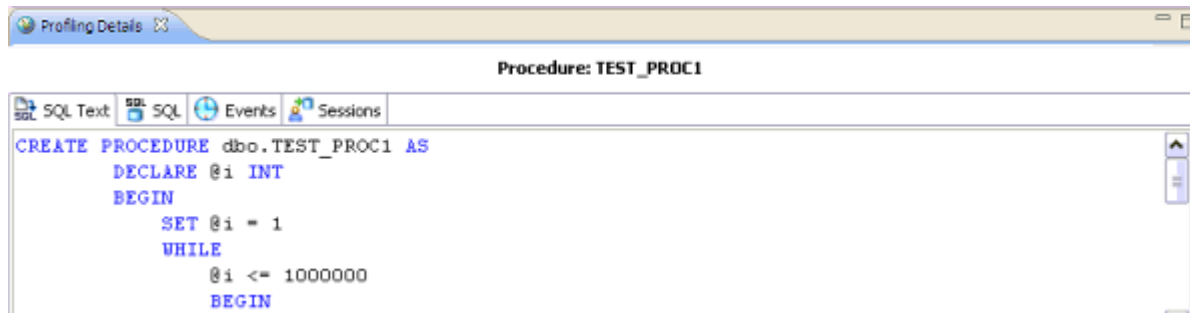
In the **Top Activities Section**, selecting a procedure entry on the **Procedure** tab displays information in the **Profiling Details** view. The graph portion and details on the procedure category tabs on the new editor pertain only to the selected procedure. Additionally, new tabs become available.

Selecting a procedure type entry on a procedure category tab opens a new profiling editor page. The graph portion and details on the **Procedure** tab and procedure category tabs on the new editor page pertain only to the selected procedure and to SQL statements that waited in that procedure.

- The **SQL Text** tab shows the SQL of the procedure. For more information, see "[SQL Text](#)" on page 97.
- The **SQL** shows the statements involved in the procedure. For more information, see "[SQL](#)" on page 97.
- The **Events** displays the time and parameter information about the selected procedure. For more information, see "[Events](#)" on page 98.
- The **Sessions** displays information about the sessions that the selected procedure is associated with. For more information, see "[Sessions](#)" on page 98.

SQL Text

The SQL Text tab displays the full code of the procedure.



SQL

The SQL tab displays information about the SQL statements involved in the selected procedure.

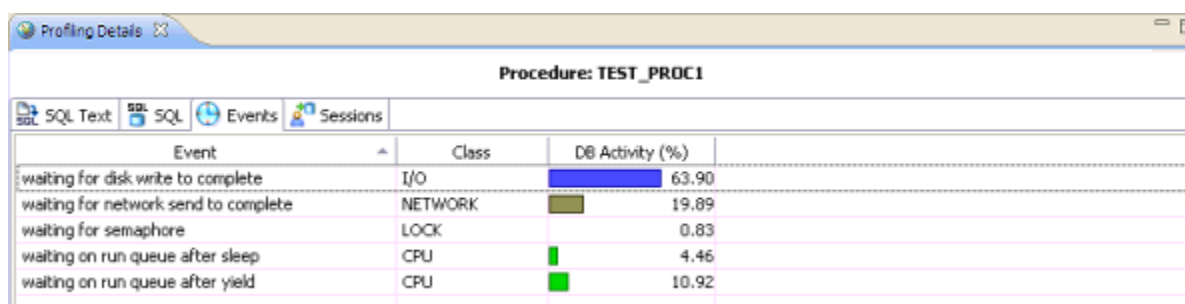
Statements	CPU	Physical IO	Memory Usage	Execu
INSERT INTO codruta.t1(j1j, j2, i, u) VALUES (@i, @i, @i, @i)	2	3	8	
WHILE @i <= 1000000 BEGIN INSERT INTO codrut...LUES (@i, @i, @i, @i) SET @i = @i + 1 END	1	0	8	
SET @i = @i + 1	1	0	8	

The SQL tab displays the following parameters about the statement:

Value	Notes
Statement	The name of the statement.
SQL ID	The ID value of the SQL statement.
Child Number	The child number in the database.
Parsing User ID	The ID of the user who parsed the statement.
Plan Hash Value	The execution value of the statement.
CPU	Cumulative CPU time for the process. (measured in "ticks", an arbitrary unit of time)
Physical IO	Cumulative disk reads and writes for the process. (total count)
Memory Usage	Number of pages in the procedure cache that are currently allocated to this process. A negative number indicates that the process is freeing memory allocated by another process.
Executions	The number of times the statement was executed.
Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.

Events

The Events tab provides details about the events that the session is associated with

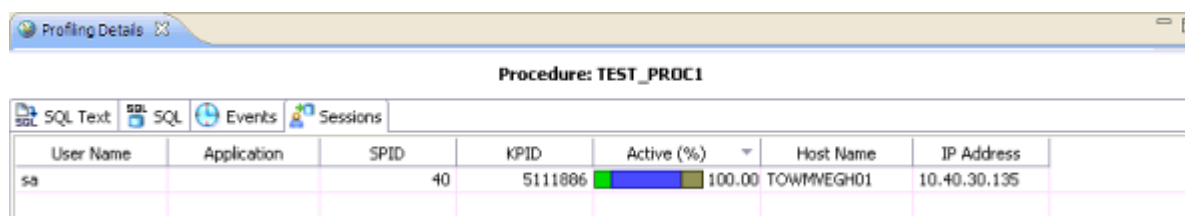


Events are listed by the following values:

Value	Notes
Event Name	The name of the event.
Class	The wait group the event in the selected procedure belongs to.
Activity (%)	A graphical representation of the distribution of execution and wait time for the event.

Sessions

The Sessions tab displays the sessions and related information regarding those that were associated with the selected procedure.



The following parameters are displayed on the Sessions tab:

Value	Notes
User Name	The user name under which the session was run.
Program	The name of the executable under which the session was run.
SID	The SID value of the session.
Serial Number	The serial number of the machine from which the session executed.
Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.
Machine	The machine name and network location of the machine from which the session executed.
Session Type	The type of session.

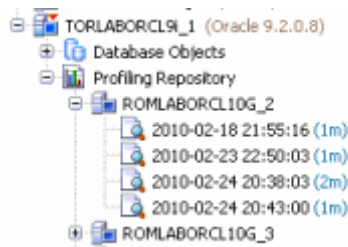
SAVING PROFILING SESSIONS

The profiling session is saved automatically or when you try to close it according to the choices made in the Profile Configuration dialog. For information on configuring your profiling sessions, see "[Building Launch Configurations](#)" on page 107. Profiling sessions can be saved in the current workspace in an archive file with a **.oar** suffix or for Oracle users, into the Profiling Repository.

The .oar archive file is named with a default file name of:

- The name of the data source if the session was not initiated from a named launch configuration
- The name of the launch configuration if the session was initiated from a named launch configuration.

If you are using an Oracle data source and have configured DB Optimizer to automatically save profiling sessions in a Profiling Repository within an Oracle data source, then the profiling session is saved in the Profiling Repository under the name of the data source. Each profiling session for that data source is named using a date and time stamp. As you can see below, the duration of the profiling session is also saved with the session data.

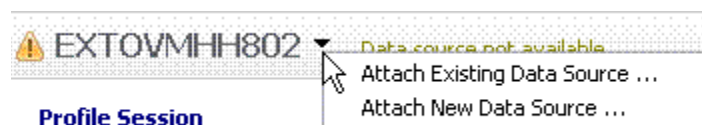


For information on working with the Profiling Repository, see "[Work with the Profiling Repository](#)" on page 100.

The time period of the saved session is the amount of data on the chart. The maximum amount of data on the chart is determined when profiling is started (1 hour default). You can specify the amount of time to profile the data source in the Profile Configurations dialog and you can also stop the profiling at any time.

Saving the profile lets you open the archive at a later time for subsequent analysis by yourself or by other DB Optimizer users. Use standard DB Optimizer file techniques to save, open, or close SQL Profiling archives.

If you open a profiling archive on a machine on which the associated data source is not registered, a **Data source not available** warning appears in the profiling editor header. Use the associated control to specify a data source already defined on the machine or to register a new data source.



WORK WITH THE PROFILING REPOSITORY

NOTE: Saving wait-time statistics to the Profiling Repository is not supported in DB Optimizer XE Developer.

The Profiling Repository is only available when profile session data is saved to an Oracle data source. For more information, see ["Building Launch Configurations"](#) on page 107. When the system is configured to automatically save profiling information to the Profiling Repository DB Optimizer can profile 24 hours a day 7 days a week, thus providing much more statistical data for analysis. Also, since the Profiling Repository resides on a data source and not on the local disk, other DB Optimizer users can also view and analyze the profiles.

To start saving profile sessions to the Profiling Repository:

- 1 From the Data Source Explorer right-click the data source you want to profile for and select **Profile As** from the menu, then choose **Profile Configurations**.

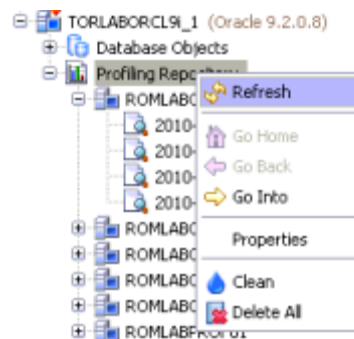
The **Profile Configurations** dialog appears.

- 2 In the **Name** field of the **Profile** tab, enter a name for the profile group for this data source.
- 3 In the **Profiling Repository** area of the **Profile** tab, click **Save to data source** and then from the list of available datasources, choose the Oracle data source where you want to store your profiling information.
- 4 Click **Apply** and then click **Profile** to start a profiling session immediately.

Any new profiling session that you start continue until you manually stop it. The profile session can be for as long as you like, days or weeks even. When the profiling session has been stopped a profile file is stored in the profile group for this data source. The name of the profile file is the date and time when the profile finished.

To delete profile sessions saved in the Profiling Repository:

- 1 In the **Data Source Explorer**, locate and then click the Profiling Repository.



DB Optimizer connects to the Profiling Repository data source.

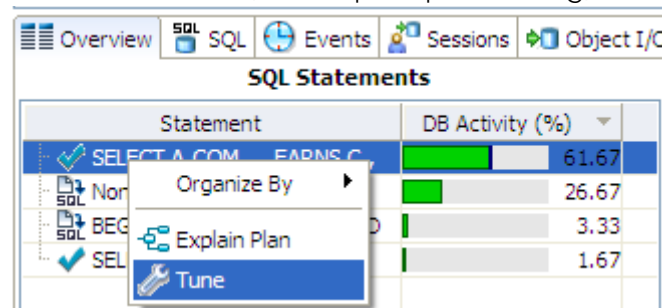
- 2 To delete all sessions in the Profiling Repository, right-click **Profiling Repository** and then select **Delete All**.

To delete a specific profiling session, expand the Profiling Repository and the data source containing the profiling session, and right-click the name of the profiling session, and then select **Delete**.

The profiling session data is deleted, however, some information about the data source is retained in order to expedite future profiling on this data source. If you are certain you will not want to retain this information, right-click the Profiling Repository and then select **Clean**.

IMPORT STATEMENTS TO TUNING

The profiling feature lets you submit one or more SQL tab statements for tuning by the tuning feature. This lets you take advantage of tuning's hint-based and transformation-based suggestions, detailed execution statistics, and explain plan costing, in tuning a statement.



To open a tuning job on a statement appearing on the SQL tab of the profiling editor:

- Select one or more statements, right-click and select **Tune** from the context menu. Tuning opens on the selected statement.

NOTE: The SQL will be tuned as the user/schema that profiling was running under. If the query being tuned was run by another user/schema, it is recommended to connect to the database as that user/schema and copy/paste the query into tuning, rather than import the statement directly from profiling.

For more information, see "Tuning SQL Statements" in the SQL Tuner help.

OTHER PROFILING COMMANDS

In addition to the default viewing options provided by the views, profiling also provides the following features and functionality:

- **Zooming In and Out.** For more information, see "[Zooming In and Out](#)" on page 102.
- **Filtering Results.** For more information, see "[Filtering Results](#)" on page 102.

ZOOMING IN AND OUT

To zoom in or out on the Load Graph:

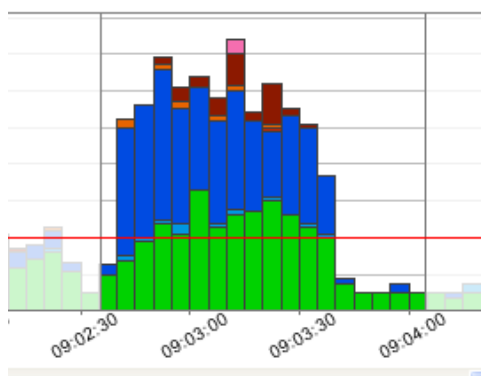
In the upper right-hand corner of the Load graph, click the Zoom In or Zoom Out icons, respectively.

NOTE: The Zoom In and Zoom Out commands are only available when a session has been stopped.

By default, the information contained on the Load Chart spans the entire length of the profiling session. You can select one or more bars of the graph to have the tabbed view populated with statistics for only the selected subset of the graph.

To display statistics for one or more bars on the graph, use one of the following methods:

Click-drag across one or more bars.



FILTERING RESULTS

You can display filtered subsets of the original profiling results set for each section of profiling based on DBMS platform type:

- **IBM DB/2 for Windows, Unix, and Linux:** Creator ID, Cursor Name, Package Name, and Statement Type
- **Microsoft SQL Server:** Application Name, Command, Database Name, and Hostname
- **Oracle:** Action Hash, Module Hash, and Program
- **Sybase:** Application, Database ID, Host, IP Address, and Process priority

You filter results using the filter controls in the upper, right-hand part of the profiling editor.



Additionally, on Oracle platforms, you can filter results by user, or foreground, or background activity. Select All, User (Foreground), or Background to filter out the specified process activity, respectively.

To filter profile editor results:

- 1 Use the **Filter By** menu to select a filter type. The second menu becomes active based on your selection in the first menu.
- 2 Use the second menu to specify a value.

The profiling editor is updated to show only results associated with your choice.

TIP: Select **-None-** from the **Filter by** dropdown to restore the unfiltered results.

CONFIGURING PROFILING

This section addresses the following topics:

- [Configuring DBMS Properties and Permissions](#) on page 103
- [Building Launch Configurations on page 107](#)

CONFIGURING DBMS PROPERTIES AND PERMISSIONS

Profiling supports the following DBMS platforms:

- IBM DB/2 for WIndows, Unix, and Linux
- Microsoft SQL Server
- Oracle
- Sybase

The following describe how to set up a platform to utilize Optimizer on supported database platforms:

- [Configuring IBM DB/2 for Windows, Unix, and Linux](#) on page 103
- [Configuring Microsoft SQL Server](#) on page 105
- [Configuring Oracle](#) on page 106
- [Configuring Sybase](#) on page 106

CONFIGURING IBM DB/2 FOR WINDOWS, UNIX, AND LINUX

NOTE: The connected profiling user should be a member of the DB2 SYSMON group.

By default, DB2 Monitor flags are set to OFF. As a result, when attempting to launch a Profile job on a DB2 data source, users may encounter the following message: "One or more errors have occurred that prevent session profiling against this data source." Examine the details below and consult your data source administrator and/or the data source documentation to resolve the problem(s)."

You can resolve this error using one of two methods:

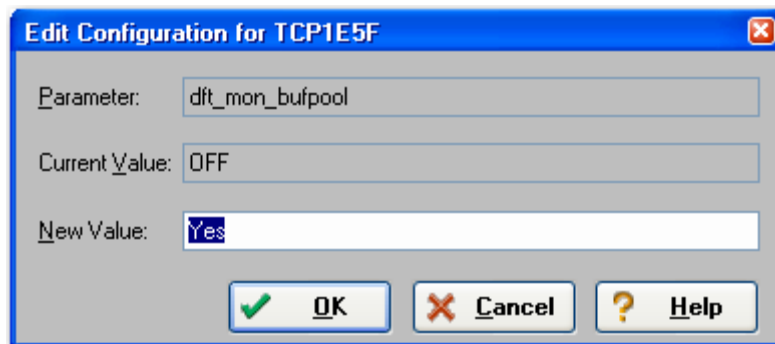
- Enabling DB2 Monitor Flags via Embarcadero DBArtisan
- Command Line Option

To resolve the error through DBArtisan:

1 Ensure the following DB2 Monitor Flags are turned on in DB2:

- dft_mon_uow
- dft_mon_uow
- dft_mon_stmt
- dft_mon_timestamp
- dft_mon_lock
- dft_mon_bufpool
- dft_mon_table
- dft_mon_timestamp

You can set view and set Monitor Flags via DBArtisan. Ensure that the New Value field for each variable is set to Yes, as shown below.



2 Restart the DB2 data source to enable the changes, then launch DB Optimizer and begin profiling.

To resolve the error through the command line:

This solution must be performed from DB2 CLP, on the DB2 server. If you attempt to perform these tasks through a client, an error message will result.

- 1 Navigate to **start > Programs > IBM DB2/CMDLINE TOOLS > COMMAND LINE PROCESSOR**.
- 2 Turn the monitor switches on using the following commands:

```
db2 update dbm cfg using dft_mon_lock on dft_mon_bufpool on dft_mon_sort on
dft_mon_stmt on dft_mon_table on timestamp on dft_mon_uow on
db2stop
db2start
```

- 3 Ensure that the switches are turned on by connecting to the server with the following command:

```
Db2 connect to database username password password
```

The following screenshot provides an example of the input and output from the server:

```
db2 => connect to gim user db2admin
Enter current password for db2admin
db2 => get monitor switches

      Monitor Recording Switches

Switch list for db partition number 0
Buffer Pool Activity Information (BUFFERPOOL) = ON 03/05/2009 19:14:06.61257
Lock Information (LOCK) = ON 03/05/2009 19:14:06.61257
Sorting Information (SORT) = ON 03/05/2009 19:14:06.61257
SQL Statement Information (STATEMENT) = ON 03/05/2009 19:14:06.61257
Table Activity Information (TABLE) = ON 03/05/2009 19:14:06.61257
Take Timestamp Information (TIMESTAMP) = ON 03/05/2009 18:50:44.00034
Unit of Work Information (UOW) = ON 03/05/2009 19:14:06.61257
```

CONFIGURING MICROSOFT SQL SERVER

Perform the following tasks to ensure that SQL Server is compatible with Optimizer:

- If you are setting up SQL Server 2000, ensure the current user is a member of the sysadmin group.
- If you are setting up later versions of SQL Server, the current user must meet one of the following requirements:
 - Be a member of sysadmin, or have the VIEW SERVER STATE permission enabled.
 - Be a member of sysadmin, or have the SELECT permission enabled.

On SQL Server 2000 only:

You can enable profiling to capture more SQL by adding the following flag:

```
DBCC TRACEON(2861)
```

Trace flag 2861 instructs SQL Server to keep zero cost plans in cache, which SQL Server would typically not cache (such as simple ad-hoc queries, set statements, commit transaction, and others). In other words, the number of objects in the procedure cache increases when trace flag 2861 is turned on because the additional objects are so small, there is a slight increase in memory that is taken up by the procedure cache.

Ensure you restart the server for your changes to take affect.

CONFIGURING ORACLE

Oracle users need access to V\$ views. In order to configure Oracle to provide users with these privileges:

- If you are setting up Oracle 10 or later, ensure you are logged in as sys or system with the sysdba role, or the SELECT_CATALOG_ROLE has been granted to *user_name*.
- If you are setting up an earlier version of Oracle, ensure you are logged in as sys or system with the sysdba role.

CONFIGURING SYBASE

Perform the following tasks to ensure that Sybase is compatible with Optimizer:

- Ensure the following system configuration properties are activated:
 - Enabling Monitoring (sp_configure "enable monitoring", 1)
 - Wait Event Timing (sp_configure "wait event timing", 1)
 - Max SQL Text Monitored (sp_configure "SQL batch capture", 1)
 - SQL Batch Capture (sp_configure "max SQL text monitored", 4096)

Additionally, perform the following tasks, as necessary:

- If a user does not have mon_role enabled, the user will not be able to access Adaptive Server's monitoring tables.
- If the monProcess table is missing, the user will not be able to view currently connected sessions.
- If the sysprocesses table is missing, the user will not be able to view information about Adaptive Server processes.
- If the monWaitEventInfo table is missing, the user will not be able to view information about wait events.

- If the monProcessSQLText table is missing, the user will not be able to view currently executing SQL statements.

NOTE: These packages should only be installed by the DBA.

Profiling enables you to create a set of launch configurations to store the basic properties for each profiling session that you run on a regular basis. A launch configuration enables you to start profiling sessions from a single menu command, rather than re-define configuration parameters each time you want to run one.

BUILDING LAUNCH CONFIGURATIONS

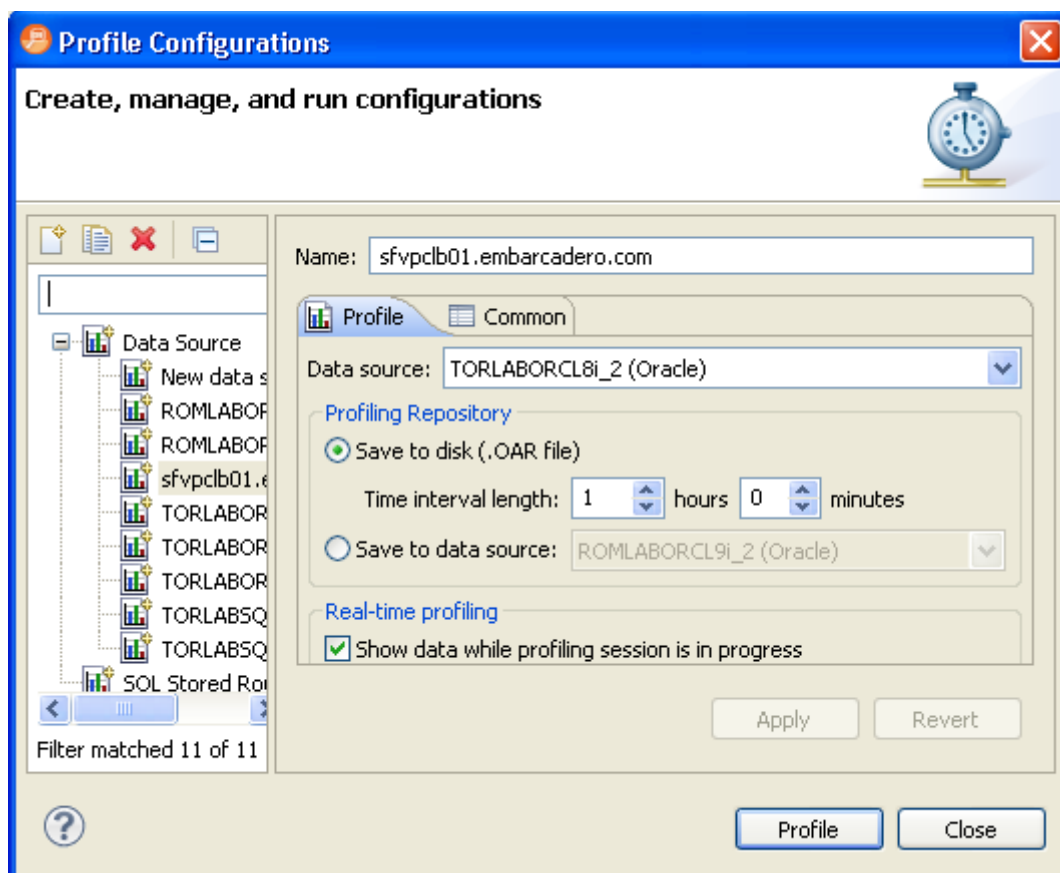
Profiling enables you to store parameters related to specific profiling sessions, in a launch configuration for stored routines. Multiple configurations can be created for each data source in your enterprise and saved with unique names that identify them in the application.

NOTE: On all supported platforms, support for stored routines includes functions and procedures. On Oracle, stored routine support also includes package functions and package procedures.

To create a launch configuration:

- 1 Right-click on the data source you want to build a configuration for and select **Profile As** from the menu, then choose **Profile Configurations**.

The **Profile Configurations** dialog appears.



- 2 Select the name of the data source and modify the parameters on the **Profile** tab, as needed.
- 3 In the **Name** field, provide a name for the launch configuration. You should select a name that will make the launch configuration unique and easily identified once it is saved in the application.

- 4 In the **Profiling Repository** area, choose to save the profile session to disk or if you are profiling an Oracle data source then you can choose to save the profile session to the data source.

If you choose to save the session to disk, in the **Time interval length** area specify the length of the profiling session. When you try to close the Profile Session, you will be prompted to save the file and can then name the file as desired.

A .oar file saved to disk opens very quickly from **File > Open** dialog and has a limit of just under 1000 hours of profiling data. Profiling to a data source directly allows the system to capture more data for a longer period of time, until you decide to stop the profiling session. The profiling session is automatically saved to the Profiling Repository where other DB Optimizer users can also view the session for their own analysis.

- 5 Click **Apply**. The launch configuration is stored in the application.

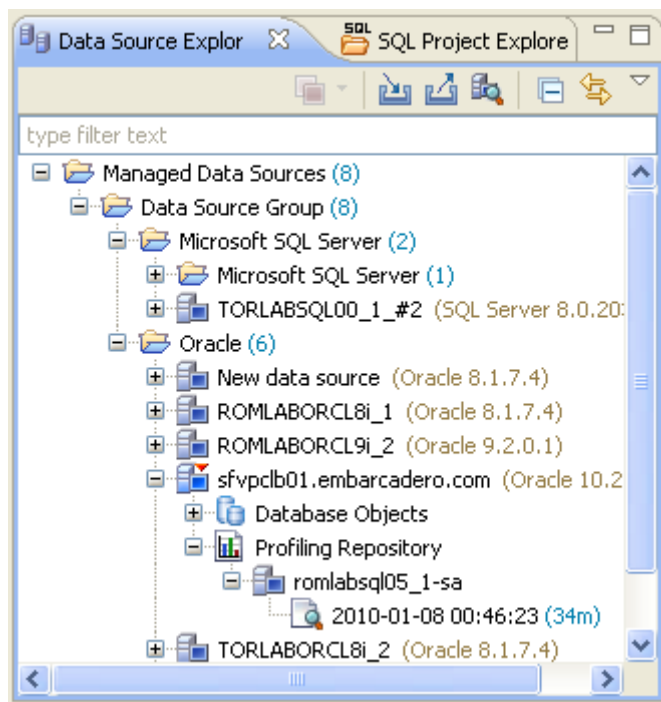
Once a launch configuration is defined, you can execute it in profiling. For more information, see ["Running a Profiling Session" on page 73](#).

NOTE: The parameters provided when you select the data source name in the left pane control session parameters for the specified data source. To set these controls, see ["Configuring DBMS Properties and Permissions" on page 103](#).

The following describes fields and options of the Profile tab that require further explanation.

- **Name** indicates the name of the profile configuration.
- **Data source** indicates the name of the data source to which the profile applies.

- **Save to disk/Save to data source** gives you the option to save your profiling session to a .oar file which you can access from within DB Optimizer or you can save the profiling session to a data source of your choice if you are using an Oracle data source. DB Optimizer will create a Profiling Repository, similar to the following on the selected Oracle data source. The structure of the Profiling Repository is created from the name of the data source and the date and time of each specific profiling session.

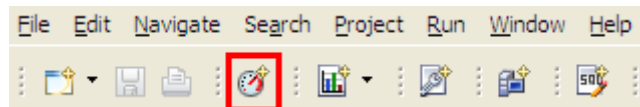


Saving your profiling sessions to a live data source enables you to better organize your profile session data for later review.

- **Time Interval Length** indicates how many hours of the session to save to disk. Since the profile session continues until you manually stop it the session length may exceed the time interval length. For example, the time interval length is set to four hours but the profiling session continues for 10 hours. In this case only the last four hours of data is retained. This parameter also indicates the total width of the time load graph. The longer a profile is, the larger the saved file will be. For heavily loaded databases, the time interval length value should not exceed eight hours.
- The **Show Data While Profile Session is in Progress** check box enables "real time" profiling, which refreshes the data of the session as profiling runs. The **Refresh Interval** specifies how often in seconds profiling updates this data.

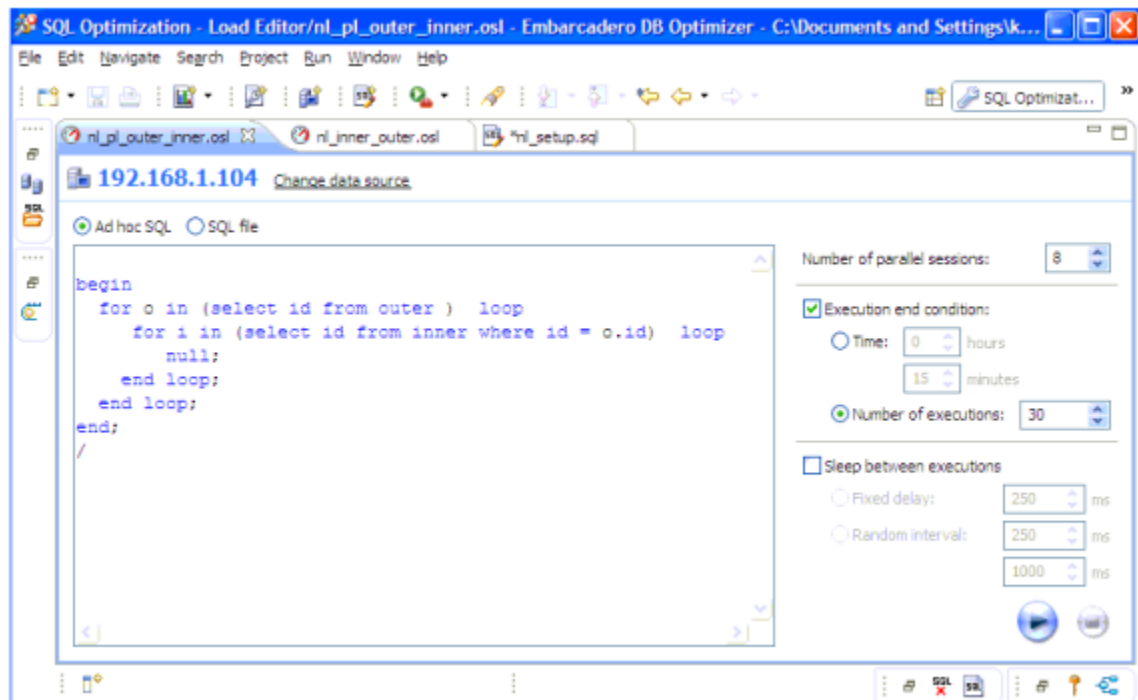
NOTE: Profiling can run sessions based on ad hoc parameters you designate before executing the profiling process. However, by building profile launch profiles, it is a much more efficient method of managing standard, frequent, or common profiling sessions.

USING LOAD TESTER



The load editor can be run either with **File > New > Load Editor** or with the load editor icon shown above in the red square.

The icon depicts an RPM meter on a car with a red line. The idea behind the icon is that we can run a load on a database and stress the database with the load similar to red lining.



The load editor page has space on the left to show the sql to be run. The sql can be typed in or pasted in or read from a file if the **SQL File** option at the top right-hand side of the window is selected.

On the right are options on how to run the SQL

- Number of parallel sessions
- Length of test
- Number of executions

Sleep between executions

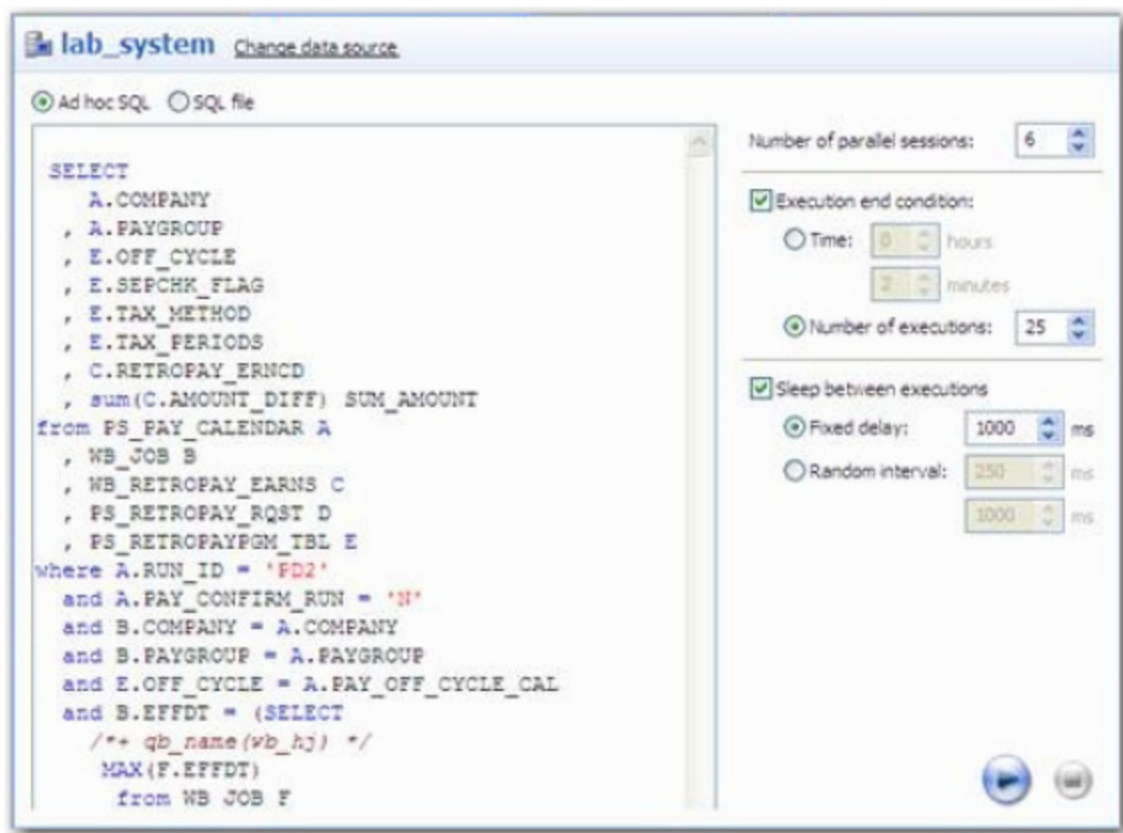
- No sleep
- Fixed sleep

- Random sleep between a max and min

Methodology

- Write SQL with Editor
- Set up Load with Load Editor
- Kick off profiling the database
- Run the load in the Load Editor
- Verify the database load profile to see if there are any major issues

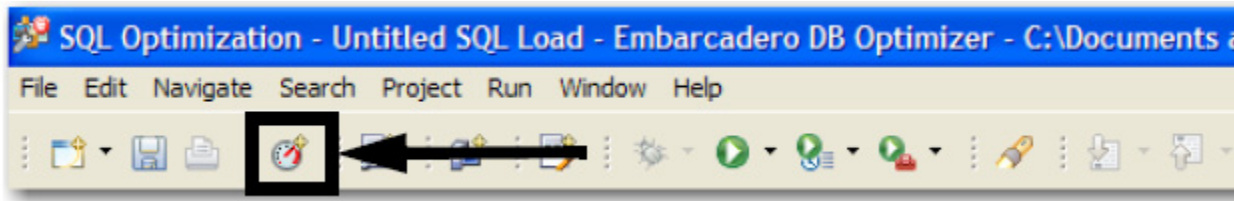
The SQL Load Editor enables you to configure and execute SQL code against a data source.



This feature enables you to specify a data source against which the code will be executed, and then provides options that enable you to choose a period of time that you want the script to execute for, and at what intervals the execution “loop” occurs.

On execution, SQL Load Editor runs in the background. It can therefore be run in conjunction with a profiling session in order to analyze the effects of the executing load against the specified data source. Once you run a SQL script via Load Editor, you can start the SQL Profiling function and analyze the results of the load.

The Load Editor is accessed via the **Load Editor** icon on the Toolbar:



When you open Load Editor, click **Select Data Source** to specify the data source against which you want the SQL script to run.

Choose **Ad hoc SQL** and manually type (or copy/paste) the SQL code into the window provided, or select **SQL file** and navigate to the SQL file you want to run. The window populates with the code from the selected file.

The following configuration parameters are set with Load Editor prior to executing the SQL script:

Configuration Parameter	Description
Number of Parallel Sessions	Specifies the number of jobs that the execution script will operate on.
Execution End Condition	Specifies if the script execution process runs for a set amount of time or script executions. Choose Time if you want the script to execute over a specific period of time, or Number of Executions if you want the script to execute a specific number of times.
Sleep Between Executions	Specifies if Load Editor will wait before running the execution script again. Select the check box and choose Fixed Delay or Random Interval, depending on whether you want the script to execute at a specific time, or at random intervals within a specified range of time.

To run Load Editor:

- 1 Access Load Editor by selecting the icon on the Toolbar. **Load Editor** opens.
- 2 Click **Select Data Source** and choose a data source you want to run the SQL code against.
- 3 Choose **Ad hoc SQL** or **SQL file**, and then copy/paste or manually type the code you want to execute in the window provided, or navigate to the location of the file, respectively.
- 4 In the right-hand panel, choose the execution configuration parameters to specify how you want Load Editor to handle the script.
- 5 Click the **Execute** icon in the lower right-hand corner of the screen. The script starts to execute against the specified data source, using the configuration parameters you selected.

- 6 If you are profiling a data source, start and run a new profiling session on the data source you specified in Load Editor. The session will reflect how your SQL script executes against the specified data source.

USING TUNING

This section provides information on tuning, its functionality, and is structured so a user can follow the information provided to fully tune their enterprise in terms of more efficient query paths at the SQL statement level of individual data sources.

Tuner has three parts

- Query rewrites and quick fixes
- Alternative execution plans generated via optimizer directives
- Analysis of Query showing
 - Indexes used, not used, missing (suggested to create)
 - Graphic display of query

The SQL tuner will take a query and add database optimizer directives to change the execution path of the query. A list of all the unique execution paths will be generated with all duplicates eliminated from the list. The final list of alternative paths can be executed. Any path that takes more than 150% of the base case will be canceled because we are only interested on paths that could be faster than the base case so no need to waste time and resources continuing to run cases that are slower than the original. After the cases have been executed they can be sorted in order of elapsed time. If a better path is found then those optimizer directives can be included in the original query to achieve optimal response time.

You can save the entire content of a tuning job for later analysis or for sharing with other users.

This section contains the following topics:

- [Introduction to Database Tuning](#) on page 115
- [Understanding the Tuner Interface](#) on page 128
- [Tuning SQL Statements](#) on page 135
- [Using Oracle-Specific Features](#) on page 171
- [Additional Tuning Commands](#) on page 176
- [Configuring Tuning](#) on page 180
- [Examples of Transformations and SQL Query Rewrites](#) on page 185
- [DBMS Hints](#) on page 186

INTRODUCTION TO DATABASE TUNING

This discussion will help you understand the methodology behind DB Optimizer's tuning functionality and how you can use it to optimize database performance. This discussion is comprised of the following topics:

- [Introduction to DB Optimizer's Tuner](#) on page 116
- [SQL Tuning Methodology](#) on page 118
- [SQL Tuner Overview](#) on page 119
- [What's happening on the databases?](#) on page 120
- [Tuning Example](#) on page 123
 - [The Database is Hanging or the Application has Problems](#) on page 123
 - [The Database Caused the Problem](#) on page 125
 - [The Machine Caused the Problem](#) on page 126
- [Finding and Tuning Problem SQL](#) on page 128

INTRODUCTION TO DB OPTIMIZER'S TUNER

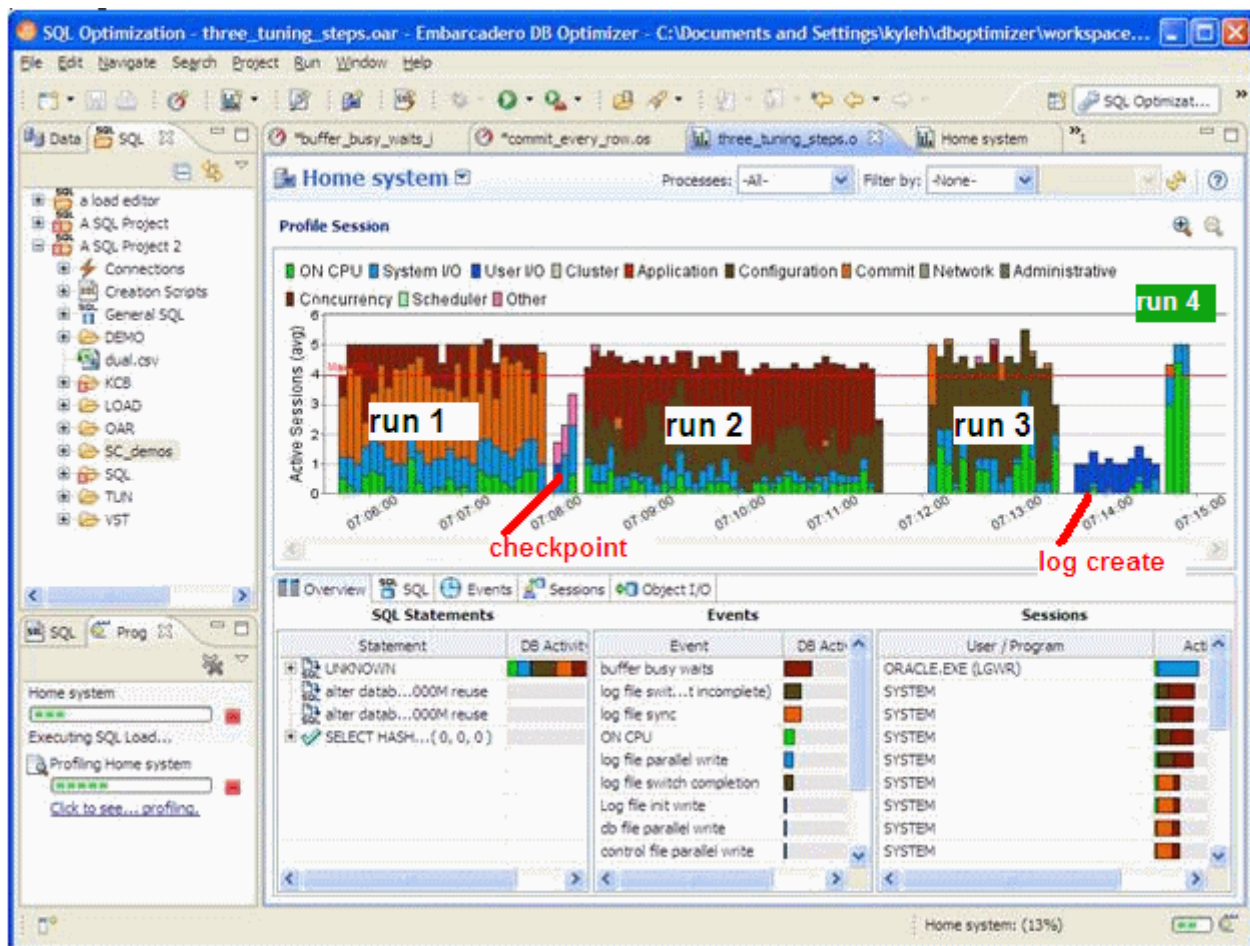
DB Optimizer's methodology grew out of the impossible predicament presented by the defacto method of database tuning. The standard method was trying to collect 100% of the statistics 100% of the time. Trying to collect all the statistics as fast as possible ends up putting load on the monitored database and creating problems. Stories of problems created by database monitoring products abound in the industry. In order to avoid putting load on the target database, performance monitoring tools have to collect less often as a compromise. Oracle compromised in 10g with AWR (their automated performance data collector), only running it once an hour because of the performance impact. Not only is the impact on the monitored target high, but the amount of data collected is staggering, but the worst problem of all though, is the impossibility of correlating statistics with the sessions and SQL that created the problems or suffered the consequences.

