

IDERA SQL QUERY TUNER 17.1.0

I D E R A

Table of Contents

SQL Query Tuner optimizes SQL queries	8
Release notes	9
New features and fixed issues	9
17.1.0 New features.....	9
17.1.0 Fixed issues.....	9
Known issues.....	9
Welcome to SQL Query Tuner	11
What is IDERA SQL Query Tuner?	11
Configuring SQL Query Tuner.....	12
Product requirements.....	12
Hardware Requirements.....	12
Supported Operating Systems	12
DBMS Support	12
Initial setup.....	12
Specify a workspace	13
License SQL Query Tuner.....	13
Installing SQL Query Tuner.....	15
Customizing SQL Query Tuner (Preferences)	17
Specify Data Sources Preferences.....	18
Specify SQL Development Preferences.....	18
Specify SQL Editor Preferences	20
Specify SQL Execution Preferences.....	27
Specify SQL Filters Preferences.....	28
Specify Profile Alerts Preferences	29
Specify Tuning Job Editor Preferences.....	29
Specify VST Diagrams Tuning Preferences	29
Specify File Encoding Preferences	29
Introduction to Database Tuning.....	32
Introduction to SQL Query Tuner's Tuner	32
Tuning Example	32
SQL Tuning Methodology	34

SQL Tuner Overview.....	34
What's Happening on the Databases?	35
Databases	36
AverageActiveSessions(AAS)Loadofselecteddatabase.....	36
MaximumCPULine.....	36
TopSQL, TopBottlenecks, and TopSessions.....	36
Tuning Example	38
The Database is Hanging or the Application has Problems.....	38
The Database Caused the Problem.....	39
The Machine Caused the Problem	40
Finding and Tuning Problem SQL	41
Using SQL Query Tuner	43
Working with Data Sources	43
Register data sources.....	43
Add a New Data Source.....	44
Import and Export Data Souces	46
Categorize Data Sources.....	48
Customizing Data Source Categories.....	48
Browse a Data Source	48
View Database Object Properties	49
Search for Database Objects	50
Filter database objects	52
Drop a Database Object.....	53
Working with SQL Projects	53
Create a New SQL Project.....	54
Open an Existing Project.....	55
Add Files to a Project	55
Search a Project	56
Delete a Project	56
Creating and Editing SQL Files (SQL Editor)	57
Semantic Validation.....	58
Create an SQL File	58
Open an Existing SQL File	59
Working in SQL Editor.....	59

View Change History	74
Revert to an Old Version of a File	75
Delete an SQL File	75
Executing SQL Files	76
Associate an SQL file with a data source	76
Configure an SQL Session	77
Execute SQL Code	78
View and Save Results	79
Troubleshooting.....	80
View Log Details	81
Maintain Logs	81
Filter Logs	82
Import and Export Error Logs	84
Find and Fix SQL Code Errors	84
Find and Fix Other Problems	85
Using Profiling.....	86
Understanding Profiling	86
Understanding the Interface	86
Running a Profiling Session	87
Execute a Profiling Session.....	88
Working with Session Results.....	89
Creating Profiling Reports	110
Saving Profiling Sessions.....	112
Import Statements to Tuning.....	112
Other Profiling Commands.....	112
Configuring Profiling.....	113
Building profiling configurations	113
Configuring DBMS Properties and Permissions	114
Specify Profile Alerts Preferences during Configuration	115
Using SQL Load Editor/Tester	117
Using Tuning	120
Understanding the Tuner Interface.....	120
Understanding the Overview tab	120
Understanding the Analysis Tab	125

Using the Table Statistics tab.....	127
Using the Column Statistics and Histograms tab.....	128
Using the Outlines tab	129
Tuning SQL statements in the System Global Area.....	130
Tuning SQL Statements	131
Create a New Tuning Job.....	131
Specify a data source	132
Add SQL Statements	132
Managing Bind Variable Errors	133
Run a tuning job	136
Analyze tuning results.....	137
Modify tuning results	141
Using the Analysis tab.....	143
Visual SQL Tuning	145
Interpreting the VST Diagram Graphics	153
Additional Tuning Commands	164
View the Source Code of Tuning Candidates.....	164
View Statement or Case Code in SQL Viewer.....	164
Open an Explain Plan for a Statement or Case.....	164
Executing a Session from the Command Line	165
Saving a Tuning Job.....	165
Configuring Tuning	166
Configuring Tuning: Specify Tuning Job Editor Preferences.....	166
Specify case generation preferences	168
Specify VST Diagrams Tuning Preference.....	169
Examples of Transformations and SQL Query Rewrites	170
Cartesian Product Elimination	170
Expression Transformation	170
Invalid Outer Join.....	170
Transitivity.....	170
Move Expression to WHERE Clause	170
NULL Column	170
Push Subquery	171
Mismatched column types	171

Reference.....	172
Database objects.....	172
JDBC Connection Parameters.....	176

SQL Query Tuner optimizes SQL queries

- Pinpoint problematic SQL queries with wait time analysis
- Receive automated tuning suggestions
- Visually tune complex SQL queries
- Load test simulated production environments
- Streamline SQL query tuning for SQL Server

Release notes

Thank you for using IDERA SQL Query Tuner, an enterprise SQL development and optimization tool that gives database developers and administrators the perfect environment for building the best performing SQL code. This application easily identifies inefficient SQL code and boosts developer productivity by ensuring SQL is right before it becomes a problem in mission-critical applications. SQL Query Tuner also provides a rich SQL IDE with code completion, real-time error checking, code formatting and sophisticated object validation tools. Lastly, SQL Query Tuner provides advanced project and data source management tools.

To get a quick glimpse into the newest features, fixed issues, and known issues in this release of SQL Query Tuner, review the following sections of the Release Notes:

- [Learn about key new features in this release](#)
- [Review issues fixed by this release](#)
- [See known issues](#)

New features and fixed issues

IDERA SQL Query Tuner provides the following new features and fixed issues.

17.1.0 New features

Improves operating system and platform support

SQL Query Tuner 17.1.0 now supports the following operating system and database platforms:

- Windows Server 2019 and 2012 R2
- SQL Server 2014 SP2

17.1.0 Fixed issues

Issue	Description	DBMS
OPT-5281	DBO gives a COUNT field incorrect error	SQL Server

Known issues

Idera strives to ensure our products provide quality solutions for your SQL Server needs. The following known issues are described in this section. If you need further assistance with any issue, please contact [Support](#) (www.idera.com/support).

The SQL Server sqlvariant data type is not currently supported by the Microsoft JDBC driver. If a query is used to retrieve data from a table that contains a column of the sqlvariant data type, an exception will occur. There are two workarounds to fix this issue:

- Use CAST to type cast the sqlvariant columns to the data type you want.

or

- Use the jTDS driver to connect.

Welcome to SQL Query Tuner

IDERA SQL Query Tuner 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. SQL Query Tuner's user interface helps improve overall productivity with integrated development, monitoring, and tuning components.

SQL Query Tuner 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 \(SQL Editor\)](#).
- **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 SQL Query Tuner. For more information, see [Using SQL Load Editor/Tester](#).
- **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](#).
- **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 more information, see [Using Tuning](#).

What is IDERA SQL Query Tuner?

IDERA SQL Query Tuner 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. SQL Query Tuner's user interface helps improve overall productivity with integrated development, monitoring, and tuning components.

SQL Query Tuner 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 \(SQL Editor\)](#).
- **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 SQL Query Tuner. For more information, see [Using SQL Load Editor/Tester](#).
- **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](#).
- **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 more information, see [Using Tuning](#).

Configuring SQL Query Tuner

This section contains information on configuring SQL Query Tuner. 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.

- [Product requirements](#)
- [Initial setup](#)
- [Customizing SQL Query Tuner \(Preferences\)](#)

Product requirements

Before installing IDERA SQL Query Tuner, verify that your environment meets the following requirements.

Hardware Requirements

The following are the minimum hardware requirements:

- 1 GHz or faster processor
- 3 GB of RAM
- 1 GB of free disk space
- 1024 x 768 screen resolution

Supported Operating Systems

SQL Query Tuner is supported on the following operating systems (32- and 64-bit):

- Microsoft Windows 7, 8, 8.1, and 10
- Microsoft Windows Server 2019, 2016, 2012, 2012 R2, 2008 SP1, and 2008 R2

DBMS Support

SQL Query Tuner is supported on the following SQL Server versions:

- MS SQL Server 2019* (tested against community technology preview 3.2), 2017*, 2016*, 2014*, 2014 SP2*, 2012, 2008, and 2005

* Supports only a subset of this database version's features/functions

Initial setup

The following topics provide general help for configuring SQL Query Tuner:

- [Specify a workspace](#)
- [License SQL Query Tuner](#)
- [Installing SQL Query Tuner](#)

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

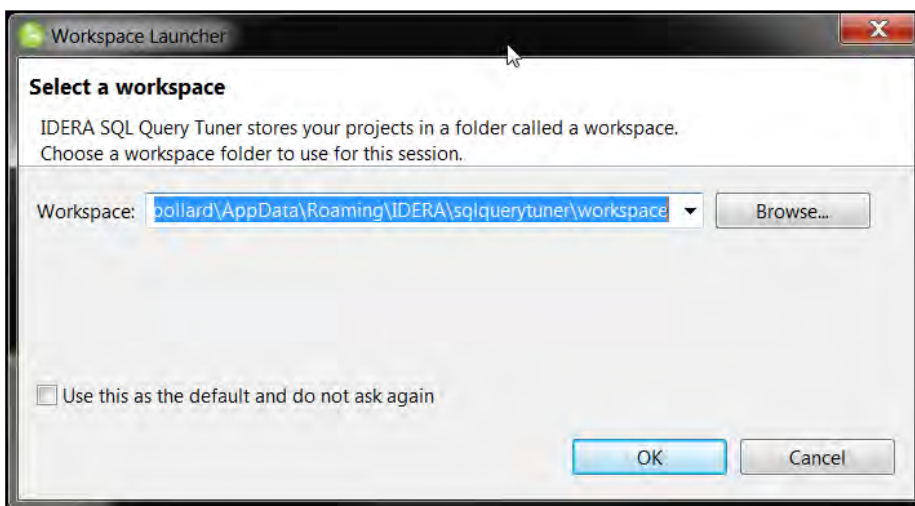
- [Specify Data Sources Preferences](#)
- [Specify SQL Development Preferences](#)

- [Specify SQL Editor Preferences](#)
- [Specify SQL Execution Preferences](#)
- [Specify SQL Filters Preferences](#)
- [Specify Profile Alerts Preferences](#)
- [Specify Tuning Job Editor Preferences](#)
- [Specify VST Diagrams Tuning Preferences](#)
- [Specify File Encoding Preferences](#)

Specify a workspace

When you install SQL Query Tuner, you are prompted to create a workspace. Then when you launch SQL Query Tuner, you have an opportunity to choose your workspace. At any time while running SQL Query Tuner, you can change your workspace.

To access the Workspace Launcher, select **File > Switch Workspace**.



License SQL Query Tuner

Manage your SQL Query Tuner license

SQL Query Tuner requires a license key. This key allows you to connect to SQL Server instances and use all the features of SQL Query Tuner.

Though you can immediately begin using SQL Query Tuner with the included evaluation key, consider updating the evaluation license as soon as possible. A registered license key allows you to unlock the evaluation time and registered instance limits so you can begin using SQL Query Tuner to the fullest extent.

SQL Query Tuner provides an intuitive interface for license key management. You can view the status of your license keys and add licenses to have additional instances to which you can connect. Each license allows you to connect to a predetermined number of SQL Server instances.

When you reach your license limit, SQL Query Tuner prevents you from connecting to any new instances until you get a new license or remove some licensed instances. When your trial period expires, SQL Query Tuner prevents you from using any instances.

To open the License Keys window, select **Help > IDERA SQL Query Tuner Licenses** from the Toolbar menu.

Use the trial license

SQL Query Tuner includes a limited-time trial that allows you to create your SQL Query Tuner Repository on the first start. As the end of the trial period nears, SQL Query Tuner warns you each time you start the application that your trial period is ending. At the end of this trial period, you must upgrade the application license to continue to function properly. This trial license is shared by all SQL Query Tuner applications using the same repository.

SQL Query Tuner limits the number of SQL Server instances to one when using a trial license.

Add a new license

If you would like to connect to additional SQL Server instances beyond the limits of your existing license, contact [IDERA Sales](#) to purchase additional licenses.

Once you receive a new license, open the IDERA Product License Manager window by clicking **Help > IDERA SQL Query Tuner Licenses**. Click **Add License Key** or right-click in the **License Manager** tree, and then select **Add License Key** from the menu to bring up the Add License Key window. Type your license key in the available space, and then click **OK**. Your new license lists the number of additional SQL Server instances to which you can connect. Each new license you add to SQL Query Tuner Repository adds to the total number of SQL Server instances to which you can connect.

To get a new license, contact your [IDERA Sales representative](#).

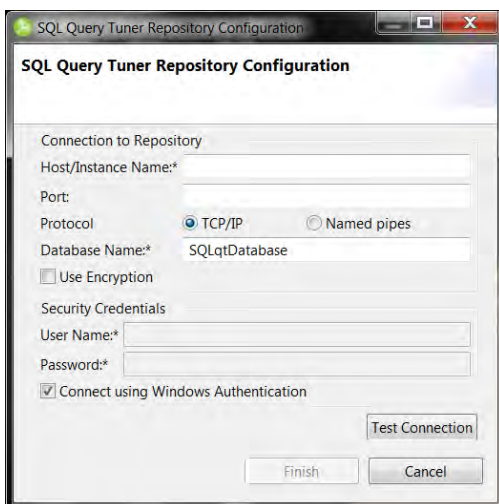
Manage your SQL Server instances

The IDERA Product License Manager window (**Help > IDERA SQL Query Tuner Licenses**) allows you to see all the SQL Server instances that you can connect to by a SQL Query Tuner application. You can:

- Edit the SQL Query Tuner Repository connection information used by the application.
- Remove a SQL Server instance from the list of instance nodes that allow connection.
- Add a new license to the SQL Query Tuner Repository.

Configuring the repository database

The first time you open the SQL Query Tuner, the SQL Query Tuner Repository Configuration dialog appears. Use this dialog to select the SQL Server instance to use for the IDERA SQL Query Tuner repository database. If you want to use a repository that already exists, then enter the information to that repository.



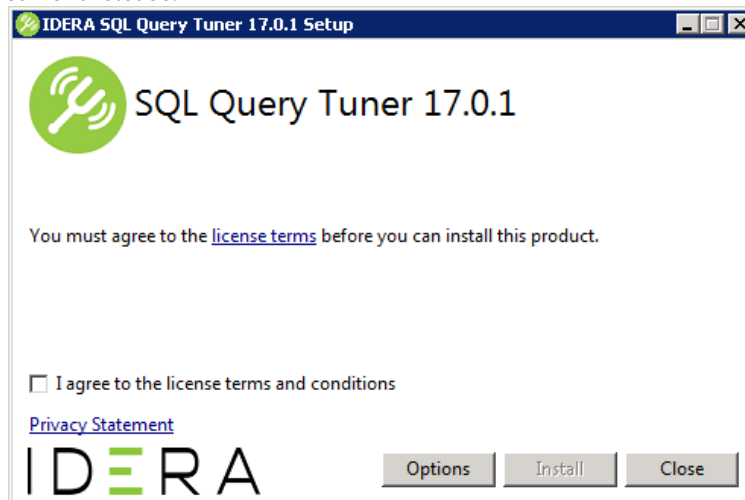
To specify the SQL Server instance where you want to install the Repository database:

1. In **Host/Instance Name**, specify the host SQL Server instance you want to use for the IDERA SQL Query Tuner Repository database.
2. *Optional.* Type a port to use for your SQL Server instance if it is not the default port.
3. Choose the appropriate connection protocol.
4. If you do not want to use the default name for the IDERA SQL Query Tuner Repository database, type the appropriate name. If your SQL Server environment uses SSL to connect, check **Use Encryption**.
5. **If your SQL Server environment uses SQL Server Authentication**, enter the username and password used to connect to the repository. **If your SQL Server environment instead uses Windows Authentication**, click the checkbox and the current user's Windows credentials are used to connect to the repository.
NOTE: If the database does not already exist, this user must have the proper permissions to create a database on the SQL Server instance. Additionally, the user must have create table privileges to create the repository tables in that database.
6. Click **Test Connection** to verify that all the information entered is correct.
7. Click **Finish**.

Installing SQL Query Tuner

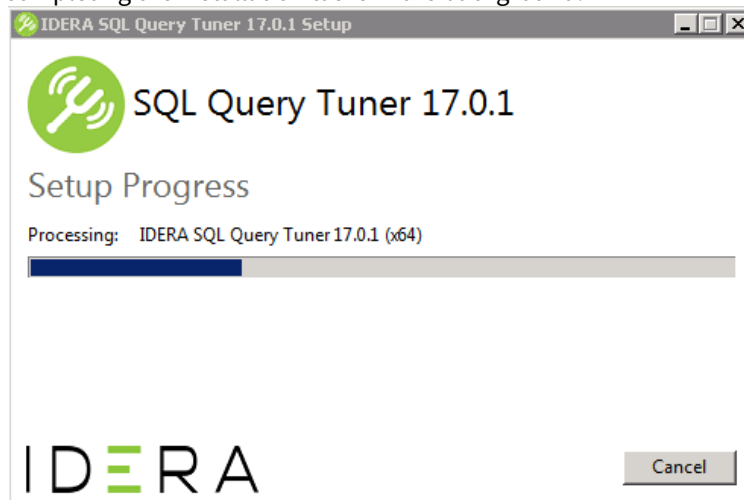
Read all of the installation steps before proceeding to make sure you have all the necessary information available.

1. Run the SQL Query Tuner installer. The following images are from version 17.0.1, but are still valid in the current release.

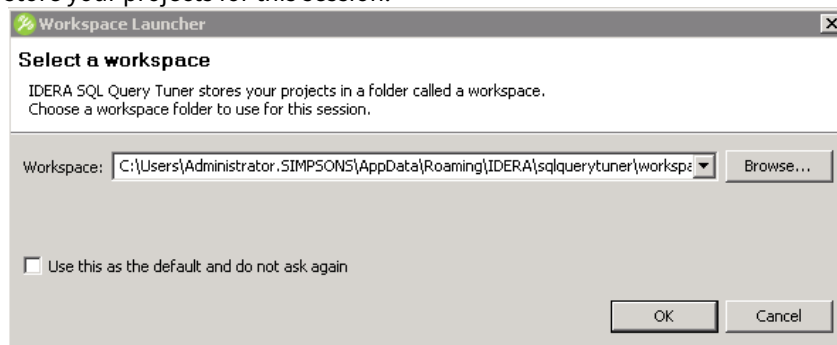


- i** If you need to change the default install location, click **Options**, and then browse for and select the appropriate location.

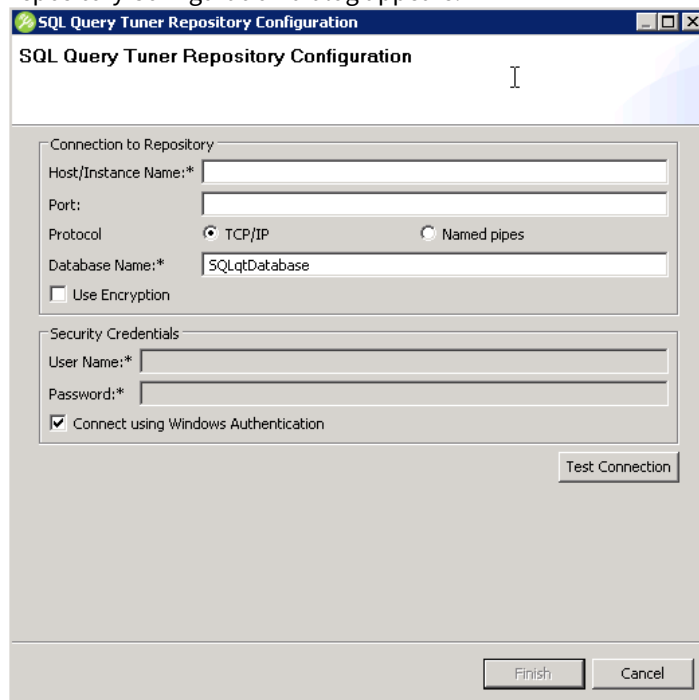
2. Agree to the license terms, and then click **Install**. The setup window displays a progress bar while completing the installation tasks in the background.



3. After a successful setup, click **Launch**. The Workspace Launcher appears asking you to select a workspace to store your projects for this session.



4. Select a workspace, and then click **OK**. SQL Query Tuner creates the workspace. The SQL Query Tuner Repository Configuration dialog appears.



5. In the **Host/Instance Name** and **Port** fields, type the hostname of the machine that contains the Repository database.
6. In the Security Credentials area, enter the username and password used to access the Repository database.
7. *Optional.* Click **Test Connection** to check your database connection and to create or update any tables necessary.

Customizing SQL Query Tuner (Preferences)

To customize various aspects of SQL Query Tuner, select the aspect you want to customize from the **Preferences** menu.

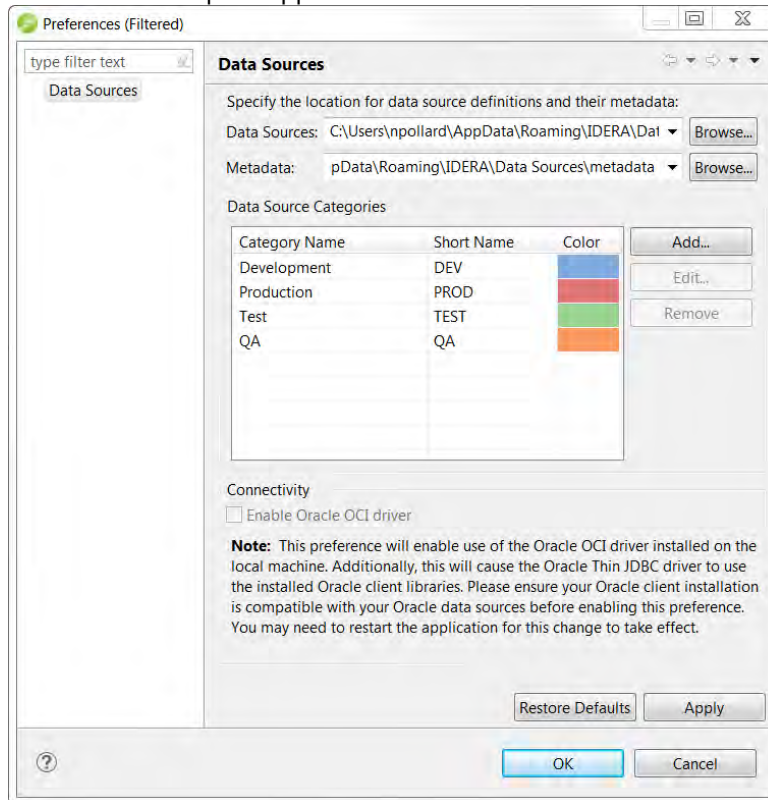
This section is comprised of the following topics:

- [Specify Data Sources Preferences](#)
- [Specify SQL Development Preferences](#)
- [Specify SQL Editor Preferences](#)
- [Specify SQL Execution Preferences](#)
- [Specify SQL Filters Preferences](#)
- [Specify Profile Alerts Preferences](#)
- [Specify Tuning Job Editor Preferences](#)
- [Specify VST Diagrams Tuning Preferences](#)
- [Specify File Encoding Preferences](#)

Specify Data Sources Preferences

When you add a data source to your list of Managed Data Sources in the Data Source Explorer, SQL Query Tuner stores the definition and metadata for the data source in the location you specify here. For information on adding data sources, see [Register data sources](#).

1. From the **Preferences** menu, select **Data Sources**.
The **DataSources** pane appears.



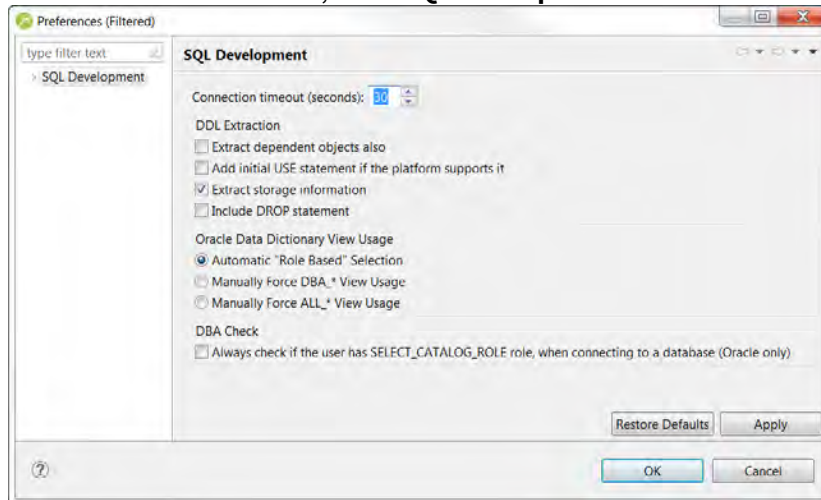
2. Specify the location for data source definitions and their metadata.
3. Click **Apply**.

i For information on adding custom categories, see [Customizing Data Source Categories](#).

Specify SQL Development Preferences

The SQL Development Preferences specified on the first page of the SQL Development Preferences determines SQL Query Tuner behavior when connecting to and extracting DDL from a data source. For information on preferences accessible by expanding SQL Development, see [Customizing SQL Query Tuner \(Preferences\)](#).

1. From the **Preferences** menu, select **SQL Development**.



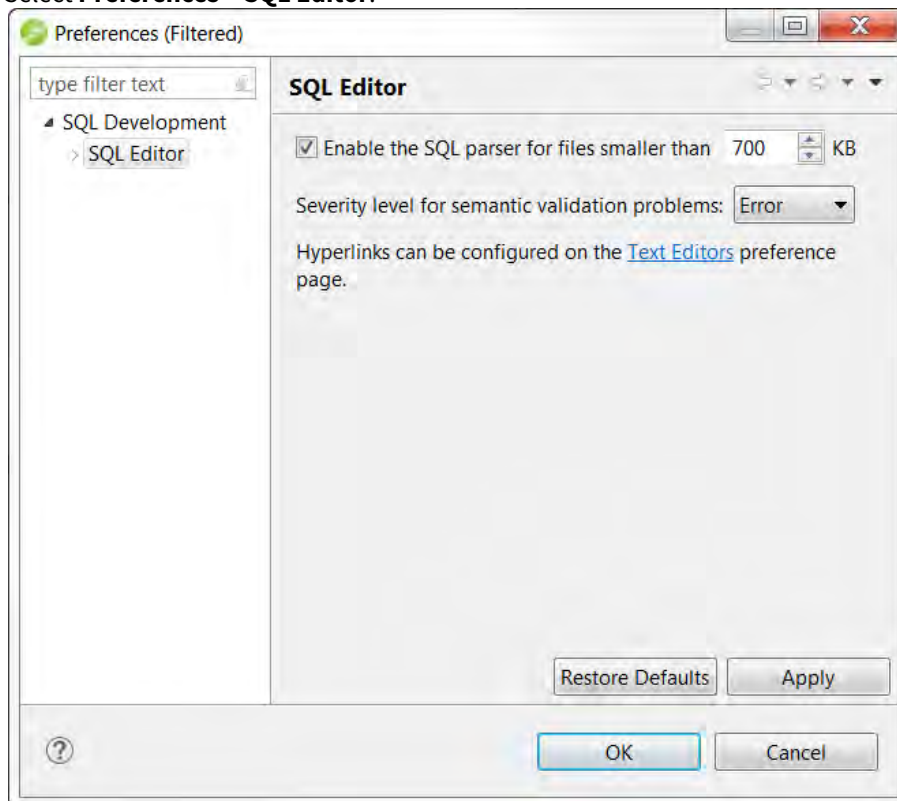
2. Choose your preferences, and then click **Apply**.

The following describes the SQL Development preferences:

- **Connection timeout (seconds):** Specify the connection timeout before the connection to the database fails.
- **Extract dependent objects also:** If selected, when extracting DDL dependent objects such as indexes are also extracted.
- **Add initial USE statement if the platform supports it:** If selected, a USE statement is added to the DDL extracted. Adding the USE statement ensures that when you run the DDL, you are using the correct database context.
- **Extract storage information:** If selected, when extracting DDL object storage information is also extracted.
- **Include DROP statement:** If selected, the DROP statement will be added to the DDL so you can easily execute the statement.

Specify SQL Editor Preferences

1. Select **Preferences > SQL Editor**.



2. Change the settings as appropriate in each section and then click **Apply**.

- **Enable the SQL parser...:** For performance reasons, you may want to enable the SQL parser only if a SQL file is smaller than the size you specify here.
- **Severity level for semantic validation problems:** Choose a security level from this list. This 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.

i 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

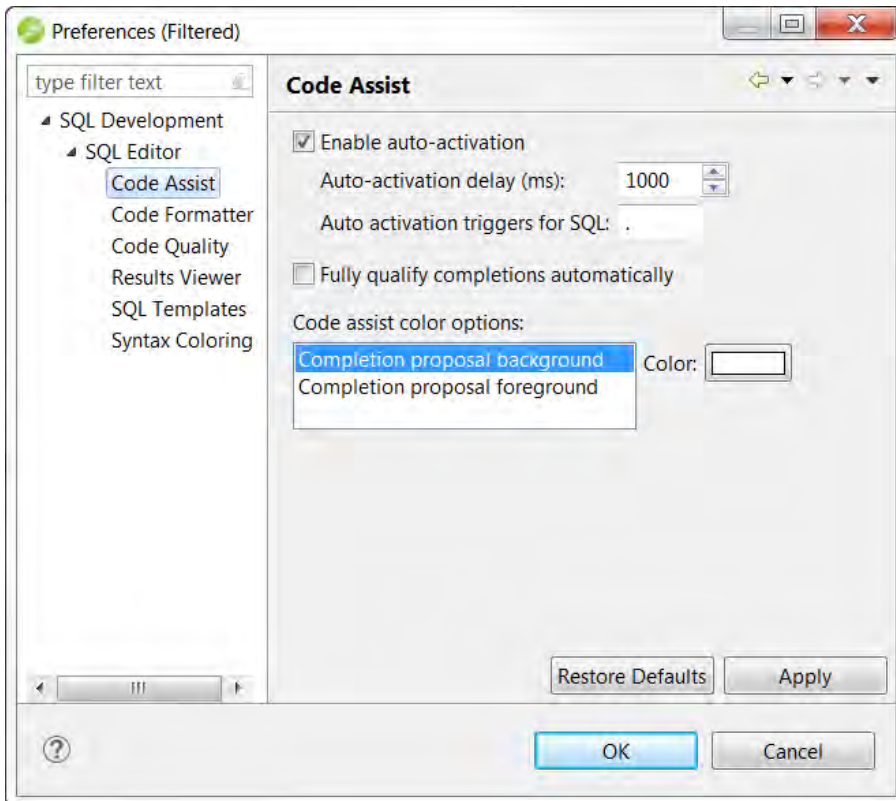
See the following topics to configure other SQL Editor preferences.

- [Specify Code Assist Preferences](#)
- [Specify Code Formatter Preferences](#)
- [Specify Code Quality Preferences](#)
- [Specify Results Viewer Preferences](#)
- [Specify SQL Templates Preferences](#)
- [Specify Syntax Coloring Preferences](#)

Specify Code Assist Preferences

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

Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand **SQL Editor** and click **Code Assist**.



The following describes the options on the Code Assist Preferences page.

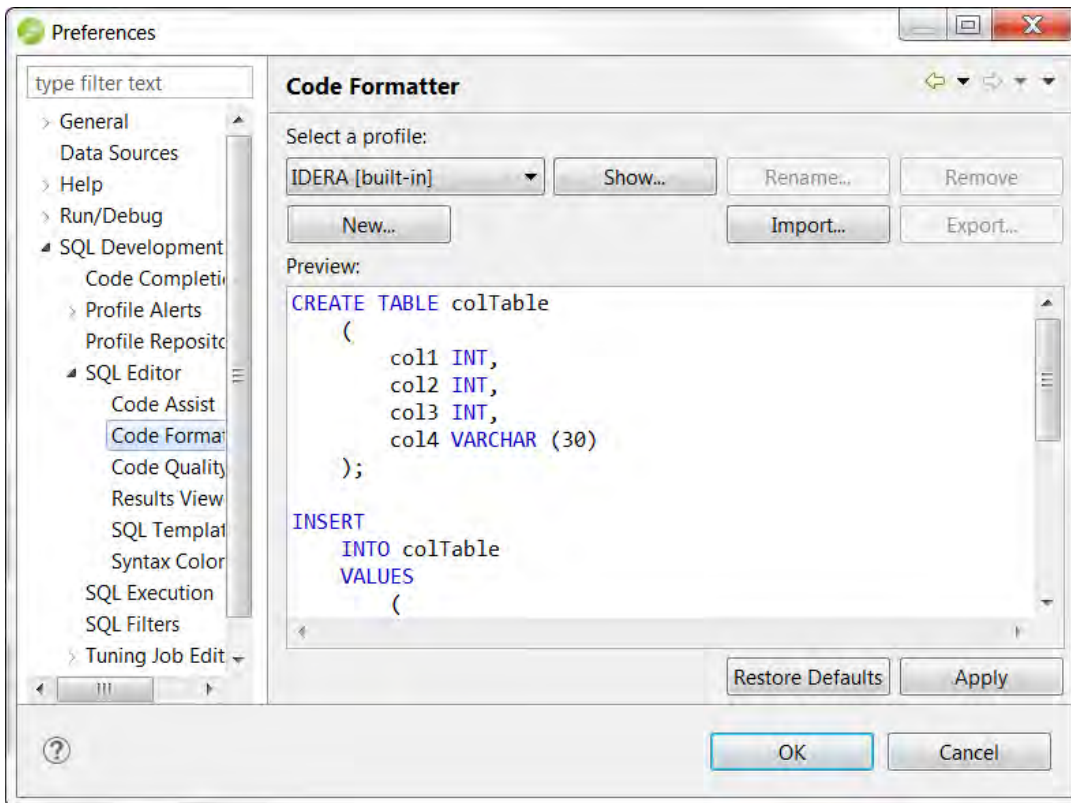
- **Enable auto-activation:** When selected enables code assist functionality with the Ctrl + Space command. If this option is selected, the code assist window automatically appears when you stop typing.
- **Auto-activation delay (ms):** Specify the amount of time in milliseconds that the window automatically appears.
- **Auto activation triggers for SQL:** Enter a trigger character or trigger characters. When you enter an activation trigger in the SQL Editor, you will see the code assist options available.
- **Fully qualify completions automatically:** When selected, 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.

