Embarcadero® DB Optimizer™ 1.0 User Guide

Contents

Welcome to Embarcadero DB Optimizer

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

Technical Requirements

Before installing DB Optimizer, verify that your environment meets the following requirements.

Hardware Requirements

The following minimum hardware requirements are required to run DB Optimizer:

- Pentium 4-Level Processor
- 1024 MB of memory
- 500 MB of disk space
- 1024 x 768 screen resolution

Operating System Requirements

DB Optimizer supports the following operating systems:

- Microsoft Windows XP (x86-32, Win32)
- Microsoft Vista (x86-32, Win32)
- Microsoft Windows Server 2003
- Red Hat Enterprise Linux 5.0, x86-32, GTK 2
- SuSe Linux Enterprise Server (x86) GTK+ 2.x

DBMS Support

DB Optimizer provides native support for the following platforms (no additional DBMS client software is required):

- Generic JDBC
- IBM DB2 LUW 8.0 - 9.0
- Microsoft SQL Server 2000 and 2005
- Oracle 8i - 11g
- Sybase 12.5 - 15.0

Additional Product Resources

The Embarcadero Web site is an excellent source for additional product information, including white papers, articles, FAQs, discussion groups, and the Embarcadero Knowledge Base.

Go to www.embarcadero.com/support, or click any of the links below, to find:

- Documentation
- Online Demos
- Technical Papers
- Discussion Forums
- Knowledge Base

Embarcadero Technologies Technical Support

If you have a valid maintenance contract with Embarcadero Technologies, the Embarcadero Technical Support team is available to assist you with any problems you have with our applications. Our maintenance contract also entitles registered users of Embarcadero Technologies' products to download free software upgrades during the active contract period.

To save you time, Embarcadero Technologies maintains a Knowledge Base of commonly-encountered issues and hosts Discussion Forums that allow users to discuss their experiences using our products and any quirks they may have discovered.

To speak directly with Embarcadero Technical Support, see Contacting Embarcadero Technologies Technical Support below.

NOTE: Evaluators receive free technical support for the term of their evaluation (14 days).

Contacting Embarcadero Technologies Technical Support

When contacting Embarcadero Technologies Technical Support please provide the following to ensure swift and accurate service:

Personal Information

- Name
- Company name and address
- Telephone number
- Fax number
- Email address

Product and System Information

- Embarcadero product name and version number. This information is found under Help, About.
- Your client operation system and version number.
- Your database and version number.

Problem Description

A succinct but complete description of the problem is required. If you are contacting us by telephone, please have the above information, including any error messages, available so that an Embarcadero Technical Support Engineer can reproduce the error and clearly understand the problem.

There are three ways to contact Embarcadero's Technical Support department:

- Via the Web
- Via Phone (for Standard and Professional editions only)
- Via Email

Via the Web

Embarcadero Technical Support provides an online form that lets you open a Support case via the Web. To access this form, go to http://www.embarcadero.com/support/open_case.jsp.

We usually acknowledge the receipt of every case on the same day, depending on the time of submission.

Via Phone

Telephone support is available for the Standard and Professional editions only.

United States

Embarcadero Technologies Technical Support phone number is (415) 834-3131. Select option 2 and then follow the prompts. The hours are Monday through Friday, 6:00 A.M. to 6:00 P.M. Pacific time.

For licensing issues, including Product Unlock Codes, call (415) 834-3131. Select option 2 and then follow the prompts. The hours are Monday through Friday, 6:00 A.M. to 6:00 P.M. Pacific time.

The Embarcadero Technologies Technical Support fax number is (415) 495-4418.

EMEA

Embarcadero Technologies Technical Support phone number is +44 (0)1628 684 499. The hours are Monday to Friday, 9 A.M. to 5:30 P.M. U.K. time.

For licensing issues, including Product Unlock Codes, call +44 (0)1628-684 494. The hours are Monday to Friday, 9 A.M. to 5:30 P.M. U.K. time.

The Embarcadero Technologies Technical Support fax number is +44 (0)1628 684 401.

Via Email

United States

Depending on your needs, send your email to one of the following:

- support@embarcadero.com - Get technical support for users and evaluators.
- upgrade@embarcadero.com - Request upgrade information.
- key@embarcadero.com - Request a product key.
- wish@embarcadero.com - Make a suggestion about one of our products.

EMEA

Depending on your needs, send your email to one of the following:

- uk.support@embarcadero.com - Get technical support for users and evaluators.
- uk.upgrade@embarcadero.com - Request upgrade information.
- uk.key@embarcadero.com - Request a product key.
- uk.wish@embarcadero.com - Make a suggestion about one of our products.

DB Optimizer Overview

Embarcadero DB Optimizer simplifies SQL development for application developers, with many features for improving productivity and reducing errors. Key features include:

A rich SQL IDE with data source and project management facilities, key DB Optimizer features include:

- Running profiles against data sources to locate and diagnose wait-based bottlenecks and inefficiencies.
- Tuning multiple SQL statements using explain plan costing, execution statistics, and SQL Optimizer hint-based suggestions
- Migrating data sources from Eclipse DTP (Data Tools Project) or Quest TOAD or automatically detecting them on the network.
- Data Source Explorer enables users to easily navigate, search, extract DDL, execute commands, and browse outline views without opening SQL files.
- Code Assist provides context-sensitive database object names to eliminate misspellings and the need to look them up. Also available offline.
- Formatting Profiles ensure consistent, quality code layout for easy review and extension. Profiles can be customized and shared.
- Real-time SQL validation ensures there are no parser violations or object names used that are not found in the database.

Using DB Optimizer

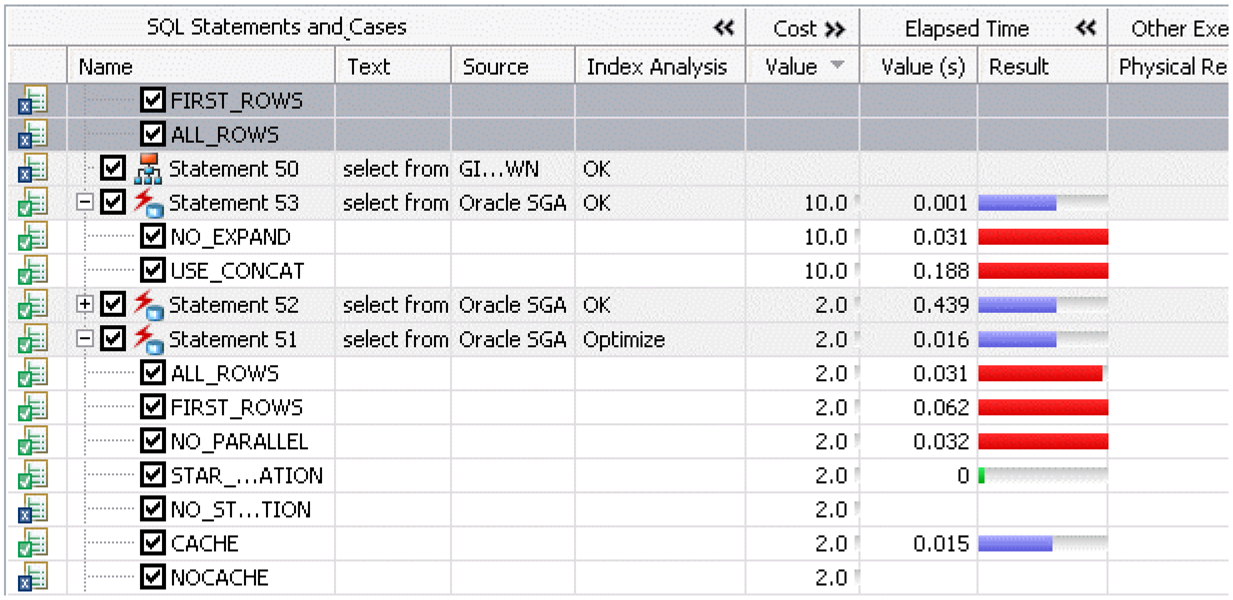
This section contains detailed instructions for Working with Data Sources, Working with Projects, Tuning SQL Statements, Profiling Data Sources, Creating and Editing SQL Files (SQL Editor), Executing SQL Files, and Troubleshooting.

Tuning SQL Statements

The SQL Tuner feature lets you view cost details on groups of SQL statements on an Oracle data source. As a means of identifying bottlenecks or problem queries, SQL Tuner lets you:

- View high-level explain plan costs for statements as well as offering the option to view operation-by-operation explain plan details
- Generate execution statistics such as CPU time, Read and Sort statistics, and other details
- View explain plan and execution statistics for automatically-generated SQL Optimizer hint-based variations and a transformation-based variation on all statements in a tuning job

Used as a development aid, SQL Tuner lets you fine tune queries before putting them into production. As a maintenance tool, SQL Tuner can help you isolate and optimize queries in production systems.



SQL Statements and Cases				Cost	Elapsed Time		Other Exe
Name	Text	Source	Index Analysis	Value	Value (s)	Result	Physical Re
FIRST_ROWS							
ALL_ROWS							
Statement 50	select from	GI…WN	OK				
Statement 53	select from	Oracle SGA	OK	10.0	0.001	🟦	
NO_EXPAND				10.0	0.031	🟥	
USE_CONCAT				10.0	0.188	🟥	
Statement 52	select from	Oracle SGA	OK	2.0	0.439	🟦	
Statement 51	select from	Oracle SGA	Optimize	2.0	0.016	🟦	
ALL_ROWS				2.0	0.031	🟥	
FIRST_ROWS				2.0	0.062	🟥	
NO_PARALLEL				2.0	0.032	🟥	
STAR_…ATION				2.0	0	🟩	
NO_ST…TION				2.0			
CACHE				2.0	0.015	🟦	
NOCACHE				2.0			

Support for simultaneous tuning of multiple statements lets you perform high-level, “at a glance” comparisons as well as detailed, low-level analysis. In combination with the ability to save your tuning job, this lets you group logically-related or functionally-related code, revisit your analysis, add or remove SQL, and routinely generate new statistics for critical systems.

For information on how to use SQL Tuner, see the following topics:

- Preliminary steps in setting up the SQL Tuner feature
- Running Tuning Jobs

Preliminary steps in setting up the SQL Tuner feature

Prior to starting to run tuning jobs, you must perform the following setup tasks:

- Setting up roles/permissions required to use the SQL Tuner feature
- Set SQL Tuner Feature Preferences

Setting up roles/permissions required to use the SQL Tuner feature

In order to take advantage of all SQL Tuner features, each user must have a specific set of permissions. The SQL below creates a role with all required permissions. To create the required role, execute the SQL against the target Oracle data source, modified according to the specific needs of your site:

```
/* Create the role */
CREATE ROLE SQLTUNING NOT IDENTIFIED
/
GRANT SQLTUNING TO "CONNECT"
/
GRANT SQLTUNING TO SELECT_CATALOG_ROLE
/
GRANT ANALYZE ANY TO SQLTUNING
/
GRANT CREATE ANY OUTLINE TO SQLTUNING
/
GRANT CREATE ANY PROCEDURE TO SQLTUNING
/
GRANT CREATE ANY TABLE TO SQLTUNING
/
GRANT CREATE ANY TRIGGER TO SQLTUNING
/
GRANT CREATE ANY VIEW TO SQLTUNING
/
GRANT CREATE PROCEDURE TO SQLTUNING
/
GRANT CREATE SESSION TO SQLTUNING
/
GRANT CREATE TRIGGER TO SQLTUNING
/
GRANT CREATE VIEW TO SQLTUNING
/
GRANT DROP ANY OUTLINE TO SQLTUNING
/
GRANT DROP ANY PROCEDURE TO SQLTUNING
/
GRANT DROP ANY TRIGGER TO SQLTUNING
/
GRANT DROP ANY VIEW TO SQLTUNING
/
GRANT SELECT ON SYS.V_$SESSION TO SQLTUNING
/
GRANT SELECT ON SYS.V_$SESSTAT TO SQLTUNING
/
GRANT SELECT ON SYS.V_$SQL TO SQLTUNING
/
GRANT SELECT ON SYS.V_$STATNAME TO SQLTUNING
/
```

Once complete, you can assign the role to users who will be running tuning jobs:

```
/* Create a sample user*/
CREATE USER TUNINGUSER IDENTIFIED BY VALUES '05FFD26E95CF4A4B'
    DEFAULT TABLESPACE USERS
    TEMPORARY TABLESPACE TEMP
    QUOTA UNLIMITED ON USERS
    PROFILE DEFAULT
    ACCOUNT UNLOCK
/
GRANT SQLTUNING TO TUNINGUSER
/
ALTER USER TUNINGUSER DEFAULT ROLE SQLTUNING
/
```

Set SQL Tuner Feature Preferences

The following preferences directly influence the behavior of the SQL Tuner feature:

- Caching Required Object Definitions
- Setting Tuning job Editor Preferences
- Specifying Generated Cases Preferences

Caching Required Object Definitions

When connecting to a data source, the application caches a subset of the object definitions on the data source. SQL Tuner feature preferences allow you to modify the types of objects for which definitions are cached. To properly process transformations, a specific set of database object definitions must be cached.

When not running tuning jobs and taking advantage of other SQL Tuner functionality, SQL editing for example, you might disable caching of some object definitions. You may have done this to speed up data source caching for example, or because some object definitions were not necessary to the task at hand. If you are going to run tuning jobs however, you must ensure that SQL Tuner is caching required objects when connecting to a data source.

To ensure SQL Tuner automatically caches required object definitions when connecting to a data source:

1 On the **Window** menu, choose **Preferences**.

A **Preferences** dialog opens.

2 In the left-hand pane expand the **SQL Development** item and then click **Cache Configuration**.

3 Select the check boxes associated with the following list of minimally-required object definitions:

- **Foreign keys**
- **Functions**
- **Indexes**
- **Materialized view**
- **Primary keys**
- **Procedures**
- **Stored outline**
- **Tables**
- **Unique keys**
- **Views**

4 Click **OK**.

Setting 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.

To set Tuning Job editor preferences:

1 On the **Window** menu, choose **Preferences**.

A **Preferences** dialog opens.

2 In the left-hand pane expand the **SQL Development** node and then click **Tuning Job Editor**.

3 Use the following table as a guide to setting the Tuner Job editor preferences:

Option	Description
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 SQL Tuner. **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, the SQL Tuner feature 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. For more information, see Understanding the Graphical Representation of Cost and Elapsed Time.
Case execution	Lets you dictate how execution statistics are gathered. For more information, see Executing Statements and Cases on the Generated Cases Tab.
Table analysis	Lets you specify an estimation sample percentage to be used with the Analyze Tables function.

4 Click **OK**.

Specifying Generated Cases Preferences

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. For more information, see Working with Generated Cases.

To specify preferences for automatic hint generation:

1 On the **Window** menu, choose **Preferences**.

A **Preferences** dialog opens.

2 In the left-hand pane expand the **SQL Development** item, expand the **Tuning Job Editor** item, and then click **Case Generation**.

3 Use the **Generate cases automatically after extracting tuning candidates** control to enable or disable automatic generation of hint-based cases.

4 Use the hint type check boxes to specify which hint-based cases are generated for a statement added to a tuning job.

5 Click **OK**.

Running Tuning Jobs

Prior to running tuning jobs, you should set tuning job preferences. For details, see Set SQL Tuner Feature Preferences.

A tuning job consists of a set of SQL statements and any analysis results you generated against an Oracle data source using the SQL Tuner feature. The SQL statements and analysis results that make up a tuning job can be saved in a tuning file (**.tun** suffix). This lets you open the a tuning job at a later time for inspection and analysis, add, delete, or modify SQL statements, or generate new execution statistics. For details, see Opening a Tuning Job and Closing and Saving Tuning Jobs.

In adding SQL candidates to tuning job, SQL Tuner lets you type or paste SQL code or add SQL extracted from files, database objects, and other data source resources. For more information on adding SQL candidates to a tuning job for analysis, see Specifying the SQL for a Tuning Job.

As you add SQL candidates, SQL Tuner-supported DML statements (SELECT, INSERT, DELETE, and UPDATE) are parsed out and added to the **Generated Cases** tab of the Profile editor. Where appropriate, hint-based cases of each DML statement are also added as well as a transformation-based case. The **Generated Cases** tab is where you perform your SQL Analysis. You can:

- View explain plans and associated costs
- Set bind variables and execute statements or cases to view elapsed time and other key execution statistics such as physical and logical reads.
- Use additional analysis options such as statement comparisons and index optimization suggestions

For detailed information on working with generated cases, see Working with Tuning Results on the Generated Cases Tab.

Opening a Tuning Job

If you want to start a new tuning job or open an existing tuning job file, follow the steps in the following tasks, as appropriate:

- Creating a New Tuning Job
- Opening a Saved Tuning Job

Creating a New Tuning Job

Tuning jobs can be saved in a file (**.tun** suffix) and consist of a set of tunable SQL statements associated with a single data source as well as statistics and other results gathered. The first step in creating a new tuning job is to open the Tuning Job editor and provide basic information such as a job name and the data source associated with the job.

To create a new tuning job:

1 On the **File** menu, select **New** and then select **Other**.

NOTE: As a shortcut, if you are already in a tuning perspective, you can click the New button on the toolbar.

A **New** dialog opens.

2 Select **Tuning Job** and then click **Finish**.

The Tuning Job editor opens.

3 In the **General Information** area, provide a **Name** and **Description** for the tuning job.

NOTE: When you close the tuning job or shut down the application, you are prompted to save the tuning job with the name you specify in the **Name** field. For more information, see Closing and Saving Tuning Jobs.

You are now ready to specify the SQL that you want to profile for this tuning job. For details, see Specifying the SQL for a Tuning Job.

You can also save your new tuning job. For details, see Closing and Saving Tuning Jobs.

Opening a Saved Tuning Job

A tuning job can be saved in a file with a **.tun** suffix. It consists of the SQL candidates you have added to the tuning job along with the most recently collected statistics. For more information, see Closing and Saving Tuning Jobs.

You can subsequently open the tuning archive to inspect results, add or delete SQL statements, generate new statistics, or work with other tuning job options.

To open an existing tuning job file, use one of the following techniques:

- On the **File** menu, select **Open** and then use the **Open File** dialog to locate and select the tuning job archive.
- Use the SQL Project Explorer to locate and open the tuning job archive.

  The tuning job archive opens in the Tuning Job editor.

After opening an existing tuning job, you can:

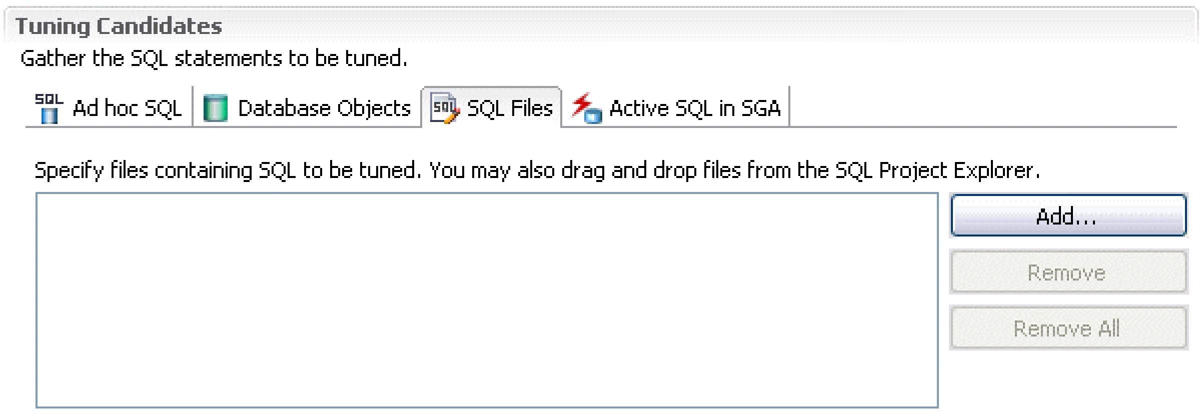
- Optionally specify additional SQL that you want to profile. For details, see Specifying the SQL for a Tuning Job.
- Work with tuning job results. For details, see Working with Tuning Results on the Generated Cases Tab.

Specifying the SQL for a Tuning Job

After Creating a New Tuning Job or Opening a Saved Tuning Job, you can add the SQL statements that are to be tuned. All standard DML statements can be tuned (SELECT, INSERT, DELETE, UPDATE). Your tuning job can include:

- Typed or pasted SQL
- SQL statements extracted from supported SQL-containing database objects
- Supported statements included in entire SQL Files
- For Oracle only, SQL statements active in the System Global Area (SGA)

The tabs in the **Tuning Candidates** area of the Tuning Job editor reflect these sources.



Gather the SQL statements to be tuned.

Specify files containing SQL to be tuned. You may also drag and drop files from the SQL Project Explorer.

TIP: 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.

To add SQL to a tuning job:

1 Open an existing tuning job or create a new one. For details, see Opening a Tuning Job.

2 In the **Tuning Candidates** view, add SQL as follows:

- Click on the **Ad Hoc SQL** tab and type or paste SQL
- Click on the **Database Objects** tab and in the Data Source Explorer, expand a connected data source tree down to the level of individual objects of valid SQL-containing types:
  - Functions
  - Materialized Views
  - Packages
  - Package bodies
  - Procedures
  - Stored Outlines
  - Triggers
  - Views

  You can then drag objects containing SQL from the Data Source Explorer into the **Overview** tab.

  NOTE: Alternatively, after clicking the **Database Objects** tab, you can click the **Add** button. This opens an dialog that lets you search objects by name, and select specific objects from the results to add to the tuning job.

- Click the **SQL Files** tab and then either click the **Workspace** button to open a dialog that lets you open a SQL file that you have in your workspace or click the **File System** button to open a file residing elsewhere in the file system.

- Click the **Active SQL in SGA** tab and then click the **Scan** button. This opens a **Scan SGA** wizard that lets you set filtering criteria for an SGA scan and then run the scan. You can then select the specific SQL statements that satisfied the filtering criteria, and add them to the tuning job.

As you add SQL items to the **Tuning Candidates** view, the SQL is parsed and the extracted SELECT, INSERT, UPDATE, and DELETE statements are added to the **Generated Cases** tab. For information on interpreting and working with these results, see Working with Tuning Results on the Generated Cases Tab.

Working with Tuning Results on the Generated Cases Tab

As you add SQL to the **Overview** tab of a tuning job, tuning results for statements are added to the **Generated Cases** tab. For information on adding SQL to a tuning job, see Specifying the SQL for a Tuning Job.

To view the current results for a tuning job:

1 Click the **Generated Cases** tab, located at the bottom of the **Tuning Candidates** view.

In general, the **Generated Cases** tab displays a numbered record for each SELECT, INSERT, DELETE, or UPDATE statement extracted from the SQL sources specified on the **Overview** tab. In the case of chained queries, in which SELECT statements are chained using set operators (UNION ALL, MINUS, INTERSECT), a numbered record is generated for each distinct SELECT statement within the chained query and can be tuned separately.