The solution to collecting performance data required letting go of the old problematic paradigm of trying to collect as many performance counters possible as often as we could and instead freeing ourselves with the simple approach of sampling session state. Session state includes what the session is, what its state is (active, waiting, and if waiting, what it is waiting on) and what SQL it is running. The session state method was officially packaged by Oracle in 10g when they introduced Active Session History (ASH). ASH is an automated collection of session state sampling. The rich robust data from ASH in its raw form is difficult to read and interpret. The solution for this was Average Active Sessions (AAS). AAS is a single powerful metric which measures the load on the database based on the ASH data. AAS data provided the perfect road map for what data to drill into. The main drill downs are "top sql", "top session", "top event", and "top objects".

Other aggregations are possible based on the different dimensions in the ASH data.

Tuning Example

Here is an example screen shot of the same batch job being run four times. Between each run performance modifications are made based on what was seen in the in the profiling load chart:



Run:

- 1 In run 1, "log file sync" event is the primary bottleneck. To correct this we moved the log files to a faster device. (You can see the checkpoint activity just after run 1 where we moved the log files.)
- 2 In run 2, the "buffer busy wait" event is the primary bottleneck. To correct this we moved the table from a normal tablespace to an Automatic Segment Space Managed tablespace.
- 3 In run 3 the "log file switch" (checkpoint incomplete) event is the primary bottleneck. To correct this we increased the size of the log files. (You can see the IO time spent creating the new redo logs just after run 3.)
- 4 The run time of run4 is the shortest and all the time is spent on the CPU which was our goal, take advantage of all the processors and run the batch job as quickly as possible.

NOTE: To view an explanation of the event, hover over the even name in the Event section.

Conclusion:

With the load chart we can quickly and easily identify the bottlenecks in the database, take corrective actions, and see the results. In the first run, almost all the time is spent waiting, in the second run we eliminated a bottleneck but we actually spent more time - the bottleneck was worse. Sometime this happens as eliminating one bottleneck causes great contention on the next bottleneck. (You can see the width of the run, the time it ran, is wider in run 2). By the third run, we are using more CPU and the run time is faster and finally by the 4th run all the time spent is on CPU, no waiting, and the run is the fastest by far.

SQL TUNING METHODOLOGY

- 1 Verify that the execution path is the optimal for the query

If not either use the tuning directives (such as hints on Oracle) or

Identify why the native optimizer failed to pick the optimal path

- 2 If the query is still slow then look at adding indexes

- 3 If the query is still slow, then you know you are going to have to look at the architecture

What information is the query trying to get?

Is this information necessary?

Are there alternative ways to get this information?

DB Optimizer's SQL Tuner can help with 1 and 2. Step 3 will have to be done by a developer or DBA but knowing that step 1 and 2 have already been validated can indicate to management that step 3 is necessary and therefore allocate sufficient resources for step 3.

How do we know if the native database optimizer chose the optimal path? How long would it take to check this by hand?

DB Optimizer's SQL Tuner is a solid fast sanity test to verify the plan chosen by the native database SQL optimizer. Tuner quickly generates as many alternative paths as possible and allows the user to execute them to see if there are more efficient execution paths. DB Optimizer's SQL Tuner is successful at tuning queries that have a suboptimal execution path.

A query has a sub-optimal execution path when the database optimizer has miscalculated the cost of the various possible access paths and mistakenly chosen a bad path. The access path calculations can be miscalculated because of the following reasons:

- The table/index statistics are missing or wrong. (For example, the number of rows is missing or way off.)
- The data is skewed, for example, the number of orders with an open status is usually low compared to all the orders that have a closed status because the work is complete. (For example, orders get filled every day, but only a few are open and needing to be processed.) Looking for open orders should probably use an index and return fewer rows than looking for closed orders which should probably just do a full table scan.

- The predicates used are correlated. The optimizer treats two predicate filters on a table as more selective than just one, but this is not always the case such is the case in the query, how many Swedes speak Swedish which basically returns the same number of results as just asking for the number of Swedes alone. Another example is how many Swedes speak Swahili, which is probably more selective than the optimizer would guess.
- A bug in the optimizer

DB Optimizer's SQL tuner will take a query and try to produce as many execution paths as possible. These alternative execution paths can then be run to see if there is a faster or less resource expensive execution path. The execution of each alternative case is timed and if the execution exceeds 1.5 X the original case then its execution is stopped and we move on to the next case. This avoids wasting time and resources on execution plans that are clearly suboptimal.

SQL TUNER OVERVIEW

Tuning provides an easy and optimal way to discover efficient paths for queries that may not be performing as quickly or as efficiently as they could be.

The application enables the optimization of poorly-performing SQL code through the detection and modification of execution paths used in data retrieval. This process is performed through the following functions:

- Hint Injection
- Index Analysis
- Statistic Analysis (Oracle only)
- Query re-writes such as suggesting joins to eliminate Cartesian joins, adding transitivity predicates, and unnesting subqueries in the WHERE clause.

Tuning analyzes an SQL statement and supplies execution path directives to the application that encourage the database to use different paths.

For example, if tuning is selecting from two tables (A and B), it will enable the joining of A to B, or B to A as well as the join form. Additionally, different joining methods such as nested loops or hash joins can be used and will be tested, as appropriate. Tuning will select alternate paths, and enable you to change the original path to one of the alternates. Execution paths slower than the original are eliminated, which enables you to select the quickest of the returned selections and improve query times, overall.

This enables the DBA to correctly optimize queries in the cases where the native optimizer failed.

WHAT'S HAPPENING ON THE DATABASES?

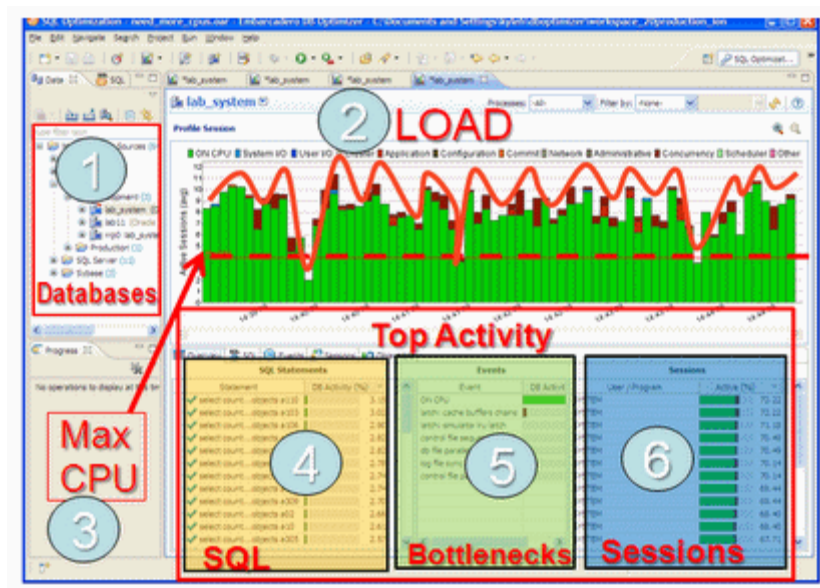
Is the database idle, working or bottlenecked?

When a bottleneck happens how can you know which of these problems are causing the problem? A bottleneck could be caused by:

- An application problem
- An undersized machine
- SQL requiring optimization
- A misconfigured database

All of these can be easily identified from DB Optimizer's performance profiling screen.

Let's look at the components of the performance profiling screen



The screen has six important parts

1. **Databases.** For more information, see "[Databases](#)" on page 121.
2. **Average Active Sessions (AAS) Load of selected database.** For more information, see "[Average Active Sessions \(AAS\) Load of selected database](#)" on page 121.
3. **Maximum CPU line.** For more information, see "[Maximum CPU line](#)" on page 121.
4. **Top SQL.** For more information, see "[Top SQL, Top Bottlenecks, and Top Sessions](#)" on page 122.
5. **Top Bottlenecks.** For more information, see "[Top SQL, Top Bottlenecks, and Top Sessions](#)" on page 122.
6. **Top Sessions.** For more information, see "[Top SQL, Top Bottlenecks, and Top Sessions](#)" on page 122.

Databases

First, on top left, is a list of our databases we have registered.

Average Active Sessions (AAS) Load of selected database

The most important part of the screen is the Average Active Sessions (AAS) graph. AAS shows the performance of the database measured in the single powerful unified metric AAS. AAS easily and quickly shows any performance bottlenecks on the database when compared to the Maximum CPU line. The Max CPU line is a yardstick for performance on the database. When AAS is larger than the Max CPU line there is a bottleneck on the database. Bottleneck identification is that easy.

AAS or the average number of sessions active, shows how many sessions are active on average (over a 5 second range in DB Optimizer) and what the breakdown of their activity was. If all the users were running on CPU then the AAS bar is all green. If some users were running on CPU and some were doing IO, represented by blue, then the AAS bar will be partly green and partly blue.

Maximum CPU line

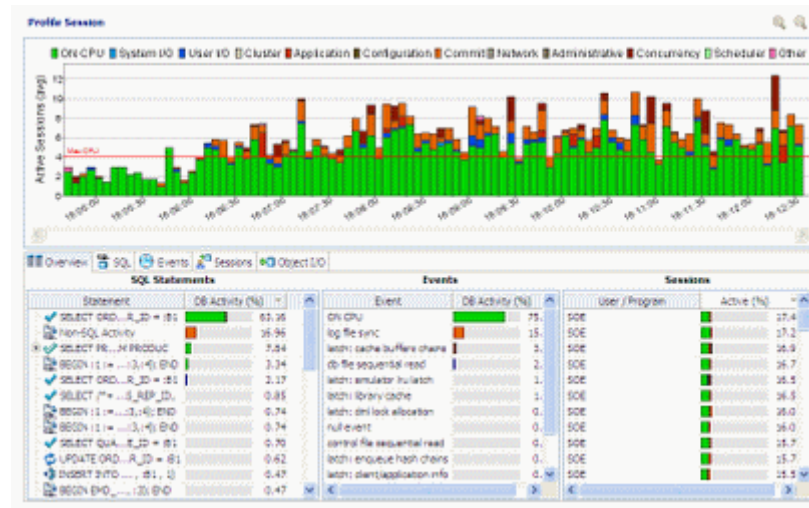
The line "Max CPU" represents the number of CPU processors on the machine. If we have one CPU then only one user can be running on the CPU at a time. If we have two CPUs then only 2 users can be on CPU at any instant in time. Of course users can go on and off the CPU extremely rapidly. When we talk about sessions on the CPU we are talking about the average number of sessions on CPU. A load of one session on the CPU, thus would be an average which could represent one user who is consistently on the CPU or many users who are on the CPU for short time periods. When a CPU becomes a resource bottleneck on the database we will see the average active sessions in CPU state go over the Max CPU line. The number of sessions above the max CPU line is the average number of sessions waiting for CPU.

The Max CPU is a yardstick for performance on the database.

From looking at the previous chart the problem is a machine resource problem.

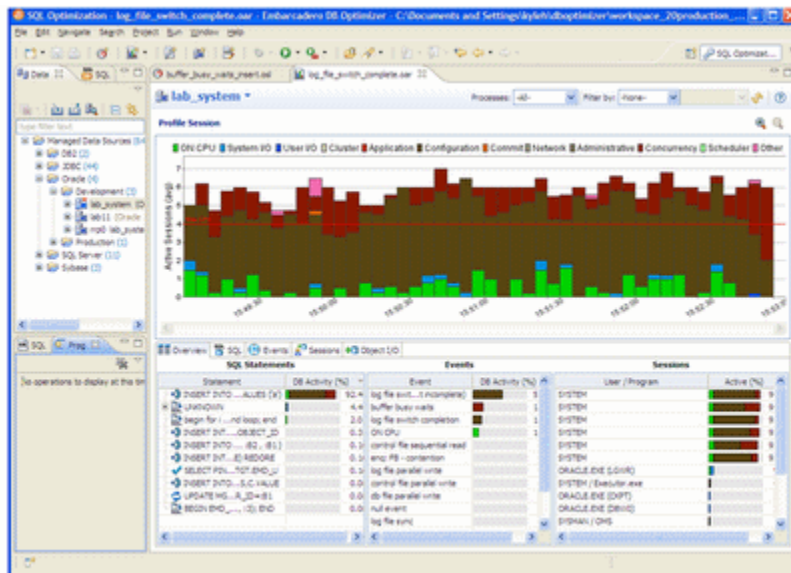
Top SQL, Top Bottlenecks, and Top Sessions

In order to know what the problem is, we have to find out where that demand is coming from. To find out where the demand is coming from we can look at Top SQL and Top Session tables below the load chart. In our case shown here the load is well distributed over all SQL in Top SQL and all sessions in Top Session. There is no outlier or resource hog. In this case it's the machine that's underpowered. What does a case look like where we should tune the application? The following screenshot depicts such a problem.



In this case, again the CPU demand is more than the machine can supply but if we look at "Top SQL" we can see that the first SQL statement (with the large green bar) uses up much more CPU than any of the rest, actually 60%! If we could get it down to 10% CPU then we'd save 50% of the CPU usage on the machine! Thus in this case it's worth our while to spend a day or week or even a couple weeks trying to tune that one SQL statement instead of buying a bigger machine.

Finally, how do we know when the database configuration is a problem? We know it's a configuration problem when we are seeing something other than CPU as the bottleneck in Top Bottleneck section. Here's an example



In this case we can see the load is higher than the Max CPU line but the load is coming from brown colored bars and the green CPU colored bars. If we look at Top SQL we see that there is only one SQL taking up almost all the load, but it's not because of CPU which would be a green bar, but some other color. What does this other color represent? We can look at the Top Bottleneck section and see that it is "log file switch (incomplete)" which basically means the log files are too small, the database is not correctly configured. This bottleneck can be resolved simply by increasing the log size.

TUNING EXAMPLE

This example is comprised of the following parts:

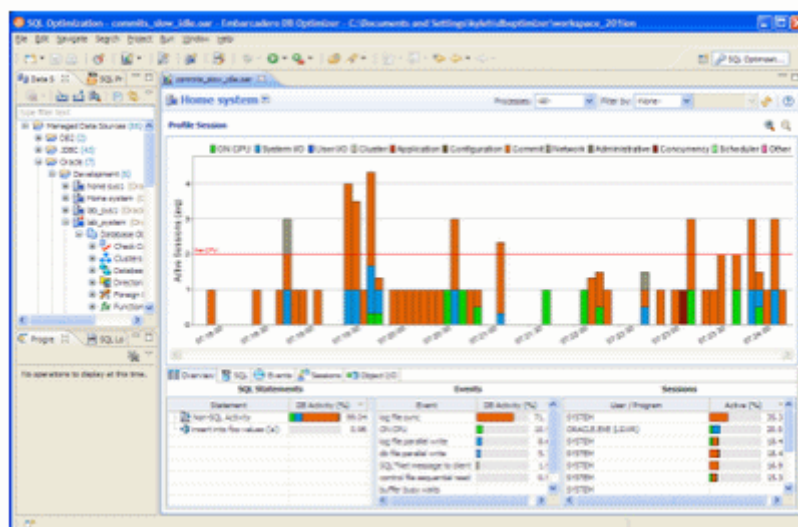
- [The Database is Hanging or the Application has Problems](#) on page 123
- [The Database Caused the Problem](#) on page 125
- [The Machine Caused the Problem](#) on page 126

THE DATABASE IS HANGING OR THE APPLICATION HAS PROBLEMS

I wonder if you can imagine, or have had the experience of the application guys calling with anger and panic in their voices saying, "The database is so slow, you've got to speed it up."

What's your first reaction? What tools do you use? How long does it take to figure out what's going on?

Let's take a look at how it would work with DB Optimizer.



WE can clearly see that the database is not bottlenecked and there must be a problem on the application.

Why do we think it's the application and not the database? The database is showing plenty of free CPU in the load chart, the largest chart, on the top, in the image above. In the load chart, there is a horizontal red line. The red line represents the number of CPUs on the system, which in this case is 2 CPUs. The CPU line is rarely crossed by bars which represent the load on the database, measured in average number of sessions. The session activity is averaged over 5 samples over 5 seconds, thus bars are 5 seconds wide. The bars above fall mostly about one average active session and the bars are rarely green. Green represents CPU load. Any other color bar indicates a sessions waiting. The main wait in this case is orange, which is log file sync, waits for commits. Why is the database more or less idle and why are most of the waits we do see for "commit"? When we look at the code coming to the database we see something like this:

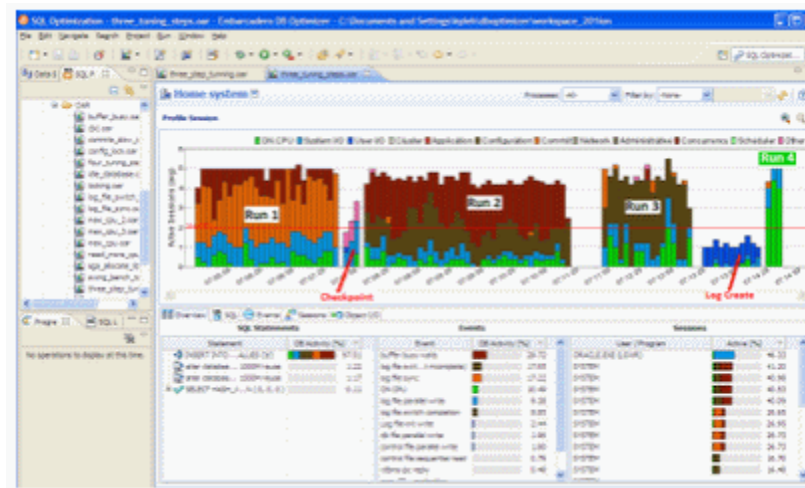
```
insert into foo values ('a');  
commit;  
insert into foo values ('a');  
commit;  
insert into foo values ('a');  
commit;  
insert into foo values ('a');  
commit;  
insert into foo values ('a');  
commit;  
insert into foo values ('a');  
commit;
```

Doing single row inserts and committing after each is very inefficient. There is a lot of time wasted on network communication which is why the database is mainly idle. When the application thinks it's running full speed ahead, it is actually waiting mainly on network communication and commits. If we commit less and batch the work we send to the database, reducing network communications, we will run much more efficiently. Changing the code to

```
begin
  for i in 1..1000 loop
    insert into foo values ('a');
    -- commit;
  end loop;
end;
/
commit;
```

improves the communication delay and now we get a fully loaded database but we run into database configuration issues.

THE DATABASE CAUSED THE PROBLEM



In the above DB Optimizer screen, the same workload was run 4 times. We can see that the time (width of the load) reduced, and the percent of activity on CPU increased.

Runs:

1. "log file sync", the orange color, is the biggest color area, which means users are waiting on commits, still even though we are committing less in the code. In this case we moved the log files to a faster device. (you can see the checkpoint activity just after run 1 where we moved the log files)

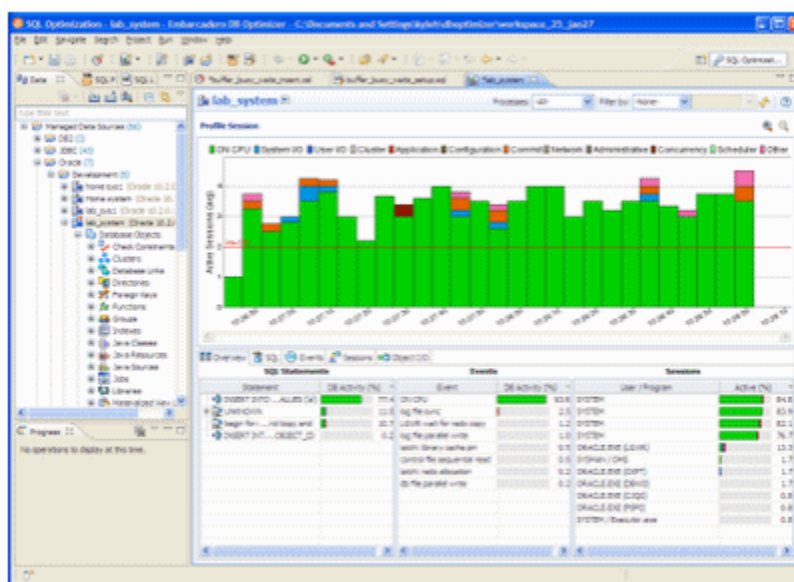
2 "buffer busy wait", the burnt red, is the biggest color area. We drilled down on the buffer busy wait event in the Top Event section and the details tells use to move the table from a normal tablespace to an Automatic Segment Space Managed tablespace.

3. "log file switch (checkpoint incomplete)", the dark brown, is the largest color area, so we increased the size of the log files. (You can see the IO time spent creating the new redo logs just after run 3.)

4. The run time is the shortest and all the time is spent on the CPU which was our goal, to take advantage of all the processors and run the batch job as quickly as possible.

THE MACHINE CAUSED THE PROBLEM

Now that the application is tuned and the database is tuned let's run a bigger load:

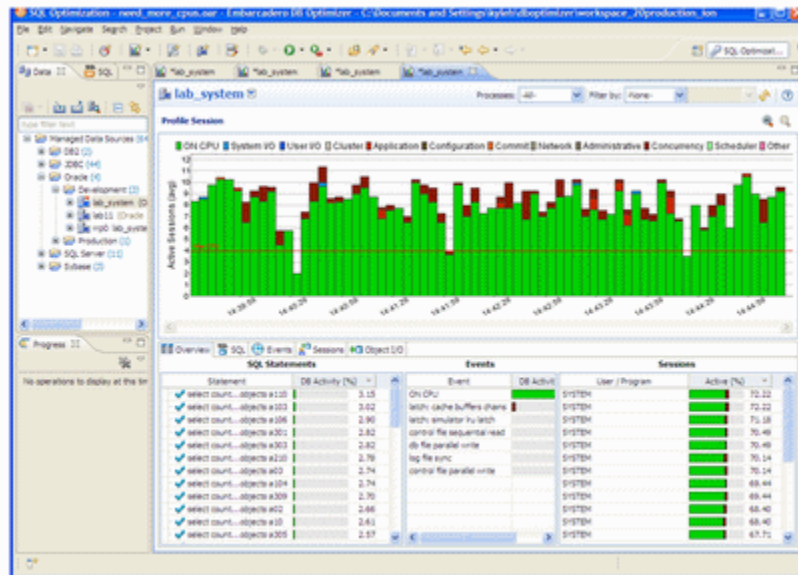


We can see that the CPU load is constantly over the max CPU line. How can we have a bigger CPU load than there are actually CPUs on the machine? Because the demand for CPU is higher than the CPU available on the machine. In the image above there are 2 CPUs on the machine but an average of three users think they are on the CPU, which means that on average one user is not really on the CPU but ready to run on the CPU and waiting for the CPU.

At this point we have two options. In this case we are only running one kind of load, the insert. For inserts we can actually go even further tuning this insert and use Oracle's bulk load commands:

```
declare
  TYPE IDX IS TABLE OF Integer INDEX BY BINARY_INTEGER;
  MY_IDX IDX;
BEGIN
  for i in 1..8000 loop
    MY_IDX(i) := 1;
  end loop;
  FORALL indx IN MY_IDX.FIRST .. MY_IDX.LAST
  INSERT INTO foo ( dummy )
  VALUES ( MY_IDX(indx) );
  COMMIT;
end;
/
```

But if this was an application that had a lot of different SQL and the SQL load was well distributed across the system then we'd have a case for adding more hardware to the system. Making the decision to add more hardware can be a difficult decision because in general the information to make the decision is unknown, unclear or just plain confusing, but DB Optimizer makes it easy and clear, which can save weeks and months of wasteful meetings and debates. For example

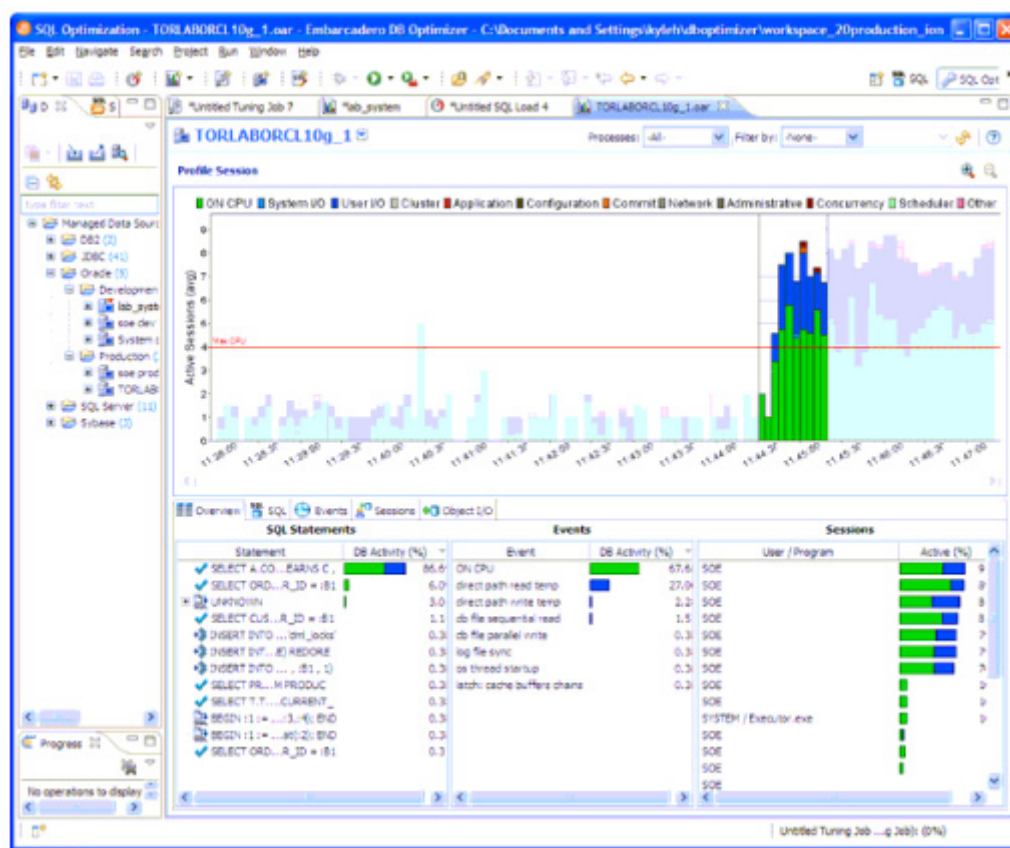


If we look in the bottom left, there is no SQL that takes up a significant amount of load, there is no outlier SQL that we could tune and gain back a lot of wasted CPU. We'd have to tune many SQL and make improvements on most of them to gain back enough CPU to get our load down below the max CPU line. In this case, adding CPUs to the machine might be the easiest and most cost effective solution.

Conclusion:

- With the load chart we can quickly and easily identify the bottlenecks in the database, take corrective actions, and see the results. In part 1 we had an application problem, in part 2 we had 3 database configuration issues and in part 3 we had a hardware sizing issue. In all 3 instances DB Optimizer provides a clear and easy presentation of the data and issues making solutions clear.

FINDING AND TUNING PROBLEM SQL



DB Optimizer is targeted at finding problem sql in a running load with the profiler and then tuning that (or those) specific queries with the tuner.

It's not efficient just to dump a bunch of procedure code into the tuner and then try and see if any of the SQL in the package or procedure are tunable. Most queries should, by default, run optimally on a database, so the goal of DBO is to tune those queries that for one reason or another are not optimally tuned by the database by default. The easiest way to find those queries is to identify them on a running system. They can be identified on a running system because they take up a lot of resources. If we find a resource intensive query then it's worth the time to generate cases and analyze it for missing indexes to see if there is a way to tune it.

UNDERSTANDING THE TUNER INTERFACE

In the application interface, tuning is composed of three tabs:

- [Input](#)
- [Overview](#)

- [Analysis](#)

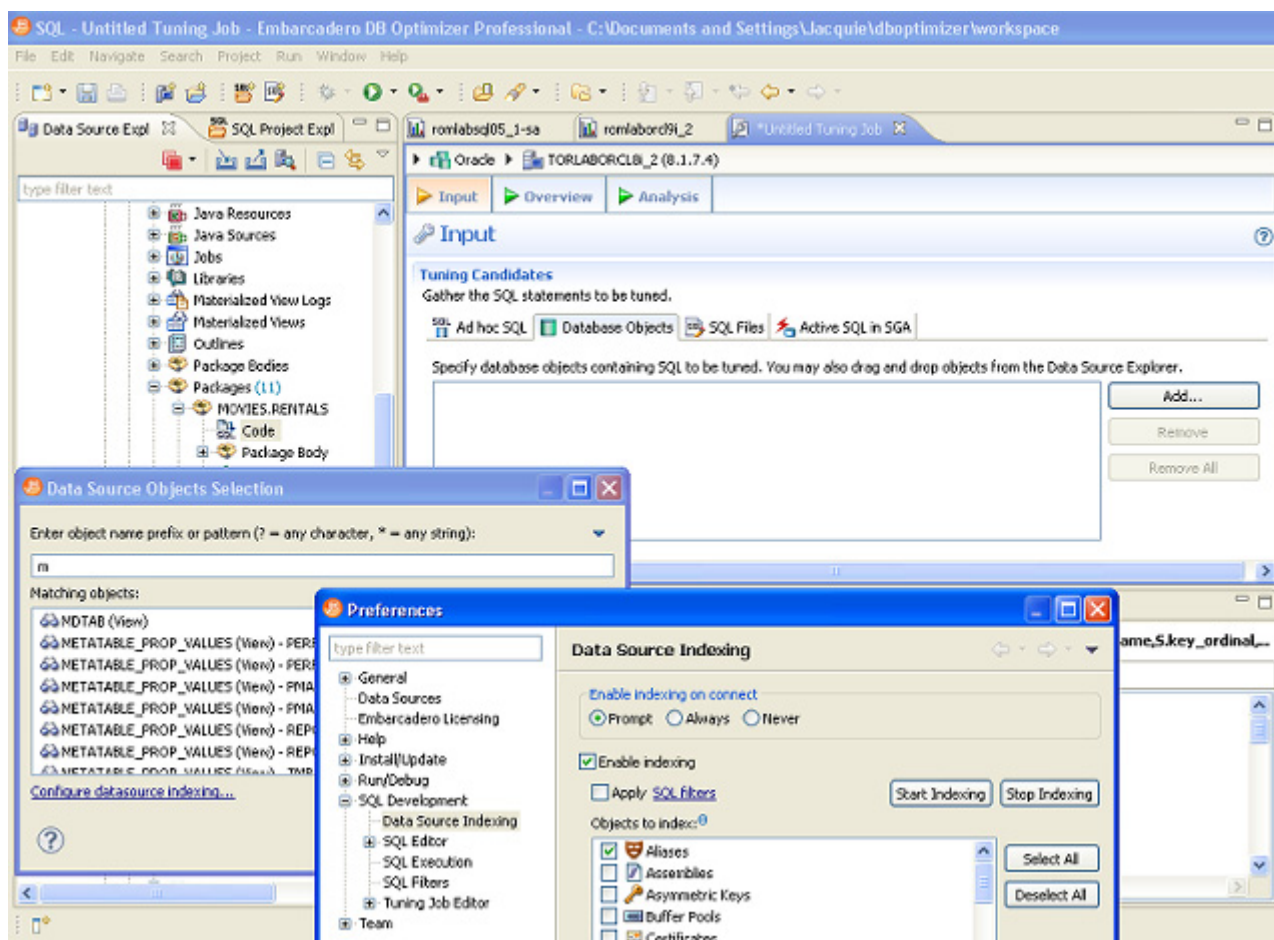
NOTE: When using tuning on Oracle sources, several additional tabs appear on the Analysis and Outlines tabs. For more information on utilizing these extra features, see "[Using Oracle-Specific Features](#)" on page 171.

UNDERSTANDING THE INPUT TAB

Use the Input tab to specify which SQL statements to tune.

- **Ad hoc SQL:** Copy/paste SQL statements to the Ad hoc SQL tab or write queries by hand.

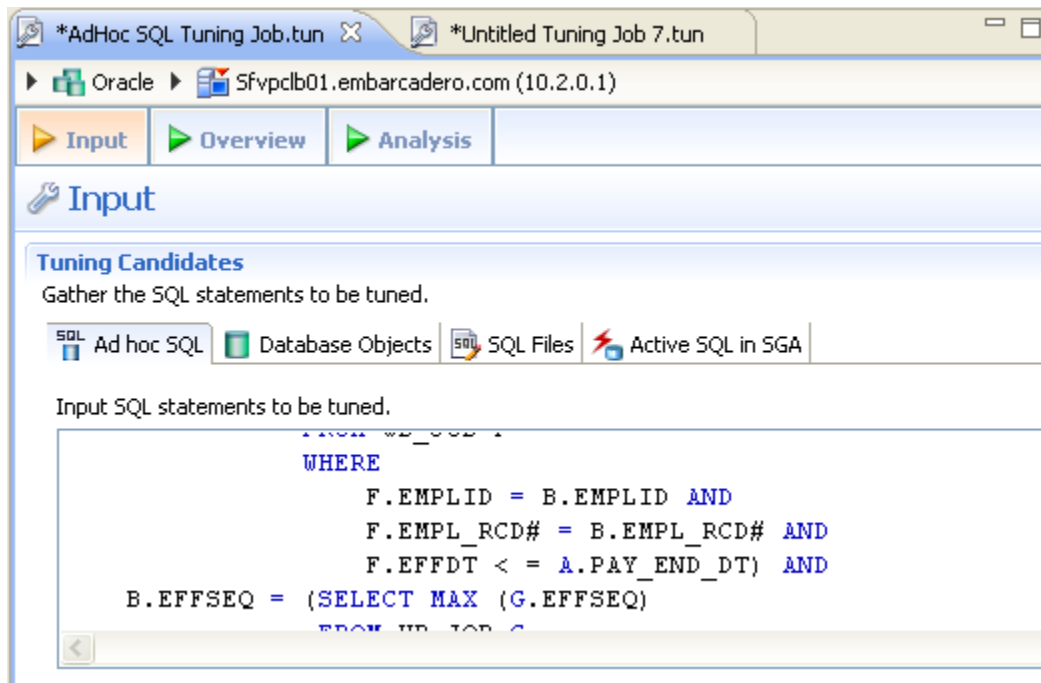
- **Database Objects:** Drag and drop data base objects containing SQL that you want to tune from the Data Source Explorer to the Database Objects tab. Alternatively, you can click Add and enter the first few letters of the database object. DB Optimizer will search through the database to find objects matching your input and presents matches for you to choose. In order for this option to work, you must enable Data Source Indexing in the properties for the database. If the data source has not already been indexed you will receive a message indexing that no indexing information is available. You can configure the database Properties dialog from the **Data Source Objects Selection** dialog by clicking **Configure datasource indexing...**



For information on setting data source indexing properties, see "[Set Index Configuration Preferences](#)" on page 15.

- **SQL Files:** Browse the workspace or file system and select SQL files.

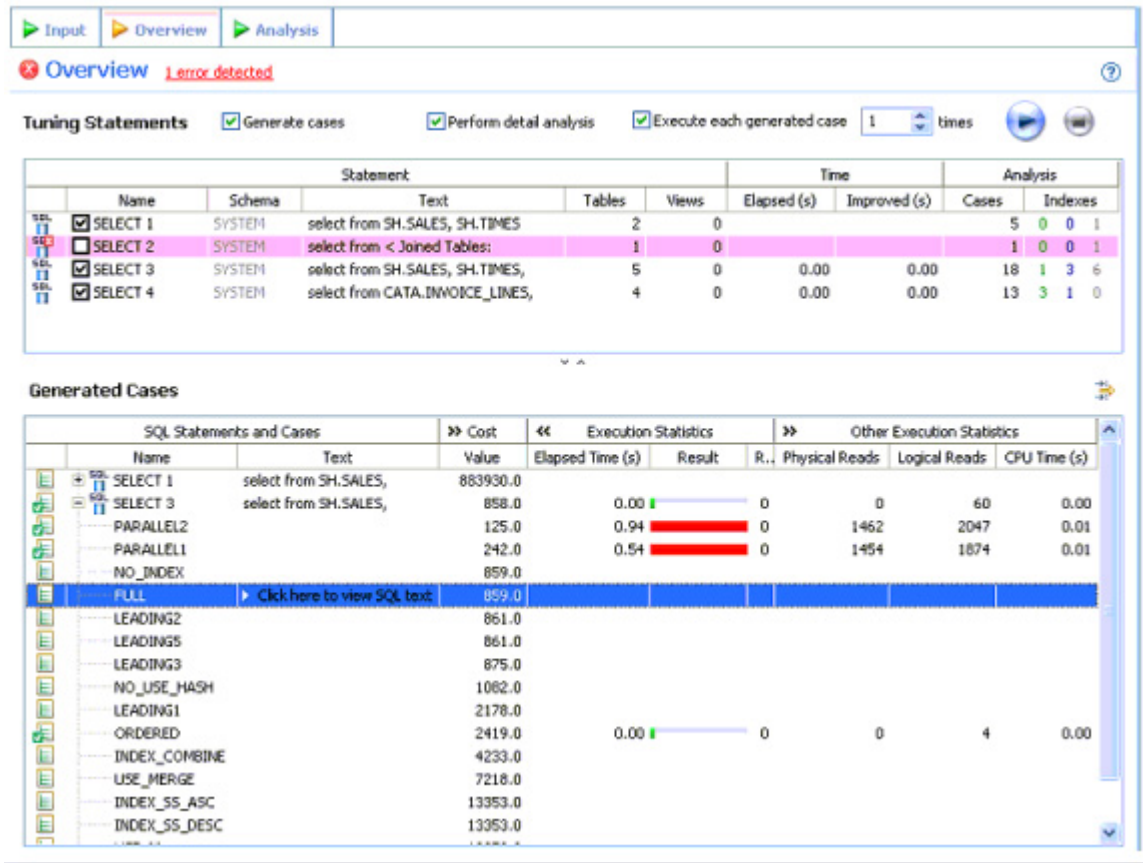
- **Active SQL in SGA:** For the Oracle platform only, you can also scan the System Global Area (SGA) for statements to tune.



UNDERSTANDING THE OVERVIEW TAB

Once you click the Run Job icon on the top right-hand side of the Overview tab, the Overview tab provides the list of statements that were analyzed by the Tuner, as well as the cases suggested by the execution process to improve them. Additional information may include statement Name, Text, Source, Cost, and Elapsed Time values, depending on the platform.

Only the Elapsed Time statistic, appears on all supported platforms. On Oracle platforms, Execution Statistics and Other Execution Statistics columns will appear. When determining the best possible path using the Overview tab, it is best to use the Elapsed Time value as the guideline. The faster the path, the more optimized the query will become.



There are three tuning options to choose from before clicking Run Job:

To analyze the SQL statement, click **Generate cases**

To perform the analysis that populates the Analysis tab now, click Perform detail analysis. Otherwise, the analysis tab is populated when you click the **Analysis** tab.

To have the system generate execution statistics, click **Execute each generate case** and then select the number of time the system should execute each generated case. Multiple executions can verify that the case results are not skewed by caching. For example, the first time a query is run, data might be read off of disk, which is slow, and the second time the data might be in cache and run faster. Thus, one case might seem faster than another but it could be just benefiting from the effects of caching. Generally, you only need to execute the cases once, but it may be beneficial to execute the cases multiple times to see if the response times and statistics stay the same.

UNDERSTANDING THE ANALYSIS TAB

Index analysis is started when you either generate cases with **Perform detail analysis** selected on the **Overview** tab, or when you click the **Analysis tab**. If any columns referenced in the WHERE clause of the tuning candidate are not the first column of an index, tuning will recommend that you create an index on that column.

The color-coded Index Analysis feature highlights missing indexes as well as shows which indexes are used and which are not used in the default execution path. The Index Analysis feature highlights issues where the database optimizer might not be using the preferred indexes. DB Optimizer also lists indexes on the tables that do not have fields in the WHERE clause helping the designer to see if adding an additional predicate in the WHERE clause might make use of an existing index.

The layout of the Analysis tab shows the SQL text and Visual SQL Tuning (VST) diagram on the top, and the indexes on the tables in the query below.

The screenshot displays the SQL Analysis tab with five numbered components:

- Statement selector:** A dropdown menu at the top right labeled "Select statement of interest:" with "SELECT 1" selected.
- Statement text:** A text area on the left containing the SQL query:


```
SELECT *
FROM sh.countries
WHERE country_region_id = 1
```
- Graphical diagram:** A Visual SQL Tuning (VST) diagram on the right showing the table "COUNTRIES" with columns "COUNTRY_REGION_ID: NUMBER" and "COUNTRY_ID". It also shows indexes "COUNTRIES_PK" and "IDX_COUNTRIES_0".
- Index analysis table:** A table at the bottom titled "Collect and create indexes" with columns: Index Name, Table Owner, Table Name, Column Name, and Index Type.

Index Name	Table Owner	Table Name	Column Name	Index Type
<input checked="" type="checkbox"/> IDX_COUNTRIES_0	SH	COUNTRIES	COUNTRY_REGION_ID	Normal
<input type="checkbox"/> COUNTRIES_PK	SH	COUNTRIES	COUNTRY_ID	Unique
- Index analysis details:** A text area on the right providing a detailed explanation: "Table SH.COUNTRIES is scanned via full table scan but it has a filter: country_region_id = 1 on it and we created a virtual index: IDX_COUNTRIES_0 which the optimizer picked up, so we suggest implementing this index."

The Analysis tab has five important components as depicted in the previous illustration:





- 1 **Statement selector**, if there are multiple statements in the tuning set.
- 2 **Statement text** for selected statement.
- 3 **Graphical diagram** of the SQL statement.
- 4 **Index analysis** of the SQL statement.

5 **Description of the selected index**, including the reasoning behind DB Optimizer recommendations.

NOTE: For the Oracle platform, there are several other tabs available, including Table Statistics, Column Statistics And Histograms, and Outlines. For more information, see "[Using Oracle-Specific Features](#)" on page 171.

TIP: The text, diagram, and analysis sections can be resized or expanded to take up the whole page.