Specify Code Formatter Preferences

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

Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand **SQL Editor** and click **Code Formatter**.



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

- **Select a profile:** From the list, choose the profile you want to view.
- Click **Show** to view the details of the Profile. On the **Show Profiles** dialog that appears, you can edit the profile and save the changes.
- 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.

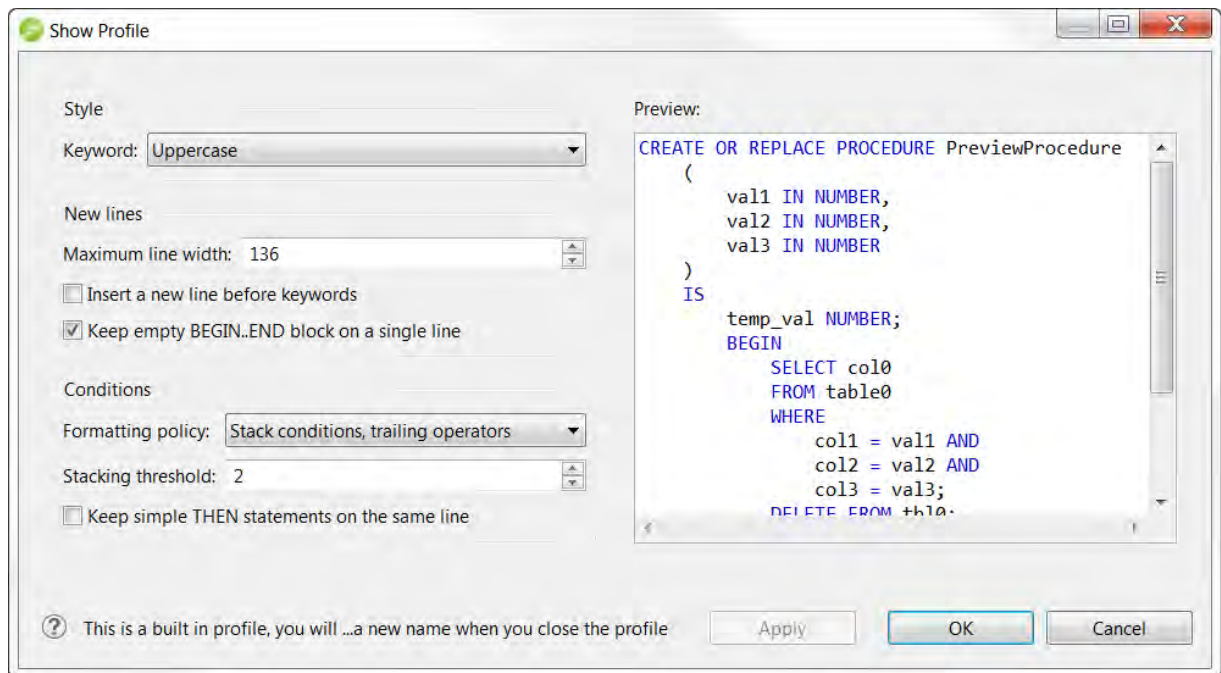
i If you create a new profile with a name that already exists in the system, a prompt will appear asking you to change the name of the new code formatting template.

Create and edit Code Formatting Profiles

You can create your own code formatting profiles that will define how your SQL code is formatted.

1. Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand SQL Editor and click **Code Formatter**.

2. On the **Code Formatter** pane, click **Show** or **New** and a dialog similar to the following appears.

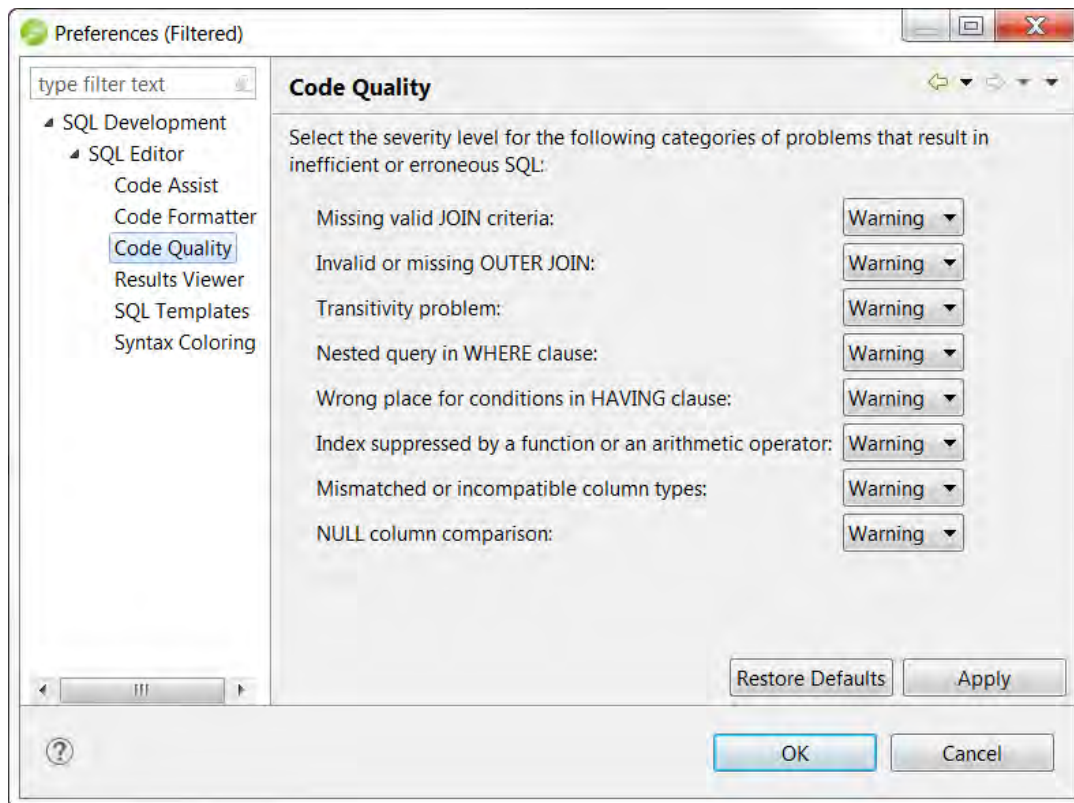


3. Make your changes, click **Apply** to preview your changes and then click **OK** to create the new profile or to exit the **Show Profile** dialog.
4. Use the preview pane to preview the changes you make to your code formatting preferences. Changes are not implemented until you click **Apply** or **OK**.

Specify Code Quality Preferences

The Code Quality preferences allow you to specify the severity level for several categories of problems that result in inefficient or erroneous SQL.

1. Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand **SQL Editor** and click **Code Quality**.

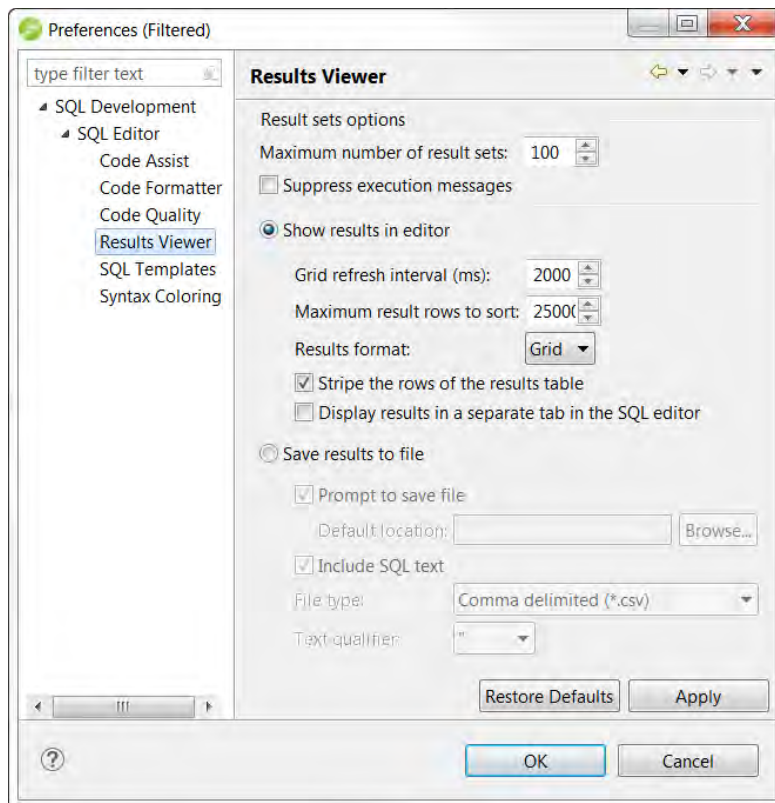


2. Change the severity levels by clicking the list next to the category and choosing the level.
3. To save your changes, click **Apply**.

Specify Results Viewer Preferences

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

1. Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand **SQL Editor** and click **Results Viewer**.



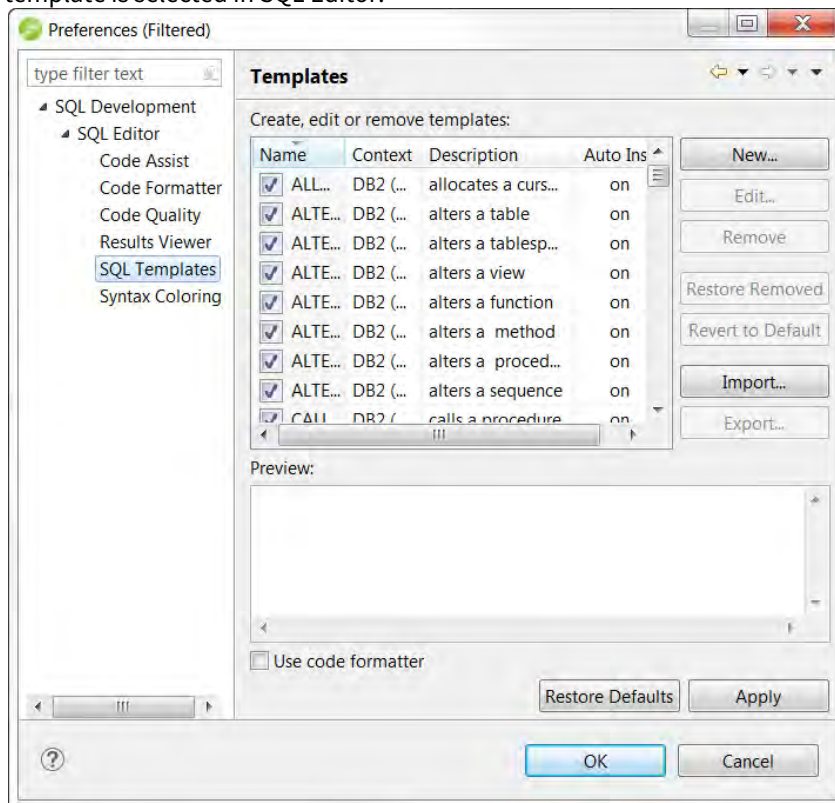
2. Make your preference changes and then save your changes by clicking **Apply**. The following describes the preference options available:
 - - **Maximum number of result sets:** If selected and the executed SQL returns more results sets than the maximum specified, result sets in excess of the maximum specified will not be displayed.
 - **Suppress execution messages:** If selected and the SQL you execute returns informational messages, they will not be displayed.
 - **Show results in editor:** Execution results can be either shown in the editor or sent to a file. If you choose
 - **Grid refresh interval:** Indicates the speed in milliseconds that the Results view refreshes.
 - **Maximum result rows to sort:** If the number of rows and results exceed this number, the column sorting in the result set is disabled.
 - **Results format:** These are the different formats that can be used to display the results in the editor.
 - **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.
 - **Save results to file:** Provides options that let you save the contents of the result sets to a file. You can also specify the file type, delimiter and text qualifier.

Specify SQL Templates Preferences

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

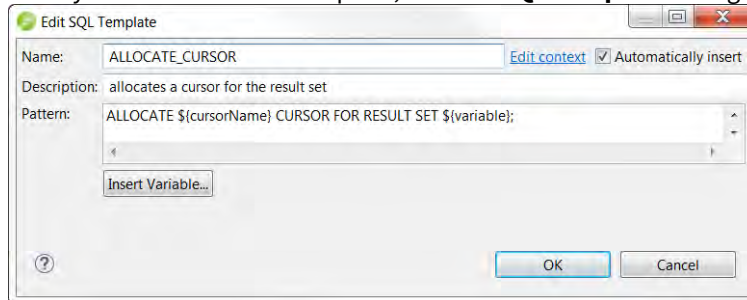
1. Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand **SQL Editor** and click **SQL Templates**.
The SQL Templates panel displays a list of all SQL code templates currently available. Additionally, when you select a template from the list, the **Preview** section displays the code block as it will appear when the

template is selected in SQL Editor.



2. 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.

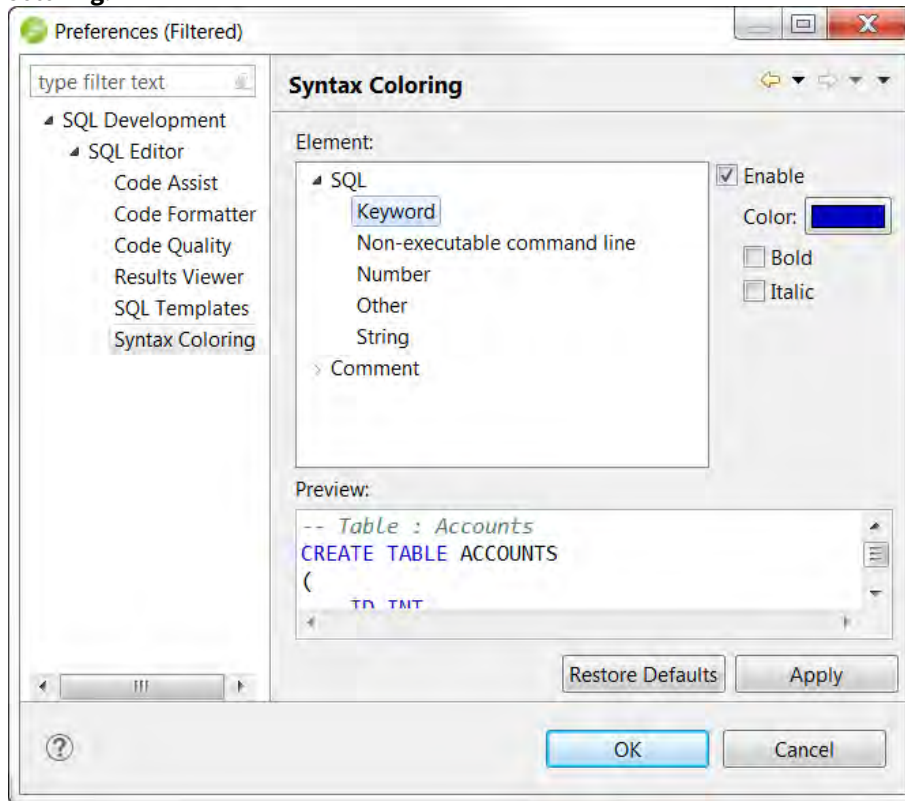


3. Enter a **Name**, **Description**, and **Pattern** in the fields provided, and click **OK**. If the template name doesn't match an existing SQL code template, your new template is added to the list, and will automatically be considered when the code assist function is executed in SQL Editor.
4. Select the **Use Code Formatter** check box to apply code formatting preferences to the specified template. See [Specify Code Formatter Preferences](#) for more information about setting code formatter preferences.

Specify Syntax Coloring Preferences

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

1. Select **Preferences > SQL Editor** and then in the **Preferences** dialog, expand **SQL Editor** and click **Syntax Coloring**.

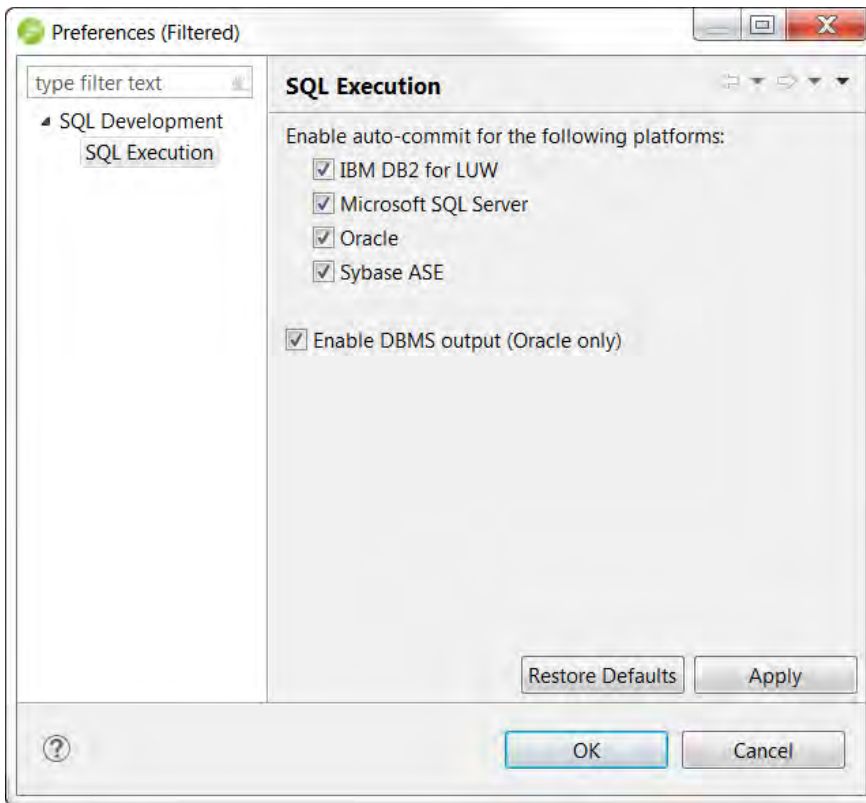


2. Use the tree view provided in the Element window to select the comment type or code element you want to modify. Select the options to the right-hand side of the window to modify it. The **Preview** window shows a piece of sample code that updates according to the changes you made.

Specify SQL Execution Preferences

The SQL Execution preferences you set determine how SQL is executed.

Select **Preferences > SQL Execution**.



The following describes the SQL Execution Preference options available.

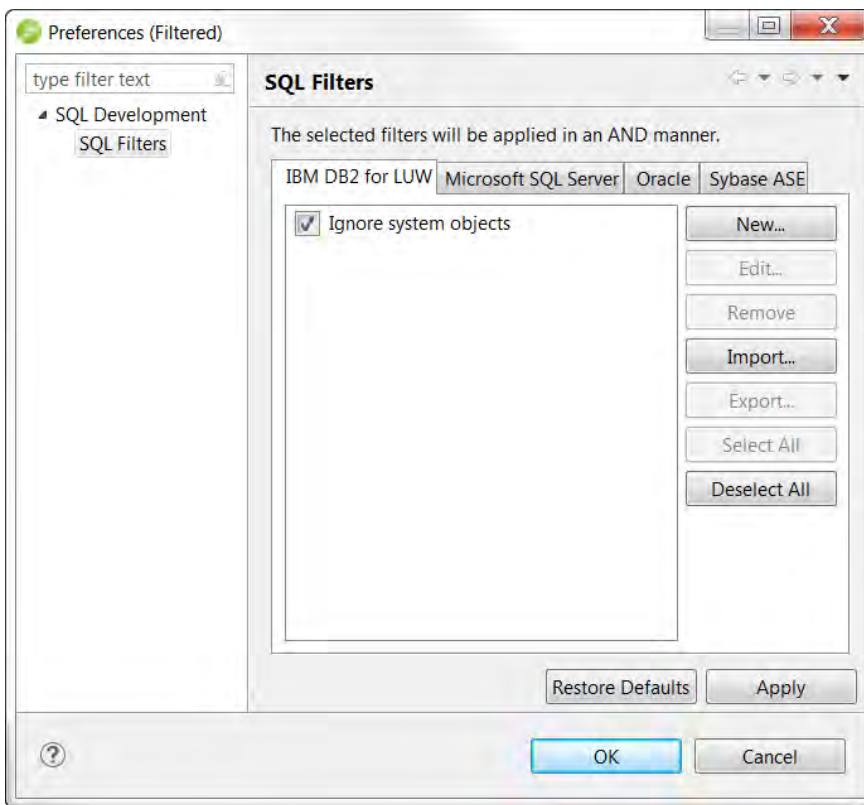
- **Enable auto-commit for the following platforms:** When disabled, the SQL is executed within transactions that must be manually committed.
- **Enable DBMS output (Oracle only):** When disabled, this omits the output statements that Oracle would otherwise display.

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

Specify SQL Filters Preferences

These are the set of controls that determine what objects are shown in the Data Source Explorer.

Select **Preferences > SQL Filters**.



Specify Profile Alerts Preferences

For information on specifying the Profile Alert Preferences, see [Configuring Profiling](#).

Specify Tuning Job Editor Preferences

For information on specifying the Tuning Job Editor Preferences, see [Configuring Tuning](#).

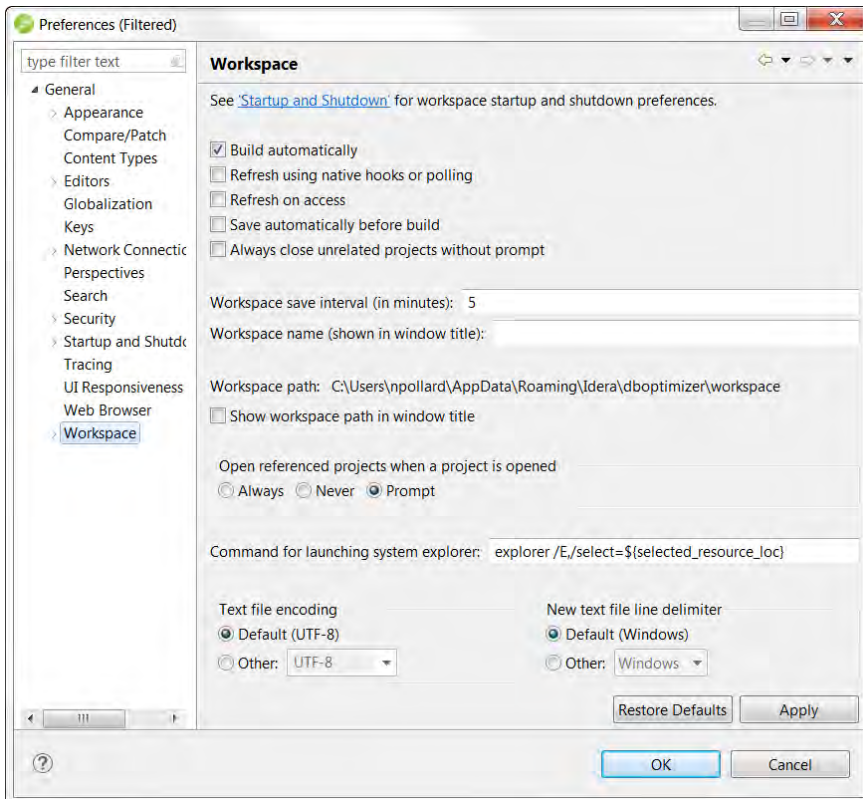
Specify VST Diagrams Tuning Preferences

For information on specifying the VST Diagrams Tuning Preferences, see [Configuring Tuning](#).

Specify File Encoding Preferences

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

Select **Preferences > General**, expand **General** and then click **Workspace** in the tree.



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.

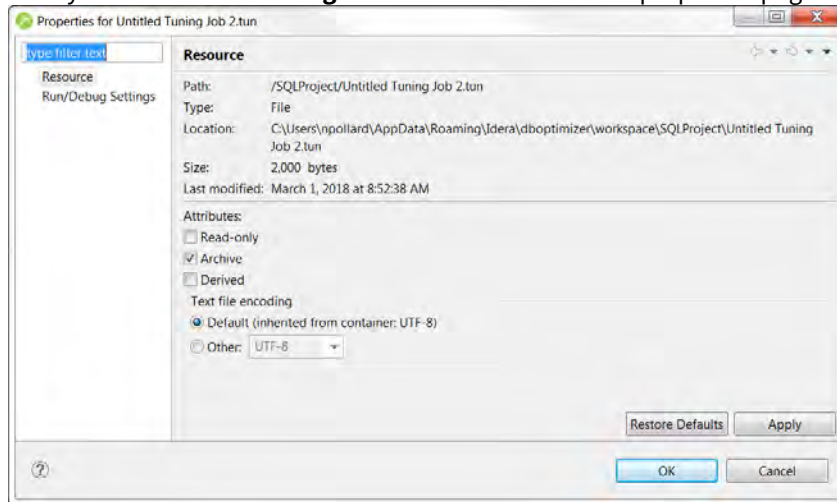
To change text file encoding in the development environment:

1. Select **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.
3. Click **Apply** to save your changes.

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

1. Right-click on the file, folder or project that you want to modify and choose **Properties**.

2. Modify the **Text file encoding** selection on the **Resource** properties page that appears.



Introduction to Database Tuning

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

- [Introduction to SQL Query Tuner's Tuner](#)
- [SQL Tuning Methodology](#)
- [SQL Tuner Overview](#)
- [What's Happening on the Databases?](#)
- [Tuning Example](#)
- [Finding and Tuning Problem SQL](#)

Introduction to SQL Query Tuner's Tuner

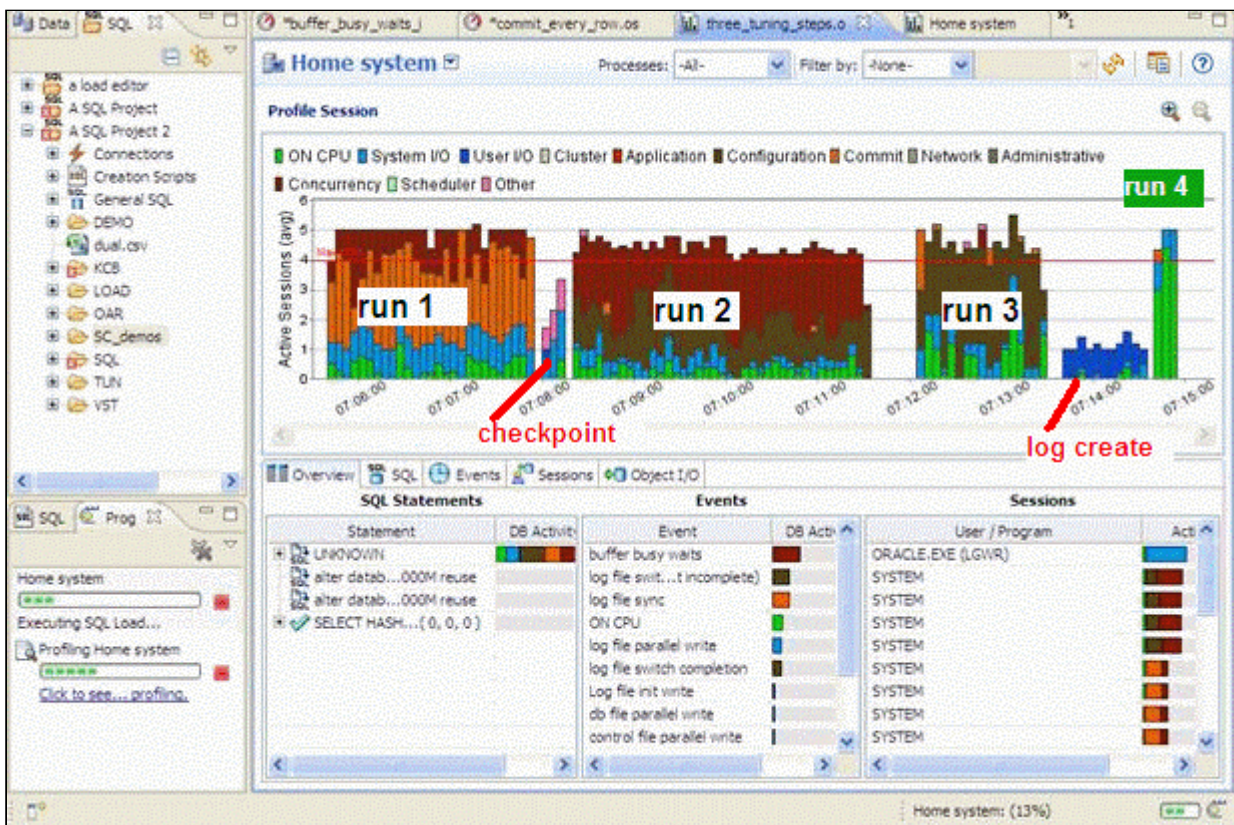
SQL Query Tuner's methodology grew out of the impossible predicament presented by the defacto method of database tuning. The standard method was trying to collect 100% of the statistics 100% of the time. Trying to collect all the statistics as fast as possible ends up putting load on the monitored database 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, the **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 run 4 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.

 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?

SQL Query Tuner'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?

SQL Query Tuner'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. SQL Query Tuner'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

SQL Query Tuner'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.

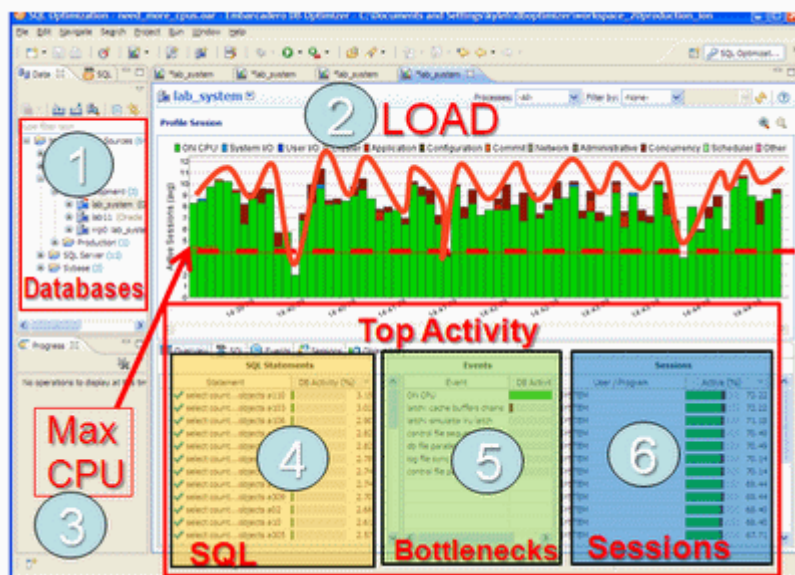
What's Happening on the Databases?

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 SQL Query Tuner'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](#).
2. **AverageActiveSessions(AAS)Loadofselecteddatabase.** For more information, see [Average Active Sessions \(AAS\) Load of selected database](#).
3. **MaximumCPUline.** For more information, see [Maximum CPU line](#).

4. **TopSQL.** For more information, see [Top SQL, Top Bottlenecks, and Top Sessions](#).
5. **TopBottlenecks.** For more information, see [Top SQL, Top Bottlenecks, and Top Sessions](#).
6. **TopSessions.** For more information, see [Top SQL, Top Bottlenecks, and Top Sessions](#).

Databases

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

AverageActiveSessions(AAS)Loadofselecteddatabase

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 SQL Query Tuner) 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.

MaximumCPUline

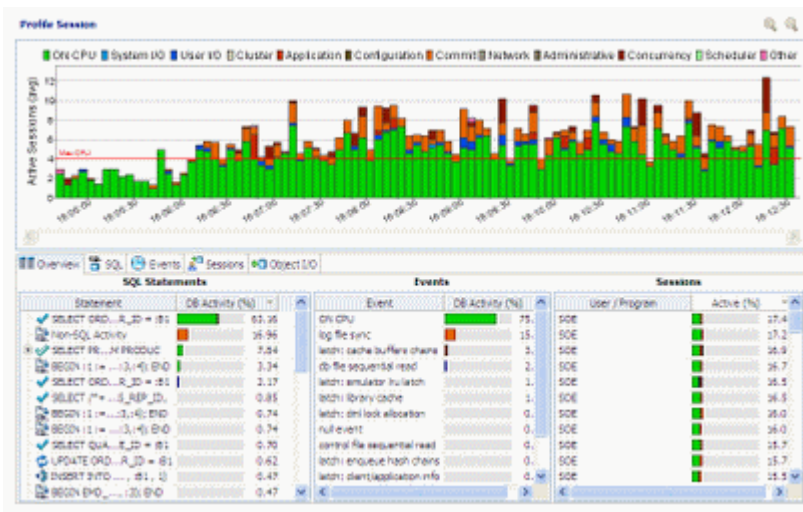
The line "Max CPU" represents the number of CPU processors on the machine. If we have one CPU then only one user can be running on the CPU at a time. If we have two CPUs then only 2 users can be on CPU at any instant in time. Of course users can go on and off the CPU extremely rapidly. When we talk about 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.

