



Embarcadero® DB Optimizer™ 1.5.1: Using SQL Tuning

Copyright © 1994-2009 Embarcadero Technologies, Inc.

Embarcadero Technologies, Inc.
100 California Street, 12th Floor
San Francisco, CA 94111 U.S.A.
All rights reserved.

All brands and product names are trademarks or registered trademarks of their respective owners.

This software/documentation contains proprietary information of Embarcadero Technologies, Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

If this software/documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with Restricted Rights, as defined in FAR 552.227-14, Rights in Data-General, including Alternate III (June 1987).

Information in this document is subject to change without notice. Revisions may be issued to advise of such changes and additions. Embarcadero Technologies, Inc. does not warrant that this documentation is error-free.

Contents

Using Tuning	2
Overview	3
Understanding the Overview Tab	4
Understanding the Generated Cases Tab	5
Tuning SQL Statements	6
Create a New Tuning Job	7
Specify a Job Name	8
Specify a Data Source	8
Add SQL Statements	9
Run Tuning Job	10
Analyze Tuning Results	11
Modify Tuning Results	15
Using Oracle-Specific Features	16
Using the Analysis Tab	16
Using the Outlines Tab	17
Tuning SQL Statements in the System Global Area (SGA)	18
Additional Tuning Commands	18
View the Source Code of a Statement or Case	19
View Statement or Case Code in SQL Viewer	20
Open an Explain Plan for a Statement or Case	21
Work with Index Analysis Options	22
Executing a Session from the Command Line	22
Configuring Tuning	22
Set Roles and Permissions on Data Sources	23
Index Required Object Definitions	24
Set Tuning Job Editor Preferences	25
Set Generated Case Preferences	27
DBMS Hints	29
Oracle Hints	30
SQL Server Hints	35
DB2 Hints	36
Sybase Hints	37

Using Tuning

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

This guide contains the following topics:

[Overview](#)

[Tuning SQL Statements](#)

[Using Oracle-Specific Features](#)

[Additional Tuning Commands](#)

[Configuring Tuning](#)

[DBMS Hints](#)

Overview

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

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

- Hint Injection
- Index Analysis (Oracle only)
- Statistic Analysis (Oracle only)

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

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

This is a better method than relying on the native platform optimizer, as it can make the wrong decisions through incorrect or missing object statistics, skewed data, correlated predicates, or a bug in the optimizer.

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

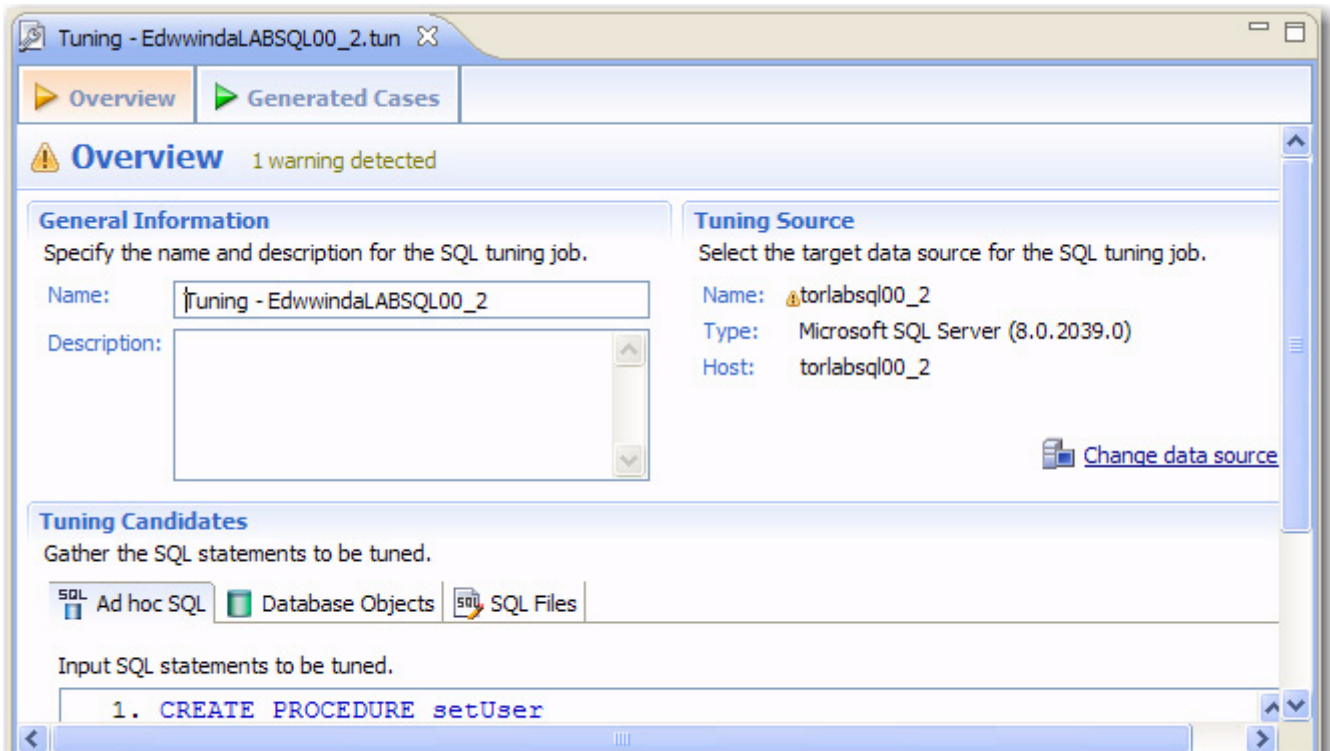
[Understanding the Overview Tab](#)

[Understanding the Generated Cases Tab](#)

Note: When using tuning on Oracle sources, two additional tabs appear: the Analysis and Outlines tabs. For more information on utilizing these extra features, see [Using the Analysis Tab](#) and [Using the Outlines Tab](#), respectively.

Understanding the Overview Tab

The **Overview** tab provides information about the SQL statements that will be tuned.



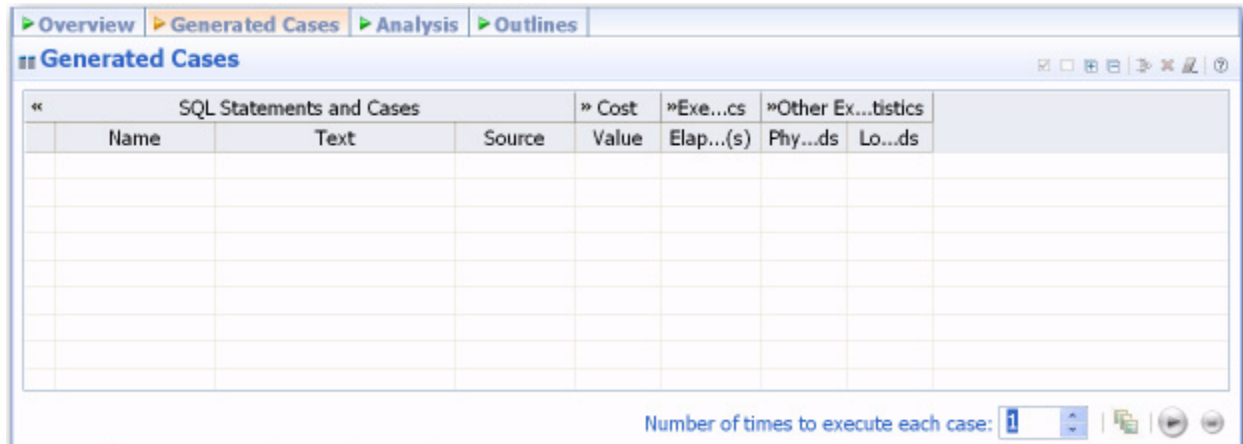
- The **Name** and **Description** fields enable you to enter the name of the tuning job, as well as a description.
- The **Tuning Source** box provides details about the data source from which the statements to be tuned reside.
- The **Tuning Candidates** section specify the statements to be tuned, and is split into three tabs: **Ad hoc SQL**, **Database Objects**, and **SQL Files**, depending on what source you want to tune the statement from, respectively.

Note: If you are tuning a specific schema on applicable platforms, this information appears below the **Host** section of the **Tuning Source** parameters. For example, **Schema: System** indicates that tuning examines **System** queries. This provides the ability to tune by user, where a table named **emp** might exist in different schemas, only **system.emp** will be examined in this case.

Understanding the Generated Cases Tab

The **Generated Cases** tab provides the list of statements that are analyzed by tuning, as well as the cases suggested by the execution process to improve them. Additional information may include statement **Name**, **Text**, **Source**, **Cost**, and **Elapsed Time** values, depending on the platform.

