

PLATINUM



SQL-Station Plan Analyzer for Oracle

User Guide

Version 2.3

Title and Publication Number

PLATINUM Publication Number: PPO-X-230-UG00-01

Printed: January 17, 1997

Information in this guide is subject to change without notice and does not constitute a commitment on the part of PLATINUM *technology, inc.* It is supplied on an "as is" basis without any warranty of any kind, either explicit or implied. Information may be changed or updated in this guide at any time.

Copyright Information

SQL-Station Plan Analyzer for oracle is ©copyright 1/17/97 by PLATINUM *technology, inc.* and its subsidiaries. This guide is ©copyright 1/17/97 by PLATINUM *technology, inc.*, and its subsidiaries and may not be reproduced in whole or in part, by any means, without the written permission of PLATINUM *technology, inc.* and its subsidiaries.

Names marked TM or ® and other company and product names may be trademarks or registered trademarks of their respective vendors or organizations.

Mailing Address

PLATINUM *technology, inc.*
1815 South Meyers Road
Oakbrook Terrace, Illinois
60181-5235

Cover Photo

Photo by Daniel J. Cox \ Tony Stone Images \ Seattle

The Polar Bear featured on the cover of this guide is a symbol of PLATINUM's corporate commitment to preserving endangered and threatened species around the globe.

Table of Contents



Preface

Philosophy	xi
Contacting Technical Support	xii
About This Guide	xiii
What's New	xv
Conventions	xv
Related Publications	xvi

1 • Installing Plan Analyzer

Overview	1-3
Network Install	1-3
License Password	1-3
Program Group	1-3
System Requirements	1-4
Hardware Requirements	1-4
Operating System Requirements	1-4
Database Software Requirements	1-4
Space Requirements	1-4
Client Install	1-5
The README.TXT File	1-9
Database Setup	1-9
Note to DBAs	1-9
Using DBInstall to Install the SYS Views and Repository	1-11
Upgrading From an Earlier Version	1-12
Upgrading Your Current Version of Plan Analyzer	1-12

Installing a New Version of Plan Analyzer	1-14
The Admin Menu Item	1-21
Migrating from an Earlier Repository	1-21
Manually Installing the SYS Views and Repository	1-24
Post-Installation Steps	1-26
Tutorial Account	1-27
Installing Using DBInstall	1-27
Installing Manually	1-28

2 • Basics

Overview	2-3
Optimization Steps	2-3
Tutorial Account	2-4
Working with Plan Analyzer	2-4
Starting Plan Analyzer	2-4
The Plan Analyzer Main Window	2-6
Standard and Expert Modes	2-8
Entering SQL Statements	2-12
Generating Optimization Plans	2-13
Different Plan Types	2-13
Creating a Rule-Based Plan	2-14
Visualizing the Plan	2-15
Analyzing the Steps	2-17
Creating a Cost-Based Plan	2-21
Specifying Hints	2-24
Performance: System Resources	2-30
Testing the Plans	2-30
Bind Variables	2-34
The Plan Analyzer Repository	2-35
Saving	2-36
Retrieving	2-38

Deleting	2-40
Verifying Optimization Plans	2-40
Real-Time SQL Capture	2-42
Capturing SQL	2-42

3 • File and Edit Menus

File Menu	3-2
New SQL	3-2
Open SQL	3-2
Save SQL	3-3
Save SQL As	3-3
Save SQL & Plans	3-3
Print	3-3
Exit	3-4
Edit Menu	3-4
Undo	3-4
Cut	3-5
Copy	3-5
Paste	3-5
Clear	3-5
Find	3-5
Replace	3-6

4 • Database Menu

Connect	4-3
Disconnect	4-3
Save to Repository	4-3
Retrieve from Repository	4-5
Search Button	4-8
Reset Button	4-9
Retrieve Button	4-9
Details Button	4-9

More Button	4-10
Verify Button	4-10
Verify All Button	4-12
Delete SQL in Repository	4-13
All Objects	4-13
Table Information	4-14
Index Information	4-18
Trigger Information	4-19
View Information	4-20
Procedure Information	4-22
Function Information	4-22
Package Information	4-22
Sequence Information	4-22
Cluster Information	4-23
Object Dependencies	4-23
Refresh Button	4-24
Create Index	4-25
Maintenance	4-29
Correcting Errors	4-30
Admin	4-31

5 • SQL Menu

Set Bind Variable	5-2
Copy SQL to Clipboard	5-4
Format	5-5
DB Objects Analysis	5-6
Analysis Dialog Statistics	5-7
Retrieve Data	5-24

6 • Capture Menu

Active Problems	6-3
-----------------------	-----

Statistics and Features of Result Sets	6-4
The CPU Tab	6-7
The Logical Reads Tab	6-8
The Logical Writes Tab	6-9
The Physical Reads Tab	6-10
The Total I/O Tab	6-10
Server SQL Snapshot	6-10
The Result Tab	6-11
The All SQL Tab	6-14
The By Session Tab	6-15
The By Ora User Tab	6-17
The By OS User Tab	6-19
The By Table Owner Tab	6-20
The By Tablename Tab	6-22
The By Tablespace Tab	6-23
The By Datafile Tab	6-25
The By Resource Tab	6-27
By SQL Text Tab	6-28
Server SQL Monitor	6-29
Top SQL Resources	6-29
The Review Window	6-35

7 • Plan Menu

Standard Mode Plans	7-3
Expert Mode Plans	7-3
Rule	7-4
Cost First Row	7-5
Cost All Rows	7-6
Hints	7-6
Invoking the Hints Dialog	7-8
Adding New Hints	7-9
Types of Hints	7-10

Step Details	7-36
What are the Database Objects in Each Step?	7-36
Distributed Queries	7-40
Parallel Queries	7-41
Cost, Rows, and Bytes	7-42
Visualize	7-43
Explaining Each Step	7-44
Traversing the Plan	7-46
Compare	7-46
Standard Translation	7-49

8 • Performance Menu

Choosing a Plan	8-2
Server Statistics	8-3
Options Tab	8-5
Resources	8-12
Testing a Plan	8-15
Additional Information	8-16
Statistics Summary	8-19

9 • Evaluate Menu

Evaluating a Statement	9-2
Execution Configuration	9-3
Evaluation Summary	9-4
Performance Statistics Section	9-4
Plan Analysis Section	9-6
Advice	9-7
Copy SQL to Application	9-8

10 • Reports Menu

Unused Indexes	10-2
----------------------	------

Index Frequency Count	10-2
Extinct Indexes Used in Plans	10-3

11 • View Menu

View Menu Options	11-3
Toolbar	11-3
Status Bar	11-6
Collapse All Steps	11-6
Expand All Steps	11-6
Preferences	11-6
Predefined Configurations Tab	11-7
Database Connect Strings Tab	11-11
Analysis Alert Parameters Tab	11-13
Other Tab	11-18
Standard vs. Expert Mode	11-22
Cached Objects	11-23
Refresh	11-24
Drop	11-24

12 • Window Menu

Arrange Icons	12-2
Cascade	12-2
Tile	12-3
Tile - SQL/Explain	12-4
SQL	12-5
Rule Plan	12-6
First Row Plan	12-6
All Rows Plan	12-6
First, Second, and Third Hints Plans	12-6

Retrieve Data 12-7

Glossary

Index



Preface

Philosophy

PLATINUM *technology, inc.*, is the leading vendor of open enterprise systems management (OESM) products, which help organizations manage all the hardware and software components of the multiplatform, multi-operating system, multivendor environment called the open enterprise environment (OEE).

By leveraging its expertise in relational technology, PLATINUM offers products and services that increase the efficiency of individual computing systems and databases, as well as the interoperability of these systems and databases in distributed environments.

Contacting Technical Support

You can contact us with any questions or problems you have. You will be directed to an experienced software engineer familiar with the product in question

For product assistance or information, contact:

USA or Canada, toll free	800-442-6861
Illinois	630-620-5000
FAX	630-691-0708 or 630-691-0406
Internet	info@platinum.com
World Wide Web	http://www.platinum.com

To send Email to PLATINUM Technical Support, use:

Internet	techsup@platinum.com
IBM MAIL Exchange	USRWNPSN

To contact PLATINUM Technical Support, use:

USA or Canada, toll free	800-833-PLAT (7528)
IBM Software Mail	PLATSM4
CompuServe	GO PLATINUM

Our Mailing Address is:

PLATINUM *technology, inc.*
1815 South Meyers Road
Oakbrook Terrace, IL 60181-5235

About This Guide

The *PLATINUM SQL-Station Plan Analyzer for Oracle User Guide* explains how to use *PLATINUM SQL-Station Plan Analyzer for Oracle* to its fullest capabilities.

This guide assumes that the appropriate *PLATINUM SQL-Station Plan Analyzer for Oracle* components have been installed at your site. The instructions for installing the product are in the Installation chapter.

Chapter Number	Chapter Name	Content Description
1	<i>Installation</i>	Describes the installation process for Plan Analyzer.
2	<i>Basics</i>	Provides an overview of Plan Analyzer and outlines some of the basic features.
3	<i>File and Edit Menus</i>	Describes the options on the File and Edit menus.
4	<i>Database Menu</i>	Describes the options on the Database menu.
5	<i>SQL Menu</i>	Describes the options on the SQL menu.
6	<i>Capture Menu</i>	Describes Plan Analyzer's Capture menu options.
7	<i>Plan Menu</i>	Describes the options on the Plan menu.
8	<i>Performance Menu</i>	Describes the options on the Performance menu.
9	<i>Evaluate Menu</i>	Describes the options on the Evaluate menu.

Chapter Number	Chapter Name	Content Description
10	<i>Reports Menu</i>	Describes the options on the Report menu.
11	<i>View Menu</i>	Describes the options on the View menu.
12	<i>Window Menu</i>	Describes the options on the Window Menu.
	<i>Glossary</i>	Provides definitions for terminology used in this manual and within the client-server environment.
	<i>Index</i>	Helps you locate information within this manual.

What's New

PLATINUM SQL-Station Plan Analyzer for Oracle Version 2.3 offers all the functions available in Version 1, plus these new features:

Enhanced SQL Capture Facilities

Plan Analyzer's new SQL Capture features allow you to examine SQL being executed on the server in order to identify code that may be using excessive resources. You can capture currently executing SQL, or capture SQL in Batch mode to examine problems on the server over a period of time.

Create Index Wizard

Plan Analyzer's Index Wizard walks you through all the steps necessary to create indexes without exiting Plan Analyzer.

Enhanced User Interface

Conventions

The following notational conventions are used throughout this manual:

- Each manual is divided into chapters. A chapter is a main division that describes a subject matter. Each chapter contains topics that are the major sections of the chapter. Each topic may contain subtopics that further break down the topic.
- The documentation's chapter number and page number appear on each page's footer. The page numbers restart with each chapter.
- The ■ symbol denotes a set of bullet points or instructions.
- NOTE text and WARNING text are designed for easy identification.

Related Publications

As you use this User Guide, you might find it helpful to have these additional books available for reference:

- PLATINUM SQL-Station Coder User Guide
- PLATINUM SQL-Station Debugger User Guide
- Oracle7 Server Concepts Manual
- Oracle7 Server Application Developer's Guide

Installing Plan Analyzer

This chapter discusses how to install SQL-Station Plan Analyzer for Oracle.

Overview	1-3
Network Install	1-3
License Password	1-3
Program Group	1-3
System Requirements	1-4
Hardware Requirements	1-4
Operating System Requirements	1-4
Database Software Requirements	1-4
Space Requirements	1-4
Client Install	1-5
The README.TXT File	1-9
Database Setup	1-9
Note to DBAs	1-9
Using DBInstall to Install the SYS Views and Repository	1-11
Upgrading From an Earlier Version	1-12
Upgrading Your Current Version of Plan Analyzer	1-12

Installing a New Version of Plan Analyzer	1-14
The Admin Menu Item	1-21
Migrating from an Earlier Repository	1-21
Manually Installing the SYS Views and Repository	1-24
Post-Installation Steps	1-26
Tutorial Account	1-27
Installing Using DBInstall	1-27
Installing Manually	1-28

Overview

There are two basic steps to installing Plan Analyzer for Oracle (called “Plan Analyzer” in the rest of this Guide):

- 1 Run the SQL-Station **SETUP.EXE** program to copy the Plan Analyzer files to your machine.
- 2 Create a repository on each database with which Plan Analyzer will be used.

Step 1 is performed just once, and step 2 is performed once for each database you will use.

Network Install

You can perform the client-side install (step 1 above) fully on your machine, or you can choose to do a network install. If you choose a network install, you specify an existing Plan Analyzer installation to access, and the icons and program group are created in Program Manager. To perform a network install, Plan Analyzer must be fully installed on a machine to which the client is connected.

License Password

Plan Analyzer requires a license password to run. You can find the license password printed on the License Agreement Card.

Program Group

Running the **SETUP** executable copies the files to user-specified directories (if full install), and creates a Platinum SQL-Station program group with two icons: The Plan Analyzer for Oracle program is the main product executable, and the Plan Analyzer DBInstall program is an executable used for installing the required objects on each database instance.

System Requirements

To run Plan Analyzer, you must have the following minimum system requirements:

Hardware Requirements

- IBM-PC or compatible with a 386DX processor (or better).
- 8MB of RAM.

Note • Your minimum screen resolution should be 800x600 (SVGA).

Note • A faster x86 processor and more RAM may be required depending on the operating system that you choose

Operating System Requirements

- Windows 95, Windows NT Workstation 3.51 or higher, or Windows NT Server 3.51 or higher.

Database Software Requirements

- Oracle Server v7.x or higher for the server.
- SQL*Net v1.x or SQL*Net v2.x for the client.

Space Requirements

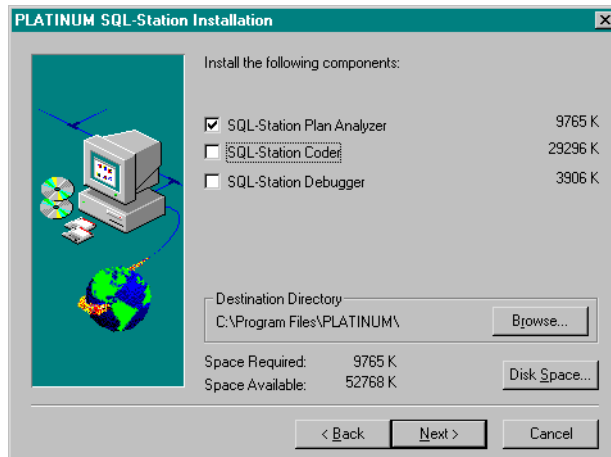
- **Full install:** at least 6 MB diskspace.
Network install: at least 1.5 MB diskspace.

- The tutorial installation creates 3 additional Oracle schemas (1 MB total). These schemas contain demonstration tables and views for the queries in the DEMO directory. Installation of the tutorial is optional.
- Plan Analyzer requires a minimum of 5MB of tablespace. Additional space may be required depending on how much will be saved into the repository.

Client Install

To execute **SETUP**, perform the following steps in Windows.

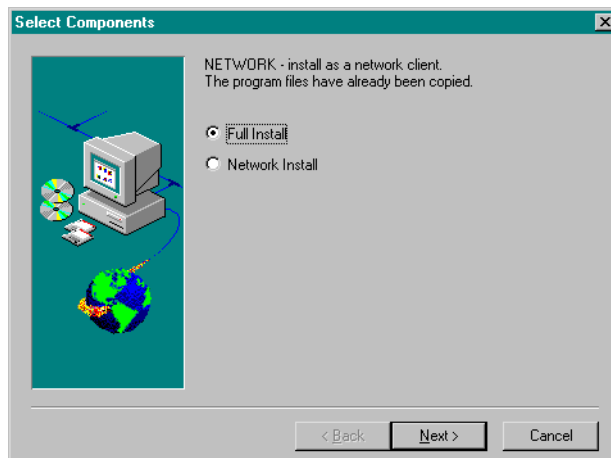
- 1 Insert the CD-ROM or Disk 1 of SQL-Station in the appropriate drive, either your CD or floppy drive. Set your machine to read the appropriate drive, and run **setup.exe** from the CD (or disk).
- 2 After a brief pause, a Welcome dialog appears. Click **Next** to continue the installation.
- 3 The Name and Company dialog appears. Enter this information and click **Next**.
- 4 A dialog appears prompting you for the SQL-Station components you want to install. If you are installing only Plan Analyzer, check only the Plan Analyzer box. See the installation chapters of the SQL-Station Debugger and SQL-Station CoderUser Guides for more information on installing those products.



Use the Browse button to specify a top-level default directory for installation. **C:\ProgramFiles\PLATINUM** is the default.

The disk space required and the disk space you have available is also listed. Be sure you have enough free disk space for the installation.

When you're satisfied with the settings, click **Next**. After a brief pause, the Select Components Dialog appears.

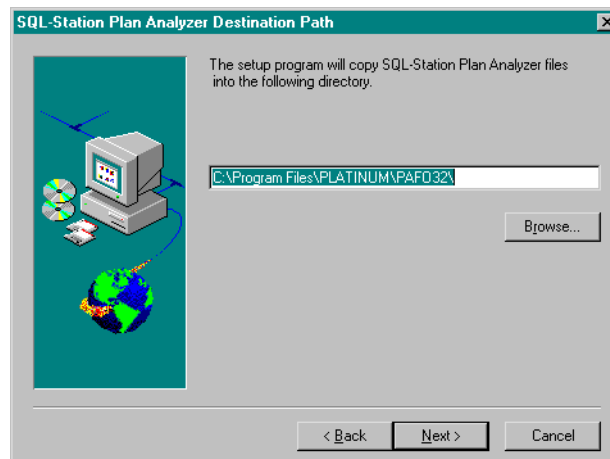


Select Components Dialog

- 5 Use the radio buttons to select either **FULL** or **NETWORK** installation. If you choose **FULL** install, the Plan Analyzer executable and all associated files will be copied to a directory on your machine. If you choose **NETWORK** install, setup simply creates icons that point to an existing full install on the network. The network install also copies two DLL files to your system32 directory.

In both cases, a program group and icons are created in Program Manager. Select Full Install or Network Install and click **Next**.

- 6 If you chose **NETWORK**, a dialog appears asking where Plan Analyzer is already installed. If you chose **FULL**, the Destination Path dialog appears.
- 7 The Destination Path dialog shows where the files for Plan Analyzer will be copied. Use the Browse button to specify a target destination shown.



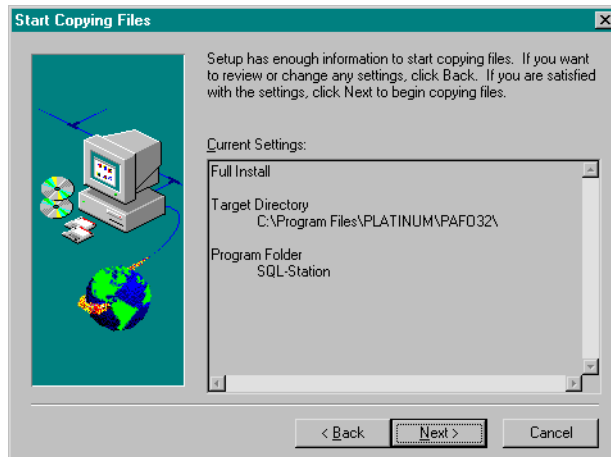
Destination Path Dialog

Clicking **Next** brings up the Select Program Folder dialog.

- 8 The Select Program folder dialog allows you to select the location of the Plan Analyzer icons. The default folder is **SQL-Station**. You can select another folder from the list, or type in a different name.

Click **Next** to invoke the Start Copying Files dialog.

- 9 The Start Copying Files dialog shows the settings you've chosen for the install: a Full or Network install, the target directory for the files, and the Program Folder name.



Start Copying Files Dialog

If any of these settings are incorrect, use the **Back** button to go back and make changes. If the settings are correct, click **Next** and Setup begins copying the files.

For Full install, Plan Analyzer copies the following DLL files:

- EXPSQL32.DLL
- DTBL32.DLL
- MFC40.DLL
- MSVCRT40.DLL
- INTEGR32.DLL

Note • It is assumed that the DLL files associated with Oracle connectivity are already installed.

- 10 Once the files are copied, the Setup Complete dialog appears. Check the box to view the ReadMe file, and click **Finish**.

The README.TXT File

All corrections to this User Guide are provided in the Release Notes. Read this file by checking the checkbox in the Setup complete dialog of the installation process.

Database Setup

Plan Analyzer requires views to be created in the SYS account, and tables and views to be created in another account (which will be referred to as the “repository account”). These database objects must be created on each database where Plan Analyzer will be used. This is considered the “database install.”

To perform the database install, Plan Analyzer DBInstall must be run *on each database* by the ORACLE DBA. Since DBInstall can only be run by the ORACLE DBA, *most users will only use the Plan Analyzer for Oracle executable*.

Note to DBAs

SYS Views

Plan Analyzer requires the creation of views in the SYS account to extend the information that the ORACLE dictionary accesses. *These extensions do not compromise the security of your system*. The views can be divided into two categories: dictionary extensions and performance views.

ORACLE7's dictionary views are incomplete. For instance there are no ALL_SEGMENTS, or ALL_CLUSTERS views. These performance views close the gaps and ensure that each user can view the dictionary information displayed in any ORACLE explain plan. Additionally, the views are present to speed access to the dictionary, since ORACLE's views can be inefficient for Plan Analyzer's purposes.

Database Setup

The performance views include views to determine the following:

- Whether an individual user is a Plan Analyzer DBA (i.e. has BECOME USER, ALTER SESSION, SELECT ANY TABLE, DELETE ANY TABLE, INSERT ANY TABLE, and UPDATE ANY TABLE permissions)
- The ORACLE SID (session ID) to retrieve performance statistics
- The SID for all sessions logged in with the same user name, or, if a DBA, the SIDs of all user sessions
- The SQL for the SIDs that are retrievable

All views conform to the security restrictions present in ORACLE and absolutely *never compromise security*.

After performing a database install, by default all ORACLE accounts will have access to the database objects created, since access is granted to PUBLIC. To restrict users from saving explain plans, statistics, and related data, the ORACLE DBA must manually revoke PUBLIC access and grant privileges directly to the specific users.

Repository

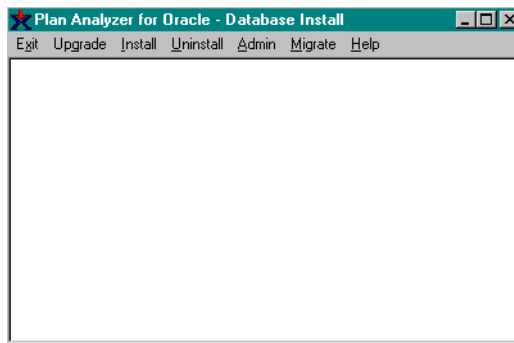
The repository is a set of tables, sequences, views, procedures, and packages used to store any results produced when using Plan Analyzer. The repository can be installed on any existing Oracle account, or on a new account. The views guarantee row level ownership, where each non-DBA can only view their own data in the repository. A DBA can view all users' work, though the DBA cannot save changes to someone else's work.

During the install of the repository, the DBA is given an opportunity to specify the default and temporary tablespaces for the repository account, as well as the initial quota and how much of the quota to initially allocate when creating the tables.

Using DBInstall to Install the SYS Views and Repository

Startup

Click the DBInstall icon in the PLATINUM program group. A menu appears with seven choices: **Exit**, **Upgrade**, **Install**, **Uninstall**, **Admin**, **Migrate**, and **Help**. (The **Uninstall** option allows you to remove all database objects associated with Plan Analyzer.)



Database Install Window

Note • If Version 1 of Plan Analyzer is already installed, Version 2 will not permit you to install the repository on the same account. You must create a Version 2 repository, then use the Migrate menu option to migrate the contents of the Version 1 repository to the new Version 2 repository. Use the Install or Upgrade menu options to create the new Version 2 repository (see note below). It is possible, however, to maintain both Version 1 and Version 2 repositories on the same server. This will allow you to run both versions of the product.

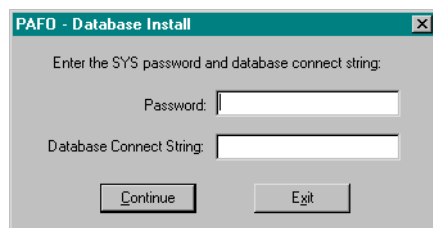
Note • If Version 2 is already installed, you must use the Upgrade menu option instead of the Install menu option. Attempting to use the Install option in this case will not work. The Upgrade menu will upgrade current views and install new views required for the latest version of Plan Analyzer.

Upgrading From an Earlier Version

Upgrading Your Current Version of Plan Analyzer

If you already have version 2.1.x of Plan Analyzer installed, you must use the **Upgrade** menu item to install 2.3.x.

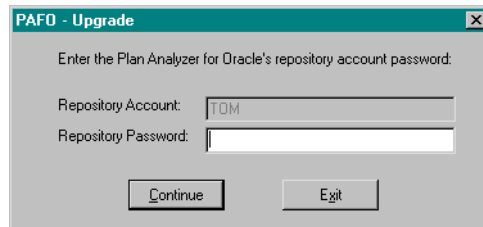
Select **Upgrade** to display a pulldown menu with two choices: Sys Views and Repository for PAFO v2.3, and Sys Views and Repository for Latest Oracle Version. Choosing either of these raises a dialog where you are prompted for the SYS password and a database connect string:



Database Install Dialog

Use the first option for upgrading from Plan Analyzer version 2.1.x to version 2.3. Use the second option for upgrading the SYS objects after you have installed a new version of Oracle.

Click **Continue**, and DBInstall creates the SYS Objects, and grants privileges to the repository account. A dialog prompting you for the repository account password appears:



PAFO - Upgrade

Enter the Plan Analyzer for Oracle's repository account password:

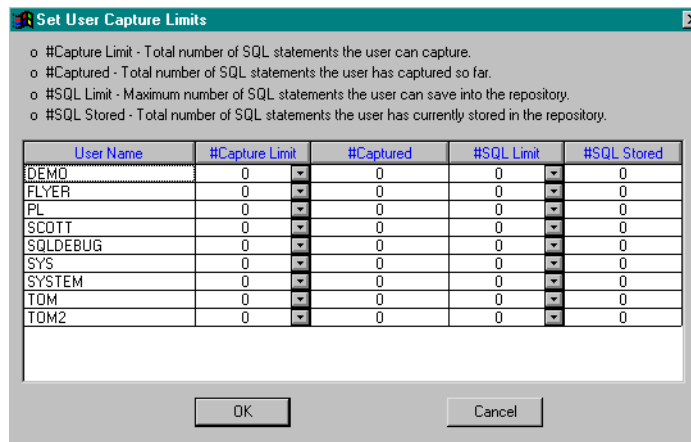
Repository Account: TOM

Repository Password:

Continue Exit

Upgrade Repository Password Dialog

Enter the password and click **Continue**. DBInstall upgrades the objects in the repository, creates the PL/SQL packages, grants privileges, and raises the **Set User Capture Limits** dialog:



Set User Capture Limits

- o #Capture Limit - Total number of SQL statements the user can capture.
- o #Captured - Total number of SQL statements the user has captured so far.
- o #SQL Limit - Maximum number of SQL statements the user can save into the repository.
- o #SQL Stored - Total number of SQL statements the user has currently stored in the repository.

User Name	#Capture Limit	#Captured	#SQL Limit	#SQL Stored
DEMO	0	0	0	0
FLYER	0	0	0	0
PL	0	0	0	0
SCOTT	0	0	0	0
SQLDEBUG	0	0	0	0
SYS	0	0	0	0
SYSTEM	0	0	0	0
TOM	0	0	0	0
TOM2	0	0	0	0

OK Cancel

Set User Capture Limits Dialog

The **#Capture Limits** column shows the maximum amount of SQL the selected user can capture. The **#Captured** column shows the number of SQL statements the user has captured currently. The **#SQL Limit** column displays the maximum number of SQL statements the user may save to the repository. The **#SQL Stored** column shows the number of SQL statements the user currently has stored in the repository.

Installing a New Version of Plan Analyzer

To change the value in either of the **Limits** columns, click once on the row in the column you want to modify, and enter a value. Or, you can select **Unlimited** from the combo box, to allow the user unlimited save and/or capture ability.

When you are satisfied with the capture limits, click **OK**. DBInstall tells you the upgrade has completed successfully.

Installing a New Version of Plan Analyzer

Select **Install** to display a pulldown menu with two choices: **SYS Views and Repository** and **Tutorial Accounts**.

You must install the SYS Views and Repository on each database where Plan Analyzer will be used. Both options are described in detail in subsequent sections.

Note • If Version 1 of Plan Analyzer is already installed, Version 2 will not permit you to install the repository on the same account. Use the **Migrate** menu option to migrate the contents of the Version 1 repository to a new Version 2 repository. It is possible, however, to maintain both Version 1 and Version 2 repositories on the same server. This will allow you to run both versions of the product.

Note • If Version 2 is already installed, you must use the **Upgrade** menu option instead of the **Install** menu option. Attempting to use the **Install** option in this case will not work. See the previous section for details on upgrading from an earlier version.

Install

When DBInstall creates the SYS Views and Repository, three things occur:

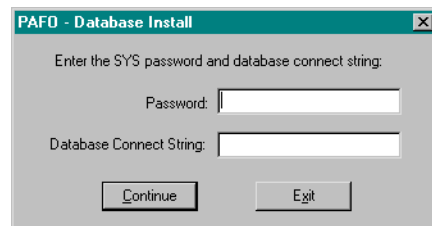
- 1 The SYS dictionary views are created.
- 2 The repository account is created.

- 3 The repository database objects are created.

Follow the steps in the next sections to install the SYS Views and Repository.

SYS Dictionary Views

- 1 Select **Install, SYS Views and Repository** from the DBInstall main menu. The following dialog displays:



Database Install Dialog

- 2 Enter the SYS password and the database connection string to log into, and click **Continue**.

Note • You can click Exit at any time to stop the install.

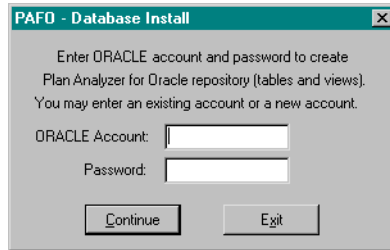
The installer checks whether an account exists containing the Version 2 repository objects. (The Version 2 repository objects can only exist in one ORACLE account.) If it finds that the objects exist, you will be requested to Uninstall all accounts, and the install will return to the main window.

If Version 2 repository objects are not found on the database, a dialog indicating the install progress displays the message, **Creating Plan Analyzer SYS objects**.

Creating a Repository Account

- 3 A dialog prompts you for the Oracle account and password for the repository tables and views.

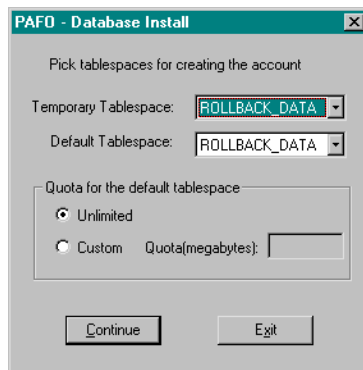
Installing a New Version of Plan Analyzer



Repository Install dialog

The Oracle account may or may not be an existing account. After entering the information, click **Continue**.

- 4 The installer checks for the existence of the account. If the account exists, it asks if you want to continue. For new accounts, the following dialog appears, prompting for the name of the tablespaces for temporary storage and default storage, and for the quota on the default tablespace.



Account Tablespace Dialog

- 5 Select the tablespaces from the combo boxes. Either select an unlimited quota, or specify a quota by clicking **Custom** and then entering the number of megabytes in the **Quota** field. Then click **Continue** to create the repository account. A window with a percentage bar shows the progress of the creation of the SYS objects.

Installing a New Version of Plan Analyzer

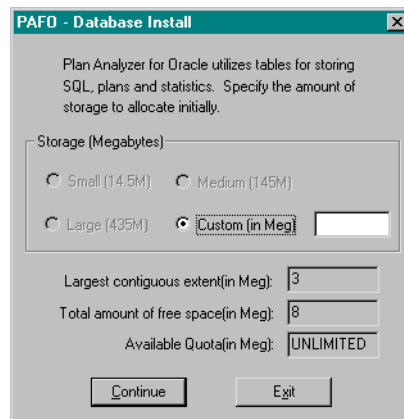
All tables and indexes created in this account will be on the default tablespace.

If `TIMED_STATISTICS` is not set to `TRUE` in the database's `INIT.ORA` file, or if it is not available on your server, a dialog displays that tells you the `TIMED_STATISTICS` parameter is set to false. See the section *CPU Timing* for more information.

After the account is created, a dialog appears with a progress bar and the message **Granting privileges to _____ account.**

Repository DB Objects

- 6 At this point `DBInstall` is ready to create the tables and indexes, but it needs to allocate storage on the default tablespace to the different objects. A dialog appears prompting for the total amount of storage to allocate to the repository tables and indexes.

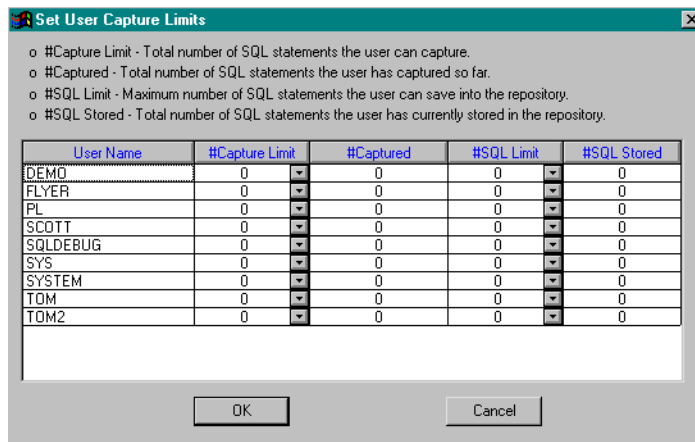


Storage Dialog

Radio buttons indicate the amount of storage in megabytes that will be divided among the objects to be created. To guide your choice, the largest contiguous extent and total free space on the default tablespace is listed, plus the remaining quota. If this is a new account, the quota granted in the previous step appears.

Installing a New Version of Plan Analyzer

- 7 Choose a large enough storage allocation to accommodate storage of SQL, optimization plans, and statistics. You can also enter a custom allocation by clicking **Custom** and entering the exact number of megabytes.
- 8 After specifying the amount of storage, click **Continue**. A dialog displays the progress as the objects are created.
- 9 A dialog appears asking if you would like to give all current Oracle users the ability to store unlimited number of SQL statements. Clicking yes raises the **Set User Capture Limits** dialog with the columns for each user set for Unlimited. Clicking no invokes the same dialog, but with the user limits set for 0, as shown below:



Set User Capture Limits Dialog

The **#Capture Limits** column shows the maximum amount of SQL the selected user can capture. The **#Captured** column shows the number of SQL statements the user has captured currently. The **#SQL Limit** column displays the maximum number of SQL statements the user may save to the repository. The **#SQL Stored** column shows the number of SQL statements the user currently has stored in the repository.

To change the value in either of the **Limits** columns, click once on the row in the column you want to modify, and enter a value. Or, you can select **Unlimited** from the combo box, to allow the user unlimited save and/or capture ability.

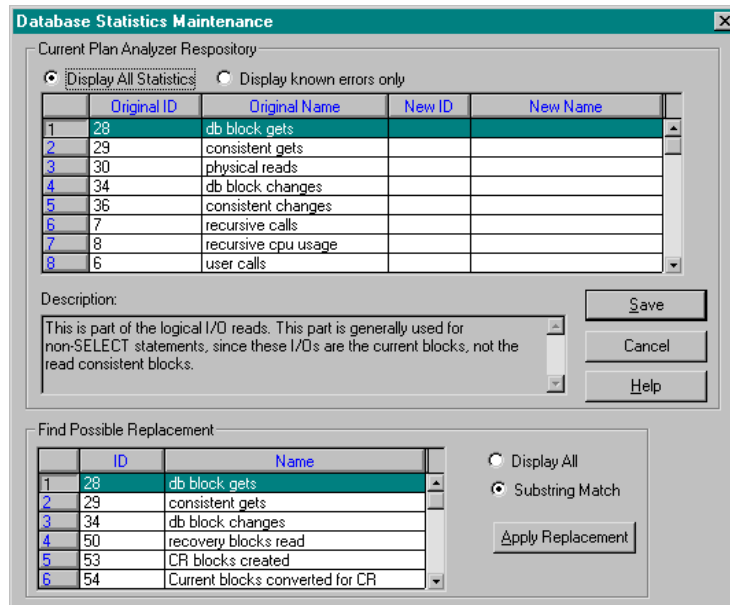
- 10 When you are satisfied with the capture limits, click **OK**. DBInstall tells you the install has completed successfully.

Oracle Statistics

One table created in the repository is EXPLAIN_STAT_IDS. This table contains statistics used in measuring the resources consumed by a SQL statement during tests. Each statistic has an ID and name that must match a row in the VSSTATNAME view. Since Oracle changes either the ID or name, or both, between versions, Plan Analyzer performs a check to ensure each statistic used is in complete agreement with the version of Oracle on which the repository is being installed. If any mismatches are found, a dialog appears that tells you an inconsistency has been found.

- 1 Click **Continue** to display the Database Statistics Maintenance dialog shown in the next figure.

Installing a New Version of Plan Analyzer



Database Statistics Maintenance Dialog

This dialog illustrates any statistic mismatches you might have. The first mismatch is highlighted, and possible matches from the current list of V\$STATNAME rows are listed at the bottom. Typically, the difference is a small name change or rearrangement of words in the name.

- 2 Select the correct replacement, and click **Apply Replacement**. After all mismatches are corrected, click **Save** to complete the installation.

Uninstall

Selecting **Uninstall** displays the same menu as **Install**. The only difference is that the objects are dropped from the specified database. Before uninstalling the repository, you should first export the data using Oracle's **EXP** utility. The ORACLE account containing the repository will not be dropped if the account contains non-repository database objects.

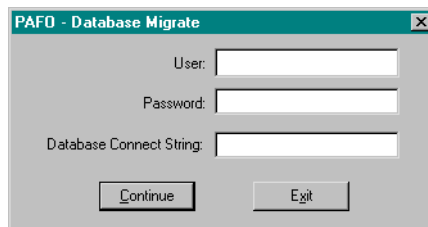
The Admin Menu Item

Clicking the Admin menu item displays a single choice: **Set User Profile**. Selecting this option invokes the **Set User Capture Limits** dialog. For a complete description of this dialog see the *Installing a New Version* section of this manual.

Migrating from an Earlier Repository

Use the **Migrate** menu option when you want to migrate a Version 1 repository to a Version 2 repository. This is the only option available under this menu.

Choosing the migrate option invokes the Database Migrate dialog:

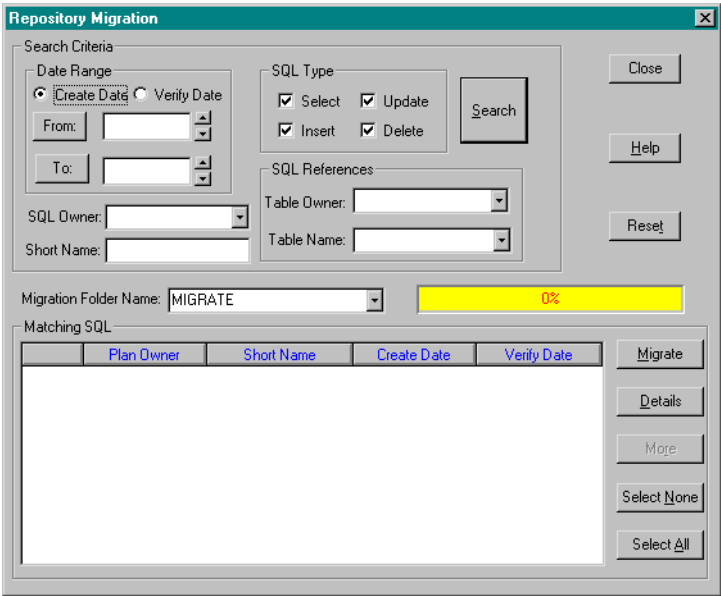


Database Migrate Dialog

Enter an account user name and password for the repository you want to migrate. Click **Continue**. This brings up the **Repository Migration** dialog:



Migrating from an Earlier Repository



Repository Migration Dialog

The Repository Migration dialog lets you search for SQL and plans to migrate to the new repository by a number of criteria, as detailed in the following table:

Component	Description
Date Range	<p>There are two different dates associated with a SQL statement: the date the SQL was originally saved or last modified, and the date the optimization plans for the SQL were last verified.</p> <p>To specify a date criterion, first select Create Date or Verify Date. Then click in the part of a date that you want to change and use the up and down arrow keys to increase/decrease it. Alternatively, click From or To.</p>

Component	Description
SQL Type	Specifies the type of SQL statements to search for. Click the checkboxes to toggle the type.
Table Owner	Queries SQL based on a table that is directly or indirectly referenced by the SQL statement. The table owner must be selected before the table name can be selected.
Table Name	Specifies which tables the user can access for the specified table owner.
SQL Owner	Identifies the ORACLE user who owns the SQL. For non-DBAs, this field lists only the current login account name. Plan Analyzer DBAs can click the combo box to list the names of all ORACLE accounts that have SQL stored in the Plan Analyzer repository.
Short Name	Gives the name assigned to the SQL statement. All ORACLE SQL wild card characters are permitted (for example, percent sign or underscore).

After you have entered the search criteria, click the **Search** button.

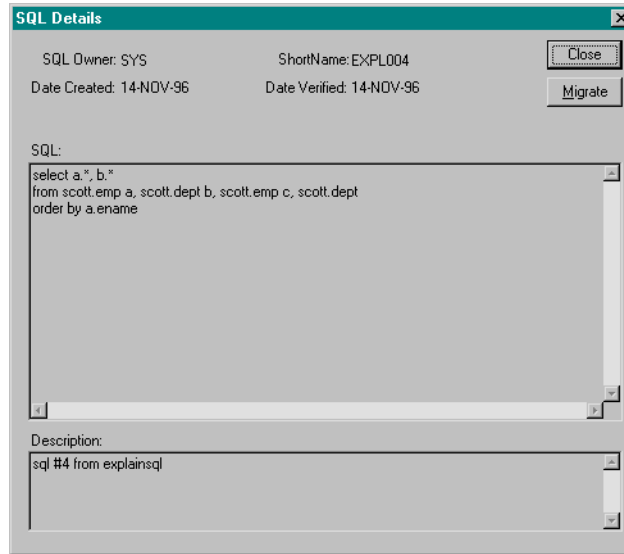
The results are displayed in the spreadsheet below the search criteria section. The spreadsheet displays the first 15 captured SQL statements. Click the **More** button to display the next 15.

The **Migration Folder Name** field shows the name of the folder the SQL will be migrated to in the new repository. **Migrate** is the default name.

Click on the number of a captured statement to select it. Alternatively, use the **Select None** or **Select All** buttons if so desired.

The **Details** button invokes the **SQL Details** dialog for the selected SQL, as shown below:

Manually Installing the SYS Views and Repository



SQL Details Dialog

The **SQL Details** dialog displays the SQL Owner, the Short Name, the Date created, the Date Verified, the SQL statement itself, and any comments that were included in the Description field. Click **Close** to return to the Repository Migration dialog, or **Migrate** to migrate the SQL to the new repository.

Clicking the Migrate button from either the SQL Details dialog, or the Repository Migration dialog causes the SQL to be removed from the spreadsheet, and migrated to the new repository.

Manually Installing the SYS Views and Repository

Follow these steps to manually install the SYS views and repository:

Manually Installing the SYS Views and Repository

Note • All the scripts needed for the manual installation are located in the Admin directory under the main install directory. Do NOT use the scripts located in the main install directory.

- 1 Login as SYS. Determine the version of Oracle you are running by typing:

```
SELECT * FROM v$version
```

- 2 Create a repository account. Execute the following statement, replacing the italicized strings with the correct values for your system:

```
CREATE user pafo identified by pafo
default tablespace user_data
temporary tablespace temporary_data
quota unlimited on user_data
```

- 3 Using SQL*Plus, run the following scripts. Use the scripts shown in the column for the version of Oracle you are running:

Version 7.0	Version 7.1	Version 7.2	Version 7.3
sys_70.sql	sys_71.sql	sys_72.sql	sys_73.sql
privs701.sql	privs701.sql	privs723.sql	privs723.sql

- 4 Connect as user **pafo** with password **pafo** to the same server.
- 5 Run the following scripts. Use the scripts shown in the column for the Oracle version you are running.

Version 7.0	Version 7.1	Version 7.2	Version 7.3
repos.sql	repos.sql	repos.sql	repos73.sql
plsql70.sql	plsql71.sql	plsql72.sql	plsql73.sql

- 6 Connect as SYS.
- 7 Run the `lstprivs.sql` script.

Post-Installation Steps

PAFO32.INI Parameters

Each client installation may differ. Specifics for each client are contained in the `PAFO32.INI` file. All settings in the `PAFO32.INI` file can be managed from the Plan Analyzer application. Simply start Plan Analyzer, and click the **Preferences** option from the **View** menu. There are four tabs: one for specifying the database connect strings, one for specifying the alarm and optimal settings for color-coded analysis, one for creating predefined configurations, and one for miscellaneous settings. The most important setting to consider at this point is the database connect strings. For more information, see the *View Menu* chapter in the User Guide.

You can enter the database connect strings as you connect, or before you connect to a database. Add connect strings for each database and save the strings. To start using Plan Analyzer, select **Database, Connect**.

CPU Timing

To obtain CPU time, you must set `TIMED_STATISTICS` in the `INIT.ORA` file to `TRUE` for each database. DBAs may be concerned that setting the parameter to `TRUE` on the production database will increase overhead. In numerous tests performed to measure the overhead, Platinum has not been able to distinguish any difference. Besides providing an invaluable statistic to determine the best optimization plan with Plan Analyzer, `TIME_STATISTICS` also benefits the DBA in many other ways when monitoring the overall system performance through `SQL*DBA`.

Even if `TIMED_STATISTICS` is set to `TRUE`, ORACLE may not be able to obtain the CPU time. This is true in the OS/2 and Netware server environments.

Tutorial Account

Plan Analyzer includes a tutorial account that you can install using the DBInstall utility (or manually). The tutorial account provides an environment in which you can practice Plan Analyzer functionality without impacting a production database.

The demonstration database utilizes three ORACLE schemas:

- One for storing human resources data
- One for storing project assignments data
- One for accessing the demonstration data

If you install the tutorial account using DBInstall, you can name these schemas anything you wish.

Additionally, Plan Analyzer includes a **demo** subdirectory which contains several sample .sql files. These files contain SQL statements that reference objects in the tutorial database. After you've installed the tutorial accounts, you can open these .sql files from within Plan Analyzer and use them to practice optimization techniques.

Installing Using DBInstall

To install the tutorial account using DBInstall, follow these steps:

Note • If you have previously installed the tutorial account and you want to re-install it, you must UnInstall it before following these steps.

- 1 Select the **DBInstall** icon from the SQL-Station program group. The Plan Analyzer - Database Install window appears.
- 2 Select **Install, Tutorial Accounts**.
- 3 When prompted, enter the password for SYS, and the connect string for the database.

Tutorial Account

- 4 Click **Continue**. DBInstall checks to see if the objects exist in another account. If a previous installation of the tutorial exists, you are prompted to Uninstall the accounts before proceeding. If Plan Analyzer objects are not found on the database, Plan Analyzer begins creating the tutorial objects.
- 5 Specify the names and storage allocation parameters for the three accounts as prompted.

Installing Manually

You can install the tutorial accounts manually if you wish. Follow these steps to manually install the tutorial accounts (all scripts are in the **admin** directory. *Do not use the scripts in the main directory for this install.*):

Note • The manual install of the tutorial account requires you to create 3 accounts with the following names: TUTORIAL, HR, and PROJECT. If you already have accounts with these names, you should use the DBInstall procedure documented above to install the tutorial and give the accounts unique names.

- 1 Use SQL*Plus to login as SYS.
- 2 Execute the following statement to create the **HR** user ID (replace italicized strings with correct values):

```
create user HR identified by HR
default tablespace default tablespace
temporary tablespace temporary tablespace
quota unlimited on default tablespace
```

- 3 Execute the following statement to grant privileges to HR:

```
grant create table, create view, create sequence,
create session, create cluster, create synonym,
create public synonym to HR
```

- 4 Execute the following statement to create the **PROJECT** user ID (replace italicized strings with correct values):

```
create user PROJECT identified by PROJECT
default tablespace default tablespace
temporary tablespace temporary tablespace
quota unlimited on default tablespace
```

- 5 Execute the following statement to grant privileges to **PROJECT**:

```
grant create table, create view, create sequence,
create session, create cluster, create synonym,
create public synonym to PROJECT
```

- 6 Execute the following statement to create the **TUTORIAL** user ID (replace italicized strings with correct values):

```
create user TUTORIAL identified by TUTORIAL
default tablespace default tablespace
temporary tablespace temporary tablespace
quota unlimited on default tablespace
```

- 7 Execute the following statement to grant privileges to **TUTORIAL**:

```
grant create session, create synonym to TUTORIAL
```

- 8 Execute the following statement:

```
insert into PAFO.EXPLAIN_TUTORIAL
values ('HR', 'PROJECT', 'TUTORIAL')
```

- 9 Use SQL*Plus to execute the file **HR.SQL** from the **HR** login.

Note • Only run scripts from the PAFO admin subdirectory! Do not run files from the main install directory.

- 10 Use SQL*Plus to execute the file **PROJ.SQL** from the **PROJECT** login.

Tutorial Account

- 11 Use SQL*Plus to execute the file **DEMO.SQL** from the **TUTORIAL** login.

Basics

This chapter discusses the basic elements of working with Plan Analyzer for Oracle.

Overview	2-3
Optimization Steps	2-3
Tutorial Account	2-4
Working with Plan Analyzer	2-4
Starting Plan Analyzer	2-4
The Plan Analyzer Main Window	2-5
Standard and Expert Modes	2-8
Entering SQL Statements	2-12
Generating Optimization Plans	2-13
Different Plan Types	2-13
Creating a Rule-Based Plan	2-14
Visualizing the Plan	2-15
Analyzing the Steps	2-17
Creating a Cost-Based Plan	2-21
Specifying Hints	2-24

Performance: System Resources	2-30
Testing the Plans	2-30
Bind Variables	2-34
The Plan Analyzer Repository	2-35
Saving	2-36
Retrieving	2-38
Deleting	2-40
Verifying Optimization Plans	2-40
Real-Time SQL Capture	2-42
Capturing SQL	2-42

Overview

You can use Plan Analyzer for Oracle to tune your databases and database applications. Plan Analyzer allows you to do the following:

- Generate optimization plans for specific SQL statements.
- Monitor server performance.
- Capture and identify SQL statements that need performance tuning.

These features are discussed later in this chapter and in detail in the Menu chapters.

Optimization Steps

The process of optimization involves trial and error. Generally, optimizing includes steps such as the following:

- 1 Generate different plans for a particular SQL statement.
- 2 Examine the plans to see if a different access path should be taken.
- 3 Revise the query; perhaps by rewriting the query, adding hints, or adding or dropping an index.
- 4 Regenerate the plans.
- 5 Execute with the new plans and compare statistics (especially CPU time).
- 6 Save the SQL, plans, and statistics into the Plan Analyzer repository.

Plan Analyzer for Oracle structures and facilitates this trial and error process. The following sections of this chapter outline the basic procedures involved in completing the above steps.

Tutorial Account

Plan Analyzer provides a tutorial account you can install and use for practicing. If you wish, you can use the tutorial account to practice the basic procedures outlined in this chapter, without having to use a production database.

For information on installing and using the tutorial account, refer to the Installation chapter.

Working with Plan Analyzer

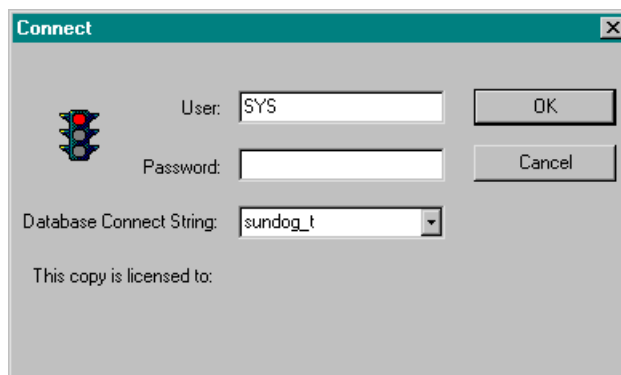
Before you can start Plan Analyzer, the Plan Analyzer DBInstall program must have been run on at least one Oracle database. For more information refer to the Installation chapter.

You can start Plan Analyzer from within SQL-Station Coder, or stand-alone.

Starting Plan Analyzer

To start Plan Analyzer for Oracle from outside of SQL-Station Coder, follow these steps:

- 1 Select SQL-Station Plan Analyzer from the SQL-Station group. The Connect dialog appears:



Use the Connect dialog to specify the account you want to access and the connect string for the ORACLE server.

- 2 In the **User** field, enter the username for an existing ORACLE account.
- 3 In the **Password** field, enter the corresponding password.
- 4 In the **Database Connect String** field, enter or select the connect string for the ORACLE database server that contains the database you want to work with (and on which you have run DBInstall).

Depending on your version of SQL*Net, the form of the connect string may differ from the example shown above.