The Analysis tab suggests missing indexes, indicates which indexes are used in the execution path and lists all indexes that exist on all the tables in the query. Indexes on the table are listed on the Analysis tab and color coded as follows:

Text Color	Interpretation
	Index is used in the query
	Index is usable but not used in the current execution path.
	This index is missing. DB Optimizer recommends that you create this index.
	This index exists on the table but not usable in this query as it is written.

From the example illustrated above, we can see the following:

```
SELECT *
  FROM
    client_transaction ct,
    client c
 WHERE
    ct.transaction_status = c.client_marital_status AND
    c.client_first_name = 'Brad'
```

Since there is no index on CLIENT.CLIENT_FIRST_NAME and there are 5600 records in CLIENT, DB Optimizer proposes creating an index:

In the **Collect and Create Indexes** table, orange-highlighted entries indicate missing indexes that DB Optimizer recommends be created to improve performance. Clicking on that index, as shown in the illustration that follows, displays text to the right outlining the rational behind this recommendation.

The screenshot displays the 'SQL Analysis' window with three tabs: 'Input', 'Overview', and 'Analysis'. The 'Analysis' tab is active, showing a query in the left pane and its execution plan in the right pane. Below the panes is a table titled 'Collect and create indexes'.

Query:

```
SELECT *
FROM
  client_transaction ct,
  client c
WHERE
  ct.transaction_status =
  c.client_first_name =
```

Execution Plan: The plan shows a join between 'CLIENT_TRANSACTION (ct)' and 'CLIENT (c)'.

Collect and create indexes table:

Index Name	Table Owner	Table Name
IDX_CLIENT...SACTION_0	SYSTEM	CLIENT_TRANSACTION
CLIENT_MULTI	SYSTEM	CLIENT
CLIENT_BROKER	SYSTEM	CLIENT
CLIENT_INCOME	SYSTEM	CLIENT

For more information on using the Analysis tab, see "[Using the Analysis Tab](#)" on page 149.

TUNING SQL STATEMENTS

A tuning job enables you to view the cost details of SQL statements on a registered data source and then select the best, or most efficient, array of execution path directives in order to make query execution faster, therefore improving the entire enterprise, overall.

There are four methods through which statement tuning can be activated:

- Ad hoc statement tuning via manual entry, or cutting and pasting into the tuning window.

- Database object selection, by selecting stored packages from a list on the registered data source.
- SQL file selection, by choosing an SQL file saved on the system.
- Importing statements directly from profiling.

A tuning job consists of a set of SQL statements and any analysis results you generate against a data source using tuning. The SQL statements and analysis results that compose a tuning job can be saved in a tuning file (.tun). This enables you to open a tuning job at a later time for inspection and analysis, to add, delete, or modify the SQL statements, or generate new execution statistics.

The following topics provide a high-level overview of the tuning process:

- 1 [Create a New Tuning Job](#) on page 136
- 2 [Specify a Data Source](#) on page 137
- 3 [Add SQL Statements](#) on page 138
- 4 [Run a Tuning Job](#) on page 140
- 5 [Analyze Tuning Results](#) on page 143
- 6 [Modify Tuning Results](#) on page 147

NOTE: For additional commands that fall outside the general tuning workflow, but may still be helpful, see "[Additional Tuning Commands](#)" on page 176.

CREATE A NEW TUNING JOB

New tuning jobs can be created via the File > New > Tuning Job command, or by importing statements directly from profiling. A New Tuning Job icon is also available on the Toolbar.

To create a new tuning job via the Menu or Icon command:

Select **File > New > Tuning Job**, or click the **New Tuning Job** icon on the Toolbar. Tuning opens.

You can now proceed to set up the parameters of the new job.

To create a new tuning job from profiling:

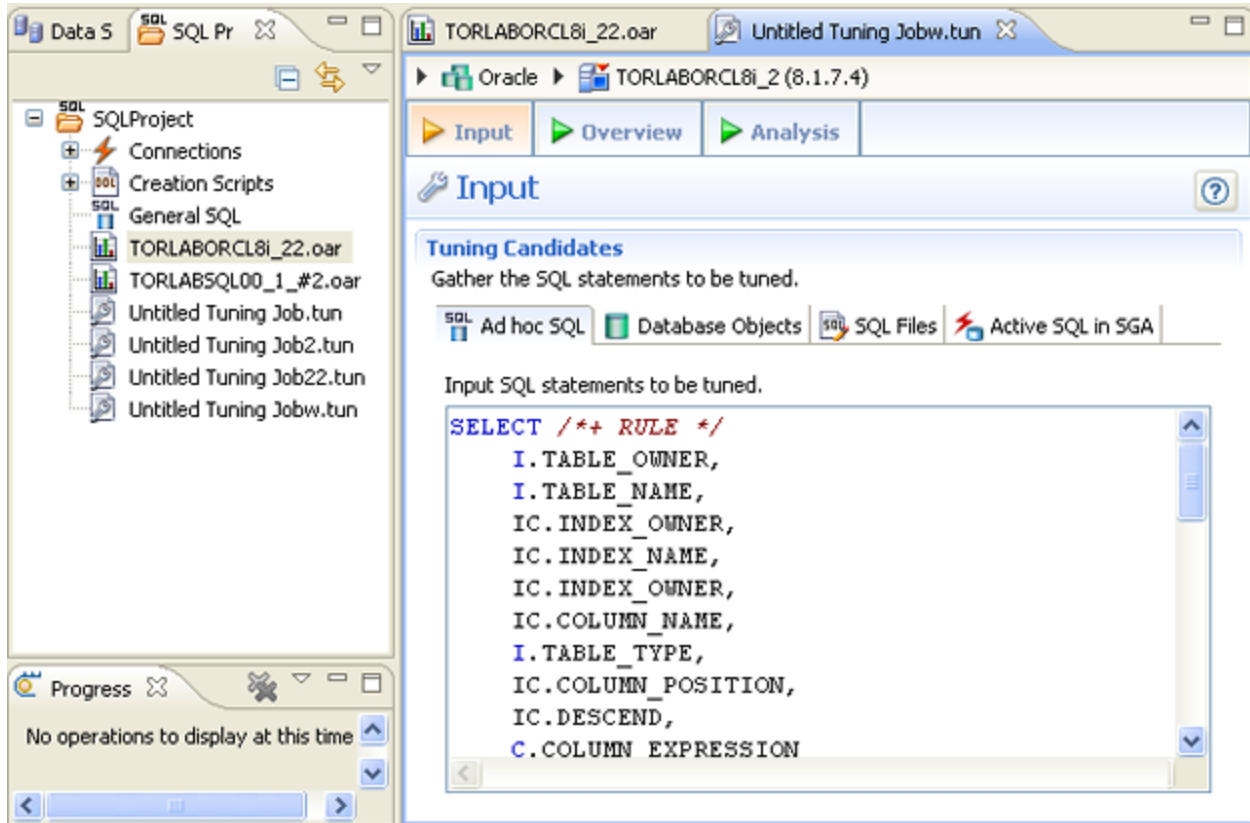
After you have run a profiling session, in profiling's Profiling Details tab, select one or more statements, right-click, and select Tune from the context menu. Tuning opens, pre-populated with parameters based on the statements you selected.

To open an existing tuning job:

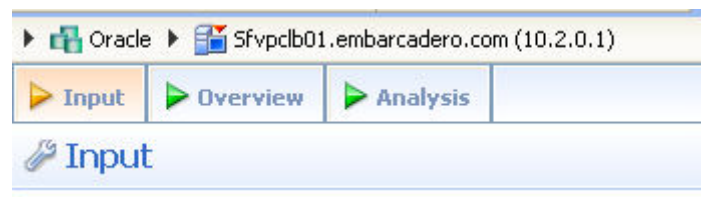
Navigate to the SQL Project tab and double-click the name of the existing tuning job.

To name a job, save it:

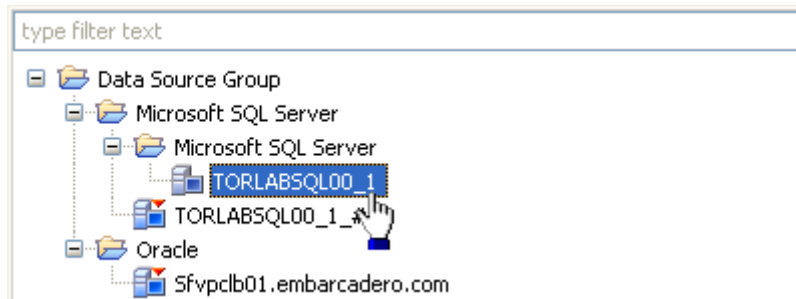
Ensure you specify a meaningful name that identifies the job in other views and dialogs. You can save the job by selecting **File > Save or File > Save All** from the Menu bar. Once a job is saved, it is added to the SQL Project view.

**SPECIFY A DATA SOURCE**

The bread crumbs at the top of the tuning job window identify the data source where the SQL statements to be tuned reside. The default data source is the one that was selected when the new tuning job was initiated. For example in the following image, we see that the data source is Sfvpc1b01.embarcadero.com., which is part of the Oracle data source group.



You can change the data source of a tuning job by clicking a breadcrumb triangle and then navigating to the datasource or using the filter to locate and then select a data source. In the following screenshot, Microsoft SQL Server was clicked and *T* was entered in the filter text area, which resulted in several matches.



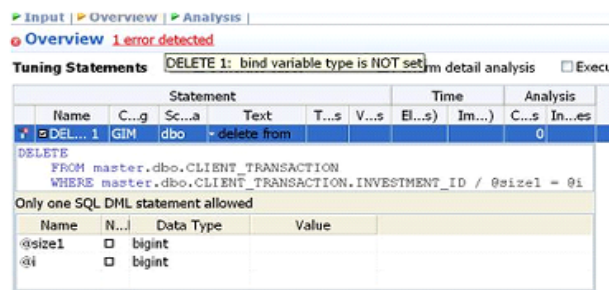
Click the name of the desired data source to affect the change.

NOTE: Multiple tuning jobs can be saved against the same data source. You can therefore set up your tuning jobs organizationally. You might for example, set up a tuning job to tune only SQL associated with procedures or a set of SQL sources that are functionally related. Alternatively, your tuning jobs may be organized by application.

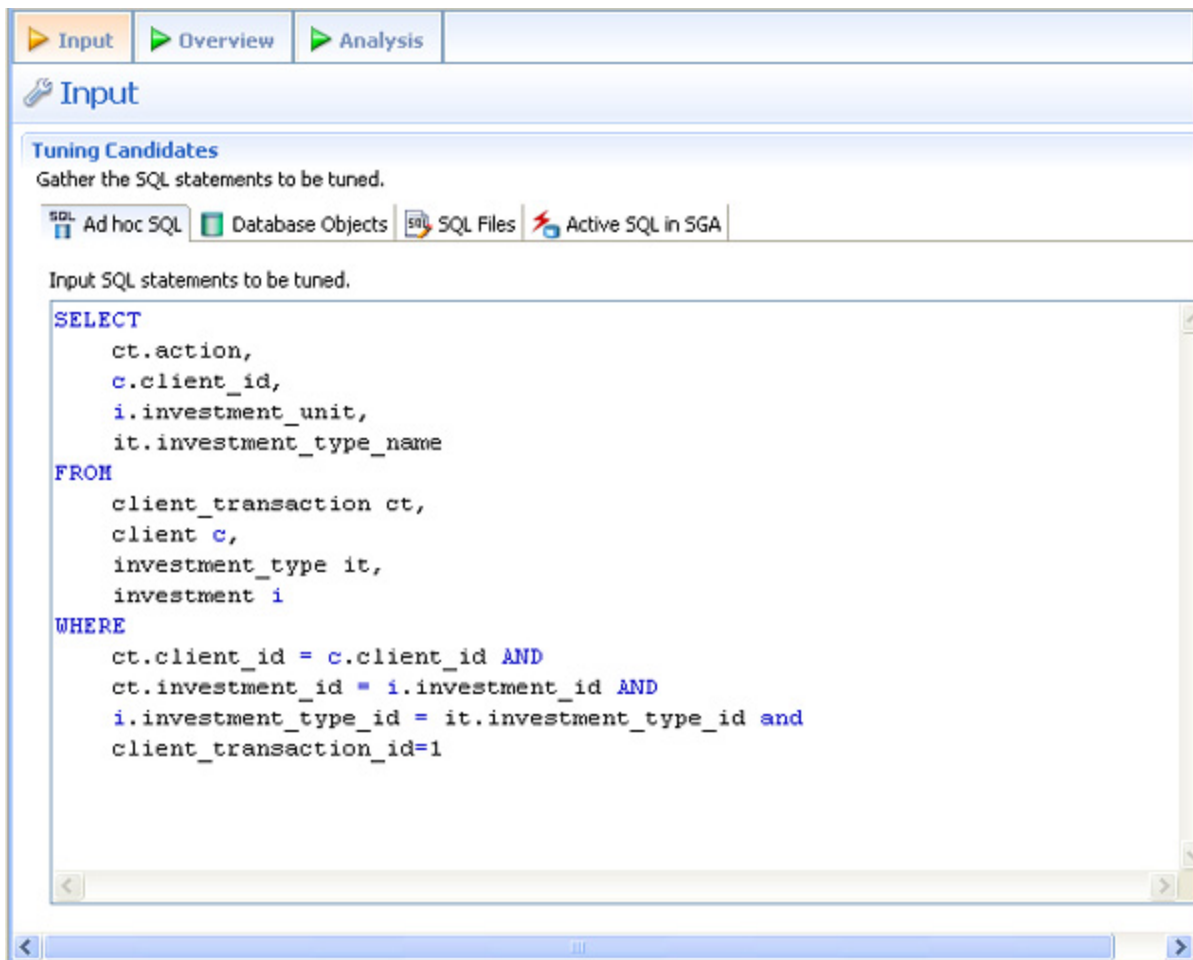
ADD SQL STATEMENTS

Once you have created a tuning job and named it, using **File > Save As**, you need to add SQL statements to the job that are to be tuned. All standard DML statements can be tuned (SELECT, INSERT, DELETE, and UPDATE).

NOTE: When you try to tune a statement containing a bind variable you will be warned that either the type is not set or the value is not set.



Statements are added to tuning via the Tuning Candidates pane.



There are several different methods for adding SQL statements to a job, as reflected by the tabs in the Tuning Candidates box:

- The **Ad hoc SQL** tab enables tuning via manual entry, or cutting and pasting into the tuning window.
- The **Database Objects** tab enables you to select stored SQL from the data source to which you are connected. You can either drag and drop objects from the Data Source Explorer or you can add database objects matching specified filters. For example, entering t in the filter area of the Data Source Objects Selection dialog, can match functions, views, and procedures, whose name begins with t.
- The **SQL Files** tab enables you to choose an SQL file saved on the system.
- The **Active SQL in SGA** tab is available for the Oracle platform only. It enables you to scan for and select active SQL in the System Global Area (SGA). For more information, see "[Tuning SQL Statements in the System Global Area \(SGA\)](#)" on page 175

To add an ad hoc SQL statement:

Select the Ad hoc SQL tab and manually type an SQL statement in the window, or copy/paste the statement from another source.

To add a database object:

- 1 Select the **Database Objects** and click **Add**. The Data Source Object Selection dialog appears.
- 2 Type an object name prefix or pattern in the field provided. The window below automatically populates with all statements residing on the specified data source that match your criteria. Database objects include functions, materialized views, packages, package bodies, procedures, stored outlines, triggers, and views.
- 3 Double-click on the statement you want to add. You can click **Add** again to repeat the process and add more objects to the job.

NOTE: Alternatively, after clicking the Database Objects tab, you can drag and drop objects from Data Source Explorer into the Database Objects window. As long as the dragged object is a valid object type, it will be added to the Database Objects tab.

To add an SQL file:

- 1 Select SQL Files and click Workspace or File System, depending on where the file you want to add is stored:
 - **Workspace files** are files that reside in the application, meaning project files or other objects generated or stored in the system.
 - **File System files** are files that reside on your machine or the network.
- 2 Select a file from the dialog that appears. It is automatically added to the job.

RUN A TUNING JOB

As you add SQL statements to the job on the Input tab of the tuner, tuning-supported DML statements (SELECT, INSERT, DELETE, and UPDATE) are parsed from the statements and added to the Overview tab in preparation for the tuning function execution.

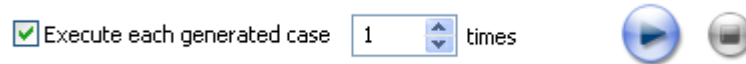
Each tuning source statement is listed by Name, Schema, Text, Tables and Views. For SQL Server and Sybase platforms, there is also a Catalog column. Additionally, each statement will have Time and Analysis values that approximate how efficiently they execute on the specified data source.

In the Generated Cases area of the Overview tab of a tuning job, the Cost and Execution Statistics columns let you compare the relative efficiency of SQL statements or statement cases. While the explain plan Cost for a statement or case is calculated when you add SQL to a tuning job, the Elapsed Time and Execution Statistics columns are not populated until you execute that statement or case.

If the Tuning Status Indicator indicates that a statement or case is ready to execute, you can execute one or more statements on the **Overview** tab. Alternatively, the Tuning Status Indicator may show that you have to correct the SQL or set bind variables before you can execute.

Once the tuning job has run, the Overview tab provides a series of cases, per statement, that you can select and modify based on your results.

In some cases, automatic case generation might be disabled (via the Preferences panel). If this is true, or if you have otherwise modified the Generated Cases table and can no longer generate a specific case, you can instead explicitly generate a case for specific statements.



To execute a tuning job:

- 1 Ensure you have registered and selected a data source. For more information, see "[Register Data Sources](#)" on page 24 and "[Specify a Data Source](#)" on page 137.
- 2 Ensure you are connected to the database by double clicking the database name in the Data Source Explorer.
- 3 Click the tuning icon on the toolbar, or click **File > New > Tuning Job**.
- 4 On the **Input** tab, specify the SQL you want to tune:
 - On the **Ad hoc SQL** tab, enter SQL statements of copy/paste SQL statements from another source.
 - Click the **Database Objects** tab and then click and drag database objects, such as Procedures, from the Data Source Explorer to the Database Object tab.
 - Click the **Database Objects** tab and then click Add to choose database objects matching the filter you provide.
 - Click the **SQL Files** tab and navigate to the SQL file you want to tune.
- 5 Navigate to the **Overview** tab and modify the number of times to execute each statement in the **Execute each generated case** field, as needed.
- 6 Click the execution icon to the right side of the case generation field.

The tuning job runs, exacting and analyzing each statement and providing values in the appropriate columns.

To explicitly generate a case for a specific statement:

- 1 Ensure you are connected to the database by double clicking the database name in the Data Source Explorer.
- 2 Navigate to the **Overview** tab.
- 3 In the **Generated Cases** area, right-click in the **Name** field of a statement or transformation case and select **Generate Cases** from the context menu, or click the **Overview Run Job** icon. The specified case is generated.

To view the generated cases for a specific statement

- 1 In the **Tuning Statements** area, click the checkbox to the left of the tuning source statement name.

A check mark appears in the checkbox and the cases displayed in the Generated Cases area are filtered to display only those cases related to the selected source statement.

Input

Overview

Analysis

Overview

Error detected

Tuning Statements

☒Generate cases

☒Perform detail analysis

☒Execute each generated case

1 times

Statement						Time		Analysis	
	Name	Schema	Text	Tables	Views	Elapsed (s)	Improved (s)	Cases	Indexes
<input checked="" type="checkbox"/>	SELECT 1	SYSTEM	select from SH.SALES, SH.TIMES	2	0			5	0 0 1
<input checked="" type="checkbox"/>	SELECT 2	SYSTEM	select from < Joined Tables:	1	0			1	0 0 1
<input checked="" type="checkbox"/>	SELECT 3	SYSTEM	select from SH.SALES, SH.TIMES,	5	0	0.00	0.00	18	1 3 6
<input checked="" type="checkbox"/>	SELECT 4	SYSTEM	select from CATA.INVOICE_LINES,	4	0	0.00	0.00	13	3 1 0

Generated Cases

SQL Statements and Cases		Cost	Execution Statistics		Other Execution Statistics				
	Name	Text	Value	Elapsed Time (s)	Result	R..	Physical Reads	Logical Reads	CPU Time (s)
<input checked="" type="checkbox"/>	SELECT 1	select from SH.SALES,	883930.0						
<input checked="" type="checkbox"/>	SELECT 3	select from SH.SALES,	858.0	0.00	0		0	60	0.00
<input checked="" type="checkbox"/>	PARALLEL2		125.0	0.94	0		1462	2047	0.01
<input checked="" type="checkbox"/>	PARALLEL1		242.0	0.54	0		1454	1874	0.01
<input checked="" type="checkbox"/>	NO_INDEX		859.0						
<input checked="" type="checkbox"/>	FULL	Click here to view SQL text	859.0						
<input checked="" type="checkbox"/>	LEADING2		861.0						
<input checked="" type="checkbox"/>	LEADING5		861.0						
<input checked="" type="checkbox"/>	LEADING3		875.0						
<input checked="" type="checkbox"/>	NO_USE_HASH		1062.0						
<input checked="" type="checkbox"/>	LEADING1		2178.0						
<input checked="" type="checkbox"/>	ORDERED		2419.0	0.00	0		0	4	0.00
<input checked="" type="checkbox"/>	INDEX_COMBINE		4233.0						
<input checked="" type="checkbox"/>	USE_MERGE		7218.0						
<input checked="" type="checkbox"/>	INDEX_SS_ASC		13353.0						
<input checked="" type="checkbox"/>	INDEX_SS_DESC		13353.0						

ANALYZE TUNING RESULTS

Once you have executed a tuning job, the Overview tab reflects tuning analysis of the specified statements. The **Analysis tab** also shows the resulting analysis of the query, including indexes used, not used, and missing (or suggested to create). For more information on using the Analysis tab, see "[Understanding the Analysis Tab](#)" on page 133

The screenshot shows the DB Optimizer Overview tab with the following components and annotations:




- Tuning Status Indicator:** A yellow warning icon and the text "1 warning detected" in the Overview tab header.
- Enable Execution Check Box:** Checkboxes for "Generate Cases", "Perform detail analysis", and "Execute each generated case" under the "Tuning Source Status" section.
- Run/Cancel Job Controls:** Blue "Run" and grey "Cancel" buttons.
- Column Set Expand/Collapse Control:** A small icon next to the "Statement" table header.
- Increase/Decrease Pane Size Control:** A vertical slider control between the "Statement" and "Generated Cases" panes.
- Generated Case Expand/Collapse Control:** A small icon next to the "Generated Cases" section header.
- Filter Control:** A search filter icon in the top right of the "Generated Cases" pane.
- Transformation Case:** A case named "[Missing a ...sformation]" highlighted in yellow.
- Hint-Based Cases:** Cases like "INDEX_FFS", "FULL", and "FIRST_ROWS" listed under the "Generated Cases" pane.
- Extracted SQL Statements:** The "Text" column in the "Generated Cases" pane showing the SQL for each case.

Name	Schema	Text	Tables	Views	Time		Cases	Index
					Elapsed (s)	Improved (s)		
SELECT 2	SYSTEM	select from BROKER,			6.32	6.32	0	
SELECT 1	SYSTEM	select from			0.00	0.00	10	

Name	Text	Cost	Elapsed Time (s)	Physical Reads	Logical Reads
SELECT 2	select from BROKER, CLIENT_TRANSACTION,	34014.0	6.32	2	0
[Missing a ...sformation]		274.0	0.03	0	0
SELECT 1	select from client_transaction, client,	4.0	0.00	0	0
USE_HASH		14.0	0.00	0	0
ORDERED		8.0	0.00	0	0
NO_USE_NL		16.0	0.00	0	0
LEADING4		8.0	0.00	0	0
LEADING3		10.0	0.01	0	0
LEADING2		7.0	0.01	0	0
LEADING1		4.0	0.00	0	0
INDEX_FFS		9.0	0.00	0	0
FULL		64.0	0.00	0	0
FIRST_ROWS		4.0	0.00	0	0

- The **Generated case Expand/Collapse** control lets you hide or display the hint-based cases and transformation-based case generated for a statement.
- The **Enable Execution** check boxes let you enable multiple statements or cases for simultaneous execution while the Run/Cancel Job controls let you start and stop simultaneous execution.
- The **Column set Expand/Collapse** controls let you expand a column set to display more of the columns within the table.

- The **Tuning Status** Indicator indicates whether a statement or case is ready to execute or has successfully executed. The following table provides information on the Tuning Status Indicator states:

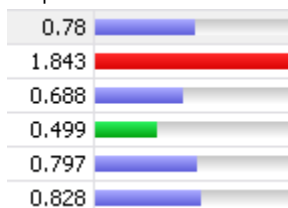
Icon	Description
	The case has not been executed. There are no errors or warnings and the case is ready to be executed.
	The case has been successfully executed.
	Execution for this case failed or was cancelled due to execution time exceeding 1.5 of original case time.

Hovering the mouse over the Tuning Status Indicator displays a tip that notes the nature of a warning or error.

NOTE: If a warning indicates that one or more tables do not have statistics, you can right-click the statement and select **Analyze Tables** to gather statistics.

A warning can indicate an object caching error. For example, a table may not exist or not be fully qualified. Cases cannot be generated for the associated statement.

- The explain plan-based **Cost** field can be expended to display a graphical representation of the values for statements or cases. Similarly, after executing a statement or case, the **Elapsed Time** field can be expended to display a graphical representation. The bar length and colors used in the representation are intended as an aid in comparing values, particularly among cases. For example:



In the case of both **Cost** and **Elapsed Time**, the values for the original statement are considered the baseline values. With respect to color-coding for individual case variants, values within a degradation threshold (default 10%) and improvement threshold (default 10%) are represented with a neutral color (default light blue). Values less than the improvement threshold are represented with a distinctive color (default green). Values greater than the degradation threshold are shown with their own distinctive color (default red).

With respect to bar length, the baseline value of the original statement spans half the width of the column. For child-cases of the original statement, if one or more cases show a degradation value, the largest degradation value spans the width of the column. Bar length for all other children cases is a function of the value for that case in comparison to the highest degradation value.

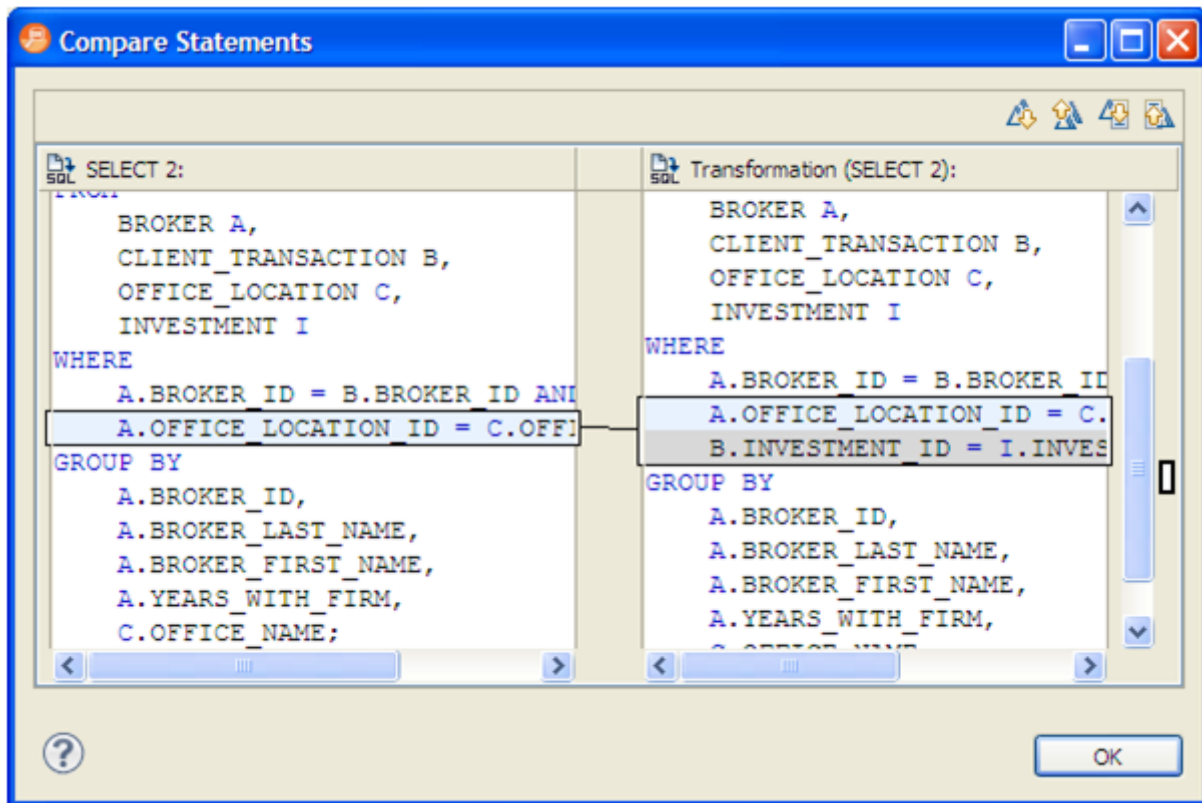
NOTE: For information on specifying colors, and the improvement threshold and degradation threshold values used in these graphical representations, see "[Set Tuning Job Editor Preferences](#)" on page 182.

Additionally, once results have been generated you can:

- **Compare Cases.** For more information, see "[Compare Cases](#)" on page 145.
- **Filter and Delete Cases.** For more information, see "[Filter and Delete Cases](#)" on page 146.
- **Visual SQL Tuning.** For more information, see "[Visual SQL Tuning](#)" on page 151.
- **Create an Outline.** For more information, see "[Create an Outline](#)" on page 146.

COMPARE CASES

You can compare cases between an original statement and one of its tuning-generated statements, or another statement case via the Compare to Parent and Compare Selected commands, respectively



To compare a case side-by-side with its parent:

Right-click in the **Name** field of a case and select **Compare to Parent** from the context menu.

To compare two cases:

Select the two cases then right-click in the **Name** field of either case and select **Compare Selected** from the context menu.

FILTER AND DELETE CASES

You filter cases from the Generated Cases table via the Filter icons on the Generated Cases Toolbar of the **Overview** tab.



Filter the cases on the **Overview** tab so that hints that are not improvements on the original statement are not displayed. You can filter:

- Non-optimizable statements
- Optimized statements
- Worst cost cases
- Worst elapsed time cases

When filtering, the criteria remain in effect until you change the criteria. That is, as new cases are generated, only those cases that do not satisfy the filtering criteria are displayed. To restore an unfiltered set of cases, open the **Filter** dialog and deselect the filtering options.

When removing cases, the criteria you set has no effect on cases subsequently generated.

To filter cases from the Overview table:

- 1 Click the **Filter** button, respectively. A **Filters** dialog opens.
- 2 Use the check boxes to select your filtering and then click **OK**.

To delete cases from the Overview table:

- 1 Right-click on the row of the case you want to delete and select **Delete**. A **Delete** dialog opens.
- 2 Use the check boxes to select your filtering and then click **OK**.

When removing cases, the criteria you set has no effect on cases subsequently generated.

CREATE AN OUTLINE

If SQL is executed by an external application or If you cannot directly modify the SQL being executed but would like to improve the execution performance, you can create an outline on the Oracle platform. An outline instructs the Oracle database on the execution path that should be taken for a particular statement.

To create an outline for a change suggested by a case:

- 1 On the Overview tab of a tuning job, right-click in the **Name** field of a case and select **Create Outline** from the context menu.
A **New Outline** wizard opens.

- 2 On the first panel, provide an **Outline name**, select an **Outline category**, and then click **Next**.
A **Preview Outline** panel opens previewing the SQL code to create the outline.
- 3 Select an **Action to take** option of **Execute** or **Open in new SQL editor** and then click **Finish**.

For more information, see "[Using the Outlines Tab](#)" on page 174.

MODIFY TUNING RESULTS

As you add SQL source to the Input tab of a tuning job, the supported DML statements are automatically parsed out and a numbered statement record for each statement is added to the Overview tab.

Cases generated from tuning candidates are alternative forms of the original statement that have been optimized or otherwise "fixed" by the tuning function. Once you have executed a tuning job, tuning automatically generates all SQL optimizer hint-based variations that can be applied to the statement:

- All SQL Optimizer hint-based variations that can be applied to a statement.

- A transformation-based case, if any of the eight common quick fixes can be applied to an SQL statement. This feature leverages the DB Optimizer Code Quality Check functionality. See "[Understanding Code Quality Checks](#)" on page 51 for more information on the eight quick fixes. A transformation case, in turn, has its own set of SQL Optimizer hint cases. For information on Oracle transformations or query rewrites, see "[Oracle Query Rewrites](#)" on page 149. For information on other transformations, see "[Examples of Transformations and SQL Query Rewrites](#)" on page 185.

The screenshot shows the Oracle SQL Tuning interface. The top navigation bar includes 'Input', 'Overview', and 'Analysis' tabs. The 'Overview' tab is active, showing a warning icon and the text '1 warning detected'. Below the navigation bar, there are checkboxes for 'Generate Cases', 'Perform detail analysis', and 'Execute each generated case'. The 'Execute each generated case' checkbox is checked.

The 'Tuning Source Stat' table lists the following statements:

Name	Schema	Text	Tables	Views	Elapsed (s)	Improved (s)	Cases	Index
SELECT 2	SYSTEM	select from BROKER,			6.32	6.32	0	
SELECT 1	SYSTEM	select from			0.00	0.00	10	

The 'Generated Cases' table lists the following cases:

Name	Text	Cost	Elapsed Time (s)	Physical Reads	Logical Reads
SELECT 2	select from BROKER, CLIENT_TRANSACTION,	34014.0	6.32	2	
[Missing a ...sformation]		274.0	0.03	0	
SELECT 1	select from client_transaction, client,	4.0	0.00	0	
USE_HASH		14.0	0.00	0	
ORDERED		8.0	0.00	0	
NO_USE_NL		16.0	0.00	0	
LEADING4		8.0	0.00	0	
LEADING3		10.0	0.01	0	
LEADING2		7.0	0.01	0	
LEADING1		4.0	0.00	0	
INDEX_FFS		9.0	0.00	0	
FULL		64.0	0.00	0	
FIRST_ROWS		4.0	0.00	0	

Annotations in the image point to the 'Transformation-based case' (SELECT 2) and the 'Hint-based cases' (USE_HASH, ORDERED, NO_USE_NL, LEADING4, LEADING3, LEADING2, LEADING1, INDEX_FFS, FULL, FIRST_ROWS).

Hint-based cases and the transformation-based case are a special case of the statement records added to the Overview tab as you add candidates to a tuning job. With the exception of the Text, Source, and Index Analysis fields, cases are identical to the standard statement record. Similarly, execution, statistics collection, and other options available for basic statement records are available for individual cases.

Once cases have been generated, if you have the required permissions on the specified data source, you can apply the changes suggested by hint and transformation based cases in the Overview table.

To apply a change:

- 1 Right-click on the **Name** field of the case that you want to use to modify the original statement and select **Apply Change**.

The **Apply Change** dialog appears.

- 2 Choose **Execute** to apply the change to the statement automatically.

Alternatively, select **Open in New SQL Editor** to open the modified statement in SQL Editor for manual changes or to save it to a file.

ORACLE QUERY REWRITES

The following query rewrites or transformations may be recommended during tuning for Oracle data sources.

Before	After
select * from t1 where EXISTS (select null from t2 where t2.key = t1.key);	select * from t1 where t1.key IN (select t2.key from t2);
select * from t1 where NOT EXISTS (select null from t2 where t2.key = t1.key);	select * from t1 where t1.key NOT IN (select t2.key from t2 where t2.key is not null);
select * from t1 where t1.key IN (select t2.key from t2);	select * from t1 where EXISTS (select null from t2 where t2.key = t1.key);
select * from t1 where t1.key NOT IN (select t2.key from t2 where t2.key is not null);	select * from t1 where NOT EXISTS (select null from t2 where t2.key = t1.key);
select * from t1 where NOT EXISTS (select null from t2 where t2.key = t1.key);	select t1.* from t1, t2 where t1.key = t2.key(+) and t2.key is null
select * from t1 where t1.key NOT IN (select t2.key from t2 where t2.key is not null);	select t1.* from t1, t2 where t1.key = t2.key(+) and t2.key is null;
select column BETWEEN X AND Y	select (column <= X AND column >= Y)
select column NOT BETWEEN X AND Y	select (column < X AND column > Y)
select (column <= X AND column >= Y)	select column BETWEEN X AND Y
select (column < X AND column > Y)	select column NOT BETWEEN X AND Y

USING THE ANALYSIS TAB

The **Analysis** tab provides detailed information about statements and cases selected from the **Overview** tab, after a tuning job has been executed. It also shows filter ratio, and table and join sizes.

The **Analysis** tab contains information about the statement or case, its full SQL code, a diagram of the SQL statement, and Index Analysis.

The screenshot displays the SQL Analysis tool interface. At the top, there are tabs for **Input**, **Overview**, and **Analysis**. The **Analysis** tab is selected. Below the tabs, the **SQL Analysis** section shows a query window on the left and an execution plan diagram on the right.

SQL Query:

```
SELECT
  ct.action,
  c.client_id,
  i.investment_unit,
  it.investment_type_name
FROM
  client_transaction ct,
  client c,
  investment_type it,
  investment i
WHERE
  ct.client_id = c.client_id;
  ct.investment_id = i.investment_id;
  i.investment_type_id = it.investment_type_id;
  client_transaction id = 1;
```

Execution Plan Diagram:

```
graph TD
    CT[CLIENT_TRANSACTION (ct)] --> C[CLIENT (c)]
    CT --> I[INVESTMENT (i)]
    I --> IT[INVESTMENT_TYPE (it)]
```

Below the query and plan, there are tabs for **Index Analysis**, **Table Statistics**, **Column Statistics And Histograms**, and **Outlines**. The **Index Analysis** tab is selected, showing a table of indexes.

Index Name	Table Owner	Table Name	Column Name
CLIENT_PK	SYSTEM	CLIENT	CLIENT_ID
CLIENT_TRANSACTION_PK	SYSTEM	CLIENT_TRANSACTION	CLIENT_TRANSACTION_ID
INVESTMENT_PK	SYSTEM	INVESTMENT	INVESTMENT_ID
INVESTMENT_TYPE_PK	SYSTEM	INVESTMENT_TYPE	INVESTMENT_TYPE_ID
CLIENT_TRANSACTION_CLIENT	SYSTEM	CLIENT_TRANSACTION	CLIENT_ID

Additionally, for the Oracle platform there are Table Statistics, Column Statistics and Histograms, and Outlines tabs. For more information, see "[Using Oracle-Specific Features](#)" on page 171.

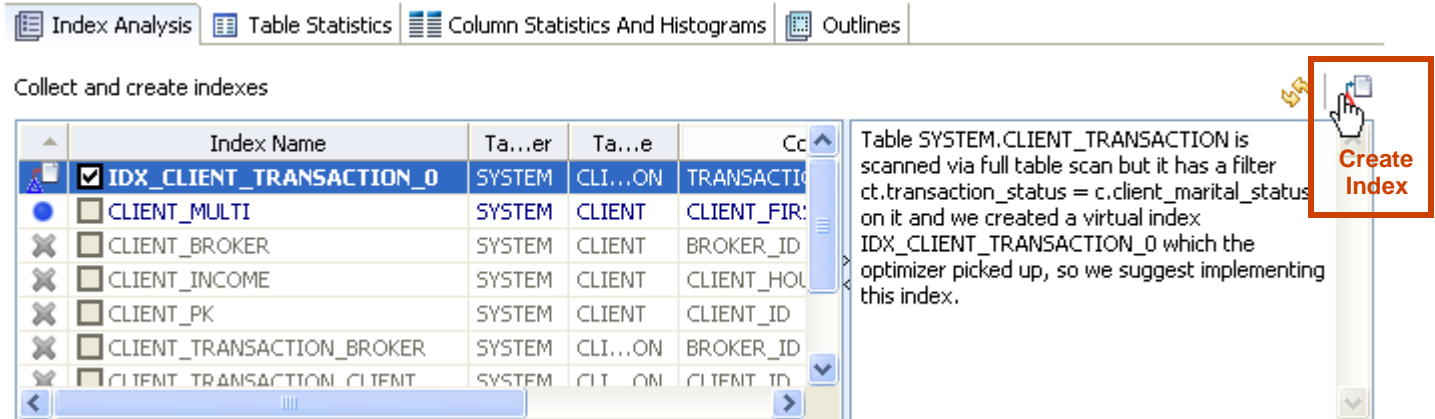
Statement analysis is performed when you click **Perform detail analysis** on the **Overview** tab and then click **Run Job** or when you click the **Analysis** tab. In order to view and analyze statement statistics, select the tab (Index Analysis, Table Statistics, Column Statistics and Histograms, or Outline) and the statements whose statistics you want to analyze.

For more information, see "[Visual SQL Tuning](#)" on page 151.

IMPLEMENTING INDEX ANALYSIS RECOMMENDATIONS

Once you have added tuning candidates to a tuning job, DB Optimizer can analyze the effectiveness of the indexes in the database and recommend the creation of new indexes where the new indexes can increase performance.

In the **Collect and create indexes** table, any indexes DB Optimizer recommends you create are marked in orange.



To accept the suggestion and have tuning automatically generate an index:

- 1 For any recommended index, click the checkbox to the left of the index.

Optionally, modify the Index type by clicking in the **Index Type** column and then selecting a type from the list.

- 2 Click the **Create Indexes** button.

The **Index Analysis** dialog appears.

- 3 To view the index SQL in an editor for later implementation, click the statement and then click **Open in a SQL editor**.
- 4 To run the index SQL and create the index on the selected database, click **Execute**.

VISUAL SQL TUNING

NOTE: Visual SQL Tuning is not available in DB Optimizer XE Developer.

DB Optimizer can now parse an SQL query and analyze the indexes and constraints on the tables in the query and display the query in graphical format on The Visual SQL Tuning (VST) diagram, which can be displayed in either Summary Mode or Detail Mode, helps developers, designers and DBAs see flaws in the schema design such as Cartesian joins, implied Cartesian joins and many-to-many relationships. The VST diagram also helps the user to more quickly understand the components of an SQL query, thus accelerating trouble-shooting and analysis.