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



SQL Statements and Cases			» Cost	»Exe...cs	»Other Ex...tistics	
Name	Text	Source	Value	Elap...(s)	Phy...ds	Lo...ds

Number of times to execute each case: 1

Additionally, **Generated Cases** can be viewed by clicking **Generated Cases** on the **Overview** tab.

To generate cases:

Once you have added SQL code to the tuning job on the **Overview** tab, click **Generate Cases** at the bottom right-hand side of the screen. DB Optimizer automatically navigates to the **Generated Cases** tab and provides a list of statements that will be analyzed by the job.

Tuning SQL Statements

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

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

- Ad hoc statement tuning via manual entry, or cutting and pasting into the tuning window.
- Database object selection, by selecting stored packages from a list on the registered data source.
- SQL file selection, by choosing an SQL file saved on the system.
- Importing statements directly from profiling.

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

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

- 1 [Create a New Tuning Job](#)
- 2 [Specify a Job Name](#)
- 3 [Specify a Data Source](#)
- 4 [Add SQL Statements](#)
- 5 [Run Tuning Job](#)
- 6 [Analyze Tuning Results](#)
- 7 [Modify Tuning Results](#)

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

Create a New Tuning Job

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

Overview

General Information
Specify the name and description for the SQL tuning job.

Name:

Description:

Tuning Source
Select the target data source for the SQL tuning job.

Name:
Type:
Host:

[Select data source](#)

Tuning Candidates
Gather the SQL statements to be tuned.

☒ Ad hoc SQL ☐ Database Objects ☐ SQL Files

Input SQL statements to be tuned.

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

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

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

To create a new tuning job from profiling:

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

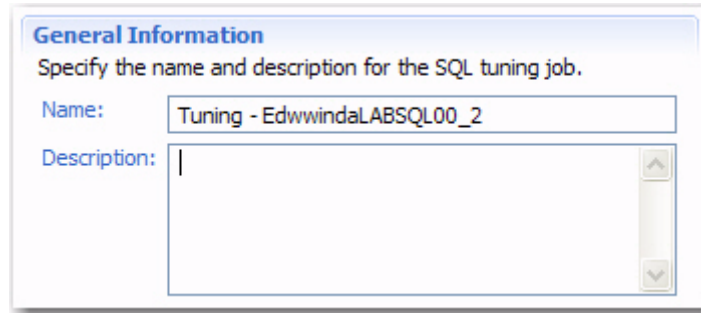
To open an existing tuning job:

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

Specify a Job Name

A job name identifies the Tuning job in the application and should be specified with this in mind.

Specify a meaningful name that clearly identifies the job in the views and dialogs of the working environment.



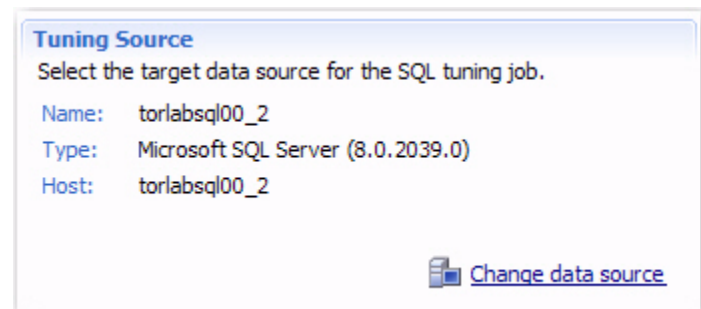
To name a job:

Type the name of the job in the **Name** field of tuning. Additionally, you can add an optional description of the job if required.

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

Specify a Data Source

The **Tuning Source** box identifies the data source where the SQL statements to be tuned reside. It is displayed by **Name**, **Type**, and **Host**.



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 a data source to a job:

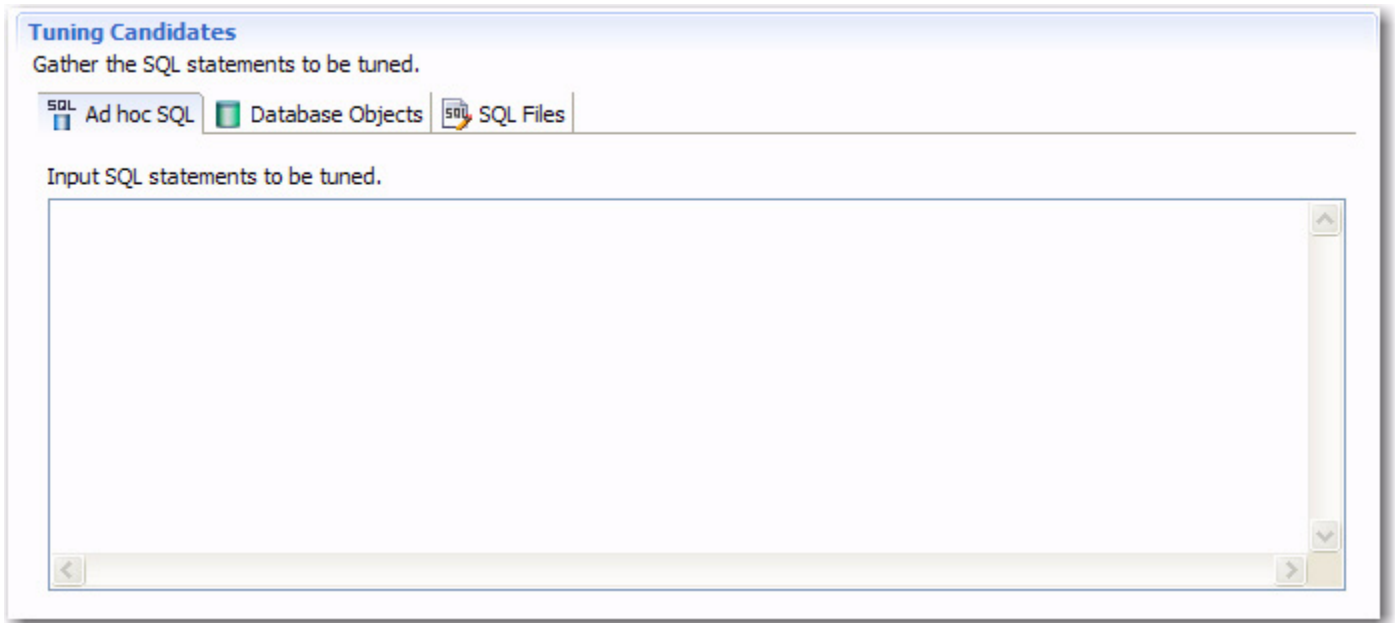
Click and drag a data source from **Data Source Explorer** to the **Tuning Source** box, or click **Select Data Source** and choose a data source from the dialog.

Note: You can change a data source selection by clicking **Change Data Source** from the appropriate box, or by dragging a different data source to the box from **Data Source Explorer**.

Add SQL Statements

Once you have created a name for the tuning job and indicated the tuning source, you need to add SQL statements to the job that are to be tuned. All standard DML statements can be tuned (SELECT, INSERT, DELETE, and UPDATE).

Statements are added to tuning via the **Tuning Candidates** box.



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

- The **Ad hoc SQL** tab enables tuning via manual entry, or cutting and pasting into the tuning window.
- The **Database Objects** tab enables you to select stored packages from a list on the registered data source.
- The **SQL Files** tab enables you to choose an SQL file saved on the system.

To add an ad hoc SQL statement:

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

To add a database object:

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

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

To add an SQL file:

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

Run Tuning Job

As you add SQL statements to the job, tuning-supported DML statements (SELECT, INSERT, DELETE, and UPDATE) are parsed from the statements and added to the **Generated Cases** tab in preparation for the tuning function to execute.

Each extracted statement is listed by **Name**, **Text**, and **Source**. Additionally, each statement will have **Cost**, **Elapsed Time** and **Other Execution Statistics** values that appropriate how efficiently they execute on the specified data source.

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.

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

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

Number of times to execute each case:    

To execute a tuning job:

Navigate to the **Generated Cases** tab and modify the number of times to execute each statement in the **Number of times to execute each case** field, as needed. Then click the execution icon in the lower-right side of the screen. The tuning job runs, exacting and analyzing each statement and providing values in the appropriate columns.

To explicitly generate a case for a specific statement:

Right-click in the **Name** field of a statement or transformation case and select **Generate Cases** from the context menu, or click the **Generated Cases** icon. The specified case is generated.

Analyze Tuning Results

Once you have executed a tuning job, the **Generated Cases** tab reflects tuning analysis of the specified statements.