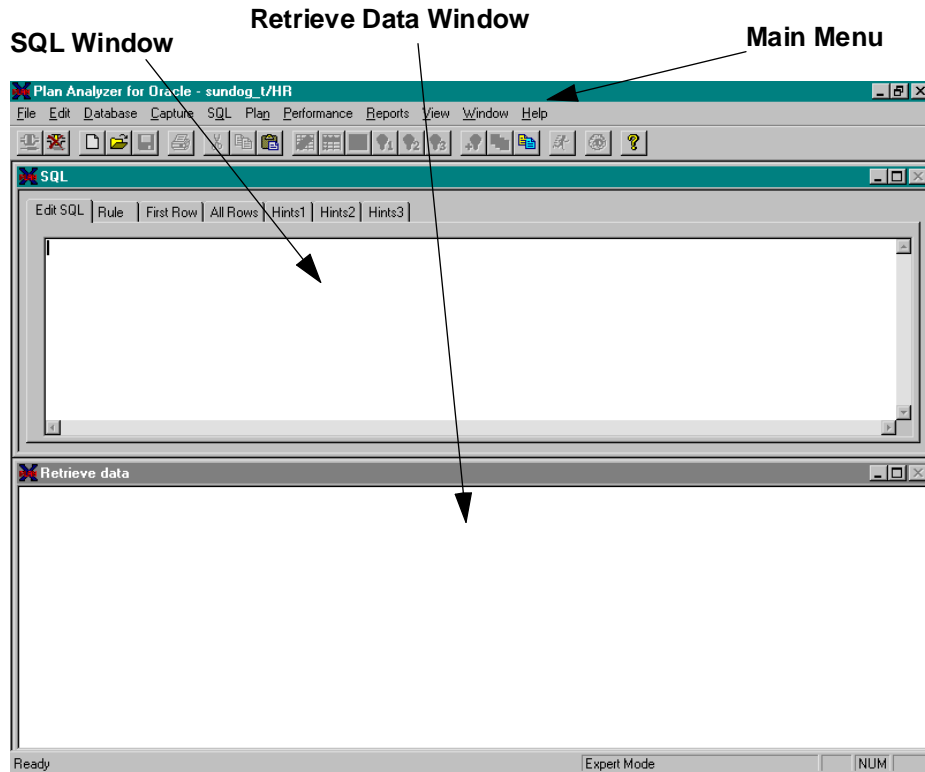
- 5 Click **OK** to connect to the specified account. The stoplight changes to yellow as the connection is being established, and then to green once the connection is successful.

Note • You can use the Preferences dialog to specify Database Connect strings for future connections. For more information, see the *Preferences* section of the *View Menu* chapter.

The Plan Analyzer main window appears.

The Plan Analyzer Main Window

Following is the Plan Analyzer main window in its default state:



Plan Analyzer's Default Windows

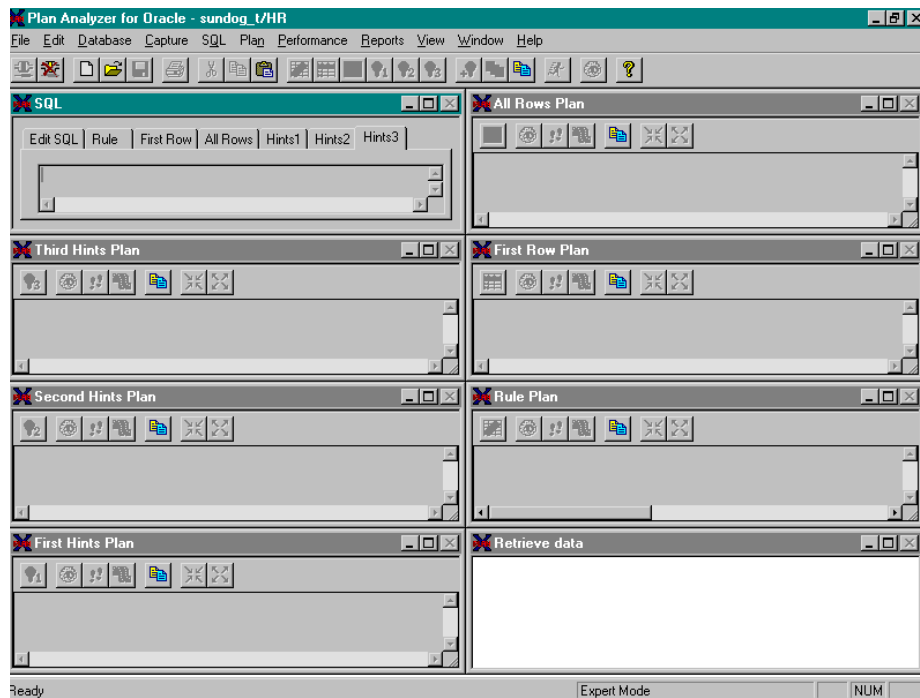
By default, Plan Analyzer starts in Expert mode. If you want to change to Standard mode, you can select **View, Standard Mode** from the main menu. For more information on Standard and Expert modes, see the *Standard and Expert Modes* section later in this chapter.

When the main window comes up, it is divided into two windows: the SQL Window, and the Retrieve Data Window. Behind the Retrieve Data Window are six additional windows:

- Rule Plan Window
- First Row Plan Window
- All Rows Plan Window
- First Hints Plan Window
- Second Hints Plan Window
- Third Hints Plan Window

You can also access these windows by choosing the tab with the same name from the tabs along the top of the SQL window.

If you choose **Window, Tile**, you can see all eight windows at one time:



Plan Analyzer Main Window -- Tiled

The SQL Window is where you insert, enter, or paste SQL statements. The six plans windows display the respective optimization plans. The Retrieve Data window displays data returned from the database when the statement in the SQL Window is executed.

Standard and Expert Modes

Plan Analyzer for Oracle has two modes of operation: Standard mode and Expert mode. If you are not familiar with Oracle's EXPLAIN PLAN facility, you should use the Standard mode. Advanced users should use the Expert mode. At any time, you can switch between the different modes and Plan Analyzer will translate the plans and statistics appropriately. The default mode is Expert. The mode can be changed in the Preferences dialog from the **View** pulldown menu. Each mode has its own menu bar and toolbar.

Expert Mode

Expert mode provides more detail and flexibility than Standard mode. For example, Expert mode permits you to produce any plan in any order you want, while still having the choice to display the plans in the simple terms used in Standard mode. Additionally, you have complete control over execution options when testing the SQL, such as the array size, and the number of array fetches to return. You have detailed options, such as separating the “parse” portion of the test from the “execute” portion, and more statistics to help determine what occurred during the execution.

Expert mode provides you with six different optimization plans for each SQL statement:

- Rule-based
- First-Rows-based
- All-Rows-based
- Three different Hints-based plans

You can also execute and compare statistics for each plan. (See the *Plan Menu* chapter for a detailed explanation of the different optimization modes.)

The Expert menu bar follows:



Expert Menu Bar

The Expert toolbar contains icons that map to the menu items:



Expert Toolbar

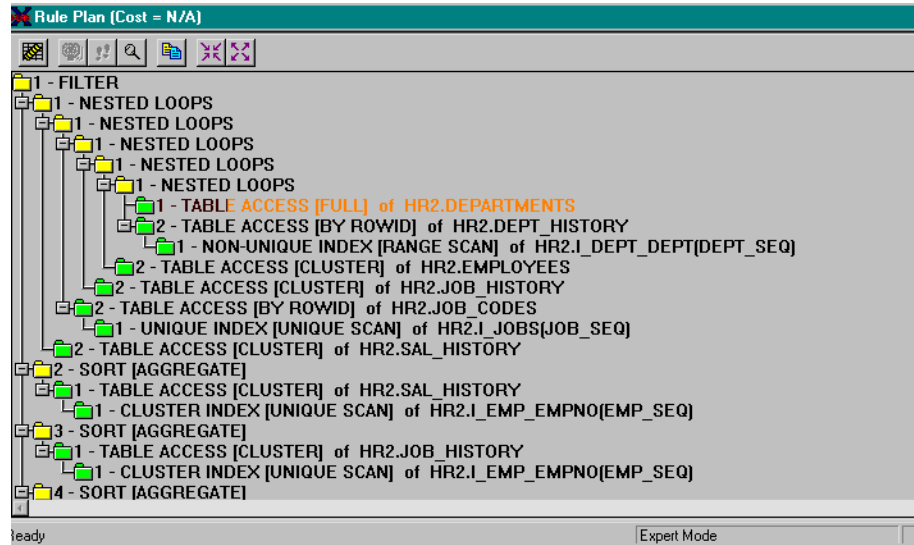
For an explanation of each toolbar button, see the *Toolbar* section of the *View Menu* chapter.

Standard Mode

Standard mode provides a simplified interface, and requires only entry of the SQL statement, and specification of whether Plan Analyzer should test the SQL statement for an interactive or batch application. Plan Analyzer performs the relevant tests, provides you with a summary, and suggests the “best” plan for optimal execution. The best plan determination is based on resource cost information entered in the Other tab of the Preferences item. (See the *View Menu* chapter for more information on the Preferences options.)

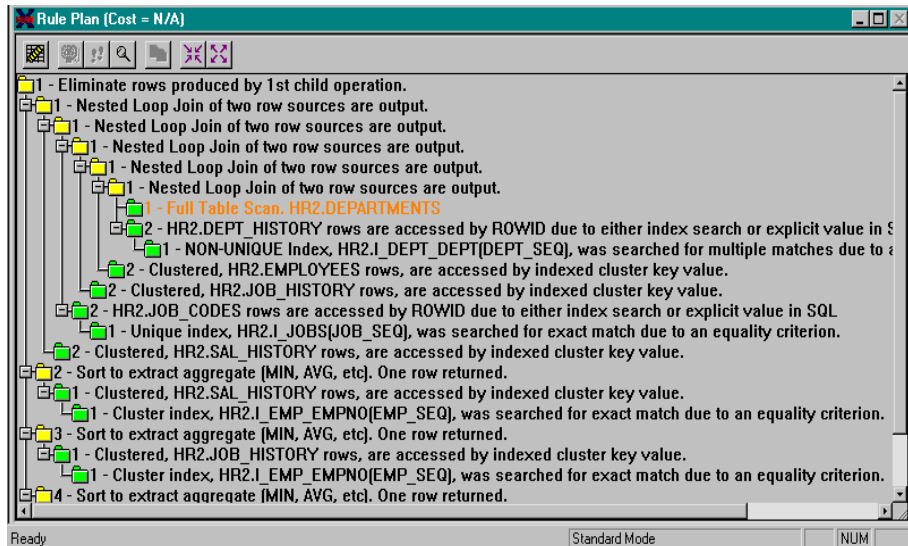
Optimization plans can be complex and difficult to understand. The following figures represent the same Rule-based optimization plan, however the first plan is displayed in Expert mode, and the second plan is displayed in Standard mode.

Working with Plan Analyzer



Expert Mode Plan

Working with Plan Analyzer



Standard Mode Plan

The Standard mode plan is easier for a beginner to understand. However, the conciseness of the steps in the Expert mode can benefit an advanced user.

The Standard menu bar follows:



Standard Mode Menu

The Standard toolbar follows:



Standard Mode Toolbar

For an explanation of each toolbar button, see the *Toolbar* section of the *View Menu* chapter.

Entering SQL Statements

When you first start Plan Analyzer, there are three ways to enter a SQL statement into the SQL Window:

- You can type or paste a statement.
- You can insert a statement from a file or repository folder using **File, Open SQL**, or **Database, Retrieve from Repository**.
- You can capture SQL using the **Capture, Active Problems** feature, the **Capture, Server SQL Snapshot** feature, or the **Capture Server SQL Monitor** feature. These features allow you to capture SQL from the server and work with it in Plan Analyzer. See the *Capture Menu* chapter for a detailed description of these features.

Note • If you are using Plan Analyzer from SQL-Station Coder, you can also transfer a SQL statement from the Coder Edit Window into the Plan Analyzer SQL window. For more information, see the *SQL-Station Coder User Guide*.

Plan Analyzer can generate optimization plans for the following types of SQL statements:

- SELECT
- INSERT
- DELETE
- UPDATE

Note • You can only enter a single SQL statement in the SQL Window at one time. If you are inserting a file, it must contain only one SQL statement (although sub-queries are allowed).

Once you have entered a SQL statement, you can generate an optimization plan for it.

Generating Optimization Plans

Different Plan Types

In Expert mode, Plan Analyzer can generate four different types of optimization plans:

- Rule
- Cost First Row
- Cost All Rows
- Hints

Note • Since Hints may already exist in the SQL, Plan Analyzer ensures that the hints are disabled when performing any optimization plan except Hints-based.

Rule

Rule-based optimization is the original technique used by ORACLE to base optimization on a set of predefined rules. Rule-based optimization does not use any storage or data statistics. In certain cases, rule-based optimization can provide the best performance.

Cost First Row

First Row optimization has the goal of fast response: returning the first row to the application immediately. In order to provide fast response, ORACLE often produces an optimization plan that requires more resources than an All Rows optimization. This is because the user is waiting at the terminal for results, and wants an initial return as soon as possible. Typically, First Row optimization is used for interactive applications.

Cost All Rows

All Rows optimization has the goal of best throughput - getting the entire result set back to the user. This means that the user may wait before any rows are returned. This is usually the best optimization type for a batch report.

Hints

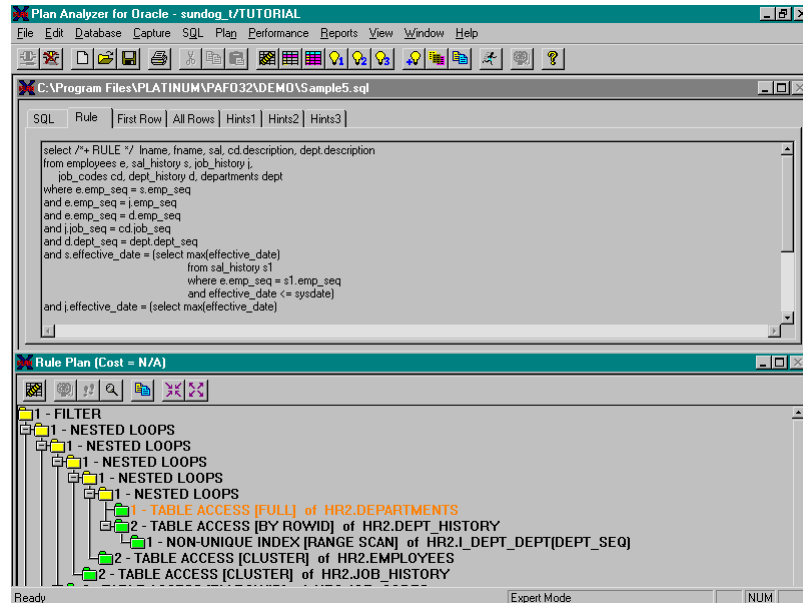
Hints optimization involves entering hints, or directions, to the ORACLE optimizer in an attempt to modify the default optimization plan. In this mode, ORACLE evaluates any hints entered (manually or through the Plan Analyzer Specify Hints dialog).

Creating a Rule-Based Plan

To create a Rule-based plan in Expert mode, follow these steps.

Note • The following steps refer to menu options and displays from Expert mode only. If you are running Standard mode, the menu options and output will differ from those described here. To switch to Expert mode, select View, Expert Mode from the menu

- 1 Enter or insert a valid SQL statement in the SQL Window. You can type a statement directly into the SQL Window, or you can select **File, Open SQL** to insert a file that contains a SQL statement.
- 2 Select **Plan, Rule** from the menu. After a brief pause, the Rule-based optimization plan appears in the Rule window. The SQL window shows the actual SQL used to generate the Rule plan. Selecting the tab for the various types of plans shows the SQL for the plan, as well as the ordered plan steps. The following screen shows a sample Rule plan output:



Sample Rule Plan

The lines are indented to indicate the hierarchy of the plan. An indented line is a child operation of the line above it. In the above example, NESTED LOOPS is a child operation of FILTER.

To further clarify the order of the plan, you can use the Visualize facility to animate the execution flow and step through each line.

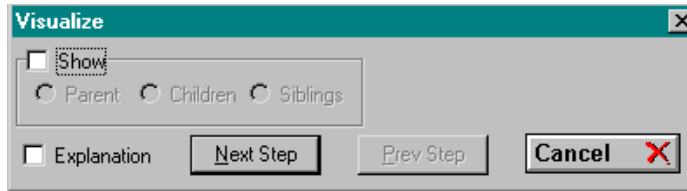
Visualizing the Plan

The **Visualize** facility allows you to examine the execution flow of the plan. The first step (or *driving step*) of an execution is the most important one. It is the driving force behind the query, and is often the step that needs the most attention. Among other things, **Visualize** allows you to examine the first step of the execution and determine its effectiveness.

Generating Optimization Plans

You can use the Visualize option to step through an optimization plan of any type. To visualize the execution flow of a Rule-based plan, follow these steps:

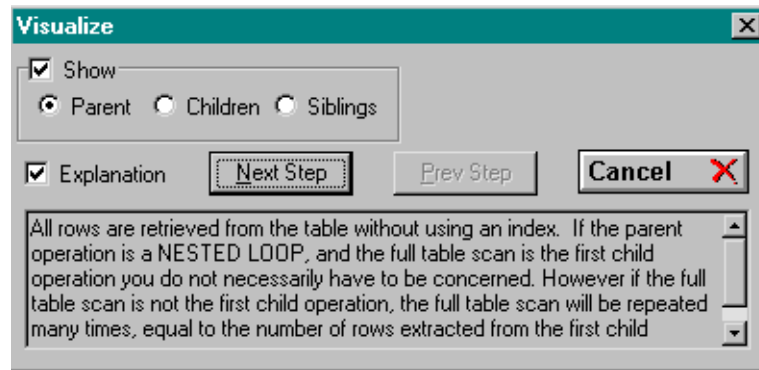
- 1 With a plan in the Rule window, select **Plan, Visualize, Rule** from the menu. The Visualize dialog appears:



Visualize Dialog

- 2 If you check the **Show** checkbox, you can choose whether to indicate the Parent, Children, or Sibling statements as you step through.
- 3 Check the **Explanation** checkbox to display an explanation of each step as it occurs.
- 4 Click **Next Step** to start animation. Plan Analyzer highlights a line in the plan. This is the first operation performed by ORACLE when executing the SQL statement. Depending on which options you have chosen in the Visualize dialog, the output differs. For example, if you checked **Explanation**, the Visualize dialog includes an explanation of each step of the plan as it is highlighted:

Generating Optimization Plans



Visualize Dialog with Explanation


- 5 If you selected **Show Parent** in the Visualize dialog, the Parent to the current statement flashes at each step of the execution. Similarly, selecting **Show Children** or **Show Siblings** causes the child or sibling statements to flash.
- 6 Click **Next Step** to move to the next operation or **Prev Step** to move to the previous step. At any point, you can click **Cancel** to dismiss the Visualize dialog and stop the animation.

When you reach the last step in the plan, the Visualize dialog is dismissed.

Analyzing the Steps

Once you have generated the plan, you can evaluate the effectiveness of the driving step. If a step in the plan does not perform the desired operation, you can consider rewriting the query or using a different optimization mode to force a different plan.

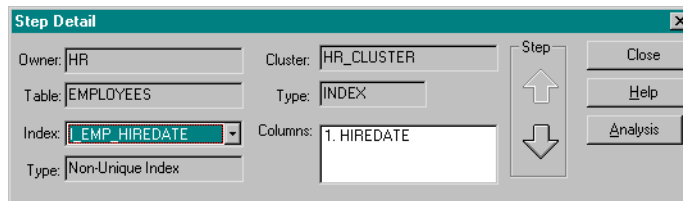
Step Detail Dialog

If you click on an optimization step that references a database object, Plan Analyzer enables the **Step Detail** menu option under the **Plan** menu, and also enables the Step Detail toolbar icon .

Generating Optimization Plans

You can use the Step Detail dialog to list information about database objects that are referenced (directly or indirectly) by the optimization step. You can then use this information to help evaluate the effectiveness of the plan.

- 1 Click on the line of the optimization plan for which you want more information.
- 2 Select **Plan, Step Details** from the menu. The Step Detail dialog appears:



Sample Step Detail Dialog

The Step Detail dialog lists information about objects related to the database object in the selected plan line. The above figure shows that the EMPLOYEES table is part of an indexed cluster, HR_CLUSTER. The **Index** combo box lists all indexes on the specified table, including the cluster index. You can use this information to search for an alternate index.

- 3 Click on the **Index** combo box down-arrow to display a list of indexes on the table. To display information about each index, click on the index names one by one.

The **Columns** field displays a list of the columns on which the index is based and the order in which they occur.

The **Type** field indicates whether the index is unique or non-unique, and whether it is the cluster index.

You can use the information to evaluate the following:

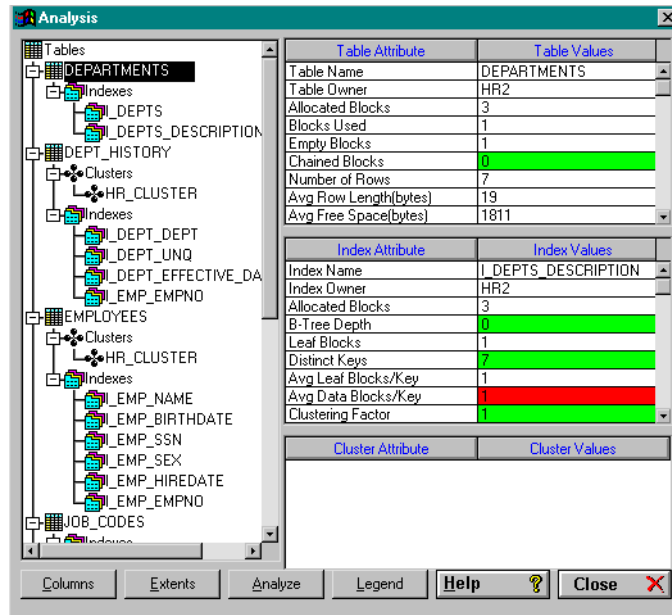
- Whether the indexes are based on optimal columns (i.e, foreign keys).
- The type of cluster.
- The table accessed by the cluster index.
- What other indexes exist on the table.
- Whether or not to use a different index to drive the query.

You can further analyze the table information by clicking the **Step Detail Analysis** button.

Analysis Dialog

To display complete storage and data statistics for a selected object, click **Analysis** from within the Step Detail dialog. The Analysis dialog displays:

Generating Optimization Plans

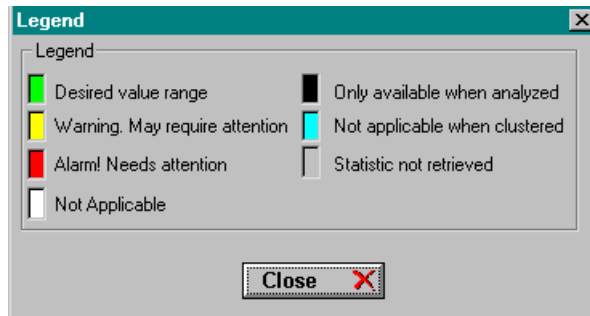


Analysis Dialog

The Analysis Dialog displays statistics on the selected index, its associated table, and its cluster index (if one exists). Since there are so many statistics, they are color coded to provide a quick summary. You can modify the thresholds associated with each color using the Preferences dialog. For more information, see the *Preferences* section of the *View Menu* chapter.

The **Table Attribute** and **Table Values** fields show the table listed in the Step Details dialog. The **Index Attribute** and **Index Values** fields show statistics on the index, if one was present in the step. The Cluster Attribute and the Cluster Values fields display information on the cluster, if present.

Clicking the **Legend** button brings up a table that lists the colors used to summarize the statistics.



Legend for Colors in the Analysis Dialog

Interpreting Statistics

The Analysis dialog contains storage statistics about the table and associated indexes (including the cluster if applicable). You must first understand what each statistic means, and then use the statistics to determine such things as:

- Should this index be used to drive a query?
- Should this index, table, or cluster be reorganized?
- Could this index be useful if the rows were sorted prior to inserting?
- Should the hashed cluster be redefined for more cluster keys?

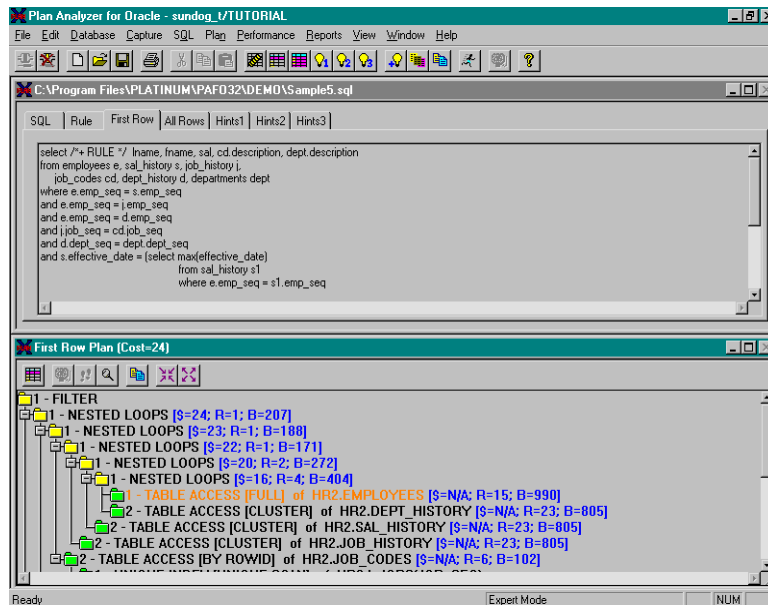
Each of the above questions in itself has many points. For example, if the table should be reorganized, is it because the table has too many extents, too many chained rows, too much free space per block, or too many empty blocks? It is possible to determine whether an index will ever be useful based upon the statistics. For more information on interpreting statistics, see the *Analysis Dialog Statistics* section of the *SQL Menu* chapter.

Creating a Cost-Based Plan

In addition to Rule-based plans, you can create First Row and All Rows optimization plans:

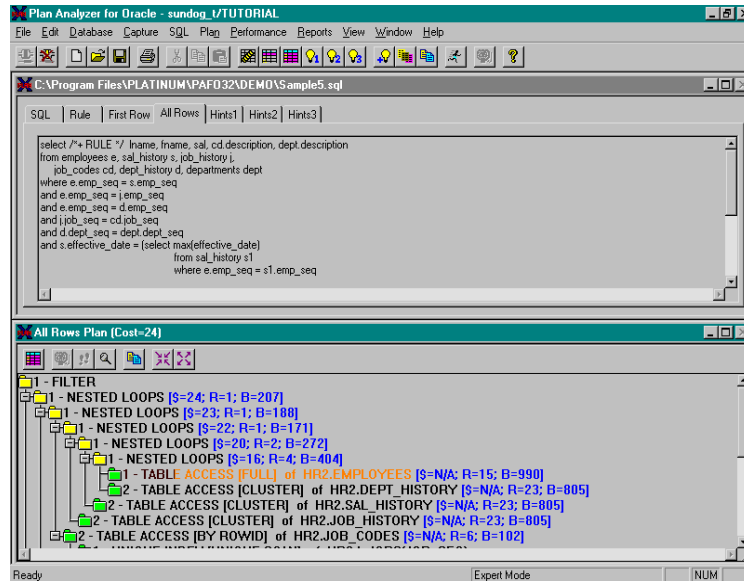
Generating Optimization Plans

- 1 To create a First Row plan, enter a SQL statement in the SQL Window, and select **Plan, Cost First Row** from the Plan Analyzer menu. After a brief pause, the First Row window comes to the front, and the optimization plan is displayed.
- 2 Similarly, you can create an All Rows plan by selecting **Plan, Cost All Rows** from the menu. After a brief pause, the All Rows plan window comes to the front, and the optimization plan is displayed.



Explain - First Row Cost Plan

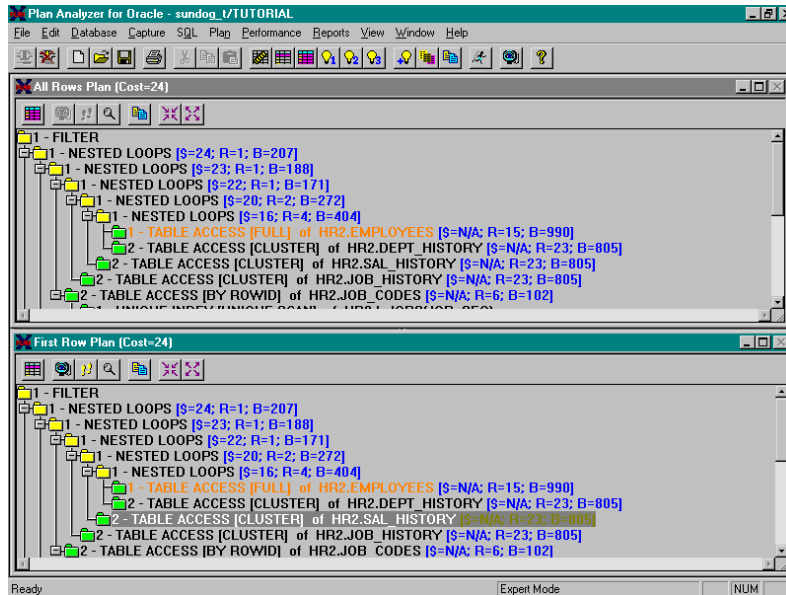
Generating Optimization Plans



All Rows Cost Plan

- 3 You can compare the cost plans by tiling the plan windows. Select **Windows, Tile** to see both Cost plans at the same time. You can minimize any empty windows, or windows you do not want to see.

Generating Optimization Plans



Comparing Cost-Based Plans

- To return the windows to the original state, you can select **Window, Tile - SQL/Explain**.

For more information about Cost-based optimization plans, see the *Cost First Row* and *Cost All Rows* sections in the *Plan Menu* chapter.

Specifying Hints

Hints are suggestions to ORACLE on how best to optimize the SQL. In Versions earlier than 7.3, ORACLE statistics do not track the actual distribution of values in each column. ORACLE knows how many different values there are, but not whether one value occurs more than another.

Note • When you specify a hint, ORACLE does not always use it.

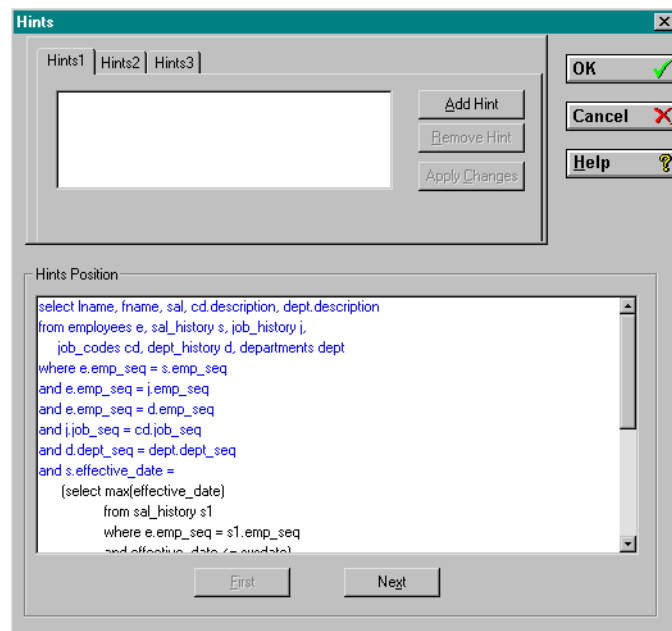
In this version of Plan Analyzer for Oracle, you may specify up to three separate Hints Plans on a single SQL statement, enabling you to compare the results of several different Hints schemes. Note that each set of hints must include either the RULE, FIRST ROW, or ALL ROWS hint. This hint controls the optimization mode for the SQL statement and any subqueries.

Using hints is a two step process:

- Specifying which hint to use and where
- Executing the hint

To specify and execute hints, follow these steps:

- 1 With a SQL statement in the SQL window, select **Plan, Hints, Specify Hints** from the menu. The Hints dialog appears:



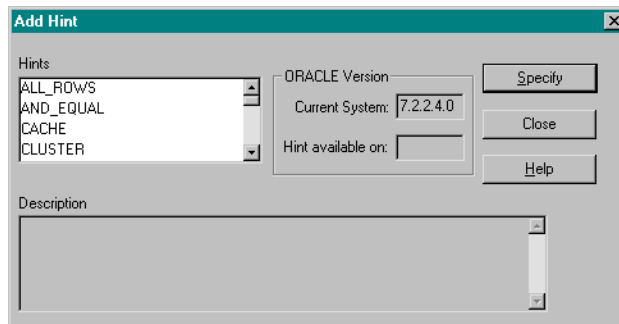
Hints Dialog

Generating Optimization Plans

Note the three tabs for **Hints1**, **Hints2**, and **Hints3**. All three plans apply to the SQL statement in the window. You may specify different hints in each of the hints plans. Thus, you can test for the most cost effective hints plan.

The SQL statement appears in the window with the first SELECT statement colored blue, indicating where the hint will be placed once the **Apply Changes** button is clicked. (For example, in the above figure, the first SELECT statement, `select lname, fname, sal etc.` is colored blue.) If the statement contains multiple SELECT statements (such as nested or correlated subqueries) you can use the **Next** button to move between them, adding hints to the various modules of the statement. The **First** button repositions the color coding at the first location where hints are permitted.

- 2 Click the **Add Hint** button to enter a new hint. The Add Hint dialog appears:

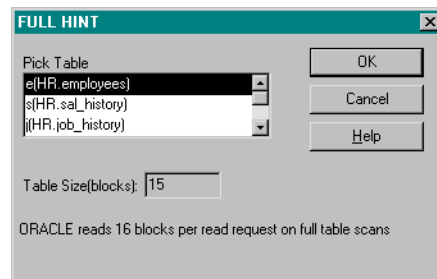


Add Hint Dialog

The Hints combo box lists all hints that can be used. Since some hints take table and index names as parameters, you will need to specify the table and/or index information for these hints.

- 3 Scroll through the Hints combo box and select the desired hint. The **Add** button will toggle to **Specify** if the hint you've selected requires any additional table or index information.

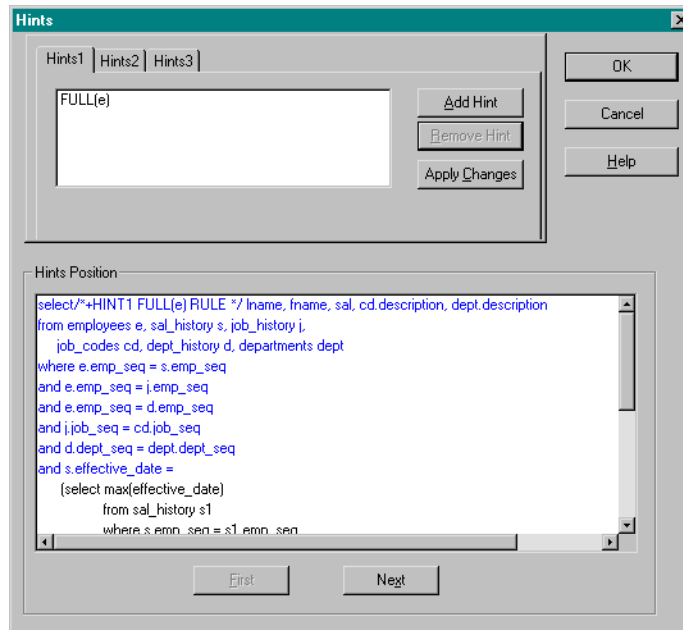
- 4 If the button toggles to **Specify**, click it to display a dialog in which you specify the dependent tables and/or indexes for the hint. For example, selecting the FULL hint and clicking **Specify** causes the following dialog to display:



Select a table or tables, and click **OK**.

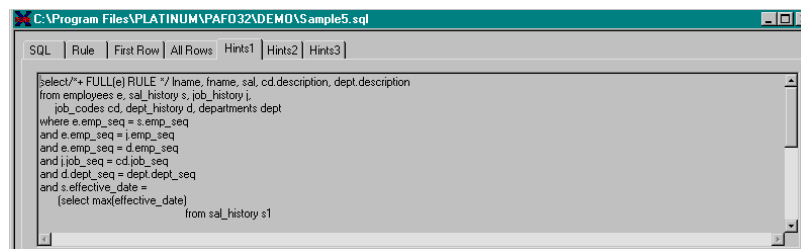
- 5 If the hint doesn't require table or index parameters, click **Add** from the Add Hint dialog. The Hint dialog reappears.
- 6 The hint is listed in the Hints list box. Click **Apply Changes** to place the hint in the SQL statement.

Generating Optimization Plans



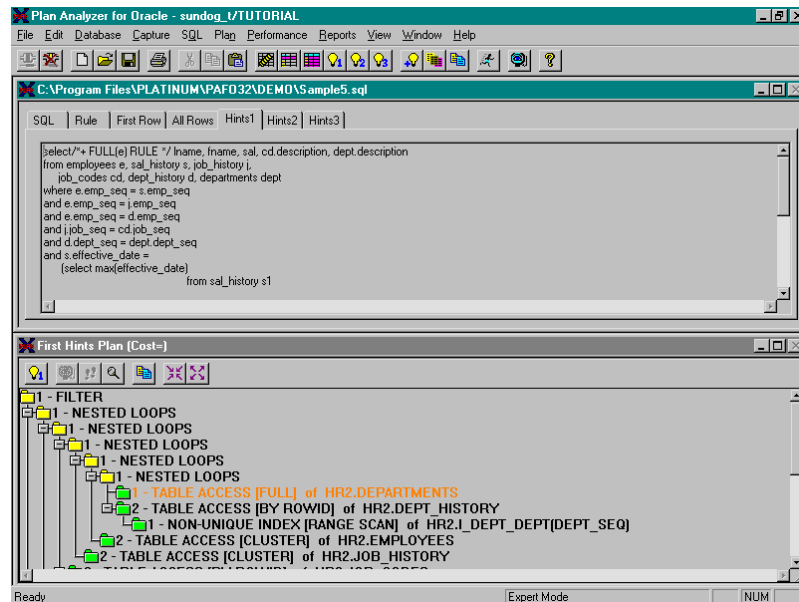
Hints Dialog After Apply Changes

- 7 Click **OK** to return to the main window. The SQL window displays the SQL statement with the new hint:



SQL Window with Hint

- 8 Select **Plan**, **Hints**, **Execute** from the menu to produce the optimization plan with the new hint.



Optimization Plan with Hints

The plan and cost are displayed in the First Hints Plan window.

Note • The Cost= value does not, in itself, represent any quantifiable value. Rather, it represents the relative cost of efficiency of each plan. The cost associated with each plan can be used for comparison purposes between plans for the same SQL statement. You cannot compare the cost value between two different statements.

You can further tune the optimization of a single hints plan by adding additional hints or by using a different hint. Use **Plan, Hints, Specify** to remove or add hints, use **Plan, Hints, Execute** to test them.

Use the other Hints plans to compare different Hints schemes. You can specify different hints for the same SQL statement in the three hints plans, save and execute them, and compare the results.

For more information about hints, see the *Hints* section of the *Plan Menu* chapter.

Performance: System Resources

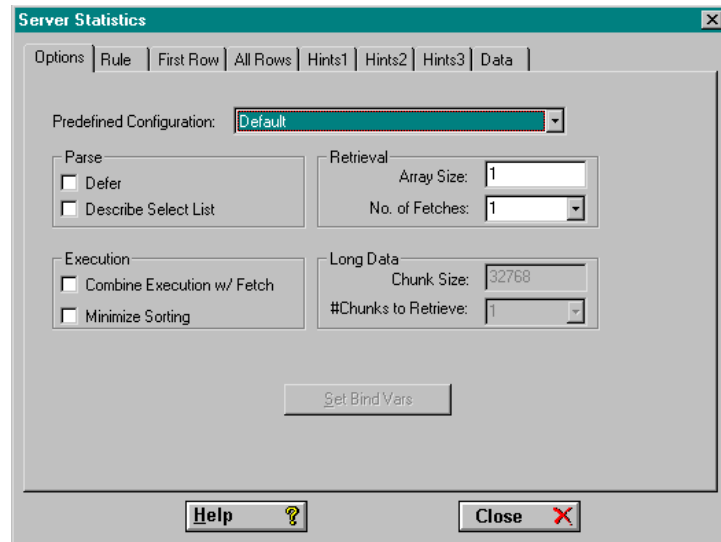
How can you decide which plan is the most efficient? Which plan uses which resources? The cost is only a single variable. Rule-based optimization does not have a cost and may often be the best choice. It is also possible that a higher-cost plan will execute the fastest. Therefore, you could execute the SQL for each optimization plan to see which one performs the fastest. Prior to measuring the resources used, Plan Analyzer allows you to specify the exact environment in which the SQL will be executed.

Testing the Plans

You can use the Server Statistics dialog to test the performance of your SQL statements for each optimization plan. The **Server Statistics** menu option is available only in Expert mode.

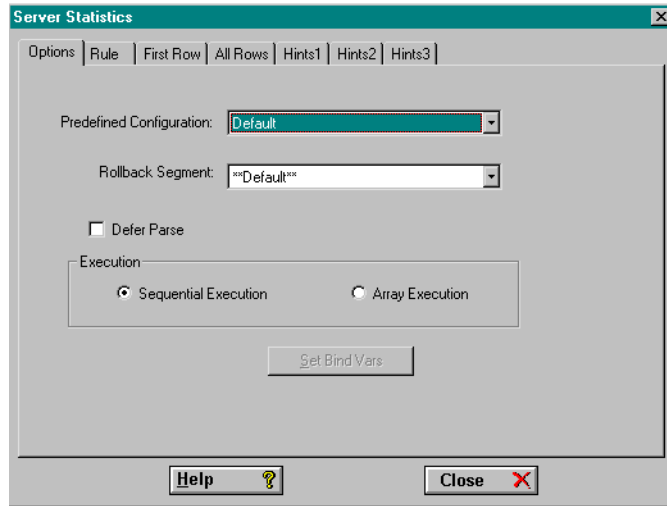
Follow these steps to test the SQL statement for each type of optimization:

- 1 With a statement in the SQL window, select **Performance, Server Statistics** from the main menu. If your statement contains bind variables, the Bind Variable dialog displays. Specify values for any bind variables and click **OK**. The Server Statistics dialog appears:



Server Statistics Dialog (Options Tab for SELECT Statement)

The contents of the Server Statistics Options tab vary depending on whether the statement is a SELECT statement, because different execution options are available. The above figure displays the Options tab for a SELECT statement. The figure below displays the Options tab for a non-SELECT statement:



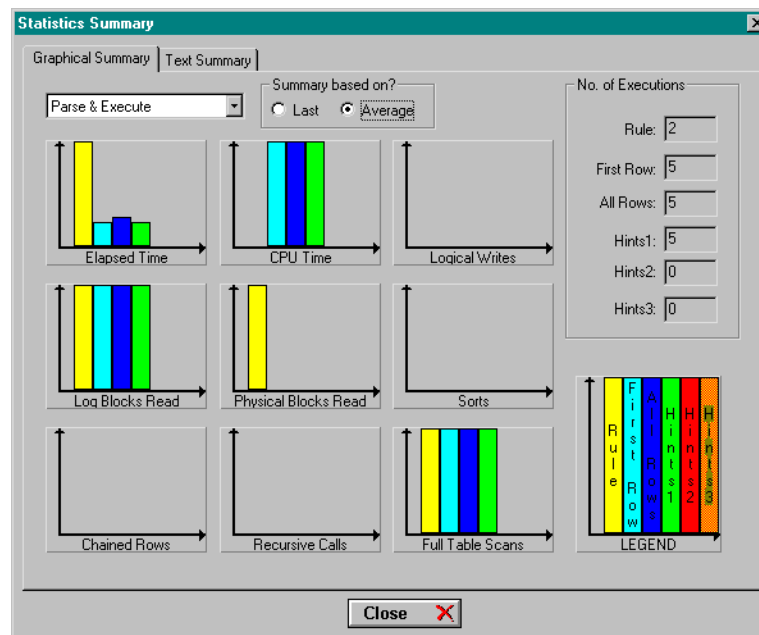
Server Statistics Dialog (Options Tab for Non-SELECT Statement)

The Server Statistics dialog contains an Options tab and a tab for each optimization plan type. In the Options tab, you can specify parameters that affect performance. Most of the options are used to emulate the exact environment in which the SQL will be embedded.

For a detailed description of the Server Statistics dialog, refer to the *Server Statistics* section of the *Performance Menu* chapter.

- 2 In the Options tab (for SELECT statements), the **Array Size** field indicates the number of rows fetched across the network in a single round-trip message to ORACLE. Network messages increase response time. Since you want to reduce messages by returning a group of rows on each call, set the Array Size to a value appropriate for the number of rows. For example, for a database with few rows, 20 is a good value.
- 3 The **No. of Fetches** field indicates how many database fetch calls to make during the test. Each fetch will attempt to return the number of rows specified in the **Array Size** field. If that is not possible, then the query is done. If there are very few rows in the database, leave the value as close to 1 as possible, meaning return up to *X* rows, where *X* is the value specified in **Array Size**.

- 4 After specifying the options, click **Test** from within each plan tab to execute the statement and gather statistics for that plan type. Make certain to select the plan tab for the SQL plan you want to test, i.e. Rule, First Row, All Rows, or one of the Hints plans.
- 5 Test each plan type multiple times. The **Test** button increments to indicate how many tests you have run.
- 6 After you have run tests for each desired plan, click **Summary** to display the Summary dialog:



Statistics Summary Dialog (Graphical Tab)

	Rule	First Row	All Rows	Hints1	Hints2	Hints3
Elapsed Time (Sec)	0.23	0.05	0.06	0.05		
CPU Time (Sec)	0.00	0.01	0.01	0.01		
Logical Blocks Read	4	4	4	4		
Physical Blocks Read	8	0	0	0		
Logical Writes	0	0	0	0		
Recursive Calls	0	0	0	0		
Database Calls	3	3	3	3		
Chained Rows	0	0	0	0		
Sorts	0	0	0	0		
Full Table Scans	1	1	1	1		
Execution Count	2	5	5	5		

Statistics Summary Dialog (Text Tab)

- 7 The Summary dialog lists statistics for each plan tested, including the elapsed time, CPU times, I/O, database calls, and full table scans.

You can use these statistics to determine which plan to use. Once you have chosen the most efficient plan type, you can close the Server Statistics dialog. You can then select **SQL, Copy SQL to Clipboard** from the main menu to copy the optimal plan to the clipboard, where it is available to paste into your application.

Bind Variables

Bind variables allow you to re-execute the same SQL statement multiple times without having to reparse the SQL statement. This is because the constants that drive the SQL are variables and are entered after the SQL statement is parsed, but before it is executed.

For example, consider the following SQL statement:


```
SELECT LNAME, FNAME, MNAME  
FROM HR.EMPLOYEES  
WHERE EMP_SEQ= :empno
```

In this example, **empno** is the bind variable. The bind variable causes this statement to be parsed once and re-executed many times. Before each execution, the bind variable gets a different value by a separate call to ORACLE to specify the value. ORACLE may also produce a different optimization plan when bind variables are used rather than explicit constants.

Note • Applications with embedded SQL that uses bind variables should be tested with the bind variables, not with substituted constants.

If your SQL statement contains bind variables, you are first prompted for values for the bind variables when you select **Performance, Server Statistics**.

Non-SELECT Statements

Non-SELECT SQL statements can also contain bind variables, but you can specify an array of values for each bind variable. If an array of values is used, the same number of values must be entered for each bind variable in the SQL statement. You can then test array operations which are unique to ORACLE (See the *Sequential vs. Array Execution* section in the *Performance Menu* chapter for more details).

The Plan Analyzer Repository

The repository is a database in which Plan Analyzer stores saved SQL, plans, statistics, and other information relevant to Plan Analyzer functionality. Plan Analyzer creates the repository when you run DBInstall and maintains it transparently.

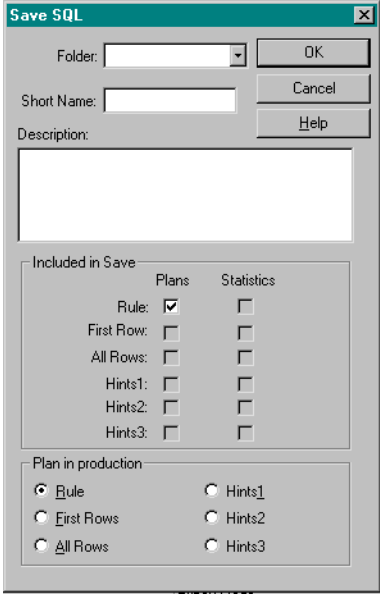
Saving

You can use the **Save to Repository** option under the **Database** menu to save SQL, optimization plans, statistics, and bind-variable values. Once saved, the stored SQL and optimization plans can be a valuable resource. Following are some features of the **Save to Repository** option:

- You can save optimization plans that are difficult to understand, and review them later with the DBA.
- Database changes, such as adding or dropping indexes or re-analyzing tables, can change optimization plans. Plan Analyzer lets you verify whether the optimization plans have changed.
- You can store SQL by application, thereby providing a means of documenting the application. To ease entry of the SQL, Plan Analyzer provides a monitoring option to capture all SQL in an applications as it executes. You can then review the captured SQL and decide which SQL statements to save.
- You can retrieve or verify SQL statements and optimization plans based on criteria such as owner, short name, creation dates, or, more importantly, by a table name that is directly or indirectly referenced in the SQL statement.

To save the current optimization plans and statistics, perform the following steps:

- 1 With a SQL statement in the SQL Window, select **Database, Save to Repository** from the main menu. The Save SQL dialog appears:

The image shows a 'Save SQL' dialog box with a title bar containing a close button. The dialog has several input fields and buttons. At the top, there is a 'Folder:' label followed by a drop-down menu and an 'OK' button. Below this is a 'Short Name:' label followed by a text input field and a 'Cancel' button. Underneath is a 'Description:' label followed by a large text area and a 'Help' button. The bottom section is titled 'Included in Save' and contains two columns of checkboxes: 'Plans' and 'Statistics'. Under 'Plans', there are checkboxes for 'Rule' (checked), 'First Row', 'All Rows', 'Hints1', 'Hints2', and 'Hints3'. Under 'Statistics', there are checkboxes for 'Hints1', 'Hints2', and 'Hints3'. At the very bottom, there is a section titled 'Plan in production' with two columns of radio buttons. The left column has radio buttons for 'Rule' (selected), 'First Rows', and 'All Rows'. The right column has radio buttons for 'Hints1', 'Hints2', and 'Hints3'.

Save SQL Dialog

- 2 The Save SQL dialog indicates which optimization plans and statistics will be saved.
- 3 In the **Folder** field, enter a name for the folder, or select an existing folder from the drop-down list
- 4 Enter a name in the **Short Name** field: this is a required field. The **Short Name** must be unique by user.
- 5 In the **Description** field, enter a description of up to 2000 characters.
- 6 In the **Included in Save** field, check the boxes for the Plans or Statistics you want to include in the save.
- 7 In the **Plan in production** field, specify which plan to use in production by selecting the appropriate radio button.
- 8 When you are finished completing the save information, click **OK**.

The Plan Analyzer Repository

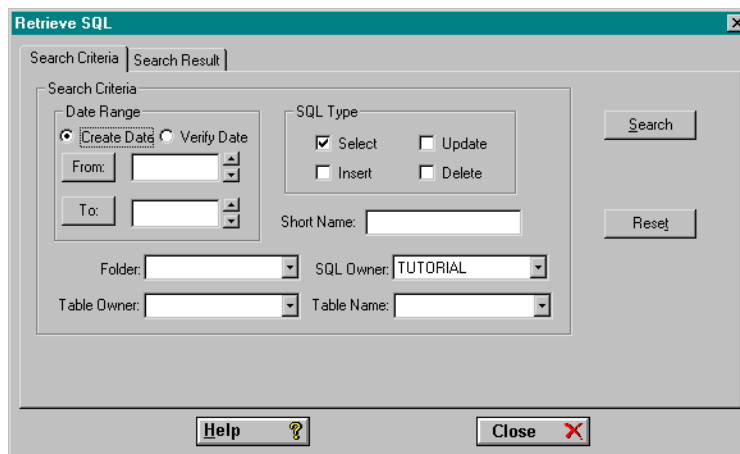
As the SQL, plans, and statistics are saved, Plan Analyzer also determines all tables that are directly or indirectly (via synonyms or views) referenced by the SQL statement, and saves those table references. Plan Analyzer can then use the saved table references to verify whether the optimization plans have changed.

The saved information is stored with the save date, the owner, and the security domain under which the SQL must be executed to reproduce the optimization plan (usually your login account). For more information about the **Database, Save to Repository** option, see the *Database Menu* chapter.

Retrieving

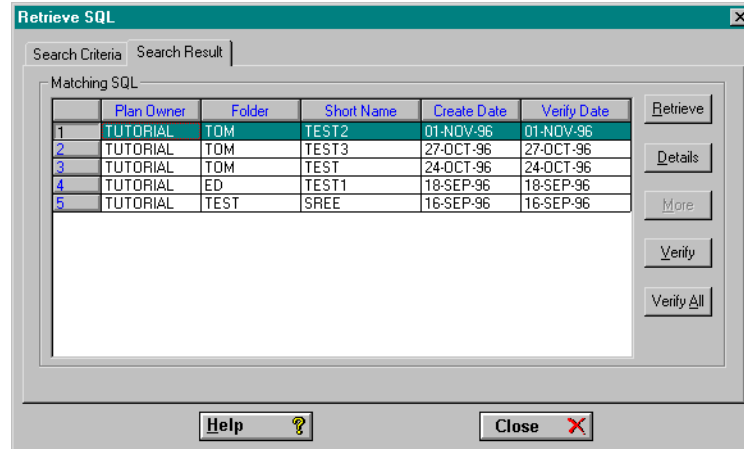
You can use the **Database, Retrieve from Repository** option to retrieve previously saved SQL, optimization plans, statistics, and tables referenced by the SQL. Follow these steps to retrieve SQL that has been saved:

- 1 Select **Database, Retrieve from Repository** from the Plan Analyzer window. The Retrieve SQL dialog appears:



Retrieve SQL Dialog

- 2 Specify search criteria in the appropriate sections (or leave all fields blank to retrieve all SQL). The most significant criterion to specify is the table that is directly or indirectly referenced by a SQL statement. Click **Search** to start the search. Plan Analyzer returns the matching SQL to the **Search Result** field, under the **Search Result** tab.



Search Result Tab of the Retrieve SQL Dialog

- 3 Select the desired row from the list of matching SQL, and click **Details**. The **SQL Details** dialog appears, displaying the SQL statement and all associated information, including the last verification date (for more information see *Verifying Optimization Plans* later in this chapter).
- 4 Click **Close** to return to the **Retrieve SQL** dialog, or **Retrieve** to retrieve the selected SQL, plans, and statistics.
- 5 To perform a new search with different criteria, you can click **Reset** from within the **Search Criteria** tab.

Note • If you re-explain the SQL or re-execute the SQL for performance statistics, Plan Analyzer tracks all changes and prompts you to save them.

If you change a SQL statement that was retrieved with all plans and statistics, Plan Analyzer invalidates all plans and statistics. This is because Plan Analyzer expects that the SQL change will produce a different set of optimization plans and performance statistics.

Deleting

To delete saved SQL and any associated information, use the **Database, Delete SQL in Repository** option.

Note • If the SQL currently displayed in the SQL window was retrieved from the Plan Analyzer repository, you cannot use the delete option on it. You must first clear the SQL from the Plan Analyzer SQL window, then select **Database, Delete SQL in Repository**.