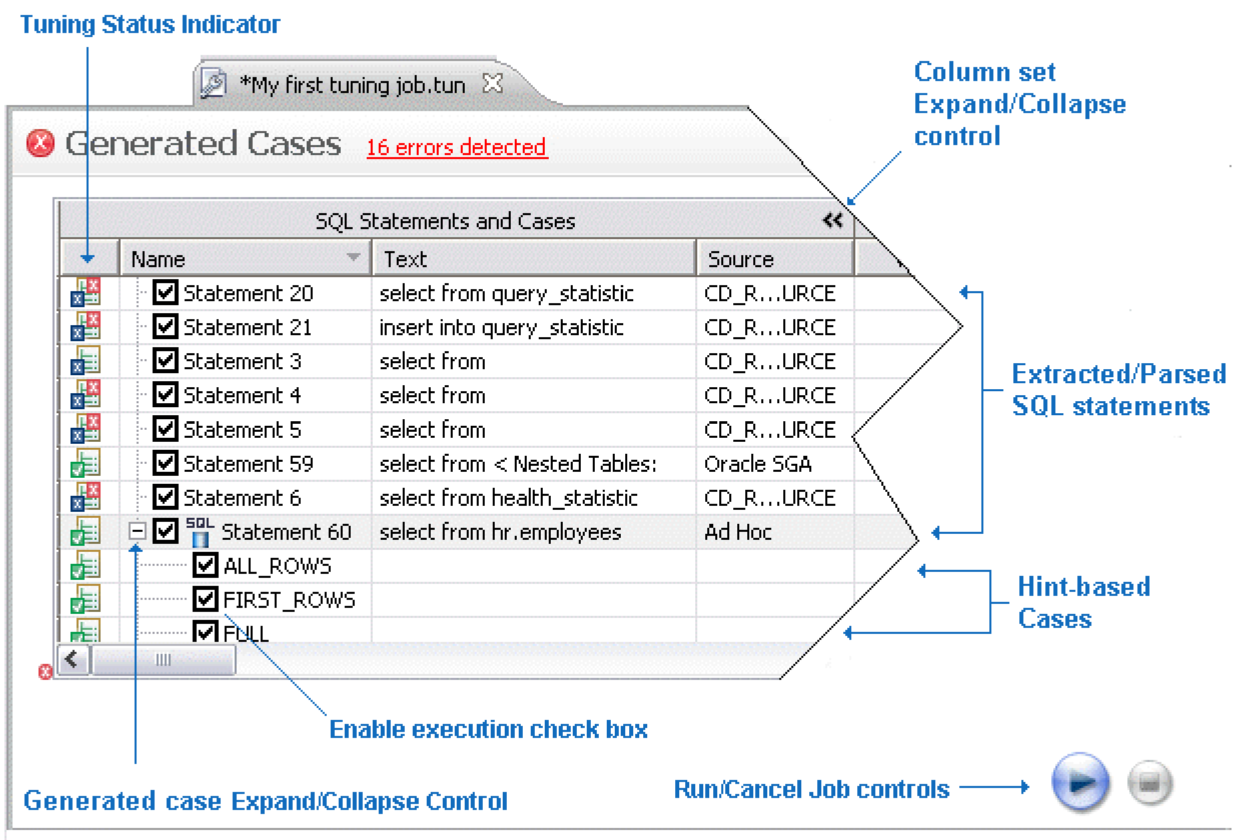
Each record can have a set of children, generated cases corresponding to:

- The supported Oracle SQL Optimizer hints applicable to the statement type and recognized content
- A transformation-based case.

For more information, see Working with Generated Cases.

The **Generated Cases** tab is where you perform your tuning analysis. Prior to executing statements in a tuning job, the statement record provides a short excerpt of the statement, an indication of the source, an explain plan cost, and results of an index analysis. Executing statements from the Generated Cases tab provides additional performance information such as elapsed time, physical and logical read information, and sort statistics, and other details. For more information on working with statements, see About Statement Records.

The **Generated Cases** tab also has a set of controls for working with the results.


Tuning Status Indicator
*My first tuning job.tun
Generated Cases
16 errors detected
Column set Expand/Collapse control
Extracted/Parsed SQL statements
Hint-based Cases
Enable execution check box
Generated case Expand/Collapse Control
Run/Cancel Job controls

SQL Statements and Cases

Name	Text	Source
Statement 20	select from query_statistic	CD_R...URCE
Statement 21	insert into query_statistic	CD_R...URCE
Statement 3	select from	CD_R...URCE
Statement 4	select from	CD_R...URCE
Statement 5	select from	CD_R...URCE
Statement 59	select from < Nested Tables:	Oracle SGA
Statement 6	select from health_statistic	CD_R...URCE
Statement 60	select from hr.employees	Ad Hoc
ALL_ROWS		
FIRST_ROWS		
FULL		

The controls are as follows:

- The Tuning Status Indicator indicates whether a statement or case is ready to execute or has successfully executed. For details on all states, see About the Tuning Status Indicator.
- The Generated case Expand/Collapse control lets you hide or display the hint-based cases and transformation-based case generated for a statement.
- The Enable Execution check boxes let you
  - Enable multiple statements or cases for simultaneous execution while the Run/Cancel Job controls let you start and stop simultaneous execution.
  - Generate cases if the preference controlling automatic case generation is enabled. For more information, see Specifying Generated Cases Preferences.

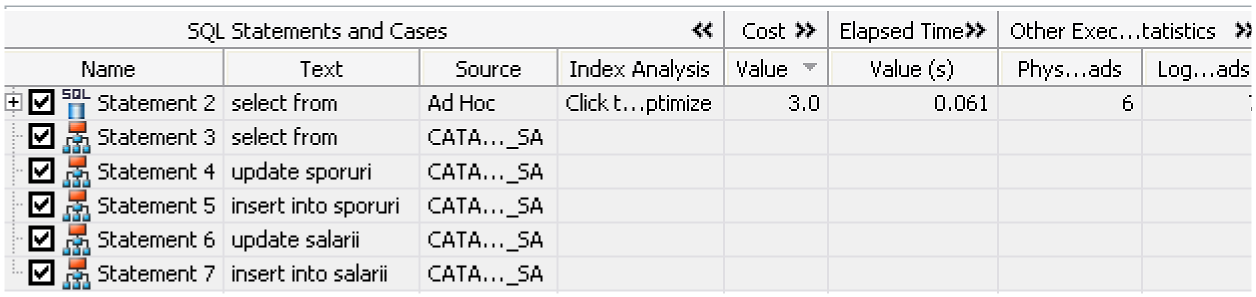
  For more information on execution options, see Executing Statements and Cases on the Generated Cases Tab.
- The Column set Expand/Collapse controls let you expand a column set to display more of the columns within the column set or to collapse the column set to show only a subset.

Other options available from the **Generated Cases** tab include:

- Opening the Source for a Statement or Case to Show it in Context
- Displaying the Full Text of a Statement or Case and Setting Bind Variables
- Opening an Explain Plan for a Statement or Case
- Working with Index Analysis Options

About Statement Records

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 **Generated Cases** tab.



SQL Statements and Cases				Cost	Elapsed Time	Other Exec...tatistics	
Name	Text	Source	Index Analysis	Value	Value (s)	Phys...ads	Log...ads
Statement 2	select from	Ad Hoc	Click t...ptimize	3.0	0.061	6	
Statement 3	select from	CATA..._SA					
Statement 4	update sporuri	CATA..._SA					
Statement 5	insert into sporuri	CATA..._SA					
Statement 6	update salarii	CATA..._SA					
Statement 7	insert into salarii	CATA..._SA					

The following table provides information on the fields of the statement or case record and describes available options:

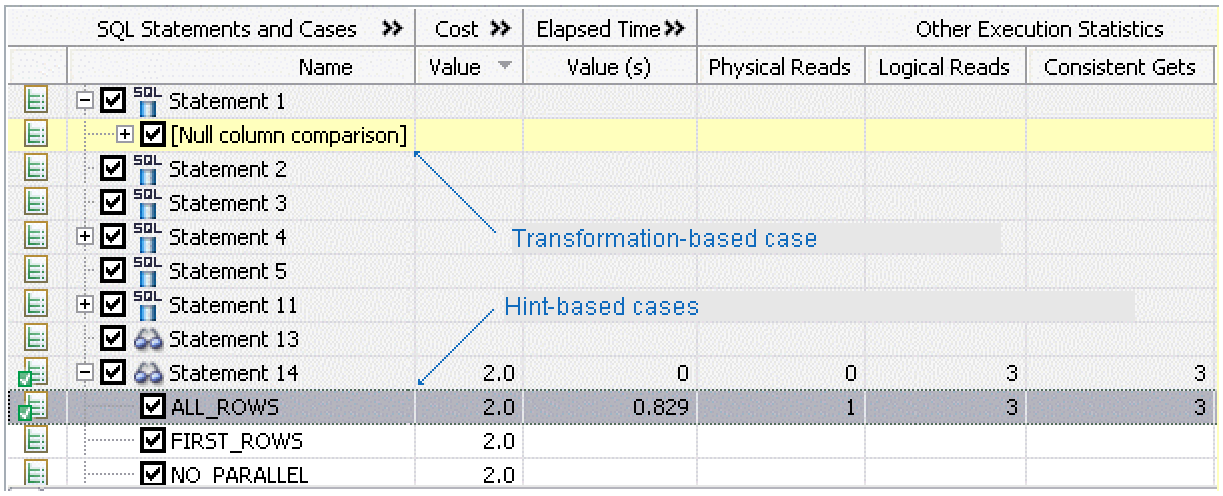
Column or column set	Description
SQL Statements and Cases	Identifiers for the generated statement or case: **Name** - Statements are assigned a numbered identifier based on the order in which they were added to a tuning job. **Text** - An excerpt of the statement or case based on the statement type (SELECT, INSERT, DELETE, and UPDATE). For details on how to view the entire statement or case, see Displaying the Full Text of a Statement or Case and Setting Bind Variables. **Source** - Corresponds to the source type from which you added the statement. Values can be **Ad Hoc**, **Oracle SGA**, a SQL file name, or the name of a SQL-containing object. For information on options available from this field, see Opening the Source for a Statement or Case to Show it in Context.
Cost	An explain plan-based cost estimate. This field is populated as soon as the statement is added to the **Generated Cases** tab. This column set can be expanded to display a graphical representation of the cost to facilitate comparisons among cases. For more information, see Understanding the Graphical Representation of Cost and Elapsed Time.
Index Analysis	The SQL Tuner feature automatically detects indexes that require optimization and offers you the option to automatically optimize the index. For more information, see Working with Index Analysis Options.
Elapsed time	The execution time during the most recent execution. This column set is not populated until you execute the statement or case. For more information, see Executing Statements and Cases on the Generated Cases Tab. This column set can be expanded to display a graphical representation of the elapsed time to facilitate comparisons among cases. For more information, see Understanding the Graphical Representation of Cost and Elapsed Time.
Other Execution Statistics	The default, collapsed view has **Physical Reads** and **Logical Reads** columns. Expanded, there are also **Consistent Gets**, **Block Gets**, **Rows Returned**, **CPU time(s)**, **Parse CPU Time(s)**, **Row Sorts**, **Memory Sorts**, **Disk Sorts,** and **Open Cursors** columns. For details on these statistics, refer to your DBMS documentation. This column set is not populated until you execute the statement or case. For more information, see Executing Statements and Cases on the Generated Cases Tab.

NOTE: For detailed information on hints-based cases of SQL statements, see Working with Generated Cases.

Working with Generated Cases

Cases generated from tuning candidates are alternative forms of the original statement, optimized or with common coding errors corrected. The SQL Tuner feature can generate:

- All SQL Optimizer hint-based variations that can be applied to a statement. For a short description off all hints supported by SQL Tuner, see Hints Reference.
- A transformation-based case if any of eight common quick fixes can be applied to a SQL statement. For details, see Understanding Code Quality Checks. A transformation case will in turn, have its own set of SQL Optimizer hint cases.



SQL Statements and Cases	Cost	Elapsed Time	Other Execution Statistics		
Name	Value	Value (s)	Physical Reads	Logical Reads	Consistent Gets
Statement 1					
[Null column comparison]					
Statement 2					
Statement 3					
Statement 4					
Statement 5					
Statement 11					
Statement 13					
Statement 14	2.0	0	0	3	3
ALL_ROWS	2.0	0.829	1	3	3
FIRST_ROWS	2.0				
NO_PARALLEL	2.0				

SQL Tuner generates all applicable cases:

- Automatically, when you add a tuning candidate to the tuning job, if automatic case generation is enabled. For details, see Specifying Generated Cases Preferences.
- When you explicitly generate cases. For details, see Explicitly generating cases.

Hint-based cases and the transformation-based case are a special case of the statement records added to the **Generated Cases** 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. For more information, see About Statement Records.

NOTE: For execution purposes, you can set bind variables in all generated cases for a statement by setting the variables in the original statement. For more information, see Displaying the Full Text of a Statement or Case and Setting Bind Variables.

Other case-specific options you can use during your analysis include:

- Comparing a case side-by-side with its parent or another case
- Filtering or removing cases that are not improvements on the parent statement

On completion of your analysis, there are two ways to apply the change suggested by a case to the original statement:

- Applying the change which modifies the original SQL
- Creating an outline based on a case for the statement record

Explicitly generating cases

Automatic generation of cases is controlled by an application preference. For details, see Specifying Generated Cases Preferences.

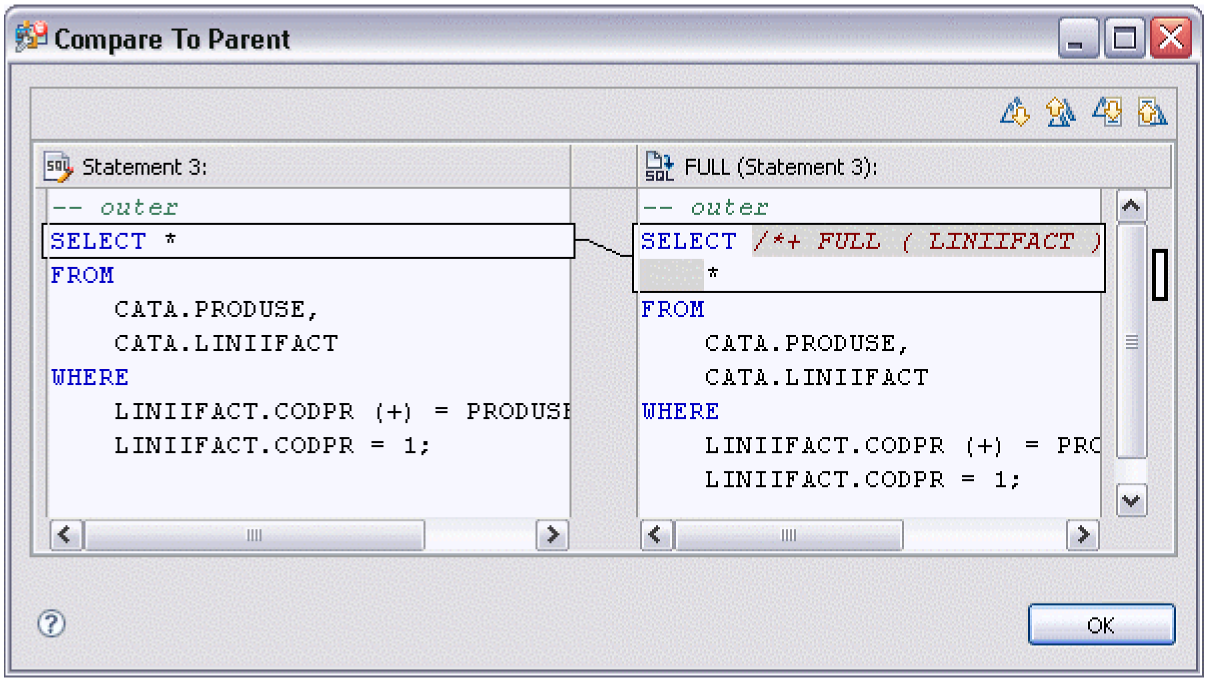
If automatic generation of cases is disabled or if you have deleted cases of a parent statement or a transformation case on the **Generated Cases** tab, you can explicitly instruct the SQL Tuner feature to generate cases for a parent statement.

To generate cases for a statement:

- Right-click in the **Name** field of a statement or transformation case and select **Generate Cases** from the context menu.

Comparing a case side-by-side with its parent or another case

To help you see the differences between cases or between an original statement and one of its cases, the SQL Tuner feature lets you open a window showing the full text of the statements.



```
-- outer
SELECT *
FROM
    CATA.PRODUSE,
    CATA.LINIIFACT
WHERE
    LINIIFACT.CODPR (+) = PRODUSE
    LINIIFACT.CODPR = 1;
```

```
-- outer
SELECT /*+ FULL ( LINIIFACT )
     *
FROM
    CATA.PRODUSE,
    CATA.LINIIFACT
WHERE
    LINIIFACT.CODPR (+) = PRO
    LINIIFACT.CODPR = 1;
```

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

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

To compare two cases

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

Filtering or removing cases that are not improvements on the parent statement

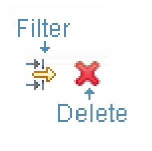
You can filter the view on the **Generated Cases** tab so that hints that are not improvements on the original statement are not displayed. Similarly, you can permanently remove cases from the tuning job. You can filter or remove:

- Non-optimizable statements

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

To filter or remove cases that are not improvements:

1. Click the Filter or Delete button.

   A **Filters** or **Delete** dialog opens.

2. Use the check boxes to select your filtering or removal criteria and then click **OK**.

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.

Applying the change which modifies the original SQL

For ad hoc tuning candidates and SQL statements extracted from SQL files or database objects, if you have the required permissions, you can apply a change suggested by a hint-based case or transformation-based case. For Oracle SGA tuning candidates you can modify the SQL but cannot change what has been executed.

To apply a change suggested by a case:

1. RIght-click in the **Name** field of a case and select **Apply Change** from the context menu.

   An **Apply Change** dialog opens previewing the SQL code to apply the change.

2. Select an **Action to take** option of **Execute** or **Open in new SQL editor** and then click **Finish**.

   NOTE: Alternatively, you can apply the change through creating an outline. For details, see Creating an outline based on a case for the statement record.

Creating an outline based on a case for the statement record

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. An outline instructs oracle on the execution path that should be taken for a particular statement.

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

1. 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**.

About the Tuning Status Indicator

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. For more information, see Executing Statements and Cases on the Generated Cases Tab.
	The case has been successfully executed.
	Transformations can be applied to this case.
	Execution for this case failed.
	The case contains invalid bind variables (types or values). For more information, see Displaying the Full Text of a Statement or Case and Setting Bind Variables.
	Execution for this case failed and the case contains invalid bind variables.

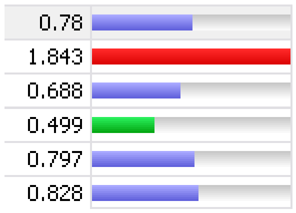
Hovering the mouse over the Tuning Status Indicator displays a tip that notes the nature of a warning or error.

NOTE: If a warning indicates that one or more tables do not have statistics, you can right-click the statement and select **Analyze Tables** to gather statistics. For more information, see Setting Tuning job Editor Preferences.

NOTE: 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.

Understanding the Graphical Representation of Cost and Elapsed Time

On the **Generated Cases** tab, the explain plan-based **Cost** field can be expended to display a graphical representation of the values for statements or cases. Similarly, after executing a statement or case, the **Elapsed Time** field can be 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 Set SQL Tuner Feature Preferences.

Executing Statements and Cases on the Generated Cases Tab

On the **Generated Cases** tab of a tuning job, the **Cost**, **Elapsed Time** and **Other 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 **Other Execution Statistics** columns are not populated until you execute that statement or case.

If the Tuning Status Indicator indicates that a statement or case is ready to execute, you can execute one or more statements on the **Generated Cases** tab. Alternatively, the Tuning Status Indicator may show that you have to correct the SQL or set bind variables before you can execute. For information on interpreting Tuning Status Indicator states, see About the Tuning Status Indicator.

To execute SQL statements or cases, use one of the following methods:

- To execute a single statement or case, right-click on the statement or case and select **Execute** from the context menu. If you execute an individual case variant while the base statement has no current execution statistics, the base statement will be executed as well.
- To execute all statements or cases with the associated Enable execution check box selected, click the Run job button.

  ▶

  For more information on the Enable execution check box, see Working with Tuning Results on the Generated Cases Tab.

Opening the Source for a Statement or Case to Show it in Context

On the **Generated Cases** tab, you can use the **Source** field of a statement record to open that statement, as follows:

- The **Ad Hoc SQL** tab of the **Overview** view for SQL statements you typed or pasted into that tab
- For SQL files you added using the **Overview** view's **SQL Files** tab, the file opens in the SQL editor
- For SQL-containing objects you added using the **Overview** view's **Database Objects** tab, the database object is extracted from the database and displayed in the SQL editor.
- The **Active SQL in SGA** tab of the **Overview** page for SQL statements you specified using an SGA scan

To open the source for a statement or case to show it in context:

1. Click in the **Source** field of a statement or case.

   The source control is activated.

2. Click the source control a second time.

The SQL statement you selected in opening the resource is highlighted.

Displaying the Full Text of a Statement or Case and Setting Bind Variables

The Tuning job's **Generated Cases** tab let you open a statement in a 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. For details on the Tuning Status Indicator, see About the Tuning Status Indicator.

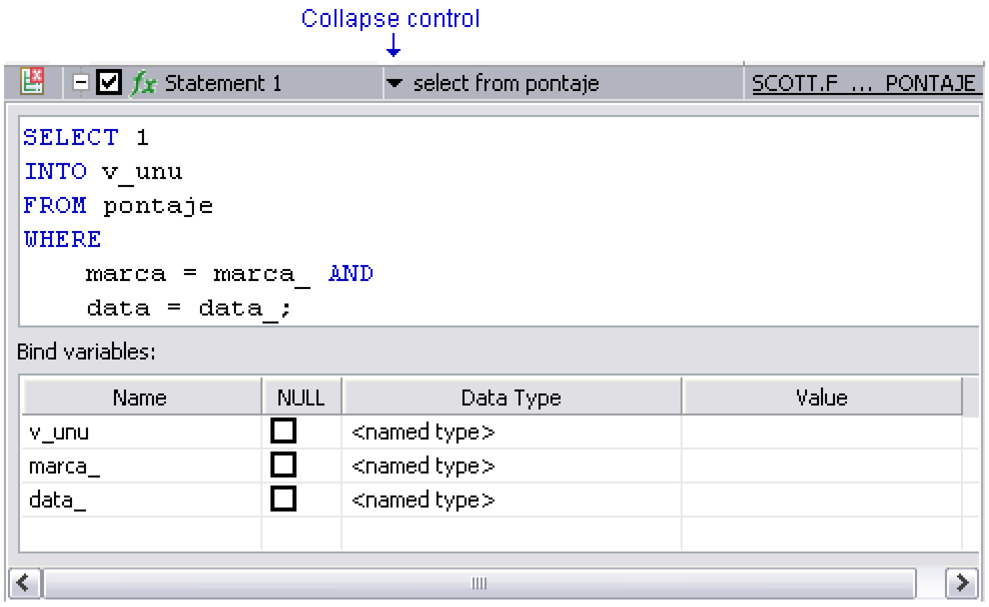
To view or set bind variables in a statement or case:

1 Click in the **Text** field of a statement or case.

A SQL Viewer opens on the statement or case. A set of controls for working with the statement or case bind variables appears at the bottom of the window.

2 Use the **Data Type** and **Value** (or **NULL**) controls to specify the type and value for each bind variable.

3 Close the window by clicking the collapse control in the **Text** field of the statement record, above the SQL Viewer.



```
SELECT 1
INTO v_unu
FROM pontaje
WHERE
    marca = marca_ AND
    data = data_;
```

Name	NULL	Data Type	Value
v_unu	☐	<named type>	
marca_	☐	<named type>	
data_	☐	<named type>	

After setting bind variables, you can execute a case. For details, see Executing Statements and Cases on the Generated Cases Tab.