Generated Cases

SQL Statements and Cases					Cost	Elapsed Time		
	Name	Text	Source	Index Analysis	Value	Value (s)	Result	Phy
☒	SQL Statement 1	select from	Ad Hoc	Click to optimize	3.0	7.825		
☒	SQL Statement 2	select from	Ad Hoc	OK				
☒	USE_CONCAT							
☒	NO_EXPAND							
☒	SQL Statement 3	select from	Ad Hoc	Unable to				
☒	SQL Statement 4	select from	Ad Hoc	OK	1.0			
☒	SQL Statement 5	select from	Ad Hoc	Unable to				
☒	SQL Statement 10	select from	Ad Hoc	Click to optimize	3.0			
☒	[Null c...rmation]				3.0			
☒	ALL_ROWS				3.0			
☒	FIRST_ROWS				3.0			
☒	FULL				3.0			
☒	INDEX_FFS				3.0			

Number of times to execute each case: 1

Run/Cancel Job controls

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

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

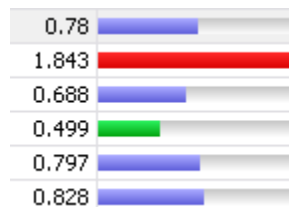
Icon	Description
	The case has not been executed. There are no errors or warnings and the case is ready to be executed.
	The case has been successfully executed.
	Transformations can be applied to this case.
	Execution for this case failed or was cancelled due to execution time exceeding 1.5 of original case time.
	The case contains invalid bind variables (types or values).
	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.

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

- The explain plan-based **Cost** field can be expended to display a graphical representation of the values for statements or cases. Similarly, after executing a statement or case, the **Elapsed Time** field can be 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.

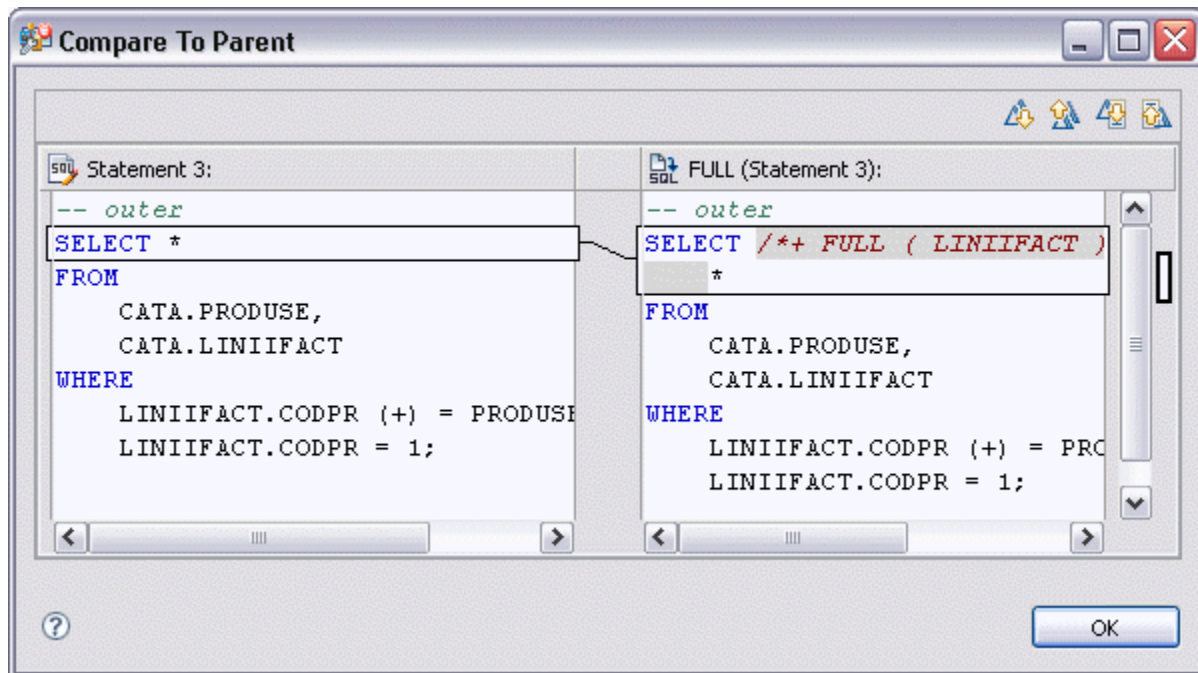
For information on specifying colors, and the improvement threshold and degradation threshold values used in these graphical representations.

Additionally, once results have been generated you can:

- [Compare Cases](#)
- [Filter and Remove Cases](#)
- [Create an Outline](#)

Compare Cases

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



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

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

To compare two cases:

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

Filter and Remove Cases

You filter or remove cases from the **Generated Cases** table via the **Filter** or **Delete** icons on the **Generated Cases** Toolbar.



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

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

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

To filter or remove cases from the **Generated Cases** table:

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

Create an Outline

If SQL is executed by an external application or If you cannot directly modify the SQL being executed but would like to improve the execution performance, you can create an outline. 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**.

Modify Tuning Results

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

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

- All SQL Optimizer hint-based variations that can be applied to a statement.
- A transformation-based case, if any of the eight common quick fixes can be applied to a SQL statement. This feature leverages the DB Optimizer **Code Quality Check** functionality. A transformation case, in turn, has 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				

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.

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

To apply a change:

- 1 Right-click on the **Name** field of the case that you want to use to modify the original statement and select **Apply Change**. The **Apply Change** dialog appears.
- 2 Choose **Execute** to apply the change to the statement automatically. Alternatively, select **Open in New SQL Editor** to open the modified statement in SQL Editor for manual changes or to save it to a file.

Using Oracle-Specific Features

[Using the Analysis Tab](#)

[Using the Outlines Tab](#)

[Tuning SQL Statements in the System Global Area \(SGA\)](#)

Using the Analysis Tab

The Analysis tab provides detailed information about statements and cases selected from the Generated Cases tab, after a tuning job has been executed.

The Analysis tab contains information about the statement or case, its full SQL code, and tabs displaying Index Analysis, Table Statistics, and Column Statistics and Histograms.

The screenshot displays the 'SQL Analysis' tab with the following components:

- Case Selection Table:**

Name	Cost	Elas... (s)
Statement 1	572.0	0.73
Custom 5		
Custom 6		
- SQL Preview:**

```
SELECT /*+ LEADING ( T31 ) */
cata.clientil.codcl,
MAX (cata.clientil.adresa),
MIN (cata.facturil.datafact)
```
- View table statistics:**

Object		Statistics						Monitoring		
Tabl...wner	Table Name	Statist... Status	Days ...aken	No of Rows	Blocks	Empt...cks	Sample Size	Monitoring	No I...rts	No ...
OCATA	CLIENTI1	Statistics OK	0	299997	1023	0	299997	YES		
OCATA	FACTURI1	Statistics OK	0	201998	649	0	201998	YES		
OCATA	LINIIFACT1	Statistics OK	0	201614	468	0	5374	YES		

Statement analysis is a manual process. In order to view and analyze statement statistics, select the tab (Index Analysis, Table Statistics, or Column Statistics and Histograms) and the statements whose statistics you want to generate. Then click the **Run** icon located in the bottom left-hand side of the tab.

Using the Outlines Tab

The **Outlines** tab provides detailed information about outlines created by the query during the statement execution process on the Generated Cases tab.

It provides information including the SQL statement name, if the outline is enabled or not, and the **Name**, **Category**, and **Hints** associated with the outline. Additionally, the **Drop** parameter specifies if it is dropped or not at execution time.