The Delete dialog is similar to the Retrieve SQL dialog. Search for SQL in the same way you search in the Retrieve dialog. Once the matching SQL is displayed in the list box, click a row and then click **Delete** to delete the SQL and its associated parts.

Verifying Optimization Plans

Plan Analyzer lets you verify whether optimization plans have changed by querying the SQL and requesting the verification.

Knowing Which SQL to Query

Plan Analyzer stores the SQL with a list of the tables that are directly or indirectly referenced by the SQL statement. Indirect references result from the use of synonyms, views, views of views, views of synonyms, synonyms pointing to views, and so on. Production optimization plans can change over time for a variety of reasons and may cause significant changes in an application's performance. Sources of change can be:

- Creation or dropping of an index
- Re-analyzing a table or index with the SQL ANALYZE command

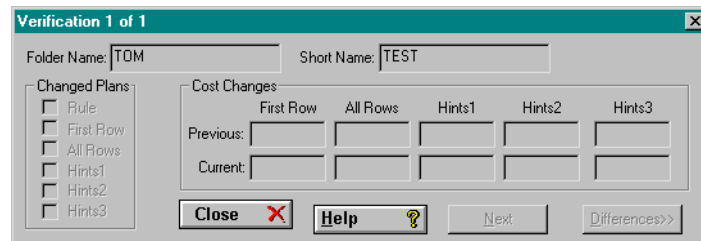
- A new version of ORACLE, which may have changes to the optimization algorithms

Since you know which table was analyzed, or which one had the index dropped or created, you should simply retrieve all SQL based on that table.

Changing Optimization Plans

To verify the state of an optimization plan, follow these steps:

- 1 Select **Database, Retrieve from Repository** from the menu. The Retrieve SQL dialog appears.
- 2 Click **Search**.
- 3 Select the name of the saved SQL you wish to verify, and click **Verify**. The Verification dialog appears:



Verification Dialog

The Verification dialog checks the boxes of the plans that have changed. The previous and current cost values are displayed.

The typical verification results from one or more tables being modified. Use **Retrieve from Repository** to retrieve the SQL based on those tables, one table at a time. Then verify each of the statements. For more information on verifying optimization plans, refer to the *Verify* section of the *Database Menu* chapter.

Real-Time SQL Capture

You can use Plan Analyzer to capture SQL from applications, either by monitoring SQL from current sessions or by capturing SQL from the shared SQL pool.

Capturing SQL

The Plan Analyzer **Capture** options allow you to monitor the SQL being executed in other applications while they are running. This simplifies the capturing of SQL. Monitoring is also useful when you do not have the source code and you want to know what SQL is being executed. As a DBA, you may be experiencing problems with system performance, and wish to identify which user is consuming the most resources. The Plan Analyzer SQL Capture facility is a substantial feature. You do not need to be a DBA to use the Capture facility. Plan Analyzer guarantees complete session-level security for all non-DBA users. The only restriction for non-DBA users is that they are limited to monitoring sessions logged in with the same ORACLE account name. A Plan Analyzer DBA can monitor any session.

For more information about monitoring and capturing SQL see the *Capture* chapter.

File and Edit Menus

This chapter discusses the options available from the File and Edit menu options.


File Menu	3-2
New SQL	3-2
Open SQL	3-2
Save SQL	3-3
Save SQL As	3-3
Save SQL & Plans	3-3
Print	3-3
Exit	3-4
Edit Menu	3-4
Undo	3-4
Cut	3-5
Copy	3-5
Paste	3-5
Clear	3-5
Find	3-5
Replace	3-6

File Menu

The **File** menu is available in both Standard and Expert modes. Its seven options are described below.

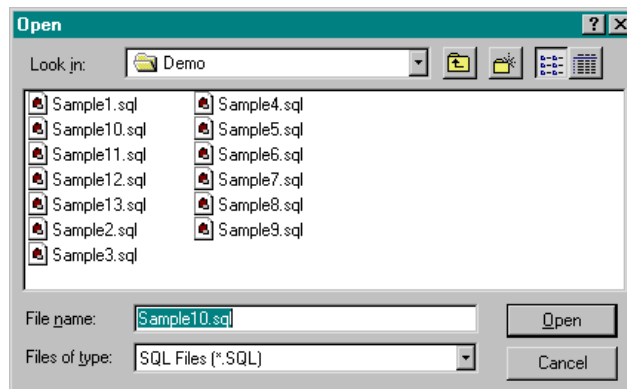
New SQL

Choose **File, New SQL** to clear the application. If there is unsaved work, Plan Analyzer displays the Save SQL dialog so that you can save the current work to a file. Click **Cancel** if you want to disregard changes and clear the windows without saving.

You can also invoke the **New SQL** menu option by clicking on the  toolbar icon.


Open SQL

Choose **File, Open SQL** to display the **Open** dialog. The dialog appears as follows:




Select a filename and click **Open** to display the contents of the file in the SQL window. The contents should include only a SQL statement. You can edit out any unwanted characters.

This option is intended to open only files with a single SQL statement. Plan Analyzer may not be able to open large files.

You can also invoke the **Open SQL** option by clicking the  toolbar icon.

Save SQL

Use the **Save SQL** option to save the contents of the SQL window to a file that has been opened or saved previously.

You can also invoke the **Save SQL** option by clicking the  toolbar icon.

Save SQL As

Use the **Save SQL As** option to save the contents of the SQL window to a new file.

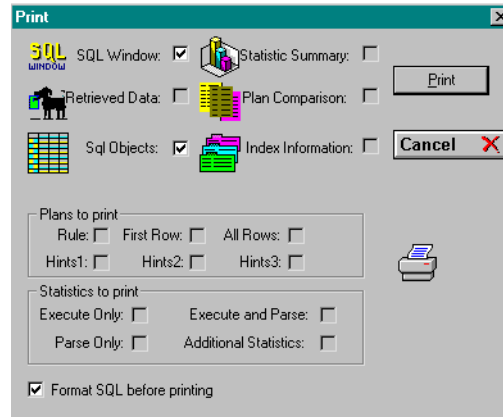
Save SQL & Plans

Save SQL & Plans saves the SQL and all explain plans to a file. The default suffix for the file is .XPL.

Print

Use the **Print** option to print the SQL, optimization plans, plan summary, execution statistics, statistics summary and data.

Edit Menu



You can choose which items to print. By default, all items that have been produced are printed. Click the checkbox next to each item to toggle it between selected and deselected for printing. The **Plans to print** and **Statistics to print** sections each have components that represent a different optimization mode.

You can also invoke the **Print** option by clicking on the  toolbar icon.

Exit

Exit disconnects from the database and quits the application. If there is unsaved work, the Save SQL dialog appears prior to exiting, prompting you to save your work to the repository.

Edit Menu


The **Edit** menu appears in both Standard and Expert modes. The seven **Edit** menu options are described below.

Undo

Undo reverses the last operation performed in the SQL window.

Cut

Cut copies the highlighted text into a buffer and deletes it from the SQL window. The text can later be pasted elsewhere.

You can also invoke the **Cut** option by clicking on the  toolbar icon.

Copy

Copy places the highlighted text into a buffer which can then be referenced by the **Paste** menu item.

You can also invoke the **Copy** option by clicking on the  toolbar icon.

Paste

Paste inserts the text from the buffer at the cursor position. You can use either **Copy** or **Cut** to place text in the buffer.

You can also invoke the **Paste** option by clicking on the  toolbar icon.

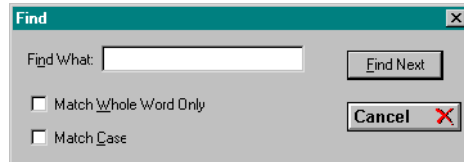
Clear

Clear blanks out all windows in the application, and resets the performance statistics and bind variable values. If unsaved work is present when **Clear** is invoked, the Save SQL dialog appears, allowing you to save the current work before clearing.

Find

The **Find** option searches for text strings in the SQL window. The Find dialog is shown in the following figure:

Edit Menu

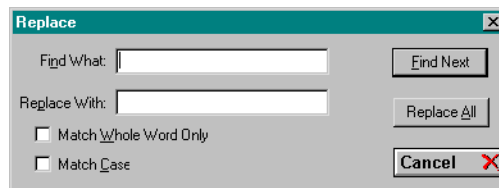


Enter the string to search for in the **Find What** field, and click **Find Next** to start searching from the current cursor position in the SQL window. After the first match is found, the following dialog displays with the option to either **Find Next** or **Cancel**.



Replace

The **Replace** option lets you find and replace a string in the SQL window.



Replace lets you specify a string in the **Replace With** field that will replace the found item when you click **Replace**. Clicking **Replace All** replaces all occurrences of the specified string with the string in the **Replace With** field.

If you click **Find Next**, the first occurrence is found and highlighted, and the following dialog displays:



Database Menu

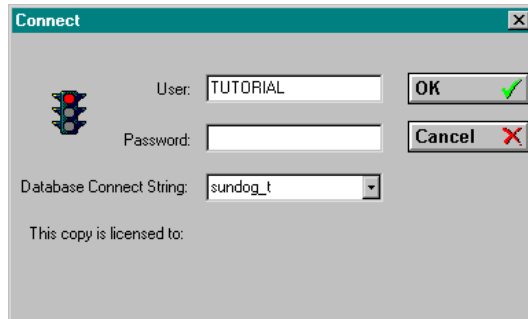
The Database menu item appears in both Standard and Expert modes. This chapter describes the nine Database menu options.

Connect	4-3
Disconnect	4-3
Save to Repository	4-3
Retrieve from Repository	4-5
Search Button	4-8
Reset Button	4-9
Retrieve Button	4-9
Details Button	4-9
More Button	4-10
Verify Button	4-10
Verify All Button	4-12
Delete SQL in Repository	4-13
All Objects	4-13
Table Information	4-14
Index Information	4-18

Trigger Information	4-19
View Information	4-20
Procedure Information	4-22
Function Information	4-22
Package Information	4-22
Sequence Information	4-22
Cluster Information	4-23
Object Dependencies	4-23
Refresh Button	4-24
Create Index	4-25
Maintenance	4-29
Correcting Errors	4-30
Admin	4-31

Connect

Connect to Database is enabled on the **Database** menu only when the application is not connected to the database. Choose **Database, Connect to Database** to display the Connect dialog.



Connect Dialog

Enter an Oracle account name into the **User** field, and its password into the **Password** field. As you enter your password, the characters are displayed as asterisks. Finally, enter the database connect string in the **Database Connect String** field, or select it from the combo box which lists the database connections contained in the **PAFO32.INI** file. If you are connecting to a local database, or if **LOCAL** is set in the **ORACLE.INI** file, you can leave the **Database Connect String** field blank. When you have entered the required information, click **OK** to login to the database.

Disconnect

To disconnect from the database without exiting the application, click **Database, Disconnect from Database**. This option is useful when moving between different Oracle accounts or databases.

Save to Repository

You can use the **Save to Repository** option to save the following items to the database repository:

Save to Repository

- SQL statements
- Optimization plans
- Statistics
- Any bind variable values used in executing the SQL statement

Select **Database, Save to Repository** to display the Save SQL dialog:

The screenshot shows the 'Save SQL' dialog box. It has a title bar with 'Save SQL' and a close button. The dialog contains the following elements:

- Folder:** A text box with a dropdown arrow.
- Short Name:** A text box.
- Description:** A large text area.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons are located to the right of the input fields.
- Included in Save:** A section with two columns: 'Plans' and 'Statistics'.

	Plans	Statistics
Rule:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
First Row:	<input type="checkbox"/>	<input type="checkbox"/>
All Rows:	<input type="checkbox"/>	<input type="checkbox"/>
Hints1:	<input type="checkbox"/>	<input type="checkbox"/>
Hints2:	<input type="checkbox"/>	<input type="checkbox"/>
Hints3:	<input type="checkbox"/>	<input type="checkbox"/>
- Plan in production:** A section with radio buttons.

<input checked="" type="radio"/> Rule	<input type="radio"/> Hints1
<input type="radio"/> First Rows	<input type="radio"/> Hints2
<input type="radio"/> All Rows	<input type="radio"/> Hints3

Save SQL Dialog

The optimization plans and execution statistics are each represented by six checkboxes in the **Save SQL** dialog. Plan Analyzer automatically sets the checkbox for each item generated. For instance, if you generated only the Rule optimization plan and no execution statistics, the dialog would look like the one in the above figure. If you choose to save execution statistics, any bind values that were used are also saved.

Before saving the items, you must provide a unique folder and short name. The folder and short name are a means of organizing your SQL and plan in the Plan Analyzer repository. Type a name in the **Folder** field, or click the combo box to list current folders owned by the current Oracle account name. If you enter a new folder, Plan Analyzer will list the folder in the combo box on subsequent saves.

Note • If at some point all items associated with a specific folder are deleted, Plan Analyzer automatically drops the folder.

A suggestion is to choose folder names that identify the application the SQL is embedded in. Besides the folder and short name, you can optionally add a description for future reference.

When saving the SQL, you can also specify which plan to use in production by selecting the appropriate radio button in the **Plan in production** group box.

Retrieve from Repository

You can use the **Database, Retrieve from Repository** option to retrieve SQL and associated data which has been saved with the **Save to Repository** option. The Retrieve SQL dialog lets you specify what types of SQL statements to retrieve.

Retrieve from Repository

The Retrieve SQL Dialog

If there is unsaved work when **Retrieve from Repository** is invoked, the **Save SQL** dialog is displayed, giving you an opportunity to save the work.

The possible criteria for retrieving SQL, enclosed in the **Search Criteria** group box, include:

Component	Description
-----------	-------------

Date Range	There are two different dates associated with a SQL statement: the date the SQL was originally saved or last modified, and the date the optimization plans for the SQL were last verified (see <i>Verify</i> in this chapter).
-------------------	--

To specify a date criterion, first select **Create Date** or **Verify Date**. Then click in the part of a date that you want to change and use the up and down arrow keys to increase/decrease it. Alternatively, click **From** or **To**.

SQL Type	Specifies the type of SQL statements to search for. Click the checkboxes to toggle the type. Select is the default type.
-----------------	---

Component	Description
Short Name	Gives the name assigned to the SQL statement. All ORACLE SQL wild card characters are permitted (for example, percent sign or underscore).
Folder	Identifies the folder the SQL was placed in. Click the combo box to list all folders associated with the Oracle account listed in the SQL Owner field.
Table Owner	Queries SQL based on a table that is directly or indirectly referenced by the SQL statement. The table owner must be selected before the table name can be selected.
SQL Owner	Identifies the Oracle user who owns the SQL. For non-DBAs, this field lists only the current login account name. Plan Analyzer DBAs can click the combo box to list the names of all Oracle accounts that have SQL stored in the Plan Analyzer repository.
Table Name	Specifies which tables the user can access for the specified table owner.

The most significant criterion to specify is the table that is directly or indirectly referenced by a SQL statement. After numerous SQL statements have been stored in the Plan Analyzer repository, retrieving the SQL by the short name becomes less likely. Using a combination of folder name and the table referenced by the SQL statement is much more likely to be successful. Querying by table name should be the only criteria used when retrieving SQL to verify the optimization plans.

Click the **To** or **From** button to set the associated date using a calendar (see figure below).

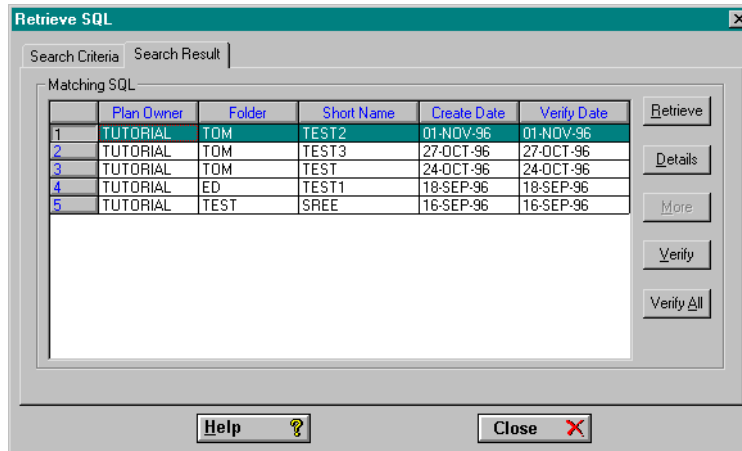
Retrieve from Repository



Calendar Dialog

Search Button

After specifying the search criteria, click **Search** to perform the query. Plan Analyzer returns the matching SQL, and displays it in the Search Results window.



If no matching SQL is found, a dialog displays the message **No Match Found**.

Reset Button

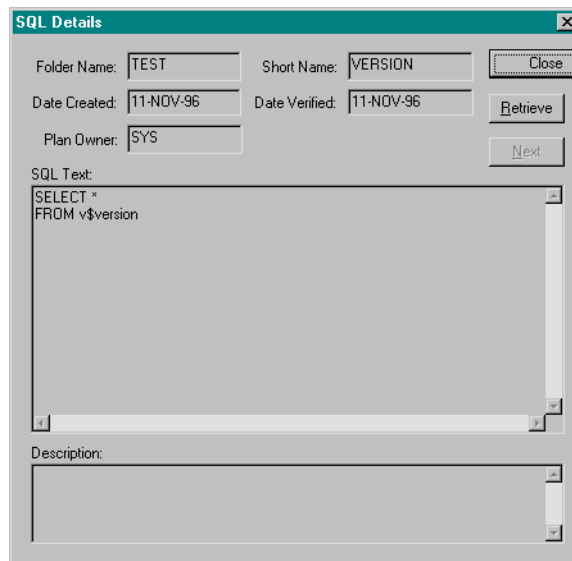
If you wish to use different search criteria, clear the selected criteria by clicking **Reset**. **Reset** clears the table name, table owner, short name, and the date range, and sets the **SQL Type** to **Select**.

Retrieve Button

Click **Retrieve** to retrieve the SQL and associated parts for the row highlighted in the **Matching SQL** list box. You can also double-click the highlighted row to retrieve the data.

Details Button

You can click the **Details** button to display the SQL statement and a description in the SQL Details dialog:



SQL Details Dialog

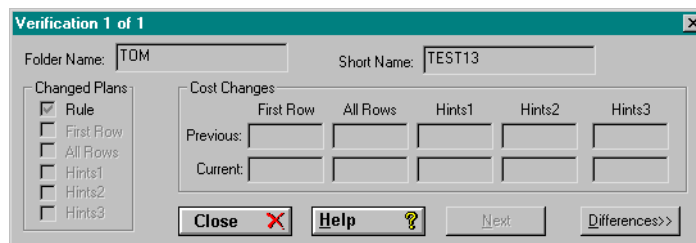
More Button

When you click **Search**, Plan Analyzer displays the first 15 rows of the result set. To see the next 15 rows, click **More**.

Verify Button

Production optimization plans can change over time for a variety of reasons and may cause significant changes in an application's performance. The plans stored in the Plan Analyzer repository should represent the plans at the time the SQL was placed into production. Since Oracle 7 dynamically creates the plans each time the SQL is parsed, a variety of factors can change the plans over time. One of the most common sources of change is the creation or dropping of an index. Re-analyzing a table or index with the SQL ANALYZE command may also change plans, since new or different statistics may become available to Oracle. Additionally, new Oracle versions generally have changes to the optimization algorithms.

Use the **Verify** button to determine whether any of the plans stored in the Plan Analyzer repository have changed. To determine if the optimization plans have changed since they were last saved, highlight a row in the **Matching SQL** list box, and click **Verify** to retrieve all optimization plans for the SQL. The Verification dialog identifies which plans have changed by setting the corresponding checkbox to true.



Verification Dialog

The **Cost Changes** section shows the previous and current costs for the stored optimization plans. The above figure shows that only the Rule optimization plan was stored for the SQL statements, and that the plan has changed. The **First Row**, **All Rows**, and **Hints** checkboxes in the **Changed Plans** group box are disabled to indicate that the plans are not even stored in the repository. If cost-based plans, such as First Row, All Rows, or Hints, were stored in the repository for the SQL, the previous and current costs for the cost-based optimizations are listed in the **Cost Changes** section.

Differences

If the plans have changed, the **Differences** button is enabled. Click **Differences** to expand the Verification dialog. This displays information that will help you understand the changes in the plans. Note that it is possible that the costs have changed but not the plans.

The screenshot shows the 'Verification 1 of 1' dialog box. It contains several sections:

- Folder Name:** TOM
- Short Name:** TEST13
- Changed Plans:** A group box with checkboxes for Rule (checked), First Row, All Rows, Hints1, Hints2, and Hints3. The latter four are disabled.
- Cost Changes:** A table showing previous and current costs for Rule, First Row, All Rows, Hints1, Hints2, and Hints3. The Rule row shows a change from 0 to 1.
- Plan Differences for:** A group box with radio buttons for Rule (selected), First Rows, All Rows, Hints1, Hints2, and Hints3.
- List Database Objects in:** A group box with radio buttons for Current plan but not in previous plan and Previous plan but not in current plan (selected).
- Table:** A table with columns Operation, Object Owner, and Object Name. It shows one row: INDEX, HR2, I_EMP_NAME.
- Full Table Scans in:** A group box with text boxes for Previous Plan (0) and Current Plan (1).
- Sorts in:** A group box with text boxes for Previous Plan (0) and Current Plan (0).
- First Step in:** A group box with text boxes for Previous Plan (INDEX-RANGE SCAN-I_EMP_NAME) and Current Plan (TABLE ACCESS-FULL-EMPLOYEES).

Buttons at the bottom include Close, Help, Next, and Differences>>.

Verification Dialog (Expanded)

The **Plan Differences** for group box indicates which plan differences are being viewed. It is important to check for changes in the first step of the plan. If the plans have changed, then it is likely that the first step has changed. Other changes in the plans that may result in dramatically different performance are the number of full table scans and sorts in the current and previous plans. Check the **Full Table Scans in** and **Sorts in** boxes.

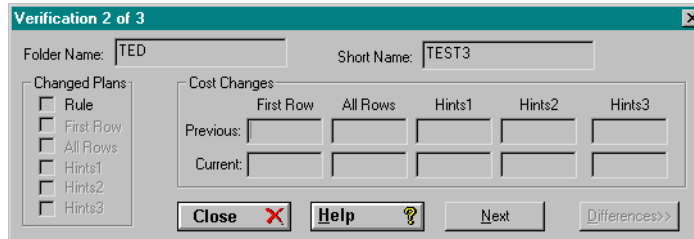
One last potential difference is among the objects in the different plans. If a new index was added, causing the plans to change, then most likely that new index will appear in the current plan, whereas the index that it replaced will appear in the previous plan. To view objects in the current plan that did not appear in the previous plan, set the radio button labeled **Current plan but not in previous plan**. For the reverse, set the radio button labeled **Previous plan but not in current plan**.

When Do You Verify?

Verify is one of the most useful options available to the DBA or developer. You can use **Verify** to determine the impact of database changes, rather than waiting for users to complain. Whenever an index is created or dropped, use **Verify** for all SQL which references the table with the index. If a table is re-analyzed, statistics change and so do the cost-based optimization plans. After re-analyzing a table, the SQL referencing the re-analyzed table should be verified for plan changes.

Verify All Button

The **Verify All** button performs the same function as the **Verify** button, but it also performs verification for all SQL satisfying the criteria specified on the search. Therefore, if multiple SQL statements match the criteria, each one will be verified. The Verification dialog displays a **Next** button, which you can use to proceed to the next verification.



Verification Dialog After Selecting Verify All

Delete SQL in Repository

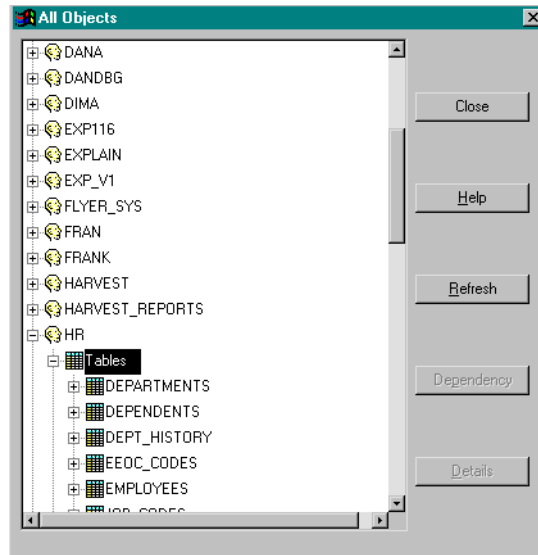
Select **Database, Delete SQL in Repository** to display the Delete SQL dialog, which is very similar to the Retrieve SQL dialog. Search for SQL in the same way that you search in the Retrieve SQL dialog. Once the matching SQL is displayed in the list box, click a row and then click **Delete** to delete the SQL and its associated parts.

To display the details of the SQL prior to deleting, click **Show Details**.


All Objects

The All Objects dialog displays all Oracle accounts owning database objects that are accessible to the user.

All Objects



All Objects Dialog

Each Oracle account name is preceded by a head icon, . A plus sign next to the account indicates that there are further levels in the hierarchy. Click an account with a plus sign to expand the hierarchy one more level. For instance, the above figure shows the HR account after clicking the HR account name.

Note • If, in the Preferences dialog, Enable Caching is set, the All Object dialog may become out of date for a dynamic schema. Use the Refresh button to periodically read the server for any object changes that occurred after All Object was invoked. For more information, see the Preferences section of the View chapter.

Table Information

To list the tables owned by an Oracle account, double-click the Tables folder icon. Single-click a table to enable the **Dependency** and **Details** buttons. The **Dependency** button lists the database objects that are

dependent on the highlighted table (see the *Object Dependencies* section in this chapter). The **Details** button displays the Table Object dialog with a complete definition of the table:

The dialog box titled "Table Object: JOB_HISTORY" contains the following configuration options and table:

Full Table Scans:

- Default Parallel Servers: 1
- No. of instances split across: 1
- Cached by default? ☒ Yes ☐ No

Transactions/Data Block:

- Initial: 0
- Maximum: 0

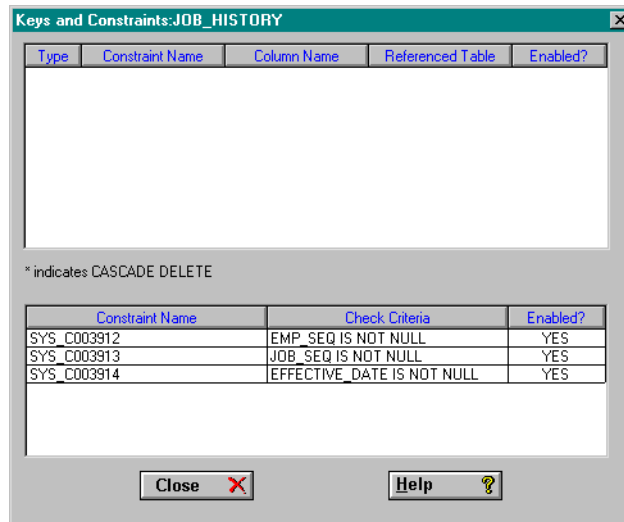
	Column Name	Data Type	Default	Null ?
1	EMP_SEQ	NUMBER		NO
2	JOB_SEQ	NUMBER		NO
3	EFFECTIVE_DATE	DATE		NO

At the bottom, there are buttons for **Keys/Checks**, **Privileges**, **Cluster**, **Close** (with a red X icon), and **Help** (with a question mark icon).

Table Object Dialog

Keys/Checks Button

The **Keys/Checks** button on the Table Object dialog displays the Keys and Constraints dialog.



Keys and Constraints Dialog

The top spreadsheet in the dialog lists the primary and foreign key referential constraints. The bottom spreadsheet lists the check constraints. The most important constraints to view are the referential constraints, to assist you in properly joining the tables.

Privileges Button

The **Privileges** button on the Table Objects dialog invokes the Table Privileges dialog:

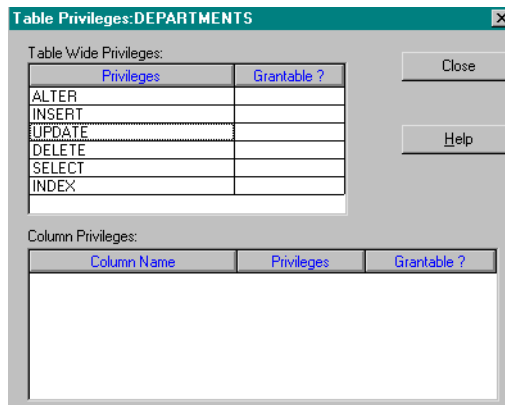
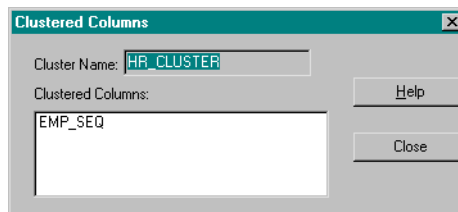


Table Privileges Dialog

The Table Privileges dialog displays the object privileges, system privileges, and role privileges that the user has on the table. Check the Privileges dialog if security errors occur while you are attempting to produce optimization plans for the SQL statement.

Cluster

If the table is part of a hash or indexed cluster, the **Cluster** button is enabled in the Table Object dialog. Click **Cluster** to display the Clustered Columns dialog, which lists the name of the cluster and the columns in the table mapped to the cluster key. The columns appear in the order defined in the cluster key.



Clustered Columns Dialog

Full Table Scans

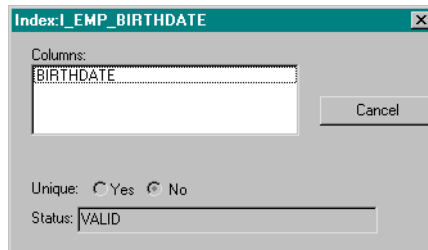
At the top of the Table Object dialog is a group box labeled **Full Table Scans** which contains information regarding the parallel execution and caching. If a table was defined to be queried in parallel, the **Default Parallel Servers** field identifies the number of servers that will be used to perform a full table scan of the table. If the Oracle server is part of a clustered environment, then **No. of instances split across** specifies the number of nodes in the cluster that will participate in the parallel operation. If a full table scan is performed on a table, the data blocks can either be placed at the beginning or at the end of the LRU (Least Recently Used) chain. If **Cached by default** is set to Yes, then the blocks will be placed at the end of the LRU chain, a potential benefit in subsequent queries of the table.

Transactions/Data Block

These are the initial number and maximum number of transactions that can occur concurrently in a single datablock of the table. If the table is clustered, then both values are zero.

Index Information

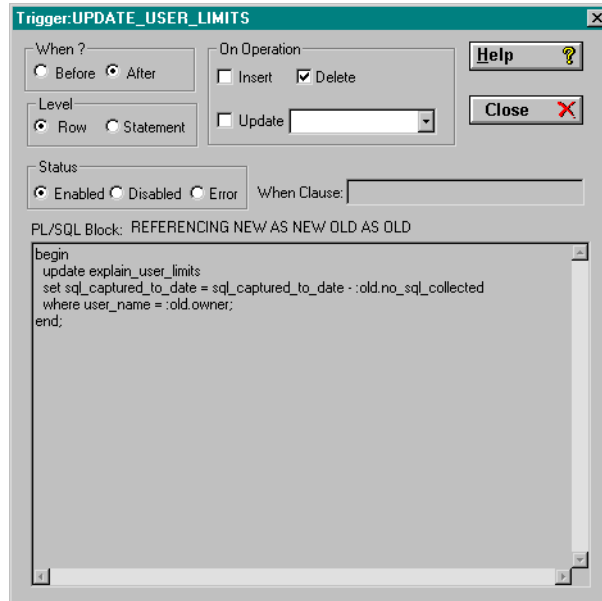
From the All Objects dialog, double-click the **Indexes** category under a specific table to list all indexes created on the table. Clicking an index name once, or double-clicking it, invokes the Index dialog. The dialog lists the columns comprising the index in the order specified in the index definition. It also lists whether or not the index is unique and whether the index is still valid.



Index Dialog

Trigger Information

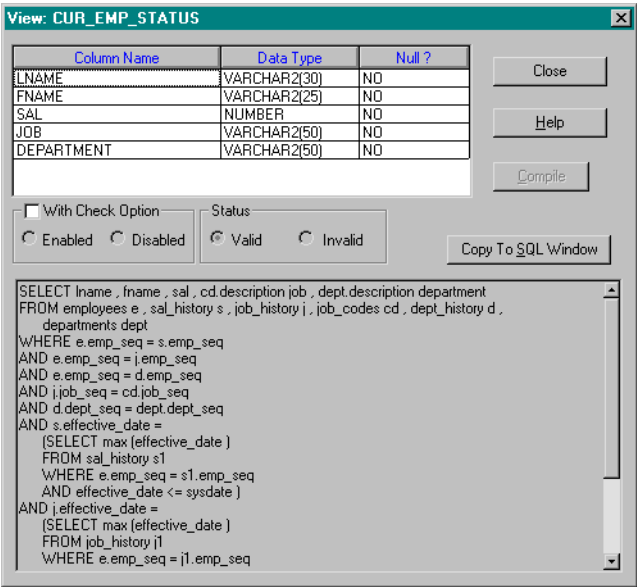
From the All Objects dialog, double-click the Triggers category under a specific table to list all triggers created on the table. Clicking a trigger once, or double-clicking it, displays the Trigger dialog. The performance of SQL INSERT, DELETE and UPDATE statements is affected by the existence of triggers. The following figure illustrates the Trigger dialog for one of Plan Analyzer's triggers.



Trigger Dialog

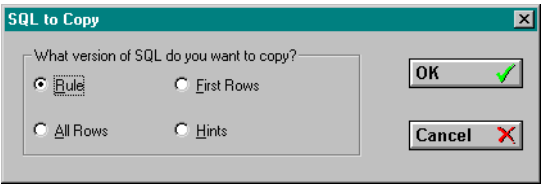
View Information

From the All Objects dialog, double click the Views icon to list the views owned by an Oracle account. Click a view name to enable the **Dependency** and **Details** buttons. The **Dependency** button lists the database objects that are referenced by the highlighted view (see the *Object Dependencies* section below). The **Details** button displays the View dialog with a complete definition of the view. The following figure illustrates the View dialog.



View Dialog

Since views are often accessed within SQL statements, the SQL definition may need to be optimized separately from the SQL referencing the view. To do this, click **Copy To SQL Window** to copy the SQL statement to the SQL window for optimization. After the SQL statement is optimized, the **Copy To SQL Window** label in the View window changes to **Paste from SQL Window**. Click **Paste from SQL Window** to display the SQL to Copy dialog, which prompts for the version of the SQL to paste back.



SQL to Copy Dialog

If the Rule version was the optimal version, select Rule and the SQL is replaced with a version containing the RULE hint. After replacing the SQL, click **Compile** in the **View** window to recompile the view definition.

Procedure Information

To list the procedures owned by an Oracle account, double-click the Procedures icon in the All Objects window. Click a procedure name to enable the **Details** and **Dependency** buttons. Double-click the procedure name or highlight it and click **Details** to display the Procedure dialog. The Procedure dialog lists the complete definition of the procedure.

Function Information

To list the functions owned by an Oracle account, double-click the Functions icon in the All Objects window. Click a function name to enable the **Details** and **Dependency** buttons. Double-click the function name or highlight it and click **Details** to display the Function dialog. The Function dialog lists the complete definition of the function.

Package Information

To list the packages owned by an Oracle account, double-click the Packages icon in the All Objects window. Click a package name to enable the **Details** and **Dependency** buttons. Double-click the package name or highlight it and click **Details** to display the Package dialog. The Package dialog lists the complete definition of the package.

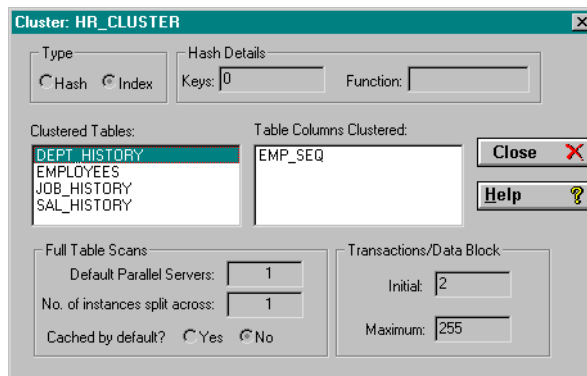
Sequence Information

To list the sequences owned by an Oracle account, double-click the Sequence icon in the All Objects window. Click a sequence name to enable the **Details** button. Double-click the sequence name or highlight it and click **Details** to display the Sequence dialog. The Sequence dialog lists the complete definition of the sequence.

Cluster Information

To list the clusters owned by an Oracle account, double-click the Cluster icon in the All Objects window. Single-click a cluster name to enable the **Details** button. Double-click the cluster name or highlight it and click **Details** to display the Cluster dialog. The Cluster dialog lists the complete definition of the cluster.

The Cluster dialog in the following figure shows one of the demonstration clusters. All tables that are part of the cluster are listed in the **Clustered Tables** box. Click a table to list the columns in the table that are part of the cluster key.

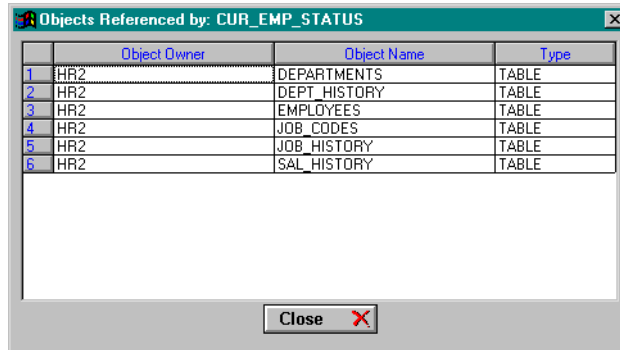


Cluster Dialog

Object Dependencies

The **Dependency** button in the All Objects dialog displays one of two dialogs, depending on the type of database object that is highlighted in the hierarchy.

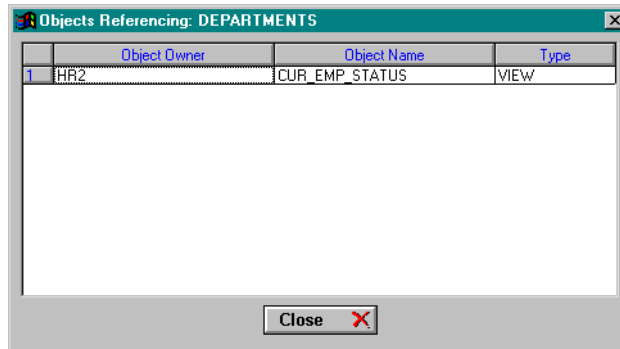
- If the highlighted object is a table, the Objects Referencing dialog displays all the other database objects that reference the table.



	Object Owner	Object Name	Type
1	HR2	DEPARTMENTS	TABLE
2	HR2	DEPT_HISTORY	TABLE
3	HR2	EMPLOYEES	TABLE
4	HR2	JOB_CODES	TABLE
5	HR2	JOB_HISTORY	TABLE
6	HR2	SAL_HISTORY	TABLE

Objects Referencing Dialog (Table)

- If the highlighted object is a view, procedure, function, or package, the Objects Referenced By dialog displays all the database objects that they reference.



	Object Owner	Object Name	Type
1	HR2	CUR_EMP_STATUS	VIEW

Objects Referenced By Dialog (View)

Refresh Button

When you invoke **Database, All Objects**, Plan Analyzer caches the retrieved account information into memory. This means that, as the underlying objects are changed, they may not be reflected accurately in

the cached **All Objects** display. You can use the **Refresh** button on the All Objects dialog to periodically re-read the account information into Plan Analyzer.

Create Index

The Create Index feature prompts you for all the parameters necessary to create an index on an object for which you have CREATE INDEX privileges.

The screenshot shows the 'Create Index' dialog box with the following details:

- Owner:** HR2
- Index Name:** (empty)
- Unique:** ☐
- On:**
 - Schema:** HR2
 - Table:** EMPLOYEES
 - Cluster:** (empty)
- Available Indexes:** I_EMP_BIRTHDATE(NONUNIQUE)
- Index Columns:** BIRTHDATE
- Table Columns:** EMP_SEQ, LNAME, FNAME, MNAME, BIRTHDATE (selected), HIREDATE, SEX, EEOC
- Index Columns:** SSN
- Buttons:** Add >>, Remove <<, Create Index, Cancel

The Create Index Window

In the **Columns** tab, specify the Owner of the new index. Give the index a name in the Index Name field. Check the Unique checkbox if you want to create a unique index.

In the **On** groupbox select the schema for the Table or Cluster you want to create the index on. Select an object from either the Table or Cluster combo boxes. These lists show all the available tables or clusters in the

Create Index

schema you've selected. **Available Indexes** shows the indexes that already exist on the table or cluster you've selected, and **Index Columns** displays the columns that are part of the selected index.

Use the **Table Columns** and **Index Columns** fields below to move selected columns from one field to another. The **Table Columns** field shows all the columns in the table. The **Index Columns** field shows the columns that will make up the new index. Select the columns you want and click either **Add** or **Remove** to place the column in the desired field. When you are satisfied with your selection, click the **Create Index** button at the bottom of the window.

Click the **Other** tab to specify other parameters for your index.

The screenshot shows the 'Create Index' dialog box with the 'Other' tab selected. The dialog has three tabs: 'Columns', 'Other', and 'Generated DDL'. The 'Other' tab contains the following fields and options:

- PCTFREE:** A text input field.
- INITRANS:** A text input field.
- MAXTRANS:** A text input field.
- Table Space:** A dropdown menu currently showing 'USER_DATA'.
- No Sort:** An unchecked checkbox.
- Recoverable:** An unchecked checkbox.
- Storage:** A text input field containing 'STORAGE (INITIAL 10K NEXT 10K MINEXTENTS 1 MAXEXT'.
- Parallel:** A text input field.

At the bottom of the dialog are two buttons: 'Create Index' and 'Cancel'.

The Create Index Other Tab

In the **PCTFREE** field, specify the portion of a data block reserved for later updates to the rows in that block. **PCTFREE** is the portion of the data block that is not filled by rows as they are inserted into the table. The default value is 10, and the minimum value is 0.

INTRANS tells the number of transactions that can update a data block concurrently. The value can range from 1 to 255. The default value is 1. When more than **INTRANS** transactions are created for the block, space is automatically allocated for them, up to the value of **MAXTRANS**.

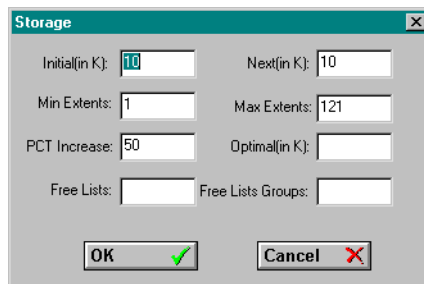
MAXTRANS is the maximum number of transactions that can concurrently update a data block. The values for **MAXTRANS** can range from 1 to 255, the default being 255.

Choose the **Table Space** allocation from the combo box.

Select the **No Sort** option to bypass the database's sort when searching the index. This is applicable only when the rows of the index are in ascending order.

Check **Recoverable** to log the creation of the index in the redo log file.

Clicking the **Storage** button brings up the **Storage** dialog:

The image shows a 'Storage' dialog box with a title bar and a close button. It contains several input fields: 'Initial(in K):' with a value of 10, 'Next(in K):' with a value of 10, 'Min Extents:' with a value of 1, 'Max Extents:' with a value of 121, 'PCT Increase:' with a value of 50, 'Optimal(in K):' which is empty, 'Free Lists:' which is empty, and 'Free Lists Groups:' which is empty. At the bottom, there are 'OK' and 'Cancel' buttons, each with a small icon (a green checkmark for OK and a red X for Cancel).

The Storage Dialog

Initial allocates the first extent of space to the object. The default value, (in Kilobytes) is 5. The smallest initial extent you can allocate is two data blocks, and the largest value is system dependent.

Next refers to the size of the extent allocated after the initial extent has been filled. The default if unspecified is five data blocks. The smallest extent you can allocate is one data block, and the largest is operating-system dependent.

If **Min Extents** is not specified it defaults to 2 for a rollback segment. This means that when the object is created, only the initial extent is allocated. A number larger than 2 will create that many more total extents, and the size of the extent will be determined by the values set with **Initial**, **Next**, and **PCT Increase**. All of these will be allocated when the object is created.

Max Extents establishes a limit on the total number of extents that can be allocated. The minimum limit is 1, and the default and maximum are operating-system dependent.

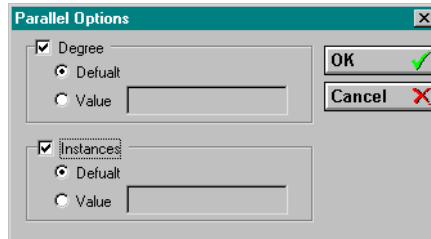
PCT Increase controls the rate of growth of extents beyond the second extent. If set to 0, every additional extent will be the same size as the second extent, specified by **Next**. If **PCT Increase** is a positive integer, each succeeding extent will be that percentage larger than the previous one.

Optimal sets an optimal size for a rollback segment. Oracle will dynamically deallocate extents in the rollback segment to maintain the optimal size. NULL means that Oracle never deallocates the rollback segment extents - this is the default behavior. You must supply a size greater than or equal to the initial space allocated for the rollback segment by the **Min Extents**, **Initial**, **Next** and **PCT Increase** parameters.

Free Lists sets the number of free lists for each free list group.

Free Lists Groups gives the number of free lists, with a default of 1. This setting applies to the Parallel Server option of Oracle7 for indexes.

Clicking the Parallel button from the **Other** tab brings up the Parallel options dialog:



The Parallel Options Dialog

Here you specify the number of possible concurrent mounts on the database. Check the **Degree** button to specify the number of servers used in the parallel operation. Check the **Instances** button to indicate the number of parallel server instances.

The Generated DDL tab shows the SQL code generated in the creation of the index.

Maintenance

Plan Analyzer uses numerous Oracle performance statistics to measure the resources consumed when a SQL statement executes. Note that these statistics may change between versions of Oracle, resulting in invalid performance statistics. Therefore, when the Plan Analyzer repository is first installed, a check is performed to ensure that statistics correspond to those Plan Analyzer expects. If they are as expected, the Oracle version is recorded in the repository. Thereafter, each time Plan Analyzer is run, a check is performed to ensure that the current Oracle version is the same as the version stored in the repository. If it is not, a check is performed to ensure that the statistics are still correct. If any errors are found in the statistics, a PL/SQL procedure is executed to correct the errors. If the procedure cannot resolve all the errors, the Database Statistics Maintenance dialog (see figure below) displays. The top spreadsheet in the dialog displays the known errors that must be corrected by an Oracle DBA.

Database Statistics Maintenance

Current Plan Analyzer Repository

☐ Display All Statistics ☒ Display known errors only

	Original ID	Original Name	New ID	New Name
1	69	table scans (short tables)		
2	70	table scans (long tables)		
3	71	table scan rows gotten		
4	72	table scan blocks gotten		
5	77	parse time cpu		
6	93	table fetch continued row		
7	98	parse count		
8	99	execute count		

Description:

Full table scans of small tables where the size limit is determined by the INIT.ORA parameter SMALL_TABLE_THRESHOLD.

Save Cancel Help

Find Possible Replacement

	ID	Name
1	46	DBWR lru scans
2	74	transaction tables consistent reads - un
3	75	transaction tables consistent read rollb
4	86	table scans (short tables)
5	87	table scans (long tables)
6	88	table scans (rowid ranges)

☐ Display All ☒ Substring Match

Apply Replacement

Database Statistics Maintenance Dialog

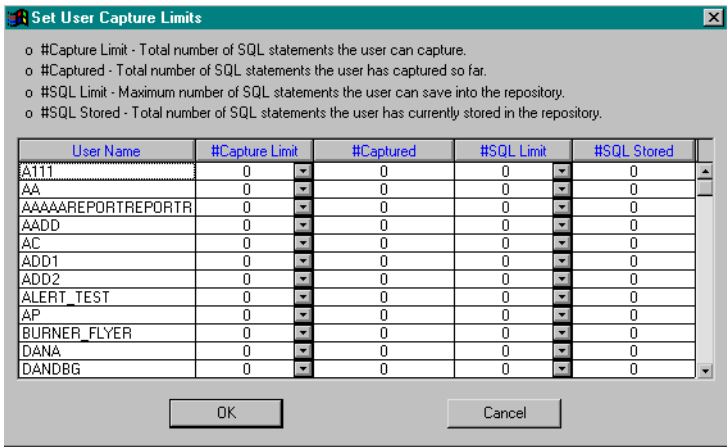
Correcting Errors

The statistics that are in error appear in the Original ID and Original Name columns of the top spreadsheet. To correct one of the statistics, highlight the statistic in the top spreadsheet and then find a replacement in the lower spreadsheet. Highlight the replacement and click **Apply Replacement**. The replacement statistic ID and name are then displayed in the New ID and New Name fields. After you have replaced all the statistics that are in error, click **Save** to save the changes.

To assist in the search for a replacement, a description of the highlighted statistic appears below the top spreadsheet. This description should only be used if the replacement is not obvious. Generally the best way to find a replacement is to simply choose a substring search by clicking the **Substring Match** radio button. All current statistics that have a substring in the name matching one in the highlighted statistic are displayed. Typically all errors are easily resolved with the substring match.

Admin

The Admin function allows DBAs to limit the amount of SQL users can capture and store. This is helpful if storage space is an issue. The following figure illustrates the **Set User Capture Limits** dialog, which you invoke by selecting the Admin menu item:



Set User Capture Limits Dialog

The **#Capture Limits** column shows the maximum amount of SQL the selected user can capture. The **#Captured** column shows the number of SQL statements the user has captured currently. The **#SQL Limit** column displays the maximum number of SQL statements the user may save to the repository. The **#SQL Stored** column shows the number of SQL statements the user currently has stored in the repository.

To change the value in either of the **Limits** columns, click once on the row in the column you want to modify, and enter a value. Or, you can select **Unlimited** from the combo box, to allow the user unlimited save and/or capture ability.

SQL Menu

This chapter describes the SQL menu item, which appears in both Standard and Expert modes with five options.

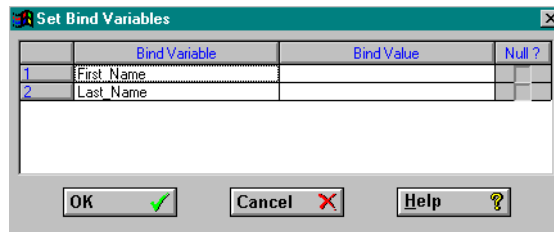
Set Bind Variable	5-2
Copy SQL to Clipboard	5-4
Format	5-5
DB Objects Analysis	5-6
Analysis Dialog Statistics	5-7
Retrieve Data	5-24

Set Bind Variable

Bind variables are the placeholders for constants that are passed prior to execution but after the parse. In order to test the performance of a SQL statement, all bind variables must have values entered. The following SQL statement has two bind variables:

```
SELECT * FROM employees
WHERE lname = :Last_Name
AND fname = :First_Name
```

When you select **SQL, Set Bind Variable** (or when the Server Statistics dialog is requested for a statement with Bind Variables), the Set Bind Variables dialog displays:



Set Bind Variables Dialog

The **Bind Variable** column lists the names of the bind variables. The bind variable can have either a NULL or a non-NULL value. A non-NULL value must be entered in the **Bind Value** column. To enter a NULL value, set the check box in the column labeled **Null**.

Non-SELECT SQL statements also permit bind variables, but you can specify an array of values for each bind variable. If an array of values is used, the same number of values must be entered for each bind variable in the SQL statement. You can then test array operations that are unique to Oracle (see *Sequential vs. Array Execution* in the *Performance Menu* chapter for more details).

To accommodate the array feature, the Set Bind Variables dialog is modified for INSERT, DELETE and UPDATE statements. The following figure illustrates the dialog with bind variables listed for the following SQL INSERT statement.

```
INSERT INTO employees (emp_seq, lname, fname, mname)
VALUES (:Empno, :Last_Name, :First_Name, :Mid_Initial)
```

The dialog box titled "Set Bind Variables" has a field "No. of values per bind variable:" set to 7. It contains two tables. The left table lists bind variables, and the right table is a spreadsheet for entering values.

Bind Variable		Bind Value		Null ?
1	First_Name	1		<input type="checkbox"/>
2	Mid_Initial	2		<input type="checkbox"/>
3	Empno	3		<input type="checkbox"/>
4	Last_Name	4		<input type="checkbox"/>
		5		<input type="checkbox"/>
		6		<input type="checkbox"/>
		7		<input type="checkbox"/>

Buttons: OK, Cancel, Help

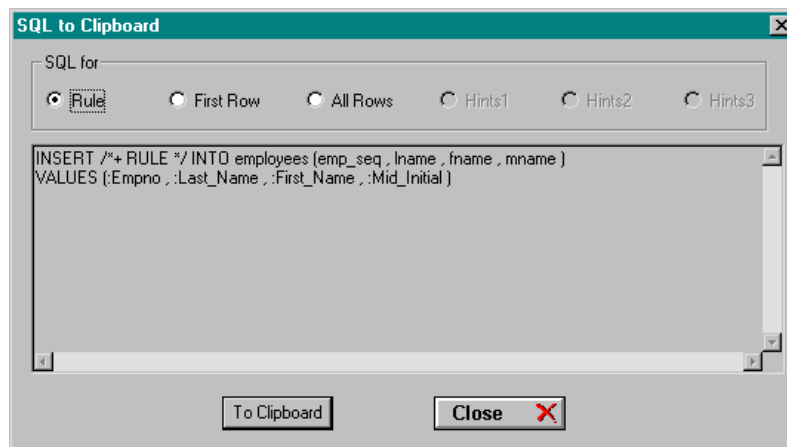
Set Bind Variables Dialog (Non-SELECT Statement)

Enter the size (cardinality) of the array in the **No. of values per bind variable** field. After you set the cardinality, the spreadsheet on the right enables the specified number of rows. Enter the values for each bind variable in this spreadsheet. To specify the values for a specific bind variable, you must first select the variable name and then enter the values in the enabled rows of the spreadsheet on the right. Enter Null values by setting the check box in the column labeled **Null**.

Note • When the SQL and execution statistics are saved in the Plan Analyzer repository, any bind values that exist in the SQL statement are also saved.

Copy SQL to Clipboard

Once the best optimization plan is found, you can use the **SQL, Copy SQL to Clipboard** option to replace the SQL statement in the application. For instance, most applications use the default optimization mode specified at the database level. If testing proves that the First Rows mode is the most optimal, you can simply paste the modified SQL to the clipboard, and paste it directly into the application.



SQL to Clipboard Dialog

To get the Rule optimization plan illustrated in the figure above, set the **Rule** radio button and click **Copy To Clipboard**.