NOTE: Setting the bind variables in a parent statement sets the bind variables in all generated cases for that statement. For more information, see Working with Generated Cases.

Opening an Explain Plan for a Statement or Case

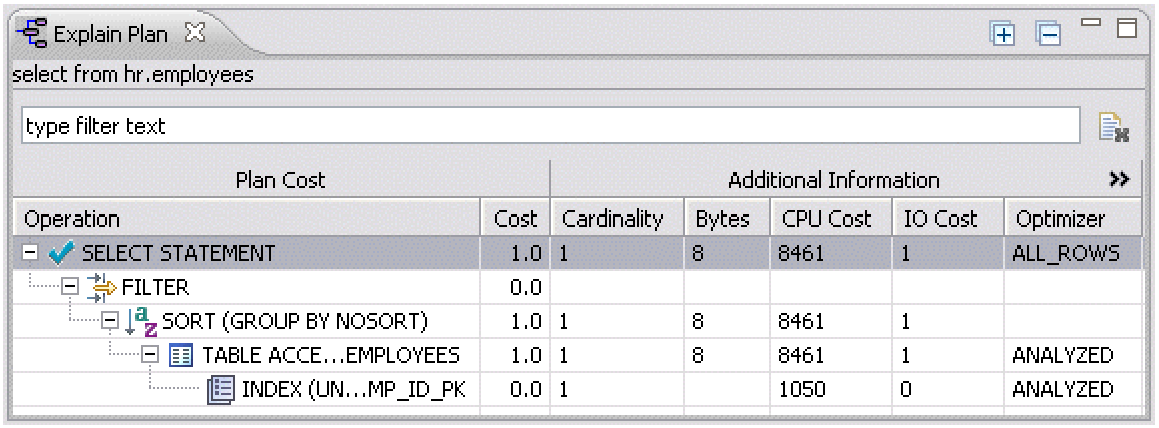
Any valid SQL statement added to the **Generated Cases** tab shows a calculated explain plan cost in the **Cost** field of the statement or case record. You can open an explain plan on these statements to view the sequence of operations used to execute the statement and the costs and other explain plan details for each operation.

To initially open an explain plan on a valid SQL statement on the Generated Cases 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 **Generated Cases** tab.



select from hr.employees

Plan Cost		Additional Information				
Operation	Cost	Cardinality	Bytes	CPU Cost	IO Cost	Optimizer
SELECT STATEMENT	1.0	1	8	8461	1	ALL_ROWS
FILTER	0.0					
SORT (GROUP BY NOSORT)	1.0	1	8	8461	1	
TABLE ACCE...EMPLOYEES	1.0	1	8	8461	1	ANALYZED
INDEX (UN...MP_ID_PK	0.0	1		1050	0	ANALYZED

Explain plan operations are shown in a typical tree structure showing parent-child relationships. The following table describes the column groups shown for each operation on the **Explain Plan** tab:

Column (group)	Description
Plan Cost	Includes the **Name** of the operation and the calculated explain plan cost.
Additional Information	The default, collapsed view shows the **Cardinality**, **Bytes**, **CPU Cost**, **IO Cost**, and **Optimizer** values. Expanded, the view also displays **Access Predicates**, **Filter Predicates**, **QB Lock Name**, **Distribution**, **Object Alias**, **Object Instance**, **Object Node**, **Partition ID**, **Partition Star**t, **Partition Stop**, **Position**, **Projection**, **Remarks**, **Search Columns**, **Temp Space**, **Time**, **Other**, and **Other Tag** values.

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

To change the Explain Plan tab display to another SQL statement:

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

Working with Index Analysis Options

As tuning candidates are added to a tuning job, the SQL Tuner feature automatically performs index analysis. If any columns referenced in the WHERE clause are not the first column of an index, SQL Tuner will recommend that you create an index on that column. This is indicated by a **Click to Optimize** link in the **Index Analysis** field for a statement.

To accept the suggestion and have SQL Tuner automatically generate an index:

1 Click the **Click to Optimize** link in the **Index Analysis** field of a statement.

A **New Indexes** dialog opens.

2 Optionally, modify the **Index Name** and select an **Index Type**.

3 Click **Next**.

The dialog is updated with an **Indexes Preview** panel displaying the SQL to create the index.

4 Click **Finish** to create the recommended index.

Closing and Saving Tuning Jobs

You can save the SQL statements and current execution statistics associated with a tuning job.

To save a tuning session:

1 With a tuning session active, on the menubar choose **File > Save**.

   If this is the first time that you are saving this tuning session, you are prompted for a file name.

Once saved, you can subsequently open the session for further analysis, to add or remove SQL statements, or to collect new statistics. For more information, see Opening a Saved Tuning Job.

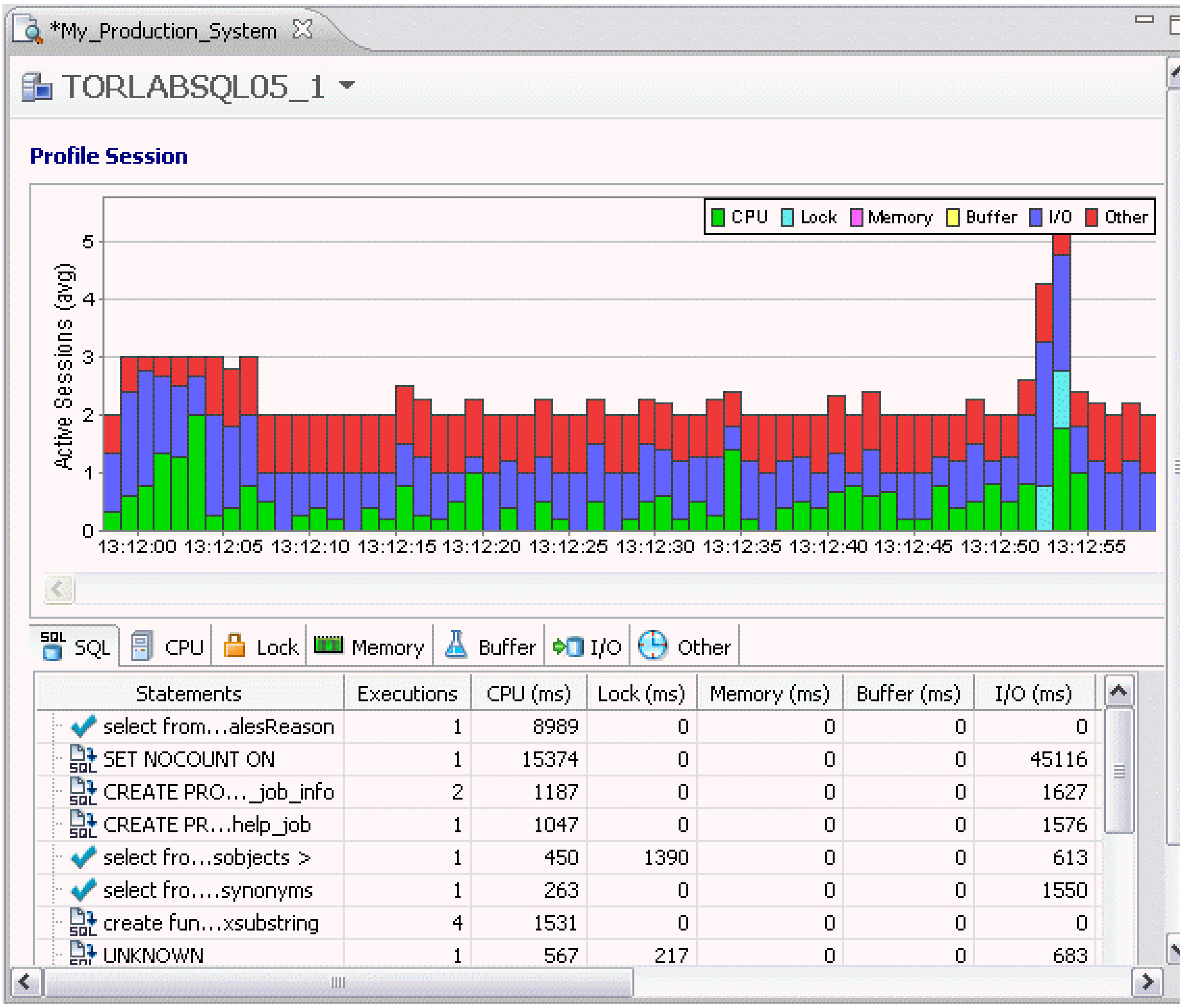
To close a tuning session:

1 With a tuning session active, on the menubar choose **File > Close**.

   If you have unsaved changes to SQL or execution statistics, you are prompted to save before closing.

Profiling Data Sources

The SQL Profiler feature lets you locate and diagnose problematic SQL and wait event-based bottlenecks. At a high level, SQL Profiler lets you take fixed-duration snapshots of query activity on a data source. At a lower level, SQL Profiler lets you investigate execution and wait event details for individual stored routines. SQL Profiler results, presented in a profiling editor, let you quickly identify problem areas and subsequently drill down to individual, problematic SQL statements.



Statements	Executions	CPU (ms)	Lock (ms)	Memory (ms)	Buffer (ms)	I/O (ms)
select from...alesReason	1	8989	0	0	0	0
SET NOCOUNT ON	1	15374	0	0	0	45116
CREATE PRO..._job_info	2	1187	0	0	0	1627
CREATE PR...help_job	1	1047	0	0	0	1576
select fro...sobjects >	1	450	1390	0	0	613
select fro....synonyms	1	263	0	0	0	1550
create fun...xsubstring	4	1531	0	0	0	0
UNKNOWN	1	567	217	0	0	683

The graphical portion of the profiling editor presents the distribution of sessions that were executing over the length of the profiling and those that were waiting in DBMS-specific events. It provides a first step in identifying problem areas. Tabs at the bottom of the profiling editor provide:

- Total number of executions, CPU time, and wait time for SQL statements waiting or executing over the profiling session

- Elapsed time details for specific wait event types and for executing sessions

Selecting one or more bars of the graph updates tab details to display only statistics for those bars, letting you find particular problem areas. You can then drill down to investigate SQL statements, statement components, and associated wait events active during that graph subset. In short, SQL Profiler provides the tools to not only indicate the general area of concern, but also to isolate the specific problem SQL statements.

As well, since results can be displayed while a profile is running, you can view results in real-time.

For details on working with SQL Profiler, see the following topics:

- Configuring DBMS properties and permissions for the SQL Profiler feature
- Running Profiling Sessions
- Saving and Opening SQL Profiling Archives

Configuring DBMS properties and permissions for the SQL Profiler feature

The following topics describe how you set up your DBMS for use with SQL Profiler:

- Setting up IBM DB/2 for Windows, Unix, and Linux for use with the SQL Profiler feature
- Setting up Microsoft SQL Server for use with the SQL Profiler feature
- Setting up Oracle for use with the SQL Profiler feature
- Setting up Sybase for use with the SQL Profiler feature

Setting up IBM DB/2 for Windows, Unix, and Linux for use with the SQL Profiler feature

In order to use SQL Profiler against IBM DB/2, the monitor flags (**BUFFERPOOL**, **LOCK**, **SORT**, **STATEMENT**, **TABLE**, **UOW**, **TIMESTAMP**) must be switched on.

Setting up Microsoft SQL Server for use with the SQL Profiler feature

In order to use SQL Profiler against SQL Server version 8 and 9, the current user needs to be a member of **sysadmin**.

In order to use SQL Profiler against 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 must be enabled for the current login.
- Be a member of **sysadmin** or the **SELECT** permission must be enabled for the current user.

Setting up Oracle for use with the SQL Profiler feature

In order to use SQL Profiler against Oracle version 10 or later, you either have to be logged in as the **sys** user with the **sysdba** role, or the **SELECT_CATALOG_ROLE** role must be granted for your current login.

In order to use SQL Profiler against earlier versions of Oracle, you must be the **sys** user with the **sysdba** role.

Setting up Sybase for use with the SQL Profiler feature

In order to use SQL Profiler against Sybase, you must meet the following requirements:

- You must have the **enable monitoring** system configuration property turned on
- You must have the **wait event timing** system configuration property turned on
- You must have the **max SQL text monitored** system configuration property turned on
- You must have the **SQL batch capture** system configuration property turned on

In addition:

- If the user doesn't have the **mon_role** enabled, the user will not be able to access Adaptive Server's monitoring tables.
- If the **monProcess** table is missing, the user will not be able to see currently connected sessions.
- If the **sysprocesses** table is missing, the user will not be able to see information about Adaptive Server processes.
- If the **monWaitEventInfo** table is missing, the user will not be able to see information about wait events.
- If the **monProcessSQLText** table is missing, the user will not be able to see currently executing SQL statements.

Running Profiling Sessions

The SQL Profiler feature lets you create a set of launch configurations to store the basic properties for each analysis you want to run on a regular basis. Whether running a profiling session against a data source or an individual stored routine, named launch configurations let you start a profiling session from a single menu command and can be used as an organizational aid. For details, see Creating Profile Launch Configurations.

With launch configurations set up, you can easily run SQL Profiler sessions on a day-to-day basis as required, to isolate and diagnose wait-based bottlenecks. For details, see Running a Profiling Session Using a Launch Configuration. Similarly, you can run ad hoc or one-off profiling sessions.

At the highest level, SQL Profiler results let you view the distribution of executing and waiting sessions over the length of the profiling session. Options let you drill down to view statistics for DBMS-specific wait class categories and on down to the level of specific wait events or to details for specific SQL statements. Related options lets you view statistics for subsets of the profile length and zoom in and out on the graph. For details, see Working with Session Results in the Profile Editor.

Creating Profile Launch Configurations

When you run a profiling session against a data source, you must specify the data source and the length of time that you want the session to run. Similarly, when you run a session against a stored routine, you must specify the function or procedure to execute and provide values for any input parameters.

NOTE: On all DBMS platform, support for stored routines includes functions and procedures. For Oracle, stored routine support also includes package functions and package procedures.

The SQL Profiler feature lets you store these basic profiling properties in a profile launch configuration. For each resource that you profile, you can create and save one or more named launch configurations.

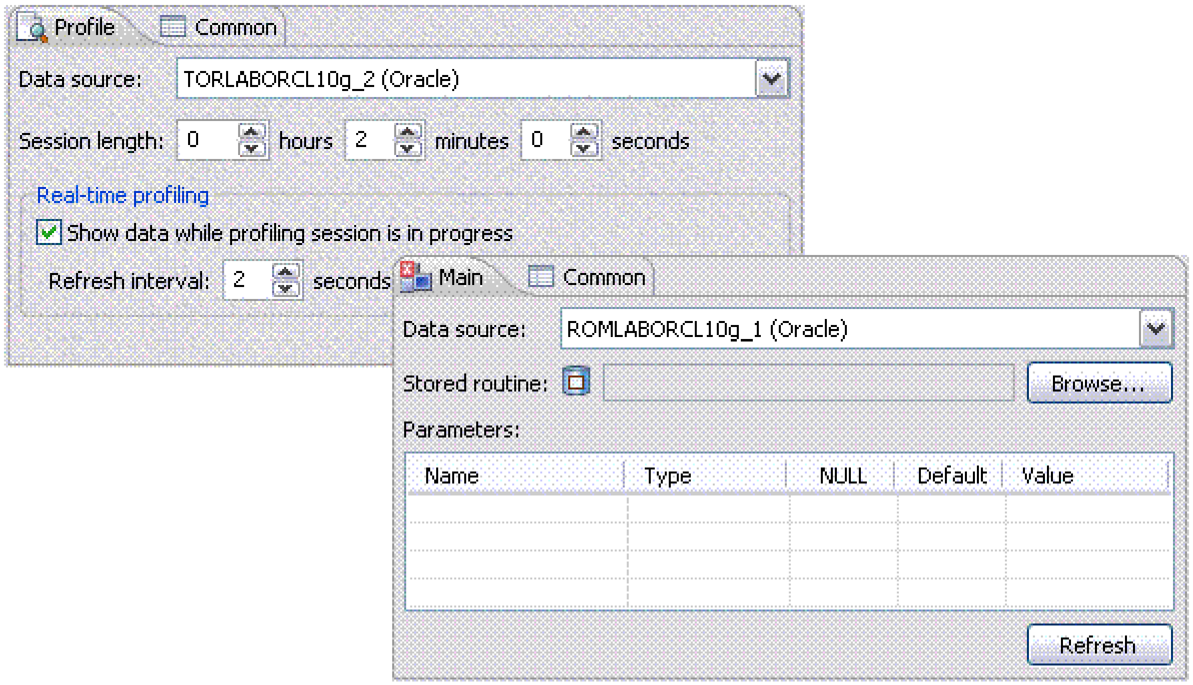
To create a SQL Profiler launch configuration:

1 Right-click in the Data Source Explorer, select **Profile As** from the context menu, and then select **Open Profile Dialog**.

A **Profile** dialog opens.

2 In the left pane of the dialog, select **Data Source** or **SQL Stored Routine**, and then on the toolbar above the left pane, click the New launch configuration button:

The right pane of the dialog is updated with the controls used to create a launch configuration of the type you chose.



Data source: TORLABORCL10g_2 (Oracle)

Session length: 0 hours 2 minutes 0 seconds

Show data while profiling session is in progress [✓]

Refresh interval: 2 seconds

Data source: ROMLABORCL10g_1 (Oracle)

Stored routine:

Parameters:

Name	Type	NULL	Default	Value

3 In the **Name** field, provide a name for this launch configuration. The name you choose could reflect the name of the resource and the profile length, for example.

4 Ensure that the selected value in the **Data source** dropdown reflects the correct data source.

5 Use one of the following methods:

- If creating a data source launch configuration, use the **Session Length** controls to specify a profile length in hours, minutes, and seconds. If you want to view results while the session is running, use the **Real-time profiling** controls to enable the feature and specify a refresh interval.
- If creating a stored routine launch configuration, use the **Browse** button to open a dialog that lets you locate and select a stored routine. If appropriate SQL Profiler will gather information on any required input parameters. You can then use the controls in the **Parameters** area to provide the values used in executions of the stored routine when executed using this launch configuration.

6 Click **Apply** to save the launch configuration.

Once you create a launch configuration, you can quickly start a profile session, referencing the named launch configuration. For details, see Running a Profiling Session Using a Launch Configuration.

NOTE: You can use the **Profile** button to immediately run a profile. In addition to letting you run a profile using the currently selected launch configuration, you can also provide or modify the current properties and run an ad hoc profiling session.

NOTE: The left pane of the **Profile** dialog lets you load existing launch configurations. The associated toolbar lets you create, copy, or delete launch configurations. You can also filter, expand, and collapse the launch configuration lists using the associated controls.

Running a Profiling Session Using a Launch Configuration

While the SQL Profiler feature lets you run ad hoc profiling sessions, the prescribed method is to create one or more launch configurations, each specifying either:

- A data source to be profiled and the length of the session
- A stored outline to execute and any required parameter values

You can then run profiling sessions using a launch configuration.

NOTE: For details on creating launch configurations, see Creating Profile Launch Configurations.

To run a profiling session using a launch configuration:

1 Connect to a data source.

NOTE: Depending on the size and configuration details for the data source, this can take several minutes.

2 In the Data Source Explorer, right click on the resource to be profiled, select **Profile As** from the context menu, and then select **1 Data Source** or **1 SQL Stored Routine**.

NOTE: If multiple launch configurations are defined for the resource, a **Profiling Session Launch Configurations** dialog opens. Select the launch configuration corresponding to the type of profile you want to run and click **OK**.

The profiling session runs for the length of time specified in the launch configuration. For information on interpreting profile results and the tasks you can perform when analyzing results, see Working with Session Results in the Profile Editor.

Working with Session Results in the Profile Editor

Profiling session results are displayed in a Profile editor. It opens whenever:

- You start a profiling session and you have enabled the real-time profiling feature on the profile dialog. For details, see Creating Profile Launch Configurations.
- A running session profile completes and the real-time profiling feature is disabled. For details, see Running a Profiling Session Using a Launch Configuration.
- You open profiling session archive file. For details, see Saving and Opening SQL Profiling Archives.

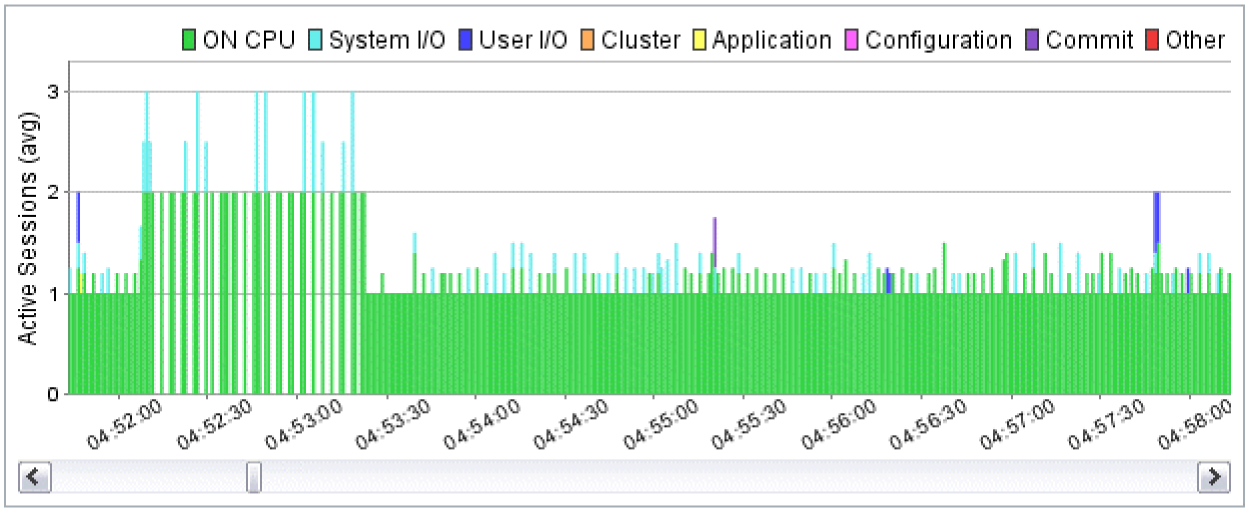
Results are displayed in graphical and tabular formats. The bar chart in the upper portion of the Profile editor presents the distribution of waiting and executing sessions over the length of the profiling session. Detailed tabular results are provided in the tabbed view in the lower portion of the Profile editor.

The following topics provide the information you need to work with results in the Profile editor:

- Understanding the Profile Editor Graph
- Understanding Profile Editor Tabs
- Tasks and Options in the Profile Editor

Understanding the Profile Editor Graph

The Profile editor's bar chart shows the distribution of waiting and executing sessions over the length of the profiling session.



Time is displayed on the X axis, with the length divided into one-second bars. The SQL Profiler feature lets you zoom in and out on the graph. For details, see Zooming in and out on the Graph.

The Y axis shows the average number of sessions waiting or executing during each one-second bar. Each supported platform has a specific set of wait event categories:

- IBM DB/2 for Windows, Unix, and Linux - **Fetch**, **Cursor**, **Execution**, **Operation**, **Transaction**, **Connectivity**, and **Other**
- Oracle - **On CPU**, **System I/O**, **User I/O**, **Cluster**, **Application**, **Configuration**, **Commit**, and **Other**
- SQL Server - **CPU**, **Lock**, **Memory**, **Buffer**, **I/O**, and **Other**
- Sybase - **CPU**, **Lock**, **Memory**, **I/O**, **Network**, and **Other**

A chart legend, showing the color-coding scheme for executing and waiting session categories is displayed in the upper, right-hand part of the graph.