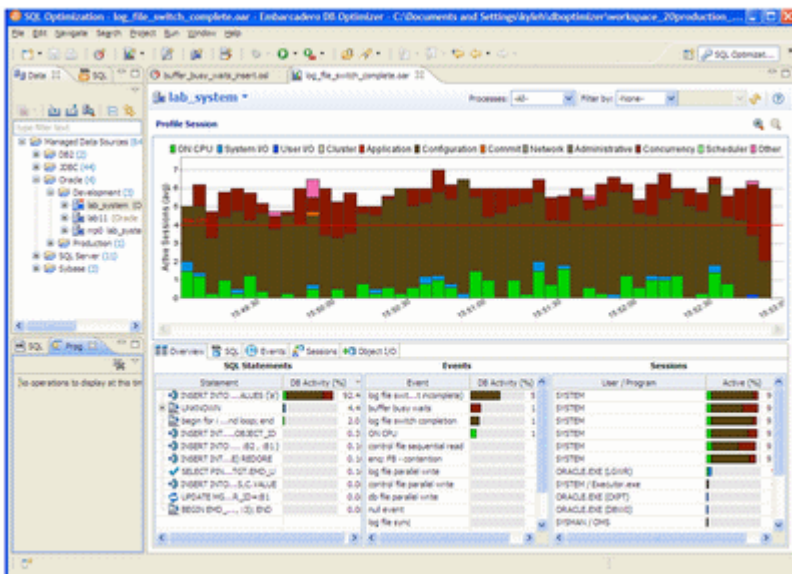
TopSQL, TopBottlenecks, and TopSessions

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:

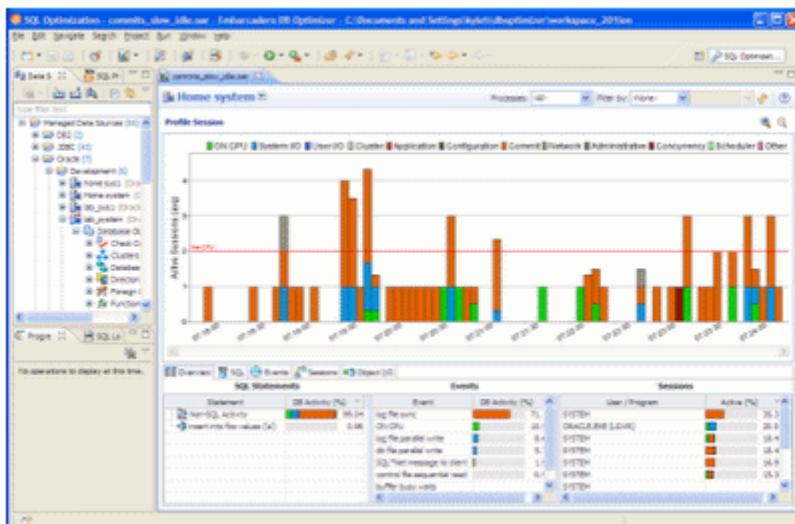
- [The Database is Hanging or the Application has Problems](#)
- [The Database Caused the Problem](#)
- [The Machine Caused the Problem](#)

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 SQL Query Tuner.



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 two 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 five samples over five seconds, thus bars are five 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;
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:

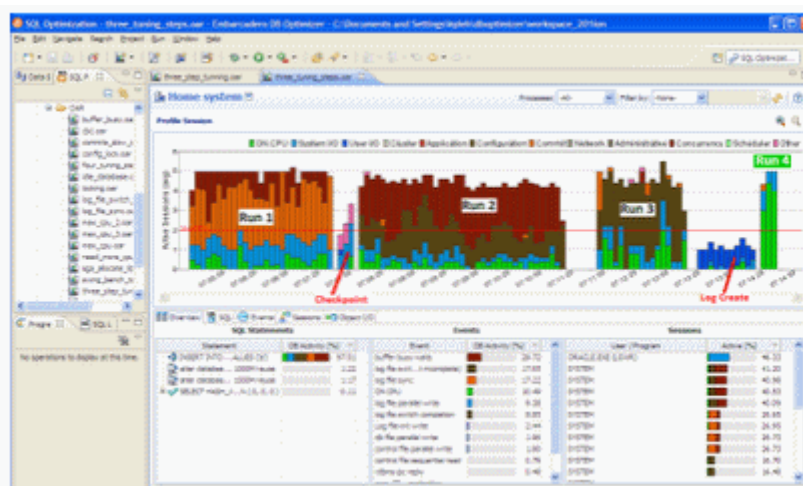
```
begin
  for i in 1..1000 loop insert into foo values
    ('a');
  -commit;
end loop; end;

/

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



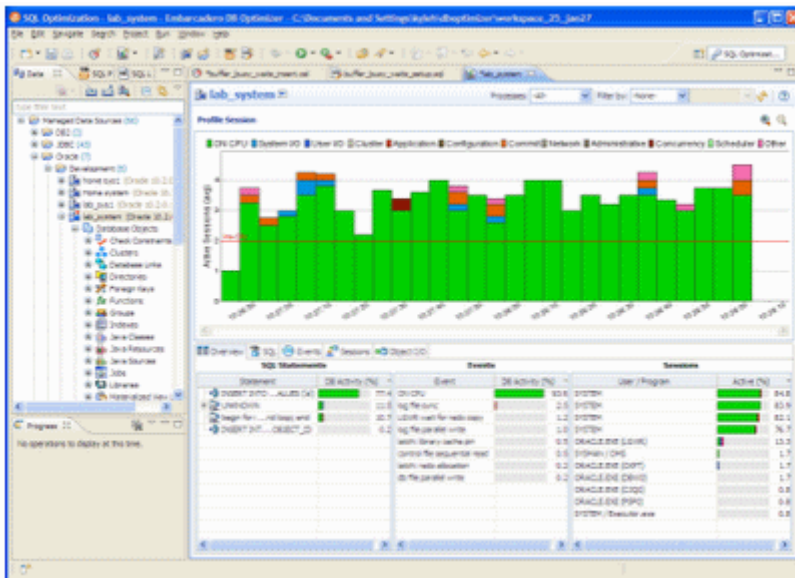
In the above SQL Query Tuner screen, the same workload was run 4 times. We can see that the time (width of the load) reduced, and the percent of activity on CPU increased.

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 tell us 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;

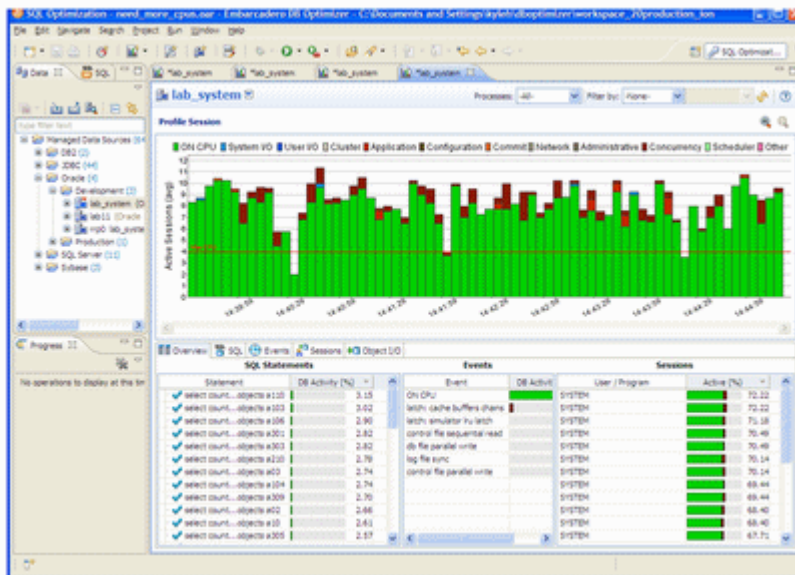
BEGIN
    for i in 1..8000 loop
        MY_IDX(i):=1;
    end loop;
    FORALL indx IN MY_IDX.FIRST .. MY_IDX.LAST
    INSERT INTO foo ( dummy )
    VALUES ( MY_IDX(indx) );
    COMMIT;

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 SQL Query Tuner makes it easy and clear, which can save weeks and months of wasteful meetings and debates. For example:

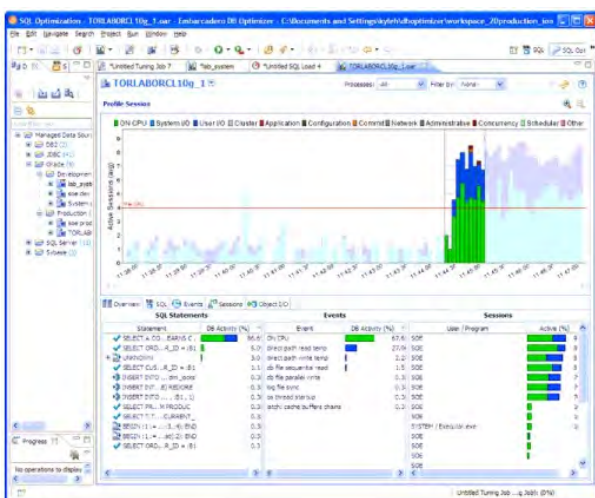


If we look in the bottom left, there is no SQL that takes up a significant amount of load, there is no outlier SQL that we could tune and gain back a lot of wasted CPU. We'd have to tune many SQL and make improvements on most of them to gain back enough CPU to get our load down below the max CPU line. 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 SQL Query Tuner provides a clear and easy presentation of the data and issues making solutions clear.

Finding and Tuning Problem SQL



SQL Query Tuner 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 is 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.

Using SQL Query Tuner

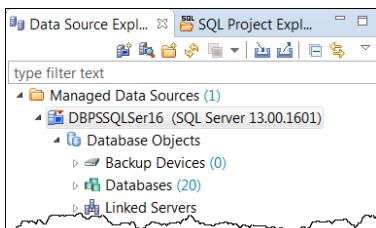
This section describes how to use the features of SQL Query Tuner to optimize your database operations. This section contains an overview of SQL Query Tuner functionality and also contains detailed instructions for:

- [Working with Data Sources](#)
- [Working with SQL Projects](#)
- [Creating and Editing SQL Files \(SQL Editor\)](#)
- [Executing SQL Files](#)
- [Troubleshooting](#)

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 SQL Query Tuner, a prompt appears and offers to populate Data Source Explorer from multiple sources on the system. This includes previously-registered data sources on other IDERA products, and third-party DBMS clients such as TOAD. If SQL Query Tuner 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 > IDERA > Data Sources > Previously Registered IDERA Data Sources (Registry)** command from the Main Menu



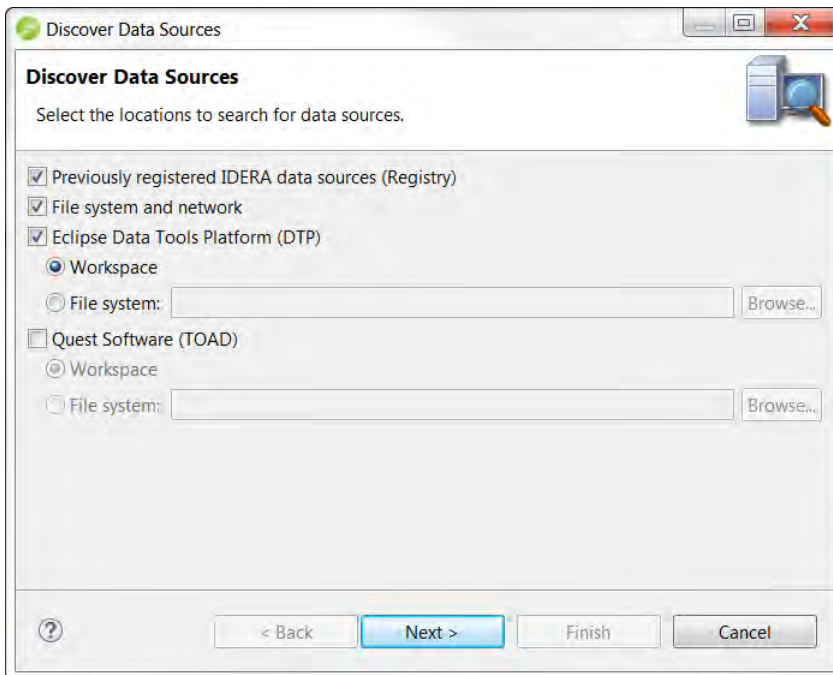
The Profiling Repository entries in the Data Source Explorer are available only when configured in the **Profile Configuration Dialog** for Oracle data sources only. These are saved profiling sessions that you can share with other SQL Query Tuner users. For information on configuring the data source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

Register data sources

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

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 SQL Query Tuner environment and click **Finish**. Data Source Explorer automatically populates with the new data source selections.

i 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.

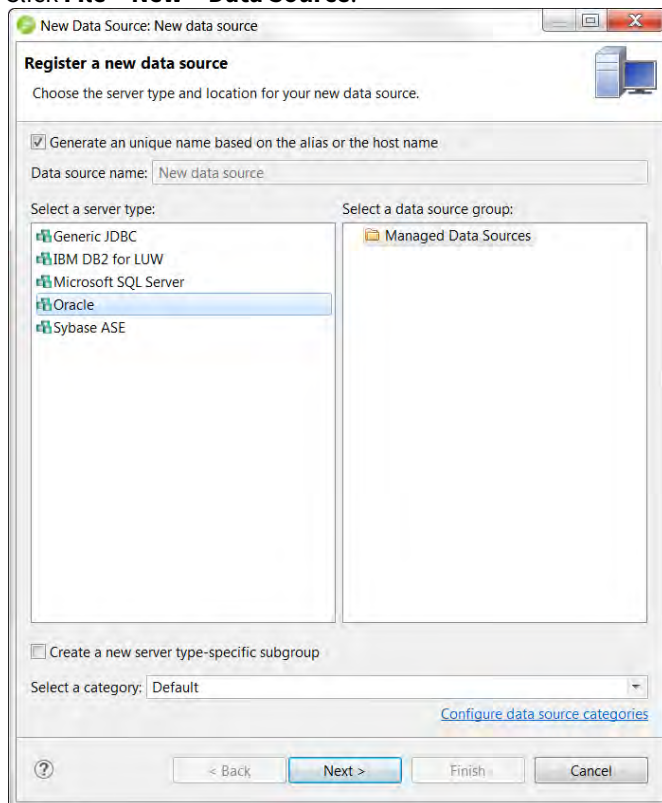
Once registered, the data source appears in the Data Source Explorer view. If you have created more than one workspace, they all share the same data source catalog.

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. SQL Query Tuner can encrypt and save user IDs and passwords to connect automatically.

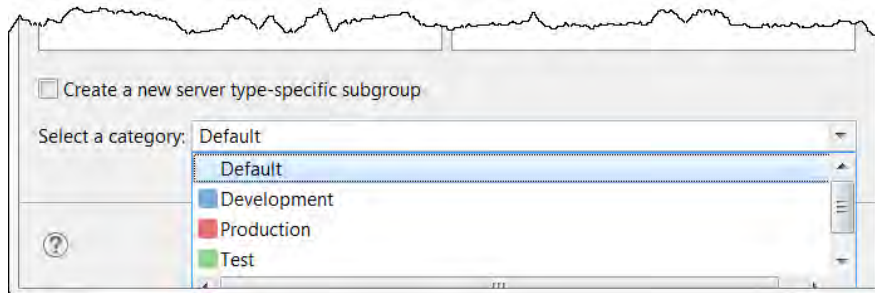
i In some cases, older versions of SQL Query Tuner and DB Artisan/Rapid SQL are not compatible with this version of SQL Query Tuner, and the methods listed above will not import these older data source catalogs. If you are experiencing difficulties, you can import the old data sources via the Windows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



Add a New Data Source

You can also add a new data source manually.

1. Click **File > New > Data Source**.

2. If you want the system to assign a name to the data sources, select **Generate a unique name based on the alias or the host name**.
If you want to enter the data source name, deselect **Generate a unique name based on the alias or the host name** and then in the **Data source name** field, enter the data source name.
3. In the **Select a server type** area, select the type you want to add.
4. In the **Select a data source group** area, select the data source group where you want the new data source to appear in the Data Source Explorer.
5. If you want to assign the new data source to a category, useful if you have a large number of data sources to manage, click the **Select a category:** list and choose one of the categories.

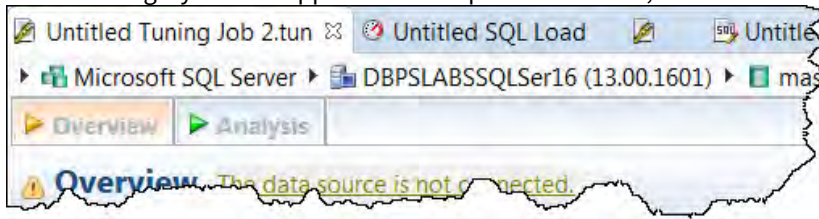


Categorized data sources appear with the color for the designated category on the bottom left of the data source icon in the bottom of such as  for the **Test** category and  for the **Production** category.

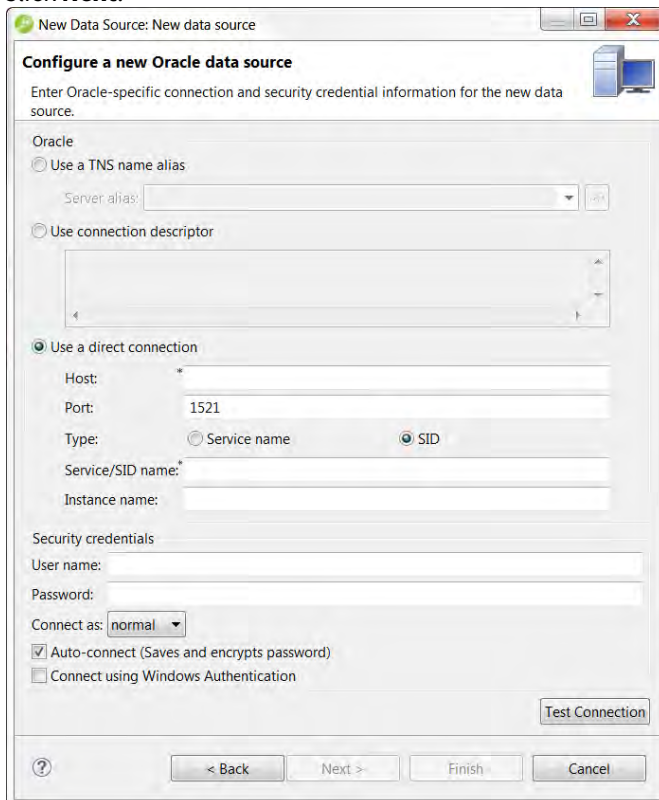
 For information on adding custom categories, see [Customizing Data Source Categories](#).

When you open a tuning job or SQL Editor window to create SQL for the categorized data source you will see

that the category color is applied to the top of the window, as follows.



6. Click **Next**.

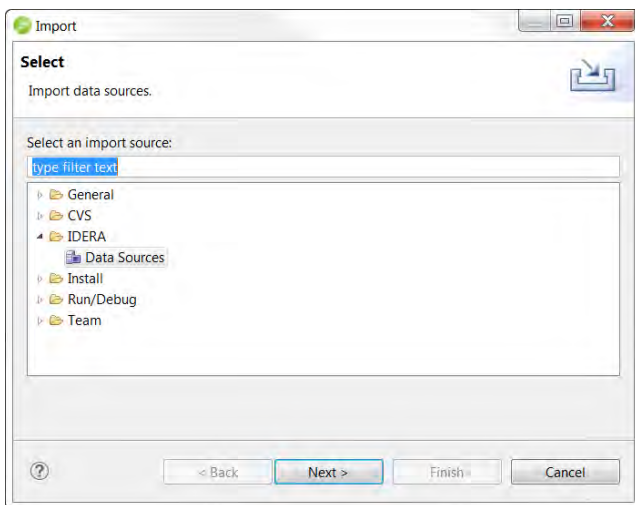


7. Complete the data source configuration and then click **Test Connection**. This will ensure your configuration is correct. If the connection test fails, make the necessary corrections and then click **Finish**. The new data source appears in the **Data Source Explorer**.

Import and Export Data Sources

Some IDERA products contain data source catalogs that are shared with SQL Query Tuner. In other words, instead of manually adding data sources to the environment, you can import an existing data source catalog from other IDERA products or third-party DBMS sources.

You import Data Sources via the **File > Import** command in the Main Menu, expanding the IDERA folder in the Data Source Selection tree, and choosing **Data Sources**.



The following types of sources can be imported to Data Source Explorer:

- Eclipse Data Tools Platform (DTP)
- Previously Registered IDERA Data Sources (File)
- Previously Registered IDERA Data Sources (Registry)
- Quest Software (TOAD)

Once a data source is registered, it automatically appears in Data Source Explorer. Connection parameters are stored locally, and SQL Query Tuner can be set to connect automatically each time you select the data source from the tree.

Conversely, you can also Export your current data source catalog to a file, which can then be imported into other instances of SQL Query Tuner via the **Previously Registered IDERA Data Sources (File)** option. This is performed using the **File > Export** command in the Main Menu, and then selecting **IDERA > Data Sources** from the tree view in the **Export** dialog.



To import data sources:

1. Select **File > Import**. The **Import** dialog appears.
2. Choose **IDERA > Data Sources** from the tree and click **Next**.
3. Choose a source from which you want to import the data sources. You can choose to import data sources from the DTP, TOAD, or an existing IDERA data source catalog stored in the Windows registry or as a file (created via the **Export** command). Click **Next**.
4. Specify the location of the import source and click **Finish**. Data Source Explorer is automatically populated with the new data sources.

To export data sources:

1. Select **File > Export**. The **Export** dialog appears.
2. Choose **IDERA > Data Sources** from the export tree and click **Next**.
3. Use the check boxes beside each listed data source to indicate which data sources you want to export. Click **Next**.
4. Click **Finish**.
5. The data sources are automatically exported in the form of an XML file. You can import this file to other instances of SQL Query Tuner via the **Import** command.

Categorize Data Sources

To make managing a large number of databases easier, you can assign a category to a data source. Categorized data sources appear with the color for the designated category on the bottom left of the data source icon in the bottom of such as  for the **Test** category and  for the **Production** category.

 For information on adding custom categories, see [Customizing Data Source Categories](#).

You can categorize a data source when you add a new data source (see [Add a New Data Source](#)) or by editing the properties of an existing data source.

1. In the **Data Source Explorer**, locate and then right-click the data source you want to add to a category.
2. Choose **Properties**.
3. From the **Category** list, choose the category you want and then click **OK**.

When you create a tuning job for the categorized data source you will see that the category color is applied to the top of the tuning job data source details.


Customizing Data Source Categories

SQL Query Tuner lets you customize your data source category scheme. A data source category has the following configurable components:

- **Category name.** The name displayed as a selection when selecting a category
- **Short Name.** An abbreviation shown in window components and icons
- **Color.** The color used to denote a categorized data source in the Explorer tree icons and window tabs.

To customize your datasource categories

1. Select **Preferences > Data Source**, and then select the **Datasource Group** panel.
2. Take one of the following actions:
 - Create a new category by clicking **Add** and selecting or providing a **Full name**, **Short name**, and **Color** combination.
 - Edit an existing category by selecting the category, clicking **Edit** and modifying the name and color combination.

 The short name for a category cannot be edited.

- Delete an existing category by selecting the category, clicking **Delete** and verifying the deletion at the prompt. .

Browse a Data Source

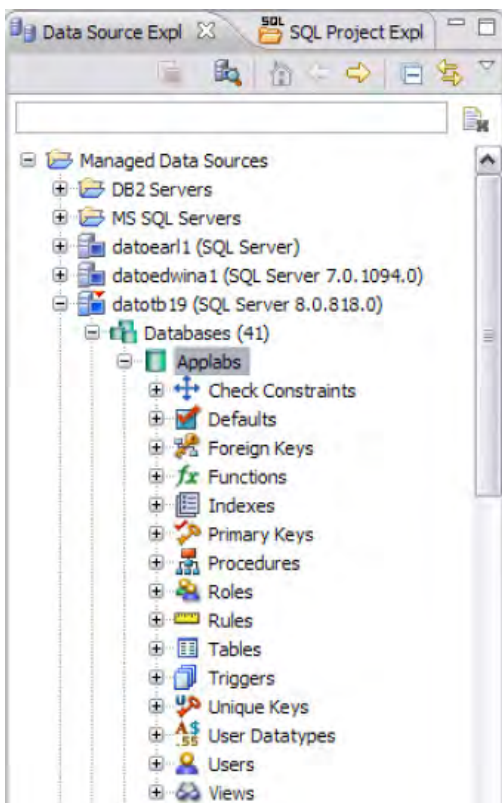
You can drill down in the Data Source Explorer tree to view registered databases on a server, and view tables, and other objects in a database. Additionally, you can view the structure of individual objects such as the columns and indexes of a table. Right-click the object for a menu of available commands, 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, SQL Query Tuner 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.

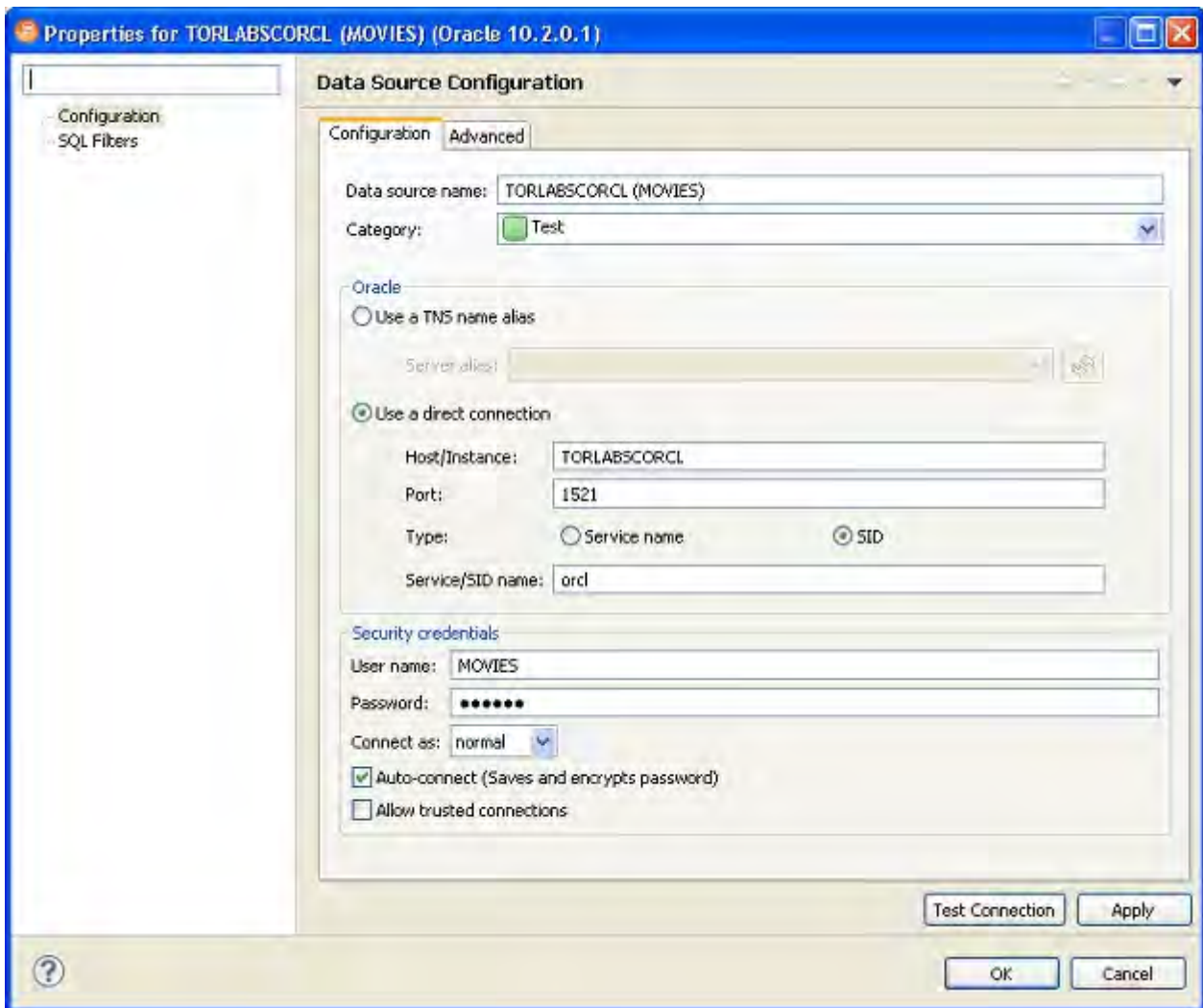
You can turn off the Auto Connect feature by right-clicking on a specified data source and toggling the Connect on Expand option. By default, when Connect on Expand is active, SQL Query Tuner automatically attempts to connect to the server each time you browse a data source.

View Database Object Properties

All objects in Data Source Explorer contain properties as they relate to the SQL Query Tuner application.



SQL Query Tuner 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 properties are accessed by right-clicking a data source in Data Source Explorer. The dialog displays properties with regards to Configuration and SQL Filters.

The Configuration node provides information about 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](#).

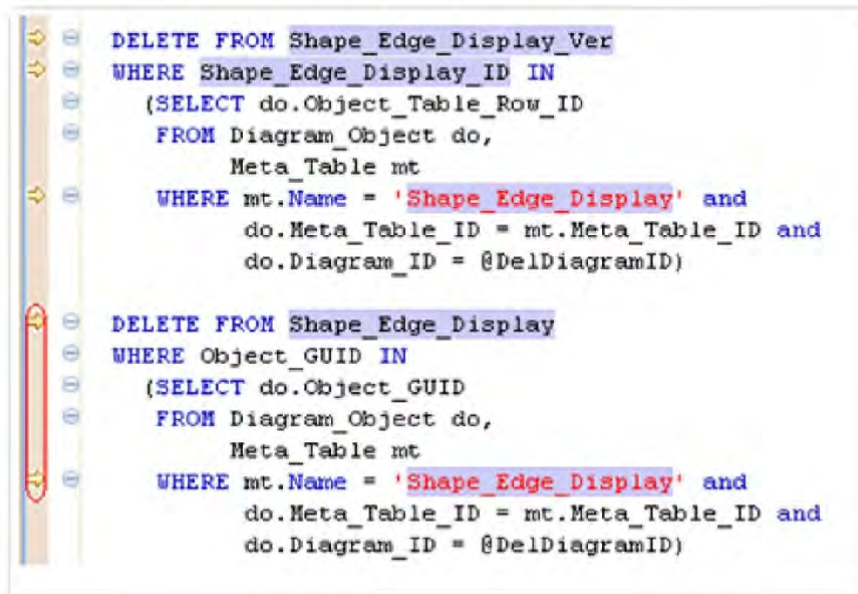
The SQL Filter node enables a developer to place filters on data source objects that appear in the Database Explorer. For more information, see [Filter Database Objects](#).

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 [Specify Data Source Indexing Preferences](#).

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.
- 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.



```

DELETE FROM Shape_Edge_Display_Ver
WHERE Shape_Edge_Display_ID IN
(SELECT do.Object_Table_Row_ID
FROM Diagram_Object do,
Meta_Table mt
WHERE mt.Name = 'Shape_Edge_Display' and
do.Meta_Table_ID = mt.Meta_Table_ID and
do.Diagram_ID = @DelDiagramID)

DELETE FROM Shape_Edge_Display
WHERE Object_GUID IN
(SELECT do.Object_GUID
FROM Diagram_Object do,
Meta_Table mt
WHERE mt.Name = 'Shape_Edge_Display' and
do.Meta_Table_ID = mt.Meta_Table_ID and
do.Diagram_ID = @DelDiagramID)

```

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 SQL Query Tuner development environment.
- **Data Source specific filters** affect only the specified data source for which they are defined.

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 Data Source-Specific Object Filters](#)
- [Define Global Database Object Filters](#)

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.
 - Click **OK**. The filter property specification is added to the Filter Template.
 - 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 SQL Query Tuner development environment. When you create and apply a global filter to a platform vendor in SQL Query Tuner, 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.
 - Click **OK**. The filter property specification is added to the Filter Template.
 - 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.

 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.

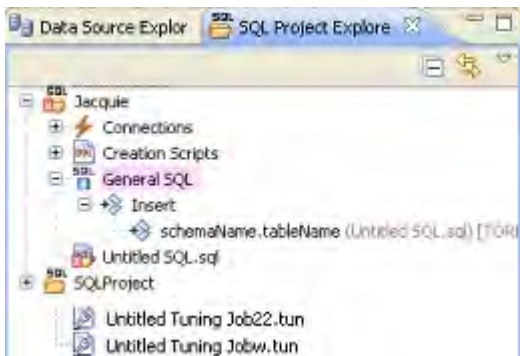
Drop a Database Object

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 SQL Query Tuner are stored under the Workspace directory you specified at the prompt when SQL Query Tuner was started. The directory and files can be shared, and other tools can be used to work on the files, outside the SQL Query Tuner 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 SQL Query Tuner 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 SQL Query Tuner 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.
- 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 SQL Query Tuner.

i 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.)

i Physically, the projects and files you create as you work in SQL Query Tuner 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 SQL Query Tuner development environment.

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.

i 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.
- Click **Finish**. The import process moves all selected folders and files into Project Explorer and thus into the SQL Query Tuner development environment.

 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.

Search a Project

You can search for certain text within a project and view the search results in the Search view on the Workbench.

To use the Search feature

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.
- Click **Search**. The results of your search are generated in the Search view on the Workbench.

Delete a Project

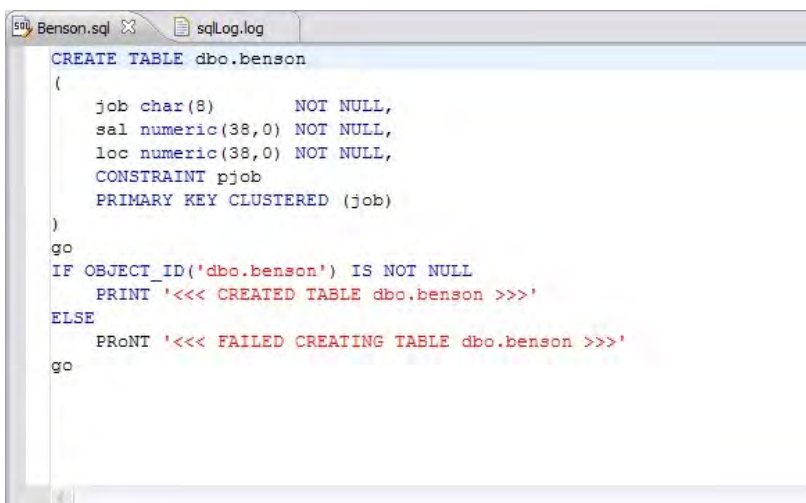
You can delete a project by right-clicking its folder in the SQL Project Explorer and selecting **Delete**.

When you delete a project, SQL Query Tuner 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 SQL Query Tuner 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.



