



Product Documentation

DB Optimizer™

New Features Guide

Version 2.0

Published October 20, 2009

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

- New Features Summary 4
- Details of Top New Features..... 4
- Index Analysis..... 4
 - Overview 4
- Visual SQL Tuning 7
 - Changing Diagram Detail Display..... 7
 - Viewing the VST Diagram in Summary Mode..... 8
 - Viewing the VST Diagram in Detail Mode 8
 - Changing Detail Level for a Specific Table 9
 - Viewing All Table Fields 10
 - Viewing Diagram Object SQL 11
 - Expanding Views in the VST Diagram 11
 - Interpreting the VST Diagram Graphics 14
 - Icons 14
 - Colors 14
 - Connecting Lines/Joins 14
 - One-to-One Join 15
 - One-to-Many Join 15
 - Cartesian Join 16
 - Implied Cartesian Join 17
 - Many-to-Many Join 18
- Implementing Index Analysis Recommendations 18
- Profiling Details..... 20

New Features Summary

The following summarizes the top new features of DB Optimizer 2.0

- [Index Analysis](#): Detailed index analysis is now available in the SQL Tuner for all supported platforms in DB Optimizer. The color-coded Index Analysis feature highlights missing indexes as well as shows which indexes are used and which are not used in the execution path. The Index Analysis feature highlights issues where the database optimizer might not be using the preferred indexes. DB Optimizer also lists indexes on the tables that do not have fields in the WHERE clause helping determine if adding an additional predicate in the WHERE clause might make use of an existing index.
- [Visual SQL Tuning](#): DB Optimizer can now parse an SQL query and analyze the indexes and constraints on the tables in the query and display the query in graphical format on the Analysis tab. The Visual SQL Tuning (VST) diagram, which can be displayed in either Summary Mode or Detail Mode, helps developers, designers and DBAs see flaws in the schema design such as Cartesian joins, implied Cartesian joins and many-to-many relationships. The VST diagram also helps the user to quickly understand the components of an SQL query, thus accelerating trouble-shooting and analysis.
- [Expanding Views in the VST Diagram](#): View and sub-views can now be expanded to show the contents within them, including the tables and the indexes on those tables. Views can now be expanded in both the Visual SQL Tuning diagram.
- [Profiling Details](#): Profiling Details of the SQL Profiler have been expanded to show Session Details for Sybase, and SQL Server and the SQL that ran in the selected session for Sybase, SQL Server, and DB2.

Details of Top New Features

The following topics describe in detail the top new features of this release.

- [Index Analysis](#)
- [Visual SQL Tuning](#)
- [Expanding Views in the VST Diagram](#)
- [Profiling Details](#)

Index Analysis

This section is comprised of the following topics:

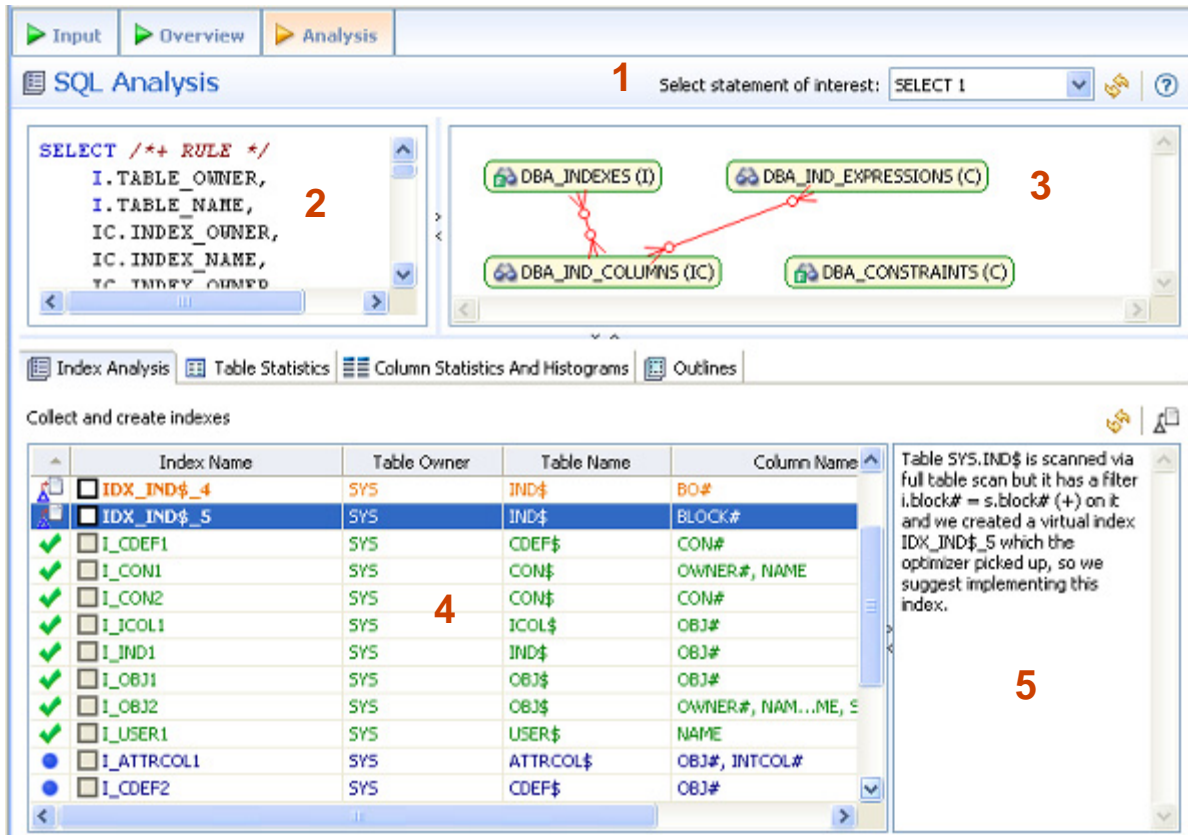
- [Overview](#)
- [Implementing Index Analysis Recommendations](#)