[illegible]

In order to view outlines, the session needs to have `USE_STORED_OUTLINES` set prior to execution. Outlines in tuning are created for the `DEFAULT` category, by default. Use the following commands to enable outlines with the default settings:

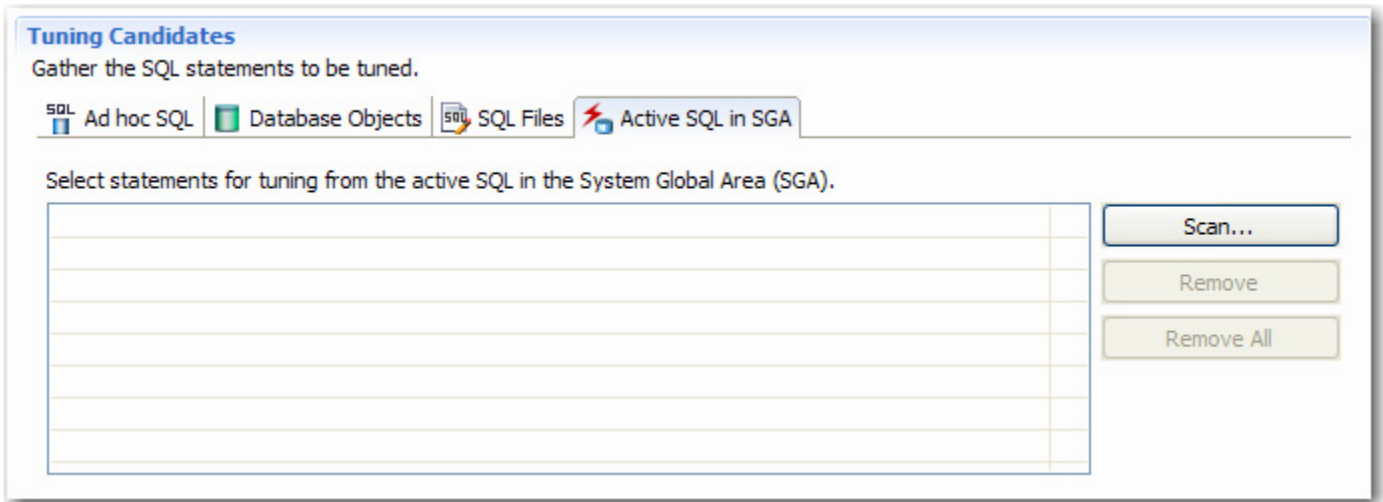
```
alter system set USE_STORED_OUTLINES=true;
alter system set USE_STORED_OUTLINES='DEFAULT';
alter session set USE_STORED_OUTLINES=true;
```

Additionally, in order for a session to USE_STORED_OUTLINES, the user requires the **create any outline** role. Use the following command to set up the proper permissions:

```
grant create any outline to [user];
```

Tuning SQL Statements in the System Global Area (SGA)

On Oracle platforms, SQL statements that reside in the SGA can also be tuned. When you create a tuning job and specify an Oracle source, an additional tab appears in the **Tuning Candidates** section of tuning, named **Active SQL in SGA**.



To add a statement active in the SGA:

- 1 Select the **Active SQL in SGA** tab and then click **Scan**. The **Scan SGA** wizard appears.
- 2 Set the filtering criteria for an SGA scan and then run the wizard. It returns all active statements on the Oracle source.
- 3 Choose the specific statements and add them to the tuning job.

Additional Tuning Commands

In addition to tuning, the interface provides additional commands and functionality that enables you to view source code, statements, and other information regarding the data source.

[View the Source Code of a Statement or Case](#)

[View Statement or Case Code in SQL Viewer](#)

[Open an Explain Plan for a Statement or Case](#)

[Work with Index Analysis Options](#)

[Executing a Session from the Command Line](#)

View the Source Code of a Statement or Case

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.

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.

View Statement or Case Code in SQL Viewer

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:

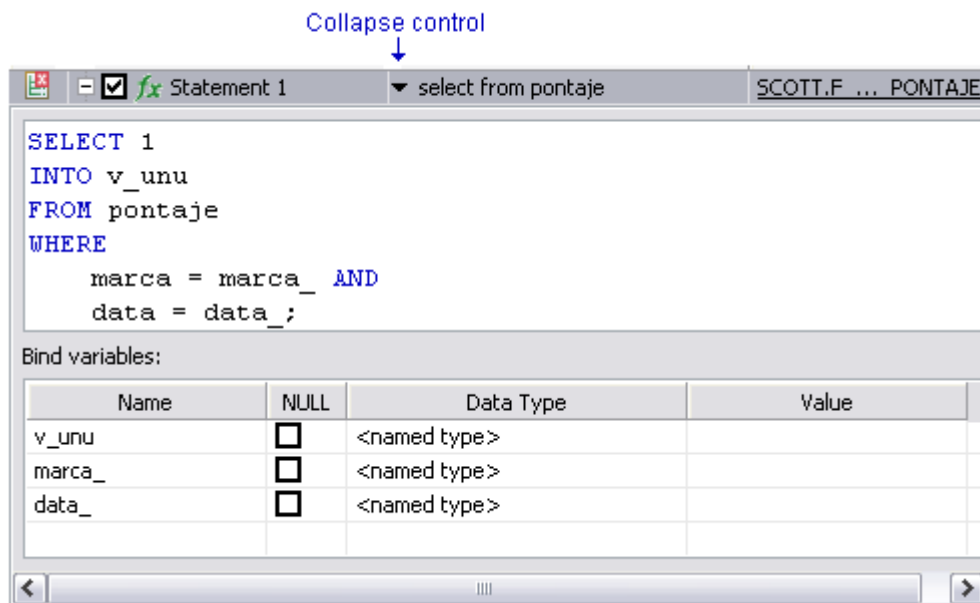
- View the entire SQL statement.
- Set bind variables. If the Tuning Status Indicator indicates a statement or case has invalid bind variables, you must set those variables before executing the statement or case.

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.



After setting bind variables, you can execute a case.

Note: Setting the bind variables in a parent statement sets the bind variables in all generated cases for that statement.

Open an Explain Plan for a Statement or Case

Any valid SQL statement added to the **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.

Operation	Plan Cost	Cost	Operation Cost	Result	Cardinality	Bytes	CPU Cost	IO Cost	Optimizer	Actual Statistics
SELECT STATEMENT	253.0	0.0			37	2183	102...107	235	ALL...WS	
PX COORDINATOR										
PX SEND - SYS.:TQ10004	253.0	0.0			37	2183	102...107	235		
HASH	253.0	0.0			37	2183	102...107	235		
PX RECEIVE	253.0	0.0			37	2183	102...107	235		
PX SEND - SYS.:TQ10003	253.0	0.0			37	2183	102...107	235		
HASH	253.0	1.0			37	2183	102...107	235		
HASH JOIN	252.0	1.0			7981	470879	95195351	235		
PX RECEIVE	67.0	0.0			37	703	24604129	63		
PX SEND -...:TQ10001	67.0	0.0			37	703	24604129	63		
PX BLOCK	67.0	0.0			37	703	24604129	63		
TABLE A...IENTI1	67.0	67.0			37	703	24604129	63	ANA...ED	
HASH JOIN	184.0	1.0			11706	468240	67387172	173		

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

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

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

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

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

Work with Index Analysis Options

As tuning candidates are added to a tuning job, the tuning feature automatically performs index analysis. If any columns referenced in the WHERE clause are not the first column of an index, tuning will recommend that you create an index on that column.

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

- 1 Navigate to the **Analysis** tab and select **Click to Optimize** in the field of a specified 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.

Executing a Session from the Command Line

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

```
dboptimizer.exe tune ds:ROM*L*ABORCL10G_1 sqlfile: C:\dboptimizer\workspace\test.sql
tofile:c:\testtune.tun overwrite
```

In the above command, the user has specified ROM*L*ABORCL10G_1 as the data source, and indicates a tuning session using the **test.sql** script. The **tofile** variable specifies the directory and name of the file to which the tuning job will be saved. The **overwrite** command indicates that a pre-existing save file will be overwritten by the new output.

Configuring Tuning

This section contains information on configuring tuning. It provides information on setting up your data sources to work with tuning functionality, as well as information regarding preferences within the application for the customization of various features and functionality.

[Set Roles and Permissions on Data Sources](#)

[Index Required Object Definitions](#)

[Set Tuning Job Editor Preferences](#)

Set Roles and Permissions on Data Sources

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

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

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

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

Index Required Object Definitions

When connecting to a data source, the application caches a subset of the object definitions on the data source. Tuning 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 tuning 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 tuning is indexing required objects when connecting to a data source.

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

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

A **Preferences** dialog opens.