This section is comprised of the following topics:

- [Changing Diagram Detail Display](#) on page 152
- [Interpreting the VST Diagram Graphics](#) on page 161

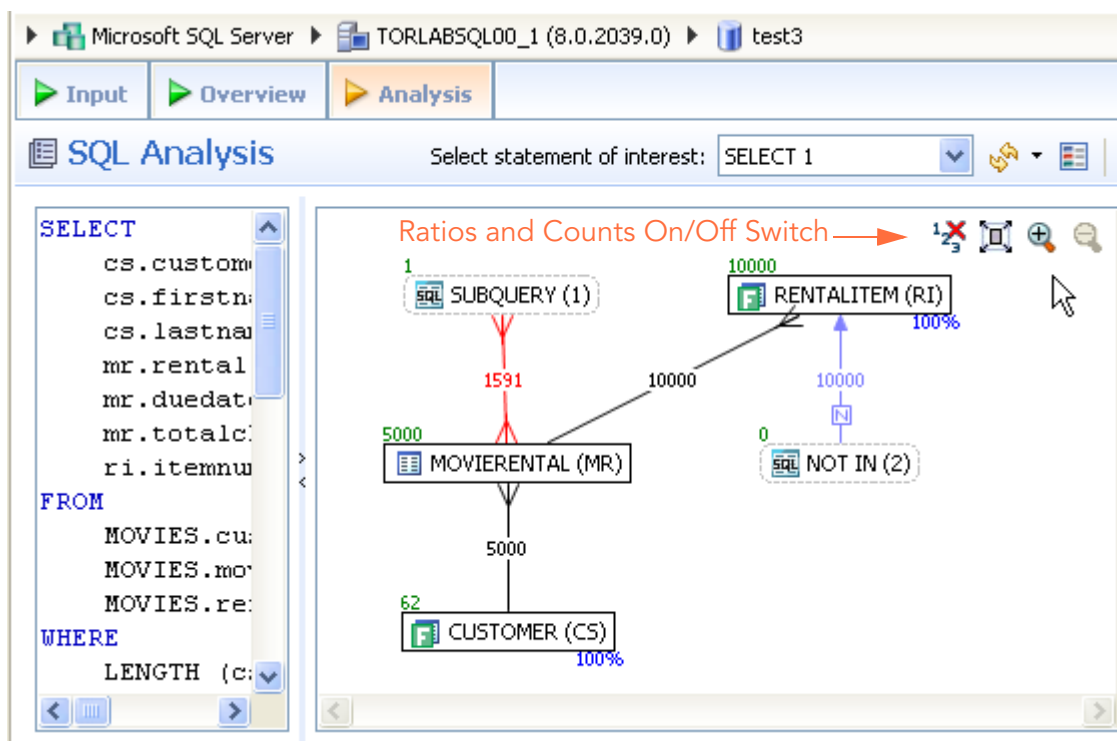
CHANGING DIAGRAM DETAIL DISPLAY

This section is comprised of the following topics:

- [Viewing Table Counts and Ratios](#) on page 152
- [Viewing the VST Diagram in Summary Mode](#) on page 153
- [Viewing the VST Diagram in Detail Mode](#) on page 154
- [Changing Detail Level for a Specific Table](#) on page 154
- [Viewing All Table Fields](#) on page 155
- [Viewing Diagram Object SQL](#) on page 157
- [Expanding Views in the VST Diagram](#) on page 158

VIEWING TABLE COUNTS AND RATIOS

By default the diagram does not show table counts, two table join sizes and filtered result set ratios. Click the Ratios and Counts on/off switch to view or hide this information

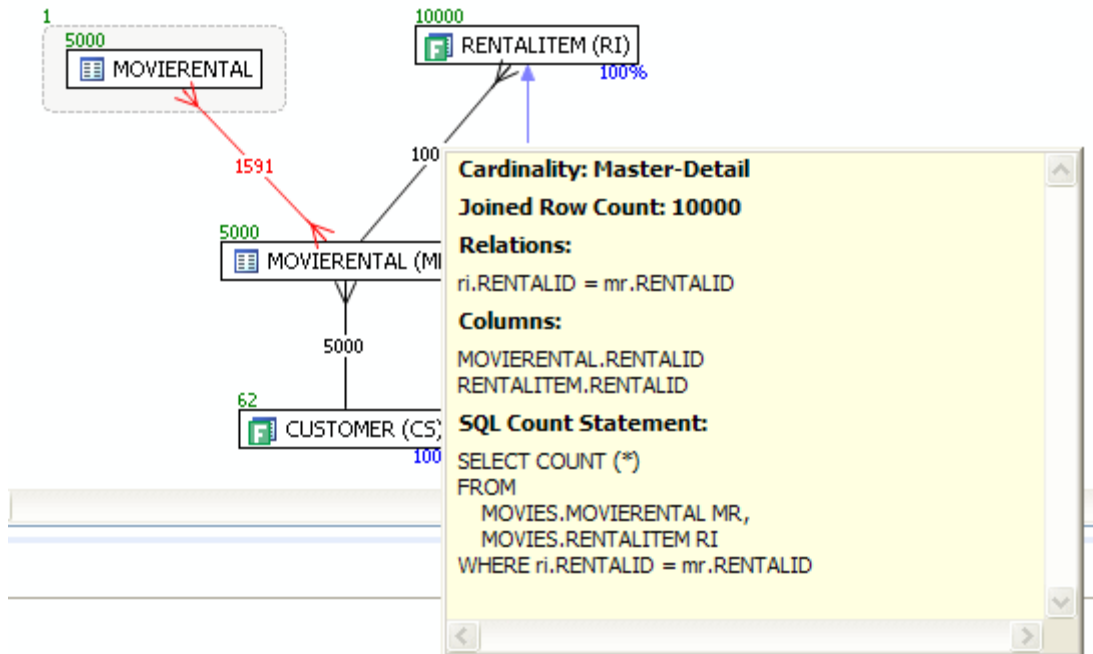


Green numbers at top left of table represent the total number of rows in that table. In the above the MOVIERENTAL (MR) table has 5000 rows.

Blue percentage at the bottom right of the table represent the percentage of rows in that table that meet the selection criteria. In the above example, 100 percent of the rows in the RENTALITEM (RI) table have met the selection criteria.

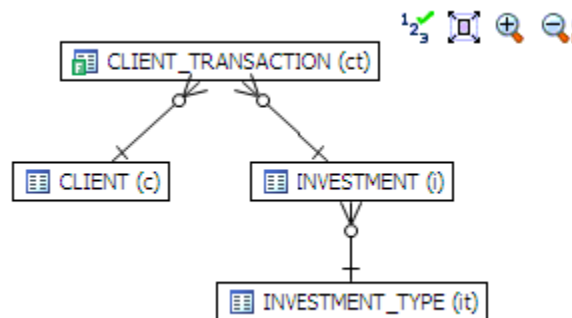
The numbers on the table joins indicate the total number of rows that meet the selection criteria for both tables.

You can also view the SQL Query that created a relationship by hovering over the relationship. If the tooltip contents is larger than the size of the tooltip rectangle, you can hover the mouse on top of the tooltip for a second, then it will turn into a dialog you can re-size, scroll in, and select text from to copy into the Clipboard.



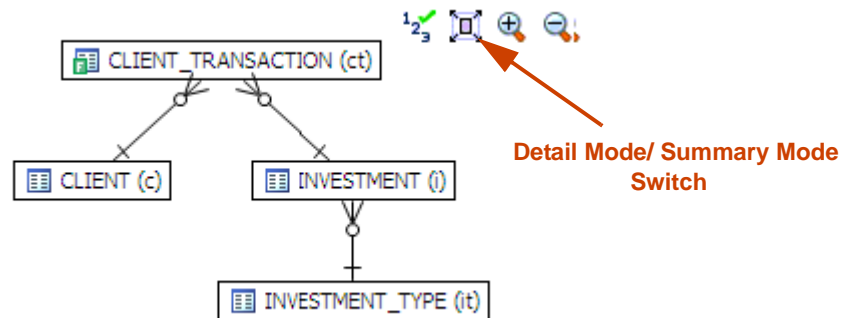
VIEWING THE VST DIAGRAM IN SUMMARY MODE

By default the diagram displays Summary Mode, showing only table names and joins, as seen in the following illustration

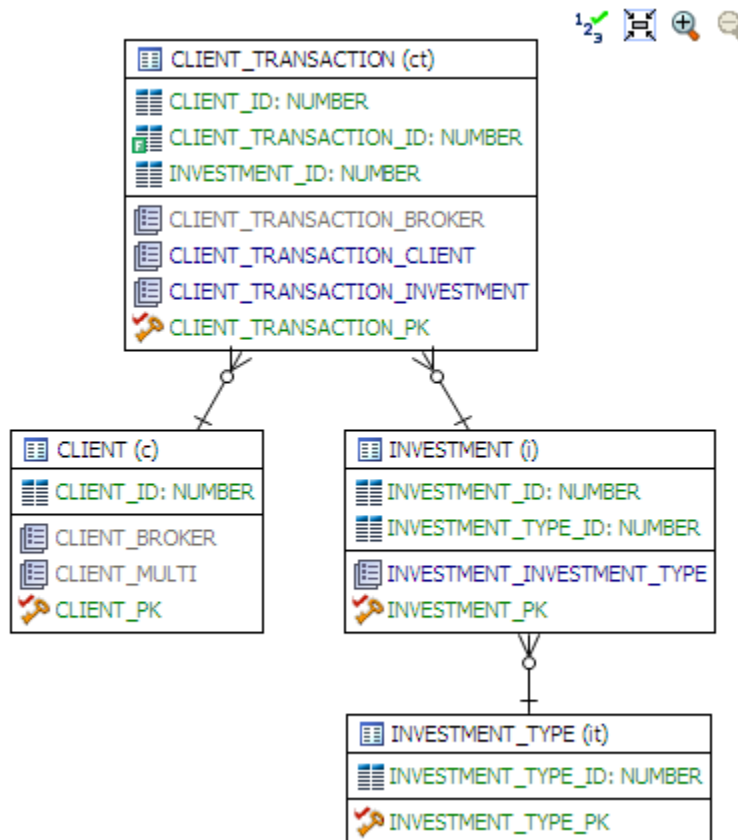


VIEWING THE VST DIAGRAM IN DETAIL MODE

By default, the VST diagram displays in Summary Mode, but by clicking the **Detail Mode/ Summary Mode** switch.



additional details of the tables display, including table columns and indexes

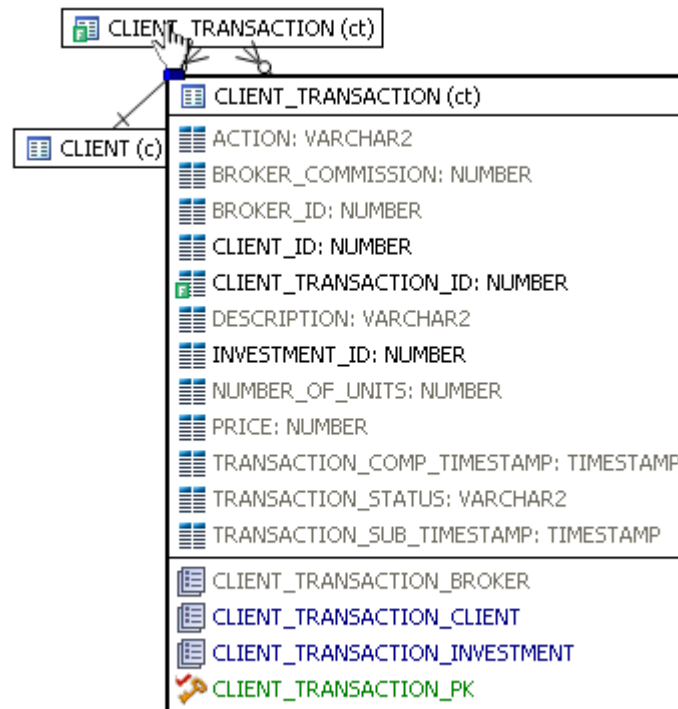


CHANGING DETAIL LEVEL FOR A SPECIFIC TABLE

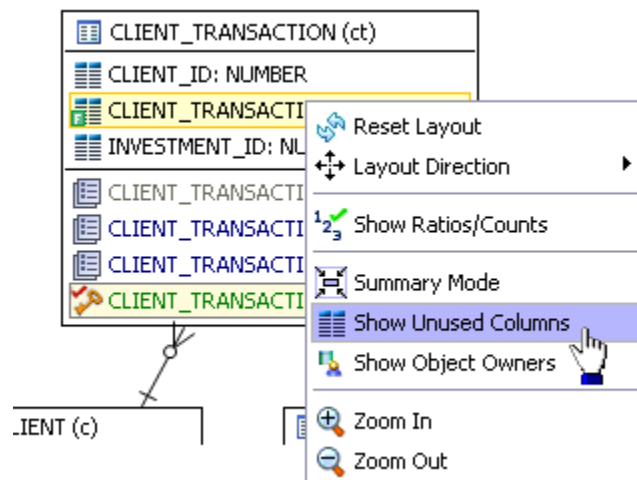
You can also switch between Summary Mode and Detail Mode for a specific table or view, by double-clicking the object name.

VIEWING ALL TABLE FIELDS

















By default, only fields that are used in the WHERE clause are displayed in detail mode; however, all fields in the table can be seen in a pop-up window when, while in Summary Mode, you hover the mouse over the table. The illustration below shows an example of a pop-up window that appears when hovering the mouse over a table.



However, if you right-click the table you can choose to display even unused columns as follows:

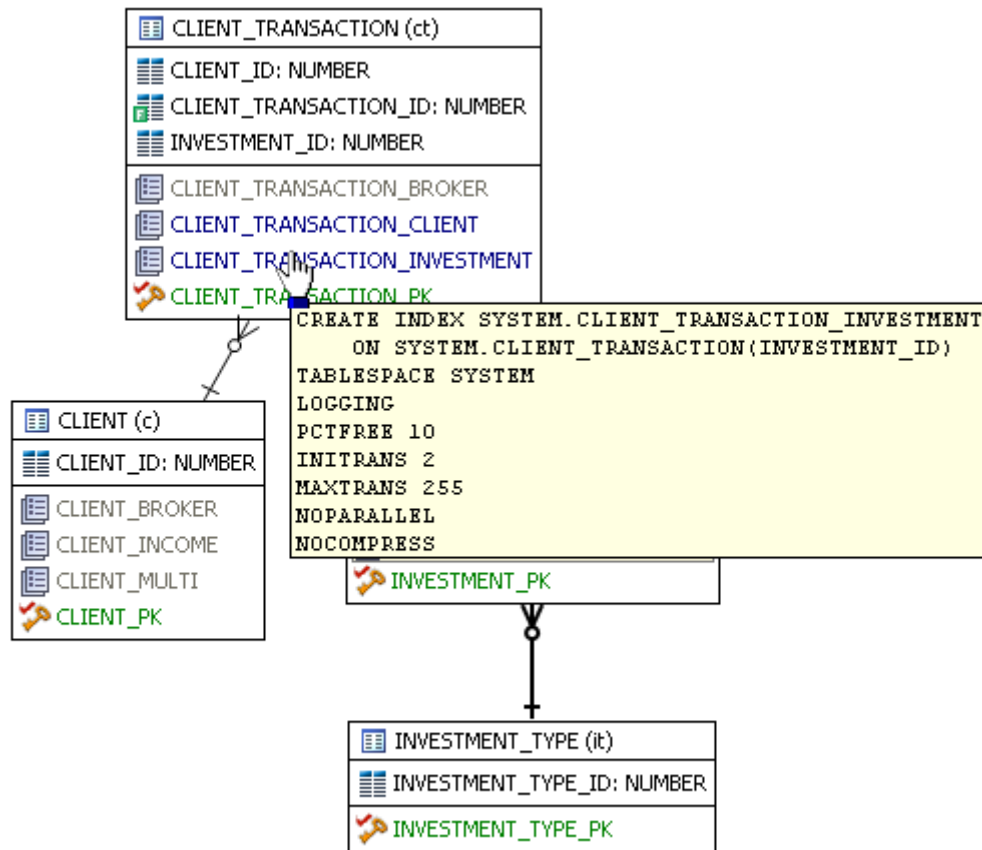


All the columns in the table are shown, and not just the ones used in the WHERE clause of the SQL statement.

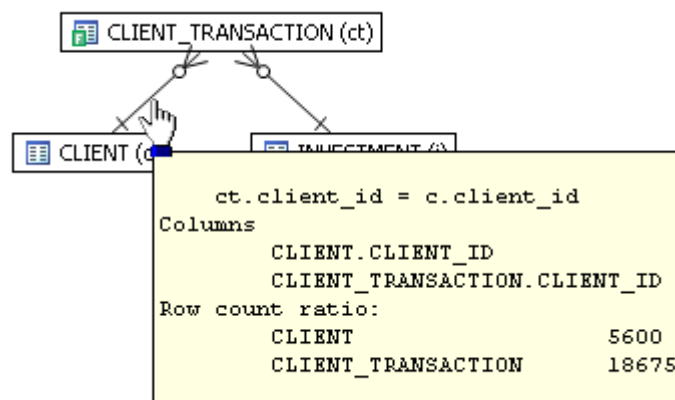
CLIENT_TRANSACTION (ct)	
	ACTION: VARCHAR2
	BROKER_COMMISSION: NUMBER
	BROKER_ID: NUMBER
	CLIENT_ID: NUMBER
	CLIENT_TRANSACTION_ID: NUMBER
	DESCRIPTION: VARCHAR2
	INVESTMENT_ID: NUMBER
	NUMBER_OF_UNITS: NUMBER
	PRICE: NUMBER
	TRANSACTION_COMP_TIMESTAMP: TIMESTAMP
	TRANSACTION_STATUS: VARCHAR2
	TRANSACTION_SUB_TIMESTAMP: TIMESTAMP
<hr/>	
	CLIENT_TRANSACTION_BROKER
	CLIENT_TRANSACTION_CLIENT
	CLIENT_TRANSACTION_INVESTMENT
	CLIENT_TRANSACTION_PK

VIEWING DIAGRAM OBJECT SQL

While in Detail Mode, hovering the mouse over the sub query, table name, field, or index displays the SQL required to create that object.



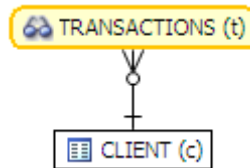
Hovering over the join between two tables displays the relationship between the two tables.



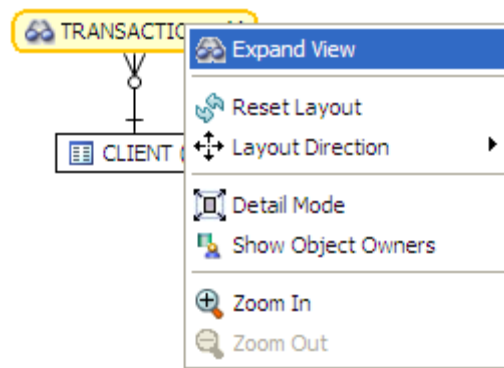
EXPANDING VIEWS IN THE VST DIAGRAM

If there are views in the Visual SQL Tuning diagram, they can be expanded by right clicking the view name and choosing **Expand View**:

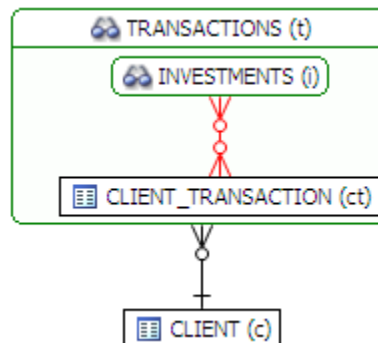
For example, the following is the default layout from query join table CLIENT (c) to view TRANSACTIONS (t):



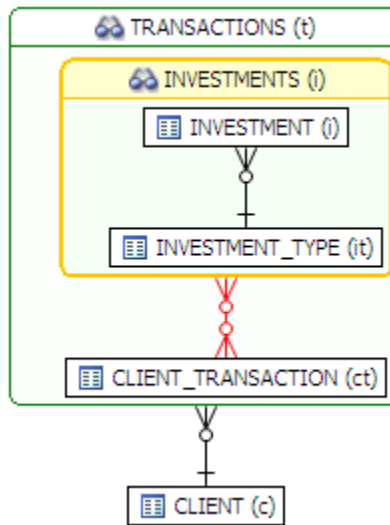
Right click on the view, TRANSACTION (t) and choose **Expand View**



Now we can see the objects in the view:

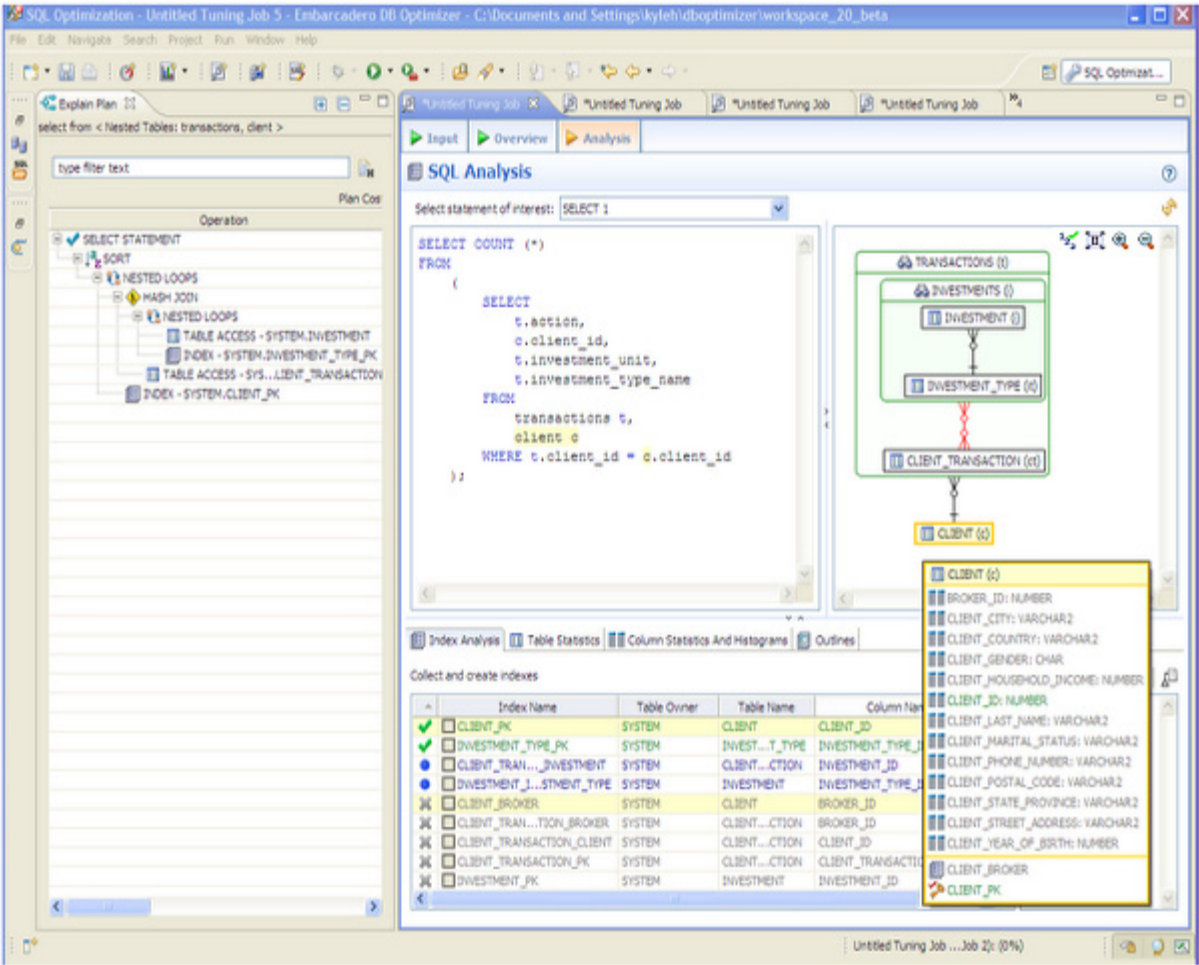


We can further expand the sub-view within the original view:



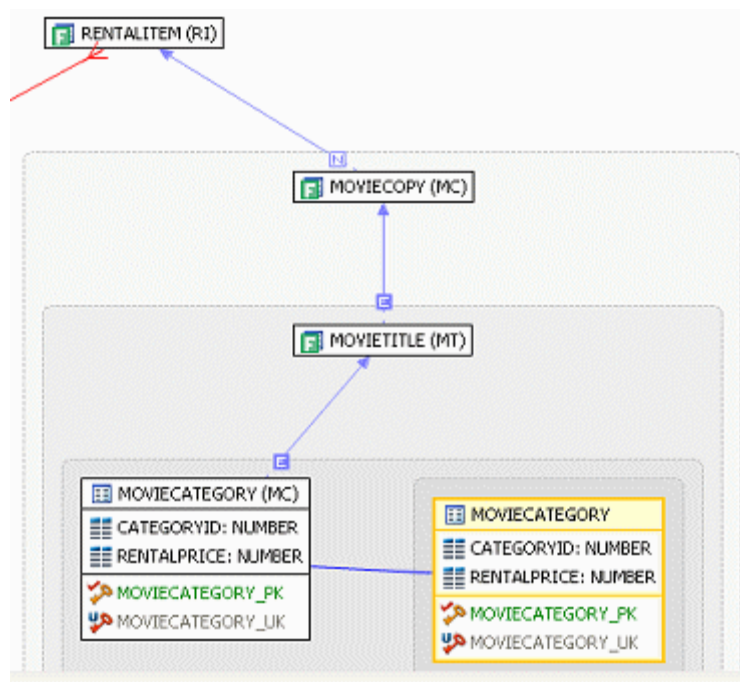
The following is an example of view expansion along with the Explain Plan to the left.

Notice in the view expansion a list of all the indexes on all the underlying tables in the views and sub views and which of those indexes is used in the default execution plan.



EXPANDING SUBQUERIES AND NESTED SUBQUERIES

Double-click queries to expand them or right-click the query and select **Expand Query** from the menu that appears. The following shows several layers of nesting queries.



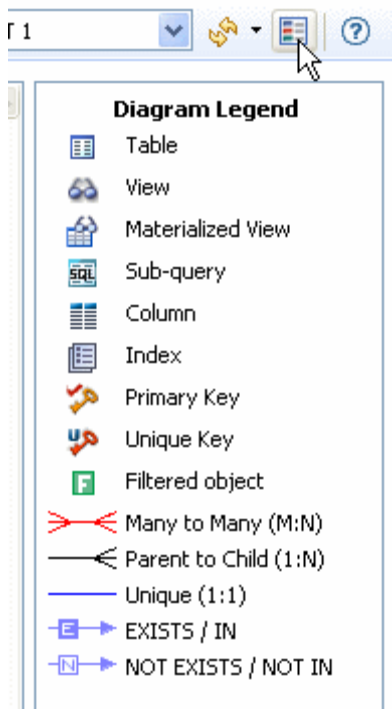
INTERPRETING THE VST DIAGRAM GRAPHICS

This section contains the following topics that will help you understand the graphics in VST diagrams:

- [Viewing the Diagram Legend](#) on page 162
- [Colors](#) on page 162
- [Connecting Lines/Joins](#) on page 162





VIEWING THE DIAGRAM LEGEND

Click the **Diagram Legend Toggle** button as shown in the diagram below to see a description of the icons and relationship lines used in VST diagrams.



COLORS







The color of the index entries in the **Collect and Create Indexes** table is interpreted as follows:

Text Color	Interpretation
	Index is used in the query.
	Index is usable but not used by the current execution path.
	This index is missing. DB Optimizer recommends that you create this index.
	This index exists on the table but not usable in this query as it is written.

CONNECTING LINES/JOINS

Joins are represented with connecting lines between nodes. You can move tables in the diagram by clicking and dragging them to the desired location. The position of the connecting lines is automatically adjusted. The following describes when a particular type of connecting line is used and the default positioning of the line.

Connecting Lines	When used
	One-to-One Join relationships are graphed horizontally using blue lines. For more information, see " One-to-One Join " on page 163.

Connecting Lines	When used
	One-to-Many Join relationships are graphed with the many table above the one table. For more information, see " One-to-Many Join " on page 163.
	Cartesian Join shows the table highlighted in red with no connectors to indicate that it is joined in via a Cartesian join. For more information, see " Cartesian Join " on page 164.
	Many-to-Many Join relationships are connected by a red line and the relative location is not restricted. For more information, see " Many-to-Many Join " on page 166.
	Indirect Relationship. For more information, see " Indirect Relationship " on page 166
	Not Exists and Not in relationship lines connect the subquery to the table being queried. Notice that when you click this relationship line, the SQL text creating the relationship is also selected. For more information, see Not In or Not Exists Join on page 169.
	Exists and In relationship lines connect the subquery to the table being queried. Notice that when you click this relationship line, the SQL text creating the relationship is also selected. For more information, see In or Exists Join on page 167.

ONE-TO-ONE JOIN

If two tables are joined on their primary key, such as:

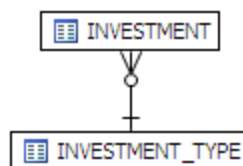
```
SELECT COUNT (*)
FROM
    investment_type it,
    office_location ol
WHERE investment_type_id = office_location_id;
```

Then graphically, these would be laid out side-by-side, with a one-to-one connector:



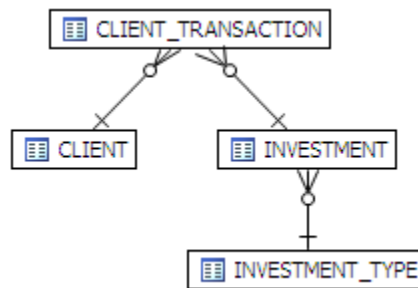
ONE-TO-MANY JOIN

This is the default positioning of a one-to-many relationship, where INVESTMENT_TYPE is the master table and INVESTMENT is the details table.



The following is an example of a query that consists of only many-to-one joins, which is more typical:

```
SELECT
    ct.action,
    c.client_id,
    i.investment_unit,
    it.investment_type_name
FROM
    client_transaction ct,
    client c,
    investment_type it,
    investment i
WHERE
    ct.client_id = c.client_id AND
    ct.investment_id = i.investment_id AND
    i.investment_type_id = it.investment_type_id and
    client_transaction_id=1
```

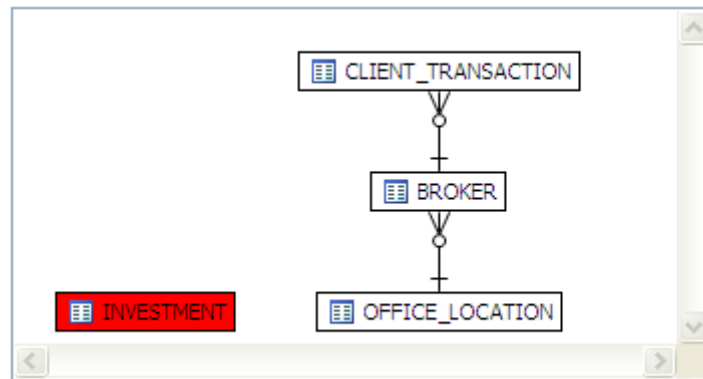


CARTESIAN JOIN

A Cartesian join is described in the following example where the query is missing join criteria on the table INVESTMENT:

```
SELECT
    A.BROKER_ID BROKER_ID,
    A.BROKER_LAST_NAME BROKER_LAST_NAME,
    A.BROKER_FIRST_NAME BROKER_FIRST_NAME,
    A.YEARS_WITH_FIRM YEARS_WITH_FIRM,
    C.OFFICE_NAME OFFICE_NAME,
    SUM (B.BROKER_COMMISSION) TOTAL_COMMISSIONS
FROM
    BROKER A,
    CLIENT_TRANSACTION B,
    OFFICE_LOCATION C,
    INVESTMENT I
WHERE
    A.BROKER_ID = B.BROKER_ID AND
    A.OFFICE_LOCATION_ID = C.OFFICE_LOCATION_ID
GROUP BY
    A.BROKER_ID,
    A.BROKER_LAST_NAME,
    A.BROKER_FIRST_NAME,
    A.YEARS_WITH_FIRM,
    C.OFFICE_NAME;
```

Graphically, this looks like:



INVESTMENT is highlighted in red with no connectors to indicate that it is joined in via a Cartesian join.

Possible missing join conditions are displayed in the **Overview** tab under **Generated Cases** in the transformations area. DB Optimize recommends that you create these joins.

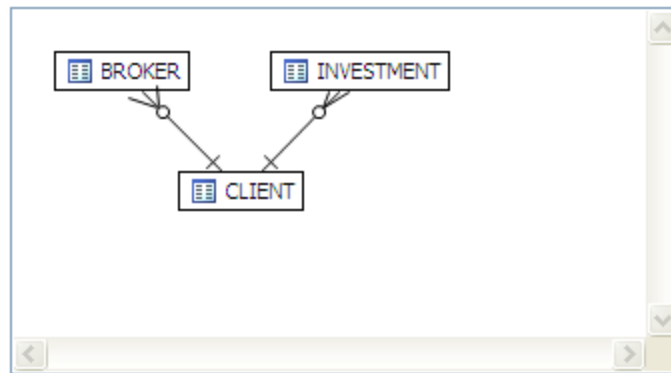
SQL Statements and Cases		Cost	Execution Statistics	Other Execution Statistics		
Name	Text	Value	Elapsed Time (s)	Physical Reads	Logical Reads	CPU
SELECT 1	select from BROKER,	34014.0	6.22	0	170	
[Missing a valid join criteria] transformation		274.0	0.04	0	167	
FULL		34019.0	6.29	0	173	
LEADING1		34017.0	6.25	0	192	
ALL_ROWS		34014.0	6.35	0	170	
LEADING3		34017.0	6.41	0	170	
INDEX		34392.0	6.58	0	414	
LEADING2		38148.0	7.94	0	170	
ORDERED		38147.0	8.61	0	170	
USE_NL		38198.0	9.03	0	37518	

NOTE: Transformations are highlighted in yellow.

IMPLIED CARTESIAN JOIN

If there are different details for a master without other criteria then a Cartesian-type join is created:

```
SELECT *
FROM
  investment i,
  broker b,
  client c
WHERE
  b.manager_id=c.client_id and
  i.investment_type_id=c.client_id;
```



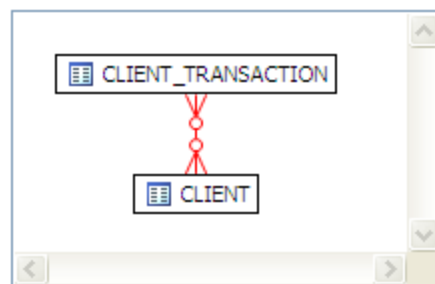
The result set of BROKER to CLIENT will be multiplied by the result set of INVESTMENT to CLIENT.

MANY-TO-MANY JOIN

If there is no unique index at either end of a join then it can be assumed that in some or all cases the join is many-to-many; there are no constraints preventing a many-to-many join. For example, examine the following query:

```
SELECT *
FROM
  client_transaction ct,
  client c
WHERE
  ct.transaction_status=c.client_marital_status;
```

There is no unique index on either of the fields being joined so the optimizer assumes this is a many-to-many join and the relationship is displayed graphically as:



If one of the fields is unique, then the index should be declared as such to help the optimizer.

INDIRECT RELATIONSHIP

Indirect relationships are produced by the following SQL, where BIG_STATEMENT2 is a Materialized View.

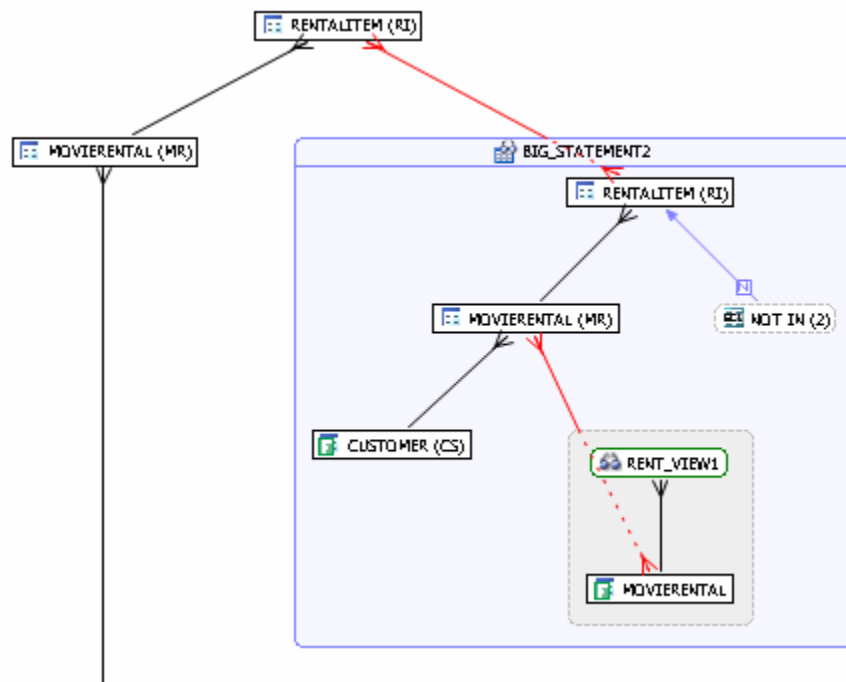
```
SELECT CS.*
FROM
  MOVIES.CUSTOMER CS,
```

```

MOVIES.MOVIERENTAL MR,
MOVIES.RENTALITEM RI,
OE.BIG_STATEMENT2
WHERE
  CS.ZIP > '75062' AND
  MR.RENTALID = RI.RENTALID AND
  RI.ITEMNUMBER = OE.BIG_STATEMENT2.ITEMNUMBER AND
  MR.CUSTOMERID = CS.CUSTOMERID;

```

The following diagram produced by the SQL above shows that an indirect relationship exists between the RENTALITEM(RI) tables inside and outside the materialized view, BIG_STATEMENT2. An indirect relationship also exists between MOVIERENTAL (MR) inside BIG_STATEMENT2 and MOVIERENTAL(MR) inside the RENT_VIEW1 view.



IN OR EXISTS JOIN

The following SQL contains a nest IN subquery (shown in bold text) that is graphically represented with the Subquery summary icon and the IN join.

```

SELECT
  cs.customerid,
  cs.firstname,
  cs.lastname,
  mr.rentalid,
  mr.duedate,
  mr.totalcharge,
  ri.itemnumber

```

```

FROM
  (
    SELECT
      c1.customerid,
      c1.firstname,
      c1.lastname,
      c1.phone
    FROM MOVIES.customer c1
    WHERE EXISTS (SELECT NULL
                  FROM MOVIES.customer c2
                  WHERE
                    c1.customerid <> c2.customerid AND
                    c1.lastname = c2.lastname AND
                    c1.phone BETWEEN 0 AND 9999569900)
  )
cs,
  (
    SELECT
      customerid,
      rentalid,
      duedate,
      totalcharge,
      rentaldate
    FROM MOVIES.movierental
    WHERE totalcharge > 10
  )
mr,
MOVIES.rentalitem ri
WHERE
  LENGTH (cs.lastname) = 10 AND
  - 1 < cs.customerid AND
  ROUND (ri.rentalid) > 10 AND
  TRUNC (ri.itemnumber) > 1 AND
  mr.totalcharge > (SELECT AVG (totalcharge)
                   FROM MOVIES.movierental
                   WHERE TOTALCHARGE >= 40) AND
  ri.moviecopyid NOT IN (SELECT mc.moviecopyid
                        FROM MOVIES.moviecopy mc
                        WHERE
                          mc.copyformat = 'vhs' AND
                          mc.copycondition = 'new' AND
                          mc.movieid IN (SELECT mt.movieid
                                       FROM MOVIES.movietitle mt
                                       WHERE
                                         mt.year < 1990 AND
                                         mt.rating IN ('pg', 'r') AND
                                         mt.categoryid IN (SELECT

```

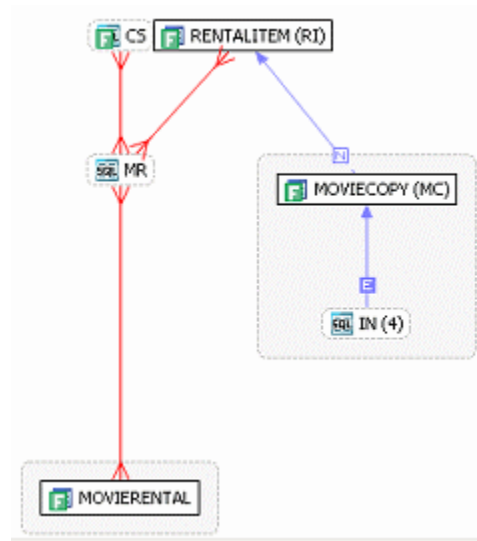
mc.categoryid


```

FROM
MOVIES.moviecategory mc
WHERE
mc.rentalprice = (SELECT MAX (rentalprice)
FROM MOVIES.moviecategory
WHERE categoryid = mc.categoryid))) AND
mr.CUSTOMERID = cs.CUSTOMERID AND
ri.RENTALID = mr.RENTALID

```

Graphically, this would display as the following when the MOVIECOPY (MC) subquery is expanded:



NOT IN OR NOT EXISTS JOIN

The following SQL contains a NOT IN subquery (shown in bold below) that is graphically represented with the Subquery summary icon and the NOT IN join.

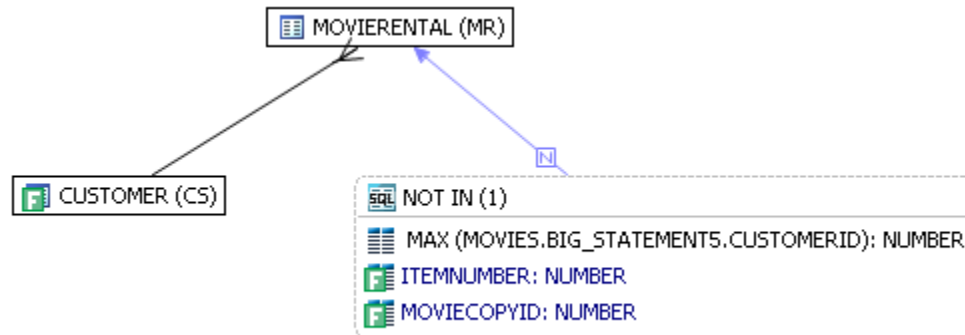
```

SELECT CS.*
FROM
  MOVIES.CUSTOMER CS,
  MOVIES.MOVIERENTAL MR
WHERE
  CS.ZIP > '75062' AND
  MR.RENTALID NOT IN (SELECT MAX (MOVIES.BIG_STATEMENT5.CUSTOMERID)
FROM
  MOVIES.RENTALITEM RI,
  MOVIES.MOVIECOPY MC,

```

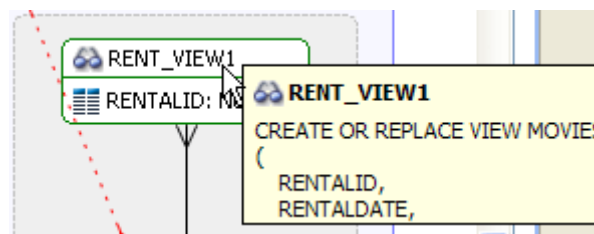
```
MOVIES.BIG_STATEMENT5
WHERE
  RI.ITEMNUMBER > 1 AND
  MC.MOVIECOPYID = 700) AND
MR.CUSTOMERID = CS.CUSTOMERID;
```

Graphically, this statement would look like this:



VIEWING OBJECT SQL

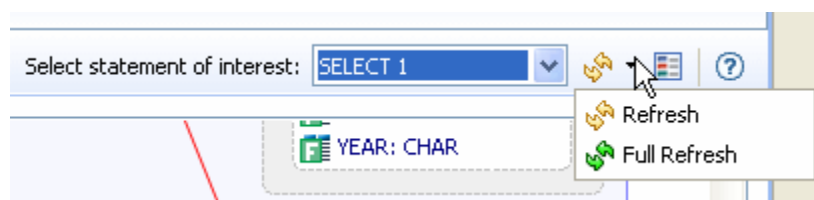
Hover over the name of an object to view the object SQL as shown in the diagram below.