Overview

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

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

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







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

- 1 Statement selector, if there are multiple statements in the tuning set.
- 2 Statement text for selected statement.
- 3 Graphical diagram of the SQL statement.
- 4 Index analysis of the SQL statement. For the Oracle platform, there are several other tabs available, including Table Statistics, Column Statistics And Histograms, and Outlines.

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

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

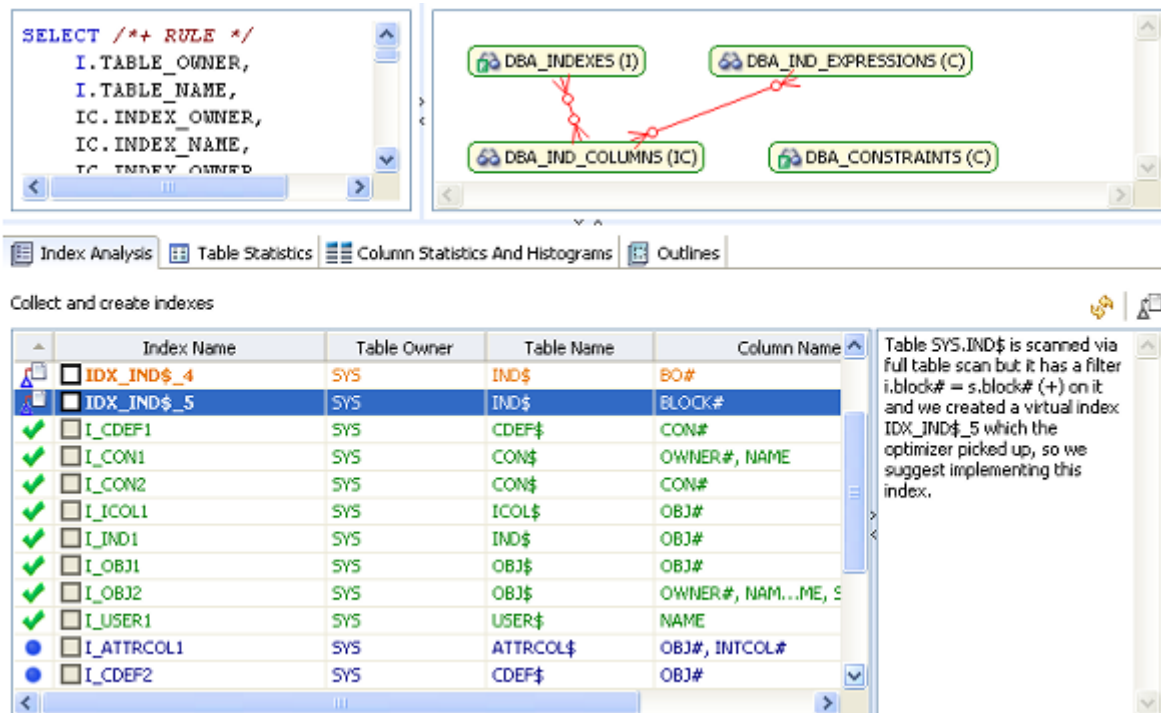
Text Color	Interpretation
	Index is used in the query
	Index is usable but not used in the current execution path.
	This index is missing. DB Optimizer recommends that you create this index.
	This index exists on the table but not usable in this query as it is written.