To achieve a specific optimization mode, Plan Analyzer embeds a hint for **Rule**, **First Rows** and **All Rows** in the outermost SQL module of the SQL statement. For instance, to ensure that the SQL statement in the above figure uses the Rule optimization mode, Plan Analyzer places the RULE hint in the first SELECT list. If the Hints version is requested, the hints entered by the user will appear instead.

Format

The **Format** menu item formats the SQL statement in the SQL window according to the formatting options set in the **Other** tab of the Preferences dialog. Properly formatted SQL code is much easier to understand. For example, consider the following unformatted SQL statement:

```
select * from employees where emp_seq = (select emp_seq
from sal_history s1 where effective_date = (select max(effective_date)
from sal_history s2 where s1.emp_seq = s2.emp_seq and effective_date <=
sysdate)
and sal = (select max(sal) from sal_history where
(emp_seq,effective_date) in
(select emp_seq, max(effective_date) from sal_history where
effective_date <= sysdate
group by emp_seq) ) )
```


It is difficult to start optimizing the statement before it has been formatted. Following is the same SQL statement after it has been formatted using **SQL, Format**:

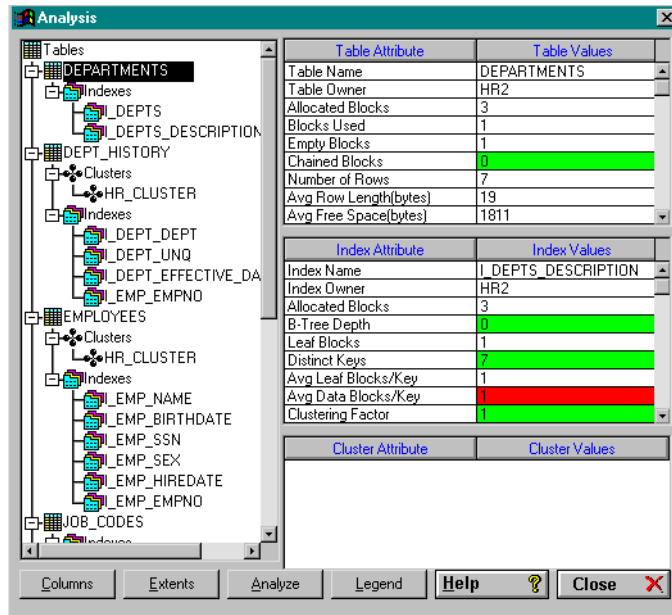
```
SELECT *
FROM employees
WHERE emp_seq =
    (SELECT emp_seq
    FROM sal_history s1
    WHERE effective_date =
        (SELECT max (effective_date )
        FROM sal_history s2
        WHERE s1.emp_seq = s2.emp_seq
        AND effective_date <= sysdate )
    AND sal =
        (SELECT max (sal )
        FROM sal_history
        WHERE (emp_seq , effective_date ) in
```

```
(SELECT emp_seq, max (effective_date )  
FROM sal_history  
WHERE effective_date <= sysdate  
GROUP by emp_seq )  
)  
)
```

Note • The specific format of statements will vary depending on the preferences set in the Preferences dialog. See the *View Menu* chapter.

DB Objects Analysis

The database objects listed in the steps of the optimization plans are generally tables and indexes. Oracle uses any statistics available on the tables and indexes to determine the best cost-based optimization plan. Select **SQL, DB Objects Analyze** or the toolbar icon  to display the Analysis dialog and list the various statistics.



Analysis Dialog

When the Analysis dialog initially appears, all fields are blank. You must first select a table from the Table tree on the left side of the dialog. If you select a clustered table, the cluster information displays immediately. After picking a table, click on the Indexes icon to dropdown a list of all the indexes on the table, including the cluster index if the table is clustered. Click on the desired index to display its properties in the Analysis Index Attributes and Values fields.

Analysis Dialog Statistics

The Analysis dialog contains storage statistics about the table and associated indexes. It includes the cluster statistics if the table is clustered. You must first understand what each statistic means, and then use the statistics to determine such things as:

- 1 Should this index ever be used to drive a query?

- 2 Should this index, table, or cluster be reorganized?
- 3 Can this index be useful if the rows were sorted prior to inserting?
- 4 Should the hashed cluster be redefined for more cluster keys?

Each of the above questions in itself has many points. For example, should the table be reorganized because the table has too many extents, too many chained rows, too much free space per block, or too many empty blocks? You can use the statistics to determine whether an index will ever be useful. This section explains how to interpret the statistics, and how to use Plan Analyzer to simplify the interpretation.

Note that most storage statistics are only available if the object has been analyzed using the Oracle **ANALYZE** command. Assuming an estimate of the statistics is not performed, statistics are only correct at the time they are created. Database objects have to be re-analyzed periodically to ensure the correctness of the statistics. Additionally, if a table is part of a cluster, certain statistics will pertain to the cluster, not the table. Plan Analyzer uses color coding to identify which statistics are not applicable when the table is clustered, and which statistics are not available unless the object is analyzed using the **ANALYZE** command.

The following color codes are defined:

Color Codes	Meaning
RED	Alarm! Needs attention.
YELLOW	Warning. May require attention.
BLUE	Not applicable when clustered.
BLACK	Only available when analyzed.
GREEN	Desired value range

The purpose of the color coding is to help you visually interpret the statistics quickly, or at least alert you to certain facts that you can research further. For instance, knowing which objects have not been analyzed will have a great deal of influence on cost-based optimizations. Cost-based optimizations require all statistics for an accurate optimization plan. Certain facts that would otherwise be blank when the table is clustered are clearly marked, so there is no mistake.

Finally, Plan Analyzer informs the user of potentially poor performance characteristics. Plan Analyzer will also indicate when statistics are in the “desirable” range. The user can specify, on an individual client basis, the parameters governing when a statistic is in the “desirable” range, the “alarm” range, or the intermediate “warning” range. See *Alert Analysis Parameters* in the *View Menu* chapter for more information.

Chained Rows

A B-tree index contains the ROWID for each row that has a non-NULL value for the columns in the index. It is important to understand that a row can span multiple data blocks if the row is too large for the current block. The ROWID is the address of the block where the row is initially inserted; this is called the header block. When a row spans multiple data blocks, reading the row means reading multiple data blocks. Since the goal is to always minimize the resources utilized, chained blocks should be eliminated by re-organizing the table to eliminate the chained rows. Re-organizing generally comes in the form of exporting the table, truncating the table, and then finally importing the table (plus disabling and re-enabling the constraints).

Chained rows cannot always be eliminated when the row is too large for the Oracle data block size. Rows with LONG columns are an example. If some LONG columns are not always retrieved, consider separating those columns along with a copy of the primary key into a separate table. Often, the LONG column is viewed only when the user wants more detail. The existence of the LONG column is still reflected in resources consumed by a query on the non-LONG columns of the table, even if the LONG column is not always retrieved, since the rows are spread across several blocks.

Keep one goal in mind when planning the storage parameters for a table: the more rows per data block, the faster full table scans can be performed. Considering that full table scans play a significant role in the optimization of Oracle V7.1's parallel query option, plan carefully to fit the maximum rows per data block.

If the table has been analyzed using the Oracle **ANALYZE** command and the table is not clustered, the number of blocks that are chained to a header block is listed in the **Chained Blocks** field.

Specify optimal and alarm values for the percentage of chained blocks that can be allocated to the table. For instance, if the optimal value for chained blocks is zero percent, the field is colored green only if the number of chained rows is zero. If the alarm threshold is 10 percent and the total blocks allocated to the table is 2000, then the field will be colored red when the number of chained blocks is 200 or more. Anything less than 200, but greater than the optimal amount, will result in the field being colored yellow.

Typically, unless the table contains a LONG column, the chained blocks should be 1 percent or less. Unacceptable values will probably best be determined by experience, but consider keeping the alarm threshold at 0.5 percent. The valid range of values for the optimal and alarm percentages is between 0 and 100, and the optimal must be less than the alarm. These values can be modified in the in the **Analysis Alert Parameters** tab, under the **View, Preferences** menu item.

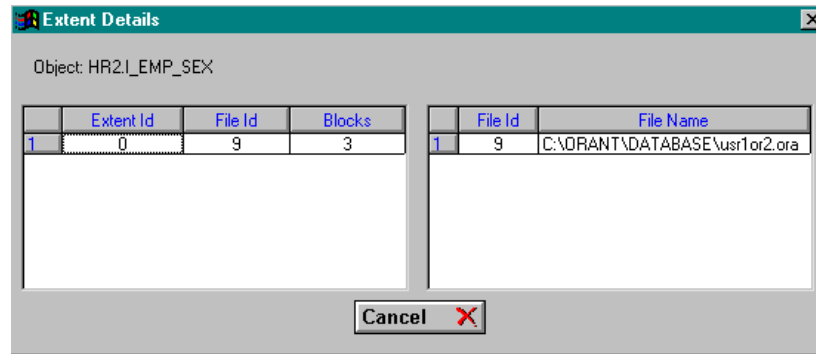
Fragmented Tables, Indexes, and Clusters

Fragmentation means that an object is spread across multiple contiguous extents. For instance, when a table, index, or cluster is created, a single extent is usually allocated. When the extent becomes full, Oracle automatically allocates another extent based on storage parameters specified on the object definition. One desirable goal is to limit the number of extents, keeping the object as contiguous as possible. This goal is not always attainable, and in some cases is not desirable. For example, *striping* a table means that rows of the table are spread across multiple storage devices, requiring multiple extents. Striping is an effective means

of spreading the access to a single object over multiple devices with the goal of improved performance. In the case of Oracle V7.1, the parallel query option benefits from having multiple servers reading the rows from multiple devices, especially in the case of full table scans. Multiple extents can also improve concurrent usage, since multiple users are more likely to access different disks. If a user does intend to use the parallel query option, then a full table scan would benefit most from one extent.

The optimal value for extents is based on the maximum number of extents for a table of a specific size. For example, the default setting is two extents per 100 data blocks. Therefore, a table is still optimal if it has 500 data blocks spread across 10 extents. By associating the number of extents with a number of data blocks, it is possible to specify an optimal threshold and an alarm threshold for large and small tables, indexes and clusters. The alarm value for extents colors the field red when the value is greater than or equal to the alarm value. These values can be modified in the **Analysis Alert Parameters** tab, under the **View, Preferences** menu item.

Click the **Extents** button in the Analysis dialog to display the Extent Details dialog (figure below). The spreadsheet on the left lists the different extents. The **Extent Id** column starts at zero and is incremented for each extent. The **File Id** column identifies the database file that contains the extent, and the **Blocks** column contains the size of the extent in Oracle blocks. The unique database files are listed in the spreadsheet on the right.



Extent Details Dialog

Allocated, Used, and Empty Blocks

Each object is allocated a certain amount of storage (**Allocated Blocks** in the Analysis dialog). To understand the size of an object other than an index, you need to know what portion of those allocated blocks are actually used (**Blocks Used**). The remaining blocks are a combination of table overhead and unused blocks (**Empty Blocks**). When a table is clustered, the **Empty Blocks** value does not pertain to the table, but the cluster. Therefore **Empty Blocks** will have the same value for all tables in the same cluster, even for tables that have no rows. The storage belongs to the cluster, not to the table.

These statistics provide information about the size of a table and the amount of wasted space. Empty space on the cluster is more significant, since the number of empty blocks indicates the number of slots available for new cluster index keys. If the cluster is a hashed cluster, the number of empty blocks, along with the average blocks per cluster key, (**Avg. Blocks/ Cluster Key**), indicates whether you need to reorganize the cluster with more storage and a higher number of hash keys (**# of Hash Keys**).

Clusters

Clustering means storing rows of related tables together in the same data blocks, rather than storing each table separately. Clustering prejoins the tables in the cluster, saving the overhead of the join process later on. Individual tables may be clustered by themselves to save storage of repeating the cluster key. This is especially useful when the cluster is a hashed cluster. Whether the cluster is an indexed cluster (meaning a B-tree index is created on the cluster key), or a hashed cluster, the potential for collision of many rows on the same cluster key can severely limit the usefulness of the cluster.

Plan Analyzer provides the **Avg. Blocks/Cluster Key** statistic to indicate the average number of data blocks associated with a key value. The goal is to keep the value as close to 1 as possible. When more than one data block is used for a key value, more I/O is performed to read rows not on the header block. If the query is always a join, the additional blocks are probably not a concern. But when the query is for a subset of rows for a cluster key value, or the operation is an update or delete, more I/O is expended.

If one of the tables in the cluster contains a LONG column, there is a significant chance that chaining will occur. Consider moving the LONG column to a new table to avoid chaining, especially if the LONG column is infrequently accessed. For hashed clusters, keep in mind that the hashing reduces the I/O for exact matches, so a few extra blocks per hash key may not be a negative.

Specify the optimal and alarm blocks value per cluster key in the **View, Preferences** dialog under the **Analysis Alert Parameters** tab. If the value in the Analysis dialog is equal to or less than the optimal value, the field is colored green. If the value is equal to or greater than the alarm value, the field is colored red. If the value is between (but not inclusive of) the optimal and alarm values, the field is colored yellow. The default optimal value for **Avg. Blocks/Cluster Key** is 1. The values for the optimal and alarm thresholds must be greater than 0 and the optimal must be less than the alarm threshold value. These values are modified from the **View** menu in the Preferences dialog under the **Analysis Alert Parameters** tab.

Hashed Clusters

Hashed clusters can offer a significant performance enhancement under certain conditions. The advantage of the hash index is that it is an algorithm which avoids additional I/O in accessing the index prior to accessing the data. Hashing is only good for exact matches, not range searches. Therefore the hash index can be used only for equality constraints and joining. Some details, such as human resources data, are never accessed except by their foreign key. The typical human resources request is for an employee record and its associated details. The details are always referenced by their foreign key, the employee id.

The number of tables included in the hashed cluster may need to be reduced if **Avg. Blocks/Cluster Key** is larger than 1, although the exact limit is in the user's control. Finally, if too many rows hash to the same key, multiple rows will be stored under the same hash key. This is different from indexed clusters (using the B-tree index), since only rows with the same cluster key value will be clustered together. Hashed clusters cluster rows together where the cluster key values are different, as long as the cluster key values hash to the same hash value. The goal is to ensure that there are enough hash keys to accommodate the hashed data values, so that each cluster key value hashes to a unique hash key.

Platinum strongly suggests investigating the use of hash clusters to take advantage of the hash index. For instance, if a B-tree index is used to access a single row, the amount of I/O is equal to the value of the B-tree depth attribute for the index plus the I/O to the datapage. Therefore if the B-tree depth equals 4, you should expect to perform at least five I/Os to retrieve a specific row. Using a hashed cluster instead avoids the I/O to the index. So pay special attention to a large B-tree depth value, and consider converting to a hashed cluster if the index is only used for exact matches. If you need to perform range searches, still consider putting the table in a hashed cluster and maintain the separate B-tree index for the range searches.

Always attempt to achieve a very small **Avg. Blocks/Cluster Key** value (the best being 1), and **# of Hash Keys** equal to the allocated blocks for the cluster.

Considering the savings on index I/O, an **Avg. Blocks/Cluster Key** value of 3 is not unreasonable.

B-Tree Depth

The depth of the B-tree (**B-Tree Depth**) is the number of blocks that must be read after reading the root block of the index. A value of 0 indicates that the root and the leaf block are one and the same—that is, a very small index. The larger the B-tree depth, the more I/O that must be performed before the first data page is accessed. For instance, if the B-tree depth is equal to 4, Oracle must read at least 4 more index blocks after the root block before reaching the leaf blocks where the ROWIDs are stored. The greater the value, the more I/O that must be performed. But everything is relative. In the following example, a range search is performed on the EMPLOYEE table in the human resources database over the HIREDATE column:

```
SELECT * FROM HR.EMPLOYEES
WHERE HIREDATE BETWEEN '01-JAN-94' AND '31-DEC-95
```

In this case, after the overhead of the extra I/O to get to the leaf blocks of the index, Oracle scans the leaf blocks. If the employees having a hiredate within the search range cover 10 index leaf blocks, and the B-tree depth is 4, Oracle would have to perform 14 reads to get the ROWIDs, not counting the reads of the data blocks.

This statistic should be used in conjunction with the size of the table, since inevitably the index will grow as the table grows. But overall, you need to be concerned with the additional I/O that is incurred when using an index. An optimal and alarm threshold can be set for the B-tree depth in the Preferences dialog under the **Analysis Alert Parameters** tab, under the **View, Preferences** menu. The optimal size should be approximately 2 -4, whereas the alarm size may be 5 or more. If the B-tree depth is less than equal to the optimal value, the field will be colored green. If the B-tree depth value is greater than or equal to the alarm value, the field will be colored red. Otherwise the field will be colored yellow, indicating a warning condition.

Distinct Keys

The most significant statistic on indexes is the number of distinct values (**Distinct Keys**) within the index. If the number of distinct keys is low for a specific index, better performance will be achieved by choosing an alternative index or a full table scan instead of using that index. That applies to both join and non-join criteria. Since Oracle optimizes full table scans by performing multi-block reads (DB_FILE_MULTIBLOCK_READ_COUNT parameter in the INIT.ORA file), a full table scan can often provide a more efficient access method. The new parallel option in Oracle V7.1 provides increased efficiency for full table scans.

The reason for not using the index is more complicated than just looking at the number of distinct keys. In general, the more distinct keys there are, the fewer data blocks will match a specific key value.

When more than one key value is being searched (for example, a range search), utilizing the index to perform the search becomes less efficient. But assuming an exact match, the larger the number of distinct keys, the less I/O that is required to retrieve the rows. The ultimate case is that of the unique index, in which the percentage of distinct keys is 100. In this case the I/O required to retrieve a single row from the table is equal to the B-tree depth plus 1 for index I/O and one more for the data block. If the B-tree depth is 2, the total I/O performed equals 4. But once the number of distinct keys decreases, more data blocks must be read; but how many more? If the search is for a value that is contained in 4 rows of the table, the amount of I/O is now dependent on whether the 4 rows are on the same data block, or on 4 distinct blocks. The average data blocks per key and the clustering factor help answer the question. (See *Clustering Factor* and *Average Data Blocks Per Key* in this chapter for details.)

Consider the effects of join criteria when you use an index to join two tables. If most of the rows in the child table will be joined to the parent table, using the index to perform the join is actually slower. The total I/O is higher since many leaf blocks of the index must be read, in addition to the data blocks. Also, the logical I/O is increased, because the same data blocks may be read multiple times when the rows in a single data block have different index key values. The alternative is to perform a sort

join, where the index is not used for the join. Instead, a full table scan of the child table is performed and then sorted by the join criteria. Finally, a merge join is performed to join the parent rows to the child rows. This sort join also eliminates the redundant reading of the data blocks.

Note that indexes support other operations such as sorts, or satisfying the query within the index. If the index is used for sorting the rows, you still must consider whether the increase in data block reads outweighs the benefit of using the index for the sort. It may be better to let Oracle perform a full table scan to sort the table. The best index to perform the sort is an index created on a table that was sorted by the index columns prior to loading the table. For example, if the EMPLOYEES data is sorted by LNAME prior to loading the rows into the table, the index on LNAME will provide superior performance when used to list the rows sorted by LNAME, since there will never be redundant reading of the same data block. However the data is ordered, Oracle does perform multi-block reads efficiently, and the use of an index precludes the use of multi-block reads.

The index is still useful, regardless of the number of distinct values, when the index precludes the need to read the actual data page. This happens in two places. First, when the SELECT list references only columns that are in the index and the only criteria specified can be satisfied in the index, then the rows need not be read from the data pages. That eliminates redundant reads, though depending on the number of qualifying rows, the full table scan might still be more efficient. The following is an example:

```
SELECT LNAME, FNAME, MNAME
FROM HR.EMPLOYEES
WHERE LNAME LIKE 'SM%'
```

In this case, the criteria can use the I_EMP_NAME concatenated key index where the leading index column is the LNAME column. Since the index consists of all three columns on the SELECT list, there is no need to query the data blocks. The following is the rule-based optimization plan for the SQL:

```
1 - UNIQUE INDEX [RANGE SCAN] of HR.I_EMP_NAME[LNAME,FNAME,MNAME]
```

The following SQL would require retrieving the data blocks:

```
SELECT LNAME, FNAME, MNAME
FROM HR.EMPLOYEES
WHERE LNAME LIKE 'SM%'
AND BIRTHDATE BETWEEN '01-JAN-52' AND '10-FEB-53'
```

Assuming the I_EMP_NAME index was used to drive the query, the criterion on BIRTHDATE requires retrieving the rows where the final verification of the BIRTHDATE criterion is performed. If the hint INDEX(EMPLOYEES, I_EMP_NAME) is used, the hints-based optimization plan looks like this:

```
1 - TABLE ACCESS [BY ROWID] of HR.EMPLOYEES
  1 - UNIQUE INDEX [RANGE SCAN] of HR.I_EMP_NAME(LNAME,FNAME,MNAME)
```

The rule of thumb is to use the index only if the number of distinct values is greater than 15-20 percent. Recall that the **average data block per key** and **clustering factor** are major determinants of index effectiveness. An optimal and alarm threshold can be specified for the percentage of the distinct keys. By default, the optimal value is 90 percent, and the alarm is 20 percent. If the percentage of distinct values is greater than or equal to the optimal threshold, the **Distinct Keys** field is colored green. If the percentage of distinct values is less than or equal to the alarm threshold, the field is colored red. Otherwise the field is colored yellow to signal an alarm condition.

Average Leaf Blocks Per Key

Leaf blocks are blocks in the B-tree index where ROWIDs associate the index keys with the Oracle data blocks containing rows. The other blocks in the index are the root and branch blocks. The root block is the first block entered when accessing the index. It is also considered a branch block. The branch blocks are those index blocks that contain index data pointing to other index blocks. The purpose of the branch blocks is to guide the search through the index to get to the desired leaf block. Within the leaf blocks are the full index values, not partial values as in branch blocks.

The statistic **Avg Leaf Blocks/Key** indicates how much space is required to hold an average index key value and associated ROWIDs. Using the index to access a specific index key value requires reading the associated branch blocks to reach the leaf blocks and reading all the leaf blocks for the index key value. If **Avg Leaf Blocks/Key** is large, it means that either there are many matching data blocks or the index key value is large. For example, creating an index on a VARCHAR(2000) column could easily use multiple 2K Oracle blocks. Concatenated key indexes can also be large. The largest index key size is platform dependent, but is generally based on the Oracle block size. To determine if the latter is the case, perform the following query to get the average key size for the I_EMP_NAME concatenated index, consisting of columns LNAME, FNAME and MNAME:

```
SELECT /*+ index (employees, i_emp_name) */  
      AVG(LENGTH(LNAME)+LENGTH(FNAME)+LENGTH(MNAME))  
FROM   HR.EMPLOYEES  
WHERE  LNAME >= 'A'
```

The query must have some criteria that reference the index. The hint is added to ensure that the correct index is used.

Plan Analyzer does not provide any thresholds to mark acceptable regions. The statistic should be used in conjunction with distinct values, since the reason for the large number of leaf blocks per key is that there are few distinct values.

Average Data Blocks Per Key

The leaf blocks list the full index value and ROWIDs of all rows having the index values. The data block address is within the ROWID. The average data blocks per key (**Avg. Data Blocks/Key**) is the average number of data blocks referenced by an index key. The more data blocks, the more I/O performed on an index key match. For example, an equality criterion using the index will generally require the I/O to access both the index and the average data blocks per key. To determine the effectiveness of the index requires measuring the percentage of the total used data blocks represented by **Avg. Data Blocks/Key**. If the **Avg. Data Blocks/Key**

equals 10 and the total number of used data blocks is 5000, then an equality search on the index requires, on average, reading only 0.2% of the data blocks. If the total number of used blocks equals 50, then the same equality search would, on average, read 20% of the data blocks.

The thresholds for **Avg. Data Blocks/Key** are based on the percent of data blocks used. If **Avg. Data Blocks/Key** is less than or equal to the optimal percent of used data blocks, the field is colored green. If **Avg. Data Blocks/Key** is greater than or equal to the alarm value, the field is colored red. Otherwise, the field is in the warning range and is colored yellow. The optimal and alarm threshold percents are specified from the **View** menu, in the **Preferences** dialog under the **Analysis Alert Parameters** tab.

Clustering Factor

The clustering factor is an interesting statistic that is expected to become more accurate as Oracle Corporation provides a clearer definition. The Oracle manuals describe the clustering factor as a measure of the “order of the rows in the table based on the values in the index”. For example, a unique index is considered *well-ordered*, while an index on a column that has the same values randomly distributed over data blocks would be considered *randomly-ordered*.

Oracle manuals explain that an index is well-ordered if the clustering factor is near the number of blocks used by the table. The index is randomly-ordered if the index is near the number of rows in the table. You might wonder what it means when the number of blocks equals the number of rows, or worse, when the number of blocks is greater than the number of rows. Using two different attributes to distinguish the two opposite meanings of another attribute does not make good sense. For that reason, the following example will hopefully shed some light on the clustering factor. Consider the following table definition using the default storage clause:

```
CREATE TABLE TEST
(COL_UNQVARCHAR2(30),
COL_SAME_FOR_BLOCKVARCHAR2(30),
COL_ALL_SAMEVARCHAR2(30),
COL_RANDOMVARCHAR2(30) )
```

The table was created with 4096 rows and updated with the following statement:

```
UPDATE TESTSET COL_UNQ = ROWID,
COL_SAME_FOR_BLOCK =
SUBSTR(ROWID,1,8) || SUBSTR(ROWID,15,4),
COL_ALL_SAME = '123456789012345678901234567890',
COL_RANDOM = MOD(ROWNUM, 25)
```

This ensures that the COL_UNQ column has a unique value for each row by setting the value equal to the ROWID. The COL_SAME_FOR_BLOCK column is set to the Oracle block address, by setting the column value to the concatenation of the block portion of the ROWID (SUBSTR(ROWID,1,8)) and the file portion of the ROWID (SUBSTR(ROWID,15,4)). Therefore all rows in the same Oracle block have the same value for COL_SAME_FOR_BLOCK. The values for COL_ALL_SAME are all the same, and the values for COL_RANDOM are made more random by setting the value to the remainder of the ROWNUM divided by 25 (MOD(ROWNUM, 25)). The random values range from 0 to 24. Since each data block contains about 27 rows, each data value is in each data block. (Note that this is not truly random data, since the sequence repeats every 25 rows; the storage space dictates where the cycle restarts.)

The next command creates a separate index on each column, gives the index the same name as the column, and then analyzes the table.

```
ANALYZE TABLE TEST COMPUTE STATISTICS
```

The following table lists the clustering factor by index name:

Index Name	Clustering Factor
COL_UNQ	152
COL_SAME_FOR_BLOCK	152
COL_ALL_SAME	152
COL_RANDOM	3794

Since the table uses 152 blocks and the number of rows is 4096, a perfectly well-ordered index will have a value of 152 and a perfectly random-ordered index will have a value of 4096. The first three indexes are considered well-ordered. But there is a difference between a unique index and an index in which each value is the same. So what *does* “well-ordered” mean? No single statistic can indicate whether or not the index is a good index to use as an access path. The clustering factor combined with the number of distinct keys is a good combination, since in the case of COL_ALL_SAME, the distinct keys equal 1, which means that it’s useless. Since the number of distinct values for the COL_UNQ column equals the number of rows in the table, there is no need to check the clustering factor, because COL_UNQ is an ideal index—a unique index. But consider the index COL_SAME_FOR_BLOCK. The distinct values equals 152, which is good. The average data blocks per key (**Avg. Data Blocks/Key**) equals 1, which is great. COL_SAME_FOR_BLOCK is considered a good access path.

What about the COL_RANDOM index? Since the index has a clustering factor of 3794, which is closer to 4096 than 152, should the index be considered closer to randomly-ordered? Certainly it is not well-ordered. The number of distinct values is 25, which is also considered poor. In fact the average data blocks per key statistic equals 151, meaning that each key value references almost the entire table. In this case a full table scan is far more effective.

To further expand on the example, update the COL_RANDOM column as follows:

```
UPDATE TEST SET COL_RANDOM = MOD(ROWNUM, 50)
```

Re-analyze the index, COL_RANDOM. The clustering factor for the index is 4096, indicating that the index is even more random than the previous one, even though the number of distinct values is far greater and the average data blocks per key is still high. Its value is 81, which is over half the data blocks.

But what happens when the number of rows equals or exceeds the number of blocks? Then it's a case of how close the clustering factor is to the number of blocks. Oracle states that the clustering factor indicates a well-ordered index when the clustering factor is "near" the number of blocks. But the question should be, "how near?"

Plan Analyzer attempts to simplify the specification of thresholds by using a scale between 0 and 10, where 0 indicates a perfectly-ordered index and 10 is a completely random index. When the number of blocks is greater than or equal to the number of rows, the index is considered perfectly ordered. An optimal and alarm threshold can be set for the clustering factor from the **View** menu, in the **Preferences** dialog under the **Analysis Alert Parameters** tab. The recommended optimal threshold is 0.5, and the recommended alarm threshold is 5.

Column Analysis

Click the **Columns** button to display the **Column Analysis** dialog. Each row in the spreadsheet represents a column of the table. If the **ANALYZE** command has not been executed for the table, then only the **Column Name**, **Data Type** and **Null?** columns have values. If the table is analyzed, the number of distinct values for each column is listed in the **#Distinct** column and the second-lowest and highest values are stored in the **Low** and **High** columns respectively.

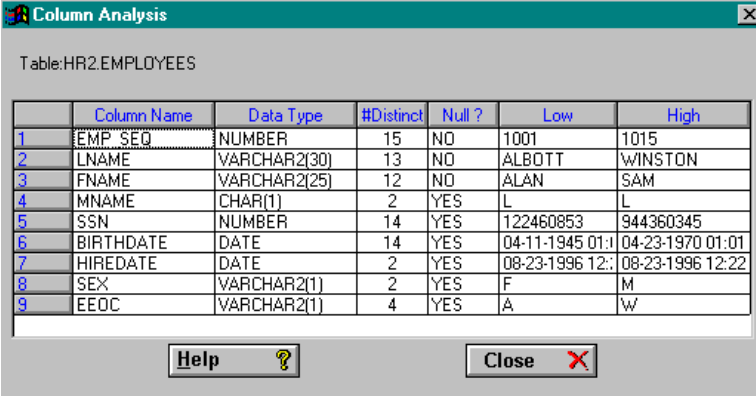


Table: HR2.EMPLOYEES

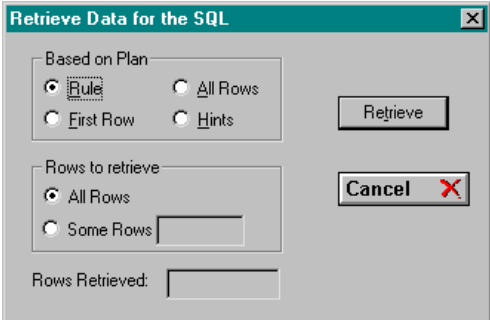
	Column Name	Data Type	#Distinct	Null ?	Low	High
1	EMP_SED	NUMBER	15	NO	1001	1015
2	LNAME	VARCHAR2(30)	13	NO	ALBOTT	WINSTON
3	FNAME	VARCHAR2(25)	12	NO	ALAN	SAM
4	MNAME	CHAR(1)	2	YES	L	L
5	SSN	NUMBER	14	YES	122460853	944360345
6	BIRTHDATE	DATE	14	YES	04-11-1945 01:1	04-23-1970 01:01
7	HIREDATE	DATE	2	YES	08-23-1996 12:1	08-23-1996 12:22
8	SEX	VARCHAR2(1)	2	YES	F	M
9	EEOC	VARCHAR2(1)	4	YES	A	W

Buttons: Help, Close

Column Analysis Dialog

Retrieve Data

Retrieving the result set of a SQL query can also assist in verifying that the query is producing the expected results. The **Retrieve Data** menu item returns all or some rows for the query listed in the SQL window. Select **SQL, Retrieve Data** to display the **Retrieve Data for the SQL** dialog:



Retrieve Data for the SQL

Based on Plan:

☒ Rule ☐ All Rows

☐ First Row ☐ Hints

Retrieve

Rows to retrieve:

☒ All Rows ☐ Some Rows

Cancel

Rows Retrieved:

Specify the optimization plan to use and the number of rows to return. It is possible, depending upon the optimization mode chosen, that the rows will be returned in a different order.

The following figure displays the result set for the SAMPLE3 query.

The screenshot shows the Plan Analyzer for Oracle interface. The top window displays a sample SQL query. The bottom window shows the result set for the query, which consists of three rows of employee data.

Query:

```
select /*+ RULE */ * from employees
where sex = 'M'
and EEOC = (select code from eeo_codes
            where description = 'BLACK')
and hiredate >= add_months(sysdate,-12)
```

Result Set:

	EMP_SEQ	LNAME	FNAME	MNAME	SSN	BIRTHDATE	HIR
1	1004	LONG	LOU		144460809	11-APR-45	23-AU
2	1012	WINSTON	MIKE		122460853	23-APR-70	23-AU
3	1014	BAKER	ELLIOT		222460853	20-AUG-50	23-AU

Result Set for Sample Query

Capture Menu

The Capture menu appears in both Expert and Standard modes. This chapter describes the options available from the Capture menu.

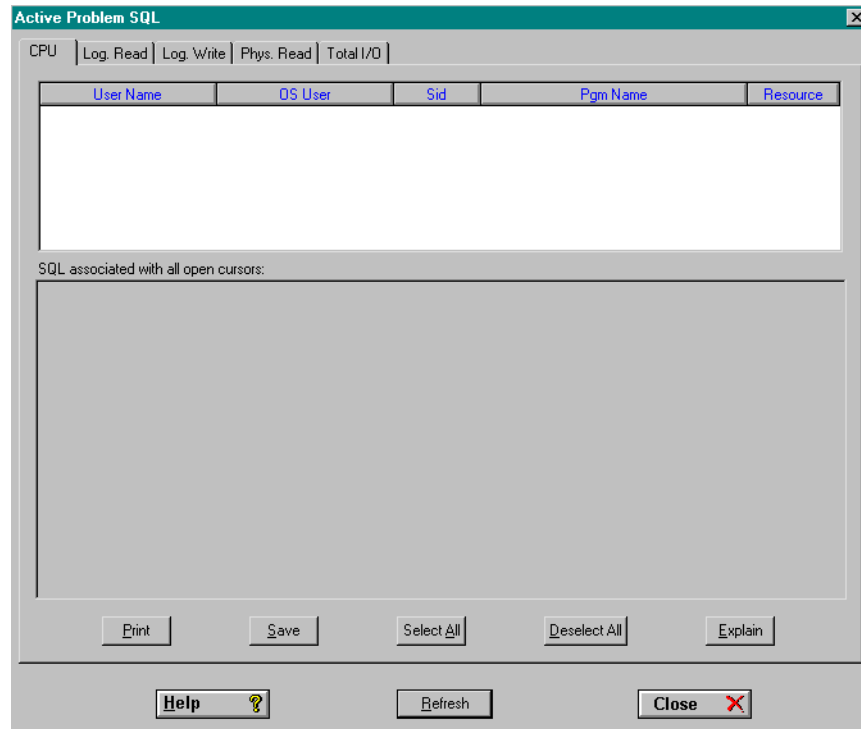
Active Problems	6-3
Statistics and Features of Result Sets	6-4
The CPU Tab	6-7
The Logical Reads Tab	6-8
The Logical Writes Tab	6-9
The Physical Reads Tab	6-10
The Total I/O Tab	6-10
Server SQL Snapshot	6-10
The Result Tab	6-11
The All SQL Tab	6-14
The By Session Tab	6-15
The By Ora User Tab	6-17
The By OS User Tab	6-19
The By Table Owner Tab	6-20
The By Tablename Tab	6-22
The By Tablespace Tab	6-23
The By Datafile Tab	6-25

The By Resource Tab	6-27
By SQL Text Tab	6-28
Server SQL Monitor	6-29
Top SQL Resources	6-29
The Review Window	6-35

Active Problems

Database administrators are often asked, “Why is the server response time so slow?” One of the first things a DBA typically does is identify who is consuming the most resources. Too often, DBAs rely on products that produce historical statistics to identify which session or SQL statement should be examined. This methodology is problematic in that historical statistics do not reflect what is causing the current poor server performance.

Plan Analyzer’s Active Problems facility examines what is *currently* being executed on the server. It provides a powerful yet easy to use method of viewing and identifying the SQL statements that are using the most resources. It captures information on the top CPU and I/O consumers, as well as who is performing the most logical reads, logical writes, and physical reads. Note that the Active Problems feature is for DBAs only: Users without DBA privileges cannot run this.



The Active Problem SQL Window

Statistics and Features of Result Sets

Plan Analyzer provides a comprehensive list of statistics for the sessions captured using the Active Problem facility. The table and two figures that follow illustrate and describe the statistics for a typical SQL capture. The first figure shows the statistics as they appear by default; the second shows a continuation of the view when the window is scrolled to the right:

ALL ACTIVE SQL:
 No. 1 User: HR OS User: CONNT SID: 14 OS Pgm: OraPgm
 Logical Reads: 9 Physical Reads: 3 Parses: 1 Sorts: 3 Executions: 1
 select sal_history.emp_seq, job_history.emp_seq, dependents.emp_seq, employees.emp_seq from job_history, sal_histo

Share Mem: 12377 Versions: 1 Curr Users: 1
 , dependents, employees

Typical SQL Statistics

Statistic Name	Definition
No.	Shows the ordinal number of the SQL statements in the display.
User	The ORACLE account name of the user generating the SQL.
OS User	The user's operating system login name.
SID	The ORACLE session ID number.
OS Pgm	The program the SQL was generated from.
Logical Reads	The number of reads from ORACLE's buffer cache.
Physical Reads	The number of reads that required disk access.
Parses	The number of times this statement has been parsed.

Statistic Name	Definition
Sorts	The number of sorts the query performed.
Executions	The number of times the statement was executed.
Share Mem	The size of shareable memory used by the statement in the shared pool.
Versions	The number of loaded versions.
Current Users	The number of current users.
Optimizer Mode	The optimization mode used by the SQL statement. Available on ORACLE 7.3 and higher.

In addition to these statistics, the text of the entire SQL statement is reproduced in the window.

Other Features

To take advantage of the other powerful features available in the Active Problem feature, use the buttons at the bottom of the window, as shown in the following figure:



Click **Print** button to send the selected SQL statement and statistics to your printer. This is useful for generating reports on the current status of the server.

Click **Save** to save the selected SQL and statistics. Later, you can analyze and generate optimization plans for the SQL.

Select All selects all the currently captured SQL and statistics.

Deselect All deselects whatever captured SQL you have selected.

Click **Explain** to automatically copy the SQL into Plan Analyzer's SQL window and generate an optimization plan. This is a quick and powerful way to analyze SQL that may be causing a server slowdown.

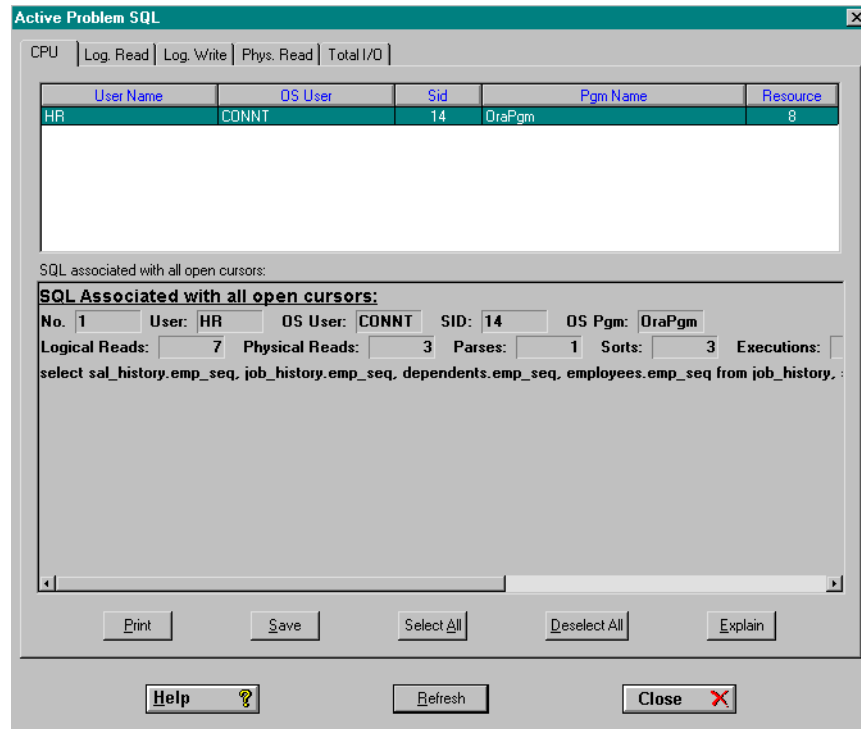
The **Refresh** button causes Plan Analyzer to take another snapshot of the server.

The **Help** button brings up Plan Analyzer's online help.

Use **Close** to exit the Active Problem window.

The CPU Tab

The CPU Tab captures the SQL currently on the server that is using the most CPU. The following figure illustrates the CPU Tab:



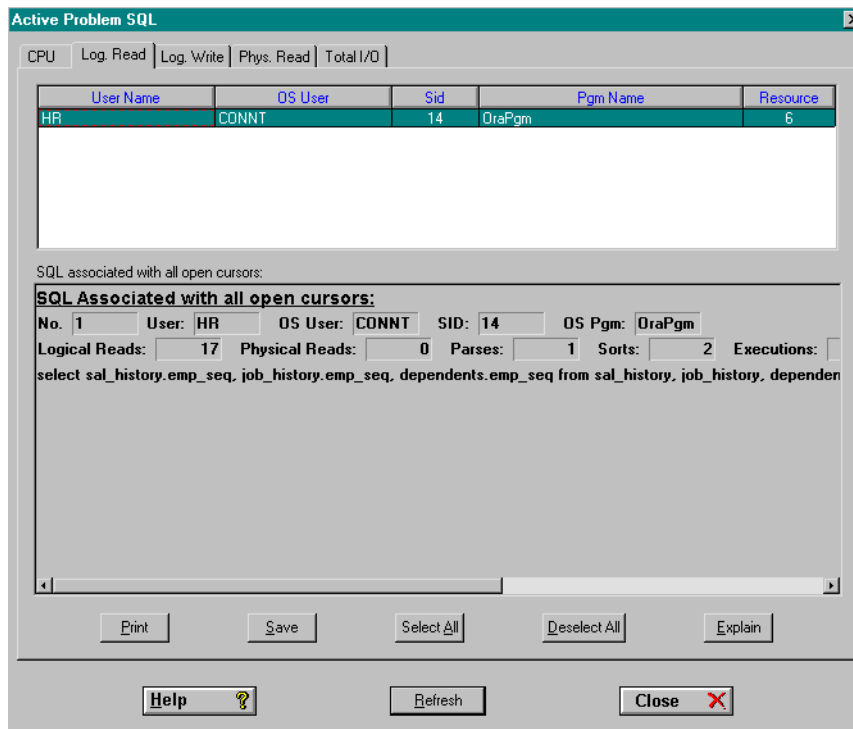
The Active Problem CPU Tab

The spreadsheet at the top identifies the User Name, the OS User, the Session ID (SID), the Program Name, and Resource consumed by the session.

The SQL statements using the most CPU are displayed below.

The Logical Reads Tab

The Logical Reads tab captures the SQL that is performing the largest number of logical reads. The following figure illustrates:



The Logical Reads Tab

The spreadsheet shows the User Name, the OS User, SID, Pgm Name, and Resource, as in the CPU tab. The SQL displayed represents what that session is currently doing on the server.

The Logical Writes Tab

This tab displays the SQL that performed the greatest number of Logical Writes. The columns and fields are identical to the Logical Reads tab (see above).



The Physical Reads Tab

The Physical Reads Tab displays the SQL performing the greatest amount of physical reads. The columns and fields are identical to the Logical reads tab (see above).

The Total I/O Tab

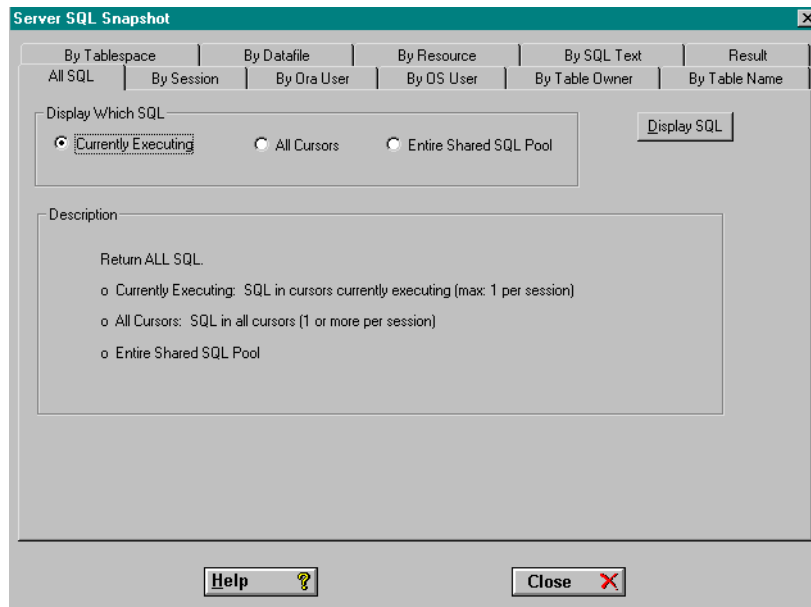
The Total I/O tab displays the SQL using the greatest total amount of I/O. The columns and fields are identical to the Logical Reads tab (see above).

Server SQL Snapshot

The Snapshot facility allows users to capture detailed information on the current state of the server. The Snapshot facility enables you to get the following information quickly and with minimal effort:

- Who is currently consuming the most resources.
- What SQL is currently executing.
- What SQL is associated with a session.
- What SQL is in the Shared SQL Pool.

Additionally, the information can be retrieved using a variety of search criteria. The following figure shows the **Server SQL Snapshot** dialog for a user with DBA privileges. Each tab represents a dialog in which you specify different criteria for searching the current SQL. For non-DBAs, only the All SQL, By Session, By Ora User, By OS User, By SQL Text, and Result tabs will be enabled. Non-DBA users can only capture and view SQL that they have privileges on.

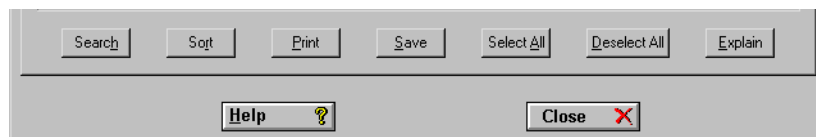


The Server SQL Snapshot Dialog

The Result Tab

When you capture SQL using the Snapshot facility, the SQL and its associated statistics are displayed on the Result tab. See the section *Statistics and Features of Result Sets* earlier in this chapter for a complete description of the statistics PAFO provides on captured SQL.

To take advantage of the other powerful features of the Snapshot function, use the buttons at the bottom of the Result tab window, as shown in the following figure:



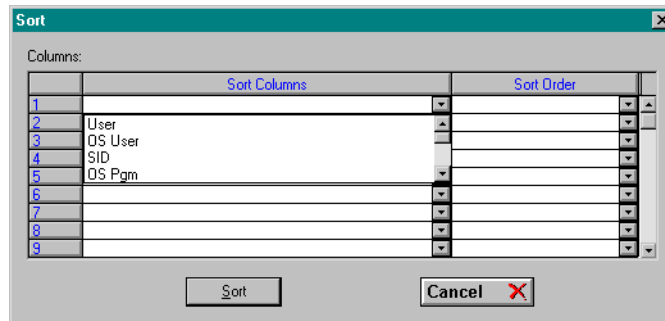
Click **Search** to bring up the Search dialog:

The Search Dialog

The Search dialog lets you search the captured SQL by a number of different criteria. You can select a User name from the User combo box. You can specify a range for the criteria by clicking one of the arrows and selecting either a greater than (>) or less than (<) operator. Then enter the value you want to search against in the Value column.

You can specify as many or as few search criteria as you want. For instance, the above example will search for Sorts per Statement greater than 1, and Physical Reads greater than 100.

Click **Sort** to bring up the Sort dialog:



Here you click on an arrow in the Sort Columns field to display the drop down scrollable menu as shown above. Use the Sort dialog to choose the criteria by which you wish to sort the captured SQL. Select a value from the Sort Columns combo box. In the Sort Order column, choose Ascending or Descending from the combo box.

Click **Print** to send the selected SQL statement and statistics to your printer. This is useful for generating reports on the current status of the server.

Click **Save** to save the selected SQL and statistics to a file or to the repository. Later, you can analyze and generate optimization plans for the SQL.

Select All selects all the currently captured SQL and statistics.

Deselect All deselects whatever captured SQL you have selected.

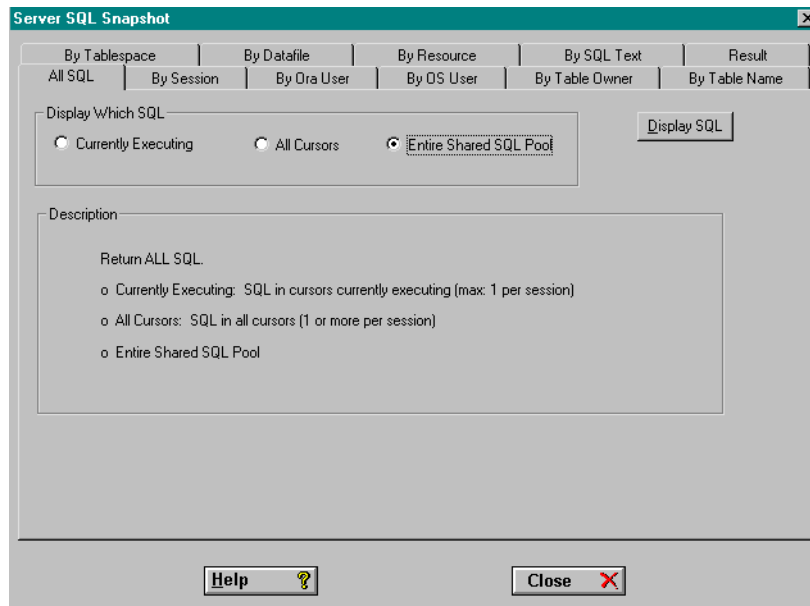
Click **Explain** to automatically copy the SQL into Plan Analyzer's SQL window and generate an optimization plan. This is a quick and powerful way to analyze SQL that may be causing a server slowdown.

The **Help** button brings up Plan Analyzer's online help.

Use **Close** to exit the Active Problem window.

The All SQL Tab

Use the All SQL tab to retrieve all the SQL on the server, without any filters.



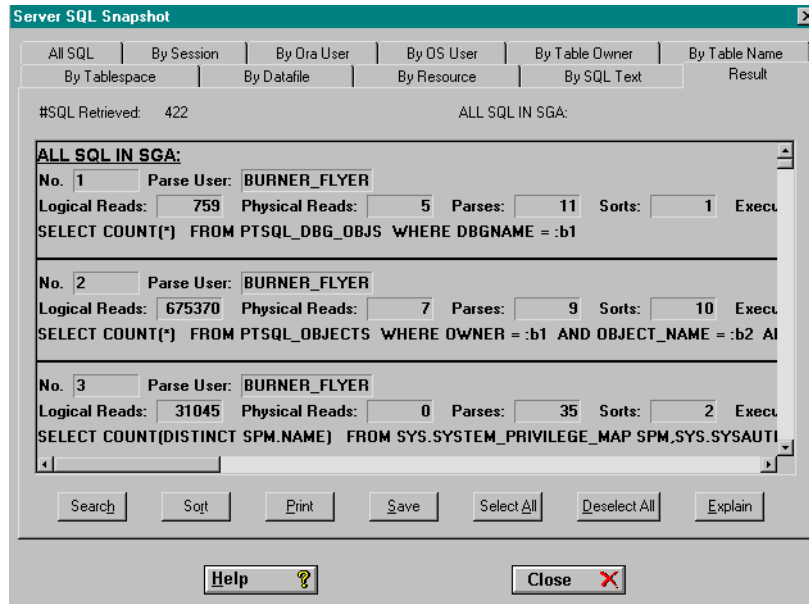
The All SQL Tab

Choose one of the three available radio buttons:

- **Currently Executing**, which retrieves only the SQL executing at the time the search is performed.
- **All Cursors**, which retrieves the SQL for all open cursors.
- **Entire Shared SQL Pool**, which returns all the SQL in the SGA (Shared Global Area).

In all three cases, users with DBA roles will retrieve all the SQL available in a specific search criteria, and users with non-DBA roles will retrieve only the SQL that they have permission to access.

The figure below illustrates a typical result set.