REFRESHING THE VST DIAGRAM

I'm not sure if this is correct. Please check thoroughly.

There are two refresh options available: **Refresh** and **Refresh All**. Click the **Refresh** list as shown below to gain access to these options



- **Refresh:** Regenerates the Analysis tab including the VST diagram. Any changes made on the tab are reflected in the diagram.

- **Full Refresh:** Re-caches all objects used in (or related to) the query, then regenerates the Analysis tab including the VST diagram.

USING ORACLE-SPECIFIC FEATURES

This section describes the tuning features available for the Oracle platform. These features are not available for other database platforms.

- [Using the Table Statistics Tab](#) on page 172
- [Using the Column Statistics And Histograms Tab](#) on page 173
- [Using the Outlines Tab](#) on page 174
- [Tuning SQL Statements in the System Global Area \(SGA\)](#) on page 175

USING THE TABLE STATISTICS TAB

The Table Statistics area of the Analysis tab indicates when and if table statistics were last taken. Using the Table Statistics you can view the information the optimizer uses to choose a path and assess the validity of the various hints presented on the Overview tab.

The screenshot shows the SQL Analysis tool interface. The top section displays a SQL query and its execution plan. The bottom section shows a table of statistics for various tables.

SQL Query:

```
SELECT
  A.COMPANY,
  A.PAYGROUP,
  E.OFF_CYCLE,
  E.SEPCHK_FLAG,
  E.TAX_METHOD,
  E.TAX_PERIODS,
  C.RETROPAY_ERNCN,
  SUM (C.AMOUNT_DIFF) SUM_AMOUNT
FROM
  PS_PAY_CALEDAR A,
  WB_JOB B,
  WB_RETROPAY_EARNS C,
  PS_RETROPAY_RQST D,
  PS_RETROPAYPGM_TBL E
WHERE
```


Execution Plan:



```
graph TD
  PS_RETROPAYPGM_TBL[E] --> PS_RETROPAY_RQST[D]
  PS_RETROPAYPGM_TBL[E] --> WB_JOB[B]
  PS_RETROPAYPGM_TBL[E] --> WB_RETROPAY_EARNS[C]
  PS_RETROPAYPGM_TBL[E] --> PS_PAY_CALEDAR[A]
  PS_RETROPAYPGM_TBL[E] --> WB_JOB[G]
```

Table Statistics:

Object	Statistics	Monitoring	Attributes		
Table Owner	Table Name	Statistics Status	Days Since Stats Taken	Monitoring	Cache
<input type="checkbox"/> SYSTEM	PS_RETROPAYPGM_TBL	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	PS_PAY_CALEDAR	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	WB_JOB	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	WB_RETROPAY_EARNS	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	PS_RETROPAY_RQST	Statistics OK	200	YES	N

This table draws attention to:

- **Missing statistics:** Missing statistics can cause the optimizer to choose the wrong path because the optimizer uses table statistics to make decisions. If the statistics are missing, you can click the select a table and then click **Collect Statistics**  on the far right of the tab. This sends a request to the database to analyze the table and calculate the statistics.

- **Out-of-date statistics:** Like missing statistics, out-of-date statistics can also cause the optimizer to choose the wrong path. You can update the statistics by selecting a table, and then clicking Display Statistics , which refreshes the statistics from the database or by clicking Collect Statistics , which requests the database to analyze the table and calculate the statistics.

NOTE: Collecting Statistics may be time-consuming, depending on how many tables the database is analyzing and the number of rows in each table.

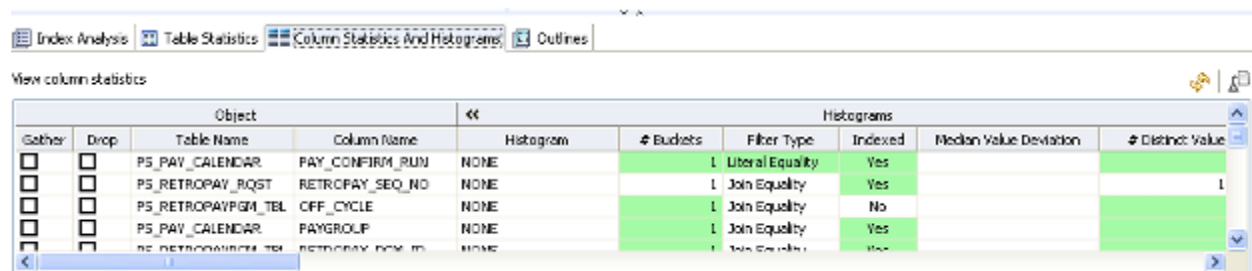
- **Useful statistics:** The number of rows in a table and whether the table has been modified since the statistics were last collected can help you to determine which hints you should implement in the SQL code. These statistics can help the DB Administrator to better understand the database.

TIP: You can right-click anywhere in a row and choose options such as Collect Statistics, Display Statistics, and Copy from the short-cut menu.

USING THE COLUMN STATISTICS AND HISTOGRAMS TAB

Histograms are special statistics that exist for a limited number of columns and are created by the database administrator. Column histograms should be created only when there are highly-skewed values in a column, such as is the case of an order details table with an Order Status column where the number of closed orders for a business operating for several years is far greater than the number of open orders. The Order Status column therefore meets the criteria of a useful target for a histogram because the data is highly skewed. Using histograms the optimizer determines that a full-scan is recommended when searching for closed orders, but an index scan is more useful when searching for open orders.

DBOptimizer looks at the columns that have histograms and using statistics tries to determine whether the column is a good or bad candidate for a histogram and presents this information on the Column Statistics And Histograms tab.



Object		Table Name	Column Name	Histogram	# Buckets	Filter Type	Indexed	Median Value Deviation	# Distinct Value
<input type="checkbox"/>	<input type="checkbox"/>	PS_PAY_CALENDAR	PAY_CONFIRM_RUN	NONE	1	Literal Equality	Yes		
<input type="checkbox"/>	<input type="checkbox"/>	PS_RETROPAY_RQST	RETROPAY_SEQ_NO	NONE	1	Join Equality	Yes		1
<input type="checkbox"/>	<input type="checkbox"/>	PS_RETROPAYPGM_TEL	OFF_CYCLE	NONE	1	Join Equality	No		
<input type="checkbox"/>	<input type="checkbox"/>	PS_PAY_CALENDAR	PAYGROUP	NONE	1	Join Equality	Yes		
<input type="checkbox"/>	<input type="checkbox"/>	PS_RETROPAYPGM_TEL	RETROPAY_SEQ_NO	NONE	1	Join Equality	Yes		

The row shading indicates the following:

- **Green:** Good histogram candidate
- **Red:** Bad histogram candidate
- **No shading:** Not determined to be a good or bad histogram candidate

Median Value Deviation

For columns that have histograms, the median value deviation is presented. Understanding the median value deviation can help you determine whether an index scan or a full-table scan would be more efficient.

The median value deviation represents the number of values that have duplicates away from the median. In the case of the Order Status column, there are only three possible values, open, processing, and closed. Consider the following:

10 open orders

100,000 closed orders

1 order in processing

In this case the median is the middle value, 10. The number of closed orders is 10,000 times the median which indicates that the column data is highly skewed. In this case the value in the Median Value Deviation column would be presented as

1, 0, 0, 0, 1, 0, 0, 0

There are 1's at the first and 5th spot in the median value deviation field indicating one column value (value of orders in the *processing* state which appears once) is 1 factor of 10 away from the median and there is a 1 at the 5th position indicating there is a column value (orders in the *closed* state) that appears 5 factors of 10 more often (10,000) than the median value of 10.

A column with a median value deviation of 0, 0, 0, 0, 0, 0, 0, 0 indicates that the column data is not skewed and it is a bad candidate for a histogram, and therefore a full scan of the table would more efficiently satisfy a query than an index scan.

To update the statistics of any object, you can select **Gather** for that column and then click **Display Statistics** or **Collect Statistics**.

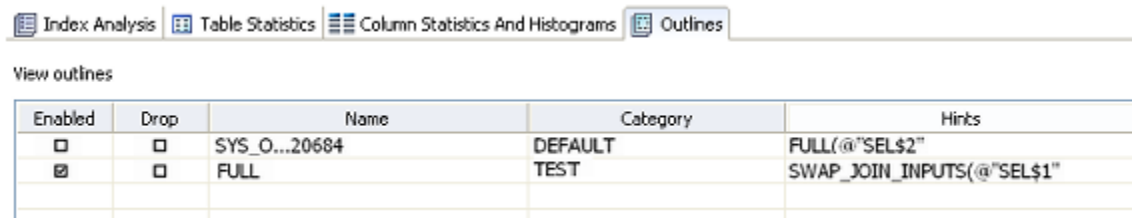
To stop gathering statistics for an object, such as a bad candidate for a histogram, select **Drop** for that column and then click **Display Statistics** or **Collect Statistics**.

TIP: If you are gathering statistics for a column for which the statistics were missing or out-of-date, then once the statistics collection is complete, you should return to the Overview tab and rerun the cases, because the characteristics of the column may have changed, so the hints to improve performance would also change.

USING THE OUTLINES TAB

The Outlines tab provides detailed information about outlines created by the query during the statement execution process on the **Overview** tab.

It provides information including the SQL statement name, if the outline is enabled or not, and the Name, Category, and Hints associated with the outline. Additionally, the Drop parameter specifies if it is dropped or not at execution time.



Enabled	Drop	Name	Category	Hints
<input type="checkbox"/>	<input type="checkbox"/>	SYS_O...20684	DEFAULT	FULL(@"SEL\$2"
<input checked="" type="checkbox"/>	<input type="checkbox"/>	FULL	TEST	SWAP_JOIN_INPUTS(@"SEL\$1"

In order to view outlines, the session needs to have `USE_STORED_OUTLINES=true` set prior to execution. Outlines in tuning are created for the `DEFAULT` category, by default. Use the following commands to enable outlines with the default settings:

```
alter system set USE_STORED_OUTLINES=true;
alter system set USE_STORED_OUTLINES='DEFAULT';
alter session set USE_STORED_OUTLINES=true;
```

Additionally, in order for a session to `USE_STORED_OUTLINES`, the user requires the `create any outline` role. Use the following command to set up the proper permissions:

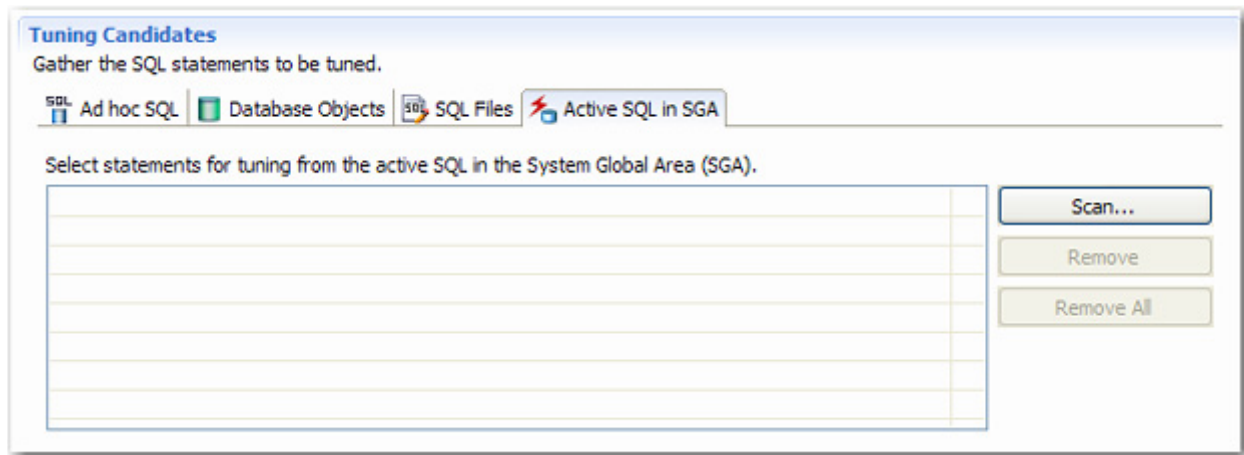
```
grant create any outline to [user];
```

TUNING SQL STATEMENTS IN THE SYSTEM GLOBAL AREA (SGA)

On Oracle platforms, SQL statements that reside in the SGA can also be tuned. When you create a tuning job and specify an Oracle source, an additional tab appears in the Tuning Candidates section of tuning, named `Active SQL in SGA`.

The SGA contains all the SQL since the database has been started up, except for those that have been purged when the system runs out of memory. When analyzing the causes of a database bottleneck, it is perhaps more useful to view and tune the SQL statements most recently run, than those that have run in the last month, for example. `DBOptimizer` cannot tell you which statements have most recently run by looking in the SGA. However, by profiling the

database using DBOptimizer Profiling and then optimizing the code by executing and running the generated cases, you will be able to see which paths are most likely causing a bottleneck and can be altered to enhance performance. Also, you can use Embarcadero Performance Center to continually monitor a database over a longer period of time to help you analyze and optimize database performance.



To add a statement active in the SGA:

- 1 Select the Active SQL in SGA tab and then click **Scan**. The **Scan SGA** wizard appears.
- 2 Set the filtering criteria for an SGA scan and then run the wizard. It returns all active statements on the Oracle source.
- 3 Choose the specific statements and add them to the tuning job.

ADDITIONAL TUNING COMMANDS

In addition to tuning, the interface provides additional commands and functionality that enables you to view source code, statements, and other information regarding the data source.

- [View the Source Code of Tuning Candidates](#) on page 176
- [View Statement or Case Code in SQL Viewer](#) on page 177
- [Open an Explain Plan for a Statement or Case](#) on page 178
- [Executing a Session from the Command Line](#) on page 179

VIEW THE SOURCE CODE OF TUNING CANDIDATES

You can view the source code of a tuning candidate as follows:

- On the **Ad hoc SQL** tab of the **Input** tab, you can see the SQL statements you typed or pasted into that tab.
- On the **Database objects, SQL Files, and Active SQL in SGA** tabs of the **Input** tab, you can double-click the name of any object added to that tab and an SQL session will open that displays the SQL of that database Object. The SQL editor in use is actually Rapid SQL, an Embarcadero product that is integrated with DB Optimizer.

VIEW STATEMENT OR CASE CODE IN SQL VIEWER

The Tuning job's **Overview** tab let you open a statement in an SQL Viewer if you want to perform either of the following tasks:

- View the entire SQL statement.
- Set bind variables. If the Tuning Status Indicator indicates a statement or case has invalid bind variables, you must set those variables before executing the statement or case.

To view or set bind variables in a statement or case:

- 1 Click in the **Text** field of a statement or case.

an SQL Viewer opens on the statement or case. A set of controls for working with the statement or case bind variables appears at the bottom of the window.
- 2 Use the **Data Type** and **Value** (or **NULL**) controls to specify the type and value for each bind variable.

- 3 Close the window by clicking the collapse control in the **Text** field of the statement record, above the SQL Viewer.

InputOverviewAnalysis

Overview1 error detected

Tuning Source Stat:

☒Generate Cases☐Perform detail analysis☐Execute each generated case

Statement

	Name	Schema	Text	Tables	Views	Time		Cases	An
						Elapsed (s)	Improved (s)		Inde
SQL	SELECT 1	SYSTEM	select from hr.employees					0	

SELECT *
FROM hr.employees
WHERE employee_id = :1

Only one SQL DML statement allowed

Name	NULL	Data Type	Value
:1	<input type="checkbox"/>	number	

Generated Cases

SQL Statements and Cases		>> Cost	>>Executi...istics	>> Ot
Name	Text	Value	Elapsed Time (s)	Physical Reads
SQL SELECT 1	select from hr.employees			

After setting bind variables, you can execute a case.

NOTE: Setting the bind variables in a parent statement sets the bind variables in all generated cases for that statement.

OPEN AN EXPLAIN PLAN FOR A STATEMENT OR CASE

Any valid SQL statement added to the **Overview** tab shows a calculated explain plan cost in the **Cost** field of the statement or case record. You can open an explain plan on these statements to view the sequence of operations used to execute the statement and the costs and other explain plan details for each operation.

178

DB OPTIMIZER™ XE AND DB OPTIMIZER™ USER GUIDE

To initially open an explain plan on a valid SQL statement on the **Overview** tab:

- 1 Right-click in the **Name** field of any statement record showing a value in the **Cost** field.
- 2 Select **Explain Plan** from the context menu.

An Explain Plan tab opens below the **Overview** tab.

Operation	Plan Cost	Cost	Operation Cost	Result	Cardinality	Bytes	CPU Cost	IO Cost	Optimizer	Actual Statistics
SELECT STATEMENT	253.0	0.0			37	2183	102...107	235	ALL...WS	
PX COORDINATOR										
PX SEND - SYS.:TQ10004	253.0	0.0			37	2183	102...107	235		
HASH	253.0	0.0			37	2183	102...107	235		
PX RECEIVE	253.0	0.0			37	2183	102...107	235		
PX SEND - SYS.:TQ10003	253.0	0.0			37	2183	102...107	235		
HASH	253.0	1.0			37	2183	102...107	235		
HASH JOIN	252.0	1.0			7981	470879	95195351	235		
PX RECEIVE	67.0	0.0			37	703	24604129	63		
PX SEND - SYS.:TQ10001	67.0	0.0			37	703	24604129	63		
PX BLOCK	67.0	0.0			37	703	24604129	63		
TABLE A...IENT11	67.0	67.0			37	703	24604129	63	ANA...ED	
HASH JOIN	184.0	1.0			11706	468240	67387172	173		

Explain plan operations are shown in a typical tree structure showing parent-child relationships. The following table describes the column groups shown for each operation on the **Explain Plan** tab:

Column (group)	Description
Plan Cost	Includes the Name of the operation and the calculated explain plan cost.
Additional Information	The default, collapsed view shows the Cardinality , Bytes , CPU Cost , IO Cost , and Optimizer values. Expanded, the view also displays Access Predicates , Filter Predicates , QB Lock Name , Distribution , Object Alias , Object Instance , Object Node , Partition ID , Partition Start , Partition Stop , Position , Projection , Remarks , Search Columns , Temp Space , Time , Other , and Other Tag values.

With the **Explain Plan** tab open, you can quickly switch the view to an explain plan for another SQL statement.

To change the Explain Plan tab display to another SQL statement:

- 1 Click in the **Name** field of another statement record showing a value in the **Cost** field.

EXECUTING A SESSION FROM THE COMMAND LINE

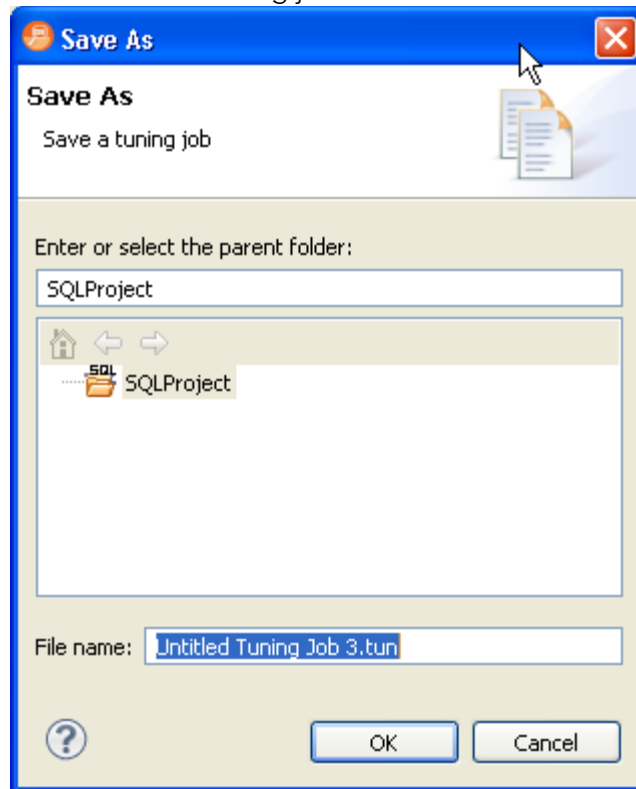
You can launch a tuning job from the command line using the following syntax:

```
dboptimizer.exe tune ds:ROM*L*ABORCL10G_1 sqlfile: C:\dboptimizer\workspace\test.sql
```

In the above command, the user has specified ROM*L*ABORCL10G_1 as the data source, and indicates a tuning session using the test.sql script.

SAVING A TUNING JOB

A tuning session can be saved to a file with a `.tun` suffix. This enables you to open the file at a later time for analysis and to share the tuning job results with other users.



Profiling sessions can be saved as `.oar` files for use at a later time.

Once you have saved a tuning session to disk as a `.tun` file, it appears in the SQL Project Explorer under the name you saved it as. It can be opened again by double-clicking the project name.

To save a tuning session:

Select the tuning session and then choose **File > Save As....** Specify the project location you want to save the file in and modify the file name, as needed. Click **OK**. The tuning job project is added to SQL Project Explorer.

CONFIGURING TUNING

This section contains information on configuring tuning. It provides information on setting up your data sources to work with tuning functionality, as well as information regarding preferences within the application for the customization of various features and functionality.

This section is comprised of the following topics:

- [Set Roles and Permissions on Data Sources](#) on page 181
- [Set Tuning Job Editor Preferences](#) on page 182

- [Set Tuning Job Editor Preferences](#) on page 182
- [Set Generated Case Preferences](#) on page 184

SET ROLES AND PERMISSIONS ON DATA SOURCES

In order to take advantage of all tuning features, each user must have a specific set of permissions. The code below creates a role with all required permissions. To create the required role, execute the SQL against the target data source, modified according to the specific needs of your site:

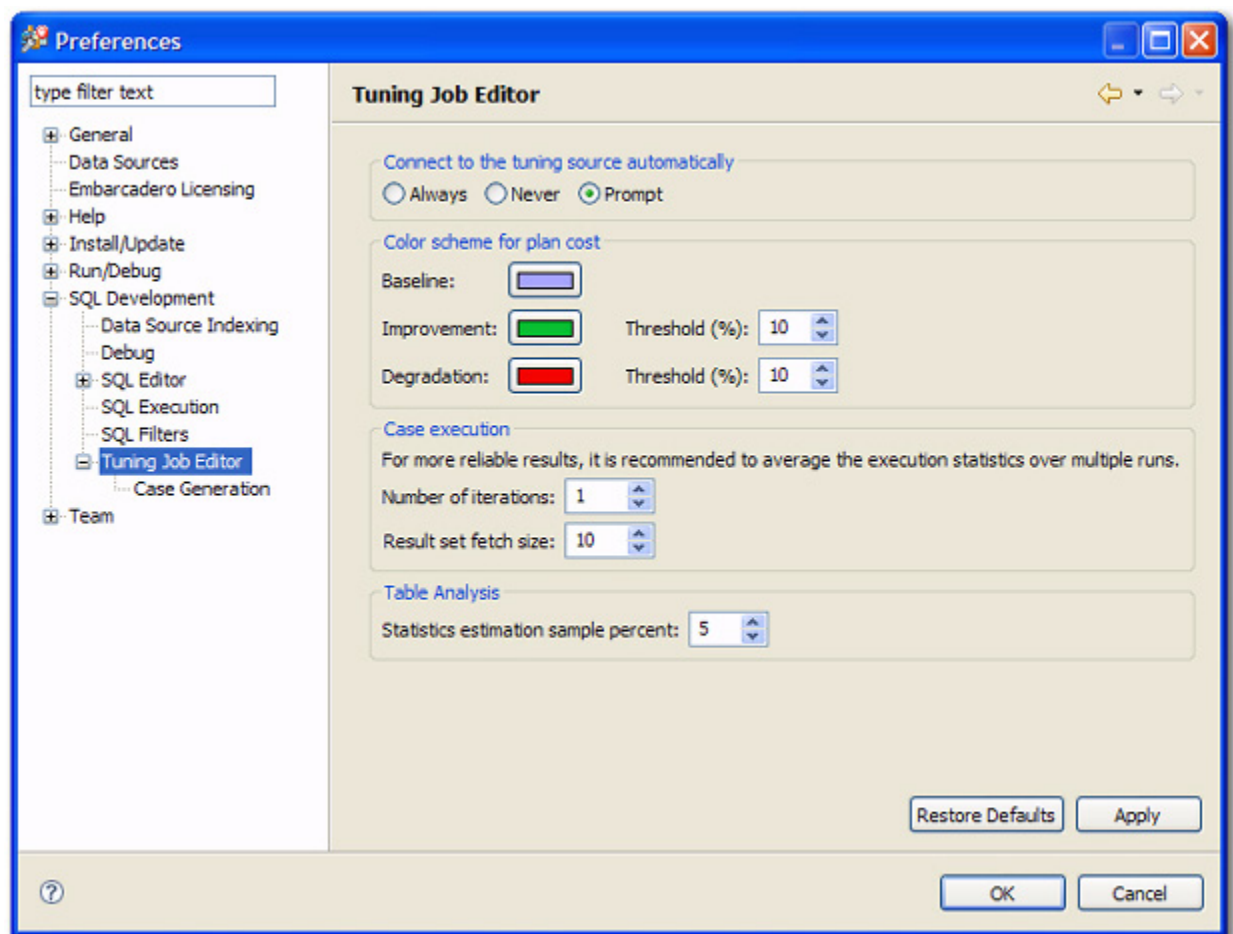
```
/* Create the role */
CREATE ROLE SQLTUNING NOT IDENTIFIED
/
GRANT SQLTUNING TO "CONNECT"
/
GRANT SQLTUNING TO SELECT_CATALOG_ROLE
/
GRANT ANALYZE ANY TO SQLTUNING
/
GRANT CREATE ANY OUTLINE TO SQLTUNING
/
GRANT CREATE ANY PROCEDURE TO SQLTUNING
/
GRANT CREATE ANY TABLE TO SQLTUNING
/
GRANT CREATE ANY TRIGGER TO SQLTUNING
/
GRANT CREATE ANY VIEW TO SQLTUNING
/
GRANT CREATE PROCEDURE TO SQLTUNING
/
GRANT CREATE SESSION TO SQLTUNING
/
GRANT CREATE TRIGGER TO SQLTUNING
/
GRANT CREATE VIEW TO SQLTUNING
/
GRANT DROP ANY OUTLINE TO SQLTUNING
/
GRANT DROP ANY PROCEDURE TO SQLTUNING
/
GRANT DROP ANY TRIGGER TO SQLTUNING
/
GRANT DROP ANY VIEW TO SQLTUNING
/
GRANT SELECT ON SYS.V_$SESSION TO SQLTUNING
/
GRANT SELECT ON SYS.V_$SESSTAT TO SQLTUNING
/
GRANT SELECT ON SYS.V_$SQL TO SQLTUNING
/
GRANT SELECT ON SYS.V_$STATNAME TO SQLTUNING
/
```

Once complete, you can assign the role to users who will be running tuning jobs:

```
/* Create a sample user*/
CREATE USER TUNINGUSER IDENTIFIED BY VALUES '05FFD26E95CF4A4B'
  DEFAULT TABLESPACE USERS
  TEMPORARY TABLESPACE TEMP
  QUOTA UNLIMITED ON USERS
  PROFILE DEFAULT
  ACCOUNT UNLOCK
/
GRANT SQLTUNING TO TUNINGUSER
/
ALTER USER TUNINGUSER DEFAULT ROLE SQLTUNING
/
```

SET TUNING JOB EDITOR PREFERENCES

Tuning job editor preferences let you control certain aspects of the appearance of items in the tuning job editor as well as default behaviors.

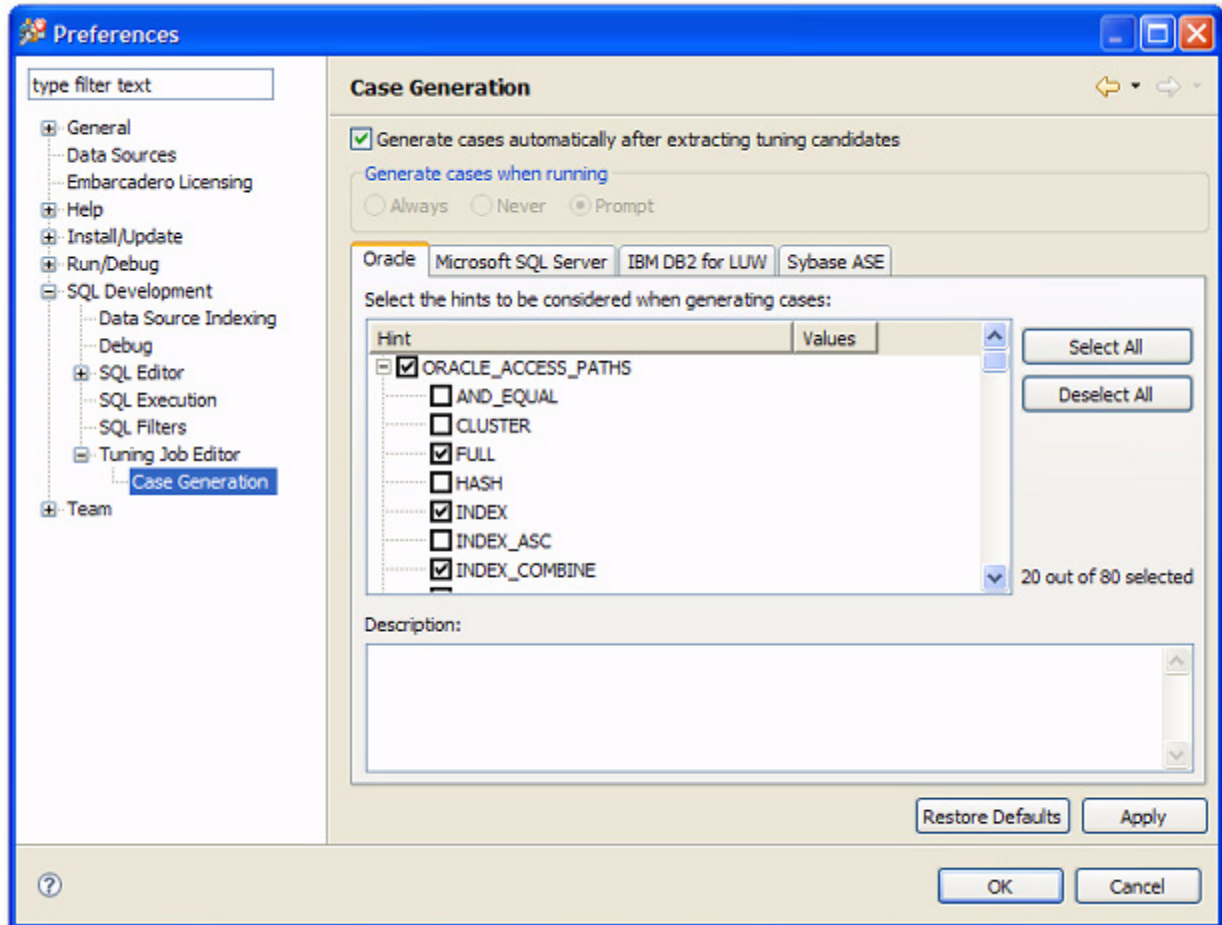


Select Window > Preferences > SQL Development > Tuning Job Editor

Option	Description
Connect to the tuning source automatically	<p>When you open a tuning perspective, it automatically opens the last saved tuning jobs that were open when you closed the application. This option lets you specify whether, in addition, you want to automatically connect to the data sources associated with these tuning jobs. If you typically review existing tuning job archives rather than run new tuning jobs, you may wish to explicitly connect to a data source rather than connect automatically. The options are:</p> <p>Always: Automatically connects to data sources associated with tuning jobs that were open last time you shut down tuning.</p> <p>Never: Automatically opens tuning job archives that were open last time you shut down the application but does not automatically connect to the associated data sources.</p> <p>Prompt: Prompts you to connect to data sources associated with tuning jobs that were open last time you shut down the application.</p>
Color scheme for plan cost	<p>In the graphical representations of explain plan cost and elapsed time, tuning uses a color scheme to highlight differences among generated cases. Values for the original statement are treated as a baseline, and values for individual cases that are within a specified threshold range of the baseline value are represented with a Baseline color. For cases whose values are outside the threshold range, Improvement and Degradation colors are used to represent values in those cases.</p> <p>Tip: You can set the threshold in the application preferences, by selecting Window > Preferences > Tuning Job Editor and then changing the threshold levels as required.</p>
Case execution	Lets you dictate how execution statistics are gathered.
Table analysis	Lets you specify an estimation sample percentage to be used with the Analyze Tables function.

SET GENERATED CASE PREFERENCES

Additionally, the Generated Case preference page lets you enable or disable the automatic generation of SQL Optimizer hint-based cases of SQL statements added to a tuning job. It also lets you indicate which specific hint types are generated when the feature is enabled.



Select **Window > Preferences > SQL Development > Tuning Job Editor > Case Generation**.

Use the Generate cases option automatically after extracting tuning candidates control to enable or disable automatic generation of hint-based cases, and then select the check boxes to specify the hint-based cases that are generated for a statement added to a tuning job.

About Statement Records

Column or column set	Description
SQL Statements and Cases	<p>Identifiers for the generated statement or case:</p> <p>Name: Statements are assigned a numbered identifier based on the order in which they were added to a tuning job.</p> <p>Text: An excerpt of the statement or case based on the statement type (SELECT, INSERT, DELETE, and UPDATE). For details on how to view the entire statement or case.</p>
Cost	<p>An explain plan-based cost estimate. This field is populated as soon as the statement is added to the Overview tab.</p> <p>This column set can be expanded to display a graphical representation of the cost to facilitate comparisons among cases.</p>
Index Analysis	<p>Tuning automatically detects indexes that require optimization and offers you the option to automatically optimize the index. For more information, see "Implementing Index Analysis Recommendations" on page 150.</p>
Elapsed time	<p>The execution time during the most recent execution. This column set is not populated until you execute the statement or case.</p> <p>This column set can be expanded to display a graphical representation of the elapsed time to facilitate comparisons among cases.</p>
Other Execution Statistics	<p>The default, collapsed view has Physical Reads and Logical Reads columns. Expanded, there are also Consistent Gets, Block Gets, Rows Returned, CPU time(s), Parse CPU Time(s), Row Sorts, Memory Sorts, Disk Sorts, and Open Cursors columns. For details on these statistics, refer to your DBMS documentation.</p> <p>This column set is not populated until you execute the statement or case.</p>

EXAMPLES OF TRANSFORMATIONS AND SQL QUERY REWRITES

Cartesian Product Elimination: Detects Cartesian Joins and propose corrections based on analysis of statement, for example suggesting dept.deptno = emp.deptno if emp and dept had no join criteria.

Expression Transformation: Identifies actions on predicates that might suppress index usage such as "where empid + 1 = 1 ", should be "where empid=0"

Invalid Outer Join: Identifies invalid outer joins and suggests more efficient alternatives.

Before	After
SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id (+) AND c.state = 'CA'	SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id (+) AND c.state(+) = 'CA'

Transitivity:

Before	After
<pre>SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id</pre>	<pre>SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id AND i.product_id = pr.product_id</pre>

Move Expression to WHERE Clause

Before	After
<pre>SELECT col_a, SUM(col_b) FROM table_a GROUP BY col_a HAVING col_a > 100</pre>	<pre>SELECT col_a, SUM(col_b) FROM table_a WHERE col_a > 100 GROUP BY col_a</pre>

NULL Column

Before	After
<pre>SELECT * FROM employee WHERE manager_id != NULL</pre>	<pre>SELECT * FROM employee WHERE manager_id IS NUL</pre>

Push Subquery

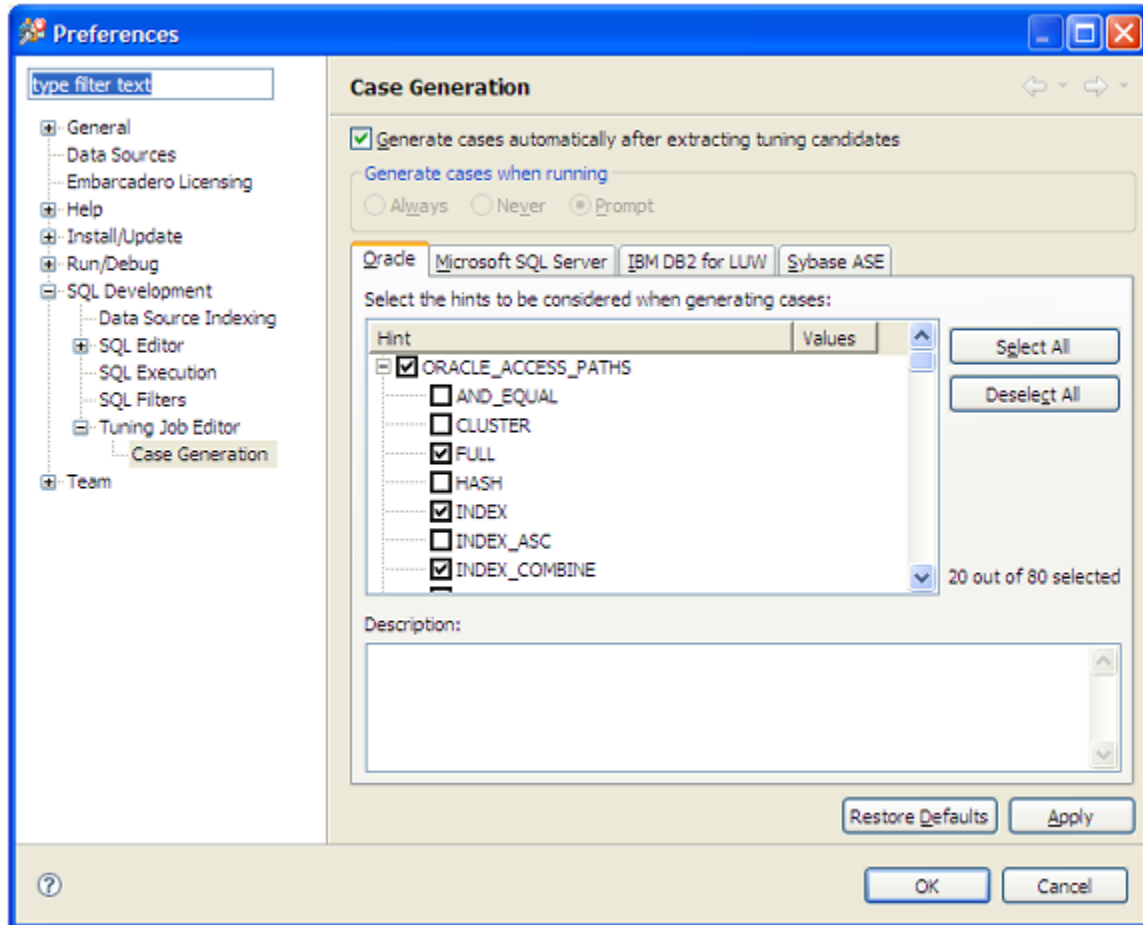
Before	After
<pre>SELECT * FROM employee WHERE employee_id = (SELECT MAX(salary) FROM employee)</pre>	<pre>SELECT employee.* FROM employee, (SELECT DISTINCT MAX(salary) col1 FROM employee) t1 WHERE employee_id = t1.col1</pre>

Mismatched column types: identify joins type mismatch such as number = character which might suppress use of Index

DBMS HINTS

Users can provide hints to a specified platform in order to instruct data source optimizer on the best way to execute SQL statements. Tuning automatically generates cases using these hints.

Hints can be enabled or disabled when cases are being generated by tuning on the **Window > Preferences > Tuning Job Editor > Case Generation** panel. Choose a tab as it pertains to the platform you want to modify and use the check boxes to select and de-select the hints you want to enable or disable, respectively.



The following topics describe platform hints that are packaged in tuning to provide optimal efficiency when executing jobs:

- [Oracle Hints](#) on page 187
- [SQL Server Hints](#) on page 193
- [DB2 Hints](#) on page 194
- [Sybase Hints](#) on page 195

ORACLE HINTS

NOTE: Hint Analysis through SQL hint injection for Oracle data sources is not supported in DB Optimizer XE Developer.