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

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

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

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



The screenshot shows the Oracle SQL Developer interface. The top part displays the Index Analysis tab with a diagram showing relationships between DBA_INDEXES (I), DBA_IND_EXPRESSIONS (C), DBA_IND_COLUMNS (IC), and DBA_CONSTRAINTS (C). Below this is the 'Collect and create indexes' table:

Index Name	Table Owner	Table Name	Column Name
IDX_IND\$_4	SYS	IND\$	BO#
IDX_IND\$_5	SYS	IND\$	BLOCK#
I_CDEF1	SYS	CDEF\$	CON#
I_CON1	SYS	CON\$	OWNER#, NAME
I_CON2	SYS	CON\$	CON#
I_ICOL1	SYS	ICOL\$	OBJ#
I_IND1	SYS	IND\$	OBJ#
I_OBJ1	SYS	OBJ\$	OBJ#
I_OBJ2	SYS	OBJ\$	OWNER#, NAM...ME, S
I_USER1	SYS	USER\$	NAME
I_ATTRCOL1	SYS	ATTRCOL\$	OBJ#, INTCOL#
I_CDEF2	SYS	CDEF\$	OBJ#

A tooltip for the orange-highlighted index IDX_IND\$_5 contains the following text:

Table SYS.IND\$ is scanned via full table scan but it has a filter i.block# = s.block# (+) on it and we created a virtual index IDX_IND\$_5 which the optimizer picked up, so we suggest implementing this index.

Visual SQL Tuning

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

This section is comprised of the following topics:

- [Changing Diagram Detail Display](#)
- [Interpreting the VST Diagram Graphics](#)
- [Implementing Index Analysis Recommendations](#)

Changing Diagram Detail Display

This section is comprised of the following topics:

- [Viewing the VST Diagram in Summary Mode](#)
- [Viewing the VST Diagram in Detail Mode](#)
- [Changing Detail Level for a Specific Table](#)
- [Viewing All Table Fields](#)
- [Viewing Diagram Object SQL](#)
- [Expanding Views in the VST Diagram](#)

Viewing the VST Diagram in Summary Mode

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

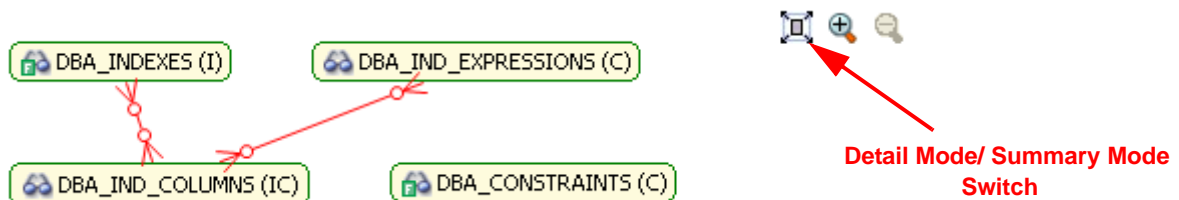
The screenshot shows the Oracle SQL Analysis tool interface. The top navigation bar includes 'Input', 'Overview', and 'Analysis'. The 'SQL Analysis' section shows a 'SELECT' statement of interest. The VST diagram on the right shows four tables: DBA_INDEXES (I), DBA_IND_EXPRESSIONS (C), DBA_IND_COLUMNS (IC), and DBA_CONSTRAINTS (C) connected by red lines. Below the diagram is a table of indexes and constraints.

Index Name	Table Owner	Table Name	Column Name	Inv
<input type="checkbox"/> IDX_IND\$_4	SYS	IND\$	TYPE#	Norm
<input type="checkbox"/> IDX_IND\$_5	SYS	IND\$	BO#	Norm
<input checked="" type="checkbox"/> I_CDEF1	SYS	CDEF\$	CON#	Uniqu
<input checked="" type="checkbox"/> I_CON1	SYS	CON\$	OWNER#, NAME	Uniqu
<input checked="" type="checkbox"/> I_CON2	SYS	CON\$	CON#	Uniqu

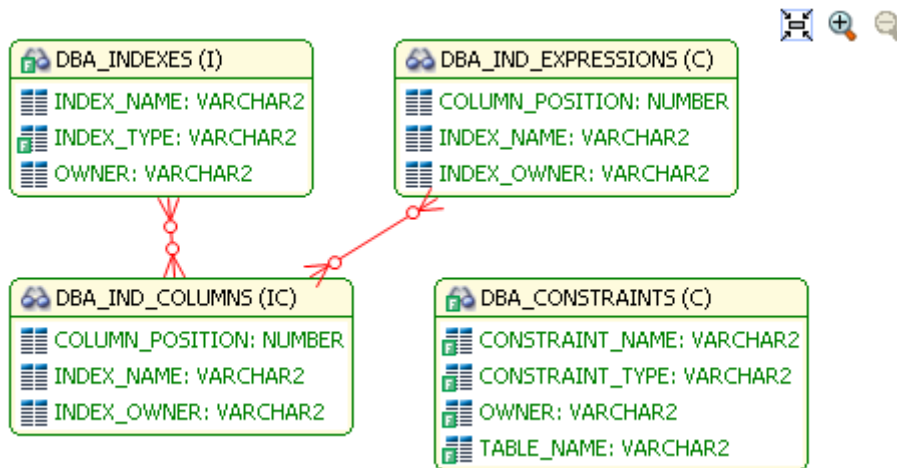
Table SYS.IND\$ is scanned via full table scan but it has a filter i.bo# = i.obj# on it and we created a virtual index IDX_IND\$_5 which the optimizer picked up, so we suggest implementing this index.