```

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](#).
 - **Code templates.** Templates for creation of tables, procedures, etc. For more information, see [Understanding SQL Templates](#).
 - **Hyperlinks.** For more information, see [Understanding Hyperlinks](#).
 - **Semantic validation.** For more information, see [Semantic Validation](#).
 - **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](#).
- **Code correction and transformations.** For more information, see [Examples of Transformations and SQL Query Rewrites](#).
- **Object indexing.** For more information, see [Specify Data Source Indexing Preferences](#).
- **SQL Project Explorer.** For more information, see [Working with SQL Projects](#).

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, SQL Query Tuner 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.

Semantic 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 Comma	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](#).

Create an SQL File

To create an SQL file

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

 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

To open an existing SQL file

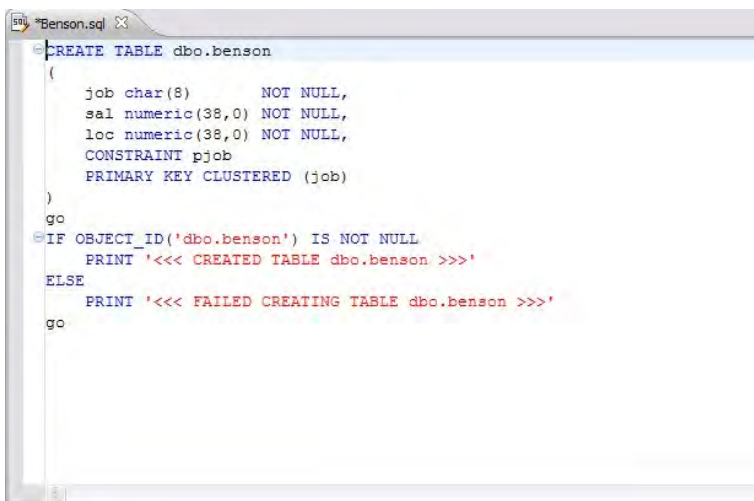
1. Open the SQL project containing the file, or that you want to contain the file.
2. If necessary, add the file to the project. See [Add Files to a Project](#).
3. In the SQL Project Explorer, double-click the file to open it in SQL Editor.

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 SQL Query Tuner, as well, but these are handled by other editors.

For example, if a text file is opened in the Workbench, SQL Query Tuner 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.




```

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
  
```

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.

 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](#)
- [Understanding Code Assist](#)
- [Understanding Hyperlinks](#)
- [Understanding Code Formatting](#)
- [Understanding Code Folding](#)
- [Understanding Code Quality Checks](#)
- [Understanding SQL Templates](#)

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.

```

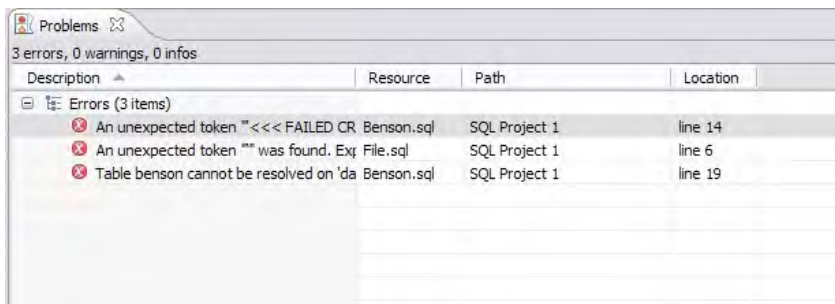
CREATE TABLE dbo.benson
(
    job CHAR (8) NOT NULL,
    sal NUMERIC (38, 0) NOT NULL,
    loc NUMERIC (38, 0) NOT NULL,
    CONSTRAINT pjob PRIMARY KEY CLUSTERED (job)
)

go
IF OBJECT_ID('dbo.benson') IS NOT NULL
    PRINT '<<< CREATED TABLE dbo.benson >>>'
ELSE
    PRINT '<<< FAILED CREATING TABLE dbo.benson >>>'

go
SELECT *
FROM tbo.benson;

```

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



Right-click the error and select **Go To** in order to find the error. SQL Query Tuner opens and navigates to the specific line of code containing the specified error.

- i** Automatic error detection functions, such as syntax checking and semantic validation are suspended if #define or #include directives are detected in an editor window. SQL Query Tuner does not perform #define/#include substitutions on execution.

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 nested within DDL statements. This functions in the same manner as code assist for statements that are not nested, 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

 Function suggestions are only available for Oracle.

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.


Statement Syntax	Object Suggestion Behavior
A period is typed.	<p>If the word prior to the period is a name of a table or view, a list of columns is displayed.</p> <p>If the word prior to the period is a schema name, a list of table names is displayed.</p> <p>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</p>

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.

 On a per platform basis, auto-suggestion behavior may vary.

To modify object suggestion parameters, including setting it from automatic to manual, see [Specify Code Assist Preferences](#).

You can speed up the performance of the code assist functionality by enabling data source indexing either when you connect to the data source, see [Working with Data Sources](#) or on the Preferences page, see [Specify Data Source Indexing Preferences](#).


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, <https://www.idera.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.

 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 [Specify Data Source Indexing Preferences](#).

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 [Specify Code Formatter Preferences](#) 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

Custom Code Formatting Parameter	Value (if applicable)
Number of new lines to insert	1
Indent Size	5
Right Margin?	80
Stacked parentheses when they contain multiple items?	No
Stacked parentheses when the list contains __ or more items.	_
Indent size?	5
New line after first parentheses?	No
Indent content of conditional and looping constructs?	Yes
Number of new lines to insert?	1
Indent size?	5

```

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
Stacked parentheses when the list contains ___ or more items.	2
Indent size?	2
New line after first parentheses?	Yes

Custom Code Formatting Parameter	Value (if applicable)
Indent content of conditional and looping constructs?	--
Number of new lines to insert?	1
Indent size?	5

```
B'1=file.sql
```

```
  Begin
```

```
  If x=S
```

```
    SELECT
```

```
      apple ,
      pear ,
      orange Big Orange' ,
      strawberry ,
      orchard name ,
      owner
```

```
    FROM
```

```
      fruit F ,
      orchard 0
```

```
    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.

```

CREATE DEFAULT ET_FALSE AS 0
go
IF OBJECT_ID('PERFCNTR_KS.ET_FALSE') IS NOT NULL
go
CREATE TABLE PERFCNTR_KS.ADDRESS_ROLE
(
    ADDRESS_ROLE_ID    numeric(10,0) IDENTITY(20000,1),
    NAME               varchar(128) NOT NULL,
    DESCRIPTION        varchar(255) NULL,
    LIST_ORDER         int          NOT NULL,
    IS_DEFAULT         bit          NOT NULL,
    IS_SYSTEM_REQUIRED bit          NOT NULL,
    ROWTIMESTAMP       datetime     CONSTRAINT DF_ADDRESS_R_ROWTI__1BF
    CONSTRAINT ADDRESSROLEPK
    PRIMARY KEY CLUSTERED (ADDRESS_ROLE_ID)
)
go
EXEC sp_bindefault 'PERFCNTR_KS.ET_FALSE', 'ADDRESS_ROLE.IS_DEFAULT'
go
EX sp_bindrule 'PERFCNTR_KS.TRUEORFALSE', 'ADDRESS_ROLE.IS_DEFAULT'

```


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 employeee,customerc,sales_orders 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 employeee,customerc,sales_orders WHERE e.employee_id = c.salesperson_id AND s.customer_id = c.customer_id</pre> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p> This code quality check is valid for Oracle join conditions.</p> </div>
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(+)ANDc.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>

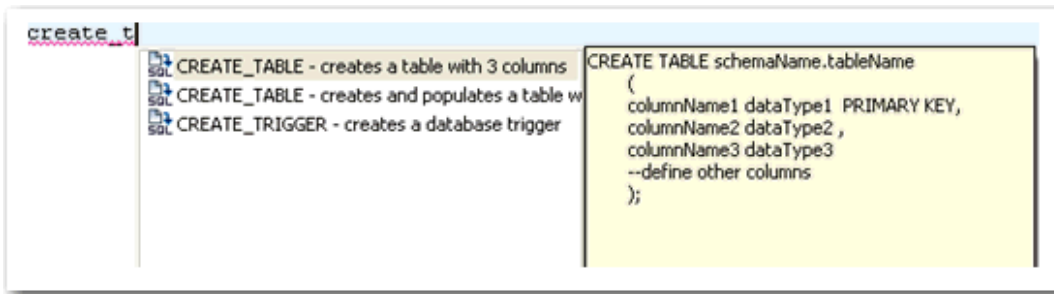
Code Quality Check Type	Definition
Transitivity issues	<p>The performance of statements can sometimes be improved by adding join criteria, even if a join is fully defined. If this alternate join criterion is missing in a statement, it can restrict the selection of an index in Oracle's optimizer and cause 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. SQL Query Tuner 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>

Code Quality Check Type	Definition
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>
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

SQL Query Tuner 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.

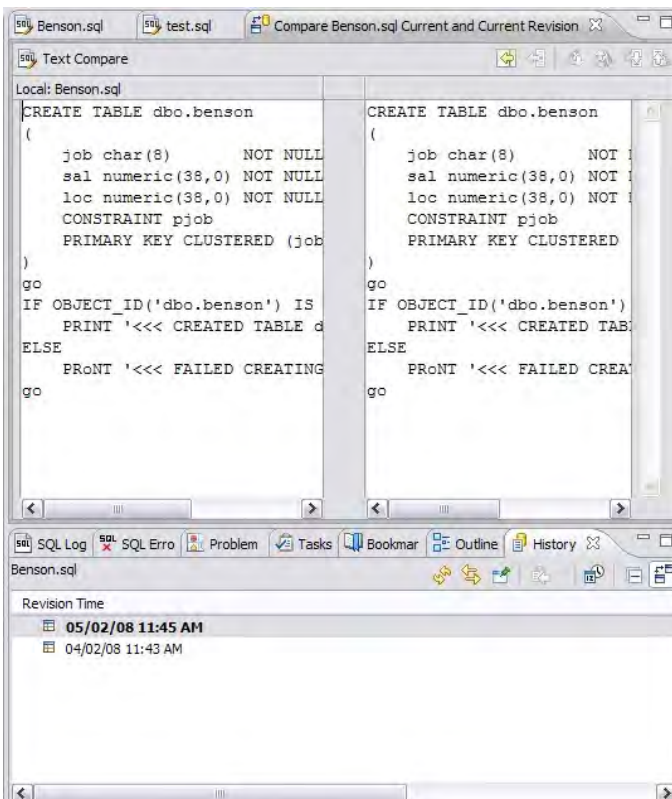


Code templates are available for DML, ALTER, DROP, CREATE, and platform specific commands.

A comprehensive set of DDL/DML templates are available, with statement alternatives varying by DBMS and specific DBMS versions. You can modify and create new templates via the SQL Templates panel on the Preferences dialog. See 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.

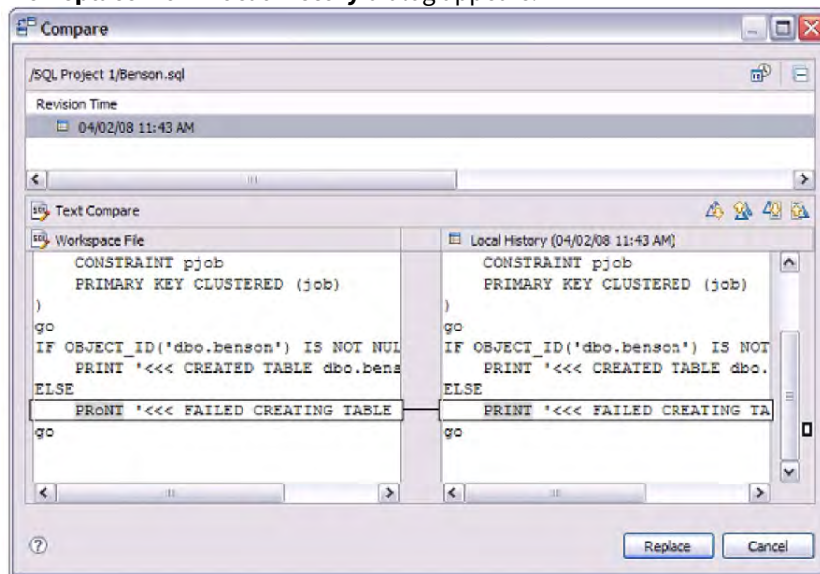
- Double-click a time in the **History** view to access the **Text Compare** panel. It displays the text of the file after the change occurred at the time indicated in the **History** view.

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 SQL Query Tuner's last recorded history point.

Delete an SQL File

To delete an SQL file, right-click its icon in the SQL Project Explorer, and then select **Delete**. This process removes the file from both the SQL project and the file system.

Executing SQL Files

SQL Query Tuner 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**. SQL Query Tuner 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 [Specify SQL Execution Preferences](#) 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. SQL Query Tuner executes the transaction on the data source you specified.

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

 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**. SQL Query Tuner commits the transaction on the data source you specified.

 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](#)
- [Configure an SQL Session](#)
- [Execute SQL Code](#)
- [View and Save Results](#)

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 does 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.

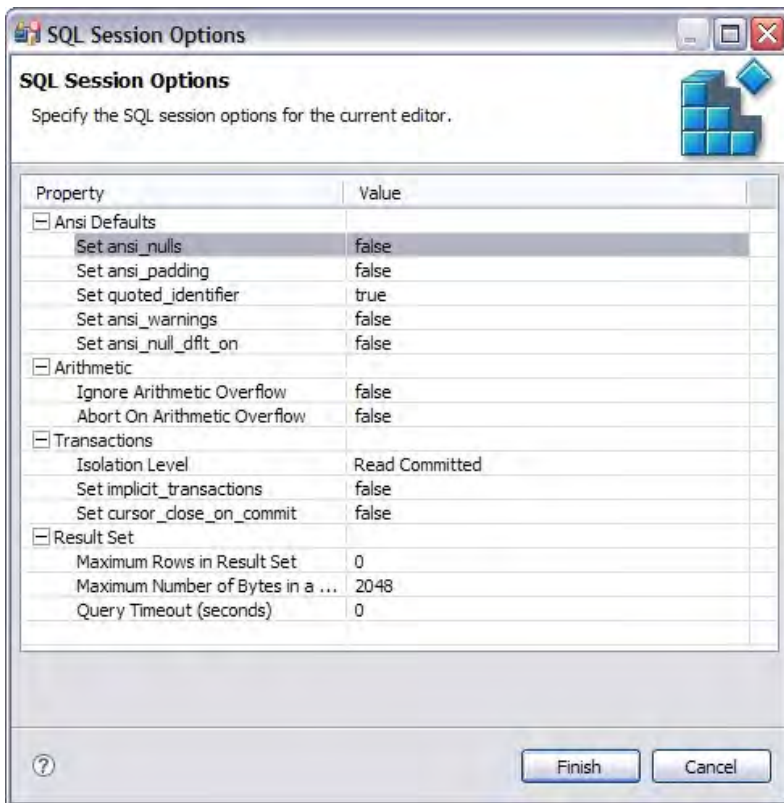
To change or associate a registered data source with a SQL file

- 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, SQL Query Tuner's behavior differs when the connection is made.

i 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 SQL Query Tuner 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 SQL Query Tuner 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 SQL Query Tuner 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. SQL Query Tuner 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. SQL Query Tuner 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. SQL Query Tuner commits the transaction on the data source you specified.

i You can set transactions to auto-commit prior to execution on the SQL Execution node of the Preferences panel in SQL Query Tuner.

View and Save Results

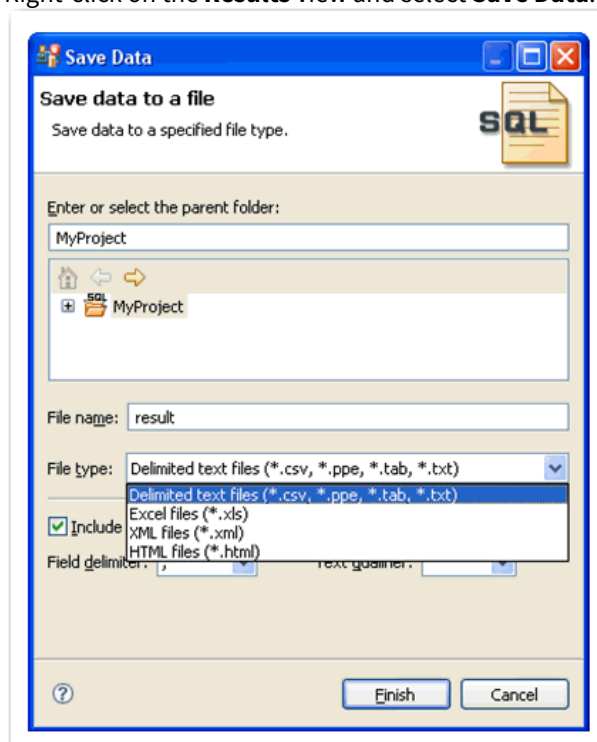
Once a file has been executed, the results are displayed in the Results view. Here, you can examine the outcome of the execution process, as well as save the results in other file formats, as needed.

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

SQL Query Tuner 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 '<<< CREAT...	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 SQL Query Tuner. 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 PRoNT.	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 SQL Query Tuner detects, as well as locate other problems that have similar attributes.

Description	Resource	Path	Location

See Also

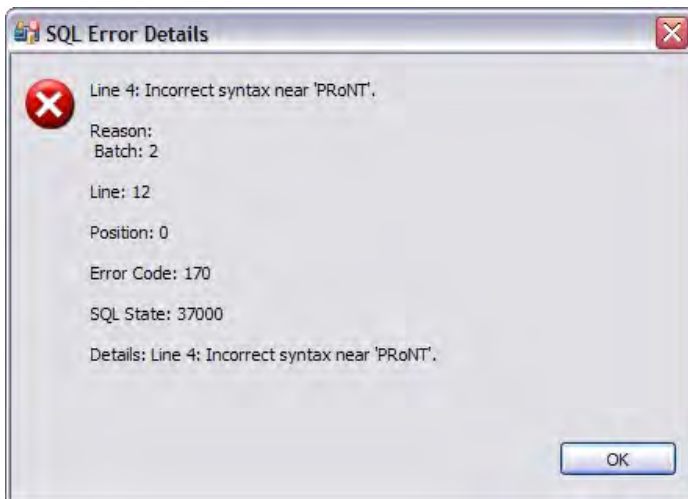
- [View Log Details](#)
- [Maintain Logs](#)
- [Filter Logs](#)
- [Import and Export Error Logs](#)
- [Find and Fix SQL Code Errors](#)
- [Find and Fix Other Problems](#)

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 SQL Query Tuner. 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.

- Right-click in the SQL Log view 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, SQL Query Tuner will automatically generate a new .log file in the .metadata subfolder.


i 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 Errors views to limit searches when troubleshooting and pinpointing specific processes within the system.

To filter the SQL Log


1. On the SQL Log view, 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.

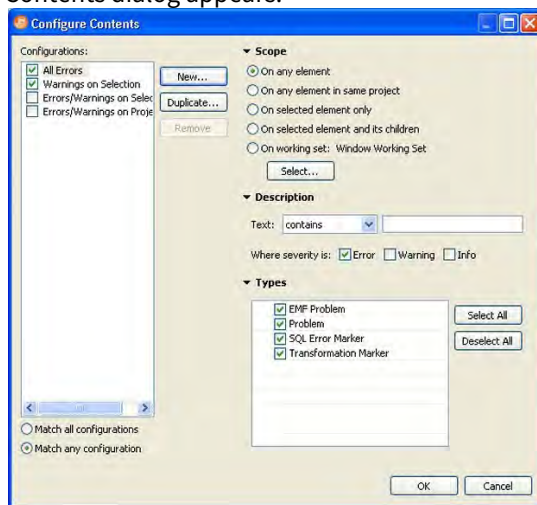



2. In the SQL Statement Types frame, select **Successful** or **Failed** to filter by the type of Error Log entries.
3. Select **Limit display statements** to indicate a maximum limit of the number of entries displayed in the Error Log, and enter the maximum entry value in the corresponding field.
4. 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.
5. 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 view and filter the Problems log

1. On the Main Toolbar, click **Window > Show View > Other > General > Problems**.

2. On the Problems view, select the options icon  and choose **Configure Contents**. The Configure Contents dialog appears.



The Configure Contents dialog enables you to create multiple filter profiles that you can apply to the log by clicking the options icon , clicking Show, and then choosing the filter to apply. The Configurations panel on the left-hand side of the dialog displays all existing filter profiles stored in the Workspace. Selecting a configuration 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 criterion 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 Scope area 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 SQL Query Tuner. 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	<p>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.</p>

Profile Criteria	Description
Problem Severity	In the Where severity is area, 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 Types list on the right-hand side of the dialog enable you to filter problems by type. For example, 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 **Show** 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

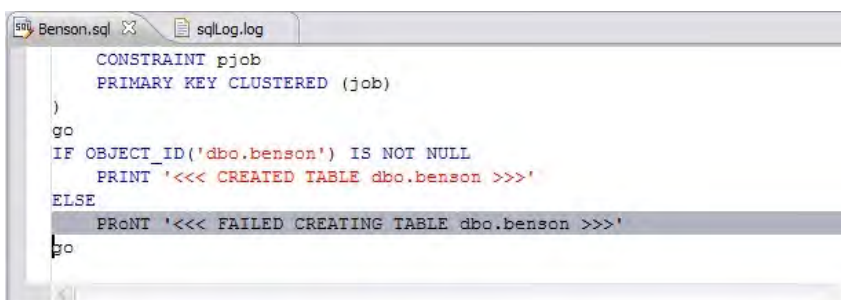
Right-click the SQL Log view, and then choose **Export Log**. The log is saved in the specified directory path with a .log extension.

To import the Error Log

Right-click the SQL Log view, and then 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.