The following table highlights Oracle hints based on Oracle hints optimization:

Category	Hint	Available For	Notes
ACC PATH	AND_EQUAL	/*+ CLUSTER (tablesppec) */	-
ACC PATH	CLUSTER	/*+ FULL (tablesppec) */	Use on Clustered Tables only
ACC PATH	FULL	/*+ HASH (tablesppec) */	Forces a table scan even if there are indexes.
ACC PATH	HASH	/*+ INDEX (tablesppec [TAL: indexspec]) */	Only to tables stored in a table cluster.
ACC PATH	INDEX	/*+ INDEX_ASC (tablesppec [TAL: indexspec]) */	If no indexspec is supplied, the optimizer will try to scan with each avail index.
ACC PATH	INDEX_ASC	/*+ INDEX_COMBINE (tablesppec [indexspec [TAL: indexspec]...]) */	Essentially the same as INDEX.
ACC PATH	INDEX_COMBINE	/*+ INDEX_DESC (tablesppec [indexspec [TAL: indexspec]...]) */	Forces the optimizer to try multiple boolean combinations of indexes.
ACC PATH	INDEX_DESC	/*+ INDEX_DESC (tablesppec [indexspec [TAL: indexspec]...]) */	Essentially the same as INDEX.
ACC PATH	INDEX_FFS	/*+ INDEX_FFS (tablesppec [indexspec [TAL: indexspec]...]) */	Forces an index scan using specified index(es).
ACC PATH	INDEX_JOIN	/*+ INDEX_JOIN (tablesppec [indexspec [TAL: indexspec]...]) */	Indexes used should be based on columns in the where clause.
ACC PATH	INDEX_SS	/*+ INDEX_SS (tablesppec [indexspec [TAL: indexspec]...]) */	Useful with composite indexes where the first column is not used in the query, but others are.
ACC PATH	INDEX_SS_ASC	/*+ INDEX_SS_ASC (tablesppec [indexspec [TAL: indexspec]...]) */	Essentially the same as INDEX_SS.
ACC PATH	INDEX_SS_DESC	/*+ INDEX_SS_DESC (tablesppec [indexspec [TAL: indexspec]...]) */	Essentially the same as INDEX_SS.
ACC PATH	NO_INDEX	/*+ NO_INDEX (tablesppec [indexspec [TAL: indexspec]...]) */	Directs the Optimizer not to use specified index(es).
ACC PATH	NO_INDEX_FFS	/*+ NO_INDEX_FFS ([tablesppec [indexspec [TAL: indexspec]...]) */	Directs the Optimizer to exclude a fast full scan of the specified index(es).
ACC PATH	NO_INDEX_SS	/*+ NO_INDEX_SS (tablesppec [indexspec [TAL: indexspec]...]) */	Directs the Optimizer to exclude a skip scan of the specified index(es).
ACC PATH	ROWID	-	-
JOIN OP	HASH_AJ	-	-

Category	Hint	Available For	Notes
JOIN OP	HASH_SJ	-	-
JOIN OP	MERGE_AJ	-	-
JOIN OP	MERGE_SJ	-	-
JOIN OP	NL_AJ	-	-
JOIN OP	NL_SJ	-	-
JOIN OP	NO_USE_HASH	/*+ NO_USE_HASH (tablespec [TAL: tablespec]...) */	Negates the use of hash joins for the table specified.
JOIN OP	NO_USE_MERGE	/*+ NO_USE_MERGE (tablespec [TAL: tablespec]...) */	Negates the use of sort-merge joins for the table specified.
JOIN OP	NO_USE_NL	/*+ NO_USE_NL (tablespec [TAL: tablespec]...) */	Negates the use of nested-loop joins for the table specified.
JOIN OP	USE_HASH	/*+ USE_HASH (tablespec [TAL: tablespec]...) */	Directive to join each table specified using a hash join.
JOIN OP	USE_MERGE	/*+ NO_USE_MERGE (tablespec [TAL: tablespec]...) */	Directive to join each table specified using a sort-merge join.
JOIN OP	USE_NL	/*+ NO_USE_NL (tablespec [TAL: tablespec]...) */	Directive to use a nested-loop join with the specified tables as the inner table.
JOIN OP	USE_NL_WITH_INDEX	/*+ USE_NL_WITH_INDEX (tablespec [indexspec [TAL: indexspec]...]) */	Directive to use a nested-loop join with the specified table as the inner table using the index specified to satisfy at least one predicate.
JOIN ORDER	LEADING	/*+ LEADING (tablespec) */	Directive to join the tables in the order specified.
JOIN ORDER	ORDERED	/*+ ORDERED */	Directive to join tables in the order found in the FROM clause.
JOIN ORDER	STAR	-	-
OPT APPROACH	ALL_ROWS	/*+ ALL_ROWS */	Indicates the goal is overall throughput.
OPT APPROACH	CHOOSE	-	-
OPT APPROACH	FIRST_ROWS	/*+ FIRST_ROWS (integer) */	The goal is to retrieve the first row(s) as fast as possible.
OPT APPROACH	RULE	/*+ RULE */	Used to disable the COST based optimizer.
OTHER	CACHE	/*+ CACHE (tablespec) */	Should be used with the FULL hint. Places data in the most-recently used area of the buffer cache.
OTHER	APPEND	/*+ APPEND */	Directs the optimizer to INSERT data at the end of the existing table data using direct path I/O.

Category	Hint	Available For	Notes
OTHER	CURSOR_SHARING_EXACT	<code>/*+ CURSOR_SHARING_EXACT */</code>	Directs the Optimizer to ignore previously parsed SQL that matches, but uses bind variables. Forces the SQL to be parsed unless an exact match is found.
OTHER	DRIVING_SITE	<code>/*+ DRIVING_SITE (tablespec) */</code>	Used when data is joined remotely via DBLink. Normally data at the remote site is returned to the local and joined. This hint directs the optimizer to send the local data to the remote site for resolution of the join.
OTHER	DYNAMIC_SAMPLING	<code>/*+ DYNAMIC_SAMPLING ([TAL: tablespec] integer) */</code>	Only used in simple SELECT statements with a single table to approximate cardinality if there are no existing statistics on the table.
OTHER	MODEL_MIN_ANALYSIS	<code>/*+ MODEL_MIN_ANALYSIS */</code>	Used with spreadsheet and model analysis to minimize compile time.
OTHER	NO_PUSH_PRED	<code>/*+ NO_PUSH_PRED [TAL: (tablespec)] */</code>	Opposite of PUSH_PRED, it directs the Optimizer not to try to push the predicate into the view.
OTHER	NO_PUSH_SUBQ	<code>/*+ NO_PUSH_SUBQ] */</code>	Opposite of PUSH_SUBQ, it directs the Optimizer not to try and evaluate the subquery first.
OTHER	NO_UNNEST	<code>/*+ NO_UNNEST */</code>	Subqueries in the WHERE clause are considered nested. A subquery can be evaluated several times for multiple results in the "parent". Unnesting evaluates the subquery once and merges the results with the body of the "parent". This hint directs the Optimizer NOT to unnest.
OTHER	NOAPPEND	<code>/*+ NOAPPEND */</code>	Directs the Optimizer to utilize existing space in a table and negates parallel processing.
OTHER	NOCACHE	<code>/*+ NOCACHE (tablespec) */</code>	Should be used with the FULL hint. Places data in the least-recently used area of the buffer cache.
OTHER	OPT_PARAM	-	-
OTHER	ORDERED_PREDICATES	-	-
OTHER	PUSH_PRED	<code>/*+ PUSH_PRED [TAL: (tablespec)] */</code>	Used when one of the tables in a join is an in-line view. Forces the predicate used to join the table and the view into the view.
OTHER	PUSH_SUBQ	<code>/*+ PUSH_SUBQ *</code>	Used with an EXISTS or IN subselect to force evaluation of the subquery rather than the default behavior of the last.

Category	Hint	Available For	Notes
OTHER	UNNEST	<code>/*+ UNNEST */</code>	Subqueries in the where clause are considered nested. A subquery could be evaluated several times for multiple results in the "parent". Unnesting evaluates the subquery once and merges results with the body of the "parent".
PARALLEL	NO_PARALLEL	<code>/*+ NO_PARALLEL (tablespec) */</code>	Directs the Optimizer not to parallel the specified table.
PARALLEL	NO_PARALLEL_INDEX	<code>/*+ NO_PARALLEL_INDEX (tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optimizer not to parallel the specified index(es).
PARALLEL	NO_PX_JOIN_FILTER	<code>/*+ NO_PX_JOIN_FILTER (tablespec) */</code>	Directs the Optimizer not to try and join bitmap indexes in parallel.
PARALLEL	NOPARALLEL	<code>/*+ NOPARALLEL (tablespec) */</code>	Directs the Optimizer not to parallel the specified table.
PARALLEL	NOPARALLEL_INDEX	<code>/*+ NOPARALLEL_INDEX (tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optimizer not to parallel the specified index(es).
PARALLEL	PARALLEL	<code>/*+ PARALLEL (tablespec [integer TAL:DEFAULT]) */</code>	Number specifies degrees of parallelism (how many processes).
PARALLEL	PARALLEL_INDEX	<code>/*+ PARALLEL_INDEX (tablespec [indexspec [TAL: indexspec]...] integer DEFAULT) */</code>	Number specifies degree of parallelism (how many processes).
PARALLEL	PQ_DISTRIBUTE	<code>/*+ PQ_DISTRIBUTE(tablespec outer_distribution inner_distribution) */</code>	Used in parallel join operations to indicate how inner and outer tables of the joins should be processed. The values of the distributions are HASH, BROADCAST, PARTITION, and NONE. Only six combinations table distributions are valid.
PARALLEL	PX_JOIN_FILTER	<code>/*+ PX_JOIN_FILTER (tablespec) */</code>	Directs the Optimizer to try and join bitmap indexes in parallel.
QUERY TRANS	EXPAND_GSET_TO_UNION	<code>/*+ EXPAND_GSET_TO_UNION */</code>	Performs transformations on queries that have GROUP BY into Unions.
PARALLEL	FACT	<code>/*+ FACT (tablespec) */</code>	In the context of STAR transformation, this table should be considered a FACT table (as opposed to a DIMENSION).
PARALLEL	MERGE	<code>/*+ MERGE ([view tablespec]) */</code>	Use with either an in-line view that has a Group by or Distinct in it as a joined table, or with the use of IN subquery to "merge" the "view" into the body of the rest of the query.

Category	Hint	Available For	Notes
PARALLEL	NO_EXPAND	/*+ NO_EXPAND */	Used when OR condition (including IN lists) is present in the predicate to not consider transformation to compound query.
PARALLEL	NO_FACT	/*+ NO_FACT (tablespec) */	In the context of STAR transformation this table should not be considered a FACT table.
PARALLEL	NO_MERGE	/*+ NO_MERGE [([view TAL:tablespec]) */	Directs the Optimizer not to "merge" the view into the query.
PARALLEL	NO_QUERY_TRANSFORMATION	/*+ NO_QUERY_TRANSFORMATION */	Directs the Optimizer not to transform OR, in-lists, in-line views, and subqueries. Try it whenever any of these conditions are present.
PARALLEL	NO_REWRITE	/*+ NO_REWRITE */	Directs the Optimizer not to use a Materialized View, even if one is available.
PARALLEL	NO_STAR_TRANSFORMATION	/*+ NO_STAR_TRANSFORMATION */	Directs the Optimizer not to try a Star Transformation.
PARALLEL	NO_XML_QUERY_REWRITE	/*+ NO_XML_QUERY_REWRITE */	Use only if the query is using XML functionality.
PARALLEL	NO_XMLINDEX_REWRITE	/*+ NO_XMLINDEX_REWRITE */	Use only if the query is using XML functionality.
PARALLEL	NOFACT	/*+ NOFACT (tablespec) */	In the context of STAR transformation, this table should not be considered a FACT table.
PARALLEL	NOREWRITE	/*+ NOREWRITE	Directs the Optimizer not to use a Materialized View, even if one is available.
PARALLEL	REWRITE	/*+ REWRITE [(view [TAL: view]...)] */	Directs the Optimizer to use a Materialized View instead of the underlying tables. Specify REWRITE without additional parameters. Oracle will determine if it can use a Materialized View or not.
PARALLEL	STAR_TRANSFORMATION	/*+ STAR_TRANSFORMATION */	Directs the Optimizer to try Star Transformation. Only try with a 3 table or more join.
PARALLEL	USE_CONCAT	/*+ USE_CONCAT */	Used when the OR condition (including IN lists) is present in the predicate to transform the query into a compound UNION ALL.

Category	Hint	Available For	Notes
REAL TIME	MONITOR	/*+ MONITOR */	Effective only if STATISTICS_LEVEL initialization parameter is either set to ALL or TYPICAL and CONTROL_MANAGEMENT_PACK_ACCESS is set to DIAGNOSTIC+TUNING. Turns on features of the Oracle Database Tuning Pack.
REAL TIME	NO_MONITOR	/*+ NO_MONITOR */	See MONITOR hint.

SQL SERVER HINTS

The following table highlights SQL hints based on MS SQL Server hints optimization:

Category	Hint	Available For	Notes
JOIN	LOOP	SELECT/UPDATE/DELETE	Not applicable for RIGHT OUTER or FULL joins.
JOIN	HASH	SELECT/UPDATE/DELETE	-
JOIN	MERGE	SELECT/UPDATE/DELETE	-
JOIN	REMOTE	SELECT/UPDATE/DELETE	Only for INNER JOINS. Not applicable with COLLATE
		SELECT/UPDATE/DELETE	-
QUERY	RECOMPILE	SELECT/UPDATE/DELETE	-
QUERY	FORCE ORDER	SELECT/UPDATE/DELETE	-
QUERY	ROBUST PLAN	SELECT/UPDATE/DELETE	-
QUERY	KEEP PLAN	SELECT/UPDATE/DELETE	-
QUERY	KEEPFIXED PLAN	SELECT/UPDATE/DELETE	-
QUERY	EXPAND VIEWS	DML Statements	Only for statement containing views.
QUERY	HASH GROUP	SELECT	Only when GROUP BY, COMPUTE and DISTINCT clauses are used.
QUERY	ORDER GROUP	SELECT/UPDATE/DELETE	Only when GROUP BY, COMPUTE and DISTINCT clauses are used.
QUERY	MERGE UNION	SELECT	Only for statements chained using UNION
QUERY	HASH UNION	SELECT	Only for statements chained using UNION
QUERY	CONCAT UNION	SELECT	Only for statements chained using UNION
QUERY	LOOP JOIN	SELECT/UPDATE/DELETE	-
QUERY	MERGE JOIN	SELECT/UPDATE/DELETE	-
QUERY	HASH JOIN	SELECT/UPDATE/DELETE	-
TABLE	INDEX()	DML Statements	Only for tables and views with indexes.

Category	Hint	Available For	Notes
TABLE	KEEPIDENTITY	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	KEEPDEFAULTS	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	HOLDLOCK	DML Statements	Not applicable for SELECT statements using FOR BROWSE clause.
TABLE	IGNORE_CONSTRAINTS	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	IGNORE_TRIGGERS	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	NOLOCK	SELECT/UPDATE/COMPLETE	Not applicable for the target table in UPDATE/DELETE statements.
TABLE	NOWAIT	DML Statements	-
TABLE	PAGLOCK	DML Statements	-
TABLE	READCOMMITTED	DML Statements	-
TABLE	READCOMMITTEDLOCK	SELECT/UPDATE/COMPLETE	-
TABLE	READPAST	SELECT/UPDATE/COMPLETE	Not applicable for the target table in UPDATE/DELETE statements.
TABLE	READUNCOMMITTED	SELECT/UPDATE/COMPLETE	Not applicable for the target table in UPDATE/DELETE statements.
TABLE	REPEATABLEREAD	DML Statements	-
TABLE	ROWLOCK	DML Statements	-
TABLE	SERIALIZABLE	DML Statements	Not applicable for SELECT statements using FOR BROWSE clause.
TABLE	TABLOCK	DML Statements	-
TABLE	TABLOCKX	DML Statements	-
TABLE	UPDLOCK	DML Statements	-
TABLE	XLOCK	DML Statements	-
TABLE	FASTFIRSTROW	DML Statements	-

DB2 HINTS

The following table highlights SQL hints based on IBM DB2 hints optimization:

Category	Hint	Notes
Command	SET OPTIMIZATION LEVEL	For top-level SELECT statements only
Clause	optimize for <n> rows	For top-level SELECT statements only

Category	Hint	Notes
Clause	fetch first <n> rows only	For SELECT statements only

SYBASE HINTS

The following table highlights SQL hints based on Sybase hints optimization:

Category	Hint	Notes
Logical	distinct	No explicit implementation
Logical	group	No explicit implementation
Logical	g_join	No explicit implementation
Logical	nl_g_join	Not applicable for: statements with chained queries; select statements with group by clause and having clause or group by clause and order by clause
Logical	m_g_join	Not applicable for: statements with chained queries; select statements with group by clause and having clause or group by clause and order by clause
Logical	join	No explicit implementation
Logical	nl_join	Not applicable for: select statements with group by clause and having clause or group by clause and order by clause
Logical	m_join	Not applicable for: select statements with group by clause and having clause or group by clause and order by clause
Logical	h_join	Not applicable for: select statements with group by clause and having clause or group by clause and order by clause
Logical	union	No explicit implementation
	scan	No explicit implementation
Logical	scalar_agg	Only used in combination with other operators. It does not change the execution plan itself.
Logical	sequence	Is a keyword that will be used in the implementation of scalar_agg operator.
Logical	hints	We don't support a combination of hints
Logical	prop	Uses a set of pre-defined values.
Logical	table	Used only in combination with other operators, when referring tables from subqueries
Logical	work_t	This operator is applicable only together with store operator

Category	Hint	Notes
Logical	in	Used only in combination with other operators, when referring tables from subqueries
Logical	subq	Used only in combination with other operators, when referring tables from subqueries
Physical	distinct_sorted	Only for SELECT statements containing DISTINCT, and only for tables
Physical	distinct_sorting	Only for SELECT statements containing DISTINCT, and only for tables
Physical	distinct_hashing	Only for SELECT statements containing DISTINCT, and only for tables
Physical	group_sorted	Only for SELECT statements (not working for views) with no having and no order by clause.
Physical	group_hashing	Only for SELECT statements (not working for views) with no having and no order by clause.
Physical	group_inserting	Not implemented
Physical	append_union_all	Not applicable for: UNION chained clauses, nested sub-selects in a from clause, if a group by clause is present or if scalar aggregation is present
Physical	merge_union_all	Not applicable for: UNION ALL chained clauses, nested sub-selects in a from clause, or if a group by clause is present.
Physical	merge_union_distinct	Not applicable for: UNION ALL chained clauses, nested sub-selects in a from clause, or if a group by clause is present.
Physical	hash_union_distinct	Not applicable for: UNION ALL chained clauses, nested sub-selects in a from clause, if a group by clause is present, or if scalar aggregation is present.
Physical	i_scan	Applied to all table references in the from clause of the main select and of the sub select statements except: 1. statement has sub-selects. 2. table references has no indexes.

Category	Hint	Notes
Physical	t_scan	Applied to all the table references in the from clause of the main select and of the sub select statements except: On Sybase 12.5 not applied for tables in the main query if: 1. statement has chained queries. 2. Sub queries have group by and having clauses; and not applied to the tables in sub selects if: 1. has select statements in from clause of the main select. 2. sub queries have group by and having clauses. 3. statement has select statements in select clause. 4. statement has parent statement and insert statement; on Sybase 15 not applied for tables in sub selects if: 1. has select statements in from clause of the main select. 2. statement has chained queries.
Physical	m_scan	Applied for all tables if in the where clause there is a condition like: table1.indexedColumn1 condition body OR table1.indexedColumn2 condition body; Not applied if the LIKE operator is used. For columns that belong to a primary key only the first column is considered.
Physical	store	-
Physical	store_index	-
Physical	sort	-
Physical	xchg	-

TUTORIALS

The tutorials enable you to get started using DB Optimizer. Their purpose is to provide you with the foundation needed to fully utilize the features and benefits of the products, and apply them to your own enterprise. This section contains information on how to register data sources from your enterprise, and how to profile, analyze, and tune SQL statements, sessions, and events for the purpose of optimizing the efficiency of your data sources, as well as identify and prevent system bottlenecks and other wait-related issues.

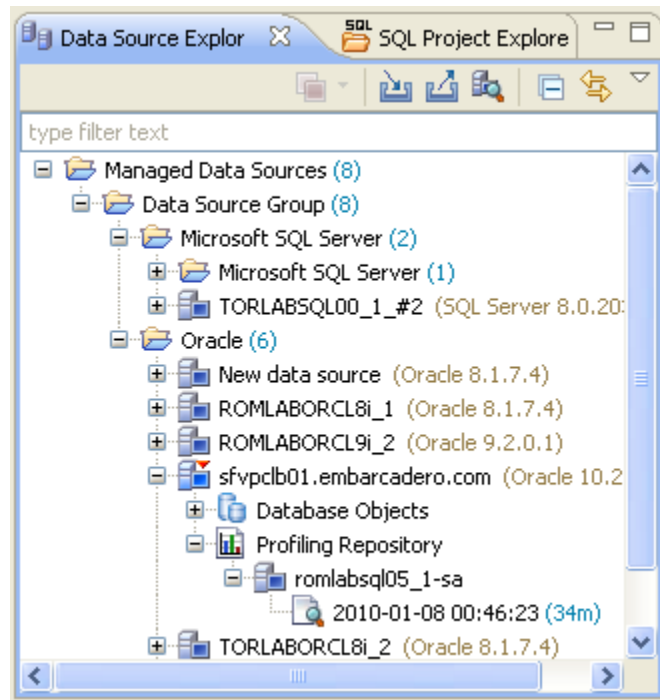
This guide is comprised of the following sections:

- [Working with Data Source Explorer](#) on page 200
- [Profiling a Data Source](#) on page 204
- [Tuning SQL Statements](#) on page 214
- [SQL Code Assist and Execution](#) on page 226

Once you have started the application, you can select Help from the Menu Bar to find additional resources that compliment and build upon the features and tasks presented in this guide.

WORKING WITH DATA SOURCE EXPLORER

Data Source Explorer provides a tree view of all data sources registered in DB Optimizer. In addition, it breaks down the components of each data source and categorizes databases and corresponding database objects by object type and underlying SQL code. This feature provides you with a view of the contents of data sources in your enterprise in an easily navigated and cataloged interface. If you have implemented a Profiling Repository for your Oracle data sources then you can access saved profiling sessions from the Data Source Explorer

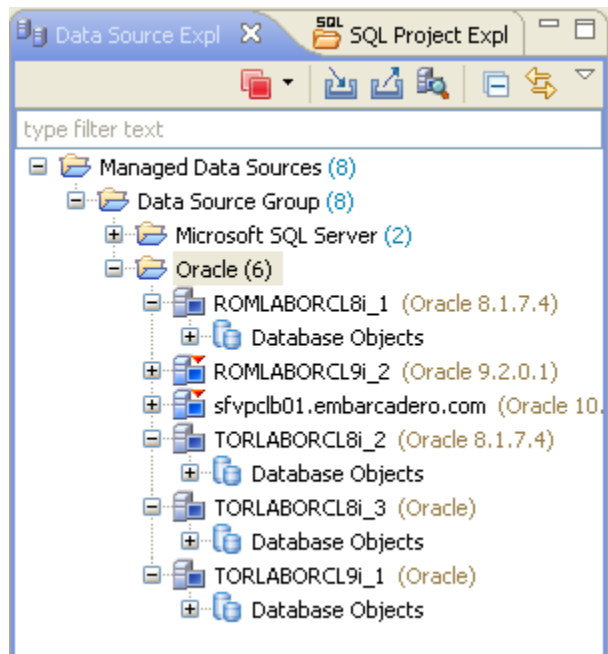


Data Source Explorer sorts databases and database objects by category within DB Optimizer.

If data sources are particularly large or complex, or you are only developing specific objects, you can apply database object filters on the view in Data Source Explorer.

ADDING DATA SOURCES

In order to profile and tune statements, you need to register the data sources to be analyzed in the environment by providing connection information and other details to DB Optimizer. Data sources are registered and managed in Data Source Explorer. Each time you register a new data source you need to specify its connection information and organize it in the View, as needed. Once a data source has been registered, it remains stored in a catalog and does not need to be registered again each time you open DB Optimizer. Furthermore, it can be used in multiple jobs, archived, or otherwise “shared” with regards to the general functionality of the application.

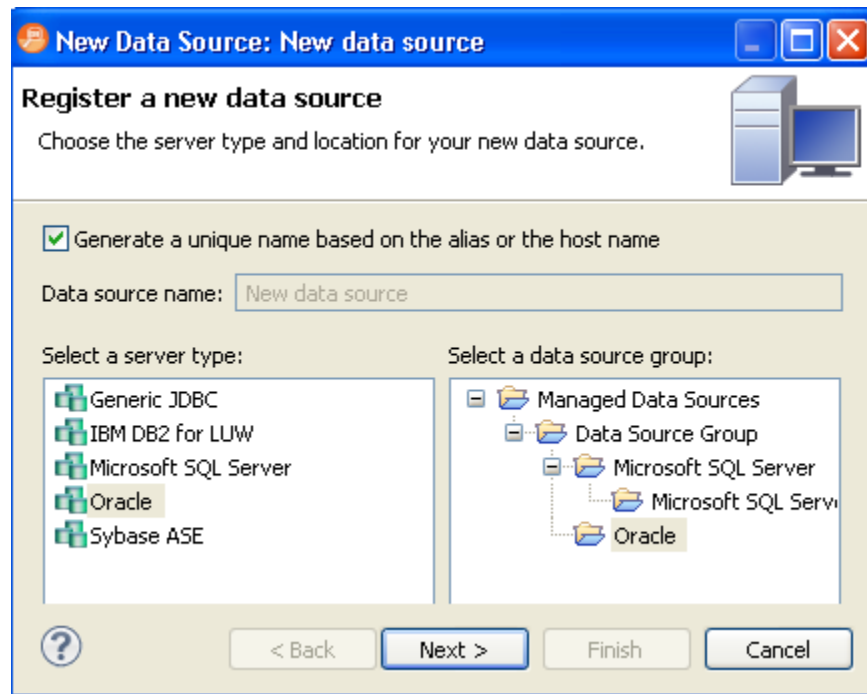


The Data Source Explorer view provides an organizational tree of registered data sources and the parameters associated with them.

To add a data source:

- 1 Select **Data Source Explorer** and choose **File > New > Data Source** from the **Menu** bar.

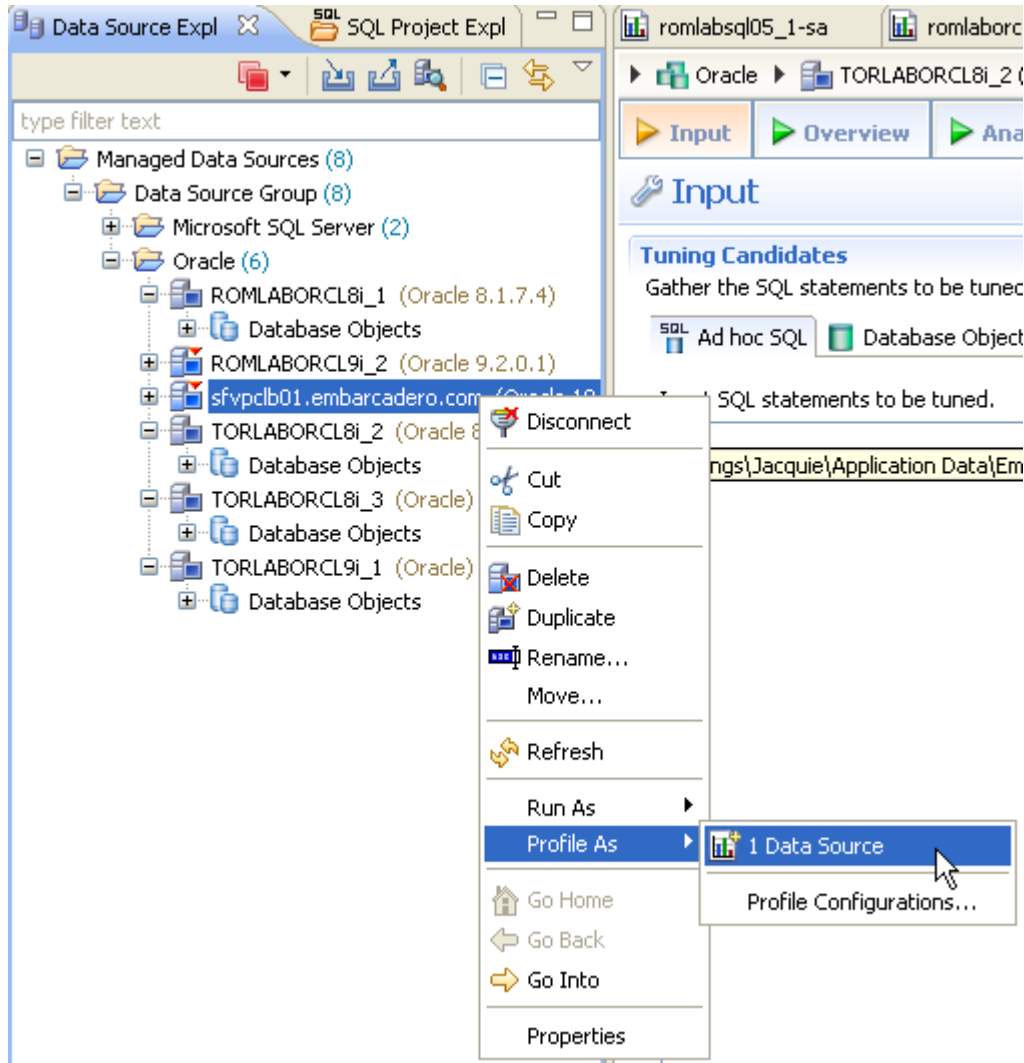
The **Data Source Wizard** appears.



- 2 Follow the steps provided to register the data source. When you are finished, the data source is added to Data Source Explorer.

BROWSING DATA SOURCES

In order to analyze and tune statements on data sources within your enterprise, you need to access the data source objects within the application environment. It is important to be able to view databases and underlying code in an organized manner, especially when maintaining a large system. Data Source Explorer's tree structure can be used to view databases, tables, and other information about data sources. Expand the nodes of each registered data source to view details about each data source. Data Source Explorer is also used to launch profiling sessions, by enabling you to select the data source that you want to execute, and then launching a profiling session from your selection.



The Data Source Explorer tree provides a list of databases, and enables you to launch Optimizer features, such as SQL Profiler, via the context menu.

All objects listed in Data Source Explorer are organized, in descending order, by data source. Additionally, if you double-click on an object, SQL Editor will open and display the underlying SQL code.

Notice in the screenshot above, there is a Profiling Repository on the last data source in the list of Oracle data sources. For Oracle data sources only, you can save your profiling sessions to the Profiling Repository for later examination. We'll discuss this more later in "[Saving a Profiling Session](#)" on page 212.

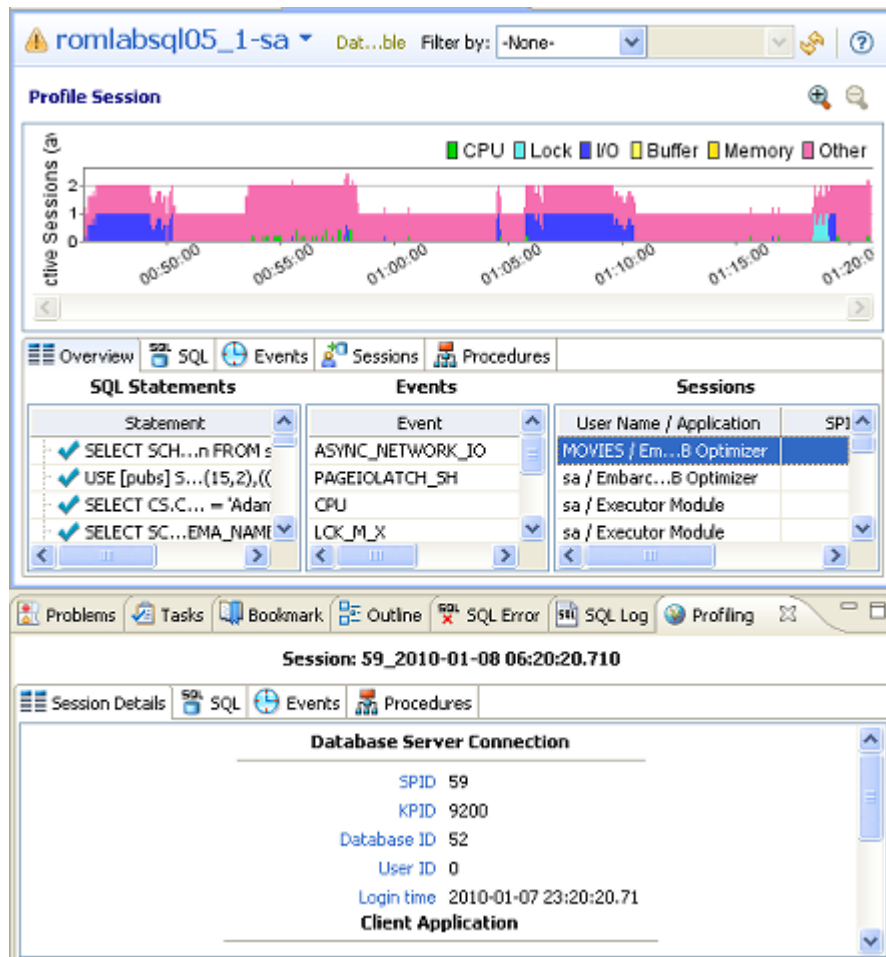
To browse a data source:

Expand the node of the data source that you want to examine. Double-click an object to view the code in SQL Editor.

PROFILING A DATA SOURCE

NOTE: The SQL Profiler is not available in the "Basic" version of DB Optimizer.

SQL Profiler is an interface component that constantly samples a data source and builds a statistical model of the load on the database. Profiling is used to locate and diagnose problematic SQL code and event-based bottlenecks. Additionally, Profiler enables you to investigate execution and wait time event details for individual stored routines. Results are presented in the profiling editor, which enables you to identify problem areas and view individual SQL statements, as needed.



SQL Profiler analyzes and provides a statistical model of database load that identifies problematic code and event-based bottlenecks.

SQL Profiler is composed of the following diagnostic parts:

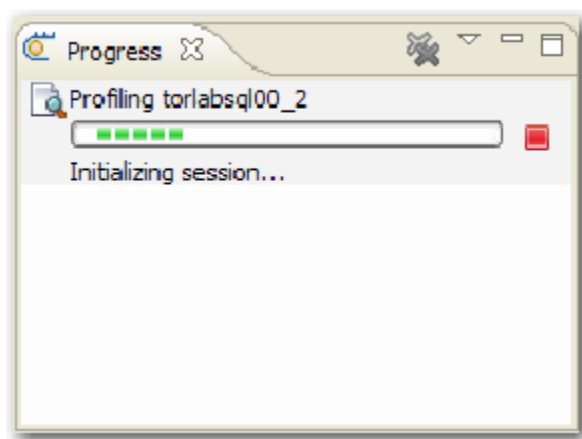
- The **Load Graph** provides a display of the overall load on the system. The bars represent individual aspects of the data source.
- The **Top Activity** section displays where load originates. Specifically, Top Activity displays the top SQL statements, the top events the database spends time in, and the top activity sessions that may be causing issues in terms of query time and response.
- The **Profiling Details** view displays detailed information for any item selected from Top Activity. For example, examining a SQL statement from Top Activity displays the identification parameters and the execution statistics of that specific statement selection.

STARTING A PROFILING SESSION

In order to access SQL Profiler, you need to run a profiling session on a data source registered in Data Source Explorer. You can do this by right-clicking on a data source and selecting **Profile As ...> Data Source** from the context menu. You can also click the Profile icon in the Toolbar, which initiates a profile session on the last selected data source in Data Source Explorer.

Once a profiling session launches, it runs until you stop it or it has run for its specified length of time. You can stop a session by clicking the Stop icon on the right-hand side of the Profiling progress bar.

When the profiling session is complete, the first two sections (Load Chart and Top Activity) will be populated with information about the database load. You can then begin analyzing the data and identifying problem areas.



The Progress view indicates that the profiling session has been initiated.

To run a profiling session:

In Data Source Explorer, right-click on a data source and select **Profile As ...> Data Source**. The profiling session begins.

ANALYZING SESSION DATA

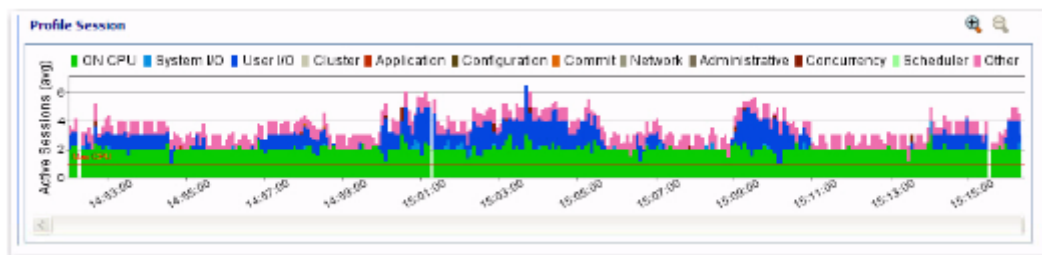
Profiler is composed of three essential diagnostic views that provide information about load on a particular database in the system. These views enable you to identify bottlenecks and view details about specific queries that execute and wait over the course of a profiling session.

SQL Profiler is composed of the following three components, listed in descending order of granularity:

- **Load Chart.** For more information, see "[Load Chart](#)" on page 207.
- **Top Activity.** For more information, see "[Top Activity](#)" on page 208.
- **Profiling Details.** For more information, see "[Profiling Details](#)" on page 209.

LOAD CHART

The Load Chart provides an overview of the load on the system, and is designed as a high level entry point to reading session results. The colored bars represent individual aspects of the database, and the graph can be used to discover bottlenecks.

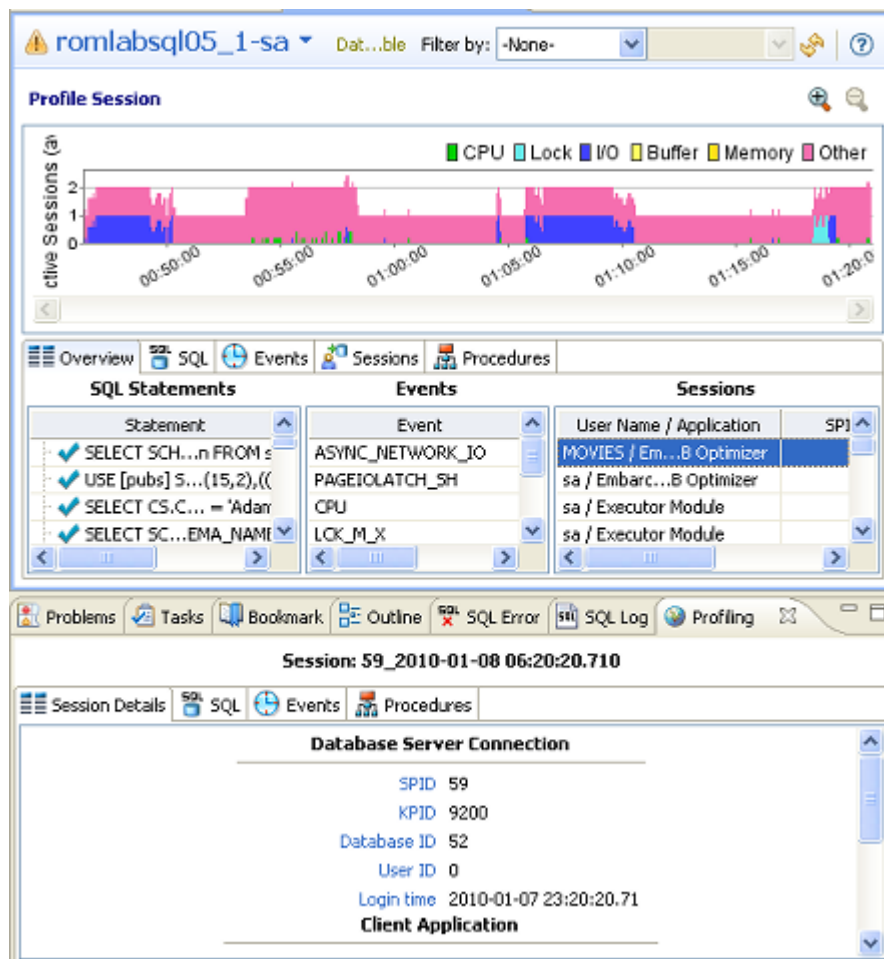


The Load Chart displays the overall load of the database that you analyzed with Profiler.

Time is displayed on the X axis, and the Y axis shows the average number of sessions waiting or executing. Each support platform type has a specific set of wait event times. For example, Sybase platforms will display CPU, Lock, Memory, I/O, Network, and Other. Use the chart legend to understand the graph. It displays a color and code scheme for executing and waiting session categories in the upper right-hand corner of the chart.

TOP ACTIVITY

Below the Load Graph, the Top Activity section displays where the load originates and outlines the top SQL statements, top events, and the top activity sessions on the database. It is composed of a series of tabs that provide detailed statistics on individual SQL statements and sessions that are waiting or executing over the length of the profiling session.



The Top Activity section displays a more detailed view of the Load Graph. It identifies top statements, events, and sessions that are waiting or executing over the length of the profiling session.

- The **Overview** tab provides summary information about SQL statements and events and their activity levels, and sessions, their system process IDs and their activity level. You can reorder the rows in any of the three sections of this tab. For example, clicking the Event column in the Events section changes the alphabetical order to ascending or descending.
- The **SQL** tab provides information about SQL statements and procedures. This includes all INSERT, SELECT, DELETE, and UPDATE statements that are executing or waiting to execute over the length of the profiling session.
- The **Events** tab displays information about wait events, and should be used to tune at the application or database configuration level. For example, if the top events are locks, then application logic needs to be examined. If the top events are related to database configuration, then the database setup should be investigated.

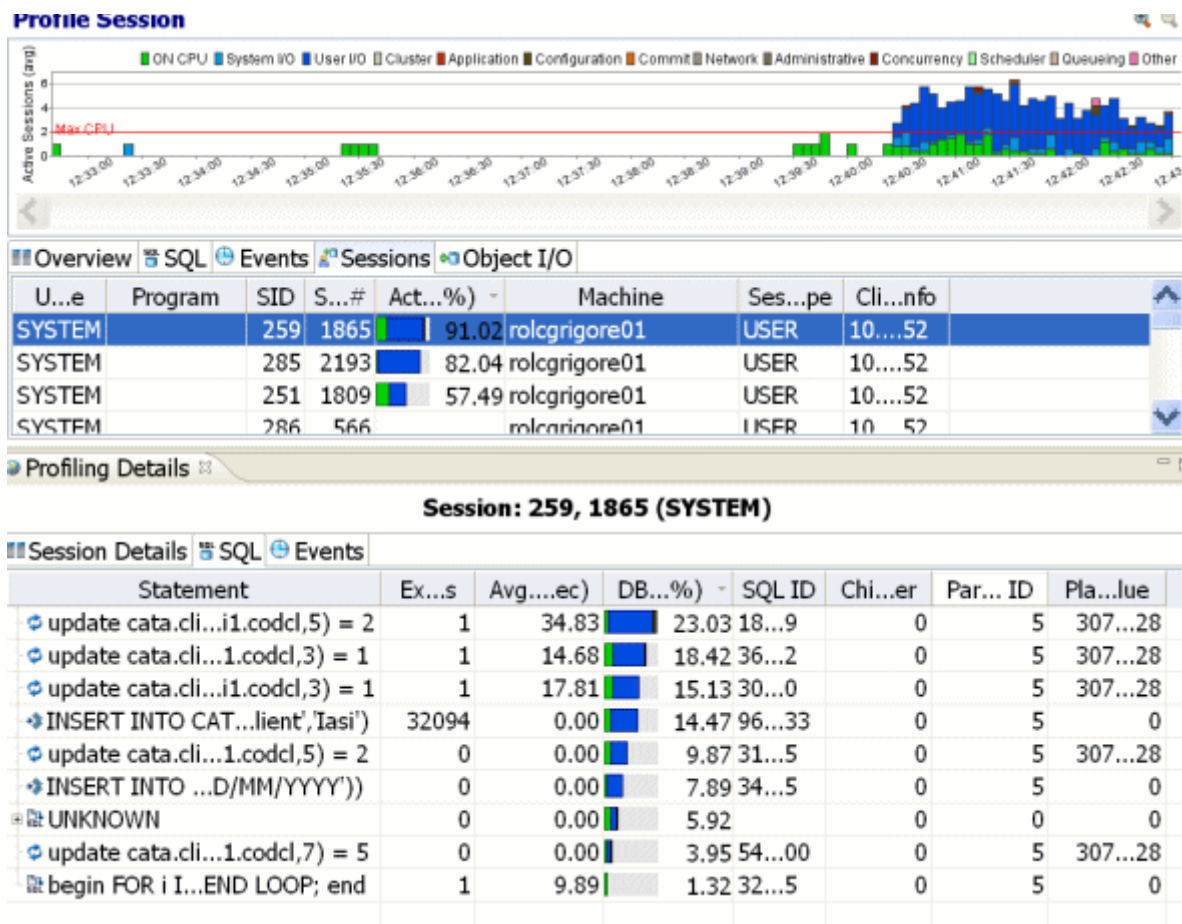
- The **Sessions** tab displays information about sessions, and can be used to discover sessions that are very active or bottlenecked.
- The **Procedures** tab displays information about procedures profiled by the execution process. This is available for Sybase and SQL Server only.
- The **Object I/O** tab provides information about I/O. This tab will be displayed only if you are profiling a database on the Oracle platform.