The graph is intended as a high-level entry point to profile session results. You can subsequently use the tabbed view to find more detailed information on waiting/executing sessions across the length of the profiling session. Alternatively, you can select one or more bars of the graph to populate the tabbed view with detailed information on just that subset of the graph. For details, see Displaying Detailed Statistics for a Subset of the Profile.

Understanding Profile Editor Tabs

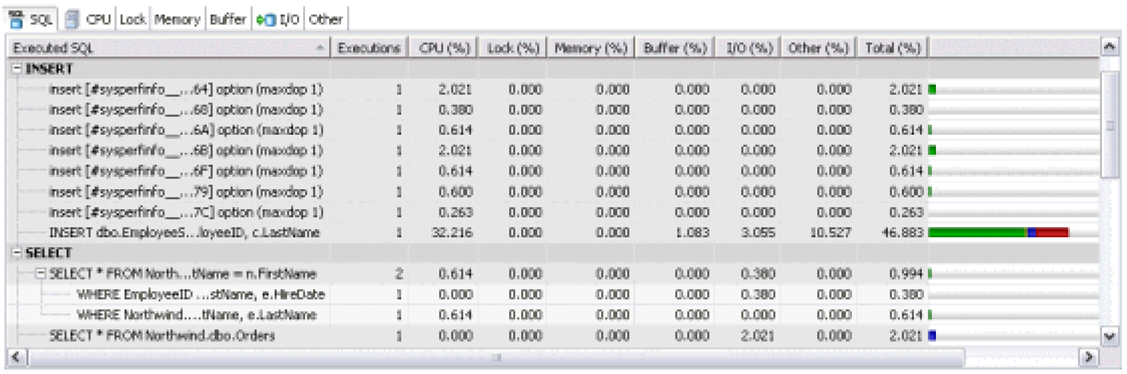
The Profile editor's tabbed view lets you view detailed statistics on individual SQL statements and sessions that were waiting or executing over the length of the profiling session. Profile editor tabs provide:

- Execution and wait time statistics on individual SQL statements that were active. For details, see SQL Tab.

- Details on sessions waiting or executing on events over the entire length or a subset of the profiling session. For details, see Event Category Tabs.

SQL Tab

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



Executed SQL	Executions	CPU (%)	Lock (%)	Memory (%)	Buffer (%)	I/O (%)	Other (%)	Total (%)
INSERT								
insert [#sysperfinfo__...64] option (maxdop 1)	1	2.021	0.000	0.000	0.000	0.000	0.000	2.021
insert [#sysperfinfo__...68] option (maxdop 1)	1	0.380	0.000	0.000	0.000	0.000	0.000	0.380
insert [#sysperfinfo__...6A] option (maxdop 1)	1	0.614	0.000	0.000	0.000	0.000	0.000	0.614
insert [#sysperfinfo__...6B] option (maxdop 1)	1	2.021	0.000	0.000	0.000	0.000	0.000	2.021
insert [#sysperfinfo__...6F] option (maxdop 1)	1	0.614	0.000	0.000	0.000	0.000	0.000	0.614
insert [#sysperfinfo__...79] option (maxdop 1)	1	0.600	0.000	0.000	0.000	0.000	0.000	0.600
insert [#sysperfinfo__...7C] option (maxdop 1)	1	0.263	0.000	0.000	0.000	0.000	0.000	0.263
INSERT dbo.EmployeeS...loyeeID, c.LastName	1	32.216	0.000	0.000	1.083	3.055	10.527	46.883
SELECT								
SELECT * FROM North...tName = n.FirstName	2	0.614	0.000	0.000	0.000	0.380	0.000	0.994
WHERE EmployeeID ...stName, e.HireDate	1	0.000	0.000	0.000	0.000	0.380	0.000	0.380
WHERE Northwind....tName, e.LastName	1	0.614	0.000	0.000	0.000	0.000	0.000	0.614
SELECT * FROM Northwind.dbo.Orders	1	0.000	0.000	0.000	0.000	2.021	0.000	2.021

Grouping by DML statement type is the default organization on the **SQL** tab. In addition to the **INSERT**, **SELECT**, **DELETE**, and **UPDATE** statement groupings, the SQL tab can display two other groupings. The **OTHER** grouping includes all recognized SQL statements other than INSERT, SELECT, UPDATE, and DELETE statements. The **UNKNOWN** grouping represents statements that could not be captured.

Within each statement type, statements are displayed in a tree structure with the following statement components:

- **Subject** - the DML statement type (and FROM clause where appropriate)
- **Predicate** - the WHERE clause
- **Remainder** - any statement component following the WHERE clause

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

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

Statistics are provided at each level of the tree structure. This lets 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:

Column	Description
Executions	The number of active executions for this statement or statement component over the length of the profiling session or the selected graph bars.
Elapsed Time	The total time that the statement or statement component spent actively executing or waiting to execute.

Column	Description
Activity	A visual representation of the distribution of execution and waiting time for the statement or statement component.
Event category columns (only available when the **Wait Analysis** column group is expanded)	For each event category, the time that the statement or statement component spent executing or waiting in an event type within that category.

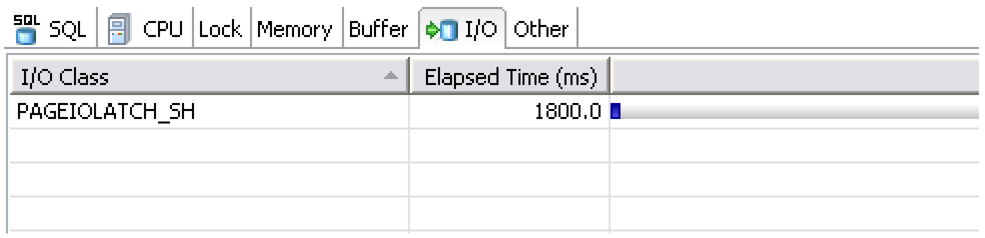
For information on how to open a new profiling editor with details pertaining only to a **SQL** tab statement entry, see Drilling down into entries on the SQL tab.

Event Category Tabs

The event category tabs displayed differ by DBMS platform as follows:

- IBM DB\2 for Windows, Unix, and Linux - **Fetch**, **Cursor**, **Execution**, **Operation**, **Transaction**, **Connectivity**, and **Other**
- Oracle - **On CPU**, **System I/O**, **User I/O**, **Cluster**, **Application**, **Configuration**, **Commit**, and **Other**
- SQL Server - **CPU**, **Lock**, **Memory**, **Buffer**, **I/O**, and **Other**
- Sybase - **CPU**, **Lock**, **Memory**, **I/O**, **Network**, and **Other**

An entry on the **CPU**, **On CPU**, or **Execution** tabs, indicates that one or more sessions, last waiting on the specified wait event, spent the **Elapsed Time** actively executing, over the length of the profiling session or within the currently selected graph bars. Entries on all other tabs indicate that sessions were waiting in events in that category over the length of the profiling session or within the currently selected graph bars. For example, against a Microsoft SQL Server data source:



I/O Class	Elapsed Time (ms)	
PAGEIOLATCH_SH	1800.0	

The graphical representation shows the elapsed time for each entry as a function of the entry with the longest elapsed time.

For information on how to open a new profiling editor with details pertaining only to an event category tab event entry, see Drilling down into events on event category tabs.

Tasks and Options in the Profile Editor

In addition to the default viewing options provided by the Profile editor's graph and tabbed view, the SQL Profiler feature also provides the following options:

- Zooming in and out on the Graph
- Displaying Detailed Statistics for a Subset of the Profile

- Filtering results in the profile editor
- Drilling down into entries on the SQL tab
- Drilling down into events on event category tabs
- Submitting SQL statements for tuning

Zooming in and out on the Graph

To zoom in or out on the profile session chart:

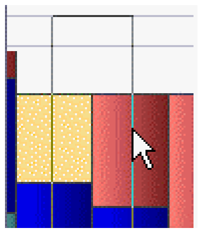
1 Click the Zoom In or Zoom Out control.

Displaying Detailed Statistics for a Subset of the Profile

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

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

- Click-drag across one or more bars.

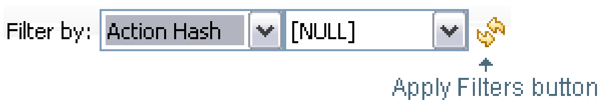


Filtering results in the profile editor

The profiling editor lets you display a filtered subset of the original profiling results set. The criteria you can filter on varies by DBMS platform:

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

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



To filter profile editor results:

1 Use the **Filter By** dropdown to select a filter type.

2 Use the second dropdown to select a specific value.

3 Click the Apply Filters button.

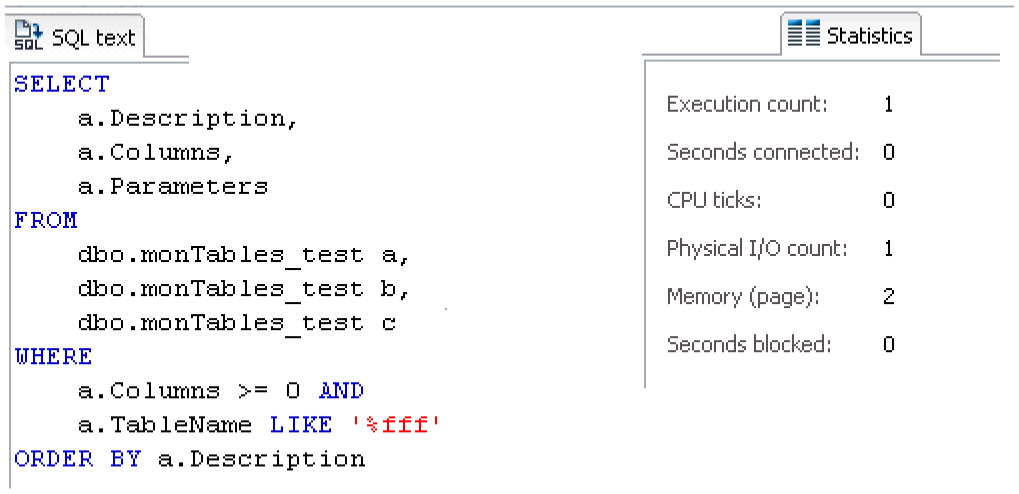
The profiling editor is updated to show only results associated with your choice.

TIP: Select **-None-** from the **Filter by** dropdown and click the Apply Filters button to restore the unfiltered results.

Drilling down into entries on the SQL tab

Drilling down on a statement entry on a **SQL** tab opens a new profiling editor page. The graph portion and details on the event category tabs on the new editor pertain only to the selected statement. In addition, two new tabs become available:

- A **SQL Text** tab showing the full text of the SQL statement
- A **Statistics** tab providing execution details



```
SELECT
    a.Description,
    a.Columns,
    a.Parameters
FROM
    dbo.monTables_test a,
    dbo.monTables_test b,
    dbo.monTables_test c
WHERE
    a.Columns >= 0 AND
    a.TableName LIKE '%fff'
ORDER BY a.Description
```

```
Execution count:      1
Seconds connected:    0
CPU ticks:            0
Physical I/O count:   1
Memory (page):        2
Seconds blocked:      0
```

To drill down on a SQL tab statement entry:

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

NOTE: If you click on a statement entry that has children nodes, an intermediate profiling editor page opens and you will have to repeat these steps to drill down to one of the child statements. For information on the tree structure representation of statements, see SQL Tab.

2 Click a second time to open the statement or event in the tab mentioned above.

The new profiling editor page opens, as reflected by the breadcrumb trail at the top left of the editor. Since you drilled down into a single statement, no **SQL** tab appears in the new profiling editor page. You cannot drill down further on event category tabs in the new editor page.

Drilling down into events on event category tabs

Drilling down on an event type entry on an event category tab opens a new profiling editor page. The graph portion and details on the **SQL** 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.

To drill down on an event entry on one of the event category tabs:

1. On an event category tab, click on an event type to activate the drilldown control.
2. Click a second time to open the open a new profiling editor page for that event type.

A new profiling editor page opens, as reflected by the breadcrumb trail at the top left of the editor. The graph and tabbed areas of the editor are updated as follows:

- The **SQL** tab has entries for each statement that waited on the selected event.
- The updated event category tabs contain entries pertaining to the statements on the new **SQL** tab.

NOTE: You can subsequently drill down on the **SQL** tab and event category tabs on the new profiling editor page. For more information, see Drilling down into entries on the SQL tab.

Submitting SQL statements for tuning

The SQL Profiler feature lets you submit one or more SQL tab statements for tuning by the SQL Tuner feature. This lets you take advantage of SQL Tuner'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.

  A tuning editor opens on the selected statement.

For more information, see Tuning SQL Statements.

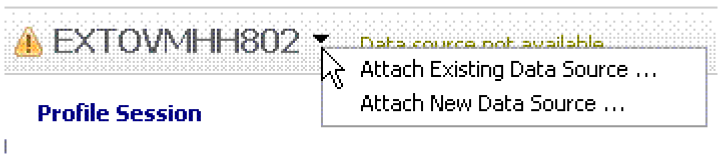
Saving and Opening SQL Profiling Archives

A profiling session can be saved in the current workspace in an archive file with a **.oar** suffix and 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

This lets you open the archive at a later time for subsequent analysis. Use standard DB Optimizer file techniques to save, open, or close SQL Profiling archives.

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


EXTOVMHH802
Data source not available
Attach Existing Data Source ...
Attach New Data Source ...
Profile Session

Working with Data Sources

The **Data Source Explorer** provides a browseable tree view of all DB Optimizer-registered data sources and associated database objects. Data Source Explorer is automatically populated with data sources that have been pre-registered in *Rapid SQL®*, *DBArtisan®*, or previous versions of DB Optimizer. If DB Optimizer cannot detect a data source, you can register it manually.

Register Data Sources

When DB Optimizer is started, it automatically imports the data source catalog created by any previously installed Embarcadero products or other instances of DB Optimizer.

The first time DB Optimizer is started, it also scans the local machine for the client software of supported third-party DBMS. These data sources are automatically added to the data source catalog. To manually initiate a scan later, click the **Auto-Discover Data Sources** icon at the top of Data Source Explorer.

To register a data source manually, right-click **Managed Data Sources** in the Data Source Explorer tree, select **New > Data Source**, and enter the connectivity parameters as prompted. For additional information on these parameters, see DBMS Connection Parameters by Platform.

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

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

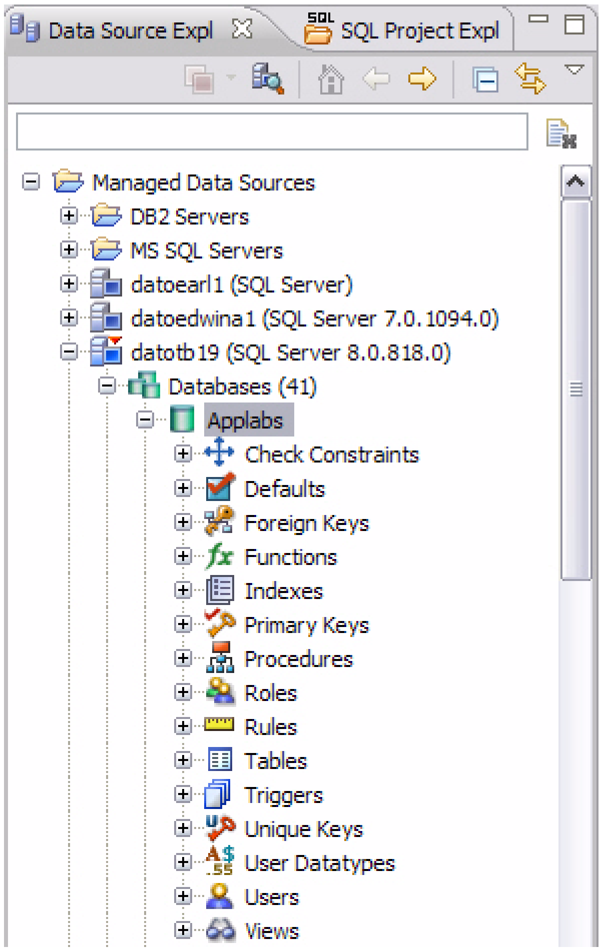
Browse a Data Source

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

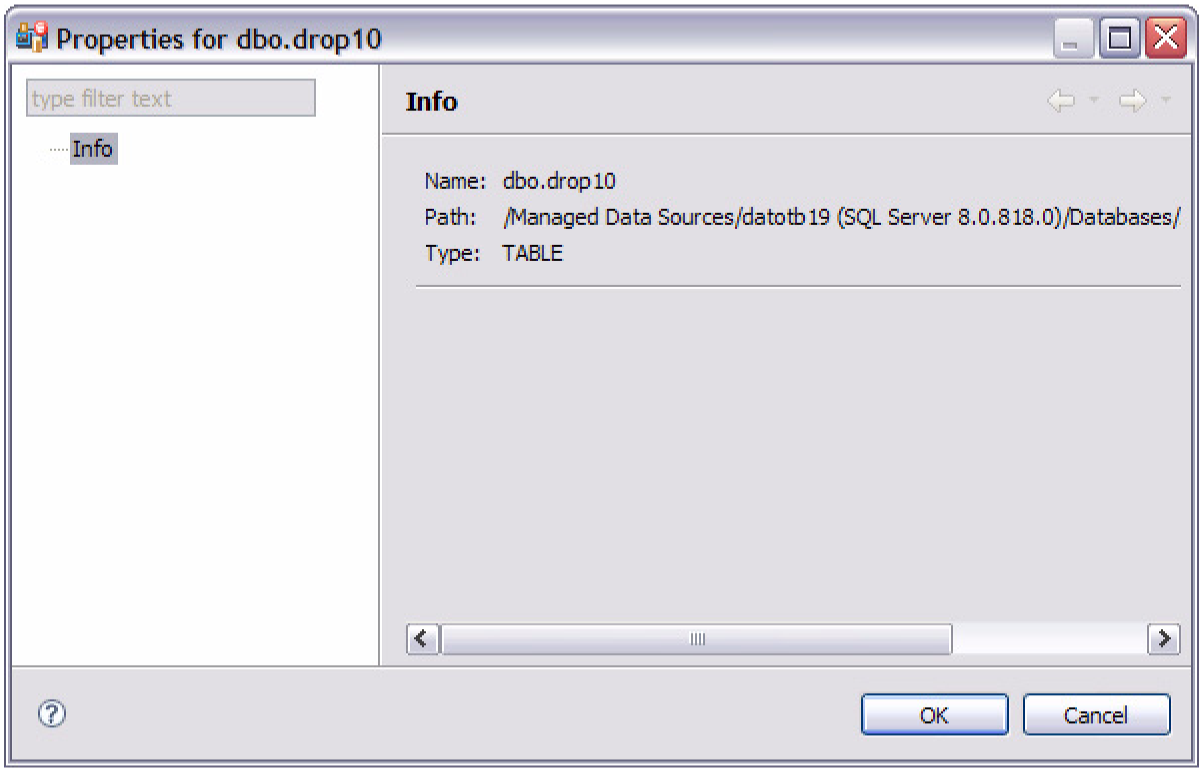
In most cases, whenever you browse a data source, DB Optimizer requires login information in order to connect with the data source. Enter a valid user name and password in the fields provided. The **Auto Connect** option retains your login credentials for future connections to the same data source.

View Database Object Properties

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



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



Name: dbo.drop10
Path: /Managed Data Sources/datotb19 (SQL Server 8.0.818.0)/Databases/
Type: TABLE

To view Data Source Explorer object properties:

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

The dialog displays the following three properties:

- **Name:** The name of the node as it appears in the Database Explorer. This value cannot be modified.
- **Path:** The folder path and file name where the object is stored on the system. This value cannot be modified.
- **Type:** The type of data source object that the node represents. For example, the **Type** value for a table node would be **TABLE**. This value cannot be modified.

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

The Configuration node is composed of:

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

These nodes are identical to the parameters used to initially define the data source during the data source registration process. For more information on these values and how to modify them, see Register Data Sources.

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 cache when returning results. By default, caching is set to configure only parts of a database. To configure the object cache to expand object searches, see Set Cache Configuration 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 Cached Data** to indicate that the search function should read the database cache. This increases the performance of the search function and will typically result in faster returns on any hits the search might make.
   - Select **Apply SQL Filters** to apply any relevant database or vendor filters to the search.
   - Choose **Declarations**, **References**, or **All Occurrences** to specify what the search is restricted to in terms of database objects.
   - A declaration is an instance where an object is declared. For example, an object is declared in a CREATE table.
   - A reference is an instance where an object is used or referred to. For example, an object is referred to in a procedure or as a foreign key in a table.
   - Choose **All Occurrences** to return both declarations and references in the search results.
   - Use the check boxes beside the database object panel to select and deselect the specific database objects that you want to be included in the search process.
3. Click **Search**.

   The results of your search are generated in the **Search** view.

Filter Database Objects

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

There are two types of data source filters available:

- **Global filters** that affect all registered data sources in the DB Optimizer 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 Global Database Object Filters

Define Data Source-specific 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 Filter** node and deselect **Enable Data Source Specific Settings**. The other controls on the dialog become enabled.
3. Click **Add**. 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.

5. When you have finished defining the filter template, click **OK**. The template name is added to the **Properties** dialog. It can be enabled and disabled by selecting or deselecting the check box beside its name, respectively.

Define Global Database Object Filters

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

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

To define a global filter:

1. Select **Window > Preferences** from the Main Menu. The **Preferences** dialog appears.
2. Expand the **SQL Development** node and select the **SQL Filter** subnode. The **SQL Filter** pane appears.
3. Click **New**. The **Filter Template** dialog appears.

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

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** asks you to confirm removal of the object and provides a DDL preview of the deletion code.

Working with 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 a project are organized under the following categories:

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

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

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

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

Open an Existing Project

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

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

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

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

Search a Project

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

Add Files to a Project

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

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

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

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

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

To use the Import command:

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

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

Delete a Project

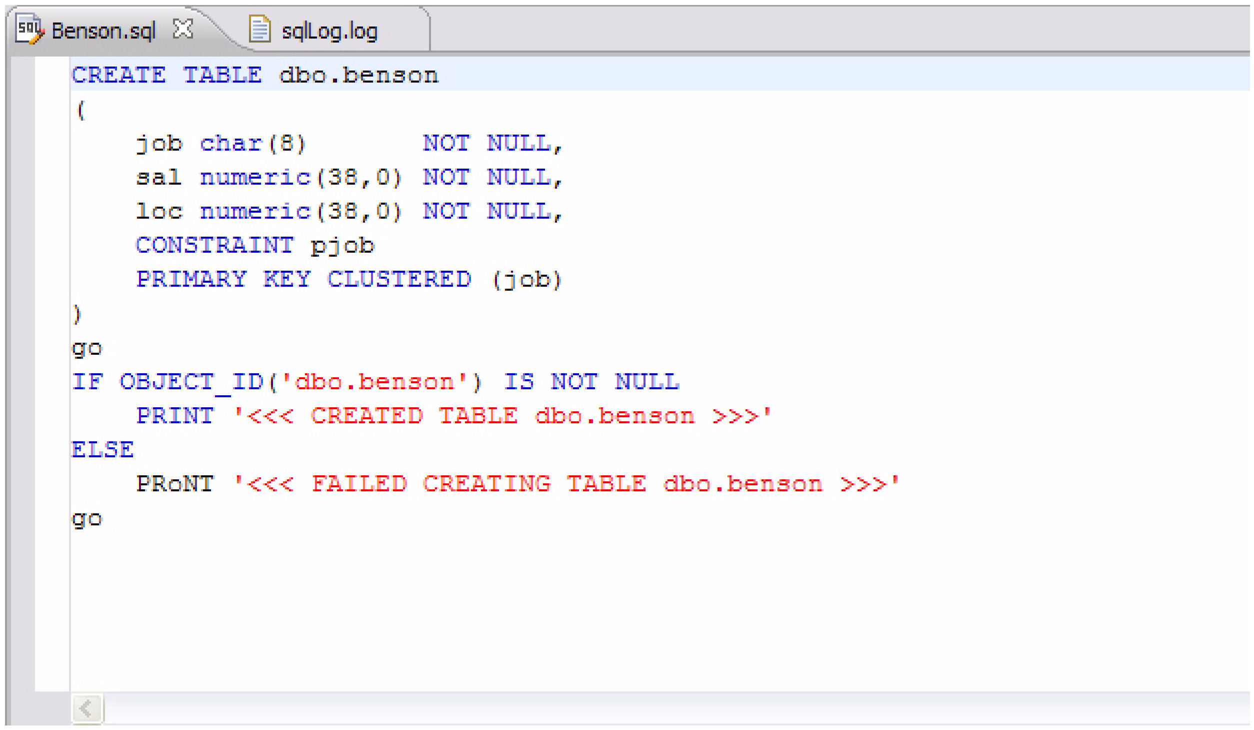
You can delete a project by right-clicking its folder in the SQL Project Explorer and selecting **Delete**.