Viewing the VST Diagram in Detail Mode

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



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

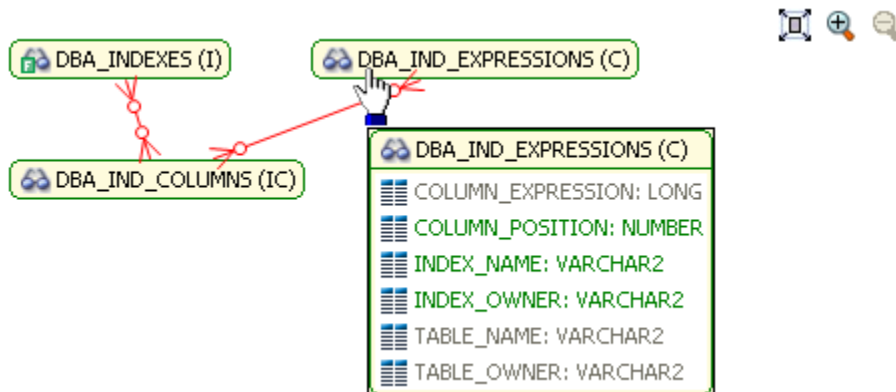


Changing Detail Level for a Specific Table

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

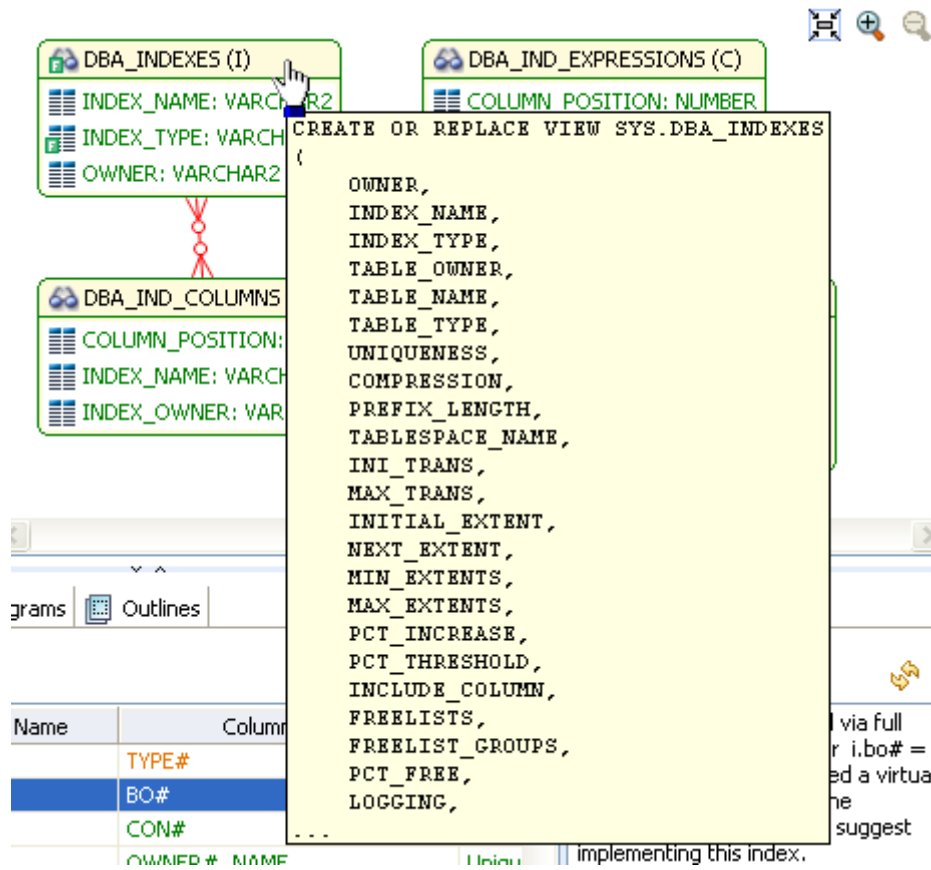
Viewing All Table Fields

Only fields that are used in the WHERE clause are displayed in detail mode; however, all fields in the table can be seen in a pop-up window when you hover the mouse over the table. The illustration below shows the pop-up window that appears when hovering over the CLIENT (c) table.