PROFILING DETAILS

When you select any item from Top Activity, details are displayed on the Profiling Details view.

TIP: You may have to select **Windows > Show View > Profiling Details** to display the Profiling Details view.

The tabs that compose the Profiling Details view are dependent on the nature of the object selected in Top Activity, in order to reflect that item's specific information. The tabs are also dependent on the data source platform. For example, the Object I/O tab is available only for the Oracle platform whereas the Procedures tab is available only for the SQL Server and Sybase platforms.



Profiling Details displays detailed parameter information on a statement, event, or session that was selected in Top Activity. The data displayed also varies by database platform.

Depending on the data source platform you have specified, the tabs that appear in the view will be different, in order to accommodate the parameter specifics of the statement you have selected.

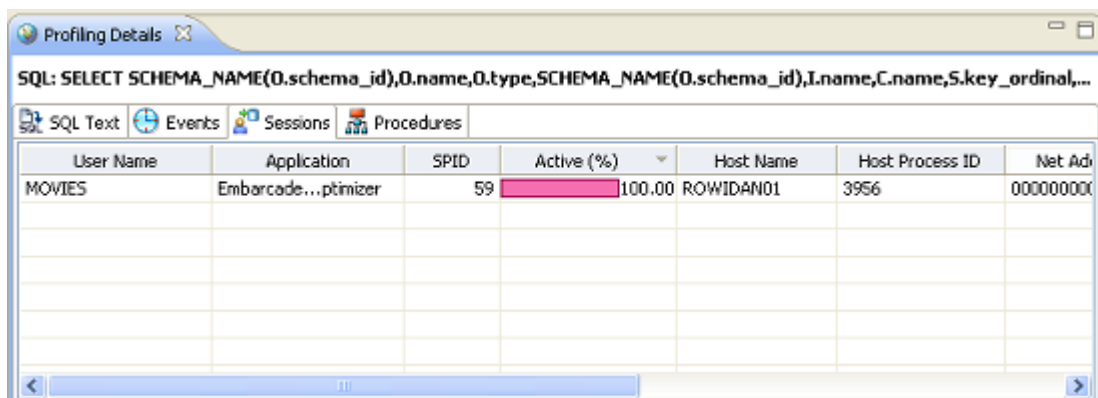
Depending on the top activity selected and the profiled platform types, some tabs may not be available.

NOTE: When right-clicking on a SQL statement in the Top Activity Section in Profiling, if the SQL statement is run by a different user than the user who is running DBO, than the User Mismatch dialog appears, with an example of the following message: "This query was executed by [SOE] and you are currently connected as [system]. We recommend you reconnect as [SOE] to tune the SQL. Would you like to continue anyway?" This message indicates that the statement is being tuned by a user other than the user who originally ran the query, and tables may be missing based on the different schemas. Click OK to run the query, or click Cancel and run tuning under the original user.

To view session details:

- 1 In the **Profile Session** area of the **SQL Profiler**, in the Sessions column click anywhere in the row of an application that ran during the profiling session.
- 2 In the **Profiling Details** area, click the **Sessions** tab.

Details of the session are displayed.

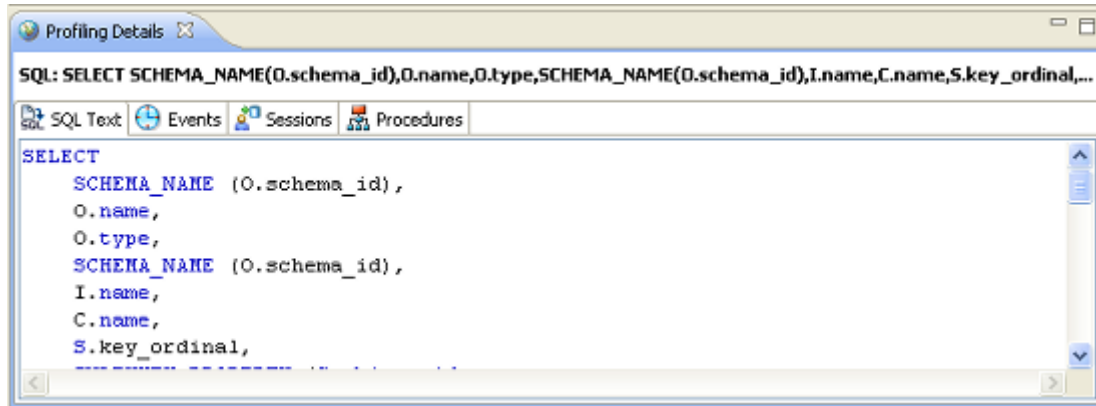


User Name	Application	SPID	Active (%)	Host Name	Host Process ID	Net Ad
MOVIES	Embarcade...ptimizer	59	100.00	ROWIDAN01	3956	00000000

To view SQL:

- 1 In the **Profile Session** area of the **SQL Profiler**, click anywhere in the row of an application that ran during the profiling session.
- 2 In the **Profiling Details** area, click the **SQL Text** or **SQL** tab.

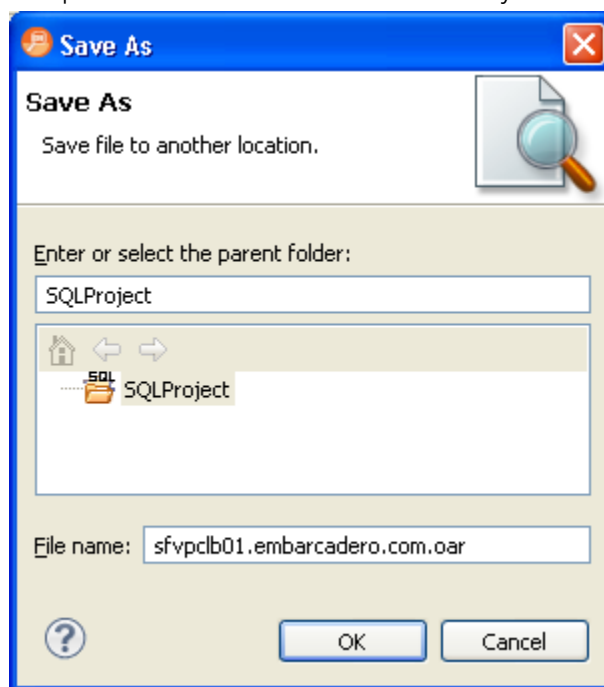
The SQL text for the application that selected object displays.



TIP: From the SQL tab you can easily tune a statement by right-clicking a statement on the SQL tab to initiate the tuner, which then opens with the selected statement in the Ad hoc SQL tab of the Tuner Input.

SAVING A PROFILING SESSION

A profiling session can be saved to a file with a .oar suffix that contains the name of the data source. This enables you to open the file at a later time for analysis.



Profiling sessions can be saved as .oar files for use at a later time.

Once you have saved a profiling session to disk as a .oar file, it appears in the SQL Project Explorer under the name you saved it as. It can be opened again by double-clicking the project name.

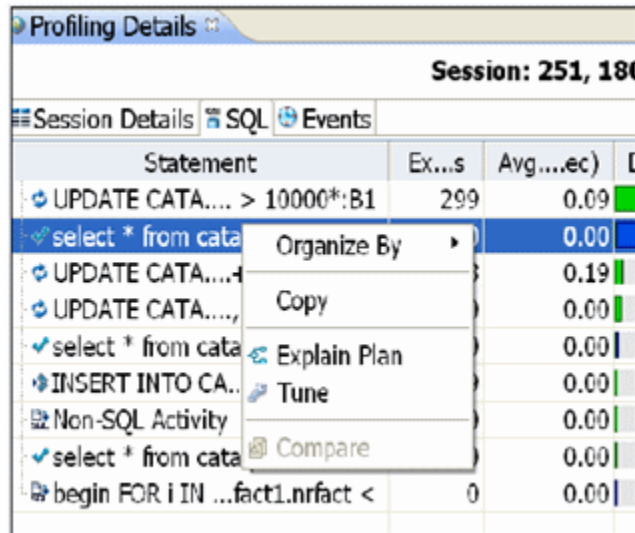
To save a profiling session:

Select the profiling session and then choose **File > Save As**. Specify the project location you want to save the file in and modify the file name, as needed. Click **OK**. The project is added to SQL Project Explorer.

When profiling Oracle datasources you can alternatively save your profiling sessions to the Profiling Repository which appears in the Data Source Explorer. This enables you to profile a data source for an extended period of time, for days or weeks even. Through the Profiling Repository you can also share your profiling session data for other DB Optimizer users to view and analyze. For more information, see "[Work with the Profiling Repository](#)" on page 100.

IMPORTING STATEMENTS TO SQL TUNER

SQL Profiler enables you to submit one or more statements into SQL Tuner. This enables you to take advantage of Tuner's hint-based and transformation-based suggestions if you want to tune a problem statement that you detected over the course of a profiling session.



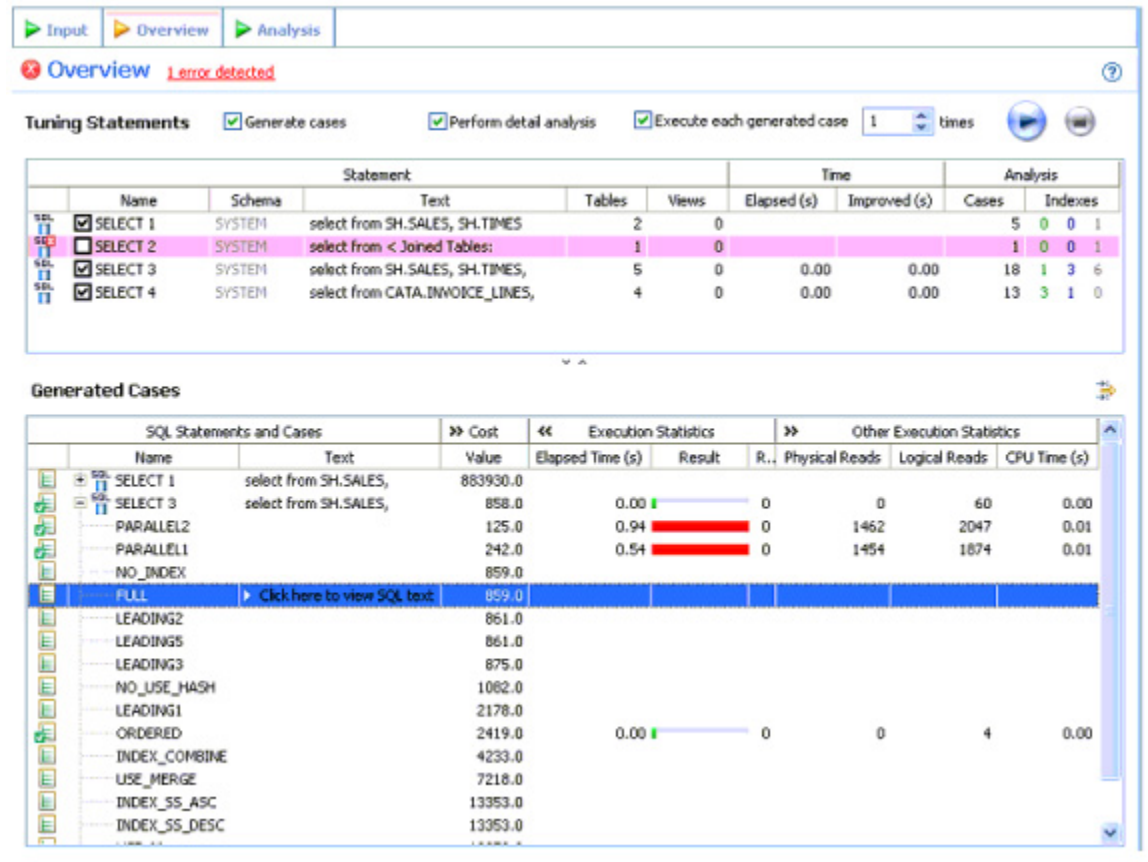
Context menu commands in SQL Profiler enable you to import statements to a tuning job directly from the Profiler interface.

To Import a statement from Profiler into Tuner

Select one or more statements in Profiler, right-click and select **Tune** from the context menu. SQL Tuner opens and contains the selected statements in a new tuning job. You can now proceed to tune the problematic statements.

TUNING SQL STATEMENTS

SQL Tuner provides an easy and optimal way to discover efficient paths for queries that may not be performing as quickly or as efficiently as they could be. Tuner enables the optimization of poorly-performing SQL code through the detection and modification of execution paths used in data retrieval. This is primarily performed through hint injection, and index and statistics analysis.



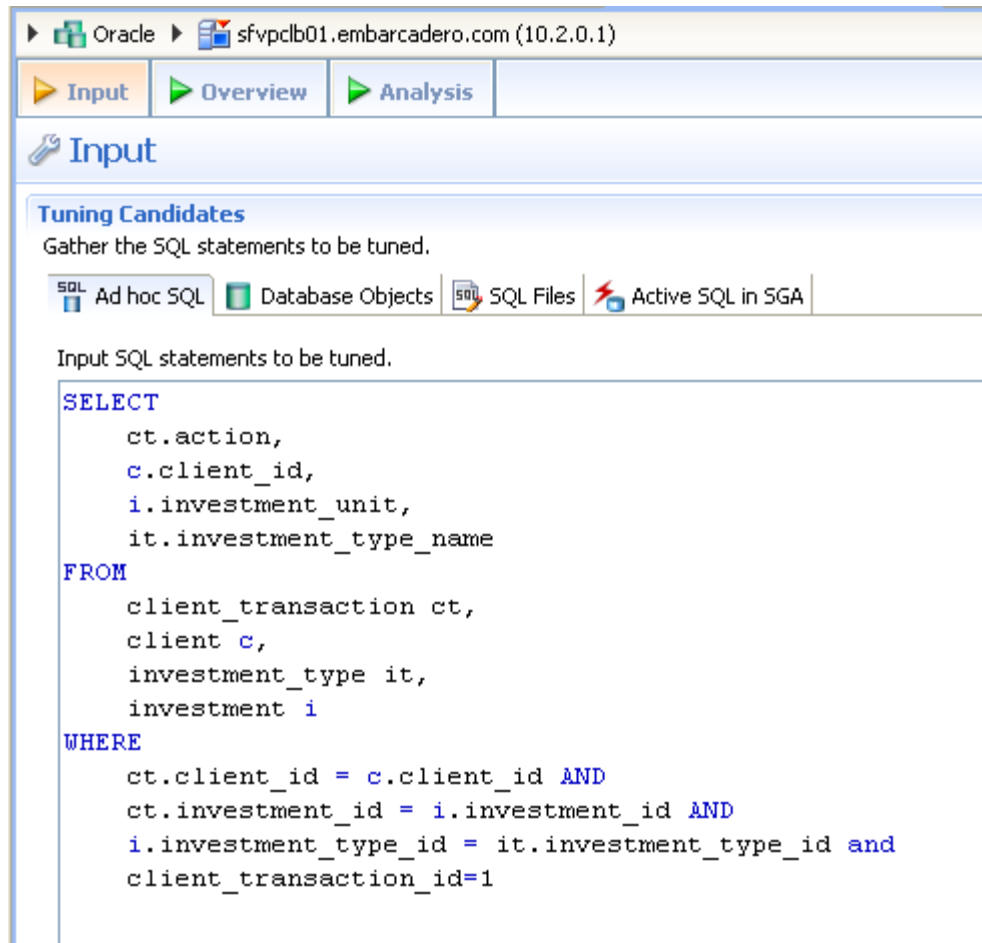
Tuner analyzes specified SQL statements and then supplies execution path directives. This enables you to select alternate paths for queries, thus optimizing system performance based on analysis.

For example, if tuning is selecting from two tables (A and B), it will enable the joining of A to B, or B to A as well as the join form. Additionally, different joining methods such as nested loops or hash joins can be used and will be tested, as appropriate. Tuning will select alternate paths, and enable you to change the original path to one of the alternates. Execution paths slower than the original are eliminated, which enables you to select the quickest of the returned selections and improve query times, overall.

This enables the DBA to correctly optimize queries in the cases where the native optimizer failed.

CREATING A NEW TUNING JOB

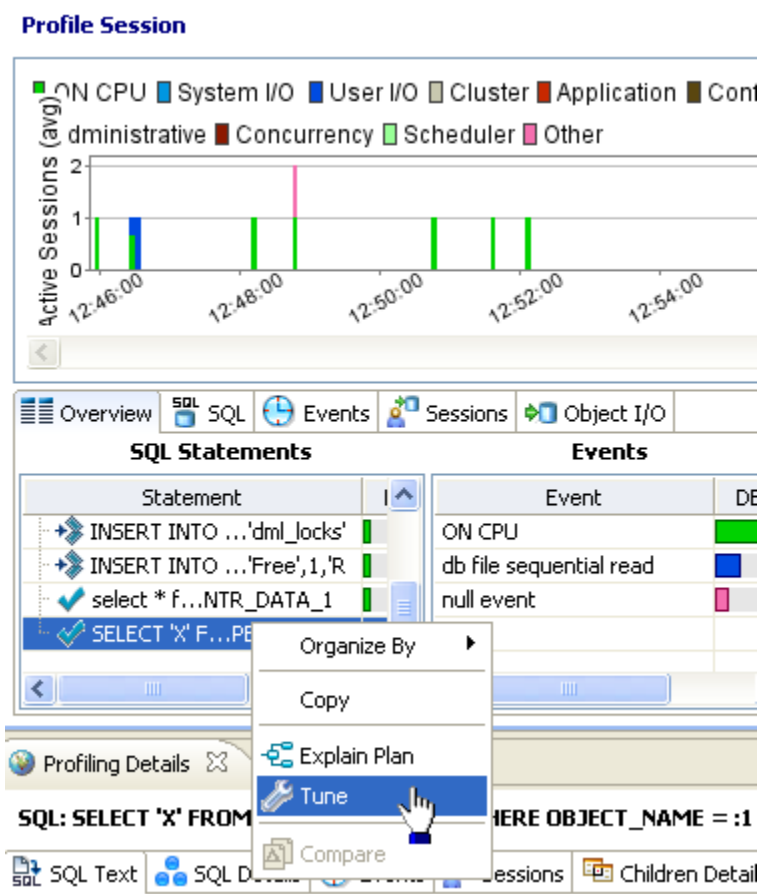
New tuning jobs are created from scratch where you can specify the statements to be tuned from a variety of sources, or statements can be directly imported from existing profiling sessions on data sources currently registered in the environment.



Tuning jobs are defined in SQL Tuner by specifying the data source and corresponding statements to be tuned and then executing the tuning job process.

To create a new tuning job:

If you determined from a profiling session that a specific SQL statement should be tuned, in the Profile Session, right-click the statement and select **Tune** as follows.



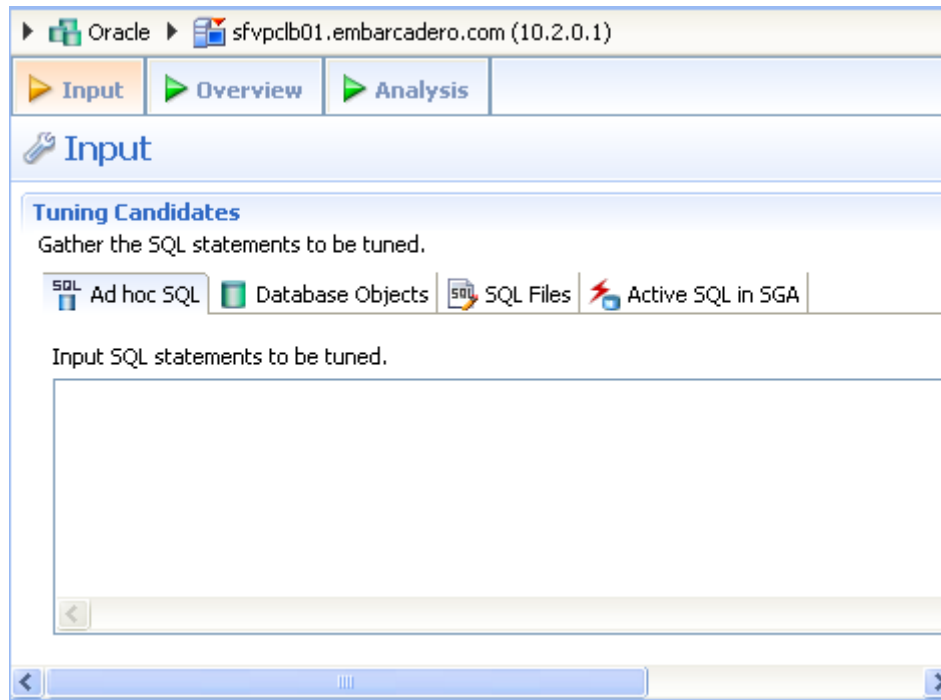
OR

Select **File > New > Tuning Job**, or click the New Tuning Job icon on the Toolbar. SQL Tuner opens and you can proceed to set up the parameters of the new job.

Once you have defined the input of the tuning job, you can save the file with a `.tun` suffix via the **Save As...** command. The job is added to the SQL Project Explorer and can be re-opened and re-run at any time once it is saved to the system.

ADDING SQL STATEMENTS

Once you have created a name for the tuning job and indicated its source, you can add SQL statements that you want the job to tune. Statements are added to a job via the Ad hoc SQL box. All standard DML statements (SELECT, INSERT, DELETE, UPDATE) are viable for the tuning procedure.



Statements are added to a tuning job via the Input tab.

There are three or four different ways to add SQL statements to a job, as reflected by the tabs in the Tuning Candidates box:

Use the **Input** tab to specify which SQL statements to tune.

- **Ad hoc SQL:** Copy/paste SQL statements to the Ad hoc SQL tab or write queries by hand.
- **Database Objects:** Drag and drop database objects from the Data Source Explorer to the Database Objects tab.
- **SQL Files:** Browse the workspace or file system and select SQL files.
- **Active SQL in SGA:** For the Oracle platform only, you can also scan the System Global Area (SGA) for statements to tune.

To add an ad hoc statement:

Select the Ad Hoc SQL tab and manually type an SQL statement in the window. Alternatively, copy/paste the statement from another source.

To add a database object:

Select the Database Objects tab and click Add. The Data Source Objects Selection dialog appears. Type an object name prefix or pattern in the field provided and choose a statement from the window as it populates to match what you typed.

To add a saved SQL file:

Select the SQL Files tab and click Workspace or File System, depending on where the file you want to add is stored. Select a file from the dialog that appears and it will be added to the job.

To add Active SQL in SGA:

Select the Active SQL in SGA tab and click Scan. Specify any filters as required and then click Next. From the list of SQL statements retrieved from the SGA, select those you want to optimize, and then click Finish. The selected statements are copied to the text area of the Active SQL in SGA tab.

RUNNING A TUNING JOB

After you add SQL statements to the job, click the Overview tab. Once you choose your tuning options and click the Run Job icon, the DML is parsed from the statements and added to the Generated Cases area. The Generated Cases are alternative execution paths or explain paths that could be better or worse than the default path the database uses. When these cases are executed you can use the execution statistics to determine which case would optimize performance.

Each extracted statement is listed by Name and Text. Additionally, each statement has a Cost, Elapsed Time, and Other Execution Statistics value that provide information on how effectively each case executes on the specified data source. These parameters let you compare the efficiency of the original statements to the cases generated by the tuning process when it is executed.

TIP: You can click the Text field of a generated case to view the SQL source of the statement.

The screenshot displays the DB Optimizer interface with the 'Overview' tab selected. The 'Tuning Source Status' section shows a table of statements with columns for Name, Schema, Text, Tables, Views, Elapsed (s), Improved (s), Cases, and Index. Two statements are listed: 'SELECT 2' and 'SELECT 1'. The 'Generated Cases' section shows a table of SQL statements and cases with columns for Name, Text, Cost, Elapsed Time (s), Physical Reads, and Logical Reads. The table includes a list of generated cases such as 'USE_HASH', 'ORDERED', 'NO_USE_NL', 'LEADING4', 'LEADING3', 'LEADING2', 'LEADING1', 'INDEX_FFS', 'FULL', and 'FIRST_ROWS'.

Annotations in the image include:

- Tuning Status Indicator:** Points to the 'Tuning Source Status' tab.
- Enable Execution Check Box:** Points to the 'Execute each generated case' checkbox.
- Increase/Decrease Pane Size Control:** Points to the vertical scrollbar on the right side of the 'Generated Cases' table.
- Column Set Expand/Collapse Control:** Points to the expand/collapse icons on the right side of the 'Generated Cases' table.
- Run/Cancel Job Controls:** Points to the 'Run' and 'Cancel' buttons in the 'Tuning Source Status' section.
- Generated Base Expand/Collapse Control:** Points to the expand/collapse icons on the left side of the 'Generated Cases' table.
- Filter Control:** Points to the filter icon on the right side of the 'Generated Cases' table.
- Transformation Case:** Points to the 'Transformation Case' row in the 'Generated Cases' table.
- Hint-Based Cases:** Points to the 'Hint-Based Cases' row in the 'Generated Cases' table.

The Generated Cases tab enables you to measure the various load costs of the original tuning statements and generated cases for each, which suggest alternative query paths to optimizing your data source.

The Tuning Status Indicator provides the status of each statement or case, and indicates if they are ready for execution. In some cases, SQL code may need to be corrected or bind variables may need to be set prior to executing statements. When you try to tune a statement containing a bind variable you are now warned that either the type is not set or the value is not set.

Use the check boxes to select which statements and cases you want to run and then click the Run icon in the lower right-hand corner of the screen. The Execute each generated case field enables you to execute each selected statement or case.

Once you have executed a tuning job, the Generated Cases tab will reflect SQL Tuner's analysis of the specified statements. Once these have been analyzed, you can proceed to modifying the Tuner results and applying specified cases on the data source to optimize its performance.

To execute a tuning job:

- 1 Once you have a SQL statement that is a tuning candidate, navigate to the Overview tab.
- 2 In the Tuning Statements area, select the checkbox next to the Statement name that you want to analyze and then
- 3 *To analyze the SQL statements*, click Generate cases.

To perform the analysis that populates the Analysis tab now, click Perform detail analysis. Otherwise, this analysis tab is performed when you click the Analysis tab.

To have the system generate execution statistics, click Execute each generated case and then select the number of times the system should execute each generated case. Multiple executions can verify that the case results are not skewed by caching. For example, the first time a query is run, data might be read off of disk, which is slow, and the second time the data might be in cache and run faster. Thus, one case might seem faster than another but it could be just benefiting from the effects of caching. Generally, you only need to execute the cases once, but it may be beneficial to execute the cases multiple times to see if the response times and statistics stay the same.

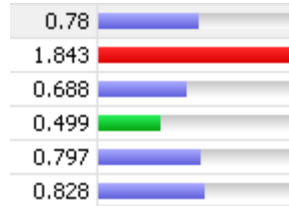
- 4 Then click the Run Job icon at the top right-hand side of the window. The tuning job runs, analyzing each statement and case, and providing values in the appropriate columns.

ANALYZING TUNER RESULTS ON THE OVERVIEW TAB

When you have executed a tuning job, the Generated Cases area of the Overview tab reflects Tuner's analysis of the specified statements and cases. The Generated cases create alternative execution or explain paths that could be more or less efficient than the default path the databases uses. Executing these cases provides the statistics necessary to optimize performance.

Once a tuning job has executed, use the Cost and Execution Statistics value columns to determine the fastest execution path for each statement. The Cost column shows the performance cost of an execution path as determined by the database. The Execution Statistics are the actual results of running the SQL statement using the generated case. This is where DB Optimizer can help you find where the database default path is actually not the optimal path. The Elapsed Time(s) and Results columns can more accurately show the most efficient execution path.

In the Cost and Execution Statistics columns, the values of the original statement are considered to be the baseline values. A Cost column can be expanded to provide a graphical representation of the values for statements and cases. Similarly, the Execution Statistics column can also be expanded to display a graphical representation of values as well. The bar length and colors are intended as an aid in comparing values, particularly among cases.



Case query times based on the original statement can be represented as colored bars on the Generated Cases area of the Overview tab to help you determine the fastest execution path for the given selections.

The baseline value of the original statement spans half the width of the column, in terms of bar length. For cases of the original statement, if one or more cases show a degradation value, the largest value will span the width of the column. Bar lengths for all other cases will then be displayed in comparison length to the highest degradation value.

The cost and execution results are color-coded as follows:

- **Light blue:** These cases are within the degradation and improvement threshold. Applying these changes may marginally improve or degrade the efficiency of the SQL statement.
- **Green:** These cases have values less than the improvement threshold. There is a high probability that changing the SQL statement with this alternative execution path will improve efficiency.
- **Red:** These cases have values higher than the improvement threshold. Implementing these changes will degrade the efficiency of the SQL statement.

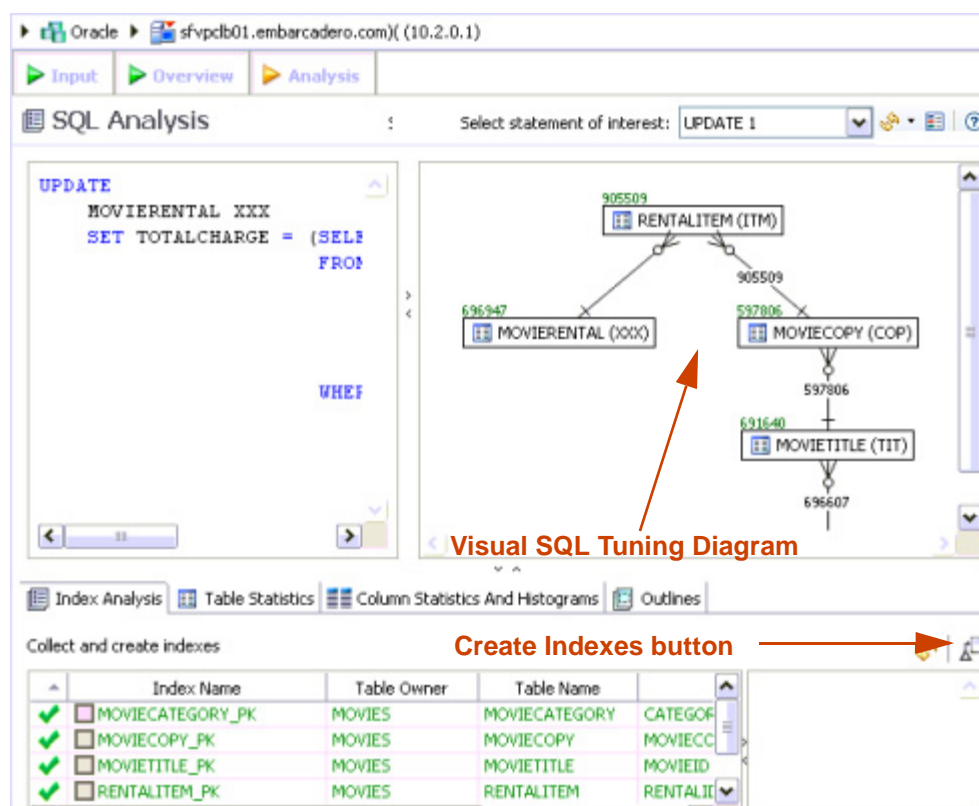
To determine the best cases for statement execution path time:

Once the tuning job has executed, view the Generated Cases area of the Overview tab and determine the best possible case in terms of the Execution Statistics column values. This will indicate the most optimized query path for a given statement. Once you have determined the best case, you can execute that case on the specified data source and alter the database code to run the statement as that case on the native environment.

If you don't find an acceptably fast path, go to the Analysis tab. The Analysis tab can identify missing indexes and by examining the diagram you may be able to determine if there is something wrong with the SQL or schema.

FINDING MISSING INDEXES AND SQL PROBLEMS

The tuner performs the index and SQL analysis as part of the tuner run job performed on the Overview tab if Perform Detail Analysis is selected. Otherwise, the analysis is performed when you click the Analysis tab.



DB Optimizer can parse an SQL query and analyze the indexes and constraints on the tables in the query and display the query in graphical format on the Visual SQL Tuning (VST) diagram. The VST diagram which can be displayed in either Summary Mode or Detail Mode, helps developers, designers and DBAs see flaws in the schema design such as Cartesian joins, implied Cartesian joins and many-to-many relationships. The VST diagram also helps the user to more quickly understand the components of an SQL query, thus accelerating troubleshooting and analysis.

This section is comprised of the following topics:

- [Finding Missing Indexes](#) on page 223
- [Applying Tuner Results to the Data Source](#) on page 223

FINDING MISSING INDEXES

Missing indexes are color-coded **orange** in the Collect and create indexes area of the Overview tab. Creating a missing index can improve the execution path of the SQL statement being analyzed.

TIP: Indexes that are used are color-coded **green**. Indexes that are present but not used in this execution path are color-coded **grey**.

The screenshot displays the SQL Analysis tool interface. The top navigation bar includes 'Input', 'Overview', and 'Analysis' tabs. The 'Analysis' tab is active, showing a SQL query in the left pane and its execution plan in the right pane. Below the query pane is the 'Index Analysis' tab, which contains a table titled 'Collect and create indexes'.

SQL Query:

```
SELECT
  ct.action,
  c.client_id,
  i.investment_unit,
  it.investment_type_name
FROM
  client_transaction ct,
  client c,
  investment_type it,
  investment i
WHERE
  ct.client_id = c.client
  ct.investment_id = i.in
  i.investment_type_id =
```

Execution Plan: The plan shows a join between CLIENT_TRANSACTION (ct) and CLIENT (c), and another join between INVESTMENT (i) and INVESTMENT_TYPE (it).

Index Analysis Table:

Index Name	Table Owner	Table Name	Index Name
✓ INVESTMENT_PK	SYSTEM	INVESTMENT	INVESTMEI

APPLYING TUNER RESULTS TO THE DATA SOURCE

Once Tuner has generated cases and statement results and analyzed the indexes, you can apply the suggested changes to the data source from the Generated Cases area of the Overview tab. You can create any recommended indexes from the Index Analysis area of the Analysis tab.

IMPLEMENTING RECOMMENDATIONS ON THE OVERVIEW TAB

To change an SQL statement based on a transformation or hint-based case:

- 1 In the **Generated Cases** area, right-click the **Name** field of the case you want to modify the original statement with and choose **Apply Change**.

The **Apply Change** dialog appears.

- 2 Choose **Execute** to apply the change to the statement automatically.

TIP: Alternatively, you can select Open in New SQL Editor to make manual changes or save it to a file.

SQL Statements and Cases >>		Cost >>	Elapsed Time >>	Other Execution Statistics		
	Name	Value	Value (s)	Physical Reads	Logical Reads	Consistent Gets
<input checked="" type="checkbox"/>	SQL Statement 1					
<input checked="" type="checkbox"/>	[Null column comparison]					
<input checked="" type="checkbox"/>	SQL Statement 2					
<input checked="" type="checkbox"/>	SQL Statement 3					
<input checked="" type="checkbox"/>	SQL Statement 4					
<input checked="" type="checkbox"/>	SQL Statement 5					
<input checked="" type="checkbox"/>	SQL Statement 11					
<input checked="" type="checkbox"/>	SQL Statement 13					
<input checked="" type="checkbox"/>	SQL Statement 14	2.0	0	0	3	3
<input checked="" type="checkbox"/>	ALL_ROWS	2.0	0.829	1	3	3
<input checked="" type="checkbox"/>	FIRST_ROWS	2.0				
<input checked="" type="checkbox"/>	NO PARALLEL	2.0				

Cases can be selected and applied on the data source where the statement originated. This process improves the former statement's execution path and therefore lessens overall data source load.

IMPLEMENTING RECOMMENDATIONS ON THE INDEX ANALYSIS TAB

Once you have added tuning candidates to a tuning job, DB Optimizer can analyze the effectiveness of the indexes in the database and recommend the creation of new indexes where the new indexes can increase performance.

In the Collect and create indexes table, any indexes DB Optimizer recommends you create are marked in orange and have the little create index.

Index Analysis Table Statistics Column Statistics And Histograms Outlines					
Collect and create indexes					
	Index Name	Table Owner	Table Name	Column Name	Index
<input checked="" type="checkbox"/>	IDX_CLIENT_TRANSACTION_0	SYSTEM	CLIENT_TRANSACTION	TRANSACTION_STATUS	Normal
<input type="checkbox"/>	CLIENT_MULTI	SYSTEM	CLIENT	CLIENT_FIRST...NT_LAST_NAME	Normal
<input type="checkbox"/>	CLIENT_BROKER	SYSTEM	CLIENT	BROKER_ID	Normal
<input type="checkbox"/>	CLIENT_INCOME	SYSTEM	CLIENT	CLIENT_HOUSEHOLD_INCOME	Normal
<input type="checkbox"/>	CLIENT_PK	SYSTEM	CLIENT	CLIENT_ID	Unique
<input type="checkbox"/>	CLIENT_TRANSACTION_BROKER	SYSTEM	CLIENT_TRANSACTION	BROKER_ID	Normal
<input type="checkbox"/>	CLIENT_TRANSACTION_CLIENT	SYSTEM	CLIENT_TRANSACTION	CLIENT_ID	Normal

To accept the suggestion and have tuning automatically generate an index:

- 1 For any recommended index, click the checkbox to the left of the index you want to create.

For a selected index you can modify the Index type by clicking in the **Index Type** column and then selecting a type from the list: **Normal**, **Bitmap**, **Reverse Key**, **Reverse Key Unique**, or **Unique**.

Index Analysis Table Statistics Column Statistics And Histograms Outlines					
Collect and create indexes					
	Index Name	Table Owner	Table Name	Column Name	Index
<input checked="" type="checkbox"/>	IDX_CLIENT_TRANSACTION_0	SYSTEM	CLIENT_TRANSACTION	TRANSACTION_STATUS	Normal
<input type="checkbox"/>	CLIENT_MULTI	SYSTEM	CLIENT	CLIENT_FIRST...NT_LAST_NAME	Normal
<input type="checkbox"/>	CLIENT_BROKER	SYSTEM	CLIENT	BROKER_ID	Normal
<input type="checkbox"/>	CLIENT_INCOME	SYSTEM	CLIENT	CLIENT_HOUSEHOLD_INCOME	Normal
<input type="checkbox"/>	CLIENT_PK	SYSTEM	CLIENT	CLIENT_ID	Unique
<input type="checkbox"/>	CLIENT_TRANSACTION_BROKER	SYSTEM	CLIENT_TRANSACTION	BROKER_ID	Normal
<input type="checkbox"/>	CLIENT_TRANSACTION_CLIENT	SYSTEM	CLIENT_TRANSACTION	CLIENT_ID	Normal

Table SYSTEM.CLIENT_TRANSACTION is scanned via full table scan but it has a filter `ct.transaction_status = c.client_marital_status` on it and we created a virtual index `IDX_CLIENT_TRANSACTION_0` which the optimizer picked up, so we suggest implementing this index.

Create Index

- 2 Click the **Create Indexes** button.

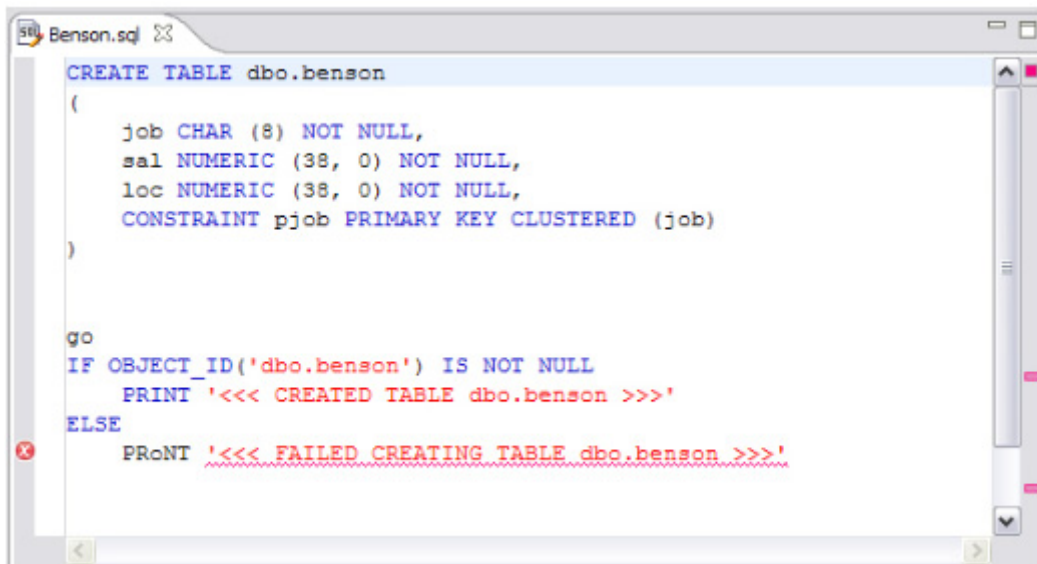
The **Index Analysis** dialog appears.

- 3 To view the index SQL in an editor for later implementation, click the statement and then click **Open in a SQL editor**.
- 4 To run the index SQL and create the index on the selected database, click **Execute**.

SQL CODE ASSIST AND EXECUTION

SQL Code Assist is an interface component that enables the development and formatting of SQL code for the purposes of creating and modifying database objects. It provides a front-end application for the delivery of code through its object code extraction capabilities.

Code Assist provides a number of key features that assist in the coding process, ensuring greater accuracy in coding, faster development cycles, and a general increase in efficiency overall.



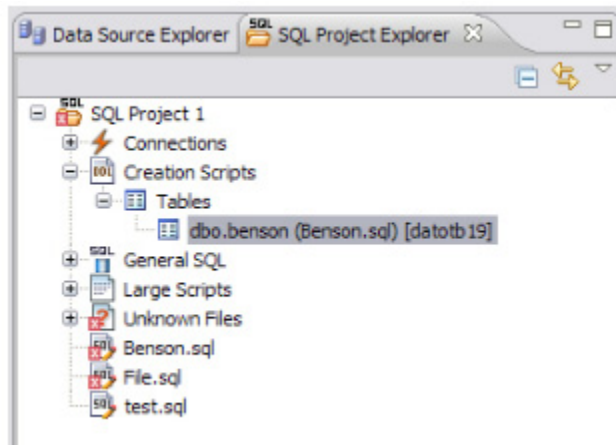
SQL Editor provides key features to ensure an increase in code accuracy and overall development efficiency.

The following key features are provided with SQL Code Assist:

- Code Extraction
- Code Highlighting
- Automatic Error Detection
- Code Complete
- Hyperlinks
- Code Formatting
- Code Folding
- Code Quality Checks

In order to access SQL Editor, you need to create a new file or edit existing code from Data Source Explorer.