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

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

Creating and Editing SQL Files (SQL Editor)

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



```
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
    PRoNT '<<< FAILED CREATING TABLE dbo.benson >>>'
go
```

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

If SQL Editor does not recognize a selected file format, DB Optimizer 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.

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

Code	Formatting
Strings	Red font
Non-Executable Command Line Commands	Aqua font

Single line and multiple line comments appear in different colors.

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

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

See also:

Working in SQL Editor

Create an SQL File

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

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

Open an Existing SQL File

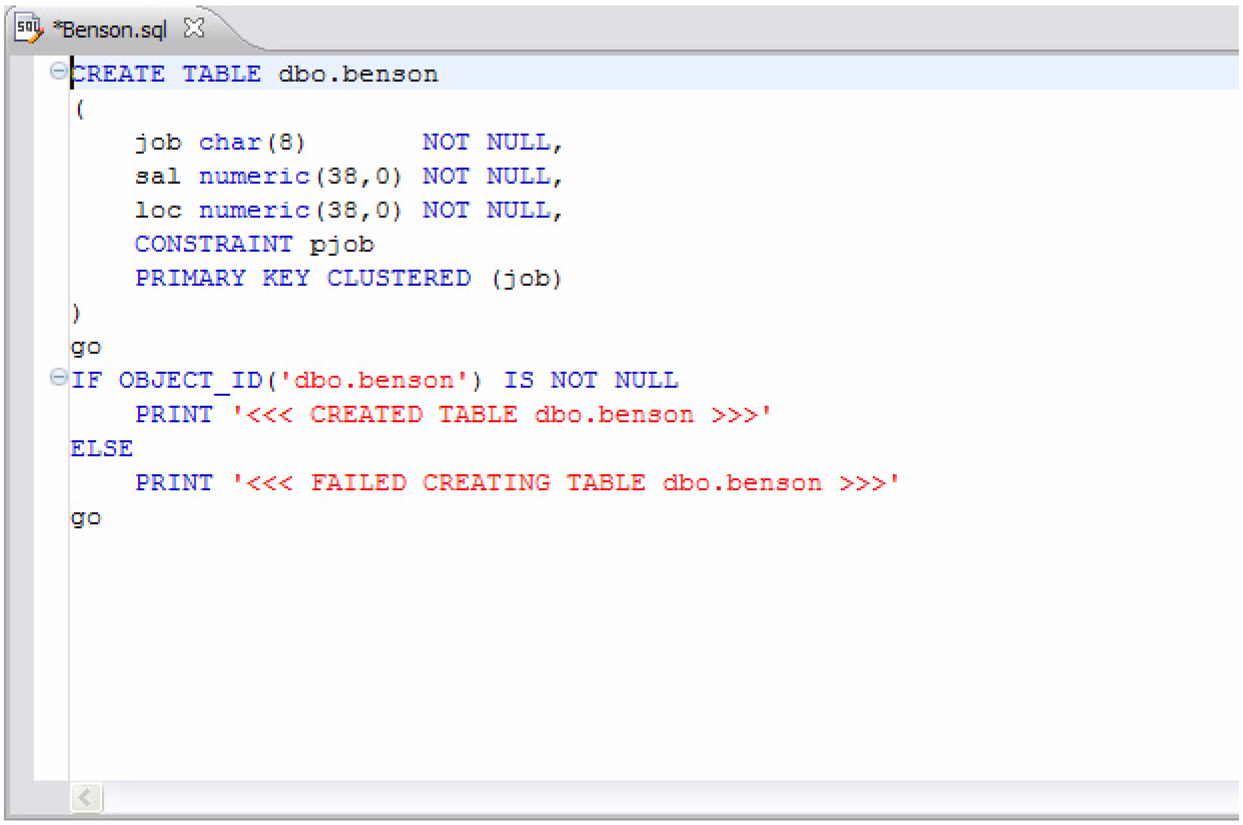
1. Open the SQL project containing the file, or that you want to contain the file.
2. If necessary, add the file to the project (see Add Files to a Project).
3. In the SQL Project Explorer, double-click the file to open it in SQL Editor.

Working in SQL Editor

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

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

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



```
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.

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

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

See also:

Understanding Automatic Error Detection

Understanding Code Assist

Understanding Hyperlinks

Understanding Code Formatting

Understanding Code Folding

Understanding Code Quality Checks

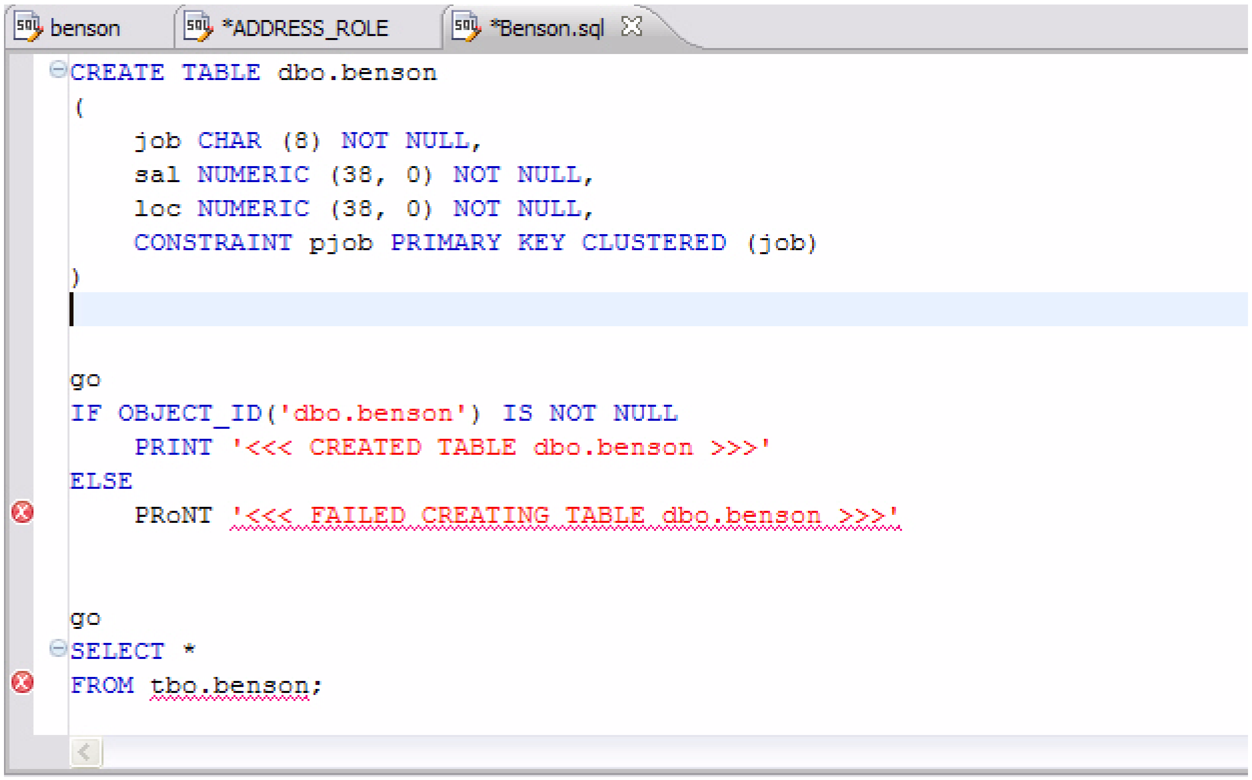
Understanding Automatic Error Detection

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:

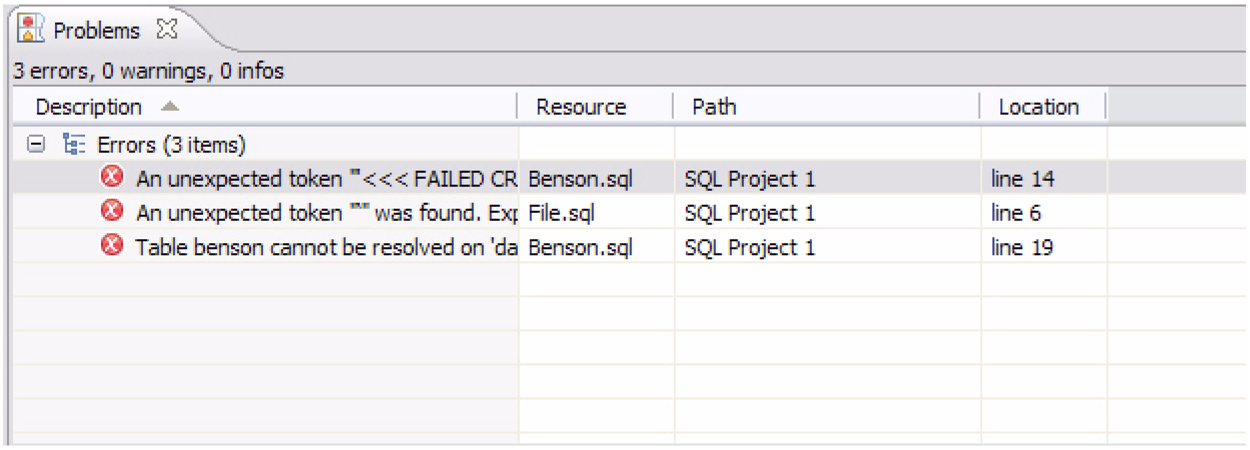
- **SELECT:** Specifies the field, constants, and expressions to display in the query results.
- **FROM:** Specifies one or more tables containing the data that the query retrieves from.
- **WHERE:** Specifies join and filter conditions that determine the rows that query returns. Join operations in a 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
    PRoNT '<<< FAILED CREATING TABLE dbo.benson >>>'


go
SELECT *
FROM tbo.benson;
```

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



3 errors, 0 warnings, 0 infos

Description	Resource	Path	Location
Errors (3 items)			
An unexpected token "<<< FAILED CR	Benson.sql	SQL Project 1	line 14
An unexpected token "" was found. Exp	File.sql	SQL Project 1	line 6
Table benson cannot be resolved on 'da	Benson.sql	SQL Project 1	line 19

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

Understanding Code Assist

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

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

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

Generally, insertion suggestions use the following format:

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

Content assist is available for SELECT, UPDATE, INSERT, and DELETE statements.

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 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)

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.

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

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

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

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

To activate object suggestion:

1 Click the line that you want SQL Editor to suggest an object for.

2 Press **CTRL + Spacebar** on your keyboard. SQL Editor ‘reads’ the line and presents a list of tables, views or columns as appropriate based on statement elements.

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

To modify object suggestion parameters, see Set SQL Editor 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, www.embarcadero.com) and database object links.

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

NOTE: Hyperlink functionality relies on certain objects being cached in the Object Cache. If the cache 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 cache.) To specify object cache types, see Set Cache Configuration 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. (See Set Code Formatter Preferences for more information on these formatting options.)

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.

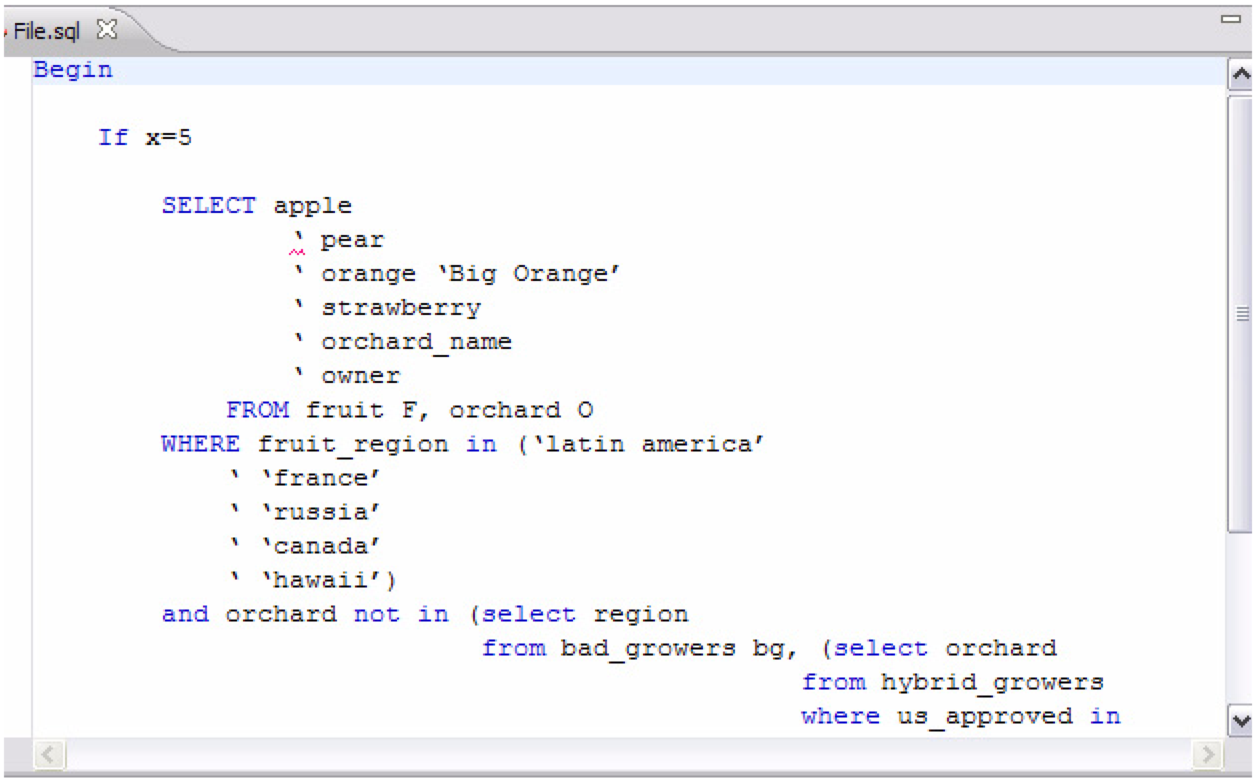
The following examples display a list of code formatting parameters and the resultant output in SQL Editor, based on the same set of SQL statements.

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

Custom Code Formatting Parameter	Value (if applicable)
Place SQL keywords on their own line?	No
Indent size?	--
Indent batch blocks?	Yes
Number of new lines to insert	1
Indent Size	5
Right Margin?	80
Stacked parentheses when they contain multiple items?	No
Stack parentheses when list contains ___ or more items.	--
Indent Size?	5
New line after first parentheses?	No
Indent content of conditional and looping constructs?	Yes
Number of new lines to insert?	1
Indent size?	5

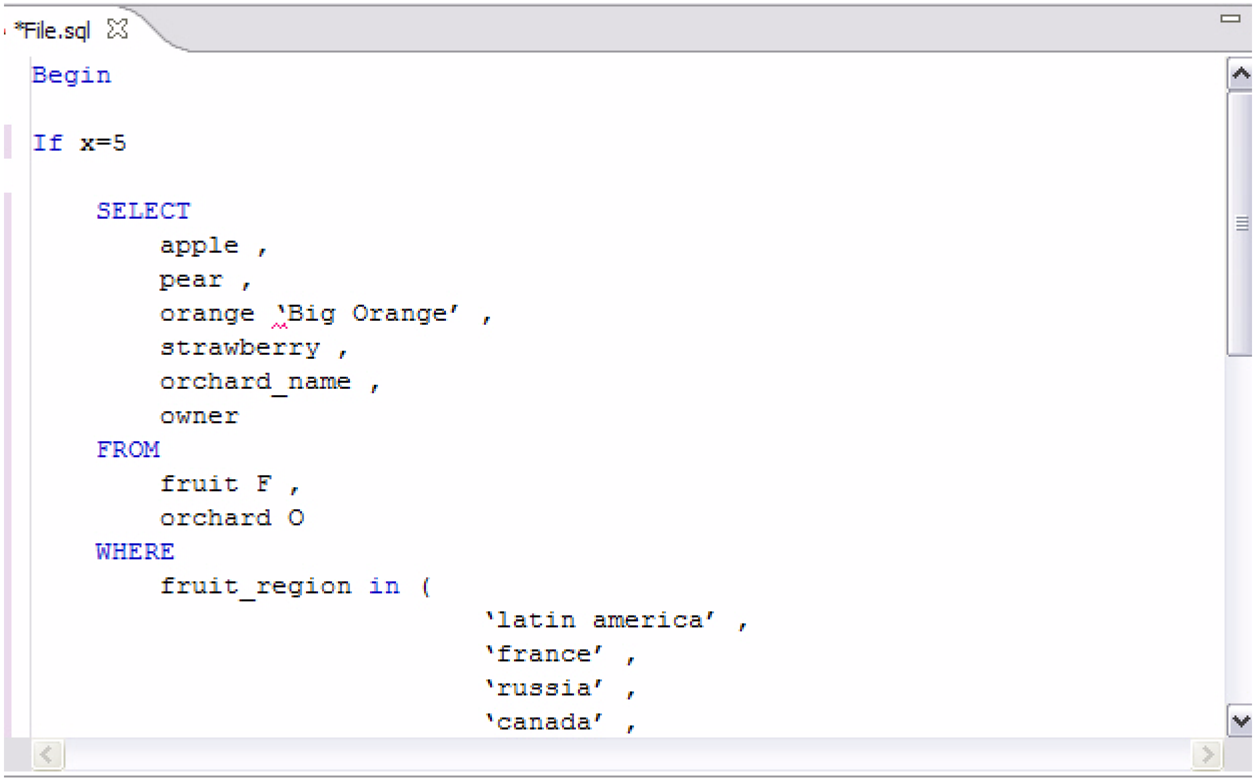


```
Begin

    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.

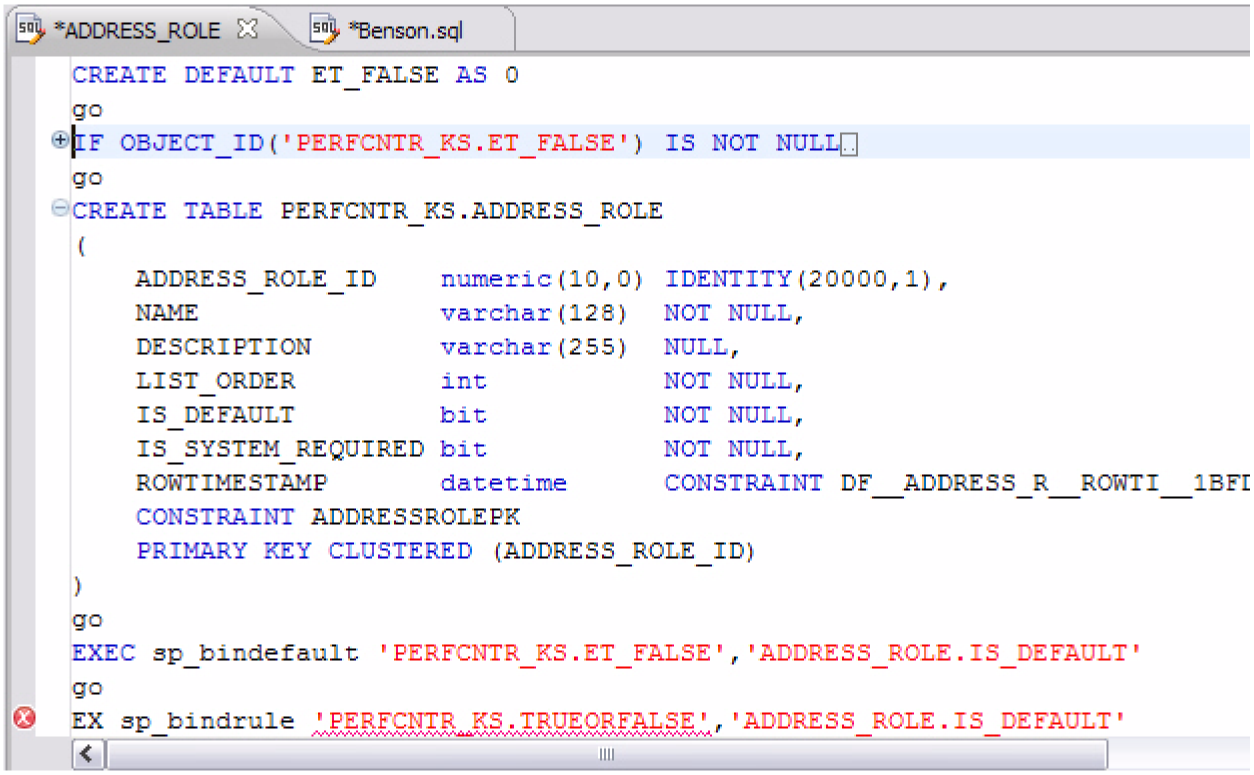
Custom Code Formatting Parameter	Value (if applicable)
Stack commas separated by lists?	Yes
Stack Lists with ___ or more items.	2
Indent Size?	0
Preceding commas?	--
Spaces after comma?	Yes
Trailing commas?	Yes
Spaces before comma?	2
Right align FROM and WHERE clauses with SELECT statement?	No
Align 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
Stack parentheses when list contains ___ or more items.	2
Indent Size?	2
New line after first parentheses?	Yes
Indent content of conditional and looping constructs?	--
Number of new lines to insert?	1
Indent size?	5


*File.sql