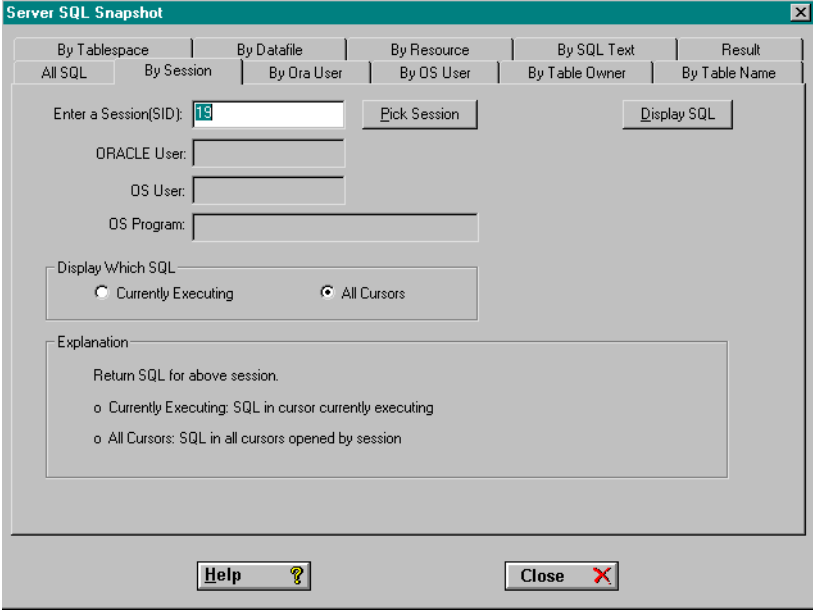


Typical Result Set from an All SQL Search

All the same data and data management options are available here as in the **Active Problem** feature (see above). See also the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

The By Session Tab

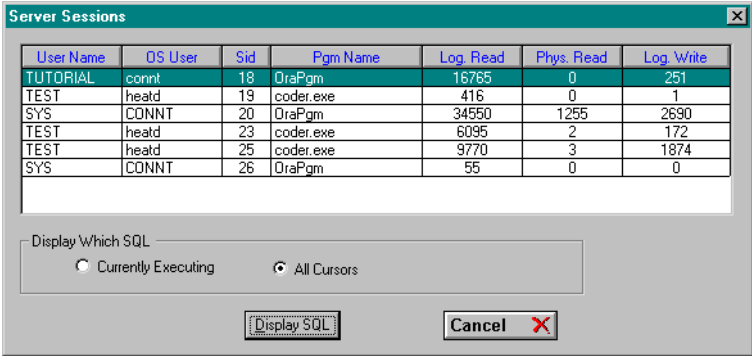
Use the By Session tab to capture SQL by Session ID. The figure below illustrates the By Session tab:



The **Server SQL Snapshot** dialog box features a tabbed interface at the top with the following tabs: **By Tablespace**, **By Datafile**, **By Resource**, **By SQL Text**, **Result**, **All SQL**, **By Session**, **By Ora User**, **By OS User**, **By Table Owner**, and **By Table Name**. The **By Session** tab is currently selected. Below the tabs, there is a section for session identification: **Enter a Session(SID):** with a text field containing '18', a **Pick Session** button, and a **Display SQL** button. Below this are fields for **ORACLE User:**, **OS User:**, and **OS Program:**. A section titled **Display Which SQL** contains two radio buttons: **Currently Executing** and **All Cursors**, with **All Cursors** selected. An **Explanation** box below provides details: 'Return SQL for above session.' followed by two bullet points: 'o Currently Executing: SQL in cursor currently executing' and 'o All Cursors: SQL in all cursors opened by session'. At the bottom are **Help** and **Close** buttons.

By Session Tab

Enter a Session ID in the **Enter a Session (SID)** field, or click the **Pick Session** button to bring up the Server Sessions dialog:



The **Server Sessions** dialog box displays a table of active sessions. Below the table is a **Display Which SQL** section with radio buttons for **Currently Executing** and **All Cursors**, and **Display SQL** and **Cancel** buttons at the bottom.

User Name	OS User	Sid	Pgm Name	Log_Read	Phys_Read	Log_Write
TUTORIAL	connit	18	OraPgm	16765	0	251
TEST	heatd	19	coder.exe	416	0	1
SYS	CONNIT	20	OraPgm	34550	1255	2690
TEST	heatd	23	coder.exe	6095	2	172
TEST	heatd	25	coder.exe	9770	3	1874
SYS	CONNIT	26	OraPgm	55	0	0

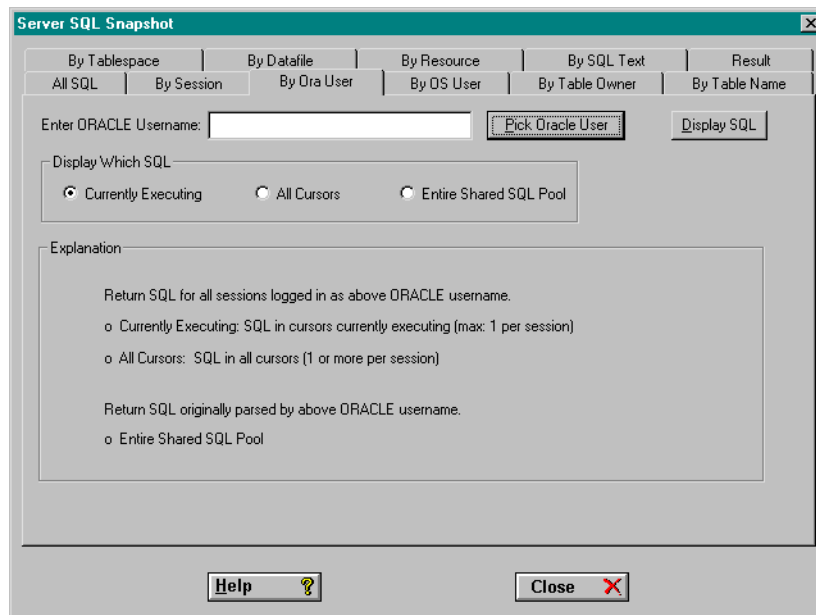
Server Sessions Dialog

Click on the row of the session for which you want to search. The **Log. Read, Phys. Read, and Log. Write** columns make it easy to identify the sessions consuming the most resources on the server. Choose either the **Currently Executing** or **All Cursors** radio buttons. From either the By Session tab, or the Server Sessions dialog, click the **Display SQL** button to commence the search. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

Return to the By Session tab to see the information for the Oracle User, OS User, and OS Program for the Session ID SQL you searched for.

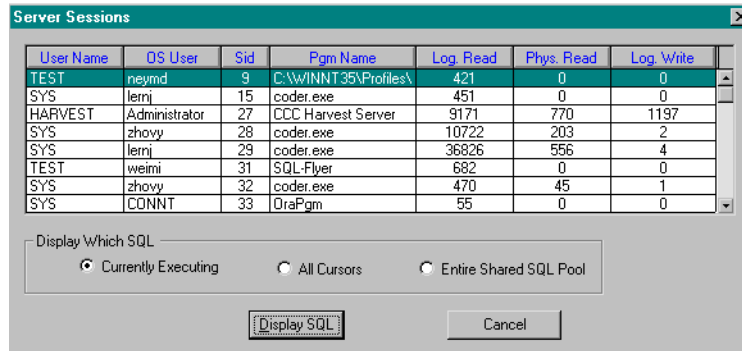
The By Ora User Tab

Use the By Ora User tab to search for SQL by ORACLE user name. The following figure illustrates the By Ora User tab:



By Ora User Tab

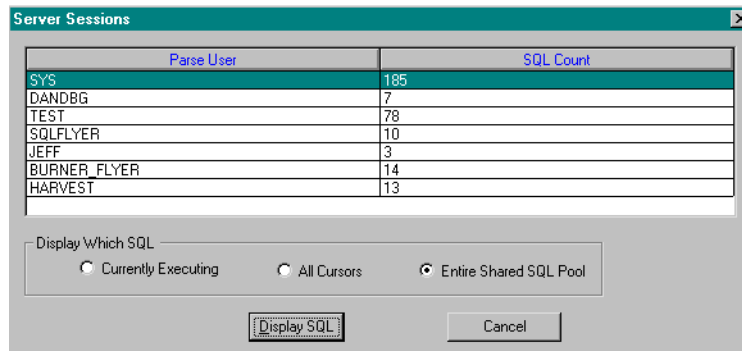
Enter an ORACLE user name in the **Enter ORACLE Username** field. Clicking the **Pick Oracle User** button, when either the **Currently Executing**, or **All Cursors** radio button is selected, brings up the By Oracle User - Server Sessions dialog.



By Oracle User Server Sessions Dialog

The spreadsheet at the top displays the User Name, the OS User, the SID number, the Program Name, the number of Logical Reads, Physical Reads and Logical Writes for the selected User. These columns help identify what sessions are likely candidates for optimization.

Selecting the **Pick Oracle User** button when the **Entire Shared SQL Pool** radio button is checked invokes the following dialog:



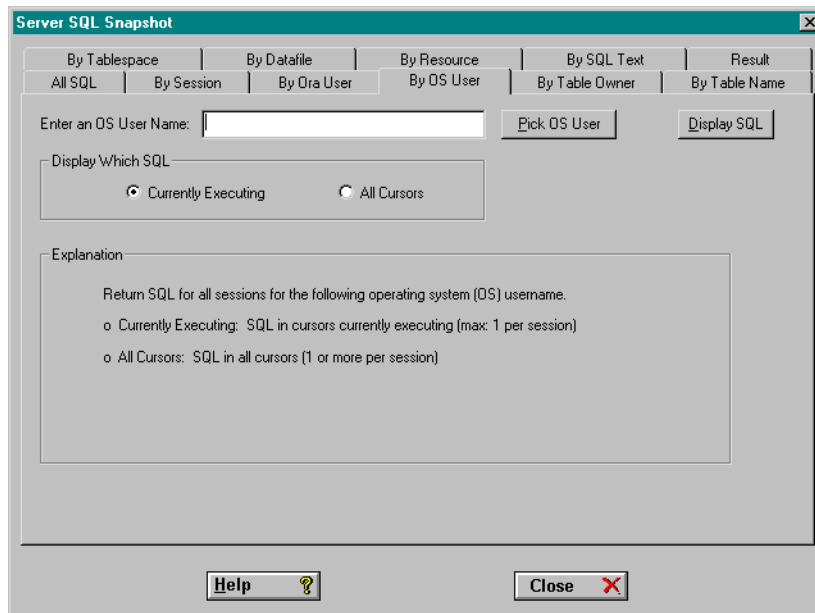
Parse User, SQL Count

The **Parse User** column shows who originally parsed the SQL. The **SQL Count** column shows how many SQL statements the user has parsed.

From either the Server Sessions dialog or the By Ora User tab, choose one of the three radio buttons to specify the SQL you want to search for: **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the result window.

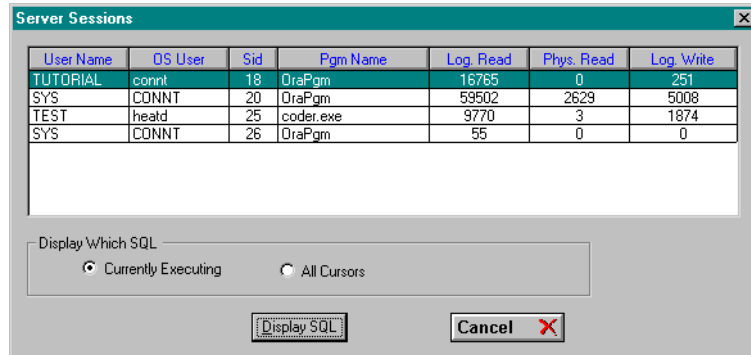
The By OS User Tab

Use the By OS User tab to search for SQL by OS User name. The following figure illustrates the By OS User tab:



By OS User Tab

Enter an OS user name in the **Enter an OS User Name** field, or click the **Pick OS User** button to display the Server Sessions dialog for the OS User name tab:



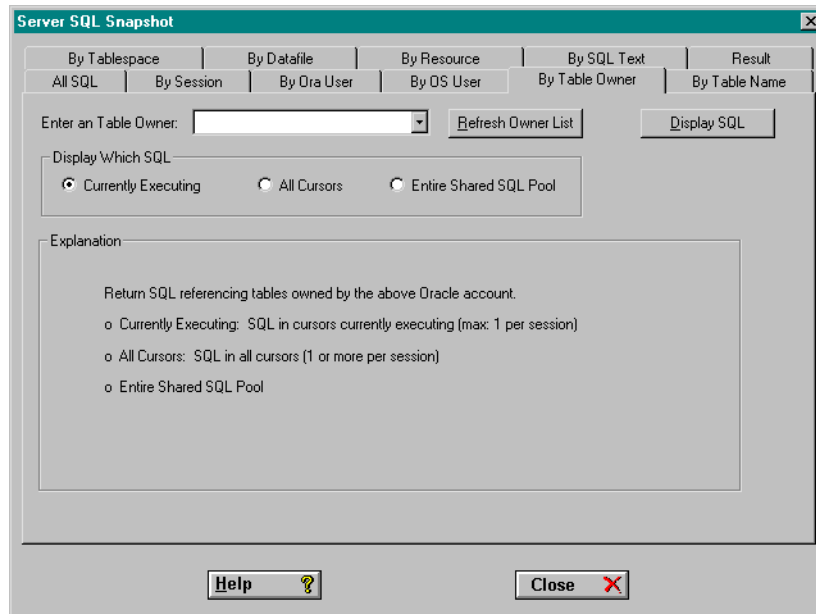
Server Sessions Dialog for the OS User Tab

The spreadsheet at the top displays the User Name, the OS User, the SID number, the Program Name, the number of Logical Reads, Physical Reads and Logical Writes for the selected OS User. This data makes it easy to spot SQL that is using excessive resources.

From either the Server Sessions dialog or the By OS User tab, choose one of the two radio buttons to specify the SQL you want to search for: **Currently Executing** or **All Cursors**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

The By Table Owner Tab

Use the By Table Owner tab to search for SQL that references objects owned by a specific account. This option is available only to DBAs. The following figure illustrates the By Table Owner tab:



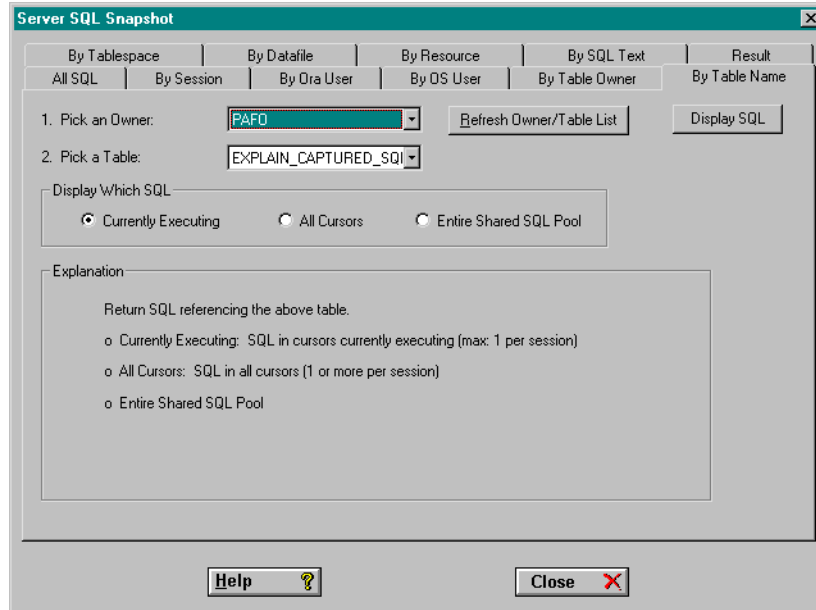
By Table Owner Tab

Choose one of the table owners from the dropdown menu. Click the **Refresh Owner List** button to get the most recent table owners from the server. Choose one of the three radio buttons to specify the SQL you want to search for: **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

Note that the **Entire Shared SQL Pool** option is available on ORACLE version 7.2 or higher.

The By Tablename Tab

Use the By Tablename tab to search for SQL that references a specific table. This option is only available to DBAs. The following figure illustrates the By Tablename tab:



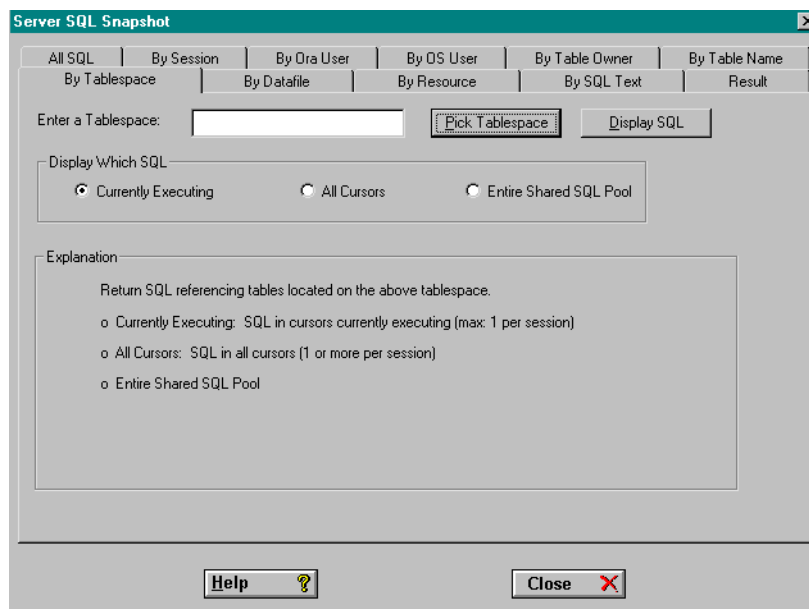
By Tablename Tab

First, pick a table owner from the combo box. Then choose a table from the **Pick a Table** combo box, which is updated with the owner's tables. Clicking the **Refresh Owner/Table List** button updates the lists with the most current tables from the server. Choose one of the three radio buttons to specify the SQL you want to search for: **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

Note that the **Entire Shared SQL Pool** option is available on ORACLE version 7.2 or higher.

The By Tablespace Tab

You can search the for SQL that is referencing objects in a particular tablespace. Note that this option is only available to DBAs. The following figure illustrates the By Tablespace tab:



By Tablespace Tab

Specify the tablespace by entering the tablespace name in the window, or click the **Pick Tablespace** button. This brings up the Server Tablespace Dialog:

Tablespace	Size (KB)	Blocks Read	Read Time (msec)	Block Writes	Write Time (msec)
ROLLBACK_DATA	25360	45	112	5028	34823
SYSTEM	168720	13411	14322	6248	34968
TEMPORARY_DATA	117400	0	0	287	9819
USER_DATA	184080	129	201	726	4345

Full Table Scans are performed with Multi Block Reads equal to 32.
Other reads are due to indexed searches.

Requests % Blocks %

Multi Block Reads: 0 0 0 0

Single Block Reads: 45 100 45 100

Display Which SQL

☒ Currently Executing ☐ All Cursors ☐ In Shared SQL Pool

Display SQL Cancel

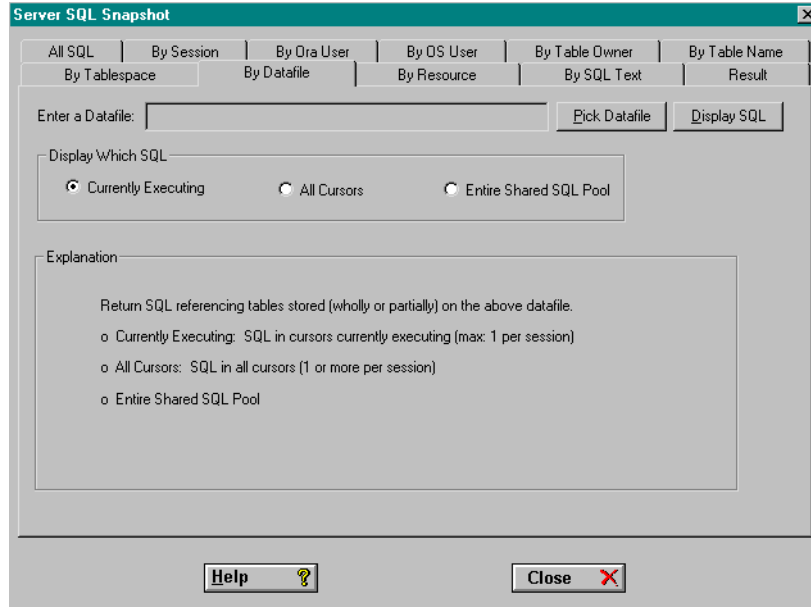
The Server Tablespaces Dialog

The spreadsheet at the top of the window displays the Tablespace Name, the Size (in bytes), the Blocks Read, the Read Time (in milliseconds), the Block Writes, and the Write Time (in milliseconds). To select a table space, click anywhere in its row. Below the top spreadsheet is the Multi Block Reads and Single Block Reads data for the datafile you have selected. The Multi Block Reads information shows the number of requests and the percentage of all requests that performed multi-block reads. This is indicative of the number of full table scans performed on objects in this tablespace. A tablespace with a high number of multi-block reads should be examined.

Once the tablespace is selected (in either the By Tablespace window or the Server Tablespace dialog), choose the SQL you want returned by clicking one of the radio buttons - **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with the parameters you have specified, click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

The By Datafile Tab

Use the By Datafile tab to search for SQL statements that reference objects on a particular datafile. This option is only available to DBAs. The following figure illustrates the By Datafile tab:



By Datafile Tab

Enter a datafile name in the **Enter a Datafile** field, or click the **Pick Datafile** button to display the Server Datafiles dialog:

File Name	Tablespace	Size (KB)	Blocks Read	Read Time (msec)	Block Writes	Write Time (msec)
C:\ORANT\DATA	ROLLBACK_DATA	5000	36	125	603	3471
C:\ORANT\DATA	ROLLBACK_DATA	5000	28	77	214	1129
C:\ORANT\DATA	ROLLBACK_DATA	5120	142	347	1398	6186
C:\ORANT\DATA	SYSTEM	10000	4242	6075	1704	7289
C:\ORANT\DATA	SYSTEM	153600	12554	18324	5095	15037
C:\ORANT\DATA	TEMPORARY_DA	10000	0	0	139	754
C:\ORANT\DATA	TEMPORARY_DA	5000	0	0	0	0
C:\ORANT\DATA	TEMPORARY_DA	99328	0	0	0	0

Full Table Scans are performed with Multi Block Reads equal to 16.
Other reads are due to indexed searches.

Requests % Blocks %

Multi Block Reads: 0 0 0 0

Single Block Reads: 36 100 36 100

Display Which SQL

☒ Currently Executing ☐ All Cursors ☐ Entire Shared SQL Pool

Display SQL Cancel

The Server Datafiles Dialog

Click on the row for the datafile you want to select. The spreadsheet at the top of the window displays the Datafile Name, the Tablespace it is part of, the Size (in kilobytes), the Blocks Read, the Read Time (in milliseconds), the Block Writes, and the Write Time (in milliseconds). Below the top spreadsheet is the Multi Block Reads and Single Block Reads data for the datafile you have selected. The Multi Block Reads information shows the number of requests and the percentage of all requests that performed multi-block reads. This is indicative of the number of full table scans performed on objects in this tablespace. A tablespace with a high number of multi-block reads should be examined.

From either the Server Datafiles dialog or the By Datafile tab, choose one of the three radio buttons to specify the SQL you want to search for: **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

The By Resource Tab

Use the By Resource Tab to search for currently executing SQL by specifying resource usage criteria. Any SQL that matches the criteria you select is retrieved. This option is available only to DBAs.

The following figure illustrates the By Resource Tab.

Server SQL Snapshot

By Tablespace | By Session | By Ora User | By OS User | By Table Owner | By Table Name

By Tablespace | By Datafile | **By Resource** | By SQL Text | Result

Search Column	Operator	Value	Max Value	Min Value
Logical Reads	<	194392	194392	0
Executions			4860	0
Curr Users			0	0
Parses			4860	0
Physical Reads			1998	0
Sorts	>	1	24	0
Versions			2	0
Share Mem			403850	919

Show Range

Display SQL

Display Which SQL

☒ Currently Executing ☐ All Cursors ☐ Entire Shared SQL Pool

Explanation

Return SQL which meets resource usage criteria specified above.

- o Currently Executing: SQL in cursors currently executing (max: 1 per session)
- o All Cursors: SQL in all cursors (1 or more per session)
- o Entire Shared SQL Pool

Help ? Close X

The By Resource Tab

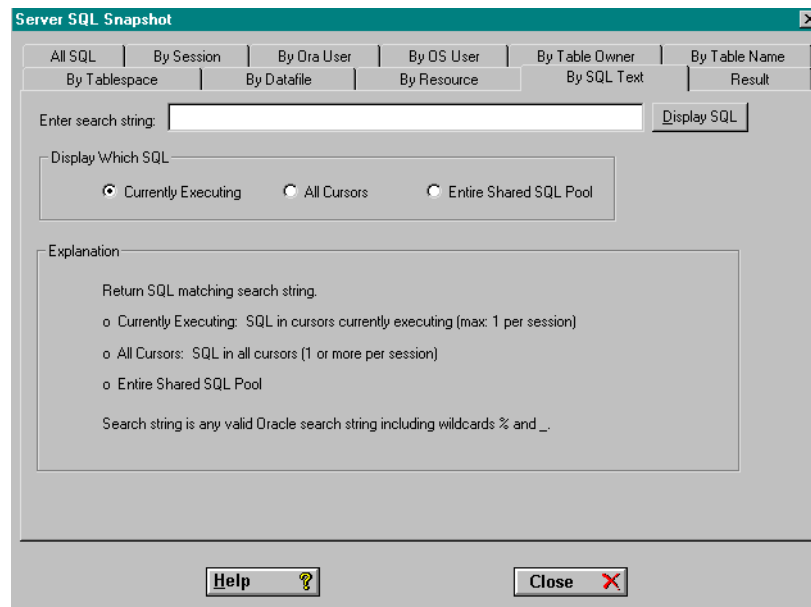
You can specify a range for the criteria in the spreadsheet by clicking one of the arrows and selecting either a greater than (>) or less than (<) operator. Then enter the value you want to search against in the Value column. Clicking the **Show Range** button displays the current values from the server.

You can specify as many or as few search criteria as you want. For instance, the above example will search for SQL by specifying Logical Reads less than 194392, and Sorts greater than 1.

Choose one of the three radio buttons to specify the SQL you want to search for: **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

By SQL Text Tab

Use the By SQL Text tab to search for SQL by a string of text. The following figure illustrates the By SQL Text tab:



The By SQL Text Tab

Enter a string from the SQL you are searching for in the **Enter Search String** field. For example, to search for all SQL that has the string 'ptsql' in the text, enter the search string '%ptsql%'. To search for all SQL that starts with 'select name, date', enter the search string 'select name, date%'. Note that the search string is case-sensitive.

Choose one of the three radio buttons to specify the SQL you want to search for: **Currently Executing**, **All Cursors**, or **Entire Shared SQL Pool**. When you are satisfied with your selection click the **Display SQL** button. The SQL is returned on the Result tab. See the section on *The Result Tab* earlier in this chapter for a detailed description of the returned SQL fields and other features of the Result window.

Server SQL Monitor

The Server SQL Monitor facility is a powerful new feature in Plan Analyzer that allows DBAs to capture SQL over a specified length of time. While the Snapshot facility captures SQL in one time slice, the Monitor facility actually monitors the server over time - capturing SQL, computing statistics, and storing both for later review and/or analysis. In addition to capture in the Interactive mode, Monitor jobs can be submitted to the ORACLE batch queue where the job will execute at a later time. It is also possible to set up a batch job to execute repeatedly.

The Monitor feature is available only to DBAs or users with DBA privileges.

Top SQL Resources

The Top SQL Resources menu option allows DBAs to capture a specified number of the SQL statements using the most CPU, and performing the most logical reads and writes. Thus, if you specify, for example, the top five SQL statements, Plan Analyzer will return the top five statements according to the three criteria mentioned above. In this case, you may capture as many as fifteen SQL statements, i.e., the five top users according to the three criteria. However, if the same SQL statement is the top user of *both* CPU and I/O, Plan Analyzer will show the SQL only once.

The figure below illustrates the Set up tab of the Monitor window. Use the Set up tab for both interactive and batch jobs.

The screenshot shows the 'Monitor SQL' dialog box with the 'Set up' tab selected. The 'Capture Result' sub-tab is also active. The 'Collection Name' field contains 'TOM2'. The 'Capture Top' field is set to '5' and the unit is 'SQL'. The 'Include SQL executed by SYS' checkbox is checked. The 'Monitor Frequency' group box contains two fields: 'Time between samples' set to '5' with a 'Seconds' dropdown, and 'Monitor Run Time' set to '20' with a 'Seconds' dropdown. The 'Description' text area contains 'Top Resources'. At the bottom are 'Help' and 'Close' buttons.

Server SQL Monitor Set Up Tab

Set up Parameters

In the **Collection Name** field, enter a unique name for the SQL you will be capturing. If the name already exists, Plan Analyzer will prompt you to enter a unique name. The collection names are unique only to your ORACLE account. A different ORACLE account can use the same name. In the **Capture Top *n* SQL** field, specify how many of the top CPU and I/O consuming statements you wish to capture. With the **Include SQL executed by SYS** check box you can choose to capture or ignore the SQL being executed by the server itself: Check the box if you want to include the background server SQL.

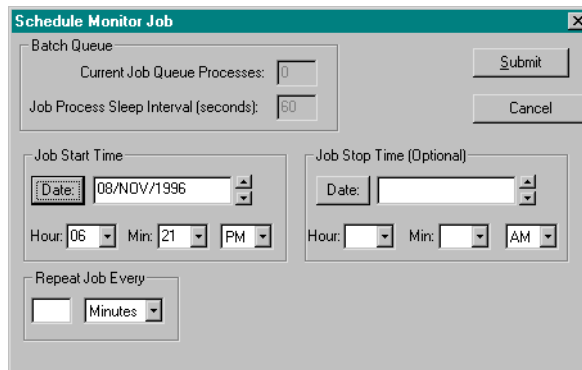
The **Monitor Frequency** group box allows you to specify the time between sample captures, as well as the duration of the sampling period. For example, in the figure above, Monitor is set to take a snapshot of the server every 5 seconds for 20 seconds. Enter values in the first column of boxes, and use the combo boxes in the second column to specify Seconds, Minutes, or Hours.

Enter any desired information regarding the SQL you are capturing in the **Description** field.

For jobs in interactive mode, click the **Run Now** button to begin the capture.

Jobs in Batch Mode

Batch executions are enabled for the criteria specified in the Monitor Set up window by clicking the **Schedule** button. This brings up the Schedule Monitor Job dialog. Here you specify the scheduling parameters for a batch job. The figure below illustrates:

The image shows a Windows-style dialog box titled "Schedule Monitor Job". It has a "Batch Queue" section with "Current Job Queue Processes" (a text box with "0") and "Job Process Sleep Interval (seconds)" (a text box with "60"). There are "Submit" and "Cancel" buttons. Below is a "Job Start Time" section with a "Date:" field showing "08/NOV/1996", and "Hour:", "Min:", and "PM" dropdown menus. To the right is a "Job Stop Time (Optional)" section with a "Date:" field and "Hour:", "Min:", and "AM" dropdown menus. At the bottom is a "Repeat Job Every" section with a text box and a "Minutes" dropdown menu.

Schedule Monitor Job Dialog

The Batch Queue group box contains details on the Current Job Queue Processes on the server, and the Job Process Sleep Interval.

The **Job Start** and the **Job Stop Time** fields indicate when you want your batch job to start and stop. Position the cursor in the date field and click the arrows on the right to increment or decrement the portion of the date where the cursor is positioned. Use the combo boxes to specify the Hour, Minute, and AM or PM fields. Clicking the **Date** button in either field invokes the calendar, with which you may specify the values for the Year, Month, and Day:



The arrows at the top increment or decrement the year and month. To select a day, click on the desired number. Click **OK** to apply the changes.

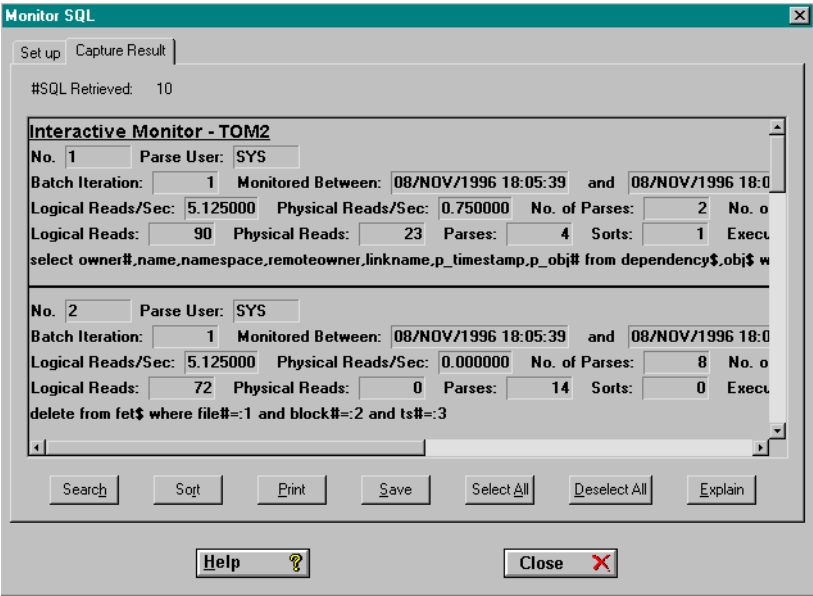
If no value is specified for the **Job Stop Time**, the job will run continuously until stopped by the user.

The **Repeat Job Every** group box allows you to specify the interval at which the job will execute again. Enter a value in the first field, and use the combo box to specify Minutes, Hours, or Days. If you only want to run the job once, do not enter a repeat value.

When you are satisfied with the scheduling information, click the **Submit** button to submit your batch job to the ORACLE batch queue. Once a job is submitted, the application can be closed. To review SQL captured during a batch job, use the Review dialog.

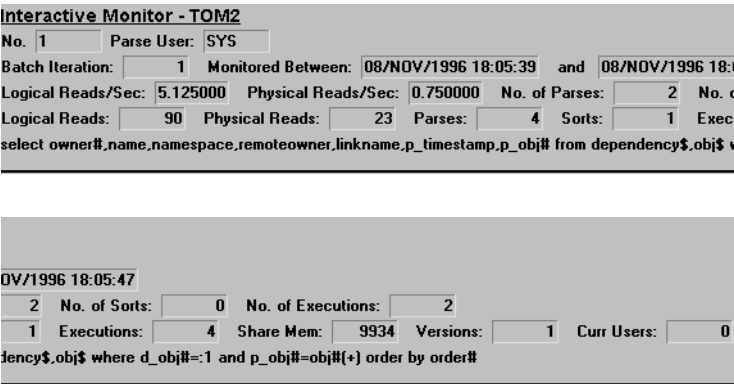
Capture Results

After submitting your job, the Monitor facility returns the SQL captured according to the criteria you specified. The figure below illustrates a typical Monitor capture result:



Typical Capture Result

In addition to the SQL statement itself, Monitor returns a variety of other statistics regarding the captured statement. The following table and figures illustrate and describe these statistics. The first figure shows the result set as it appears by default. The second figure shows the additional results visible when the window is scrolled to the right:



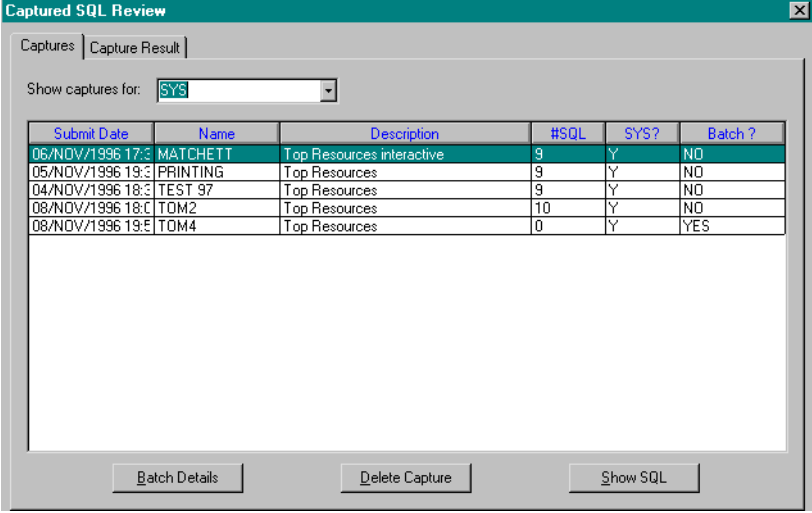
Statistic Name	Definition
No.	Shows the ordinal number of the SQL statements in the display.
Parse User	The user that first parsed the SQL.
Batch Iterations	The number of times the job ran, if it was a batch job.
Monitored Between...And	The date and time the SQL was first collected by Monitor.
Logical Reads/Sec.	The number of logical reads per second.
Physical Reads/Sec.	The number of physical reads per second.
No. of Parses	The number of parses during the time Monitor collected the SQL.
No. of Sorts	The number of sorts performed during the time Monitor collected the SQL.
No. of Executions	The number of times the SQL executed during the time Monitor collected the SQL.
Logical Reads	The number of reads from ORACLE's buffer cache.
Physical Reads	The number of reads that required disk access.

Statistic Name	Definition
Parses	The number of times the statement has been parsed.
Sorts	The number of sorts the query performed.
Executions	The number of times the statement was executed.
Share Memory	The size of shareable memory used by the statement in the shared pool.
Versions	The number of loaded versions.
Current Users	The number of current users.
Optimization	The Optimization mode for the SQL. Available on ORACLE 7.3 or higher.
In addition to these statistics, the text of the SQL statement itself is displayed below the statistics.	

The buttons at the bottom of the Capture Result tab are identical to those in the Snapshot result set window. See the *Result Tab* section earlier in this chapter for a complete description.

The Review Window

The Review menu item enables you to review details of the various interactive or batch jobs currently stored in Monitor, as well as the SQL captured by them. The following figure illustrates the Captured SQL Review window:



The screenshot shows a window titled "Captured SQL Review" with two tabs: "Captures" and "Capture Result". The "Captures" tab is active. Below the tabs is a dropdown menu labeled "Show captures for:" with "SYS" selected. Below this is a table with the following data:

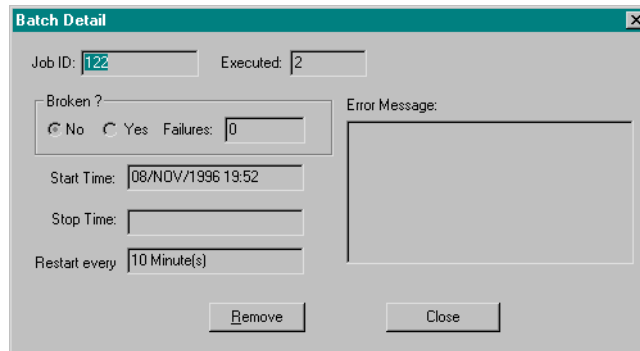
Submit Date	Name	Description	#SQL	SYS?	Batch ?
06/NOV/1996 17:3	MATCHETT	Top Resources interactive	9	Y	NO
05/NOV/1996 19:3	PRINTING	Top Resources	9	Y	NO
04/NOV/1996 18:3	TEST 97	Top Resources	9	Y	NO
08/NOV/1996 18:0	TQM2	Top Resources	10	Y	NO
08/NOV/1996 19:5	TQM4	Top Resources	0	Y	YES

Below the table are three buttons: "Batch Details", "Delete Capture", and "Show SQL".

Captured SQL Review Window

The **Submit Date** column shows the date the job was submitted. The **Name** column shows the unique name given the job in the Collection Name field of the Set up window. The **Description** column contains the text entered in the Description field of the Set up window. The **#SQL** column shows the number of SQL statements captured by the job. The **SYS?** column tells whether the job included background server SQL in its collection criteria. The **Batch?** column indicates whether the job is or is not a batch job.

Clicking the **Batch Details** button raises the Batch Detail dialog:

The image shows a 'Batch Detail' dialog box with a title bar containing a close button. The dialog contains several input fields: 'Job ID' with the value '122', 'Executed' with the value '2', 'Broken?' with radio buttons for 'No' (selected) and 'Yes', and a 'Failures' field with the value '0'. Below these are 'Start Time' (08/NOV/1996 19:52), 'Stop Time' (empty), and 'Restart every' (10 Minute(s)). On the right is a large 'Error Message' text area. At the bottom are 'Remove' and 'Close' buttons.

Batch Detail Dialog

The **Job ID** field shows the unique number given the job by ORACLE. The **Executed** field shows the number of times the job executed. The **Broken?** field indicates whether the job was corrupted in any way by the server, and also indicates the number of failures, if any. The **Error Message** field shows any errors invoked by the job. The **Start Time**, **Stop Time**, and **Restart every** fields show the schedule details entered in the Schedule Monitor Job window when the job was defined. Clicking the **Remove** button removes the job from the server's batch queue. The **Close** button closes the Batch Details window.

The **Delete Capture** button on the Captured SQL Review window deletes the job from the repository. The **Show SQL** button displays the SQL captured by the job, and displays it along with its statistics on the Capture Results tab. See the section *Capture Results*, above, for a detailed description of the Capture Results tab.



Plan Menu

The Plan menu items are associated with optimization plans. Different menu items appear depending on whether the operation mode is Standard or Expert. Expert mode permits you to save and reuse plan configurations. Under Standard mode, plans apply only to the current SQL being tested.

Standard Mode Plans	7-3
Expert Mode Plans	7-3
Rule	7-4
Cost First Row	7-5
Cost All Rows	7-6
Hints	7-6
Invoking the Hints Dialog	7-8
Adding New Hints	7-9
Types of Hints	7-10
Step Details	7-36
What are the Database Objects in Each Step?	7-36
Distributed Queries	7-40
Parallel Queries	7-41

Cost, Rows, and Bytes	7-42
Visualize	7-43
Explaining Each Step	7-44
Traversing the Plan	7-46
Compare	7-46
Standard Translation	7-49

Standard Mode Plans

In Standard mode, the optimization plans are invoked from the **Evaluate** menu, since the plans are created in conjunction with the execution. See The *Evaluate Menu* chapter for more information. The Standard Mode **Plan** menu contains all other functions associated with optimization plans, such as **Specify Hints** and **Compare**.

The menu items for Standard mode are:

- Specify Hints
- Step Details
- Visualize
- Compare

Expert Mode Plans

The Expert Mode **Plan** menu contains the following options:

- Rule
- Cost First Row
- Cost All Rows
- Hints
- Step Details
- Visualize
- Compare
- Standard Translation

ORACLE7 introduced cost-based optimization, and introduced hints to assist in customizing optimization plans. Plan Analyzer separates optimization into four types: **Rule**, **Cost First Row**, **Cost All Rows**, and

Hints. When the optimization plan is Rule-based, hints embedded in the SQL are disabled, providing a “pure” optimization plan based on rules. The same holds true for First Row-based optimization: The plan is cost-based with a goal of First Row, and no hints are enabled. Hints are only evaluated under Hints-based optimization. Cost First Row produces an optimization plan based only on the underlying statistics. The goal is to get something back to the user quickly. Cost All Rows produces an optimization plan that produces the best throughput; that is, it gets all rows returned as quickly as possible. If the SQL statement has hints embedded in it, they can be evaluated only in Hints mode. Producing all plans except Hints provides considerable insight into how ORACLE thinks the SQL should be optimized. If you think a good alternative is being missed, use Hints to get ORACLE to optimize it differently.

There are two ways to produce a specific plan. One way is to choose **Rule**, **Cost First Row**, or **Cost All Rows** from the **Plan** menu. This immediately produces the associated plan. Choosing **Plan**, **Hints** displays a submenu with the **Specify Hints** and **Execute Hints** options. **Execute Hints** produces the Hints plan. **Plan**, **Specify Hints** assists in entering the hints. You can also use these toolbar icons:



The buttons, from left to right are: **Rule Plan**, **First Row Plan**, **All Rows Plan**, and the three **Hints Based Plans**.

Rule

This menu option is available only in Expert mode.

Rule-based optimization was the only optimization mode available prior to ORACLE7. Rule-based optimization is based on a set of heuristic rules. The rules are based solely on access paths available, not statistics. (See Oracle's *ORACLE7 Server Concepts Manual*, Chapter 13 for a listing of all rules.) Rule-based optimization knows nothing of size of the tables, or selectivity of the indexes. Worse, rule-based optimization will force the use of an index even if the best path is to perform a full table scan. To

prevent ORACLE from choosing indexes in rule-based optimization, users typically resort to modifying the SQL statement to inhibit the use of an index path.

The following example shows how to modify the SQL to inhibit the use of the index. The following query utilizes the index on the SEX column of the HR.EMPLOYEES table.

```
SELECT * FROM HR.EMPLOYEES  
WHERE SEX = 'M'
```

Rule-based optimization would use the index on SEX, though the better solution would be to perform a full table scan. To inhibit the use of the index on SEX, concatenate a null character (two single quotes) to the column name as follows:

```
SELECT * FROM HR.EMPLOYEES  
WHERE SEX||' ' = 'M'
```

To inhibit the use of an index on a numeric column, add zero to the numeric column.

Since cost-based optimization is still quite new, rule-based optimization, in many cases, produces the better plan. Choose **Plan, Rule** to produce a rule-based optimization plan.

Cost First Row

This menu option is only available in Expert mode.

First Row based optimization is cost-based with the goal of best response time: It returns something to the application as soon as possible. This means the first row of the result set is returned as fast as possible at the expense of a longer overall time and more resources to return all rows. Typically this is the desired goal for interactive applications, where a user is waiting for some type of response. The cost, as perceived by ORACLE, is listed in the menu bar of the associated window.

First, do not consider the First Rows approach for anything but a query. All other SQL statements should be All Rows based, because ORACLE doesn't return anything in those cases until the entire statement is processed. Additionally some queries will have the same optimization plans for both First Rows and All Rows, simply because there are no means of returning a single row until all are processed. For example, a GROUP BY query must be totally completed before the first row can be returned. Typically the First Rows approach benefits joins, but not GROUP BY, ORDER BY, and set operations.

Choose **Plan, Cost First Row** to produce the Cost First Row optimization plan.

Cost All Rows

This menu option is only available in Expert mode.

All Rows-based optimization is cost-based with the goal of best throughput, meaning rows will be returned faster, with a slower initial response. The goal is to complete the job, and to not worry about showing progress, as the First Rows optimization approach does. Typically, this is the desired goal for batch reports, where the report is not read until the entire report is complete. This is also the goal applied to all DELETE, UPDATE and INSERT statements when cost-based optimization is used.

Choose **Plan, Cost All Rows** to produce the Cost All Rows optimization plan.

Hints

In Expert mode, the Plan menu contains the Hints option, which in turn contains sub-options to specify or execute any of the three possible Hints Plans. In Standard mode, The Plan menu contains the Specify Hints options with no further sub-options. (See *Invoking the Hints Dialog* in this chapter.)

ORACLE has a sophisticated optimizer, but like other RDBMSs, it does not attempt to search every possible optimization path for the minimum cost. That would potentially be disastrous, since the time to execute might be less than the time to optimize. ORACLE does not know the exact distribution of the data values per column, and when the analysis is performed using an estimate, the data is even more inaccurate. If ORACLE must make a choice between multiple paths, will it make the right choice, or would one of the other paths have provided better performance? Since the developer knows more about the data than ORACLE does, there is a need for hints.

Hints are supposed to direct the optimizer to use a specific access path, if the access path exists. But this is not always the case. Sometimes you must combine hints in order to make one hint work. Platinum regards hints as suggestions to the optimizer, since the optimizer can refuse to use the hint. Plan Analyzer provides the following features to guide you through the specification of hints:

- **Eliminates syntactically incorrect hints**

Users often make syntax mistakes when specifying hints. ORACLE regards these defective hints as comments and doesn't inform the user. When you choose a hint, Plan Analyzer displays the parameter requirements for the hint in the status bar. Unless the minimal hints are specified, the hint will not be accepted.

- **Inhibits hints that are contextually incorrect**

Plan Analyzer assists you in choosing appropriate hints and in setting the parameters for those hints. Plan Analyzer does not allow a hint such as HASH if none of the objects in the query are members of a hashed cluster. If the query is not a join, it does not allow hints requiring multiple tables, such as ORDERED. If you select the AND-EQUAL hint, only single column indexes are listed for a specific table, and the table must have more than one single column index available. With all hints,

- **Clearly identifies all locations where hints may be applied**

Each subquery can have its own set of hints. For example, the following query is rule-based on the nested select, but cost first row on the outer query.

```
SELECT /*+ INDEX (e, i_emp_birthdate) */ FROM
      hr.employees e

WHERE birthdate BETWEEN '01-JAN-78'
      AND '01-JAN-79'

AND emp_seq IN
      (SELECT /*+ FULL ( d) */ emp_seq
      FROM hr.dependents d
      WHERE birthdate > '01-JAN-93')
```

- **Automates the parameter selection**

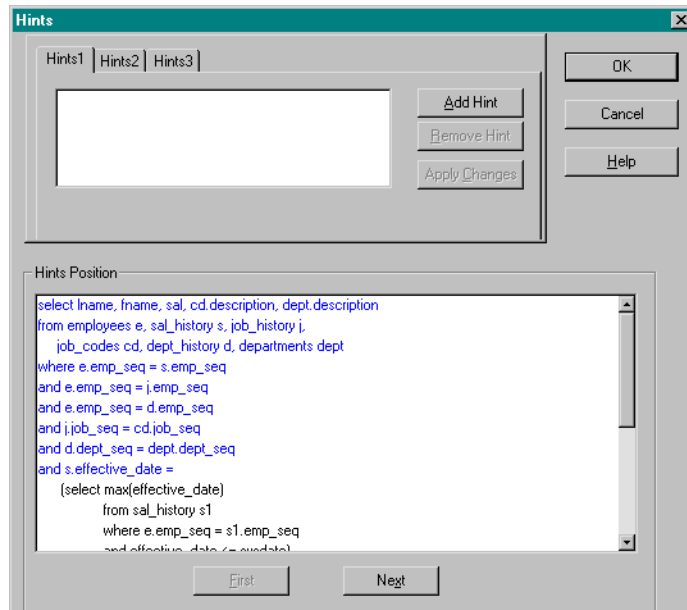
The most cumbersome part of specifying hints is the need to know the table names and associated indexes. If you use aliases in the FROM clause, Plan Analyzer uses the alias automatically, as dictated by ORACLE. It is even more difficult to know the available indexes and the columns in the indexes. By automating parameter selection, Plan Analyzer simplifies this process.


- **View utilization**

When you include views in the FROM clause, you can apply hints to the underlying tables. However you must first determine what are the underlying tables. Plan Analyzer tracks all views in the FROM clause and displays all indexes for tables in the initial FROM clause of the view.

Invoking the Hints Dialog

In Expert mode, the **Plan, Hints** menu option displays the following sub-options: **Specify Hints**, **Execute Hints1**, **Execute Hints2**, and **Execute Hints3**. Select **Plan, Hints, Specify Hints** to display the Hints dialog:

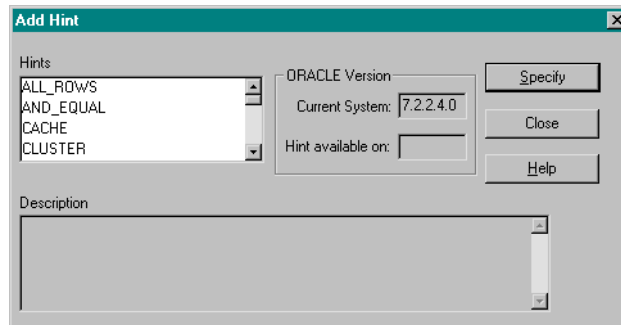


In Standard mode, the **Specify Hints** option appears directly under the **Plan** menu item. In either mode, clicking the following toolbar icon also invokes the Hints dialog: 

The Hints dialog displays the SQL statement in the list box with the first SQL module in blue. If the SQL module contains hints, the hints are displayed in the list box at the top. Use the **Hints1**, **Hints2**, or **Hints3** tabs to select which Hints plan you want to work with. The **Next** button selects the next SQL module, and the **First** button reselects the first SQL module. Use the **Add Hint** button to add new hints, and the **Remove Hint** button to delete the hint that is selected in the list box to the left. The **Apply Changes** button places the hints in the top list box into the selected SQL module.

Adding New Hints

Click **Add Hints** to add new hints to the selected SQL module (highlighted in blue). The Add Hint dialog displays:



All available hints are listed in the list box at the top left. The ORACLE Version group box to the right identifies the ORACLE version to which Plan Analyzer is currently connected (**Current System** field), and also the earliest ORACLE version to which the hint is applicable (**Hint available on** field). If the hint is not available on the current system, the hint is in red. When you select a hint, the Add Hint dialog displays the ORACLE versions for which the hint is available and a description of the hint and its purpose.

The RULE, NOCOST, FIRST_ROWS and ALL_ROWS hints do not take any parameters. Most other hints take parameters and have a specific dialog associated with them. When you select a RULE, NOCOST, FIRST_ROWS, or ALL_ROWS hint, the **Specify** button changes to **Add**, since there are no parameters to specify.

For each set of hints, an optimizer mode must be chosen in the top query. This optimizer mode affects the entire SQL statement, including all subqueries. The options are RULE, FIRST ROWS, and ALL ROWS. If an optimizer mode is not specified, the program will prompt you to specify one.

Types of Hints

This section describes all hints available as of ORACLE V7.3. Other hints will be added to new versions of Plan Analyzer as ORACLE provides them. For more detailed information about ORACLE hints, you can refer to the *Oracle7 Server Application Developer's Guide*.

ALL_ROWS

The ALL_ROWS hint either forces an All Rows plan or instructs ORACLE to use the other embedded hints to achieve an All Rows type of plan. An All Rows plan always chooses a merge join over a nested loop join, resulting in full table scans and sorts to achieve the overall best plan when the entire result set for the query must be returned.

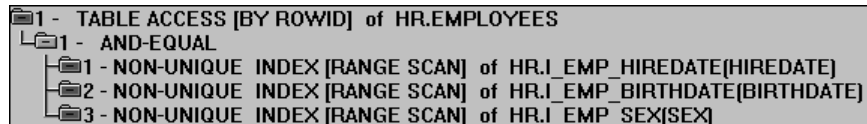
ALL_ROWS does not take any parameters and is applicable only to the first SQL module. The ALL_ROWS hint is colored red if the Add Hints dialog is displayed for any other SQL module.

AND_EQUAL

ORACLE can optimize multiple equality criteria on a single table if there are single column indexes on each column in the criteria. For instance, the following query has three equality criteria and each column in the criteria has an index defined on it.

```
SELECT * FROM employees
WHERE hiredate = '01-apr-86'
AND birthdate = '10-apr-52'
AND sex = 'M'
```

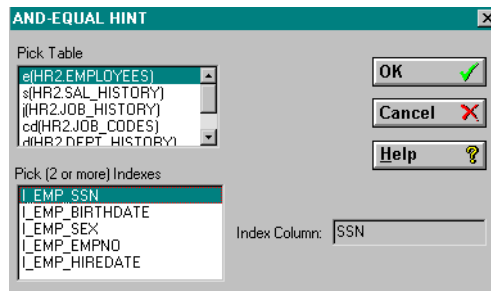
The Rule plan for the query follows:



The AND-EQUAL operation has three child operations, which are the three single-column indexes. ORACLE positions within each index, extracts the corresponding ROWIDs, and then performs the intersection of the ROWIDs before reading the rows. Using multiple indexes permits the rows to be qualified prior to actually retrieving the row. The number of indexes used determines how well qualified the rows are prior to

retrieval, but at a cost of more I/O to the indexes. Checking the object statistics in the Analysis dialog can help in deciding how many indexes to use.

The AND_EQUAL hint forces an AND-EQUAL operation if possible, but with more flexibility since the hint takes the indexes to be merged as parameters. The figure below illustrates the AND-EQUAL HINT dialog. Clicking one of the indexes displays all single-column indexes for the selected table. Selecting one of the indexes displays the name of the indexed column in the **Index Column** field. The AND-EQUAL hint requires one table name and two or more index names.