Additionally, SQL Project Explorer provides a tree structure for all files created in DB Optimizer. You can access files from here by double-clicking the file name you want to open as well.



SQL Project Explorer provides a tree of all files developed and saved in DB Optimizer.

To create a new file:

Choose **File > New > SQL File**.

A blank instance of SQL Editor appears in the Workbench. If you save this file, it is automatically added to the SQL Project Explorer.

To edit an existing file:

Use Data Source Explorer or Project Explorer to navigate to the code you want to modify and double-click it.

An instance of SQL Editor appears in the Workbench, populated with the extracted code of the specified object or SQL project file.

CODE EXTRACTION

SQL Editor provides the ability to extract the underlying SQL code of database objects registered in DB Optimizer to provide a front-end application for the development and modification of data sources in your enterprise.

To extract underlying SQL code:

- Navigate to a database object in Data Source Explorer and select Extract via the right-click menu.

The object's underlying SQL code appears in SQL Editor and is ready for editing and further modification.

CODE HIGHLIGHTING

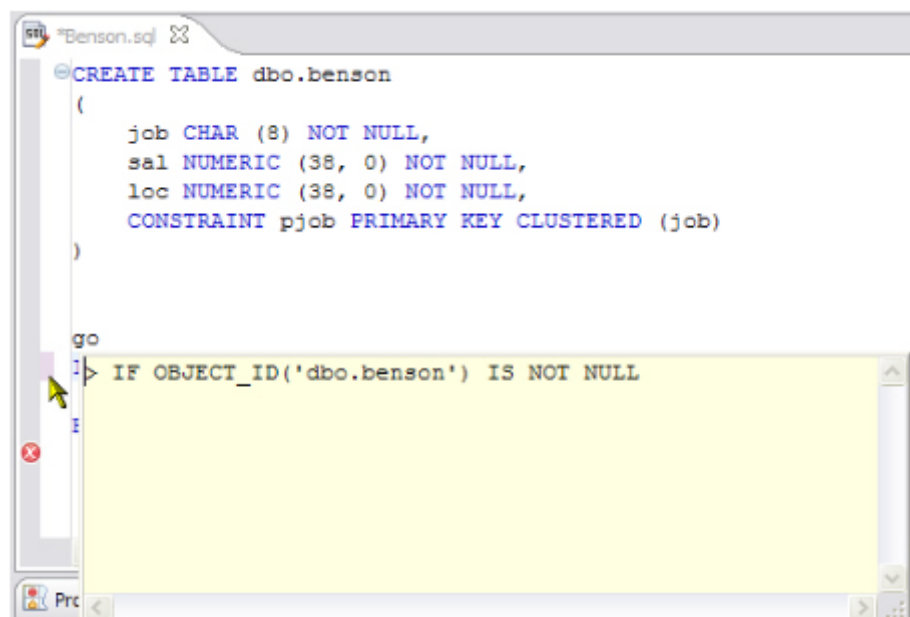
SQL Editor identifies commands and provides syntax highlighting changes that are automatically added to the code as you add lines, which enables you to clearly and quickly understand code when you read it in the interface.

The following syntax highlighting is automatically added to lines of code in SQL Editor:

Code	Formatting
Comments	Green italic font
SQL Commands	Dark blue font
Syntax Errors	Red underlined font
Coding Errors	Red font

- Comments: green italics
- SQL Commands: dark blue
- Syntax Errors: red underline
- Coding Errors: red

Additionally, SQL Editor provides a purple change bar in the left-hand column that indicates if a line of code has been modified from the original text. You can hover over this change bar to view the original code line. A red square icon in the right-hand column indicates that there are errors in the code line. You can hover over the icon to view the error count.

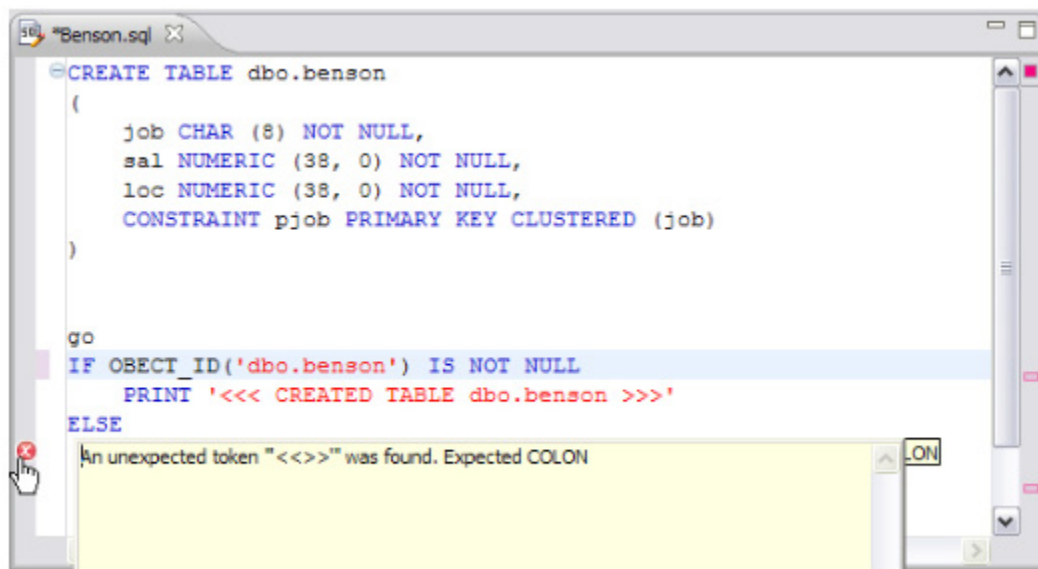


The purple change bar indicates if a line of code has been changed from its original text. Hover your mouse over the change bar to view the original text.

AUTOMATIC ERROR DETECTION

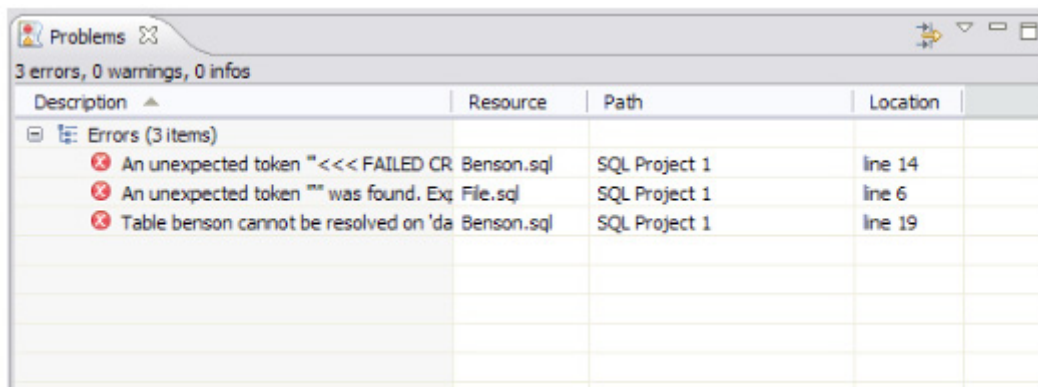
The automatic error detection functionality of the editor highlights errors and typos in the code as you work in “real time”.

Automatic error detection automatically identifies and analyzes SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY statements. If it detects any syntax errors while you type these statements, the line is automatically flagged by the error icon in the left-hand column of SQL Editor. You can hover your mouse over the icon to view any errors.



Syntax errors are automatically flagged by line as you work with code in SQL Editor. Hover the mouse over the error icon to view the specific error message.

Additionally, all semantic errors are recorded in the Problems view, an interface component that automatically logs errors and warnings as you work with files.



The Problems view logs errors and warnings as you work with files in SQL Editor.

You can double-click on a line in the Problems view and DB Optimizer will automatically navigate you to that issue in SQL Editor.

CODE COMPLETE

SQL Editor provides suggestion capabilities for both DML statements and objects. It has the ability to look up object names in order to avoid errors when defining table names, columns, etc. in the development process. This feature provides lists of object and code suggestions that will make the development process more efficient as you will not be forced to manually look up object names and other statement values. SQL Editor provides code assist for SELECT, UPDATE, INSERT, and DELETE statements and object suggestion support for tables, alias tables, columns, alias columns, schemas, and catalogs.

To activate code assist:

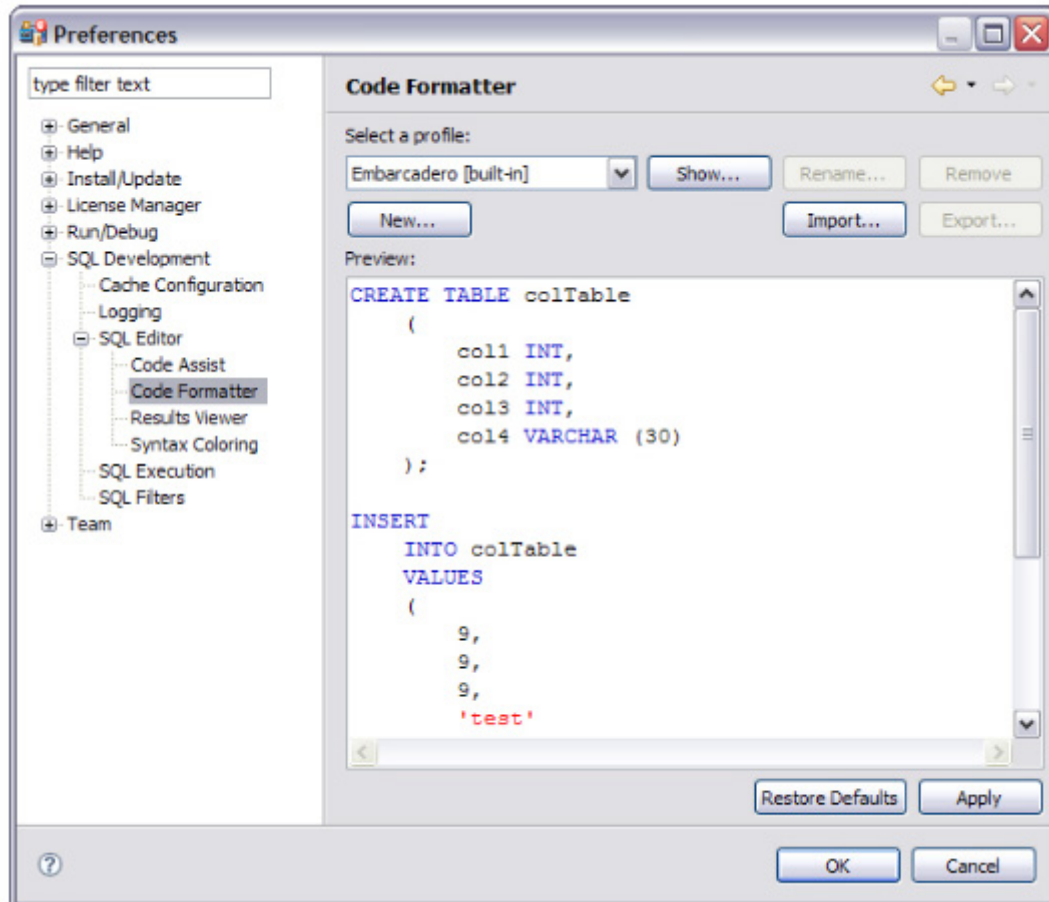
- 1 Click the line on which you want to activate the code assist feature.
- 2 Press **CTRL + Spacebar** on your keyboard. Code assist analyzes the line and presents a list of suggestions as appropriate based on the elements of the statement.

HYPERLINKS

Hyperlinks are used in SQL Editor to provide links to tables, columns, packages and other reference objects. When you select a hyperlinked object from a piece of code, a new editor opens and displays the source. Additionally, hyperlinks can be used to link procedures or the function of a call statement, as well as function calls in DML statements. To enable a hyperlink, hover your mouse over the object name and hold the CTRL key. It becomes underlined and changes color.

CODE FORMATTING

Code formatting is automatically applied to a file as you develop it in SQL code. This enables you to set global formatting preferences one time, and then apply it to all code development, saving time and allowing for a more efficient code development process. The code formatter can be accessed by selecting **CTRL + Shift + F** in the editor. All code is automatically formatted based on parameters specified on the Code Formatter node of the Preferences panel. (Select Window > Preferences in the Main Menu to access this panel.)



Code formatting parameters can be globally set and then applied to all development work in SQL Editor.

In addition to formatting code per individual file, you can also format an entire group of files from Project Explorer. Select the directory of files that you want to apply formatting to and execute the **Format** command from the right-click menu. The files will be automatically formatted based on the global preferences.

CODE FOLDING

The code folding feature automatically sorts code into a tree-like outline structure in SQL Editor. This increases navigation and clarification capacities during the code development process. It ensures that the file will be easily understood should any future work be required on the code. As you work in SQL Editor, collapsible nodes are automatically inserted into the appropriate lines of code. Statements can then be expanded or collapsed as needed, and this feature is especially useful when working on parts of particularly large or complicated files.

CODE QUALITY CHECKS

On Oracle-based code, the code quality checking feature provides suggestions regarding improved code on a statement-by-statement basis. As you work in SQL Editor, markers provides annotations that prevent and fix common mistakes in the code.

Notes regarding code quality suggestions appear in a window on any line of code where the editor detects an error, or otherwise detects that the code may not be as efficient as it might be. Code quality check annotations are activated by clicking the light bulb icon in the margin, or by selecting **Ctrl + I** on your keyboard.

The following common errors are detected by the code quality check function in the editor:

- Statement is missing valid JOIN criteria
- Invalid or missing outer join operator
- Transitivity issues
- Nested query in WHERE clause
- Wrong place for conditions in a HAVING clause
- Index suppressed by a function or an arithmetic operator
- Mismatched or incompatible column types
- Null column comparison

To activate code quality checks:

- Click the light bulb icon in the margin of the editor or select **Ctrl + I** on your keyboard.

The editor suggestions appear in a window beneath the selected statement. When you click a suggested amendment, the affected code is automatically updated.

SQL EXECUTION

When you have finished developing or modifying code, you can then execute the file from within the DB Optimizer environment, on the database of your choosing. This enables you to immediately execute code upon completion of its development. Alternatively, you can save files for execution at a later point in time.

In order to execute a file, you must first associate it with a target database. This is performed by using the drop-down menus located in the Toolbar. When a SQL file is open in the Workbench, the menus are enabled. Select a data source and a corresponding database to associate the file with and then click the green arrow icon to execute the file.



The pair of drop-down menus indicate that the SQL file is associated with the dataotb19 data source and EMBCM database. When the green arrow icon on the right-hand side of the menus is selected, the file is executed on the specified data source and database.

Additionally, if you have turned off auto-committal in the Preferences panel (**Window > Preferences**) you can commit and execute transactions via the Commit Transaction and Start Transaction icons located beside the Execute icon.

To execute a file:

Open the file you want to run and ensure it is associated with the correct database, then click the Execute icon. DB Optimizer executes the code on the database you specified.

To execute a transaction:

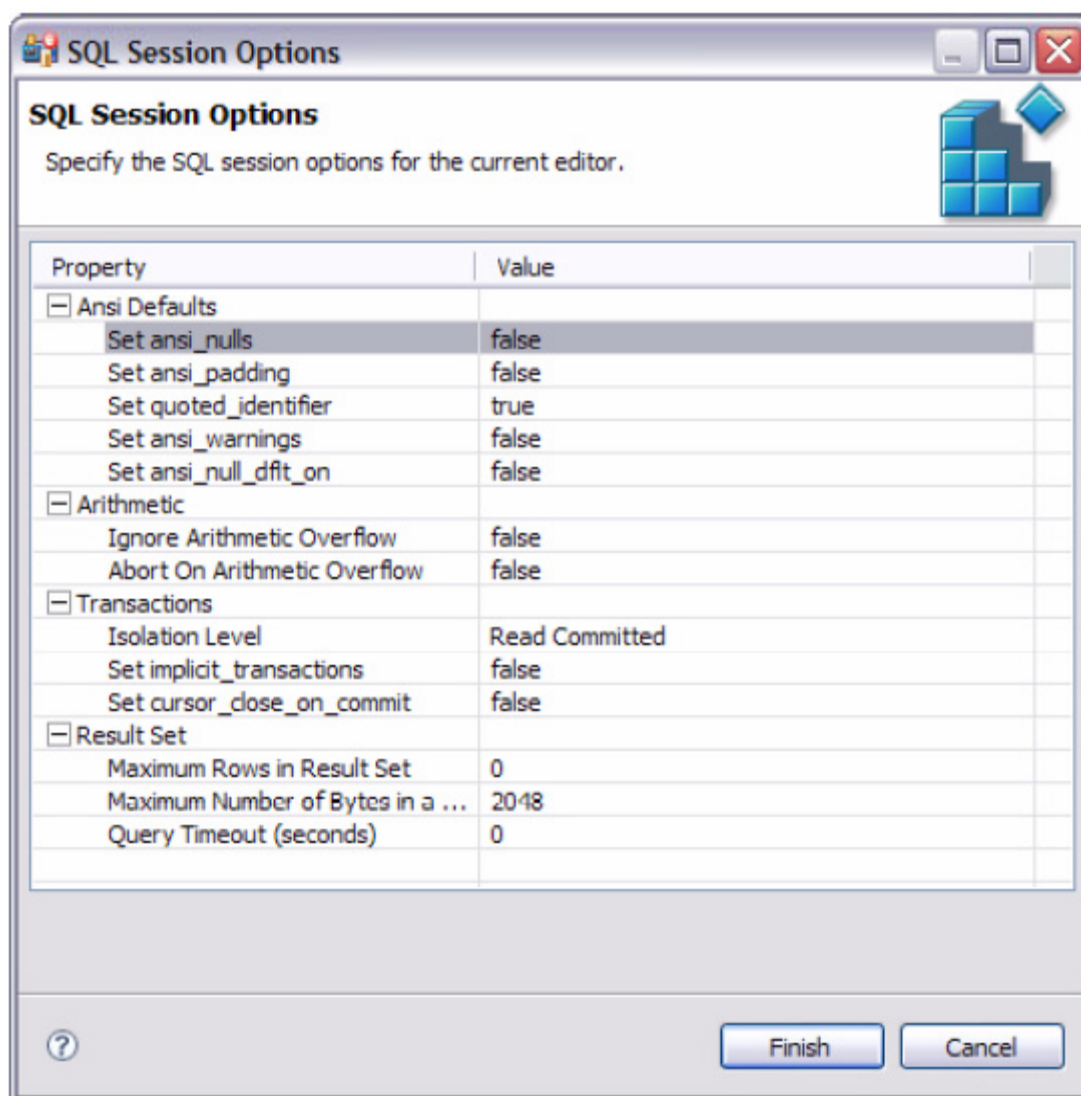
Open the transaction file you want to run and ensure it is associated with the correct database, then click the Start Transaction icon. DB Optimizer executes the transaction on the database you specified.

To commit a transaction:

Open the transaction file you want to commit, ensure it is associated with the correct database, and click the Commit Transaction icon. DB Optimizer commits the transaction on the database you specified.

CONFIGURING SQL EXECUTION PARAMETERS

If you do not want to use the default execution options provided by DB Optimizer, use the SQL Session Options dialog to modify the configuration parameters that determine how DB Optimizer executes code. These options ensure that code is executed the way you want on an execution-per-execution basis, ensuring accuracy and flexibility when running new or modified code.



SQL Sessions Options provide you with the flexibility to adjust execution parameters on a session-by-session basis.

To modify SQL session options:

- 1 Click the SQL Session Options icon on the toolbar. The **SQL Sessions Options** dialog appears.
- 2 Click on the individual parameters in the **Value** column to change the configuration of each property, as specified.

- 3 Click **Finish**. The session options are changed and DB Optimizer executes the code as specified by your options.

NOTE: SQL Session Options are only applied to the currently-selected code, and are not retained across different files with regards to execution.

REFERENCE

The following topics provide reference details:

- [Database Objects](#)
- [DBMS Connection Parameters by Platform](#)

DATABASE OBJECTS

The following table describes the database objects displayed in DB Optimizer™ XE and DB Optimizer™ and contains information regarding each one, including object name, DBMS platform, and any notes pertaining to the specified object.

In DB Optimizer™ XE and DB Optimizer™, database objects are stored in Data Source Explorer as subnodes of individual, pertinent databases.

Database Object	DBMS Platforms	Notes
Aliases	DB2	<p>An alias is an alternate name that references a table, view, and other database objects. An alias can also reference another alias as long as the aliases do not reference one another in a circular or repetitive manner.</p> <p>Aliases are used in view or trigger definitions in any SQL statements except for table check-constraint definitions. (The table or view name must be referenced in these cases.)</p> <p>Once defined, an alias is used in query and development statements to provide greater control when specifying the referenced object. Aliases can be defined for objects that do not exist, but the referenced object must exist when a statement containing the alias is compiled.</p> <p>Aliases can be specified for tables, views, existing aliases, or other objects. Create Alias is a command available on the shortcut menu.</p>
Check Constraints	All	<p>A check constraint is a search condition applied to a table. When a check constraint is in place, Insert and Update statements issued against the table will only complete if the statements pass the constraint rules.</p> <p>Check constraints are used to enforce data integrity when it cannot be defined by key uniqueness or referential integrity restraints.</p> <p>A check condition is a logical expression that defines valid data values for a column.</p>

Database Object	DBMS Platforms	Notes
Clusters	Oracle	<p>A cluster is a collection of interconnected, physical machines used as a single resource for failover, scalability, and availability purposes.</p> <p>Individual machines in the cluster maintain a physical host name, but a cluster host name must be specified to define the collective as a whole.</p> <p>To create a cluster, you need the CREATE CLUSTER or CREATE ANY CLUSTER system privilege.</p>
Database Links	Oracle	<p>A database link is a network path stored locally, that provides the database with the ability to communicate with a remote database.</p> <p>A database link is composed of the name of the remote database, a communication path to the database, and a user ID and password (if required).</p> <p>Database links cannot be edited or altered. To make changes, drop and re-create.</p>
Foreign Keys	All	<p>A foreign key references a primary or unique key of a table (the same table the foreign key is defined on, or another table and is created as a result of an established relationship). Its purpose is to indicate that referential integrity is maintained according to the constraints.</p> <p>The number of columns in a foreign key must be equal to the number of columns in the corresponding primary or unique key. Additionally, the column definitions of the foreign key must have the same data types and lengths.</p> <p>Foreign key names are automatically assigned if one is not specified.</p>
Functions	DB2, Oracle	<p>A function is a relationship between a set of input data values and a set of result values.</p> <p>For example, the TIMESTAMP function passes input data values of type DATE and TIME, and the result is TIMESTAMP.</p> <p>Functions can be built-in or user-defined. Built-in functions are provided with the database. They return a single value and are part of the default database schema. User-defined functions extend the capabilities of the database system by adding function definitions (provided by users or third-party vendors) that can be applied in the database engine itself.</p> <p>A function is identified by its schema, a function name, the number of parameters, and the data types of its parameters.</p> <p>Access to functions is controlled through the EXECUTE privilege. GRANT and REVOKE statements are used to specify who can or cannot execute a specific function or set of functions.</p>
Groups	All	<p>Groups are units that contain items. Typically, groups contain the result of a single business transaction where several items are involved.</p> <p>For example, a group is the set of articles bought by a customer during a visit to the supermarket.</p>

Database Object	DBMS Platforms	Notes
Indexes	All	<p>An index is an ordered set of pointers to rows in a base table.</p> <p>Each index is based on the values of data in one or more table columns. An index is an object that is separate from the data in the table. When an index is created, the database builds and maintains it automatically.</p> <p>Indexes are used to improve performance. In most cases, access to data is faster with an index. Although an index cannot be created for a view, an index created for the table on which a view is based can improve the performance of operations on that view.</p> <p>Indexes are also used to ensure uniqueness. A table with a unique index cannot have rows with identical keys.</p> <p>DB2: Allow Reverse Scans, Percent Free (Lets you type or select the percentage of each index page to leave as free space when building the index, from 0 to 99), Min Pct Used (Lets you type or select the minimum percentage of space used on an index leaf page. If, after a key is removed from an index leaf page, the percentage of space used on the page is at or below integer percent, an attempt is made to merge the remaining keys on this page with those of a neighboring page. If there is sufficient space on one of these pages, the merge is performed and one of the pages is deleted. The value of integer can be from 0 to 99.</p> <p>Oracle: The Logging, No Sort, Degrees, and Instances properties are documented in the editor.</p>
Java Classes	Oracle	<p>A model or template, written in Java language, used to create objects with a common definition and common properties, operations and behavior.</p> <p>Java classes can be developed in Eclipse (or another Java development environment such as Oracle JDeveloper) and moved into an Oracle database to be used as stored procedures.</p> <p>Java classes must be public and static if they are to be used in this manner.</p> <p>When writing a class to be executed within the database, you can take advantage of a special server-side JDBC driver. This driver uses the user's default connection and provides the fastest access to the database.</p> <p>Java classes become full-fledged database objects once migrated into the database via the loadjava command-line utility or the SQL CREATE JAVA statement.</p> <p>A Java class is published by creating and compiling a call specification for it. The call spec maps a Java method's parameters and return type to Oracle SQL types.</p> <p>Once a Java class is developed, loaded, and published -- the final step is to execute it.</p>
Java Resources	Oracle	A Java resource is a collection of files compressed in a .jar file.

Database Object	DBMS Platforms	Notes
Libraries	Oracle	<p>A library is a configurable folder for storing and sharing content with an allocated quota. Multiple libraries may exist in the same database environment.</p> <p>A library is a special type of folder in Oracle Content Services. Unlike Containers and regular folders, each library has a Trash Folder and an allocated amount of disk space.</p> <p>A library is composed of a name (mandatory), description, quota, path, and library members.</p> <p>The library service allows you to create folders, list quotas, and manage categories, workflow, trash folders, and versioning. The Library service does not allow you to create or upload files.</p>
Materialized Views	Oracle	<p>A database object that contains the results of a query. They are local copies of data located remotely, or are used to create summary tables based on aggregations of table data. Materialized views are also known as snapshots.</p> <p>A materialized view can query tables, views, and other materialized views. Collectively, these are called master tables (a replication term) or detail tables (a data warehouse term).</p> <p>For replication purposes, materialized views allow you to maintain copies of remote data on your local node. These copies are read-only. If you want to update the local copies, you need to use the Advanced Replication feature. You can select data from a materialized view as you would from a table or view.</p> <p>For data warehousing purposes, the materialized views commonly created are aggregate views, single-table aggregate views, and join views.</p>
Materialized View Logs	Oracle	<p>Because Materialized Views are used to return faster queries (a query against a materialized view is faster than a query against a base table because querying the materialized view does not query the source table), the Materialized View often returns the data at the time the view was created, not the current table data.</p> <p>There are two ways to refresh data in Materialized Views, manually or automatically. In a manual refresh, the Materialized View is completely wiped clean and then repopulated with data from the source tables (this is known as a complete refresh). If source tables have changed very little, however, it is possible to refresh the Materialized View only for changed records -- this is known as a fast refresh.</p> <p>In the case of Materialized Views that are updated via fast refresh, it is necessary to create Materialized View Logs on the base tables that compose the Materialized View to reflect the changes.</p> <p>If the number of entries in this table is too high, it is an indication that you might need to refresh the Materialized Views more frequently to ensure that each update does not take longer than it needs.</p> <p>Select owner, then select from tables with Materialized Views, etc.</p>

Database Object	DBMS Platforms	Notes
Oracle Job Queue	Oracle	<p>The Oracle Job Queue allows for the scheduling and execution of PL/SQL stored procedures at predefined times and/or repeated job execution at regular intervals, as background processes.</p> <p>For example, you could create a job in the Oracle Job Queue that processed end-of-day accounting -- a job that must run every weekday, but can be run unattended, or you could create a series of jobs that must be run sequentially -- such as jobs that might be so large, that in order to reduce CPU usage, only one is run at a time.</p> <p>Runs PL/SQL code at specified time or on specified schedule, can enable/disable.</p>
Outlines	Oracle	<p>Oracle preserves the execution plans of "frozen" access paths to data so that it remains constant despite data changes, schema changes, and upgrades of the database or application software through objects named stored outlines.</p> <p>Outlines are useful for providing stable application performance and benefit high-end OLTP sites by having SQL execute without having to invoke the cost-based optimizer at each SQL execution. This allows complex SQL to be executed without the additional overhead added by the optimizer when it performs the calculations necessary to determine the optimal access path to the data.</p>

Database Object	DBMS Platforms	Notes
Packages	All	<p>A package is a procedural schema object classified as a PL/SQL program unit that allows the access and manipulation of database information.</p> <p>A package is a group of related procedures and functions, together with the cursors and variables they use, stored together in the database for continued use as a unit. Similar to standalone procedures and functions, packaged procedures and functions can be called explicitly by applications or users.</p> <p>DB applications explicitly call packaged procedures as necessary with privileges granted, a user can explicitly execute any of the procedures contained in it.</p> <p>Packages provide a method of encapsulating related procedures, functions, and associated cursors and variables together as a unit in the database. For example, a single package might contain two statements that contain several procedures and functions used to process banking transactions.</p> <p>Packages allow the database administrator or application developer to organize similar routines as well as offering increased functionality and database performance.</p> <p>Packages provide advantages in the following areas: encapsulation of related procedures and variables, declaration of public and private procedures, variables, constraints and cursors, separation of the package specification and package body, and better performance.</p> <p>Encapsulation of procedural constructs in a package also makes privilege management easier. Granting the privilege to use a package makes all constructs of the package assessable to the grantee.</p> <p>The methods of package definition allow you to specify which variables, cursors, and procedures are: public, directly accessible to the users of a package, private, or hidden from the user of the package.</p>
Package Bodies	Oracle	<p>A package body is a package definition file that states how a package specification will function.</p> <p>In contrast to the entities declared in the visible part of a package, the entities declared in the package body are only visible within the package body itself. As a consequence, a package with a package body can be used for the construction of a group of related subprograms in which the logical operations available to clients are clearly isolated from the internal entities.</p>

Database Object	DBMS Platforms	Notes
Primary Keys	All	<p>A key is a set of columns used to identify or access a row or rows. The key is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key.</p> <p>A unique key is a key that is constrained so that no two of its values are equal. The columns of a unique key cannot contain NULL values.</p> <p>The primary key is one of the unique keys defined on a table, but is selected to be the key of the first importance. There can only be one primary key on a table.</p> <p>Oracle: If an index constraint has been defined for a table, the constraint status for the table's primary key cannot be set to Disabled.</p>
Procedures	All	<p>A procedure is an application program that can be started through the SQL CALL statement. The procedure is specified by a procedure name, which may be followed by arguments enclosed within parenthesis.</p> <p>The argument or arguments of a procedure are individual scalar values, which can be of different types and can have different meanings. The arguments can be used to pass values into the procedure, receive return values from the procedure, or both.</p> <p>A procedure, also called a stored procedure, is a database object created via the CREATE PROCEDURE statement that can encapsulate logic and SQL statements. Procedures are used as subroutine extensions to applications, and other database objects that can contain logic.</p> <p>When a procedure is invoked in SQL and logic within a procedure is executed on the server, data is only transferred between the client and the database server in the procedure call and in the procedure return. If you have a series of SQL statements to execute within a client application, and the application does not need to do any processing in between the statements, then this series of statements would benefit from being included in a procedure.</p>
Profiles	Oracle	<p>Profiles are a means to limit resources a user can use by specifying limits on kernel and password elements. Additionally, Profiles can be used to track password histories and the settings of specific profiles may be queried.</p> <p>The following kernel limits may be set: maximum concurrent sessions for a user, CPU time limit per session, maximum connect time, maximum idle time, maximum blocks read per session, maximum blocks read per call, and maximum amount of SGA.</p>

Database Object	DBMS Platforms	Notes
Roles	Oracle	<p>A role is a set or group of privileges that can be granted to users to another role.</p> <p>A privilege is a right to execute a particular type of SQL statement or to access another user's object. For example: the right to connect to a database, the right to create a table, the right to select rows from another user's table, the right to execute another user's stored procedure.</p> <p>System privileges are rights to enable the performance of a particular action, or to perform a particular action on a particular type of object.</p> <p>Roles are named groups of related privileges that you grant users or other roles. Roles are designed to ease the administration of end user system and object privileges. However, roles are not meant to be used for application developers, because the privileges to access objects within stored programmatic constructs needs to be granted directly.</p>
Sequences	DB2, Oracle	<p>A sequence generates unique numbers.</p> <p>Sequences are special database objects that provide numbers in sequence for input into a table. They are useful for providing generated primary key values and for the input of number type columns such as purchase order, employee number, sample number, and sales order number.</p> <p>Sequences are created by use of the CREATE SEQUENCE command.</p>
Structured Types	DB2	<p>Structured Types are useful for modeling objects that have a well-defined structure that consists of attributes. Attributes are properties that describe an instance of the type.</p> <p>A geometric shape, for example, might have as attributes its list of Cartesian coordinates. A person might have attributes of name, address, and so on. A department might have a name or some other attribute.</p>
Synonyms	Oracle	<p>A synonym is an alternate name for objects such as tables, views, sequences, stored procedures, and other database objects.</p> <p>A synonym is an alias for one of the following objects: table, object table, view, object view, sequence, stored procedure, stored function, package, materialized view, java class, user-defined object type or another synonym.</p>
Tables	All	<p>Tables are logical structures maintained by the database manager. Tables are composed of columns and rows. The rows are not necessarily ordered within a table.</p> <p>A base table is used to hold persistent user data.</p> <p>A result table is a set of rows that the database manager selects or generates from one or more base tables to satisfy a query.</p> <p>A summary table is a table defined by a query that is also used to determine the data in the table.</p>

Database Object	DBMS Platforms	Notes
Tablespaces	DB2, Oracle	A tablespace is a storage structure containing tables, indexes, large objects, and long data. Tablespaces reside in database partition groups. They allow you to assign the location of database and table data directly onto containers. (A container can be a directory name, a device name, or a file name.) This can provide improved performance and more flexible configuration.
Triggers	All	<p>A trigger defines a set of actions that are performed when a specified SQL operation (such as delete, insert, or update) occurs on a specified table. When the specified SQL operation occurs, the trigger is activated and starts the defined actions.</p> <p>Triggers can be used, along with referential constraints and check constraints, to enforce data integrity rules. Triggers can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions to perform tasks such as issuing alerts.</p>
Undo Segments	Oracle	<p>In an Oracle database, Undo tablespace data is an image or snapshot of the original contents of a row (or rows) in a table. The data is stored in Undo segments in the Undo table space.</p> <p>When a user begins to make a change to the data in a row in an Oracle table, the original data is first written to Undo segments in the Undo tablespace. The entire process (including the creation of the Undo data is recorded in Redo logs before the change is completed and written in the Database Buffer Cache, and then the data files via the database writer (DBW) process.)</p>

Database Object	DBMS Platforms	Notes
Unique Keys	All	<p>A unique key is a key that is constrained so that no two of its values are equal. The columns of a unique key cannot contain null values. The constraint is enforced by the database manager during the execution of any operation that changes data values, such as INSERT or UPDATE. The mechanism used to enforce the constraint is called a unique index. Thus, every unique key is a key of a unique index. Such an index is said to have the UNIQUE attribute.</p> <p>A primary key is a special case of a unique key. A table cannot have more than one primary key.</p> <p>A foreign key is a key that is specified in the definition of a referential constraint.</p> <p>A partitioning key is a key that is part of the definition of a table in a partitioned database. The partitioning key is used to determine the partition on which the row of data is stored. If a partitioning key is defined, unique keys and primary keys must include the same columns as the partitioning key, but can have additional columns. A table cannot have more than one partitioning key.</p> <p>Oracle: You cannot drop a unique key constraint that is part of a referential integrity constraint without also dropping the foreign key. To drop the referenced key and the foreign key together, check the Delete Cascade option for the foreign key.</p> <p>Clustered: A cluster composes of a group of tables that share the same data blocks, and are grouped together because they share common columns and are often used together.</p> <p>Filegroup: Lets you select the filegroup within the database where the constraint is stored.</p> <p>Fill Factor: Lets you specify a percentage of how large each constraint can become.</p>
Views	All	<p>A view provides an alternate way of looking at the data in one or more tables.</p> <p>A view is a named specification of a result table and can be thought of as having columns and rows just like a base table. For retrieval purposes, all views can be used just like base tables.</p> <p>You can use views to select certain elements of a table and can present an existing table in a customized table format without having to create a new table.</p>

DBMS CONNECTION PARAMETERS BY PLATFORM

The following topics provide connection details:

- [IBM DB2 LUW](#)
- [Microsoft SQL Server](#)

- [JDBC Connection Parameters](#)
- [Oracle Connection Parameters](#)
- [Sybase Connection Parameters](#)

IBM DB2 LUW

Connection Parameter	Description
Use Alias from IBM Client or Generic JDBC Configuration	If you choose to use the alias from the IBM client, select the appropriate alias name. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Schema ID (Optional)	The name of the database schema.
Function Path	Optional. Enter an ordered list of schema names to restrict the search scope for unqualified function invocations.
Security Credentials	The log on information required by DB Optimizer™ XE and DB Optimizer™ to connect to the data source.
Auto Connect	Automatically attempts to connect to the data source when selected in Data Source Explorer, without prompting the user for connection information.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer™ XE and DB Optimizer™ to initiate a JDBC standard access connection.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a data source. Often contains host and port numbers, as well as the name of the data source to which it connects. For example: <code>jdbc:postgresql://host:port/database</code> <code>jdbc:derby://host:port/database</code>
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

MICROSOFT SQL SERVER

Connection Parameter	Description
Use Network Library Configuration	If the data source utilizes a network library, select this parameter. The corresponding connection parameter fields become available. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Host/Instance (JDBC Configuration)	The name of the data source.
Port (JDBC Configuration) (optional)	The listening port used in TCP/IP communications between DB Optimizer™ XE and DB Optimizer™ and the data source.
Protocol (JDBC Configuration)	The communication mechanism between DB Optimizer™ XE and DB Optimizer™ and the data source. Choose TCP/IP or Named Pipes.
Default Database (Optional)	The default SQL database name, as defined by the schema.
Security Credentials	The log on information required by DB Optimizer™ XE and DB Optimizer™ to connect to the data source.
Auto Connect	Automatically attempts to connect to the data source when selected in Data Source Explorer, without prompting the user for connection information.

Connection Parameter	Description
Allow Trusted Connections	Enables trusted connections to the data source from DB Optimizer™ XE and DB Optimizer™.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer™ XE and DB Optimizer™ to connect and communicate with the database.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: <code>jdbc:postgresql://host:port/database</code> <code>jdbc:derby://host:port/database</code>
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

JDBC CONNECTION PARAMETERS

Connection Parameter	Description
Connect String	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: <code>jdbc:postgresql://host:port/database</code> <code>jdbc:derby://host:port/database</code>
Data Source Name	The name of the data source to which you want DB Optimizer™ XE and DB Optimizer™ to connect.

ORACLE CONNECTION PARAMETERS

Connection Parameter	Description
Use TNS Alias	If the data source is mapped to a net service name via tnsnames.ora, select this parameter. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Host/Instance (JDBC Configuration)	The name of the host machine on which the data source resides.
Port (JDBC Connection)	The listening port used in TCP/IP communications between DB Optimizer™ XE and DB Optimizer™ and the data source.
Type (JDBC Configuration)	Indicates if the data source is defined via a system identifier (SID) or a service name.
Service/SID Name (JDBC Configuration)	The name of the system identifier (SID) or service name that identifies the data source.
Security Credentials	The log on information required by DB Optimizer™ XE and DB Optimizer™ to connect to the data source.
Auto Connect	Automatically attempts to connect to the data source when selected in data source Explorer, without prompting the user for connection information.
Allow Trusted Connections	Enables trusted connections to the data source from DB Optimizer™ XE and DB Optimizer™.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer™ XE and DB Optimizer™ to connect and communicate with the database.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: <code>jdbc:postgresql://host:port/database</code> <code>jdbc:derby://host:port/database</code>
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

SYBASE CONNECTION PARAMETERS

Connection Parameter	Description
Use Alias Information from your SQL.INI File	If the data source is mapped to a name via SQL.INI, select this parameter to use that name for connection. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Host/Instance (JDBC Connection)	The name of the host machine on which the data source resides.
Port (JDBC Connection)	The listening port used in TCP/IP communications between DB Optimizer™ XE and DB Optimizer™ and the data source.
Default Database (JDBC Connection) (Optional)	The default database name, as defined by the schema.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer™ XE and DB Optimizer™ to connect and communicate with the database.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: <code>jdbc:postgresql://host:port/database</code> <code>jdbc:derby://host:port/database</code>
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

Index

A

- archives, profiling
 - opening/saving 99
- associating 59
- Average Active Sessions (AAS) 77
- Average Active Sessions (AAS) graph 77

B

- bind variable 138

C

- cases, generated
 - opening in context 176
- change history 56
- Clean 101
- code folding 51
- code formatting 46

D

- data source 23
- Data source indexing 24
- data source indexing 24
- database objects 30, 245
- DB2 LUW 256
- delete 36

E

- editing 39
- error 69
- error detection 41
- error logs 68
- execute 61
- Exists 163
- explain plans
 - opening from tuning job 178

F

- files 35
- filtering profile results 102
- filters 30

G

- global filters 31

H

- hints
 - opening in context 176
- hyperlinks 46

I

- IBM DB2 LUW 256
- Import 36
- In 163
- index analysis, SQL Tuner 150, 224
- Indirect Relationship 166

J

- JDBC connection 258

L

- license 14
- Load Chart 76
- Load Chart legend 78
- local history 57
- log 68

M

- Max CPU 77
- Max Engine 77

N

- Nested Subqueries 161
- new_project_wizard_page 34
- Not Exists 163
- Not in 163

O

- object properties 27
- objects 27
- opening 34
- Oracle 258

P

- permissions, SQL Profiler 103
- permissions, SQL Tuner 181
- Profiling Repository 100
- profiling sessions
 - configuring DBMS for 103
 - filtering results 102
 - opening/saving 99
 - submitting tuning sessions from 101
- project_info_page 34
- Projects 33

Q

- Query Rewrites 149

R

- roles, SQL Tuner 181

S

- Saving
 - Tuning Job 180
- Saving Tuning Job 115
- SQL
 - tuning 135
 - Viewing Diagram SQL 170
- SQL file 39
- SQL Server 256
- Statements
 - grouping 80
- Sybase 259

T

- Top Activity 78
- Top Object I/O Tab (Oracle-Specific) 79
- Top Procedures Tab (SQL Server and Sybase Specific) 79
- transformations 149
- Tuning Job
 - Saving 180

INDEX

tuning jobs

- editor preferences 182
- index analysis 150, 224
- introduced 135
- opening explain plans from 178
- roles/permissions required 181
- understanding generated statements 185

tuning sessions

- opening from profiling session 101

U

UNKNOWN statements 81

W

Wait Event 78

workspace 14