```
Begin

If x=5

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

Understanding Code Folding

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



```
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__1BFI
    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.

NOTE: Code quality checks are only applicable to Oracle data sources.

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	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. 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. **Example** The following statement is missing a valid JOIN criteria: SELECT * FROM employee e,customer c, sales_order s WHERE e.employee_id = c.salesperson_id The code quailty check fixes the above statement by adding an AND clause: SELECT * FROM employee e,customer c, sales_order s WHERE e.employee_id = c.salesperson_id **AND s.customer_id = c.customer_id**
Invalid or missing outer join operator	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. The code quality check detects invalid or missing join operators in the code and suggests fixes with regards to using the the proper join operators. **Example** The following statement is missing an outer join operator: SELECT * FROM employee e, customer c WHERE e.employee_id = c.salesperson_id (+) AND c.state = 'CA' The code quality check fixes the above statement by providing the missing outer join operator to the statement: SELECT * FROM employee e,customer c WHERE e.employee_id = c.salesperson_id(+) AND **c.state(+)** = 'CA'
Transitivity issues	The performance of statements can sometimes be improved by adding join criteria, even if a join is fully defined. If this alternate join critieria is missing in a statement, it can restrict the selection of an index in Oracle's optimizer and cause performance problems. The code quality check detects possible join conditions by analyzing the existing conditions in a statement and calculating the missing, alternative join criteria. **Example** The following statement contains a transitivity issue with an index problem: SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id The code quality check fixes the above statement with a tranisitivity issue by adding the missing join condition: 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**

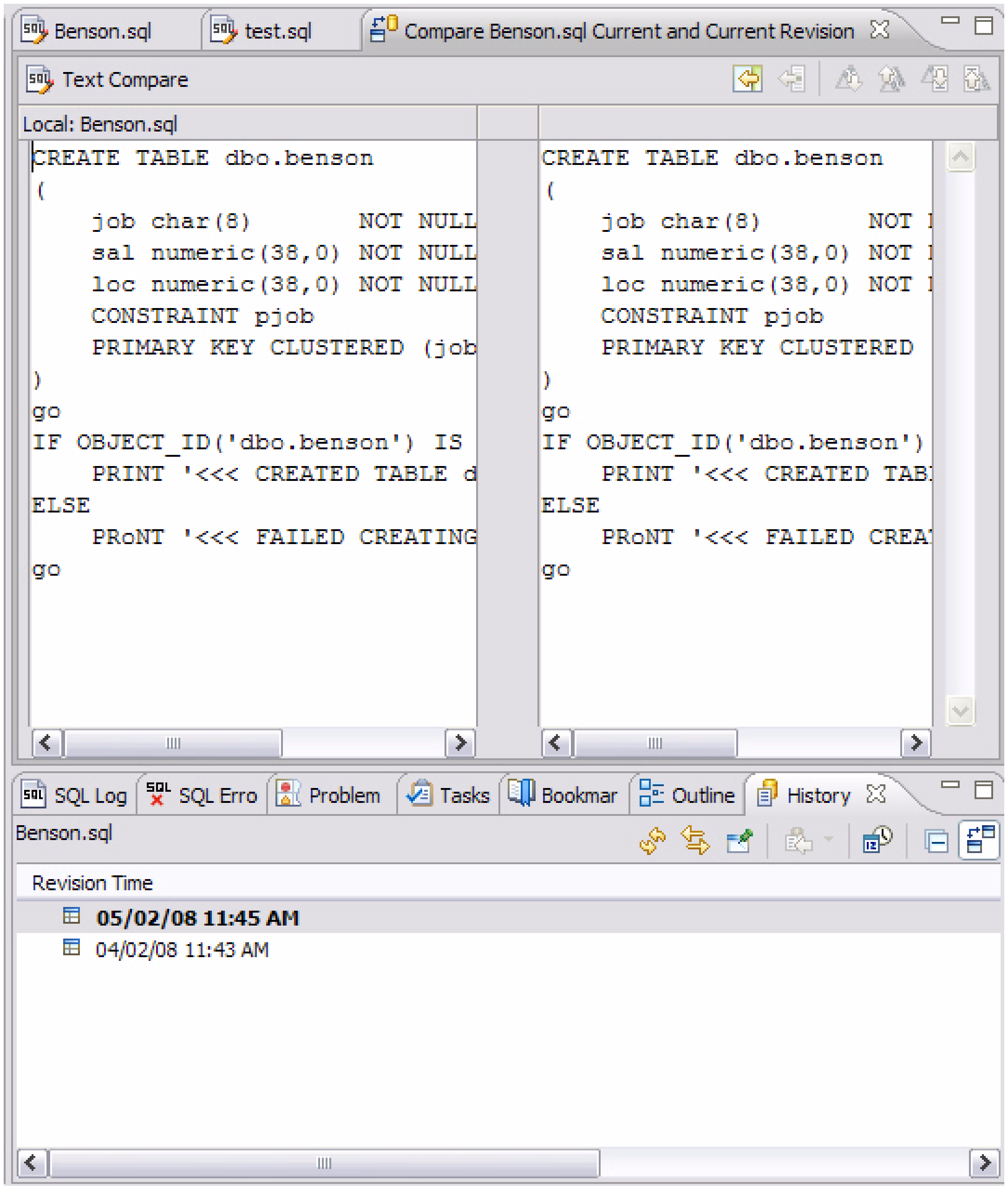
Code Quality Check Type	Definition
Nested query in WHERE clause	It is considered bad format to place sub-queries in the WHERE clause of a statement, and such clauses can typically be corrected by moving the sub-query to the FROM clause instead, which preserves the meaning of the statement while providing more efficient code. 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. **Example** The following statement contains a sub-query that contains an incorrect placement of a WHERE statement: SELECT * FROM employee WHERE employee_id = (SELECT MAX(salary) FROM employee) The code quality check fixes the above statement by correcting the sub-query issue: SELECT employee.* FROM employee **(SELECT DISTINCT MAX(salary) col1** FROM employee) t1 WHERE employee_id = t1.col1
Wrong place for conditions in a HAVING clause	When utilizing the HAVING clause in a statement It is recommended to include as few conditions as possible while utilizing the HAVING clause in a statement. DB Optimizer detects all conditions in a given HAVING statement and suggests equivalent expressions that can benefit from existing indexes. **Example** The following statement contains a HAVING clause that is in the wrong place: SELECT col_a, SUM(col_b) FROM table_a GROUP BY col_a HAVING col_a > 100 The code check fixes the above statement by replacing the HAVING clause with equivalent expressions: SELECT col_a, SUM(col_b) FROM table_a WHERE col_a > 100 GROUP BY col_a
Index suppressed by a function or an arithmetic operator	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. The code quality check detects these conditions and suggests equivalent expressions that benefit from existing indexes. **Example** The following statement includes an indexed column as part of an arithmetic operator: SELECT * FROM employee WHERE 1 = employee_id - 5 The code quality check fixes the above statement by reconstructing the WHERE clause: SELECT * FROM employee **WHERE 6 = employee_id**

Code Quality Check Type	Definition
Mismatched or incompatible column types	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. The code quality check flags mismatched or incompatible column types and warns that it is not valid code. **Example** Consider the following statement if Table A contains the column col int and Table B contains the column col 2 varchar(3): SELECT * FROM a, b WHERE a.col = b.col; 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.
Null column comparison	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. The code quality check flags occurrences of the !=NULL condition and replaces them with the IS NULL operator. **Example** The following statement includes an incorrect col = NULL expression: SELECT * FROM employee WHERE manager_id = NULL The code quality check replaces the incorrect expression with an IS NULL clause: SELECT * FROM employee WHERE manager_id **IS NULL**

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**.



```
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
    PRINT '<<< CREATED TABLE d
ELSE
    PRoNT '<<< FAILED CREATING
go
```

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

Revision Time
05/02/08 11:45 AM
04/02/08 11:43 AM

- 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.

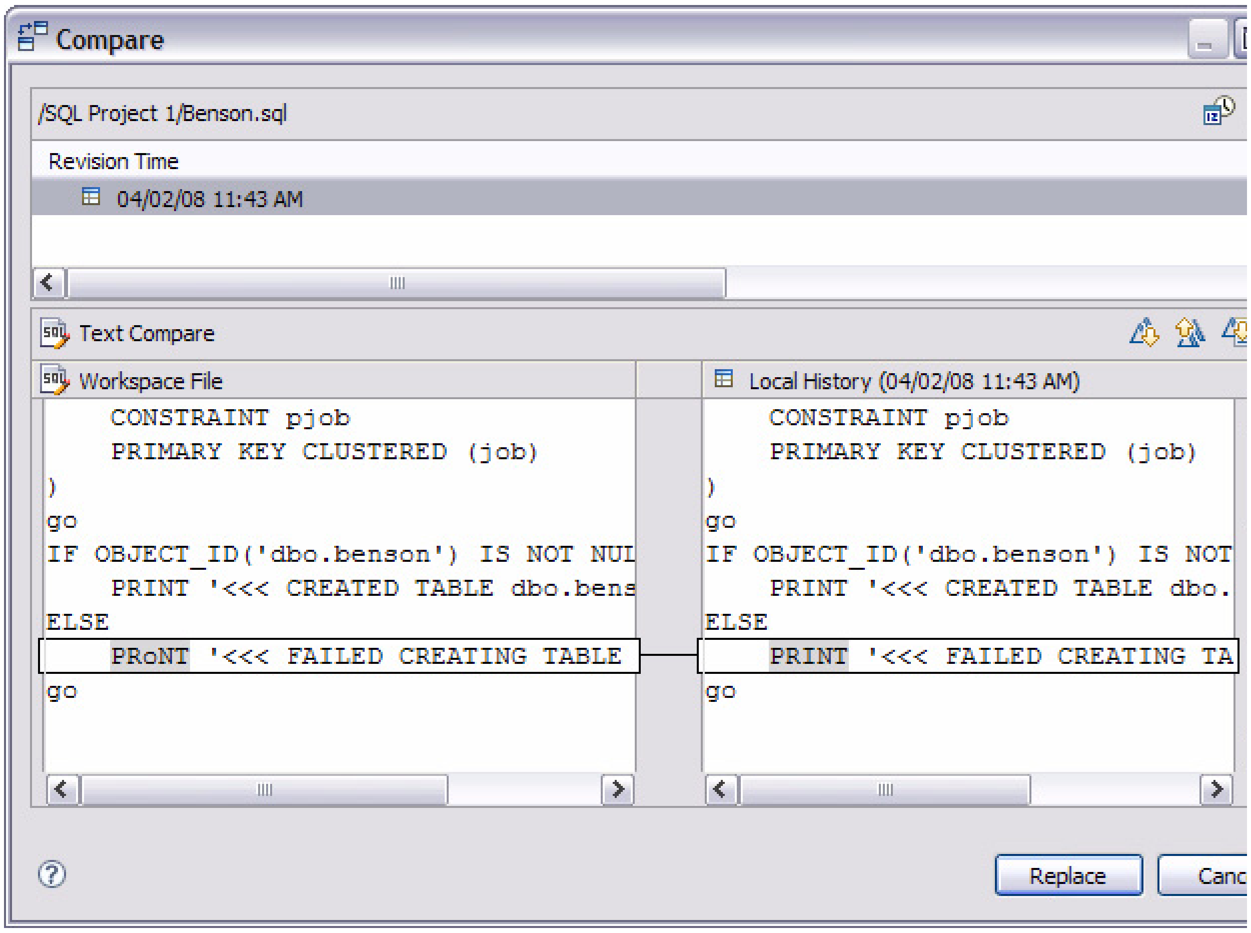
Revert to an Old Version of a File

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.



```
    CONSTRAINT pjob
    PRIMARY KEY CLUSTERED (job)
)
go
IF OBJECT_ID('dbo.benson') IS NOT NUL
    PRINT '<<< CREATED TABLE dbo.bens
ELSE
    PRoNT '<<< FAILED CREATING TABLE
go
```

```
    CONSTRAINT pjob
    PRIMARY KEY CLUSTERED (job)
)
go
IF OBJECT_ID('dbo.benson') IS NOT
    PRINT '<<< CREATED TABLE dbo.
ELSE
    PRINT '<<< FAILED CREATING TA
go
```

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

3 Click **Replace**.

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

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

Delete a SQL File

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

Executing SQL Files

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

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

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

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

To execute a file:

Open the SQL file you want to run, ensure it is associated with the correct database, and click **Execute**. DB Optimizer 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.

To execute a transaction:

To execute transactions, you need to ensure that the auto commit feature is turned off. See Set 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**. DB Optimizer executes the transaction on the data source you specified.

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

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

To commit a transaction:

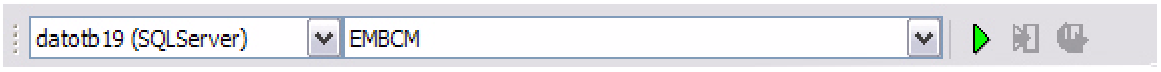
Open the transaction file you want to commit, ensure it is associated with the correct database, and click **Commit Transaction**. DB Optimizer commits the transaction on the data source you specified.

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

Associate a 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 lists on the Toolbar are used to display and specify a data source in relation to the specified SQL Editor file. The menus contain a list of all registered data sources. Additionally, on platforms that support catalogs, files can be associated with these as well.



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

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

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

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

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

Execute SQL Code

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

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

To execute code:

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

To execute a transaction:

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

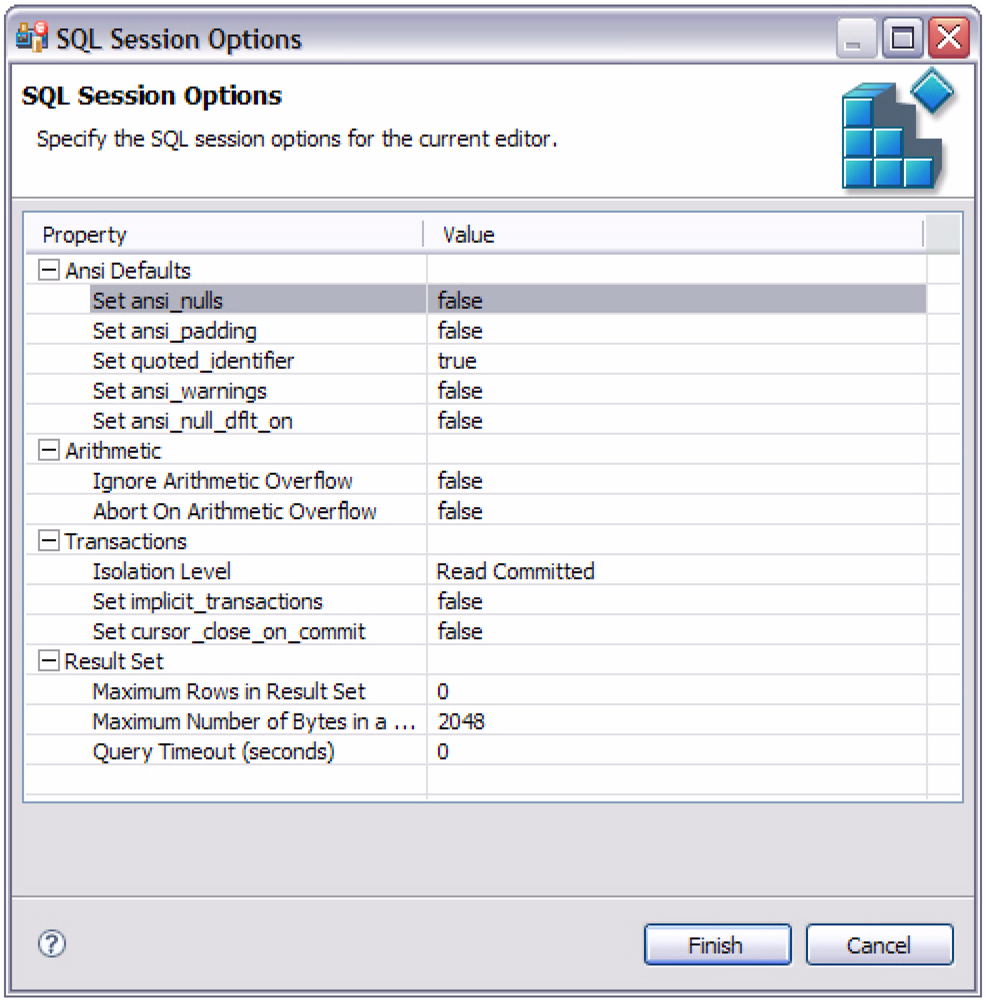
To commit a transaction:

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

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

Configure a SQL Session

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



Property	Value
Ansi Defaults	
Set ansi_nulls	false
Set ansi_padding	false
Set quoted_identifier	true
Set ansi_warnings	false
Set ansi_null_dflt_on	false
Arithmetic	
Ignore Arithmetic Overflow	false
Abort On Arithmetic Overflow	false
Transactions	
Isolation Level	Read Committed
Set implicit_transactions	false
Set cursor_close_on_commit	false
Result Set	
Maximum Rows in Result Set	0
Maximum Number of Bytes in a ...	2048
Query Timeout (seconds)	0

To modify SQL session options:

1 Click the SQL Session Options icon in the Toolbar.

The **SQL Session Options** dialog appears.

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

3 Click **Finish**.

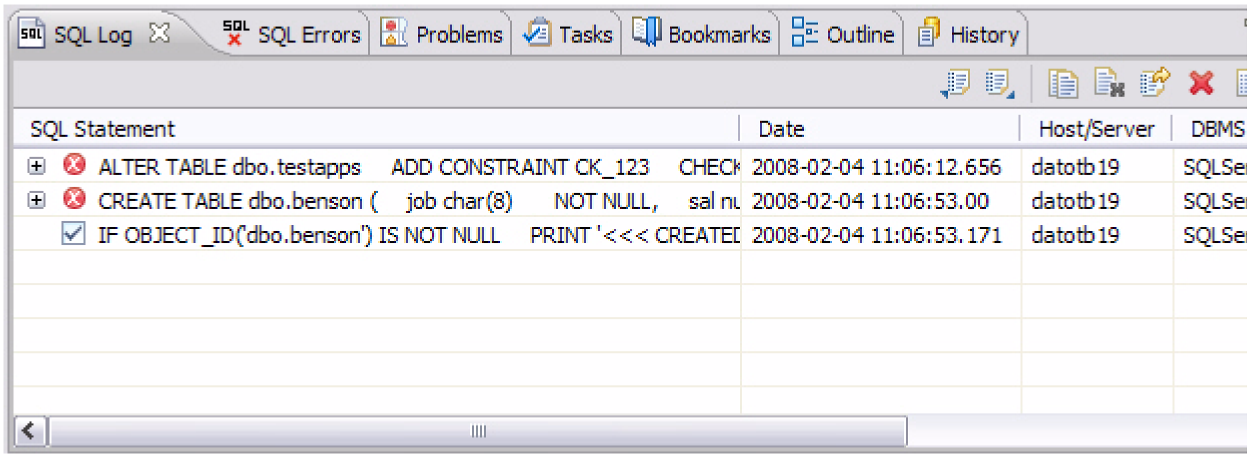
The session options will be changed and DB Optimizer will execute the code as specified when you execute it.

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

Troubleshooting

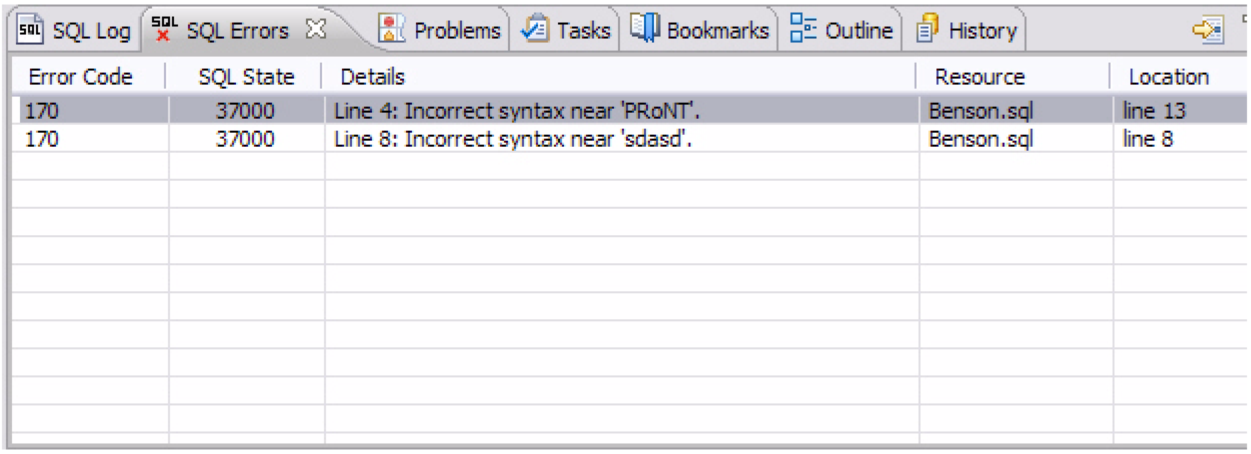
DB Optimizer contains a number of views used exclusively to log and monitor the SQL development process.

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



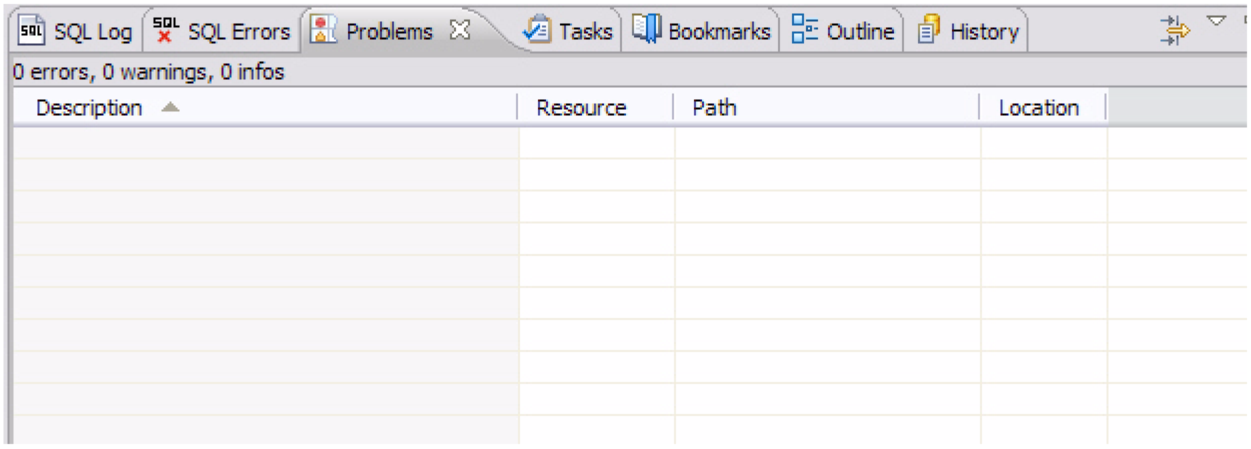
SQL Statement	Date	Host/Server	DBMS
ALTER TABLE dbo.testapps ADD CONSTRAINT CK_123 CHECK	2008-02-04 11:06:12.656	datotb19	SQLSe
CREATE TABLE dbo.benson (job char(8) NOT NULL, sal nu	2008-02-04 11:06:53.00	datotb19	SQLSe
IF OBJECT_ID('dbo.benson') IS NOT NULL PRINT '<<< CREATEI	2008-02-04 11:06:53.171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

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



Error Code	SQL State	Details	Resource	Location
170	37000	Line 4: Incorrect syntax near '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**.



0 errors, 0 warnings, 0 infos

Description	Resource	Path	Location

See also:

[View Log Details]

[Maintain Logs]

[Filter Logs]

[Import and Export Error Logs]

[Find and Fix SQL Code Errors]

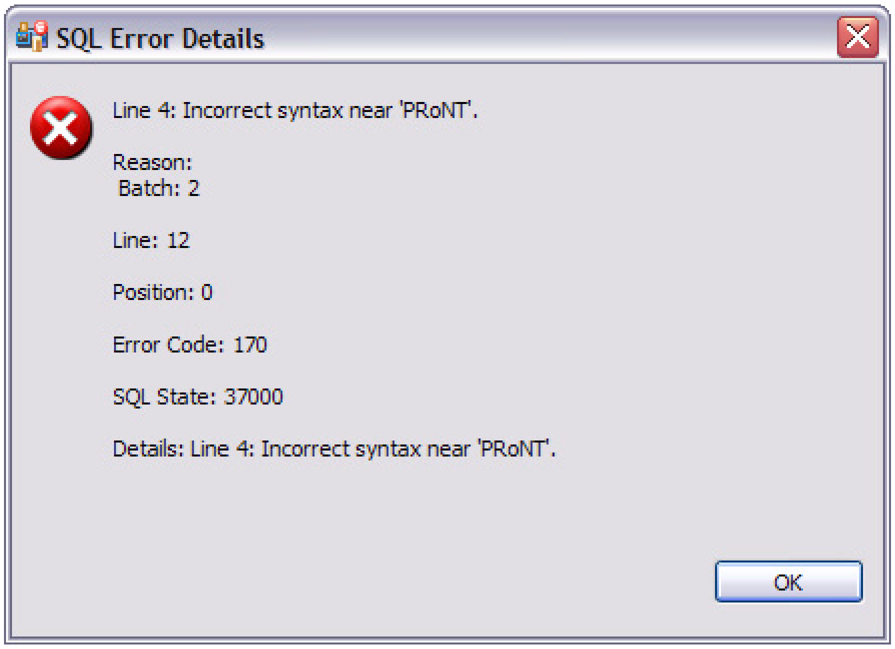
[Find and Fix Other Problems]

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**.



Line 4: Incorrect syntax near 'PRoNT'.

Reason:
 Batch: 2

Line: 12

Position: 0

Error Code: 170

SQL State: 37000

Details: Line 4: Incorrect syntax near 'PRoNT'.

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

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

To view details about Problems

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

Maintain Logs

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

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

To maintain log entries:

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

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

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

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

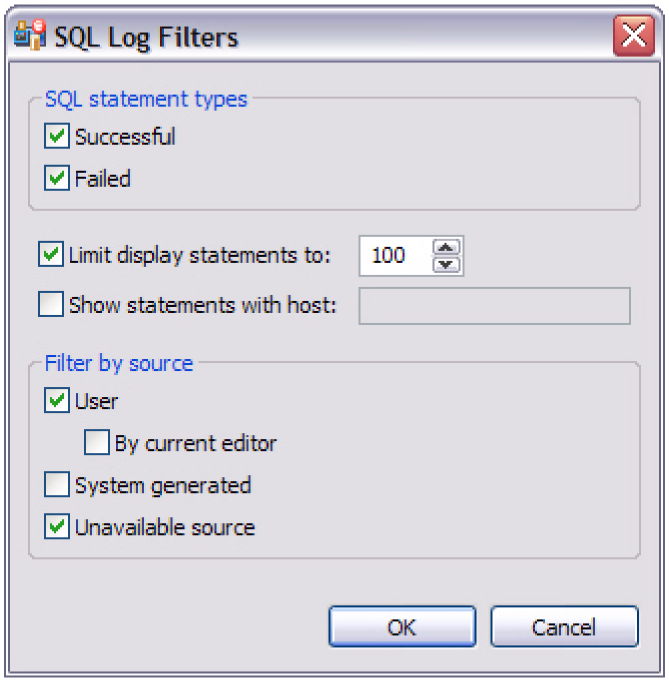
- To restore the **Error Log** view based on the entries contained in the **.log** file, select **Restore Log** from the shortcut menu. The View is restored based on the entries in the **.log** file.

Filter Logs

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

To filter the SQL Log:

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

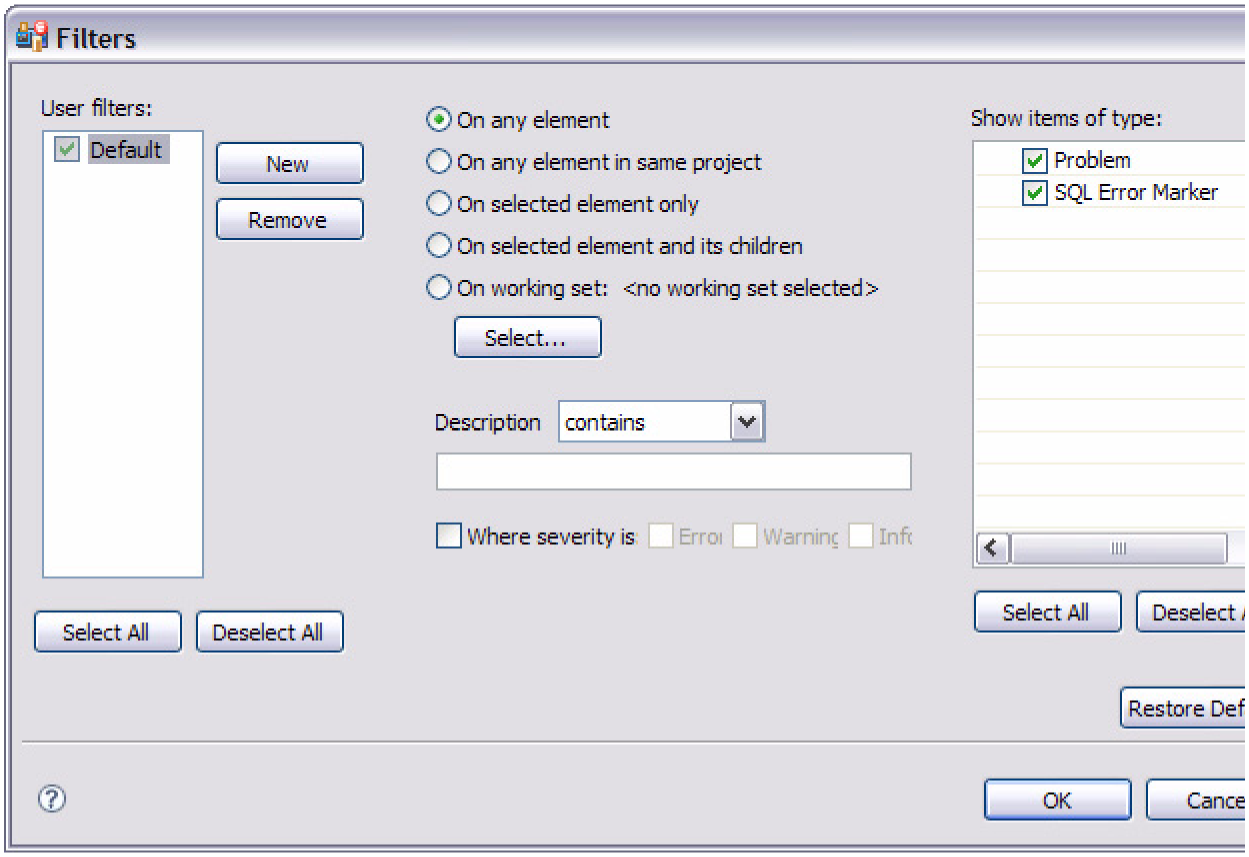


- [✓] Successful
- [✓] Failed
- [✓] Limit display statements to: 100
- [ ] Show statements with host:
- [✓] User
  - [ ] By current editor
- [ ] System generated
- [✓] Unavailable source

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

To filter the Problems log:

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



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

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

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

Profile Criteria	Description
Working Set	The options located in the center of the dialog enable you to filter problems based on defined Working Sets. A Working Set is a collection of user-defined Project files that you can organize, as needed, in DB Optimizer. 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. 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.
Character String	Use the **Description** list to select **contains** or **doesn’t contain**, as needed, and type the character string in the field below the list. The Problems view is filtered to only contain, or omit, problem descriptions that fully or partially match the string value.
Problem Severity	Select the **Where severity is** check box and choose **Error**, **Warning**, **Info**, or some combination of the three check boxes. Only entries whose severity matches the check boxes you have selected remain visible in the Problems view.
Problem Type	The options in the **Show items of type** list on the right-hand side of the dialog enable you to filter problems by type. Deselect **Problem** to remove any system entries from the view, or deselect **SQL Error Marker** to remove any SQL code entries from the view.

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

Import and Export Error Logs

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

To export the SQL Log:

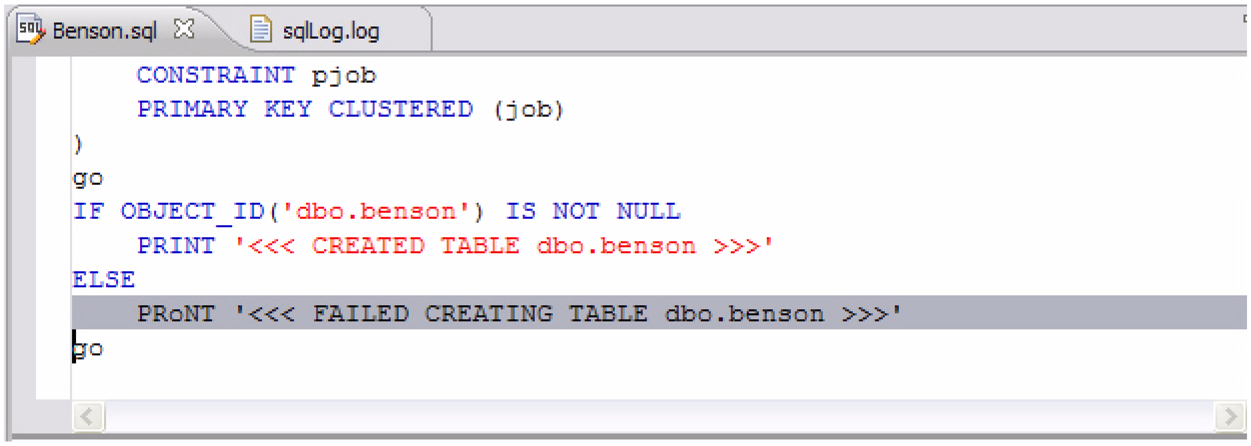
Right-click the **SQL Log** view and choose **Export Log**. The log is saved in the specified directory path with a **.log** extension.

To import the Error Log:

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

Find and Fix SQL Code Errors

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