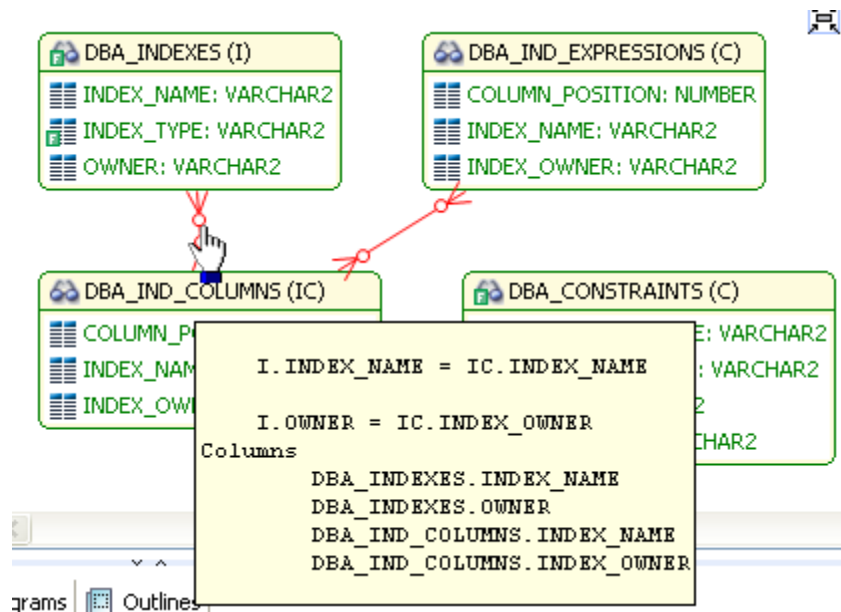


Viewing Diagram Object SQL

Hovering the mouse over the table name, field, or index displays the SQL required to create that object.



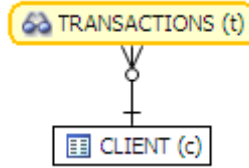
Hovering over the join between two tables displays the relationship between the two tables.



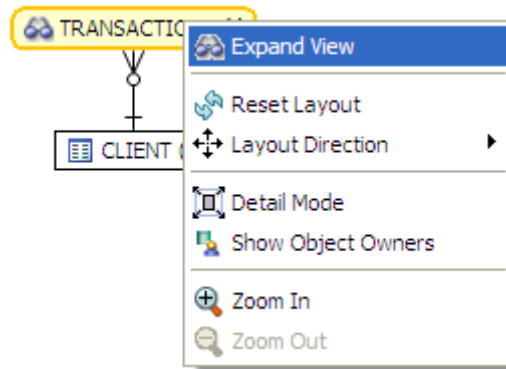
Expanding Views in the VST Diagram

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

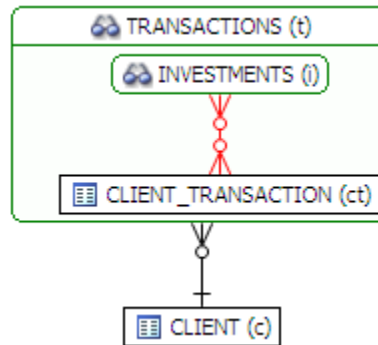
For example, the following is the default layout from query join table CLIENT (c) to view TRANSACTIONS (t):



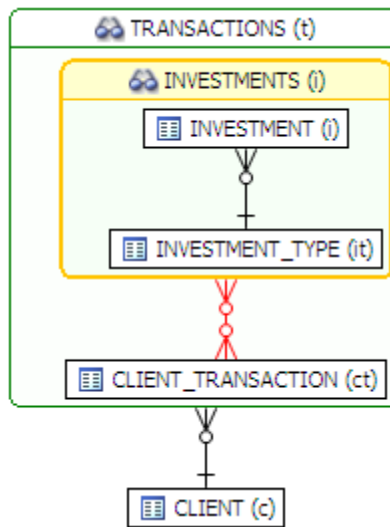
Right click on the view, TRANSACTION (t) and choose **Expand View**



Now we can see the objects in the view:



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



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

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

SQL Analysis

Select statement of interest: SELECT 1

```

SELECT COUNT (*)
FROM
  (
    SELECT
      t.transaction_id,
      t.client_id,
      t.investment_unit,
      t.investment_type_name
    FROM
      transactions t,
      client c
    WHERE t.client_id = c.client_id
  )
  
```

Index Name	Table Owner	Table Name	Column Name
<input checked="" type="checkbox"/> CLIENT_PK	SYSTEM	CLIENT	CLIENT_ID
<input checked="" type="checkbox"/> INVESTMENT_TYPE_PK	SYSTEM	INVESTMENT_TYPE	INVESTMENT_TYPE_ID
<input checked="" type="checkbox"/> CLIENT_TRAN_INVESTMENT	SYSTEM	CLIENT_TRANSACTION	INVESTMENT_ID
<input checked="" type="checkbox"/> INVESTMENT_TRANSACTION_TYPE	SYSTEM	INVESTMENT_TRANSACTION	INVESTMENT_TYPE_ID
<input type="checkbox"/> CLIENT_BROKER	SYSTEM	CLIENT_TRANSACTION	BROKER_ID
<input type="checkbox"/> CLIENT_TRAN_TRAN_BROKER	SYSTEM	CLIENT_TRANSACTION	BROKER_ID
<input type="checkbox"/> CLIENT_TRANSACTION_CLIENT	SYSTEM	CLIENT_TRANSACTION	CLIENT_ID
<input type="checkbox"/> CLIENT_TRANSACTION_PK	SYSTEM	CLIENT_TRANSACTION	CLIENT_TRANSACTION_ID
<input type="checkbox"/> INVESTMENT_PK	SYSTEM	INVESTMENT	INVESTMENT_ID






Interpreting the VST Diagram Graphics

This section will help you understand the following graphic usages:

- [Icons](#)
- [Colors](#)
- [Connecting Lines/Joins](#)





Icons

The following describes the icons used in tables displayed in Detail Mode.