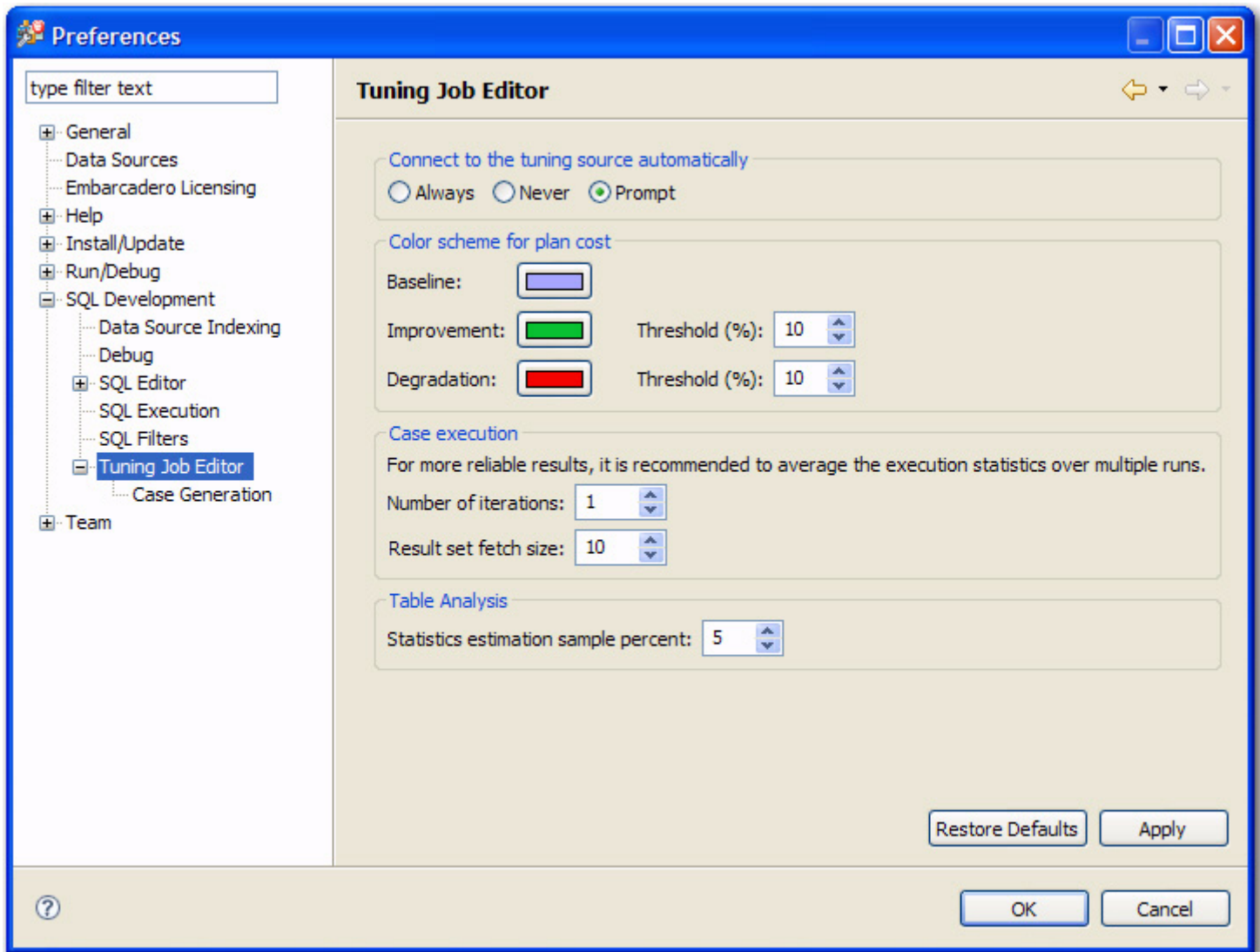
- 2 In the left-hand pane expand the **SQL Development** item and then click **Cache Configuration**.
- 3 Select the check boxes associated with the following list of minimally-required object definitions:

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

- 4 Click **OK**.

Set Tuning Job Editor Preferences

Tuning job editor preferences let you control certain aspects of the appearance of items in the tuning job editor as well as default behaviors.

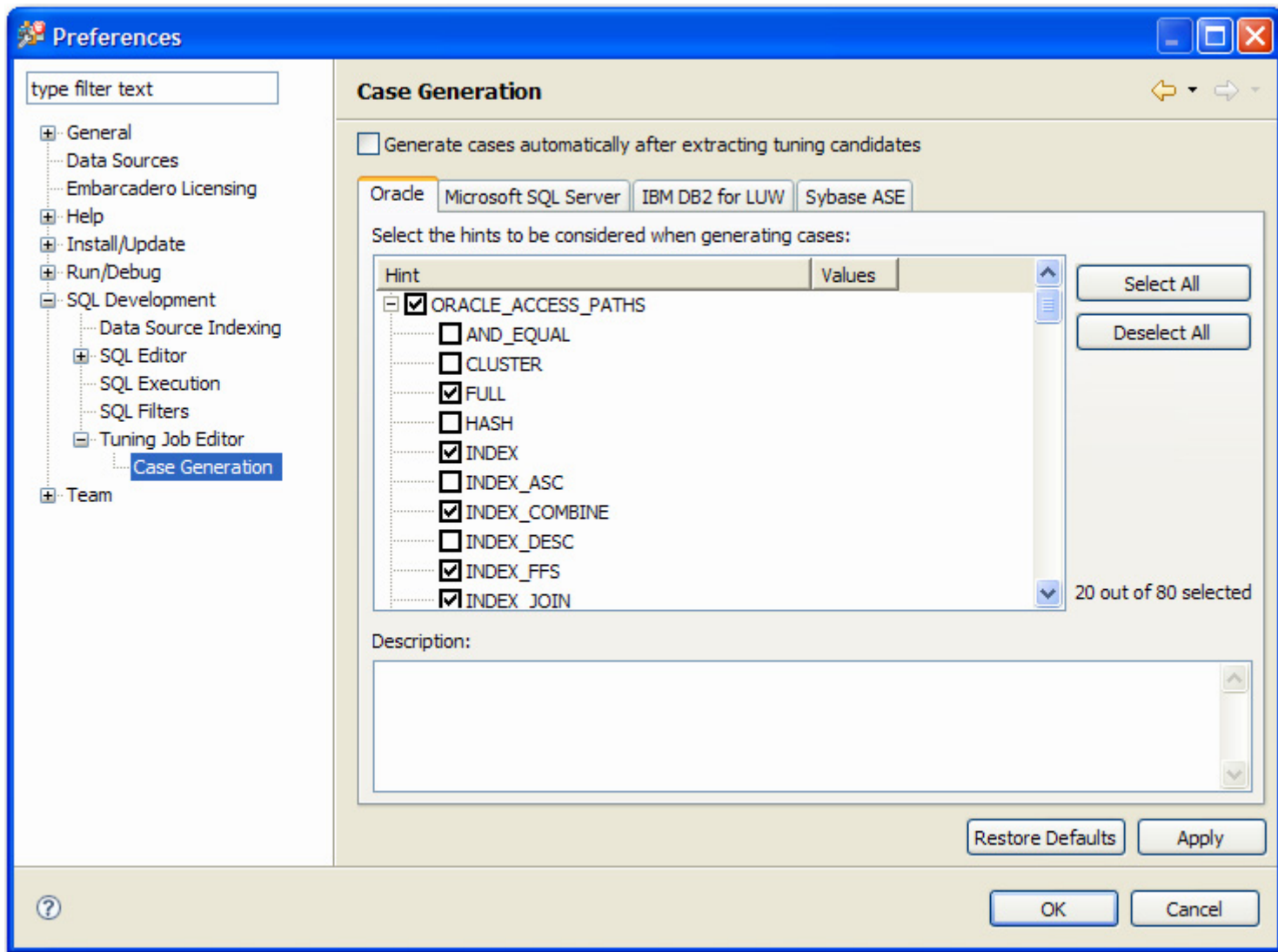


Select **Window > Preferences > SQL Development > Tuning Job Editor**

Option	Description
Connect to the tuning source automatically	When you open a tuning perspective, it automatically opens the last saved tuning jobs that were open when you closed the application. This option lets you specify whether, in addition, you want to automatically connect to the data sources associated with these tuning jobs. If you typically review existing tuning job archives rather than run new tuning jobs, you may wish to explicitly connect to a data source rather than connect automatically. The options are: Always - automatically connects to data sources associated with tuning jobs that were open last time you shut down tuning. Never - automatically opens tuning job archives that were open last time you shut down the application but does not automatically connect to the associated data sources. Prompt - prompts you to connect to data sources associated with tuning jobs that were open last time you shut down the application.
Color scheme for plan cost	In the graphical representations of explain plan cost and elapsed time, tuning uses a color scheme to highlight differences among generated cases. Values for the original statement are treated as a baseline, and values for individual cases that are within a specified threshold range of the baseline value are represented with a Baseline color. For cases whose values are outside the threshold range, Improvement and Degradation colors are used to represent values in those cases.
Case execution	Lets you dictate how execution statistics are gathered.
Table analysis	Lets you specify an estimation sample percentage to be used with the Analyze Tables function.

Set Generated Case Preferences

Additionally, the Generated Case preference page lets you enable or disable the automatic generation of SQL Optimizer hint-based cases of SQL statements added to a tuning job. It also lets you indicate which specific hint types are generated when the feature is enabled.



Select **Window > Preferences > SQL Development > Tuning Job Editor > Case Generation**

Use the **Generate cases automatically after extracting tuning candidates** control to enable or disable automatic generation of hint-based cases, and then select the check boxes to specify the hint-based cases that are generated for

a statement added to a tuning job.