```
    CONSTRAINT pjob
    PRIMARY KEY CLUSTERED (job)
)
go
IF OBJECT_ID('dbo.benson') IS NOT NULL
    PRINT '<<< CREATED TABLE dbo.benson >>>'
ELSE
    PROoNT '<<< FAILED CREATING TABLE dbo.benson >>>'
go
```

To navigate to the source of a SQL error entry:

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

Find and Fix Other Problems

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

The first column of the Problems view displays an icon that denotes the type of line item, the category, and the description. Click the problem and DB Optimizer 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.

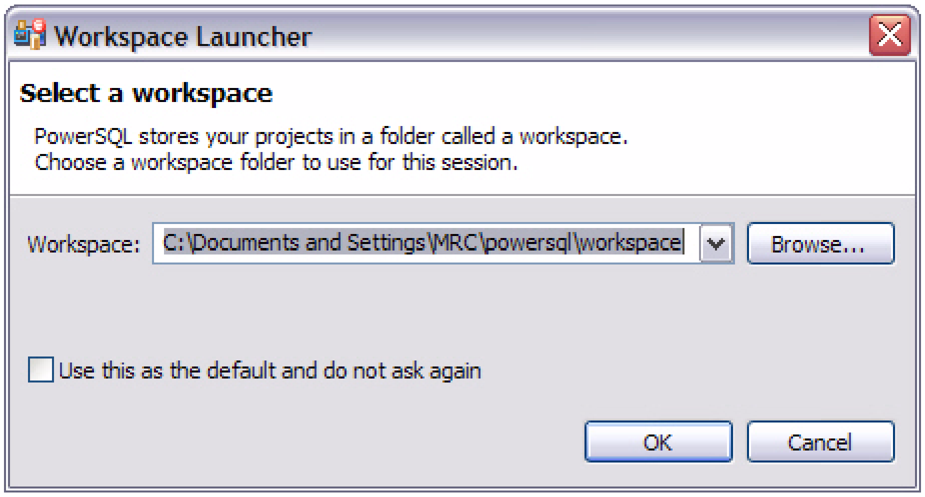
Problems can sometimes be fixed via the **Quick Fix** command in the shortcut menu. The list of possible resolutions is displayed. To fix other similar problems, use the **Find Similar Problems** command and DB Optimizer will locate other problems in the view to which the same fix can be applied.

Configuring DB Optimizer

Initial Setup

Specify a Workspace

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



Select a workspace

PowerSQL stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\MRC\powersql\workspace

Use this as the default and do not ask again

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

License DB Optimizer

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

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

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

Customizing DB Optimizer (Preferences)

To customize various aspects of DB Optimizer, select **Window > Preferences > SQL Development**. For information on the other preference categories, see **Help > Help Contents > Workbench User Guide**.

Set Cache Configuration Preferences

The **Data Source Object Cache** is a local repository that stores the schema of registered data sources in DB Optimizer. It is automatically set to cache information about data sources registered in the development environment.

By default, the **Data Source Object Cache** caches all catalogs, functions, procedures, tables, and views. Additionally, after the initial cache, the object cache performs incremental caches.

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

By default, when a data source connects to DB Optimizer, the Object Cache automatically begins indexing its schema.

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

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

To configure the cache:

1. Select **Window > Preferences > SQL Development > Cache Configuration**. Change the settings as appropriate:
   - The tree view displays a list of database objects as they are organized in the Database Explorer view. Use the check boxes beside each object to specify the data sources that are to be included in the cache process.
   - Select **Clear Cache on Startup** to delete the Object Cache each time the application is started.
   - Select **Apply SQL Filters** if you want to apply any pre-defined filters to the cache.
   - If you are having performance problems due to a caching issue (such as a configuration error), the **Stop Caching**, **Clear Cache**, and **Start Caching** buttons enable you to stop, clear, and/or restart the cache process, respectively.
   - The **Max. number of objects to cache** field indicates how many logical data source objects can be cached before the Object Cache has reached maximum cache size.
   - The options in the **When the maximum cache size is reached** pane specify how the Object Cache should handle the caching process once it has reached the maximum. Select **Grow the cache automatically until all data is cached** to override the maximum cache setting, or select **Stop caching** to terminate the cache process when the maximum cache size is reached.
   - The **Objects to include in cache** pane contains a list of data source objects. Select or clear the check boxes beside each data source object to indicate the specific data source objects that are included and excluded, respectively, from the cache process.
   - The **Cache Expiration Time (hours)** setting indicates that a cache job will not start automatically until the specified number of hours have passed. The cache can also be started manually via **Start Caching**.
2. When you are finished configuring the Object Cache, click **Apply** to save your changes.

Set SQL Editor Preferences

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

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

- **Enable SQL Parser** indicates that the parser that provides code development features in SQL Editor is enabled. If this option is disabled, functions such as code formatting, auto completion, semantic validation, and hyperlinks, will not be available. The options below the check box enable you to limit (by file size) when the parser is enabled when dealing with code on a file-by-file basis.
- **Parsing Delay** specifies the amount of time that the parser delays prior to activating features after a keystroke.
- **Severity Level for Semantic Validation Problems** determines how semantic code errors are flagged in the editor and the **Problems** view.

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

Set SQL Execution Preferences

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

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

Set Code Assist Preferences

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

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

- **Enable Auto Activation** enables or disables code assist functionality.
- **Insert Single Proposals Automatically** specifies if only a single code completion suggestion is returned, it is inserted automatically.
- **Fully Qualified Completions Automatically** specifies if code completion results are returned specific (fully qualified), rather than the minimum required to identify the object.
- **Code Assist Color Options** specifies the color formatting of code completion proposals. Select background or foreground options from the menu and modify them as appropriate.

Set Code Formatter Preferences

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

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

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

- Click **New** to define additional code formatting profiles.
- Click **Edit** to modify existing profiles. You can modify how code characters appear in the interface and how SQL Editor determines line breaks.

- Click **Rename** to change the name of an existing profile. The new name cannot be the same as another existing profile.

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

Set Results View Preferences

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

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

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

Set Syntax Coloring Preferences

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

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

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

Set SQL Filter Preferences

See Filter Database Objects for instructions.

Reference

The following topics provide reference details:

- Database Objects
- DBMS Connection Parameters by Platform
- Hints Reference

Database Objects

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

In DB Optimizer, database objects are stored in **Data Source Explorer** as subnodes of individual, pertinent databases.

Database Object	DBMS Platforms	Notes
Aliases	DB2	An alias is an alternate name that references a table, view, and other database objects. An alias can also reference another alias as long as the aliases do not reference one another in a circular or repetitive manner. Aliases are used in view or trigger definitions in any SQL statements except for table check-constraint definitions. (The table or view name must be referenced in these cases.) Once defined, an alias is used in query and development statements to provide greater control when specifying the referenced object. Aliases can be defined for objects that do not exist, but the referenced object must exist when a statement containing the alias is compiled. Aliases can be specified for tables, views, existing aliases, or other objects. Create Alias is a command available on the shortcut menu.
Check Constraints	All	A check constraint is a search condition applied to a table. When a check constraint is in place, Insert and Update statements issued against the table will only complete if the statements pass the constraint rules. Check constraints are used to enforce data integrity when it cannot be defined by key uniqueness or referential integrity restraints. A check condition is a logical expression that defines valid data values for a column.
Clusters	Oracle	A cluster is a collection of interconnected, physical machines used as a single resource for failover, scalability, and availability purposes. Individual machines in the cluster maintain a physical host name, but a cluster host name must be specified to define the collective as a whole. To create a cluster, you need the CREATE CLUSTER or CREATE ANY CLUSTER system privilege.

Database Object	DBMS Platforms	Notes
Database Links	Oracle	A database link is a network path stored locally, that provides the database with the ability to communicate with a remote database. A database link is composed of the name of the remote database, a communication path to the database, and a user ID and password (if required). Database links cannot be edited or altered. To make changes, drop and re-create.
Foreign Keys	All	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. 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. Foreign key names are automatically assigned if one is not specified.
Functions	DB2, Oracle	A function is a relationship between a set of input data values and a set of result values. For example, the TIMESTAMP function passes input data values of type DATE and TIME, and the result is TIMESTAMP. Functions can be built-in or user-defined. Built-in functions are provided with the database. They return a single value and are part of the default database schema. User-defined functions extend the capabilities of the database system by adding function definitions (provided by users or third-party vendors) that can be applied in the database engine itself. A function is identified by its schema, a function name, the number of parameters, and the data types of its parameters. Access to functions is controlled through the EXECUTE privilege. GRANT and REVOKE statements are used to specify who can or cannot execute a specific function or set of functions.
Groups	All	Groups are units that contain items. Typically, groups contain the result of a single business transaction where several items are involved. For example, a group is the set of articles bought by a customer during a visit to the supermarket.

Database Object	DBMS Platforms	Notes
Indexes	All	An index is an ordered set of pointers to rows in a base table. 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. 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. Indexes are also used to ensure uniqueness. Atable with a unique index cannot have rows with identical keys. DB2: Allow Reverse Scans, Percent Free (Lets you type or select the percentage of each index page to leave as free space when building the index, from 0 to 99), Min Pct Used (Lets you type or select the minimum percentage of space used on an index leaf page. If, after a key is removed from an index leaf page, the percentage of space used on the page is at or below integer percent, an attempt is made to merge the remaining keys on this page with those of a neighboring page. If there is sufficient space on one of these pages, the merge is performed and one of the pages is deleted. The value of integer can be from 0 to 99. Oracle: The Logging, No Sort, Degrees, and Instances properties are documented in the editor.
Java Classes	Oracle	A model or template, written in Java language, used to create objects with a common definition and common properties, operations and behavior. Java classes can be developed in Eclipse (or another Java development environment such as Oracle JDeveloper) and moved into an Oracle database to be used as stored procedures. Java classes must be public and static if they are to be used in this manner. When writing a class to be executed within the database, you can take advantage of a special server-side JDBC driver. This driver uses the user’s default connection and provides the fastest access to the database. Java classes become full-fledged database objects once migrated into the database via the loadjava command-line utility or the SQL CREATE JAVA statement. A Java class is published by creating and compiling a call specification for it. The call spec maps a Java method’s parameters and return type to Oracle SQL types. Once a Java class is developed, loaded, and published -- the final step is to execute it.
Java Resources	Oracle	A Java resource is a collection of files compressed in a .jar file.

Database Object	DBMS Platforms	Notes
Libraries	Oracle	A library is a configurable folder for storing and sharing content with an allocated quota. Multiple libraries may exist in the same database environment. A library is a special type of folder in Oracle Content Services. Unlike Containers and regular folders, each library has a Trash Folder and an allocated amount of disk space. A library is composed of a name (mandatory), description, quota, path, and library members. The library service allows you to create folders, list quotas, and manage categories, workflow, trash folders, and versioning. The Library service does not allow you to create or upload files.
Materialized Views	Oracle	A database object that contains the results of a query. They are local copies of data located remotely, or are used to create summary tables based on aggregations of table data. Materialized views are also known as snapshots. A materialized view can query tables, views, and other materialized views. Collectively, these are called master tables (a replication term) or detail tables (a data warehouse term). For replication purposes, materialized views allow you to maintain copies of remote data on your local node. These copies are read-only. If you want to update the local copies, you need to use the Advanced Replication feature. You can select data from a materialized view as you would from a table or view. For data warehousing purposes, the materialized views commonly created are aggregate views, single-table aggregate views, and join views.
Materialized View Logs	Oracle	Because Materialized Views are used to return faster queries (a query against a materialized view is faster than a query against a base table because querying the materialized view does not query the source table), the Materialized View often returns the data at the time the view was created, not the current table data. There are two ways to refresh data in Materialized Views, manually or automatically. In a manual refresh, the Materialized View is completely wiped clean and then repopulated with data from the source tables (this is known as a complete refresh). If source tables have changed very little, however, it is possible to refresh the Materialized View only for changed records -- this is known as a fast refresh. In the case of Materialized Views that are updated via fast refresh, it is necessary to create Materialized View Logs on the base tables that compose the Materialized View to reflect the changes. If the number of entries in this table is too high, it is an indication that you might need to refresh the Materialized Views more frequently to ensure that each update does not take longer than it needs. Select owner, then select from tables with Materialized Views, etc.

Database Object	DBMS Platforms	Notes
Oracle Job Queue	Oracle	The Oracle Job Queue allows for the scheduling and execution of PL/SQL stored procedures at predefined times and/or repeated job execution at regular intervals, as background processes. For example, you could create a job in the Oracle Job Queue that processed end-of-day accounting -- a job that must run every weekday, but can be run unattended, or you could create a series of jobs that must be run sequentially -- such as jobs that might be so large, that in order to reduce CPU usage, only one is run at a time. Runs PL/SQL code at specified time or on specified schedule, can enable/disable.
Outlines	Oracle	Oracle preserves the execution plans of “frozen” access paths to data so that it remains constant despite data changes, schema changes, and upgrades of the database or application software through objects named stored outlines. Outlines are useful for providing stable application performance and benefit high-end OLTP sites by having SQL execute without having to invoke the cost-based optimizer at each SQL execution. This allows complex SQL to be executed without the additional overhead added by the optimizer when it performs the calculations necessary to determine the optimal access path to the data.
Packages	All	A package is a procedural schema object classified as a PL/SQL program unit that allows the access and manipulation of database information. 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. DB applications explicitly call packaged procedures as necessary with privileges granted, a user can explicitly execute any of the procedures contained in it. 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. Packages allow the database administrator or application developer to organize similar routines as well as offering increased functionality and database performance. 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. 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. 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.

Database Object	DBMS Platforms	Notes
Package Bodies	Oracle	A package body is a package definition file that states how a package specification will function. In contrast to the entities declared in the visible part of a package, the entities declared in the package body are only visible within the package body itself. As a consequence, a package with a package body can be used for the construction of a group of related subprograms in which the logical operations available to clients are clearly isolated from the internal entities.
Primary Keys	All	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. 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 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. Oracle: If an index constraint has been defined for a table, the constraint status for the table’s primary key cannot be set to Disabled.
Procedures	All	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. 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. 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. 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.
Profiles	Oracle	Profiles are a means to limit resources a user can use by specifying limits on kernel and password elements. Additionally, Profiles can be used to track password histories and the settings of specific profiles may be queried. The following kernel limits may be set: maximum concurrent sessions for a user, CPU time limit per session, maximum connect time, maximum idle time, maximum blocks read per session, maximum blocks read per call, and maximum amount of SGA.

Database Object	DBMS Platforms	Notes
Roles	Oracle	A role is a set or group of privileges that can be granted to users to another role. A privilege is a right to execute a particular type of SQL statement or to access another user's object. For example: the right to connect to a database, the right to create a tale, the right to select rows from another user's table, the right to execute another user's stored procedure. System privileges are rights to enable the performance of a particular action, or to perform a particular action on a particular type of object. Roles are named groups of related privileges that you grant users or other roles. Roles are designed to ease the administration of end user system and object privileges. However, roles are not meant to be used for application developers, because the privileges to access objects within stored progmmatic constructs needs to be granted directly.
Sequences	DB2, Oracle	A sequence generates unique numbers. Sequences are special database objects that provide numbers in sequence for input into a table. They are useful for providing generated primary key values and for the input of number type columns such as purchase order, employee number, sample number, and sales order number. Sequences are created by use of the CREATE SEQUENCE command.
Structured Types	DB2	Structured Types are useful for modeling objects that have a well-defined structure that consists of attributes. Attributes are properties that describe an instance of the type. A geometric shape, for example, might have as attributes its list of Cartesian coordinates. A person might have attributes of name, address, and so on. A department might have a name or some other attribute.
Synonyms	Oracle	A synonym is an alternate name for objects such as tables, views, sequences, stored procedures, and other database objects. A synonym is an alias for one of the following objects: table, object table, view, object view, sequence, stored procedure, stored function, package, materialized view, java class, user-defined object type or another synonym.
Tables	All	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. A base table is used to hold persistent user data. 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. A summary table is a table defined by a query that is also used to determine the data in the table.
Tablespaces	DB2, Oracle	A tablespace is a storage structure containing tables, indexes, large objects, and long data. Tablespaces reside in database partition groups. They allow you to assign the location of database and table data directly onto containers. (A container can be a directory name, a device name, or a file name.) This can provide improved performance and more flexible configuration.

Database Object	DBMS Platforms	Notes
Triggers	All	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. 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.
Undo Segments	Oracle	In an Oracle database, Undo tablespace data is an image or snapshot of the original contents of a row (or rows) in a table. The data is stored in Undo segments in the Undo table space. When a user begins to make a change to the data in a row in an Oracle table, the original data is first written to Undo segments in the Undo tablespace. The entire process (including the creation of the Undo data is recorded in Redo logs before the change is completed and written in the Database Buffer Cache, and then the data files via the database writer (DBW) process.)