The screenshot shows a SQL script editor with two tabs: 'Benson.sql' and 'sqlLog.log'. The script content is as follows:

```

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 error message 'FAILED CREATING TABLE dbo.benson' is highlighted in a grey box.

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.

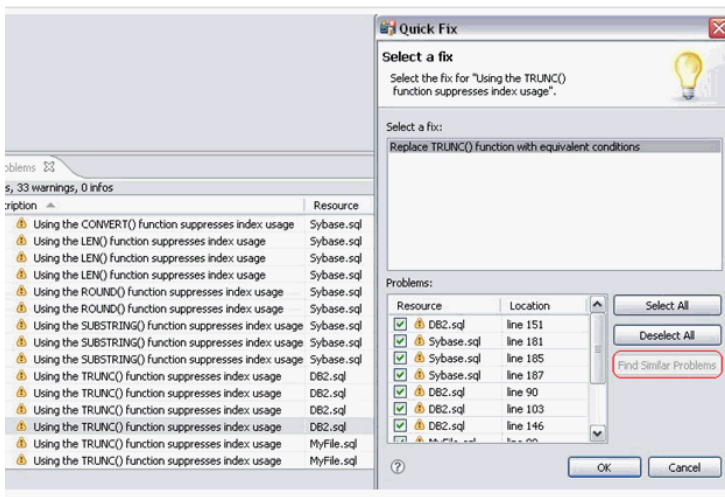
Find and Fix Other Problems

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 SQL Query Tuner 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**. SQL Query Tuner 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**. SQL Query Tuner attempts to resolve all of the specified issues.

Using Profiling

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

- [Understanding Profiling](#)
- [Understanding the Interface](#)
- [Running a Profiling Session](#)
- [Configuring Profiling](#)

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 SQL 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 is 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

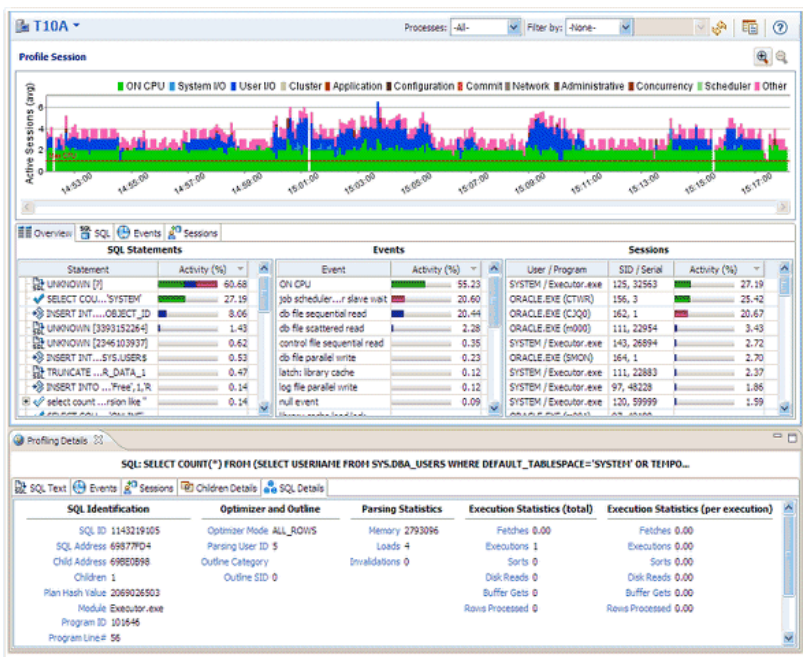
Problems can come from any one or more of four areas:

- Machine CPU or Engine, slow disks (network)
- Application locks, invalid SQL
- Database cache sizes, log files, etc
- Inefficient SQL

Details of profiling sessions can be saved to an .oar file, which you can access through the SQL Project Explorer.

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.

- i** 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 user's 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**

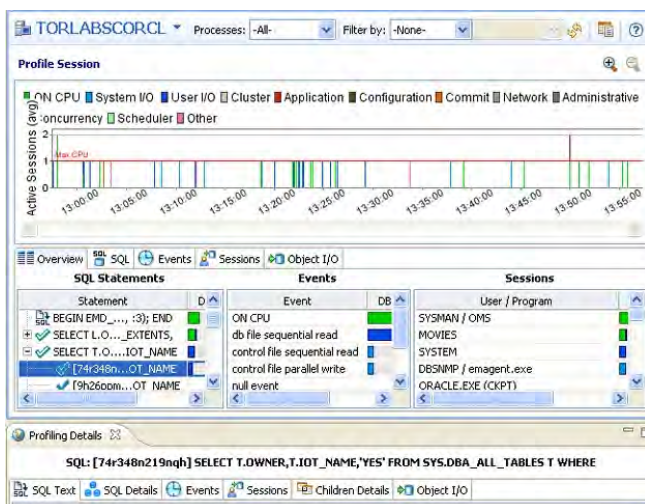
2. [Work with Session Results](#)
3. [Creating Profiling Reports](#)
4. [Saving Profiling Sessions](#)
5. [Import Statements to Tuning](#)

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](#).

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](#) 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 **Profile Configurations** dialog, select the configuration to use for this profiling session. If you haven't already created a profile, see [Building profiling configurations](#) for a description of the profiling configuration options you can choose.


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 [Working with Session Results](#).

To stop a profiling session

You can stop a profiling session at any time by clicking the **Stop** button [] in the upper left-hand side of the Profile Session screen or by clicking the Stop button in the Progress Window.

Executing a Session from the Command Line

 This is not supported when using SQL Query Tuner InstantOn.

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.

Working 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](#)
- [Filtering results](#)
- [Analyze the Load Chart](#)
- [Analyze the Top Activity section](#)
- [Analyze profiling details](#)

Opening an Existing Profiling Session

Saved profiling session data is stored in a SQL Project. You can find profiling sessions saved as .oar files in the SQL Project Explorer.

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

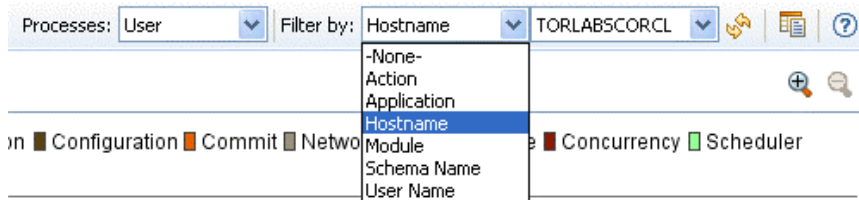
Filtering results

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:** Application, Creator ID, Cursor Name, Package Name, Statement Type, and User Name.

- **Microsoft SQL Server:** Application, Command, Database, Hostname, NT domain, Net Address, and User Name.
- **Oracle:** Processes (Background or User), Action, Application, Hostname, Module, Schema Name, and User Name. When profiling a RAC, there is also an instance filter that appears to let you limit the profiling results shown to a specific instance.

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



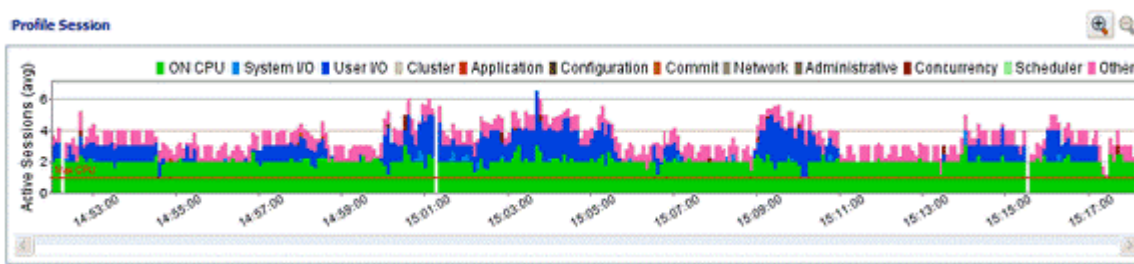
To filter profile editor results

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.
3. Click **Refresh** [🔄] to update the profiling details.
The profiling editor is updated to show only results associated with your choice.

i Select **-None-** from the **Filter by** list to restore the unfiltered results.

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 previous screenshot 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 CPU 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 SQL Query Tuner) 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 SQL Query Tuner 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 **EngineCount** dialog that appears select **Useacustomvalue**, enter a new value, and then click **OK**. The AAS or Load Chart **MaxCPU** or **MaxEngine** 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.

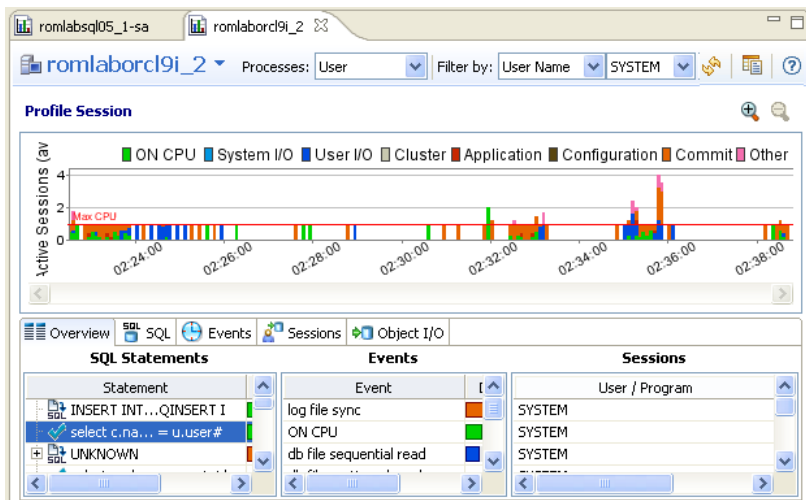
DBMS	Wait Event Category
Oracle	On CPU, System I/O, User I/O, Cluster, Application, Configuration, Commit, Other

* 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](#).
- The top **Events** tab displays information about wait events profiled by the execution process. For more information, see [Top Events tab](#).
- The top **Sessions** tab displays information about sessions profiled by the execution process. For more information, see [Top Sessions tab](#).
- The top **Blockers** tab displays information about blocking sessions. For more information, see [Top Blockers tab](#).
- The top **Object/I/O** tab displays information about the I/O profiled by the execution process. For more information, see [Top Object I/O tab](#).
- The top **Procedures** tab displays information about procedures observed during profiling. For more information, see [Top Procedures tab](#).

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.

Statements

Statements can be grouped by type by right-clicking the view and selecting **Organize > By Type**.

i 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, DELETE, and MERGE statements.

Group	Description
UNKNOWN	Statements that are not recognized by the application. SQL Query Tuner 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.

 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:

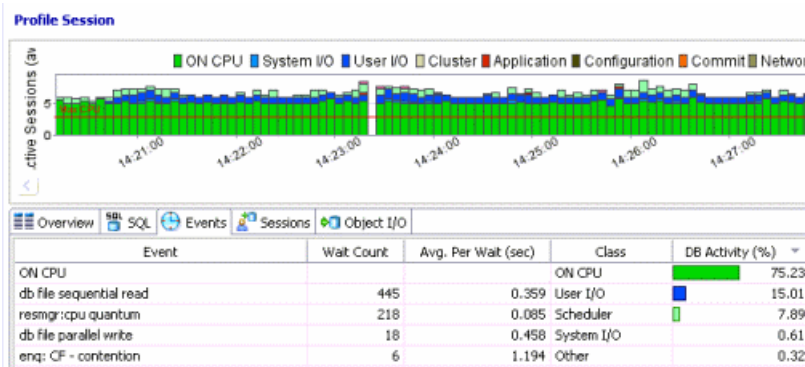
 Columns displayed on the top SQL tab differ depending on the data source platform.

Statistic	Notes
Exeutions	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)	The average amount of time that elapsed while executing the statement during the profiling period.
DB Activity (%)	A graphical representation of the distribution of execution and wait time for the statement or statement component.
SQL ID	The ID value of the SQL statement.

Statistic	Notes
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.

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.



Top Sessions tab

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

The 'Sessions' tab displays a table of active sessions with the following columns: User Name, Program, SID, Serial#, Activity (%), Machine, and Session Type.

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

Top Blockers tab

The Blockers tab provides details on sessions holding blocking locks.

User Name	Program	SID	Serial#	Blocking (%)	Machine	Session Type	Client Info
SYSTEM	JDBC T...lient	91	113	79.10	rowcb...ru02	USER	
	ORA...WR)	166	1	15.67	ROML...RS03	BACK...UND	
SYSTEM	JDBC T...lient	124	26549	4.48	rowcb...iu02	USER	
	ORA...ON)	161	1	0.75	ROML...RS03	BACK...UND	

The following parameters are displayed on the Blockers tab:

Value	Description
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	The serial number of the machine from which the session executed.
Blocking (%)	A graphical representation of the percentage of total blocked sessions being blocked by a blocking session.
Machine	The machine name and network location of the machine from which the session executed.
Session Type	The type of session.
Client Info	The name/type of the client from which the session initiated.

For more detailed information, see [Viewing details on the Blockers tab.](#)

Top Object I/O tab

The **Object I/O** tab 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

The **Procedures** tab displays information about Procedure loads on the profiled data source.

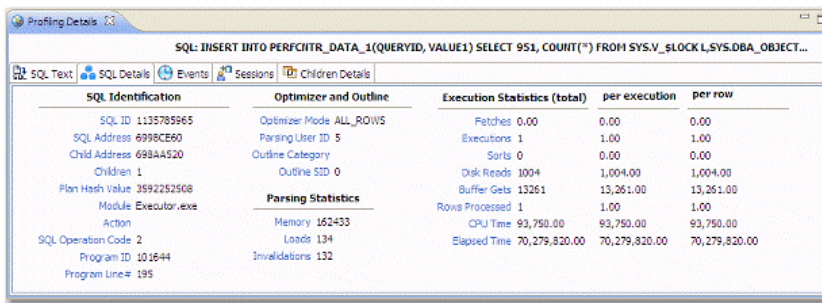
Procedure Name	Database Name	Procedure ID	Executions	DB Activity (%)
TEST_PROC2	codruta	850099038	1	50.00
TEST_PROC1	codruta	1842102572	1	50.00

The following parameters are displayed on the Procedures tab:

Value	Description
Procedure Name	The name of the procedure affecting the database performance.
Oracle	The owner of the schema in which the procedure resides.
Procedure ID	A unique ID created when the procedure is created.

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.



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
SQL Text	Displays the full code of the selected SQL statement.
SQL Details	Provides details on statement, like execution statistics.
Events	Provides database activity details about events the statement is associated with.
Sessions	Shows which sessions executed this statement.
Blockers	Shows which sessions held blocking locks against the session associated with this statement. Double-clicking an entry on this tab opens that session in the Top Blockers tab, letting you find more information on the blocking session. For details, see Top Blockers tab .
Children Details	Lists all copies of the cursor or SQL query, if Oracle has cached multiple copies of the same statement.
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.
Bind Variable Details	Shows bind variable information for SQL captured during the Profiling session.

Event selected

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

Tab Name	Description
SQL	Shows which SQL statements waited on this event.
Sessions	Provides information about the sessions associated with the event.
Blockers	Shows which sessions held blocking locks against the session associated with this event. Double-clicking an entry on this tab opens that session in the Top Blockers tab, letting you find more information on the blocking session. For details, see Top Blockers tab .
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
Analysis	Displays for "buffer busy waits" and "cache buffer chains latch" waits. The analysis shows data and documentation to assist in solving these bottlenecks.

Session selected

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

Tab Name	Description
Sessions	Provides parameters regarding the session. For example, database server connection information, and data regarding the client tool and application.
Blockers	Shows which sessions held blocking locks while this session was active. Double-clicking an entry on this tab opens that session in the Top Blockers tab, letting you find more information on the blocking session. For details, see Top Blockers tab .
SQL	Shows which SQL statements this session ran.
Events	Shows which events this session waited on.

i 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.

Blocking Session selected

When a **Blocking Session** is selected, the following Profiling Detail tabs are available.

Tab Name	Description
Blocked Sessions	Provides identifier and V\$SESSION session information on the sessions being locked by the blocking session.
Session Details	Provides parameters regarding the session. For example, database server connection information, and data regarding the client tool and application.
SQL	Shows the SQL statements associated with the lock.
Events	Shows which events the blocking session waited on.

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](#).
- **SQL Details tab:** Displays execution details. For more information, see [SQL Details](#).
- **Events tab:** Displays information about the events the selected statement is associated with.

For more information, see [Events](#).

- **Sessions tab:** Displays information about the sessions that the selected statement is associated with. For more information, see [Sessions](#).

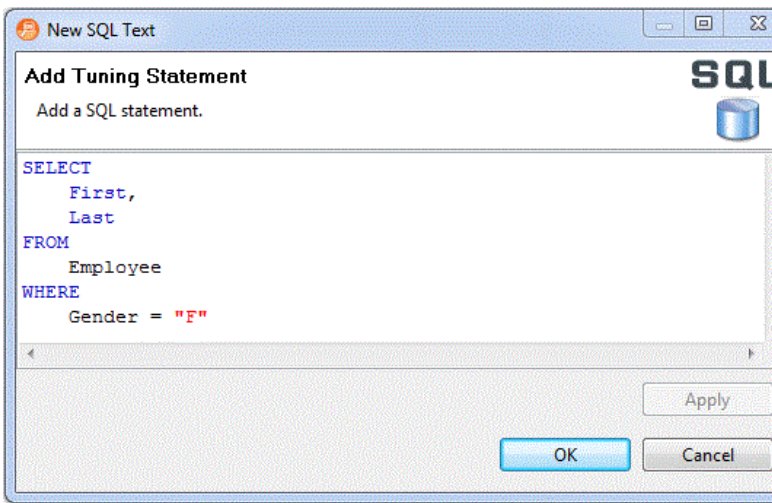
To select a SQL tab statement entry

- 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.

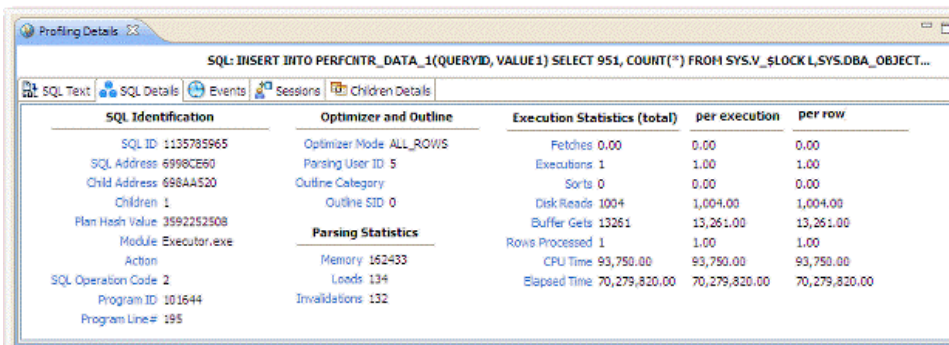
SQL text

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.

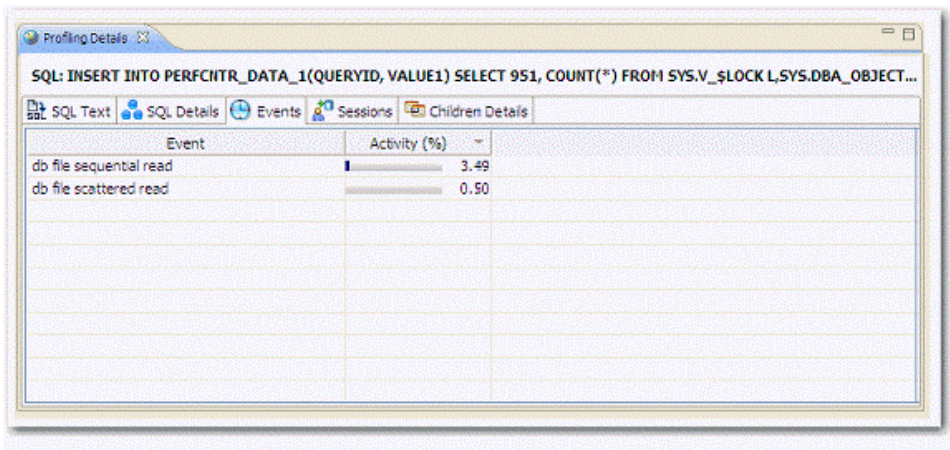


SQL Details include:

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	Executior.exe	145	9180	3.14	EMBARCADERO\ROWSNOVAC01	USER
SYSTEM	Executior.exe	145	9242	0.85	EMBARCADERO\ROWSNOVAC01	USER

Session details include information on different parameters, such as User Name, Program, SID, Serial #, Activity (%), Network Machine Name, and Session Type.

Viewing details on the Sessions tab

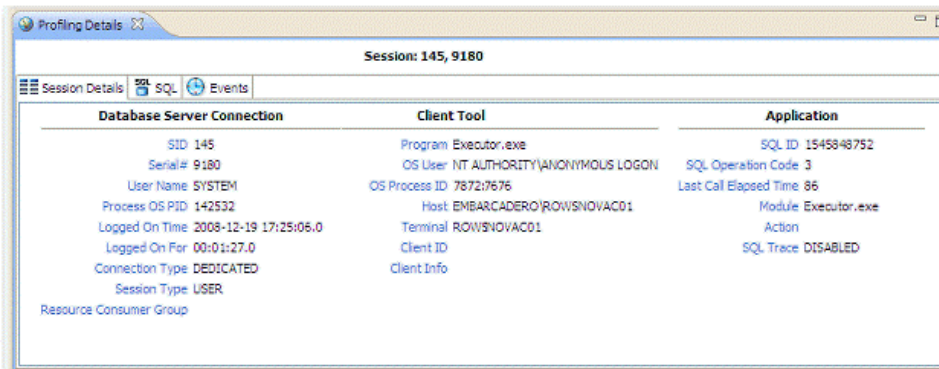
In the **Top Activities Section**, selecting a statement entry on the **Sessions** tab displays information in the **Profiling Details** view. The graph portion and details on the event category tabs on the new editor pertain only to the selected statement. Additionally, new tabs become available.

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](#).
- **SQL tab:** Displays information about the SQL files that the selected session is associated with. For more information, see [SQL](#).
- **Events tab:** Displays the time and parameter information about the selected session. For more information, see [Events](#).

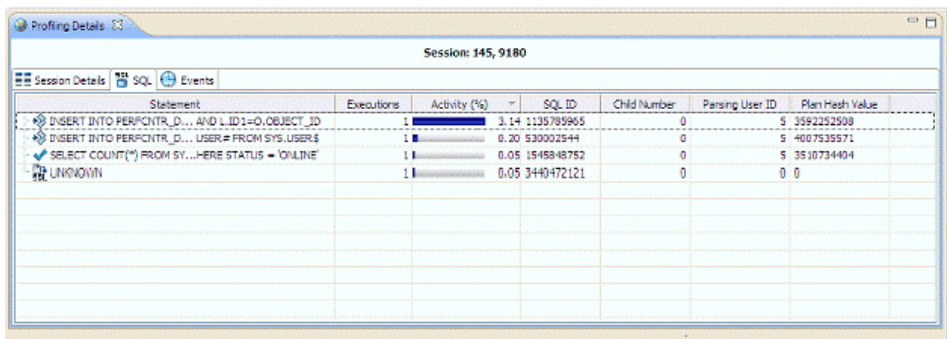
Session details

The Session tab provides further information about the selected session.



SQL

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

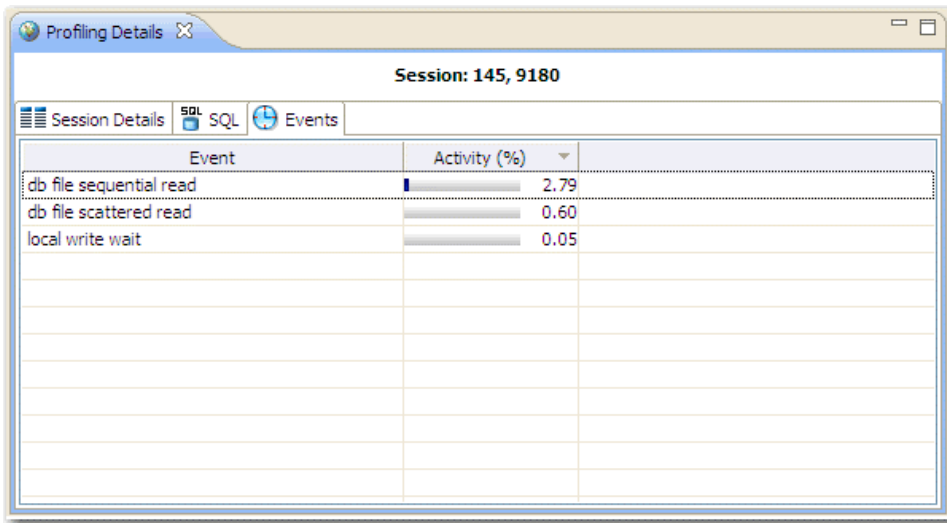


SQL statements are listed by the following parameters:

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 ValueThe execution value of 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.

Bind variable details

Profiling captures the bind variables and their attributes. Select an SQL statement in the Profiling Session and the details of the captured bind variables for that statement are displayed here.

SQL ID	Child Number	Position	Variable Name	Variable Type	Variable Value
dzzdqaujhrpft	0	1		NUMBER	625

The following parameters are displayed on the Bind Variable Details tab:

Value	Description
SQL ID	SQL identifier used by the data source.
Child Number	A new child number is generated for the SQL ID of the query whenever the plan changes, for example the value of a bind variable is changed, and the query is executed again.
Position	The position of the variable within the SQL text. For example, given the query, select * from T1 where C1 = :a and C2 = :b and C3 = :c and C4 = :d, the position of a is 1, b is 2, c is 3 and d is 4.

Value	Description
Variable Name	The name of the variable.
Variable Type	The data type of the variable.
Variable Value	The value of the variable.

Viewing details on the Blockers tab

In the **Top Activities Section**, selecting an entry on the **Blocked Sessions** tab displays information on sessions holding blocking locks in the **Profiling Details** view.

Blocked sessions

The Blocked Sessions tab provides general information on blocked sessions and the details identifying the specific row locked.

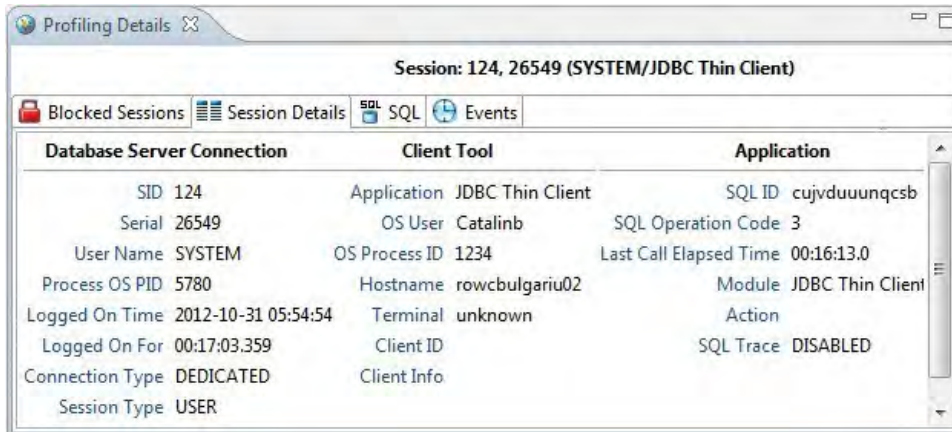
User Name	SID	ROW_WAIT_OBJ#	ROW_WAIT_FILE#	ROW_WAIT_BLOCK# NUMBER	ROW_WAIT_ROW#
SYSTEM	137	-1	0	0	0
SYSTEM	136	-1	0	0	0
SYSTEM	110	-1	0	0	0
SYSTEM	107	-1	0	0	0
SYSTEM	100	-1	0	0	0
SYSTEM	96	-1	0	0	0

This tab provides the following columns for each blocked session:

Value	Notes
User Name	The user name under which the blocking session was run.
SID	The SID value of the blocking session.
ROW_WAIT_OBJ#	Object ID of the table containing the row specified in ROW_WAIT_ROW#.
ROW_WAIT_FILE#	Identifier of the datafile containing the row specified in ROW_WAIT_ROW#.
ROW_WAIT_BLOCK#	Identifier of the block containing the row specified in ROW_WAIT_ROW#.
ROW_WAIT_ROW#	The current row being locked.

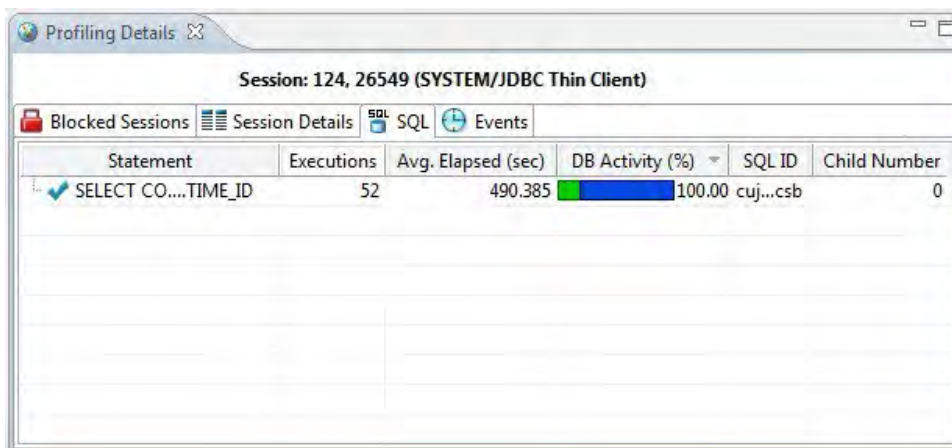
Session details

The Session Details tab provides information on the server connection, client, and application associated with the blocking session.



SQL

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

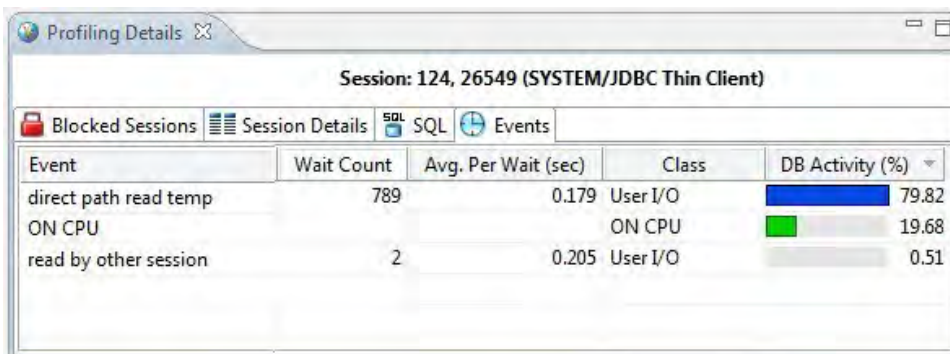


SQL statements are listed by the following parameters:

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.

Events

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



Events are listed by the following values:

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

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](#).
- **Sessions** tab: Displays information about the sessions that the selected event is associated with. For more information, see [Sessions](#).

SQL

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

Statements	CPU	Physical IO	Memory Usage	Execu
DELETE FROM codruta.t1 WHERE ijj = (select max(ijj) from codruta.t1)	0	3036	8	
WHILE (SELECT COUNT(*) FROM codruta.t1) > 1 BE...E ijj = (select max(ijj) from codruta.t1) END	0	1625	8	

Value	Notes
Statement	The name of the statement.
SQL ID	The ID value of the SQL statement.

Value	Notes
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.

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 (DBWR)	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 (PSPO)	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

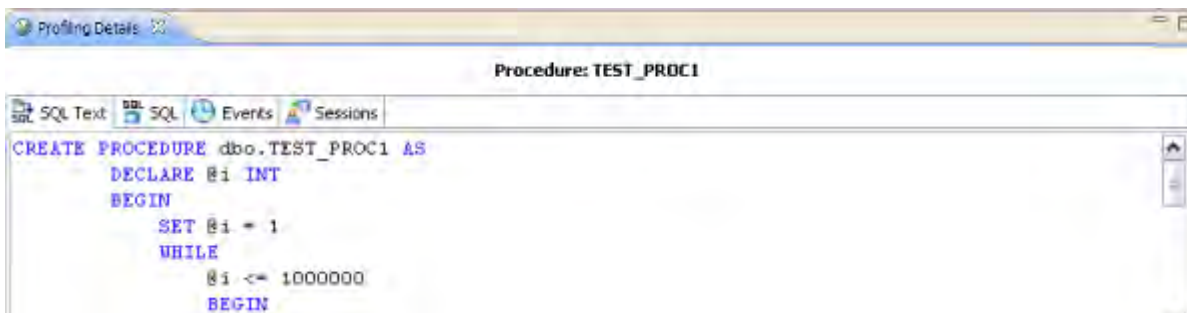
The following parameters are displayed on the Sessions tab:

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.

Value	Notes
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.

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.

The screenshot shows the 'SQL' tab of the 'Profiling Details' window for 'Procedure: TEST_PROC1'. It displays a table with the following data:

Statements	CPU	Physical IO	Memory Usage	Execu
INSERT INTO codruta.t1(jii, ji, 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	

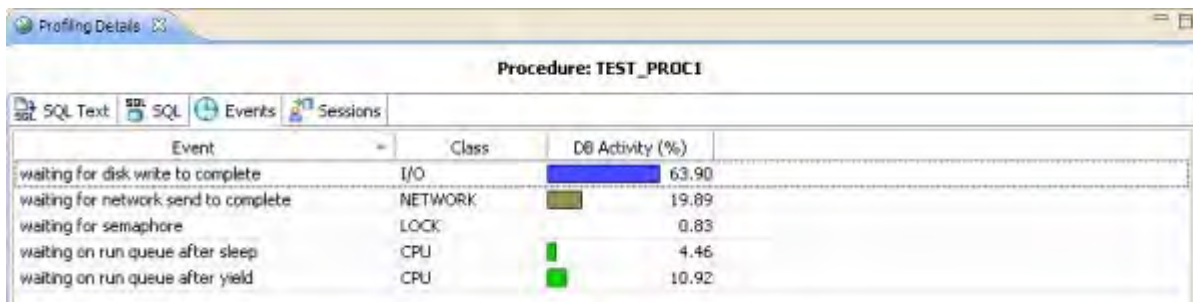
The SQL tab displays the following parameters about the statement:

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.

Value	Notes
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.

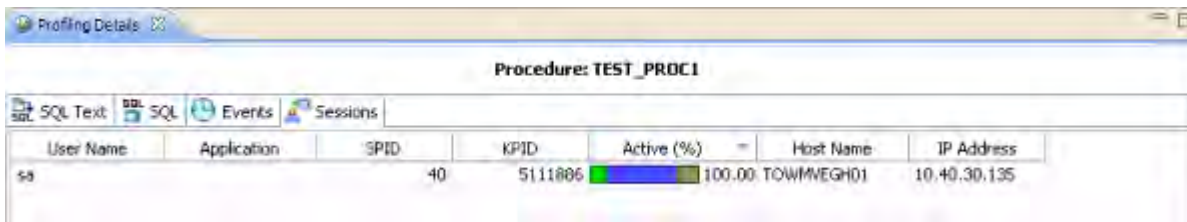


Events are listed by the following values:

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:

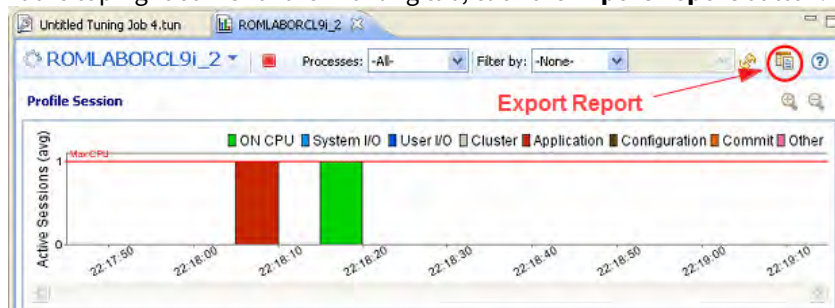
Value	Notes
User Name	The user name under which the session was run.
Program	The name of the executable under which the session was run.
SID	The SID value of the session.
Serial Number	The serial number of the machine from which the session executed.
Activity (%)	A graphical representation of the distribution 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.

You can kill a session by right-clicking the entry on the Sessions tab and choosing **Kill Session**. You can start a trace on a session by right-clicking the entry on the Sessions tab, and then choosing **Trace**.

Creating Profiling Reports

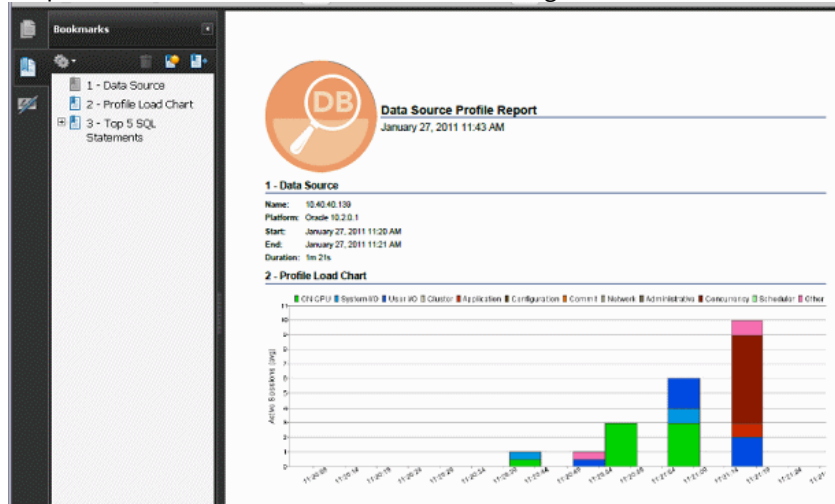
After profiling a data source, you can create an HTML or PDF Report of the profiling session. You can choose the details to include in the report.

1. At the top right corner of the Profiling tab, click the **Export Report** button.



The **Export a Profile Report** dialog appears.

2. Enter a **Report Title** and **Description**.
3. In the **Profile Report Options** area, click the triangles to expand the options.
4. Select your options, enter the location for the report, and then click **Export**. A report in PDF format will resemble the following:



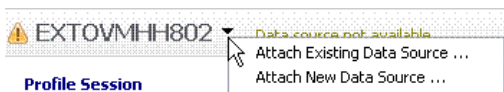
Saving Profiling Sessions

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 profiling sessions, see [Building Profiling Configurations](#). Profiling sessions can be saved in the current workspace in an archive file with a **.oar** suffix.

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

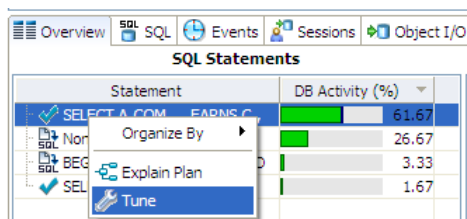
- The name of the data source if the session was not initiated from a named launch configuration
- The name of the launch configuration if the session was initiated from a named launch configuration.

Saving the profile lets you open the archive at a later time for subsequent analysis by yourself or by other SQL Query Tuner users. Use standard SQL Query Tuner 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.



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.

For more information, see [Tuning SQL Statements](#).

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](#).
- **Filtering Results.** For more information, see [Filtering Results](#).

Zooming In and Out

To zoom in or out on the Load Graph

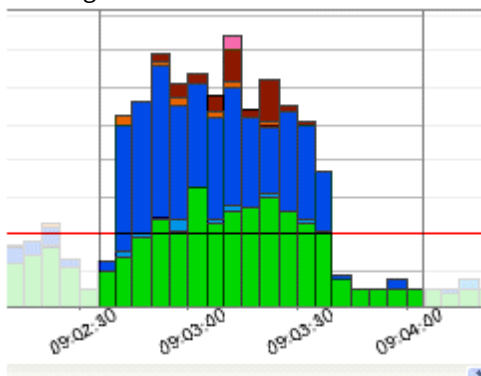
In the upper right-hand corner of the Load graph, click the Zoom In or Zoom Out icons, respectively.

i 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

Click-drag across one or more bars.



Configuring Profiling

This section addresses the following topics:

- [Building profiling configurations](#)
- [Configuring DBMS Properties and Permissions](#)
- [Specify Profile Alerts Preferences during Configuration](#)
- [Using SQL Load Editor/Tester](#)

Building profiling configurations

Profiling enables you to store parameters related to specific profiling sessions, in a profile 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.

i Support for stored routines includes functions and procedures as well as package functions and package procedures.

To create a profile configuration

1. Right-click 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 Target** area, click **Real Application Cluster (RAC) mode** if the target database is an Oracle RAC. This enables you to profile the entire cluster in one profiling session. (In general, profiling a RAC entails querying the GV\$ views.)

i When profiling a RAC, you can also filter the profiling details to show only the details for a selected instance. At the top of the Profiling view, click the Instances list and choose the instance you want to examine.

5. Click **Apply**. The launch configuration is stored in the application.
6. Once a launch configuration is defined, you can execute it in profiling. For more information, see [Running a Profiling Session](#).

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 SQL Query Tuner.

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.

i 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.

Configuring DBMS Properties and Permissions

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.

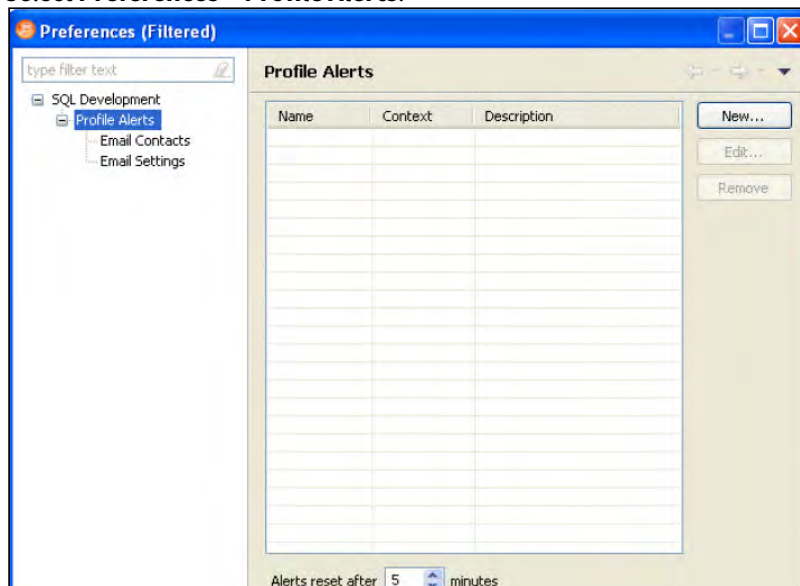
Ensure you restart the server for your changes to take affect.

For tuning, you will need to be a member of sysadmin or have the SHOWPLAN permission enabled.

Specify Profile Alerts Preferences during Configuration

You can configure SQL Query Tuner to send you an alert via email if during a profiling session it detects that database activity has met or exceeded the threshold that you specified.

1. Select **Preferences > Profile Alerts**.



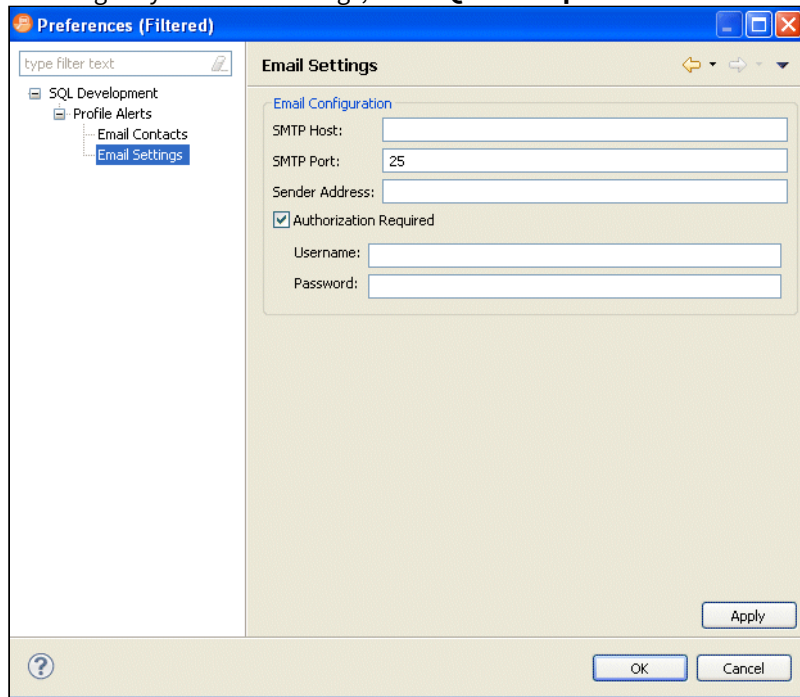
- To create a profile alert, click **New**.

To edit an existing profile alert, select the alert you want to modify and then click **Edit**. The **Create Profiling Alert** or **Edit Profiling Alert** dialog appears.

- Complete the create/edit profiling alert dialog as required and then click **OK**.
- In the **Preferences** tree, click **Email Contacts**.
If you select **Display system notification when alert fires**, you will receive an alert notification in your Windows system tray when the alert fires.
If you select **Send email when alert fires**, you must specify email contacts and email server settings.
- To create a new email contact, click **New**.
To edit an existing email contact, select the contact you want to modify and then click **Edit**.

- From the list of **Data Sources**, select the data sources for which this contact should receive an email notification that an alert has fired, and then click **OK** to save this contact.

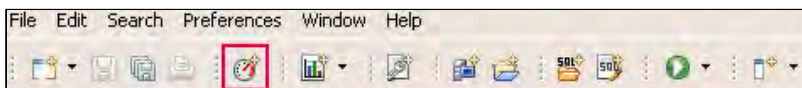
7. To configure your email settings, click **SQL Development > Profile Alerts > Email Settings**.



8. Complete the Email Settings as required and click **Apply**.

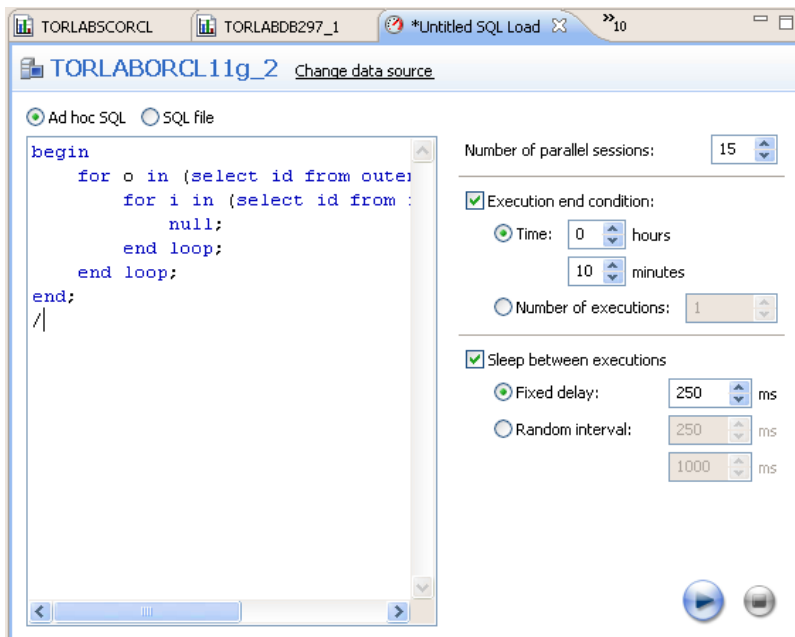
- **Sender Address:** This is an email user configured on your email server. SQL Query Tuner uses this address to send alerts to the email contacts defined.

Using SQL Load Editor/Tester



The load editor can be run either with **File > New > SQL Load** or with the **New SQL Load** 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/Tester 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/Tester 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 SQL Load Editor/Tester is accessed via the **New SQL Load** icon on the Toolbar:



When you open SQL Load Editor/Tester, 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 SQL Load Tester/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 SQL Load Editor/Tester by selecting the icon on the Toolbar. The **SQL Load Editor/ Tester** opens.
2. Click **Change 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 SQL Load Editor/Tester 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:

- [Understanding the Tuner Interface](#)
- [Tuning SQL Statements](#)
- [Additional Tuning Commands](#)
- [Configuring Tuning](#)
- [Examples of Transformations and SQL Query Rewrites](#)

Understanding the Tuner Interface

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

- [Overview](#)
- [Analysis](#)

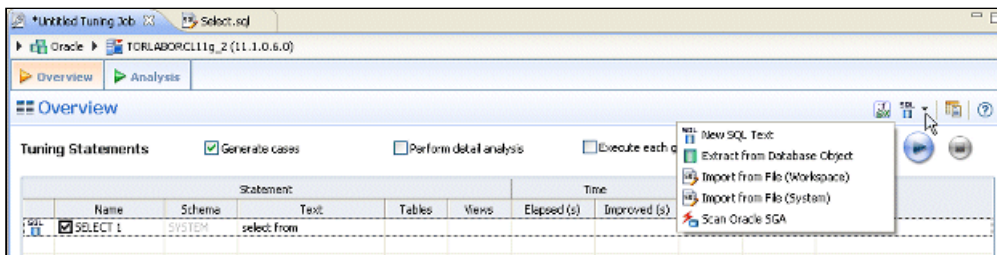
Several additional tabs appear on the Analysis and Outlines tabs. For more information on utilizing these extra features, see:

- [Using the Table Statistics tab](#)
- [Using the Column Statistics and Histograms tab](#)
- [Using the Outlines tab](#)

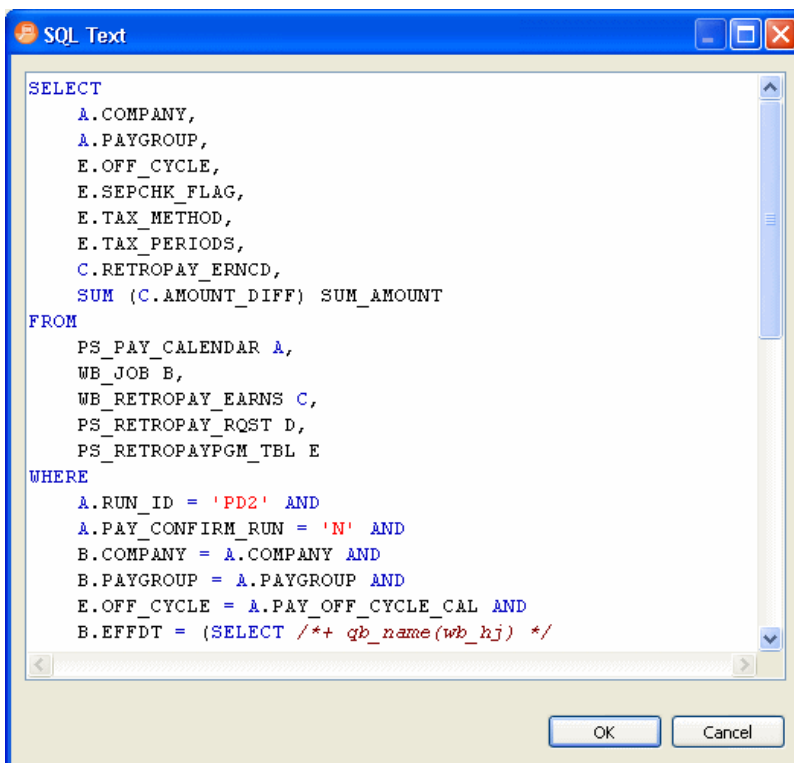
Understanding the Overview tab


Inputting SQL to tune

Click the SQL button on the **Overview** tab to specify the source of SQL statements you want to tune.



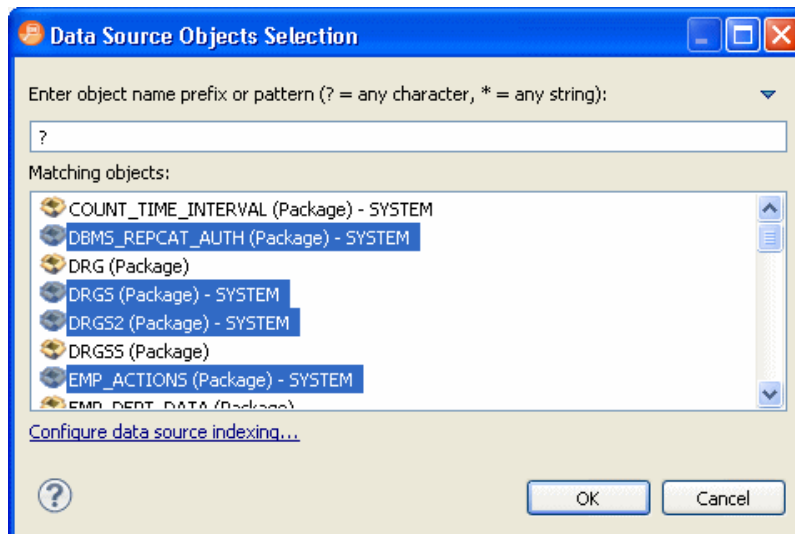
- **New SQL Text:** From the SQL button menu, select **New SQL Text**, and then copy/paste SQL statements to the **SQL Text** dialog or write queries by hand and then click **OK**.



 You can also input SQL by clicking anywhere in the Tuning Statements area and pressing Ctrl-V.

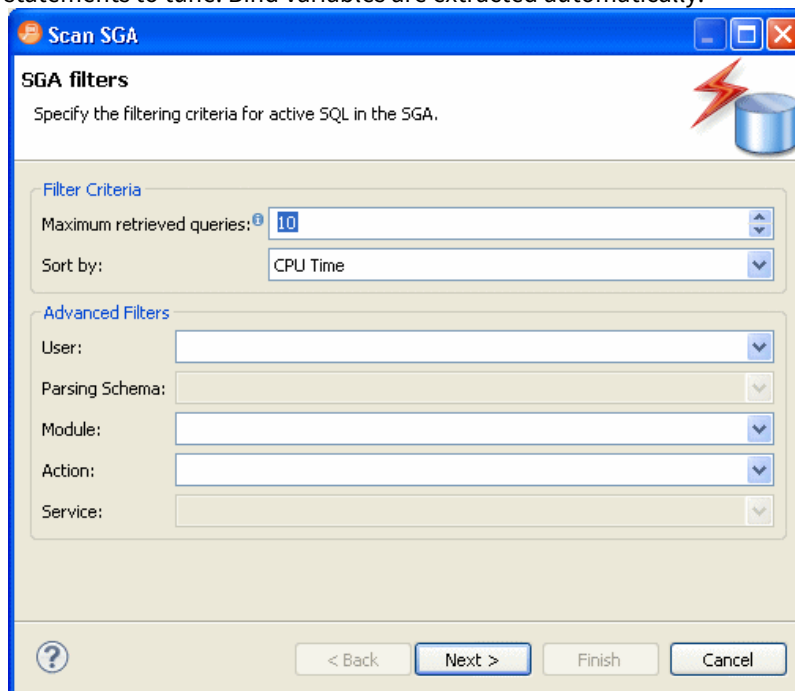
Once you have input the SQL and click **OK**, you can later edit the text by right-clicking an entry in the Tuning Statements area and selecting **Edit**.

- **Extract from Database Objects:** Search for and then select (Ctrl-click) data base objects containing SQL that you want to tune from the selected data source. SQL Query Tuner 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 data source indexing....**



For information on setting data source indexing properties, see [Specify Data Source Indexing Preferences](#).

- **Import from File (Workspace) and Import from File (System):** Browse the workspace or file system and select an SQL file from which to extract statements to tune.
- **Scan Oracle SGA:** For the Oracle platform only, you can also scan the System Global Area (SGA) for statements to tune. Bind variables are extracted automatically.

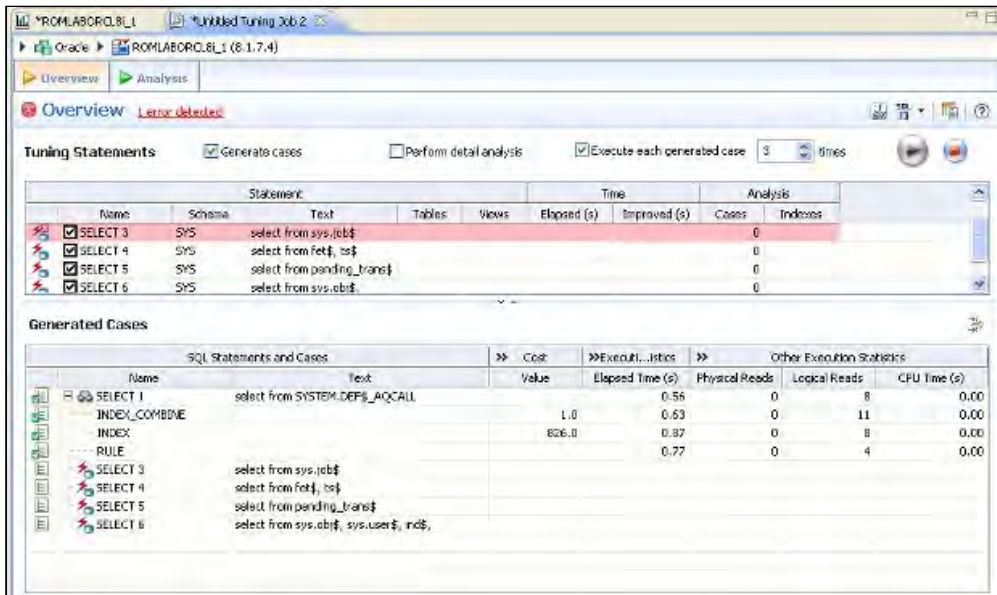


- You can also drag and drop Materialized Views, Procedures, and Views from the Data Source Explorer to the Tuning Statements grid and they will be added to the list of statements to tune.

Running a tuning job

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, 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.



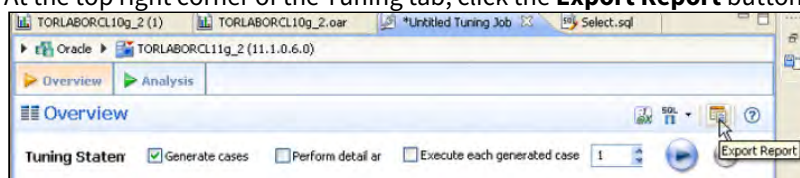
There are three tuning options to choose from before clicking Run Job:

- 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.

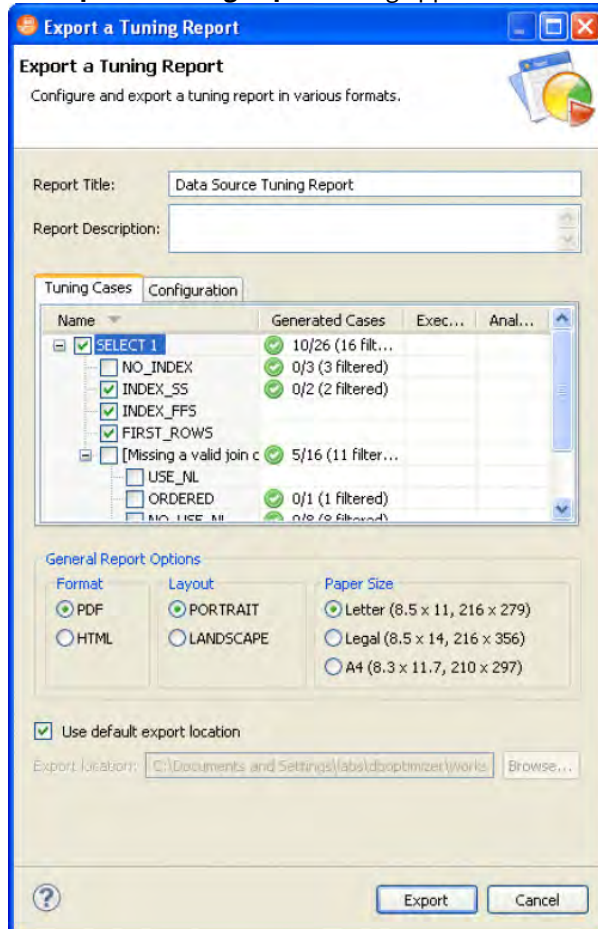
Creating tuning reports

After tuning SQL you can create an HTML or PDF Report of the tuning session. You can choose the details to include in the report.

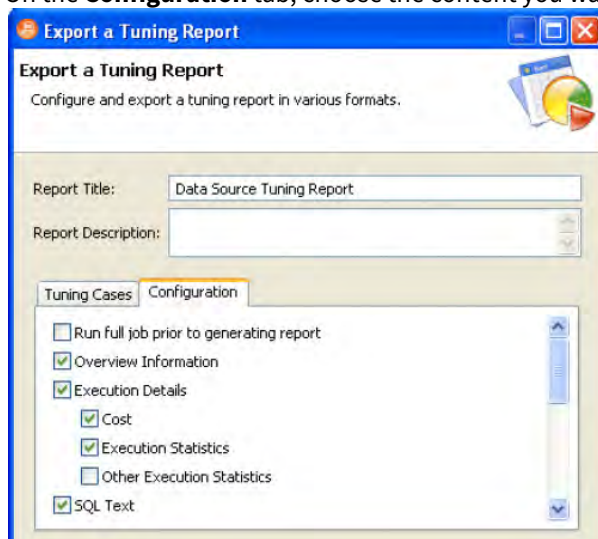
1. At the top right corner of the Tuning tab, click the **Export Report** button.



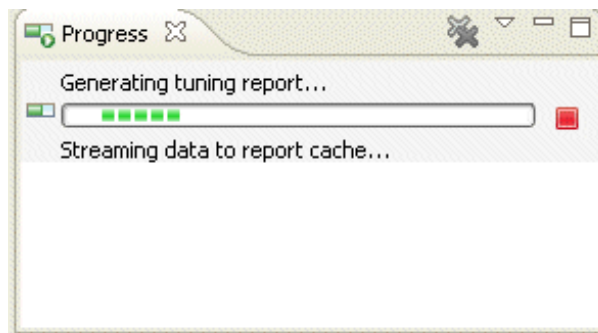
The **Export a Tuning Report** dialog appears.



2. Enter a **Report Title** and **Description**.
3. On the **Tuning Cases** tab, choose the cases you want to report on. Click the + to expand the cases.
4. On the **Configuration** tab, choose the content you want to include in the report.



5. Select the **General Report** options, enter the location for the report, and then click **Export**. You will see the progress of the report generation in the Progress pane.



When complete, the report is stored at the top level of your workspace.
A report in PDF format will resemble the following:

Data Source Tuning Report
March 6, 2012 10:07 AM
Test report

1 - Data source:
Name: TORLABORCL11g_2
Platform: Oracle 11.1.0.6.0
Tuning Job Name: Untitled Tuning Job

2 - Overview

Name	Source	Schema	Text	Tables	Views	Elapsed Time(s)	Improved Time(s)	Cases Analyzed	Indexes Analyzed
SELECT 1	Custom Case	SYSTEM	select from HR.EMPLOYEES, HR.DEPARTME...					10	

3 - Case SELECT 1

3.1 - Generated Cases

Name	Text	Cost Result	Elapsed Time (s)	Rows Returned
SELECT 1	select from HR.EMPLOYEES, HR.DEPARTMENTS	41.0		
FIRST_ROWS		4.0		
Transformation		7.0		
INDEX_FFS		40.0		
INDEX_SS		69.0		

3.2 - SQL Text