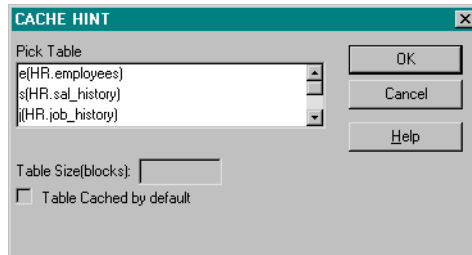
CACHE

ORACLE normally stores every data block read at the end of the LRU (Least Recently Used) chain in the SGA. This action becomes an issue when a full table scan is being performed. If the table is small and will be read by many applications, it is desirable to ensure that the data blocks are stored at the end of the LRU chain. The CACHE hint ensures this, regardless of the default setting specified when the table was created.

The CACHE hint takes a table name as its only parameter. Plan Analyzer lists all objects in the FROM clause. If the object does not translate to a single table, the object is colored yellow as a warning. When a table is highlighted, Plan Analyzer populates the **Table Size (blocks)** field with the number of blocks occupied by the table rows if the table is analyzed, or the total number of blocks allocated to the table if the table is not

analyzed. If the table is cached by default, the **Table Cached by default** check box is set. When the CACHE hint is specified for a table, the FULL hint is also added for the same table.

The CACHE HINT dialog is illustrated in the following figure.



CLUSTER

Use the CLUSTER hint to access a table in a cluster using the cluster index, or to join a second table that is clustered to the first without using the cluster index. In the first case, a criterion must exist on the table referencing the cluster key. In the second case, the tables must be joined over the cluster key, and the table parameter is the table in the second child operation of a nested loop. For instance, if TABLE_A and TABLE_B are joined together in the SQL as well as being members of the same cluster, then if TABLE_A is accessed by an index, TABLE_B rows are already in the same data block as the row retrieved for TABLE_A. Using a CLUSTER hint instructs ORACLE to avoid an index join to TABLE_B.

The following SQL statement contains two tables from the Human Resources tutorial database. The tables are members of the same indexed cluster, and the cluster key is on the EMP_SEQ column of both tables.

```
SELECT *
FROM employees e, sal_history s
WHERE e.emp_seq = 101
AND e.emp_seq = s.emp_seq
```

The Rule-based plan follows:

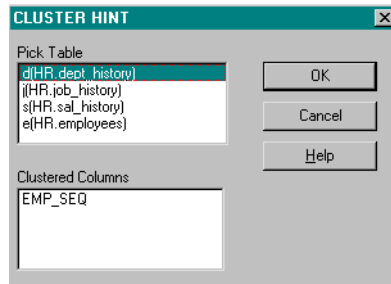
```

1 - NESTED LOOPS
  1 - TABLE ACCESS [CLUSTER] of HR.EMPLOYEES
    1 - CLUSTER INDEX [UNIQUE SCAN] of HR.I_EMP_EMPNO[EMP_SEQ]
    2 - TABLE ACCESS [CLUSTER] of HR.SAL_HISTORY
  
```

The cluster index, I_EMP_EMPNO accesses the EMPLOYEES table. After the corresponding EMPLOYEES data block is retrieved, the cluster access of SAL_HISTORY no longer requires the cluster index since the SAL_HISTORY rows are in the same data block.

Plan Analyzer lists all tables in the SQL module that are part of an indexed cluster (see following figure). The object is not listed if one of the objects in the FROM clause is a view of a join, or if it translates to a single object that the user does not have privileges on.

Select a table to display the clustered columns.



FIRST_ROWS

The FIRST_ROWS hint either forces a First Rows plan or instructs ORACLE to use the other embedded hints to achieve a First Rows type of plan. A First Rows plan chooses a nested loop join over a merge join, to limit full table scans and sorts, with the goal of returning a subset of rows as quickly as possible.

The hint is ignored by ORACLE if the SQL statement does not contain one of the following:

- set operators
- GROUP BY
- FOR UPDATE
- aggregate functions
- DISTINCT

FIRST_ROWS does not take any parameters, and is only applicable to the first SQL module. If the Add Hints dialog is displayed for any other SQL module, the FIRST_ROWS hint is colored red.

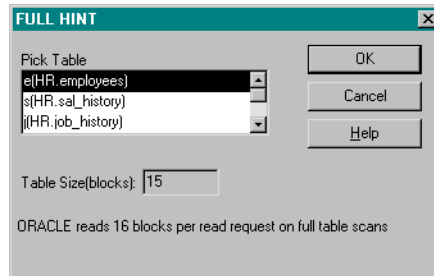
FULL

The FULL hint attempts a full table scan of a table. A full table scan should be considered on an All Rows plan where a majority of the rows will be retrieved. If the goal is a First Rows plan, combining the FULL hint with the ORDERED hint may be helpful if the full table scan is performed on the leading table with the goal of returning most of the rows. If the table is large, consider combining the NOCACHE hint.

If the full table scan resulting from the hint is not the first child operation of a nested loop join parent operation, the full table scan will be colored red as an alarm. In that case, if the table size is large (more than the B-tree depth of the join index), consider using a merge join or removing the FULL hint.

The FULL Hint dialog is shown in the next figure. Plan Analyzer lists all objects in the FROM clause. If the object does not translate to a single table, the object is colored yellow as a warning and the **Table Size (blocks)** field is disabled. When a table is highlighted, Plan Analyzer populates the **Table Size (blocks)** field with the number of blocks occupied by the table rows if the table is analyzed, or with the total number of blocks allocated to the table if the table is not analyzed.

ORACLE is optimized to perform full table scans by reading multiple data blocks per read request. The number of blocks read on each request is displayed at the bottom of the dialog. If the table consists of 40 data blocks and the read size for ORACLE is 8 blocks, then ORACLE will make eight requests for reads.



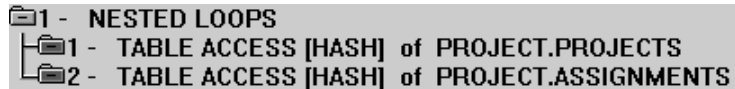
HASH

Use the HASH hint either to access a table in a cluster using the hash index, or to join a second table that is clustered to the first without using the hash index. In the first case, a criterion must exist on the table referencing the cluster key. In the second case, the tables must be joined over the cluster key, and the table parameter is the table in the second child operation of a nested loop. For instance, if TABLE_A and TABLE_B are joined together in the SQL as well as being members of the same cluster, then if an index accesses TABLE_A, TABLE_B rows are already in the same data block as the row retrieved for TABLE_A. Using a HASH hint instructs ORACLE to avoid an index join to TABLE_B.

The following SQL statement contains two tables from a Project Tracking database. The tables are members of the same hash cluster, and the cluster key is on the PROJ_SEQ column of both tables.

```
SELECT *
FROM assignments a, projects p
WHERE p.proj_seq = 101
AND a.proj_seq = p.proj_seq
```

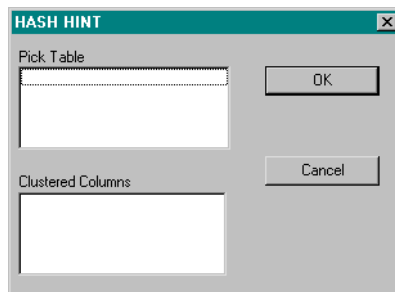
The Rule-based plan follows:



This differs from the CLUSTER hint in that no step in the plan shows that the hash index is used to access the PROJECTS table. Since the hash index is not really an index, but an algorithm, it makes sense. Once the hash algorithm is used to access the PROJECTS row, the corresponding ASSIGNMENTS rows are in the same data block and do not require the use of the hash algorithm to complete the join.

Plan Analyzer lists all tables in the SQL module that are part of a hash cluster (see figure below). If one of the objects in the FROM clause is a view of a join, or translates to a single object that the user does not have privileges on, then the object is not listed.

Select a table to display the clustered columns.



HASH_AJ

HASH_AJ refers to an anti-join, where the join is a hash merge join. There are a number of criteria for the HASH_AJ hint to be applicable.

- Anti-joins are only possible on SQL statements containing a subquery interfaced to the surrounding query via a NOT IN operator. HASH_AJ is applicable to all columns interfaced between the surrounding subquery and the nested subquery that have simple references to the column names, that is, without expressions. The selected items in the

nested subquery can have columns embedded in the aggregate functions MIN, MAX, AVG, COUNT, SUM, STDDEV, or VARIANCE. The following examples illustrate the point:

Good:

```
SELECT * FROM employees
WHERE emp_seq NOT IN
(SELECT emp_seq FROM dependents)
```

Bad:

```
SELECT* FROM sal_history
WHERE (emp_seq, sal+100) NOT IN
(SELECT emp_seq, max(sal) FROM sal_history
GROUP BY emp_seq)
```

The second example is bad because “sal+100” is not a simple reference: It is part of an expression.

- A GROUP BY clause must be specified if an aggregate function is used in the subquery. The following examples illustrate:

Good:

```
SELECT * FROM sal_history
WHERE sal not in
(SELECT sum(distinct sal) FROM sal_history
GROUP BY emp_seq)
```

Bad:

```
SELECT * FROM sal_history
WHERE sal not in
(SELECT max(sal) FROM sal_history)
```


The last example cannot except the HASH_AJ hint, as it lacks the GROUP BY clause.

- All column references must return non-null values. The following example illustrates:

Good:

```
SELECT * FROM employees
WHERE eeoc is not null
and eeoc not in
(SELECT /*+HASH_AJ*/ code FROM eeoc_codes)
```

Bad:

```
SELECT * FROM employees
WHERE eeoc not in
(SELECT /*+HASH_AJ*/ code FROM eeoc_codes)
```

By adding the “eeoc is not null” ORACLE knows that NULLs will not occur, regardless of the column definition, and therefore performs the anti-join. If the EEOC_CODES.CODE column was not defined as NOT NULL, then the appropriate criteria needs to be entered there as well.

- No correlation is allowed in the subquery. The following example has a correlated subquery and cannot use the HASH_AJ hint:

```
SELECT *
FROM employees e , sal_history s
WHERE e.emp_seq = s.emp_seq
AND effective_date not in
      (SELECT max (effective_date )
       FROM sal_history s1
       WHERE s1.emp_seq = s.emp_seq )
```

- The WHERE clause of the surrounding query must not have criteria joined by an OR operator.
- Anti-joins can only be performed with the cost-based approach. This means the top level query can not have the RULE hint.

INDEX

Use an INDEX hint to instruct ORACLE to use one of the indexes specified as a parameter. The INDEX hint is helpful in a number of situations. If the FROM clause contains a single table, then only those indexes whose leading columns are referenced in criteria can be used. In the following query, it doesn't make sense to use the INDEX hint with the I_EMP_HIREDATE index, since the index consists of the HIREDATE column, and there is no criterion referencing the HIREDATE column.

```
SELECT * FROM employees
WHERE lname like :Last
AND birthdate BETWEEN :Start_Date AND :End_Date
```

The following query consists of a join with non-join criteria on both tables.

```
SELECT * FROM employees e, sal_history s
WHERE e.emp_seq = s.emp_seq
AND birthdate BETWEEN :Start_Date AND :End_Date
AND sal > :Salary
```

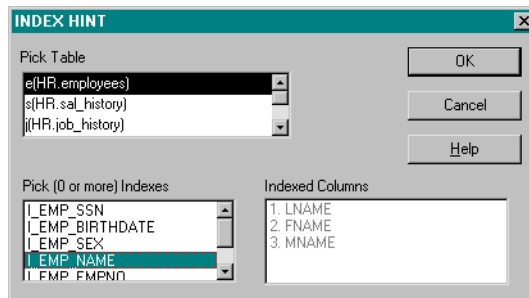
In this case, the INDEX hint makes sense only on the I_EMP_BIRTHDATE index on the BIRTHDATE column of the EMPLOYEES table, or on the I_SAL_SAL index on the SAL column of the SAL_HISTORY table when a nested loop join is performed. On nested loop joins, the table referenced by the first step of the plan is considered the driving table. The INDEX hint can be used for non-equality criteria only on the driving table. Additional indexes can be used only on

equality criteria for the non-driving table. For instance, if EMPLOYEES is the driving table, the INDEX hint could still be used on the SAL column index in the following query:

```
SELECT * FROM employees e, sal_history s
WHERE e.emp_seq = s.emp_seq
AND birthdate BETWEEN :Start_Date AND :End_Date
AND sal = :Salary
```

If a merge join is performed, the INDEX hint could be used for all non-equality criteria. The following figure illustrates the INDEX Hint dialog.

Select a table to list the indexes created on the table. Click an index to display the columns in the index.



INDEX_ASC

The INDEX_ASC hint is intended to signify that the index must be used in ascending order. The INDEX_ASC hint is ideal if the rows are loaded into the database in order by the index columns, or if the index columns are referenced in an ORDER BY. The INDEX_ASC HINT dialog is the same as the INDEX HINT dialog.

INDEX_COMBINE

The INDEX_COMBINE hint works only with bitmap indexes that are available in V7.3.3. If multiple bitmap indexes are specified, ORACLE attempts a boolean combination of the indexes.

INDEX_DESC

The INDEX_DESC hint follows the same rules as the INDEX hint, but the purpose is different. The INDEX_DESC hint requests ORACLE to read the index in descending order, which retrieves the corresponding rows in descending order of the index columns. This hint is useful when the ORDER BY is in descending order. For example, to retrieve the youngest employees from a Human Resources database, the following query could be used:

```
SELECT * FROM employees
WHERE birthdate =
(SELECT max(birthdate) FROM employees)
```

Using the INDEX_DESC hint in the following query achieves the same goal more efficiently:

```
SELECT /*+ INDEX_DESC
      (EMPLOYEES,I_EMP_BIRTHDATE) */ *
FROM employees
WHERE birthdate <= sysdate
AND rownum = 1
```

The INDEX_DESC Hint dialog is the same as the INDEX Hint dialog.

INDEX_FFS

The INDEX_FFS hint refers to fast full index scans, where the entire index is read using the multiblock I/O option available in ORACLE. Normal index searches read only one index page at a time. The fast full index scan is only possible when the index contains all the columns needed for the query. The following examples illustrate when the index contains all the columns:

```
SELECT col1, col2
FROM tab1
```

Assuming TAB1 has a concatenated index on either the combination COL1|COL2 or COL2|COL1, rather than perform a full table scan to extract the columns, the INDEX_FFS hint with TAB1 and the name of the concatenated index will use only the index. This is faster than specifying the INDEX hint, which reads only one index page at a time. INDEX_FFS performs multi-block reads of the index.

The following SQL statement cannot use the INDEX_FFS hint, as INDEX_FFS should only be used with hash and merge joins:

```
SELECT count(*)
FROM tab1
WHERE col1 > 1
AND col2 BETWEEN 1 and 100
```

The INDEX_FFS dialog is the same as the INDEX Hint dialog.

MERGE_AJ

MERGE_AJ refers to an anti-join, where the join is a sort merge join. There are a number of criteria for the MERGE_AJ hint to be applicable.

- Anti-joins are only possible on SQL statements containing a subquery interfaced to the surrounding query via a NOT IN operator. MERGE_AJ is applicable to all columns interfaced between the surrounding subquery and the nested subquery that have simple references to the column names, that is, without expressions. The selected items in the nested subquery can have columns embedded in the aggregate functions MIN, MAX, AVG, COUNT, SUM, STDDEV, or VARIANCE. The following examples illustrate the point:

Good:

```
SELECT * FROM employees
WHERE emp_seq not in
      (SELECT emp_seq
       FROM dependents )
```

Bad:

```
SELECT * FROM sal_history
WHERE (emp_seq , sal+100 ) not in
      (SELECT emp_seq , max (sal )
       FROM sal_history
       GROUP by emp_seq )
```

The second example is bad because “sal+100” is not a simple reference: It is part of an expression.

- A GROUP BY clause must be specified if an aggregate function is used in the subquery. The following examples illustrate:

Good:

```
SELECT * FROM sal_history
WHERE sal not in
      (SELECT sum (distinct sal )
       FROM sal_history
       GROUP by emp_seq )
```

Bad:

```
SELECT * FROM sal_history
WHERE sal not in
      (SELECT max (sal )
       FROM sal_history )
```

The last example cannot accept the MERGE_AJ hint, as it lacks the GROUP BY clause.

- All column references must return non-null values. The following example illustrates:

Good:

```
SELECT * FROM employees
WHERE eeoc is not null
AND eeoc not in
      (SELECT /*+MERGE_AJ*/ code
       FROM eeoc_codes )
```

Bad:

```
SELECT * FROM employees
WHERE eeoc not in
      (SELECT /*+MERGE_AJ*/ code
       FROM eeoc_codes )
```

By adding the “eeoc is not null” ORACLE knows that NULLs will not occur, regardless of the column definition, and therefore performs the anti-join. If the EEOC_CODES.CODE column was not defined as NOT NULL, then the appropriate criteria needs to be entered there as well.

- No correlation is allowed in the subquery. The following example has a correlated subquery and cannot use the MERGE_AJ hint:

```
SELECT * FROM employees e , sal_history s
WHERE e.emp_seq = s.emp_seq
AND effective_date not in
      (SELECT max (effective_date )
       FROM sal_history s1
       WHERE s1.emp_seq = s.emp_seq )
```

- The WHERE clause of the surrounding query must not have criteria joined by an OR operator.

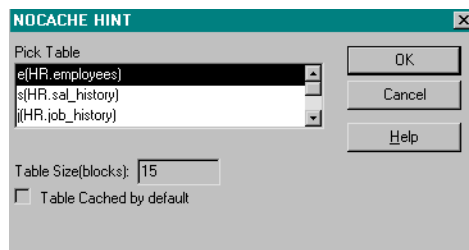
- Anti-joins can only be performed with the cost-based approach. This means the top level query can not have the RULE hint.

NOCACHE

ORACLE normally stores every data block read at the end of the LRU (Least Recently Used) chain in the SGA. This action becomes an issue during a full table scan. It is not desirable to store a large table at the end of the LRU chain. It is preferable to store those blocks at the beginning of the LRU chain so that they are immediately flushed from the system by the next user. If the blocks are stored at the end of the chain, blocks that are needed by other sessions are likely to be flushed, requiring a physical read to return the block to the SGA. The NOCACHE hint ensures this, regardless of the default setting specified when the table was created.

The NOCACHE hint takes a table name as its only parameter. Plan Analyzer then lists all objects in the FROM clause. If the object does not translate to a single table, the object is colored yellow. When a table is highlighted, Plan Analyzer populates the **Table Size (blocks)** field with the number of blocks occupied by the table rows if the table is analyzed, or with the total number of blocks allocated to the table if the table is not analyzed. If the table is cached by default, the **Table Cached by default** check box is set.

The NOCACHE HINT dialog is illustrated in the figure below.



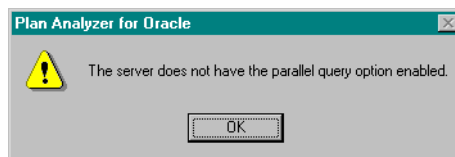
NOCOST

Use the NOCOST hint to force ORACLE to not use the cost-based optimizer. Other hints can be used with this hint. NOCOST is functionally the same as the RULE hint.

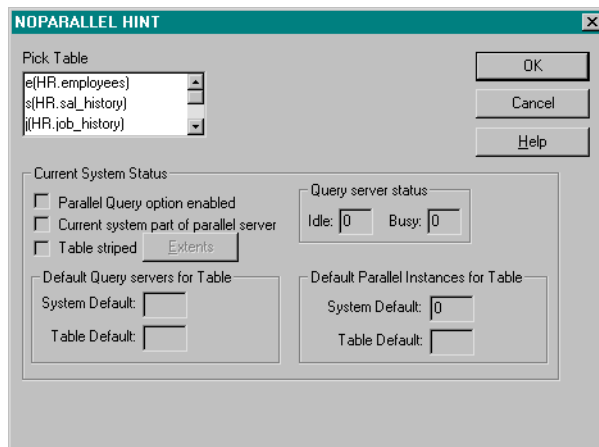
NOPARALLEL

Use the NOPARALLEL hint to instruct ORACLE not to use multiple server processes to service the operations on the specified table. This overrides database settings in the INIT.ORA file and table attributes that specify a parallel operation.

If the parallel query option is not enabled on the system, the hint is colored red, and choosing the hint displays the following dialog:



When you select the NOPARALLEL hint, Plan Analyzer displays a dialog that lists all objects on the FROM clause.



If the object is a view or translates to a view of multiple tables, or the user does not have privileges on the underlying objects, then the hint is colored yellow and the following components are disabled in the NONPARALLEL HINT dialog.

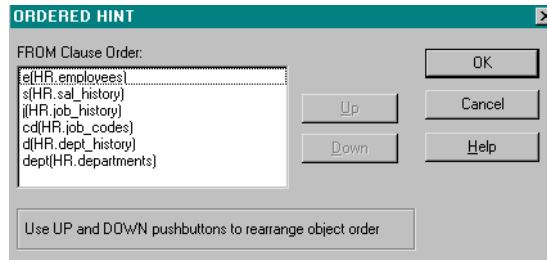
- **Table striped** checkbox
- **Extents** button
- **Default Query Servers for Table** group box
- **Default Parallel Instances for Table** group box

The **Default Query Servers for Table** and **Default Parallel Instances for Table** options are specified in table definitions and also at the system level in the INIT.ORA file. These values provide important information about the normal parallelism that will occur unless overridden by the NOPARALLEL hint.

ORDERED

Use the ORDERED hint to access the tables in the order they appear in the FROM clause. You must make sure that the join is possible. The ORDERED dialog permits the re-order of the tables. It is important to understand that the requested re-order is used only in the Hints mode. Plan Analyzer records the specified order as a parameter to the hint and physically changes the FROM clause order when Hints mode is requested. ORDERED is a powerful hint to consider when you are directing ORACLE to construct a custom plan.

The ORDERED Hint dialog is illustrated in the figure below. All objects in the FROM clause are listed in the order they currently appear. To re-order the objects, click the object to move, and then click either the **up** or **down** button.



PARALLEL

Use the PARALLEL hint to request multiple server processes to simultaneously service operations on the specified table. The PARALLEL hint can be used on each table in the FROM clause. Since the PARALLEL hint is useless without a full table scan, Plan Analyzer automatically includes a FULL hint on each table that has a PARALLEL hint. Besides parallelizing full table scans, the parallel servers can be used to parallelize the following operations in the plans:

- AGGREGATE (GROUP BY)
- MERGE JOIN
- NESTED LOOPS
- SORT (GROUP BY)
- SORT (JOIN)
- SORT (ORDER BY)
- SORT (UNIQUE)

Selecting the PARALLEL hint lists all objects on the FROM clause (see following figure). If the object is a view or translates to a view of multiple tables, or if the user does not have privileges on the view's underlying objects, then the hint is colored yellow and the following items are disabled:

- Is the table striped?

Hints

- EXTENTS pb
- Default query servers
- Default parallel instances

The PARALLEL HINT dialog displays a variety of information on the current status of the query servers and on instances that can assist in parallelizing the operations for a specific table. To use the PARALLEL hint, **Parallel Query option enabled** must be set. If the current server is part of a parallel server, **Current system part of parallel server** is set. If the current server is not part of a parallel server, then it is meaningless to enter a value for **No. Instances**.

Parallelizing a full table scan can achieve a significant performance gain if the table is striped and if there are CPUs that are currently not busy. Neither ORACLE nor Plan Analyzer can advise on the number of idle CPUs, but Plan Analyzer does contain the current status of the query servers. Each time the PARALLEL HINT dialog is invoked, the number of idle and busy query servers is listed in the **Idle** and **Busy** fields. The **Table striped** check box is checked if the extents in the table are placed on more than one database file. Click the **Extents** button to see a list of the extents and the unique database files they are on. Ideally, the database files

should also be on distinct disk drives. It is also possible that a single database file is a logical filename striped over multiple disk drives at the operating system level.

Selecting a table sets the **Table striped** check box if the table is striped, and also displays the number of default query servers and instances. **Default Query Servers for Table** and **Default Parallel Instances for Table** are defined at both the table level (table definition) and at the system level in the INIT.ORA file. These values provide important information about the normal parallelism that will occur unless overridden by the PARALLEL hint.

Select a table to parallelize, and then enter the number of query servers (**No. Query Servers**). If the current instance is part of a parallel server, enter the number of instances (**No. Instances**). The maximum values for each parameter are listed at the right.

PUSH_SUBQ

The goal of this hint is to force subqueries to be evaluated earlier rather than as a filter operation. This becomes important when multiple tables are being joined together. If the subquery was performed as a filter operation in a join, the filter takes place after the join. If the subquery is executed earlier, the row may never have to be joined to the other tables. Normally subqueries are performed after the join.

With the PUSH_SUBQ hint the subqueries are performed as early as possible, avoiding needless joins, as in the following example:

```
SELECT /*+push_subq*/ * FROM employees e
WHERE not exists
      (SELECT null
       FROM dependents d
       WHERE e.emp_seq = d.emp_seq
       AND relation = 'SPOUSE' )
AND exists
      (SELECT null
```

```
FROM sal_history s
WHERE e.emp_seq = s.emp_seq
AND effective_date <= sysdate
HAVING max (sal ) > 1000
```

Here the default cost plan would have two filters. With the addition of the PUSH_SUBQ hint there is only one filter.

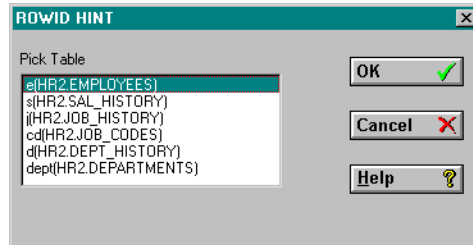
The PUSH_SUBQ hint is only applicable where the statement has subqueries, none of which access remote tables. The following example illustrates:

```
SELECT * FROM employees e
WHERE not exists
      (SELECT null
       FROM employees@remote_db as r_e
       WHERE r_e.emp_seq = e.emp_seq )
AND emp_seq in
      (SELECT max (emp_seq )
       FROM sal_history s
       WHERE sal > 1000 )
```

In this case the PUSH_SUBQ hint can be used on the third module, since this module doesn't reference a remote object. If the third module did not exist, (i.e. there was only one subquery), the hint would not be applicable.

ROWID

Use the ROWID hint to access the table by a ROWID value or range. This hint is useful only when the query references the ROWID pseudo-column. The figure below shows the ROWID HINT dialog.



RULE

Use the RULE hint to force ORACLE to use the Rule-based optimizer. The RULE hint can be used with other hints. If the RULE optimization plan has the best performance, then embedding the RULE hint will force the Rule plan regardless of the system default setting in the INIT.ORA file.

RULE does not take any parameters and is applicable to only the first SQL module. If you display the Add Hints dialog for any other SQL module, the RULE hint is colored red.

STAR

The STAR hint is first available on V7.3 to instruct Oracle to perform a star join. The STAR hint is useful when one table in the join is larger than the other tables and the larger table has a concatenated index. The concatenated key includes the foreign keys to the smaller tables. The STAR plan basically performs a Cartesian product join of only the smaller tables. Each combination of the primary key values of the smaller tables in the Cartesian join is then used to sample the concatenated key index on the larger table.

The STAR hint is applicable under the following conditions:

- 1 The FROM clause must have three or more tables.
- 2 The Fact table (the large table) is joined individually to the smaller tables (the Dimension tables). There are no join criteria between the small tables - only from the large table to the individual small tables.

- 3 The `USE_MERGE` or `USE_HASH` hints should not be used. Use only the `USE_NL` hint.

You should use the `ORDERED` hint where the Dimension tables appear first, based on the order of the keys in the concatenated index, followed by the Fact table. Add the `USE_NL` hint to specify the Fact table as the parameter. Add the `INDEX` hint with the Fact table as the first parameter and the name of the concatenated index.

For example, assume the large table (the Fact table) is named `SALES`, and the `SALES` table has a foreign key that relates it to the `ITEMS` table, and a foreign key on `CUSTOMER_ID` relating back to the `CUSTOMERS` table. `SALES` has a concatenated key index (named `SALES_ITEM_CUST`) on `ITEM_ID` followed by `CUSTOMER_ID`. In that case, since the concatenated key starts with `ITEM_ID`, the tables should be reordered as follows:

```
FROM items, customers, sales
```

The hints would be:

```
SELECT /*+STAR ORDERED USE_NL(sales) INDEX(SALES, SALES_ITEM_CUST)
FROM items, customers, sales
```

USE_CONCAT

This hint is used when the SQL statement module has multiple criteria connected by an `OR` operator. `USE_CONCAT` is applicable if the SQL module contains multiple `OR`'d criteria where the columns in the criteria are indexed.

For instance, the following query,

```
SELECT * FROM employees
WHERE birthdate BETWEEN '01-APR-52' AND '01-JUN-52'
OR hiredate > sysdate - 90
```

may only use the index on `BIRTHDATE`, but `ORACLE` can rewrite the query as follows and still obtain the same result set:

```
SELECT * FROM employees
```



```
WHERE birthdate BETWEEN '01-APR-52' AND '01-JUN-52'  
AND (hiredate <= sysdate - 90 OR hiredate IS NULL)  
UNION ALL  
SELECT * FROM employees  
WHERE (birthdate NOT BETWEEN '01-APR-52' AND '01-  
-JUN-52' OR birthdate IS NULL)  
AND hiredate > sysdate -90
```

Now the query is rewritten as a UNION ALL which is basically the CONCATENATION operation. The difference is that now each piece of the SQL statement can be optimized differently.

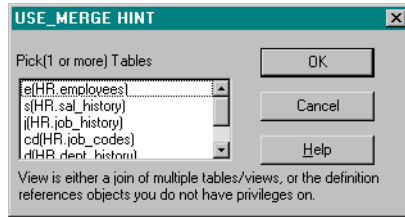
USE_HASH

The USE_HASH hint is applicable when the multiple tables are joined. This hint is only available when two or more tables are joined. The USE_HASH dialog prompts you for one or more tables.

USE_MERGE

The USE_MERGE hint instructs ORACLE to perform a sort merge join rather than a nested loop join or a hash join. The merge join works by sorting the two result sets over the join columns and then merging the results via the join columns. You can avoid a sort operation if the table has an index on the join columns, because the index sorts the rows by definition. Since USE_MERGE may initiate sorts, a lack of real memory at the server can result in performance degradation. However, the normal nested loop join requires constantly revisiting the B-tree index, and reusing the same data block. Merge joins are typical for an All Rows plan. The USE_NL hint is the opposite of the USE_MERGE hint.

The figure below illustrates the USE_MERGE HINT dialog. It lists all tables in the FROM clause.



USE_NL

The USE_NL hint instructs ORACLE to perform a nested loop join rather than a merge join or a hash join. The nested loop join retrieves one row from one result set and then joins the row to rows in the other result set. A nested loop is preferable if the goal is a First Rows plan, or if only a few rows are extracted. The USE_NL HINT dialog is the same as the USE_MERGE HINT dialog.

Step Details


This menu option is available in both Expert and Standard modes.

What does each line of the optimization plan mean? The optimization plan is quite complex in some cases and benefits from explanation. This section explains the database objects that are referenced in the plans.

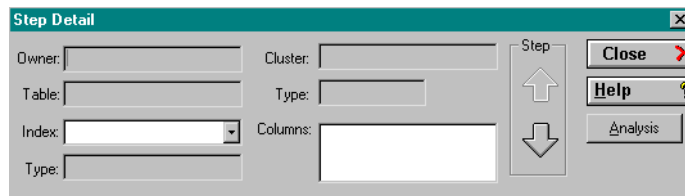
What are the Database Objects in Each Step?

It is not always easy to know what database objects are being referenced by a line of the optimization plan. Common questions are:

- What is the table for the index?
- What columns comprise the index?
- What table is being accessed by the cluster index?
- What type of cluster is it?
- What other indexes exist on the table?

If you click an optimization step that contains a database object, Plan Analyzer enables the **Step Details** menu option under the **Plan** menu item and also enables the Step Details toolbar icon .

Clicking the **Step Details** menu option or icon displays the Step Detail dialog box (see following figure). The Step Detail dialog lists information on database objects that are directly or indirectly referenced by the optimization step. At this point in evaluating an optimization plan, you need to decide whether or not to use a different index to drive the query.



Indirect object references are an important feature of Plan Analyzer. For example, what if the database object on the optimization step is a synonym?

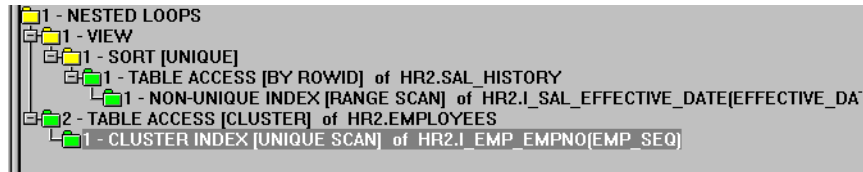
ORACLE lists the synonym as the object, but you need to know the table name because the indexes belong to the table, not to the synonym. The available indexes are vital in determining an alternative index path. Plan Analyzer translates synonyms to the underlying table name. It also goes one step further: It performs multiple layers of translation, for example, a synonym which points to another synonym which points to a table.

Another indirect object reference deals with clusters. Clusters can contain rows from multiple tables, and a step in an optimization plan will sometimes show only the cluster index as the database object. The following example SQL illustrates the point:

```
SELECT * FROM employees
WHERE emp_seq in
      (SELECT emp_seq
       FROM sal_history
```

WHERE effective_date >= sysdate - 30)

The Rule-based optimization plan follows for the SQL:

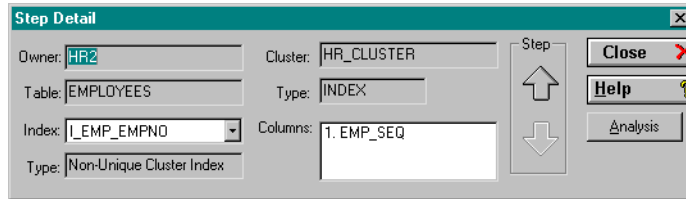


Step 7 of the optimization plan accesses the database object I_EMP_EMPNO, which is a cluster index. But the cluster contains multiple tables. Which table is being indirectly referenced by the index? First, note that the operation is a CLUSTER access. The cluster key was referenced by the WHERE EMP_SEQ IN criterion for the EMPLOYEES table. The Index dialog box will display the following objects:

- I_EMP_EMPNO index, indicating that it is a non-unique cluster index
- EMPLOYEES table owned by HR
- HR_CLUSTER, which is an indexed cluster containing the table

Additionally, since Plan Analyzer knows that the cluster index is being used to access the EMPLOYEES table, the cluster key is translated to the underlying columns in the EMPLOYEES table.

The following figure illustrates the Step Details dialog for step 7 of the above plan. The **Index** combo box lists all other indexes on the EMPLOYEES table. Whenever a table is a member of a cluster, the cluster index is displayed in the index combo box along with the regular indexes on the table. If the cluster is a hash cluster, the hash index is listed as **** HASH INDEX ****, and the hash columns are translated into the table column names, just as they are for the indexed cluster.



To determine which index will provide the fastest access path to a table, you must know the other non-join criteria on the table and then look through the indexes listed in the index combo box. Clicking an index displays the columns in the index, the column name with the position of the column in the index, and the type of the index.

Plan Analyzer overcomes certain bugs in ORACLE's listing of the optimization plan. ORACLE shows the creator of a public synonym as the owner of the synonym, rather than PUBLIC. For example, if HR creates a public synonym EMPS for HR.EMPLOYEES, ORACLE would show HR as the owner of EMPS in the optimization plan instead of PUBLIC. The same holds for local synonyms (synonyms owned by the user logged into Plan Analyzer). The owner is the owner of the underlying object, not the owner of the synonym. For example, if SQL_TOOL creates the synonym PEOPLE for HR.EMPLOYEES, ORACLE lists HR as the owner of the object PEOPLE. Plan Analyzer corrects both of these errors.

The **Analysis** button invokes the DB Objects Analysis dialog (see *DB Objects Analysis* in the *SQL Menu* chapter). Initially, the **Table** name field shows the table listed in Step Details and the **Index** field shows the index if one was present in the step. In addition, Plan Analyzer indicates whether statistics are used in the evaluation of a step by coloring the folder icon green if statistics are used, or red if none are used.

Click the up and down arrow buttons to move through the optimization plan; you don't need to return to the plan to highlight a different step.

Distributed Queries

Distributed queries may reference only remote database objects, or a combination of remote and local database objects. (A local database is one that you are currently logged onto, and a remote database is one that you are not currently logged onto.) If a step of the plan involves a distributed query, the operation on the plan is remote. When you invoke **Step Details** for a step involving a remote operation, the **Distributed** button appears in the lower right corner of the Step Details dialog.

Click **Distributed** to display the distributed query. The following figure shows a query sent to a remote database using Rule-based optimization.



Notice that the query does not have a WHERE clause. This means that the remote query performs a full table scan. The query being evaluated is a join between a local and a remote table, and the distributed query is the first child operation for a nested loop operation. The problem is that there is a non-join criterion on the local table and it references an indexed column. Optimally, the local table would be processed first to minimize the number of rows qualified, and it would then be joined to a remote table. That would require that the remote query have a WHERE clause that references the join criterion.

The following figure shows the distributed query for the First Rows plan and contains a WHERE clause on the join criteria.



The Rule plan is not optimal because of the distributed query. Also, a distributed query can be quite different for different optimization modes.

Parallel Queries

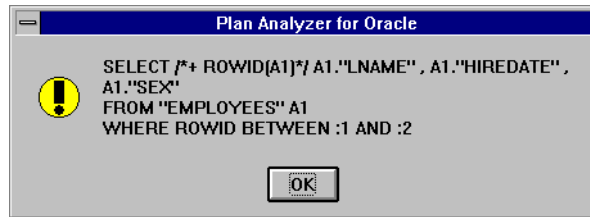
When a SQL statement is parallelized by distributing the load to multiple processors, the optimization plan contains queries to represent the division of labor between the processors. The queries are a clear indication that the plan has indeed been parallelized. The following query contains the **PARALLEL** hint:

```
SELECT /*+ PARALLEL (employees,2) */ lname ,
      hiredate , sex
FROM employees
```

The Rule-based plan follows

1 - TABLE ACCESS [FULL] of HR.EMPLOYEES

The plan contains only a single step. Invoking **Step Details** for the step displays a **Parallel** button in place of the **Distributed** button. Clicking the **Parallel** button displays the query (see following figure) that illustrates how the full table scan is distributed among the processors.



Note that the WHERE clause criterion uses the BETWEEN operator for the ROWID column, so each processor is given a different starting and ending ROWID value. Each processor is thus responsible for retrieving rows from a different set of data blocks.

Cost, Rows, and Bytes

ORACLE V7.3 extends the plan table with additional information on each step of a plan. It estimates the number of rows that can be extracted per step, the cost of each step used in computing the total cost, and the number of bytes anticipated per step. Each of these values is cumulative: The cost of a parent operation incorporates the cost of the child operations. ORACLE's TKPROF utility also provides this information, but TKPROF produces the exact values for the rows and bytes because the values are computed while the SQL statement is being processed. Without actually executing the SQL statement, it is impossible to determine actual values. The plan table columns therefore can contain only estimates for rows and bytes.

When you are connected to ORACLE V7.3, the cost, rows and bytes per step are displayed at the bottom of the window in the status bar. In the following example, the cost of the highlighted step is N/A (Not Available), the anticipated number of rows is 1, and the number of bytes that are anticipated to be returned from the step is 11.

Cost = N/A; Rows = 1; Bytes = 11

Visualize

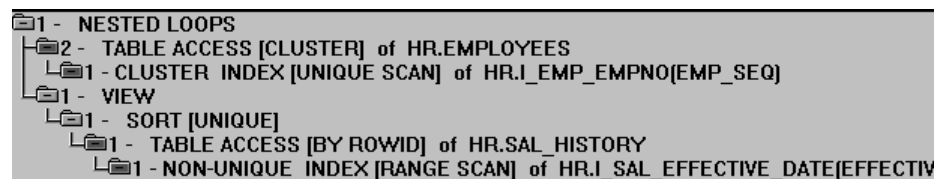
This menu option is available in both Expert and Standard modes.

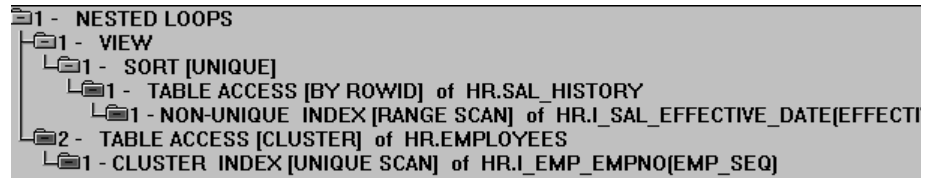
SQL is a *non-procedural* database language: ORACLE is not instructed step by step how to access and filter the tables, but is given only high-level specification of what is requested. Since all computers process in a procedural fashion, the non-procedural language must be translated into a set of procedural steps: optimization plans. Traversing the steps of the plan, from the first to the last, illustrates how ORACLE processes a specific SQL statement.

ORACLE supplies three important columns in the underlying plan table:

Column	Description
ID	the unique identifier of a step
PARENT_ID	the ID of the parent operation
POSITION	the order of the operation among all sibling operations for a specific parent operation

Since a plan is a hierarchical set of procedural operations, there is no SQL query that can return the plan steps in the exact order they will be executed. In fact, re-explaining the same SQL statement can produce what appears to be a different optimization plan, though in fact it is the same plan in a different order. For instance, the plans in the following figures are identical, though they look different.





The **Visualize** menu option, located in the **Plan** menu, visually illustrates the order of step execution and optionally gives an explanation of the operation in the step.

Explaining Each Step

Each step of the explain plan is one of the following:

- The combination of an object and the operation performed on the object, where an option determines the “flavor” of the operation.
- An operation, and optionally the flavor of the operation.

The first step to be executed must include an object. The only steps that don't include an object are ones that are processing the results of previous steps. For example, the following SQL statement:

```

SELECT eeoc.description, e.sex, count(*)
FROM hr.employees e, hr.eeoc_codes eeoc
WHERE e.eeoc = eeoc.code
GROUP BY eeoc.description, e.sex
  
```

has the following Rule-based optimization plan (the lines are numbered for discussion purposes only):

- 1 0 - SORT [GROUP BY]
- 2 1 - NESTED LOOPS
- 3 1 - TABLE ACCESS [FULL] of HR.EEOC_CODES
- 4 2 - TABLE ACCESS [BY ROWID] of HR.EMPLOYEES
- 5 1 - NON-UNIQUE INDEX [RANGE SCAN] of HR.I_EMP_EEOC

In the first step, line 3, the object being operated on is the HR.EEOC_CODES table, the operation is a TABLE ACCESS, and the type of that access is FULL: a full table scan is being performed on the HR.EEOC_CODES table. The reason for the full table scan is that there is no criterion on the EEOC_CODES table other than the join criterion, and rule-based optimization will pick the last table in the FROM clause as the driving table. The parent of line 3 is a nested loops operation. Nested loops indicate that one source of rows will be joined to another source of rows.

The second source is identified by line 4, which is the second child of line 2. Line 4 accesses the HR.EMPLOYEES table. But line 5, the scan of the I_EMP_EEOC index, is the child of line 4, and that child operation must be completed before line 4. Line 5 is the scan of the I_EMP_EEOC index. The I_EMP_EEOC index is searched after retrieving an EEOC_CODES row, taking the CODE column value and searching the I_EMP_EEOC index for a match. For each ROWID found in the I_EMP_EEOC index, the EMPLOYEES with that ROWID is retrieved. That's the nested loop!

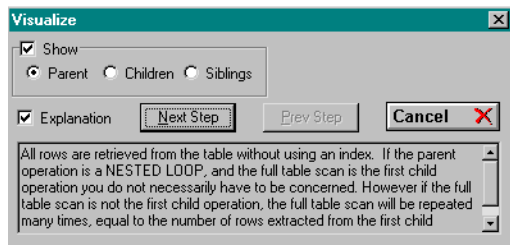
After the join is performed, the nested loop parent operation in line 2 is performed on the results of the join: the result of line 2 is the input source to line 1. Line 1 is the final GROUP BY, which sorts the joined rows by the columns specified in the GROUP BY clause, EEOC.DESCRPTION, and E.SEX.

Traversing the Plan

Select **Plan, Visualize** to display a submenu which lists the different optimization modes. The menu options are enabled only for those plans displayed. Choosing one of the plans brings the plan to the front and displays the Visualize dialog:



Click **Next Step** to highlight the first step of the plan. To see an explanation of the operation being performed, check the **Explanation** check box. This expands the Visualize dialog to include an explanation. The explanation of the operation often references the parent operation, or child and sibling operations. If it's not clear what those operations are, enable **Show** and set the radio button for the steps you want to identify. Those steps then blink. The following figure illustrates the explanation for a plan step involving a range scan of an index.

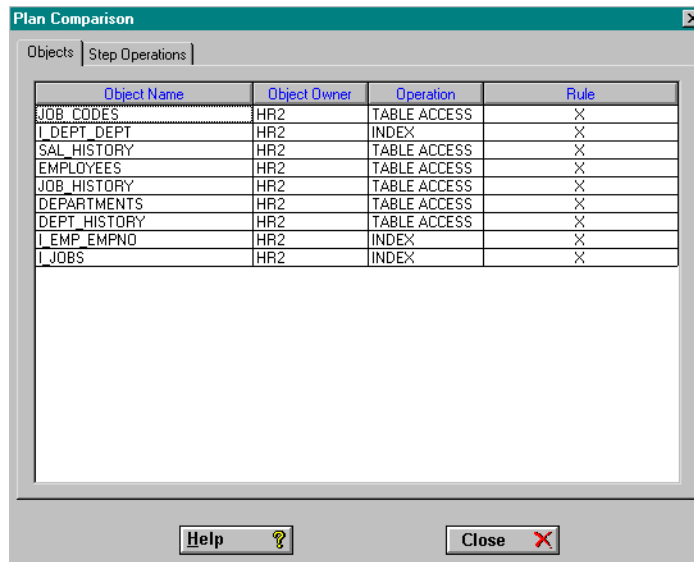


Compare

This menu option is available in both Expert and Standard modes.

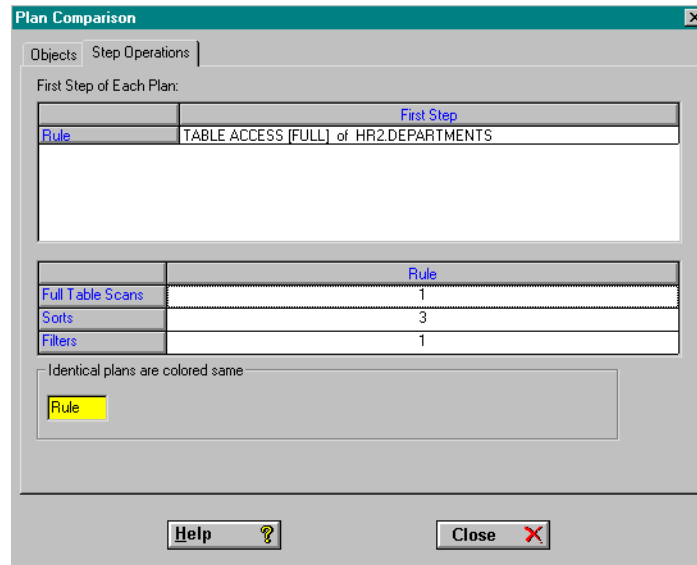
After displaying the different optimization plans for a SQL statement, Plan Analyzer lets you compare the plans in several ways. Choosing **Plan, Compare** displays the Plan Comparison dialog with two tabs: **Objects** and **Step Operations**. The **Objects** tab displays the database objects

accessed by each plan. The form of the display makes it easy to see which indexes are used on which plans. The **Objects** tab displays a spreadsheet of the database objects and the different optimization modes. If the object exists in a specific plan, the row is checked. The following figure illustrates the **Objects** tab for the sample SQL statement in **SAMPLE4.SQL**. The spreadsheet in the figure below makes it clear, for instance, that the **I_DEPT_DEPT** index is used only in the Rule plan.



Object Name	Object Owner	Operation	Rule
JOB_CODES	HR2	TABLE ACCESS	X
I_DEPT_DEPT	HR2	INDEX	X
SAL_HISTORY	HR2	TABLE ACCESS	X
EMPLOYEES	HR2	TABLE ACCESS	X
JOB_HISTORY	HR2	TABLE ACCESS	X
DEPARTMENTS	HR2	TABLE ACCESS	X
DEPT_HISTORY	HR2	TABLE ACCESS	X
I_EMP_EMPNO	HR2	INDEX	X
I_JOBS	HR2	INDEX	X

The **Step Operations** tab of the Plan Comparison dialog displays details on the differences between plans. Most important is the first step executed in each plan (the “driving step”). One of the first goals of tuning a plan is to drive the query in a specific way. The following figure illustrates the **Step Operations** tab for **SAMPLE4.SQL**.



The Step Operations tab indicates that the Rule plan starts with a different step.

The spreadsheet lists three specific operations that typically have an important impact on performance: Full Table Scans, Sorts, and Filters. A **Full Table Scan** requires a complete scan of each data block in a table. If the table is large and only a subset of the rows is required, it would be preferable to eliminate the full table scan. A **Sort** indicates the rows returned from child operations must be sorted. It is possible that the sort operation will not actually occur if an index can be used to access the data in sorted order. **First Rows** plans typically avoid sorts since they may result in disk writes if insufficient server memory is available. **Filters** typically represent correlated subqueries being processed within the SQL statement. The number of times each operation appears in a plan is displayed in the appropriate column for each plan.

Since plans can become large and complex, it can be difficult to discern whether one plan is the same as another, especially when the child operations are returned in different orders. The group box at the bottom of the Step Operations tab uses color coding to identify those plans that are exactly the same.

Standard Translation

This menu option is available only in Expert mode.

Plan Analyzer's Expert mode offers greater flexibility and more options for testing, but the resulting plans are more complex. The **Standard Translation** menu option, located in the **Plan** menu, translates the plans into simpler terms. For example, the figure below shows **SAMPLE2.SQL**, a Rule-based plan using the Expert mode terminology. The Standard Translation version of the plan is shown in the second of the figures below.

Choosing **Standard Translation** presents the current plan and all subsequent plans in simple terminology. To return to the default translation, choose **Standard Translation** again. Use **Standard Translation** and **Visualize** together for greatest ease in understanding the plans.

Standard Translation

```

1 - NESTED LOOPS
  1 - VIEW
    1 - SORT [UNIQUE]
      1 - TABLE ACCESS [BY ROWID] of HR.SAL_HISTORY
        1 - NON-UNIQUE INDEX [RANGE SCAN] of HR.I_SAL_EFFECTIVE_DATE[EFFECTIVE_DATE]
    2 - TABLE ACCESS [CLUSTER] of HR.EMPLOYEES
      1 - CLUSTER INDEX [UNIQUE SCAN] of HR.I_EMP_EMPNO[EMP_SEQ]
  
```

SAMPLE2.SQL Plan

```

1 - Nested Loop Join of two row sources are output.
  1 - Results of subquery are returned.
    1 - Sort to output unique rows.
      1 - Row accessed by ROWID from either index search or explicit value in SQL.HR.SAL_HISTORY.
        1 - NON-UNIQUE Index, HR.I_SAL_EFFECTIVE_DATE[EFFECTIVE_DATE], was searched for exact match due to :
    2 - Clustered rows accessed by indexed cluster key value.HR.EMPLOYEES
      1 - Cluster index, HR.I_EMP_EMPNO[EMP_SEQ], was searched for exact match due to :
  
```

SAMPLE2.SQL Plan (Standard Version)

Performance Menu

This menu option appears on the main menu only in Expert mode. The equivalent in Standard mode is the Evaluate menu option.

Choosing a Plan	8-2
Server Statistics	8-3
Options Tab	8-5
Resources	8-12
Testing a Plan	8-15
Additional Information	8-16
Statistics Summary	8-19

Choosing a Plan


Plan Analyzer can produce at least three potentially different optimization plans; rule-based, first row-based, and all rows-based. You can then use hints to customize these plans. The cost value for the cost-based plans (all except rule-based) is one measure of the relative performance differences. It may not always be correct, especially when the analysis is old or estimated. The best way to decide on an optimization plan is to execute a SQL statement and see which plan completes it first and utilizes the fewest resources. CPU time is the most basic statistic, but statistics such as database calls also assist you in reducing the elapsed time.

Statistics returned after executing the SQL not only assist in determining the best optimization plan, they also provide insight into how the SQL is being executed. For example, excessive logical I/O can indicate that the tables in a join are being joined in an inefficient manner. If each child row is retrieved before the parent row, the parent row will be retrieved multiple times, meaning that each data block is read many times. All statistics generated by Plan Analyzer and the execution options used to generate the statistics can be saved in the Plan Analyzer repository.

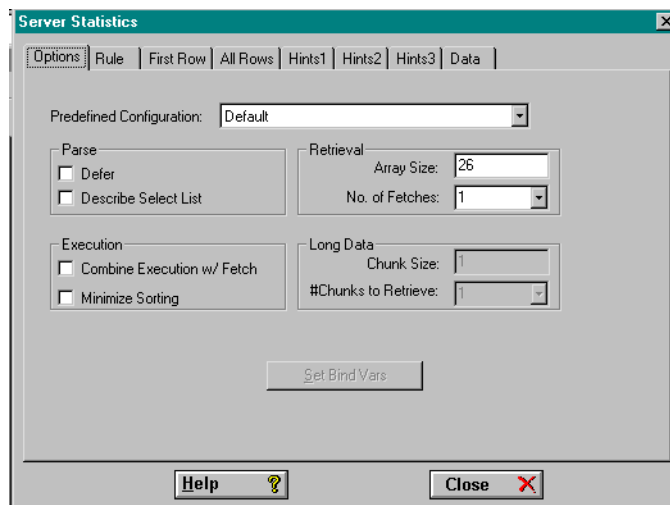
The **Performance** menu contains the **Server Statistics** and **Summary** options.

Server Statistics

The **Server Statistics** option is available only in Expert mode.