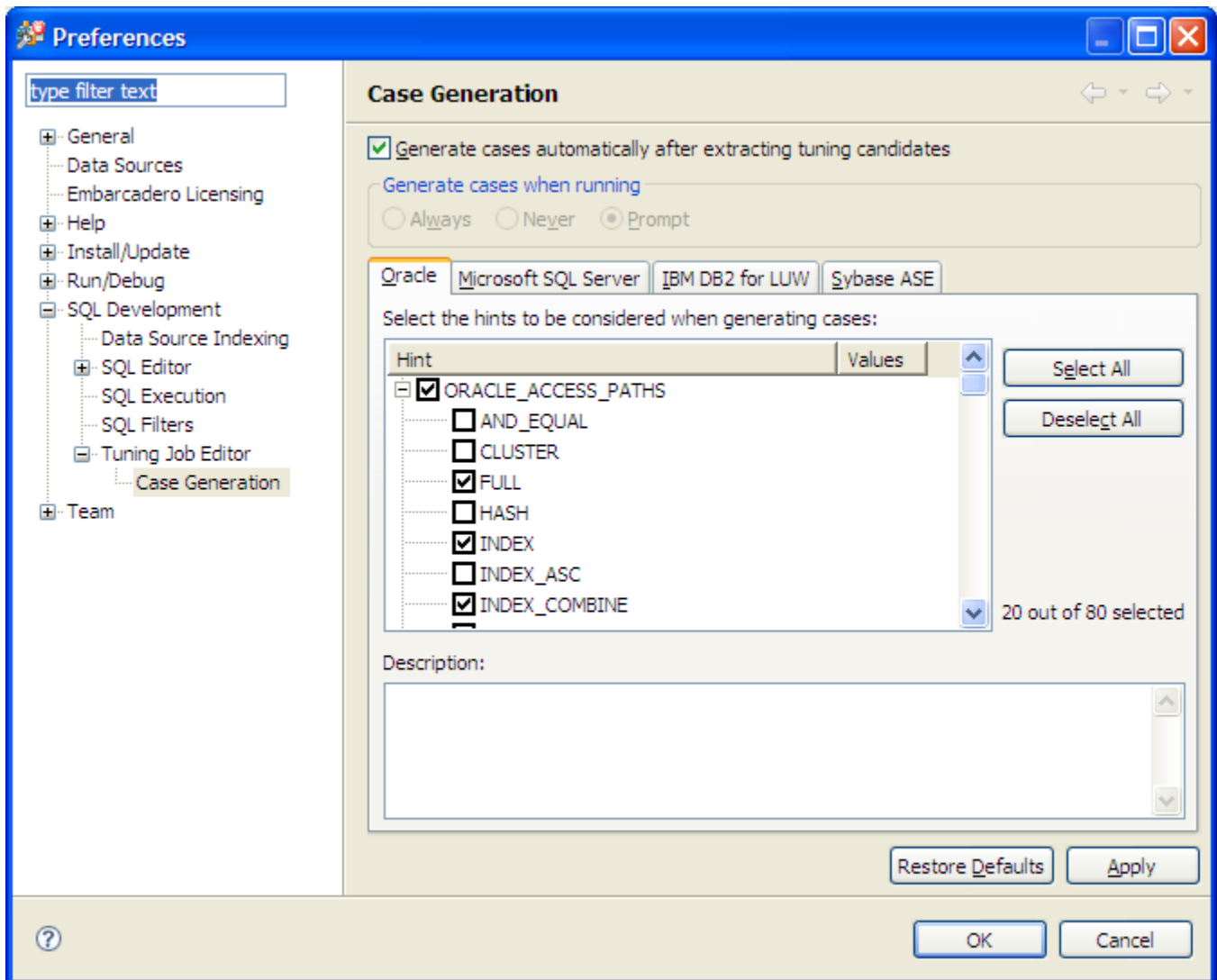
About Statement Records

Column or column set	Description
SQL Statements and Cases	Identifiers for the generated statement or case: 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. 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.
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.
Index Analysis	Tuning automatically detects indexes that require optimization and offers you the option to automatically optimize the index. For more information, see Work 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. This column set can be expanded to display a graphical representation of the elapsed time to facilitate comparisons among cases.
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.

DBMS Hints

Users can provide hints to a specified platform in order to instruct data source optimizer on the best way to execute SQL statements. Tuning automatically generates cases using these hints.

Hints can be enabled or disabled when cases are being generated by tuning on the **Window > Preferences > Tuning Job Editor > Case Generation** panel. Choose a tab as it pertains to the platform you want to modify and use the check boxes to select and de-select the hints you want to enable or disable, respectively.



The following platform hints are packaged in tuning to provide optimal efficiency when executing jobs:

[Oracle Hints](#)

[SQL Server Hints](#)

[DB2 Hints](#)

[Sybase Hints](#)

Oracle Hints

The following table highlights Oracle hints based on Oracle hints optimization:

Category	Hint	Available For	Notes
ACC PATH	AND_EQUAL	<code>/*+ CLUSTER (tablespec) */</code>	-
ACC PATH	CLUSTER	<code>/*+ FULL (tablespec) */</code>	Use on Clustered Tables only
ACC PATH	FULL	<code>/*+ HASH (tablespec) */</code>	Forces a table scan even if there are indexes.
ACC PATH	HASH	<code>/*+ INDEX (tablespec [TAL: indexspec]) */</code>	Only to tables stored in a table cluster.
ACC PATH	INDEX	<code>/*+ INDEX_ASC (tablespec [TAL: indexspec]) */</code>	If no indexspec is supplied, the optimizer will try to scan with each avail index.
ACC PATH	INDEX_ASC	<code>/*+ INDEX_COMBINE (tablespec [indexspec [TAL: indexspec]...]) */</code>	Essentially the same as INDEX.
ACC PATH	INDEX_COMBINE	<code>/*+ INDEX_DESC (tablespec [indexspec [TAL: indexspec]...]) */</code>	Forces the optimizer to try multiple boolean combinations of indexes.
ACC PATH	INDEX_DESC	<code>/*+ INDEX_DESC (tablespec [indexspec [TAL: indexspec]...]) */</code>	Essentially the same as INDEX.
ACC PATH	INDEX_FFS	<code>/*+ INDEX_FFS (tablespec [indexspec [TAL: indexspec]...]) */</code>	Forces an index scan using specified index(es).
ACC PATH	INDEX_JOIN	<code>/*+ INDEX_JOIN (tablespec [indexspec [TAL: indexspec]...]) */</code>	Indexes used should be based on columns in the where clause.
ACC PATH	INDEX_SS	<code>/*+ INDEX_SS (tablespec [indexspec [TAL: indexspec]...]) */</code>	Useful with composite indexes where the first column is not used in the query, but others are.
ACC PATH	INDEX_SS_ASC	<code>/*+ INDEX_SS_ASC (tablespec [indexspec [TAL: indexspec]...]) */</code>	Essentially the same as INDEX_SS.
ACC PATH	INDEX_SS_DESC	<code>/*+ INDEX_SS_DESC (tablespec [indexspec [TAL: indexspec]...]) */</code>	Essentially the same as INDEX_SS.
ACC PATH	NO_INDEX	<code>/*+ NO_INDEX (tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optimizer not to use specified index(es).
ACC PATH	NO_INDEX_FFS	<code>/*+ NO_INDEX_FFS ([tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optmizer to exclude a fast full scan of the specified index(es).
ACC PATH	NO_INDEX_SS	<code>/*+ NO_INDEX_SS (tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optmizer to exclude a skip scan of the specified index(es).
ACC PATH	ROWID	-	-
JOIN OP	HASH_AJ	-	-
JOIN OP	HASH_SJ	-	-
JOIN OP	MERGE_AJ	-	-
JOIN OP	MERGE_SJ	-	-