```
SELECT *
FROM
  HR.EMPLOYEES,
  HR.DEPARTMENTS;
```

3.3 - Case FIRST_ROWS

3.3.1 - Generated Cases

Name	Text	Cost Result	Elapsed Time (s)	Rows Returned
FIRST_ROWS		4.0		

3.3.2 - SQL Text

```
SELECT (* FIRST_ROWS ( 10 ) *)
FROM
  HR.EMPLOYEES;
```

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. SQL Query Tuner 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 screenshot shows the SQL Analysis tool interface. At the top, there is a 'Select statement of interest:' dropdown menu (1) containing 'SELECT I'. Below this is the SQL text (2) for a query selecting employee details. To the right is a Visual SQL Tuning (VST) diagram (3) showing the execution path between tables: EMPLOYEES (E), DEPARTMENTS (D), JOBS (J), LOCATIONS (L), and COUNTRIES (C). Below the diagram is a table (4) listing indexes for the tables in the query. The selected index is 'DEPT_ID_PK' on the 'DEPARTMENTS' table. To the right of the table is a description (5) of the selected index.

Index Name	Table Owner	Table Name	Column Name	Index
<input type="checkbox"/> COUNTRY_C_ID_PK	HR	COUNTRIES	COUNTRY_ID	Unique
<input type="checkbox"/> DEPT_LOCATION_IX	HR	DEPARTMENTS	LOCATION_ID	Normal
<input type="checkbox"/> JOB_ID_PK	HR	JOBS	JOB_ID	Unique
<input type="checkbox"/> LOC_COUNTRY_IX	HR	LOCATIONS	COUNTRY_ID	Normal
<input type="checkbox"/> REG_ID_PK	HR	REGIONS	REGION_ID	Unique
<input checked="" type="checkbox"/> DEPT_ID_PK	HR	DEPARTMENTS	DEPARTMENT_ID	Unique
<input type="checkbox"/> EMP_DEPARTMENT_IV	HR	EMPLOYEES	DEPARTMENT_ID	Normal

This index is defined on a column present in the predicates, so it could be used by the database optimizer when you run the statement.





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

1. **Statement selector**, if there are multiple statements in the tuning set. Here you can select the statement and the generated case you want to analyze.
2. **Statement text** for selected statement.
3. **Graphical diagram** of the SQL statement.
4. Index analysis, statistics, and settings relating to the SQL statement and referenced elements.
5. **Description of the selected index**, including the reasoning behind SQL Query Tuner recommendations.

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:

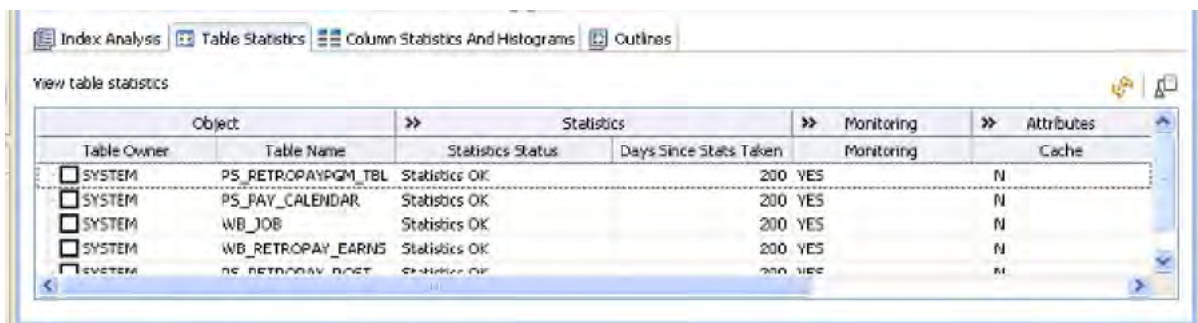
Text Color	Interpretation
	Index is used in the query
	Index is usable but not used in the current execution path.
	This index is missing. SQL Query Tuner recommends that you create this index.
	This index exists on the table but not usable in this query as it is written.

In the **Collect and Create Indexes** table, orange-highlighted entries indicate missing indexes that SQL Query Tuner recommends be created to improve performance. Clicking on that index, displays text to the right outlining the rationale behind this recommendation.

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




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.



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_CALENDAR	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	WB_JOB	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	WB_RETROPAY_EARN5	Statistics OK	200	YES	N
<input type="checkbox"/> SYSTEM	PS_RETROPAY_POST	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.

i 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.

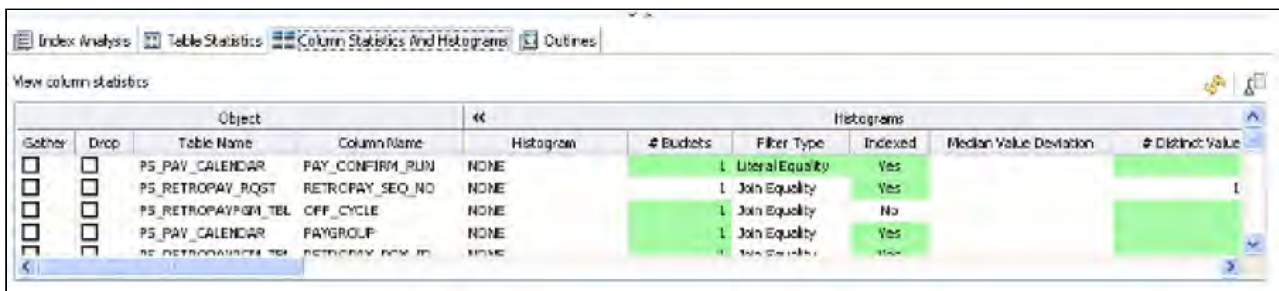
i You can right-click anywhere in a row and choose options such as Collect Statistics, Display Statistics, and Copy from the short-cut menu.

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.

Column Statistics and Histograms tab example

SQL Query Tuner 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		Column Name	Histogram	# Buckets	Filter Type	Indexed	Median Value Deviation	# Distinct Value
<input type="checkbox"/>	<input type="checkbox"/>	PS_PAV_CALENDAR.PAY_CONFIRM_RUN	NONE	1	Literal Equality	Yes		
<input type="checkbox"/>	<input type="checkbox"/>	PS_RETROPAY_BQST.RETROPAY_SEQ_NO	NONE	1	Join Equality	Yes		1
<input type="checkbox"/>	<input type="checkbox"/>	PS_RETROPAYBQST.TBL.OFF_CYCLE	NONE	1	Join Equality	No		
<input type="checkbox"/>	<input type="checkbox"/>	PS_PAV_CALENDAR.PAYGROUP	NONE	1	Join Equality	Yes		
<input type="checkbox"/>	<input type="checkbox"/>	PS_RETROPAYBQST.RETROPAY_BQST	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**.

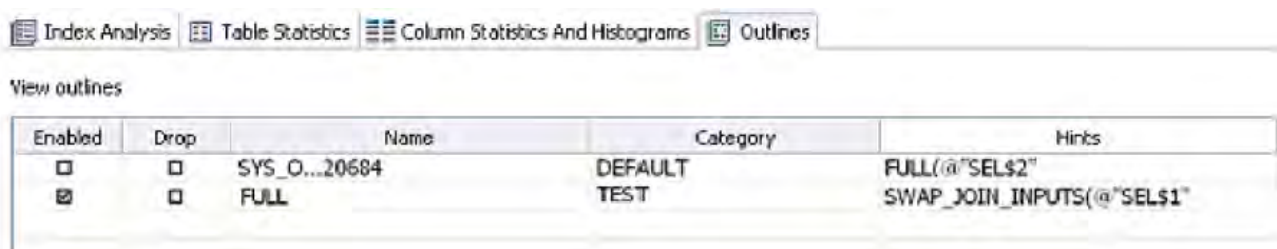
To stop gathering statistics for an object, such as a bad candidate for a histogram, select **Drop** for that column and then click **Display Statistics** or **Collect Statistics**.

i 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:

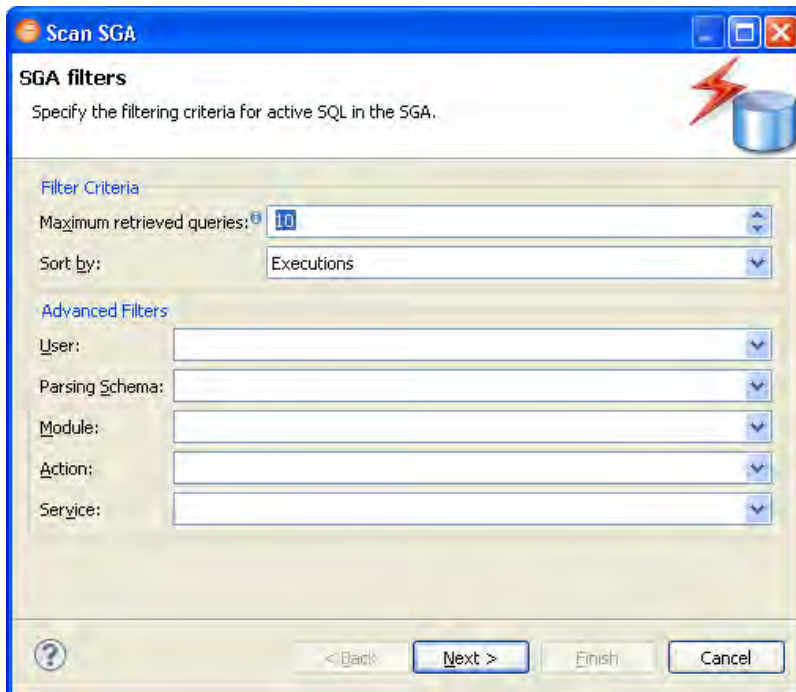
```
alter system set USE_STORED_OUTLINES=true; alter system set
USE_STORED_OUTLINES='DEFAULT'; alter session set USE_STORED_OUTLINES=true;
```

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

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. SQL Query Tuner cannot tell you which statements have most recently run by looking in the SGA. However, by profiling the database using SQL Query Tuner 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 IDERA 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. From the **Overview** tab, click the SQL icon and select **Scan Oracle SGA**. The **Scan SGA** wizard appears.
2. Set the filtering criteria for an SGA scan and then run the wizard.
3. Choose the specific statements and add them to the tuning job.


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.

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](#)
2. [Specify a Data Source](#)
3. [Add SQL Statements](#)
4. [Run a Tuning Job](#)
5. [Analyze Tuning Results](#)
6. [Modify Tuning Results](#)

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

 For information on working with data sources such as adding and browsing them, see [Working with Data Sources](#).

Create a New Tuning Job

You can create a new tuning job via the **File > New > Tuning Job** command, or by importing statements directly from profiling. The 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

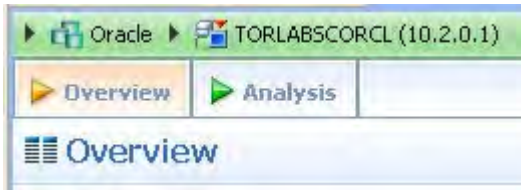
Navigate to the **SQL Project** tab and double-click the name of the existing tuning job.

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 TORLABSCORCL, which is part of the Oracle data source group. The color of the bar at the top of the tuning window shows the category of the data source as defined in the data source configuration properties.



You can change the data source of a tuning job by clicking a bread crumb triangle and then navigating to the data source or using the filter to locate and then select a data source.

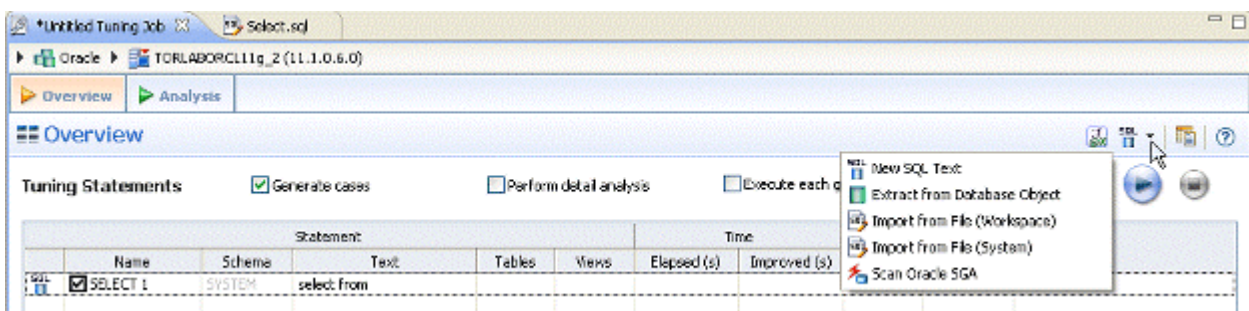
Click the name of the desired data source to affect the change.

i 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.

Statements are added to tuning via the Overview pane.



There are several different methods for adding SQL statements to a job, as reflected by the option in the New SQL text menu.

- **New SQL Text** enables tuning via manual entry, or cutting and pasting into the tuning window.
- **Extract from Database Objects** 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, materialized views, procedures, and views, whose name begins with t. You can then drag and drop the matches from the Data Source Explorer to the Tuning Statements grid.


- The **Import from File (Workspace)** and **Import from File (System)** options enables you to choose an SQL file saved in your workspace or elsewhere on your computer or network.
- The **Scan Oracle SGA** option 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](#).

To add an ad hoc SQL statement:


Select the **New SQL Text** option 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 **Extract from Database Objects** option.
The Data Source Object Selection dialog appears where you can search for and then select the object you want to tune.
2. Type an object name prefix or pattern in the field provided. The Matching objects window 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.

 In order to find matching objects, data source indexing must be enabled. To enable data source indexing, click Configure data source indexing, select **Enable indexing**, and then click **OK**.

3. Click the object you want to add. Ctrl-click to add more than one object to the job.
Click **OK**.

 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*

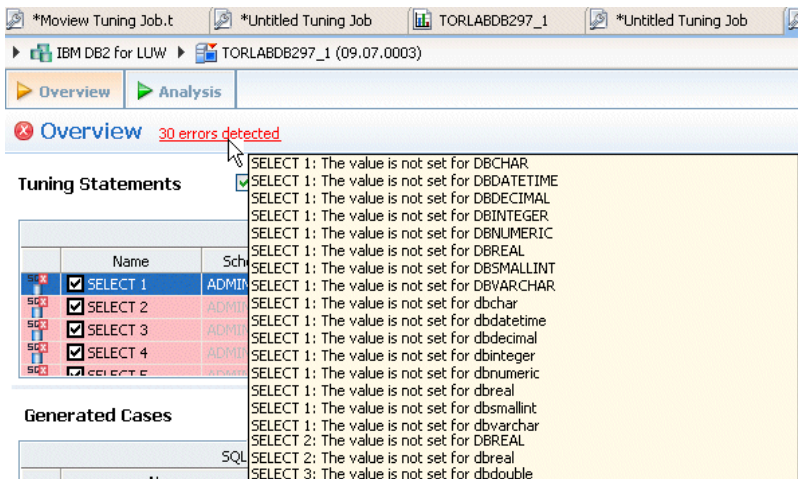
1. From the New SQL Text menu, select either **Import from File (Workspace)** or **Import from File (System)**, depending on where the file you want to add is stored:
 - Workspace files are files that reside in the application, meaning project files or other objects generated or stored in the system.
 - 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.

To add SQL from the Oracle SGA

1. From the **New SQL Text** menu, select **Scan Oracle SGA**.
2. The system scans for SQL text which you can filter on the **Scan SGA** dialog that appears.
3. Choose the statement to be tuned and then click **Finish**.

Managing Bind Variable Errors

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. Mouse over the error to learn what problems were detected.

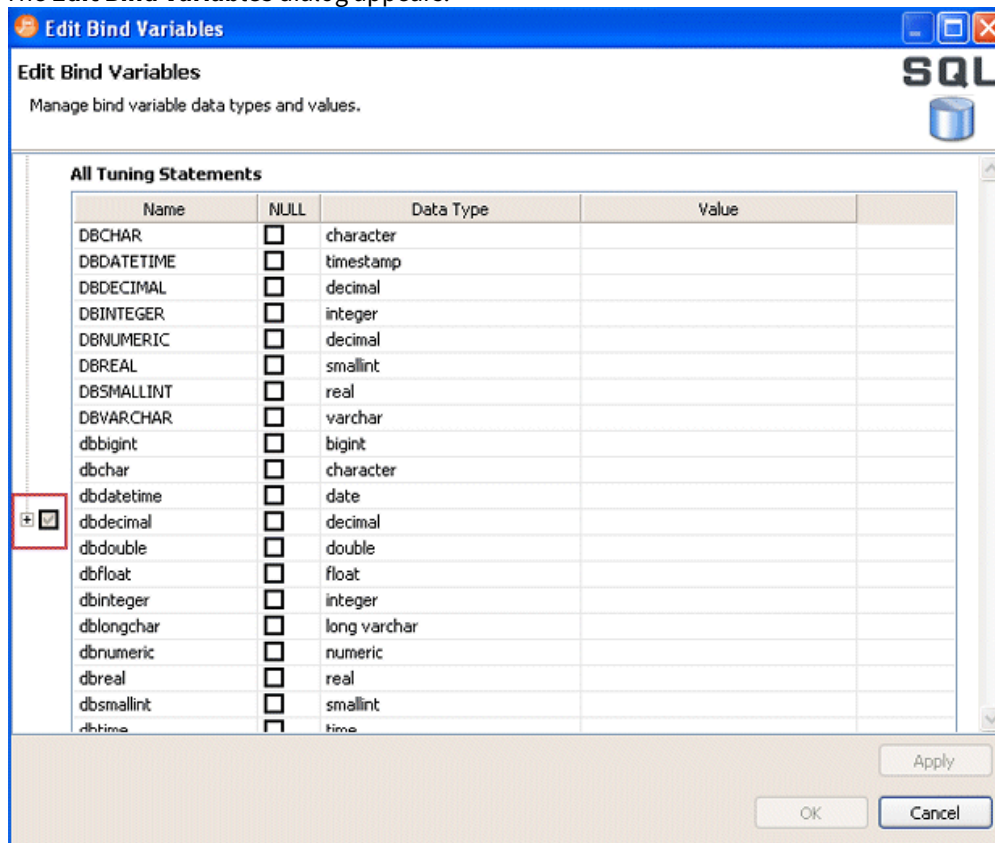


You can use the Bind Variable Editor to set the types or variables.

1. Click the **Edit Bind Variable** icon as shown below.



The **Edit Bind Variables** dialog appears.



You can either set the bind variables for all tuning statements at once or you can set the bind variables for each select statement individually. If you set the bind variable in the All Tuning Statement section, you can still override that setting for an individual select statement.

2. Set the bind variable by clicking the **NULL** box, clicking in the **Data Type** column and selecting the data type from the list, or by clicking in the **Value** column and entering the value.

- If you want to set the bind variable for an individual select statement, click the expand button to see all the select statements. The expand button is marked with a red box in the previous image.

SELECT 11 (select from MOVIES.CUSTOMER, MOVIES.MOVIERENTAL)

Name	NULL	Data Type	Value
<input checked="" type="checkbox"/> DBCHAR	<input type="checkbox"/>	character	
dbchar	<input type="checkbox"/>	date	
dblougchar	<input type="checkbox"/>	integer	
		bigint	
		blob	
		boolean	

In this manner you can set the bind variables from all tuning statements and then override that setting by setting the bind variable for a specific select statement.

You may find it easier to set the bind variables when you can see the tuning statement. In the Generated Cases section, you can double click a statement and an editor appears where you can edit the SQL statement and set the variable data types and values.

Edit Case - SELECT 1

Edit Tuning Statement

Update the SQL statement text as well as manage variable data types/values.