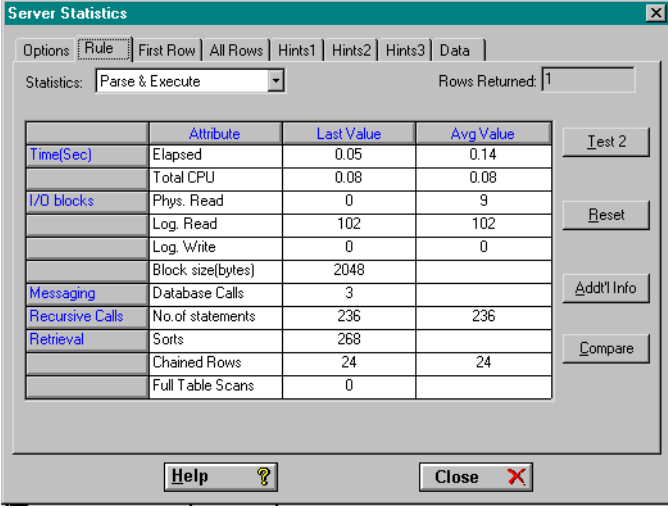
Click the **Server Statistics** menu option or the  toolbar icon to display the Server Statistics dialog. If the SQL statement has bind variables, the Set Bind Variable dialog appears first. In order to access the Server Statistics dialog, you must provide values for all bind variables. Once all bind values are entered, the Server Statistics dialog displays. The first time the Server Statistics dialog is invoked for a SQL statement, Plan Analyzer performs analysis of the SQL statement prior to displaying the dialog.

The Server Statistics dialog varies depending upon whether the SQL statement is a SELECT statement, because different execution options are available. The figures below display the Server Statistics dialog (Options tab and Rule tab) for the execution of one of the sample queries.



Server Statistics Dialog Options Tab (SELECT Statement)

Server Statistics



Options: Rule | First Row | All Rows | Hints1 | Hints2 | Hints3 | Data

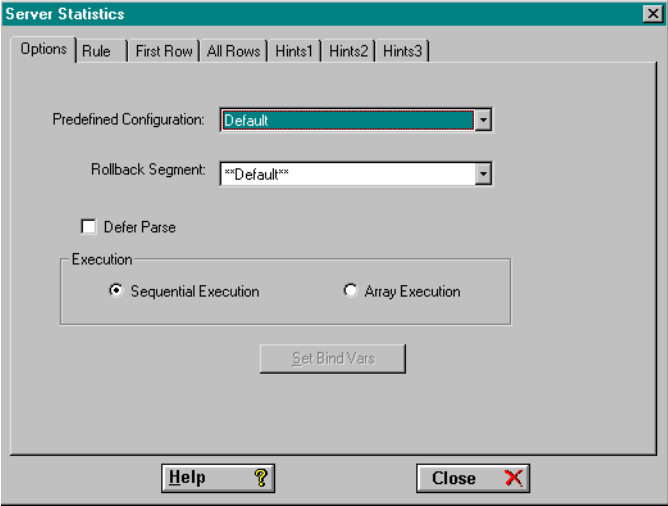
Statistics: Parse & Execute Rows Returned: 1

	Attribute	Last Value	Avg Value
Time(Sec)	Elapsed	0.05	0.14
	Total CPU	0.08	0.08
I/O blocks	Phys. Read	0	9
	Log. Read	102	102
	Log. Write	0	0
	Block size(bytes)	2048	
Messaging	Database Calls	3	
Recursive Calls	No. of statements	236	236
Retrieval	Sorts	268	
	Chained Rows	24	24
	Full Table Scans	0	

Buttons: Test 2, Reset, Add'l Info, Compare, Help, Close

Server Statistics Dialog Rule Tab (SELECT Statement)

Following is the Server Statistics dialog Options Tab for a non-SELECT statement.



Options: Rule | First Row | All Rows | Hints1 | Hints2 | Hints3

Predefined Configuration: Default

Rollback Segment: Default

☐ Defer Parse

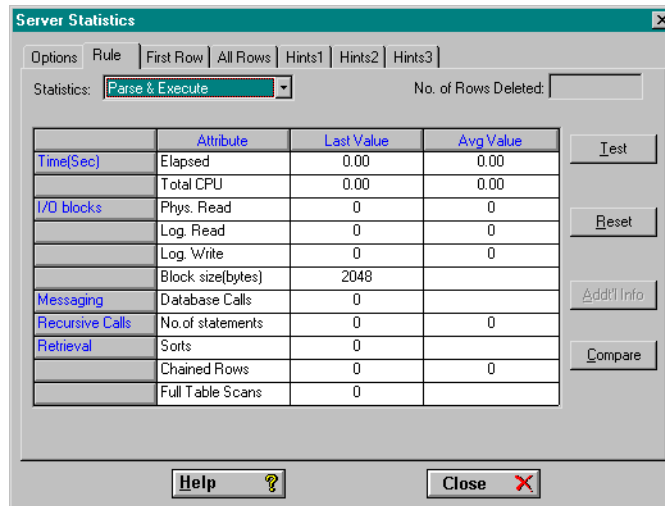
Execution

☒ Sequential Execution ☐ Array Execution

Set Bind Vars

Buttons: Help, Close

Server Statistics Dialog Options Tab (Non-SELECT Statement)



Server Statistics Dialog Rule Tab (Non-SELECT Statement)

The Options tab of the Server Statistics dialog lists the options for emulating the application in which the SQL statement will eventually be embedded.

The other tabs represent the actual resources consumed in each plan test, as well as (in the SELECT statement dialog) the data returned during the execution of a query.

Options Tab

The options under which the SQL statement can be executed can produce a wide range of elapsed time values, and other resources. Plan Analyzer offers every execution option available from ORACLE. Use these options to emulate the environment (4GL or 3GL) in which the SQL will eventually run: that is, execute the SQL using the same ORACLE API calls (OCI). For example, if the SQL will eventually be embedded in a SQL Windows or a Powerbuilder application, the execution options should mirror those used in the application.

Predefined Configuration Field

To simplify the setting of options, several predefined configurations ship with Plan Analyzer. These configurations automatically set the options for the following applications:

- SQL*Plus
- SQL*Windows
- Powerbuilder V3
- Powerbuilder V4

If you do not select a configuration, the **Predefined Configuration** field specifies Default. If a configuration is selected, and you then change one of the options, the configuration becomes Default once the **Test** button is clicked.

New configurations can be defined through the **Predefined Configurations** tab of the Preferences dialog. See the *Preferences* section of the *View Menu* chapter for more information.

Parse: Defer

This option is available for all SQL statements.

Unless a parse is deferred, it is executed immediately, forcing a message to be sent to the server. Since network messaging becomes increasingly more expensive as the network distance increases, deferring messages is a good idea. However if the 4GL product does not defer the parse, this option must not be set.

If the parse is deferred, it executes when the statement executes. The exception is a SELECT statement that is also described, which must be parsed at the time of the describe. (See <Italic>Parse: Describe Select List.</Italic>) Setting the option defers the parse until either the describe or the execution. If the option is not set, the parse occurs immediately.

Parse: Describe Select List

This option is only available for SELECT statements.

When a SELECT statement is executed within an application, the application must use program variables to inform ORACLE where to return the values from each SELECT list item. This is called the *define step*. To determine the items on the SELECT list, ORACLE provides an OCI call to describe the SELECT list. The describe call returns the item name, the datatype in the database, length, etc. ORACLE7 optimizes the describe by returning 16 item descriptions on each call, although the application in fact issues 16 calls. (SQL*Net maintains the buffered item descriptions on the client side.) The larger the SELECT list, the more network calls.

Since network messages are expensive, the key is to not describe the SELECT list. Ad hoc SQL tools such as SQL*Plus from Oracle Corporation must describe each SELECT statement. In that case, there is absolutely no way to avoid the describe, otherwise SQL*Plus would have no way to know how much memory to allocate for each SELECT list item and for the column headings. Custom applications do not perform ad hoc SQL and therefore should not have to resort to describing the SELECT statements.

Most 4GLs, however, do perform describes for various reasons. Some 4GLs, such as SQLWindows, perform numeric manipulation in decimal arithmetic. They prefer to return numeric data in ORACLE's internal numeric format so that SQLWindows can capture the exact precision stored in the database. SQLWindows also returns data columns in ORACLE internal date format. Therefore SQLWindows can present the date in SQLWindows' default format, or any format the application needs, again without losing any portion of the date, such as the time portion. SQLWindows' reasons for describing the SELECT lists are fine, and actually are quite a benefit. A better solution would allow the developer to identify the datatypes as an option. Custom 3GL applications should know what the SELECT list items are, and therefore avoid describing.

If **Describe SELECT List** is set, the SELECT list will be described, and the resources to perform the describes will be included in the parse portion of the statistics. If **Describe SELECT List** is set, and **Defer** is set, the parse occurs at the describe. If **Describe SELECT List** is not set, the SELECT list will not be described.

Execution: Combine Execution w/ Fetch

This option is available for all SQL statements.

Prior to ORACLE7, the SQL was executed using one OCI call, and if the SQL was a SELECT, the rows had to be fetched with a separate call. In fact, there were two different calls for fetching: one for fetching a single row (the older call) and one for performing an array fetch. ORACLE7 introduced a new OCI call that combines the execute with the array fetch call, further reducing the messaging over the network.

Most 4GL products do not utilize this call, though all 3GL applications should. Setting this option combines the execute with the fetch, saving one network message. If the option is not set, Plan Analyzer issues two separate calls to ORACLE.

Execution: Minimize Sorting

This option is available for all SQL statements.

This is the only option that does not pertain to an OCI call. This option eliminates the overhead associated with sorting in SQL statements by actually modifying the SQL prior to execution. The following SQL statement illustrates the problem.

```
SELECT * FROM employees
WHERE hiredate BETWEEN :Start_date and :End_Date
AND birthdate < '01-JAN-52'
ORDER BY lname, fname
```

Assuming that many EMPLOYEES rows would satisfy the criteria, all rows must to be retrieved and sorted before the first row is returned to Plan Analyzer. If you eliminate the ORDER BY, the first row qualified is

returned immediately, and the whole process is significantly speeded up. Most users have experienced a query that never ended. If the test is being performed on an MS Windows client, there is no way to interrupt the execution without rebooting the PC.

If the SQL contains an ORDER BY, GROUP BY, or HAVING statement, the Minimize Sorting option eliminates the clauses and any aggregates associated with the GROUP BY clause. The following SQL statement,

```
SELECT sex, count(*)
FROM employees
WHERE hiredate BETWEEN :Start_date and :End_Date
AND birthdate < '01-JAN-52'
GROUP BY sex
```

is transformed into the following when the **Minimize Sorting** option is selected:

```
SELECT sex
FROM employees
WHERE hiredate BETWEEN :Start_date and :End_Date
AND birthdate < '01-JAN-52'
```

Retrieval: Array Size

This option is available for all SQL statements.

Typically, each row of a query is individually returned from the server, requiring one network message per row. That means the number of network messages to return a result set is $n+1$, where n is the number of rows returned. To minimize the network messages, you need to use an array fetch to return multiple rows per network message. Values for the array size typically range from 10 to 100. If the array size is too large, the network packages will not be able to contain the entire array, resulting in multiple network messages. Different SQL SELECT statements may have different optimal array sizes. Try a large array size first to minimize the database calls, and then decrease the array size until you have minimized the overall elapsed time.

Not all products allow you to specify the array sizes for fetching. ORACLE*Forms and SQL*Windows do permit the array fetches, though only ORACLE*Forms allows the developer to specify different array sizes for most SELECTs. SQL*Windows allows only one array size per application. The goal is to find the best array size for each query.

Retrieval: No. of Fetches

This option is available for all SQL statements.

No. of Fetches tells Plan Analyzer how many fetches to perform. If an array size greater than 1 is specified, then each fetch will be an array fetch. If the array size is 20 and this option is set to 5, then up to 100 rows can be returned. Plan Analyzer will only perform another fetch if the last fetch returns the number of rows specified in the array size. You can return all rows by selecting **ALL** from the combo box.

Long Data: Chunk Size

This option is available for all SQL statements.

The maximum length of LONG and LONG RAW datatypes in ORACLE7 is 2 gigabytes. ORACLE7 provides an option to retrieve the LONG column in pieces, and for specifying the size of each piece. Oracle Corporation's *Programmer's Guide to the ORACLE Call Interface* contains a multimedia application example that retrieves a piece of the data, passes the piece to some device, and then retrieves the next piece.

Some products limit maximum size of the LONG that can be retrieved and stored. This is generally acceptable. Platforms such as MS Windows have limits of 64K bytes. If the SELECT list contains one or more LONG or LONG RAW datatypes, array fetching is disabled, and the **Chunk Size** is used for each LONG column.

Long Data: # Chunks to Retrieve

This option is available for all SQL statements.

This option indicates the number of LONG or LONG RAW pieces to retrieve per row. For example, if the **Chunk Size** is set to 1000 and **# Chunks to Retrieve** equals 20, then 20,000 bytes will be retrieved per LONG column per row. If the query has 2 LONG columns on the SELECT list, Plan Analyzer would perform single row fetches, retrieving 1000 bytes on the fetch, and subsequently performing 19 more calls per LONG to return the additional 19000 bytes per LONG column.

This option can be set to a number greater than 1. If the entire LONG column data is returned prior to completing all the specified calls (**# Chunks to Retrieve**), Plan Analyzer stops performing calls to the server for that column. You can return all rows by selecting **ALL** from the combo box.

Sequential vs. Array Execution

This option is only for non-SELECT statements.

One of the significant options offered by ORACLE is the ability to perform multiple executions of the same INSERT, UPDATE or DELETE statement in a single network message. For example, an invoicing application typically displays a single invoice master row followed by multiple invoice line item rows. When an invoice is entered, multiple line items are entered. Typical 4GLs perform individual inserts for each row inserted (sequential execution), even though ORACLE provides a means through the OCI to insert all details in a single message (array execution).

In order to take advantage of array execution, you must use bind variables in the SQL statement. To insert five rows, enter five values for each bind variable. When **Test** is invoked, one message to the server transmits both the arrays of bind values and the request to perform the execution.

The array option is also available for UPDATES and DELETES. Using the same invoice example, assume an invoice is queried, two line items are deleted, and three are updated. The two deletes and the three updates can be combined into a single execution. In the case of the updates, the SET clause columns must also reference bind variables with an array of three values each.

Array execution is superior to sequential execution for two reasons. First, the server gets one request quickly rather than multiple requests with time delays between them. This makes more efficient use of the server by reducing the chance of losing the CPU while another session gets its time slice. More importantly, the network traffic is the real winner. The greater the span between the client and server and more routers in between, the better the performance of array execution. Expect a 50% improvement in most cases.

Rollback Segment

This option is only for non-SELECT statements.

When non-SELECT statements are executed, ORACLE permits you to identify a specific rollback segment to be used for storing the undo information. Typically, you specify this only on batch applications where many rows are modified. You pick a large rollback segment to avoid extensions to the rollback segment. Extensions occur when a large amount of undo information is generated and other users block the use of the rollback segment extents when they have active transactions.

By default, ORACLE chooses any available rollback segment. To pick a specific rollback segment, click the **Rollback Segment** combo box.

Resources

ORACLE tracks a number of resources per session and displays them at the bottom of the Server Statistics dialog. These resources are captured by Plan Analyzer when a SQL statement is tested. The top-level resource categories are on the left side of the spreadsheet. These categories associate resources into groups for easy evaluation. The actual resources are listed in the **Attribute** column. For example, **Elapsed Time** and **CPU Time** are in the **Time** category. The actual amount of each resource consumed in the last test is listed in the **Last Value** column. The **Avg Value** column keeps a running average over all tests performed. Categories such as the **Block Size** whose value is the same for each test have no **Avg Value** displayed. Each resource is reviewed below.

Statistics

The **Statistics** combo box contains three values:

- Parse Only
- Execute Only
- Parse & Execute

These choices provide a means of assessing the resources consumed by each phase of execution. If the SQL statement is a **SELECT**, the **Parse Only** option consists of the parse, describes, defines, and binds. For non-**SELECT** statements, **Parse Only** consists of the parse and binds. The **Execute Only** option consists of the execute and fetches for **SELECT** statements, and of the execute for non-**SELECT** statements. Different plans consume more resources in different parts of the execution. Cost-based plans typically require more resources during the Parse Only phase.

Elapsed Time

This resource is the number of seconds that have expired at the client side. By tracking the elapsed time at the client side, all network overhead is accounted for. Keep in mind that different locations on the network will produce different elapsed times. The best time resource for determining the optimal plan is the CPU time.

Total CPU

This is the number of seconds of CPU time consumed at the server. This resource is not available on some **ORACLE** platforms, such as Personal Oracle, Netware, and OS/2. Additionally the **INIT.ORA** parameter, **TIMED_STATISTICS**, must be set to **TRUE** for **ORACLE** to capture this resource usage.

Physical Read

This is the number of **ORACLE** blocks that are read from disk. These reads are far more expensive than the logical reads.

Logical Read

Logical reads are requests for ORACLE blocks that are in the SGA, and thus do not require a disk read. Logical reads can generally be reduced by increasing the array size on queries.

Logical Write

Sessions at the server perform only logical writes. Only the database write process performs physical writes. Writes may occur during queries when a read-consistent block is created.

Block size (bytes)

This is the size of the blocks used in the database for storing data.

Database calls

This resource is in the **Messaging** category because it relates to the network traffic caused by the execution of the SQL statement. A database call is a single round-trip message over the network. (A bind or array fetch may require multiple packets transferred, depending on the amount of data being transferred.) The fewer the database calls, the better the network performance.

No. statements

This resource is in the **Recursive Calls** category. Recursive calls generally occur only during the parse phase of a SELECT. Non-SELECT statements may generate recursive calls during the execute phase if they cause requests for new extents.

Sorts

This resource combines all sorts performed during the test of the SQL statement. They generally occur only in the execute phase. Though a plan may show only one step with a sort operation, that sort may in fact be executed many times, or not at all. Sorts can be expensive, especially when the sort is too large to be performed in the server's memory.

Chained rows

Any row that spans multiple data blocks is considered a chained row. One plan may access more chained rows than another due to the access path. Chained rows mean additional I/O. If the chained rows are not due to LONG data, then reorganizing the table should eliminate all chaining.

Full Table scans

Like **Sorts**, **full table scans** may appear only once in a plan but may execute many times. Some full table scans are not bad, depending upon the size of the table.

Testing a Plan

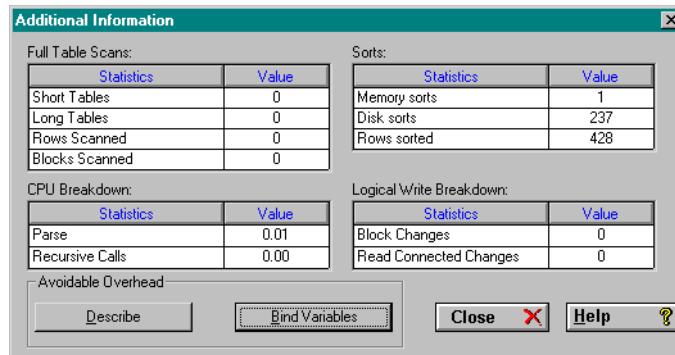
Individual optimization plans can be tested to view the resources consumed. Click the tab for the optimization plan and then click **Test**. After each test, a number is appended to the **Test** button label to indicate the number of tests performed on that plan. In the previous figure, the **Test 2** label indicates that the Rule plan was tested twice. If the **Execution result data** check box in Preferences is set (see *Performance* in the *View Menu* chapter), the Data tab will contain the rows returned from each test of a SELECT statement. After each test of a non-SELECT statement, a ROLLBACK is performed.

The basic objective is to test each plan, excluding the Hint plan if no hints were added. Each plan should generally be tested twice. If the plans are only tested once, then the first plan tested may pay a higher price by performing more physical reads than the other plans. The other plans can read the blocks from the SGA rather than disk.

When reviewing the resources consumed, check the parse separately from the execute. The total (**Parse & Execute**) may be less for one plan, but if the SQL is intended to be parsed once and executed many times, the plan with the fewest **Execute Only** resources will make the best plan.

Additional Information

The **Additional Information** button in the Server Statistics dialog offers more detailed resource consumption information. Click **Additional Information** to display the Additional Information dialog. The figure below illustrates the Additional Information dialog. The four spreadsheets are a breakdown of the four resources listed in the Server Statistics spreadsheet.



Additional Information Dialog

The **Full Table Scans** spreadsheet breaks down the full table scans into the following statistics:

- number of short full table scans
- number of long full table scans
- the number of rows actually scanned
- number of data blocks actually scanned

Short table scans are generally not a reason for concern, though long table scans are. If there are many long table scans, try customizing the optimization plan. If the scans cannot be avoided, use the NOCACHE hint to minimize the effect on other users.

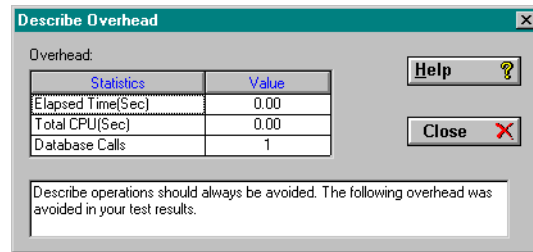
The **CPU Breakdown** spreadsheet breaks the CPU consumption down into parse and recursion. If the recursion is significant, try modifying the buffer cache in the SGA to eliminate or reduce the overhead. (Recall that CPU cannot be tracked on some ORACLE platforms, and that the TIME_STATISTICS parameter in the INIT.ORA file must be set to TRUE.)

The **Sorts** spreadsheet gives the number of memory sorts and of disk sorts, and lists the number of rows that were actually sorted. Disk sorts start out as memory sorts and become disk sorts when insufficient memory is available for the sort, resulting in writes to the temporary tablespace. You should avoid disk sorts whenever possible. If you can't avoid the sort completely, try using SQL criteria to limit the rows sorted, or request that the DBA modify the SORT_AREA_SIZE.

The **Logical Write Breakdown** spreadsheet separates the logical writes into actual block changes and read connected changes. **Read connected changes** refers to blocks that were modified to provide read consistency. **Block changes** are directly related to INSERT, UPDATE and DELETE statements. **Read connected changes** are generally associated with long-running queries of tables that are being modified while the query is still executing.

Describe Overhead

Plan Analyzer captures the overhead associated with performing a describe regardless of whether the **Describe Select List** option is set. The describe overhead should always be avoided, but most 4GLs do perform the describe. Since most products perform an operation that ORACLE allows them to avoid, the performance degradation associated with the describe is available in a single dialog. Clicking **Describe** on the Additional Information dialog displays the Describe Overhead dialog.



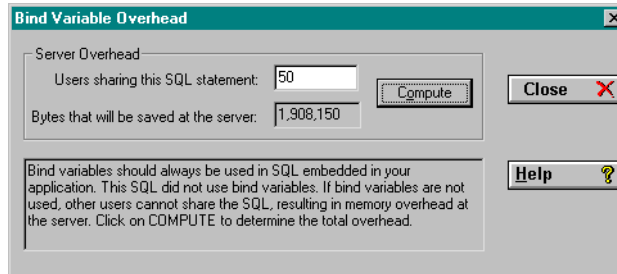
Describe Overhead Dialog

The Describe Overhead dialog contains a spreadsheet that lists the resources associated with the describe operation. The elapsed time increases as the number of database calls increases. (Recall that CPU cannot be tracked on some ORACLE platforms, and that the `TIME_STATISTICS` parameter in the `INIT.ORA` file must be set to `TRUE`.) The text window at the bottom provides some advice and indicates whether the describe overhead was avoided in the results on the Server Statistics dialog.

Bind Variable Overhead

Click **Bind Variables** in the Additional Information dialog to display the Bind Variable Overhead dialog.

If a SQL statement does not have bind variables, it is not likely that the SQL statement is shared with other users in the Shared SQL Cache at the server. Since most applications are designed for multiple users, constructing SQL so that they are shared avoids wasting memory at the server. The Bind Variable Overhead dialog (see following figure) informs developers—and hopefully managers of developers—of the high cost of not using bind variables.

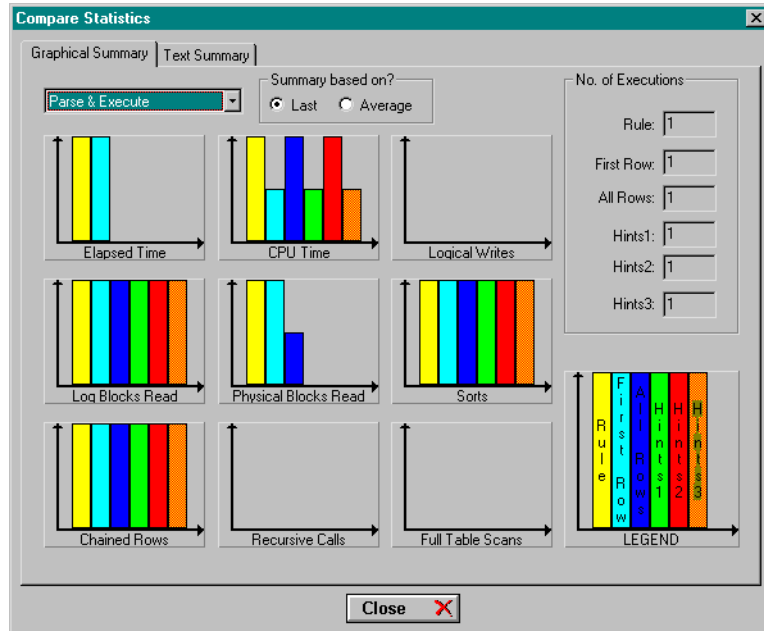


Bind Variable Overhead Dialog

Enter the number of users that are anticipated to use the application simultaneously in the **Users sharing this SQL statement** field. The default value is 50. Click **Compute** to compute the amount of memory at the server that is saved if the SQL is shared properly.

Statistics Summary

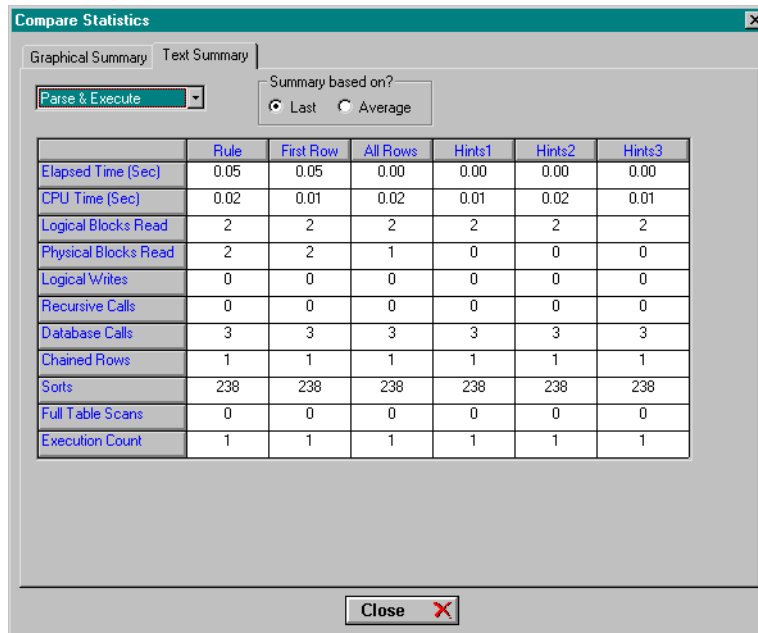
After testing the SQL for the different optimization plans, a comparison of the resources consumed is necessary to determine which is the best plan. Clicking **Summary** on the Server Statistics dialog displays the Compare Statistics dialog (see figure below), which initially displays a graphical comparison of the values.



Server Statistics Dialog (Graphical Summary)

The Graphical Summary tab contains a color coding legend in the lower right corner to identify each plan. The **No. of Executions** group box displays the number of times each plan was tested.

The **Text Summary** tab displays the actual values for precise comparisons:



	Rule	First Row	All Rows	Hints1	Hints2	Hints3
Elapsed Time (Sec)	0.05	0.05	0.00	0.00	0.00	0.00
CPU Time (Sec)	0.02	0.01	0.02	0.01	0.02	0.01
Logical Blocks Read	2	2	2	2	2	2
Physical Blocks Read	2	2	1	0	0	0
Logical Writes	0	0	0	0	0	0
Recursive Calls	0	0	0	0	0	0
Database Calls	3	3	3	3	3	3
Chained Rows	1	1	1	1	1	1
Sorts	238	238	238	238	238	238
Full Table Scans	0	0	0	0	0	0
Execution Count	1	1	1	1	1	1

Server Statistics Dialog (Text Summary Tab)

When the Statistics Summary is initially displayed, the values are based on the last set of values for each test, plus for the combination of parse and execute. The **Summary based on?** group box contains two radio buttons which determine whether the displayed statistics are the most recent set of values, or the average values per test. The combo box at the top left of the dialog provides options for viewing the resource consumption for either the **Parse** phase, **Execute** phase, or both.

It is generally best to view the summary based on the last test of each plan to account for possible physical I/O required on only the first test.

Evaluate Menu

This menu option appears only in Standard mode. It allows you to evaluate a SQL statement without first creating and testing the individual optimization plans.

Evaluating a Statement	9-2
Execution Configuration	9-3
Evaluation Summary	9-4
Performance Statistics Section	9-4
Plan Analysis Section	9-6
Advice	9-7
Copy SQL to Application	9-8

Evaluating a Statement

To simplify testing for non-experts, Standard mode simply requires the user to identify whether the SQL will be embedded in a batch application or an interactive application. Plan Analyzer then evaluates the SQL statement. *Interactive* applications usually display on a terminal where the user browses through small subsets of the data at a time. *Batch* applications are typically submitted through a utility and operate on or retrieve most of the rows in the accessed tables.

The **Evaluate** menu allows you to evaluate a SQL statement without creating and testing each individual optimization plan. The **Evaluate** menu contains two sub-options: **Batch** and **Interactive**. When you select either option, Plan Analyzer does the following:

- 1 Produces the relevant optimization plans.
- 2 Performs execution testing on each plan twice.
- 3 Displays a summary of the test.

You can also use the toolbar icons to invoke the Evaluate options. The icons are displayed below:



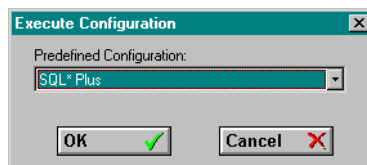
By default, Plan Analyzer tests the Rule and All Rows plans if a batch evaluation is requested. The Rule plan is always tested, since the Rule optimization algorithms are more mature than the cost-based algorithms, and may prove superior. During batch execution tests, Plan Analyzer computes an array fetch size for queries that is based upon the items on the SELECT list and on the network packet size specified in the **Other** tab of the Preferences dialog.

Plan Analyzer tests the Rule and First Rows plans if an interactive evaluation is requested. In both the batch and interactive cases, if the SQL statement contains hints, the Hints plan is also tested. An array fetch size

of 1 is used in the interactive execution tests, since the objective is to return results back to the application as quickly as possible. Using a larger array size could greatly delay the return of rows to the application, since ORACLE is not allowed to return any rows until the entire array is filled.

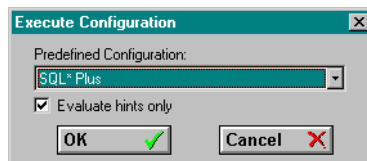
Execution Configuration

When you request one of the evaluations, the Execution Configuration dialog appears. Since the resources consumed can vary widely for different execution options, you should choose a predefined configuration that conforms as much as possible to the application in which the SQL will eventually execute. If the desired configuration does not exist in the list, you can add a new configuration in the **Predefined Configurations** tab of the Preferences dialog (**View, Preferences**).



Execute Configuration Dialog

The final evaluation provides further advice on how to add hints to improve performance. After you have added hints, the Hints plan is the only plan you need to create and test. Since testing can be time-consuming, retesting displays a different Execution Configuration dialog (if hints have been modified). Check the **Evaluate hints only** checkbox to limit retesting to only the Hints plan.



Evaluation Summary

When the evaluation is complete, Plan Analyzer displays the Evaluation Summary dialog.

Default Optimization Mode: **CHOOSE** Suggested Plan for Use: **HINTS1**

Performance Statistics

	Rule	All Rows	Hints1
Elapsed Time (Sec)	0.16	0.08	0.05
CPU Time (Sec)	0.08	0.09	0.09
Logical Blocks Read	102	28	28
Physical Blocks Read	9	0	0
Sorts	268	268	268
Full Table Scans	0	0	0

Plan Analysis:

	Rule	All Rows	Hints1
Full Table Scans	0	0	0
Repeated Table Scans	0	0	0
Sorts	3	9	9

Buttons: Close, Help, Advice, To Clipboard

Evaluation Summary Dialog

The **Default Optimization Mode** field at the bottom shows the default optimization mode that is used on the server. The **Suggested Plan for Use** field displays the plan that Plan Analyzer recommends as optimal. This recommendation is based on the combination of a cost algorithm and the Performance Statistics results in the top spreadsheet. The spreadsheet at the bottom lists operations in the plan that may help in understanding the outcome of the evaluation and operations that may need attention.

Performance Statistics Section

Plan Analyzer uses color coding to identify the best (green) and worst (red) value for each resource (also referred to as *performance statistic*). The best is always the plan with the smallest value. The statistics in the

spreadsheet are a subset of the statistics actually collected. To view the full set of statistics, switch to Expert mode and invoke the Summary dialog (the **Summary** option on the **Performance** menu).

Elapsed Time

Elapsed Time is the number of seconds that have expired at the client side. Tracking elapsed time at the client side includes all network and server overhead. Keep in mind that different locations on the network produce different elapsed times. The best time resource for determining the optimal plan is the CPU time.

CPU Time

CPU Time is the number of seconds of CPU time consumed at the server. The `TIMED_STATISTICS` parameter in the `INIT.ORA` file must be set to `TRUE` in order for ORACLE to capture resource usage. This resource is not available on some ORACLE platforms, such as Personal Oracle, Netware, and OS/2.

Logical Blocks Read

Logical reads are requests for ORACLE blocks that are found in the SGA, rather than on a physical disk.

Physical Blocks Read

Physical Blocks Read is the number requested blocks that had to be read from a physical disk. These reads are far more expensive than logical reads. If a plan containing physical block reads is compared to plans that do not, the final recommendation can be skewed in the wrong direction because the cost per physical read is far more than any other resource.

Sorts

This resource combines all sorts performed during the test of the SQL statement. Although the Plan Analysis spreadsheet may show only one sort operation, that sort may be executed many times or not at all. Sorts can be expensive, especially when the sort is too large to be performed in the server's memory.

Full Table Scans

Like **Sorts**, full table scans may appear only once in a plan but may be executed many times. Some full table scans are not bad, depending upon the size of the table.

Plan Analysis Section

The Plan Analysis spreadsheet presents the number of times a specific operation occurs in the plan being evaluated. Check the **Advice** dialog for recommendations on how to avoid sorts and full table scans, and improve the plan. The operations are detailed below.

Full Table Scans

A full table scan means that all blocks of the table are read. The alternative is to use an index to qualify which data blocks to read. If the table is small, a full table scan may be better than using an index first.

Repeated Table Scans

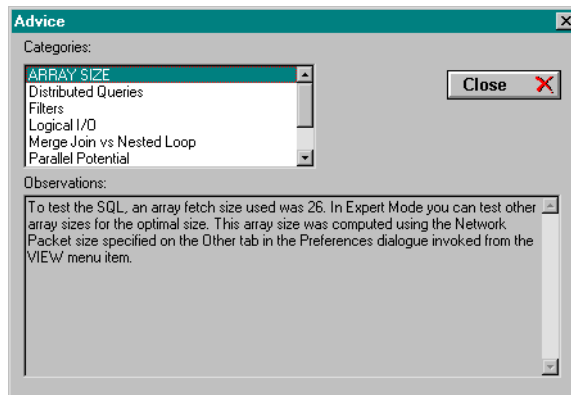
The worst type of full table scan is what Plan Analyzer refers to as a "repeated full scan." This is a full table scan whose location in an optimization plan would cause it to occur many times. Once is bad; many times is *very* bad. The Advice dialog, invoked through the **Advice** button, tells how to use hints to eliminate repeated full table scans.

Sorts

Sorts are operations in the plan that are performed to produce data output in a specific order, or to perform MIN, MAX, and other aggregate functions. A sort that appears in a plan may not need to be performed if an index can be used to return the data in the desired order. If the **Sorts** statistic in the **Performance Statistics** spreadsheet has a value of zero, it means that an index was used to satisfy the sort.

Advice

After the evaluation is complete, clicking **Advice** produces advice for improving the performance. Select a category of advice from the **Categories** list box to display a list of advice items in the **Observations** scroll box. These items assist you in understanding the current performance and suggest ways to improve performance, usually through the use of ORACLE hints.



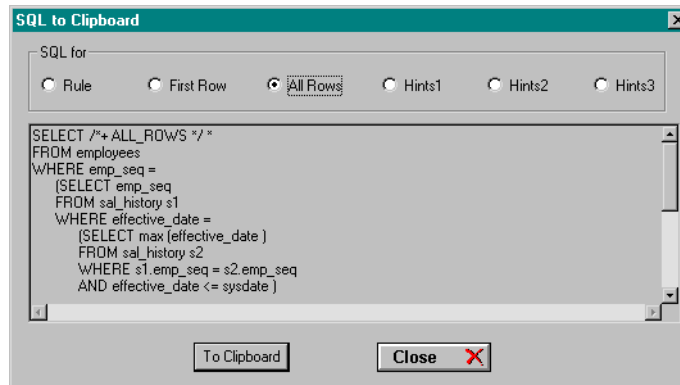
Advice Dialog

Copy SQL to Application

Once the best optimization plan is found, the SQL to Clipboard dialog (see following figure) helps you to replace the SQL statement in the application to obtain a specific optimization mode. Click **SQL To Clipboard** to display the SQL to Clipboard dialog. When the dialog appears, the radio button for the recommended optimization plan is set.

Most applications use the default optimization mode specified at the database level. If the final evaluation recommends the First Rows mode as the most optimal, what must you do to modify the SQL statement in the application so that the First Rows optimization mode is used? The SQL to Clipboard dialog will modify the SQL to ensure that the requested optimization plan is used. You can then click **Copy To Clipboard** to copy the modified SQL into a paste buffer. Find the SQL in the application and replace it by pasting. If you want an optimization plan other than the one recommended, click the radio button next to the desired plan type and then click **Copy To Clipboard** to copy the modified SQL into the paste buffer.

To achieve a specific optimization mode, Plan Analyzer embeds a hint for Rule, First Rows and All Rows in the outermost SQL module of the SQL statement. The example in the following figure uses the Hints optimization mode, so Plan Analyzer displays the hints entered by the user in each SELECT module. If you click the First Rows radio button, Plan Analyzer enters the FIRST_ROWS hint in the first SQL module and eliminates all hints entered by the user.



SQL to Clipboard Dialog

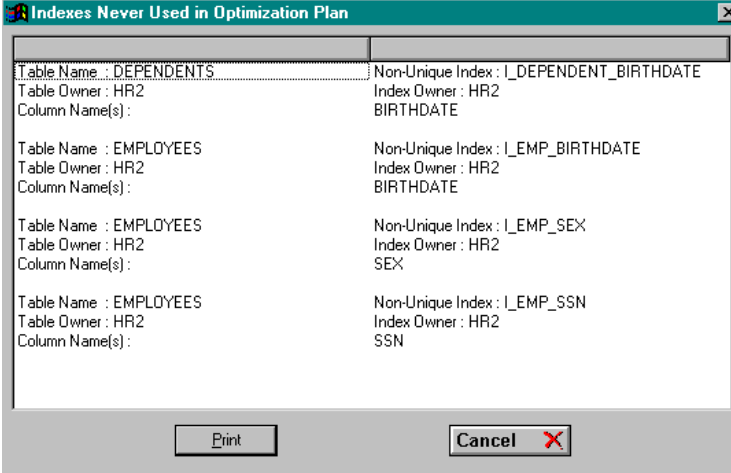
Reports Menu

This menu option appears on the main menu for both Expert and Standard modes. Plan Analyzer can generate three types of reports to assist DBAs in deciding which indexes should be dropped, which indexes are accessed the most, and which indexes are referenced in the Plan Analyzer repository but not in the database.

Unused Indexes	10-2
Index Frequency Count	10-2
Extinct Indexes Used in Plans	10-3

Unused Indexes

An index may be created on a table and then never be used to provide an access path to the table's data blocks. Choosing **Unused Indexes** creates a report which lists all existing indexes that are not referenced in any of the plans stored in the Plan Analyzer repository. The report is initially presented in a dialog with a **Print** button to print the report. The figure below shows a sample report.



Indexes Never Used in Optimization Plan	
Table Name : DEPENDENTS	Non-Unique Index : I_DEPENDENT_BIRTHDATE
Table Owner : HR2	Index Owner : HR2
Column Name(s) :	BIRTHDATE
Table Name : EMPLOYEES	Non-Unique Index : I_EMP_BIRTHDATE
Table Owner : HR2	Index Owner : HR2
Column Name(s) :	BIRTHDATE
Table Name : EMPLOYEES	Non-Unique Index : I_EMP_SEX
Table Owner : HR2	Index Owner : HR2
Column Name(s) :	SEX
Table Name : EMPLOYEES	Non-Unique Index : I_EMP_SSN
Table Owner : HR2	Index Owner : HR2
Column Name(s) :	SSN

Print Cancel

Unused Indexes Sample Report

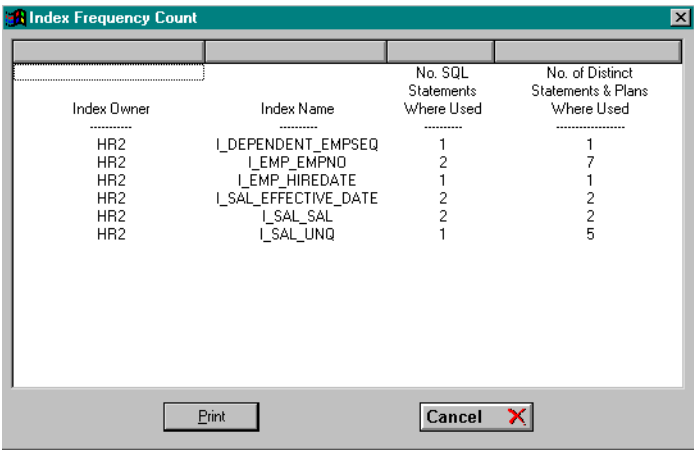
Index Frequency Count

The indexes that are used the most may be candidates for reorganization. Ensuring that the index blocks are contiguous and that the leaf pages are stored in order improves access time. These indexes may also be candidates for striping or mirroring. Mirroring probably provides the most improvement, since most mirrored disks allow sessions to read from either disk, thus increasing concurrent use.

Choosing **Index Frequency Count** creates the Index Frequency Count report, which lists:

- The index owner
- Each index stored in a Plan Analyzer repository plan
- The number of SQL statements that use the index, regardless of the number of plans stored for the SQL statement
- The number of distinct SQL statements and plans referencing the index

The report is presented in a dialog with a **Print** button so that you can print it. The figure below is an example report.



Index Owner	Index Name	No. SQL Statements Where Used	No. of Distinct Statements & Plans Where Used
HR2	I_DEPENDENT_EMPSEQ	1	1
HR2	I_EMP_EMPNO	2	7
HR2	I_EMP_HIREDATE	1	1
HR2	I_SAL_EFFECTIVE_DATE	2	2
HR2	I_SAL_SAL	2	2
HR2	I_SAL_UNQ	1	5

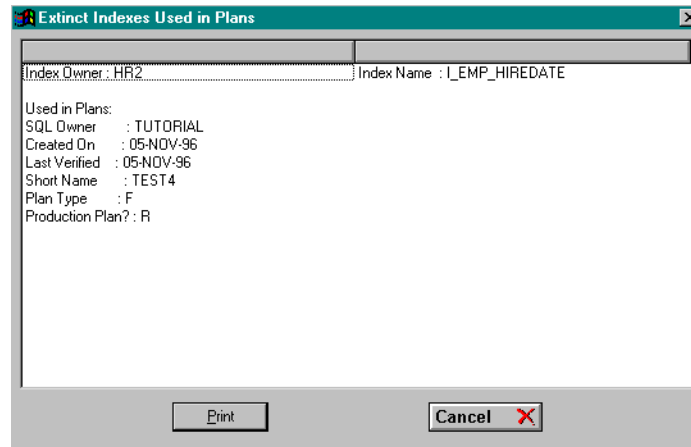
Index Frequency Count Report

Extinct Indexes Used in Plans

Indexes that no longer exist in the database may be referenced in an optimization plan stored in the Plan Analyzer repository. This is a problem if any of the plans that reference the index are used in production. The report includes information on whether the plan referencing the index is a production plan.

Choose **Extinct Indexes Used In Plans** to create the report. The figure below shows a sample report.

Extinct Indexes Used in Plans



Extinct Indexes Used in Plans Report

View Menu

This menu option is menu option is available for both Expert and Standard modes. The View menu contains options for controlling the display of the toolbar and status bar. It also contains the Preferences option and the options for collapsing and expanding the optimization plans.

View Menu Options	11-3
Toolbar	11-3
Status Bar	11-6
Collapse All Steps	11-6
Expand All Steps	11-6
Preferences	11-6
Predefined Configurations Tab	11-7
Database Connect Strings Tab	11-11
Analysis Alert Parameters Tab	11-13
Other Tab	11-18
Standard vs. Expert Mode	11-22

Cached Objects	11-23
Refresh	11-24
Drop	11-24

View Menu Options

The View menu contains a number of useful “housekeeping” options. From this menu, you set preferences about how Plan Analyzer will work. This is also where you choose whether to display the toolbar and status bar, whether to operate in Standard mode or Expert mode, examine cached objects, and where you control how many layers of steps are displayed.

Toolbar

Standard and Expert modes have different toolbars. The toolbars contain a number of shortcuts for invoking certain Plan Analyzer functions.


This is the Expert mode toolbar




This is the Standard mode toolbar:





To get help on an icon, position the mouse pointer over it and right-click. A message describing the icon displays in the left corner of the status bar at the bottom of the window.

The  icon invokes the Connect to Database function. It is only enabled when Plan Analyzer is not connected to a database.


The  icon invokes the Disconnect from Database function. It is only enabled when Plan Analyzer is connected to a database.


The  icon invokes the Clear function, which clears the entire application. You will be prompted to save unsaved work.


The  icon invokes the **Open SQL** command to copy the contents of a file into the SQL window.


The  icon invokes the Save SQL option which saves the SQL in the SQL window to the file the SQL was copied from. If the SQL did not originate from a disk file, the icon is disabled. For instance, if a file named SAMPLE.SQL is opened, clicking the third icon will save the SQL back to the same filename.

The  icon is equivalent to the **Print** command on the **File** menu.

The  icon cuts the highlighted section of the SQL text and puts it on the clipboard. You can undo the cut by clicking the **Undo** menu item on the Edit menu.


The  icon copies the highlighted section of the SQL text and places a copy on the clipboard.


The  icon pastes the contents of the clipboard into the SQL window at the current cursor position.



These six icons  (available only in Expert mode) produce the six types of optimization plans. The buttons in order from left to right create the: Rule based plan, the First Row plan, the All Rows plan, and the three possible Hints plans.

The  icon displays the Hints dialog.

The  icon displays the Server Statistics dialog in Expert mode.


The Plan Compare function is invoked by this button 

The  icon translates the current plan into standard mode.

These two icons   in Standard mode evaluate the SQL statement. The first is for batch evaluation, and the second is for interactive evaluation.

The  icon invokes the Server Statistics dialog.

The  icon invoke the DB Objects Analysis dialog.


The  icon displays the Help dialog box.

The Plans Toolbar (shown below) contains several other options:





As in the Expert mode toolbar, the  icon creates a Rule based plan.

The  icon invoke the DB Objects Analysis dialog on the selected object.

The  icon invokes the Step Detail dialog. This icon is enabled only when a step of the current optimization plan is highlighted and the step references a database object.

The icon invokes the Visualize dialog.

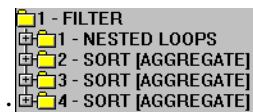
These two icons   are used for collapsing and expanding the steps of the optimization plans. The icon with the arrows pointed inwards collapses the plan steps, leaving only the top two levels of the plan. The icon with the arrows pointed outwards expands the steps, displaying all levels of the plan. The icons affect only the plan window that has the current focus. If both icons are disabled, click a plan window. If the current plan is completely expanded, the Expand Steps icon is disabled. As soon as a portion of the plan is collapsed, the Expand Steps icon is enabled.

Status Bar

The Status bar is at the bottom of the main window. The left side displays messages describing the different menu items. The mode—Expert or Standard—is displayed in the center section. The cost, rows, and bytes for a step in an optimization plan are shown at the right end of the Status bar (Oracle 7.2 or greater only).

Collapse All Steps

The **Collapse Steps** menu option collapses the steps of an optimization plan so that only the top two levels of the plan are visible. This gives you an overview of the operations being performed. The collapsed plan for the SQL statement in SAMPLE5.SQL is shown below



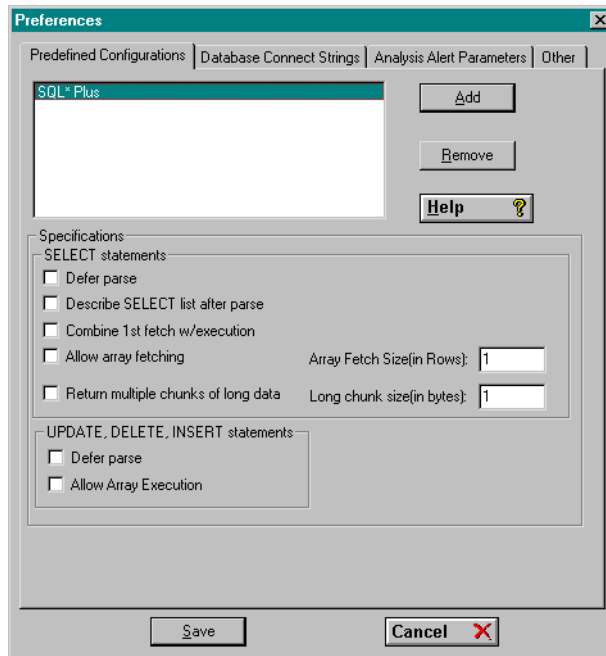
The collapsed plan indicates that the first operation is the nested loop followed by three sort operations. At this point, you could further explore the contents of the nested loop by expanding the step to display the child operations.

Expand All Steps

The **Expand Steps** menu option expands all steps of the current plan. Once all steps are expanded, the Expand Steps option is disabled.

Preferences

The Preferences menu option invokes the Preferences dialog in which you specify user-specific parameters. The Preferences dialog contains four tabs: **Predefined Configurations**, **Database Connect Strings**, **Analysis Alert Parameters**, and **Other**.



Preferences Dialog

Predefined Configurations Tab

The performance you can expect and the resources that can be consumed when you are embedding the SQL statement in an application are dependent on the product used to create the application—and on the skill of the developer. The predefined configurations are the specification of several parameters that determine the exact way in which the application will operate at the OCI level. If you are developing 3GL applications, the job of determining the configurations is simple, since you have complete control.

The Server Statistics dialog box allows you to execute the SQL statement being optimized to determine the resources required by the statement. The actual resources used are determined not only by the SQL statement, but also by a combination of the OCI (Oracle Call Interface) calls used

in the interface and the OCI options, such as array fetches. Consider, for example, whether the 4GL product in which the SQL will be embedded allows you to maintain a parsed SQL statement, or whether the product reparses the SQL each time it is executed. That one fact is extremely important to understanding the resources utilized.

Another significant choice 4GL products must make is whether to describe the SQL SELECT statements embedded in the application. Describing a SELECT statement means requesting information from the ORACLE database to identify each of the items on the SELECT list. The information includes the item name, datatype, length, etc. Describes are performed by 4GL products because the product has no knowledge of the SQL statement being entered by the developer, even though the developer could and does provide the information. Most 4GL products want to accommodate potential changes in the ORACLE data structures automatically, even though there is an expense involved.

ORACLE 7 provides a new parameter for the OCI parse call that defers the parsing until necessary. If the statement is a query, a deferred parse will not occur until the statement is described or executed. If the describe is not performed, the parse can be deferred until the execution. And if the SQL statement is a query, the execution can be combined with the first fetch. The point of deferring or combining calls to ORACLE is to reduce the amount of network traffic.

One final point is that ORACLE allows you to fetch an array or rows in a single fetch call, thereby reducing the overall volume of network messages. Does your product perform array fetches? And if so, does it allow you to specify a different array size per query? Again, the purpose of array fetches is to reduce the network messages.

The available configurations are listed in the **Predefined Configurations** tab of the Preferences dialog and can be modified on-line. Five configurations of popular tools are supplied with Plan Analyzer. Clicking a configuration displays the settings in the **Specifications** group box.

Specifications

Since SELECT and non-SELECT statements have different execution options, the specifications are divided appropriately. The following paragraphs describe the behavior of each option.

Defer parse defers a parse until the SELECT describe, if it occurs, or until the execution for a non-SELECT statement. If **Defer parse** is not set, the parse occurs immediately, requiring a round-trip network message.

Describe SELECT list after parse causes a describe for each query. A describe can have significant overhead on a network relative to the number of columns on the SELECT list, since a round-trip network message is required to describe each group of 16 fields on the SELECT list. If the query contains 32 fields, three round trip messages are needed: the third to determine that there are no more fields.

Combine 1st fetch w/execution combines the call to execute the SQL SELECT with the initial fetch. Combining the execute call with the initial fetch reduces the round-trip network messages from two to one. If array fetching is used, even more network messages are avoided. The recommendation is to combine the execute with the fetch and also use array fetching.

Allow array fetching indicates whether or not array fetching is provided by the product. By returning multiple rows, the number of round-trip network messages can be greatly reduced. If each fetch returns 20 rows, then 20 network messages have been reduced to one. Also, ORACLE permits a different array size per execution, so some queries may use an array size of 10 and others of 20. Queries returning only one row do not require an array fetch. It is strongly recommended to always perform array fetches, unless you know in advance that the SQL will return only one row. Array fetching should not be used if the query is returning LONG or LONG RAW columns, and multiple chunks will be returned.

Array Fetch Size (in Rows) is the default array size used in the tool. This number can always be changed, though this is the default array size used when the configuration is chosen.

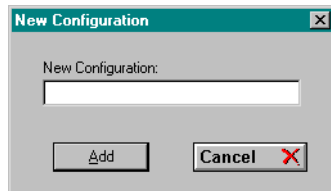
Return multiple chunks of long data refers to queries that return LONG or LONG RAW columns on the SELECT list. When one of these columns is returned, ORACLE 7 will only return the first piece of the column. The size of the piece is determined by the configuration parameter, LONG_CHUNK. To return subsequent pieces, a special OCI call must be made to retrieve each piece. Therefore, if you wish to return multiple chunks, you cannot perform array fetches.