Unique Keys	All	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. A primary key is a special case of a unique key. A table cannot have more than one primary key. A foreign key is a key that is specified in the definition of a referential constraint. 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. Oracle: You cannot drop a unique key constraint that is part of a referential integrity constraint without also dropping the foreign key. To drop the referenced key and the foreign key together, check the Delete Cascade option for the foreign key. 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. Filegroup: Lets you select the filegroup within the database where the constraint is stored. Fill Factor: Lets you specify a percentage of how large each constraint can become.

Database Object	DBMS Platforms	Notes
Views	All	A view provides an alternate way of looking at the data in one or more tables. 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. 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.

DBMS Connection Parameters by Platform

The following topics provide connection details:

- IBM DB2 LUW
- Microsoft SQL Server
- JDBC Connection Parameters
- Oracle Connection Parameters
- Sybase Connection Parameters

IBM DB2 LUW

Connection Parameter	Description
Use Alias from IBM Client or Generic JDBC Configuration	If you choose to use the alias from the IBM client, select the appropriate alias name. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Schema ID (Optional)	The name of the database schema.
Function Path	Optional. Enter an ordered list of schema names to restrict the search scope for unqualified function invocations.
Security Credentials	The log on information required by DB Optimizer to connect to the data source.
Auto Connect	Automatically attempts to connect to the data source when selected in Data Source Explorer, without prompting the user for connection information.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer to initiate a JDBC standard access connection.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a data source. Often contains host and port numbers, as well as the name of the data source to which it connects. For example: **jdbc:postgresql://*host:port/database*** **jdbc:derby://*host:port/database***
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

Microsoft SQL Server

Connection Parameter	Description
Use Network Library Configuration	If the data source utilizes a network library, select this parameter. The corresponding connection parameter fields become available. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Host/Instance (JDBC Configuration)	The name of the data source.
Port (JDBC Configuration) (optional)	The listening port used in TCP/IP communications between DB Optimizer and the data source.
Protocol (JDBC Configuration)	The communication mechanism between DB Optimizer and the data source. Choose **TCP/IP** or **Named Pipes**.
Default Database (Optional)	The default SQL database name, as defined by the schema.
Security Credentials	The log on information required by DB Optimizer to connect to the data source.
Auto Connect	Automatically attempts to connect to the data source when selected in Data Source Explorer, without prompting the user for connection information.
Allow Trusted Connections	Enables trusted connections to the data source from DB Optimizer.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer to connect and communicate with the database.

Connection Parameter	Description
Connection URL (Advanced)	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: **jdbc:postgresql://*host:port/database*** **jdbc:derby://*host:port/database***
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

JDBC Connection Parameters

Connection Parameter	Description
Connect String	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: **jdbc:postgresql://*host:port/database*** **jdbc:derby://*host:port/database***
Data Source Name	The name of the data source to which you want DB Optimizer to connect.

Oracle Connection Parameters

Connection Parameter	Description
Use TNS Alias	If the data source is mapped to a net service name via **tnsnames.ora**, select this parameter. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.
Host/Instance (JDBC Configuration)	The name of the host machine on which the data source resides.
Port (JDBC Connection)	The listening port used in TCP/IP communications between DB Optimizer and the data source.
Type (JDBC Configuration)	Indicates if the data source is defined via a system identifier (SID) or a service name.
Service/SID Name (JDBC Configuration)	The name of the system identifier (SID) or service name that identifies the data source.
Security Credentials	The log on information required by DB Optimizer to connect to the data source.
Auto Connect	Automatically attempts to connect to the data source when selected in data source Explorer, without prompting the user for connection information.
Allow Trusted Connections	Enables trusted connections to the data source from DB Optimizer.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer to connect and communicate with the database.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: **jdbc:postgresql://*host:port/database*** **jdbc:derby://*host:port/database***
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

Sybase Connection Parameters

Connection Parameter	Description
Use Alias Information from your SQL.INI File	If the data source is mapped to a name via **SQL.INI**, select this parameter to use that name for connection. Otherwise, choose Generic JDBC Configuration and enter the connection parameters, as specified.

Connection Parameter	**Description**
Host/Instance (JDBC Connection)	The name of the host machine on which the data source resides.
Port (JDBC Connection)	The listening port used in TCP/IP communications between DB Optimizer and the data source.
Default Database (JDBC Connection) (Optional)	The default database name, as defined by the schema.
JDBC Driver (Advanced)	The name of the JDBC Driver utilized by DB Optimizer to connect and communicate with the database.
Connection URL (Advanced)	Used by the JDBC Driver to connect with a database. Often contains host and port numbers, as well as the name of the database to which it connects. For example: **jdbc:postgresql://*host:port/database*** **jdbc:derby://*host:port/database***
Custom JDBC Driver Properties (Advanced)	The name and property value of any custom JDBC drivers associated with the data source.

Hints Reference

The following table lists the hints supported by DB Optimizer:

ALL_ROWS	AND_EQUAL	APPEND
CACHE	CHOOSE	CLUSTER
CURSOR_SHARING_EXACT	DRIVING_SITE	DYNAMIC_SAMPLING
EXPAND_GSET_TO_UNION	FACT	FIRST_ROWS
FULL	HASH	HASH_AJ
HASH_SJ	INDEX	INDEX_ASC
INDEX_COMBINE	INDEX_DESC	INDEX_FFS
INDEX_JOIN	INDEX_SS	INDEX_SS_ASC
INDEX_SS_DSC	LEADING	MERGE
MERGE_AJ	MERGE_SJ	MODEL_MIN_ANALYSIS
MONITOR	NL_AJ	NL_SJ
NO_EXPAND	NO_FACT	NO_INDEX
NO_INDEX_FFS	NO_INDEX_SS	NO_MERGE
NO_MONITOR	NO_PARALLEL	NO_PARALLEL_INDEX
NO_PARALLEL	NO_PUSH_SUBQ	NO_PX_JOIN_FILTER
NO_QUERY_TRANSFORMATION	NO_REWRITE	NO_STAR_TRANSFORMATION
NO_UNNEST	NO_USE_HASH	NO_USE_MERGE
NO_USE_NL	NO_XML_QUERY_REWRITE	NO_XMLINDEX_REWRITE
NOAPPEND	NOCACHE	NOFACT
NOPARALLEL	NOPARALLEL_INDEX	NOREWRITE
OPT_PARAM	ORDERED	ORDERED_PREDICATES
PARALLEL	PARALLEL_INDEX	PQ_DISTRIBUTE
PUSH_PRED	PUSH_SUBQ	PX_JOIN_FILTER
REWRITE	ROWID	RULE
STAR	STAR_TRANSFORMATION	UNNEST
USE_CONCAT	USE_HASH	USE_MERGE
USE_NL	USE_NL_WITH_INDEX	

NOTE: The following topics provide a quick reference, only. For detailed information, see your Oracle documentation.

ALL_ROWS

ALL_ROWS is a cost-based optimization approach that gathers the SELECT statement’s requested information from the database as fast as possible, letting you generate reports quickly.

AND_EQUAL

AND_EQUAL tells Oracle's optimizer to choose an execution plan that takes an access path that merges the scans of no less than two, and no more than five, single-column indexes.

APPEND

APPEND attaches data to a table when used as a hint for INSERT statements.

CACHE

CACHE places the blocks retrieved for your specified table at the most recently used end of the LRU in the buffer cache for full table scans. This a useful technique when working with small lookup tables.

CHOOSE

CHOOSE lets Oracle's optimizer select either a rule or cost-based approach for your SELECT statement. This choice is based on the presence of statistics for the tables referred to in your SQL. If the data dictionary does not have statistics for the tables, the optimizer chooses the rule-based approach. If at least one table has statistics in the data dictionary, the optimizer chooses the cost-based approach.

CLUSTER

CLUSTER tells Oracle's optimizer to access data by conducting a cluster scan for the table (name or alias) you specify in parentheses.

CURSOR_SHARING_EXACT

If the CURSOR_SHARING initialization parameter is set to replace literals in SQL statements with bind variable, the CURSOR_SHARING_EXECT hint disables this behavior. The SQL statement is executed without replacing literals with bind variables.

DRIVING_SITE

DRIVING_SITE specifies where a query is to be executed if it is not to be executed locally. If an execution takes place remotely using this hint, the results are returned to the local site.

DYNAMIC_SAMPLING

DYNAMIC_SAMPLING is used with simple SELECT statement with a single table to approximate cardinality if there are no existing statistics on the table. This hint instructs the optimizer how to control dynamic sampling to determine optimal predicate selectivity and statistics for tables and indexes.

EXPAND_GSET_TO_UNION

EXPAND_GSET_TO_UNION performs transformation on queries containing grouping sets (GROUP BY GROUPING SET or GROUP BY ROLLUP, for example). It transforms the original query into a revised query with UNION ALL of individual groupings.

FACT

Used in the context of the star transformation, FACT hint directs Oracle to treat the specified table parameter as a fact table.

FIRST_ROWS

FIRST_ROWS is a cost-based optimization approach that returns the first group of rows before the server collects all remaining requested information from the database and provides improved response time for applications. The hint is ignored by Oracle's optimizer in SELECT statements that contain a GROUP BY or FOR UPDATE clause, a set operator (like UNION, MINUS, and so on), an aggregate function, or a DISTINCT operator, because these conditions require the return of all rows prior to the return of the first row.

FULL

FULL tells Oracle's optimizer to access data by conducting a full table scan for the table (name or alias) you specify in parentheses.

HASH

HASH tells Oracle's optimizer to access data by conducting a hash scan for the table (name or alias) you specify in parentheses.

HASH_AJ

HASH_AJ tells Oracle's optimizer to turn a NOT IN subquery into a HASH ANTI JOIN to access the table (name or alias) you specify in parentheses.

HASH_SJ

HASH_SJ tells Oracle's optimizer to turn a correlated EXISTS subquery into a HASH SEMI JOIN to access the table (name or alias) you specify in parentheses.

INDEX

INDEX tells Oracle's optimizer to conduct an index scan on the table (name or alias) and index you specify in parentheses. This hint is ideal for domain and b-tree indexes, but Oracle recommends using the INDEX_COMBINE hint for bitmap indexes. In situations where a table has many indexes, Oracle optimizer chooses the least expensive index scan from the indexes listed after the hint.

INDEX_ASC

INDEX_ASC tells Oracle's optimizer to conduct an index scan on the table (name or alias) and index you specify in parentheses. Use this hint on SELECT statements which have index ranges to conduct an index range scan in ascending order of index value. SELECT statements that access more than one table always perform range scans in ascending order.

See also INDEX_DESC.

INDEX_COMBINE

INDEX_COMBINE tells Oracle's optimizer to use a bitmap access path to access the specified table. If an index is not specified, the optimizer chooses the least expensive combination of bitmap indexes. This hint is useful when more than one bitmap index is available, and distributed query optimization is your goal.

INDEX_DESC

INDEX_DESC tells Oracle's optimizer to conduct an index scan on the table (name or alias) and index you specify in parentheses. Use this hint on SELECT statements that have index ranges to conduct and index range scan in descending order of index value. SELECT statements that access more than one table are unaffected by this hint because they always perform range scans in ascending order.

See also INDEX_ASC.

INDEX_FFS

INDEX_FFS does a fast full index scan and is useful when distributed query optimization is your goal.

INDEX_JOIN

When working with a sufficiently small number of indexes containing all the columns required to resolve a query, the INDEX_JOIN hint directs DB Optimizer to use an index join as an access path.

INDEX_SS

INDEX_SS is useful with composite indexes where first column is not used in query but others are not used. It chooses an index skip scan for the specified table parameter:

- For statements using an index range scan, index entries are scanned in ascending order of their indexed values.
- For partitioned indexes, results are in ascending order within each partition.

INDEX_SS_ASC

Provides the same functionality as the INDEX_SS hint.

INDEX_SS_DSC

INDEX_SS chooses an index skip scan for the specified table parameter:

- For statements using an index range scan, index entries are scanned in descending order of their indexed values.
- For partitioned indexes, results are in descending order within each partition.

LEADING

The LEADING hint specifies the ordering of the set of tables used as the prefix in the execution plan.

MERGE

Used with either an in-line view that has a Group by or Distinct in it as a joined table, or with the use of IN subquery to "merge" the "view" into the body of the rest of the query.

MERGE_AJ

MERGE_AJ tells Oracle's optimizer to turn a NOT IN subquery into a MERGE ANTI JOIN to access the table (name or alias) you specify in parentheses.

MERGE_SJ

MERGE_SJ tells Oracle's optimizer to turn a correlated EXISTS subquery into a MERGE SEMI JOIN to access the table (name or alias) you specify in parentheses.

MODEL_MIN_ANALYSIS

MODEL_MIN_ANALYSIS is used with spreadsheet and model analysis to minimize compile time. It omits certain time-consuming compile-time optimizations of spreadsheet rules (detailed dependency graph analysis, for example).

MONITOR

Forces SQL monitoring of the statement. This hint is effective only if the STATISTICS_LEVEL initialization parameter is either set to ALL or TYPICAL and the CONTROL_MANAGEMENT_PACK_ACCESS parameter must be set to DIAGNOSTIC+TUNING.

NL_AJ

The NL_AJ hint directs DB Optimizer to turn a NOT IN subquery into a NESTED LOOP ANTI JOIN.

NL_SJ

The NL_AJ hint direct DB Optimizer r to turn an EXISTS subquery into a NESTED LOOP SEMI JOIN.

NO_EXPAND

NO_EXPAND tells Oracle's optimizer not to consider OR-expansion for a query that has OR conditions or INLISTS in its WHERE clause.

NO_FACT

Used in the context of the star transformation, the FACT hint directs DB Optimizer to NOT treat the table specified in tablespec as a fact table.

NO_INDEX

NO_INDEX tells Oracle's optimizer not to include the specified index or indexes in an index scan. When this hint is used without specifying any indexes, Oracle's optimizer does not scan any indexes. This hint applies to function-based, b-tree, bitmap, cluster, and domain indexes, and is useful if distributed query optimization is your goal.

NO_INDEX_FFS

NO_INDEX_FFS directs the Optimizer to exclude a fast full scan of the specified index(es).

NO_INDEX_SS

NO_INDEX_SS directs the Optimizer to exclude a skip scan of the specified index(es).

NO_MERGE

NO_MERGE keeps Oracle from merging views that can be merged, giving you more control over how views are accessed.

NO_MONITOR

Disables SQL monitoring of the statement.

NO_PARALLEL

Directs the Optimizer not to parallelize the specified table.

NO_PARALLEL_INDEX

Directs the Optimizer not to parallelize the specified index(es).

NO_PUSH_PRED

NO_PUSH_PRED prevents the pushing of a join predicate into the view.

NO_PUSH_SUBQ

NO_PUSH_SUBQ performs the opposite of the PUSH_SUBQ hint. It directs the Optimizer to not try evaluating the subquery first.

NO_PX_JOIN_FILTER

NO_PX_JOIN_FILTER disables parallel join bitmap filtering.

NO_QUERY_TRANSFORMATION

NO_QUERY_TRANSFORMATION directs the optimizer to not transform OR, in-lists, in-line views, and subqeueries.

NO_REWRITE

NO_REWRITE directs the Optimizer to not use a Materialized View, even if one is available.

NO_STAR_TRANSFORMATION

NO_STAR_TRANSFORMATION directs the Optimizer to no attempt a star transformation.

NO_UNNEST

NO_UNNEST disables unnesting of subqueries. Subqueries in the where clause are considered nested. A subquery could be evaluated several times for multiple results in the "parent". Unnesting evaluates the subquery once and merges results with the body of the "parent".

NO_USE_HASH

Negates the use of hash joins for the table specified.

NO_USE_MERGE

Negates the use of sort-merge joins for the table specified.

NO_USE_NL

Negates the use of nested-loop joins for the table specified.

NO_XML_QUERY_REWRITE

Use only if the query is using XML functionary.

NO_XMLINDEX_REWRITE

Use only if the query is using XML functionary.

NOAPPEND

NOAPPEND enables conventional INSERT by disabling parallel mode for the duration of the INSERT statement. (Conventional INSERT is the default in serial mode, and direct-path INSERT is the default in parallel mode).

NOCACHE

NOCACHE places the blocks retrieved for your specified table at the least recently used end of the LRU in the buffer cache for full table scans.

NOFACT

In the context of STAR transformation this table should NOT be considered a FACT table.

NOPARALLEL

NOPARALLEL overrides a PARALLEL specification in the table clause.

NOPARALLEL_INDEX

NOPARALLEL_INDEX overrides a PARALLEL attribute setting on an index to avoid a parallel index scan operation.

NOREWRITE

NOREWRITE disables query rewrite for a query block, overriding the setting of the parameter QUERY_REWRITE_ENABLED. This hint works on any query block of a request.

OPT_PARAM

Allows you to set an initialization parameter for the duration of the current query only. This hint is valid only for the following parameters: OPTIMIZER_DYNAMIC_SAMPLING, OPTIMIZER_INDEX_CACHING, OPTIMIZER_INDEX_COST_ADJ, OPTIMIZER_SECURE_VIEW_MERGING, and STAR_TRANSFORMATION_ENABLED.

ORDERED

ORDERED tells Oracle's optimizer to join tables in the order listed in the FROM clause. This hint gives you the ability to choose inner and outer tables based on your knowledge of the number of rows selected.

ORDERED_PREDICATES

ORDERED_PREDICATES forces the optimizer to preserve the order of predicate evaluation (except predicates used in index keys), as specified in the WHERE clause of SELECT statements.

PARALLEL

PARALLEL specifies the number of concurrent servers usable for a parallel operation. You can follow this hint with two values, separated by commas, that specify the degree of parallelism for the table and the manner in which the table is to split between the instances of a parallel server.

PARALLEL_INDEX

PARALLEL_INDEX specifies the number of concurrent servers usable to parallelize index range scans for partitioned indexes. You can follow this hint with two values, separated by commas, that specify the degree of parallelism for the table and the manner in which the table is to split between the instances of a parallel server.

PQ_DISTRIBUTE

Used in parallel join operations to indicate how inner and outer tables of the joins should be processed. The values of the distributions are HASH, BROADCAST, PARTITION, and NONE. For valid combinations of table distributions, refer to your Oracle SQL Optimizer documentation.

PUSH_PRED

PUSH_PRED forces the pushing of a join predicate into the view.

PUSH_SUBQ

PUSH_SUBQ evaluates nonmerged subqueries at their earliest possible place in the execution plan.

PX_JOIN_FILTER

PX_JOIN_FILTER forces parallel join bitmap filtering.

REWRITE

REWRITE is an important hint for use with views. You can use this hint with or without a view list. If you use REWRITE with a view list and the list contains an eligible materialized view, then Oracle uses the target view regardless of cost. In this instance, Oracle does not consider views outside of the list. If you do not specify a view list, Oracle searches for an eligible materialized view and always uses it, regardless of cost.

ROWID

ROWID tells Oracle’s optimizer to access data by conducting a table scan by row ID for the table (name or alias) you specify in parentheses.

RULE

RULE tells Oracle’s optimizer to optimize your SELECT statement according to rule-based goals and ignore other hints specified for your SELECT statement.

STAR

STAR makes Oracle’s optimizer use a star query plan, which places the largest table in the query last in the join order, and joins it with a nested loops join on a concatenated index. Use this hint when you are not using any other access or join method hints, when you have no less than three tables, and when the largest table’s concatenated index has no less than three columns.

STAR_TRANSFORMATION

STAR_TRANSFORMATION transforms a join into a query against a small fact table and subqueries targeting larger tables. This method is useful when employing fact tables to provide faster access to larger tables.

UNNEST

Subqueries in the where clause are considered nested. A subquery could be evaluated several times for multiple results in the "parent". Unnesting evaluates the subquery once and merges results with the body of the "parent".

USE_CONCAT

USE_CONCAT transforms combined OR conditions in the WHERE clause of your SELECT statement into a query using UNION ALL.

USE_HASH

USE_HASH tells Oracle’s optimizer to use a hash join to join a specified table to another row source using the specified table as the inner table.

USE_MERGE

USE_MERGE tells Oracle’s optimizer to use a sort-merge join to join a specified table to another row source using the specified table as the inner table. Oracle recommends using this with the ORDERED hint.

USE_NL

USE_NL tells Oracle’s optimizer to use nested loops to join a specified table to another row source using the specified table as the inner table. Oracle recommends using this with the ORDERED hint. This method is useful for quickly returning the first row of your query, and in many cases does so faster than a SORT-MERGE JOIN.

USE_NL_WITH_INDEX

Directive to use a nested-loop join with the specified table as the inner table using the index specified to satisfy at least one predicate.

Index