```

SELECT
  cs.customerid,
  cs.firstname,
  cs.lastname,
  mr.rentalid,
  mr.duedate,
  mr.totalcharge,
  ri.itemnumber
FROM
  MOVIES.customer cs,
  MOVIES.movierental mr,
  MOVIES.rentalitem ri,
  (SELECT
   DISTINCT
   AVG (totalcharge) COL10
   FROM MOVIES.movierental) T31
WHERE
  cs.FIRSTNAME = dbchar AND
  cs.zip > dbvarchar AND
  dbsmallint < cs.customerid AND
  
```

Name	NULL	Data Type	Value
DBCHAR	<input type="checkbox"/>	character	
DBDATETIME	<input type="checkbox"/>	timestamp	
DBDECIMAL	<input type="checkbox"/>	decimal	
DBINTEGER	<input type="checkbox"/>	integer	
DBNUMERIC	<input type="checkbox"/>	decimal	
DBREAL	<input type="checkbox"/>	smallint	
DBSMALLINT	<input type="checkbox"/>	real	
DBVARCHAR	<input type="checkbox"/>	varchar	
dbchar	<input type="checkbox"/>	character	
dbdatetime	<input type="checkbox"/>	date	
dbdecimal	<input type="checkbox"/>	decimal	
dbinteger	<input type="checkbox"/>	integer	
dbnumeric	<input type="checkbox"/>	numeric	

Buttons: Apply, OK, Cancel

Run a tuning job

As you add SQL statements to the job on the Overview tab of the tuner, tuning-supported DML statements 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. 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 (and Other Execution Statistics columns, if available) 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](#) and [Specify a data source](#).
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 **Overview** tab, specify the SQL you want to tune:
 - Modify the number of times to execute each statement in the **Execute each generated case** field at the top right of the tuner, as needed.
5. Click the execution button [] on the right side of the case generation field.
6. 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. Click 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.

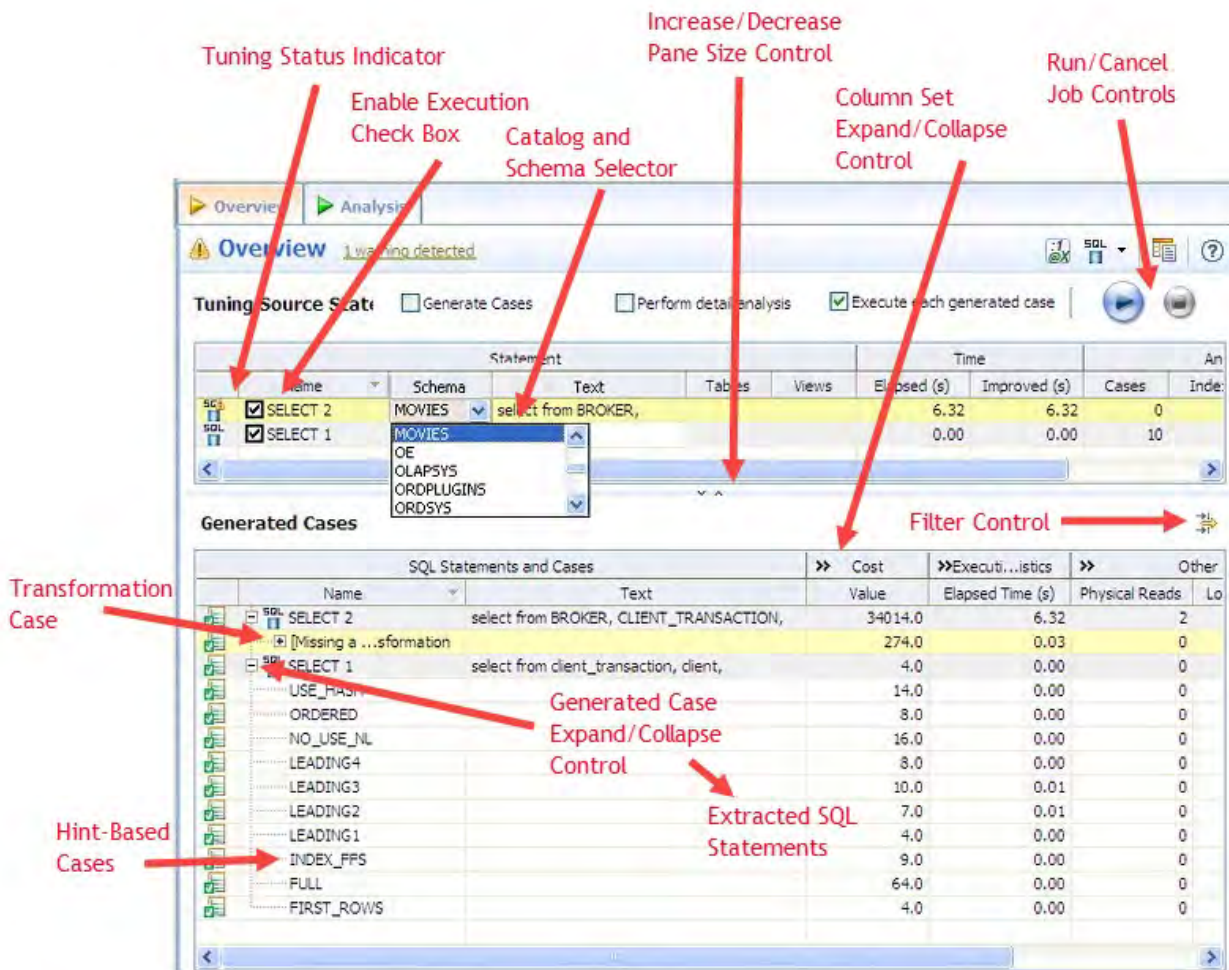
The screenshot shows the SQL Query Tuner interface. The top bar indicates the database is Oracle 17.1.7.4. The 'Overview' tab is active, showing a 'Error detected' message. Below this, there are controls for 'Tuning Statements' with options for 'Generates cases', 'Perform detail analysis', and 'Execute each generated case' (set to 3 times). The main area is divided into two tables:

Statement					Time		Analysis	
Name	Schema	Text	Tables	Views	Elapsed (s)	Improved (s)	Cases	Indexes
<input checked="" type="checkbox"/> SELECT 3	SYS	select from sys.job\$					0	
<input checked="" type="checkbox"/> SELECT 4	SYS	select from fet\$, ts\$					0	
<input checked="" type="checkbox"/> SELECT 5	SYS	select from pending_trans\$					0	
<input checked="" type="checkbox"/> SELECT 6	SYS	select from sys.obj\$,					0	


SQL Statements and Cases		Cost	Execution Statistics	Other Execution Statistics		
Name	Text	Value	Elapsed Time (s)	Physical Reads	Logical Reads	CPU Time (s)
SELECT 1	select from SYSTEM.DEFS_AQCALL		0.56	0	8	0.00
INDEX_COMBINE		1.0	0.63	0	11	0.00
INDEX		826.0	0.87	0	8	0.00
RULE			0.77	0	4	0.00
SELECT 3	select from sys.job\$					
SELECT 4	select from fet\$, ts\$					
SELECT 5	select from pending_trans\$					
SELECT 6	select from sys.obj\$, sys.user\$, ind\$,					



Analyze tuning results

Once you have executed a tuning job, the **Overview** tab reflects tuning analysis of the specified statements. The **Analysis** tab 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](#).





- The **Generated case Expand/Collapse** control lets you hide or display the hint-based cases and transformation-based case generated for a statement.
- The **Perform detail analysis** and **Execute each generated case** 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.
- Use the **Schema Selector** to select a schema for the tuning job. By specifying the schema, the tuner can use the path of the schema selected to find the tables queried in the job rather than use the path of the schema used to connect to the data source. If you change the schema used in a tuning statement you will need to refresh the tuning statements in order for new cases to be generated, which take into consideration the schema used. Right-click a tuning statement, and then select **Refresh Tuning Statements**.
- 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.

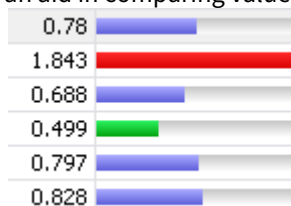
Icon	Description
	The case has been successfully executed.
	Execution for this case failed or was cancelled due to execution time exceeding 1.5 of original case time.

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


 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 may also indicate that the tuning statements are out of sync, in which case you can right-click a tuning statement and select **Refresh Tuning Statements**.

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 expanded 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 expanded 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.

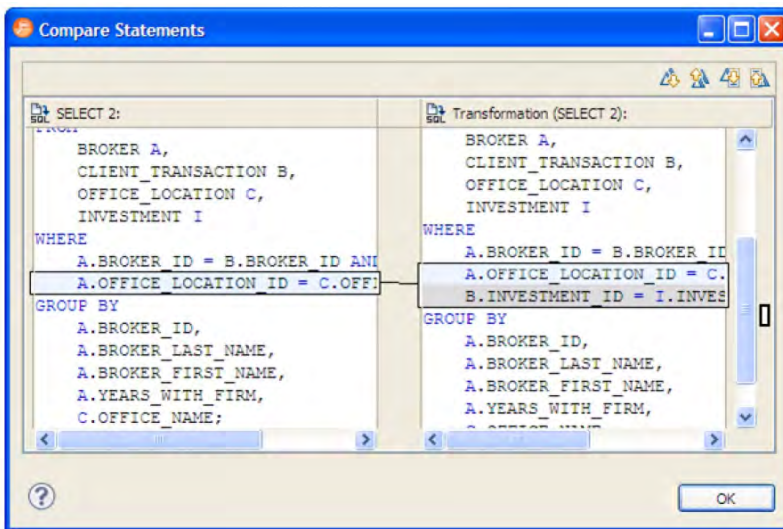
 For information on specifying colors, and the improvement threshold and degradation threshold values used in these graphical representations, see [Specify Tuning Job Editor Preferences](#).

Additionally, once results have been generated you can:

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

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, and then right-click in the **Name** field of either case. 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](#).

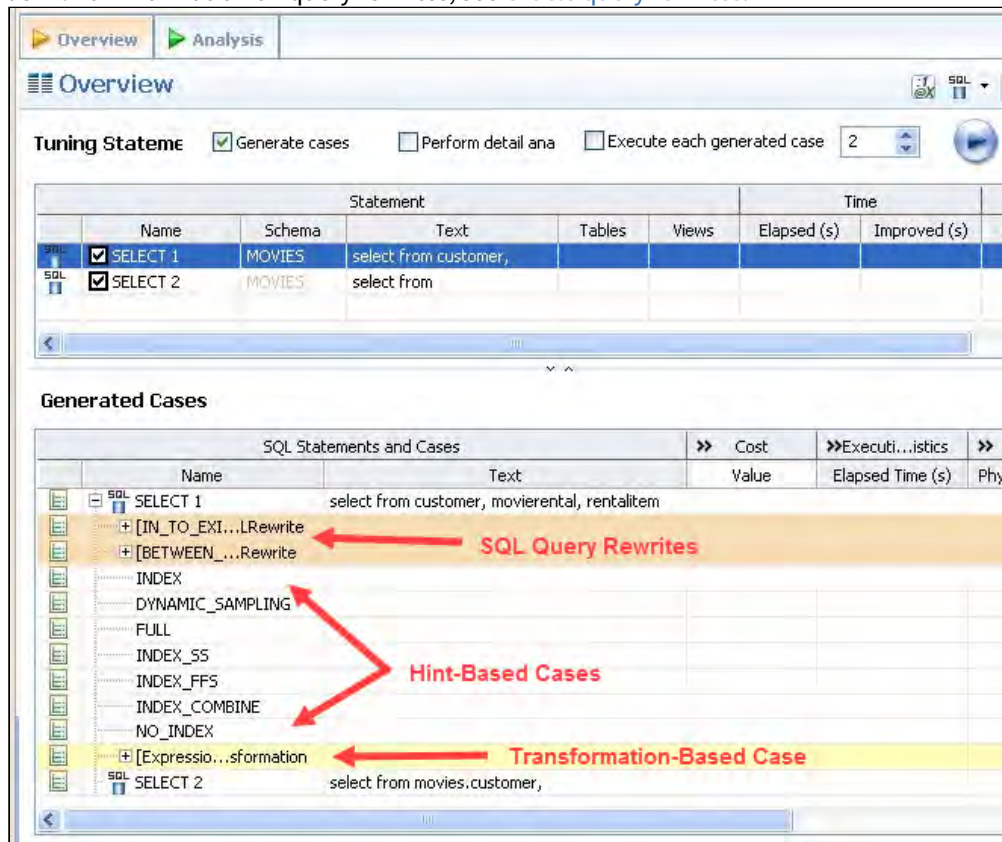
Modify tuning results

As you add SQL source to the Overview 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: If you change the schema of a 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 SQL Query Tuner Code Quality Check functionality. See [Understanding Code Quality Checks](#) 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 query rewrites, see [Oracle query rewrites](#). For information on other transformations, see [Examples of Transformations and SQL Query Rewrites](#).

- SQL Query Rewrites may be suggested when tuning. For example, a recommended rewrite for EXISTS may be IN. For information on query rewrites, see [Oracle query rewrites](#).



Hint-based cases and the transformation-based cases 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.

Oracle query rewrites

The following query rewrites or transformations may be recommended during tuning.

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);

Before	After
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
select t1.* from t1, t2 where t1.key = t2.key and t2.col = 10;	select t1.* from t1, (select * from t2 where t2.col = 10) inline_alias where t1.key= inline_alias.key;
select t2.* from t1, t2 where t1.key = t2.key and t1.col is null	select * from t2 where t2.key IN (select t1.key from t1 where t1.col is null)

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 Query Tuner interface for an Oracle database. The main window is titled "Untitled Tuning Job" and shows the "Analysis" tab. The "SQL Analysis" section displays a SQL statement and its execution plan.

SQL Statement:

```

SELECT
  cs.customerid,
  cs.firstname,
  cs.lastname,
  mr.rentalid,
  mr.duedate,
  mr.totalcharge,
  ri.itemnumber
FROM
  MOVIES.customer cs,
  MOVIES.movierental mr,
  MOVIES.rentalitem ri
WHERE
  LENGTH (cs.lastname) = 5 AND
  cs.zip > 75062 AND
  1 < cs.customerid + 2 AND
  cs.phone BETWEEN 9625569900 AND 5
  ROUND (ri.rentalid) > 10 AND
  TRUNC (ri.itemnumber) > 1

```

Execution Plan:

```

graph TD
  CS[Customer (CS)] --> MR[Movierental (MR)]
  MR --> SQ1[Subquery (1)]
  MR --> RI[Rentalitem (RI)]
  SQ1 --> RI
  RI --> NI[SQL NOT IN (2)]

```

The execution plan shows a join between the CUSTOMER (CS) and MOVIERENTAL (MR) tables. The MOVIERENTAL (MR) table is joined to a subquery (SQ1) and the RENTALITEM (RI) table. The subquery (SQ1) is joined to the RENTALITEM (RI) table. The RENTALITEM (RI) table is joined to a subquery (SQ2) labeled "SQL NOT IN (2)".

Below the SQL statement and execution plan, there are tabs for "Index Analysis", "Table Statistics", "Column Statistics And Histograms", and "Outlines". The "Index Analysis" tab is selected, showing a table with the following data:

Index Name	Table Owner	Table Name	Column Name
IDX_MOVIERENTAL_0	MOVIES	MOVIERENTAL	TOTALCH.
CUSTOMER_PK	MOVIES	CUSTOMER	CUSTOME
MOVIECOPY_PK	MOVIES	MOVIECOPY	MOVIECO
RENTALITEM_FKI	MOVIES	RENTALITEM	RENTALID
CUSTOMER_JE1	MOVIES	CUSTOMER	LASTNAMI

Additionally, there are Table Statistics, Column Statistics and Histograms, and Outlines/Plan Guides tabs.

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.

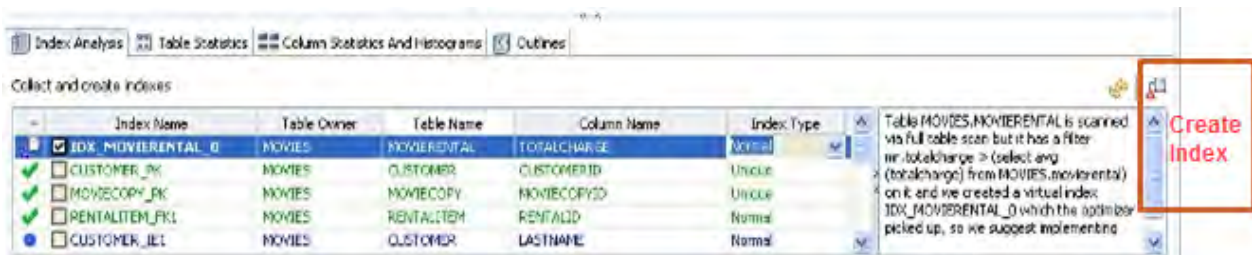
Next to the **Select statement of interest** list at the top, you choose to see an analysis of the **>ROOT** statement, or you can click the list and see an analysis of any one of the generated cases produced by running the tuning job from the **Overview** tab.

For more information, see [Visual SQL Tuning](#).

Implementing index analysis recommendations

Once you have added tuning candidates to a tuning job, SQL Query Tuner 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 SQL Query Tuner 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.

To view the index SQL in an editor for later implementation, click the statement and then click **Open in a SQL editor**.

To run the index SQL and create the index on the selected database, click **Execute**.

Visual SQL Tuning

Visual SQL Tuning is not available in SQL Query Tuner Developer.

SQL Query Tuner 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. This 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](#)
- [Interpreting the VST Diagram Graphics](#)

Changing Diagram Detail Display

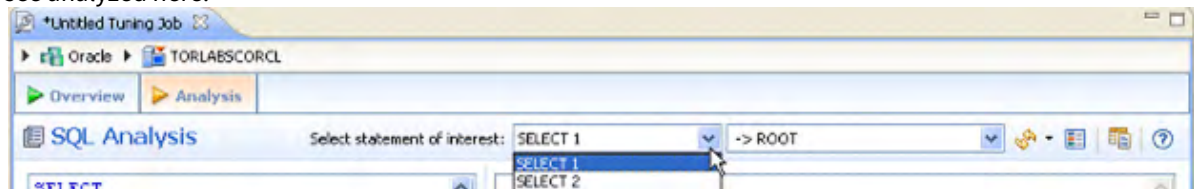
This section is comprised of the following topics:

- [Choosing the Tuning Statement and Generated Case to Analyze](#)
- [Viewing the VST Diagram Legend](#)
- [Viewing Table Counts and Ratios](#)
- [Viewing the Explain Plan](#)
- [Viewing the VST Diagram in Summary Mode](#)
- [Viewing the VST Diagram in Detail Mode](#)
- [Changing Detail Level for a Specific Table](#)
- [Viewing All Table Fields](#)
- [Viewing Diagram Object SQL](#)
- [Expanding Views in the VST Diagram](#)
- [Viewing the Oracle Explain Plan Overlay](#)

Choosing the Tuning Statement and Generated Case to Analyze

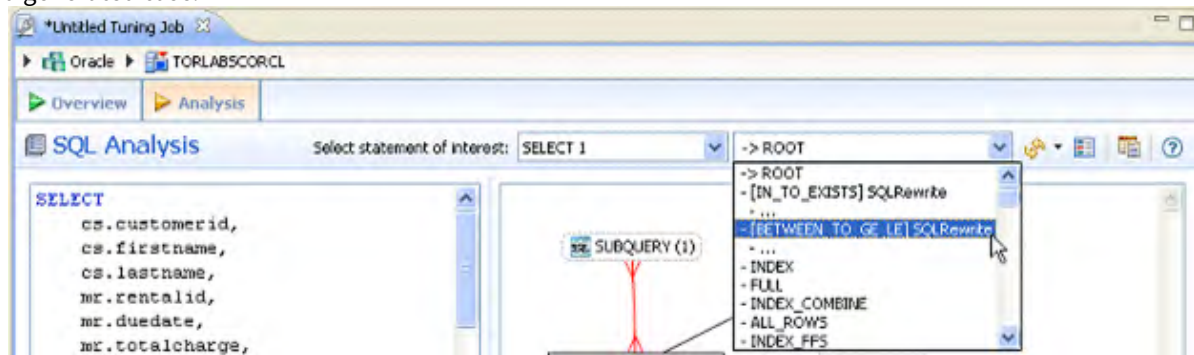
If, from the Overview tab, you have run the tuning job using more than one tuning statement, from the Analysis tab, you can choose to see the SQL analysis of any one of the tuning statements.

1. On the Analysis tab, click the **Select statement of interest** list and choose the tuning statement you want to see analyzed here.




Notice that next to the statement of interest box another list **->ROOT**. This shows that the statement being analyzed is the original statement, without any of the generated cases. This is the default selection.

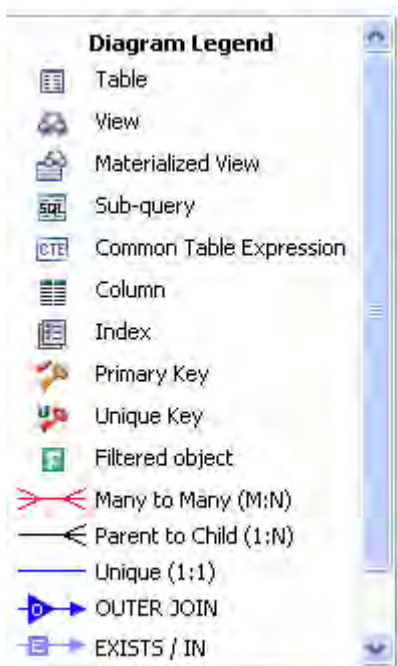
2. To choose the generated case to be analyzed, click the second **Select statement of interest** list and choose a generated case.



After you make your selections, a new analysis is performed taking into consideration the statement and case you chose. A new diagram is displayed and the Index Analysis, Table Statistics, Column Statistics And Histograms, and Outlines are recalculated and updated.

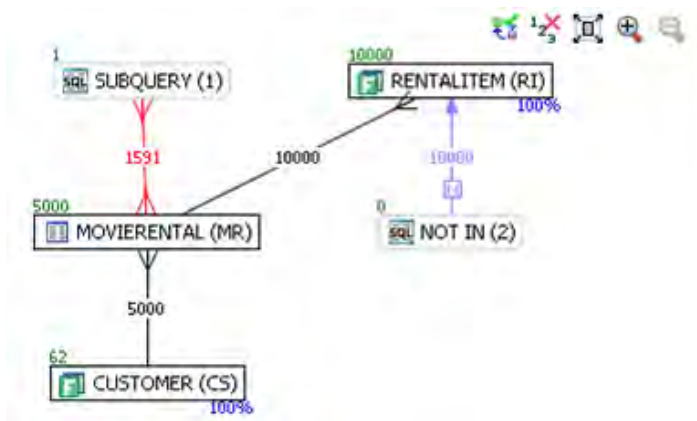
Viewing the VST Diagram Legend

Click the **Diagram Legend** toggle [] to view the legend and then click it again to hide it. All the icons used in the VST diagram are identified and in the Diagram Legend.



Viewing Table Counts and Ratios

To view or hide table counts, two table join sizes and filtered result set ratios, click the Ratios and Counts icon [].

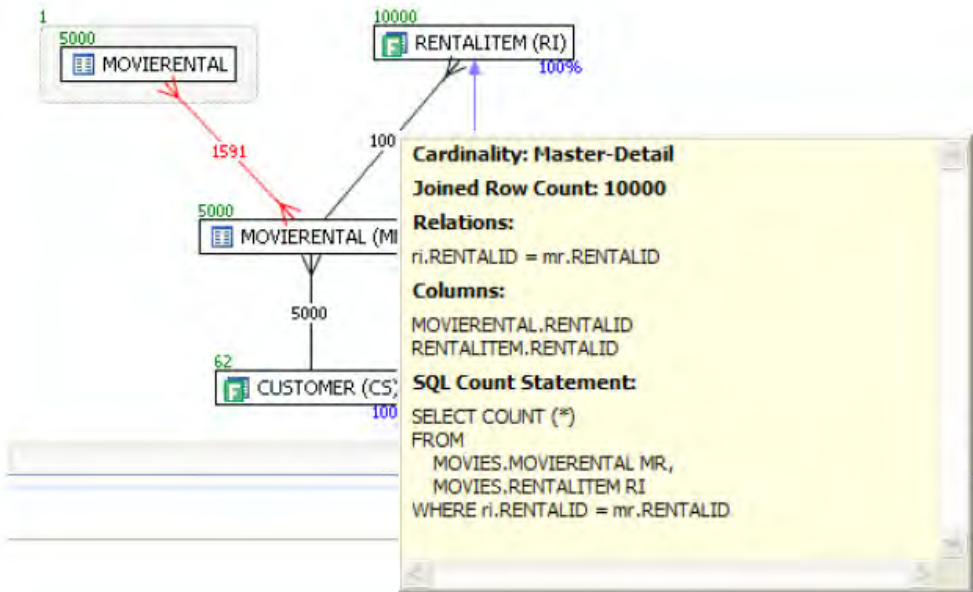


Green numbers at top left of table represent the total number of rows in that table. In the above the MOVIERENTAL (MR) table has 5000 rows.

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.

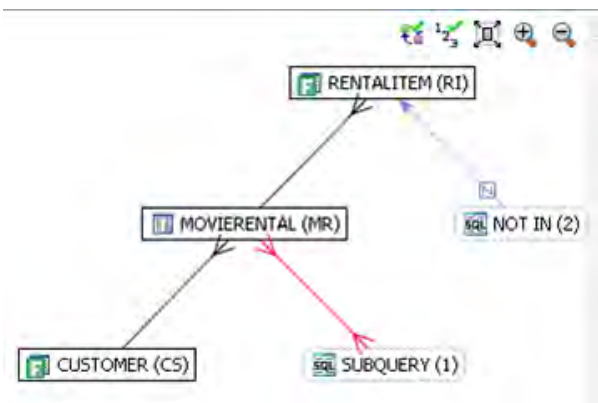
The numbers on the table joins indicate the total number of rows that meet the selection criteria for both tables.

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



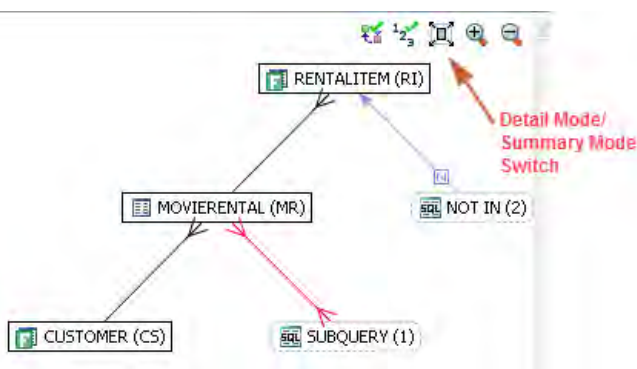
Viewing the VST Diagram in Summary Mode

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

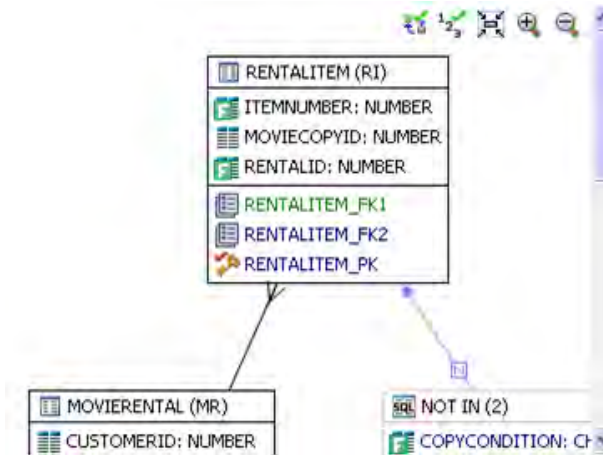


Viewing the VST Diagram in Detail Mode

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

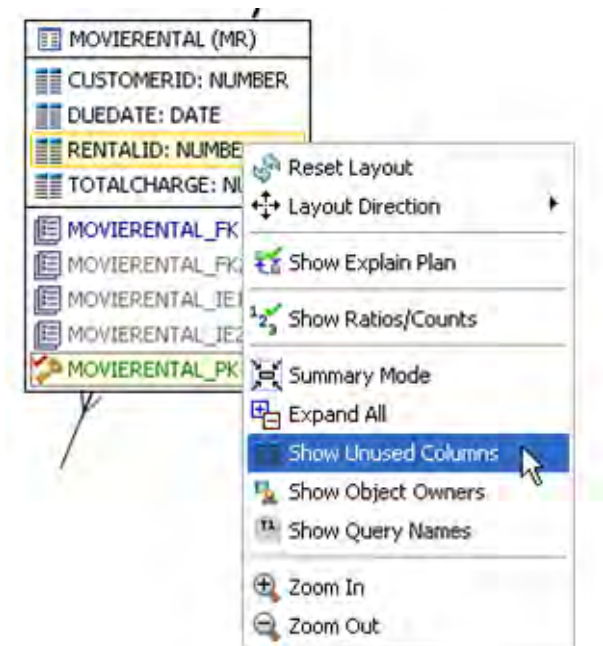


Changing Detail Level for a Specific Table

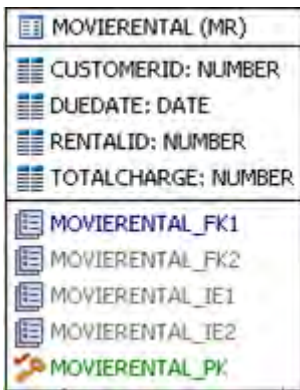
You can also switch between Summary Mode and Detail Mode for a specific table or view, by double-clicking the object name.

Viewing All Table Fields

By default, only fields that are used in the WHERE clause are displayed in detail mode; however, if you right-click the table you can choose to display even unused columns as follows:

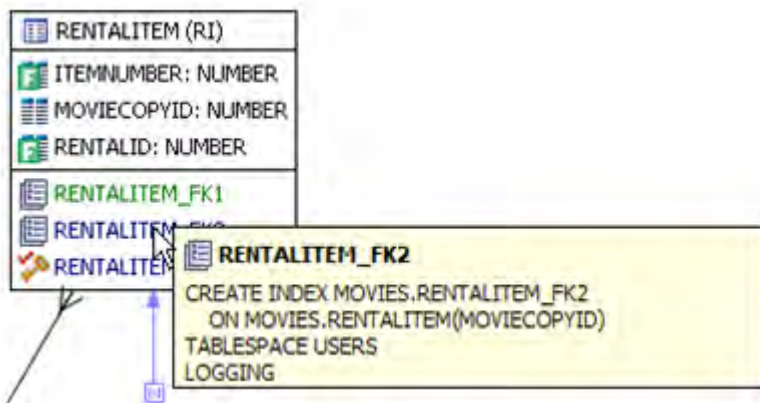


All the columns in the table are shown, and not just the ones used in the WHERE clause of the SQL statement.

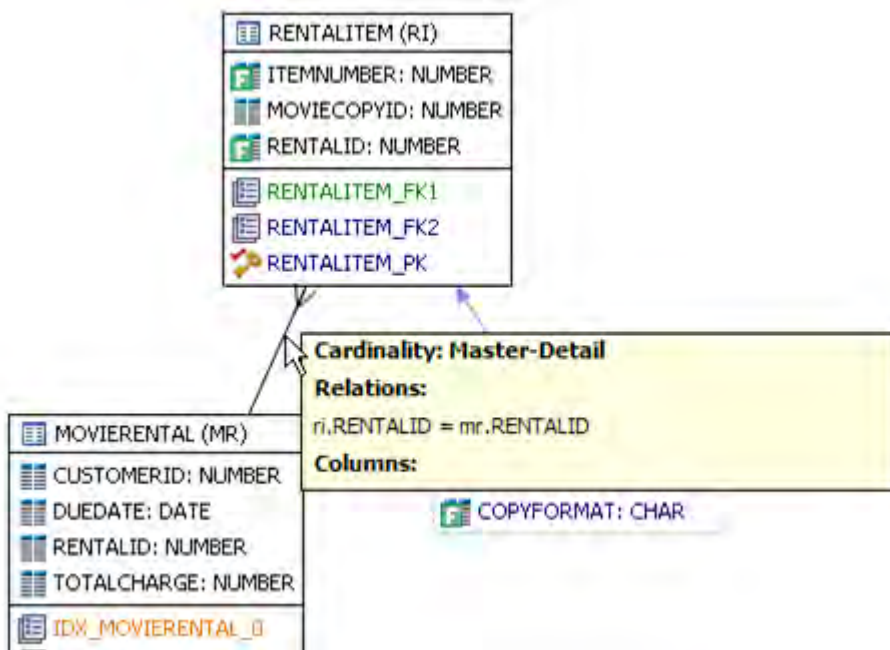


Viewing Diagram Object SQL

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



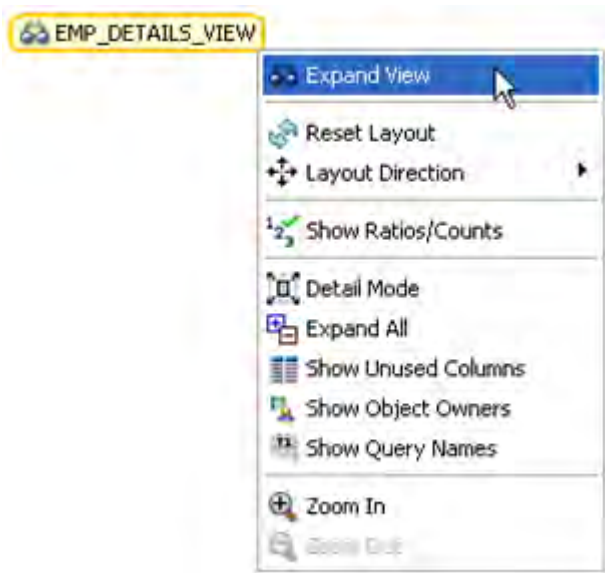
Hovering over the join between two tables displays the relationship between the two tables.



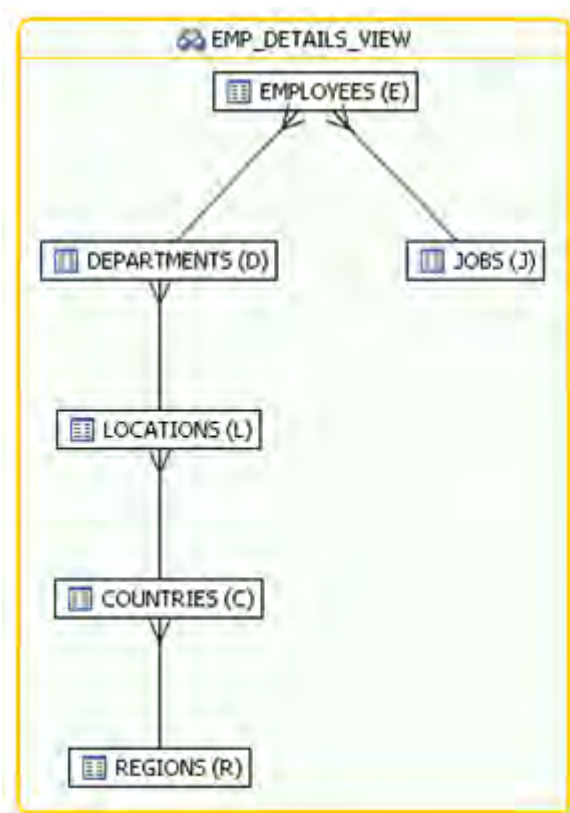
Expanding Views in the VST Diagram

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

Right click on the view, and choose **Expand View**.



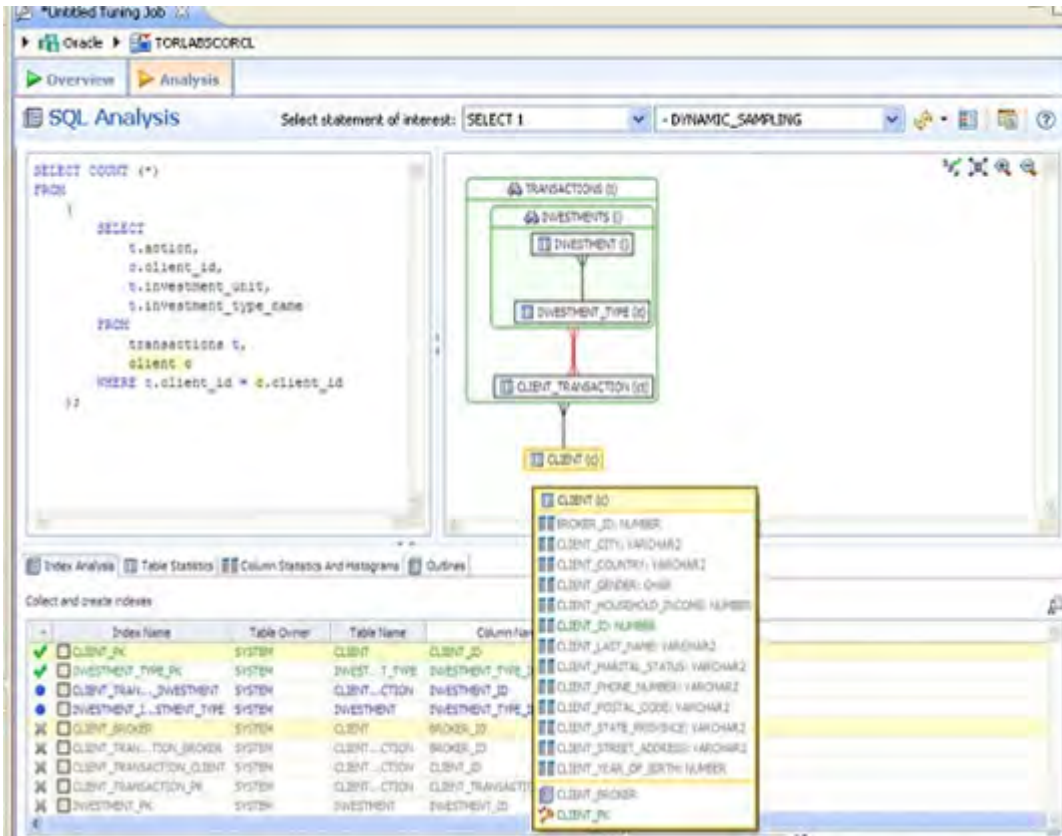
Now we can see the objects in the view:



You can further expand the sub-view within the original view also.


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

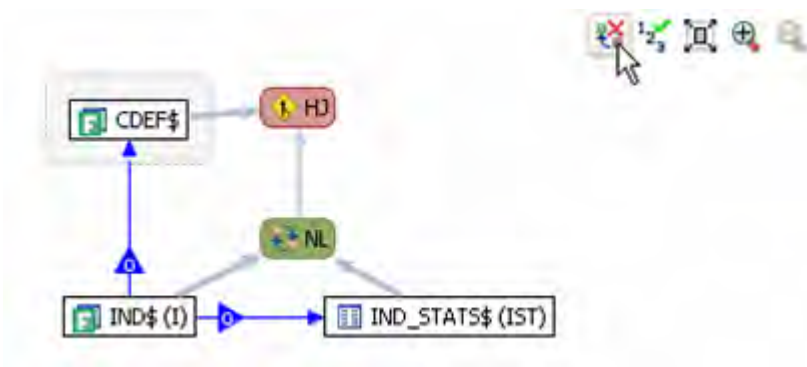
Notice in the view expansion a list of all the indexes on all the underlying tables in the views and sub views and which of those indexes is used in the default execution plan.



Viewing the Oracle Explain Plan Overlay

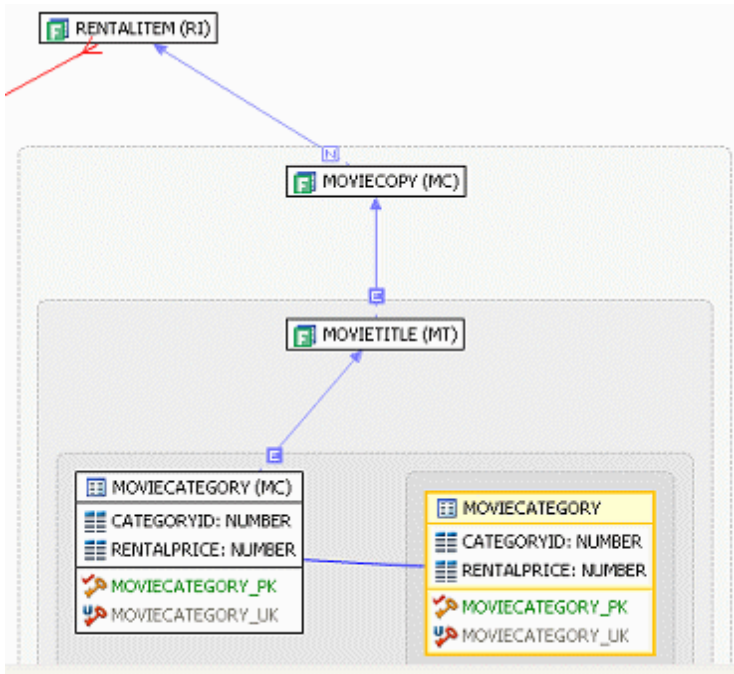
i This option is only available for Oracle versions 10 and higher.

Click the Explain Plan toggle [], you can choose to view or hide the Explain Plan details. The additional nodes shown in the Explain Plan overlay provide details on the flow of the query plan, with operations (such as nested loops, sorts, and joins) showing connecting tables and other operations. Hover the mouse over the objects or relationships to view additional details.



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.



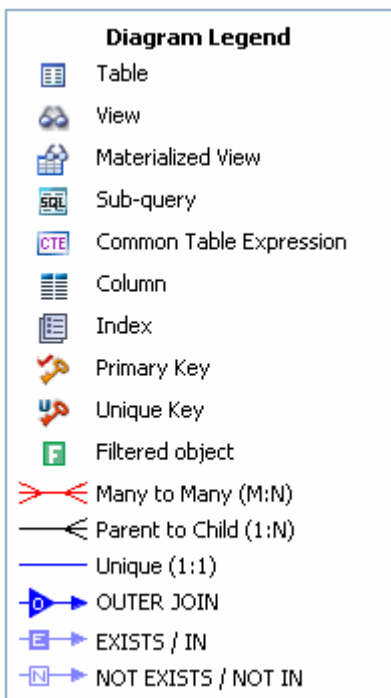
Interpreting the VST Diagram Graphics

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

- [Viewing the Diagram Legend](#)
- [Colors](#)
- [Connecting Lines/Joins](#)





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. SQL Query Tuner 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 .

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 .
	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 .
	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 .
	Indirect Relationship . For more information, see Indirect Relationship .
	Outer Join . For more information, see Outer Join .
	Unique . For more information, see Unique .
	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 .
	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 .

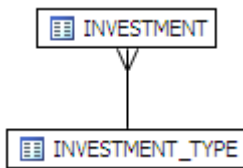
One-to-One Join

If two tables are joined on their primary key, then graphically, these would be laid out side-by-side, with a one-to-one connector:



One-to-Many Join

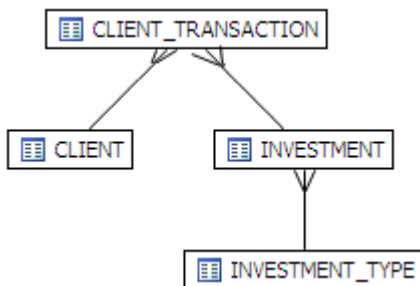
This is the default positioning of a one-to-many relationship, where INVESTMENT_TYPE is the master table and INVESTMENT is the details table.



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,
    INVESTMENT
  