Table Icon	Description
	Table Name
	Field
	Field with a filter, used in the WHERE clause
	Index
	Primary Key




Colors


The color of the index entries in the **Collect and Create Indexes** table is interpreted as follows:

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

Connecting Lines/Joins

Joins are represented with connecting lines between nodes. You can move tables in the diagram by clicking and dragging them to the desired location. The position of the connecting lines is automatically adjusted. The following describes when a particular type of connecting line is used and the default positioning of the line.

Connecting Lines	When used
	One-to-One Join relationships are graphed horizontally using blue lines.
	One-to-Many Join relationships are graphed with the many table above the one table.
	Cartesian Join shows the table highlighted in red with no connectors to indicate that it is joined in via a Cartesian join.

Connecting Lines	When used
	Many-to-Many Join relationships are connected by a red line and the relative location is not restricted.

One-to-One Join

If two tables are joined on their primary key, such as:

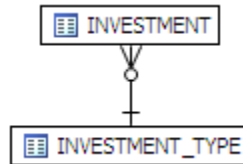
```
SELECT COUNT (*)
FROM
    investment_type it,
    office_location ol
WHERE investment_type_id = office_location_id;
```

Then graphically, these would be laid out side-by-side, with a one-to-one connector:



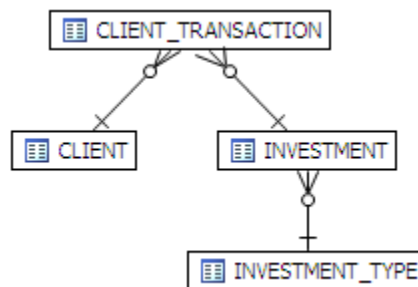
One-to-Many Join

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



The following is an example of a query that consists of only many-to-one joins, which is more typical:

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

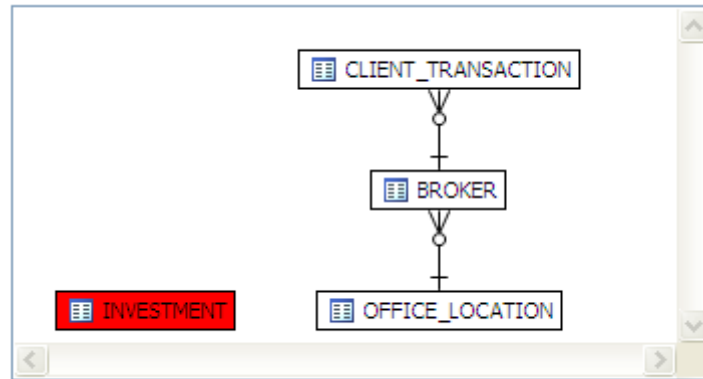


Cartesian Join

A Cartesian join is described in the following example where the query is missing join criteria on the table INVESTMENT:

```
SELECT
    A.BROKER_ID BROKER_ID,
    A.BROKER_LAST_NAME BROKER_LAST_NAME,
    A.BROKER_FIRST_NAME BROKER_FIRST_NAME,
    A.YEARS_WITH_FIRM YEARS_WITH_FIRM,
    C.OFFICE_NAME OFFICE_NAME,
    SUM (B.BROKER_COMMISSION) TOTAL_COMMISSIONS
FROM
    BROKER A,
    CLIENT_TRANSACTION B,
    OFFICE_LOCATION C,
    INVESTMENT I
WHERE
    A.BROKER_ID = B.BROKER_ID AND
    A.OFFICE_LOCATION_ID = C.OFFICE_LOCATION_ID
GROUP BY
    A.BROKER_ID,
    A.BROKER_LAST_NAME,
    A.BROKER_FIRST_NAME,
    A.YEARS_WITH_FIRM,
    C.OFFICE_NAME;
```

Graphically, this looks like:



INVESTMENT is highlighted in red with no connectors to indicate that it is joined in via a Cartesian join.

Possible missing join conditions are displayed in the **Overview** tab under **Generated Cases** in the transformations area. DB Optimize recommends that you create these joins.