Category	Hint	Available For	Notes
JOIN OP	NL_AJ	-	-
JOIN OP	NL_SJ	-	-
JOIN OP	NO_USE_HASH	/*+ NO_USE_HASH (tablespec [TAL: tablespec]...) */	Negates the use of hash joins for the table specified.
JOIN OP	NO_USE_MERGE	/*+ NO_USE_MERGE (tablespec [TAL: tablespec]...) */	Negates the use of sort-merge joins for the table specified.
JOIN OP	NO_USE_NL	/*+ NO_USE_NL (tablespec [TAL: tablespec]...) */	Negates the use of nested-loop joins for the table specified.
JOIN OP	USE_HASH	/*+ USE_HASH (tablespec [TAL: tablespec]...) */	Directive to join each table specified using a hash join.
JOIN OP	USE_MERGE	/*+ NO_USE_MERGE (tablespec [TAL: tablespec]...) */	Directive to join each table specified using a sort--merge join.
JOIN OP	USE_NL	/*+ NO_USE_NL (tablespec [TAL: tablespec]...) */	Directive to use a nested-loop join with the specified tables as the inner table.
JOIN OP	USE_NL_WITH_INDEX	/*+ USE_NL_WITH_INDEX (tablespec [indexspec [TAL: indexspec]...]) */	Directive to use a nested-loop join with the specified table as the inner table using the index specified to satisfy at least one predicate.
JOIN ORDER	LEADING	/*+ LEADING (tablespec) */	Directive to join the tables in the order specified.
JOIN ORDER	ORDERED	/*+ ORDERED */	Directive to join tables in the order found in the FROM clause.
JOIN ORDER	STAR	-	-
OPT APPROACH	ALL_ROWS	/*+ ALL_ROWS */	Indicates the goal is overall throughput.
OPT APPROACH	CHOOSE	-	-
OPT APPROACH	FIRST_ROWS	/*+ FIRST_ROWS (integer) */	The goal is to retrieve the first row(s) as fast as possible.
OPT APPROACH	RULE	/*+ RULE */	Used to disable the COST based optimizer.
OTHER	CACHE	/*+ CACHE (tablespec) */	Should be used with the FULL hint. Places data in the most-recently used area of the buffer cache.
OTHER	APPEND	/*+ APPEND */	Directs the optimizer to INSERT data at the end of the existing table data using direct path I/O.

Category	Hint	Available For	Notes
OTHER	CURSOR_SHARING_EXACT	<code>/*+ CURSOR_SHARING_EXACT */</code>	Directs the Optimizer to ignore previously parsed SQL that matches, but uses bind variables. Forces the SQL to be parsed unless an exact match is found.
OTHER	DRIVING_SITE	<code>/*+ DRIVING_SITE (tablespec) */</code>	Used when data is joined remotely via DBLink. Normally data at the remote site is returned to the local and joined. This hint directs the optimizer to send the local data to the remote site for resolution of the join.
OTHER	DYNAMIC_SAMPLING	<code>/*+ DYNAMIC_SAMPLING ([TAL: tablespec] integer) */</code>	Only used in simple SELECT statements with a single table to approximate cardinality if there are no existing statistics on the table.
OTHER	MODEL_MIN_ANALYSIS	<code>/*+ MODEL_MIN_ANALYSIS */</code>	Used with spreadsheet and model analysis to minimize compile time.
OTHER	NO_PUSH_PRED	<code>/*+ NO_PUSH_PRED [TAL: (tablespec)] */</code>	Opposite of PUSH_PRED, it directs the Optimizer not to try to push the predicate into the view.
OTHER	NO_PUSH_SUBQ	<code>/*+ NO_PUSH_SUBQ] */</code>	Opposite of PUSH_SUBQ, it directs the Optimizer not to try and evaluate the subquery first.
OTHER	NO_UNNEST	<code>/*+ NO_UNNEST */</code>	Subqueries in the WHERE clause are considered nested. A subquery can be evaluated several times for multiple results in the “parent”. Unnesting evaluates the subquery once and merges the results with the body of the “parent”. This hint directs the Optimizer NOT to unnest.
OTHER	NOAPPEND	<code>/*+ NOAPPEND */</code>	Directs the Optimizer to utilize existing space in a table and negates parallel processing.
OTHER	NOCACHE	<code>/*+ NOCACHE (tablespec) */</code>	Should be used with the FULL hint. Places data in the least-recently used area of the buffer cache.
OTHER	OPT_PARAM	-	-
OTHER	ORDERED_PREDICATES	-	-
OTHER	PUSH_PRED	<code>/*+ PUSH_PRED [TAL: (tablespec)] */</code>	Used when one of the tables in a join is an in-line view. Forces the predicate used to join the table and the view into the view.
OTHER	PUSH_SUBQ	<code>/*+ PUSH_SUBQ *</code>	Used with an EXISTS or IN subselect to force evaluation of the subquery rather than the default behavior of the last.

Category	Hint	Available For	Notes
OTHER	UNNEST	<code>/*+ UNNEST */</code>	Subqueries in the where clause are considered nested. A subquery could be evaluated several times for multiple results in the “parent”. Unnesting evaluates the subquery once and merges results with the body of the “parent”.
PARALLEL	NO_PARALLEL	<code>/*+ NO_PARALLEL (tablespec) */</code>	Directs the Optimizer not to parallel the specified table.
PARALLEL	NO_PARALLEL_INDEX	<code>/*+ NO_PARALLEL_INDEX (tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optimizer not to parallel the specified index(es).
PARALLEL	NO_PX_JOIN_FILTER	<code>/*+ NO_PX_JOIN_FILTER (tablespec) */</code>	Directs the Optimizer not to try and join bitmap indexes in parallel.
PARALLEL	NOPARALLEL	<code>/*+ NOPARALLEL (tablespec) */</code>	Directs the Optimizer not to parallel the specified table.
PARALLEL	NOPARALLEL_INDEX	<code>/*+ NOPARALLEL_INDEX (tablespec [indexspec [TAL: indexspec]...]) */</code>	Directs the Optimizer not to parallel the specified index(es).
PARALLEL	PARALLEL	<code>/*+ PARALLEL (tablespec [integer TAL:DEFAULT]) */</code>	Number specifies degree of parallelism (how many processes).
PARALLEL	PARALLEL_INDEX	<code>/*+ PARALLEL_INDEX (tablespec [indexspec [TAL: indexspec]...] integer DEFAULT) */</code>	Number specifies degree of parallelism (how many processes).
PARALLEL	PQ_DISTRIBUTE	<code>/*+ PQ_DISTRIBUTE(tablespec outer_distribution inner_distribution) */</code>	Used in parallel join operations to indicate how inner and outer tables of the joins should be processed. The values of the distributions are HASH, BROADCAST, PARTITION, and NONE. Only six combinations table distributions are valid.
PARALLEL	PX_JOIN_FILTER	<code>/*+ PX_JOIN_FILTER (tablespec) */</code>	Directs the Optimizer to try and join bitmap indexes in parallel.
QUERY TRANS	EXPAND_GSET_TO_UNION	<code>/*+ EXPAND_GSET_TO_UNION */</code>	Performs transformations on queries that have GROUP BY into Unions.
PARALLEL	FACT	<code>/*+ FACT (tablespec) */</code>	In the context of STAR transformation, this table should be considered a FACT table (as opposed to a DIMENSION).
PARALLEL	MERGE	<code>/*+ MERGE ([view tablespec]) */</code>	Use with either an in-line view that has a Group by or Distinct in it as a joined table, or with the use of IN subquery to “merge” the “view” into that body of the rest of the query.

Category	Hint	Available For	Notes
PARALLEL	NO_EXPAND	<code>/*+ NO_EXPAND */</code>	Used when OR condition (including IN lists) is present in the predicate to not consider transformation to compound query.
PARALLEL	NO_FACT	<code>/*+ NO_FACT (tablespec) */</code>	In the context of STAR transformation this table should not be considered a FACT table.
PARALLEL	NO_MERGE	<code>/*+ NO_MERGE [([view TAL:tablespec]) */</code>	Directs the Optimizer not to “merge” the view into the query.
PARALLEL	NO_QUERY_TRANSFORMATION	<code>/*+ NO_QUERY_TRANSFORMATION */</code>	Directs the Optimizer not to transform OR, in-lists, in-line views, and subqueries. Try it whenever any of these conditions are present.
PARALLEL	NO_REWRITE	<code>/*+ NO_REWRITE */</code>	Directs the Optimizer not to use a Materialized View, even if one is available.
PARALLEL	NO_STAR_TRANSFORMATION	<code>/*+ NO_STAR_TRANSFORMATION */</code>	Directs the Optimizer not to try a Star Transformation.
PARALLEL	NO_XML_QUERY_REWRITE	<code>/*+ NO_XML_QUERY_REWRITE */</code>	Use only if the query is using XML functionality.
PARALLEL	NO_XMLINDEX_REWRITE	<code>/*+ NO_XMLINDEX_REWRITE */</code>	Use only if the query is using XML functionality.
PARALLEL	NOFACT	<code>/*+ NOFACT (tablespec) */</code>	In the context of STAR transformation, this table should not be considered a FACT table.
PARALLEL	NOREWRITE	<code>/*+ NOREWRITE</code>	Directs the Optimizer not to use a Materialized View, even if one is available.
PARALLEL	REWRITE	<code>/*+ REWRITE [(view [TAL: view]...)] */</code>	Directs the Optimizer to use a Materialized View instead of the underlying tables. Specify REWRITE without additional parameters. Oracle will determine if it can use a Materialized View or not.
PARALLEL	STAR_TRANSFORMATION	<code>/*+ STAR_TRANSFORMATION */</code>	Directs the Optimizer to try Star Transformation. Only try with a 3 table or more join.
PARALLEL	USE_CONCAT	<code>/*+ USE_CONCAT */</code>	Used when the OR condition (including IN lists) is present in the predicate to transform the query into a compound UNION ALL.