Long chunk size (in bytes) is the size, in bytes, of each piece of a LONG or LONG RAW column that is fetched. (32512 or some integer value greater than zero).

Allow array execution is only available for non-SELECT statements, and indicates whether the configuration permits array INSERTs, UPDATEs, and DELETEs. Some 4GLs, such as Oracle*Forms, provide only array INSERTs. (See *Server Statistics* in the *Performance Menu* chapter for more details.)

Adding Predefined Configurations

Click **Add** to display the New Configuration dialog (see following figure). Enter a name for the new configuration and click **Add**.



New Configuration Dialog

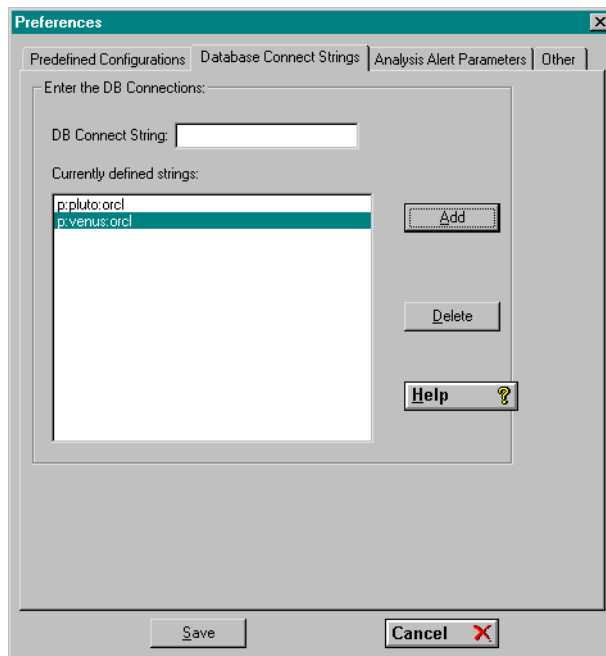
After you close the New Configuration dialog, the configuration name is highlighted in the list box, and all check boxes are set to false and the long chunk size to 1. Set the check boxes as desired and enter the default chunk size to use. When you're done, click **Save**.

Changing Predefined Configurations

You can change or delete current configurations. To change or delete a configuration, first click the configuration. Once the configuration is highlighted, set the check boxes as desired or change the long chunk size. To delete the configuration, click **Remove**. Once you're satisfied, click **Save**.

Database Connect Strings Tab

When connecting to a database, you are required to enter the name of the ORACLE account, the password, and a database connection string that identifies the database. The connection string can be either a SQL*Net V1 or V2 string. To save the connection strings, click **View, Preferences**. The Preferences dialog displays. Select the **Database Connect Strings** tab to display the following dialog:



Preferences Dialog (Database Connect Strings Tab)

Saving Changes

To save all changes and close the Preferences dialog, click **Save**. If you click **Cancel**, the changes will not be saved.

Removing Database Connection Strings

To remove a database connection string, select the string to delete and click **Delete**.

Adding Database Connection Strings

To add a database connection string, enter the string in the **DB Connect String** field, and click **Add**.

If you are not certain of the database connection string syntax, consult your DBA. SQL*Net Version 1 connect strings start with a letter identifying the protocol followed by a colon. After the colon comes the node name (the name of the server) followed by another colon. Finally the SID (instance name) is appended to the end. The following examples may help.

■ 2:

This is the connect string for Personal Oracle and Oracle on OS/2. Notice there is no node name.

■ X:ORASRV

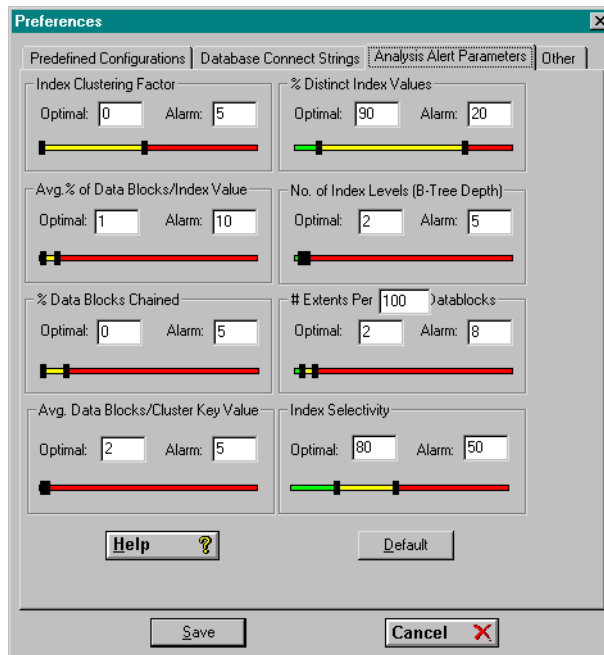
If the server is Netware, x represents the SPX protocol, and ORASRV is the name of the server.

■ T:MONGO:PROD

The protocol is TCP, the node name is MONGO, and the SID is PROD. If the server is a UNIX machine, ensure the correct case of the SID.

Analysis Alert Parameters Tab

The color coding in the Analysis dialog (see *DB Objects Analyze* in the *SQL Menu* chapter) is based on the settings for the optimal, warning, and alarm thresholds. These settings are specified in the Preferences dialog under the **Analysis Alert Parameters** tab.



Preferences Dialog (Analysis Alert Parameters Tab)

The sections below explain each of the parameters. Plan Analyzer default parameters can be reset by clicking **Default**. There are two ways to change a parameter: you can enter the value directly into the data field or you can use the slide bars to adjust the value. To use the slide bars, position the mouse pointer over the black tab on the slide bar and drag the tab until the field displays the desired value. The color of the slide bar corresponds to the color in the Analysis dialog (see the *SQL Menu* chapter), where red indicates alarm, yellow indicates warning, and green indicates the optimal setting.

Index Clustering Factor

The amount of order in the rows within the table determines the amount of I/O required to retrieve all rows for a specific index key. The best performance occurs when all rows for a specific index key are stored in the same data block. The worst performance occurs when all rows for a specific index key are located on different data blocks. The clustering factor is to some degree a measure of how well ordered the rows are.

To simplify the interpretation of the clustering factor, Plan Analyzer provides a scale from 0 to 10, where 0 means the table is perfectly ordered and 10 means it is random. When the number of data blocks used equals or exceeds the number of rows, Plan Analyzer considers the clustering factor to be optimal. Otherwise, Plan Analyzer divides the difference between the number of rows and the number of data blocks used by 10. The 10 represents the scale range. The closer the resulting figure is to the number of data blocks used, the more ordered the rows. The closer it is to the number of rows, the more random the distribution. The scale of 10 provides a means of identifying how close to one bound the clustering factor must be in order to classify the index as well ordered.

The default optimal value is 0.5 and the default alarm value is 5. Plan Analyzer ensures that the optimal value is always less than the alarm value.

Avg. % Data Blocks/Index Value

If an equality criterion references all columns in an index, the number of ROWIDs associated with the key value indicates the number of data blocks that must be retrieved. The desirable number of associated data blocks is 1, although realistically a larger value does not preclude the use of the index. Plan Analyzer displays the average number of data blocks per key (**Avg. Data Blocks/Index Value**) in the Analysis dialog box. The point is to use the percentage of matching data blocks rather than the average number of matching data blocks to determine whether the index is good. The percentage is the percentage of used data blocks compared with the average data blocks per key.

If **Avg. Data Blocks/Index Value** is less than or equal to the optimal threshold, then the field is colored green. If it is equal to or greater than the alarm threshold, the field is colored red. Otherwise, the field is yellow, as a warning.

The default optimal value is 1, and the default alarm value is 10. Plan Analyzer ensures that the optimal threshold is always less than the alarm threshold.

% Data Blocks Chained

Chained blocks increase the amount of I/O required to retrieve the data. The objective is to eliminate chained rows whenever possible. The number of chained blocks, like the average data blocks per key, is not as significant as the percentage of chained blocks compared to the total number of used data blocks. For example, if a table has 10 chained blocks out of 5000 used blocks, then only 0.2% of the blocks are chained; 10 chained blocks out of 100 used blocks is 10%.

Plan Analyzer provides parameters to specify the optimal and alarm thresholds for the percentage of the total used blocks. If the percentage of chained blocks is less than or equal to optimal value, the **Chained Blocks** field on the Analysis dialog box is colored green. If the percentage of chained blocks is greater than or equal to alarm value, the field is colored red. Otherwise the field is colored yellow to indicate a warning.

The default optimal value is 0, and the default alarm value is 0.5 (one half of one percent). Plan Analyzer ensures that the optimal value is less than alarm value.

Avg. Data Blocks/Cluster Key Value

A cluster key index always has one block address per cluster key, since ORACLE permits a maximum of one header block per cluster key. For that reason when too many rows are entered for the same cluster key, chaining occurs. But cluster chaining is quite different than normal chaining within an unclustered table. If all rows for a specific cluster key are always accessed together, then the chaining is meaningless. But if an individual row must be extracted for a cluster key value, then the

chaining forces additional I/O. Also different from normal chaining, the amount of chaining in clusters is not a factor of the percentage of used blocks, but simply the average chain length.

If the average cluster chain length (**Avg Data Blocks/Cluster Key Value** field in the Analysis dialog box) is less than or equal to the optimal value, the field is colored green. If the average cluster chain length is greater than or equal to alarm value, the field is colored red. Otherwise, the field is colored yellow to indicate a warning.

The default optimal value is 2, and the default alarm value is 5. Plan Analyzer ensures that the optimal value is always less than the alarm value.

% Distinct Index Values

The number of distinct keys in the index compared to the total number of rows is a good indication of the selectivity of the index. If the number of distinct keys is high compared to the total number of rows, the index has good selectivity. The number of distinct keys is displayed in the **Distinct Keys** field on the Analysis dialog box (see the *SQL Menu* chapter).

The comparison is made by computing the percentage of rows represented by the distinct keys. If the percentage is less than or equal to the optimal value, the **Distinct Keys** field is green. If the percentage is greater than or equal to the alarm value, the field is red. Otherwise the field is yellow indicating a warning condition.

The default optimal value is 90, and the default alarm value is 20. Plan Analyzer ensures that the optimal value is always greater than the alarm value.

No. of Index Levels (B-Tree Depth)

The size of the B-tree index can be measured in several ways. One measure is the number of I/Os performed to reach the bottom of the index. If the root and the leaf blocks are one and the same, then only one

I/O is performed and the **B-Tree Depth** field in the Analysis dialog box would equal 0. To determine the number of index blocks read to reach the leaf nodes of an index, add one to the B-Tree Depth.

If the B-tree depth is less than or equal to the optimal value, the field, **B-Tree Depth**, is colored green. If the B-tree depth is greater than or equal to alarm value, the field is colored red. Otherwise the field is colored yellow to indicate a warning.

The default optimal value is 2 and the default alarm value is 5. Plan Analyzer ensures that the optimal value is always less than the alarm value.

Extents Per n Datablocks

The performance guideline of always limiting the fragmentation with a table, index, or cluster is not exactly true. If the server has only one session active and only one CPU, it is true. Otherwise it isn't. To increase concurrence, a table's extents should span multiple disks (striping) allowing multiple users to access different parts of the table with no interference (which is the case if each user accesses an extent on a different disk). If the table is small, however, it is still best to have a single contiguous extent. To utilize the parallel query options in ORACLE V7.1 properly, the large tables must be striped. But even on a single disk it is not always possible or desirable to maintain a single extent.

Plan Analyzer allows you to specify the alarm number and optimal number of extents. In order to scale with small and large tables, the thresholds are relative to the number of blocks: a threshold is set for every n blocks of the table. The default value for the optimal setting is 2 per 100 blocks, and 8 per 100 for the alarm value. If the table is 500 blocks in size, it is still optimal if there are 10 or fewer extents. Plan Analyzer ensures that the optimal value is less than the alarm value.

If the number of extents for a database object (considering the size) is less than or equal to optimal value, the corresponding field on the Analysis dialog box is colored green. The alarm value specifies the alarm threshold. If the number of extents for a database object is greater than or equal to the alarm value, the corresponding field in the Analysis dialog

box is colored red. If the number of extents for a database object is greater than the optimal value and less than alarm value, the corresponding field in the Analysis dialog box is colored yellow.

Index Selectivity

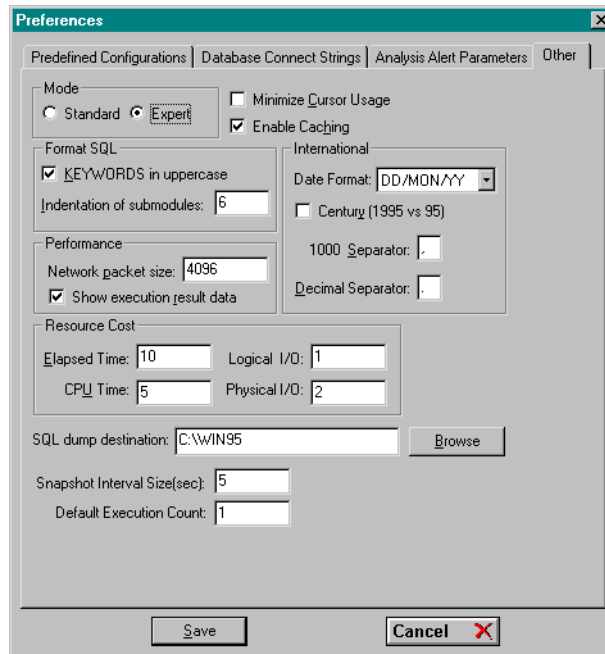
This is the ratio of distinct keys of an index divided by the number of rows in the table. A value of 100 represents a unique index since every row of the index is different.

A low value means that for every index, there are multiple rows in the table with the same value. An index with a low ratio should be dropped or modified.

When you are done altering the alarm values, click **Save** to save the settings and close the Preferences dialog

Other Tab

The **Other** tab of the Preferences dialog contains all other Plan Analyzer preferences. The following figure illustrates the **Other** tab.



Preferences Dialog (Other Tab)

Mode

In the Mode section you can set the default mode of operation to be Standard or Expert. For more information on Standard and Expert Modes, see the *File and Edit Menus* chapter.

Minimizing Cursor Usage

Plan Analyzer uses a number of cursors so as to maintain SQL parsed for optimal performance. Most of the cursors are opened immediately after logging in, thereby taking a period of time before any work can be done. The time period is especially long when connecting to a remote server, or connecting over an asynchronous line (for example, telephone).

To minimize the start-up delay, check the **Minimize Cursor Usage** check box to ensure that Plan Analyzer uses fewer cursors on subsequent logins.

Enable Caching

The process of producing optimization plans, showing step details, and requesting object analysis information requires a substantial number of queries. Since these queries typically access information on the same database objects, Plan Analyzer permits the retention of the information to prevent subsequent queries of the same information. If the **Enable Caching** check box is checked, the information is saved at the client side. This information can then be viewed by choosing **View, Cached Objects**.

Note • If **Enable Caching** is set, schema changes that occur after Plan Analyzer has been invoked will not be reflected. If you are working with a dynamic database, or you have re-ANALYZED an object, you will want to periodically dump or refresh the cache. For more information, see the **Cached Objects** section later in this chapter.

Format SQL

The **Format** menu option on the **SQL** menu uses the **Format SQL** setting specified here to format the SQL statement. There are two settings associated with SQL formatting: **KEYWORDS in uppercase** and **Indentation of submodules**. If **KEYWORDS in uppercase** is set, SQL syntax keywords such as **SELECT**, **AND**, and **WHERE** are changed to uppercase when **Format SQL** is invoked. Enter the number of character positions by which each SQL submodule (subquery) should be indented in the **Indentation of submodules** field.

International

You can change the format that Plan Analyzer uses to display dates and numeric values. By default, the date format is **DD/MON/YY**. Commas are used to separate 1000s within numbers, and a period is used to separate the decimal portion of numbers, but you can modify these with any characters or no characters in the **1000 Separator** and **Decimal Separator** fields. Clicking the **Date Format** combo box lists four different date formats:

- DD/MM/YY
- MM/DD/YY
- YY/MM/DD
- DD/MON/YY

To specify a four-digit year, set the check box for **Century (1996 vs. 96)**.

Performance

In Standard mode, Plan Analyzer calculates the array size for fetches when a batch evaluation is made. Plan Analyzer computes the array size by describing the SELECT list to determine the size of each column, totaling the columns for one row, adding column overhead, and dividing the network block size by the total. The network packet size is specified in bytes in the **Network packet size** field. The default is 4096 bytes.

When you test a plan in the Server Statistics dialog, you can view the returned rows in the Data tab if the **Show execution result data** check box is checked.

Resource Cost

In Standard mode, Plan Analyzer suggests the optimal plan after completing an evaluation of the SQL statement. A cost algorithm is used to weigh the resources and decide which is the optimal plan. The resources are the statistics in the Performance Statistics spreadsheet of the Evaluation Summary dialog, and the costs are specified in the Other tab of the Preferences dialog. The default costs used when Plan Analyzer is initially installed are listed below.

Elapsed Time	3
CPU Time	5
Logical I/O	1
Physical I/O	3

To ensure that one resource doesn't skew the recommendation, the values for each statistic in the Performance Statistics spreadsheet are totaled. The percentage of the total for each plan is then computed and multiplied by the cost.

SQL Dump Destination

This field allows you to specify the directory and file name for the location where you would like to dump selected SQL.

Snapshot Interval Size

This field allows you to specify the Snapshot interval size with a value in seconds. This is the time elapse for the SQL Capture, Active Problems function. For more information on the Snapshot function, see the *Capture* chapter.

Default Execution Count

This parameter controls how many times the SQL statement will be executed when the user hits the **Test** button in the Plan Statistics dialog. It is recommended that each SQL statement be executed three times to minimize the effects of server and network load.

Standard vs. Expert Mode

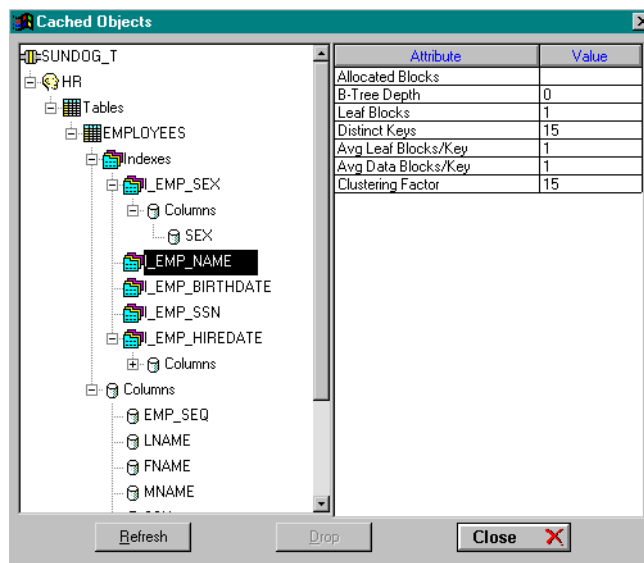
Plan Analyzer has two modes of operation, Standard mode and Expert mode. Expert mode offers greater flexibility and more options for testing, but the resulting plans are more complex. The current mode is always listed in the middle of the status bar. The mode can be dynamically changed in the **View** menu by clicking the opposite mode. For instance, if the current mode is Standard, the **Expert Mode** menu item appears in the **View** menu; if the current mode is Expert, then the **Standard Mode** option is present.

For more information about the specifics of Standard and Expert Mode, see the *File and Edit Menus* chapter.

Cached Objects

If **Enable Caching** is set in the **Other** tab of the Preferences dialog, clicking the **Cached Objects** menu option under **View** displays the Cached Objects dialog. The Cached Objects dialog contains a scroll list of the objects that are cached on the left. To view the cached information for an object, select it from the list. The cached information is displayed to the right of the list.

The figure below illustrates the **Index Information** for the I_EMP_NAME index, highlighted on the left.



Cached Objects Dialog

Refresh

Clicking **Refresh** retrieves all statistics for the selected database object.

If a database object is re-analyzed using the `SQL ANALYZE` command after it has already been cached in Plan Analyzer, its statistics must be manually refreshed using the **Refresh** button.

Drop

Since these statistics consume client memory, it may be necessary to drop some of the cached statistics if memory resources become low.

Highlighting a database object and clicking **Drop** removes the statistics from the cache.

Window Menu

The Window menu provides options to arrange the windows on the screen, and to identify which of the six Plan Analyzer windows has the cursor focus.

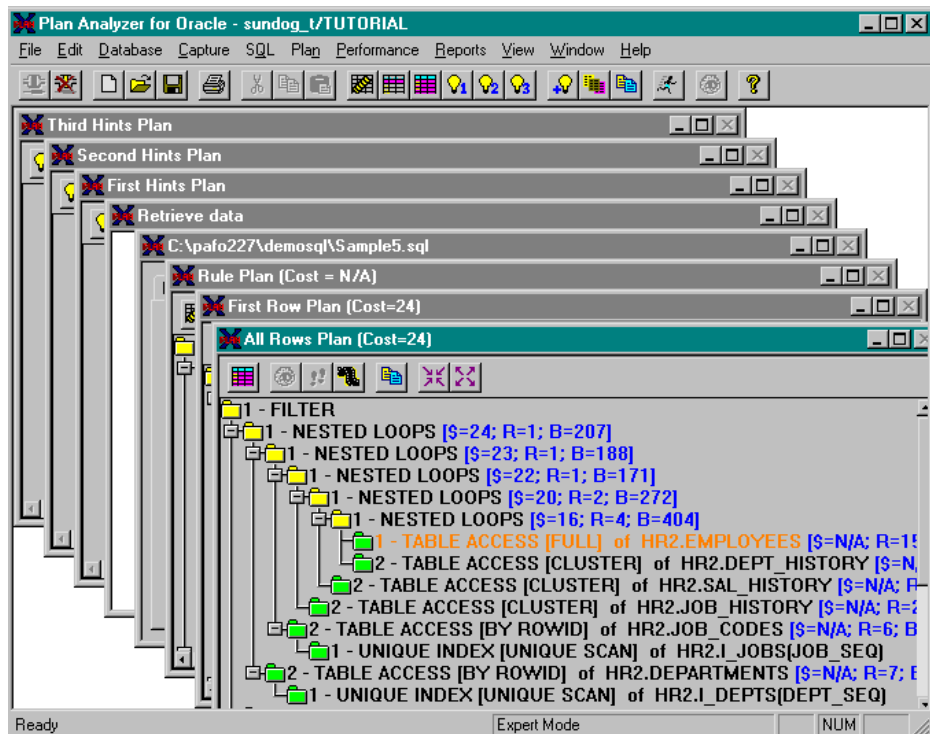
Arrange Icons	12-2
Cascade	12-2
Tile	12-3
Tile - SQL/Explain	12-4
SQL	12-5
Rule Plan	12-6
First Row Plan	12-6
All Rows Plan	12-6
First, Second, and Third Hints Plans	12-6
Retrieve Data	12-7

Arrange Icons

Choose **Window, Arrange Icons** to arrange iconized windows into a single line at the bottom of the main window.

Cascade

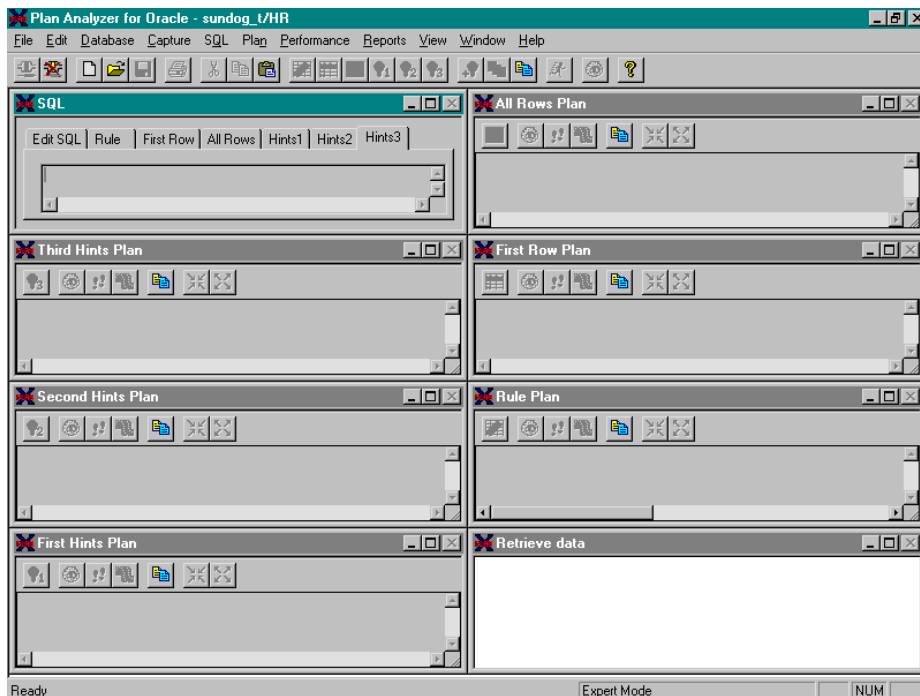
One way of viewing all non-minimized windows is to choose **Window, Cascade**. Cascade overlaps the windows so that the title bar of each is visible.



Cascaded Windows

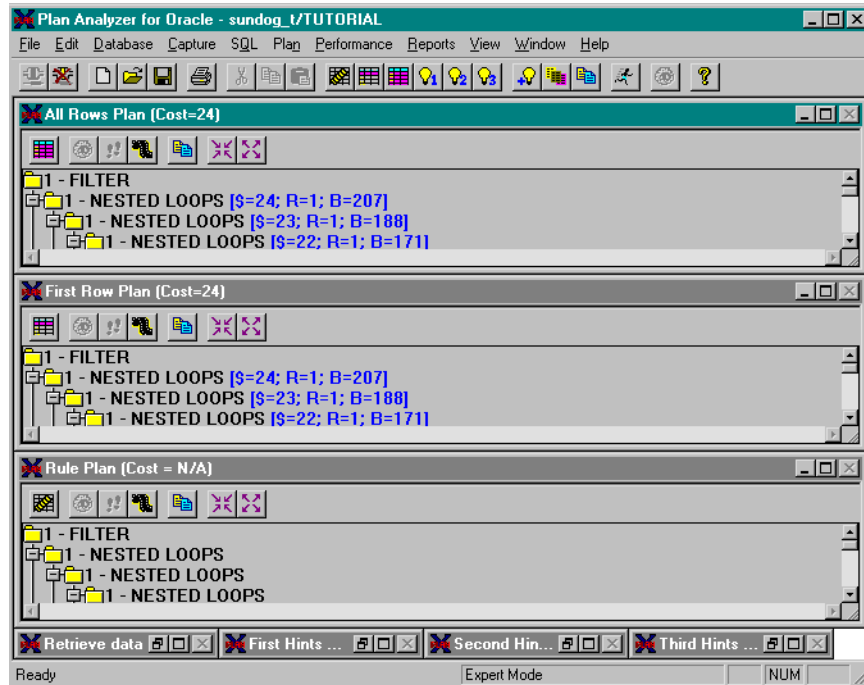
Tile

Tiling windows is another way to display all non-minimized windows at once. Choose **Window, Tile** to tile the windows. The following screen illustrates all eight windows displayed in a tiled fashion.



Tiled Windows

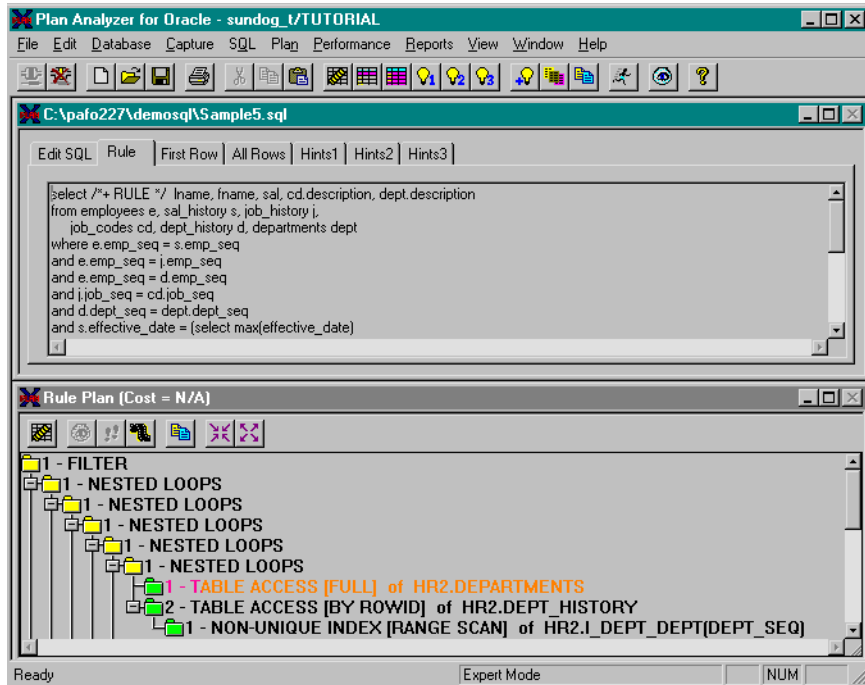
Tiling all windows leaves little room for viewing the contents of the different windows. Tiling is best used for displaying only a subset of the windows. For instance, to compare the optimization plans for Rule, Cost-First Row, and Cost-All Rows, minimize the other windows and then choose **Window, Tile**. The following figure shows the resulting screen.



Tiled Windows (with Minimized Windows)

Tile - SQL/Explain

Windows, Tile - SQL/Explain causes Plan Analyzer to display in its default state: with the SQL Window on top, and the Explain Windows layered on the bottom.



Tile - SQL Explain

SQL

Choosing **Window, SQL** positions the cursor in the SQL window that contains the SQL text. If you make any changes to the SQL statement in the SQL window, the execution data and all optimization plans and statistics are invalidated. If any hints are modified using the Plan Analyzer Hints dialog, only the hints optimization plan window is cleared.

Rule Plan

Choosing **Window, Rule Plan** positions the cursor in the Explain window for the Rule-based optimization plan. If the application is in **Tile - SQL/Explain** mode, the Rule Plan window displays at the bottom portion of the main window. If the application is in **Cascade** mode, the window comes to the top of the other windows, but retains its position.

First Row Plan

Choosing **Window, First Row Plan** positions the cursor in the Explain window for the first row-based optimization plan. If the application is in **Tile - SQL/Explain** mode, the First Row Plan window displays at the bottom of the main window. If the application is in **Cascade** mode, the window comes to the top of the other windows, but retains its position.

All Rows Plan

Choosing **Window, All Rows Plan** positions the cursor in the Explain window for the all rows-based optimization plan. If the application is in **Tile - SQL/Explain** mode, the Explain-All Rows window displays at the bottom of the main window. If the application is in **Cascade** mode, the window comes to the top of the other windows, but retains its position.

First, Second, and Third Hints Plans

Choosing the **Window, First, Second or Third Hints Plan** positions the cursor in the Explain window for the selected hints-based optimization plan. If the application is in **Tile - SQL/Explain** mode, the selected Hints Plan window displays at the bottom portion of the main window. If the application is in **Cascade** mode, the window comes to the top of the other windows, but retains its positions.

Retrieve Data

Choosing **Window, Retrieve Data** positions the Retrieve Data window below the SQL window in **Tile - SQL/Explain** mode. If the application is in **Cascade** mode, the window comes to the top of the other windows, but retains its position.



Glossary

of Hash Keys. • This is the number of hash values allocated for a hash cluster. Since each hash value is the address of a data block in the hash cluster, it indicates the number of distinct locations in which the rows will be stored. If the value is low and the number of rows is large, there may be a great deal of chaining.

A

analyze. • The SQL command ANALYZE can be used on a table, index, or cluster to produce the statistics used by the cost-based optimizer.

Avg. Blocks/Cluster Key • .The average number of data blocks associated with a specific cluster key value, whether the cluster is an indexed or hashed cluster. If the value is greater than 1, it indicates chaining: there are too many rows that have the same hash value and they have overflowed to other blocks.

B

batch application • . A batch application is one that is submitted to the server; none of the results of the execution are available until the entire application is complete. In the case of a SQL statement in a batch application, all of the results are returned after the SQL statement execution is complete. The All Rows cost-based optimization plan is considered the “batch oriented” plan.

B-Tree Depth • The number of blocks from the top of the index (root) to the leaf blocks. If the value equals 0, then the root block and the leaf block are one and the same.

bind variables • Tokens in a SQL statement that represent a variable value which will be changed between executions. The purpose of bind variables is to eliminate the expense of reparsing the SQL each time. The token is preceded by a colon and may be any valid name or number. Bind variables are mapped to program variables.

blocks used • The number of ORACLE blocks used to store rows of a table. Compare this number to the total number of blocks allocated to the table, or if clustered, to the cluster.

C

cluster index • This is the index created specifically for the cluster, where the columns are the cluster columns. This index can be used to access any of the tables stored in the cluster, and will appear as one of the indexes for each table in the cluster, when displaying the Index dialog for one of the tables.

clustered table • The rows of multiple tables can be physically stored together based on common columns (generally the join columns). By default, the rows of a single table are never mixed with those of another table. When rows of multiple tables are physically stored together, the tables are *clustered*.

Clustering Factor. • An index statistic representing the randomness of the data as represented by the index. For instance, if you are searching for a range of values in the index, you would prefer that the ROWIDs also be in order; you don't want to read the first data block, then the last, then the middle, then the 2nd to last, etc.

cost-based optimization. • Optimization of the SQL statement is based on rules and statistics that are gathered from the storage and from the table's data. Cost-based optimization is based solely on the statistics of the objects involved in the SQL, whereas rule-based optimization does not use any object statistics, only rules.

criteria. • The individual statements in the WHERE clause where either there is a column name reside on both sides of a relational operator, or there is a column name on one side and a constant on the other.

D

DBA. • An ORACLE user with SELECT ANY TABLE, DELETE ANY TABLE, UPDATE ANY TABLE, INSERT ANY TABLE and BECOME USER system privileges. This user can monitor sessions of other users and can retrieve the SQL and plans for other users.

describe. • Call to Oracle's OCI (ORACLE Call Interface) to return a description of the SELECT list; such as item name, datatype, default length, etc.

driving table. • The table in the SQL statement that is first accessed. For example, if the FROM clause has two tables, the query execution will be started by retrieving rows from one of the tables or from an index of one of the tables. The starting table is considered the driving table.

E

extent. • A contiguous group of database blocks allocated to a database object such as a table or index.

H

hints. • Instruction embedded in the SQL statement that to direct how the SQL is optimized. Hints provide maximum control over the cost-based optimization plan.

I

inner query. • See “outer query.”

interactive application. • An interactive application is typically an application that executes on a desktop computer. When the user requests data, the goal is to deliver the data immediately. This generally means returning a single row, rather than performing an array fetch. The First Rows plan is considered the “interactive-oriented” plan.

instances. • Each Oracle database is managed by a separate set of processes. Each set of processes is called an instance.

J

join clause. • A criterion in the WHERE clause that relates two tables; for example, the criterion where no constants appear on either side of the relational operator—only column names.

join criterion. • A criterion relating two tables together. A join criterion can never be used to drive the access of the tables, but can be used only after the access is started to relate one set of rows in one table to another set of rows in a different table.

K

key value. • A unique value in an index that corresponds to the value of the columns comprising the index. For each key value, there will be many ROWIDs if the index is non-unique.

L

leaf block. • The ORACLE blocks at the bottom of the index containing the actual row addresses (ROWIDs).

LRU chain. • When Oracle reads blocks on behalf of a user, those blocks are first copied to the SGA (Shared Global Area). Since the SGA is common to all users, the blocks can be read by other users without having to read them from disk. As blocks are copied to the SGA, other blocks may have to be replaced to accommodate the new blocks. The algorithm used to decide which blocks to replace is called the *least recently used* algorithm, where the blocks that have not been used for the longest time are replaced first. Oracle maintains the block timestamp information in a LRU (Least Recently Used) chain.

N

non-join criterion. • All criteria in a SQL statement appear in the WHERE clauses. A non-join criterion is one in which a column from different tables appears on each side of a relational operator.

O

OCI. • Oracle Call Interface. A set of API calls to the database for purposes such as logging in, opening a cursor, parsing a SQL statement, or executing a SQL statement.

outer query. • In a nested SELECT or correlated SELECT, the first SELECT keyword contains the items to be displayed upon execution. That SELECT is considered the “outer query,” and the nested or correlated subquery is the “inner query.”

P

parallel server. • A parallel server is a specific configuration of Oracle instances, where the database is common to all instances. All instances can participate in a parallel query if enabled.

parse. • A call to Oracle's OCI (Oracle Call Interface) to prepare the SQL statement for execution. The parse validates the SQL syntax, checks for proper security permissions, and produces an optimization plan.

R

Plan Analyzer repository. • The tables associated with the Plan Analyzer product, used for producing the explain plans and for storing the SQL and associated parts produced by the Plan Analyzer application.

result set. • The rows operated on by the SQL statement. If the SQL statement is a query, the result set is the set of rows returned.

root block. • The ORACLE block at the top of the index. This block is the first block read by ORACLE to utilize the index. The root block is also considered a branch block.

S

selectivity. • The characteristic of an index that indicates how well suited an index is for driving a search. A high selectivity indicates that an equality constraint will perform well, since only a small percentage of the

rows within the table would match. A low selectivity means the index would retrieve a large percentage of the rows. The goal is to drive the query using an index with a high selectivity.

SQL module. • If a SQL statement has a nested SELECT in the statement, the statement has more than one SQL module. Each nested SELECT is considered a separate SQL module.

T

TIMED_STATISTICS. • An `INIT.ORA` parameter that determines whether time will be available to measure several performance statistics, such as the reads/sec or CPU time used by a session. Time is not available unless `TIMED_STATISTICS` is set to `TRUE`; specifically, Plan Analyzer will not be able to display the CPU time in the Server Statistics dialog.

W

WAN. • Wide area network. The basic difference between a LAN (local area network) and WAN is that WAN network messages generally travel a far greater distance.



Index

A

- accounts, listing 4-13
- Active Problems 6-3
 - CPU tab 6-7
 - Logical Reads tab 6-8
 - Logical Writes tab 6-9
 - Physical Reads tab 6-10
 - Total I/O tab 6-10
- Advice dialog 9-6
- advice for improving performance 9-7
- All Objects dialog 4-13
- All Rows 5-21, 11-15
- All Rows hint 4-11
- All Rows Plan window 2-6
- ALL_ROWS hint 7-11
- Allocated Blocks 5-12, 5-14
- AND_EQUAL hint 7-12
- array fetch 8-8, 11-9
- Array Size 8-9, 11-8, 11-9
- ARRAY_FETCH 11-9
- Avg. Blocks/Cluster Key 5-12

- Avg. Data Blocks/Key 5-19, 5-22, 11-14
- AVG_BLOCKS_PER_CLUSTER_KEY_ALA
RM 11-16

B

- batch evaluation 9-2
- bind variables
 - Bind Variable Overhead 8-18
 - setting 5-2, 5-3
- Blocks Used 5-20, 11-14
- B-Tree Depth 5-14
- B-tree index 5-9

C

- CACHE hint 7-12
- Cached by default 4-18
- cached objects 11-23
- categories of advice 9-7
- Chained Blocks 5-9, 5-10
- Chained Rows 5-8, 5-9, 11-15
- CHAINED_BLOCKS_ALARM_PCT 11-15
- Chunk Size 8-10



CLUSTER hint 7-13
cluster index 5-12, 7-36
cluster keys 11-15
clustered columns, displaying 4-17, 7-14, 7-17
Clustering Factor 5-16, 5-20, 11-14
clusters 5-10, 7-37, 11-16
 average blocks per cluster key 5-12
 owned by an account 4-23
Coder 2-4
coder 2-12
Collapse Steps menu option 11-6
Combine Execution w/ Fetch 8-8
COMBINE_FETCH 11-9
Compare, menu item 7-46
Compile 4-21
connect strings 2-5
connecting to a database 2-5, 4-3
Copy SQL to Application 9-8
Copy SQL to Clipboard 5-4, 9-8
Copy To Clipboard 9-8
Copy to SQL Window 4-21
Cost All Rows optimization plan 2-14, 7-3, 7-6
Cost First Row optimization plan 2-13, 7-3, 7-5
cost-based optimization plans 4-11, 5-6, 5-9, 7-3, 7-5, 8-13, 9-2
CPU timing 1-26

D

database
 connecting to 2-5, 4-3
 disconnect from 4-3
 installation 1-9

 listing objects 4-13
Database Calls 8-2
Database Connect String 2-5, 4-3
date range 1-22, 4-6
DBInstall 1-3, 1-9, 1-11
 manually running 1-25
Default Optimization Mode 9-4
Default Parallel Servers field 4-18
DEFER_PARSE 11-9
Delete SQL option 4-13
demonstration files 1-27
dependency
 listing 4-23
 of procedures 4-22
 of tables 4-14
 of views 4-20
DESCRIBE 5-20, 11-8
Describe Select List 8-7
describing SELECT statement 5-20, 11-8
Details button 4-9
Differences button 4-11
disconnect from database 4-3
Distinct Keys 5-16, 5-22, 11-16

E

Elapsed Time 8-2, 8-9
Empty Blocks 5-8, 5-12
errors
 correcting 4-30
 in listing owners 7-39
 in statistics 4-29, 4-30
evaluating
 advice 9-7
 hints 9-3
 optimization plans 9-2, 9-3

- performance statistics 9-4
 - statements 9-2, 9-3
- Evaluation Summary dialog 9-4
- Execution Configuration dialog 9-3
- Expand Steps menu option 11-6
- Expert Mode 2-8
- Expert mode 2-6, 7-49, 11-3, 11-6, 11-19
 - optimization plans 7-3
- Expert mode plans 7-3
- Expert vs. Standard mode 2-9, 11-22
- Extents 5-8, 5-10
- F**
- First Row 2-13, 8-2
- First Row Plan window 2-6
- FIRST_ROWS hint 7-14
- folder
 - save SQL in 4-5
 - search for 4-7
- Format menu item 5-5
- formatting SQL statements 5-5
- FULL hint 7-15
- full table scans 4-18, 5-10, 5-16
- functions
 - listing 4-22
 - owned by an account 4-22
 - showing dependencies 4-24
- H**
- HASH hint 7-16
- HASH_AJ hint 7-17
- header block 5-9, 5-13, 11-15
- hints
 - adding 4-21, 7-9
 - ALL_ROWS 7-11
 - and optimization 5-4, 7-3, 9-8
 - AND_EQUAL 7-12
 - CACHE 7-12
 - CLUSTER 7-13
 - customizing plans with 8-2
 - discussion of 2-14, 7-6, 7-8
 - evaluating 9-3
 - FIRST_ROWS 7-14
 - FULL 7-15
 - HASH 7-16
 - HASH_AJ 7-17
 - in subqueries 7-8
 - INDEX 7-20
 - INDEX_ASC 7-21
 - INDEX_COMBINE 7-21
 - INDEX_DESC 7-22
 - INDEX_FFS 7-22
 - MERGE_AJ 7-23
 - modifying 12-5
 - NOCACHE 7-26
 - NOCOST 7-27
 - NOPARALLEL 7-27
 - ORDERED 7-28
 - PARALLEL 7-29
 - PUSH_SUBQ 7-31
 - ROWID 7-32
 - RULE 7-33
 - specifying 7-4, 7-6
 - STAR 7-33
 - USE_CONCAT 7-34
 - USE_HASH 7-35
 - USE_MERGE 7-35
 - USE_NL 7-36
- Hints dialog 7-8, 11-4

Hints optimization plan 4-11, 7-3, 7-6

I

INDEX hint 7-20

INDEX_ASC hint 7-21

INDEX_COMBINE hint 7-21

INDEX_DESC hint 7-22

INDEX_FFS hint 7-22

indexes 5-7, 5-10, 5-16, 5-19, 5-22, 7-4, 7-8, 7-36

 allocated blocks 5-12, 5-14

 average data blocks per key value 5-19, 5-22, 11-14

 B-tree depth 5-14

 clustering factor 5-16, 5-20, 11-14

 displaying available indexes 5-7, 5-10, 5-16, 5-19, 5-22, 7-4, 7-8, 7-36

 leaf blocks 5-15, 5-16, 11-16

 list 4-18

 number of distinct key values 5-16, 5-22

 root block 5-15, 5-18

 selectivity 7-4, 11-16

 statistics 5-6

 storage statistics 5-7, 5-10, 5-16, 5-19, 5-22, 7-4, 7-8, 7-36

indirect object references 7-37

installation 1-3

 client 1-5

 full 1-7

 manual 1-25

 network 1-7

 SQL-Station 1-5

interactive evaluation 9-2

J

join criterion 7-45

K

Keys/Checks on tables 4-15

L

Leaf Blocks 5-15, 5-16, 11-16

license password 1-3

LOCAL variable 4-3

logging into the database 1-23, 4-3, 4-7

login 1-23, 4-3, 4-7

 to database 4-3

LONG_CHUNK 11-10

M

merge join 5-17

MERGE_AJ hint 7-23

migrating a repository 1-21

modes of operation 11-19

Monitor facility 6-29

 Batch details 6-36

 Capture Results tab 6-32

 Jobs in batch mode 6-31

 Review Window 6-35

 Top SQL Resources 6-29

More button 4-9

multiple extents 5-10

MULTIPLE_CHUNKS 11-10

N

network installation 1-3, 1-7

No. of Chunks to Retrieve 8-11

No. of Fetches 8-10

No. of instances split across 4-18

No. of values per bind variable 5-3
 NOCACHE hint 7-26, 8-16
 NOCOST hint 7-27
 NOPARALLEL hint 7-27
 Number of Rows 5-20, 8-9

O

optimization 2-3, 4-4, 5-9, 5-17, 7-36, 8-2, 12-3
 optimization plans
 advice 9-6
 All Rows 5-21
 choosing 8-2
 choosing a plan 8-2
 Cost All Rows 2-14, 7-3, 7-6
 Cost First Row 2-13, 7-3, 7-5
 cost, rows, and bytes 7-42
 cost-based 4-11, 5-9, 7-3, 7-5, 8-13, 9-2
 DB Objects Analyze 5-6
 displaying steps of 11-5, 11-6
 evaluating 9-2, 9-3
 evaluation 9-4
 explaining 2-16
 First Row 7-8
 Hints 2-14, 5-18, 7-3, 7-6
 hints in 8-2, 9-8
 overview 2-3
 performance statistics 9-4
 Rule 7-3, 7-4
 rule-based 2-9, 2-13, 2-14, 5-17, 7-38, 9-2
 rules-based 7-8
 testing 9-2
 verification 4-11
 visualizing 2-15, 2-16

optimizing
 views 4-21
 ORACLE.INI file 4-3
 ORDERED hint 7-28
 owner
 of database objects 4-13

P

packages, owned by an account 4-22
 PAFO32.INI file 1-26, 4-3
 PARALLEL hint 7-29, 7-41
 password 1-3
 Paste from SQL Window 4-21
 Performance Options
 Array Size 8-9, 11-8, 11-9
 Chunk Size 8-10
 Combine Execution w/ Fetch 8-8
 Describe Select List 8-7
 No. of Chunks to Retrieve 8-11
 No. of Fetches 8-10
 performance statistics 3-5, 9-4
 Chained Rows 5-8, 5-9, 11-15
 CPU Time 9-5
 Database Calls 8-2
 Elapsed Time 8-2, 8-9, 9-5
 Full Table Scans 9-6
 Logical Blocks Read 9-5
 maintaining 4-29
 Physical Blocks Read 9-5
 Plan Analysis 9-6
 Repeated Table Scans 9-6
 Sorts 9-6, 9-7
 Plan Analyzer repository 1-23, 4-7, 8-2
 retrieving stored SQL 4-5, 5-14, 8-10, 11-10, 11-14, 11-15

- storing SQL and plans 5-13, 8-8
- Plan in production (Save SQL dialog) 4-5
- plans
 - finding changed 4-10
 - save to file 3-3
 - saving 4-5
- predefined configuration
 - ARRAY_FETCH 11-9
 - COMBINE_FETCH 11-9
 - DEFER_PARSE 11-9
 - DESCRIBE 5-20, 11-8
 - LONG_CHUNK 11-10
 - MULTIPLE_CHUNKS 11-10
- predefined configurations
 - adding 11-10
 - changing 11-11
- preferences
 - # Extents Per n Datablocks 11-17
 - % Data Blocks Chained 11-15
 - % Distinct Index Values 11-16
 - Allow array execution 11-10
 - Allow array fetching 11-9
 - Analysis Alert Parameters 11-13
 - Array Fetch Size (in Rows) 11-9
 - Avg. % Data Blocks/Index Value 11-14
 - Avg. Data Blocks/Cluster Key Value 11-15
 - Combine 1st fetch w/execution 11-9
 - Database Connect Strings 11-11
 - Date Format 11-20
 - default mode of operation 11-19
 - Defer parsing 11-9
 - Describe SELECT list after parse 11-9
 - Enable Caching 11-20
 - Format SQL 11-20

- Index Clustering Factor 11-14
- Long chunk size (in bytes) 11-10
- Minimize Cursor Usage 11-19
- Network packet size 11-21
- No. of Index Levels (B-Tree Depth) 11-16
- Resource Cost 11-21
- Return multiple chunks of long data 11-10
- setting 11-6
- privileges
 - displaying 4-16
- procedures
 - dependencies 4-22
 - details 4-22
 - listing 4-22
- PUSH_SUBQ hint 7-31

R

- repository 1-3, 1-10
 - migrating 1-21
- resources
 - measuring 4-29
- retrieve 5-14, 8-10, 11-10, 11-14, 11-15
- Retrieve Data window 2-8
- retrieve SQL 4-9
 - set search criteria 4-6
 - specify criteria 4-5
- root block 5-15, 5-18
- ROWID hint 7-32
- Rule 5-17, 7-8, 7-38, 8-2
- RULE hint 7-33
- Rule optimization plan 7-3, 7-4
- Rule Plan window 2-6
- rule-based optimization plans 2-9, 2-13,

2-14, 7-4, 9-2

S

save

optimization plans 4-3

SQL as file 3-3

SQL file 4-3

statistics 4-4

Save SQL As 3-3

save SQL file 4-3

saving 5-13, 8-8

search criteria, setting 4-6

Second Hints Plan window 2-6

selectivity 7-4, 11-16

sequences, owned by an account 4-22

Sessions filter 4-13

Set Bind Variable 5-2

Show Children 2-17

Show Parent 2-17

Show Siblings 2-17

Snapshot facility 6-10

All SQL tab 6-14

By Datafile tab 6-25

By Ora User tab 6-17

By OS User tab 6-19

By Resource tab 6-27

By Session tab 6-15

By SQL Text tab 6-28

By Table Owner tab 6-20

By Tablename tab 6-22

By Tablespace tab 6-23

Result tab 6-11

Search dialog 6-12

Sort dialog 6-13

sort join 5-17

Sorts statistic 9-7

SQL

Copy To Clipboard 9-8

copying to SQL window 4-21

formatting 5-5

modifying in application 9-8

pasting back 4-21

retrieving saved 4-5, 4-9

retrieving stored 8-10, 11-10, 11-14,
11-15

save as file 3-3

saving 4-4

storing 5-13, 8-8

SQL owner

search for 1-23, 4-7

SQL statements

copy to application 9-8

deleting 4-13

details 4-9, 4-13

displaying 4-9

entering 2-12

evaluating 9-2, 9-3

formatting 5-5

retrieving 4-5

saving 4-3

short name 1-23, 4-7

verifying 4-12

SQL to Clipboard dialog 9-8, 9-9

SQL type, search for 1-23, 4-6

SQL window 2-6, 2-8, 2-12

copying to 4-21

formatting statements in 5-5

pasting to 4-21

retrieving data from 5-24

SQL-Station

- installation 1-5
- SQL-Station Coder 2-4, 2-12
- Standard mode 2-6, 2-9, 7-49, 9-2, 11-3, 11-6, 11-19
- Standard Translation menu option 7-49
- Standard vs. Expert mode 2-9, 11-22
- STAR hint 7-33
- starting Plan Analyzer 2-4
- statistics
 - checking for errors 4-29
 - DB Objects Analyze 5-6
 - errors in 4-30
 - for optimization plans 5-6
 - saving 4-4
 - viewing 9-4
- Statistics, of result sets 6-4
- statistics, performance 4-29
- Status bar 11-6
- Storage Statistics
 - Allocated Blocks 5-12, 5-14
 - Avg. Blocks/Cluster 5-12
 - Avg. Data Blocks/Key 5-19, 5-22, 11-14
 - Blocks Used 5-20, 11-14
 - B-Tree Depth 5-14
 - Chained Blocks 5-9
 - Clustering Factor 5-16, 5-20, 11-14
 - Distinct Keys 5-16, 5-22
 - Empty Blocks 5-8, 5-12
 - Extents 5-8, 5-10
 - Leaf Blocks 5-15, 5-16, 11-16
 - Number of Rows 5-20, 8-9
- striping 5-10
- Suggested Plan for Use 9-4

T

- tables
 - clustered columns in 4-17
 - data blocks used 5-20, 11-14
 - dependency 4-14
 - details 4-14
 - full table scans 4-18
 - in a cluster 4-23
 - keys/checks 4-15
 - list triggers on 4-19
 - listing 4-14
 - lsit indexes on 4-18
 - privileges on 4-16
 - statistics 5-6, 5-7
 - table name 1-23, 4-7
 - table owner 1-23, 4-7
- Third Hints Plan window 2-6
- Transactions/Data Block 4-18
- triggers
 - listing 4-19
 - Trigger dialog 4-20
- tutorial account 1-27, 2-4

U

- Uninstall 1-20
- upgrading Plan Analyzer 1-11
- USE_CONCAT hint 7-34
- USE_HASH hint 7-35
- USE_MERGE hint 7-35
- USE_NL hint 7-36
- used blocks 5-20, 11-15

V

- variables
 - and array execution 8-11

- Bind Variable Overhead 8-18
- LOCAL environment variable 4-3
- set bind variable 5-2
- setting to return SELECT values 8-7
- Verification dialog 4-11
 - specifying criteria 4-12
 - using the Verify option 4-12
 - viewing plan differences 4-12
- views
 - dependencies 4-20
 - details 4-20
 - optimizing 4-21
 - owned by an account 4-20
- Visualize facility 2-15
- visualizing the plan 2-15

W

- Window
 - Tile command 2-7
- window
 - All Rows Plan 2-6
 - First Row Plan 2-6
 - Rule Plan 2-6
 - Second Hints Plan 2-6
 - Third Hints Plan 2-6
- windows
 - viewing 2-6