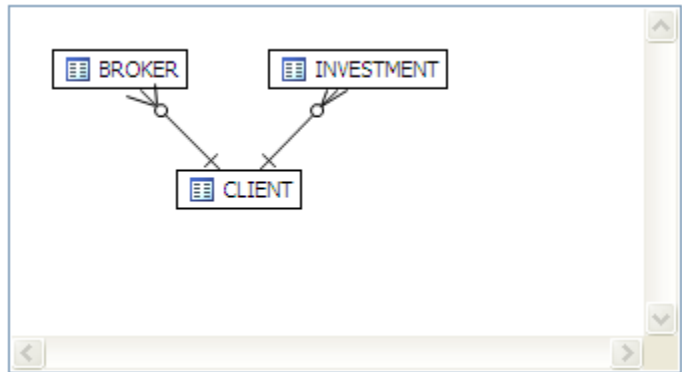
SQL Statements and Cases		Cost	Execution Statistics	Other Execution Statistics		
Name	Text	Value	Elapsed Time (s)	Physical Reads	Logical Reads	CPI
SELECT 1	select from BROKER,	34014.0	6.22	0	170	
[Missing a valid join criteria] transformation		274.0	0.04	0	167	
FULL		34019.0	6.29	0	173	
LEADING1		34017.0	6.25	0	192	
ALL_ROWS		34014.0	6.35	0	170	
LEADING3		34017.0	6.41	0	170	
INDEX		34392.0	6.58	0	414	
LEADING2		38148.0	7.94	0	170	
ORDERED		38147.0	8.61	0	170	
USE_NL		38198.0	9.03	0	37518	

NOTE: Transformations are highlighted in yellow.

Implied Cartesian Join

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

```
SELECT *
FROM
  investment i,
  broker b,
  client c
WHERE
  b.manager_id=c.client_id and
  i.investment_type_id=c.client_id;
```

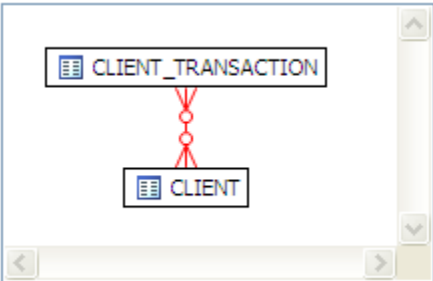
The result set of BROKER to CLIENT will be multiplied by the result set of INVESTMENT to CLIENT.

Many-to-Many Join

If there is no unique index at either end of a join then it can be assumed that in some or all cases the join is many-to-many; there are no constraints preventing a many-to-many join. For example, examine the following query:

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

There is no unique index on either of the fields being joined so the optimizer assumes this is a many-to-many join and the relationship is displayed graphically as:

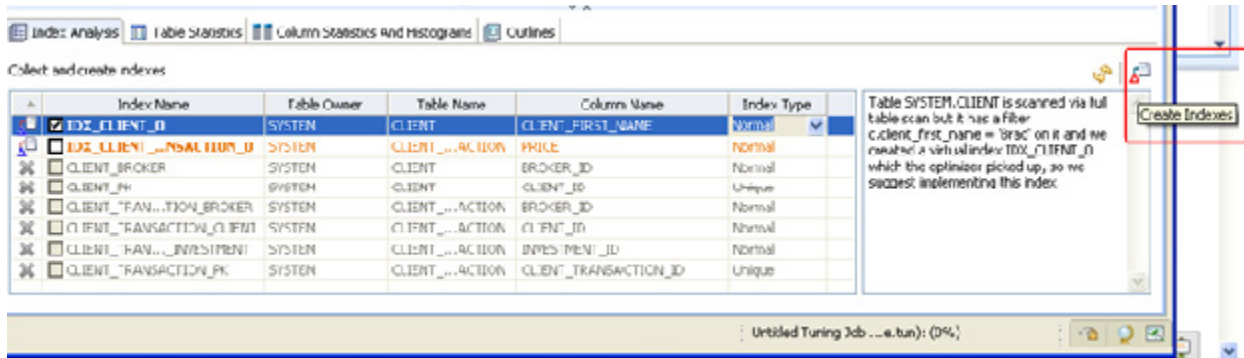


If one of the fields is unique, then the index should be declared as such to help the optimizer.

Implementing Index Analysis Recommendations

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

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



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

- 1 For any recommended index, click the checkbox to the left of the index.
 Optionally, modify the Index type by clicking in the **Index Type** column and then selecting a type from the list.
- 2 Click the **Create Indexes** button.
 The **Index Analysis** dialog appears.
- 3 To view the index SQL in an editor for later implementation, click the statement and then click **Open in a SQL editor**.
- 4 To run the index SQL and create the index on the selected database, click **Execute**.

Profiling Details

Profiling Details of the SQL Profiler have been expanded to show Session Details for Sybase, and SQL Server and the SQL that ran in the selected session for Sybase, SQL Server, and DB2.

To view session details:

- 1 In the **Profile Session** area of the **SQL Profiler**, click anywhere in the row of an application that ran during the profiling session.
- 2 In the **Profiling Details** area, click the **Session Details** tab.

Details of the session are displayed.

The screenshot shows the SQL Profiler interface. At the top, there are several tabs for tuning jobs, with the active one being 'TORLABSQL00_1_#2'. Below the tabs, there's a 'Filter by:' dropdown set to '-None-'. The main area is titled 'Profile Session' and contains a bar chart showing 'Active Sessions (avg)' over time from 09:40:00 to 09:49:00. The chart has a legend for CPU (green), Lock (cyan), Memory (yellow), Buffer (orange), and I/O (blue). Below the chart is a table with columns: User Name, Application, SPID, Active (%), Host Name, Host Process ID, and Net Address. The table has three rows, with the second row selected. Below the table is a 'Profiling Details' window for the selected session: 'Session: 55_2009-09-30 09:40:34.940 (sa / Executor Module)'. This window has two tabs: 'Session Details' and 'SQL'. The 'Session Details' tab is active, showing 'Database Server Connection' and 'Client Application' information.