Category	Hint	Available For	Notes
REAL TIME	MONITOR	/*+ MONITOR */	Effective only if STATISTICS_LEVEL initialization parameter is either set to ALL or TYPICAL and CONTROL_MANAGEMENT_PACK_ACCESS is set to DIAGNOSTIC+TUNING. Turns on features of the Oracle Database Tuning Pack.
REAL TIME	NO_MONITOR	/*+ NO_MONITOR */	See MONITOR hint.

SQL Server Hints

The following table highlights SQL hints based on MS SQL Server hints optimization:

Category	Hint	Available For	Notes
JOIN	LOOP	SELECT/UPDATE/DELETE	Not applicable for RIGHT OUTER or FULL joins.
JOIN	HASH	SELECT/UPDATE/DELETE	-
JOIN	MERGE	SELECT/UPDATE/DELETE	-
JOIN	REMOTE	SELECT/UPDATE/DELETE	Only for INNER JOINS. Not applicable with COLLATE
		SELECT/UPDATE/DELETE	-
QUERY	RECOMPILE	SELECT/UPDATE/DELETE	-
QUERY	FORCE ORDER	SELECT/UPDATE/DELETE	-
QUERY	ROBUST PLAN	SELECT/UPDATE/DELETE	-
QUERY	KEEP PLAN	SELECT/UPDATE/DELETE	-
QUERY	KEEPFIXED PLAN	SELECT/UPDATE/DELETE	-
QUERY	EXPAND VIEWS	DML Statements	Only for statement containing views.
QUERY	HASH GROUP	SELECT	Only when GROUP BY, COMPUTE and DISTINCT clauses are used.
QUERY	ORDER GROUP	SELECT/UPDATE/DELETE	Only when GROUP BY, COMPUTE and DISTINCT clauses are used.
QUERY	MERGE UNION	SELECT	Only for statements chained using UNION
QUERY	HASH UNION	SELECT	Only for statements chained using UNION
QUERY	CONCAT UNION	SELECT	Only for statements chained using UNION
QUERY	LOOP JOIN	SELECT/UPDATE/DELETE	-
QUERY	MERGE JOIN	SELECT/UPDATE/DELETE	-
QUERY	HASH JOIN	SELECT/UPDATE/DELETE	-
TABLE	INDEX()	DML Statements	Only for tables and views with indexes.
TABLE	KEEPIDENTITY	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.

Category	Hint	Available For	Notes
TABLE	KEEPDEFAULTS	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	HOLDLOCK	DML Statements	Not applicable for SELECT statements using FOR BROWSE clause.
TABLE	IGNORE_CONSTRAINTS	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	IGNORE_TRIGGERS	INSERT	Only for INSERT statements using OPENROWSET clause with BULK option.
TABLE	NOLOCK	SELECT/UPDATE/COMPLETE	Not applicable for the target table in UPDATE/DELETE statements.
TABLE	NOWAIT	DML Statements	-
TABLE	PAGLOCK	DML Statements	-
TABLE	READCOMMITTED	DML Statements	-
TABLE	READCOMMITTEDLOCK	SELECT/UPDATE/COMPLETE	-
TABLE	READPAST	SELECT/UPDATE/COMPLETE	Not applicable for the target table in UPDATE/DELETE statements.
TABLE	READUNCOMMITTED	SELECT/UPDATE/COMPLETE	Not applicable for the target table in UPDATE/DELETE statements.
TABLE	REPEATABLE_READ	DML Statements	-
TABLE	ROWLOCK	DML Statements	-
TABLE	SERIALIZABLE	DML Statements	Not applicable for SELECT statements using FOR BROWSE clause.
TABLE	TABLOCK	DML Statements	-
TABLE	TABLOCKX	DML Statements	-
TABLE	UPDLOCK	DML Statements	-
TABLE	XLOCK	DML Statements	-
TABLE	FASTFIRSTROW	DML Statements	-

DB2 Hints

The following table highlights SQL hints based on IBM DB2 hints optimization:

Category	Hint	Notes
Command	SET OPTIMIZATION LEVEL	For top-level SELECT statements only
Clause	optimize for <n> rows	For top-level SELECT statements only
Clause	fetch first <n> rows only	For SELECT statements only

Sybase Hints

The following table highlights SQL hints based on Sybase hints optimization:

Category	Hint	Notes
Logical	distinct	No explicit implementation
Logical	group	No explicit implementation
Logical	g_join	Noexplicit implementation
Logical	nl_g_join	Not applicable for: statements with chained queries; select statements with group by clause and having clause or group by clause and order by clause
Logical	m_g_join	Not applicable for: statements with chained queries; select statements with group by clause and having clause or group by clause and order by clause
Logical	join	Noexplicit implementation
Logical	nl_join	Not applicable for: select statements with group by clause and having clause or group by clause and order by clause
Logical	m_join	Not applicable for: select statements with group by clause and having clause or group by clause and order by clause
Logical	h_join	Not applicable for: select statements with group by clause and having clause or group by clause and order by clause
Logical	union	Noexplicit implementation
	scan	Noexplicit implementation
Logical	scalar_agg	Only used in combination with other operators. It does not change the execution plan itself.
Logical	sequence	Is a keyword that will be used in the implementation of scalar_agg operator.
Logical	hints	We don't support a combination of hints
Logical	prop	Uses a set of pre-defined values.
Logical	table	Used only in combination with other operators, when referring tables from subqueries
Logical	work_t	This operator is applicable only together with store operator
Logical	in	Used only in combination with other operators, when referring tables from subqueries
Logical	subq	Used only in combination with other operators, when referring tables from subqueries
Physical	distinct_sorted	Only for SELECT statements containing DISTINCT, and only for tables
Physical	distinct_sorting	Only for SELECT statements containing DISTINCT, and only for tables

Category	Hint	Notes
Physical	distinct_hashing	Only for SELECT statements containing DISTINCT, and only for tables
Physical	group_sorted	Only for SELECT statements (not working for views) with no having and no order by clause.
Physical	group_hashing	Only for SELECT statements (not working for views) with no having and no order by clause.
Physical	group_inserting	Not implemented
Physical	append_union_all	Not applicable for: UNION chained clauses, nested sub-selects in a from clause, if a group by clause is present or if scalar aggregation is present
Physical	merge_union_all	Not applicable for: UNION ALL chained clauses, nested sub-selects in a from clause, or if a group by clause is present.
Physical	merge_union_distinct	Not applicable for: UNION ALL chained clauses, nested sub-selects in a from clause, or if a group by clause is present.
Physical	hash_union_distinct	Not applicable for: UNION ALL chained clauses, nested sub-selects in a from clause, if a group by clause is present, or if scalar aggregation is present.
Physical	i_scan	Applied to all table references in the from clause of the main select and of the sub select statements except: 1. statement has sub-selects. 2. table references has no indexes.
Physical	t_scan	Applied to all the table references in the from clause of the main select and of the sub select statements except: On Sybase 12.5 not applied for tables in the main query if: 1. statement has chained queries. 2. Sub queries have group by and having clauses; and not applied to the tables in sub selects if: 1. has select statements in from clause of the main select. 2. sub queries have group by and having clauses. 3. statement has select statements in select clause. 4. statement has parent statement and insert statement; on Sybase 15 not applied for tables in sub selects if: 1. has select statements in from clause of the main select. 2. statement has chained queries.
Physical	m_scan	Applied for all tables if in the where clause there is a condition like: table1.indexedColumn1 condition body OR table1.indexedColumn2 condition body; Not applied if the LIKE operator is used. For columns that belong to a primary key only the first column is considered.
Physical	store	-
Physical	store_index	-
Physical	sort	-
Physical	xchg	-

C

- caching, transformations requirements 24
- cases, generated
 - opening in context 19

E

- explain plans
 - opening from tuning job 21

H

- hints
 - opening in context 19

I

- index analysis, SQL Tuner 22

P

- permissions, SQL Tuner 23

R

- roles, SQL Tuner 23

S

- SQL
 - tuning 6

T

- transformations
 - caching requirement 24
- tuning jobs
 - editor preferences 25
 - index analysis 22
 - introduced 6
 - opening explain plans from 21
 - roles/permissions required 23
 - understanding generated statements 28