```

```
OFFICE_LOCATION C,
INVESTMENT I
```

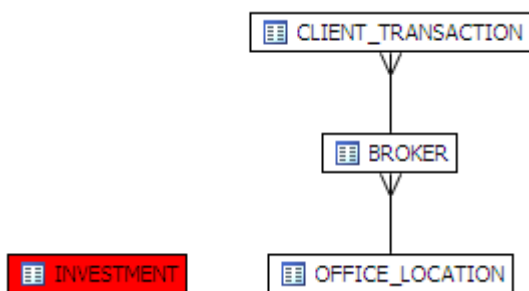
```
WHERE
```

```
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	Executi...istics	Other Execution Statistics		
Name	Text	Value	Elapsed Time (s)	Physical Reads	Logical Reads	CPU T
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	

i Transformations are highlighted in yellow.

Implied Cartesian Join

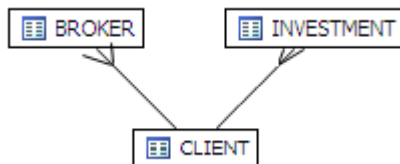
If there are different details for a master without other criteria then a Cartesian-type join is created:

```
SELECT *
FROM
```

```
investment i,
broker b,
client c
```

```
WHERE
```

```
b.manager_id=c.client_id and
i.investment_type_id=c.client_id;
```



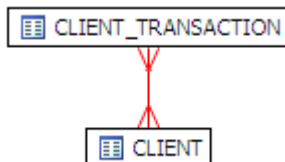
The result set of BROKER to CLIENT will be multiplied by the result set of INVESTMENT to CLIENT.

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:



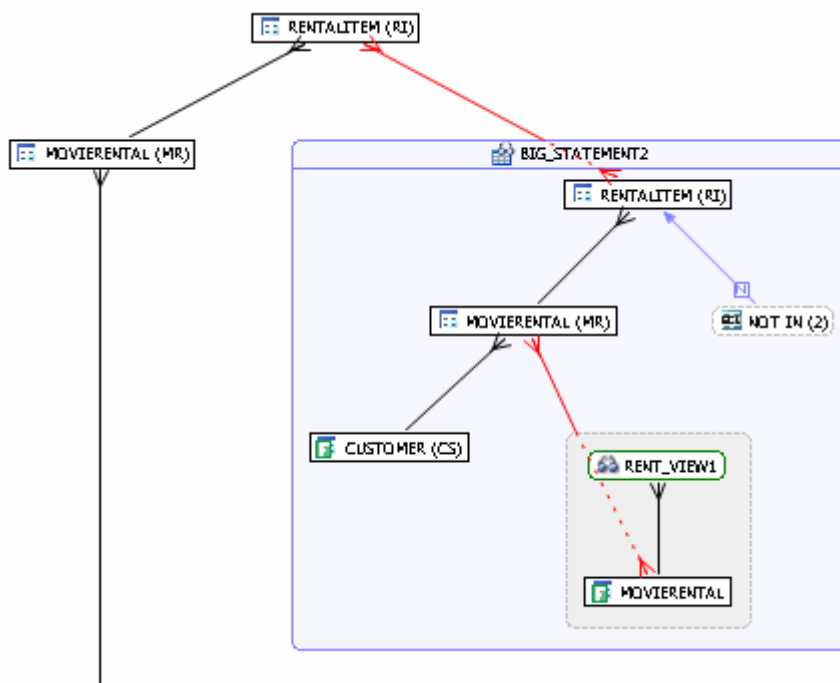
If one of the fields is unique, then the index should be declared as such to help the optimizer.

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) cs, (FROM MOVIES.customer c2
        WHERE
            c1.customerid <>c2.customerid AND c1.lastname=c2.lastname AND c1.phone
            BETWEEN 0 AND 9999569900)
        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

```

Outer Join

The bold SQL predicate in the statement below defines the outer join between customer and movierental.


```
select cs.*
```

```
from MOVIES.customercs, MOVIES.movierentalmr
```

```
where
```

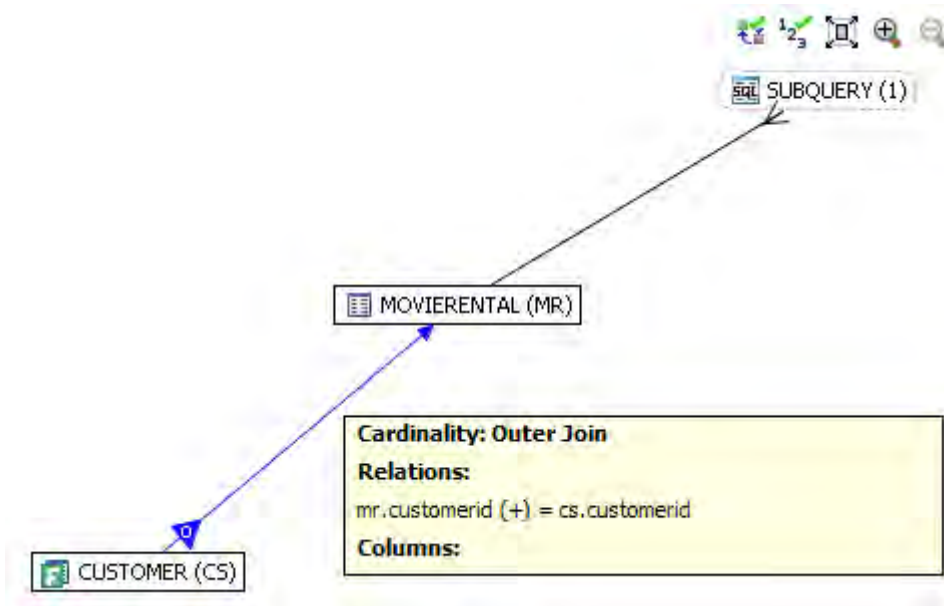
```
length(cs.lastname) = 8 and cs.zip > 75062 and 1 < cs.customerid + 2 and cs.phone between 9625569900
and 9999569900 and mr.rentalid = (select max(ri.rentalid)
```

```
from MOVIES.rentalitem ri, MOVIES.moviecopy mc
```

```
where
```

```
ri.itemnumber > 1 and mc.moviecopyid = 700) and mr.customerid(+)=cs.customerid;
```

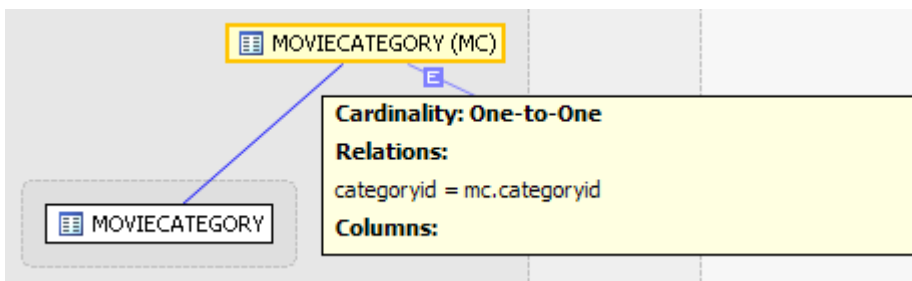
The following screen shot illustrates how the outer join is displayed in the VST diagram.



Unique

The subquery below illustrates a unique relationship between two primary keys.

```
...select max(rentalprice) from MOVIES.moviecategory where categoryid = mc.categoryid...
```



Not In or Not Exists Join

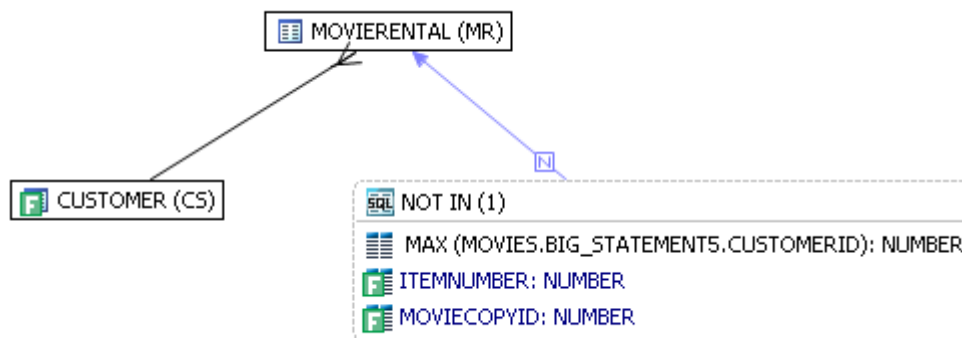
The following SQL contains a NOT IN subquery (shown in bold below) that is graphically represented with the Subquery summary icon and the NOT IN join.

```

SELECT CS.*
FROM
    MOVIES.CUSTOMERCS,
    MOVIES.MOVIERENTALMR
WHERE
    CS.ZIP > '75062'
    AND MR.RENTALID NOT IN (SELECT MAX ( MOVIES.BIG_STATEMENTS5.CUSTOMERID)
FROM
    MOVIES.RENTALITEMRI,
    MOVIES.MOVIECOPYMC,
    MOVIES.BIG_STATEMENTS5
WHERE
    RI.ITEMNUMBER>1
    AND MC.MOVIECOPYID=700)
    AND MR.CUSTOMERID=CS.CUSTOMERID;

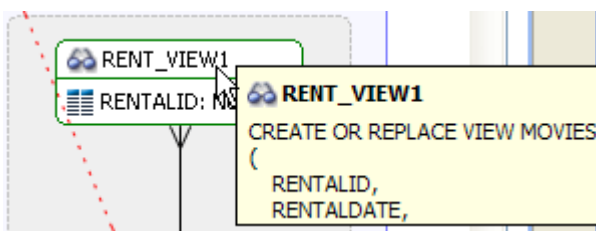
```

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.

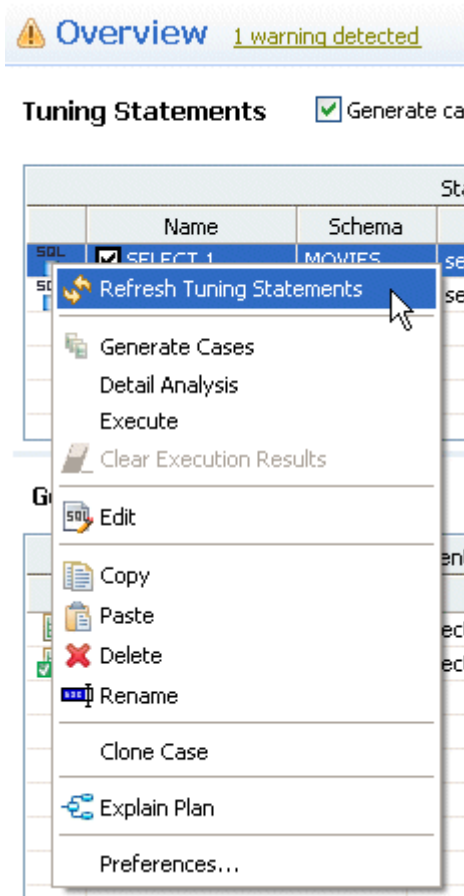


Refreshing Tuning Statements

At times you may see an error on the Overview page, which when you mouse over it, indicates that the tuning statements are out of sync and need to be refreshed. This can happen, for example, if you tune a statement, then delete it, and insert another SQL query for tuning.

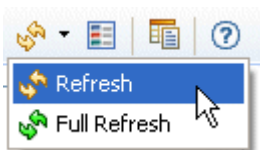
To refresh the tuning statements

In the Tuning Statements area of the Overview tab, right-click the tuning statement and select **Refresh Tuning Statements**.



Refreshing the VST Diagram

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. This option is typically used when the underlying objects have been recently changed.

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](#)
- [View Statement or Case Code in SQL Viewer](#)
- [Open an Explain Plan for a Statement or Case](#)
- [Executing a Session from the Command Line](#)
- [Saving a Tuning Job](#)

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 IDERA product that is integrated with SQL Query Tuner.


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. Right-click a statement or case and select **Edit**.
2. Use the **Data Type** and **Value** (or **NULL**) controls to specify the type and value for each bind variable. After setting bind variables, you can execute a case.

 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.

On Oracle version 9 and higher, SQL Query Tuner attempts to get the Explain Plan from V\$SQL_PLAN when possible. Otherwise, the Explain Plan is generated by the Oracle EXPLAIN PLAN command.

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		Result	Estimated Statistics					Actual Statistics
	Cost	Operation Cost		Cardinality	Bytes	CPU Cost	IO Cost	Optimizer	
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 ACCESS (INDEX) TABLE A...IENT1	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.

With the **Explain Plan** tab open, you can quickly switch the view to an explain plan for another SQL statement.

To change to another SQL statement

- Click in the **Name** field of another statement record showing a value in the **Cost** field.

Executing a Session from the Command Line

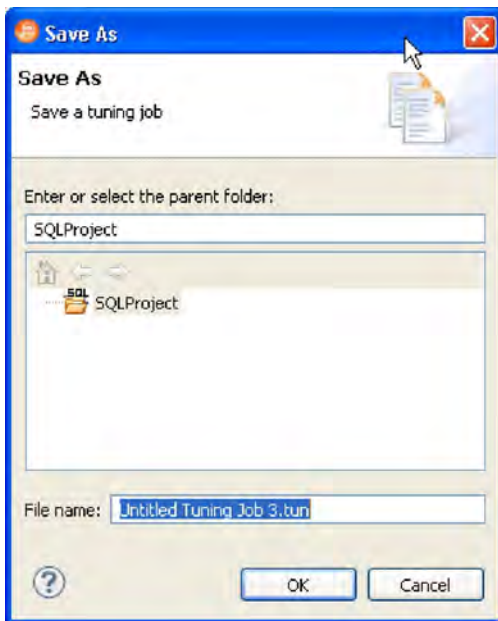
You can launch a tuning job from the command line using the following syntax:

```
dboptimizer.exe tune ds:ROMLABORCL10G_1 sqlfile: C:\dboptimizer\workspace\test.sql
```

In the above command, the user has specified ROMLABORCL10G_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.



i Tuning sessions can be saved as .tun 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.

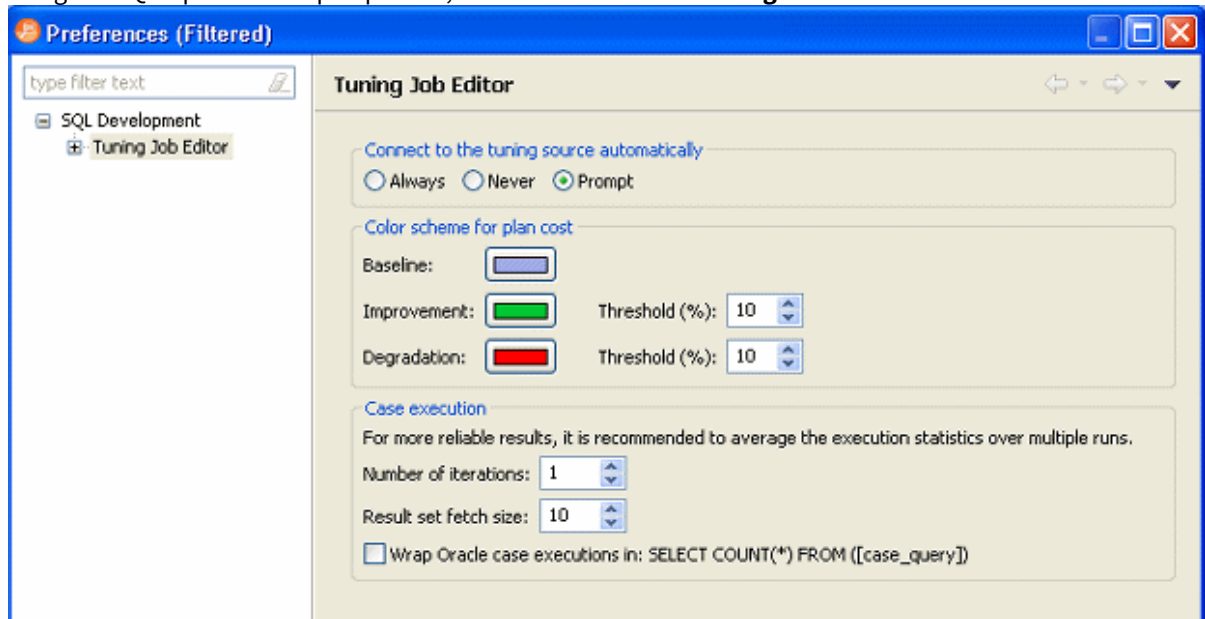
This section is comprised of the following topics:

- [Configuring Tuning: Specify Tuning Job Editor Preferences](#)
- [Specify case generation preferences](#)
- [Specify VST Diagrams Tuning Preference](#)

Configuring Tuning: Specify 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.

- Using the SQL Optimization perspective, select **Preferences > Tuning Job Editor**.



- Make your changes and then to save your changes, click **Apply**.

The following describes the options available:

General Preferences

Connect to the tuning source automatically: 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:

- **Always:** Automatically connects to data sources associated with tuning jobs that were open last time you shut down tuning.
- **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.
- **Prompt:** Prompts you to connect to data sources associated with tuning jobs that were open last time you shut down the application.

Color scheme for plan cost: 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.

Information: You can set the threshold in the application preferences, by selecting **Window > Preferences > Tuning Job Editor** and then changing the threshold levels as required.

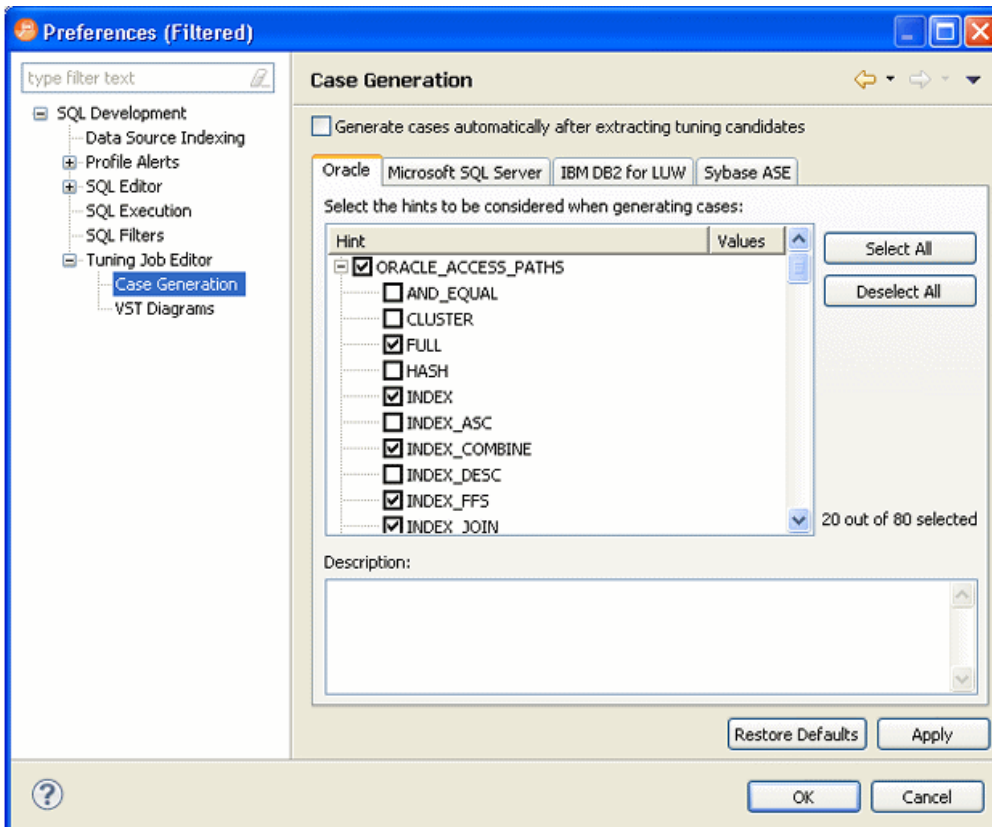
Case execution: Lets you dictate how execution statistics are gathered.

See Also

- [Specify Case Generation Preferences](#)
- [Specify VST Diagrams Tuning Preferences](#)

Specify case generation 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.



Using the SQL Optimization perspective, select **Preferences > Tuning Job Editor > Case Generation**.

Use the Case Generation 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	Identifiers for the generated statement or case: <ul style="list-style-type: none"> • Name: Statements are assigned a numbered identifier based on the order in which they were added to a tuning job. • Text: An excerpt of the statement or case based on the statement type. For details on how to view the entire statement or case.

Column or Column Set	Description
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.</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>Against Oracle datasources, the default, collapsed view has Physical Reads and Logical Reads columns. Expanded, there are also Consistent Gets, Block Gets, and Rows Returned, CPU time(s), Parse CPU Time(s), Row Sorts, Memory Sorts, Disk Sorts, and Open Cursors columns.</p> <p>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>

Specify VST Diagrams Tuning Preference

The preferences on this page allow you to change the default presentation of count information and sub-query names in Visual SQL Tuning diagrams.

Using the SQL Optimization perspective, select **Preferences > Tuning Job Editor > VST Diagrams**.

- **Show count information.** If enabled, shows the ratios and count information when the VST diagram is generated. If not enabled, you must click the **Ratios/Counts** icon () on the VST diagram to view ratio and count information.
- **Hide sub-query names when expanded.** If enabled, shows the sub-query name when the VST diagram is generated. If not enabled, you must right-click anywhere in the VST and from the menu that appears, choose Show Query Names.

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
<pre>SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id (+) AND c.state = 'CA'</pre>	<pre>SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id (+) AND c.state(+) = 'CA'</pre>

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 NULL</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.

Reference

The following topics provide reference details:

- [Database objects](#)
- [JDBC Connection Parameters](#)

Database objects

The following table describes the database objects displayed in SQL Query Tuner and contains information regarding each one, including object name, DBMS platform, and any notes pertaining to the specified object.

In SQL Query Tuner, database objects are stored in Data Source Explorer as subnodes of individual, pertinent databases.

Database Object	Notes
Check Constraints	<p>A check constraint is a search condition applied to a table.</p> <p>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>
Foreign Keys	<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>
Groups	<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	Notes
Indexes	<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>
Packages	<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>

Database Object	Notes
Primary Keys	<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>
Procedures	<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>
Tables	<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	Notes
Triggers	<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>
Unique Keys	<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>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	<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>

JDBC Connection Parameters

Connection Parameter	Description
Connect String	<p>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.</p> <p>For example:</p> <pre>jdbc:postgresql://host:port/database jdbc:derby://host:port/database</pre>
Data Source Name	<p>The name of the data source to which you want SQL Query Tuner to connect.</p>