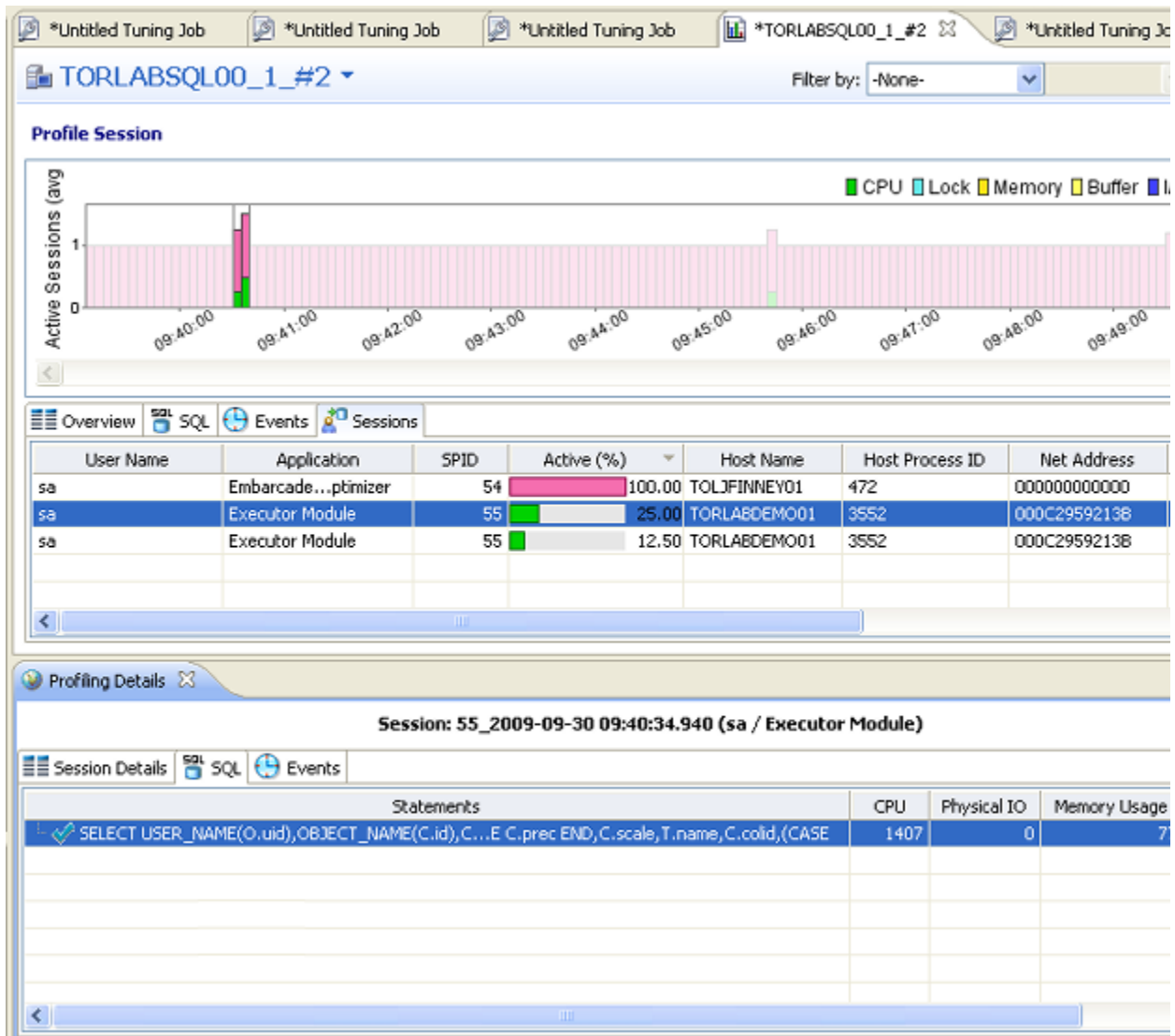
User Name	Application	SPID	Active (%)	Host Name	Host Process ID	Net Address
sa	Embarcade...ptimizer	54	100.00	TOLJFINNEY01	472	000000000000
sa	Executor Module	55	25.00	TORLABDEMO01	3552	000C2959213B
sa	Executor Module	55	12.50	TORLABDEMO01	3552	000C2959213B

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

To view session SQL:

- 1 In the **Profile Session** area of the **SQL Profiler**, click anywhere in the row of an application that ran during the profiling session.
- 2 In the **Profiling Details** area, click the **SQL** tab.

The SQL for the application that selected displays.



TIP: From the SQL tab you can easily tune a statement by right-clicking a statement on the SQL tab to initiate the tuner, which then opens with the selected statement in the Ad hoc SQL tab of the Tuner Input.